

HP 3000 MPE/iX Computer Systems

ALLBASE/SQL

Database Administration Guide



HP Part No. 36216-90005
Printed in U.S.A. August 1997

Seventh Edition
E0897

The information contained in this document is subject to change without notice.

Hewlett-Packard makes no warranty of any kind with regard to this material, including, but not limited to, the implied warranties of merchantability or fitness for a particular purpose. Hewlett-Packard shall not be liable for errors contained herein or for direct, indirect, special, incidental or consequential damages in connection with the furnishing or use of this material.

Hewlett-Packard assumes no responsibility for the use or reliability of its software on equipment that is not furnished by Hewlett-Packard. This document contains proprietary information which is protected by copyright. All rights reserved. Reproduction, adaptation, or translation without prior written permission is prohibited, except as allowed under the copyright laws.

Restricted Rights Legend

Use, duplication, or disclosure by the U.S. Government is subject to restrictions as set forth in subparagraph (c) (1) (ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013. Rights for non-DOD U.S. Government Departments and Agencies are as set forth in FAR 52.227-19 (c) (1,2).

Hewlett-Packard Company
3000 Hanover Street
Palo Alto, CA 94304 U.S.A.

Printing History

The following table lists the printings of this document, together with the respective release dates for each edition. The software version indicates the version of the software product at the time this document was issued. Many product releases do not require changes to the document. Therefore, do not expect a one-to-one correspondence between product releases and document editions.

Edition	Date	Software Version
First Edition	December 1987	36216-02A.01.00
Second Edition	October 1988	36216-02A.12.00
Third Edition	October 1989	36216-02A.20.00
Fourth Edition	December 1990	36216-02A.E1.00
Fifth Edition	June 1992	36216-02A.F0.00
Sixth Edition	April 1994	36216-02A.G0.00
Seventh Edition	August 1997	36216-02A.G2.00

ALLBASE/SQL Documents

Title	Customer Order Number
<i>ALLBASE/NET User's Guide</i>	36216-90031
<i>ALLBASE/SQL Advanced Application Programming Guide</i>	36216-90100
<i>ALLBASE/SQL C Application Programming Guide</i>	36216-90023
<i>ALLBASE/SQL COBOL Application Programming Guide</i>	36216-90006
<i>ALLBASE/SQL Database Administration Guide</i>	36216-90005
<i>ALLBASE/SQL FORTRAN Application Programming Guide</i>	36216-90030
<i>ALLBASE/SQL Message Manual</i>	36216-90009
<i>ALLBASE Pascal Application Programming Guide</i>	36216-90007
<i>ALLBASE/SQL Performance and Monitoring Guidelines</i>	36216-90102
<i>ALLBASE/SQL Reference Manual</i>	36216-90001
<i>ISQL Reference Manual for ALLBASE/SQL and IMAGE/SQL</i>	36216-90096
<i>Up and Running with ALLBASE/SQL</i>	36389-90011
<i>ODBCLINK/SE Reference Manual</i>	36217-90403

Preface

This manual describes how to design, create, and maintain ALLBASE/SQL databases on HP 3000 computers running under the MPE/iX operating system. ALLBASE/SQL is Hewlett-Packard's proprietary relational database management product.

MPE/iX, Multiprogramming Executive with Integrated POSIX, is the latest in a series of forward-compatible operating systems for the HP 3000 line of computers. In HP documentation and in talking with HP 3000 users, you will encounter references to MPE XL, the direct predecessor of MPE/iX. MPE/iX is a superset of MPE XL. All programs written for MPE XL will run without change under MPE/iX. You can continue to use MPE XL system documentation, although it may not refer to features added to the operating system to support POSIX (for example, hierarchical directories).

This manual contains advanced information about ALLBASE/SQL database administration. It is intended for experienced users of SQL and SQL application programmers. Topics are discussed in separate chapters, as follows:

- Chapter 1, "DBA Tasks and Tools," presents the basic concepts and terms relating to ALLBASE/SQL database administration.
- Chapter 2, "Logical Design," shows how to create a database schema prior to creating an ALLBASE/SQL DBEnvironment.
- Chapter 3, "Physical Design," describes the physical files used in ALLBASE/SQL, and shows how to calculate the amount of space required for various database objects.
- Chapter 4, "DBEnvironment Configuration and Security," details the steps in creating a new DBEnvironment.
- Chapter 5, "Database Creation and Security," shows how to create specific database objects.
- Chapter 6, "Backup and Recovery," describes procedures for routine backup and for recovery when needed.
- Chapter 7, "Maintenance," presents the tasks required for modifying a DBEnvironment once it has been in operation.
- Chapter 8, "System Catalog," contains a complete description of all the system catalog pseudotables and views.

The following appendixes contain additional reference material.

- Appendix A describes system limits.
- Appendix B presents a table of the authorities required to execute ALLBASE/SQL commands.
- Appendix C is a summary of SQL syntax.
- Appendix D is a summary of ISQL syntax.
- Appendix E is a list of locks obtained on the system catalog by various SQL commands.
- Appendix F contains the syntax of SQLUtil commands.
- Appendix G contains the syntax of SQLGEN commands.
- Appendix H contains the syntax of SQLMigrate commands.
- Appendix I contains the syntax of SQLAudit commands.

Most of the examples in this manual are based on the tables, views, and other objects in the sample DBEnvironment PartsDBE. For complete information about PartsDBE, refer to the *ALLBASE/SQL Reference Manual*, appendix C.

What's New in this Release

G.1 and G.2 New Features

The following table highlights the new or changed functionality added in G.1 and G.2 releases, and shows you where each feature is documented.

New Features in ALLBASE/SQL Releases G.1 and G.2

Feature (Category)	Description	Documented in . . .
New operand to concatenate strings (Standards)	Adds an operand to concatenate character or binary strings in an expression. New operand:	<i>ALLBASE/SQL Reference Manual</i> , "Expressions."
RENAME Column or Table (Usability)	Adds capability of defining a new name for an existing table or column in a DBEnvironment. You cannot rename a table or column that has check constraints or an IMAGE/SQL table. New commands: RENAME COLUMN, RENAME TABLE.	<i>ALLBASE/SQL Reference Manual</i> , RENAME COLUMN and RENAME TABLE in "SQL Statements."
CAST function added to Expression syntax (Usability)	Adds the CAST function to allow explicitly converting from one data type to another. It allows conversion between compatible data types and between normally incompatible data types such as CHAR and INTEGER. New Expression function: <i>CastFunction</i> .	<i>ALLBASE/SQL Reference Manual</i> , "Cast" in "Expressions."
Syntax added to VALIDATE (Usability, Performance)	Automates execution of COMMIT WORK after each module or procedure is validated when WITH AUTOCOMMIT is used. All sections are revalidated whether valid or invalid when FORCE is used. This can reduce log space and shared memory requirements for the VALIDATE statement. New syntax for VALIDATE: FORCE, WITH AUTOCOMMIT.	<i>ALLBASE/SQL Reference Manual</i> , VALIDATE in "SQL Statements."

New Features in ALLBASE/SQL Releases G.1 and G.2 (continued)

Feature (Category)	Description	Documented in . . .
Syntax added to DELETE (Usability, Performance)	Automates execution of COMMIT WORK at the beginning of the DELETE and after each batch of rows is deleted when WITH AUTOCOMMIT is used. Reduces log-space and shared-memory requirements. WITH AUTOCOMMIT cannot be used in some cases (see the DELETE statement). New syntax for DELETE: WITH AUTOCOMMIT.	<i>ALLBASE/SQL Reference Manual</i> , DELETE in "SQL Statements."
Decimal operations (Usability)	Increases maximum precision from 18 to 27.	<i>ALLBASE/SQL Reference Manual</i> , "Decimal Operations" in "Data Types."
Terminate a query (Usability, Performance)	Allows termination of a query for a connection or transaction. New statement: TERMINATE QUERY. New syntax for SET SESSION, SET TRANSACTION.	<i>ALLBASE/SQL Reference Manual</i> , TERMINATE QUERY, SET SESSION, SET TRANSACTION in "SQL Statements."
Terminate a transaction (Usability, Performance)	Allows stopping of a given transaction. New statement: TERMINATE TRANSACTION. New syntax for SET SESSION, SET TRANSACTION.	<i>ALLBASE/SQL Reference Manual</i> , TERMINATE TRANSACTION, SET SESSION, SET TRANSACTION in "SQL Statements."
Timeout enhanced to allow specifying what is rolled back or terminated (Usability, Performance)	Allows specifying the action when a timeout expires. New attributes for SET SESSION and SET TRANSACTION: TERMINATION AT LEVEL, USER TIMEOUT, ON TIMEOUT ROLLBACK.	<i>ALLBASE/SQL Reference Manual</i> , SET SESSION in "SQL Statements."
Allow or disallow SQLMON for users. (Usability)	Grants or revokes the ability to run SQLMON for specific users. New attribute for GRANT and REVOKE: MONITOR.	<i>ALLBASE/SQL Reference Manual</i> , GRANT, REVOKE in "SQL Statements."

New Features in ALLBASE/SQL Releases G.1 and G.2 (continued)

Feature (Category)	Description	Documented in . . .
Allow or disallow authority to create modules. (Usability)	Grants or revokes the ability to create modules for specific users. New attributes for GRANT and REVOKE: INSTALL.	<i>ALLBASE/SQL Reference Manual</i> , GRANT, REVOKE in “SQL Statements.”
Script for migration to a new release (Usability, Tools)	Provides SQLINSTL script for migration to a new release of ALLBASE/SQL. Read the SQLINSTL file on your system for more information.	SQLINSTL file; <i>Communicator 3000 MPE/iX Release 5.5 (Non-Platform Software Release C.55.00)</i> , “ALLBASE/SQL Enhancements”; <i>ALLBASE/SQL Database Administration Guide</i> in “SQLINSTL” section of the “DBA Tasks and Tools” chapter.
GENPLAN on a section (Usability)	Obtains an access plan of a stored static query by specifying the module and section number. Changed syntax: GENPLAN.	<i>ALLBASE/SQL Reference Manual</i> , GENPLAN in “SQL Statements.”
POSIX support (Tools)	Starting with G.1, the ALLBASE/SQL preprocessor (PSQLCOB) supports preprocessing and generation of Microfocus COBOL source code under POSIX (Portable Operating System Interface).	<i>Communicator 3000 MPE/iX Release 5.5 (Non-Platform Software Release C.55.00)</i> , “ALLBASE/SQL Enhancements.”
Terminate a user’s connections (Connectivity)	Terminates one or more connections for a user. New syntax for TERMINATE USER: CID <i>ConnectionID</i> .	<i>ALLBASE/SQL Reference Manual</i> , TERMINATE USER in “SQL Statements.”
Run Queue Control for ALLBASE/NET (Connectivity)	Allows running HPDADVR in D queue for an MPE/iX session or HP-UX connection or C queue for an MPE/iX job connection. New environment variable: HPSQLJOBTYPE.	<i>Communicator 3000 MPE/iX Release 5.5 (Non-Platform Software Release C.55.00)</i> , “ALLBASE/SQL Enhancements.”
PC ODBC 16-bit and 32-bit support (Connectivity, Client/server)	ODBCLINK/SE allows connectivity to ALLBASE and IMAGE/SQL servers from a PC running MS Windows using ODBC.	<i>ODBCLINK/SE Reference Manual</i>
Year 2000 solution (Standards)	Provides the JCW HPSQLSPLITCENTURY to use in setting a value between 0 and 99. This value is used to change the century part of the DATE and DATETIME functions to override the default of 19.	“Date/Time Functions” in the “Expressions” chapter of the <i>ALLBASE/SQL Reference Manual</i>

G.0 Features

The following table highlights the new or changed functionality in release G.0, and shows you where each feature is documented.

New Features in ALLBASE/SQL Release G.0

Feature (Category)	Description	Documented in . . .
Stored procedures (Usability)	Provides additional stored procedure functionality for application programs. Allows declaration of a procedure cursor and fetching of multiple rows within a procedure to applications. New statement: ADVANCE. Changed syntax: CLOSE, CREATE PROCEDURE, DECLARE CURSOR, DESCRIBE, EXECUTE, EXECUTE PROCEDURE, FETCH, OPEN.	<i>ALLBASE/SQL Reference Manual</i> , “SQL Statements” and “Using Procedures” in “Constraints, Procedures and Rules;” <i>ALLBASE/SQL Advanced Application Programming Guide</i> , “Using Procedures in Application Programs.”
Case insensitivity (Usability)	Adds an optional attribute to the character and varchar type column attributes of tables. Allows search and compare of these columns in a case insensitive manner. Four new SQLCore data types are added. Changed syntax: ALTER TABLE, CREATE TABLE.	<i>ALLBASE/SQL Reference Manual</i> , “Comparison Predicate” in “Search Conditions;” CREATE TABLE in “SQL Statements.”
Support for 1023 columns (Usability)	Increases the maximum number of columns per table or view to 1023. Increases maximum sort columns and parameters in a procedure to 1023.	<i>ALLBASE/SQL Reference Manual</i> , CREATE TABLE and CREATE VIEW in “SQL Statements;” <i>ALLBASE/SQL Database Administration Guide</i> , “ALLBASE/SQL Limits” appendix.
ISQL HELP improvements (Usability)	Gives help for entire command instead of only the verb.	<i>ISQL Reference Manual for ALLBASE/SQL and IMAGE/SQL</i> , HELP in “ISQL Commands.”
EXTRACT command (Usability)	Extracts modules from the database and stores them in a module file. Allows for creation of a module file at any time based on the current DBEnvironment without preprocessing. New command: EXTRACT. Changed syntax: INSTALL.	<i>ISQL Reference Manual for ALLBASE/SQL and IMAGE/SQL</i> , “Using Modules” in “Using ISQL for Database Tasks;” EXTRACT, INSTALL in “ISQL Commands.”

New Features in ALLBASE/SQL Release G.0 (continued)

Feature (Category)	Description	Documented in . . .
New SQLGEN GENERATE parameters (Usability)	Generates SQL statements necessary to recreate modified access plans for module sections. New syntax for GENERATE: DEFAULTSPACE, MODOPTINFO, PARTITION, PROCOPTINFO, SPACEAUTH.	<i>ALLBASE/SQL Database Administration Guide</i> , "SQLGEN Commands" appendix.
Row level locking (Usability)	Permits multiple transactions to read and update a table concurrently because locking is done at row level. Since the transaction will obtain more locks, the benefits must be weighed against the costs. (Previously documented in an addendum after F.0 release.)	<i>ALLBASE/SQL Reference Manual</i> , "Concurrency Control through Locks and Isolation Levels;" <i>ALLBASE/SQL Database Administration Guide</i> , "Effects of Page and Row Level Locking" in "Physical Design."
Increased number of users (Usability)	Removes the limitation of 240 users supported by pseudotables. (Maximum is system session limits: 2000 on HP-UX; 1700 on MPE/iX.)	<i>ALLBASE/SQL Database Administration Guide</i> , "ALLBASE/SQL Limits" appendix.
POSIX support (Usability)	Improves application portability across MPE/iX and HP-UX. Enhances the ALLBASE/SQL preprocessors to run under POSIX (Portable Operating System Interface) on MPE/iX.	<i>ALLBASE/SQL Advanced Application Programming Guide</i> , "POSIX Preprocessor Invocation" in "Using the Preprocessor."
Application thread support (Performance, Usability)	Provides the use of threads in an application. Allows ALLBASE/SQL to be used in an application threaded environment on MPE/iX. Application threads are light weight processes that share some resources and last for the duration of a transaction. Threaded applications reduce the overhead of context switching and improve the performance of OpenTP applications.	<i>ALLBASE/SQL Advanced Application Programming Guide</i> , "Using the Preprocessor."

New Features in ALLBASE/SQL Release G.0 (continued)

Feature (Category)	Description	Documented in . . .
High Availability	Provides a collection of features to keep systems available nonstop including: Partial STORE and RESTORE, Partial rollforward recovery, DBEFiles in different groups (MPE/iX), detaching and attaching database objects, CHECKPOINT host variable, changing log files, console messages logged to a file, generating fewer log records by using TRUNCATE TABLE to delete rows, and new system catalog information. See the following features for new and changed syntax.	<i>ALLBASE/SQL Reference Manual</i> , “SQL Statements;” <i>ALLBASE/SQL Database Administration Guide</i> , “Maintaining a Nonstop Production System” in “Maintenance” chapter and “SQLUtil” appendix.
Partial rollforward recovery (High Availability)	Supports partial rollforward recovery through PARTIAL option on SETUPRECOVERY. Used to recover specific DBEFiles while allowing access to other DBEFiles.	<i>ALLBASE/SQL Database Administration Guide</i> , “Backup and Recovery” chapter and SETUPRECOVERY PARTIAL in “SQLUtil” appendix.
Partial STORE and RESTORE (High Availability)	Gives more flexibility in backup and recovery strategies by allowing partial store and restore of DBEFiles, DBEFileSets or combinations of both. See “New and changed SQLUtil commands for increased availability” later in this table.	<i>ALLBASE/SQL Database Administration Guide</i> , “Backup and Recovery” chapter and “SQLUtil” appendix.
DBEFile group change on MPE/iX (High Availability)	Manages DBEFiles so they can be placed in a particular group or on a particular volume (MPE/iX). Use either CREATE DBEFILE or MOVEFILE.	<i>ALLBASE/SQL Reference Manual</i> , CREATE DBEFile in “SQL Statements;” <i>ALLBASE/SQL Database Administration Guide</i> , “Maintaining a Nonstop Production System” in “Maintenance” chapter and MOVEFILE in “SQLUtil” appendix.
Detaching and attaching database objects (High Availability)	Detaches or attaches a DBEFile or DBEFileSet from the DBEnvironment. This is useful for data that is accessed infrequently such as tables containing historical data only. New SQLUtil commands: DETACHFILE, ATTACHFILE.	<i>ALLBASE/SQL Database Administration Guide</i> , “Maintaining a Nonstop Production System” in “Maintenance” chapter and DETACHFILE, ATTACHFILE in “SQLUtil” appendix.

New Features in ALLBASE/SQL Release G.0 (continued)

Feature (Category)	Description	Documented in . . .
New and changed SQLUtil commands for increased availability (High Availability)	Adds support for high availability and System Management Intrinsic. Intended for non-stop, continuously available operations. New SQLUtil commands: ATTACHFILE, CHANGELOG, DETACHFILE, RESTORE PARTIAL, STORE PARTIAL, STOREINFO, STOREONLINE PARTIAL, WRAPDBE. Modified SQLUtil commands: MOVEFILE, RESTORE, RESTORELOG, SHOWDBE, SETUPRECOVERY, STORE, STORELOG, STOREONLINE.	<i>ALLBASE/SQL Database Administration Guide</i> , "SQLUtil" appendix.
List files on backup device (High Availability)	Lists physical names of files stored on backup device with new SQLUtil command: STOREINFO.	<i>ALLBASE/SQL Database Administration Guide</i> , "Backup and Recovery" chapter and STOREINFO in "SQLUtil" appendix.
Log file improvements (High Availability)	Allows changing log files, switching of console messages to a file, and gives advance warning for log full. Increased maximum size of a single DBE log file to 4 gigabytes. A DBEnvironment can have up to 34 log files configured. Changed syntax: CHECKPOINT. New SQLUtil command: CHANGELOG.	<i>ALLBASE/SQL Reference Manual</i> , CHECKPOINT in "SQL Statements;" <i>ALLBASE/SQL Database Administration Guide</i> , "Maintaining a Nonstop Production System" in "Maintenance" chapter, CHANGELOG in "SQLUtil" appendix, and "ALLBASE/SQL Limits" appendix.
New SET SESSION and SET TRANSACTION statements (Standards, Performance)	Provides additional flexibility and improved performance. Allows setting and changing transaction and session attributes.	<i>ALLBASE/SQL Reference Manual</i> , SET SESSION and SET TRANSACTION in "SQL Statements."
FIPS flagger (Standards)	Meets Federal Information Processing Standard (FIPS) 127.1 flagger support. Flags non-standard statement or extension. Invoked with a flagger option in the preprocessor command line or the SET FLAGGER command in ISQL. Updatability rules are different when flagger is invoked. New syntax: DECLARE CURSOR, WHENEVER. Changes to C and COBOL host variable declaration.	<i>ALLBASE/SQL Reference Manual</i> , DECLARE CURSOR in "SQL Commands" and "Standards Flagging Support" appendix; <i>ALLBASE/SQL Advanced Application Programming Guide</i> , "Flagging Non-Standard SQL with the FIPS Flagger;" <i>ISQL Reference Manual for ALLBASE/SQL and IMAGE/SQL</i> , SET in "ISQL Commands."

New Features in ALLBASE/SQL Release G.0 (continued)

Feature (Category)	Description	Documented in . . .
Optimizer enhancement (Performance)	Uses a more efficient algorithm that significantly reduces the time to generate the access plan.	<i>ALLBASE/SQL Performance and Monitoring Guidelines</i> , "Optimization" in "Basic Concepts in ALLBASE/SQL Performance."
Access plan modification (Performance)	Allows modification of access plans for stored section to optimize performance. View the plan with SYSTEM.SETOPTINFO. New statement: SETOPT.	<i>ALLBASE/SQL Reference Manual</i> , SETOPT in "SQL Statements;" <i>ALLBASE/SQL Database Administration Guide</i> , SYSTEM.SETOPTINFO in "System Catalog."
Syntax added to disable access plan optimization (Performance, Usability)	Specifies that the optimization information in the module file is not to be used. Changed syntax: EXTRACT, INSTALL, VALIDATE.	<i>ALLBASE/SQL Reference Manual</i> , VALIDATE in "SQL Statements;" <i>ISQL Reference Manual for ALLBASE/SQL and IMAGE/SQL</i> , EXTRACT, INSTALL in "ISQL Commands."
Application Development Concurrency (Performance, Usability)	Provides enhancements to improve preprocessing performance when simultaneously accessed by multiple users. Page or row level locking on any system base table and processing without storing sections. See the related features in this table. New SQL parameter: SET DEFAULT DBEFileSet. SQL changed syntax: ALTER TABLE, GRANT, REVOKE, UPDATE STATISTICS. ISQL changed syntax: INSTALL. Changed SYSTEM and CATALOG view. New STOREDSECT tables. Special owners HPRDBSS and STOREDSECT. Changed syntax for Full Preprocessing Mode.	<i>ALLBASE/SQL Reference Manual</i> , "Names" and "SQL Statements;" <i>ALLBASE/SQL Advanced Application Programming Guide</i> , "Using the Preprocessor;" <i>ISQL Reference Manual for ALLBASE/SQL and IMAGE/SQL</i> , "ISQL Commands;" <i>ALLBASE/SQL Database Administration Guide</i> , "Database Creation and Security" and "System Catalog."
System Catalog tables (Performance)	Provides greater concurrency by allowing users to specify table, page, or row level locking of any system table owned by STOREDSECT through the ALTER TABLE statement.	<i>ALLBASE/SQL Reference Manual</i> , "Names;" <i>ALLBASE/SQL Database Administration Guide</i> , "System Catalog."
Preprocessors (Performance)	Allows optional specification of a DBEFileSet for storage of sections. Allows preprocessing without storing sections in DBEnvironment.	<i>ALLBASE/SQL Advanced Application Programming Guide</i> , "Using the Preprocessor."

New Features in ALLBASE/SQL Release G.0 (continued)

Feature (Category)	Description	Documented in . . .
I/O performance improvement (Performance)	Optimizes I/O for initial load, index build, serial scans, internal data restructuring, file activity, pseudo mapped files and temporary files. See the following features for new and changed syntax.	<i>ALLBASE/SQL Reference Manual</i> , "SQL Statements."
TRUNCATE TABLE statement (Performance)	Deletes all rows in a specified table leaving its structure intact. Indexes, views, default values, constraints, rules defined on the table, and all authorizations are retained. TRUNCATE TABLE is faster than the DELETE statement and generates fewer logs. New statement: TRUNCATE TABLE.	<i>ALLBASE/SQL Reference Manual</i> , TRUNCATE TABLE in "SQL Statements."
New scans (Performance)	Reads tables with a new parallel sequential scan. The tables are partitioned and files are read in a round robin fashion to allow OS prefetch to be more effective. Allows the I/O for a serial scan to spread across multiple disc drives.	<i>ALLBASE/SQL Performance and Monitoring Guidelines</i> , "Using Parallel Serial Scans" in "Guidelines on Query Design."
Load performance improvement (Performance)	Improves performance with new SET and SET SESSION attributes, a new binary search algorithm, and deferred allocation of HASH pages. New attributes for SET SESSION statement: FILL, PARALLEL FILL.	<i>ALLBASE/SQL Reference Manual</i> , SET SESSION in "SQL Statements."
ISQL enhanced to improve the performance of LOADs (Performance)	Uses new parameters of the ISQL SET command to set load buffer size and message reporting. Improves load performance. Choose a procedure, command file, or new ISQL command to set constraints deferred, lock table exclusively, and set row level DML atomicity. Changed syntax: SET (see the following feature).	<i>ISQL Reference Manual for ALLBASE/SQL and IMAGE/SQL</i> , SET in "ISQL Commands."

New Features in ALLBASE/SQL Release G.0 (continued)

Feature (Category)	Description	Documented in . . .
Modified SET options (Performance)	Provides better performance for LOADs and UNLOADs. Specify buffer size, status reporting for LOAD/UNLOAD or exclusive lock for data table. AUTOSAVE row limit increased to 2147483647. New and changed SET options: LOAD_BUFFER, LOAD_ECHO, AUTOLOCK, AUTOSAVE.	<i>ISQL Reference Manual for ALLBASE/SQL and IMAGE/SQL</i> , SET in “ISQL Commands;” <i>ALLBASE/SQL Performance and Monitoring Guidelines</i> , “Initial Table Loads” in “Guidelines on Logical and Physical Design.”
SQLMON (Tools)	Monitors the activity of ALLBASE/SQL DBEnvironment. Provides information on file capacity, locking, I/O, logging, tables, and indexes. Summarizes activity for entire DBEnvironment or focuses on individual sessions, programs, or database components. Provides read-only information.	<i>ALLBASE/SQL Performance and Monitoring Guidelines</i> , chapters 6-9.
Audit (Tools)	Provides a series of features to set up an audit DBEnvironment which generates audit log records that you can analyze with the new SQLAudit utility for security or administration. Includes the ability to set up partitions. See <i>ALLBASE/SQL Database Administration Guide</i> for SQLAudit commands. Modified statements: ALTER TABLE, CREATE TABLE, START DBE NEW, START DBE NEWLOG. New statements: CREATE PARTITION, DROP PARTITION, DISABLE AUDIT LOGGING, ENABLE AUDIT LOGGING, LOG COMMENT.	<i>ALLBASE/SQL Reference Manual</i> , “SQL Statements;” <i>ALLBASE/SQL Database Administration Guide</i> , “DBEnvironment Configuration and Security” chapter and “SQLAudit” appendix.
Wrapper DBEnvironments (Tools)	Creates a DBEnvironment to wrap around the log files orphaned after a hard crash of DBEnvironment. New SQLUtil command: WRAPDBE.	<i>ALLBASE/SQL Reference Manual</i> , “Wrapper DBEnvironments” in “Using ALLBASE/SQL;” <i>ALLBASE/SQL Database Administration Guide</i> , WRAPDBE in “SQLUtil.”
HP PC API is now bundled with ALLBASE/SQL.	PC API is an application programming interface that allows tools written with either the GUPTA or the ODBC interface to access ALLBASE/SQL and IMAGE/SQL from a PC.	<i>HP PC API User’s Guide for ALLBASE/SQL and IMAGE/SQL</i> .

New Features in ALLBASE/SQL Release G.0 (continued)

Feature (Category)	Description	Documented in . . .
Increased memory for MPE/iX (HP-UX shared memory allocation is unchanged) (Performance)	Increases memory up to 50,000 data buffer pages and 2,000 run time control block pages. Increases the limits significantly allowing allocation of enough data buffer pages to keep the entire DBEnvironment in memory if desired for performance.	<i>ALLBASE/SQL Reference Manual</i> , STARTDBE, STARTDBE NEW, and START DBE NEWLOG in "SQL Statements;" <i>ALLBASE/SQL Database Administration Guide</i> , "ALLBASE/SQL Limits" appendix.
ALLBASE/NET enhancements (Connectivity, Performance)	Improves performance of ALLBASE/NET, allows more client connections on server system, and reduces number of programs on MPE/iX.	<i>ALLBASE/NET User's Guide</i> , "Setting up ALLBASE/NET."
ALLBASE/NET commands and options for MPE/iX (Connectivity, Usability)	Adds option ARPA. Adds option NUMSERVERS to check status of listeners and number of network connections. Changed syntax: ANSTART, ANSTAT, ANSTOP. Changed NETUtil commands: ADD ALIAS, CHANGE ALIAS.	<i>ALLBASE/NET User's Guide</i> , "Setting up ALLBASE/NET" and "NETUtil Reference."
ALLBASE/NET and NetWare (Connectivity)	ALLBASE/NET listener for NetWare now works with the 3.11 version of Novell's NetWare for UNIX (HP NetWare/iX).	<i>ALLBASE/NET User's Guide</i> , "Setting up ALLBASE/NET."
Changed restrictions for executing NETUtil commands for MPE/iX (Connectivity, Usability)	Adds SM or AM (in the specified account) to MANAGER.SYS for adding, changing, or deleting users for MPE/iX.	<i>ALLBASE/NET User's Guide</i> , "Setting up ALLBASE/NET."
ARPA is only TCP/IP interface for data communication through ALLBASE/NET beginning with HP-UX 10.0 (Connectivity)	Remote database access applications that specify NS will not work if the client and/or server machine is an HP 9000 Series 700/800 running HP-UX 10.0 or greater. Server Node Name entry must be changed from NS node name to ARPA host name. For the NETUsers file, the "Client Node Name" must be changed from the NS node name to the ARPA host name. New NETUtil commands: MIGRATE USER, MIGRATE ALIAS.	<i>ALLBASE/NET User's Guide</i> , "Setting up ALLBASE/NET" and "NETUtil Reference."

Conventions

UPPERCASE In a syntax statement, commands and keywords are shown in uppercase characters. The characters must be entered in the order shown; however, you can enter the characters in either uppercase or lowercase. For example:

COMMAND

can be entered as any of the following:

command Command COMMAND

It cannot, however, be entered as:

comm com_mand comamnd

italics In a syntax statement or an example, a word in italics represents a parameter or argument that you must replace with the actual value. In the following example, you must replace *filename* with the name of the file:

COMMAND *filename*

punctuation In a syntax statement, punctuation characters (other than brackets, braces, vertical bars, and ellipses) must be entered exactly as shown. In the following example, the parentheses and colon must be entered:

(*filename*):(*filename*)

underlining Within an example that contains interactive dialog, user input and user responses to prompts are indicated by underlining. In the following example, yes is the user's response to the prompt:

Do you want to continue? >> yes

{ } In a syntax statement, braces enclose required elements. When several elements are stacked within braces, you must select one. In the following example, you must select either **ON** or **OFF**:

**COMMAND { ON }
 { OFF }**

[] In a syntax statement, brackets enclose optional elements. In the following example, **OPTION** can be omitted:

COMMAND *filename* [OPTION]

When several elements are stacked within brackets, you can select one or none of the elements. In the following example, you can select **OPTION** or *parameter* or neither. The elements cannot be repeated.

**COMMAND *filename* [OPTION
 parameter]**

Conventions (continued)

[...] In a syntax statement, horizontal ellipses enclosed in brackets indicate that you can repeatedly select the element(s) that appear within the immediately preceding pair of brackets or braces. In the example below, you can select *parameter* zero or more times. Each instance of *parameter* must be preceded by a comma:

[, *parameter*] [...]

In the example below, you only use the comma as a delimiter if *parameter* is repeated; no comma is used before the first occurrence of *parameter*:

[*parameter*] [, ...]





| ... | In a syntax statement, horizontal ellipses enclosed in vertical bars indicate that you can select more than one element within the immediately preceding pair of brackets or braces. However, each particular element can only be selected once. In the following example, you must select **A**, **AB**, **BA**, or **B**. The elements cannot be repeated.




$\left\{ \begin{array}{l} \mathbf{A} \\ \mathbf{B} \end{array} \right\} | \dots |$

... In an example, horizontal or vertical ellipses indicate where portions of an example have been omitted.

Δ In a syntax statement, the space symbol Δ shows a required blank. In the following example, *parameter* and *parameter* must be separated with a blank:

(*parameter*)Δ(*parameter*)

 The symbol  indicates a key on the keyboard. For example,  represents the carriage return key or  represents the shift key.

 *C*  followed by an uppercase character indicates a control character. For example, **Y** means that you press the control key and the Y key simultaneously.

Contents

1. DBA Tasks and Tools	
Tasks for the DBA	1-2
Creating Logical and Physical Objects	1-2
Starting and Stopping DBE Sessions	1-6
Establishing Multiple DBEnvironment Connections	1-6
Managing ALLBASE/SQL Logs	1-7
Defining Logs	1-7
Logs and Recovery	1-7
LOG FULL Condition	1-8
Maintaining Buffers for Data and Log Pages	1-8
Managing Transactions and Locks	1-10
Setting Parameters in the DBECon File	1-10
Monitoring the System Catalog	1-10
Managing Nonstop Production Systems	1-11
Backing Up and Restoring DBEnvironments	1-12
Migrating DBEnvironments Between Releases of ALLBASE/SQL	1-12
Tools for the DBA	1-13
Using ISQL	1-13
Using SQLUtil	1-13
Starting SQLUtil	1-14
JCWs Set by SQLUtil	1-14
Using SQLGEN	1-15
Starting SQLGEN	1-15
SQLGEN Commands	1-16
SQLGEN Schema Files	1-16
SQLGEN Conventions	1-16
Entering Object Names	1-17
JCWs Set by SQLGEN	1-18
Using SQLMigrate	1-18
Running SQLMigrate	1-19
Types of Forward Migration	1-19
Steps for Forward Migration	1-20
Steps for Backward Migration	1-22
JCWs Set by SQLMigrate	1-23
Using SQLINSTL	1-24
Using SQLMON	1-24
Using SQLVer	1-25
Using SQLAudit	1-27
SQLAudit Conventions	1-28
Understanding Audit Points	1-29
Example of Getting Audit Points	1-31
Establishing SQLAudit Log Locks	1-31

Performing an Audit	1-31
SQLAudit Result Files	1-33
Checking the Exit Status	1-33
System Administration for ALLBASE/SQL	1-34
Shared Memory Usage	1-34
Native Language Support	1-34
Network Administration	1-35
Using NETUtil	1-35

2. Logical Design

Identifying the Data for Tables	2-1
Normalizing the Data	2-4
First Normal Form	2-4
Second Normal Form	2-5
Third Normal Form	2-6
Arranging Data in Tables	2-7
Defining Tables	2-8
Planning Joins	2-8
Final Form of Sample Database Tables	2-9
Defining Columns	2-10
Defining Column Names	2-10
Defining Column Data Type	2-10
Defining Column Size	2-12
Defining Null Values For Columns	2-12
Designing Views	2-13
Designing Indexes	2-15
Determining Index Keys	2-16
How Index Keys are Used	2-17
Determining Index Type	2-17
Defining Unique Indexes	2-18
Defining Clustering Indexes	2-18
Designing Hash Structures	2-19
Understanding the Hash Function	2-19
Choosing Hash Keys	2-20
Choosing the Number of Primary Pages	2-20
Designing Integrity Constraints	2-21
Defining Security Levels	2-22
Authority Types	2-22
Special Authorities	2-23
Owner Authority	2-24
Table and View Authorities	2-24
RUN Authority	2-25
EXECUTE Authority	2-25
Space Authorities	2-25
Defining Authorization Groups	2-25
Determining Group Membership	2-26
Defining Classes	2-27
Differences Between Groups and Classes	2-27
Guidelines for Creating Classes	2-28
Defining the DBEnvironment Scope	2-28

3. Physical Design

Calculating Storage for Database Objects	3-1
Understanding DBEFile Characteristics	3-2
Calculating Storage for Tables	3-3
Calculating Row Length	3-4
Calculating Rows per Page	3-5
Calculating Number of Pages	3-5
Calculating Directory Overhead	3-6
Calculating Storage for Indexes	3-6
Calculating the Index Key Length	3-6
Calculating the Size of the Index Header	3-7
Calculating the Number of Rows per Leaf Page	3-7
Calculating the Number of Rows per Non-Leaf Page	3-7
Calculating the Number of Leaf Pages	3-8
Calculating the Number of Non-Leaf Pages	3-8
Calculating the Number of Directory Overhead Pages	3-9
Calculating Total Number of Index Pages	3-9
Arranging Tables and Indexes in DBEFileSets	3-9
Grouping Tables in DBEFileSets	3-9
Choosing DBEFile Types and Devices	3-11
Using a Single MIXED DBEFile	3-11
Using Separate DBEFiles for Tables and Indexes	3-11
Using Different Storage Devices	3-12
Estimating DBEFile Size	3-13
Calculating Storage for Hash Structures	3-14
Calculating Primary Pages	3-14
Allowing for Overflow	3-15
Calculating the Size of DBEFiles for Hash Structures	3-15
Allocating DBEFiles for Hash Structures	3-15
Mapping Logical Page Number to Physical File Location in Hash Structures	3-16
Calculating Storage for Integrity Constraints	3-16
Unique Constraints	3-16
Referential Constraints	3-17
Hashing on Constraints	3-17
Check Constraints	3-17
Calculating Storage for the System Catalog	3-17
Storage of Definitions for Newly Created Objects	3-18
Number of Columns in Tables	3-18
Definitions of Rules, Procedures, Constraints, and Views	3-18
Storage of Sections	3-19
Calculating Space Needed for Sections	3-19
Monitoring System Catalog Size	3-20
Calculating Storage for Logging	3-20
Understanding Log File Characteristics	3-20
Log Records and Transactions	3-21
Using Archive or Nonarchive Logs	3-21
Using Single or Dual Logging	3-22
Using Multiple Log Files	3-22
Sample Log Configuration	3-23
Disk Space for the Log	3-23
Determining the Number of Log Files	3-23

Estimating Log File Size	3-24
Example	3-24
Calculating Temporary Disk Space	3-26
Controlling the Use of Temporary Space	3-27
Estimating Shared Memory Requirements	3-27
Estimating Runtime Control Block Buffer Pages	3-28
Effects of Page and Row Level Locking	3-28
Running out of Shared Memory	3-29
Estimating Data Buffer Pages	3-30
Setting the Memory Resident Buffer Flag	3-30
Estimating Log Buffer Pages	3-31
Estimating the Number of Transactions	3-32
Implementing the Design	3-32
4. DBEnvironment Configuration and Security	
Required MPE/iX Capabilities	4-1
Using START DBE NEW	4-2
Supplying Startup Parameters with START DBE NEW	4-3
Log Files	4-4
Dual Logging	4-4
Archive Logging	4-5
Multiple Log Files	4-5
Specifying a Native Language Parameter	4-6
Looking at the DBEnvironment Elements	4-6
Examining MPE Files	4-6
Examining DBECon Parameters	4-6
Examining the System Catalog	4-7
Examining Log File Characteristics	4-9
Creating Audit DBEnvironments	4-10
Example of Setting Up an Audit DBEnvironment	4-11
Defining Additional Audit DBEnvironment Log Files	4-12
Disabling Audit Logging	4-12
Creating DBEFileSets and DBEFiles	4-13
Creating DBEFileSets	4-13
Assigning Default DBEFileSets	4-14
Creating DBEFiles	4-14
Adding DBEFiles to DBEFileSets	4-15
Allocating Expandable DBEFile Space	4-16
Allocating Expandable DBEFile Space in the SYSTEM DBEFileSet	4-17
Partial DBEFile Expansion	4-17
Obtaining Information about Expandable DBEFiles	4-18
Creating the DBEnvironment Security Scheme	4-18
Creating Authorization Groups	4-18
Managing Authorization Groups	4-18
Using DBA Authority	4-19
The DBECreator	4-19
DBA Functions	4-20
Granting Authorities to PUBLIC	4-20
Granting/Revoking CONNECT Authority	4-20
Granting/Revoking RESOURCE Authority	4-21
Granting/Revoking RUN Authority	4-22

Granting/Revoking EXECUTE Authority	4-22
Granting/Revoking SECTIONSPACE or TABLESPACE Authority	4-23
Verification of Authority	4-23
Managing DBEnvironment Sessions	4-23
Using Autostart	4-24
Using START DBE	4-25
Starting a DBE Session in Single-User Mode	4-25
Overriding DBECon Parameters	4-25
Starting DBE Sessions without Autostart	4-26
Rollback Recovery with START DBE	4-26
Connecting to a DBE	4-26
Terminating a DBE Session	4-26
Using RELEASE	4-26
Using STOP DBE	4-27
Using TERMINATE USER	4-27
Terminating Transactions and Queries	4-28
Setting Timeout Values	4-28
Remote Database Access	4-28

5. Database Creation and Security

Creating Tables	5-1
Table Type	5-2
Revoking and Granting Authorities on PUBLICROW and PUBLIC Tables	5-3
Altering Table Type	5-3
Owner and Table Name	5-3
Column Definition	5-4
Column Name and Data Type	5-4
Language Clause	5-5
DEFAULT Clause	5-5
Constraint Definitions	5-5
Unique Constraints	5-5
Referential Constraints	5-5
Check Constraints	5-6
DBEFileSet Name	5-6
Examining Table Attributes	5-7
Defining Partitions and Tables	5-8
Creating Views	5-9
Creating Hash Structures	5-10
Creating Indexes	5-11
Creating Procedures and Rules	5-12
Creating a Procedure	5-12
Creating a Rule	5-12
Creating the Database Security Scheme	5-13
Controlling Table Access with Authorities	5-13
Authorities for Single Users	5-14
Authorities for Groups	5-14
Creating Classes	5-14
Revoking Table and View Authorities	5-15
Controlling Table Access with Views	5-15
Using the GRANT OPTION Clause	5-17
Orphaned Privileges	5-18

Using the WITH GRANT OPTION Clause and Authorization Groups . . .	5-19
Creating a Database Schema	5-20
Loading Tables	5-20
Loading from an External File	5-21
Loading from an Internal File	5-21
Loading Tables with Constraints on Them	5-22
Loading Tables with Rules Built on Them	5-22
Using Command Files for Loading	5-22

6. Backup and Recovery

Choosing an Approach to Backup and Recovery	6-1
Choosing Nonarchive Logging	6-2
Choosing Archive Logging	6-2
Understanding Log File Types	6-3
Understanding the LOG FULL Condition	6-3
Using Single or Dual Logs	6-3
Using Multiple Log Files	6-3
Log Names and Numbers	6-4
Log File Names	6-4
Identifier Numbers	6-4
Sequence Numbers	6-4
Using Nonarchive Logs	6-4
Multiple Files in Nonarchive Mode	6-5
Using Archive Logs	6-6
Multiple Files in Archive Mode	6-7
Choosing Full or Partial Backup Procedures	6-9
Choosing Full or Partial Recovery Procedures	6-10
Backup and Recovery Procedures for Nonarchive Logging	6-11
Nonarchive Backup Procedures	6-11
Adding Files to the Nonarchive Log	6-11
Nonarchive Full Recovery Procedures	6-12
Nonarchive Partial Recovery Procedures	6-12
Backup and Recovery Procedures for Archive Logging	6-13
Online Backup Procedures in Archive Mode	6-13
Static Full or Partial Backup Procedures in Archive Mode	6-14
Adding Files to the Archive Log	6-15
Archive Recovery Procedures	6-15
Rollback Recovery	6-15
Full Rollforward Recovery	6-16
Partial Rollforward Recovery	6-17
ALLBASE/SQL Interface to True Online Backup	6-20
Managing Log Files	6-21
Monitoring the Log with SHOWLOG	6-21
Displaying Files in the Log	6-21
Log File Status Types	6-23
Displaying Available File Space	6-23
Using the CHECKPOINT command	6-23
Adding Log Files with ADDLOG	6-24
Storing Log Files with STORELOG	6-24
Rescuing Log Files with RESCUELOG	6-25
Restoring Log Files with RESTORELOG	6-25

Purging Log Files with PURGELOG	6-26
Moving Log Files with MOVELOG	6-26
Starting a New Log	6-27
Monitoring the Log with SQLMON	6-28
Setting up a Wrapper DBEnvironment	6-28
Selecting Valid Log Files with SHOWLOG	6-29
Single Logs	6-29
Dual Logs	6-30
Selecting Log Files when the DBECon File is Inaccessible	6-31
Creating a DBEnvironment	6-31
Wrapping the DBEnvironment Around the Log Files	6-32
Example of Setting Up a Wrapper DBE	6-33

7. Maintenance

Using Simple and Complex Maintenance Operations	7-1
Maintaining the DBEnvironment	7-2
Adjusting Startup Values	7-2
Determining Behavior of Rules in a DBEnvironment Session	7-5
Updating System Catalog Statistics	7-6
Changing System Table Lock Types	7-7
Managing DBEFiles and DBEFileSets	7-7
Adding a New DBEFile	7-8
Changing DBEFile Type	7-8
From TABLE or INDEX to MIXED	7-8
From MIXED to TABLE or INDEX	7-9
Dropping a DBEFile	7-9
Maintaining Tables	7-10
Changing a Table's Locking Behavior	7-10
Dropping Tables	7-10
Adding Columns	7-11
Deleting Columns	7-11
Removing Rows from a Table	7-12
Merging Tables	7-13
Dividing Tables	7-14
Renaming Tables or Columns	7-14
Dropping and Recreating Hash Structures	7-15
Maintaining Indexes	7-15
Monitoring Index Space	7-15
Monitoring the Cluster Count	7-16
Dropping and Recreating an Index	7-16
Reloading a Table to Improve Index Performance	7-17
Altering the Index Key	7-17
Maintaining Constraints	7-17
Adding Constraints	7-17
Dropping Constraints	7-18
Maintaining Rules and Procedures	7-18
Granting and Revoking Procedure Authorities	7-18
Examining the Inventory of Rules and Procedures	7-19
Dropping and Recreating Rules and Procedures	7-19
Validating Procedure Sections	7-20
Maintaining Sets of Interrelated Objects	7-20

Maintaining Applications	7-21
Invalidation and Revalidation of Sections	7-21
Information in the System Catalog on Validity of Sections	7-21
Monitoring File Space for Modules and Sections	7-23
Causes for Invalidation of Sections	7-23
Avoiding the Need for Re-Preprocessing	7-23
Determining Available Space for Sections	7-24
Determining Number of Sections in the DBEnvironment	7-24
Module Related Authorities	7-24
Sharing Modules Between DBEnvironments	7-25
Dropping Modules	7-26
Maintaining a Nonstop Production System	7-26
DBEFiles in Different Groups and Volumes	7-27
Moving DBEFiles to Different Groups	7-27
Detaching and Attaching Database Files	7-27
Using a Host Variable with the CHECKPOINT Statement	7-28
Using Console Message Files	7-28
Making Changes to a New Log File	7-29
Checking the System Catalog	7-29
Maintaining Security	7-30
Disabling Data Definition	7-30
Judging Maintenance Expenses	7-31
Cleaning Up after Abnormal Termination	7-31

8. System Catalog

Views owned by SYSTEM and CATALOG	8-1
Summary of System Catalog Views by Function	8-2
Using the System Catalog	8-4
System Catalog Views	8-5
Using UPDATE STATISTICS on System Views	8-5
Locking of the System Catalog	8-6
Storedsect.System	8-7
Storedsect.DBFileSetName	8-8
System.Account	8-9
System.Call	8-10
System.CheckDef	8-14
System.Colauth	8-15
System.Coldefault	8-18
System.Column	8-19
System.Constraint	8-23
System.Constraintcol	8-25
System.ConstraintIndex	8-26
System.Counter	8-29
System.DBFile	8-31
System.DBFileSet	8-34
System.Group	8-35
System.Hash	8-36
System.Imagekey	8-38
System.Index	8-40
System.Installauth	8-44
System.Modauth	8-45

System.Paramdefault	8-46
System.Parameter	8-47
System.Partition	8-50
System.Plan	8-51
System.Procauth	8-53
System.Procedure	8-54
System.ProcedureDef	8-56
System.ProcResult	8-57
System.Rule	8-59
System.RuleColumn	8-61
System.RuleDef	8-62
System.Section	8-64
System.Setoptinfo	8-66
System.Spaceauth	8-67
System.Spacedefault	8-68
System.Specauth	8-69
System.Tabauth	8-71
System.Table	8-74
System.TempSpace	8-77
System.TPIndex	8-78
System.Transaction	8-80
System.User	8-82
System.ViewDef	8-83

A. ALLBASE/SQL Limits

B. Authorities Required by ALLBASE/SQL Statements

C. SQL Syntax Summary

D. ISQL Syntax Summary

E. Locks Held on the System Catalog by SQL Statements

F. SQLUtil

ADDLOG	F-2
ALTDDBE	F-4
ATTACHFILE	F-7
CHANGELOG	F-9
DETACHFILE	F-10
ENDRECOVERY	F-12
EXIT	F-14
HELP	F-15
MOVEFILE	F-17
MOVELOG	F-19
PURGEALL	F-21
PURGEDBE	F-22
PURGEFILE	F-23
PURGELOG	F-25
QUIT	F-26
RECOVERLOG	F-27

RESCUELOG	F-29
RESTORE	F-30
RESTORELOG	F-31
RESTORE PARTIAL	F-35
SET	F-37
SETDBEMAJNT	F-40
SETUPRECOVERY	F-41
SETUPRECOVERY PARTIAL	F-44
SHOWACCESS	F-47
SHOWDBE	F-48
SHOWDBE-ALL	F-50
SHOWDBE-EXIT	F-51
SHOWDBE-HELP	F-52
SHOWDBE-LANG	F-54
SHOWDBE-MAINT	F-55
SHOWDBE-QUIT	F-56
SHOWDBE-STARTPARMS	F-57
SHOWLOG	F-58
SHOWSET	F-61
STORE	F-62
STOREINFO	F-64
STORELOG	F-65
STOREONLINE	F-67
STOREONLINE PARTIAL	F-69
STORE PARTIAL	F-72
SYSTEM	F-75
WRAPDBE	F-76

G. SQLGEN

EDITOR	G-2
EXIT	G-3
GENERATE ALL	G-4
GENERATE DBE	G-9
GENERATE DBEFILES	G-10
GENERATE DEFAULTSPACE	G-12
GENERATE GROUPS	G-13
GENERATE INDEXES	G-15
GENERATE INSTALLAUTH	G-17
GENERATE LOAD	G-18
GENERATE MODAUTH	G-20
GENERATE MODOPTINFO	G-22
GENERATE PARTITION	G-24
GENERATE PROCAUTH	G-25
GENERATE PROCEDURES	G-27
GENERATE PROCOPTINFO	G-29
GENERATE RULES	G-31
GENERATE SPACEAUTH	G-33
GENERATE SPECAUTH	G-34
GENERATE STATISTICS	G-35
GENERATE TABAUTH	G-37
GENERATE TABLES	G-39

GENERATE TEMPSPACES	G-42
GENERATE VIEWAUTH	G-43
GENERATE VIEWS	G-45
HELP	G-47
RELEASE	G-48
SET ECHO_ALL OFF	G-49
SET ECHO_ALL ON	G-50
SET EDITOR	G-51
SET EXIT_ON_DBERR OFF	G-52
SET EXIT_ON_DBERR ON	G-53
SET SCHEMA	G-54
STARTDBE	G-55
:	G-56

H. SQLMigrate

ADD DBEFILE	H-2
CREATE DBEFILE	H-3
EXIT	H-4
HELP	H-5
MIGRATE	H-6
PREVIEW	H-9
QUIT	H-12
REPAIR	H-13
SET	H-14
SHOW 'DBEnvironmentName' VERSION	H-16
SHOW VERSIONS	H-17
:	H-18

I. SQLAudit

AUDIT	I-2
EDITOR	I-5
EXIT	I-6
GET AUDITPOINT	I-7
HELP	I-9
LOCK AUDITPOINT	I-10
MODIFY AUDITPOINT	I-12
QUIT	I-14
SET	I-15
SET DBENVIRONMENT	I-16
SET ECHO_ALL	I-17
SET EDITOR	I-18
SET EXIT_ON_DBERR	I-19
SET RECOVERFILE	I-20
SHOW AUDITPOINT	I-21
UNLOCK AUDITPOINT	I-22

Index

Figures

1-1. Tables, DBEFiles, and DBEFileSets	1-3
1-2. Databases and DBEFileSets	1-4
1-3. Elements of an ALLBASE/SQL DBEnvironment	1-5
1-4. The Relationship between Files and Buffers	1-9
2-1. Identifying Data Categories	2-2
2-2. Entities, Attributes, and Keys	2-3
2-3. First Normal Form: Removing Repeating Groups	2-5
2-4. Second Normal Form: Establishing Dependency	2-6
2-5. Third Normal Form: Removing Transitive Dependencies	2-7
2-6. Common Columns for Joins	2-9
2-7. B-Tree Index Design	2-15
2-8. Relationship among Authorities	2-23
2-9. Authorization Group Chain	2-27
2-10. DBEnvironment used by Integrated Peripherals, Inc.	2-29
3-1. DBEFiles in DBEFileSets	3-2
3-2. Data Stored in DBEFiles within a DBEFileSet	3-3
3-3. DBEFileSets in the Sample DBEnvironment	3-10
3-4. Table and Index DBEFiles in the OrdersFS DBEFileSet	3-12
3-5. DBEFiles, DBEFileSets, and Direct-Access Storage	3-13
4-1. The Sample DBEnvironment Immediately After Configuration	4-3
4-2. SQL Commands for Authorization Group Management	4-19
4-3. Autostart and User Mode Dependencies	4-24
5-1. Views Restricting Access	5-16
5-2. Example Database Security Scheme	5-18
6-1. Nonarchive Log	6-5
6-2. Archive Log	6-6
6-3. Log Switching in Archive Mode	6-8

Tables

1-1. JCWs Set by SQLUtil	1-15
1-2. SQLGEN General Conventions	1-17
1-3. SQLGEN Name Entry Conventions	1-18
1-4. JCWs Set by SQLGEN	1-18
1-5. JCWs Set by SQLMigrate	1-24
1-6. SQLVERERR JCW	1-25
1-7. SQLAudit General Conventions	1-29
2-1. Sample Database Tables	2-10
2-2. Column Attributes for Two Tables	2-13
3-1. Data Type Storage Requirements	3-4
3-2. Page Requirements for Table Data	3-14
3-3. Page Requirements for Index Data	3-14
3-4. Logical Page Number and DBEFile Location in Hash Structure	3-16
3-5. Maximum Numbers of Locks Obtained at Different Granularities	3-28
4-1. DBECon Default Startup Parameters	4-2
5-1. Table and View Authorities	5-13
6-1. SQLMON Log Monitoring Tasks	6-28
6-2. Example Log File Names and Sequence Numbers	6-30
7-1. DBECon Parameters	7-3
8-1. System Catalog Views by Function	8-2
8-2. System.Account	8-9
8-3. System.Call	8-10
8-4. System.CheckDef	8-14
8-5. System.Colauth	8-16
8-6. System.Coldefault	8-18
8-7. System.Column	8-19
8-8. System.Constraint	8-23
8-9. System.ConstraintCol	8-25
8-10. System.ConstraintIndex	8-27
8-11. System.Counter	8-29
8-12. System.DBEFile	8-31
8-13. System.DBEFileSet	8-34
8-14. System.Group	8-35
8-15. System.Hash	8-36
8-16. System.Imagekey	8-38
8-17. System.Index	8-40
8-18. System.Installauth	8-44
8-19. System.Modauth	8-45
8-20. System.Paramdefault	8-46
8-21. System.Parameter	8-47
8-22. SYSTEM.PARTITION	8-50
8-23. System.Plan	8-51
8-24. System.Procauth	8-53

8-25. System.Procedure	8-54
8-26. System.ProcedureDef	8-56
8-27. System.ProcResult	8-57
8-28. System.Rule	8-59
8-29. System.RuleColumn	8-61
8-30. System.RuleDef	8-62
8-31. System.Section	8-64
8-32. System.Setoptinfo	8-66
8-33. System.Spaceauth	8-67
8-34. System.Spacedefault	8-68
8-35. System.Specauth	8-69
8-36. System.Tabauth	8-71
8-37. System.Table	8-74
8-38. System.TempSpace	8-77
8-39. System.TPIndex	8-78
8-40. System.Transaction	8-80
8-41. System.User	8-82
8-42. System.ViewDef	8-83
A-1. System Control Limits	A-1
A-2. Logical Data Limits	A-1
A-3. Space Management Limits	A-2
B-1. Authorities Required By ALLBASE/SQL Statements	B-1
E-1. Mapping the System Views to the Base System Tables	E-2
E-2. Locks Held on the System Catalog by SQL Statements	E-4

DBA Tasks and Tools

This guide shows how to perform database administration tasks for your ALLBASE/SQL relational database management system. The present chapter introduces the following topics:

- Tasks for the DBA
- Tools for the DBA
- System administration for ALLBASE/SQL

Chapters 1 through 3 describe the concepts behind ALLBASE/SQL database administration tasks. Subsequent chapters describe DBA concepts and tasks in detail. Experienced database administrators may wish to turn directly to Chapter 4, “DBEnvironment Configuration and Security.” Before you read this manual, you should be familiar with the material in the *ALLBASE/SQL Reference Manual* and the *ISQL Reference Manual for ALLBASE/SQL and IMAGE/SQL*.

Most of the examples presented in this guide are based on a DBEnvironment named PartsDBE, which is provided with the ALLBASE/SQL product. This DBEnvironment contains the sample database PurchDB, which includes several tables: Parts, SupplyPrice, Vendors, Orders, OrderItems, and Inventory. Refer to the “Sample DBEnvironment” appendix in the *ALLBASE/SQL Reference Manual* for complete details.

This guide assumes that a single individual is carrying out the tasks of **database administrator**, also known as the DBA. The DBA uses SQL statements, usually through ISQL, to create and maintain ALLBASE/SQL DBEnvironments. The DBA also uses a variety of utility programs, explained later in this chapter, to perform specific kinds of maintenance. If you create and maintain your own DBEnvironment, then you are its **DBECreator** as well as its DBA.

The individual who serves as database administrator normally needs **DBA authority** for the DBEnvironments that are to be administered. If you are the DBECreator of a DBEnvironment, you automatically receive DBA authority in that DBEnvironment, and you can therefore use almost all the commands described in this manual.

Tasks for the DBA

The database administrator is responsible for the overall operation of DBEnvironments. This includes:

- Creating logical and physical objects.
- Starting and stopping DBE sessions.
- Establishing multiple DBEnvironment connections.
- Managing ALLBASE/SQL logs.
- Maintaining buffers for data and log pages.
- Managing transactions and locks.
- Setting parameters in the DBECon file.
- Monitoring the system catalog.
- Managing nonstop production systems.
- Backing up and restoring DBEnvironments.
- Migrating DBEnvironments between releases of ALLBASE/SQL.

Creating Logical and Physical Objects

The largest physical unit in ALLBASE/SQL is the DBEnvironment, which is a collection of files for one or more logical databases. A DBEnvironment is the maximum unit of transaction scope and recovery. The DBEnvironment contains:

- DBEFiles for storage of data and index pages
- a DBEFile for system information
- a DBECon file containing startup parameters for the DBE
- log files.

A **DBEFile** is an MPE/iX file that can be associated with a DBEFileSet. ALLBASE/SQL database tables are stored in one or more DBEFiles. Indexes for a table are also stored in DBEFiles. Figure 1-1 illustrates the relationship among tables, indexes, DBEFileSets, and DBEFiles.

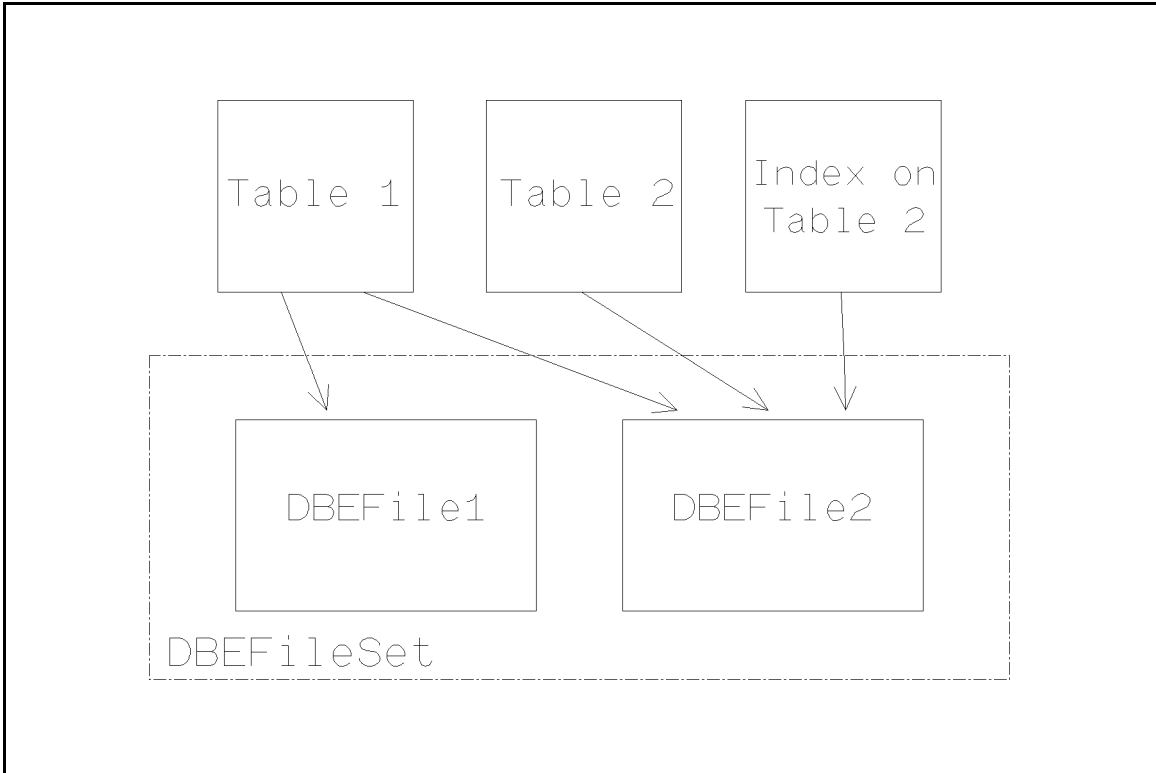


Figure 1-1. Tables, DBEFiles, and DBEFileSets

A **DBEFileSet** is a collection of DBEFiles containing data for one or more tables. The tables and indexes associated with a DBEFileSet do not have to be for the same database.

Figure 1-2 illustrates that while a DBEFileSet can contain data for all the tables in a database, a DBEFileSet can also contain data for some of the tables in a database, or for tables in more than one database.

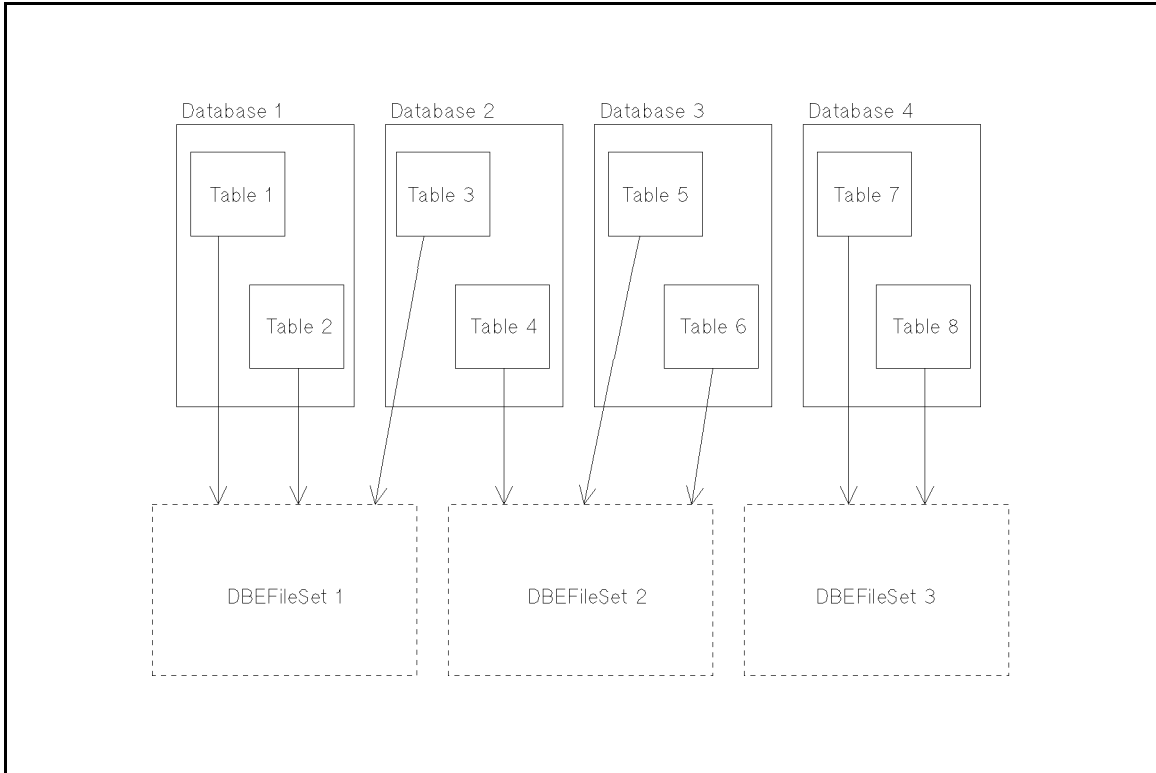


Figure 1-2. Databases and DBEFileSets

Thus, DBEFileSets offer a way to allocate data storage independently of how users think about the data. During physical design and database creation, the DBA determines space requirements for the tables and indexes and creates DBEFiles and DBEFileSets to accommodate them.

A **DBEnvironment**, illustrated in Figure 1-3, houses the DBEFiles for one or more ALLBASE/SQL databases.

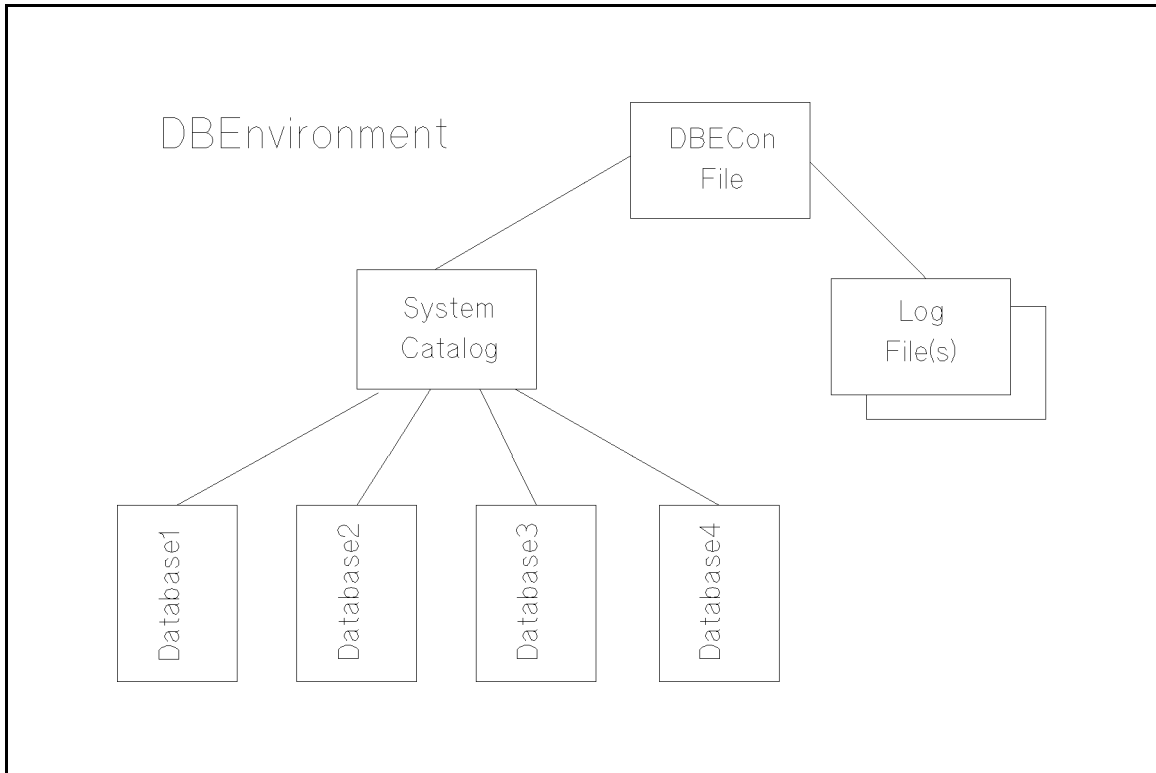


Figure 1-3. Elements of an ALLBASE/SQL DBEnvironment

The DBEnvironment also contains the following entities, which contain information for **all** databases in the DBEnvironment:

- **A DBECon file.** This file contains information about the DBEnvironment's configuration, such as the size of various buffers and other startup parameters. The name of the DBECon file is the same as the name of the DBEnvironment.
- **Log files.** A log file contains a record of DBEnvironment changes. ALLBASE/SQL uses log files to undo (**roll back**) or redo (**roll forward**) changes made in the DBEnvironment. You can add additional log files as needed. In the case of dual logging, two sets of log files are maintained.
- **A system catalog.** The system catalog is a collection of tables and views that contain data describing DBEnvironment structure and activity. The parts of the system catalog necessary for DBEnvironment startup reside in a DBEFile known as **DBEFile0**. All system catalog DBEFiles are associated with a DBEFileSet called **SYSTEM**.

The DBA determines the configuration, the startup parameters, the name and size of DBEFile0, the name and size of data and index DBEFiles, and the name and size of initial log files before configuring the DBEnvironment. ALLBASE/SQL uses defaults for any of these values if a choice is not made.

Starting and Stopping DBE Sessions

The DBA controls access to each DBEnvironment by turning the AUTOSTART flag ON or OFF and by issuing START DBE and STOP DBE commands. Use the SQLUtil ALTDBE command (described later in this chapter) to turn AUTOSTART mode ON or OFF.

When users have the proper authorization, they access a database by first **connecting** to the DBEnvironment in which the database resides. To connect, you use a CONNECT statement, as in the following example:

```
isql=> CONNECT TO 'PartsDBE';
```

Following a successful CONNECT, ALLBASE/SQL establishes a **DBE session** for the user, which allows SQLCore to process commands.

If AUTOSTART is OFF, the DBA must start the DBEnvironment using the START DBE statement:

```
isql=> START DBE 'PartsDBE';
```

Following this statement, the DBEnvironment remains available to users until the DBA issues the STOP DBE statement.

A DBEnvironment can be started in one of two user modes:

- In **single-user mode**, only one user or program can access a DBEnvironment at a time. Single-user mode is employed by the DBA to perform maintenance and restructuring tasks.
- In **multiuser mode**, more than one user and/or program can access a DBEnvironment at a time. Multiuser mode is for production, in which the DBA's responsibility is to maintain the DBEnvironment for the multiple users that access it.

You can access a DBEnvironment interactively or through an application program.

Establishing Multiple DBEnvironment Connections

Users can access multiple DBEnvironments at the same time. Each connection is assigned a different connection name, as in the following:

```
CONNECT TO 'PartsDBE' AS 'DBE1'  
CONNECT TO 'MusicDBE' AS 'DBE2'
```

The SET CONNECTION statement establishes the current DBEnvironment connection:

```
SET CONNECTION 'DBE1'
```

To support the use of multiple connections, the DBA should set default and maximum user timeout values in each DBEnvironment. For additional information, see the section “Using Multiple Connections and Transactions with Timeouts” in the *ALLBASE/SQL Reference Manual*.

Managing ALLBASE/SQL Logs

For each DBEnvironment you create, ALLBASE/SQL automatically starts a log containing **log records** which reflect the DBEnvironment's activities. ALLBASE/SQL uses **writeahead** logging. This means that actual changes are not made to the DBEnvironment until the changes are first written to the log files as log records. Log records enable an ALLBASE/SQL DBEnvironment to roll back transactions and to **recover** in the event of a soft crash or a media failure.

For additional security, you can specify dual logging, which means that ALLBASE/SQL maintains two identical logs. If there is a write or read failure in one log, the other will then be used. ALLBASE/SQL has two log modes: nonarchive and archive. **Nonarchive mode**, the default, permits only rollback recovery. **Archive mode**, which you enable with the SQLUtil STOREONLINE command, permits both rollback and rollforward recovery (that is, recovery from an earlier stored version of the DBEnvironment).

Defining Logs

You choose single or dual logging initially in the START DBE NEW statement. You also determine the size and location of initial logs using the LOG clause of this statement:

```
LOG DBEFILE DBELog1ID [AND DBELog2ID] WITH PAGES = DBELogSize, NAME =  
'SystemFileName1' [AND 'SystemFileName2' ]
```

You can define a new log with the START DBE NEWLOG statement. This lets you change the log file name and size, turn archive mode on or off, and change from single to dual logging and back.

Detailed information about START DBE NEW and START DBE NEWLOG appears in the “DBEnvironment Configuration and Security” chapter. The syntax of both statements appears in the “SQL Statements” chapter of the *ALLBASE/SQL Reference Manual*.

The DBA must also manage the size, number, and location of all ALLBASE/SQL logs. This is done through using the SQLUtil log commands:

```
ADDLOG          MOVELOG  
PURGELOG        RESCUELOG  
RESTORELOG      SHOWLOG  
STORELOG
```

SQLUtil commands are fully explained in the “SQLUtil” appendix. Detailed information about managing logs appears in the “Backup and Recovery” chapter.

Logs and Recovery

After a system failure (other than a media failure), all the data within a DBEnvironment is automatically recovered to a consistent state the next time the DBEnvironment is started. Changes performed by any transactions that were incomplete at failure time are rolled back. Changes performed by transactions that were complete at failure time are written to the data files on disk from the log. In the case of a media failure, you must initiate a manual recovery of the DBEnvironment from backups. This process is described fully in the “Backup and Recovery” chapter.

Rollback recovery is an automatic feature of both nonarchive and archive log modes. Rollback recovery has two purposes:

- to let users roll back the effects of a transaction with the `ROLLBACK WORK` statement
- to let ALLBASE/SQL automatically roll back the DBEnvironment to a consistent state after a soft crash, and whenever the `START DBE` statement is executed.

A **soft crash** is a program abort or a system failure that does not damage the storage media. ALLBASE/SQL always does rollback recovery when a DBEnvironment starts up, and this ensures that whether or not there was a crash, all complete transactions are made permanent to disk, and all incomplete transactions are undone (rolled back).

Rollforward recovery is possible only with archive log mode. It allows you to reconstruct a DBEnvironment from a backup copy and one or more stored archive log files in the event of a hard crash. A **hard crash** is a failure, such as a disk head crash, that damages files on disk. Complete details about rollforward recovery from archive log files is presented in the “Backup and Recovery” chapter.

LOG FULL Condition

Under some circumstances, the log can become full, which means that no additional transactions can be logged until log space is provided. When a LOG FULL condition arises, ALLBASE/SQL performs a special rollback operation which rolls back all transactions and issues the following error message:

```
Log Full. (DBERR 14046)
```

To avoid a LOG FULL condition, make sure there are enough log files available for all the concurrent transactions running on your system. Refer to the “Backup and Recovery” chapter for additional information about managing log files.

Note When LOG FULL arises, in most cases *all* transactions are rolled back. This includes transactions that have performed no updates.

Maintaining Buffers for Data and Log Pages

ALLBASE/SQL uses two kinds of buffers to hold data as it is passed between your applications and the operating system:

- the **log buffer**, which holds log records that reflect changes made to data pages by active transactions.
- the **data buffer**, which holds DBEFile pages from tables and indexes currently being accessed.

The DBA must decide on the appropriate number of log and data buffer pages for the system. Buffers are **flushed** (written to disk) only at specific times. Once they are flushed, the buffers can be used by other transactions.

ALLBASE/SQL flushes log buffers to the log file when one of the following occurs:

- a `COMMIT WORK` ends a transaction that modified the DBEnvironment.
- the data buffer is full, so changes to the DBEnvironment must be written to disk to free data buffer space. Log buffers must also be flushed because of writeahead logging.

- the log buffer is full, so changes to the DBEnvironment must be written to disk to free log buffer space.
- when a checkpoint is taken by means of a user's CHECKPOINT statement or by ALLBASE/SQL internally. The checkpoint writes a **system checkpoint record** to the log, and it flushes the log buffer as well as the data buffer.

ALLBASE/SQL flushes pages from the data buffer to DBEFiles when one of the following occurs:

- the data buffer is full, so individual changed pages are written to disk to free data buffer space.
- when a checkpoint is taken by means of a user's CHECKPOINT statement or by ALLBASE/SQL internally. The checkpoint writes a system checkpoint record to the log, and it flushes the log buffer as well as the data buffer.

Figure 1-4 shows the relationship between files and buffers.

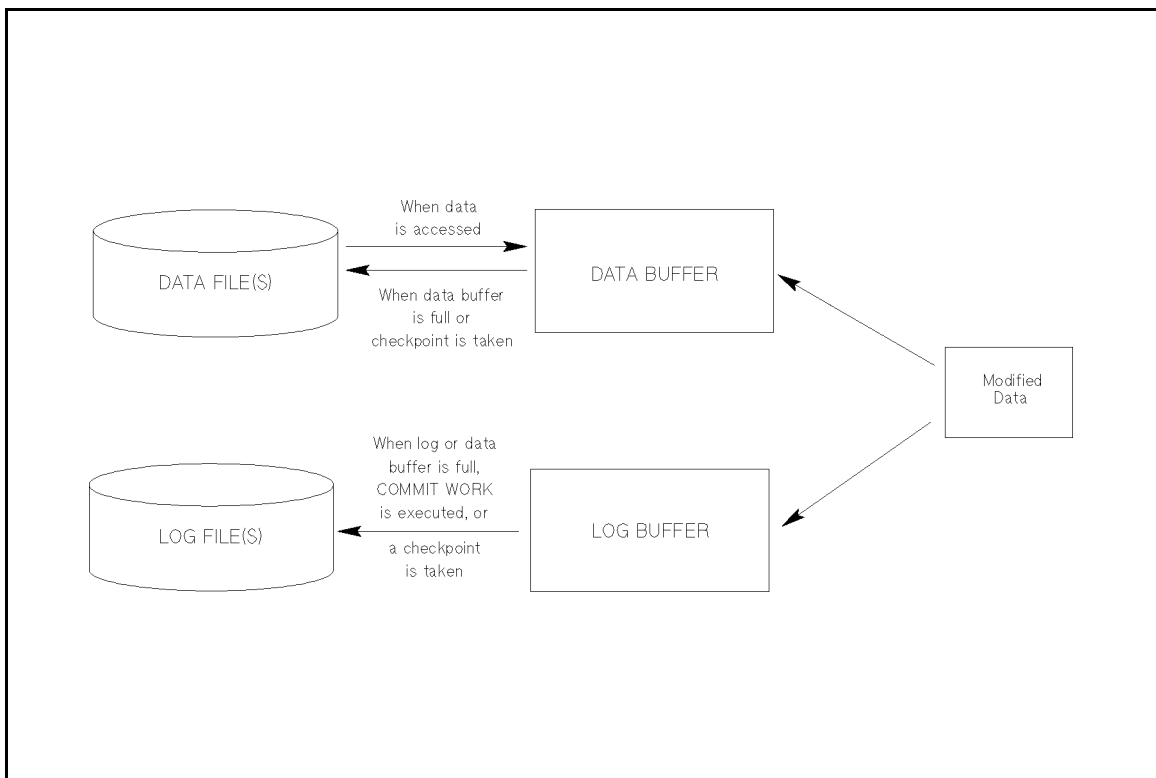


Figure 1-4. The Relationship between Files and Buffers

As you see from the figure, data is transferred from DBEFiles on disk and loaded into the data buffer when an SQL statement requiring data is executed. When user or system data must be changed, log records are first written to the log buffer, and then DBEFile pages in the data buffer are modified. If the data buffer is full or if a checkpoint is taken, some data and log buffer pages will be flushed to disk.

Managing Transactions and Locks

Within a DBEnvironment, ALLBASE/SQL manipulates data in units of recoverable work known as transactions. A **transaction** is one or more SQL statements that together perform a unit of work on one or more databases in a DBEnvironment. A transaction begins with an SQL statement and ends with either a COMMIT WORK statement or a ROLLBACK WORK statement. All work done within a transaction can be made permanent (committed) or undone (rolled back).

Transactions acquire locks, which regulate concurrent access to the DBEnvironment. The DBA keeps track of the locking behavior of the DBEnvironment, monitoring the number of lock waits and deadlocks, and choosing approaches to locking and isolation levels that can minimize deadlock while obtaining the greatest system throughput. Refer to the chapter “Concurrency Control through Locks and Isolation Levels” in the *ALLBASE/SQL Reference Manual* for basic information about transactions and locking. Additional information is found in the *ALLBASE/SQL Performance and Monitoring Guidelines*.

Logging and recovery are also performed in terms of transactions. For more information, refer to the “Backup and Recovery” chapter.

Setting Parameters in the DBECon File

The DBE configuration file (DBECon file) contains startup parameters for each DBEnvironment. The DBA adjusts these parameters as needed as the DBEnvironment is developed, put into production, and modified. Some DBECon parameters are quantitative:

- Run time control blocks
- Maximum transactions
- Number of log and data buffer pages
- Timeout values

Others are ON/OFF:

- DDL Enabled
- Autostart mode
- Single or multi startup
- Archive mode

All these are useful in tuning the performance of the DBE for your specific installation's needs.

The “DBEnvironment Configuration and Security” chapter describes the initial state of each DBECon parameter at START DBE NEW time, and the “Maintenance” chapter shows how to alter DBECon parameters using SQLUtil.

Monitoring the System Catalog

The system catalog contains information about all the objects stored in the DBEnvironment and about ongoing processes while the DBEnvironment is active. The DBA can monitor this information to determine when it is necessary to add objects, remove them, add file space, reallocate buffer space, or adjust other parameters.

The system catalog contains information about:

- What tables, views, and indexes exist.
- How much DBEFile space is available.

1-10 DBA Tasks and Tools

- The size of rows and columns in tables.
- The cluster count of indexes.
- Which transactions are waiting for locks to be released.
- The names of users on the system and their session ids.
- View and table definitions.

The “Maintenance” chapter shows how to perform many useful maintenance tasks using system catalog information. The “System Catalog” chapter describes each view and pseudotable in the system catalog with examples of its contents. Much of the information contained in the system catalog is displayed by SQLMON, the online monitoring tool. See the *ALLBASE/SQL Performance and Monitoring Guidelines* for more information.

Managing Nonstop Production Systems

A collection of ALLBASE/SQL features can be used to help keep systems available with as few stops as possible. These features also help users who have large databases servicing a large number of concurrent sessions and requiring lengthy backup and recovery times. Users who experience a large amount of Online Transaction Processing (OLTP) and have their systems run for significant periods without an operator present will also find these features useful for performing tasks when operator time permits.

These features are implemented through SQLUtil commands and ALLBASE/SQL statements. The commands and statements comprise the tasks of database creation, maintenance, and recovery.

The SQLUtil features are implemented in the following commands:

STORE or STOREONLINE PARTIAL	Stores parts of a DBEnvironment (DBEFiles or DBEFileSets).
STOREINFO	Displays DBEFile information stored on a backup device.
RESTORE PARTIAL	Restores a set of DBEFiles.
SETUPRECOVERY PARTIAL	Rolls forward a set of DBEFiles.
ATTACHFILE or DETACHFILE	Attaches or detaches a DBEFile or DBEFileSet.
CHANGELOG	Causes a DBEnvironment to change to a new log file.
SHOWDBE	Displays database attribute information including audit DBEnvironment parameters and whether it is a wrapper database.
MOVEFILE	Moves a DBEFile.

The syntax for the SQLUtil commands is found in Appendix F of this manual.

The following ALLBASE/SQL statements can be used interactively or programmatically:

CHECKPOINT statement	Retrieves the number of free blocks available in a log file.
CREATE DBEFILE statement	Creates a DBEFile in a particular group or on a particular volume.

The full syntax for the SQL statements is found in the “SQL Statements” chapter of the *ALLBASE/SQL Reference Manual*.

Backing Up and Restoring DBEnvironments

The DBA is also responsible for routine backup and, when necessary, the restoring of DBEnvironments following a system failure. This means:

- Choosing either archive or nonarchive logging.
- Implementing a backup strategy.
- Backing up DBEnvironments and log files regularly.
- Adding and dropping log files as needed.
- Restoring the system from backups as needed.

The subject is discussed fully in the chapter “Backup and Recovery.”

Migrating DBEnvironments Between Releases of ALLBASE/SQL

The internal structure of a DBEnvironment must be compatible with the particular release of ALLBASE/SQL software being used. After installing a new version of ALLBASE/SQL, use SQLMigrate to migrate a DBEnvironment forward to the current release or backward to an earlier release. Under normal conditions, you would not need to perform a backward migration. This functionality is provided so that if you ever choose to restore older software, you will be able to migrate your DBEnvironment backward quickly and easily.

Prior to release A.20.00 of ALLBASE/SQL, it was sometimes necessary to use the ISQL UNLOAD command, recreate the DBEnvironment, and use the LOAD command to migrate your DBEnvironment to one that was compatible with a new release. This approach may still be useful. The process of unloading and reloading is described in the “Maintenance” chapter. Additional information is provided about SQLMigrate later in this chapter under “Using SQLMigrate.”

Tools for the DBA

In addition to the general set of SQL statements, the DBA uses several utility programs in creating and maintaining DBEnvironments. These are:

- ISQL
- SQLUtil
- SQLGEN
- SQLMigrate
- SQLMON
- SQLVer
- SQLAudit

The use of each toolset is described in the following paragraphs. Complete command syntax for ISQL, SQLUtil, SQLGEN, SQLMigrate, and SQLAudit is in the appendices. In addition, there is a discussion of SQLINSTL. SQLINSTL is a script used when moving to a new version of ALLBASE/SQL.

Using ISQL

ISQL lets you issue most SQL commands interactively. In addition, it lets you load and unload tables using the LOAD and UNLOAD statements. ISQL also includes a help facility which explains the syntax of SQL and ISQL commands.

ISQL **command files** offer a shortcut to creating databases. Command files allow you to store a series of ISQL and SQL commands and then, with a single START command, execute all the commands in that file. This is very useful for groups of commands you execute together frequently. In addition, if an entire DBEnvironment is created by using command files, it is easy to recreate the DBEnvironment, as well as examine or modify any part of its definition. The sample DBEnvironment used in all the examples in this manual can actually be created with the series of command files found in the “Sample DBEnvironment” appendix in the *ALLBASE/SQL Reference Manual*.

SQL data definition statements such as CREATE and maintenance commands such as UPDATE STATISTICS obtain locks on the system catalog. To avoid contention, you can use command files to execute these statements on a DBEnvironment during off hours. This will reduce the amount of waiting on locks during peak working hours.

You can use SQLGEN to create command files for use through ISQL. See the section “Using SQLGEN” later in this chapter. For complete information on ISQL command files, refer to the *ISQL Reference Manual for ALLBASE/SQL and IMAGE/SQL*.

Using SQLUtil

SQLUtil is an ALLBASE/SQL utility program that lets you manage the startup parameters for each DBEnvironment and perform several other maintenance tasks. With SQLUtil, you can:

- Display and change startup parameters in the DBECon file.
- Move DBEFiles or log files from one location to another.
- Purge DBEFiles.
- Purge the DBEnvironment.
- Back up and restore a DBEnvironment.
- Monitor the log.

- Add, purge, store, and restore log files.
- Perform rollforward recovery.

Starting SQLUtil

There are two ways to run the SQLUtil program: directly from MPE/iX, or within ISQL. For the first method, type the following command:

```
: RUN SQLUTIL.PUB.SYS
```

To execute SQLUtil within ISQL, type:

```
isql=> sqlutil;
```

No matter which method you use, you will see the SQLUtil banner and prompt (>>).

The SQLUtil commands are described in the appendix, “SQLUtil.” Now you can execute SQLUtil commands until you enter an EXIT or QUIT command.

SQLUtil requires the use of the message catalog, SQLCT000.PUB.SYS. If you are using native language data, the name of the catalog is SQLCTxxx.PUB.SYS, where xxx is the number of your current language. See “Native Language Support” for information about specifying a native language as the current language. SQLUtil accepts responses to prompts for a DBEnvironment name or for a filename in either the current language or NATIVE-3000.

Instead of entering commands at the keyboard, you may redirect input from a command file by specifying the STDIN option in the run string, as in the following example:

```
: RUN SQLUTIL.PUB.SYS;STDIN=Commands.SomeGrp.SomeAcct
```

As with ISQL, you can use a semicolon to terminate an SQLUtil command. Semicolons are not needed, however. All responses to commands must be contained on one line; continuation of responses is not supported.

Some SQLUtil commands display a subsidiary prompt (“->”). From this prompt, you can enter // to return to the SQLUtil prompt (“>>”). Also, if input from within the ISQL session was coming from a command file or the command buffer when SQLUtil was invoked, that input is suspended during the SQLUtil session until the user enters either EXIT or QUIT. Finally, prompting-mode is the only mode of operation in SQLUtil. You cannot specify a complete command line as you would in ISQL.

JCWs Set by SQLUtil

When running SQLUtil in batch mode, your job can check the following JCWs to ensure that SQLUtil completed successfully.

Table 1-1. JCWs Set by SQLUtil

JCW Name	Contents	Range
UTILERR	number of SQL errors encountered	0 - 32767
UTILWARN	number of SQL warnings encountered	0 - 32767
JCW	FATAL if SQL error encountered, 0 if SQL error not encountered. The system JCW is set only if the SET EXIT_ON_DBERR command is ON.	FATAL or 0

Using SQLGEN

SQLGEN is a utility that generates the commands used to recreate all or part of an existing ALLBASE/SQL DBEnvironment. It also generates LOAD/UNLOAD and UPDATE STATISTICS commands. These commands are placed in one or more data files called schema files which can then be used as ISQL command files to recreate the DBEnvironment.

SQLGEN has several uses. It aids in migrating to new releases of ALLBASE/SQL when unloading or reloading are necessary. In addition, SQLGEN helps with other database tasks. DBEnvironments can be designed and tuned on a development system and then the entire schema can be transferred to a production system. SQLGEN can also help move part of a DBEnvironment. For example, if a department transfers from one site to another, that department's portion of a DBEnvironment can be easily moved to a new system.

You can also use SQLGEN to create a replicate DBEnvironment schema from the master DBEnvironment. When you do so, you must use the START DBE NEWLOG statement to specify the AUDIT LOG, AUDIT NAME, DEFAULT PARTITION, MAXPARTITIONS, and AUDIT ELEMENTS parameters. You must also use CREATE PARTITIONS to create partitions and ALTER TABLE to assign tables to the partitions.

Starting SQLGEN

You must have DBA authority to use SQLGEN. Enter commands at the keyboard or redirect the input by specifying a command file name in the run string. To run SQLGEN directly from the operating system, type:

```
: RUN SQLGEN.PUB.SYS
```

To run SQLGEN from within ISQL, type:

```
isql => sqlgen;
```

No matter which method you use, the SQLGEN prompt (>>) appears, ready for you to enter commands.

SQLGEN requires the use of the message catalog, SQLCT000.PUB.SYS. If you are using native language data, the name of the catalog is SQLCTxxx.PUB.SYS, where xxx is the number of your current language. See "Native Language Support" for information about specifying a native language as the current language.

Usually, STARTDBE will be the first command you enter. This command connects to the DBEnvironment and checks to make sure you have DBA authority. It is recommended that the DBEnvironment be shut down prior to issuing the STARTDBE command to ensure that

system catalogs are not being altered during SQLGEN execution. When you finish entering commands, use the EXIT command to return either to MPE/iX or to ISQL.

When you run SQLGEN directly from MPE/iX, you can specify the editor name as an INFO string. When you run SQLGEN from ISQL, the current editor name is passed to and displayed by SQLGEN. Refer to the SET EDITOR command for more information.

SQLGEN Commands

SQLGEN commands can be divided into two categories, generate commands and auxiliary commands. Generate commands such as GENERATE ALL build the SQL statements necessary to recreate all or part of a DBEnvironment. Auxiliary commands provide services needed to use SQLGEN. For example, the STARTDBE command connects to the DBEnvironment. The EDITOR command allows you to access an editor from within SQLGEN. With SET commands you redefine the editor or designate a schema file name. HELP, RELEASE, and EXIT are also auxiliary commands. SQLGEN also allows you to enter MPE/iX commands when they are preceded by the character `:`.

The SQLGEN commands are described in the appendix, "SQLGEN."

SQLGEN Schema Files

SQLGEN places the statements it generates in schema files which are then used as ISQL command files for recreating the DBEnvironment. All GENERATE commands prompt for a schema file name unless you have already designated the name with the SET SCHEMA command. If you enter a carriage return instead of a name at the schema file name prompt, the output is displayed on your terminal screen. In most cases, GENERATE commands automatically include a COMMIT WORK at the end of the schema file. However, if the generate command was not able to build any commands (for example, GENERATE GROUP cannot build commands if no groups exist in the DBEnvironment), then no COMMIT WORK is generated.

When a GENERATE command encounters an unexpected ALLBASE/SQL error, SQLGEN automatically returns the schema file in use to the state it was in before the command began. For example, if an error occurs during a GENERATE ALL command, any commands that have already been placed in the schema file are automatically rolled back. To execute the command, correct the problem and re-enter GENERATE ALL.

SQLGEN Conventions

To use SQLGEN effectively you must be familiar with the conventions it uses. The SQLGEN prompt is >>. Unlike ISQL, SQLGEN does not require a command-terminating semicolon. Once you enter a command, SQLGEN prompts for your responses. At an object name prompt, enter the name of the object for which you wish to generate commands. When you enter an object name, SQLGEN prompts you for another. When you finish entering object names, a carriage return returns you to the SQLGEN prompt. If you wish to select all qualifying names, enter an '@' at the object name prompt. To see a list of all qualifying names, type a '?'. If no objects qualify, SQLGEN displays a message and automatically returns you to the previous prompt.

When choosing from two options, (y/n) for example, the first choice listed is always the default (carriage return). To return to the SQLGEN prompt from any command, type '/' or '/'. To access the operating system, enter a colon (:) and a carriage return. Type RESUME to return to SQLGEN. To enter system commands from within SQLGEN, type a colon (:) as the

first character of the command. Only commands allowed in break can be executed. Table 1-2 summarizes SQLGEN's general conventions.

Table 1-2. SQLGEN General Conventions

Convention	Explanation	Example
No Semicolons	Command terminating semicolons are not needed, but allowed.	EXIT or EXIT;
:<Command> to enter system commands	Use any command allowed in break.	>> :LISTF <input type="button" value="Return"/>
: to escape to MPE	Use any command allowed in break Type resume to return to SQLGEN.	>> : <input type="button" value="Return"/> : resume <input type="button" value="Return"/> >>
'/' OR '/' to return to the SQLGEN prompt	Single or double slashes end a command.	Table Name >> // <input type="button" value="Return"/> >>

Entering Object Names

SQLGEN automatically upshifts all object names not entered in quotes. To enter an object name, in most cases all you need to do is type the name. SQL pattern matching symbols (% and _) can be used. For example, at a table name prompt, P% selects all tables starting with P.

Single and double quotation marks are used for special cases.

- Use single quotes around lower case names. Pattern matching symbols are allowed within the single quotes. For example, to select all objects starting with Na, enter 'Na%'.
- Use double quotes around object names that contain one of the pattern matching symbols (% or _). In this case, pattern matching cannot be used. For example, the table name NJS% must be entered "NJS%".

Table 1-3 summarizes SQLGEN's naming conventions.

Table 1-3. SQLGEN Name Entry Conventions

Convention	Explanation	Example
Pattern Matching Using % and _	% replaces 0 or more characters. _ replaces one character.	P% all objects starting with the character P P_ all two-letter names starting with P
No Quotes	Use in most cases. Name is upshifted. Pattern matching allowed.	PurchDB PURCHDB P% all names starting with P
Single Quotes	Use around lower case names. Names are not upshifted. Pattern matching allowed.	'tab' tab 'T%s' all names starting with T and ending with s
Double Quotes	Use when the name itself contains a % or_. Names are taken exactly as entered. No pattern matching allowed.	"t%s" t%s

JCWs Set by SQLGEN

When running SQLGEN in batch mode, your job can check the following JCWs to ensure that SQLGEN completed successfully.

Table 1-4. JCWs Set by SQLGEN

JCW Name	Contents	Range
GENERR	number of SQL errors encountered	0 - 32767
GENWARN	number of SQL warnings encountered	0 - 32767
JCW	FATAL if SQL error encountered, 0 if SQL error not encountered. The system JCW is set only if the SET EXIT_ON_DBERR command is ON.	FATAL or 0

Using SQLMigrate

If you are updating from an earlier release of ALLBASE/SQL, you must perform the ALLBASE/SQL migration to migrate your DBEnvironment. The method used depends upon the version of ALLBASE/SQL that you are currently using. Use SQLMigrate to convert the DBEnvironment between major releases such as from F.0 to G.0. Use the SQLINSTL script to migrate between versions of a release (such as from G1.14 to G1.15) or minor releases (such as from G.0 to G.1).

The SQLINSTL script is provided in the G.1 and later versions of ALLBASE/SQL to make it easy for the DBA to move to a delta release. Using SQLINSTL ensures that you will have access to the most recent version of the System and Catalog views, and it also uses VALIDATE FORCE statements to revalidate all stored sections to be compatible with the new release.

If SQLINSTL is not executed on a DBEnvironment after installing a new version, stored sections may not be properly revalidated causing run-time errors. Revalidating stored sections

at run-time during production hours can also cause concurrency problems due to write locks placed on the system catalog. You must use SQLINSTL whenever a new version of ALLBASE/SQL is installed; however, SQLINSTL does not need to be executed separately if SQLMigrate is being executed to migrate between major releases. Refer to the section, “Using SQLINSTL,” later in this chapter for more information.

Note ALLBASE/SQL no longer supports pseudo-mapped files. You must convert all pseudo-mapped DBEFiles back to mapped DBEFiles before you upgrade.

Use the SQLUtil SHOWACCESS command to list DBEFiles. If any are listed as pseudo-mapped, use the SQLUtil MOVEFILE command to convert the pseudo-mapped files to mapped before you upgrade ALLBASE/SQL. Refer to the commands in the appendix, “SQLUtil.”

Warning **SQLINSTL and SQLMigrate drop all system catalog views. If any view has been created upon a system catalog view, that view will also be dropped. To avoid dropping views, before executing SQLINSTL or SQLMigrate, use the GENERATE VIEWS command in SQLGEN to create a script to recreate the user view.**

Running SQLMigrate

Use SQLMigrate to migrate between major revisions of ALLBASE/SQL. Backup the DBEnvironment prior to running SQLMigrate.

You must be the DBECreator or have SM capability to migrate a DBEnvironment.

To run SQLMigrate, use the following command:

```
:RUN SQLMIG.PUB.SYS
```

You will see the SQLMigrate banner and prompt (SQLMIGRATE=>). Command entry follows ISQL rules; that is, commands are terminated with a semicolon. The SQLMigrate commands are described in the appendix “SQLMigrate.”

Forward and backward migration steps are discussed below. Only one DBEnvironment can be migrated at a time.

Types of Forward Migration

There are two flavors of forward migration to consider:

1. Migrating a DBEnvironment *without* audit logging into audit logging releases.
2. Migrating a DBEnvironment *with* audit logging into audit logging releases.

For both cases, all tables are placed in the default partition. In the first case, no audit logging is enabled for the DBEnvironment. In the second case, the only audit logging element allowed is DATA because SQLMigrate does not support any other audit elements.

If you migrate an audit logging DBEnvironment, it should *not* be migrated to a non-audit logging DBEnvironment. Therefore, you must specify AUDIT LOG in any START DBE NEWLOG statement that SQLMigrate performs.

For backward migration, since previous releases do not support partitions, the only supported option for a DBEnvironment with audit logging in use is to place all the tables in the default partition with the only audit logging element specified being DATA. Then the DBEnvironment can be migrated backward unchanged.

If the audit elements include something other than DATA, one of the following may happen:

- The migration is not allowed to proceed.
- The elements are reset to be only DATA.

Steps for Forward Migration

1. Prior to updating the operating system and ALLBASE/SQL software, do the following for each DBEnvironment that will be migrated:
 - a. Run ISQL and issue a START DBE statement. This ensures that the DBEnvironment is logically consistent.
 - b. Run SQLUtil and issue the STORE command to backup each DBEnvironment. Note: Log files are not stored using this command. Application programs associated with the DBEnvironment must be backed up separately.
2. Backup the ALLBASE/SQL software.
3. Update the operating system and the ALLBASE/SQL software. If you are updating the operating system, make sure you have a backup of the operating system (including the old release of ALLBASE/SQL software).
4. Type the following command at the operating system prompt:

```
:RUN SQLMIG.PUB.SYS
```

5. Issue the SHOW VERSIONS command to determine the possible version values that can be entered as the version parameter in both the PREVIEW and the MIGRATE commands. The version parameter indicates to SQLMigrate the release of ALLBASE/SQL software with which you wish the DBEnvironment to be compatible.

For example, you can enter the following command:

```
SQLMIGRATE=> show versions;

VERSION          RELEASE

E                HP36216-02A.E
F                HP36216-02A.F
G                HP36216-02A.G

SQLMIGRATE=>
```

This shows the MPE/iX ALLBASE/SQL versions and releases for which you can use SQLMigrate.

6. For each DBEnvironment that will be migrated, issue the PREVIEW command to check for errors that might occur during migration. The syntax for PREVIEW of a forward migration is:

```
SQLMIGRATE=> PREVIEW 'DBEnvironmentName' FORWARD [TO 'Version'];
```

Note that the version parameter is optional. If this parameter is omitted, the most recent version supported by SQLMigrate will be used as the default.

If you receive the message that there is not enough space in the SYSTEM DBEFileSet to complete the migration successfully, the number of DBEFile pages needed will be returned. Use the following commands to create a new DBEFile and add it to the SYSTEM DBEFileSet:

```
SQLMIGRATE=> CREATE DBEFILE DBEFileName
    WITH PAGES = DBEFileSize, NAME = 'SystemFileName';

SQLMIGRATE=> ADD DBEFILE DBEFileName TO DBEFILESET SYSTEM;
```

Note that the syntax of these commands is the same as in ISQL.

Repeat this step until no errors are encountered and SQLMigrate returns the message:

```
The proposed migration should be successful.
```

If you encounter errors during the PREVIEW step that you do not understand, contact your HP Service Representative or Response Center.

7. Once you have completed the PREVIEW step, issue the MIGRATE command to modify the DBEnvironment so that it is compatible with the release of ALLBASE/SQL that you wish to use. The MIGRATE FORWARD syntax is:

```
SQLMIGRATE=> MIGRATE 'DBEnvironmentName' FORWARD [TO 'Version'];
```

Note that the version parameter is optional. If it is omitted, the most recent version supported by SQLMigrate will be used as the default.

When the MIGRATE command is finished, SQLMigrate automatically purges the old user log files and performs a START DBE NEWLOG statement to create new user log files. The options of the START DBE NEWLOG statement match the startup parameters contained in the DBECon file. SQLMigrate does not issue a START DBE NEWLOG statement if the NEWLOG option has been set to OFF.

If the START DBE NEWLOG statement fails you must exit from SQLMigrate, run ISQL, and issue a START DBE NEWLOG statement. This creates a new log file that will be compatible with the target release of ALLBASE/SQL. Note that you cannot use your DBEnvironment after it has been migrated until the START DBE NEWLOG statement executes successfully.

If you encounter errors during the MIGRATE step that you do not understand, contact your HP Service Representative or Response Center.

8. Make a backup of the migrated DBEnvironment immediately after the START DBE NEWLOG statement completes. If you wish to use nonarchive logging, run SQLUtil and use the STORE command. For archive logging, you should choose one of the following:
 - a. If you have TurboSTORE software, do a *concurrent backup* using the SQLUtil STOREONLINE command.

- b. If you do not have TurboSTORE, issue the following SQL statements *in single user mode* in ISQL:

```
isql=> BEGIN ARCHIVE;  
isql=> COMMIT ARCHIVE;
```

Exit from ISQL and run SQLUtil, then issue the STORE command.

9. Run SQLUtil (if you are not already in SQLUtil from the previous step) and issue the SHOWDBE command to check the parameters of the new version of the DBEnvironment. Use the ALTDBE command if changes are necessary. Use the SHOWLOG command to display current log information.
10. You may have received a message stating that stored sections were invalidated. This is to be expected. Stored sections will be revalidated automatically when they are executed. For some users, production may be faster if revalidation is done before that time. Revalidation can be accomplished by preprocessing the application programs that contain the stored sections. If you are migrating from release F (or later), you can revalidate stored sections with the VALIDATE statement in ISQL.

Steps for Backward Migration

1. Run SQLMigrate and issue the SHOW VERSIONS command to determine the possible values that can be entered as the version parameter in both the PREVIEW and the MIGRATE commands. The version parameter indicates to SQLMigrate the release of ALLBASE/SQL software with which you wish the DBEnvironment to be compatible.
2. Prior to restoring the backup version of the operating system and ALLBASE/SQL, do the following for each DBEnvironment that will be migrated:
 - a. Run ISQL and issue a START DBE statement. This ensures that the DBEnvironment is logically consistent.
 - b. Run SQLUtil and issue the STORE command to make a backup of the DBEnvironment. Note: Log files are not stored using this command. Application programs associated with the DBEnvironment must be backed up separately.
 - c. Run SQLMigrate and issue the PREVIEW command to check for errors that might occur during migration. The DBEnvironment is not modified during this command. The syntax for the PREVIEW of a backward migration is:

```
SQLMIGRATE=> PREVIEW 'DBEnvironmentName' BACKWARD TO 'Version';
```

If you receive the message that there is not enough space in the SYSTEM DBEFileSet to complete the migration successfully, the number of DBEFile pages needed will be returned. To increase the number of pages, you need to add another DBEFile to the SYSTEM DBEFileSet. To do this, use the following syntax:

```
SQLMIGRATE=> CREATE DBEFILE DBEFileName  
WITH PAGES = DBEFileSize, NAME = 'SystemFileName';
```

```
SQLMIGRATE=> ADD DBEFILE DBEFileName TO DBEFILESET SYSTEM;
```

Repeat this step until no errors are encountered and SQLMigrate returns the message:

```
The proposed migration should be successful.
```

If you encounter errors during the PREVIEW step that you do not understand, contact your HP Service Representative or Response Center.

- d. Run SQLMigrate and issue the MIGRATE command to modify your DBEnvironment to make it compatible with the old release of ALLBASE/SQL. The MIGRATE BACKWARD syntax is:

```
SQLMIGRATE=> MIGRATE 'DBEnvironmentName' BACKWARD TO 'Version';
```

If you encounter errors during the MIGRATE step that you do not understand, contact your HP Service Representative or Response Center.

3. Restore the backup versions of the operating system and of ALLBASE/SQL that were made during the forward migration steps.
4. For each DBEnvironment that was backward migrated, do the following:
 - a. Run ISQL and issue the START DBE NEWLOG statement. This creates a new log file that will be compatible with the old release of ALLBASE/SQL. Note that you will not be able to use your DBEnvironment after it has been migrated until this step has been completed. SQLMigrate does not automatically perform a START DBE NEWLOG statement during backward migration.
 - b. Run SQLUtil and issue the SHOWDBE command to check the parameters of the new version of the DBEnvironment. Use the ALTDBE command if changes are necessary.

To enable archive mode logging, use the following steps:

- i. If you have TurboSTORE software, run SQLUtil and use the STOREONLINE command.
- ii. If you do not have TurboSTORE, issue the following SQL statements from ISQL:

```
isql=> BEGIN ARCHIVE;  
isql=> COMMIT ARCHIVE;
```

Exit from ISQL and run SQLUtil, then issue the STORE command.

- c. Drop views and any stored sections created under the later release, because these views and sections are no longer usable. Refer to the DROP VIEW and DROP MODULE commands in the *ALLBASE/SQL Reference Manual*. Do not try to install modules that were created under the release you are migrating back from as they will not be compatible with the release of ALLBASE/SQL you are migrating to.
- d. You may have received a message stating that stored sections were invalidated. This is to be expected. Stored sections will be revalidated automatically when they are executed. For some customers, production may be faster if revalidation is done before that time. Revalidation can be accomplished by preprocessing the application programs that contain the stored sections. If you are migrating from release F (or later), you can revalidate stored sections with the VALIDATE statement in ISQL.

JCWs Set by SQLMigrate

When running SQLMigrate in batch mode, your job can check the following JCWs to ensure that SQLMigrate completed successfully.

Table 1-5. JCWs Set by SQLMigrate

JCW Name	Contents	Range
MIGERR	number of SQL errors encountered	0 - 32767
MIGWARN	number of SQL warnings encountered	0 - 32767
JCW	FATAL if SQL error encountered, 0 if SQL error not encountered. The system JCW is set only if the SET EXIT_ON_DBERR command is ON.	FATAL or 0

Using SQLINSTL

SQLINSTL is a script that you use instead of SQLMigrate when migrating between releases of versions of ALLBASE/SQL (such as G1.14 to G1.15) or minor releases (such as G.0 to G.1). When migrating between major releases (such as F.0 to G.0 or F.0 to G.1), you must instead use SQLMigrate to migrate your DBEnvironment. Refer to the previous section, “Using SQLMigrate,” for more information.

When using SQLINSTL, execute SQLINSTL on each DBEnvironment on the system to ensure that you have access to the most recent version of the system catalog views. SQLINSTL also ensures that stored sections are properly revalidated to be compatible with the new release. If SQLINSTL is not executed, errors may result when stored sections are executed due to compatibility problems.

Warning **SQLINSTL and SQLMigrate drop all system catalog views. If any view has been created upon a system catalog view, that view will also be dropped. To avoid dropping views, before executing SQLINSTL, use the GENERATE VIEWS command in SQLGEN to create a script to recreate the user view.**

The following is an example of using SQLINSTL from ISQL:

```
:isql.pub.sys  
isql=>start sqlinstl.pub.sys (mydbe);  
isql=>exit;
```

Read the SQLINSTL file on your system for more information.

If you are using ARCHIVE MODE LOGGING, you must make a backup of the DBEnvironment after using SQLINSTL. This backup must be used if rollforward recovery is to be performed at some time in the future.

Using SQLMON

SQLMON is an online diagnostic tool that monitors the activity of an ALLBASE/SQL DBEnvironment. SQLMON screens provide information on file capacity, locking, I/O, logging, tables, and indexes. They summarize activity for the entire DBEnvironment, or focus on individual sessions, programs, or database components. SQLMON is a read only utility, and cannot modify any aspect of the DBEnvironment. SQLMON is documented fully in the *ALLBASE/SQL Performance and Monitoring Guidelines*.

To run SQLMON, log on as system manager, as the database administrator, or a user that was granted MONITOR authority, and issue the following command:

```
: SQLMON
```

Using SQLVer

SQLVer allows you to check the version strings of the ALLBASE/SQL files. The syntax of SQLVer is as follows:

```
SQLVER [file]
```

The file parameter specifies the fully qualified name of the ALLBASE/SQL file whose version you want to check. If the file parameter is omitted, SQLVer verifies all of the ALLBASE/SQL files and summarizes the following:

- current ALLBASE/SQL version
- number of missing files
- number of version incompatibilities

SQLVer sets the SQLVERERR JCW to the following values:

Table 1-6. SQLVERERR JCW

SQLVERERR Value	Explanation
FATAL	SQLVer internal error
0	no version incompatibilities or missing files detected
1	missing files detected
2	version incompatibilities detected
3	version incompatibilities and missing files detected

In the example that follows, the version strings of all the ALLBASE/SQL files are checked:

```
MPE:sqlver
```

```
ALLBASE/SQL G1 Version Checker
WED, FEB 5, 1997 2:42 PM
```

```
ALLBASE/SQL:
-----
```

```
dbmon.pub.sys (created THU, OCT 17, 1996):
    @(#)HPDBCORE A.G1.18 09/24/96          ALLBASE/SQL

hpsqludc.pub.sys (created THU, OCT 17, 1996):

isql.pub.sys (created THU, OCT 17, 1996):
    HP36216-02A.G1.18      Interactive SQL/3000      ALLBASE/SQL

psqlc.pub.sys (created THU, OCT 17, 1996):
    HP36216-02A.G1.18      C Preprocessor/3000      ALLBASE/SQL

psqlcob.pub.sys (created THU, OCT 17, 1996):
    HP36216-02A.G1.18      COBOL Preprocessor/3000  ALLBASE/SQL
```

```

psqlfor.pub.sys (created THU, OCT 17, 1996):
    HP36216-02A.G1.18      FORTRAN Preprocessor/3000      ALLBASE/SQL

psqlpas.pub.sys (created THU, OCT 17, 1996):
    HP36216-02A.G1.18      PASCAL Preprocessor/3000      ALLBASE/SQL

sqlct000.pub.sys (created THU, OCT 17, 1996):

sqlgen.pub.sys (created THU, OCT 17, 1996):
    HP36216-02A.G1.18      SQL Command Generator/3000      ALLBASE/SQL

sqlmonp.pub.sys (created THU, OCT 17, 1996):
    HP36216-02A.G1.18      SQLMonitor/3000      ALLBASE/SQL

sqlmig.pub.sys (created THU, OCT 17, 1996):
    HP36216-02A.G1.18      SQLMigrate/3000      ALLBASE/SQL

sqlutil.pub.sys (created THU, OCT 17, 1996):
    HP36216-02A.G1.18      DBE Utility/3000      ALLBASE/SQL

sqlwl000.pub.sys (created THU, OCT 17, 1996):

ALLBASE/NET:
-----

anutil.pub.sys (created THU, OCT 17, 1996):
    @(#)HP36216-02A.G1.18      ALLBASE/NET 900      ALLBASE/SQL

hpdalstn.pub.sys (created THU, OCT 17, 1996):
    @(#)HP36216-02A.G1.18      ALLBASE/NET 900      ALLBASE/SQL

hpdadvr.pub.sys (created THU, OCT 17, 1996):
    HP36216-02A.G1.18      ALLBASE/NET 900      ALLBASE/SQL

hpidvdr.pub.sys (created THU, OCT 17, 1996):
    @(#)HP36216-02A.G1.18      ALLBASE/PC API 900      ALLBASE/SQL

hpipns.pub.sys (created THU, OCT 17, 1996):
    @(#)HP36216-03A.G1.12      ALLBASE/PC API 900      ALLBASE/SQL

netutil.pub.sys (created THU, OCT 17, 1996):
    HP36216-02A.G1.18      ALLBASE/NET 900      ALLBASE/SQL

hpipefile.net.sys (created THU, OCT 17, 1996):

hpipeast.net.sys (created THU, OCT 17, 1996):

hpipeasrv.net.sys (created THU, OCT 17, 1996):

Common ALLBASE/SQL Libraries:
-----
xl.pub.sys (created THU, OCT 17, 1996):
    @(#)HPDBCORE A.G1.18 09/24/96      ALLBASE/SQL
    HP36216-02A.G1.18 Replicate/3000      ALLBASE/SQL
    HP36216-02A.G1.18      Query Processor/3000      ALLBASE/SQL
    HP36216-02A.G1.18      SQL Parser/Linearizer/3000      ALLBASE/SQL
    HP36216-02A.G1.18      ALLBASE/NET 900      ALLBASE/SQL
    HP36216-02A.G1.18      SQLX/3000      ALLBASE/SQL
    HP36216-02A.G1.18      SQLX/3000      ALLBASE/SQL
    HP36216-02A.G1.18      SQLX/3000      ALLBASE/SQL
    HP36216-02A.G1.18      PASCAL Preprocessor/3000      ALLBASE/SQL
    HP36216-02A.G1.18      C Preprocessor/3000      ALLBASE/SQL
    HP36216-02A.G1.18      FORTRAN Preprocessor/3000      ALLBASE/SQL
    HP36216-02A.G1.18      COBOL Preprocessor/3000      ALLBASE/SQL

```

SUMMARY:

Current ALLBASE/SQL version installed: A.G1.18

ALLBASE/SQL has 0 file(s) missing.

ALLBASE/NET has 0 file(s) missing.

Number of version incompatibilities found: 0

Using SQLAudit

SQLAudit is an ALLBASE/SQL utility program that can be used in conjunction with audit DBEnvironments to view the changes that have been made to the DBEnvironment. They include such groups as DML commands (INSERT, UPDATE, DELETE), and DDL commands (data definition, storage, and authorization commands, for example).

The committed transactions are audited by extracting information from the audit log records in the log files. These audit log records are generated when audit logging is enabled on the DBEnvironment through the START DBE NEW or START DBE NEWLOG commands. Besides enabling audit logging, these commands include other audit logging parameters that define the groups of SQL commands you want to audit. These groups are discussed under “Creating Audit DBEnvironments” elsewhere in this manual. They include such groups as DML commands (INSERT, UPDATE, DELETE), and DDL commands (data definition, storage, and authorization commands, for example).

SQLAudit is designed with a number of commands for viewing the changes made to the DBEnvironment. The following list of SQLAudit commands provides an overview of these commands:

AUDIT	Audits changes made between two given audit points.
EDITOR	Invokes the currently set editor.
EXIT	Terminates SQLAudit execution.
GET AUDITPOINT	Determines the current audit point information from the DBEnvironment.
HELP	Displays and describes all SQLAudit commands.
LOCK AUDITPOINT	Locks an audit point to prevent overwriting transactions in the log files that have not been audited
MODIFY AUDITPOINT	Modifies the current audit point information for the DBEnvironment.
QUIT	Terminates SQLAudit execution.
SET	Displays current SQLAudit settings.
SET DBENVIRONMENT	Establishes or releases the connection to the DBEnvironment.
SET ECHO_ALL	Enables or disables echoing of user input to the standard output.
SET EDITOR	Defines the current editor.

SET EXIT_ON_DBERR	Sets SQLAudit to exit or continue when a database error is encountered.
SET RECOVERFILE	Defines the recovery file used by the AUDIT command.
SHOW AUDITPOINT	Displays audit point information contained in a file.
UNLOCK AUDITPOINT	Removes a locked audit point from the log.
:	Escapes temporarily to the operating system for the execution of operating system commands.

Refer to the appendix “SQLAudit” for the complete SQLAudit command reference.

SQLAudit Conventions

The following conventions should be noted when using SQLAudit:

- Unlike ISQL, SQLAudit does not require a command terminating semicolon. (If a semicolon is entered, SQLAudit ignores it.)
- When choosing from two options, (y/n), for example, the first choice listed is always the default (carriage return).
- To return to the main prompt (SQLAudit >>) at any time, type a double or single slash (// or /).
- If an SQLAudit command creates a file, the file is a Portable Operating System Interface (POSIX) file. (POSIX is a command shell that is part of MPE/iX and causes MPE/iX to be more similar to HP-UX.) When you specify a file name in an SQLAudit command, it is best to enter it in all uppercase letters. This way, you can see the file in a file list from within SQLAudit. If the file name contains lowercase letters, you must exit SQLAudit and enter an *ls* command from the POSIX shell in order to see the file in a list.
- To escape to the command interpreter (MPE/iX) enter a colon (:) and a carriage return. To enter operating system commands from within SQLAudit, type a colon (:) as the first character of the command. (In either case, only commands allowed in MPE/iX can be executed.) You then type RESUME RETURN to return to SQLAudit.

Table 1-7 summarizes these conventions.

Table 1-7. SQLAudit General Conventions

Convention	Explanation	Example
No Semicolons	Command terminating semicolons are not needed, but allowed.	EXIT or EXIT;
:<Command> to enter system commands	Use any command allowed in MPE/iX.	SQLAudit >> :listf (RETURN)
: to escape to MPE/iX	Use any command allowed in MPE/iX.	SQLAudit >> : (RETURN) : RESUME (RETURN) SQLAudit >>
// OR / to return to the SQLAudit prompt	Single or double slashes end a command.	DBEnvironment Name >> // SQLAudit >>
ALL CAPS	When you use an SQLAudit command to create a file, it is recommended that you enter the filename in all uppercase letters, so that you can see it from the POSIX shell.	AUDIT BEGINFILE

Understanding Audit Points

SQLAudit requires a beginning and ending point in the log files to determine what portion of the log history to audit. These audit points are defined by using the command GET AUDITPOINT. GET AUDITPOINT determines all the current transaction information for the DBEnvironment and stores it and identification information in a file for later use. The audit point file can then be used by other commands, such as AUDIT or SHOW AUDITPOINT.

The beginning audit point provides the AUDIT command with information about the previously committed transactions. The AUDIT command uses this information to audit all transactions that have committed after the beginning audit point was defined by GET AUDITPOINT.

The ending audit point is used by the AUDIT command to provide a point where the AUDIT command can stop processing audit log records. In addition, the ending point also provides a list of partitions that have committed transactions, and when compared with the beginning audit point, a list of partitions to be audited.

If you do not specify a beginning audit point, the AUDIT command assumes by default that you want to audit every transaction, starting from the first committed transaction for each partition being audited. If you do not specify an ending audit point file, the AUDIT command assumes by default that you want to use the last committed transaction in the log file as the stopping criteria.

Whether you specify a beginning audit point or assume the default, SQLAudit must still be able to find the first committed transaction for the partition in the log files. Therefore, if the first committed transaction for an audited partition cannot be found, an error is returned and the partition is dropped from the list of partitions being audited.

Suppose, for example, that at the time the beginning audit point is defined, the current list of partitions and their committed transactions exists as follows:

Partition	Transaction Identification Information
1	Transaction # 10
2	Transaction # 21
6	Transaction # 16

Transaction number 10 was the last transaction that changed data in partition number 1. Similarly, transaction numbers 21 and 16 were the last transactions to change data in partition numbers 2 and 6, respectively.

Suppose that the following transaction information is reflected in the ending audit point information:

Partition	Transaction Identification Information
1	Transaction # 45
2	Transaction # 21
4	Transaction # 51
6	Transaction # 45

In comparing these two audit points, SQLAudit can determine several facts:

- No new transactions have been committed on partition number two.
- Transactions have been committed (since the beginning audit point) on partitions 1 and 6.
- Partition number 4 has had its first transaction committed since the beginning audit point.

With this information, the AUDIT command can determine that the only partitions that need to be audited are 1, 4, and 6. The AUDIT command would still include partition 2 in the list of partitions being audited, but no transactions should be found.

Note that in this example, two partitions have the same transaction number (partitions 1 and 6). This means that transaction number 45 made changes to data in both partitions 1 and 6.

Although this example is an oversimplification of what the transaction information looks like, the concepts still hold true.

Example of Getting Audit Points

The following example illustrates how GET AUDITPOINT defines and displays an audit point:

```
SQLAudit >> GET AUDITPOINT

Audit Point File >> STARTPT
Lock Log for Current Audit Point (no/yes) >> NO
Display Current Audit Point Information (no/yes) >> YES
Creator Name: dbauser
Lockpoint: Log Sequence No (5)   Page No (327)
No. of Partition Instances: 3

Audit      Partition  Global      Newlog      Local
Name       Number      CommitID    Timestamp   CommitID
-----
DBE1       DEF         0000000400000121  2C13D50F000896EB  0000000400000121
DBE1       1           0000000A00000105  2C13D50F000896EB  0000000A00000105
DBE1       2           00000004000000E0  2C13D50F000896EB  00000004000000E0

SQLAudit >>
```

Refer to the SHOW AUDITPOINT command for an explanation of the display of current audit point information.

Establishing SQLAudit Log Locks

Since audit log records for transactions are written to the same DBEnvironment log files as nonaudit log records, it is possible that transactions that have not been audited can be overwritten. To prevent this, the LOCK AUDITPOINT command can be used to prevent users from overwriting transactions that you have not audited.

While it is possible to have many different audit points defined for a DBEnvironment, it is only possible to have one audit point locked for the DBEnvironment. Therefore, only one database administrator should be assigned to determine which audit point should remain locked. This log lock defines the point in the log file which no user can write beyond. If any user transaction attempts to write beyond this point, a LOG FULL message is issued and the transaction is rolled back.

The audit point can be unlocked with the UNLOCK AUDITPOINT. Since only one audit point can be locked for the DBEnvironment, locking a new audit point automatically unlocks any previously locked audit point. Audit points can also be locked when they are defined through the GET AUDITPOINT command.

Performing an Audit

Assume that a company (perhaps a bank) wants to determine what changes have been made to their data during the course of a business day. They have previously enabled audit logging for the DBEnvironment and are about to start their business day.

The first thing that must be done is to determine the beginning audit point in the DBEnvironment log files. This point will be used as the starting point of the audit process used at the end of the business day.

Since the bank does not want to lose any information before they audit it, they also lock the beginning audit point to make sure the transactions are not overwritten. This is done with the SQLAudit command GET AUDITPOINT as follows:

```
SQLAudit >> SET DBENVIRONMENT BankDBE
SQLAudit >> GET AUDITPOINT
Audit Point File >> STARTPT
Lock Log for Current Audit Point (no/yes) >> YES
Display Current Audit Point Information (no/yes) >> NO
```

This sequence of commands creates a file called StartPt containing the beginning audit point information and locks the log files to prevent unaudited transactions from being overwritten.

Since log files are being locked, the database administrator makes sure that they have enough log space in the DBEnvironment to last the full day. If they find that they are running out of log space during the day, additional log files will be added.

At the end of the day the bank wishes to audit the changes that have been made to the data during the course of the business day. Since the AUDIT command needs an ending audit point, the following sequence of commands can be used to define it:

```
SQLAudit >> SET DBENVIRONMENT BankDBE
SQLAudit >> GET AUDITPOINT
Audit Point File >> ENDPT
Lock Log for Current Audit Point (no/yes) >> NO
Display Current Audit Point Information (no/yes) >> NO
```

Notice that this audit point is not locked since that would drop the lock on the previous audit point and allow transactions committed between the two audit points to be overwritten.

Now the actual auditing of the database changes can be performed as follows:

```
SQLAudit >> AUDIT

Beginning Audit Point File >> STARTPT
Ending Audit Point File >> ENDPT
Result File to be generated >> RESULTS
Do you wish to specify Partition Numbers (n/y) >> no

Generating Results ...

Records Audited: 10000   Records In Result File: 10000
Records Audited: 20000   Records In Result File: 20000
Records Audited: 21427   Records In Result File: 21427

Finished Generating Results.

SQLAudit >>
```

If the bank wants to do continuous auditing, the ending point of one day can be used as the starting point for the next day. You should lock the new starting audit point so that no transactions get overwritten, as illustrated in the following example:

```
SQLAudit >> LOCK AUDITPOINT

Audit Point File >> ENDPT
Lock Log for Audit Point (n/y) >> yes

SQLAudit >> :Purge StartPt
SQLAudit >> :Rename EndPt,StartPt
```

By using this process every day, you should always be able to audit the last 24 hours without the transactions ever being overwritten.

SQLAudit Result Files

The results of an audit are put into a file in a user readable format. The following example shows what the contents of this file could look like for an audit of partition number 2:

```
***** SQLAUDIT: GENERATING RESULTS *****
Creator: dbauser   Creation Time: 1993-05-11 14:22:16.531
BEGIN
INSERT (2)  USER1.TABLE1  (123, 'test data', NULL, 1.23)
UPDATE (2)  USER1.TABLE1  (123, 'test data', NULL, 1.23) ((3) 0x0000123C)
COMMIT User: USER1      Label: TRANS1   Time: 1993-05-11 10:15:00.123
BEGIN
DELETE (2)  USER1.TABLE1  (123, 'test data', 0x0000123C, 1.23)
COMMIT User: USER1      Label: TRANS2   Time: 1993-05-11 10:15:01.455
End of File
```

In this example, the first transaction audited shows that a record was inserted into table USER1.TABLE1 and that the third column in the record was then updated. In the next transaction, the record was deleted. The number in parenthesis following the operation type is the partition number that the operation was performed against (partition number two in this case).

For performance and log space reasons, some information is not contained in the audit log record. For example, column names of the table being updated are not contained in audit log records. Since log records are being audited rather than an active DBEnvironment, it is not possible to get the column names from the system catalogs. For example, a table may have been dropped and recreated a number of times between the time the audit log record was created and the time you run SQLAudit. Therefore you should keep the following points in mind:

- DATE, DATETIME, TIME, INTERVAL, and DECIMAL data are printed in the hexadecimal format.
- DDL commands (as they are categorized under the audit elements DEFINITION, STORAGE, AUTHORIZATION, and SECTION) return only the operation type such as whether the command was a CREATE TABLE or GRANT, for example.
- Long fields in a record are implemented as a pointer to the data. The pointer value is printed, not the data itself.

Checking the Exit Status

When running ISQL, SQLUtil, SQLGEN, or SQLMigrate in a script, you can check the exit status to ensure that the utility completed successfully. The exit status is set to the number of DBERRs encountered. If a DBERR is not encountered, the exit status is set to 0. The exit status may contain a value in the range 0 - 255.

System Administration for ALLBASE/SQL

The DBA must cooperate with the system administrator on several crucial matters. These include:

- Shared Memory Usage
- Native Language Support
- Network Administration

Shared Memory Usage

Since ALLBASE/SQL may have to coexist on a system with other applications, the amount of available shared memory for log buffers, data buffers, and runtime control blocks must be negotiated with the system administrator. Refer to the section on “Estimating Shared Memory Requirements” in the “Physical Design” chapter.

Native Language Support

ALLBASE/SQL lets you manipulate databases in a wide variety of native languages in addition to the default language, known as **NATIVE-3000**. You can use either 8-bit or 16-bit character data, as appropriate for the language you select. In addition, you can always include ASCII data in any database, since ASCII is a subset of each supported character set. The collating sequence for sorting and comparisons is that of the native language selected.

You can use native language characters in a wide variety of places, including:

- character literals
- host variables for CHAR or VARCHAR data (but not variable names)
- ALLBASE/SQL object names
- WHERE and VALUES clauses

If your system has the proper message files installed, ALLBASE/SQL displays prompts, messages and banners in the language you select, and it displays dates and time according to local customs. In addition, ISQL accepts responses to its prompts in the native language selected. However, regardless of the native language used, the syntax of ISQL and SQL commands—including punctuation—remains in ASCII.

Note that MPE/iX does not support native language file names or DBEnvironment names.

In order to use a native language other than the default, you must do the following:

1. Make sure your I/O devices support the character set you wish to use.
2. Set the MPE job control word NLUSERLANG to the number (*LangNum*) of the native language you wish to use. Use the following MPE/iX command:

```
SETJCW NLUSERLANG = LangNum
```

This language then becomes the *current language*. (If NLUSERLANG is not set, the current language is NATIVE-3000.)

3. Use the LANG = *LanguageName* option of the START DBE NEW command to specify the language of a DBEnvironment when you create it. Run the MPE/iX utility program NLUTIL.PUB.SYS to determine which native languages are supported on your system. Here is a list of supported languages, preceded by the *LangNum* for each:

0 NATIVE-3000	9 ITALIAN	52 ARABICW
1 AMERICAN	10 NORWEGIAN	61 GREEK
2 C-FRENCH	11 PORTUGUESE	71 HEBREW
3 DANISH	12 SPANISH	81 TURKISH
4 DUTCH	13 SWEDISH	201 CHINESE-S
5 ENGLISH	14 ICELANDIC	211 CHINESE-T
6 FINNISH	41 KATAKANA	221 JAPANESE
7 FRENCH	51 ARABIC	231 KOREAN
8 GERMAN		

Resetting NLUSERLANG while you are connected to a DBEnvironment has no effect on the current DBE session.

Network Administration

Administering the network that permits operation of a distributed database is a task that may involve both the DBA and the system administrator. You manage the network using a utility program called NETUtil, as explained in the next paragraphs.

For an end user to connect to a remote DBEnvironment, two files must exist: the **AliasDB file** and the **NETUsers file**. The AliasDB file, residing on the client node, contains an entry for each remote DBEnvironment that is available on the network. The entry contains an alias name for the DBEnvironment along with the server node and DBEnvironment name on the server. The NETUsers file, residing on the server node, contains an entry for each user on a client node that has access to a DBEnvironment on the server node.

Using NETUtil

The system administrator uses a utility called **NETUtil** on the client and server nodes to add entries to and to maintain the AliasDB and NETUsers files. Each file is automatically created when you add the first entry to it. On the client node, the system administrator invokes NETUtil and uses the ADD ALIAS command to create an entry in the AliasDB file. All NETUtil commands prompt the user for information, so in this case, the administrator is prompted for details about the specific DBEnvironment to be accessed. (The entry added to the AliasDB file will contain the answers to the prompts.)

On the server node, the system administrator invokes NETUtil and uses the ADD USER command to create an entry in the NETUsers file. Again, the administrator will be prompted by NETUtil for the appropriate information. The entry added to the NETUsers file will include specifics about the user who will be accessing the remote DBEnvironment.

Additionally, users of DBEnvironments can invoke NETUtil to display the entries in the AliasDB file so that alias names of DBEnvironments can be checked or confirmed.

Complete information about NETUtil is found in the *ALLBASE/NET User's Guide*.

Logical Design

An ALLBASE/SQL database is defined both logically and physically. Logically, an ALLBASE/SQL database is one or more tables or views with the same owner. This chapter will give guidelines on how to design the logical structure of a relational database and to use ALLBASE/SQL table, view, and index definition features to meet the needs of users and applications. The logical design process lets you create efficient databases so that your applications will perform smoothly and be easy to maintain and support. The process consists of several distinct steps:

- Identifying the data for tables
- Normalizing the data
- Arranging data in tables
- Designing views
- Designing indexes
- Designing hash structures
- Designing integrity constraints
- Defining security levels
- Defining the DBEnvironment scope

The following sections describe each of these steps using the sample DBEnvironment PartsDBE as an illustration.

Identifying the Data for Tables

Before designing the tables, indexes, and views of the database, you must determine what the data is. Where will you get the data? How will the data be used? Consider the following test case.

Integrated Peripherals Incorporated uses purchase order documents that track the purchasing of parts from external vendors. The goal of the Data Management Group is to create an automatic process for tracking purchases, paying accounts, mapping external parts to internal parts, and keeping track of internal part stock levels.

Figure 2-1 shows the purchase order used by Integrated Peripherals.

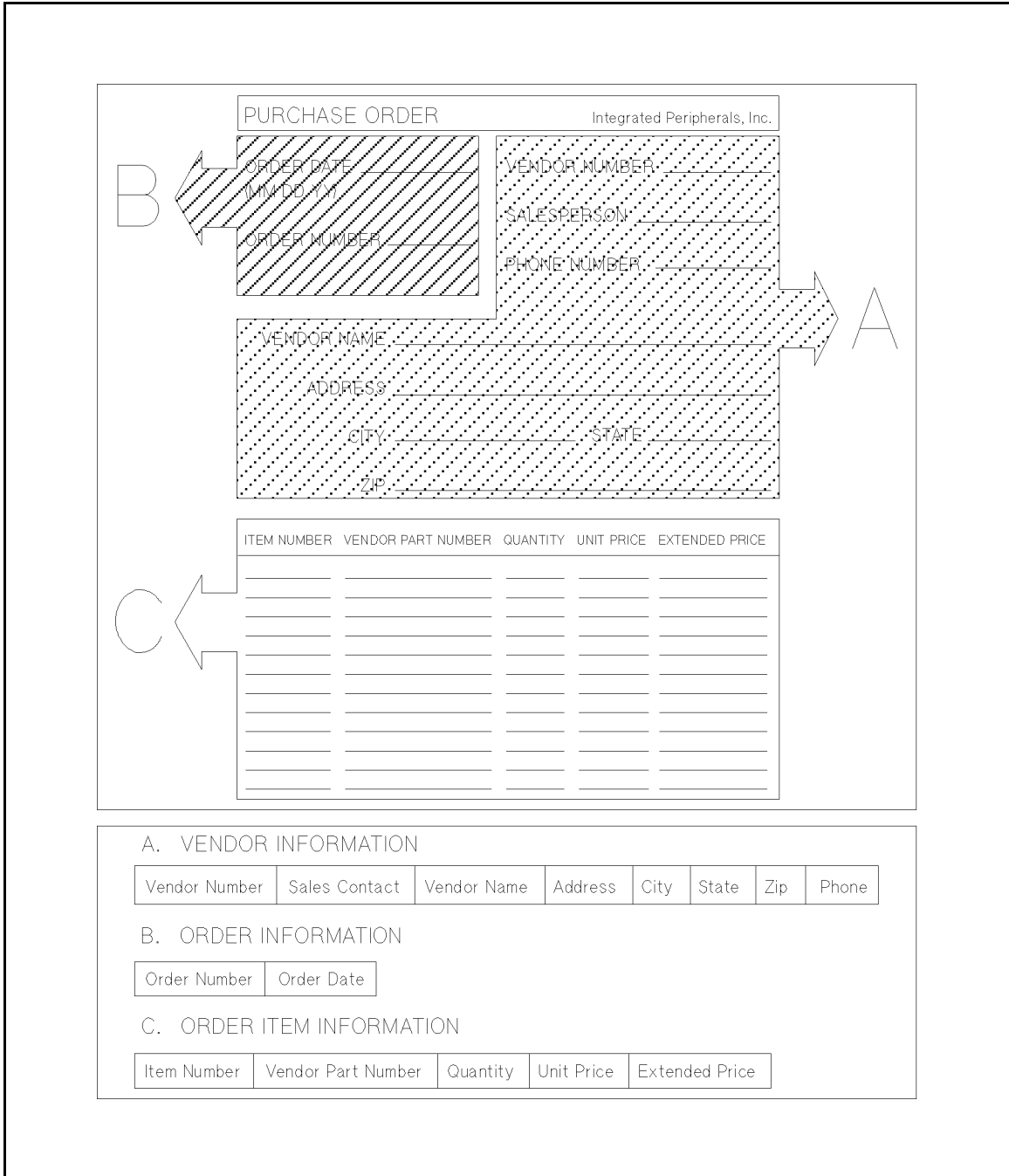


Figure 2-1. Identifying Data Categories

You can use similar documents as a starting point for designing the tables in your own databases. Data from the purchase order can be arranged in categories.

Once the data has been identified, it must be broken up into tables. This is done by identifying **entities**, **attributes**, and **keys** within the data. An entity is a category of information to be tracked by the database; an attribute is a characteristic of an entity; and a key is one or more attributes that uniquely identify a member of an entity.

For example, on the purchase order for Integrated Peripherals, the database designer might identify the following general groups of information as entities:

- Vendor information
- Order information
- Order item information

The Vendor Information entity has several attributes, including vendor number, vendor name, vendor address, sales contact, and phone number. The vendor number attribute might be identified as a key for the Vendor Information entity, since it is a unique code that distinguishes vendors from one another.

Make a list of the entities used by your organization in creating applications. For each of the entities, list the attributes, including keys. Refer to Figure 2-2 for the entities, attributes, and keys taken from the purchase order used by Integrated Peripherals.

ENTITIES	ATTRIBUTES	KEYS
Vendor Information	Vendor Number Vendor Name Sales Contact Address Phone	Vendor Number
Order Information	Order Number Order Date	Order Number
Order Item Information	Order Number Item Number Vendor Part Number Quantity Unit Price Extended Price	Order Number Item Number

Figure 2-2. Entities, Attributes, and Keys

The entities become tables, and the attributes and keys become the columns in the tables. Each entity must have a unique value or combination of values for each row in the database table in which it will appear. This value or combination is known as the **primary key**.

Note that the key for order item information is composed of two items. This is because neither the *Item Number* nor the *Order Number* is significant by itself. Remember that a primary key must be unique. Item number alone cannot be the primary key because every order can have an item #1. The order number alone cannot be the primary key either because one order can have several items, forcing the order number to be repeated (not unique). Therefore, the two pieces of information are combined to yield a unique key.

Once the data has been defined and put into groups, it can be normalized to produce the tables that will become the database.

Normalizing the Data

Normalization is a process that helps you arrange data into tables. The goal of normalization is to reduce data redundancy and facilitate efficient updates. In theory, a perfectly normalized database uses data at maximum efficiency and minimum redundancy. In practice, however, normalization does not take into consideration the specific applications of a database. Consequently, normalization is recommended only as a guideline for database design. Use normalization to get your data into relational format (two-dimensional tables) and then alter the table format to fit the specific needs of users and applications.

The process of normalization entails examining the data and the table format of a database against each of the criteria for each of the **normal forms**. Although there are at least five normal forms, most database tables need to be processed only to the second or third normal form. Therefore, only the first three normal forms are discussed here. When data is normalized, each lower normal form is a prerequisite to the next higher normal form. In other words, data must be in first normal form before it can be put into second normal form.

First Normal Form

In **first normal form**, data contains rows that have the same number of columns. Unnormalized data is not so orderly. For example, some purchase orders contain one order item, others contain two or more. Thus, if each order number from the purchase order in Figure 2-1 were to occupy a single row, each row would have a different number of columns. By separating the data into three groups as in Figure 2-3, repeated categories are removed, leaving each row with the same number of columns.

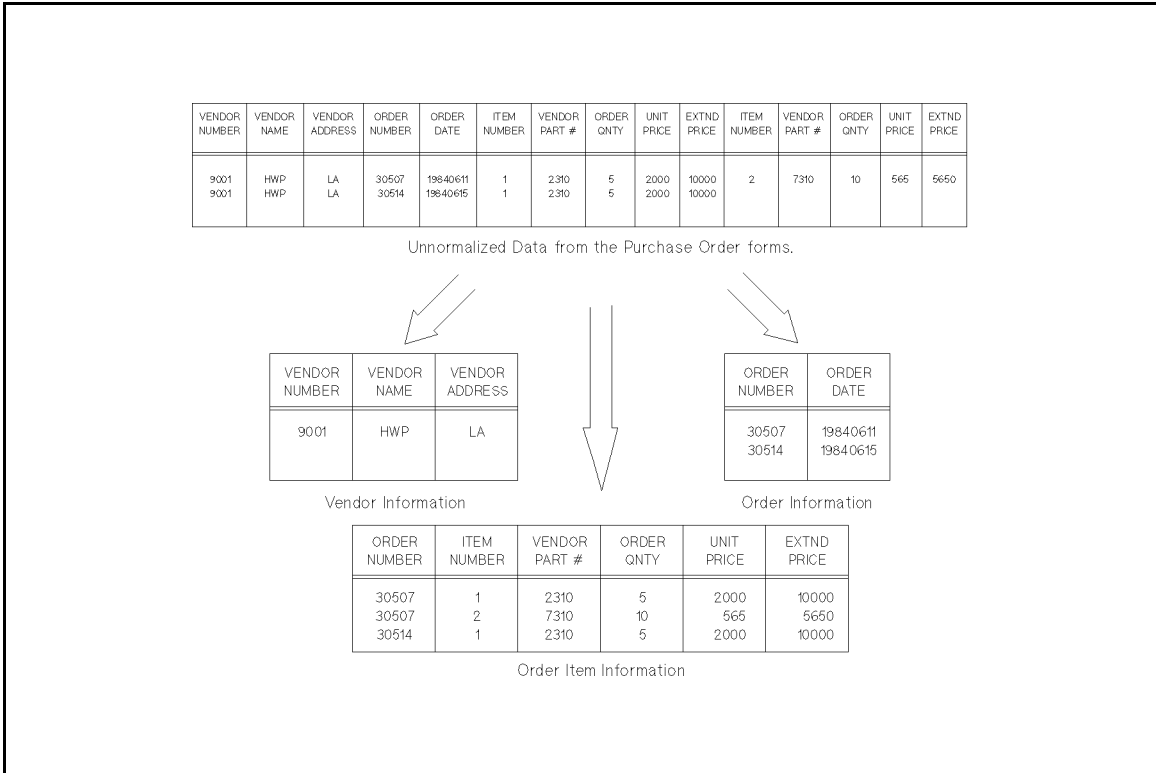


Figure 2-3. First Normal Form: Removing Repeating Groups

Second Normal Form

In **second normal form**, shown in Figure 2-4, all attributes are functionally dependent on the primary key.

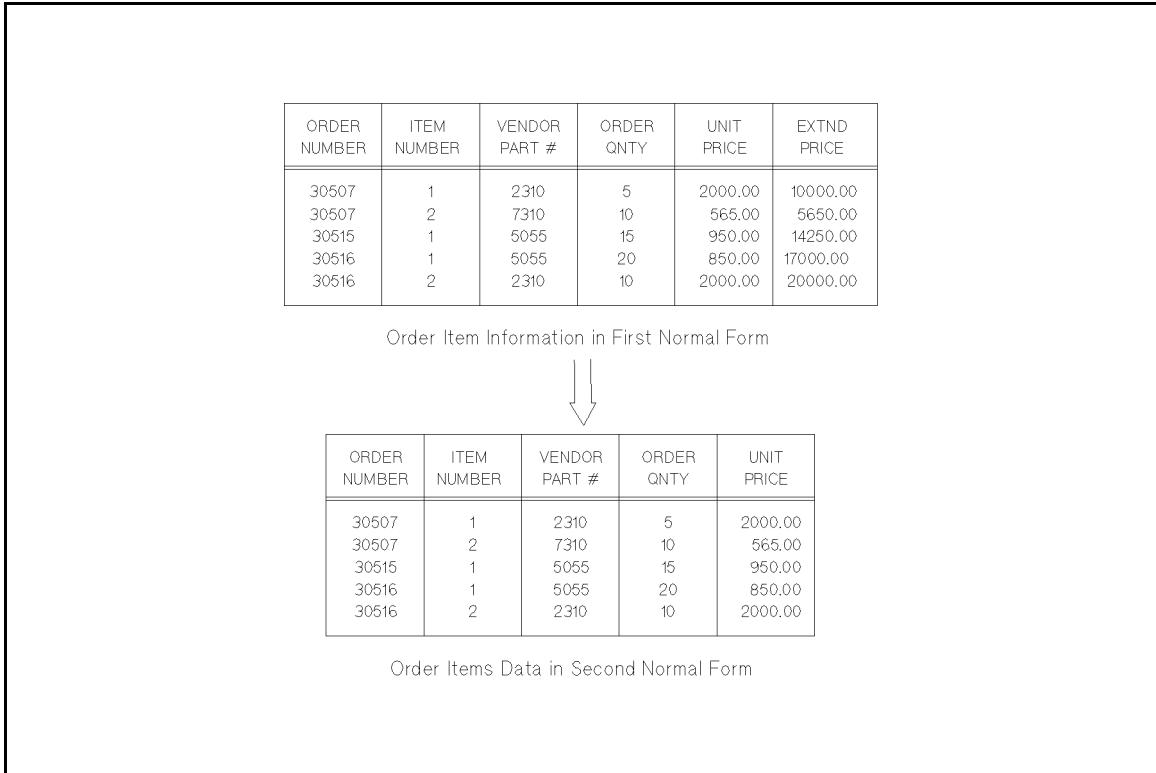


Figure 2-4. Second Normal Form: Establishing Dependency

Attributes are **functionally dependent** when their values depend directly upon the value of the primary key. The primary key in the Order Item information is the order and item number combination. Each such combination represents a unique row.

Since the vendor part number and the order quantity relate to a particular order and item number, they are functionally dependent on the primary key. However, the unit price depends on the order quantity (which, in turn, depends on the order and item number), but it does not depend directly on order or item number. This is called a **transitive dependency**.

Third Normal Form

In third normal form, data does not have transitive dependencies. In Figure 2-5, unit price is removed from the Order Items information and placed in a new set of information called Supply Price, which is used to calculate the price of an item dependent on the quantity ordered and date delivered.

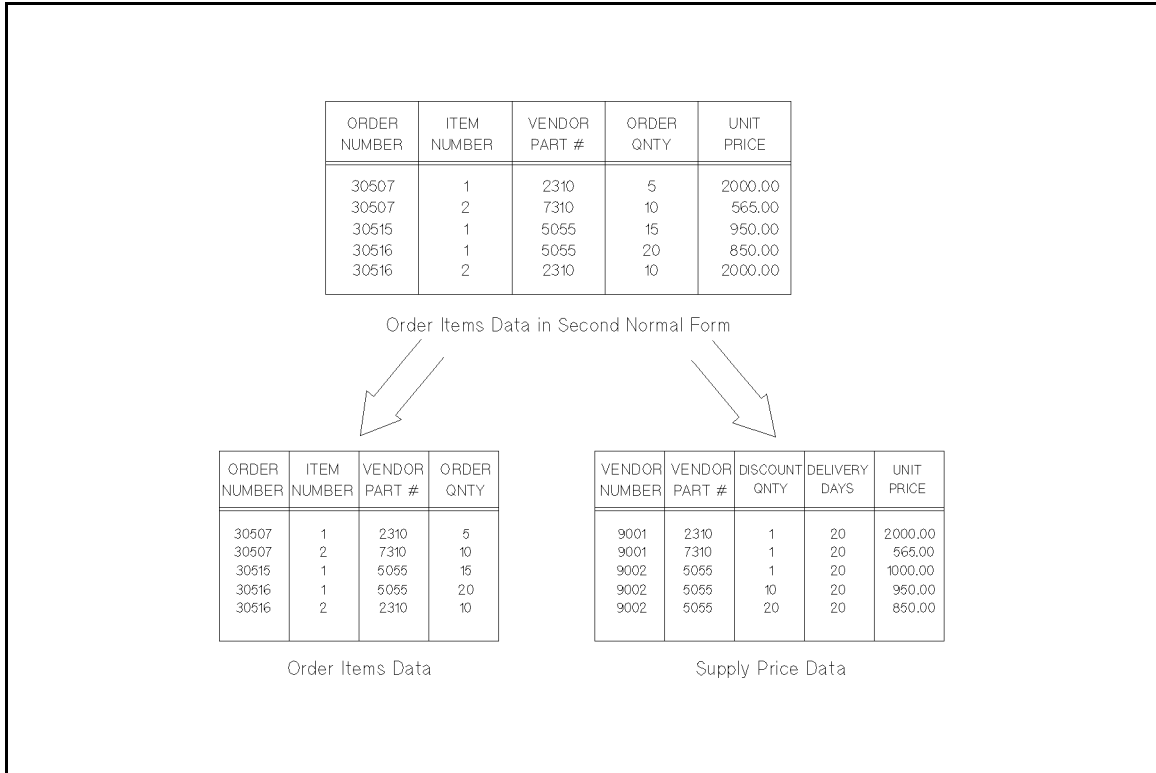


Figure 2-5. Third Normal Form: Removing Transitive Dependencies

Third normal form relieves update problems. For example, in second normal form, when an item's price changes, all orders containing that item would have to be altered. In third normal form, only the rows for that item in the Supply Price table need to be changed.

Normalized data provides a starting point for table definition. Note that the data in Figure 2-3, Figure 2-4, and Figure 2-5 is in tabular form. The next step in table design is to take the normalized data and fit it into ALLBASE/SQL columns, tables, views, and indexes.

Arranging Data in Tables

The basic data structure in an ALLBASE/SQL database is a **table**. Data is stored within a table as rows divided into columns. Indexes and views are created using the columns of a table.

To design the tables in your database from normalized data, you must:

- define tables from the normalized data according to the uses of the data in your applications
- define the columns in the tables

Defining Tables

To design tables you need to know how the applications use data in SELECT, DELETE, INSERT, and UPDATE operations. Data that is deleted, inserted, or updated at the same time should be put in the same table. Use these criteria to start grouping your data into tables.

To determine the composition of your tables, keep the following guidelines in mind:

- The maximum number of tables you can define in a DBEnvironment is $2^{31} - 1$.
- The maximum number of columns in a table is 1023.
- The maximum length of a column is 3996 bytes.
- The maximum length of a row in bytes is

$$(NC + 1) * 2 + SCL \leq 4000$$

where NC is the number of columns in the table and SCL is the sum of the column lengths.

- If you add a column to a table using the ALTER TABLE statement, the added column is placed on the right-hand side of the table.

The following paragraphs are a few more guidelines to help you determine table design.

A row is the smallest unit you can delete or insert at a time. This means two columns should be placed in different tables if they can be inserted or deleted at different times. Assume a part name is never inserted or deleted unless a corresponding part number is inserted or deleted. Part name and part number should be in the same table. On the other hand, a vendor name can be inserted or deleted independent of, say, the order number. Therefore, the vendor name and order number should be placed in separate tables.

Interactively you can only use the UPDATE statement on one table at a time. This means if you use a single statement to update one column based on the value of another column, both columns must be in the same table. The unit price of a part is updated as it pertains to a particular part number column; therefore, the unit price and part number columns should be in the same table.

Planning Joins

A **join** is a query that selects columns from at least two tables. If many applications request information that can logically reside in two tables, you may want to place the information in a single table to improve performance. A table with a large number of columns can impair performance as will several smaller tables that are joined frequently.

The WHERE clause of the SELECT statement is used to specify the condition(s) under which rows are joined. ALLBASE/SQL allows joins on compatible data types, but for maximum efficiency, joins should be performed on identical columns.

Note An application consistently containing queries that join more than six tables indicates that the tables are over-normalized and the database design should be re-examined.

Tables can be joined as Cartesian products where each row of one table is joined with every row in another table. However, in order for tables to be meaningfully joined, they must share a common column. If two tables do not share a common column, then a third table containing common columns for both tables must be introduced into the join, as shown in Figure 2-6.

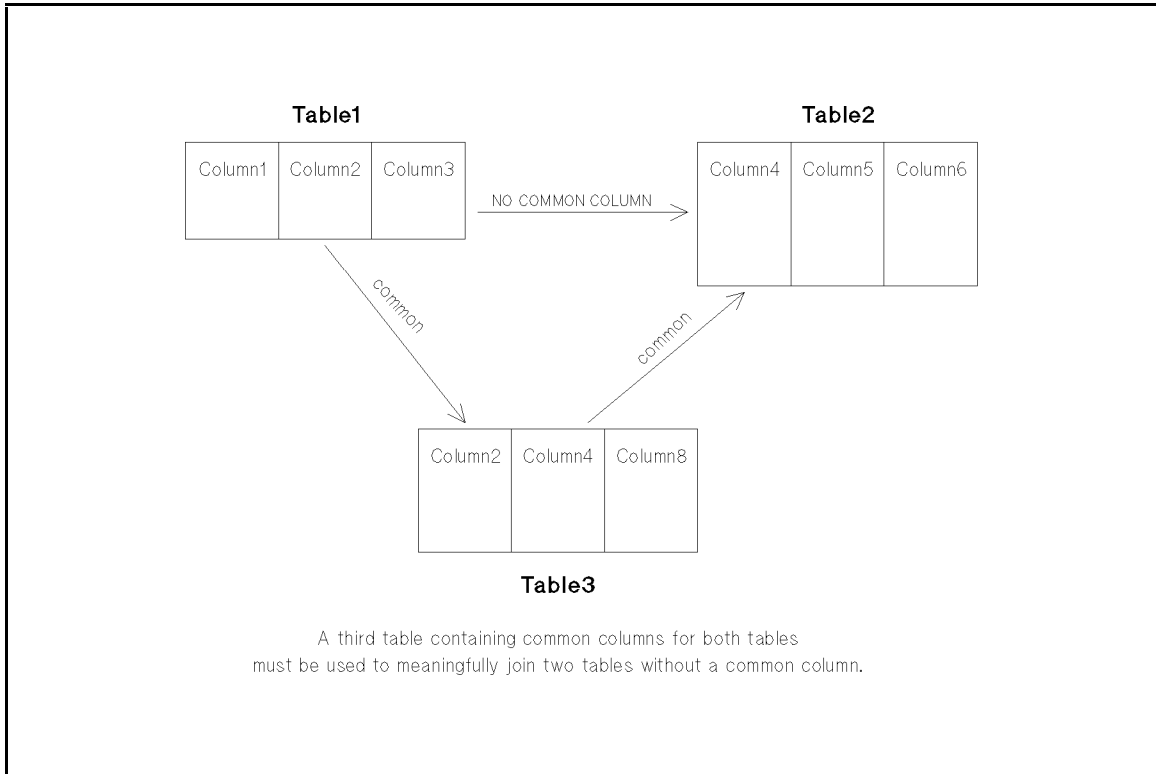


Figure 2-6. Common Columns for Joins

Several of the sample database tables must be joined to provide the required data to the users and applications. However, the normalized data does not allow some tables to be joined directly. Refer to Figure 2-3. The OrderItems table cannot be meaningfully joined with the Vendors table because there is no common column in the tables. To facilitate retrieval for those applications that use the Vendors and OrderItems tables, the VendorNumber column was added to the Orders table, as shown in Table 2-1. This allows a join to be made between the Vendors and OrderItems tables using the Orders table.

Adding the VendorNumber column, of course, violates the third normal form. However, normalization is just a tool that helps you design efficient tables. It is up to the DBA to alter the table design to meet the needs of the users.

For additional information on joins, including Cartesian products, refer to the "SQL Queries" chapter of the *ALLBASE/SQL Reference Manual*.

Final Form of Sample Database Tables

The normalized data from the purchase order form produced the Vendors, Orders, OrderItems, and SupplyPrice tables. Integrated Peripherals, Inc. also designed a Parts table and an Inventory table to keep track of the internal parts. As a result, a total of six tables were designed from the defining and normalization phase.

Columns are added to some of the tables to fit the user's needs. A VendorRemarks column is added to Vendors to keep comments for that vendor. A ReceivedQty column is added to the OrderItems table to denote when a shipment arrives. The resulting six tables are shown in Table 2-1.

Table 2-1. Sample Database Tables

TABLE	COLUMNS
Parts	PartNumber, PartName, SalesPrice
SupplyPrice	PartNumber, VendorNumber, VendPartNumber, UnitPrice, DeliveryDays, Quantity
Vendors	VendorNumber, VendorName, ContactName, PhoneNumber, VendorStreet, VendorCity, VendorState, VendorZipCode, VendorRemarks
Orders	OrderNumber, VendorNumber, OrderDate
OrderItems	OrderNumber, ItemNumber, VendPartNumber, PurchasePrice, OrderQty, ItemDueDate, ReceivedQty
Inventory	PartNumber, BinNumber, QtyOnHand, LastCountDate, CountCycle, AdjustmentQty, ReorderQty, ReorderPoint

Defining Columns

Each attribute of each entity is defined in ALLBASE/SQL as a **column** in a table. The “Data Types” chapter in the *ALLBASE/SQL Reference Manual* contains basic information on column names and data types. Use the information presented there and the following guidelines to define columns. A column is defined by specifying:

- Name
- Data type
- Size
- Whether or not it can contain null values

Defining Column Names

Be precise in your names and descriptions so no user can misunderstand a data element or its meaning. Each column name can be as long as 20 characters. Try to choose meaningful, unambiguous names. For example, a name of “Qty” in the OrderItems table could refer to order quantity or received quantity while “OrderQty” can refer only to order quantity. Long names, however, may become unwieldy in a report. In the sample database, the column and table names have been made as unambiguous as possible.

Defining Column Data Type

A data type tells ALLBASE/SQL what type of data can be stored in the column and what can be done with the column. Thus, a column with an INTEGER data type can only have integers. A column with a numeric data type (FLOAT, DECIMAL, INTEGER, or SMALLINT) can be used in arithmetic operations. A column with an alphanumeric data type (CHAR or VARCHAR) can appear in a string operation such as a comparison using the LIKE predicate.

2-10 Logical Design

Some guidelines for data types are:

- Match the data type to the data, for example:
 - Columns used for quantity or count should be defined as `INTEGER` or `SMALLINT`.
 - Columns containing real numbers or engineering data should be defined as `REAL` or `FLOAT`.
 - Columns used for money should be defined as `DECIMAL` so there is no loss of precision when rounding.
 - Columns containing alphanumeric characters should be defined as `CHAR` or `VARCHAR`. `VARCHAR` is used for particularly large alphanumeric columns like descriptions or comments. The `VARCHAR` data type is recommended when there are a few potentially large character strings, but most of the time the entire column will not be filled. The `VARCHAR` data type stores character data more efficiently because the column is not padded with blanks. The `VendorRemarks` column in the `Vendors` table is defined as `VARCHAR`.
 - Columns containing binary data or data of an unspecified kind should be defined as `BINARY` or `VARBINARY`. The `VARBINARY` data type is recommended when there are a few potentially large binary strings, but most of the time the entire column will not be filled. The `VARBINARY` data type stores binary data more efficiently because the column is not padded with zeroes.
 - For very large column values (greater than 3996 bytes), use the `LONG BINARY` or `LONG VARBINARY` data type.
 - Columns used for dates should be defined as `DATE`.
 - Columns used for times should be defined as `TIME`.
 - Columns used for timestamps should be defined as `DATETIME`.
 - Columns used for intervals should be defined as `INTERVAL`.
- Avoid data conversions by using the same data type for columns that are frequently compared or used in a `WHERE` clause. For example, if you compare two columns frequently, do not define one as `INTEGER` and the other as `FLOAT`. `ALLBASE/SQL` will have to convert one number to the data type of the other which could result in performance degradation and possible loss of precision.
- `CHAR` and `VARCHAR` columns are defined with a length in bytes. Therefore, if your log-on language is a 16-bit language such as Chinese, the number of two-byte characters you can store will be equal to half the column width.

Columns of type `VARCHAR` or `VARBINARY` may cause an extra **tuple header** to be stored. A tuple header is a description of the rows on a DBEFile page. If all rows on the page are the same, the header can be shared. A `VARCHAR` or `VARBINARY` column may be a different length in each row thus requiring each row to have its own tuple header. The calculations in Chapter 3, “Physical Design”, assume that each row has its own tuple header. Refer to the *ALLBASE/SQL Reference Manual* chapter on “Data Types” for further explanation.

The use of `VARCHAR` or `VARBINARY` data types can result in **page shifting**, that is, movement of data from one page to another when the size of the data changes. This can be a drain on performance.

Defining Column Size

You should always define your columns to be large enough to hold the largest piece of information you expect it will ever hold. This helps to avoid restructuring of tables at a later time. Column size affects both physical design and logging.

Rows are stored in 4096-byte DBEFile pages. After the space for a page header is used, 3996 bytes are left for data storage. During physical design row length is used in calculating the number of DBEFile pages needed to store data for a table. The sum of all columns plus a two byte overhead for each column is equal to the total row length:

$$RL = SC + (2 * NC)$$

where RL is row length, SC is the sum of the length of all columns in the table, and NC is the number of columns in the table.

Since a page has 3996 bytes for storage, a row of 2000 to 3000 bytes is going to waste space by taking one half to three quarters of a page and leaving the rest of the page empty. Column size can be adjusted to use pages more efficiently, or a large table can be broken into two smaller tables to improve page use. This is also discussed under “Calculating Row Length” in the “Physical Design” chapter in this guide.

Large columns that are modified frequently create proportionately large log records and consequently use more log file space. When you determine log file size in the “Physical Design” chapter, keep in mind the size of the columns that are being modified.

Defining Null Values For Columns

Columns that might not always have data available should be allowed to contain null values. A column containing a null value does not store any data. Null values are distinguished from zeros and blank character data. If the UnitPrice column contains a null value, the price is considered unavailable. However, if the UnitPrice column contains a zero, the price is \$0.00 and the item is free. The default when defining columns is to allow null values. Columns not permitted to contain null values *must* be created with the NOT NULL option. Some guidelines for null values are:

- If you create a column with the NOT NULL option, you *must* specify a value for that column whenever you insert a new row or update the column.
- If a column is to be a key value in an index, you should define it as NOT NULL.
- If an update application will not necessarily have all the information available, you should allow null values in the non-key columns (that is, omit the NOT NULL option for those columns).

There are application and physical design implications with the use of null values. Application programs that use null values must declare special variables called **indicator variables** to handle the null values. Refer to the *ALLBASE/SQL Application Programming Guide* for the language you are using for details on using indicator variables.

The use of null values can result in page shifting when the size of the row changes. This can be a drain on performance. For performance improvement, use the NOT NULL option on a column definition whenever possible. If a column is potentially null, SQLCore uses a two-byte overhead per column during query processing to check the null status of every selected column. Although SQLCore also needs to check the status for every inserted and updated value if the column is defined as NOT NULL, performance is better than if the column allowed null values.

2-12 Logical Design

In addition, a two-byte **tuple header** is stored on disk for each tuple which has a NULL value if the inserted tuple's header does not match the first shared header on the DBEFile page. The data types and sizes of the columns for the OrderItems and Vendors tables are shown in Table 2-2.

Table 2-2. Column Attributes for Two Tables

TABLE	COLUMN ATTRIBUTES		
OrderItems	OrderNumber	INTEGER	NOT NULL,
	ItemNumber	INTEGER	NOT NULL,
	VendPartNumber	CHAR(16),	
	PurchasePrice	DECIMAL(10,2)	NOT NULL,
	OrderQty	SMALLINT,	
	ItemDueDate	CHAR(8),	
	ReceivedQty	SMALLINT	
Vendors	VendorNumber	INTEGER	NOT NULL,
	VendorName	CHAR(30)	NOT NULL,
	ContactName	CHAR(30),	
	PhoneNumber	CHAR (15),	
	VendorStreet	VARCHAR(30)	NOT NULL,
	VendorCity	VARCHAR(20)	NOT NULL,
	VendorState	CHAR (2)	NOT NULL,
	VendorZipCode	CHAR (10),	
	VendorRemarks	VARCHAR(60)	

Designing Views

A view is a virtual table defined on one or more tables or views, or a combination of tables and views. The term **base table** refers to the table in which data is actually stored. Views and query results, although they appear in the form of a table, are **virtual tables** derived from one or more base tables. ALLBASE/SQL maintains information and statistics on views in the SYSTEM.TABLE, SYSTEM.SECTION, and SYSTEM.VIEWDEF views in the system catalog, which is presented fully in the chapter, "System Catalog." The following are some primary reasons for defining a view:

- Restricting access to the base tables.
- Keeping data independent of the applications that use it.

In an interactive environment, views are commonly used to restrict access to base tables. A large table containing data from several areas can be broken up into smaller views, one for each area. The data is available to those who need it, but the entire table is secure from unauthorized users. For example, in a personnel database, a view containing employee names, extensions, and locations can be derived from a table that also contains home addresses, salary, and other information to which access should be restricted. Users can then be granted access to the views without being able to access the sensitive data in the base tables. Refer to Chapter 5, "Database Creation and Security," for information on coordinating view definition with security design.

Data independence is usually a concern in programmatic environments. An application that accesses several tables would have to be modified each time the tables were altered. A view defined to look like the old tables would keep applications independent of database changes. Refer to the appropriate *ALLBASE/SQL Application Programming Guide* for more information on how views affect and are manipulated by application programs.

View definition is not restricted to base tables. You can define a view on another view's derived columns. Refer to the *ALLBASE/SQL Reference Manual* for a complete list of restrictions on defining views.

To define a view, use the same guidelines as for defining a table (refer to “Defining Tables” in this chapter), plus the following:

- ALLBASE/SQL does not limit the number of views you can define.
- A view cannot be created with an ORDER BY clause.
- You cannot define an index on a view.
- You can define as many as 1023 columns per view.
- You can use up to 32 base tables to define a view, which includes the tables that comprise a view from which another view is defined.

You can manipulate data through views as you would through tables, but certain restrictions apply. In general, you cannot INSERT, DELETE, or UPDATE through a view if any of the following are used to define it:

- Multiple tables (joins).
- A GROUP BY clause.
- An aggregate function (AVG, MAX, MIN, SUM, COUNT).
- A DISTINCT in the SELECT clause.
- Defined fields in the SELECT clause, e.g., ColumnA + ColumnB. In this case, DELETE is allowed. For more information, refer to “Updatability of Queries” in the “SQL Queries” chapter of the *ALLBASE/SQL Reference Manual*.
- A UNION clause.

A view is essentially a stored SELECT statement. Therefore you cannot alter a view by adding columns to it. To add a column to a view, simply delete the old view and create a new view, specifying the additional column in the SELECT clause.

For the most part views are used to restrict data access. However, views are actually an additional layer to the base tables. This means that each time a view is used, ALLBASE/SQL must perform an additional step to build the view before the user gets the data. Preprocessing applications also takes longer if the application contains views because the system catalog must be accessed twice: once for the view and once for the base tables on which the view is defined. Ultimately, the DBA must weigh the factors for and against creating views depending on the needs of the users.

Designing Indexes

An index is defined on one or more columns in a table to facilitate rapid retrieval of data. ALLBASE/SQL decides whether to use an index in the process of creating an optimal access path for the data during **query optimization**. By default, you cannot specify the use of an index in an SQL statement; the optimizer will decide to use an index if it will improve performance during query processing. However, if you use the SETOPT statement you can override the optimizer and specify the use of an index.

ALLBASE/SQL uses a **B-tree** (balanced tree) design for indexes, as shown in Figure 2-7.

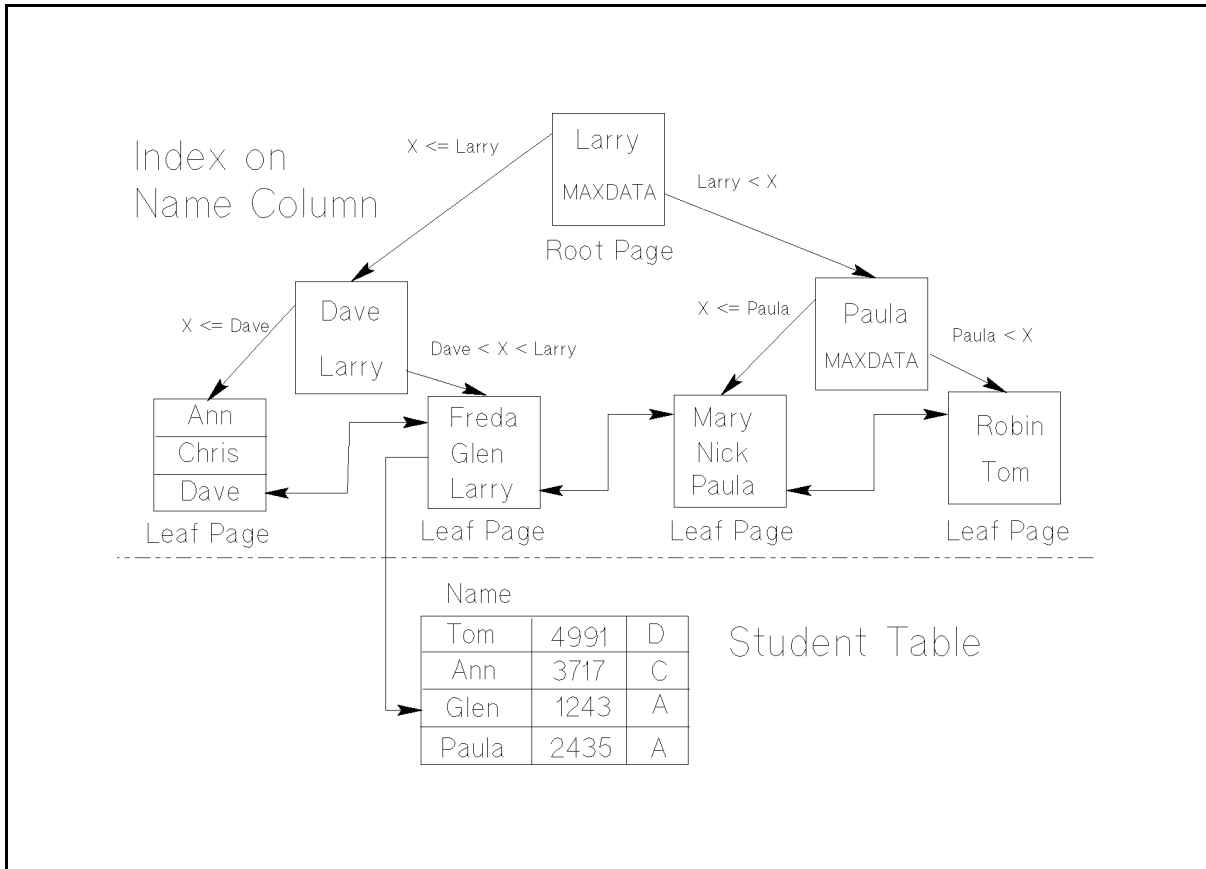


Figure 2-7. B-Tree Index Design

The tree has different levels:

- **Root level**— a single root page containing key values that subdivide the index search.
- **Intermediate levels**— pages containing key values that further subdivide an index search.
- **Leaf level**— pages which point to data values in the indexed column or columns.

When an index is used to access a row on the basis of a key value, ALLBASE/SQL first looks at the root page, then follows pointers down the tree until it finds an appropriate leaf entry, which points to a specific location in a data page. In Figure 2-7, the root page contains one subdividing value, *Larry*. In order to find student *Glen*, we would first examine the root page, which points to the leftmost page on the next level, since *Glen* comes before *Larry* in this collating sequence.

This page in turn points to the second leaf page from the left, since *Glen* comes after *Dave* and before *Larry*. Finally, the leaf page points to a specific data page where the desired row is found.

Note that ALLBASE/SQL B-tree indexes are **doubly linked**, in that the leaf pages point to other leaf pages, so that the next higher or lower value in key order can be quickly located without “backing out” to a higher level in the tree.

Keep in mind the following general guidelines when designing indexes:

- Indexes can *only* be defined on tables.
- An index can only be defined on *one* table.
- The maximum number of columns allowed per index is 16.
- There is no limit to the number of indexes you can define.
- Too many indexes can degrade performance because when a row is inserted, deleted, or updated in a table, the indexes must also be updated.
- A small table may not need an index.
- Cyclic applications requiring multiple access paths (for example, accounting applications for quarterly and annual reports) can create indexes, then drop them when finished. This reduces index overhead and maintenance and improves application performance. DDL must be enabled for this approach to be possible.
- An index may improve the performance of queries containing a WHERE clause involving a comparison on the first column of a multicolumn index key.

ALLBASE/SQL updates indexes automatically as rows in the table are updated, inserted, or deleted. This adds considerable overhead to the UPDATE, DELETE, and INSERT operations. ALLBASE/SQL maintains index information and statistics in the SYSTEM.INDEX view in the system catalog, which is presented fully in the chapter, “System Catalog.”

Determining Index Keys

All of the columns in an index together compose what is called a **key**. An index on a single column has a **simple key**. An index on multiple columns has a **compound key**.

Any column that you specify as a search item in a query will cause ALLBASE/SQL to look for an index with that column specified as part of a key. If no appropriate index is found, or if the optimizer decides not to use the index, ALLBASE/SQL performs a sequential search on that table. Sequential searches may degrade performance when most of the performed queries have a WHERE clause.

The ideal index has the one or two most common columns used in a WHERE clause defined as keys. Keep the following guidelines in mind when determining which columns to use for a key:

- A simple key can be as long as 1010 bytes.
- An index should contain a column that is used in the WHERE clause of a frequently executed query.
- An index should contain a column that is used to verify the existence of a value, especially if it is a unique index.

If you are considering a compound key, you should also keep the following in mind:

- A compound key can have up to 16 columns.
- The length of a compound key must be less than or equal to 1024 bytes. Use the following formula to determine whether or not a proposed compound key is acceptable:

$$2NC + KL + 18 \leq 1024$$

Where NC is the number of columns, and KL is the sum of the column lengths in the key. A single column index may not exceed 1010 bytes.

- The order in which you specify columns in the CREATE INDEX statement determines the sort ordering: the first column listed is the primary, or first, order of sort; the second column listed is second in order; and so on.

The most common index is created on the **primary key** of a table. For example, the order number is the primary key in the Orders table. Therefore an index with the OrderNumber column as a key should be created on the Orders table.

How Index Keys are Used

ALLBASE/SQL uses simple and compound key indexes differently. Suppose most of your queries have a WHERE clause for the VendorNumber and OrderNumber, but the two columns are never used in the *same* WHERE clause:

```
WHERE VendorNumber='StringConstant'  
WHERE OrderNumber=Integer
```

In this case, you should create two separate indexes, each with a simple key.

Even if the two columns are used in the same WHERE clause, but they are compared against each other, only one of the columns can be used as a search item.

```
WHERE VendorNumber=OrderNumber
```

In this case as well, you should create two indexes with simple keys.

On the other hand, suppose the two columns are always used together in a WHERE clause:

```
WHERE VendorNumber='StringConstant' AND OrderNumber=Integer
```

In this case you should create one index with a compound key. While it is not required to have an index on each table in a join, defining an index on one or both of the tables may improve performance. In general, you should define an index on the columns specified in the join. For example, if the sample database has an application that joins the SupplyPrice table with the OrderItems table, then an index with VendPartNumber (the common column) should be defined on one of the tables if not both of them.

Determining Index Type

You can design one of four different types of index:

- Unique
- Clustering
- Clustering and unique
- Neither clustering nor unique

The **unique index** will not permit duplicate values for the key columns specified. The **clustering index** attempts to place rows with similar key values physically close to each other on disk.

Defining Unique Indexes

If you specify the unique option when creating an index, the columns named in the index key are kept unique. Unique indexes prevent duplicate data in the columns used as keys of the index.

In the sample database, the Vendors table has a unique index created on the VendorNumber column to ensure that a vendor number refers to only one vendor. A table can have multiple unique indexes. However, ALLBASE/SQL will not allow you to create a unique index on a table that already contains rows with duplicate values in the key columns.

Defining Clustering Indexes

Clustering indexes are closely related to physical design of the database. To understand what a clustering index does, you must know that ALLBASE/SQL arranges data in sections of a DBEFile called **pages**. A clustering index attempts to place all rows with similar key values on the same or consecutive pages. Because the rows are physically close, I/O overhead is reduced and performance improved whenever the rows are retrieved in key order. This can be helpful with queries in which you make use of LIKE and BETWEEN predicates.

A clustering index should be defined on a table *after* the initial loading of the table but *before* any additional rows are inserted in the table. It is recommended that you sort data on the clustering key before you load it into the table. Rows inserted before a clustering index is created are not repositioned after the index is created. Therefore, if you want to create a clustering index on a table that already contains data, you should unload the table using a SELECT statement with an ORDER BY clause, reload the table, then create the clustering index.

Keep the following guidelines in mind when considering a clustering index:

- Regardless of the number of indexes you define on a table, only one index can be a clustering index.
- A clustering index can be either non-unique or unique.
- A clustering index should be based on the most commonly used sort sequence.
- Clustering benefits applications that must search large amounts of data to retrieve rows in sequential order based on a key value.
- A clustering index can be defined on an existing table; however, existing rows will not be repositioned.
- A clustering index may improve the performance of a query containing a DISTINCT, GROUP BY or ORDER BY clause if the columns in the sort list match the first columns in the index definition.
- A clustering index may improve the performance of a query containing a WHERE clause involving a comparison on the first few columns listed in the index.
- Any index makes INSERTs and UPDATEs more expensive.

- Clustering of data is maintained when there is a relatively large volume of DELETE operations followed by a similar volume of INSERT operations on the table. Clustering of data is not maintained when there is a larger volume of INSERT operations than DELETE operations.

Designing Hash Structures

Like an index, a hash structure specifies up to 16 key columns on a table to facilitate rapid access to data. But unlike an index, a hash structure is not a separate object; instead, it is a way of arranging the actual table data in DBEFile pages. To create a hash structure, you must define a hash key and specify a number of primary pages in a DBEFileSet containing one or more empty TABLE or MIXED DBEFiles.

Because of the nature of hashing, you can define only one hash structure for a single table. You can, however, define multiple B-tree indexes on the same table. You *cannot* define a clustering index, because clustering affects the physical placement of data, and this would be incompatible with the physical placement chosen by the hash algorithm.

Another consequence of hashing is the requirement that you specify a number of primary pages. When the table grows to include more data than will fit on the primary pages, new tuples are placed on *overflow pages*, which can be located in the same or in other DBEFiles within the same DBEFileSet. The larger the number of overflow pages, the slower the access to the average tuple.

Understanding the Hash Function

Each time a row is inserted into the table, a hash function maps the specified key value to a logical page in the DBEFileSet associated with the hash structure. If two keys hash to the same page (in what is called a **collision**), then space is allocated on the same page if possible for both rows. If space cannot be allocated on the same page, a new page is allocated (an **overflow page**) and linked to the old page.

To access data through a hash structure, ALLBASE/SQL calculates the correct data page location in the DBEFileSet from the key value by means of the hash function.

The hash function used by ALLBASE/SQL derives the primary page number in three steps:

1. Creates a concatenated form of the key.
2. Folds the concatenated key into a 4-byte integer.
3. Computes the primary page number, as follows:

$$\text{Primary Page Number} = (\text{Folded Key}) \bmod (\text{Number of Primary Pages} + 1)$$

The important thing to note is that hashing works best with a fairly even distribution of key values, spreading the corresponding rows evenly over the number of pages available. A key with a skewed distribution will attempt to place all rows on a correspondingly skewed set of pages. For good results, use a prime number of primary pages when you are hashing on a non-integer key.

Choosing Hash Keys

In choosing a hash key, one important consideration is your query design. There must be an EQUAL factor for each key column in the predicates of queries that will use the hash structure.

Another important issue is the distribution of key values. The best key results in a set of hash values that are evenly distributed among the primary pages available. The worst key results in hash values that cluster tightly in a narrow range of primary pages, leaving others empty.

Another consideration is that the key must be unique. It can either be a unique single column value or a unique combination. In addition to being unique, a hash key should be non-volatile, that is, not subject to frequent update. Since you cannot use the UPDATE statement with a hash key column, you must do a DELETE followed by an INSERT when a key modification is necessary.

An integer key such as a social security number is ideal. You can also use multiple columns to create a unique key, as in the following example:

```
CREATE PUBLIC TABLE PurchDB.OrderItems
(OrderNumber    INTEGER    NOT NULL,
 ItemNumber     INTEGER    NOT NULL,
 VendPartNumber CHAR(16),
 PurchasePrice  DECIMAL(10,2) NOT NULL,
 OrderQty       SMALLINT,
 ItemDueDate    CHAR(8),
 ReceivedQty    SMALLINT )
UNIQUE HASH ON (OrderNumber, VendPartNumber) PAGES=101
IN OrderFS
```

Note that use of a multicolumn key does not, in itself, ensure uniqueness.

Choosing the Number of Primary Pages

The number of primary pages in a hash structure is the number of data pages allocated as hash buckets. The optimal number depends partially on how much data you need to store, and partially on how sparse you wish the DBEFfile pages to be. The larger the number of primary pages, the more sparse the pages.

In general, choose a number that is large enough for the data you need to store plus space for a reasonable amount of growth. In practical terms, the smallest number of primary pages would be equal to the size of the data divided by the size of a page (about 3900 bytes, allowing for overhead). The largest number of primary pages you might choose would be equal to the total number of rows; for a unique hash structure with a completely even distribution of key values, this would mean one row per page.

For small tables subject to frequent access, create a hash structure with the number of primary pages equal to the number of rows and with an even distribution of key values. An example is a currency exchange table containing entries for 50 currencies using a currency code key with values ranging from 1001 to 1050 and 50 primary pages. This structure will enable you to lock only a single row at a time while accessing data, thereby improving concurrency.

Designing Integrity Constraints

You can enforce integrity in particular tables or in the relationships among tables in your design by creating integrity constraints on specific columns. Unique, referential, and check integrity constraints can be included as part of the `CREATE TABLE` or `ALTER TABLE` statements. A check constraint can be included as part of a `CREATE VIEW` statement.

A unique constraint lets you eliminate duplicate key values. To create the unique constraint, you use the `PRIMARY KEY` or `UNIQUE` option along with the `NOT NULL` clause in the `CREATE TABLE` or `ALTER TABLE` statement. To enforce uniqueness, `ALLBASE/SQL` automatically creates a unique index on the specified key in the same `DBFileSet` as the table. A unique constraint differs from a unique index in that you do not create it with a separate `SQL` statement, and you do not have to name it.

A referential constraint lets you enforce a relationship of dependency between different keys. This relationship implies the existence of both a referenced table and a referencing table, which may be the same table. The referential relationship means that a key value must exist in the referenced table before a row containing that key value is inserted in the referencing table. (However, the foreign key in the referencing table can contain `NULL` values, even though the primary or unique key in the referenced table cannot contain `NULL` values.)

The referenced table must be created with a `PRIMARY KEY` or `UNIQUE` clause in the `CREATE TABLE` or `ALTER TABLE` statement. The primary or unique key is then referred to in the `FOREIGN KEY` or `REFERENCES` clause in the `CREATE TABLE` or `ALTER TABLE` statement for the referencing table.

Table check constraints additionally ensure that a specified search condition does not evaluate to false for any row of a table. The search condition may evaluate to unknown if a column specified in the condition contains a `NULL` value. A search condition is defined on columns of a table in the `CREATE TABLE` or `ALTER TABLE` statement.

A check constraint for a view is defined through the `WITH CHECK OPTION` clause. This constraint ensures that no changes made through the view violate its definition. Such a check constraint is enforced during an insert or update to a table through the view. The check must satisfy first the view constraints and the constraints defined on the table on which the view is based.

You can also defer constraint checking to the transaction level and set general error checking to row level. Constraint error checking is performed at the current general error checking level unless constraint checking is deferred. Deferring constraint error checking avoids constraint errors that will be resolved by the end of the transaction. Setting general error checking to row level avoids some logging and rollback overhead if you are using nonarchive logging.

For complete information about integrity constraints, refer to “Constraints, Procedures, and Rules” chapter in the *ALLBASE/SQL Reference Manual*, and to the application programming guide for the language of your choice.

Defining Security Levels

In ALLBASE/SQL, you create security controls by granting authorities to specific users or groups. An **authority** is a privilege that enables a user to access the DBEnvironment, create database objects, use SQL statements, preprocess and run application programs containing SQL statements, or maintain the DBEnvironment. By selectively granting and revoking authorities, the DBA can control access to the DBEnvironment as well as to the tables, views, columns, groups, and modules within the DBEnvironment.

You can grant authorities to one of four entities:

- A DBEUserID
- An authorization group
- A class
- PUBLIC

A **DBEUserID** is made up of a user's MPE/iX user and account names connected with the @ symbol. An example is Wolfgang@DBMS, where Wolfgang is the user name, and DBMS is the account name. An **authorization group** is a named collection of users sharing the same authorities. A **class** is an entity which may own objects that should not belong to any individual or group.

PUBLIC is a special, nonrestrictive category of user. By granting an authority to PUBLIC, you implicitly grant that authority to any user who has CONNECT authority to the DBEnvironment.

Authority Types

There are five basic types of authority in a security design:

- Special authorities
- OWNER authority
- Table and view authorities
- RUN authority
- EXECUTE authority
- Space authorities

Except for DBA, RESOURCE, and OWNER authorities, ALLBASE/SQL authorities are discrete, not hierarchical; for example, granting CONNECT authority does not automatically grant RUN authority. This relationship is shown in this figure.

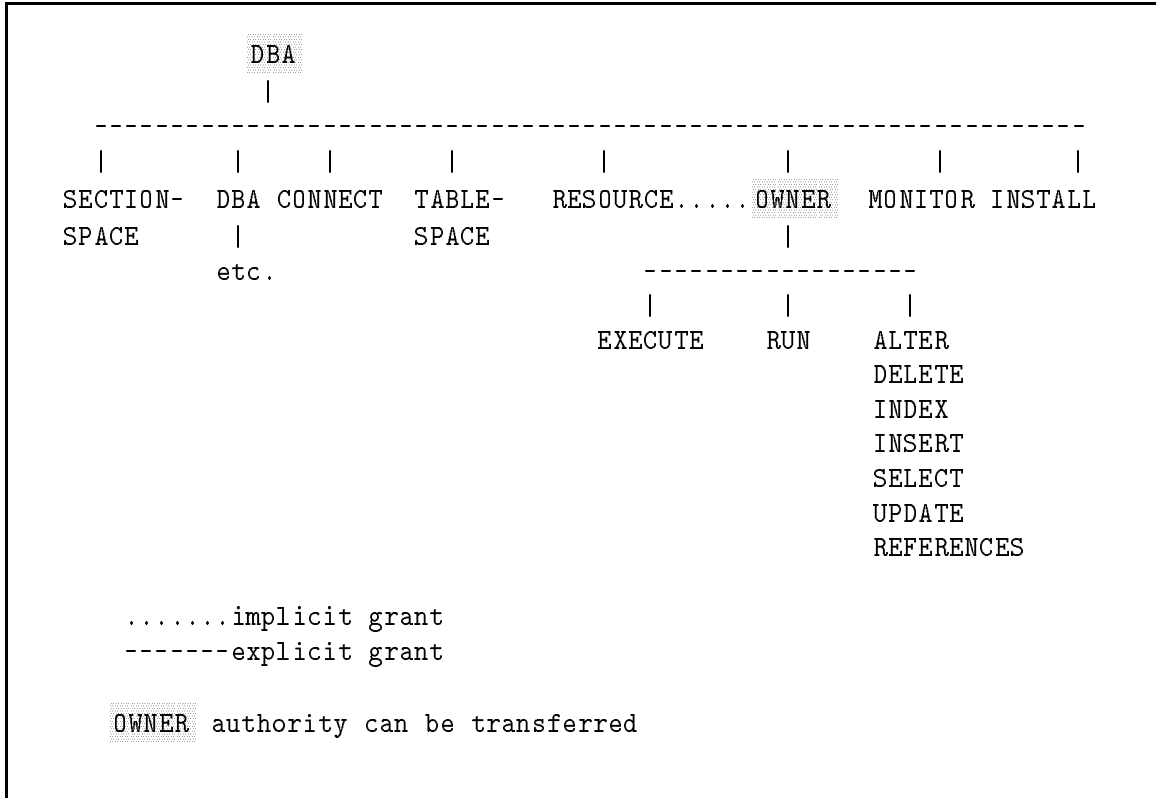


Figure 2-8. Relationship among Authorities

Special Authorities

The following authorities are special authorities:

- DBA authority
- RESOURCE authority
- CONNECT authority
- MONITOR authority
- INSTALL authority

A user with DBA authority has all other authorities plus some special capabilities for DBEnvironment maintenance. A user with DBA authority shares OWNER authority for all objects in the DBEnvironment. DBA authority is required to grant or revoke any of the special authorities.

A user with RESOURCE authority can create tables and authorization groups. The user becomes the owner of the table or group. That user has OWNER authority for the object until ownership is transferred to another user.

A user with CONNECT authority can connect to the DBEnvironment.

A user with MONITOR authority can run SQLMON on the DBEnvironment.

A user with INSTALL authority can install stored sections owned by another user, class, or group.

Owner Authority

The following objects can have owners:

- Tables
- Views
- Modules
- Procedures
- Authorization groups

The owner of an object can drop that object. The owner of a table, view, module, or procedure can grant authorities for that table, view, module, or procedure to other users. The owner of a module has RUN authority for that module plus the capability to re-preprocess the module. The owner of an authorization group can add members to the group, remove members from the group, or drop the group.

The name of an authorization group must be unique within the DBEnvironment. There cannot be either another owner or another authorization group with the same name on the system.

OWNER authority for a table or view implies all table and view authorities. Even a user with DBA authority can neither revoke table and view authorities from the owner of a table or view, nor revoke RUN authority from the owner of a module. OWNER authority cannot be granted or revoked, but it can be acquired during object creation or transferred to another owner by either the current owner or by a user with DBA authority. If ownership is transferred, the original user no longer has any authorities to access the object unless explicitly granted.

The owner of an object can be an individual user, an authorization group, or a class. OWNER authority is obtained in one of the following ways:

- By creating an object. RESOURCE authority is required for users to be able to create *ownable* objects. Only a user with DBA authority can create an object and assign another individual user, a group, or a class name as the object's owner.
- By having DBA authority. All users with DBA authority *share* ownership of all objects in the DBEnvironment with the actual owners. Users with DBA authority are also exclusive owners of objects owned by authorization groups and classes.
- By transferring ownership. Ownership can be transferred from any name to any other name by either the current owner of the object or by a user with DBA authority.

Table and View Authorities

Table and view authorities, listed below, determine which users or groups may have access to the columns of tables for specific tasks, such as selecting, updating, deleting, inserting, or indexing:

- ALTER
- DELETE
- INDEX
- INSERT

- SELECT
- UPDATE
- REFERENCES

Table and view authorities are described fully in Chapter 5.

RUN Authority

RUN authority determines who has access to specific preprocessed application modules stored in the DBEnvironment. RUN authority is further described in the “Maintenance” chapter.

EXECUTE Authority

EXECUTE authority determines who has access to specific procedures stored in the DBEnvironment. EXECUTE authority is further described in the “Maintenance” chapter.

Space Authorities

The following authorities are space authorities:

- SECTIONSPACE
- TABLESPACE

A user with SECTIONSPACE authority can store sections in the specified DBEFileSet. If the user does not have SECTIONSPACE authority, the default SECTIONSPACE DBEFileSet is used instead, even if the user has DBA authority.

A user with TABLESPACE authority can store table and long column data in the specified DBEFileSet. If the user does not have TABLESPACE authority, the default TABLESPACE DBEFileSet is used instead, even if the user has DBA authority.

For more information, refer to “Parameters—Grant DBEFileSet Authority” in the Grant section of the “SQL Statements” chapter of the *ALLBASE/SQL Reference Manual*.

Defining Authorization Groups

An authorization group can be created by any user with RESOURCE or DBA authority, and ownership can be transferred to any user. Authorization groups are created and dropped as objects, but have the attributes of a user because they can:

- Be granted any or all authorities.
- Have any or all of their authorities revoked.
- Own tables, views, modules, and other authorization groups.

When an authorization group is created, its group name is entered into the SYSTEM.GROUP table in the system catalog (for more information, refer to the “System Catalog” chapter of this manual). You cannot create a group using a name that already exists as a DBEUserID or owner name in the DBEnvironment. Conversely, you cannot grant authorities to a valid DBEUserID if the name already exists as an authorization group or as an owner name in the DBEnvironment.

The following DBEUserIDs are reserved and cannot be added to a group or used as a group name:

- HPRDBSS

- PUBLIC
- SEMIPERM
- STOREDSECT
- SYSTEM
- TEMP

In addition, the DBECreator cannot be added to an authorization group.

Each member of the authorization group has the authorities granted to the group. The owner of the group, if not explicitly a member of the group, does not have any of the authorities that have been granted to the group.

After defining groups, you can then control authorities on a group basis instead of an individual user basis.

Determining Group Membership

Group membership is determined by common authority requirements. The DBEUserIDs used in the sample database belong to different departments within the Integrated Peripherals, Inc. organization. Each department or function requires a different set of authorities. Authorization groups are created for each function and DBEUserIDs are added accordingly.

A user can be a member of any number of authorization groups. An authorization group can itself be a member of other authorization groups. A member of an authorization group that is, in turn, a member of a second authorization group is known as an **indirect** member of the second authorization group. Indirect members of groups have the authorities granted to the group of which they are a direct member *and* the group of which they are an indirect member. In Figure 2-9, the valid chain shows Kelly as a *direct* member of GroupA and an *indirect* member of GroupD and GroupE. Kelly has the authorities granted to GroupA, GroupD, and GroupE.

Although there is no limit on the number of authorization groups that an authorization group can belong to indirectly, as illustrated in Figure 2-9, the chain cannot link back to itself.

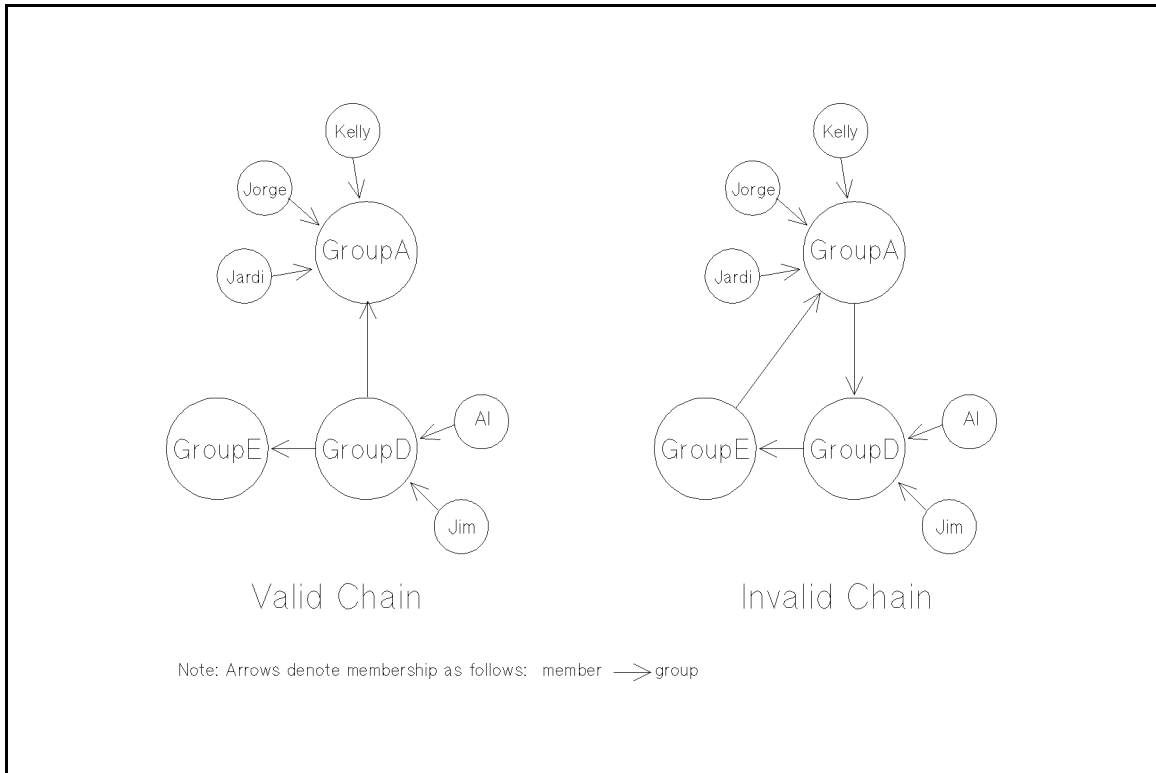


Figure 2-9. Authorization Group Chain

Defining Classes

A **class** is a special category of owner that is neither a conventional DBEUserID nor a group. You may wish to assign ownership of objects to a class when you do not want any individual or group to have automatic access to them. With class ownership, the DBA controls all authorities, since objects that belong to a class can be created and maintained only by the DBA.

Strictly speaking, a class is an owner that does not have CONNECT authority. An example of this is an owner for which there is not a corresponding userid on the system and to which you have not granted CONNECT authority. For a class to be useful, its class name must be different from the name of any existing DBEUserID or group ID.

Differences Between Groups and Classes

You create a group **explicitly** by using the CREATE GROUP statement. You create a class **implicitly** by creating objects that have a class name as owner name.

A group has members, all of which have the same group privileges. For example, if user *alex* is a member of a group *Sales* then *alex* can drop or alter objects owned by *Sales*.

A class does not have members, nor can it use any authorities, though you can grant them if you wish. This can be useful in a scenario in which you want to pre-assign ownership of objects to a DBEUserID for which there is not yet a userid on your system.

As DBA, you retain all authorities over the objects owned by a class and must explicitly grant authority to any user of those objects.

Guidelines for Creating Classes

In designing a security scheme that includes classes, follow these guidelines:

- Make sure that a new class name is different from any DBEUserID.
- Make sure the class name is not the same as any group name defined in the DBEnvironment.
- Choose a class name that reflects the concept behind the class, e.g., *PurchDB* for Purchasing Department Database.
- Do not assign CONNECT authority to a class. Doing so converts the class into a DBEUserID with OWNER authority over objects in the class. This could create problems at a later time if a real userid were created that had the same name as the class.

As an exception to the last rule, you **can** safely grant CONNECT authority when you actually want the new DBEUserID to become the owner of the object once the correct userid is added to the system.

Defining the DBEnvironment Scope

ALLBASE/SQL databases are contained in a DBEnvironment. A DBEnvironment is one or more databases that share the same system catalog and the same logging and recovery. Databases within a DBEnvironment can be related to each other through ISQL and in applications if security allows it. Databases that are contained in separate DBEnvironments cannot be linked to each other. An interactive user or application program can only access one DBEnvironment at a time. (However, the same application can maintain several DBEnvironment connections and access different DBEnvironments in sequence. Refer to the section “Using Multiple Connections and Transactions with Timeouts” in the *ALLBASE/SQL Reference Manual* chapter “Using ALLBASE/SQL.”)

As shown in Figure 2-10, the DBEnvironment used by Integrated Peripherals, Inc., has three databases.

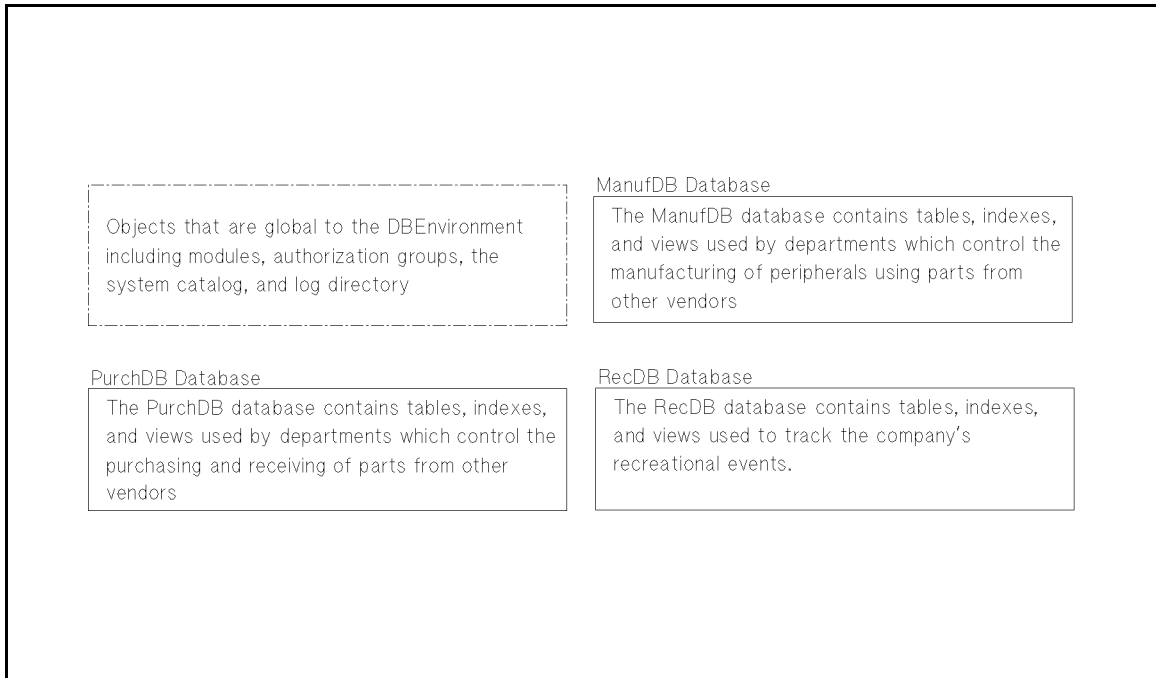


Figure 2-10. DBEnvironment used by Integrated Peripherals, Inc.

The PurchDB database has been designed in the present chapter; the other databases are included to illustrate the use of multiple databases in a single DBEnvironment. All three databases are placed in a single DBEnvironment because the data, although used separately in most applications, is sometimes used together in reports and global applications. The number of databases that you can put in a DBEnvironment is limited only by the amount of disk space available to you.

There are some reasons why you might create separate DBEnvironments for databases.

- You may have several users who need to create and drop objects frequently. Consider putting their databases into different DBEnvironments, since the frequent creation and dropping of objects by several users in the same DBEnvironment requires heavy use of the system catalog and can cause deadlocks.
- You may have databases with different users functioning as DBA. DBA authority is granted at the DBEnvironment level. Databases with different DBAs should be in different DBEnvironments.
- Recovery time increases as the size of the DBEnvironment increases.

Databases are logically separated by the owner name of the tables, indexes, and views. They can also be *physically* separated for independent manipulation through selective allocation of space using DBEFiles and DBEFileSets. The allocation of space or **physical design** of the database is covered in the next chapter.

The DBEnvironment designed for Integrated Peripherals, Inc. is the basis for the sample DBEnvironment PartsDBE, which is included with the ALLBASE/SQL product. Most of the examples in the rest of this book refer to tables or other objects in PartsDBE.

Physical Design

Physically, an ALLBASE/SQL DBEnvironment is a collection of files for the storage of databases. The physical design process helps you compute the storage requirements for the tables, indexes, and other objects that emerge at the end of the logical design process. This chapter describes several steps that are involved in developing an appropriate physical design for your DBEnvironments:

- Calculating storage for database objects
- Calculating storage for the system catalog
- Calculating storage for logging
- Calculating temporary disk space
- Estimating shared memory requirements
- Estimating the number of transactions
- Implementing the design

Before designing physical storage, you must first establish the logical design of your databases. The layout of the tables, the number of columns and column sizes, the column data types, the number and types of indexes, and the estimated number of rows per table are all factors that affect physical design. You should also have an idea of how the tables are going to be used, since you can design physical storage to optimize performance.

Physical design also requires an understanding of ALLBASE/SQL files. Therefore, this chapter also describes the characteristics of DBEFiles and log files while presenting guidelines on determining storage requirements. Familiarity with the MPE/iX file system will also help you during the physical design phase.

Calculating Storage for Database Objects

A good database storage design requires the following:

- Understanding DBEFile characteristics
- Calculating storage for tables
- Calculating storage for indexes
- Arranging tables and indexes in DBEFileSets
- Calculating storage for hash structures
- Calculating storage for integrity constraints

The numbers derived from the calculations described in the following sections are used to assign a value for a number of pages in the `PAGES=` clause of the `CREATE TABLE` and `CREATE DBEFILE` statements.

Understanding DBEFile Characteristics

DBEFiles are used to store table and index data. They are composed of 4096-byte **pages**. DBEFiles can be from 2 to 524,287 pages. The number of pages in an ALLBASE/SQL file is determined when it is created. In the case of expandable DBEFiles, the maximum size and the size of an increment are determined when the DBEFile is created.

All DBEFiles must be associated with a DBEFileSet before they can be used to store data. A DBEFileSet is a logical grouping of one or more DBEFiles. Figure 3-1 shows a DBEFileSet (represented by the dotted lines because it is a logical construct) which contains three DBEFiles (represented by solid lines because they are physical constructs).



Figure 3-1. DBEFiles in DBEFileSets

The amount of storage available in a DBEFileSet is the sum of the pages of all the DBEFiles in that DBEFileSet. The DBEFileSet in Figure 3-1 shows a total of 200 pages. When a table in the DBEFileSet illustrated needs more than 200 pages to store data, additional DBEFiles can be added to the DBEFileSet to accommodate more data.

You can specify the type of data that a DBEFile can contain. DBEFiles can be of type MIXED, TABLE, or INDEX. MIXED DBEFiles can store either table or index data. TABLE DBEFiles can store only table data. INDEX DBEFiles can store only index data. As shown in Figure 3-2, DBEFile1 can be of type TABLE, while DBEFile2 and DBEFile3 must be of type MIXED.

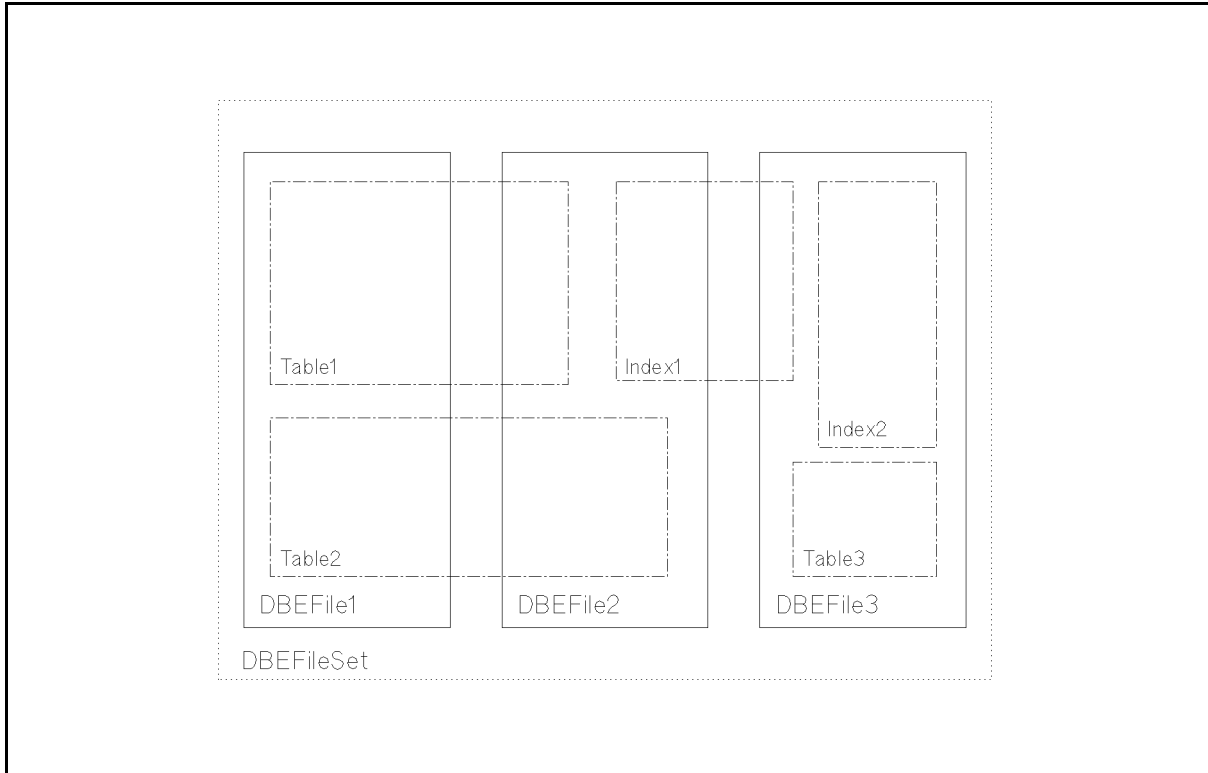


Figure 3-2. Data Stored in DBEFiles within a DBEFileSet

All the DBEFiles for a given table and its indexes must be contained in one DBEFileSet. Table and index data can span more than one DBEFile within a DBEFileSet. Figure 3-2 shows the relationship between DBEFileSets, DBEFiles, and data. Note that in the figure the tables and indexes are stored in one or more DBEFiles, but they are all contained within a single DBEFileSet.

Calculating Storage for Tables

You will need the following information to calculate the number of DBEFile pages needed by each table and index:

- The column size and data type for each column in the table
- The row length of the table
- The approximate number of rows that the table will contain initially.

Use the following procedure to calculate the necessary number of DBEFile pages needed for your tables:

1. Calculate the row length for the table (RL).
2. Calculate the number of rows that can fit on a page (NRP).
3. Calculate the number of pages needed to hold all rows (NDP).
4. Calculate the number of directory overhead pages (DO) needed by ALLBASE/SQL.
5. Add the results of steps three and four to arrive at the total number of DBEFile pages needed for that table.

The examples presented below show worst-case calculations. They assume that each row has its own tuple header. Refer to the section “Defining Null Values for Columns” in the “Logical Design” chapter for more information about tuple headers.

Calculating Row Length

The row length is dependent on the data type and size of the columns in the table.

Column size is calculated in bytes. INTEGER, SMALLINT, REAL, and FLOAT columns are a fixed number of bytes. The size of BINARY, VARBINARY, DECIMAL, CHAR, and VARCHAR columns depends on the size given in the column definition. Refer to Table 3-1 for the storage requirements of the various data types.

Table 3-1. Data Type Storage Requirements

Type	Storage Required
CHAR (n)	n bytes (where n must be an integer from 1 to 3996)
VARCHAR (n)	n bytes (where n must be an integer from 1 to 3996)
DECIMAL (p[,s])	4 bytes (where $p \leq 7$) or 8 bytes (where $7 < p \leq 15$) or 12 bytes (where $15 < p \leq 23$) or 16 bytes (where $p > 23$)
FLOAT	8 bytes
REAL	4 bytes
INTEGER	4 bytes. Integer values less than -2147483648 (-2^{31}) or larger than 2147483647 ($2^{31} - 1$) up to 15 digits long are stored as decimals with a precision of 15 and a scale of 0, i.e., equivalent to DECIMAL (15,0)
SMALLINT	2 bytes
DATE	16 bytes
TIME	16 bytes
DATETIME	16 bytes
INTERVAL	16 bytes
BINARY (n)	n bytes (where n must be an integer from 1 to 3996)
VARBINARY (n)	n bytes (where n must be an integer from 1 to 3996)
LONG BINARY (n)	n bytes (where n must be an integer from 1 to $2^{31} - 1$)
LONG VARBINARY (n)	n bytes (where n must be an integer from 1 to $2^{31} - 1$)

As you begin calculating row lengths, try to use physical space as efficiently as possible. Tables can share DBEFiles, but only one table can store rows in a given DBEFile page. A row whose columns add up to a length greater than 2000 bytes will potentially waste page space. For example, if you have a table with a row length of 2050 bytes, a DBEFile page of 4096 bytes with a 100 byte overhead would only be able to hold one 2050 byte row. The remaining 1950 bytes would be left empty.

3-4 Physical Design

The Parts table used in the examples in this chapter has three columns with the following characteristics:

PartNumber	CHAR(16),
PartName	CHAR(30),
SalesPrice	DECIMAL (10,2)

Use the following calculation to determine the row length for the table:

$$RL = 16 + 30 + 8 = 54 \text{ Bytes}$$

The calculations in the following examples assume 30,000 rows will be stored in the Parts table.

Calculating Rows per Page

The following formula is used to calculate the number of rows per page (NRP):

$$NRP = \frac{4030}{RL + (2NC) + 4}$$

where RL is row length, and NC is the number of columns in the table, divided into the number of bytes that can fit on a page (a set value of 4030). This formula includes overhead for column values that can be NULL as well as overhead for the size of VARCHAR and VARBINARY data.

Using the values for the Parts table, the number of rows that can fit on a page is calculated as follows:

$$NRP = \frac{4030}{54 + 2 * 3 + 4} = \frac{4030}{64} = 62.97$$

The value for NRP is rounded down to the next integer because a partial row cannot be stored on a page. The result is 62 rows per page.

Calculating Number of Pages

The following formula is used to calculate the number of DBEFile pages (NDP) needed to hold all rows in the table:

$$NDP = \frac{NR}{NRP}$$

where NR is the number of rows in the table, and NRP is the number of rows per page.

Using the previous value for NRP of 62 and an assumed value of 30,000 rows in the Parts table, the number of pages needed for the table is calculated as follows:

$$NDP = \frac{30000}{62} = 483.8$$

The value for NDP is rounded up to the next integer because DBEFiles are not created with partial pages. The result is 484 data pages.

Calculating Directory Overhead

For every 252 pages in a DBEFile, ALLBASE/SQL creates a **page table page** as a directory to store information about the next 252 pages. It contains pointers to data and keeps track of which pages are empty and which tables contain rows in which pages. The directory overhead (DO) can be calculated with this formula:

$$DO = \frac{NDP}{252}$$

where NDP is the number of data pages which is divided by a set value of 252.

Using the previous value for NDP of 484 pages, the directory overhead is calculated as follows:

$$DO = \frac{484}{252} = 1.9$$

The value for directory overhead is rounded up to the next integer because DBEFiles cannot have partial page table pages. The result is 2 page table pages.

The total number of DBEFile pages needed to store 30,000 rows in the Parts table is calculated as follows:

$$Pages = 484 \text{ Data Pages} + 2 \text{ Pages Overhead} = 486 \text{ Total Pages}$$

Perform the calculation for each table to get the total estimated number of DBEFile pages needed for the DBEnvironment. Be sure to leave enough space in the DBEFiles for minor expansion of the tables, so you do not need to add DBEFiles to DBEFileSets frequently.

Calculating Storage for Indexes

For indexes, ALLBASE/SQL uses a doubly linked balanced tree (B-tree) structure, which can have several levels between its initial node, or root page, and the leaf node, or leaf page, containing the pointer to the requested row. (For basic information about B-tree indexes, refer to “Designing Indexes” in the “Logical Design” chapter.) Calculate the following values to determine the number of DBEFile pages needed for an index:

- Index key length
- Size of index header
- Number of bytes per page
- Number of rows per non-leaf page
- Number of leaf pages
- Number of non-leaf pages
- Number of overhead pages

The total number of index pages needed is the sum of the number of leaf pages, non-leaf pages, and overhead pages.

Calculating the Index Key Length

The index key length (KL) is calculated by adding 10 to the sum of length of the columns (SLC) on which the index is created:

$$KL = SLC + 10$$

The 10 bytes includes 8 bytes for the data TID entry and 2 bytes needed for the slot table entry. Consult the preceding table, “Data Type Storage Requirements,” for the column length of each data type. If the index is defined upon a CHAR(16) column, an INTEGER column, and a CHAR(20) column, the index key length (KL) is calculated as follows:

$$KL = 16 + 4 + 20 + 10 = 50$$

Calculating the Size of the Index Header

The size of the index header (IH) can be calculated with the following formula:

$$IH = 2 * (NIC + 1) + 2$$

NIC is the number of columns upon which the index is defined. The constant 1 represents the byte needed for the TID. An additional 2 bytes is needed for overhead. If the index is defined upon 3 columns then the size of the index header is determined as follows:

$$IH = 2 * (3 + 1) + 2 = 10$$

Calculating the Number of Rows per Leaf Page

The formula used to calculate the number of rows per leaf page (RLP) depends on whether the index header can be shared. If the columns on which the index has been defined allow NULL values, or if their data type is VARCHAR, then the index header cannot be shared.

Use the following formula to calculate the number of rows per leaf page (RLP) when the index header can be shared :

$$RLP = \frac{4006}{KL + 2 + 8}$$

2 is 2 bytes for the slot table entry; 8 is 8 bytes for the data TID entry. If the index header cannot be shared, use the following formula:

$$RLP = \frac{4006}{KL + 2 + 8 + IH} * \frac{2}{3}$$

4006 is the number of bytes available in a page. For a conservative estimate, assume that the leaf pages are 2/3 full. If the result is a fraction, round down to the nearest integer because a partial row cannot be stored in a page. For example, if the index key length is 50 and the index header cannot be shared, the number of rows per leaf page is calculated as follows:

$$RLP = \frac{4006}{50 + 10} * \frac{2}{3} = 44$$

Calculating the Number of Rows per Non-Leaf Page

If the index header can be shared, calculate the number of rows per non-leaf page (RNLP) with the following formula:

$$RNLP = \frac{4006}{KL + 2 + 8 + 8} * \frac{1}{2}$$

Where 2 is 2 bytes for the slot table entry; 8 is 8 bytes for the data TID entry; 8 is the next data TID pointer. Use the following formula if the index header cannot be shared:

$$RNLP = \frac{4006}{KL + 2 + 8 + 8 + IH} * \frac{1}{2}$$

The value is multiplied by 1/2, because we assume that the non-leaf pages are half full. If the result is a fraction, round down to the nearest integer because a partial row cannot be stored in a page. For example, if the index key length is 50 and the index header cannot be shared, then the calculation is as follows:

$$RNLP = \frac{4006}{50 + 2 + 8 + 8 + 10} * \frac{1}{2} = 26$$

Calculating the Number of Leaf Pages

Each row of table data is pointed to by a row in a leaf page. To calculate the number of leaf pages (LP), divide the number of rows your table will contain (RT) by the number of rows per leaf page (RLP). If the result is a fraction, round down to the nearest integer because partial pages do not exist. The formula is as follows:

$$LP = \frac{RT}{RLP}$$

If the table will contain 651090 rows of data, and the value of rows per leaf page is 44, the number of leaf pages is calculated as follows:

$$LP = \frac{651090}{44} = 14797$$

Calculating the Number of Non-Leaf Pages

To determine the total number of non-leaf pages, you must calculate the number of non-leaf pages at each level in the B-tree. Start at the lowest non-leaf level, that is, the level just above the leaf pages, and move up the B-tree until the level has only one page. At each level, use the following formula:

$$NLP(n) = \frac{RL(n)}{RNLP}$$

$NLP(n)$ is the number of non-leaf pages a level n , $RL(n)$ is the number of rows needed at level n , and $RNLP$ is the number of rows per non-leaf page. If the result is a fraction, round down to the nearest integer because partial pages do not exist. Since the rows of non-leaf pages point to the leaves at the next level down, the value of RL decreases with each higher level.

In the example that follows, assume that the rows per non-leaf page is 33 and the number of leaf pages is 14797. First, calculate the number of pages at the lowest non-leaf level. The rows at this level point to the leaf pages. Since there are 14797 leaf pages, the $RL(0)$ is 14797. The calculation is as follows:

$$NLP(0) = \frac{14797}{33} = 448$$

Next, calculate the number of non-leaf pages at the next level up. The rows at this level point to 448 non-leaf pages. The calculation is as follows:

$$NLP(1) = \frac{448}{33} = 13$$

At the next level up, the page rows point to 13 non-leaf pages. After rounding up to the nearest integer, the result of the following calculation is 1, indicating that the highest level has been reached:

$$NLP(2) = \frac{13}{33} = 1$$

The total number of non-leaf pages is the sum of the non-leaf pages at each level:

$$NLP = 448 + 13 + 1 = 462$$

Calculating the Number of Directory Overhead Pages

For every 252 pages in the B-tree, a directory overhead page (DO) is required:

$$DO = \frac{LP + NLP}{252}$$

A directory overhead page is also referred to as a page table page. Using the values from the previous examples, the number of overhead pages is calculated as follows:

$$DO = \frac{14797 + 462}{252} = 61$$

Calculating Total Number of Index Pages

The total number of index pages (TIP) is the sum of the leaf pages (LP), non-leaf pages (NLP), and directory overhead pages (DO):

$$TIP = LP + NLP + DO$$

Using the values from the previous examples, the total number of index pages is calculated as follows:

$$TIP = 14797 + 462 + 61 = 15320$$

Arranging Tables and Indexes in DBEFileSets

Remember the following when allocating DBEFile storage:

- All data for a given table and its indexes (including integrity constraints) must be contained in a single DBEFileSet.
- If you do not specify a DBEFileSet when creating a table, the table is put in the SYSTEM DBEFileSet.
- Storage is allocated for tables, indexes, and constraints by adding DBEFiles with the adequate number of pages to the DBEFileSet in which you intend to create the tables and indexes.
- When you create an index on a table that is located in the SYSTEM DBEFileSet, index pages also are stored in DBEFiles associated with SYSTEM.
- Any DBEFile in a DBEFileSet can potentially store rows from any table and index associated with that DBEFileSet.

Grouping Tables in DBEFileSets

The following factors affect how tables should be grouped in DBEFileSets:

- ease of maintenance
- performance tuning capability
- independence of hash data

Grouping tables in separate DBEFileSets increases traceability of storage for particular tables. For example, if you store all tables in the SYSTEM DBEFileSet, you will not be able to tell which DBEFiles hold system catalog data and which hold data for a particular table.

As a result, all tables and the system catalog would have to be taken into consideration when storage space is added to the DBEnvironment. All maintenance functions such as the UPDATE STATISTICS statement would take longer if all tables are stored in a single DBEFileSet. Therefore, you should create a DBEFileSet for each table or group of tables that you want to maintain separately.

You may want to place tables that are used infrequently in the same DBEFileSet to use space more efficiently. You can add DBEFiles to the DBEFileSet to accommodate the space requirements of several tables.

Hash tables must be created in separate DBEFiles. It may be useful to create separate DBEFileSets for your hash tables.

Large tables should have their own DBEFileSet. If a small table is in the same DBEFileSet as a large one, the performance of sequential scans on the small table will not be as good as if the small and large tables are separated.

You may want to place related tables in the same DBEFileSet. In the sample database, one DBEFileSet is created for the internal parts information in the Parts and Inventory tables. Another DBEFileSet is for the vendor information in the Vendors and SupplyPrice tables. A third DBEFileSet contains the order data of the Orders and OrderItems tables. Since related tables are often updated simultaneously, DBEFiles can be added to the DBEFileSet to accommodate the growth of multiple tables. Figure 3-3 shows how the sample DBEnvironment is divided into three DBEFileSets.

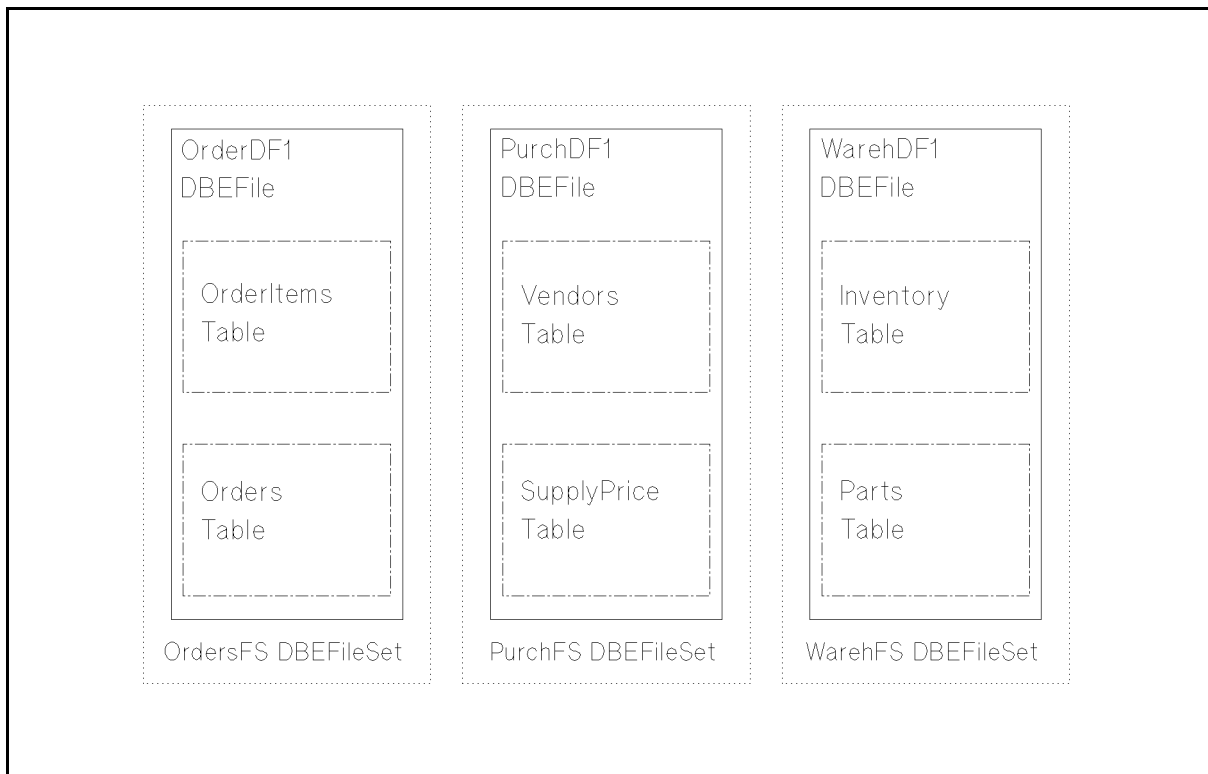


Figure 3-3. DBEFileSets in the Sample DBEnvironment

Choosing DBEFile Types and Devices

When a DBEFile is created, it can be specified as one of three types:

- TABLE, which contains only table data
- INDEX, which contains only index data
- MIXED, which can contain both table and index data

You can control performance by selecting DBEFile types carefully and by locating DBEFiles on the appropriate devices. The characteristics of the transactions to be processed determine if you should create a separate INDEX DBEFile. If your applications access indexes frequently, placing the indexes in a separate DBEFile (and possibly on a different, faster device) may improve performance. If your applications access the indexes infrequently, having index and table data share the same DBEFile uses disk space more efficiently.

Using a Single MIXED DBEFile

The default file type is MIXED. If table and index data are stored together in a MIXED DBEFile, the disk drive does not have to search multiple DBEFiles when ALLBASE/SQL uses an index. However, this may not be true for large tables that span several DBEFiles, or for tables with multiple indexes.

Using Separate DBEFiles for Tables and Indexes

You can improve performance by placing table and index data in different DBEFiles and locating the DBEFiles on different devices. Then, when an index is used during query processing, each disk drive accesses either index or table data and reads and updates are distributed over multiple devices rather than concentrated on a single disk drive.

To ensure that table and index data will be stored in different DBEFiles, create separate DBEFiles of type TABLE and INDEX, and do *not* create any MIXED DBEFiles. If there are no MIXED DBEFiles, ALLBASE/SQL must place all index data in the INDEX DBEFile and table data in the TABLE DBEFile. If space of the appropriate type is not available, an error is generated.

If you use separate TABLE and INDEX DBEFiles in a DBEFileSet, you should use the results of the calculations presented above in “Disk Space for Tables” to provide enough space in the TABLE DBEFiles to contain all the tables in the DBEFileSet. Similarly, use the results of the calculations presented above in “Disk Space for Indexes” to provide enough space in the INDEX DBEFiles for all the indexes defined on all tables in the DBEFileSet.

Figure 3-4 shows how the Orders and OrderItems tables in the sample database are stored in one TABLE DBEFile, and the indexes stored in an INDEX DBEFile.



Figure 3-4. Table and Index DBEFiles in the OrdersFS DBEFileSet

Note that the OrderFS DBEFileSet does not contain any MIXED DBEFiles.

Using Different Storage Devices

You can use the MOVEFILE command in SQLUtil to locate INDEX DBEFiles on separate devices from the TABLE DBEFiles in the same DBEFileSet. Since you can only move physical files (i.e. DBEFiles), you must keep tables physically separate to be able to place them on different devices. Simply storing tables in different DBEFiles does not ensure that they are physically separate.

You may want to create separate DBEFileSets for two tables that are accessed frequently at the same time by users or applications. Then tables can be associated with different DBEFileSets and located on different disk drives to minimize disk drive workload.

In Figure 3-5, some of the DBEFiles are located on different disks even though they belong to the same DBEFileSet.

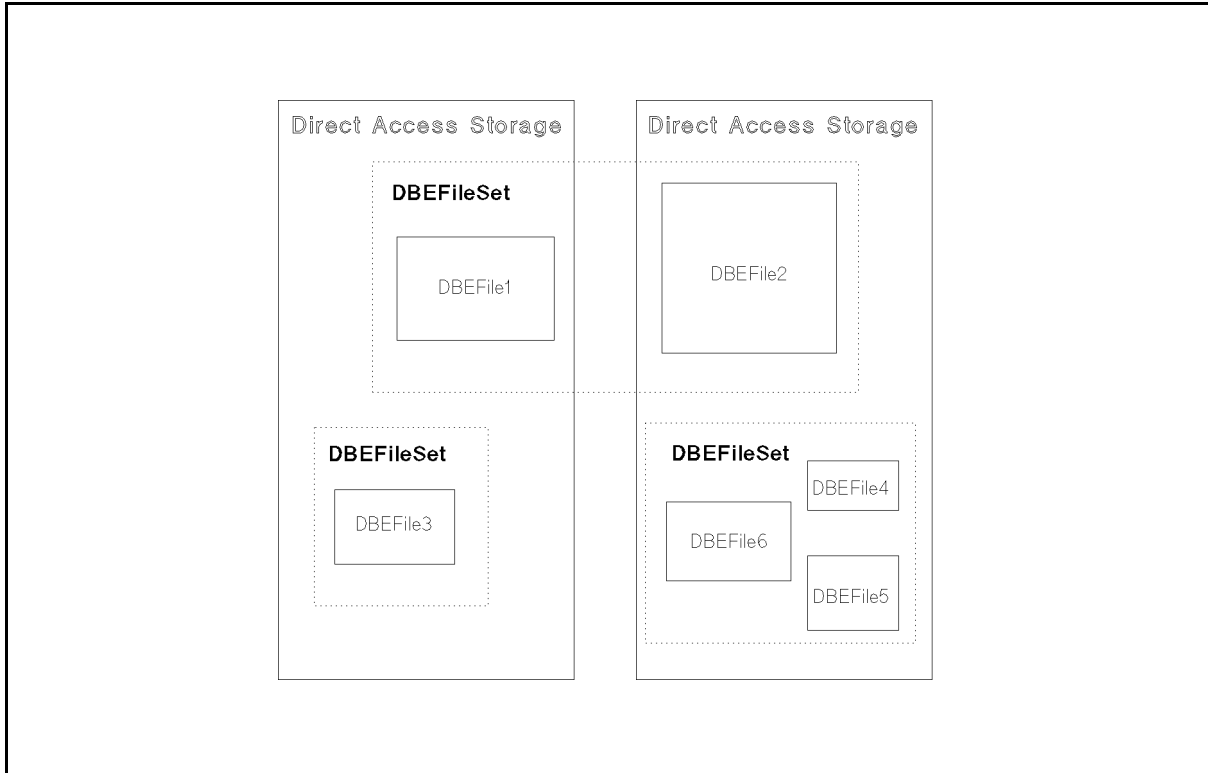


Figure 3-5. DBEFiles, DBEFileSets, and Direct-Access Storage

Estimating DBEFile Size

You specify a DBEFile size in the `PAGES` clause of the `CREATE DBEFILE` statement. Once you have determined the storage requirements for each table and index and you have determined which tables and indexes will be assigned to which DBEFileSets, you can estimate how big the DBEFiles should be.

Determining DBEFile size involves a trade-off between convenience (that is, how often you have to add additional DBEFiles) and use of direct access storage space. Remember the following when choosing a size for your DBEFiles:

- It is a more efficient use of disk space to add DBEFiles as they are needed rather than to allocate large DBEFiles before the space is required.
- DBEFiles within a DBEFileSet can be of different sizes.
- Tables and indexes can span more than one DBEFile.
- A DBEFile cannot be dropped if it is associated with a DBEFileSet.
- The size of a nonexpandable DBEFile cannot be changed without dropping and recreating it.
- Expandable DBEFiles expand as needed in increments you specify when you create them up to the maximum you indicate.

Initially, DBEFiles should be large enough to hold the estimated number of rows for all the tables and indexes in the DBEFileSet.

The DBEFile in the OrderFS DBEFileSet should be large enough to contain the data for both the Orders and the OrderItems tables. The formulas discussed earlier in this chapter were used to arrive at the following page requirements assuming 30,000 rows per table:

Table 3-2. Page Requirements for Table Data

Table Name	Size
Orders Table	195 pages
OrderItems Table	450 pages

A DBEFile of type TABLE with at least 645 pages should be created to contain the data for the two tables. You should create the DBEFiles slightly larger than the estimate to make room for minor growth. DBEFiles can be added to make room for significant growth.

The Orders table has two indexes and the OrderItems table has one index. Again, using the formulas on the previous pages, the following numbers are calculated:

Table 3-3. Page Requirements for Index Data

Index Name	Table Name	Key Column	Column Data Type	Size
OrderNumIndex	Orders	OrderNumber	Integer	430 pages
OrderVendIndex	Orders	VendorNumber	Integer	430 pages
OrderItemIndex	OrderItems	OrderNumber	Integer	430 pages

A DBEFile of type INDEX with at least 1290 pages will accommodate 30,000 keys for each of the three indexes. The DBEFiles should be created slightly larger to make room for minor growth. Again, additional DBEFiles of type INDEX can be added to accommodate significant growth.

Calculating Storage for Hash Structures

The amount of disk space used by a hash structure consists of the primary pages you define when you create the structure, page table pages used as overhead by ALLBASE/SQL, and any overflow pages used for rows which do not fit on the primary page pointed to by their hash key.

Calculating Primary Pages

You can use the following formula to estimate the approximate number of primary pages:

$$\text{Primary Pages} = \frac{\text{Number of Rows} * \text{Row Size}}{\text{Page Size} * \text{Fill Factor}}$$

Page Size is the amount of space available on a page for data. After subtracting overhead from a 4096-byte page, about 3900 bytes are free for tuples. *Fill Factor* is the percentage of each primary page that should be filled.

Suppose you will have a table with 2000 rows that are 350 bytes long and you wish to allow 30% space for additional growth:

$$\text{Primary Pages} = \frac{2000 * 350}{3900 * .70} = 256.410$$

Rounding up to the next whole page, the result is 257 primary pages. For non-integer keys, you should round up again to the next prime number of pages, which will yield the best results.

Allowing for Overflow

When ALLBASE/SQL cannot find room for a new row on a primary page, it places the data on an *overflow page*. Overflow pages can be in the same DBEFile as the primary pages, or they can be in any other TABLE or MIXED DBEFile within the same DBEFileSet. When designing the hash structure, you can create a DBEFile that accommodates both primary pages and overflow pages. Specify a DBEFile size that is larger than the number of primary pages; the extra pages will then be available *only* for use by the hash structure.

Some overflow is accommodated by the fill factor described above. You must also estimate the amount of overflow space needed due to variation in key values. For an integer key with sequential values, no additional pages are needed. For a non-integer key, add 20% of the number of primary pages for overflow.

Calculating the Size of DBEFiles for Hash Structures

When deciding how large to make the DBEFiles for your hash structures, start by calculating the total amount of space needed, as follows:

$$\text{File Space} = \text{Primary Pages} + \text{Overflow Pages} + \text{Overhead}$$

In addition to primary pages and overflow pages, allow one additional page for every 252 primary pages for page directory overhead.

You can allocate the space for primary hash pages over several DBEFiles. Using this approach, you create several smaller DBEFiles that add up to the total number of primary pages plus any overflow. These files can then be moved, if you wish, to different devices. The total number of DBEFiles allocated for primary pages in a hash structure cannot exceed 16. Remember to allow one directory page for every 252 primary pages in each DBEFile.

Remember that DBEFiles used by hash structures are considered *bound*, that is, unavailable for any other purpose.

Allocating DBEFiles for Hash Structures. The DBEFiles for a hash structure are allocated in the reverse of the order in which they were added to the DBEFileSet in which the table is created. If you create the hash structure in the same transaction as the one in which you create the DBEFiles for the hash and add them to the DBEFileSet, you can be sure that other transactions will not use the new files.

The primary pages for a hash table can be spread over no more than 16 DBEFiles. Overflow pages for a hash table can be placed in any non-bound non-index DBEFile with space for them. DBEFiles containing the hash primary pages are considered *bound* and therefore unavailable for use by any other table.

When UPDATE STATISTICS is executed, all DBEFiles bound to hash primary pages are listed as 100% full to indicate that no more new pages can be allocated from these DBEFiles. (However, data may still be inserted if the allocated pages are not yet full, and the last bound DBEFile may contain overflow pages if primary pages did not use all of it.)

Mapping Logical Page Number to Physical File Location in Hash Structures

Given a number of DBEFiles with specific sizes and a number of primary pages, it is possible to determine the physical location of a logical page number within a hash structure. This information may be useful in performance tuning.

DBEFile pages are allocated for use by the hash structure in the reverse of the order in which they were added to their DBEFileSet. For example, suppose you have created three DBEFiles, each with 100 pages, and added them to DBEFileSet HashFS in the following order: File1, File2, and File3. Then, in the same transaction, you create a hash structure in HashFS with 261 primary pages. The distribution of logical pages is shown in Table 3-4:

Table 3-4. Logical Page Number and DBEFile Location in Hash Structure

DBEFile	Physical Page Number	Contents
File3	0	Page table page (directory)
	1-99	Primary Pages 1-99
File2	0	Page table page (directory)
	1-99	Primary Pages 100-198
File1	0	Page table page (directory)
	1-63	Primary Pages 199-261
	64-99	Unused (available for overflow)

Page 0 in each DBEFile is allocated as a page table page for directory information. A new page table page must be allocated every 252 pages if the file is large enough. The files in the above example do not have additional page table pages. If you are using large DBEFiles, you should include additional page table pages in your calculations.

Calculating Storage for Integrity Constraints

Integrity constraints make use of index structures to enforce the constraint condition. In addition, each constraint definition is stored in the system catalog.

Unique Constraints

Unique constraints make use of unique indexes. When you define a primary key or specify the UNIQUE option in creating a table, ALLBASE/SQL will create a unique B-tree index. The storage required for this index is the same as for unique indexes, described in a previous section. Unique indexes for unique constraints are stored in the same DBEFileSet as the table on which the primary or unique key is defined.

Referential Constraints

In addition to the unique index that is built on a table that has a PRIMARY key, each referential constraint you define on the referencing table uses a separate index structure known as a **parent-child relationship** (PCR). A PCR is different from a standard B-tree index in that it contains pointers to both the referencing and the referenced tables. The data for a PCR is stored in the same DBEFileSet as the *referenced* table.

You can get a size estimate for a PCR by using the formulas discussed in the section, “Calculating Storage for Indexes.” Be sure to include the keys from both the referenced table and the referencing table when calculating the index length of the PCR.

Hashing on Constraints

When a unique constraint is defined as you create a new table using the HASH ON CONSTRAINT clause, the table is built as a hash structure, and space is allocated as shown in the section, “Calculating Storage for Hash Structures.”

Check Constraints

Check constraints do not depend on building indexes or PCR’s. Therefore, the only additional space required is for the constraint definition in the system catalog.

Calculating Storage for the System Catalog

Each DBEnvironment requires space allocation for the system catalog, located in DBEFile0, which is associated with the SYSTEM DBEFileSet. The **system catalog** is a collection of data that describes the contents of the DBEnvironment. It contains the definitions of all database objects (tables, indexes, views, procedures, rules, constraints, stored sections, and authorization data) as well as information about the relationship between DBEFiles and DBEFileSets. Refer to the “System Catalog” chapter for details on the contents of the system catalog.

Space for the initial contents of the system catalog is allocated in the DBEFILE0 DBEFILE clause of the START DBE NEW statement. Refer to the *ALLBASE/SQL Reference Manual* for further information on the START DBE NEW statement. When a DBEnvironment is configured, ALLBASE/SQL automatically creates one DBEFileSet called SYSTEM and one DBEFile with the default file name of DBEFile0. ALLBASE/SQL associates DBEFile0 with the SYSTEM DBEFileSet and stores in it the initial system catalog data.

The minimum and default size of DBEFile0 is 150 pages, which is sufficient space for the initial system catalog information plus 20% overhead for expansion. You can use the default name and size, or you can specify a name and size (up to 16,777,215 pages) during configuration.

The following START DBE NEW statement for the PartsDBE sample DBEnvironment creates a 150 page DBEFile0 named PartsDBE0 and an MPE/iX system file name of PartsF0:

```
isql=> START DBE 'PartsDBE.SomeGrp.SomeAcct' NEW
> DBEFILE0 DBEFILE PartsDBE0
> WITH PAGES = 150,
> NAME = 'PartsF0';
```

As the system catalog becomes larger, you can add DBEFiles to the SYSTEM DBEFileSet as needed. Alternatively, you can anticipate the growth and configure the DBEnvironment with a DBEFile0 large enough to accommodate the potential system catalog growth. Conservatively, you can figure that approximately two DBEFile pages are needed to store catalog information for every 33 user tables and every 37 indexes. Add a 20% page overhead for temporary storage during query processing and expansion.

As objects are added to the DBEnvironment, the DBA can monitor the system catalog to determine if a new DBEFile must be added to the SYSTEM DBEFileSet. Refer to the “Maintenance” chapter for more details on how to determine when to add a DBEFile to the SYSTEM DBEFileSet.

The following factors affect system catalog size:

- SQL statements that create ALLBASE/SQL objects and grant authorities.
- The number of sections stored in the system catalog from preprocessed SQL statements.
- The number of columns in all tables and views in the DBEnvironment.
- Query operations that require sorting.

Storage of Definitions for Newly Created Objects

SQL statements that create objects or alter existing objects will add or update rows in the system catalog each time they are processed. The following statements add rows to the system catalog:

ADD TO GROUP	ALTER TABLE	CREATE DBEFILE
CREATE DBEFILESET	CREATE GROUP	CREATE INDEX
CREATE PROCEDURE	CREATE RULE	CREATE TABLE
CREATE VIEW	DECLARE CURSOR	GRANT
PREPARE		

Sections are removed when any DROP statement is executed.

Number of Columns in Tables

The number of columns in the DBEnvironment affects the system catalog size because rows are stored in the system catalog for each column in each table in the DBEnvironment. Therefore, as columns are added to tables and new tables are created, the system catalog size increases.

Definitions of Rules, Procedures, Constraints, and Views

The system catalog contains a definition string for each rule, procedure, constraint, and view you create. These definitions are stored as character data, and they add additional overhead to the system catalog.

For procedures, sections and static information are also stored in the system catalog in a base table that is not exposed to the user.

Storage of Sections

Sections can be stored in the system catalog by SQL statements when they are preprocessed as part of an application program. They can also be stored interactively via ISQL.

It is difficult to put an exact number on how many DBEFile pages are needed for a given number of sections since more complex statements use more space. For example, a preprocessed SELECT statement that describes a three-table join stores a larger section than a SELECT on one table.

Refer to the *ALLBASE/SQL Reference Manual* chapter “Constraints, Procedures, and Rules” for information about sections created for procedures. Refer to the appropriate *ALLBASE/SQL Application Programming Guide* for more information on which SQL statements in an application cause sections to be stored.

Calculating Space Needed for Sections

For each section in the SYSTEM.SECTION view, ALLBASE/SQL stores at least one input tree and, conditionally, one run tree. A tree may take one or more rows depending on the SQL statement. A row is 504 bytes. SQL statements containing several column names or host variables may generate multiple-row input trees. Views only store input trees. Complex SELECT statements such as joins and queries using the GROUP BY or HAVING clauses tend to generate multiple-row run trees.

To calculate the number of pages needed to store a module, you must calculate the number of non-query SQL statements (N1) and the number of queries (N2) in the module. The formula shown below calculates the approximate number of bytes required to store sections. Only space for sections belonging to modules is calculated.

$$\text{Tree Bytes} = (N1 * 504 + N2 * 504 * 1.2) * 2$$

The above formula assumes that 20% of the queries are complex. You can alter the formula to fit a specific module by changing 1.2 to the appropriate ratio. If 50% of the queries are complex, use 1.5 instead of 1.2 to calculate tree bytes for that module.

Assume a program stores eighteen sections during preprocessing and three of those sections are queries. N2 is three and N1 is fifteen. The number of bytes needed for the module is:

$$\text{Tree Bytes} = (15 * 504 + 3 * 504 * 1.2) * 2 = 18,748.4$$

Rounded up, the value is 18,749. A DBEFile page can contain eight 504-byte rows. To translate the 18,749 bytes into pages use the following calculation:

$$\frac{18,749 \text{ Bytes}}{504 \text{ Bytes Per Row}} = 37.2$$

Rounding up, the result is 38 rows. Then,

$$\frac{38 \text{ Rows}}{8 \text{ Rows Per Page}} = 4.75$$

Rounding up, the result is 5 pages. The result is always rounded up because DBEFiles cannot be created with partial pages. These formulas help you determine how many DBEFile pages should be available in the SYSTEM DBEFileSet for preprocessed applications. Keep in mind the other factors that affect space in the SYSTEM DBEFileSet such as the number of objects in the DBEnvironment and the DBEFile pages needed to perform CREATE INDEX, ORDER BY, GROUP BY, HAVING, and DISTINCT operations. Refer to the “Physical Design” chapter for more information.

Remember to UPDATE STATISTICS and COMMIT WORK after an application is preprocessed to get an accurate reading of how many DBEFile pages are left in the SYSTEM DBEFileSet.

Monitoring System Catalog Size

As objects are added to the DBEnvironment, the DBA can monitor the system catalog to determine if a new DBEFile must be added to the SYSTEM DBEFileSet. Refer to the “Maintenance” chapter for details on how to determine when to add a DBEFile to the SYSTEM DBEFileSet.

Although user-defined tables can be created in the SYSTEM DBEFileSet, it is recommended that you create new DBEFileSets to keep user data separate from system catalog data to facilitate maintenance and performance tuning. Refer to “Arranging Tables and Indexes in DBEFileSets,” earlier in this chapter.

Calculating Storage for Logging

The DBA should calculate the appropriate size of log files for use with the ALLBASE/SQL log. Eventually, log files can become full, and the result is known as a **LOG FULL** error condition. To avoid this, you must set up logs that are large enough for your system’s heaviest needs.

Understanding Log File Characteristics

Log files are different from DBEFiles. **Log files** store log records used by ALLBASE/SQL to perform recovery. **DBEFiles** contain tables and indexes.

Log files are composed of 512-byte pages. A log file can be from 250 to 524,287 pages. The number of pages in a log file is determined when it is created. The maximum size of a single log file is 4 gigabytes. A single DBEnvironment can have up to 34 log files configured, providing a maximum of 136 gigabytes of log file space. Log files are not associated with a DBEFileSet.

Log Records and Transactions

ALLBASE/SQL logs changes made to DBEFile pages. Changes are written to a log file (or files) as **log records**. At least one log record is created for each data or index page that is changed. For example, if you update a column in a table that has three indexes created on that column, you have a minimum of four log records written to the log file: one for the change to the column, and one for each of the three indexes.

A log record is a maximum of twice the size of the updated row plus overhead of about 50 bytes. For example, updating a 200-byte column would create a log record of approximately 450 bytes. An update uses the largest amount of space in a log record; other kinds of activity (INSERT and DELETE operations) use smaller log records.

Note that a log record may be larger than a log page, which is 512 bytes. On the other hand, a log page may contain several log records. Log space is allocated in pages, whereas log entries are created as log records, which are inserted into one or more pages.

Transaction size also affects log file size. A **transaction** is one or more SQL statements that together perform a unit of work in a DBEnvironment. Transactions are implicitly begun by ALLBASE/SQL whenever a user or application program executes most SQL statements. You can also explicitly start a transaction with the BEGIN WORK statement. The transaction is not ended until a COMMIT WORK or ROLLBACK WORK is executed.

A log fileset should be large enough to hold all current transactions. If one file in the fileset fills up, ALLBASE/SQL switches to the next available file (if there is one) and continues logging. If the log fills up before a transaction in progress completes, and if there is no more file space available, ALLBASE/SQL rolls back the current transaction. In most cases, all other transactions are also rolled back, whether or not they involve updates. This can happen if your transaction is larger than available file space in the log. When such a LOG FULL condition occurs, you will not be able to process any more transactions of the same size until some log file space is recovered (in nonarchive logging only), or until a new log file is added to the DBEnvironment. Use the SHOWLOG command and the CHECKPOINT statement to monitor log file use, as described in the “Backup and Recovery” chapter. Use the SQLUtil ADDLOG command to add log files. If you are logging in archive mode, use the STORELOG command to free archive logs for reuse.

Using Archive or Nonarchive Logs

You can specify archive or nonarchive logging for the DBEnvironment. Nonarchive logging tracks all *current* DBEnvironment activity in a log file. This lets you roll back incomplete transactions when necessary, maintaining consistency in database tables. When transactions are no longer current, the space they occupy in the log file can be reused by other log records. Archive logging tracks *all* DBEnvironment activity continuously from the time it is enabled.

By default, nonarchive logging is in effect after you issue a START DBE NEW or START DBE NEWLOG statement. If you have the TurboSTORE software, you can turn archive logging on by using the SQLUtil STOREONLINE command, which backs up the DBEnvironment and then enables archive logging. If you do not have TurboSTORE, you can enable archive logging by following the procedure described under “Static Full or Partial Backup Procedures in Archive Mode” in the “Backup and Recovery” chapter. Once archive logging is on, you can only disable it with a START DBE NEWLOG statement.

For many installations, nonarchive logging is appropriate for the phase of database creation and initial loading of tables. Once the DBEnvironment has been loaded, you can use the SQLUtil STOREONLINE command to create a complete backup and turn on archive logging at the same time. (For complete information about the process of backing up the DBEnvironment and turning on archive mode, refer to the “Backup and Recovery” chapter.)

Using Single or Dual Logging

When you create the DBEnvironment, you specify either SINGLE or DUAL logging. In **single logging**, ALLBASE/SQL maintains one set of log files in either archive or nonarchive mode. For greater security, you can specify **dual logging**, in which a duplicate set of log files is maintained. When dual logging is in effect, all the procedures described in this chapter apply to the files of *both logs*.

Using Multiple Log Files

By default, ALLBASE/SQL creates a single log file (or two files, in dual logging) when you create the DBEnvironment. During normal operation, transactions are logged until there is no more space available. This is known as a LOG FULL error condition. If a LOG FULL condition arises, all transactions are rolled back.

With archive logging, LOG FULL occurs when the end of the last file has been reached, and if there is no other log available to switch into. You can assure that there is always another log to switch into by making sure that all transactions commit promptly, and by using the SQLUtil STORELOG command to store log files as they fill up. Once stored, and once all the transactions in them are complete, archive log files are marked available for reuse.

With nonarchive logging, LOG FULL occurs if there is still no available space in the file after ALLBASE/SQL takes a checkpoint while using the last file in the fileset. To prevent LOG FULL in nonarchive logging, make sure the combined size of all the files in the fileset is larger than the space required for the largest number of concurrent transactions you expect in the DBEnvironment.

To prevent a LOG FULL condition from ever arising, you can use multiple log files to provide enough log file space for the largest amount of DBEnvironment activity that will ever require logging at any one time. You can do this by setting up a log containing a circular fileset (or two circular filesets, for dual logging). A circular fileset is a group of reusable files belonging to the same log. You can add log files to the circular fileset without stopping the DBEnvironment. When you are using a circular fileset, a LOG FULL condition can only arise if there is no available space in any of the files in the fileset. With either archive or nonarchive logging, you may use up to 34 additional log files in a log (or 34 additional files for each set in dual logging).

For a complete discussion of log file types, refer to the “Backup and Recovery” chapter.

Sample Log Configuration

The following START DBE statement creates a sample DBEnvironment named PartsDBE in dual logging mode with two 256-page log files named PartsDBELog1 and PartsDBELog2 and system file names of PartsLg1 and PartsLg2:

```
isql=> START DBE PartsDBE.SomeGrp.SomeAcct' NEW
> DUAL LOG, LOG DBEFILE PartsDBELog1 AND PartsDBELog2
> WITH PAGES = 256,
> NAME = 'PartsLg1' AND 'PartsLg2';
```

The following SQLUtil ADDLOG command adds another file to each of the dual logs. The result is two log filesets:

```
>> addlog
DBEnvironment Name: PartsDBE.SomeGrp.SomeAcct
Maintenance Word: 
Enter Log File Name(s) Separated by a Blank? PartsLg3 PartsLg4
New Log File Size? 300
Add Log File (y/n)?  y

Log files 'PartsLg3' and 'PartsLg4' were Added.
Log Identifier Is: 2
```

Whatever size you choose to make the individual log files, the disk space used for logging will double if you use dual logging.

Disk Space for the Log

The major difference between the archive and nonarchive log filesets is that archive log files must be backed up for possible use in the event of a media failure. Nonarchive log files do not need to be backed up. Also, certain operations (such as initial table loads) produce more logging in archive mode than in nonarchive mode. In the following sections, size estimates are based on archive logging, so the result will be more than adequate for nonarchive logging as well.

When a DBEnvironment is configured, ALLBASE/SQL creates at least one log file with a default file name of DBELOG1 (and DBELOG2 if DUAL LOG is specified), and a default size of 250 pages. The file name is relative to the DBECon file name unless you specify a fully qualified file name. You can use the default name and size or specify a name and size (up to 524,287 pages) during configuration. You use the SQLUtil ADDLOG command to add additional files to the DBEnvironment.

Determining the Number of Log Files

In nonarchive logging, log file space can be reclaimed once transactions are committed or rolled back. This means that your log file (files, for dual logging) should be large enough to hold the single longest transactions plus all concurrent transactions.

In archive logging, log file space is reclaimed and reused by storing each log file after it becomes full. As with nonarchive mode, the sizes of all component files taken together must be large enough to contain all active transactions. But for archive logging, there should be at least two separate files in the log (two files for each log, in the case of dual logging) so as to permit log switching followed by eventual log backup and reuse.

Estimating Log File Size

To estimate the total size (sum of the sizes of component files) of a log:

1. Calculate the number of log pages (LP) that the longest transaction will require.
2. Multiply the results (LP) by the maximum number of concurrent transactions allowed in the DBEnvironment to get the total number of log pages (TLP) required. (This assumes that the maximum number will all be involved in operations that require the same amount of logging. If you are running transactions in applications such as report generators that do *not* involve logging, subtract the non-logging transactions from the maximum, and use the result in the calculation.)
3. Add 38 pages per log file for overhead used by ALLBASE/SQL to maintain the log.

Step 1 calculates the number of log pages needed for the longest transaction. The longest transaction is the transaction that affects the largest number of rows in a DBEnvironment between a BEGIN WORK and a COMMIT WORK or ROLLBACK WORK statement issued either programmatically or through ISQL.

The formula used to approximate the log pages needed for the longest transaction is as follows:

$$LP = \frac{(DBE\ Size) * (PercentInserted + PercentDeleted + PercentChanged) * 2}{500\ Bytes\ Per\ Page}$$

PercentInserted, *PercentDeleted*, and *PercentChanged* are the percentages of your tables that are affected by the transaction. You can derive these values by dividing the number of rows inserted (or deleted or changed) by the total number of rows in the table. These percentages are then applied to the entire DBEnvironment to arrive at a number of log pages for storing all the information (data from tables, indexes, and constraints, plus overhead) that is logged by the transaction. DBE Size (in bytes) can be estimated by using the following query after doing an UPDATE STATISTICS for all tables in your DBEnvironment:

```
isql> SELECT 4030*SUM(DBEFUPAGES) FROM SYSTEM.DBEFILE;
```

Example. Assume that you have a DBEnvironment containing six tables with 30,000 rows each (total 180,000 rows). Together, all the tables and indexes occupy five megabytes. Your longest transaction will insert 50 rows, delete 25 rows, and change 100 rows. This calculates to .03% inserted, .01% deleted, and .06% changed.

If you apply these numbers to the calculation for LP in step 1, you get the following results:

$$LP = \frac{5,000,000 * (.0003 + .0001 + .0006) * 2}{500} = 20\ Pages$$

In step 2, you multiply the number of log pages (LP) by the total number of concurrent transactions allowed in the DBEnvironment. The total number of concurrent transactions is a value that you can set at DBEnvironment configuration and change later using SQLUtil or START DBE. It is stored in the Maximum Transactions parameter in the DBECon file, which is discussed in the next chapter, “DBEnvironment Configuration and Security.”

Note Using the value for Maximum Transactions in the preceding formula assumes that all transactions are equally involved in logging. In other words, it is assumed that each transaction contains roughly the same amount of INSERT, UPDATE, or DELETE operations.

If this is not true, you may wish to use a weighted or average value in arriving at the total. While the above calculations will provide more than enough log file space, you can arrive at an optimal log size by using weighted values.

The following is the calculation for total log pages:

$$TLP = LP * (Maximum Transactions)$$

Suppose that Maximum Transactions is set to 30. The following calculations give the total number of log pages (TLP) based on the previous calculations:

$$TLP = 20 Pages * (30 Transactions) = 600 Log Pages$$

Finally, in step 3, add 38 pages of overhead used by ALLBASE/SQL to arrive at the size your log file should be.

$$Log File Size = 600 Log Pages + 38 Overhead Pages = 638 Pages$$

For nonarchive logging, one log file larger than 638 pages should be sufficient (two log files, if you are using dual logging). If you add additional log files, ALLBASE/SQL will switch to the second file if additional space is needed.

For archive logging, you should use at least two separate files, so that ALLBASE/SQL can switch from the first to the second, thus permitting the first to be backed up. The size of each log file should be no smaller than 638 pages, but the actual size should take into account the number of log files available for log switching and the frequency with which logs can be backed up on your system. If backups are not done at frequent intervals, a larger number of log files will be necessary in archive mode.

Note Updates involving indexed pages can result in a considerable amount of B-tree page splitting, which requires additional logging.

Until your system is in full production, use a larger than necessary log. You can then monitor the free log space from time to time until you know the optimal size for your configuration. You can easily add and purge log files as needed with the SQLUtil ADDLOG and PURGELOG commands.

For all the foregoing calculations on the required size of log files, it is assumed that all users are starting and ending transactions in reasonable ways. However, if a user starts a transaction and then neglects to commit work for a very long time, files in which that transaction is active will not be available for switching as assumed in the calculation. In such a case, the calculations are meaningless.

Note If you are using dual logging, you specify two different filenames for each file you add to the DBEnvironment with the START DBE NEW, START DBE NEWLOG, or ADDLOG commands.

Additional information about creating and maintaining logs is found in the “Backup and Recovery” chapter.

Calculating Temporary Disk Space

ALLBASE/SQL uses temporary disk space of two kinds for sorting and other operations:

- Space in temporary operating system files
- Space in the SYSTEM DBEFileSet

Queries that contain the following clauses use temporary pages in operating system files to store temporary information:

- DISTINCT
- GROUP BY
- ORDER BY
- UNION (but not UNION ALL)

In addition, the following operations also use temporary table space in the SYSTEM DBEFileSet to store sorted data:

- Queries that use sort/merge join processing.
- The CREATE INDEX statement, when it operates on tables containing data.

Sorting normally requires about twice the space occupied by the result table being sorted. Three times the space is required for sorting the result of a SELECT that employs a defined key field or that joins two or more tables and uses an ORDER BY clause. Space for sorting is taken from the **temp space** available on the system. Temporary files are created in the temp space, and they are removed after the query is processed. Space is used in the SYSTEM DBEFileSet whenever a temporary table is created as a result of sorting or query processing. The SYSTEM DBEFileSet must be large enough to store these temporary tables.

ALLBASE/SQL will use space in the current group if you do not specify any other location for temporary space. If this area is not adequate for your needs, you can use the CREATE TEMPSPACE statement to define a different area, which then serves as the location for creating temporary files when they are needed for sorting.

The default number of pages used per file for temp space is 256. The total amount of space used is all that is available in the temp space area, which on MPE/iX is the current group.

The following is an example of a statement that defines an existing MPE group as a temp space:

```
CREATE TEMPSPACE Temp1 WITH MAXFILEPAGES=400, LOCATION= 'SomeGrp.SomeAcct';
```

The group SomeGrp.SomeAcct must exist already. After you issue this statement, the group SomeGrp.SomeAcct becomes available for the creation of one or more temporary files, each of which can be up to 400 pages in size.

Many files may be opened in one temp space, and each may be expanded up to 400 pages. The number 400 is an arbitrary value chosen for this example. See the “CREATE TEMPSPACE” section in the *ALLBASE/SQL Reference Manual* for the range of allowed values.

Note that the CREATE TEMPSPACE statement does not actually create the group SomeGrp.SomeAcct; it merely stores the description of the TempSpace in the system catalog table SYSTEM.TEMPSPACE. The actual temporary files for sorting are created at the time the sort is carried out, and they are dropped when the sort is finished. If you want to remove the TempSpace definition from the system catalog, use the DROP TEMPSPACE statement.

Controlling the Use of Temporary Space

The order in which you carry out database operations may determine the amount of temporary space required. For example, the following two scenarios use the same number of pages at the end of the operation, but require different amounts of temporary space because the CREATE INDEX statement calls the same sort routine for an empty table as for a non-empty table:

```
(1) CREATE TABLE
    CREATE INDEX
    INSERT 20000 rows
```

Scenario 1 does not require the use of temporary space, since the table is never sorted. Instead, the index is updated as a part of each INSERT operation.

```
(2) CREATE TABLE
    INSERT 20000 rows
    CREATE INDEX
```

Scenario 2 uses temporary files for sorting, and it also creates a temporary relation to hold the sorted table prior to the creation of the index.

Estimating Shared Memory Requirements

Each time a multi-user DBEnvironment session is started with either the START DBE statement or the first CONNECT statement, if the autostart option is set to on, a block of shared memory is reserved for this ALLBASE/SQL session. Until the DBEnvironment session is ended, all users and programs accessing the DBEnvironment share this allocated memory. This memory remains reserved until the DBEnvironment is stopped, at which point the memory is made available for re-use by the system. If AUTOSTART is ON, memory remains reserved until the last DBE session open against the DBEnvironment terminates. If AUTOSTART is OFF, memory remains reserved until the DBA issues the STOP DBE statement.

ALLBASE/SQL uses shared memory for three types of buffers:

- Runtime control block buffer
- Data buffer
- Log buffer

You can specify how much shared memory is to be allocated to each buffer when you create the DBEnvironment with the START DBE NEW statement. The parameters you specify for *ControlBlocks*, *DataBufferPages*, and *LogBufferPages* are stored in the DBECon file. A DBA can temporarily override several of the parameters that comprise shared memory by using the

START DBE statement, or alter each of the shared memory parameters by using the SQLUtil ALTDBE command. Initial memory allocation and system configuration is performed before installation of the product.

Estimating Runtime Control Block Buffer Pages

Each type of DBCore service requires a specific number and size of control block buffer pages. The number and type needed at any one time depends on such factors as the number, duration, and type of concurrent transactions, the amount of row or page level locking, and the amount of update activity occurring. Whether or not a runtime control block buffer page is available for re-use by the system depends on the type of runtime control blocks being used. Transaction lock control block pages may not be available for re-use until after the transaction successfully terminates with either a COMMIT WORK or a ROLLBACK WORK statement.

Lock management is the single greatest user of shared memory. The greater the number of concurrent locks held, the greater the number of runtime control block buffer pages needed to manage these locks. Consequently, a program designed to have shorter transactions, coarser lock granularity, or more efficient concurrency practices is less likely to deplete the amount of shared memory available.

Effects of Page and Row Level Locking

Page level locking uses more runtime control blocks than table level locking, since each page must be locked. Row level locking uses even more runtime control blocks than page level locking, since each row must be locked individually. This can cause the allocation of a considerable amount of shared memory. The following indicates the maximum number of locks associated with table, page, and row level locking:

Table 3-5. Maximum Numbers of Locks Obtained at Different Granularities

Locking Level	Maximum Number of Locks
Table Level	1
Page Level	$n + 1$
Row Level	$m + (n + 1)$
	where n is the number of pages in the table and m is the number of rows in the table.

Table level locking requires a single lock. Page level locking requires up to n page level locks plus one intention lock at the table level. Row level locking requires up to m row level locks plus up to $(n + 1)$ intention locks at the page and table levels. Because row level locking on a large table can consume a tremendous number of runtime control blocks, the use of the PUBLICROW table type on large tables is discouraged. Large tables for which maximum read/write concurrency is desired should generally be defined as PUBLIC. The PUBLICROW table type should generally be reserved for use on small tables.

As an illustration, consider a table that occupies 500 pages in a DBEFile. Assume that each page holds 20 rows. Suppose that 40% of all rows are affected by concurrent activity using index scans at the RR isolation level on this table—that is, at any one time about 40% of all rows are being read or updated. Further assume that these rows are spread out among 80% of the pages in the table.

With page level locking, the number of runtime control blocks used is as follows:

$$RCB = 500 * .8 = 400$$

Adding one for the table level intent lock, the total is as follows:

$$Total = 400 + 1 = 401$$

If each RCB occupies 90 bytes, the total memory required would be 36,090 bytes, or 9 runtime control block buffer pages.

With row level locking, the number of runtime control blocks used is as follows:

$$RCB = 500 * 20 * .4 = 4,000$$

In addition to row locks, you need to add the number of page locks (intent locks) from the earlier calculation. Finally, you should add one intent lock for the table.

The total is as follows:

$$Total = 4,000 + 400 + 1 = 4,401$$

Then, if each RCB occupies 90 bytes, the total memory required would be 396,090 bytes, or 97 runtime control block buffer pages.

In addition to the shared memory required for locks, row level locking also requires additional CPU time to fetch and release the locks.

Running out of Shared Memory

The DBCore allocates memory for runtime control blocks in 4 Kilobyte pages. More specifically, the DBCore can allocate up to 72 4-Kilobyte pages of memory for the control structures for the data buffer pages, the log buffer pages, and the runtime control block pages.

When the DBCore cannot obtain the required number of runtime control block buffer pages, the transaction requesting the additional shared memory is rolled back and ALLBASE/SQL returns the error code -4008.

Because the conditions that caused the DBEnvironment to run out of shared memory may not exist if the transaction is simply restarted, the programmatic user can check for this error return code value and re-execute the program a finite number of times if it occurs. The ISQL user can simply re-execute the transaction.

If a DBEnvironment consistently runs out of shared memory, you can increase the number of runtime control block buffers by 20 percent and re-try the affected transactions. If you are using large PUBLICROW tables, you can use the ALTER TABLE statement to convert to PUBLIC mode. With large LOAD and INSERT operations, use the LOCK TABLE statement with the EXCLUSIVE option to avoid depleting shared memory.

Note The minimum number of runtime control block buffer pages is 17. The maximum is 2,000. The default is 37.

The number of 4096-byte pages in the runtime control block buffer is set using the START DBE NEW statement or the SQLUtil ALTDDBE command. When you specify values for these parameters in START DBE and START DBE NEWLOG, you do not update the value stored in the DBECon file but change the value for the current DBEnvironment session only.

Estimating Data Buffer Pages

During query processing, pages from DBEFiles currently being accessed are held in the **data buffer**. The number of 4 Kilobyte pages to be allocated in the data buffer is specified in the **BUFFER** clause of the **START DBE NEW** statement. This number should be based on the number of concurrent users and the type of applications. You should start with a number of data buffer pages equal to slightly more than the maximum number of concurrent users on your system.

Each transaction may need from one to several buffer pages depending on the type of query being processed. The more complex a query the more buffer pages are needed. For a complex query, the required number of data buffer pages may be from 5 to 15 times the maximum number of concurrent transactions. Because some of the buffer pages are shared in a multiuser mode, the page requirement per user decreases as the number of users increases.

Note The minimum number of data buffer pages is 15. The maximum number is 50,000. The default is 100.

 You can now allocate enough data buffer pages to keep a whole DBEnvironment in memory, if necessary.

The number of pages in the data buffer can be temporarily overridden with the **START DBE** statement. The **ALTDBE** command in **SQLUtil** allows you to permanently change the number of buffer pages.

Setting the Memory Resident Buffer Flag

You can force data buffer pages to remain memory resident by setting the *MemoryResidentEnabled* flag in the **DBECon** file to **YES**. Setting the *MemoryResidentEnabled* flag to **YES** may improve database performance on machines with a small amount of memory. However, system virtual memory swapping overhead may increase for other applications on the system, since a part of the memory frame pool becomes unavailable.

Setting the *MemoryResidentEnabled* flag to **YES** is useful for database servers where very little is going on besides database work, and the database buffers occupy a small fraction of real memory in the system.

Use the **SQLUtil ALTDBE** command to change the setting of the Memory Resident Buffer Flag, as in the following example:

```

>> altdbe

DBEnvironment Name: PartsDBE
Maintenance Word:
AutoStart Mode (on/off) (opt):
User Mode (opt):
DDL Enabled (y/n) (opt):
No. of Runtime Control Block Pages (opt):
No. of Data Buffer Pages (opt):
Data Buffer Pages Memory Resident (y/n) (opt): yes
No. of Log Buffer Pages (opt):
Max. Transactions (opt):
Maximum Timeout (opt):
Default Timeout (opt):
Authorize Once per Session (on/off) (opt):
Alter DBEnvironment Startup Parameters (y/n)? yes

DBEnvironment startup parameters altered.

```

Estimating Log Buffer Pages

The log buffer holds before- and after-images of pages that are changed during a transaction. You specify the number of 512-byte pages for the log buffer in the `BUFFER` clause of the `START DBE NEW` statement.

In deciding the number of log buffer pages, you should consider the duration of a typical transaction, that is, the time between a `BEGIN WORK` statement and its corresponding `COMMIT WORK` statement. As log records are generated during a transaction, they are kept in the log buffer until any one of the following occurs:

- A `COMMIT WORK` is performed.
- A `CHECKPOINT` is performed.
- All the log buffers are full.

When any one of the above occurs, the log buffer pages are written to the log file. Once transactions in the buffer are written to disk, the buffer pages can be used again. If transactions are short, the number of log buffer pages need not be very large, since the log records will be written to disk frequently. However, if there are lengthy transactions and few log buffer pages, transactions spend time forcing log records to disk.

A minimum of 24 log buffer pages is required; this is the default value supplied by `ALLBASE/SQL`. You can request up to 1024 log buffer pages. You can temporarily override the number of log buffer pages with the `START DBE` statement. The `ALTDBe` command in `SQLUtil` allows you to permanently change the number of buffer pages. For more detailed discussion of data buffer page size, refer to the *ALLBASE/SQL Performance and Monitoring Guidelines*.

Estimating the Number of Transactions

As the number of users accessing the DBEnvironment increases, you should increase the number of concurrent transactions allowed. The maximum should be set slightly higher than the number of expected users because ALLBASE/SQL may start a second transaction on the user's behalf when executing data definition statements such as CREATE. Set the maximum to twice the number of users if you expect to be doing a lot of data definition.

The minimum number of transactions is 2, and the default value is 50. There is no limit on the number of transactions you can request. You can temporarily override the number of transactions with the START DBE statement. The ALTDBE command in SQLUtil allows you to permanently change the maximum number of transactions.

Implementing the Design

Once both logical and physical design have been established, you are ready to create a DBEnvironment and the databases within it. The next chapters lead you through the creation of the sample DBEnvironment using the design guidelines presented in the "Logical Design" and "Physical Design" chapters.

DBEnvironment Configuration and Security

Before you can create ALLBASE/SQL databases, you must configure a DBEnvironment. After you configure the DBEnvironment, you create the physical storage space for the databases, then you create the actual tables, views, and indexes.

If this is your first time using ALLBASE/SQL, you should study the examples in this chapter, which are based on the sample DBEnvironment, PartsDBE. The “Sample DBEnvironment” appendix in the *ALLBASE/SQL Reference Manual* contains a complete description of PartsDBE.

The topics described in this chapter are:

- Required MPE/iX Capabilities
- Using START DBE NEW
- Creating Audit DBEnvironments
- Creating DBEFileSets and DBEFiles
- Creating the DBEnvironment security scheme
- Managing DBEnvironment Sessions

After reading this chapter, you can configure your own DBEnvironments and create databases using your own design specifications.

Required MPE/iX Capabilities

To be able to configure a DBEnvironment, you must have the following MPE capabilities:

- Standard user capabilities (that is, you do not need SM or OP).
- File access capabilities, as follows:
 - SF, to save user files permanently.
 - IA, to gain interactive access.
 - BA, to gain batch access. This attribute is required only if you are going to use batch (job) mode to access or create the DBEnvironment.
 - ND, to be able to use non-shareable devices. This attribute is required if you wish to use devices other than disk, such as line printers and magnetic tape devices. ND is not necessary for standard job/session input and list devices \$STDIN and \$STDLIST.

Using START DBE NEW

Configuring a DBEnvironment begins with using the START DBE NEW statement, whether in interactive mode through ISQL, using ISQL command files. Refer to the *ALLBASE/ISQL Reference Manual* for more information on command files. START DBE NEW may be used only once for a given DBEnvironment:

```
START DBE 'DBEnvironmentName' NEW [StartUp Values];
```

If you try to execute a START DBE NEW statement for an existing DBEnvironment, ALLBASE/SQL returns an error. You must purge the existing DBEnvironment and the log files associated with it using SQLUtil before you can use START DBE NEW to create a DBEnvironment with the same name. The START DBE NEW statement allows you to supply **startup parameters**, which are used to set operating limits, such as the user mode and the number of log buffers, each time the DBEnvironment is started. The startup parameters are stored in a file called the DBECon file.

If no startup parameters are specified, ALLBASE/SQL provides default values. Table 4-1 shows the startup parameters that are stored in the DBECon file and their default values.

Table 4-1. DBECon Default Startup Parameters

Parameter	Default Startup Option
DBECreator	your DBEUserID
Maintenance Word	none
DBEnvironment Language	NATIVE-3000
AutoStart	ON
User Mode	SINGLE
DBEFile0 Name	DBEFILE0
Log File Name(s)	DBELOG1 DBELOG2 *
Archive Mode	OFF
DDL Enabled	YES
Number of Run Time Control Block Pages	37
Number of Data Buffer Pages	100
Memory Resident Data Buffer Pages	NO
Number of Log Buffer Pages	24
Maximum Transactions	2
Maximum Timeout	None
Default Timeout	Maximum
Authorize once per session	OFF
	* for dual logging

You can override all these parameters except DBECreator, maintenance word, autostart, archive mode, DDL Enabled, Memory Resident Data Buffer Pages, and Authorize Once per Session by specifying other values in the START DBE NEW statement.

After the DBEnvironment has been configured, you can change some startup values using SQLUtil. See Table 7-1 in the “Maintenance” chapter for a description of each parameter, including how it can be changed.

Supplying Startup Parameters with START DBE NEW

Your DBEnvironment may require startup parameters different from the default values supplied by ALLBASE/SQL. For example, you may want to create a multiuser DBEnvironment, or assign a more descriptive name to DBEFile0.

The sample DBEnvironment was configured with the following statement:

```
isql=> START DBE 'PartsDBE.SomeGrp.SomeAcct' MULTI NEW
> DUAL LOG,
> TRANSACTION = 5,
> DBEFILE0 DBEFILE PartsDBE0
> WITH PAGES = 150,
> NAME = 'PartsFO',
> LOG DBEFILE PartsDBELog1 AND PartsDBELog2
> WITH PAGES = 256,
> NAME = 'PartsLg1' AND 'PartsLg2';
```

Note that the ALLBASE/SQL defaults were used for the number of buffer pages.

Figure 4-1 is a diagram of the PartsDBE DBEnvironment immediately after configuration. The DBECon file is expanded to show startup parameters and the log file directory.

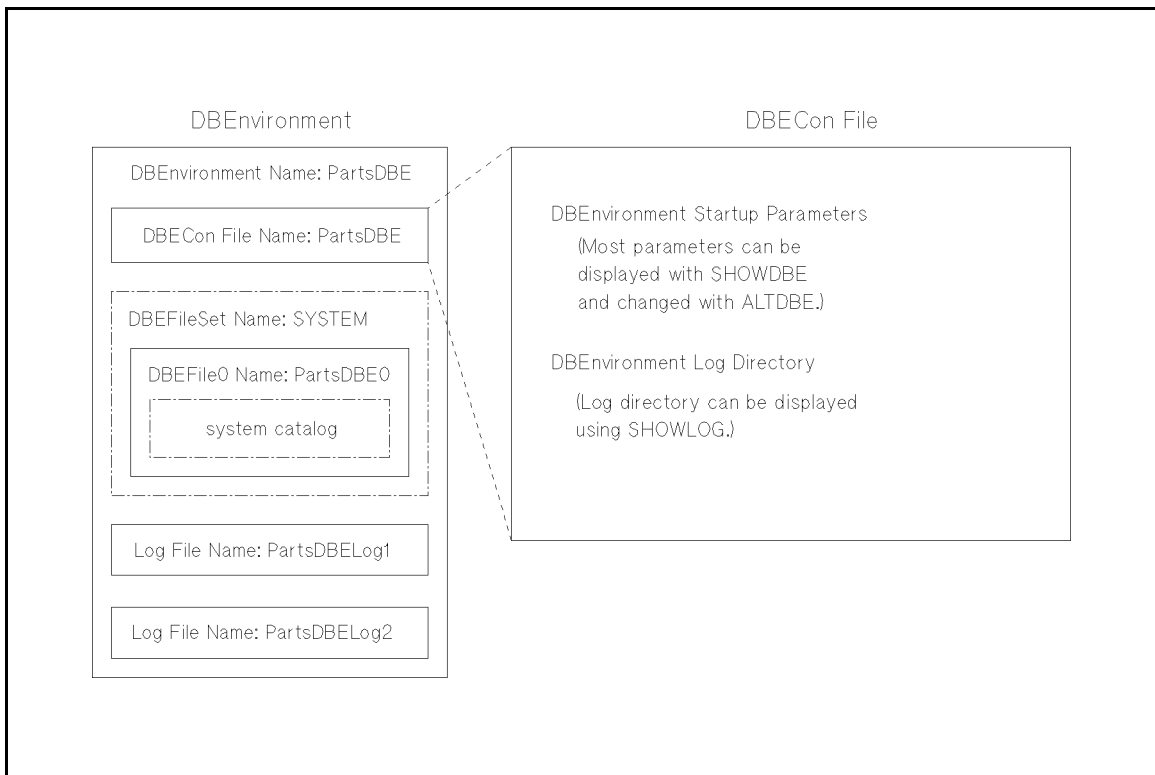


Figure 4-1. The Sample DBEnvironment Immediately After Configuration

A newly configured DBEnvironment has the following elements:

- One DBECon file containing the DBEnvironment configuration parameters. The name of the DBECon file is the same as the DBEnvironment name.
- One or two log files. The system file name(s) are specified in the NAME= clause of the LOG DBEFILE option of the START DBE NEW statement. The default name(s) are DBELOG1 and DBELOG2.
- One DBEFileSet named SYSTEM.
- One DBEFile0 DBEFile to store the initial system catalog data. The system file name is specified in the NAME= clause of the DBEFILE0 DBEFILE option of the START DBE NEW statement. The default name is DBEFILE0. This DBEFile is associated with the SYSTEM DBEFileSet.
- One system catalog to store information about the DBEnvironment. The tables, views, and indexes constituting the ALLBASE/SQL system catalog (refer to the “System Catalog” chapter in this guide) are created in the SYSTEM DBEFileSet. At first, entries in the system catalog describe the initial state of the DBEnvironment, including the pseudotables and views of the system catalog itself. As objects are added, the system catalog is updated.

Refer to the “Maintenance” chapter for additional information about the DBECon file parameters and how to change them.

Log Files

ALLBASE/SQL creates a log file when the DBEnvironment is configured. A log file can be from 250 to 524,287 pages. Each page is 512 bytes. You can set the size and name of the log file(s) using the LOG DBEFILE option of the START DBE NEW statement:

```
isql=> START DBE 'PartsDBE.SomeGrp.SomeAcct' MULTI NEW
> .
> .
> .
> LOG DBEFILE PartsLg1
> WITH PAGES= 350,
> NAME= 'PartsLg1';
```

If you do not specify a file name or log size, ALLBASE/SQL creates a log file with the default size of 250 pages and the default name of DBELOG1. The file is created in the same group and account as that of the DBECon file unless you specify a different group. The account name, if you specify it, *must* be the same as that of the DBECon file. In the above example, the name PartsLg1 was chosen for the sample DBEnvironment. Refer to the “Physical Design” chapter of this guide for guidelines on determining the size of the log file.

Dual Logging

Successful recovery requires a good copy of each log record. Since a log file is critical to the recovery procedure, ALLBASE/SQL provides dual logging which improves the probability of successful recovery by maintaining two log files.

You must specify two log file names when you specify dual logging:

```
isql=> START DBE 'PartsDBE.SomeGrp.SomeAcct' MULTI NEW
> DUAL LOG,
> LOG DBEFILE PartsDBELog1 AND PartsDBELog2
> WITH PAGES = 256,
> NAME = 'PartsLg1' AND 'PartsLg2';
```

4-4 DBEnvironment Configuration and Security

The SQLUtil SHOWLOG command will display two log file names when dual logging is in use.

A hard crash on a device containing a log file is potentially very serious, since it reduces the chances of being able to recover the DBEnvironment. Whenever you use dual logging, you can safeguard against both log files being damaged by a hard crash by locating the two logs on separate disks. Use the SQLUtil MOVELOG command to move log files to different disks.

If you cannot afford the time to reissue transactions in case a media failure corrupts your log file, use dual logging. When you use dual logging, keep in mind that disk space use is doubled and performance may be affected because the number of I/O operations is also doubled.

You can specify two archive log files or two nonarchive files, but you cannot specify one of each with dual logging.

Archive Logging

Once you have configured a DBEnvironment, you can convert to archive logging in one of the following ways:

- If you have TurboSTORE software, do a complete online backup using the SQLUtil STOREONLINE command. You use STOREONLINE after you have loaded all the database tables and are ready to start using the DBEnvironment in production.
- If you do not have TurboSTORE, stop the DBEnvironment, then connect to the DBEnvironment in single user mode and use the following SQL statements to initiate archive logging:

```
isql=> BEGIN ARCHIVE
isql=> COMMIT ARCHIVE
```

Exit from ISQL, then immediately use the SQLUtil STORE command to create a backup of the DBEnvironment.

After you enable archive logging, you should add additional log files to permit log switching, log backup, and reuse of logs. The next section shows how to add log files; for complete information about managing logs, refer to the “Backup and Recovery” chapter.

Note Once the DBEnvironment is running with archive logging, you must use the START DBE NEWLOG statement to return to nonarchive logging.

Multiple Log Files

Use the SQLUtil ADDLOG command to add additional log files to the DBEnvironment for either nonarchive or archive logging. The following example adds a second log file with 350 pages to the sample DBEnvironment *NewDBE*:

```
>> addlog
DBEnvironment Name: NewDBE
Maintenance Word: 
Enter Log File Name(s) Separated by a Blank? NewLg2
New Log File Size? 350
Add Log File (y/n)? y

Log file 'NewLg2' was Added.
Log Identifier Is: 2
```

Note ADDLOG adds a single log file at a time. In the case of dual logging, two physical log files are added. You *cannot* add more than one file in single logging mode. In dual logging mode, you *cannot* add more than two or less than two files.

Specifying a Native Language Parameter

You can specify a native language parameter in creating a DBEnvironment. Use the `LANG = LanguageName` option in the `START DBE NEW` statement to specify a native language other than `NATIVE-3000`, as in the following example:

```
START DBE 'SOMEDBE' NEW LANG = JAPANESE
```

If you want to specify the name of the DBEnvironment in a native language, then the native language you specify in the `LANG =` clause must be covered by the same character set as the current language. In other cases, your current language can be different from that of the DBEnvironment. All processing—including comparisons and sorting—will take place in accordance with the language of the DBEnvironment, but prompts and messages will appear in the current language if the appropriate message catalog is available. Also, scanning of user input will be in the current language. See “Native Language Support” in Chapter 1 for information about specifying a native language as the current language.

Looking at the DBEnvironment Elements

You can look at each of the elements created by the `START DBE NEW` statement.

Examining MPE Files

Use the MPE `LISTF` command to list all the files in the group and account where your DBEnvironment resides. The DBEFiles, log files, and DBECon file appear as privileged files:

```
: LISTF,2

ACCOUNT= SOMEACCT    GROUP= SOMEGRP

FILENAME CODE  -----LOGICAL RECORD-----  ----SPACE----
          SIZE  TYP          EOF          LIMIT R/B  SECTORS  #X MX
PARTSDBE PRIV  125W  FB           1           1  1         2  1  1
PARTSFO  PRIV  2048W FB          150         150  1        2416  5  5
PARTSLG1 PRIV  256W  FB          256         256  1         514  2  2
PARTSLG2 PRIV  256W  FB          256         256  1         514  2  2
```

Examining DBECon Parameters

Run SQLUtil from MPE/iX or from ISQL to look at the startup parameters in the DBECon file, as in the following example:

```

isql=> sqlutil; Return

                                MON, JAN 6, 1992, 11:11 AM
HP36216-02A.F0.08             DBE Utility/3000             ALLBASE/SQL
(C)COPYRIGHT HEWLETT-PACKARD CO. 1982,1983,1984,1985,1986,1987,1988,
1989,1990,1991,1992. ALL RIGHTS RESERVED.

>> SHOWDBE Return
DBEnvironment Name: PartsDBE Return
Maintenance Word: Return
Output File Name (opt): Return

-> all Return
Maintenance word:
DBEnvironment Language: NATIVE-3000
DBECreator ID: 2204
AutoStart: ON
User Mode: MULTI
DBEFileO Name: PARTSFO
DDL Enabled: YES
No. of Runtime Control Block Pages: 37
No. of Data Buffer Pages: 100
Data Buffer Pages Memory Resident: NO
No. of Log Buffer Pages: 24
Max. Transactions: 5
Maximum Timeout: NONE
Default Timeout: MAXIMUM
Authorize Once per session: OFF

->

```

The parameters are displayed as they appear in the DBECon file illustration in Table 4-1. For more information on SQLUtil, refer to the “DBA Tasks and Tools” chapter and the “SQLUtil” appendix.

Examining the System Catalog

The system catalog is a set of tables and views owned by special users CATALOG and SYSTEM that describe the contents of a DBEnvironment. You must be connected to a DBEnvironment and have SELECT authority or DBA authority in order to query the SYSTEM views. Users without DBA authority can examine the CATALOG views to see information about the objects they own. As DBA, you can also grant or revoke SELECT authority on SYSTEM views. You can query the system catalog views to look at the initial DBEFileSet, DBEFile, and system views created when a DBEnvironment is configured. You can also monitor space requirements, user access, and performance, and generally keep track of what is in the DBEnvironment. Some of the information contained in the system catalog can also be examined with SQLMON, an online monitoring tool. SQLMON is described in the *ALLBASE/SQL Performance and Monitoring Guidelines*.

A simple SELECT statement shows you all the system catalog views:

```
select name, owner, type, rtype, numc from system.table where owner = 'SYSTEM';
```

NAME	OWNER	TYPE	RTYPE	NUMC
ACCOUNT	SYSTEM	0	3	6
CALL	SYSTEM	0	3	5
CHECKDEF	SYSTEM	1	0	6
COLAUTH	SYSTEM	1	0	7
COLDEFAULT	SYSTEM	1	0	6
COLUMN	SYSTEM	1	0	13
CONSTRAINT	SYSTEM	1	0	8
CONSTRAINTCOL	SYSTEM	1	0	4
CONSTRAINTINDEX	SYSTEM	1	0	11
COUNTER	SYSTEM	0	3	3
DBEFILE	SYSTEM	1	0	10
DBEFILESET	SYSTEM	1	0	6
GROUP	SYSTEM	1	0	4
HASH	SYSTEM	1	0	11
INDEX	SYSTEM	1	0	11
MODAUTH	SYSTEM	1	0	3
PARAMDEFAULT	SYSTEM	1	0	6
PARAMETER	SYSTEM	1	0	12
PLAN	SYSTEM	0	3	7
PROCAUTH	SYSTEM	1	0	3
PROCEDURE	SYSTEM	1	0	5
PROCEDUREDEF	SYSTEM	1	0	6
RULE	SYSTEM	1	0	10
RULECOLUMN	SYSTEM	1	0	3
RULEDEF	SYSTEM	1	0	6
SECTION	SYSTEM	1	0	8
SPECAUTH	SYSTEM	1	0	4
TABAUTH	SYSTEM	1	0	14
TABLE	SYSTEM	1	0	15
TEMPSPACE	SYSTEM	1	0	4
TRANSACTION	SYSTEM	0	3	4
USER	SYSTEM	0	3	2
VIEWDEF	SYSTEM	1	0	6

Number of rows selected is 33

U[p], d[own], l[eft], r[ight], t[op], b[ottom], pr[int] <n>,or e[nd]>

To look at the DBEFile0 DBEFile created at configuration time, query the SYSTEM.DBEFILE view using the following statement:

```
isql=> SELECT * FROM System.DBEFile;
SELECT * FROM System.DBEFile;
```

DBEFILENAME	DBEFTYPE	FILEID
PARTSDBE0	90	PARTSFO

Number of rows selected is 1

U[p], d[own], l[eft], r[ight], t[op], b[ottom], pr[int] <n>,or e[nd]>

Note that the DBEFile name stored in the system catalog is the name given in the DBEFILE DBEFILE0 clause, and the FILE ID is the MPE/iX file name given in the NAME clause of the START DBE NEW statement.

Refer to the “System Catalog” chapter for a complete description of each view in the system catalog.

Examining Log File Characteristics

Use the SQLUtil SHOWLOG command to display the characteristics of a newly configured log. For example:

```
>> SHOWLOG
DBEnvironment Name: PartsDBE.SomeGrp.SomeAcct
Maintenance Word: 
Connect (y/n) (opt): y

Archive Mode: OFF
Log Sequence Number Containing Most Recent Archive Checkpoint: 0
Current Log Sequence Number: 1
First Log Sequence Number Needed for Recovery: 0
Log Mode is: Dual
Number of Free Block(s): 340

First Log Name: PartsLG1.SomeGrp.SomeAcct
First Log File Status: Useable
First Log Name: PartsLG2.SomeGrp.SomeAcct
First Log File Status: Useable
Log File Size: 250
Log Identifier Is: 1
Log Sequence Number: 1
Log Backup Status: Backup Is Not Required
```

For an explanation of each parameter, refer to the description of the SHOWLOG command in the “SQLUtil” appendix. See the “Backup and Recovery” chapter for a complete explanation of how logging operates in ALLBASE/SQL.

Creating Audit DBEnvironments

Audit functionality is a group of statements and statement parameters that allows you to generate audit log records. Audit log records contain partition information that allows you to group log records for analysis with the Audit Tool. Some types of database operations you might analyze are INSERT, UPDATE, or DELETE operations, perhaps for security reasons.

Audit log records contain identifiers such as table names in contrast to non-audit database log records which contain identifiers such as page references and data. When audit logging is enabled, these audit log records are generated in addition to non-audit database log records. You can use the Audit Tool, described in the section “Using the Audit Tool,” to audit these log records.

Audit DBEnvironments are defined by specifying audit parameters in the START DBE NEW or START DBE NEWLOG statement.

The six parameters used to make a DBEnvironment an audit DBEnvironment are listed below. One of the six, the AUDIT LOG parameter, causes the other five audit parameters to be in effect. None of the five parameters is in effect unless you specify AUDIT LOG.

AUDIT LOG	causes the audit parameters listed below to be in effect.
AUDIT NAME	identifies the DBEnvironment in each transaction.
DEFAULT PARTITION	identifies the default partition number for the DBEnvironment.
COMMENT PARTITION	identifies the comment partition number for the DBEnvironment.
MAXPARTITIONS	specifies the maximum number of partitions in an audit DBEnvironment.
AUDIT ELEMENTS	consists of the following elements:
COMMENT	allows use of the LOG COMMENT statement.
DATA	generates audit log records for user data write operations (INSERT, DELETE, and UPDATE).
DEFINITION	generates audit log records for statements that define data.
STORAGE	generates audit log records for file and storage statements.
AUTHORIZATION	generates audit log records for authorization statements.
SECTION	generates audit log records for the creation and deletion of permanent sections.
ALL	specifies generation of audit log records for all audit elements.

Audit elements are prioritized in a simple hierarchy where the following assumptions exist:

1. DATA is assumed to be specified if AUDIT LOG is in effect. In other words, if AUDIT LOG is specified in a START DBE NEW or START DBE NEWLOG statement, without specifying any audit elements, only DATA is in effect.
2. Audit elements can be explicitly specified as shown below:

```
DATA STORAGE SECTION AUDIT ELEMENTS;
```

3. Specifying ALL assumes that all audit elements are requested.

See the syntax for the START DBE NEW and START DBE NEWLOG statements in the “SQL Statements” chapter of the *ALLBASE/SQL Reference Manual* for information on how to specify audit DBEnvironment parameters.

Example of Setting Up an Audit DBEnvironment

The following examples show how to create a DBEnvironment, load it, and then enable audit logging.

First, create the DBEnvironment with a temporary log named TempLog:

```
START DBE 'DBE1' MULTI NEW
BUFFER = (240, 120),
TRANSACTION = 50,
DBEfile0 DBEFILE MyDBE1
WITH PAGES = 300,
NAME = 'MyDBE1',
LOG DBEFILE TempLog
WITH PAGES = 5000,
NAME = 'TempLog';

CREATE TABLE MyTable1
.
.
.

CREATE TABLE MyTable2
.
.
.

LOAD FROM INTERNAL LdFile1 TO MyTable1;
COMMIT WORK;
LOAD FROM INTERNAL LdFile2 TO MyTable2;
COMMIT WORK;

.
.
.

COMMIT WORK RELEASE;
```

Now you can use `START DBE NEWLOG` to enable audit logging and audit files:

```
START DBE 'DBE1' MULTI NEWLOG
AUDIT LOG,
AUDIT NAME = 'MyDBE1',
DEFAULT PARTITION = 1,
MAXPARTITIONS = 10,
ALL AUDIT ELEMENTS
LOG DBEFILE MyLog1
  WITH PAGES = 5000,
  NAME = 'MyLog1';

EXIT;
```

Now use `SQLUtil` to create the additional log file that is needed for audit DBEnvironments:

```
isql=> sqlutil
>> addlog
DBEnvironment Name: DBE1
Maintenance Word:
Enter Log File Name(s) Separated by a Blank? MyLog2
New Log File Size? 5000
Add Log File (y/n)? y

Log file 'MyLog2' Was Added.
Log Identifier Is: 2

>> exit
```

Log files need to be made slightly larger to account for audit log records generated in addition to non-audit log records. Audit log records are generated for all the statement types specified in the `AUDIT ELEMENTS` parameter, so log files may fill up more quickly with audit logging specified.

Defining Additional Audit DBEnvironment Log Files

Audit DBEnvironments require that at least one additional log file be added. This is performed with the `SQLUtil ADDLOG` command. It is recommended that *several* additional log files be added because log files will fill up more quickly.

When `START DBE NEWLOG` is executed for an existing audit DBEnvironment, most audit-related parameters not specified remain unchanged. The `AUDIT LOG` parameter is an exception. If `AUDIT LOG` is in effect and you execute a `START DBE NEWLOG` statement to change parameter values *without* again specifying `AUDIT LOG`, audit logging is then *not* in effect.

Disabling Audit Logging

You can disable audit logging for a particular *session* where you are entering statements that should not generate audit log records. This allows all other sessions to continue to generate audit log records. The following statement is used to disable audit logging for a session:

```
DISABLE AUDIT LOGGING
```

Audit logging should be enabled again before the session is ended. The following statement is used to enable audit logging:

```
ENABLE AUDIT LOGGING
```

However, since disabling only lasts for the duration of a session, when the session ends, audit logging is enabled even if you do not explicitly enable it again.

4-12 DBEnvironment Configuration and Security

Creating DBEFileSets and DBEFiles

As discussed in the “Physical Design” chapter, DBEFileSets and DBEFiles are the key elements of ALLBASE/SQL storage. A DBEFile is an MPE file that contains table data, or index data, or both. A DBEFileSet is a logical group of one or more DBEFiles. The amount of storage available in a DBEFileSet can be increased by adding DBEFiles.

When a DBEnvironment is configured, ALLBASE/SQL creates the SYSTEM DBEFileSet containing DBEFile0. DBEFile0 contains all the table, view, and index data for the system catalog, as well as all stored sections for preprocessed programs. If the SYSTEM DBEFileSet runs out of space, create another DBEFile and add it to the SYSTEM DBEFileSet before you create anything else. If you never create any new DBEFileSets and DBEFiles, all tables and indexes created in the DBEnvironment are placed in the SYSTEM DBEFileSet. As discussed in the “Physical Design” chapter, separate DBEFileSets should be created for groups of tables that are maintained together.

Creating DBEFileSets

You must have DBA authority to create a DBEFileSet. Use the following syntax:

```
CREATE DBEFILESET DBEFileSetName
```

The name you specify is stored in the system catalog, as shown in the view SYSTEM.DBEFILESET. Two DBEFileSets in the same DBEnvironment cannot have the same name.

A DBEFileSet is a logical construct. There is no physical space associated with it until you create DBEFiles and add them to the DBEFileSet.

The statements to create the three DBEFileSets for the PurchDB database are as follows:

```
isql=> CREATE DBEFILESET PurchFS;  
isql=> CREATE DBEFILESET WarehFS;  
isql=> CREATE DBEFILESET OrderFS;
```

The DBEFileSet names are stored as character strings in SYSTEM.DBEFILESET. Names are all upshifted unless you enclose them in double quotes.

To look at the DBEFileSets that were created, examine the Static DBEFile screen in SQLMON, or query the system catalog as follows:

```
isql=> SELECT DBEFSNAME, DBEFSNDBEFILES,  
> DBEFSNPAGES, DBEFSUPAGES FROM System.DBEFileSet;
```

```
-----+-----+-----+-----+  
DBEFSNAME      |DBEFSNDBEFILES|DBEFSNPAGES|DBEFSUPAGES|  
-----+-----+-----+-----+  
SYSTEM         |              |        150|          0|  
PURCHFS        |              |        100|          2|  
WAREHFS        |              |        100|          2|  
ORDERFS        |              |        100|          2|  
RECFS          |              |         50|          3|  
-----+-----+-----+-----+
```

```
Number of rows selected is 5  
U[p], d[own], l[eft], r[ight], t[op], b[ottom], pr[int] <n>, or e[nd]>
```

DBEFileSet names follow the rules of ALLBASE/SQL basic names; refer to the SQL Reference Manual for ALLBASE/SQL naming conventions.

Assigning Default DBEFileSets

ALLBASE/SQL system catalog tables are stored in the SYSTEM DBEFileSet. When a new DBEnvironment is created (with the START DBE NEW statement), the SYSTEM DBEFileSet is the default DBEFileSet for storing all user tables and sections. When user data is stored in the SYSTEM DBEFileSet, system performance is impacted.

One way of insuring that user sections and data are stored in a DBEFileSet other than SYSTEM is to assign a default DBEFileSet. Then, when tables and long column data are created without being assigned a specific DBEFileSet, they are stored in the default TABLESPACE DBEFileSet you have assigned. And when sections are created without being assigned a specific DBEFileSet, they are stored in the default SECTIONSPACE DBEFileSet. For example:

```
SET DEFAULT SECTIONSPACE TO DBEFILESET SectionDBESet FOR PUBLIC;
```

```
SET DEFAULT TABLESPACE TO DBEFILESET TableDBESet FOR PUBLIC;
```

Complete syntax for the SET DEFAULT DBEFILESET statement is found in the *ALLBASE/SQL Reference Manual*.

Creating DBEFiles

You must have DBA authority to create DBEFiles. Use the following syntax:

```
CREATE DBEFILE DBEFilename WITH PAGES = DBEFileSize, NAME = 'SystemFileName'  
[ , INCREMENT = DBEFileIncrSize [ , MAXPAGES = DBEFileMaxSize ] ]  
[ , TYPE = {  
    TABLE  
    INDEX  
    MIXED  
} ]
```

All DBEFiles must be created in the group and account where the DBECon file resides. DBEFileName is the name by which a DBEFile is known to ALLBASE/SQL. It is stored in the system catalog, as shown in the view SYSTEM.DBEFILE. No two DBEFiles can have the same DBEFileName. DBEFileNames follow the rules of ALLBASE/SQL basic names. DBEFileNames are stored as character strings; they are upshifted unless you enclose them in double quotes.

PAGES is the number of 4096-byte pages that will be preallocated to the DBEFile. A number from 2 to 524,287 must be specified. The number of pages you assign to the DBEFile depends on how much data you plan to store in it. Use your calculations from “Physical Design” to determine how many pages you should assign to a DBEFile. Remember that a DBEFile is an MPE file, so its size is limited by the system resources available.

SystemFileName is the name of the DBEFile as it appears in MPE/iX. It must follow MPE/iX naming conventions. If you use the *LISTF* command to view the files on your system, you can see the system file name specified in the CREATE DBEFILE statement for all DBEFiles you have created.

Type TABLE allows only table data; type INDEX allows only index data. The default type is MIXED, which allows both table and index data. The sample database DBEFiles are created as either TABLE or INDEX so that the tables and indexes can be manipulated independently of each other. As discussed in the “Physical Design” chapter, locating index data and table data on separate devices decreases the work load for the disk drives and increases performance.

The following examples show how to create two DBEFiles for the PurchFS DBEFileSet:

```
isql=> CREATE DBEFILE PurchDataF1
> WITH PAGES = 50,
> NAME = 'PurchDF1',
> TYPE = TABLE;

isql=> CREATE DBEFILE PurchIndxF1
> WITH PAGES = 50,
> NAME = 'PurchXF1',
> TYPE = INDEX;
```

Similarly, the following examples show how to create DBEFiles for the DBEFileSets WarehFS and OrderFS respectively:

```
isql=> CREATE DBEFILE WarehDataF1
> WITH PAGES = 50,
> NAME = 'WarehDF1',
> TYPE = TABLE;

isql=> CREATE DBEFILE WarehIndxF1
> WITH PAGES = 50,
> NAME = 'WarehXF1',
> TYPE = INDEX;

isql=> CREATE DBEFILE OrderDataF1
> WITH PAGES = 50,
> NAME = 'OrderDF1',
> TYPE = TABLE;

isql=> CREATE DBEFILE OrderIndxF1
> WITH PAGES = 50,
> NAME = 'OrderXF1',
> TYPE = INDEX;
```

The characteristics of the DBEFiles are stored in the system catalog, as shown in the view SYSTEM.DBEFILE. You can perform a SELECT on SYSTEM.DBEFILE to see the DBEFiles you have created along with DBEFile0 created at DBEnvironment configuration time. Note that although you entered the DBEFile names in upper and lower case, they are recorded in the system catalog as all upper case letters unless you enclose them in double quotes.

Adding DBEFiles to DBEFileSets

Adding DBEFiles to a DBEFileSet allocates storage space in which table and index data can be stored. Although you can create tables in an empty DBEFileSet, you cannot create indexes in a DBEFileSet until a DBEFile has been added to the DBEFileSet. Creating an index causes a root page to be created, whereas creating a table does not cause any physical page to be created.

You must have DBA authority to add DBEFiles to DBEFileSets. Use the following statement syntax:

```
ADD DBEFILE DBEFileName TO DBEFILESET DBEFileSetName
```

Both the DBEFile and the DBEFileSet must already exist in the DBEnvironment.

You can add DBEFiles to the SYSTEM DBEFileSet using the following syntax:

```
ADD DBEFILE DBEFileName TO DBEFILESET SYSTEM
```

To add a file to the SYSTEM DBEFileSet, your transaction must be the only active transaction in the system. If other users are active, your transaction will wait until they complete.

The PurchDB database has three DBEFileSets, each of which contains two DBEFiles. The following series of statements adds the DBEFiles to their respective DBEFileSets and allocates storage for the sample database:

```
isql=> ADD DBEFILE PurchDataF1 TO DBEFILESET PurchFS;
isql=> ADD DBEFILE PurchIndxF1 TO DBEFILESET PurchFS;
isql=> ADD DBEFILE WarehDataF1 TO DBEFILESET WarehFS;
isql=> ADD DBEFILE WarehIndxF1 TO DBEFILESET WarehFS;
isql=> ADD DBEFILE OrderDataF1 TO DBEFILESET OrderFS;
isql=> ADD DBEFILE OrderIndxF1 TO DBEFILESET OrderFS;
```

Now query the SYSTEM.DBEFILE view to see which DBEFiles are associated with which DBEFileSets:

```
isql=> SELECT DBEFName,DBEFSName FROM SYSTEM.DBEFILE;
```

The query result is as follows:

```
SELECT DBEFName,DBEFSName FROM System.DBEFile;
-----+-----
DBEFNAME          |DBEFSNAME
-----+-----
PARTSDBEO         |SYSTEM
PURCHDATAF1       |PURCHFS
PURCHINDEXF1      |PURCHFS
WAREHDATAF1       |WAREHS
WAREHINDEXF1      |WAREHS
ORDERDATAF1       |ORDERFS
ORDERINDEXF1      |ORDERFS
-----+-----
Number of rows selected is 7
U[p], d[own], l[eft], r[ight], t[op], b[ottom], pr[int] <n>,or e[nd]>
```

DBEFiles should be created and added to DBEFileSets as they are needed so that you do not have empty DBEFiles wasting disk space.

Allocating Expandable DBEFile Space

As an alternative to creating fixed-length DBEFiles, you can create DBEFiles which are expandable, by specifying a maximum DBEFile size and an expansion increment. This approach is known as **dynamic DBEFile expansion**, which you can employ by using the optional INCREMENT and MAXPAGES clauses of the CREATE DBEFILE statement. The procedure is as follows:

1. Create expandable DBEFiles of the appropriate type (TABLE, INDEX, or MIXED) using the CREATE DBEFILE statement. Specify an expansion increment and a maximum file size. For example:

```
isql=> CREATE DBEFILE DBEFile1 WITH PAGES=100,
> NAME='DBEFile1', INCREMENT=100, MAXPAGES=5000, TYPE=TABLE;
```

2. Use the ADD DBEFile statement to add each DBEFile to a previously created DBEFileSet.

By using this method, you do not have to add DBEFiles to the DBEFileSet to increase its capacity until the space has expanded to the maximum file size specified. If a DBEFile runs

out of initial capacity during execution of an INSERT or UPDATE statement, an additional increment of space is added to the file. However, when expandable DBEFiles are in use, the DBA should still monitor the size of DBEFiles to determine the following:

- How close the file's size is to the maximum.
- Whether the file has grown unexpectedly large following a temporary condition in the database.

Note If a transaction that causes expansion of a DBEFile were to abort, the DBEFile expansion is rolled back logically and not physically. In this event, the size of the DBEFile as shown in the DBEFNPAGES column of the SYSTEM.DBEFile view can be smaller than the actual file size shown on the operating system. The discrepancy disappears the next time a successful expansion takes place.

Allocating Expandable DBEFile Space in the SYSTEM DBEFileSet

The SYSTEM DBEFileSet contains the system catalog and its indexes, together with space used by temporary tables during query processing. When you create a DBEnvironment using the START DBE NEW statement, ALLBASE/SQL creates a fixed-length file called DBEFILE0 and adds it to the SYSTEM DBEFileSet. You specify the size of the file in the START DBE NEW statement. Because DBEFile0 is of fixed length, it is not expandable. Therefore, in order to provide dynamic space expansion for the system catalog or for sort operations during query processing, you must create one or more additional expandable DBEFiles and add them to the SYSTEM DBEFileSet. You cannot create an expandable DBEFile using the START DBE NEW statement.

Caution An expandable DBEFile cannot be compressed again once expanded. Also a DBEFile cannot easily be removed from the SYSTEM DBEFileSet once added because it could be used by the system catalog. Large sorted queries can easily require large amounts of sort space from the SYSTEM DBEFileSet causing expandable SYSTEM DBEFiles to expand to their maximum size. These DBEFiles cannot be reduced or dropped once expanded. Therefore, care should be taken to limit the maximum size of expandable SYSTEM DBEFiles.

Partial DBEFile Expansion

Partial expansion of files can occur if the following conditions exist:

- The current size of a DBEFile is not a multiple of *DBEFileIncrSize*. In this case, the number of pages by which the DBEFile is expanded is determined as follows:
 - If the number of pages needed to bring the size of the DBEFile to a multiple of its increment is less than or equal to half the increment size, the DBEFile is expanded by that many pages plus those specified in the increment size.
 - Otherwise, the DBEFile is expanded by the number of pages needed to make its size a multiple of its increment.
- If the file system gets full during expansion, the DBEFile remains partially expanded.

Obtaining Information about Expandable DBEFiles

The following query displays information about expandable DBEFiles as it appears in the SYSTEM.DBEFile view:

```
isql=> SELECT DBEFNAME, DBEFTYPE, FILEID,  
> DBEFNPAGES, DBEFUPAGES, DBEFINCRSZ, DBEFIPAGES,  
> DBEFMPAGES, CTIME, DBEFSNAME  
> FROM SYSTEM.DBEFILE;
```

Creating the DBEnvironment Security Scheme

After configuring the DBEnvironment, you must create the security scheme that will control its use. To do this, you grant and revoke authorities for specific users and groups with the GRANT and REVOKE statements, respectively, as described in the *ALLBASE/SQL Reference Manual*. You can also control the authority to perform certain maintenance tasks using the SQLUtil maintenance word, which is a password for SQLUtil. For more information on the maintenance word, refer to the “SQLUtil” appendix.

Creating Authorization Groups

You can grant all authorities to each DBEUserID individually. However, if several users require the same set of authorities, you can use an authorization group. First, create the groups, then add specific users to them. For example, if all managers need the same authorities, you can create a group called PurchManagers and add the DBEUserIDs of the managers to it, as follows:

```
isql=> CREATE GROUP PurchManagers;  
isql=> ADD Margy@Ryan TO GROUP PurchManagers;  
isql=> ADD Ron@Hart TO GROUP PurchManagers;  
isql=> ADD Sharon@Muldoon TO GROUP PurchManagers;
```

Managing Authorization Groups

Authorization group management consists of adding members to the group and removing them when appropriate. An authorization group is owned and managed by the owner of the group and/or by a user with DBA authority.

To delegate group management to another user, do one of the following:

- assign the user as the owner in the CREATE GROUP statement (requires DBA authority)
- create the group and transfer ownership to the user (requires DBA or OWNER authority)

Owners of groups are able to indirectly grant and revoke authorities by adding and removing members. An example is given in “Granting/Revoking CONNECT Authority” later in this chapter.

The SQL statements used to manage authorization groups are shown in Figure 4-2.

```

CREATE GROUP [ Owner. ] GroupName

ADD { DBEUserID
      GroupName
      ClassName } [ , ... ] TO GROUP TargetGroupName

REMOVE { DBEUserID
          GroupName
          ClassName } [ , ... ] FROM GROUP TargetGroupName

DROP GROUP GroupName

TRANSFER OWNERSHIP OF { [ TABLE ] [ Owner. ] TableName
                        [ VIEW ] [ Owner. ] ViewName
                        GROUP GroupName } TO NewOwnerName

```

Figure 4-2. SQL Commands for Authorization Group Management

For more information about these statements, refer to the *ALLBASE/SQL Reference Manual*.

Using DBA Authority

DBA authority is for database administrators and users who create, maintain, and control access to the DBEnvironment.

DBA authority is granted to (and cannot be revoked from) the user who configures the DBEnvironment (DBECreator). A user with DBA authority can execute any SQL statement or ISQL command in the DBEnvironment. DBA authority also gives the user co-ownership of all objects in the DBEnvironment. A user with DBA authority can even revoke DBA authority from any other user except the DBECreator.

ALLBASE/SQL does not restrict the number of users that can have DBA authority, but because DBA authority is so powerful and therefore potentially dangerous, you should be selective in granting it to others. You need DBA authority to grant DBA, RESOURCE, CONNECT, RUN and EXECUTE, and DBEFileSet authority to other users, as in the following examples:

```

isql=> GRANT DBA TO John@Brock;
isql=> GRANT CONNECT TO John@Brock;
isql=> GRANT RESOURCE TO John@Brock;

```

The DBECreator

The DBECreator is the user who configures a DBEnvironment with the START DBE NEW statement.

The DBECreator is able to execute all of the SQLUtil commands and is given irrevocable DBA authority for the DBEnvironment. The DBECreator's DBEUserID appears in the DBECon file and is entered into the system catalog, as shown in the SYSTEM.SPECAUTH view.

There can be only one DBECreator for each DBEnvironment. DBA authority cannot be revoked from the DBECreator. The DBECreator cannot be added to an authorization group.

The DBECreator has special capabilities that general users do not have. The DBECreator can:

- execute the `START DBE NEWLOG` statement.
- execute the `SQLUtil` commands (except `PURGEFILE`).

DBA Functions

Specific functions requiring DBA authority, and their appropriate SQL statements, are listed in the appendix “Authorities Required by ALLBASE/SQL Statements.”

Granting Authorities to PUBLIC

PUBLIC is a special, nonrestrictive category of user. By granting RUN authority or table and view authorities to PUBLIC, you implicitly grant that authority to any user who has CONNECT authority to the DBEnvironment.

When you explicitly grant an authority to PUBLIC, you are granting that authority to all users in the DBEnvironment. Granting authorities to PUBLIC on an object is not the same as creating a table PUBLIC. Although you implicitly perform a `GRANT ALL TO PUBLIC` when you create a table PUBLIC, you are also specifying the locking strategy for the table. You can revoke the authorities on a PUBLIC table, but the locking strategy remains unchanged.

Granting/Revoking CONNECT Authority

Users cannot access the DBEnvironment until they are explicitly granted CONNECT authority. A user with CONNECT authority has all table and view authorities that may have been granted to the special user PUBLIC; however, granting CONNECT authority does not grant a user any other privilege. To grant CONNECT authority, use the following statement:

```
isql=> GRANT CONNECT TO Peter@Crane;
```

You can revoke CONNECT authority at any time. A user that is currently connected to the DBEnvironment is allowed to continue their DBE session after CONNECT authority is revoked, but cannot reconnect once the session is terminated. Once CONNECT authority is revoked, the user cannot access the DBEnvironment regardless of any other authorities (except DBA authority) previously granted.

If CONNECT authority is revoked from a user who owns objects, these objects, as well as any other authorities the user may have been granted, are unaffected by the revocation. To revoke CONNECT authority, use the following statement:

```
isql=> REVOKE CONNECT FROM Peter@Crane;
```

Since granting and revoking CONNECT authority does not affect a module's ownership or other user authorities, DBEnvironment access can be restricted without restructuring the security scheme. DBA authority is required to execute the `GRANT` and `REVOKE` statements for special authorities, including `CONNECT`.

You can centralize access control to the DBEnvironment using authorization groups. In the sample DBEnvironment the group Purch has CONNECT authority and all other groups

and users are members of Purch. The DBA can revoke CONNECT authority from Purch to temporarily keep all users out of the DBEnvironment for maintenance and backup purposes. Note that if the group being added as a member of another group does not exist, no error results, since the entry is assumed to be a class name and not an authorization group. Therefore, assure that a group is created before it is granted any authority or added to another authorization group. The following series of statements creates the sample database CONNECT authority scheme:

```
isql=> CREATE GROUP Purch;
isql=> ADD PurchManagers TO GROUP Purch;
isql=> ADD PurchDBMaint TO GROUP Purch;
isql=> ADD Purchasing TO GROUP Purch;
isql=> ADD Receiving TO GROUP Purch;
isql=> ADD WareHouse TO GROUP Purch;
isql=> ADD AccountsPayable TO GROUP Purch;
isql=> ADD Tom@Ash TO GROUP Purch;
isql=> GRANT CONNECT TO Purch;
```

Only a user with DBA authority can grant or revoke CONNECT authority directly to or from an individual user or an authorization group. However, you can delegate the ability to indirectly grant CONNECT authority to a user without granting DBA authority to that user by transferring ownership of a group with CONNECT authority to that user.

```
isql=> TRANSFER OWNERSHIP OF GROUP Purch TO Ron@Hart;
```

Ron owns this authorization group and can control access to the DBEnvironment by adding members to or removing members from Purch. You can get the same results by granting CONNECT authority to an authorization group already owned by a non-DBA user.

Note Remember to COMMIT WORK to make your changes permanent.

Granting/Revoking RESOURCE Authority

RESOURCE authority gives a user the ability to create:

- tables
- authorization groups

When you grant RESOURCE authority to an individual user, you are, in effect, giving the user the capability to create a database. Any resources created by the user are owned by that user's DBEUserID and are treated as a separate logical database.

```
isql=> GRANT RESOURCE TO Annie@Melchoir;
```

Suppose Annie@Melchoir, a member of the PurchDBMaint group, creates a new table called Employees. The table's fully qualified name is Annie@Melchoir.Employees, and belongs to the Annie@Melchoir database rather than the PurchDB database.

If RESOURCE authority is revoked from Annie@Melchoir, she still has OWNER authority for those objects she created, but she cannot create any more objects:

```
isql=> REVOKE RESOURCE FROM Annie@Melchoir;
```

In the sample DBEnvironment the group PurchDBMaint has RESOURCE authority. The members of PurchDBMaint are DBA assistants that create tables and groups. Once the

design of the objects is approved, the DBA can transfer their ownership so they become part of an existing database. The following statements establish the PurchDBMaint group:

```
isql=> CREATE GROUP PurchDBMaint;
isql=> GRANT RESOURCE TO PurchDBMaint;
isql=> GRANT ALL ON PurchDB.Parts TO PurchDBMaint;
isql=> GRANT ALL ON PurchDB.Inventory TO PurchDBMaint;
isql=> GRANT ALL ON PurchDB.SupplyPrice TO PurchDBMaint;
isql=> GRANT ALL ON PurchDB.Vendors TO PurchDBMaint;
isql=> GRANT ALL ON PurchDB.Orders TO PurchDBMaint;
isql=> GRANT ALL ON PurchDB.OrderItems TO PurchDBMaint;
isql=> GRANT SELECT ON PurchDB.VendorStatistics TO PurchDBMaint;
isql=> GRANT SELECT ON PurchDB.PartInfo TO PurchDBMaint;
```

Users are then added to the PurchDBMaint group:

```
isql=> ADD Annie@Melchoir TO GROUP PurchDBMaint;
isql=> ADD Doug@Dolan TO GROUP PurchDBMaint;
isql=> ADD David@Bloom TO GROUP PurchDBMaint;
```

You can remove a user from the PurchDBMaint group to remove the user's associated RESOURCE authority:

```
isql=> REMOVE David@Bloom FROM GROUP PurchDBMaint;
```

To remove a user's OWNER authority, you can transfer ownership of the object, then grant the required table and view authorities to the user.

Granting/Revoking RUN Authority

RUN authority permits a user or group to execute an already preprocessed program that the user or group does not own. RUN authority can be granted by the DBA or the owner of the module.

```
isql=> GRANT RUN ON PurchDB.Program TO John@Brock;
```

RUN authority can also be revoked by the DBA or the owner of the module:

```
isql=> REVOKE RUN ON PurchDB.Program FROM John@Brock;
```

RUN authority can be granted to a group. For more information on module authorities, refer to the "Maintenance" chapter.

Granting/Revoking EXECUTE Authority

EXECUTE authority permits a user or group to execute a section stored in the DBEnvironment that the user does not own. EXECUTE authority can be granted by the DBA or the owner of the procedure.

```
isql=> GRANT EXECUTE ON PurchDB.ReportMonitor TO John@Brock;
```

EXECUTE authority can also be revoked by the DBA or the owner of the procedure:

```
isql=> REVOKE RUN ON PurchDB.ReportMonitor FROM John@Brock;
```

EXECUTE authority can be granted to a group. For more information on procedure authorities, refer to the "Maintenance" chapter.

Granting/Revoking SECTIONSPACE or TABLESPACE Authority

SECTIONSPACE authority permits the grantee to store sections in the specified DBEFileSet, as in the following grant to PUBLIC:

```
isql=> GRANT SECTIONSPACE ON DBEFILESET DBEFileSet1 TO PUBLIC;
```

TABLESPACE authority permits the grantee to store table and long column data in the specified DBEFileSet, as in the following grant to the Warehse group:

```
isql=> GRANT TABLESPACE ON DBEFILESET DBEFileSet2 TO Warehse;
```

Refer to complete syntax for the GRANT statement in the *ALLBASE/SQL Reference Manual*.

Verification of Authority

ALLBASE/SQL checks the DBECreator name in the DBECon file to make sure you are the DBECreator whenever you execute the START DBE NEWLOG statement and certain SQLUtil commands. In addition, ALLBASE/SQL checks your DBEUserID in the system catalog authorization tables for:

- DBA authority when you perform a START DBE
- CONNECT authority when you enter a CONNECT statement
- DBA or the appropriate authority when you:
 - specify an SQL statement that accesses the database (refer to the appendix, “Authorities Required by SQL Statements,” for a listing of SQL statements and their respective authorities)
 - run an application program
 - preprocess an application program

Interactively, ALLBASE/SQL checks authority each time you issue a statement. Programmatically, ALLBASE/SQL checks authority of the embedded SQL statements during preprocessing and for DBA, OWNER, or RUN authority when a user attempts to run the application program.

Managing DBEnvironment Sessions

This section shows the different ways to start and terminate a DBEnvironment session for daily operations and gives guidelines on when to use each method discussed.

A **DBE session** is a period of time between establishing and terminating access to a DBEnvironment by a user or a program. Each user or program has a unique session. When you start a DBEnvironment with the START DBE statement, you also start a DBE session. Once a DBEnvironment is started, provided that it is in multiuser mode, all users must use the CONNECT statement to start a DBE session. A user must be in a DBE session to execute any statements except START DBE or CONNECT.

The DBEnvironment is not stopped until the STOP DBE statement is executed or until the last DBE session terminates. The STOP DBE, RELEASE, and TERMINATE USER statements are different ways to terminate DBE sessions.

The START DBE statement must not be confused with the other two START DBE statements available with ALLBASE/SQL. START DBE NEW, covered at the beginning of this chapter, is used to configure a DBEnvironment and is performed one time only for a DBEnvironment. The other START DBE statement, START DBE NEWLOG, is used only to create a new log file. This statement is described in the “Backup And Recovery” chapter of this guide.

Using Autostart

Autostart automates the DBEnvironment startup procedure. When autostart is ON, the DBEnvironment can be started with either a START DBE or a CONNECT statement. When START DBE is used, the user mode must be specified as MULTI or the default of SINGLE is used. All other parameters not specified are taken from the DBECon file. When CONNECT is used, all startup parameters, including user mode, are taken from the DBECon file. ALLBASE/SQL executes a START DBE statement on behalf of the first CONNECT statement. All subsequent CONNECT statements are treated as conventional CONNECT statements.

Figure 4-3 shows the relationship between autostart and user mode.

		AUTOSTART	
		ON	OFF
USER MODE	MULTI	START DBE with the MULTI option starts the DBE in multiuser mode. If the DBE is already started, use CONNECT.	START DBE with the MULTI option starts the DBE in multiuser mode. If the DBE is already started, use CONNECT.
	SINGLE	CONNECT starts the DBE in multiuser mode. If the DBE is already started, CONNECT will start a DBE session.	CONNECT will fail if the DBE is not started. If the DBE is already started, CONNECT will start a DBE session.
	MULTI	START DBE without the MULTI option starts a single-user DBE session. If the DBE is already started, START DBE will fail.	START DBE without the MULTI option starts a single-user DBE session. If the DBE is already started, START DBE will fail.
	SINGLE	CONNECT starts a single-user session.	CONNECT will fail. START DBE without the MULTI option must be used to start a single-user session.

Figure 4-3. Autostart and User Mode Dependencies

The recommended procedure for automating multiuser access using autostart is as follows:

- Use SQLUtil to modify the startup parameters for your needs.
- Make sure Autostart is ON.
- Make sure User Mode is set to MULTI.

Once the configuration is set up with autostart ON and user mode MULTI, the startup procedure becomes transparent to any user. A user with CONNECT or DBA authority can start the DBEnvironment in multiuser mode with the following statement:

```
isql=> CONNECT TO 'PartsDBE.SomeGrp.SomeAcct';
```

Using START DBE

Using the CONNECT statement with autostart ON is the recommended way to start a DBE session. However, the DBA might want to use START DBE for any one of the following reasons:

- to start a DBE session in single user mode to perform functions such as creating new objects, loading large amounts of data, database restructuring and database maintenance
- to temporarily override the configuration parameters in the DBECon file
- to start a DBEnvironment that does not use autostart
- to perform rollback recovery after a system failure in which the DBEnvironment was not damaged

The START DBE statement starts an existing DBEnvironment and initiates a DBE session. You must have DBA authority to use START DBE. The START DBE statement is only successful if no users are currently accessing the DBEnvironment. Complete syntax to start a DBEnvironment is presented in the *ALLBASE/SQL Reference Manual*.

Starting a DBE Session in Single-User Mode

To start a DBE session in single user mode simply issue the START DBE statement without specifying a user mode. The user mode defaults to SINGLE, and all other parameters not specified default to the values in the DBECon file. The following statement starts a single user session in the sample DBEnvironment:

```
isql=> START DBE 'PartsDBE.SomeGrp.SomeAcct';
```

Overriding DBECon Parameters

You can use the START DBE statement to temporarily override the values in the DBECon file when the default options are not sufficient. For example, a database may undergo several updates on a certain day, requiring a greater than usual number of log and data buffer pages. The following statement will increase the number of buffer pages so that performance will not be impaired:

```
isql=> START DBE 'PartsDBE.SomeGrp.SomeAcct' MULTI  
> BUFFER = (300,45);
```

The data buffer pages and log buffer pages are increased to 300 and 45, respectively. Note that the TRANSACTION option was not specified in the START DBE statement. The default value in the DBECon file is used. Once the STOP DBE statement is issued, the page numbers return to the default values in the DBECon file. To permanently change the number of buffer pages or other startup parameters, use the ALTDBE command in SQLUtil. A DBA who needs more control over how users access the DBEnvironment might want to leave autostart OFF and start the DBEnvironment every time users need access.

Starting DBE Sessions without Autostart

If a DBEnvironment does not use autostart, the DBA must perform a START DBE before additional users can connect to the DBEnvironment. The following statement simply starts the sample DBEnvironment in multiuser mode:

```
isql=> START DBE 'PartsDBE.SomeGrp.SomeAcct' MULTI;
```

Rollback Recovery with START DBE

ALLBASE/SQL ensures the DBEnvironment is consistent when it is opened by the START DBE statement by automatically doing the following:

- permanently reapplying all changes made by transactions that were terminated by a COMMIT WORK statement but were not written to disk prior to the termination or failure
- rolling back (undoing) any changes made by incomplete or aborted transactions

The first CONNECT statement executed with autostart ON executes an implicit START DBE which performs a rollback for any incomplete transactions.

Connecting to a DBE

The CONNECT statement only works when one of the following is true:

- The DBEnvironment has been started using the START DBE statement with the MULTI option.
- Autostart is ON.

You must have CONNECT or DBA authority in a DBEnvironment before you can connect to it.

Once the DBEnvironment has been started with the MULTI option, all subsequent users must use the CONNECT statement to start a DBE session.

Terminating a DBE Session

There are three statements available that terminate a DBE session:

```
RELEASE  
STOP DBE  
TERMINATE USER
```

Using RELEASE

Any user that can connect to the DBEnvironment can execute the RELEASE statement. To terminate a DBE session in a single user or multiuser DBEnvironment, simply type:

```
isql=> RELEASE;
```

Before releasing the current DBE session, either COMMIT WORK to make changes permanent, or ROLLBACK WORK to undo the transaction. All transactions that were not committed are aborted, and their changes are rolled back when the DBEnvironment is started again.

ALLBASE/SQL keeps the DBEnvironment open until the last session terminates. When the last session is terminated, ALLBASE/SQL closes the DBEnvironment by performing a STOP DBE on the user's behalf.

Type either **END** or **EXIT** at the ISQL prompt to terminate ISQL. If you try to exit ISQL before entering the **RELEASE** statement, ISQL asks if you want to **COMMIT WORK**. Type **Y** or **YES** to keep all changes made in the current transaction. Type **N** or **NO** to roll back the current transaction.

Using **STOP DBE**

The **STOP DBE** statement is used to close a DBEnvironment. DBA authority is required to execute this statement. The **STOP DBE** statement stops all DBE sessions, whether in a single or multiuser DBEnvironment. The statement is:

```
isql=> STOP DBE;
```

Transactions that have not been committed before the DBEnvironment is stopped are aborted. Their changes are rolled back when a **START DBE** statement is executed. The **STOP DBE** statement can be used to stop the DBEnvironment before making a backup, recovering a DBEnvironment, performing maintenance, or changing DBECon parameters. The DBA may wish to query the **SYSTEM.USER** table to see which users are currently accessing the DBEnvironment, and advise those users to use the **COMMIT WORK** statement with the **RELEASE** option to make all changes permanent before the DBA issues the **STOP DBE** statement.

To exit ISQL type **E** or **EXIT**. If you try to exit ISQL before entering the **STOP DBE** statement, ISQL will ask if you want to **COMMIT WORK**. Type **Y** or **YES** to keep all changes made in the current transaction. A **YES** response will *not* keep changes made in other DBE sessions. Type **N** or **NO** to abort all current transactions.

Using **TERMINATE USER**

The **TERMINATE USER** statement allows you to terminate your own or another user's DBE session. You can always terminate your own session with the **TERMINATE USER** statement. You must have DBA authority to terminate another user's session.

A DBA might use the **TERMINATE USER** statement to terminate a DBE session that is using excessive system resources and causing deadlocks, or a session that is impeded. The statement to abort all active sessions for DBEUserID Kelly@Cota is:

```
isql=> TERMINATE USER Kelly@Cota;
```

You can also use the **TERMINATE USER** statement to terminate one of several **ALLBASE/SQL** sessions running under the same DBEUserID. Specify a session ID instead of a DBEUserID in the statement:

```
isql=> TERMINATE USER 108;
```

where 108 is the session identifier as shown in the **SYSTEM.USER** view or the process identifier displayed on the **SQLMON Overview Session** screen. If a single application (or ISQL session) has established more than one connection to the same DBEnvironment, all connections have the same session ID, and all are terminated.

If a user's session is impeded or waiting, **TERMINATE USER** aborts the session immediately. However, if a user's session is active and not waiting, **TERMINATE USER** does not take effect until the session makes its next call to **ALLBASE/SQL**.

You can monitor the **SYSTEM.USER** view in the system catalog to determine the users currently connected to a DBEnvironment and the session identifiers for each user. Refer to the "System Catalog" chapter for more information. To identify current DBEnvironment users,

you can also monitor the Overview Session screen in SQLMON. The session identifier column in the SYSTEM.USER view is equivalent to the process identifier field in SQLMON. For more information, see the *ALLBASE/SQL Performance and Monitoring Guidelines*.

Terminating Transactions and Queries

Use the TERMINATE TRANSACTION statement to terminate a given transaction. Use the TERMINATE QUERY statement to terminate a running query. Also the TERMINATE AT QUERY LEVEL or the TERMINATE AT TRANSACTION LEVEL option must have been set for the specified connection or transaction. See the “SQL Statements” chapter in the *ALLBASE/SQL Reference Manual* for more information.

Setting Timeout Values

You can set default and maximum timeout values for a DBEnvironment with the SQLUtil ALTDDBE command. These values apply to both lock waits and throttle waits. In addition, individual users can set timeout values for individual user sessions.

The timeout action can also be set to abort the command being processed instead of the entire transaction for a particular session or transaction. Both SET SESSION and SET TRANSACTION have parameters to specify which action the system should take when a timeout expires. The setting of timeout values is also incorporated into these commands. The SQLUtil SHOWDBE command displays the current, default, and maximum values of the timeout parameter in the DBECon file.

Remote Database Access

You can access ALLBASE/SQL database files on remote nodes by using ALLBASE/NET and a local area network. The local machine is known as the client node, and the remote machine, where the DBEnvironment is located, is known as the server node. Refer to the *ALLBASE/NET User's Guide* for complete details.

Database Creation and Security

Once the DBEnvironment is configured and storage is allocated, you can begin creating the tables, views, indexes, and other objects that comprise databases. The examples in this chapter show the statements for creating specific tables, views, and authorities within the sample DBEnvironment PartsDBE. A complete set of statements for creating all the tables, views, and authorities in PartsDBE is found in the “Sample DBEnvironment” appendix of the *ALLBASE/SQL Reference Manual*.

Tasks presented in this chapter are:

- Creating tables
- Creating views
- Creating hash structures
- Creating indexes
- Creating procedures and rules
- Creating the database security scheme
- Creating a database schema
- Loading tables

Creating Tables

You can create a table using the following basic syntax:

```

CREATE [ PRIVATE
       PUBLICREAD
       PUBLIC
       PUBLICROW ] TABLE [ Owner. ] TableName
[ LANG = TableLanguageName ]
( { ColumnDefinition
  { UniqueConstraint
  { ReferentialConstraint
  { CheckConstraint } [ , ... ] )
[ UNIQUE HASH ON ( HashColumnName [ , ... ] ) PAGES = PrimaryPages ]
[ HASH ON CONSTRAINT [ ConstraintID ] PAGES = PrimaryPages ]
[ CLUSTERING ON CONSTRAINT [ ConstraintID ] ]
[ IN PARTITION { PartitionName
                { DEFAULT
                { NONE } } ]
[ IN DBEFileSetName ]

```

Refer to the CREATE TABLE statement in the *ALLBASE/SQL Reference Manual* for complete syntax and semantics.

Table Type

The first parameter in the CREATE TABLE statement specifies the table type. If you do not specify a table type, the default is PRIVATE. The table type option creates a table with the following implied attributes:

- The initial security level of the table:
 - PRIVATE (the default type) - gives no authorities to PUBLIC.
 - PUBLICREAD - causes ALLBASE/SQL to perform an implicit GRANT SELECT TO PUBLIC. This gives any user with CONNECT authority the authority to look at the table.
 - PUBLIC and PUBLICROW - causes ALLBASE/SQL to perform an implicit GRANT ALL TO PUBLIC. This gives any user with CONNECT authority the authority to look at and modify the table as well as alter the table and create indexes on it.
- The locking mode of the table:
 - PRIVATE (the default type) - causes ALLBASE/SQL to hold exclusive (X) locks at the table level for both reads and writes. These locks are easy for ALLBASE/SQL to manage and unlikely to cause a deadlock condition because each table is always accessed exclusively by one user. Tables that must not be accessed by more than one user at a time should be PRIVATE.
 - PUBLICREAD - causes ALLBASE/SQL to hold share (S) locks at the table level for reads and exclusive (X) locks at the table level for writes. A table created PUBLICREAD can be read by several users, which increases concurrency, but can only be modified by one transaction at a time, which increases data consistency. Tables that are rarely updated should be PUBLICREAD.
 - PUBLIC - causes ALLBASE/SQL to hold share (S) locks at the page level for reads (SELECT) and hold exclusive (X) locks at the page level for writes (INSERT, UPDATE, DELETE). When locks are obtained at the page level for reads, an intention share (IS) lock is obtained on the table, and an S lock is obtained on the page. When locks are obtained at the page level for reads with an intention to update or write a row, an intention exclusive (IX) lock is obtained on the table, and a share and intention exclusive (SIX) lock is obtained on the page. If a page is actually written to, the SIX lock must become an X lock.

PUBLIC mode provides higher concurrency than PUBLICREAD and PRIVATE tables for both reads and writes because a user does not have to wait for a locked table to get released. Moderate to large size tables for which you want to maximize concurrency should be PUBLIC.

- PUBLICROW - causes ALLBASE/SQL to hold share (S) locks at the row level for reads and exclusive (X) locks at the row level for writes. When locks are obtained at the row level for reads, an intention share (IS) lock is obtained on the table and on the page, and an S lock is obtained on the row. When locks are obtained at the row level for reads with an intention to update or write a row, an intention exclusive (IX) lock is obtained on the table and on the page, and a share and intention exclusive (SIX) lock is obtained on the row. If a row is actually updated, the SIX lock must become an X lock.

PUBLICROW provides the greatest concurrency for both reads and writes because a user does not have to wait for a lock on a page or table to get released. A side effect of PUBLICROW mode is the large number of locks that must be obtained when accessing

data from moderate or large size tables. Locks are obtained at table, page, and row levels for PUBLICROW tables under normal circumstances. PUBLICROW mode is often the best choice for a small table that you expect to be accessed by a large number of concurrent transactions.

ALLBASE/SQL automatically uses the locking mode implicit in the table type whenever you access that table. You can use the LOCK TABLE statement to temporarily override this automatic locking behavior. With LOCK TABLE, you can increase the granularity of locking from page to table level or from row to table level. However, you cannot decrease the granularity of locking from table to page level or from page to row level by using LOCK TABLE.

Revoking and Granting Authorities on PUBLICROW and PUBLIC Tables

The REVOKE and GRANT statements may be used by the DBA or the table's owner to change the automatic grant implied at creation time; however, the locking mode remains unchanged.

You can get PUBLIC, PUBLICROW, or PUBLICREAD locking on a table without the security implications by creating the table PUBLIC, PUBLICROW, or PUBLICREAD and revoking the implied authority from PUBLIC:

```
isql=> CREATE PUBLIC TABLE SomeTable (SomeColumn ... );
isql=> REVOKE ALL ON SomeTable FROM PUBLIC;
```

Likewise, you can have PRIVATE locking on a table and grant authorization to PUBLIC:

```
isql=> CREATE PRIVATE TABLE SomeTable (SomeColumn ... );
isql=> GRANT SELECT ON SomeTable TO PUBLIC;
```

Table locking strategy is discussed in more detail in the chapter “Concurrency Control Through Locks and Isolation Levels” in the *ALLBASE/SQL Reference Manual*.

Altering Table Type

You can alter a table's type (lock mode) by using the ALTER TABLE statement. The following example shows how to change locking mode to PUBLICROW without changing the authority scheme for the table:

```
isql=> ALTER TABLE PurchDB.Vendors
> SET TYPE PUBLICROW PRESERVE AUTHORITY;
isql=>
```

Owner and Table Name

The next parameter in the CREATE TABLE statement, OwnerName, defaults to the DBEUserID of the user who creates the table. DBA authority is required to create tables with an owner name other than the default. You can set the default owner's name to something other than the creator's DBEUserID by using the ISQL SET OWNER command. Refer to the “Logical Design” chapter for information on the different types of owners (class names, DBEUserIDs, or authorization groups) and OWNER authority.

A table name may be up to 20 bytes long in any combination of letters, digits, \$, #, @, or underscore. The first character, however, cannot be a decimal digit or an underscore. A table name can contain any characters if it is enclosed in double quotes. However, the name must then be enclosed in double quotes each time it is referenced, and the first character following the first double quote cannot be a blank.

Specify the owner name any time you reference a table that is not your own.

Use the `LANG = TableLanguageName` clause in the `CREATE TABLE` statement to specify a language other than the DBEnvironment's language. For example, you might wish to specify `NATIVE-3000` (ASCII) for a certain table although the DBEnvironment language is Japanese.

```
isql=> CREATE TABLE NewTable LANG = "NATIVE-3000"
> (Column1 char(20), Column2 char(10));
```

You must use double quotes around the name "NATIVE-3000" because it contains a hyphen. Normally, native language names do not require quotes.

Column Definition

$$\begin{array}{l}
 \text{ColumnName} \left\{ \begin{array}{l} \text{ColumnDataType} \\ \text{LongColumnType} \text{ [IN DBEFileSetName2]} \end{array} \right\} \\
 \text{[LANG = ColumnLanguageName]} \\
 \text{[[NOT] CASE SENSITIVE]} \\
 \left[\begin{array}{l} \text{DEFAULT} \left\{ \begin{array}{l} \text{Constant} \\ \text{USER} \\ \text{NULL} \\ \text{CurrentFunction} \end{array} \right\} \end{array} \right] \\
 \left[\begin{array}{l} \text{NOT NULL} \left\{ \begin{array}{l} \text{UNIQUE} \\ \text{PRIMARY KEY} \end{array} \right\} \text{[CONSTRAINT ConstraintID]} \\ \text{REFERENCES RefTableName [(RefColumnName)] [CONSTRAINT ConstraintID]} \\ \text{[...]} \\ \text{CHECK (SearchCondition) [CONSTRAINT ConstraintID]} \\ \text{[IN DBEFileSetName3]} \end{array} \right] \text{ [...]}
 \end{array}$$

The column definition includes the following parts, of which only the column name and data type are required:

- Column name
- Data type
- Language clause
- DEFAULT clause
- Constraint definitions

Column Name and Data Type

A table must have at least one column and each column must be given a name and a data type. In addition, the `NOT NULL` attribute, which disallows null values from being entered in the column, can be assigned. Several columns in the sample DBEnvironment `PartsDBE`, including `PartNumber`, `VendorNumber` and `OrderNumber`, are defined as `NOT NULL` and consequently are required to contain data.

Language Clause

Use the `LANG = ColumnLanguageName` clause in the `CREATE TABLE` statement to specify a column with a language different from that of the default table language. You can only specify `NATIVE-3000` (ASCII) or the current native language. Example:

```
isql=> CREATE TABLE NewTable
> (Column1 char(20) LANG = "NATIVE-3000",
> Column2 char(20));
```

DEFAULT Clause

Use the `DEFAULT` clause to specify a default value for a column. Example:

```
isql=> CREATE TABLE Table5
> (Column1 char(20) DEFAULT 'Empty',
> Column2 integer NOT NULL);
```

For further information on data types in creating tables, refer to the chapters “Names” and “Data Types” in the *ALLBASE/SQL Reference Manual*.

Constraint Definitions

In creating a table definition, you can include the following types of integrity constraints:

- Unique constraints
- Referential constraints
- Check constraints

Unique Constraints

Use the `UNIQUE` clause to specify a unique constraint on a table. Use the `PRIMARY KEY` clause to specify a unique constraint that also defines the primary key for a table. Example:

```
CREATE PUBLIC TABLE RecDB.Clubs
(ClubName CHAR(15) NOT NULL PRIMARY KEY,
ClubPhone SMALLINT,
Activity CHAR(16))
IN RecFS;
```

Note that the primary key must be on a column that is `NOT NULL`.

The difference between a `PRIMARY KEY` and a `UNIQUE` constraint is that the `PRIMARY KEY` designation lets you reference the key in a referenced table without specifying column names.

Referential Constraints

The `REFERENCES` clause lets you specify the manner in which a referencing table points to a unique or primary key of another table. You use the `REFERENCES` clause within a column definition to define a referential constraint in which only that column references a key in another table.

Example:

```
CREATE PUBLIC TABLE RecDB.Members
(MemberName CHAR(20) NOT NULL,
Club CHAR(15) NOT NULL REFERENCES RecDB.Clubs (ClubName),
MemberPhone SMALLINT) IN RecFS;
```

This assumes that RecDB.Clubs has already been created with ClubName as a unique or primary key.

You use REFERENCES along with the FOREIGN KEY clause to define a referential constraint on multiple columns at the table level. In order to illustrate this, it is necessary to show an alternate way of creating the RecDB.Members table, in which MemberName and Club are defined as a two-column primary key:

```
CREATE PUBLIC TABLE RecDB.Members
(MemberName CHAR(20) NOT NULL,
Club CHAR(15) NOT NULL,
MemberPhone SMALLINT,
PRIMARY KEY (MemberName, Club),
FOREIGN KEY (Club) REFERENCES RecDB.Clubs (ClubName)) IN RecFS;
```

Based on this referenced table, we can define the RecDB.Events table as follows with a two-column foreign key:

```
CREATE PUBLIC TABLE RecDB.Events
(SponsorClub CHAR(15),
Event CHAR(30),
Date DATE,
Time TIME,
Coordinator CHAR(20),
FOREIGN KEY (Coordinator, SponsorClub)
REFERENCES RecDB.Members (MemberName, Club)) IN RecFS;
```

Note that since (MemberName, Club) is specified as the PRIMARY KEY for RecDB.Members, the use of the column names in the REFERENCES clause of the example is optional.

Check Constraints

The following example shows a table created with a check constraint. The check constraint ensures that the Date column will not be updated with a date earlier than January 1, 1990.

```
CREATE PUBLIC TABLE RecDB.Events
(SponsorClub CHAR(15),
Event CHAR(30),
Date DATE,
Time TIME,
Coordinator CHAR(20),
CHECK (Date >= '1990-01-01') )
IN RecFS
```

For more information about integrity constraints, see the chapter “Constraints, Procedures, and Rules” in the *ALLBASE/SQL Reference Manual*.

DBEFileSet Name

The last parameter in the CREATE TABLE statement specifies the DBEFileSet with which the table and its indexes are associated. If you do not specify a DBEFileSet and you have not assigned a default DBEFileSet with the SET DEFAULT DBEFILESET statement, tables are created in the SYSTEM DBEFileSet, which also contains the system catalog. It is recommended practice to keep the system catalog apart from your data and index files. Refer to the “Physical Design” chapter under “Grouping Tables in DBEFileSets” for information on why you might wish to create separate DBEFileSets for tables.

In addition, two parameters in the column definition syntax allow specification of a DBEFileSet for long column data and for check constraint sections.

Examining Table Attributes

After creating tables, you can query the SYSTEM.TABLE and SYSTEM.COLUMN views to see how their definitions appear. The following query on SYSTEM.TABLE will display information about all the tables in the PurchDB database:

```
isql=> SELECT * FROM System.Table WHERE Owner='PURCHDB';
SELECT * FROM System.Table WHERE Owner='PURCHDB';
```

NAME	OWNER	DBFILESSET	TYPE	RT
INVENTORY	PURCHDB	WAREHFS		0
ORDERITEMS	PURCHDB	ORDERFS		0
ORDERS	PURCHDB	ORDERFS		0
PARTINFO	PURCHDB	SYSTEM		1
PARTS	PURCHDB	WAREHFS		0
REPORTS	PURCHDB	ORDERFS		0
SUPPLYPRICE	PURCHDB	PURCHFS		0
VENDORS	PURCHDB	PURCHFS		0
VENDORSTATISTICS	PURCHDB	SYSTEM		1

```
-----
Number of rows selected is 9
U[p], d[own], l[eft], r[ight], t[op], b[ottom], pr[int] <n>, or e[nd]>
```

You can also use the Static subsystem of SQLMON to see which tables are contained in a DBEFileset. For more information on SQLMON, see the *ALLBASE/SQL Performance and Monitoring Guidelines*.

You must use an additional query of the system catalog to display the column definitions of particular tables:

```
isql=> select * from system.column
> where owner= 'PURCHDB' and tablename = 'PARTS';
```

The query result is as follows:

```
select * from system.column where owner= 'PURCHDB' and tablename = 'PARTS';
```

COLNAME	TABlename	OWNER	COLNUM	
PARTNUMBER	PARTS	PURCHDB		1
PARTNAME	PARTS	PURCHDB		2
SALESPRICE	PARTS	PURCHDB		3

```
-----
Number of rows selected is 3
U[p], d[own], l[eft], r[ight], t[op], b[ottom], pr[int] <n>, or e[nd] >
```

You can also use the ISQL INFO command to display column definitions for a table:

```
isql=> INFO PURCHDB.PARTS;
```

Column Name	Data Type (length)	Nulls Allowed	Language
PARTNUMBER	Char (16)	NO	NATIVE-3000
PARTNAME	Char (30)	YES	NATIVE-3000
SALESPRICE	Decimal (10, 2)	YES	

The CREATE TABLE statement only enters the table definition into the system catalog. The table does not occupy storage in the DBEFileSet until you insert a row.

All names that are stored as character strings in the system catalog are upshifted when they are stored unless they are in double quotes. For example, the statement

```
isql=> CREATE PUBLIC TABLE PurchDB.SomeTable
> (Column1 INTEGER, Column2 INTEGER) in FS;
```

would store PURCHDB as the owner and SOMETABLE as the table name in the system catalog, whereas the statement

```
isql=> CREATE PUBLIC TABLE "PurchDB"."SomeTable"
> (Column1 INTEGER, Column2 INTEGER) in FS;
```

would store PurchDB as the owner and SomeTable as the table name. To examine the attributes of the first, you would use the following statement:

```
isql=> SELECT * FROM SYSTEM.TABLE
> WHERE NAME = 'SOMETABLE' AND OWNER = 'PURCHDB';
```

For the second, you would use lower case spelling:

```
isql=> SELECT * FROM SYSTEM.TABLE
> WHERE NAME = 'SomeTable' AND OWNER = 'PurchDB';
```

To eliminate any possible confusion, avoid using double quotes in defining objects.

Defining Partitions and Tables

To create partitions, you use the CREATE PARTITION statement. Then, to assign a table to a partition, you can use the IN PARTITION parameter of the CREATE TABLE statement or the SET PARTITION parameter of the ALTER TABLE statement. If you do not assign a table to a partition, the table is assigned to the DEFAULT partition.

If you do not want audit logging done on the table, you can specify NONE with either IN PARTITION or SET PARTITION; then, operations on the table do not generate audit log records.

If you assign a table to a partition, and if you specified DATA AUDIT ELEMENTS when you started the DBEnvironment (either explicitly or by default), then any inserts, updates, or deletes you perform on the table generate audit log records.

Partitions are dropped with the DROP PARTITION statement. Before you can drop a partition, you must assign each of its tables to a new partition with the statement ALTER TABLE SET PARTITION.

Example

In the following example, the IN PARTITION clause of the CREATE TABLE statement is used to assign a table to the already created partition, P1:

```
CREATE PARTITION P1 with ID = 1;

CREATE PUBLIC TABLE table1 (Col1 integer not null,
                           Col2 char(12) not null,
                           Col3 integer not null)
IN PARTITION P1
IN FileSetA;
```

Refer to the CREATE TABLE statement and ALTER TABLE statement syntax in the “SQL Statements” chapter of the *ALLBASE/SQL Reference Manual*.

Creating Views

A view is a table derived by placing a “window” over one or more tables to let users or programs see only certain data. Views are useful for limiting data visibility; they are also useful for pulling together various tables to simplify complex SELECT statements. Refer to “Designing Views” in the “Logical Design” chapter of this guide.

A view is actually a SELECT statement that is permanently stored as a section in the system catalog. You can store a SELECT statement that extracts part of a table, or, as in the sample database, you can store a SELECT that joins two or more tables together. Refer to the “Maintenance” chapter for further explanation of sections.

The following is the basic syntax for creating a view:

```
CREATE VIEW [ Owner. ] ViewName [ ( ColumnName [ , ... ] ) ]  
AS QueryExpression [ IN DBEFileSetName ]  
[ WITH CHECK OPTION [ CONSTRAINT ConstraintID ] ]
```

Refer to the *ALLBASE/SQL Reference Manual* for complete syntax and semantics.

Owner names for views are established the same way they are for tables.

View names must conform to the same conventions as table names. The view name is stored in both SYSTEM.TABLE and SYSTEM.SECTION. The view definition is stored in SYSTEM.VIEWDEF.

Constraint information is stored in the SYSTEM.CONSTRAINT table.

The section containing the SELECT statement for the view is stored in one of three places:

- the DBEFileSet you specify in the CREATE VIEW statement
- the default section space DBEFileSet you specify with SET DEFAULT DBEFILESET
- the system DBEFileSet

If you omit the column names, they are derived from the column names of the table entered as part of the query expression, which is a SELECT that defines the contents of the view.

The CHECK OPTION clause defines a check constraint that enforces the condition defining the view when changes are made through the view. When the table from which the view is derived is modified through the view, this check constraint must be satisfied. The view must also meet the requirements for updatability.

Note The query expression that defines a view cannot include an ORDER BY clause. If you need to use ORDER BY, use it in the SELECT on the view.

The following statement creates a view that allows authorized users to see the relationship between the PartNumber column, the VendPartNumber, VendorName, and ListPrice:

```
isql=>CREATE VIEW PurchDB.PartInfo
> (PartNumber,PartName,VendorNumber,VendorName,
> VendorPartNumber,ListPrice,Quantity) AS
> SELECT PurchDB.SupplyPrice.PartNumber,
> PurchDB.Parts.PartName,
> PurchDB.SupplyPrice.VendorNumber,
> PurchDB.Vendors.VendorName,
> PurchDB.Supplyprice.VendPartNumber,
> PurchDB.SupplyPrice.UnitPrice,
> PurchDB.SupplyPrice.DiscountQty
> FROM PurchDB.Parts,
> PurchDB.SupplyPrice,
> PurchDB.Vendors
> WHERE PurchDB.SupplyPrice.PartNumber =
> PurchDB.Parts.PartNumber
> AND PurchDB.SupplyPrice.VendorNumber =
> PurchDB.Vendors.VendorNumber;
```

Note that the SELECT statement joins three tables. Views can facilitate query processing and can also be used to limit data visibility.

A check constraint is defined on an updatable view, as follows:

```
CREATE VIEW RecDB.EventView
    (Event,
    Date) AS
SELECT RecDB.Event,
    RecDB.Date
    FROM RecDB.Events
    WHERE Date >= '1992-01-01'
WITH CHECK OPTION
```

See the section “Controlling Table Access with Views,” below.

Creating Hash Structures

You create a hash structure at the same time you create a table, using the CREATE TABLE statement, as in the following syntax.

```
CREATE [ PUBLIC
      [ PUBLICREAD
      PRIVATE ] ] TABLE [ Owner. ] TableName [ LANG = TableLanguageName ]
( { ColumnName ColumnDataType [ LANG = ColumnLanguageName ] [ NOT NULL ] }
[ , ... ] ) [ UNIQUE HASH ON ( HashColumnName [ , ... ] ) PAGES = PrimaryPages ]
[ IN DBEFileSetName ]
```

Refer to the *ALLBASE/SQL Reference Manual* for complete syntax and semantics.

The following sequence of statements shows how to create the PurchDB.Vendors table using a hash structure. Commands to create and add a dedicated DBEFile to contain the hash structure are also shown, since there must be sufficiently large empty DBEFiles of type TABLE or MIXED available when you issue the CREATE TABLE statement in creating a hash structure:

```

isql=> BEGIN WORK;

isql=> CREATE DBEFile PurchHashF1
> WITH PAGES = 120, NAME = 'PurchHF1',
> TYPE = TABLE;

isql=> ADD DBEFile PurchHashF1
> TO DBEFILESET PurchFS;

isql=> CREATE PUBLIC TABLE PurchDB.Vendors
> (VendorNumber      INTEGER          NOT NULL,
> VendorName        CHAR(30)         NOT NULL,
> ContactName       CHAR(30),
> PhoneNumber       CHAR(15),
> VendorStreet      CHAR(30)         NOT NULL,
> VendorCity        CHAR(20)         NOT NULL,
> VendorState       CHAR(2)          NOT NULL,
> VendorZipCode     CHAR(10)         NOT NULL,
> VendorRemarks    VARCHAR(60) )
> UNIQUE HASH ON (VendorNumber) PAGES = 101
> IN PurchFS;

isql=> COMMIT WORK;

```

Note that all these statements are shown as part of the same transaction. This ensures that the DBEFiles you create will not be used by any other transaction before the hash structure is created.

Creating Indexes

ALLBASE/SQL uses the data in the columns as a key to facilitate data retrieval. Use the CREATE INDEX statement as in the following syntax:

```

CREATE [UNIQUE] [CLUSTERING] INDEX [ Owner. ] IndexName ON

[ Owner. ] TableName ( ColumnName [ ASC ] [ DESC ] [ , ColumnName [ ASC ] [ DESC ] ] [ ... ] )

```

The index name must conform to ALLBASE/SQL naming conventions explained in the *ALLBASE/SQL Reference Manual*. If you do not specify an owner name, then the index name must be unique throughout the DBEnvironment. Refer to the *ALLBASE/SQL Reference Manual* for complete syntax and semantics.

There are several indexes in the sample database. The following examples show how to create each type. The first is a nonunique, nonclustering index:

```

isql=> CREATE INDEX VendPartIndex
> ON PurchDB.SupplyPrice (VendorNumber);

```

The next example shows the creation of a unique index:

```

isql=> CREATE UNIQUE INDEX PartNumIndex
> ON PurchDB.Parts (PartNumber);

```

The following shows the creation of a clustering index:

```

isql=> CREATE CLUSTERING INDEX PartToNumIndex
> ON PurchDB.SupplyPrice (PartNumber);

```

For details on how ALLBASE/SQL uses each type of index, refer to the “Logical Design” and “Maintenance” chapters.

Note

Before you build a large index, it is advisable to use the CREATE TEMPSPACE statement to designate an area where the system can open temporary files that are large enough for the sorting that is required. By default, the current group is used as a tempspace. If there is not enough space in the current group, the CREATE INDEX statement may fail for large indexes, unless alternate tempspaces are designated. Refer to “Physical Design” chapter for more information about creating tempspaces. Also refer to the “CREATE TEMPSPACE” statement in the *ALLBASE/SQL Reference Manual*.

Creating Procedures and Rules

Rules and procedures work together to provide a method of enforcing the relationships in a database design without application programming. You create procedures, which are stored in the DBEnvironment; then you create rules that invoke the procedures when certain conditions are met. This section shows how to create simple rules and procedures. For more detailed information, refer to the chapter “Constraints, Procedures and Rules” in the *ALLBASE/SQL Reference Manual*.

Creating a Procedure

You can create procedures that perform most database operations when fired by a rule or when invoked in ISQL or in an application program. The following example shows how to create procedure PurchDB.RemovePart, which is invoked by the rule described in the following section:

```
CREATE PROCEDURE PurchDB.RemovePart (PartNum CHAR(16) NOT NULL)
AS BEGIN
    DELETE FROM PurchDB.Inventory WHERE PartNumber = :PartNum;
    DELETE FROM PurchDB.SupplyPrice WHERE PartNumber = :PartNum;
END;
```

For more detailed information, refer to the “Using Procedures” section of the “Constraints, Procedures, and Rules” chapter in the *ALLBASE/SQL Reference Manual* and to the “Using Procedures in Application Programs” chapter in the *ALLBASE/SQL Advanced Application Programming Guide*.

Creating a Rule

You can define rules that operate on specific tables in a database whenever a particular type of data manipulation is performed. The following example shows how to create a rule tied to an update of the PurchDB.Parts table:

```
CREATE RULE PurchDB.RemovePart
AFTER DELETE FROM PurchDB.Parts
WHERE SUBSTRING(PartNumber,1,4) < > 'XXXX'
EXECUTE PROCEDURE PurchDB.RemovePart (OLD.PartNumber);
```


The table on which the rule is defined is PurchDB.Parts. The statement type required to trigger the procedure is the DELETE operation. The condition that must be satisfied in addition to the statement type of DELETE is that the first four characters in PartNumber must not be “XXXX.” The procedure to be executed is PurchDB.RemovePart.

Creating the Database Security Scheme

In addition to the security provided by the operating system and the security inherent in special authorities, which you grant at the DBEnvironment level, you can create a security scheme for individual databases by granting and revoking authorities or by creating view definitions. The next sections describe how to create a database security scheme.

Controlling Table Access with Authorities

You create a security scheme to protect the tables in each database by granting specific authorities to individual DBEUserIDs or groups and revoking them from other DBEUserIDs or groups. Table 5-1 describes the types of table and view authorities.

Table 5-1. Table and View Authorities

Type	Description
ALTER	Lets a user or group add columns to a table. ALTER authority does not apply to views.
DELETE	Lets a user or group delete rows from a table or view. DELETE authority can be granted on any view, but rows cannot be deleted unless the view meets certain criteria. See the section “Designing Views” in the “Logical Design” chapter.
INDEX	Lets a user or group create and drop indexes for a table. INDEX authority does not apply to views.
INSERT	Lets a user or group insert rows into a table or view. INSERT authority can be granted on any view, but rows cannot be inserted unless the view meets certain criteria. See the section “Designing Views” in the “Logical Design” chapter.
SELECT	Lets a user or group look at a table or view and create views on a table or view.
UPDATE	Lets a user or group update all columns in the table or view. UPDATE authority on specific columns lets a user or group update only those columns. UPDATE authority can be granted on any view, but columns cannot be updated unless the view meets certain criteria. See the section “Designing Views” in the “Logical Design” chapter.
REFERENCES	Lets a user or group define referential constraints on all columns in the table. REFERENCES authority on specific columns lets a user or group define referential constraints on only those columns. REFERENCES authority cannot be granted on views.

The owner of a table or view has all table and view authorities for that table or view.

Table and view authorities cannot be revoked from the owner of a table or view; however, ownership can be transferred. When ownership is transferred from a user, that user no longer has any authorities for the table or view unless they are explicitly granted again. You can grant all table and view authorities to a user with the ALL option of the GRANT statement:

```
isql=> GRANT ALL ON PurchDB.Parts TO Peter@Crane;
```

Authorities for Single Users

The following examples grant and revoke table and view authorities for a single DBEUserID:

```
isql=> GRANT SELECT ON PurchDB.Inventory TO Tom@Ash;  
isql=> REVOKE SELECT ON PurchDB.Inventory FROM Tom@Ash;
```

Authorities for Groups

You can also assign authorities to groups. Create a group as in the following example:

```
isql=> CREATE GROUP Purchasing;
```

Next, add members to the group, as follows:

```
isql=> ADD AJ@BROWN TO GROUP Purchasing;
```

Finally, assign the appropriate table authorities to the group:

```
isql=> GRANT SELECT, INSERT, DELETE, UPDATE  
> ON PurchDB.OrderItems TO Purchasing;
```

Note that the creator of a group does not receive the authorities that are assigned to the group by other users. A user with authorization through group membership cannot issue the WITH GRANT OPTION clause of the GRANT statement.

Creating Classes

If you want to create objects that do not have users or groups as owners (and are therefore controlled solely by the DBA), use class ownership. Choose an appropriate class name, using the guidelines in the “Logical Design” chapter, then do one of the following to create the class:

- create a table or view with the class name as the owner name
- preprocess an application with the class name as owner name
- transfer ownership of an object to the class name

For example, the sample DBEnvironment contains several tables owned by the class PurchDB. The table PurchDB.Parts was created with the following statement:

```
isql=> CREATE TABLE PurchDB.Parts  
(PartNumber CHAR(16) NOT NULL,  
PartName CHAR(30),  
SalesPrice DECIMAL(10,2))  
IN WarehFS;
```

To create a module belonging to a class, specify the class name as the owner in parenthesis in the preprocessor command line:

```
: RUN PSQLPAS.PUB.SYS;INFO="(PartsDBE OWNER(PurchDB))"
```

After creating objects owned by the class, you must grant the specific authorities you wish users or groups to have. Suppose there is a group PurStaff, consisting of DBEUserIDs for members of the Purchasing Department. You could grant authorities to the group with the following statements:

```
isql=> GRANT SELECT, UPDATE ON PurchDB.Parts TO PurStaff;  
isql=> GRANT RUN ON PurchDB.Program TO PurStaff;
```

Revoking Table and View Authorities

When you revoke a user's authority or remove a user from a group, that user can no longer perform the functions allowed by that authority. However, any other authorities granted to that user are not affected. By removing AJ@Brown, who is not a DBA, from the PurStaff group, you do not affect his authorities associated with the Receiving group. Also, you do not affect any authorities that AJ@Brown may have granted, as a member of the group, to other users.

Controlling Table Access with Views

The SELECT, INSERT, and DELETE authorities operate at the row level. The UPDATE authority operates at either the row or column level. To restrict SELECT, INSERT, or DELETE authority to certain columns, or to restrict UPDATE authority to certain columns and rows of a table, you can create a view and grant the required authorities on the view.

Assume you have a table containing rows for several departments, but you only want the manager for a particular department to be able to access data for that department. To accomplish this you create a view with a WHERE clause defining only those rows to be accessed, and grant authorities on the views to the appropriate managers. The statement to create the view in Figure 5-1 is:

```
isql=> CREATE VIEW PurchDB.Dept100  
>AS SELECT * FROM PurchDB.Dept WHERE DeptNo=100;
```

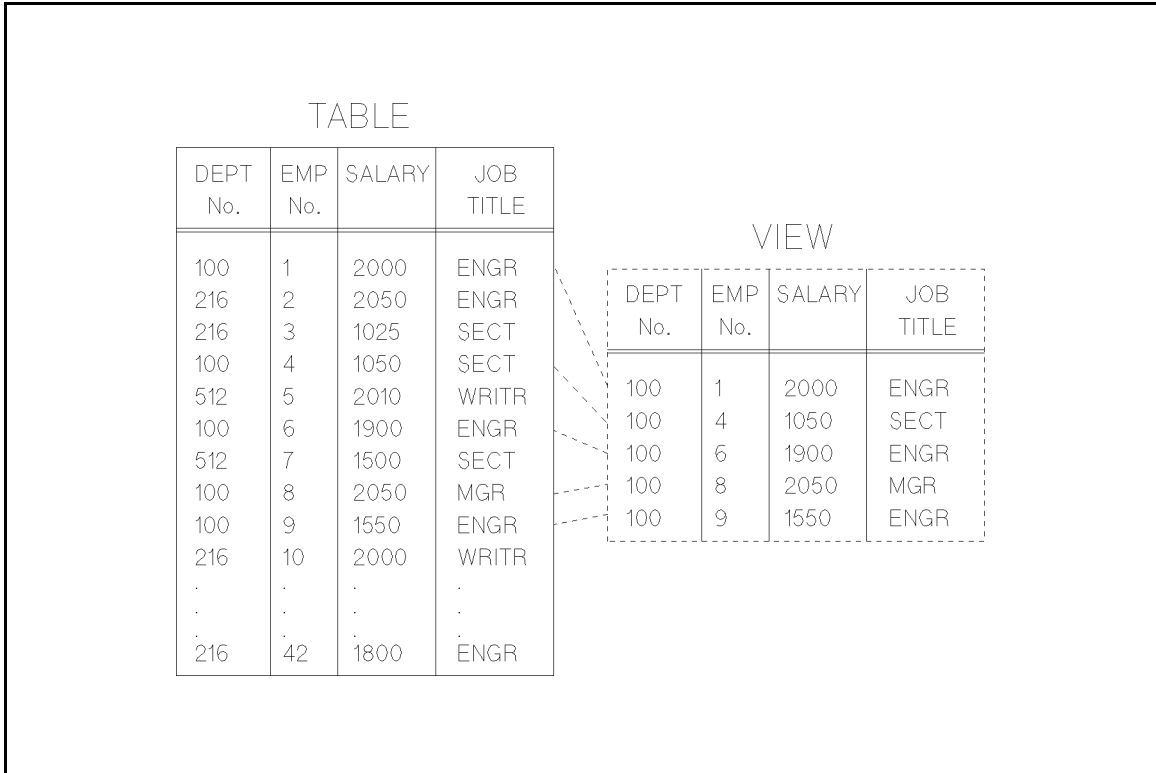


Figure 5-1. Views Restricting Access

The manager for department 100 is Tom@Ash. The statement to grant authorities for department 100 data is:

```
isql=> GRANT SELECT,INSERT,DELETE,UPDATE
> ON PurchDB.Dept100 TO Tom@Ash;
```

The example in Figure 5-1 restricts access to certain rows, but you can also restrict access to specific columns by eliminating sensitive columns from the view definition. For example, you might wish to eliminate the salary column from a view of a department's personnel. Remember, however, that if you want to update the base tables through a view, you must include in the view definition all base table columns that were created with the NOT NULL option. Other restrictions apply to using the INSERT, UPDATE, and DELETE authorities on views. Refer to "Designing Views" in the "Logical Design" chapter of this guide.

ALLBASE/SQL uses views to restrict access to the system catalog tables. The information that is used internally is critical to ALLBASE/SQL operations and should never be modified. Therefore, views are created on the system tables and SELECT authority is initially given to users with DBA authority. The base tables cannot be accessed by any user.

Using the GRANT OPTION Clause

A **grantable privilege** is a privilege obtained as a result of a grant given with the GRANT OPTION. The DBA, the owner of a table or view, or users with a grantable privilege can give a grantee a table or view privilege and, if their authority is direct (not through group membership), the ability to grant that same privilege to other users.

To revoke a grantable privilege from a user and revoke the chain of grants that may have been created by the user with the grantable privilege, use the REVOKE statement with the CASCADE option. Because a privilege cannot be revoked from the DBA or owner, cascading does not continue past that user. Therefore, a DBA or owner should not be included in the chain of grants. The DBA need not use the CASCADE option; but if it is not used, and a chain of grants exists, then an orphaned privilege is created. Orphaned privileges are discussed later in this chapter.

In this example the owner of a table grants grantable privileges to two managers:

Owner:

```
isql=> GRANT ALL ON PurchDB.Parts
> TO MGR1@DBMS, MGR2@DBMS WITH GRANT OPTION;
```

The managers grant a grantable select privilege to their employees:

MGR1@DBMS:

```
isql=> GRANT SELECT ON PurchDB.Parts
> TO EMP11@DBMS WITH GRANT OPTION;
```

MGR2@DBMS:

```
isql=> GRANT SELECT ON PurchDB.Parts
> TO EMP21@DBMS WITH GRANT OPTION;
```

The employees authorize their co-workers to also have and grant SELECT authority:

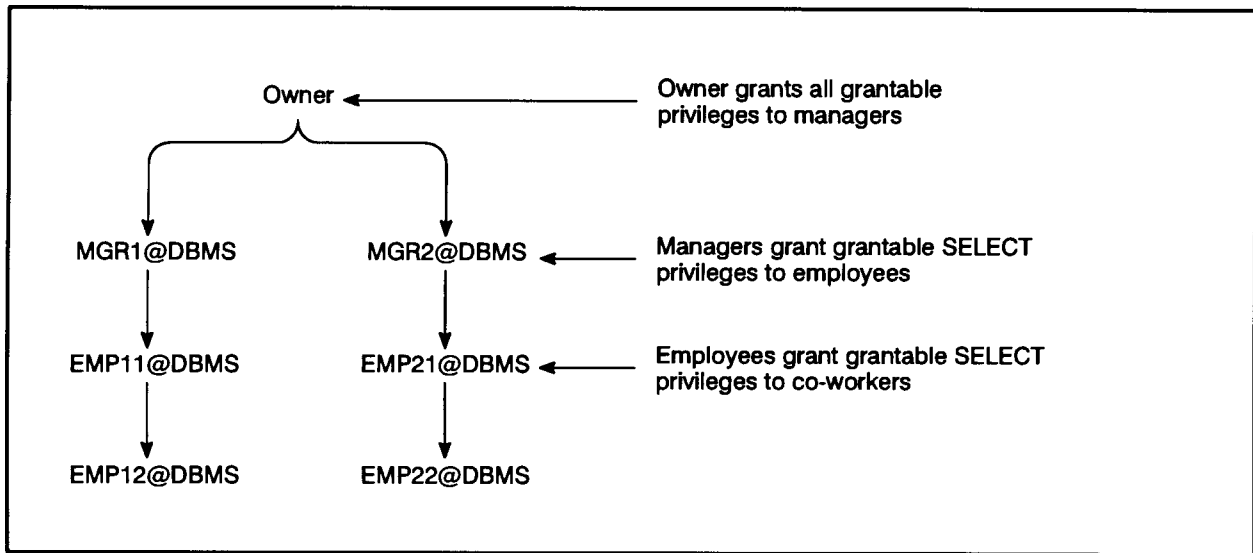
EMP11@DBMS:

```
isql=> GRANT SELECT ON PurchDB.Parts
> TO EMP12@DBMS WITH GRANT OPTION;
```

EMP21@DBMS:

```
isql=> GRANT SELECT ON PurchDB.Parts
> TO EMP22@DBMS WITH GRANT OPTION;
```

The security scheme would look like this:



LG200199_013

Figure 5-2. Example Database Security Scheme

If the table owner wants to revoke privileges from everyone, it is a simple procedure:

```
isql=> REVOKE ALL on PurchDB.Parts
> FROM MGR1@DBMS, MGR2@DBMS CASCADE;
```

However, if EMP21@DBMS is a DBA, the cascading stops on that chain at EMP21@DBMS, and EMP22@DBMS retains the grantable select privilege. Generally, a user would not grant authority to a DBA or owner, because he or she has it already; however, EMP21@DBMS could have been given DBA authority or had ownership transferred to him or her after the chain of grants was established.

If a DBA revokes SELECT authority without specifying the CASCADE option from EMP11@DBMS, then EMP12@DBMS is left with an **orphaned privilege**.

Orphaned Privileges

An orphaned privilege is one that was received from a grantor who no longer has authorization to grant or revoke that privilege. Orphaned privileges are created in the following ways:

- When a DBA revokes a privilege, without the CASCADE option, from a user who has granted privileges to others.
- When the DBA uses the BY clause of the GRANT statement to name a grantor who does not have the privilege to grant or revoke a privilege.
- When a member of a group grants a privilege but is then removed from the group.

Note Avoid orphaned privileges. Orphaned privileges make it impossible to use cascading to revoke a chain of grants.

An orphaned privilege exists if the grantor is *not* one of the following:

- Owner of the object
- A member of a group with OWNER authority
- A user with DBA authority
- A grantee with the right to grant appropriate authority

The ways to eliminate an orphaned privilege are:

- The owner of the object can grant the privilege with the WITH GRANT OPTION clause to recreate the authorization that was destroyed.
- The DBA can grant the privilege with the WITH GRANT OPTION clause and use the BY option to name a grantor who is authorized to grant and revoke privileges.

Using the WITH GRANT OPTION Clause and Authorization Groups

There are two approaches to granting authorities:

- Creating authorization groups. With this approach, authority is granted to an authorization group where members are added or removed from the group.
- Creating a chain of grants. With this approach, the WITH GRANT OPTION clause is used to grant authorities to individual users or classes, resulting in a chain of grants.

Mixing the two has complex effects, and in some cases is not allowed; we suggest that you do not mix them. For example, if you are using the WITH GRANT OPTION clause in an environment that uses group authorizations, be aware that removing a member from a group does not necessarily mean that member no longer has access to a table. The user could have been granted a privilege from another user via a grantable privilege.

If there are breaks in the chain of grants, it is difficult to maintain the security scheme. Breaks can occur in the following ways:

- When cascading stops because a DBA or owner is included in the chain of grants.
- When an orphaned privilege is created.

If you need to know the state of your security scheme, query the following system views:

- SYSTEM.TABAUTH
- SYSTEM.COLAUTH
- SYSTEM.SPECAUTH
- SYSTEM.GROUP
- SYSTEM.SPACEAUTH

Creating a Database Schema

A database is a collection of tables, views, indexes, procedures, and rules which have the same owner. A schema consists of the owner name, also called the authorization name, and the following statements, as needed, to define your database:

```
CREATE TABLE
CREATE VIEW
CREATE INDEX
CREATE PROCEDURE
CREATE RULE
CREATE GROUP
ADD TO GROUP
GRANT
```

You can define a database by either creating individual objects or using the CREATE SCHEMA statement to group the definitions within one statement. This statement allows forward and circular reference on referential integrity constraints. You can define the database within a statement so that database creation is atomic, that is, the database is either complete or nothing is created.

The appropriate system catalog views, such as SYSTEM.TABLE, SYSTEM.VIEWDEF, SYSTEM.TABAUTH, are updated to reflect the schema you have defined. After creating the schema, you can add, modify, or drop tables, and grant or revoke authorities as necessary.

Loading Tables

The easiest way to insert data into a table is by loading data from a file through ISQL using the LOAD command. The file can be either an EXTERNAL or an INTERNAL file:

- An EXTERNAL file is a sequential, ASCII, fixed-format file created either outside of ISQL or with the EXTERNAL option of the ISQL UNLOAD command. EXTERNAL files contain only data and are used when loading initial data into tables, or when restructuring tables by changing column names, column size, or data type.
- An INTERNAL file is created with the INTERNAL option of the ISQL UNLOAD command. INTERNAL files store the data format in a header that contains data and column descriptors used by ALLBASE/SQL to load data more efficiently. INTERNAL files are used to unload and load tables when restructuring without changing columns. The column definitions of the unloaded and loaded tables do not need to match exactly, since compatible data types are converted.

For details on table restructuring, refer to the “Maintenance” chapter in this guide. The syntax for the LOAD command is explained in detail in the *ISQL Reference Manual for ALLBASE/SQL and IMAGE/SQL*.

Loading from an External File

If you are loading data from a non-ALLBASE/SQL file, you must use the EXTERNAL option of the ISQL LOAD command.

To use the LOAD command, you must know the definition of the table to be loaded. The INFO command shows table definitions. The following example of the LOAD command loads data from an MPE/iX file to the PurchDB.OrderItems table:

```
isql=> LOAD FROM EXTERNAL ORDERITE
> TO PurchDB.OrderItems
> OrderNumber      1      6
> ItemNumber       9      2
> VendPartNumber   13     8  ?
> PurchasePrice    21     10
> OrderQty         31     4  ?
> ItemDueDate      37     8  ?
> ReceivedQty      47     2  ?
> END
```

The first number after the column name indicates where in the file the data for that column starts. The second number tells ISQL the length of the input. The last character is a null indicator. You must supply a null indicator if the column allows null values. The null indicator is any single character except a blank, a semicolon, a single or double quote, or the current ISQL escape character. Do not choose a null indicator that might also represent valid data. For example if you are loading integers, do not use zero as a null indicator. When a null indicator is found, a null value is loaded, not the character representing the null indicator. See the “Data Types” chapter of the *ALLBASE/SQL Reference Manual* for more information on null values.

You can also load from an external file using the description file option, as in the following example:

```
isql=> LOAD FROM EXTERNAL ORDERITE
> TO PurchDB.OrderItems
> USING orderdescrip;
```

All PurchDB tables are listed in the “Sample DBEnvironment” appendix in the *ALLBASE/SQL Reference Manual*.

An ISQL option exists for loading data from external files that contain EBCDIC, packed decimal, and zoned decimal data. Refer to the description of the LOAD command in the “ISQL Commands” chapter of the *ISQL Reference Manual for ALLBASE/SQL and IMAGE/SQL*.

Loading from an Internal File

You can unload an existing table using the INTERNAL option of the ISQL UNLOAD command, then load the file into the new table with the INTERNAL option of the LOAD command. This method is faster than loading an EXTERNAL file because the data is already in the necessary format. Use the INTERNAL option to move the data in a table in one DBEnvironment to a table with identical name, columns, and data types in another DBEnvironment.

Loading Tables with Constraints on Them

If a table has already been built with a referential integrity constraint, the LOAD command will exercise the constraint, testing each value for compliance. In order to speed loading in non-archive mode, you can use the following statements:

```
SET DML ATOMICITY AT ROW LEVEL
SET CONSTRAINTS DEFERRED
```

The SET DML ATOMICITY statement reduces logging overhead for the load operation when the DBEnvironment is running in non-archive mode to improve performance at load time. The SET CONSTRAINTS DEFERRED statement suspends integrity checking until a COMMIT WORK statement is issued, at which time the integrity checking takes place for the entire table. This approach is recommended.

Loading Tables with Rules Built on Them

If a table has a rule defined on it for the INSERT statement type, the rule will fire during load operations. If you are loading a set of tables that are related by a set of rules, and you know that the initial data already conforms to the conditions enforced by the rules, you can use the following statements to improve the performance of the initial table load:

```
SET DML ATOMICITY AT ROW LEVEL
DISABLE RULES
```

The SET DML ATOMICITY statement reduces logging overhead for the load operation when the DBEnvironment is running in non-archive mode to improve performance at load time. The DISABLE RULES statement turns off rule checking. You should only use DISABLE RULES for loads that take place in single-user mode, since the DISABLE RULES statement affects *all rules* in the DBEnvironment. As soon as the tables have been loaded, you should issue the following statements:

```
ENABLE RULES
SET DML ATOMICITY AT STATEMENT LEVEL
```

As an alternative to disabling rules, you can load the tables first, then create the rules that interrelate them. In any event, if you load data without using the rules, it is your obligation to make sure the data you are loading conforms to the rules. This is because the operation of rules is not retroactive; they will only fire on rows inserted after rule firing is enabled again.

Using Command Files for Loading

Use ISQL command files to create and test load and unload commands. Once the DBEnvironment is in operation, use the SQLGEN GENERATE LOAD and GENERATE UNLOAD commands to create load and unload scripts for you. Refer to the “Maintenance” chapter in this guide for more information and examples of using the UNLOAD command to restructure tables.

Backup and Recovery

Backup and recovery are activities that let you reconstruct your DBEnvironments in the event of database corruption or damage to your system. The type of backup and recovery you can perform depends in part on whether you choose archive or nonarchive logging for your system. The following activities are described in the sections that follow:

- Choosing an approach to backup and recovery
- Understanding log file types
- Choosing full or partial backup procedures
- Choosing full or partial recovery procedures
- Backup and recovery procedures for nonarchive logging
- Backup and recovery procedures for archive logging
- ALLBASE/SQL interface to True Online Backup
- Managing log files
- Starting a new log
- Monitoring the log with SQLMON
- Setting up a wrapper DBEnvironment

Once you have chosen a logging approach for your system, use the appropriate sections below to carry out these tasks.

Choosing an Approach to Backup and Recovery

When you configure the DBEnvironment, it runs with nonarchive logging by default. This is appropriate for data definition and table loading, since nonarchive logging results in more efficient loading, and usually there is little or no risk of data corruption.

After data definition and before doing the initial backup, you should decide which kind of backup, recovery, and logging you want to use for production. If you choose nonarchive logging, you should do a *static* backup using the SQLUtil STORE command. For archive logging, you should choose one of the following:

- If you have TurboSTORE software, do a *concurrent backup* using the SQLUtil STOREONLINE command.
- If you do not have TurboSTORE, issue the following SQL statements *in single user mode* in ISQL:

```
isql=> BEGIN ARCHIVE;  
isql=> COMMIT ARCHIVE;
```

Then immediately exit from ISQL and issue the SQLUtil STORE command.

Choosing Nonarchive Logging

If you choose nonarchive logging, you should back up the entire DBEnvironment at frequent intervals. Then, in the event of a media failure, you can restore the DBEnvironment from the most recent backup and manually enter all transactions that took place from the time of backup to the time of the failure. This approach makes sense if you use the DBEnvironment mainly for read operations, or if you process only a small number of transactions. With nonarchive logging, you do not back up the log files.

Note You cannot use the SQLUtil STOREONLINE command for DBEnvironment backups if you want to retain nonarchive logging. For backups of the DBEnvironment in nonarchive logging mode, use only the SQLUtil STORE command.

Choosing Archive Logging

If you choose archive logging, you create a backup of the entire DBEnvironment at periodic intervals (say, once a week) and back up the log files at shorter intervals (say, every day). In the event of a media failure, you restore the DBEnvironment first, and then you can apply each of the stored log files in chronological sequence to the restored copy of the DBEnvironment until you have rolled forward all completed transactions. If you wish, you can roll forward to a particular time prior to the crash by supplying a timestamp for recovery.

Archive logging affords the greatest security for the DBEnvironment. If you choose archive logging, it is best to place the log on a device that is different from any of the devices that contain other DBEnvironment files. For the greatest security, use dual logging with the two logs on different devices.

Since nonarchive logging is the default, you must explicitly turn archive logging on. If you have TurboSTORE software, the SQLUtil STOREONLINE command lets you perform the STORE without stopping the DBEnvironment; and it sets archive logging mode on for all subsequent activity until you decide to turn it off with a START DBE NEWLOG statement (omitting the ARCHIVE option).

Note Normally, you use the STOREONLINE command for DBEnvironment backups if you want to use archive logging. If your MPE/iX system does not have the TurboSTORE enhancements, you must use the SQLUtil STORE command instead. The complete procedure for this alternate approach is presented in a later section, “Static Backup Procedures in Archive Mode.”

Understanding Log File Types

It is important to understand the different conventions for naming and referring to log files in ALLBASE/SQL. This chapter assumes a basic distinction between the *log* (a logical object) and the specific log *files* which the log contains. Information about all the specific files in the log is stored in the DBECon file. You use the SHOWLOG command in SQLUtil to examine the DBECon file's directory of log files. SHOWLOG also displays general information about the log as a whole.

When you configure the DBEnvironment, you specify the names of one or two physical files (depending on whether you have chosen single or dual logging). These physical files will contain log records that are based on the transactions applied in the DBEnvironment. Later, you use the SQLUtil ADDLOG command to add additional log files to the DBEnvironment.

Understanding the LOG FULL Condition

Eventually, log files can become full, and the result is known as a **LOG FULL** error condition, which means that no additional transactions can be logged until log space is provided. When a LOG FULL condition arises, ALLBASE/SQL rolls back the current transaction and issues the following error message:

```
Log Full. (DBERR 14046)
```

In addition, in most cases other transactions are also rolled back.

Note	If a LOG FULL condition occurs while a rollback operation is taking place, <i>all</i> transactions are rolled back, including transactions that have performed no updates.
-------------	--

Using Single or Dual Logs

When you create the DBEnvironment, you specify either single or dual logs. In **single logging**, ALLBASE/SQL maintains one set of log files in either archive or nonarchive mode. The log file should be placed on a device different from the one containing the DBEnvironment. For greater security, you can specify **dual logging**, in which a duplicate set of log files is maintained. The second log file should not be on either the drive containing the DBEnvironment or the drive containing the first log file. When dual logging is in effect, all the procedures described in this chapter apply to the files of *both logs*.

Using Multiple Log Files

To avoid a LOG FULL condition, you can set up a DBEnvironment containing *multiple* log files for archive or nonarchive use, and in single or dual logging mode. When one file becomes filled, ALLBASE/SQL automatically switches to the next available file.

Log Names and Numbers

Log files are referred to in different ways, depending on the operation you are attempting to perform. In some instances, you must refer to a log by its **file name**. In other cases, you refer to the log by its **identifier number**.

Log File Names

Use a file name when you add a new log with the ADDLOG command, when you move it to a new location with MOVELOG, or when you must rescue it with RESCUELOG. The file name is an MPE file name.

Identifier Numbers

Use the identifier number when you purge a log file or when (in archive mode only) you store it. Each log file you add is given a specific identifier number; in the case of dual logging, the same identifier number refers to both files together.

Sequence Numbers

ALLBASE/SQL also keeps track of multiple log files through *sequence numbers*, which are stored along with the physical file names in the DBECon file. Sequence numbers are also written onto the tape at the time you issue a STORELOG command (described in a later section). In the case of dual logging, the same sequence number refers to both files together.

You never allocate sequence numbers yourself. They are used internally by ALLBASE/SQL to keep track of the order in which files are applied during the recovery process. If you attempt to recover a log file that has the wrong sequence number, you receive an error message.

Sequence numbers continue incrementing each time you switch into a new log file. As an example, suppose the DBEnvironment is configured for single logging in archive mode. Log 1 has the filename *DBELog1*, and initially it receives log identifier number 1 and sequence number 1. Log 2 has the filename *DBELog2*, and initially it receives log identifier number 2, and sequence number 2. When Log 1 runs out of space, ALLBASE/SQL switches to Log 2. After Log 1 has been backed up and when it no longer has active transactions in it, it becomes available for reuse. Then, when ALLBASE/SQL switches over to Log 1 again, it receives sequence number 3, although it still has identifier number 1, and it is still called *DBELog1*.

Using Nonarchive Logs

A nonarchive log behaves like a circular fileset. A **circular fileset** is a group of files in which the first one is overwritten when the last one is full. If a nonarchive log file becomes full, ALLBASE/SQL first issues a **checkpoint** in an effort to reclaim the file space taken up by transactions that have already been committed. (A checkpoint is an event that flushes data and log buffer contents to disk, and reclaims nonarchive log file space for reuse.) If a checkpoint does not reclaim enough space for the current transaction, ALLBASE/SQL attempts to switch to the next file (if it is available). A log file is considered available if it does not contain any log records for transactions active at the last checkpoint. If the log is still full, and there is no other log available to switch into, a LOG FULL condition occurs, and the transaction is rolled back.

Multiple Files in Nonarchive Mode

You can add additional physical log files to the nonarchive DBEnvironment by means of the SQLUtil ADDLOG command. You can add log files *without* stopping the DBEnvironment. Each nonarchive file has a unique **log name**, which is an MPE file name, and a **log identifier**, which is an integer by which the file is known internally to ALLBASE/SQL. If you are using dual logging, there are two MPE file names, but there is only one identifier. In nonarchive logging, you can use multiple log files to ensure that even a very large transaction will not cause a LOG FULL condition. You do *not* back up nonarchive log files.

Figure 6-1 shows a nonarchive log for a DBEnvironment that is running in multiuser mode, with different users starting and ending transactions at different times.

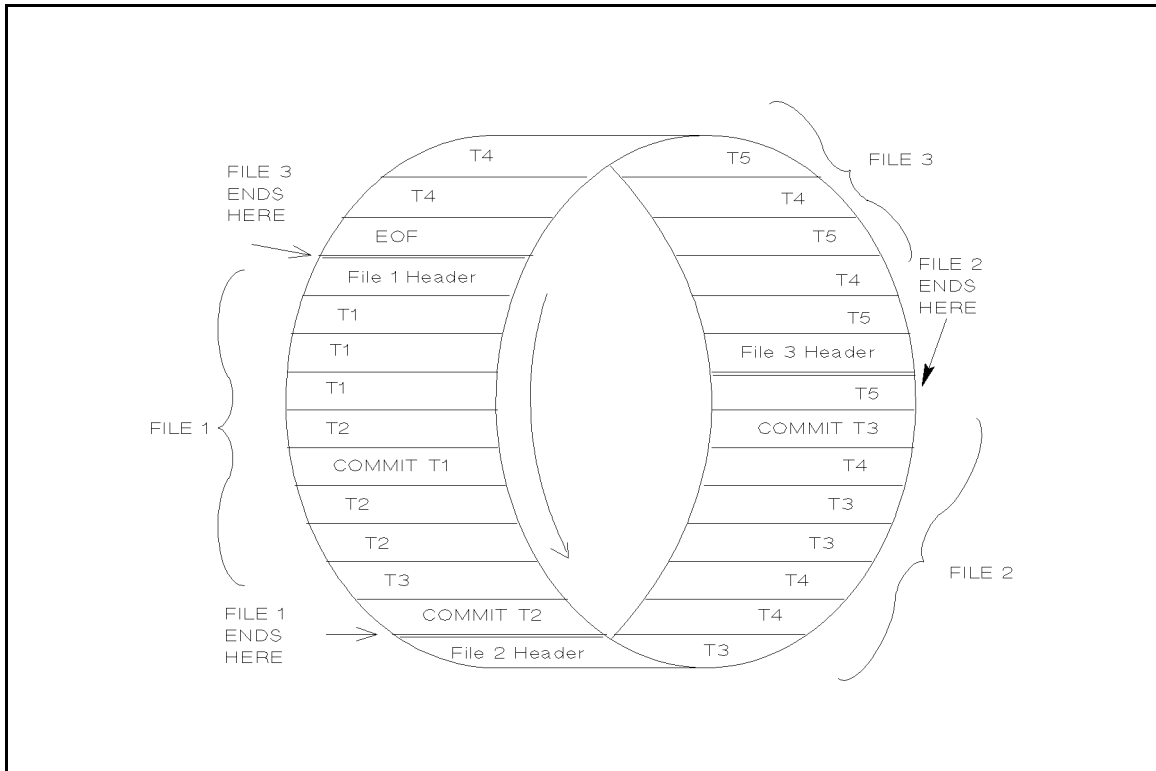


Figure 6-1. Nonarchive Log

The log has three files—File 1, File 2, and File 3; Files 1 and 2 have been filled, so ALLBASE/SQL has already switched to File 3. Assume that transactions 4 and 5 (T4 and T5) are uncommitted. When File 3 becomes full, ALLBASE/SQL will issue an internal checkpoint. Assuming there is not enough space left in File 3, ALLBASE/SQL will switch back to File 1 and continue logging. At the moment shown in the figure, both File 2 and File 3 contain active transactions.

A nonarchive log file should be used when you can frequently do a static backup of the DBEnvironment so that rollforward recovery is not needed to recover in the event of a media failure. (Static backup is described further in a later section.)

Using Archive Logs

Archive logs are also circular filesets. The difference is that file space cannot be reclaimed—even from files with no active transactions in them—until the files are backed up. If an archive log file becomes full, ALLBASE/SQL attempts to switch to the next file (if one is available). It is available if it does not contain log records for any currently active transactions, and *if it has been backed up*. If the next file is not available, a switch is attempted again after a checkpoint. (A checkpoint is an event that flushes data and log buffers to disk.) If the switch still fails, a LOG FULL condition occurs, and the transaction is rolled back.

When more space is needed, add a new log file with the ADDLOG command. You can also reclaim log file space by using STORELOG to back up a log file when it is full. Then, if there are no longer any active transactions in the file, it can be reused. You should monitor archive log files periodically with the SHOWLOG command to make sure there is enough space in them.

When you use archive logging, you must periodically back up the DBEnvironment and the log files. The backups may be used in the event of a corruption of the DBEnvironment. Refer to the section “Backup and Recovery Procedures for Archive Logging” later in this chapter for a complete description of the procedures for managing archive log files.

Figure 6-2 shows an archive log as implemented in ALLBASE/SQL.

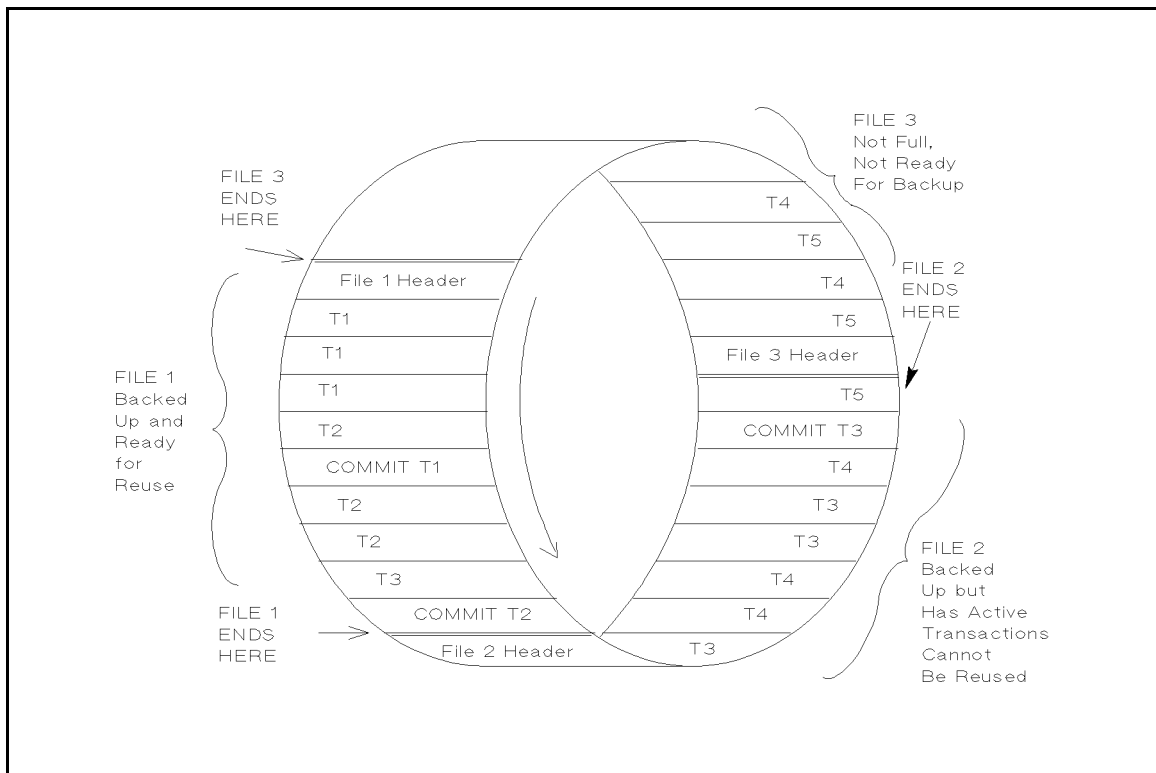


Figure 6-2. Archive Log

Multiple Files in Archive Mode

Archive log files are known by three different designations:

- Log name
- Log identifier
- Log sequence number

The *log identifier* is a number that uniquely identifies the file in the log directory. The *log name* is an MPE file name. If you are using dual logging, there are two MPE file names, but there is only one identifier. The *log sequence number* is important only for archive log files. It identifies the sequence the file occupies in the stream of rollforward recovery from the time the DBEnvironment was created. Sequence numbers are never reset.

When you use the STOREONLINE command, a new archive point is defined and the sequence number of the log file containing the first active transaction becomes the starting sequence number for the new archive log. As you cycle through a set of log files, filling them with transactions, then backing them up with STORELOG, the sequence number increments each time a new log file is used. Note that the assignment of sequence numbers has nothing to do with log identifier numbers or log names, so the same file may be used again and again with different sequence numbers.

In archive mode, you can use multiple files as a way of ensuring that there is always enough log file space. For example, when the first log file becomes full, ALLBASE/SQL will switch automatically to the second, which is given the next higher **sequence number**. Then, you can back up the first file, and when all transactions that were in progress in the first file have been committed or otherwise terminated, the file becomes available for reuse. A log file can be backed up as soon as ALLBASE/SQL has finished writing to it, but it cannot be purged or reused until all active transactions in it have been committed or rolled back.

The illustration in Figure 6-3 shows a log with three log files—DBELog1, DBELog2, and DBELog3 (identifier numbers 1, 2, and 3) over a period of several days.

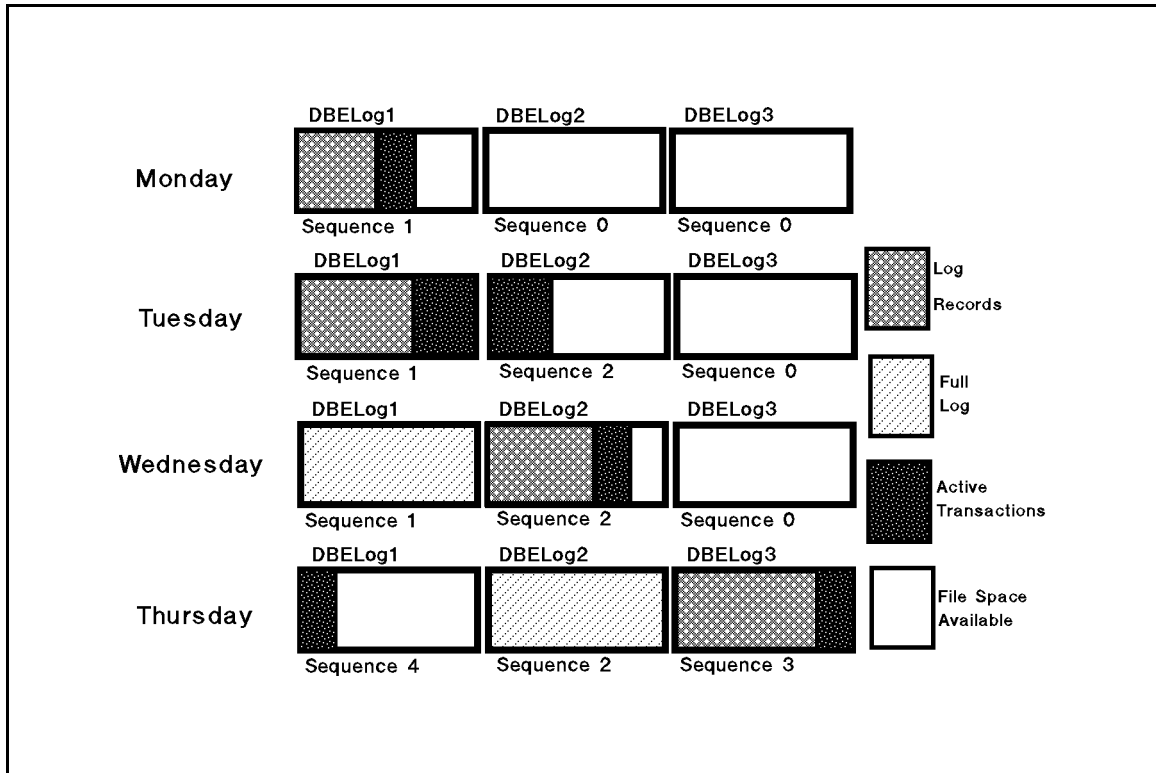


Figure 6-3. Log Switching in Archive Mode

At 5 pm on Monday, DBELog1 is not completely full, so it is not ready for backup. It has sequence number 1, whereas the two unused files DBELog2 and DBELog3 both have sequence number 0. **On Tuesday at 5 pm**, DBELog1 is full; there are still some active transactions in it, but it can be backed up using STORELOG. DBELog2 also has active transactions, but since DBELog2 is not yet full, no backup of DBELog2 is possible yet. **On Wednesday**, all the active transactions are in DBELog2, but DBELog2 is not yet full and therefore cannot be backed up. DBELog1 is now available for reuse, since it was backed up on Tuesday and now contains no active transactions. **On Thursday**, DBELog2 is full, so it is ready for backup using STORELOG. Furthermore, DBELog3 is now full and ready for backup. Note that active transactions now “wrap around” into DBELog1, which now has sequence number 4. When there are no more unused files available, ALLBASE/SQL will switch back to an earlier one and reuse it if it has been backed up and if it contains no active transactions. Note also that the sequence number of DBELog1 is 1 on Monday and 4 on Thursday. The file names and identifier numbers of log files are recycled, but the sequence numbers are never repeated.

Choosing Full or Partial Backup Procedures

ALLBASE/SQL provides the capability to either back up all the files in the DBEnvironment (**full backup**) or to back up only a subset of the files in the DBEnvironment (**partial backup**). You may choose full or partial backup when using either nonarchive or archive mode logging. If you choose to do a full backup, you can later do either a full recovery or a partial recovery from the full backup. If you do a partial backup, you can only do a partial recovery from that partial backup. The partial recovery can include all or a subset of the files contained in the partial backup.

The first time you store your DBEnvironment you should do a full backup. On subsequent stores you should do a full backup if: your entire DBEnvironment is on one drive, the entire DBEnvironment is changing (being written to) between backups, or if you need to restore the entire DBEnvironment (in the case of disk failure).

There are several reasons why you may want to do partial backups subsequent to your first full backup.

If you have DBEFiles containing a table that is used infrequently, you may choose to attach the DBEFiles only when you use the table. Then you can detach DBEFiles when the table is not in use in order to save space on the system. (As an example, you could have a history file that is used only once a month when you transfer old account information from the account table into the history table. At that time, you use `RESTORE PARTIAL` to place the files on the system, attach the DBEFiles, and then transfer the historical information. Then you do a `STORE PARTIAL` or `STOREONLINE PARTIAL` to store only those files associated with the history table, detach the files and remove them from the system.)

If your entire DBEnvironment is spread across several drives, it may be advantageous to do partial backups subsequent to your first full backup in the following two cases. In the first case, only a subset of the DBEnvironment files contains tables that are written to between backups. Then you can do partial backups of the files that are written to, and rely on the first backup to restore files that are read only. In the second case, only a subset of the DBEFiles contains essential data, and another subset of the files contains nonessential data (or data that can be derived from the essential data). In that case, you may choose to do a partial backup that includes the essential data, and then reconstruct the other data in the event of a hard crash.

Note

If you do a `STORE PARTIAL` or `STOREONLINE PARTIAL`, it is your responsibility to store *all* the DBEFiles that contain tables that will be written to by your transactions. If you fail to do this, you will later be unable to restore your DBEnvironment to a consistent state.

Choosing Full or Partial Recovery Procedures

ALLBASE/SQL also provides the capability to restore all the files in the DBEnvironment (**full recovery**) or to restore only a subset of the files in the DBEnvironment (**partial recovery**).

You may choose full or partial recovery when using either nonarchive or archive mode logging. If a crash occurs in a DBEnvironment that is contained solely on one drive, you must do a full recovery of the entire DBE (or that portion of the DBEnvironment that you stored because it was essential data).

If a crash occurs to a DBE that is spread across several drives, regardless of whether you are using nonarchive or archive logging, the kind of recovery you must do depends on certain operating system conditions and on which ALLBASE/SQL files were destroyed in the crash.

The failure conditions on the operating system must be such that you are able to restore only those files that were located on the device that failed. Your failure conditions may require you to restore files on several or all drives associated with your DBEnvironment. If you must restore files on several or all devices, partial recovery is not useful. See your system administrator for details for your operating system. Be sure that your disaster recovery plan includes the operating system considerations needed for recovery of your DBEnvironment, not just the ALLBASE/SQL considerations.

You must do a full recovery if any of the following files were destroyed in the crash:

- Any of the SYSTEM DBEFiles (DBEFile0 or any files you added to the SYSTEM DBEFileSet) and/or the DBEConFile which is stored on the same drive as the SYSTEM DBEFiles.
- In the case of single logging, your single log file; unless, in the case of archive logging, the file has never been used, or you have already stored the log file off, and the file has not yet been reused. (If you are using dual logging, the loss of one leg of the log does not require you to restore the DBEnvironment. Just repair the damaged device and create a new set of log files to restore dual logging.)

If the crash destroyed any other files than those listed above, you can repair the damaged drive and then do a partial recovery which only includes the files that were damaged in the crash. While the damaged drive is out of operation, you can still operate the remainder of the DBEnvironment so long as the operating system permits and so long as users only access the DBEFiles on the undamaged drives. Any attempt to access files on the damaged drives will result in the generation of an error message.

Backup and Recovery Procedures for Nonarchive Logging

This section describes the steps you should follow if you will be using nonarchive logging in full production after loading your tables. The following are described separately:

- Nonarchive backup procedures
- Adding files to a nonarchive log
- Nonarchive full recovery procedures
- Nonarchive partial recovery procedures

Nonarchive Backup Procedures

If you are using *nonarchive logging*, you should make either a full or partial *static* backup of the DBEnvironment at frequent intervals.

The exact frequency depends on how much the database changes between backups. If there is very little change, then the need for backup is not as great.

Use the SQLUtil STORE or STORE PARTIAL command to make a backup copy of all or a subset of the database files in a DBEnvironment.

The STORE command starts the DBEnvironment for you, thus rolling back any incomplete transactions. Follow these steps to make a backup copy of the DBEnvironment using SQLUtil STORE:

1. Stop the DBEnvironment and then exit ISQL.
2. From the group containing the DBEConFile and the SYSTEM DBEFileSet use the SQLUtil STORE or STORE PARTIAL command to make a copy of the DBEnvironment or a subset, including the DBECon file. You can use the SQLUtil STORE or STORE PARTIAL command only when the DBEnvironment is stopped.
3. Start the DBEnvironment for production only after the STORE or STORE PARTIAL is complete.

When you use nonarchive logging, you do *not* back up log files since you can not use nonarchive log files for rollforward recovery.

Caution	If you must use the MPE/iX STORE command to make a backup copy of the DBEnvironment, you should first START the DBEnvironment in single-user mode to roll back any incomplete transactions. Otherwise you risk making an inconsistent copy of the DBEnvironment. If you use MPE/iX STORE, you should also back up the DBECon file.
----------------	--

Adding Files to the Nonarchive Log

In nonarchive mode, your log files should be large enough to hold log records for the largest possible transaction as carried out by the maximum number of concurrent users. A formula for calculating the required size is given in the “Physical Design” chapter. If you develop a need for a larger log, you can use the SQLUtil ADDLOG command to provide another file. Then, when a transaction fills up the first file, ALLBASE/SQL will switch to the second one automatically.

If the need for more file space is temporary, use the ADDLOG command to add a new file, then use the PURGELOG command to remove it when it is no longer needed.

Nonarchive Full Recovery Procedures

In the event of a soft crash, simply start up the DBEnvironment again using a START DBE statement or a CONNECT (if AutoStart is enabled). Rollback recovery is automatic. If a transaction was not complete at the time of the crash, you must reenter it.

In the event of a hard crash (media failure), if you need to restore the entire DBEnvironment from your backup. Here is the procedure to use once the drive is replaced:

1. Make sure you are in the group and account the DBEnvironment DBEConFile and SYSTEM DBEFileSet were in at the time of the crash.
2. Purge any DBEnvironment files that remain.
3. Use the SQLUtil RESTORE command to restore the DBECon file and the DBEnvironment files.
4. Issue a START DBE NEWLOG statement to create a new log file. Specify SINGLE or DUAL logging, as before.
5. Use the SQLUtil PURGELOG command to purge any old log files that existed prior to issuing the START DBE NEWLOG statement. (Be careful not to purge your new log files.)
6. Use the ADDLOG command to create additional log files for the log.
7. Manually reapply the transactions that you had entered since the backup was taken.
8. If appropriate, use the SQLUtil STORE command to back up the DBEnvironment. (See “Nonarchive Backup Procedures,” above.)

Nonarchive Partial Recovery Procedures

In the event of a hard crash of a disk that does not contain either the SYSTEM DBEFileSet and the DBEConFile, or your single log file used for single mode logging, you can use the SQLUtil RESTORE PARTIAL command to do a partial recovery of the DBEnvironment. You can recover just those DBEFiles that were damaged by the crash. Here is the procedure to use once the damaged drive is replaced:

1. Make sure you are in the group and account from which you made the backup (usually the one containing the DBEConFile and the SYSTEM DBEFileSet).
2. Use the SQLUtil STOREINFO command to verify the fully qualified filename (including group and account) of the DBEFiles you are going to restore (you must use the fully qualified filename, just as shown by the STOREINFO command).
3. Use the SQLUtil RESTORE PARTIAL command to restore the DBEFileSets or DBEFiles that were damaged. The RESTORE PARTIAL may from a full backup or a partial backup, as appropriate. (As the logfiles must have been intact as a condition for doing a RESTORE PARTIAL, you should not need to create or add more log files.)

Caution

All files that will be used by the transactions to be reapplied must be restored to their state as of the last backup whether they were damaged or not. This is because all files must be synchronized to the same starting point for the reapplication of transactions to result in a consistent DBEnvironment. This is true even for transactions which only read data because the data must be in the same state it was in when the transactions were originally applied.

4. Manually reapply all transactions that were entered since the last backup against the tables lost in the crash (no transactions need to be entered against the tables that were contained in undamaged files).
5. If appropriate, use the SQLUtil STORE or STORE PARTIAL command to make a full or partial backup of the DBEnvironment before you resume operations.

Backup and Recovery Procedures for Archive Logging

This section describes the steps you should follow if you will be using archive logging in full production after loading your tables. The following are described separately:

- Online backup procedures in archive mode
- Static backup procedures in archive mode
- Adding files to the archive log
- Archive recovery procedures

Note

Online backup procedures in archive mode requires the presence of a product known as TurboSTORE (TSTORE.PUB.SYS). If your system does not have this product, you *must use static backup procedures* for archive mode backups.

Online Backup Procedures in Archive Mode

To use *archive logging*, follow these steps:

1. Load all your tables using the ISQL LOAD command before making the first backup. Preferably, the loading should be done in nonarchive mode. Refer to the description of the LOAD command in the “ISQL Commands” chapter of the *ISQL Reference Manual for ALLBASE/SQL and IMAGE/SQL* for suggestions on how to obtain best performance with the LOAD command.
2. Add an appropriate number of log files to the DBEnvironment using the ADDLOG command. A minimum of two log files is necessary.
3. For your initial backup, from the group and account that contains the DBEConFile and the SYSTEM DBEFileSET, use the SQLUtil STOREONLINE command to store a copy of the DBEnvironment. This command stores the DBEnvironment and initiates archive logging. If archive logging is already on, it remains on. It is recommended that your initial backup be a full backup but if you have decided that only a subset of the DBEnvironment is essential, you can do a STOREONLINE PARTIAL for your first backup. Backups subsequent to your initial backup may use STOREONLINE or STOREONLINE PARTIAL as appropriate.

You cannot use `STOREONLINE` or `STOREONLINE PARTIAL` if there are any pseudo-mapped files configured in your system. Before doing `STOREONLINE` or `STOREONLINE PARTIAL`, use the SQLUtil command `SHOWACCESS` to see if any DBEFiles are pseudo-mapped. Convert the pseudo-mapped files back to mapped files using the SQLUtil `MOVEFILE` command.

4. As soon as possible, use the `STORELOG` command to store a copy of the log files that were written to by active transactions while the `STOREONLINE` or `STOREONLINE PARTIAL` was being done.

These log files are necessary for later recovery, since they contain log entries for transactions that were in process at the time the `STOREONLINE` command was issued. The `STOREONLINE` or `STOREONLINE PARTIAL` command will indicate the log sequence numbers for the log files which need to be stored.

5. When a log file becomes full, back it up using the `STORELOG` command. Be sure to label each log file backup with the log sequence number and the date of the backup. To check the size of a log file, you can use the `SHOWLOG` command. Once you back up the log file, `ALLBASE/SQL` can use it again. Refer to the section “Managing Log Files,” below.

Note The online backup is *not usable for rollforward recovery* until you have backed up all log files that contain transactions that were active during the time the `STOREONLINE` command was in progress.

Once you have turned archive logging on, the only way to turn it off is to define a new log with the `START DBE NEWLOG` statement.

Static Full or Partial Backup Procedures in Archive Mode

It is also possible to carry out **static backups** in archive mode. A static backup is one that is made with the SQLUtil `STORE` or `STORE PARTIAL` command, and it requires you to `STOP` the DBEnvironment. The procedure is as follows:

1. Load all your tables using the `ISQL LOAD` command before making the first backup.
2. Stop the DBEnvironment.
3. Add an appropriate number of log files to the DBEnvironment using the SQLUtil `ADDLOG` command. A minimum of two log files is necessary.
4. Start the DBEnvironment again, and immediately issue a `BEGIN ARCHIVE` and a `COMMIT ARCHIVE` statement. Stop the DBEnvironment again.
5. From the group and account containing the `DBEConFile` and `SYSTEM DBEFileSet` use the SQLUtil `STORE` or `STORE PARTIAL` command to create a static backup of the DBEnvironment, including the `DBECon` file. Reply `Y` to the prompt “Do you wish to proceed (y/n)?” It is recommended that your initial backup be a full backup but if you have decided that only a subset of the DBEnvironment is essential, you can do a `STORE PARTIAL` for your first backup. Backups subsequent to your initial backup may use either the SQLUtil `STORE` or `STORE PARTIAL` command, as appropriate.
6. When a log file becomes full, back up the log file using the `STORELOG` command. Respond `Y` to the prompt “Use Static Store (y/n)?” Refer to the section “Managing Log Files,” below.

Once you have turned archive logging on, the only way to turn it off is to define a new log with the `START DBE NEWLOG` statement, omitting the `ARCHIVE` option.

Adding Files to the Archive Log

In archive logging, your log will contain records for all the transactions your system handles between backups. The frequency of backup depends on the total volume of data being logged. A formula for calculating the total size is given in the “Physical Design” chapter.

If you develop a need for additional log files, you can use the SQLUtil `ADDLOG` command to provide another file. Then, when a transaction fills up the first file, `ALLBASE/SQL` will switch to the second one automatically.

In archive logging, the log grows continually, so you must continue providing additional log file space until you do another full backup. You can make old log files available for reuse by issuing the SQLUtil `STORELOG` command. Once a file has been stored this way, it can be reused with a different sequence number as soon as all the active transactions in it are complete.

More detailed information about managing archive log files is found in a separate section below.

Archive Recovery Procedures

Soft crashes (program aborts or system failures) rarely damage `DBEnvironment` files and can usually be remedied with rollback recovery, which is carried out automatically the next time the `DBEnvironment` is started. This does not require any special action on your part. You simply restart any transactions that were active at the time of the crash.

Media failures, on the other hand, can be very serious and often require you to reconstruct your `DBEnvironment` from a backup. This requires rollforward recovery using the log files previously stored.

Rollback Recovery

Under normal circumstances, rollback recovery is automatic. When the `DBEnvironment` is stopped, either implicitly when the last `DBE` session terminates or explicitly when the `STOP DBE` statement is executed, `ALLBASE/SQL` writes a checkpoint to the log file. When the `DBEnvironment` is started again, `ALLBASE/SQL` will perform rollback recovery if any transactions were still uncommitted at the time the `DBEnvironment` was stopped.

Rollback recovery is all you need to recover from the most common types of system failures, such as a soft crash. To recover from a crash that does not damage the `DBEnvironment` files or the log files, you merely issue a `START DBE` or `CONNECT` statement to start the `DBEnvironment`. This will automatically use the log file to roll back any transactions that were not committed at the time of failure.

Starting the `DBEnvironment` ensures the following:

- All incomplete transactions at the time of failure are rolled back.
- All committed transactions are permanently recorded in `DBEFiles`.
- A new checkpoint is taken.

Full Rollforward Recovery

Full rollforward recovery lets you recreate an entire DBEnvironment following a media failure or logical corruption of DBEnvironment files. The process depends on the availability of backed up copies of the DBECon file, as well as all log files and DBEFiles.

Rollforward recovery begins with a restored DBEnvironment. Use the following steps in SQLUtil:

1. Use the SQLUtil SHOWLOG command (with a connect option of “no”) to determine which log files were not yet stored at the time of the crash and which show the status “backup required.”
2. Use the RESCUELOG command to store a copy of any log files that have not been stored at the time of the crash, including all log files that contained active transactions (those files with a status of “backup required”).
3. Use the RESTORE command to restore the DBEnvironment into the same group and account from which it was originally backed up. If any old DBEnvironment files are present, remove them first. RESTORE restores the DBECon file and all DBEFiles.
4. Use the SHOWLOG command (with a connect option of “no”) to display the log’s status as it was when the DBEnvironment was backed up. Make a note of the *First Log Sequence Number Needed for Recovery*.
5. Use the SETUPRECOVERY command to initiate a recovery process. If you wish, you can specify a time to recover to. SETUPRECOVERY also lets you specify the name and characteristics of the new log file for the restored DBEnvironment. The restored DBEnvironment is automatically in archive mode, since it had previously been stored that way with a STOREONLINE command.
6. Use RESTORELOG to restore a copy of each log file to the current group and account, starting with the file that corresponds to the *First Log Sequence Number Needed for Recovery* from Step 4. For each file you restore, you must specify a new file name. It is easiest to maintain an orderly process if the new file name contains the log sequence number of the log file being stored.
7. Use RECOVERLOG to apply each restored log file to the DBEnvironment. Enter the new name of the file as chosen during Step 6. ALLBASE/SQL will check the sequence number of the file as you apply it. If the file is out of sequence, you will see an error message.

Once a log file is no longer needed for recovery, it is purged automatically.
8. Repeat steps 6 and 7 until you have restored and recovered all the log files desired.

Note

If you have enough space on your system, you can use several RESTORELOG commands one after the other to restore all the log files, then several RECOVERLOG commands one after the other to apply all the log files to the DBEnvironment. When restoring and recovering groups of log files in this way, it is a good idea to rename each restored file to a distinct file name that indicates the order in which the file will be applied to the DBEnvironment.

Example:

```
Log00001
Log00002
Log00003
Log00004
```

9. After you have recovered all log files, issue the ENDRECOVERY command to complete the recovery process. (This step is not needed if you specified a recovery time in the SETUPRECOVERY command.)
10. Use the SQLUtil PURGEFILE command to purge any old log files that remain. Be careful *not* to purge the log file you specified when using the SETUPRECOVERY command.
11. Use the SQLUtil ADDLOG command to create additional log files as needed. You must have at least two files if you use a dual log and you want ALLBASE/SQL to automatically switch to a new log file when the current log file is full.
12. Exit from SQLUtil, then start the DBEnvironment as you would normally.

Each RESTORELOG and RECOVERLOG step in this process could be carried out with a different invocation of SQLUtil. Also, rollforward recovery is possible from an earlier archive backup of the DBEnvironment, ignoring an intervening backup, as long as the logs are available. Simply apply all the logs in sequence up to the desired recovery time after restoring the DBEnvironment from the earlier backup. (SHOWLOG will display the *First Log Sequence Number Needed for Recovery*.) Rollforward recovery across new logs is also supported, provided the logs are archive logs.

Partial Rollforward Recovery

Partial rollforward recovery lets you recreate a subset of the DBEnvironment following a media failure or logical corruption of DBEnvironment files. The process depends on the availability of backed up copies of the DBECon file, as well as all log files and DBEFiles for the subset of the DBEnvironment damaged by the failure.

Partial rollforward recovery begins with a DBEnvironment in which the DBEConFile and the SYSTEM DBEFileSet are intact, and the appropriate log files are intact or have been properly stored. Use the following steps in SQLUtil:

1. Use the SQLUtil SHOWLOG command (with a connect option of “no”) to determine which log files were not yet stored at the time of the crash and which show the status “backup required.”
2. Use the RESCUELOG command to store a copy of any log files that have not been stored at the time of the crash, including all log files that contained active transactions.
3. Use the SQLUtil STOREINFO command to verify the fully qualified filename (including group and account) of the DBEFiles you are going to restore (you must use the fully qualified filename, just as shown by the STOREINFO command).

4. Use the SQLUtil DETACHFILE command to detach from the DBEnvironment the files to be operated on by the partial rollforward recovery. This prevents users from attempting to use these files until you have them rolled forward to the appropriate period in time. You must roll all the way forward to the moment of the failure (or to the moment the file was detached, if the file is detached.)
5. From the group and account from which you stored the DBEnvironment (usually the one containing the DBEConFile and the SYSTEM DBEFileSet) use the RESTORE PARTIAL command to restore the appropriate subset of the DBEnvironment into the same group and account from which it was originally backed up. The RESTORE PARTIAL may be from a full backup or a partial backup, as appropriate. (As the log files must have been intact as a condition for doing the RESTORE PARTIAL, you should not need to create or add more log files.) If any damaged DBEnvironment files are still present, remove them first. RESTORE PARTIAL restores the specified DBEFiles.
6. Use the SHOWLOG command (with a connect option of “no”) to display the log’s status as it was when the DBEnvironment was backed up. Make a note of the *First Log Sequence Number Needed for Recovery*.
7. Create a new group that does not contain any DBEConFile or SYSTEM DBEFileSet.
8. From that new group use the SETUPRECOVERY PARTIAL command to initiate a recovery process, specifying the name for a temporary DBEnvironment and the DBEFiles needed for recovery. Rollforward recovery of the damaged files will be done using the temporary DBEnvironment so that the original DBEnvironment can remain in use while the damaged files are being brought up to the desired state. You must roll forward all the way to the time of the failure (or the time the file was detached, if the file is detached.) SETUPRECOVERY PARTIAL also creates a default temporary log file and DBEFile0 which are automatically removed at the end of the partial recovery process. The temporary DBEnvironment is automatically in archive mode, since the original DBEnvironment had previously been stored that way with a STOREONLINE or STOREONLINE PARTIAL command.
9. Use RESTORELOG to restore a copy of each log file to the current group and account, containing the temporary DBEnvironment, starting with the file that corresponds to the *First Log Sequence Number Needed for Recovery* from Step 6. For each file you restore, you must specify a new file name. It is easier to maintain an orderly process if the new file name contains the log sequence number of the log file being stored.
10. Use RECOVERLOG to apply each restored log file to the DBEnvironment. Enter the new name of the file as chosen during Step 6. ALLBASE/SQL will check the sequence number of the file as you apply it. If the file is out of sequence, you will see an error message. Once a log file is no longer needed for recovery, it is purged automatically.
11. Repeat steps 9 and 10 until you have restored and recovered all the log files desired.

Note

If you have enough space on your system, you can use several RESTORELOG commands one after the other to restore all the log files, then several RECOVERLOG commands one after the other to apply all the log files to the DBEnvironment. When restoring and recovering groups of log files in this way, it is a good idea to rename each restored file to a distinct file name that indicates the order in which the file will be applied to the DBEnvironment.

Example:

```
Log00001
Log00002
Log00003
Log00004
```

12. After you have recovered all log files, issue the ENDRECOVERY command to complete the partial recovery process. (This step is not needed if you specified a recovery time in the SETUPRECOVERY PARTIAL command.) The ENDRECOVERY command will remove the temporary DBEnvironment, including the temporary DBEFile0 and log files, that were created for the partial recovery process.
13. Use the SQLUtil ADDLOG command to create additional log files as needed. You must have at least two files in each leg of the log if you want ALLBASE/SQL to automatically switch to a new log file when the current log file is full.
14. Exit from SQLUtil, and move to the group and account containing the original DBEnvironment DBEConFile and SYSTEM DBEFileSET.

Each RESTORELOG and RECOVERLOG step in this process could be carried out with a different invocation of SQLUtil. Also, rollforward recovery which skips an intervening backup is possible from an earlier archive backup of the DBEnvironment, as long as all the logs are available. Simply apply all the logs in sequence up to the desired recovery time after restoring the DBEnvironment from the earlier backup. (SHOWLOG will display the *First Log Sequence Number Needed for Recovery*.) Rollforward recovery across new logs is also supported, provided the logs are archive logs.

15. Use the SQLUtil ATTACHFILE command to re-attach the fully recovered DBEFile(s) to the DBEnvironment so they can again be accessed by users. No additional log files should be necessary because the original log files had to be intact as a condition for doing partial rollforward recovery.
16. Continue using the DBEnvironment as you would normally.

ALLBASE/SQL Interface to True Online Backup

ALLBASE/SQL files can also be stored along with other files online without shutting down the DBEnvironment. This enhancement is done to help you when you do not want to use STORE/STOREONLINE in SQLUtil. The normal STORE process in SQLUtil requires the DBEnvironment to be shutdown while taking the backup, whereas, STOREONLINE calls TurboSTORE internally and the DBEnvironment being backed up is quiesced during the attach period. After the attach period, the normal activity is resumed.

With the above technique, ALLBASE/SQL users are expected to close their applications or files when backup is done in SQLUtil. This mechanism does not provide a complete backup solution to the MPE users. This becomes more complicated if the system has more than one DBEnvironment because additional backups must be scheduled on the system. Consequently, extra overhead is added in terms of time and number of tapes.

To provide a complete backup solution, the true online backup enhancement is supported by ALLBASE/SQL. To store an ALLBASE/SQL DBEnvironment online with all other files in mygroup and myaccount, execute the following command in the command mode:

```
STORE @.mygroup.myaccount;*TAPE;ONLINE=START
```

The store process will do the following:

1. Call an ALLBASE/SQL routine to get the list of files associated with the DBEnvironment. If the DBEnvironment is in archive-mode, the list includes DBECon and DBEFiles. In addition the list includes the log files.
2. Quiesce each DBEnvironment by calling an ALLBASE/SQL routine. This routine also writes BEGIN ARCHIVE checkpoints if the DBEnvironment is in archive mode.
3. Make a snapshot of ALLBASE files taken.
4. Unquiesce the DBEnvironment by calling another ALLBASE/SQL routine.
5. Call an ALLBASE/SQL routine when the store is finished. This routine indicates successful completion or failure. If the store is completed successfully, it writes a COMMIT ARCHIVE checkpoint; otherwise an ABORT ARCHIVE checkpoint is written.

Now if the restore is done, the files are in a consistent state, because the database files were quiesced when the snapshot was taken. However, for the archive-mode DBEnvironment, rollforward recovery must be applied.

The other option associated with the ONLINE keyword is END which takes a snapshot at the end of the backup.

```
STORE @.mygroup.myaccount;*TAPE;ONLINE=END
```

In this case, the sequence of calls to ALLBASE/SQL routines remains the same except that the snapshot occurs after the backup. However, it is still guaranteed that when the restore is done, files will be the same as the time when the snapshot was taken. Files are therefore consistent.

This enhancement acts as an interface to True Online Backup and allows the DBEnvironment to be synchronized when many users are working.

Managing Log Files

The DBA must manage the size, number, and location of ALLBASE/SQL log files. This is done by performing the following tasks using a special set of SQLUtil log management commands:

- Monitoring the log with SHOWLOG
- Adding log files with ADDLOG
- Storing log files with STORELOG
- Rescuing log files with RESCUELOG
- Purging log files with PURGELOG
- Restoring log files with RESTORELOG
- Moving log files with MOVELOG

Monitoring the Log with SHOWLOG

Use the SHOWLOG command to display the names of the individual log files (single or dual) associated with a DBEnvironment, the archive mode of the log, and the available file space remaining.

Displaying Files in the Log

You can use the SHOWLOG command to display the log file directory either **dynamically** or **statically**. The dynamic SHOWLOG is done if you respond Y to the prompt

```
Connect? (y/n) (opt):
```

The default option is a dynamic SHOWLOG.

Normally, you should do a dynamic SHOWLOG so as to display the most current information about available log space and about the backup status of log files. You use the static SHOWLOG only when it is not possible to connect to the DBEnvironment, as after a media failure.

Here is an example of a dynamic SHOWLOG:

```
>> showlog
DBEnvironment Name: PARTSDBE.SOMEGRP.SOMEACCT
Maintenance Word: 
Connect? (y/n) (opt): y

Archive Mode: ON
Log Sequence Number Containing Most Recent Archive Checkpoint: 1
Current Log Sequence Number: 1
First Log Sequence Number Needed for Recovery: 1
Log Mode is: Single
Number of Free Blocks: 340

First Log Name: DBELOG1.SOMEGRP.SOMEACCT
First Log File Status: Useable
Log File Size: 250
Log Identifier Is: 1
Log Sequence Number: 1
Log Backup Status: Not Ready for Backup
```

```
First Log Name: DBELOG2.SOMEGRP.SOMEACCT
First Log File Status: Useable
Log File Size: 250
Log Identifier Is: 2
Log Sequence Number: 0
Log Backup Status: Not Ready for Backup
```

In this example, SHOWLOG displays the following information:

- Archive logging is on, and the log mode is single, meaning that only one set of files is maintained.
- The archive checkpoint—the point at which the last STOREONLINE command began storing the DBEnvironment—is in the file with log sequence number 1.
- The current log file is DBELOG1, which has a log sequence number of 1.
- If rollforward recovery should become necessary, it would have to start at log sequence number 1. After subsequent STOREONLINE commands, this sequence number will become larger.
- DBELOG1 and DBELOG2 combined still have a total of 340 free log blocks (pages).
- DBELOG1 has not yet been backed up. It is not yet ready for backup, since it is the current log file, that is, it is still being written to.
- A second log file, DBELOG2, has been added to the DBEnvironment with the ADDLOG command. This file has not been used, so it still has log sequence number 0. It has also not been backed up, since it does not contain any log records, and thus has never needed to be backed up.
- Note that since the DBEnvironment is in single logging mode, both files have only a *First Log Name*.

The following is an example of a dynamic SHOWLOG display for a DBEnvironment running in nonarchive mode with dual logging:

```
>> showlog
DBEnvironment Name: PARTSDBE.SOMEGRP.SOMEACCT
Maintenance Word: 
Connect? (y/n) (opt): y

Archive Mode: OFF
Log Sequence Number Containing Most Recent Archive Checkpoint: 0
Current Log Sequence Number: 1
First Log Sequence Number Needed for Recovery: 0
Log Mode is: Dual
Number of Free Block(s): 259

First Log Name: DBELOG1.SOMEGRP.SOMEACCT
First Log File Status: Useable
Second Log Name: DBELOG2.SOMEGRP.SOMEACCT
Second Log File Status: Useable

Log File Size: 300
Log Identifier Is: 1
Log Sequence Number: 1
Log Backup Status: Backup Is Not Required
```

In this example, the *Log Sequence Number Containing Most Recent Archive Checkpoint* and the *First Log Sequence Number Needed for Recovery* are both 0, since archive logging is off.

Log File Status Types

For a dynamic SHOWLOG, the types of backup status are as follows:

Not Ready for Backup	This is the current file, which is not full yet.
Ready for Backup	The file is full, and the DBE is now using a different file, so this one is ready to be stored with STORELOG.
Backup is Done	The file has already been backed up.
Backup is Not Required	The log file does not need to be backed up if the DBEnvironment is in nonarchive mode, or the DBEnvironment is in archive mode and this log file has never been used.

For a static SHOWLOG, the types of backup status are as follows:

Backup is Required	The file has not been backed up yet.
Backup is Done	The file has already been backed up.
Backup is Not Required	The DBEnvironment is in nonarchive mode, so the file does not need to be backed up.

Displaying Available File Space

You can also use the dynamic SHOWLOG command to display the number of free log blocks (pages) available for logging. In archive mode, SHOWLOG does not tell how many pages are available in previously used log files that have not already been backed up. Until the files are backed up, these log pages are not free.

Using the CHECKPOINT command

For nonarchive log files, a CHECKPOINT command tells you how many free log file pages there are, and it frees log file pages held by completed transactions. If long transactions tend to fill up the log file, you should increase the log file size.

For archive log files, a CHECKPOINT command also tells you how many free log file pages there are, and it frees log file pages in files that have been stored. CHECKPOINT does not tell how many pages are available in previously used log files that have not already been backed up. Moreover, CHECKPOINT uses more system resources than SHOWLOG, which is the preferred method for obtaining the number of free pages.

To free log file pages in archive mode, you must back up some of the log files using STORELOG. After backup, these files become available for reuse as soon as there are no active transactions in them. As the number of free log pages approaches zero, you should add log files, or use STORELOG to make a backup of any non-current log file, which then becomes available for reuse.

Adding Log Files with ADDLOG

Using multiple log files gives you the greatest flexibility with logging. In nonarchive mode, you can add a file when it is needed for unusually large transactions, and you can recover easily from a LOG FULL condition without having to shut the DBEnvironment down. In archive mode, you can achieve continuous DBEnvironment availability by adding a group of log files, then developing a schedule of DBEnvironment and log backup.

To expand the capacity of a nonarchive or archive log, use the ADDLOG command, as in the following example:

```
>> addlog
DBEnvironment Name: PARTSDBE.SOMEGRP.SOMEACCT
Maintenance Word:
Enter Log File Name(s) Separated by a Blank? DBELOG2.SOMEGRP.SOMEACCT
New Log File Size? 250
Add Log File (y/n)? y

Log file 'DBELOG2' was Added.
Log Identifier Is: 2
```

If you are using dual logging, enter two log file names separated by a blank. No more than two names are accepted. The minimum size you can specify for an added log file is 250 pages.

After you add the file to the DBEnvironment, it will be used when the current log file runs out of space.

Storing Log Files with STORELOG

If you are using archive logging, you must use the STORELOG command to create backups of each log file as it fills up. When a file's status (shown in the dynamic SHOWLOG display) is *Ready for Backup*, you should use the STORELOG command to store a copy of the log. One method of indicating which file to store is by entering its *Log Identifier*. Here is an example:

```
>> storelog
DBEnvironment Name: PartsDBE
Maintenance Word: Return
Log Identifier (opt): 2
To File name? Save1
Use Static Store (y/n)? n
Store Log File (y/n)? y

Log file 'lgn1' with Sequence Number      2  was stored.
```

The second method of initiating a log file backup is to issue a STORELOG command and press Return when you see the prompt for a log identifier. In this case, STORELOG will back up the file with the lowest possible sequence number that has *Ready for Backup* status in the dynamic SHOWLOG display. If the STORELOG operation succeeds, the file will be given the new backup status *Backup is Done*.

A third method of initiating a log file backup is to issue a STORELOG command and enter 0 for the log identifier. In this case, STORELOG will back up all the files which are ready for backup, in the proper sequence by log sequence number, prompting you for filenames as each file is stored. Using this method, all log files can be stored on one tape.

Important! For each successful STORELOG, label the backup tape with the log file name, sequence number, and the date and time of the backup. The sequence number will let you restore and recover log files in the correct order in the event that rollforward recovery is needed.

Rescuing Log Files with RESCUELOG

At certain times, it is necessary to store a copy of a log file that is not yet ready for backup with the STORELOG command. One such time is immediately following a media failure. In this event, STORELOG does not work, since it is impossible to CONNECT to the DBEnvironment. Use the RESCUELOG command to store a copy of any log files not yet backed up at the time of the media failure, as in the following:

```
>> rescuelog
Log File Name: DBELOG6
Size Of The Log File: 250
To File Name? save6
Rescue Log File (y/n)? y

Log File 'DBELOG6' with Sequence Number      12 Was Rescued.
```

If the DBECon file is intact, you can use the static SHOWLOG command after a media failure to display the sequence numbers of files with a status of *Backup Required*. Then you can store these files with RESCUELOG.

Restoring Log Files with RESTORELOG

After a media failure, you prepare for rollforward recovery by restoring the DBEnvironment and the log files. After using RESTORE to restore the DBECon file and all DBEFiles, use the RESTORELOG command to restore each individual log file. The *Rename* prompt lets you enter a new file name for the log. Each file is restored initially with its original name; then ALLBASE/SQL prompts for a new file name.

The easiest way to restore and apply log files is to use the RESTORELOG and the RECOVERLOG commands one after another for each file, as in the following example:

```
>> restorelog
DBEnvironment Name: PARTSDBE.SOMEGRP.SOMEACCT
Maintenance Word: 

Input Device: TAPE
Local (y/n) (opt): y
Restore the Log File (y/n)? y
Log File 'lgn1' was Restored.
Rename 'lgn1' Log File To: log0001
Log File 'lgn1' was Renamed to 'log0001'.

>> recoverlog
DBEnvironment Name: PARTSDBE.SOMEGRP.SOMEACCT
Maintenance Word: 
Next Log File To Recover: log0001
Recover Log File (y/n)? y

Log File Recovered.
Next Possible Log Sequence Number is      2.
```

When a log file has been applied to the DBEnvironment as a part of rollforward recovery, it will be purged by ALLBASE/SQL when it no longer contains active transactions. For that

reason, if doing partial recovery in which your current log files are undamaged, make sure that the names given restored log files while using the RESTORELOG command do not conflict with the names of existing log files. Otherwise your existing log files could be purged as RECOVERLOG completes its work.

(In fact, partial rollforward recovery is best conducted in a group which is separated from the one containing your DBEConFile, SYSTEM DBEFileSet, and log files to avoid undesired interaction between files with duplicate names.) Do not attempt to purge logs yourself during the rollforward recovery process.

Purging Log Files with PURGELOG

The only way to purge an individual log file that is no longer needed is with the PURGELOG command. Example:

```
>> purgelog
DBEnvironment Name: PARTSDBE.SOMEGRP.SOMEACCT
Maintenance Word:
Log Identifier: 2
Purge Log File (y/n)? y

Log file(s) Purged.
```

In the case of dual logging, this command purges both filenames associated with the log identifier.

You can purge a log file that has been backed up, provided it contains no active transactions. You can also purge a log file that has never been used. You cannot use PURGELOG during the rollforward recovery process. If you wish to purge the DBEnvironment and the entire log, use the PURGEALL command. To purge the DBE alone, use the PURGEDBE command.

Note You should *not* remove log files with the MPE/iX PURGE command, since this action is not recorded in the DBECon file. For the same reasons, do *not* use the PURGEFILE command to remove a log file under any circumstances.

Moving Log Files with MOVELOG

You can move a log file from one location to another with the SQLUtil MOVELOG command. Example:

```
>> movelog
DBEnvironment Name: PARTSDBE.SOMEGRP.SOMEACCT
Current Log File Name: DBELog1.SomeGrp.SomeAcct
Current Device: DISC
New Device: 2
New Log File Name: DBELog1.OtherGrp.SomeAcct

File moved.
```

MOVELOG lets you place log files on a device that is separate from the data and index files of the DBEnvironment. Note that when you move a log file, you specify its file name; this means that you can place the different members of a dual log sequence on separate volume sets if you choose.

Starting a New Log

Use the `START DBE NEWLOG` command to create a new log file with different characteristics as you open the DBEnvironment. You must be the DBECreator to use this command.

The `START DBE NEWLOG` command may be used for the following reasons:

- To change the name or size of the current log file.
- To enable or disable dual logging.
- To return to nonarchive logging mode from archive mode.

The following procedure is recommended when using `START DBE NEWLOG` to ensure the safety of log records and the consistency of the DBEnvironment:

1. Stop the DBEnvironment.
2. If in archive mode, back up log files as desired using `STORELOG` and `RESCUELOG`. (You don't need to back up the log files in nonarchive mode because you can not use them for rollforward recovery.)
3. Perform a `START DBE NEWLOG`. You should specify file names different from the old log names in the `NAME =` clause.

The following example shows how to start a new log file for the sample DBEnvironment, using different log file names:

```
isql=> START DBE 'PARTSDBE.SOMEGRP.SOMEACCT' MULTI NEWLOG
> DUAL LOG,
> TRANSACTION=7
> LOG DBEFILE PartsLg3 AND PartsLg4
> WITH PAGES = 250,
> NAME = 'DBELog3.Log1Grp.SomeAcct' AND 'DBELog4.Log2Grp.SomeAcct';
```

The physical files are created in groups different from that of the DBECon file, but the account is (and must be) the same as that of the DBECon file.

The new log file is in nonarchive mode. If you wish, you can start a new log in archive mode by using the `ARCHIVE LOG` option in the `START DBE NEWLOG` command.

Note If you change from archive mode logging to nonarchive mode logging, rollforward recovery will be impossible beyond the last archive log file.

Monitoring the Log with SQLMON

SQLMON is an online diagnostic tool that monitors the activity of an ALLBASE/SQL DBEnvironment. In addition to providing information about locking, I/O, and file capacity, SQLMON displays information about logging activity. SQLMON is documented fully in the *ALLBASE/SQL Performance and Monitoring Guidelines*. Table 6-1 summarizes the log monitoring tasks you can perform with SQLMON.

Table 6-1. SQLMON Log Monitoring Tasks

Task	SQLMON Screens	Fields
Determining Log File Capacity	Overview	LOG FULL % Used LgPgs Max LgPgs
Identifying Log Mode	Overview IO	Archive Mode
Detecting Logging Errors	Overview	LOG ERRORS
Identifying Size of Log Buffer Pool	IO	TOTAL LOG BUFFER PAGES
Monitoring Log Buffer I/O	IO IO Log Session IO Log Program	LOG BUFF WR LOG DISK RD LOG DISK WR
Monitoring Checkpoints	IO	CHECKPOINTS

Setting up a Wrapper DBEnvironment

Wrapper DBEnvironment functionality is used to recover the audit information in the log files orphaned when you cannot connect to a DBEnvironment. Wrapping log files means associating the files with a DBEnvironment. After a DBEnvironment becomes inaccessible, its log files are not associated with any DBEnvironment. These orphaned log files are then also inaccessible.

You can try to recover the audit information in the log files with the Audit Tool (the Audit Tool is described later in this manual). Audit information allows you to group database information into partitions for processing analyses. Access to wrapped log files avoids having a gap in the ongoing record of audit information. The use of archive logging facilitates wrapper DBEnvironment use, but nonarchive logging does not prevent use of wrapper DBEnvironments.

Note Recovery of the database itself is a separate operation. It is recommended that the log files be wrapped before you attempt a recovery. The different types of recovery are described earlier in this chapter.

The following list summarizes the tasks that must be performed to create a wrapper DBEnvironment:

1. Select usable log files.
2. Create a new DBEnvironment with `START DBE NEW`.
3. Wrap the DBEnvironment around the log files with `SQLUtil WRAPDBE` command.

These tasks are described in the following sections.

Selecting Valid Log Files with `SHOWLOG`

The first step in setting up a wrapper DBEnvironment is to select the names of log files marked Useable. Only the log files themselves will be wrapped, not the DBECon file. However, log file status information is contained in the DBECon file, *not* in the log files. Because the DBECon file will not be wrapped, you must manually determine which log files are valid and usable and enter this information when prompted by the `SQLUtil WRAPDBE` command in the last step.

If the DBECon file still exists, the `SQLUtil SHOWLOG` command can be used to display the log files associated with the inaccessible DBEnvironment. (Refer to the section “Selecting Log Files when the DBECon File is Inaccessible” for guidelines on selecting log files when the DBECon file is unavailable.) The `SHOWLOG` command has two modes: one that connects to the DBEnvironment and one that does not. Use the mode that does *not* connect to the DBEnvironment because connecting to the DBEnvironment may fail. No authority is needed for the `SHOWLOG` command.

Single Logs

For single logging, *all* log file names are selected. Suppose that the names for `PartsLog1a`, `PartsLog 2A`, and `PartsLog3A` are as follows:

```
PRTSLG1A
PRTSLG3A
PRTSLG2A
```

The sequence that the log files are entered is not important to the `SQLUtil WRAPDBE` command. When all log files have been entered, the `WRAPDBE` command issues a warning if a log file is missing in the sequence even though `WRAPDBE` still allows the DBEnvironment to be converted to a wrapper DBEnvironment.

If a log file in the sequence has been purged, you will only be able to retrieve audit log records as far back in history as the beginning of a log file with a log sequence number greater than the purged log file. For example, if `PartsLog2a` was missing, you would only be able to retrieve audit log records from `PartsLog3a`. You would not be able to retrieve `PartsLog1a`.

Dual Logs

Assume that a DBEnvironment using dual logs has a log file configuration shown in Table 6-2 below:

Table 6-2. Example Log File Names and Sequence Numbers

First Log	Second Log	Log Sequence #
PartsLog1a	PartsLog1b	2
PartsLog2a	PartsLog2b	3
PartsLog3a	PartsLog3b	4

During processing, an error occurred on PartsLog2a and the log file was marked Not Useable. Since PartsLog2b was still available and considered to be a valid log file, ALLBASE/SQL logging to PartsLog2a was discontinued and PartsLog2b became the only log. Therefore, you would want to select PartsLog2b.

A message appears on the console when the switch takes place. The message is also written to the console file which an operator can read later. This switch allows users to continue accessing the DBEnvironment.

The following example shows a sample SHOWLOG command display for a DBEnvironment with dual logging:

```
>> showlog
DBEnvironment Name: PartsDBE
Maintenance Word:
Connect (y/n)? (opt): n

Archive Mode: ON
Log Sequence Number Containing Most Recent Archive Checkpoint: 0
Current Log Sequence Number: 5
First Log Sequence Number Needed for Recovery: 0
Log Mode is: Dual
Number of Free Block(s): 522

First Log Name: PRTSLG1A
First Log File Status: Useable
Second Log Name: PRTSLG1B
Second Log File Status: Useable
Log File Size: 256
Log Identifier Is: 1
Log Sequence Number: 2
Log Backup Status: Ready For Backup

First Log Name: PRTSLG2A
First Log File Status: Not Useable
Second Log Name: PRTSLG2B
Second Log File Status: Useable

Log File Size: 256
Log Identifier Is: 2
Log Sequence Number: 3
Log Backup Status: Ready For Backup
```



```
First Log Name: PRTSLG3A
First Log File Status: Useable
Second Log Name: PRTSLG3B
Second Log File Status: Useable
Log File Size: 300
Log Identifier Is: 3
Log Sequence Number: 4
Log Backup Status: Not Ready For Backup
```

As shown, PRTSLG2A has a log file status of Not Useable, indicating that the log file should not be used in the wrapper DBEnvironment. PRTSLG2B should be used instead.

Selecting Log Files when the DBECon File is Inaccessible

If single logs are being used, you are safe in using the list of log files defined when the inaccessible DBEnvironment was defined. (Such a list should be made at the time of DBEnvironment creation.) This is considered safe because when a log file becomes unusable with no backup (or dual log) to switch to, work is not allowed on the DBEnvironment and the DBEnvironment will stop processing (but will not have become inaccessible).

However, if dual logs were used, check the console file to see which log files have been marked Not Useable. The safest log files to use are those named for Second Log File. One criterion for making a decision is to check the modification timestamp for the two files with the operating system `ls -l` command (HP-UX) or `listf, logfilename, 3` command for MPE/iX. If one of the files has a considerably earlier timestamp, the system may have automatically switched to the Second Log File in the past. Therefore, the First Log File would be incomplete and logging would have continued on the Second Log File.

(To avoid this uncertainty in the future, you can issue an occasional `SHOWLOG` command while the DBEnvironment is running to see whether a log file has a Not Useable status. A console message is also issued and written to the console file when a log file becomes unusable; you can go back and check the console file later. This at least provides a reference point for later decisions as to which files are usable in the event of a hard crash.)

Creating a DBEnvironment

The next step is to create a DBEnvironment that will be converted to a wrapper DBEnvironment. Creating this new DBEnvironment is accomplished with the `START DBE NEW` statement. The DBEnvironment should be created in the subdirectory (HP-UX) or group (MPE/iX) where the log files reside.

The maximum number of transactions should be set to the same value that was allowed on the original DBEnvironment. If unknown, this value can be obtained from the DBECon File with the `SQLUtil SHOWDBE` command if the DBECon file still exists. Otherwise, you must restore the database before issuing the `SHOWDBE` command.

Any file names specified in the `START DBE NEW` command should not be the same as the log file names to be wrapped. Since no updates will be allowed on the wrapper DBEnvironment after it has been converted, options such as the number of data buffer pages and log buffer pages are irrelevant; you can use the default values for these options. However, the DBEnvironment must be created with the `AUDIT NAME`, `DEFAULT PARTITION`, `MAXPARTITIONS`, and `DATA AUDIT ELEMENTS` (and `COMMENT PARTITION`, if present) the same as in the inaccessible DBEnvironment. Audit parameters are shaded in the example below.

Since this database is used only to retrieve audit log records with the Audit Tool, most DDL statements are also not needed. Therefore, a DBEFile0 larger than the default size is not required.

The following example shows such a DBEnvironment:

```
isql=> START DBE 'WRAPPER' MULTI NEW
        AUDIT LOG,
        TRANSACTION = 100,
        DBEFile0 DBEFILE wrapDBE1
          NAME = 'WrapDBE1',

        LOG DBEFile TempLog1
          NAME = 'TempLog1',
        AUDIT NAME = 'MyDBE1',
        DEFAULT PARTITION = 1,
        MAXPARTITIONS = 10,
        DATA AUDIT ELEMENTS;
```

Wrapping the DBEnvironment Around the Log Files

The final step actually converts the created DBEnvironment to a wrapper DBEnvironment, thus associating it with the set of log files from the inaccessible DBEnvironment. The SQLUtil WRAPDBE command is used to perform the conversion.

The SQLUtil WRAPDBE command must be used on log files that are inactive. If the WRAPDBE command is used on log files still associated with a DBEnvironment, it is possible that another user may be able to connect to the database (even though it had been thought to be inaccessible) and do work that would generate log records while the log files are associated with the wrapper DBEnvironment.

You must be a superuser (HP-UX) or system administrator (MPE/iX) or DBECreator of the original inaccessible DBEnvironment to wrap the created DBEnvironment around the log files. An example of using this command is shown below:

```
>> wrapdbe
DBEnvironment Name: WRAPPER
Maintenance Word: Return
Wrapper Mode (log) (opt): LOG
Enter Log File Name (RETURN to finish): PRTSLG1A
Enter Log File Name (RETURN to finish): PRTSLG2B
Enter Log File Name (RETURN to finish): PRTSLG3A
Enter Log File Name (RETURN to finish):
Convert to Wrapper DBEnvironment (y/n): y
```

The Maintenance Word should also be that of the DBEnvironment created in the third step. The Wrapper Mode has only one option, log, at this time. If a carriage return is entered, Wrapper Mode defaults to log.

When entering log file names, SQLUtil continues prompting for the next log file name until you enter a carriage return. The maximum number of log file names that can be entered is 34. Though the log file names can be entered in any order, the complete list should constitute a correct sequence.

The final prompt, Convert to Wrapper DBEnvironment (y/n), allows you to verify that the DBEnvironment should be converted into a wrapper DBEnvironment. Entering two slashes (//) at any time returns you to the SQLUtil prompt.

The WRAPDBE command opens the new DBEnvironment in single user mode to ensure that no one else is currently accessing the DBEnvironment.

No updates can be made to the DBEnvironment after it has been converted to a wrapper DBEnvironment but updates can be made before the WRAPDBE command is issued. The WRAPDBE command removes any log files associated with the DBEnvironment before being converted. Only the wrapped log files are associated with the DBEnvironment after it is wrapped around them.

After converting the DBEnvironment, the SQLUtil command SHOWDBE displays an additional line as follows to indicate that the DBEnvironment is a wrapper DBEnvironment:

```
DBEnvironment Type: WRAPPER
```

The full display of the SHOWDBE command for a wrapper DBEnvironment is shown at the end of the next section.

Example of Setting Up a Wrapper DBE

Assume that the inaccessible DBEnvironment had the following structural information displayed for the SQLUtil SHOWDBE command:

```
>> showdbe
DBEnvironment Name: PARTSDBE
Maintenance Word: Return
Output File Name (opt): Return
-> all

DBEnvironment Language: n-computer (HP-UX)
DBECreator ID: 170 (HP-UX)
DBEnvironment Language: NATIVE-3000 (MPE/iX)
AutoStart: ON
Audit Logging Is: ON

Audit Logging Name is: PartsDBE1
Default Partition ID is: 1
Maximum Number of Partitions Is: 10
Comment Partition ID Is: 2
Audit Elements are: CHKPT, DATA, CMWT

User Mode: MULTI
DBEFile0 Name: PartsFO
DDL Enabled: YES
No. of Runtime Control Block Pages: 128
No. of Data Buffer Pages: 200

Data Buffer Pages Memory Resident: NO (MPE/iX)
No. of Log Buffer Pages: 200
Max. Transactions: 100

Maximum Timeout: NONE
Default Timeout: MAXIMUM
Authorize Once per session: OFF
Console File Name: CONSOLE
```

Assume that the following information is displayed by SHOWLOG:

```
>> showlog
DBEnvironment Name: PARTSDBE
Maintenance Word:
Connect (y/n)? (opt): n
```

Archive Mode: ON
Log Sequence Number Containing Most Recent Archive Checkpoint: 0
Current Log Sequence Number: 5
First Log Sequence Number Needed for Recovery: 0
Log Mode is: Single
Number of Free Block(s): 522

First Log Name: PRTSLG1A
First Log File Status: Useable
Log File Size: 256
Log Identifier Is: 1
Log Sequence Number: 2
Log Backup Status: Ready For Backup

First Log Name: PRTSLG2A
First Log File Status: Useable
Log File Size: 256
Log Identifier Is: 2
Log Sequence Number: 3
Log Backup Status: Ready For Backup

First Log Name: PRTSLG3A
First Log File Status: Useable
Log File Size: 300
Log Identifier Is: 3
Log Sequence Number: 4
Log Backup Status: Not Ready For Backup

First Log Name: PRTSLG4A
First Log File Status: Useable
Log File Size: 300
Log Identifier Is: 4
Log Sequence Number: 0
Log Backup Status: Not Ready For Backup

Maintenance

Maintenance is a composite of activities that let you adjust the DBEnvironment to the changing needs of your system's users. The following are presented in this chapter:

- Using simple and complex maintenance operations
- Maintaining the DBEnvironment
- Maintaining tables
- Dropping and recreating hash structures
- Maintaining indexes
- Maintaining constraints
- Maintaining rules and procedures
- Maintaining sets of interrelated objects
- Maintaining applications
- Maintaining a nonstop production system
- Maintaining security
- Disabling data definition
- Judging maintenance expenses
- Cleaning up after abnormal termination

For most maintenance operations, you use the data definition statements in SQL. For some maintenance operations, you use SQLUtil and SQLGEN, which are described in the “DBA Tasks and Tools” chapter, or the ISQL LOAD and UNLOAD commands. Complete command syntax for SQLUtil and SQLGEN appears in the appendix of this manual. ISQL commands are described in the *ISQL Reference Manual for ALLBASE/SQL and IMAGE/SQL*.

To monitor DBEnvironment performance, you use SQLMON, which is described in the *ALLBASE/SQL Performance and Monitoring Guidelines*.

Using Simple and Complex Maintenance Operations

Most maintenance operations are either simple or complex. A **simple** operation requires only a single command and can be performed without unloading and reloading tables. The following commands perform simple operations:

- ALTER TABLE
- CREATE INDEX or DROP INDEX
- ALTER DBEFIL (from INDEX or TABLE to MIXED)
- MOVEFILE
- DROP or CREATE VIEW
- ADD TO GROUP

A **complex** operation requires a series of commands that remove data from an object before modifying the object in some way and then reloading it.

The following tasks require complex operations:

- Dropping a DBEFile
- Changing a DBEFile from MIXED to INDEX or TABLE
- Regrouping data to improve performance
- Merging two existing tables
- Splitting an existing table into two tables
- Merging two columns in a table
- Modifying a column in an existing table
- Reloading tables to improve index performance

A complex operation generally uses the UNLOAD and LOAD commands in combination with some other procedure, or else it uses the INSERT statement with a newly created table and a SELECT statement that specifies columns in an old table or tables. Many maintenance tasks can be performed using more than one kind of operation, so you must often decide which to use in a given case. The next few sections show how to use both simple and complex operations to carry out a variety of tasks in database maintenance.

Maintaining the DBEnvironment

DBEnvironment maintenance includes the following tasks:

- Adjusting startup values
- Setting parameters for rule operation
- Updating system catalog statistics
- Managing DBEFiles and DBEFileSets
- Managing log files (discussed in the “Backup and Recovery” chapter)

You can monitor the DBEnvironment using the system catalog. Then, based on the changing needs of users, you can adjust DBECon parameters or increase or decrease space in DBEFileSets and log files. Refer to the “System Catalog” chapter for a complete description of each catalog view and its use in monitoring system performance.

Adjusting Startup Values

The DBECon file contains startup parameters which help the DBA automate DBEnvironment startup and access procedures. Defaults for many of these values are set when you create the DBEnvironment using the START DBE NEW statement. You can change many startup parameters temporarily using the START DBE statement or permanently using the ALLBASE/SQL tool SQLUtil.

All DBECon parameters except the DBECreator, Maintenance Word, AutoStart, DDL Enabled, Memory Resident Data Buffer Flag, and Authorize Once Per Session parameters can be specified in the START DBE NEW statement when the DBEnvironment is configured. ALLBASE/SQL assigns defaults to DBECreator, Maintenance Word, AutoStart, DDL Enabled Flag, Archive Mode, Control Block pages, Log Buffer Pages, Data Buffer Pages, Memory Resident Data Buffer Flag, and Authorize Once Per Session Flag.

Table 7-1 shows all the parameters and how they can be modified.

Table 7-1. DBECon Parameters

Parameter	Default	Modify with	Description
Maintenance Word	None	SQLUtil SETDBEMAIN	is a password for SQLUtil. Defining a maintenance word protects the DBECon file from being modified by unauthorized users. The DBECreator can set and change the Maintenance Word only with SQLUtil.
DBEnvironment Language	NATIVE-3000	Cannot be changed	is specified in the LANG= option of the START DBE NEW statement. Once the DBEnvironment is configured, you cannot change its language.
DBECreator ID	DBEuserid of the user who configures a DBEnvironment by using START DBE NEW.	Cannot be changed.	Ensures that there is a DBEUserID that has irrevocable DBA authority for each DBEnvironment. In addition, the DBECreator always has access to the SQLUtil commands for DBEnvironment maintenance and security.
AutoStart	ON	SQLUtil ALTDDBE	automates DBEnvironment startup. You can set this parameter only with SQLUtil.
User Mode	SINGLE	SQLUtil ALTDDBE to change; START DBE or START DBE NEWLOG to override	is SINGLE to allow only one user, and MULTI to allow multiple users to access the DBEnvironment simultaneously. A DBA may want to configure the DBEnvironment in single-user mode until all databases have been created and all authorization has been granted. Once the DBEnvironment is ready for production use, user mode can be changed to MULTI. User mode can be changed using SQLUtil or temporarily overridden with the START DBE statement.
Log File Name(s)	DBELOG1, DBELOG2	START DBE NEWLOG to change	are specified in the LOG DBEFILE clause of the START DBE NEW statement. If DUAL LOG is specified, you must specify two log file names. The log file names and size can be changed with the START DBE NEWLOG statement discussed in the “Backup and Recovery” chapter of this guide.
Archive Mode	Disabled	SQLUtil STOREONLINE; ARCHIVE parameter in START DBE NEWLOG statement	STOREONLINE enables archive logging for rollforward recovery. The ARCHIVE parameter in the START DBE NEWLOG statement configures a new log for a DBEnvironment that is already using archive logging.

Table 7-1. DBECon Parameters (continued)

Parameter	Default	Modify with	Description
DDL Enabled	YES	SQLUtil ALTDBE	enables the use of data definition language (DDL). You can set this parameter with the SQLUtil ALTDBE command. Disabling DDL can improve performance significantly, but it should only be done if your applications do not do any data definition.
Number of Runtime Control Block Pages	37	SQLUtil ALTDBE to change; or SQL START DBE or START DBE NEWLOG to override	The number of blocks of memory allocated for DBCore services such as locking and session management. The maximum is 2000 pages.
Number of Data Buffer Pages	100	SQLUtil ALTDBE to change; START DBE or START DBE NEWLOG to override	are specified in the BUFFER option of the START DBE NEW statement. You should make sure that the buffers are large enough to accommodate the number of concurrent transactions in the DBEnvironment. Refer to “Estimating Shared Memory Requirements” in the “Physical Design” chapter for details on determining the number of buffer pages.
Data Buffer Pages Memory Resident	NO	SQLUtil ALTDBE	When set to YES, this ensures that data buffer pages will remain memory resident and not be swapped out by the operating system. The default is NO.
Number of Log Buffer Pages	24	SQLUtil ALTDBE to change; START DBE or START DBE NEWLOG to override	are specified in the BUFFER option of the START DBE NEW statement. You should make sure that the buffers are large enough to accommodate the number of concurrent transactions in the DBEnvironment. Log buffers can be from 24 (the default) to 1024 pages. Refer to “Estimating Shared Memory Requirements” in the “Physical Design” chapter for details on determining the number of buffer pages.
Maximum Transactions	50	SQLUtil ALTDBE to change; START DBE or START DBE NEWLOG to override	is specified in the TRANSACTIONS option of the START DBE NEW statement. The number of concurrent transactions depends on the number of users that will be concurrently accessing the DBEnvironment. The default number of transactions is 50; the maximum is 240.
Maximum Timeout	NONE	SQLUtil ALTDBE to change; START DBE or START DBE NEWLOG to override	is specified in the MAXIMUM TIMEOUT clause of the START DBE NEW statement. This value is the maximum permitted timeout that can be established by a user in the DBEnvironment.

Table 7-1. DBECon Parameters (continued)

Parameter	Default	Modify with	Description
Default Timeout	MAXIMUM	SQLUtil ALTDBE to change; START DBE or START DBE NEWLOG to override	is specified in the DEFAULT TIMEOUT clause of the START DBE NEW statement. This value is the default user timeout value in the DBEnvironment.
Authorize Once Per Session	OFF	SQLUtil ALTDBE	When ON, authorization to run modules or execute procedures is only done once per session, on the first invocation.

To change any of the DBECon parameters using SQLUtil, you must be the DBECreator (creator of the DBECon file) or know the maintenance word. To temporarily override any of the DBECon parameters using START DBE, you must have DBA authority. To override using START DBE NEWLOG, you must be the DBECreator.

The START DBE and START DBE NEWLOG statements temporarily override the startup parameters in the DBECon file for just the period that the DBEnvironment is open. Note that the DBECon file contains some startup parameters that can be modified *only* through SQLUtil: Maintenance Word, Autostart, Archive Mode, DDL Enabled, Memory Resident Data Buffer Flag, and Authorize Once per Session. Defaults are provided for all of these except the maintenance word. Refer to the chapter “DBEnvironment Configuration and Security,” and see the description of ALTDBE command in the “SQLUtil” appendix for details on changing startup parameters.

Determining Behavior of Rules in a DBEnvironment Session

You can use the following SQL statements to change the behavior of rules in a DBEnvironment session:

- ENABLE RULES and DISABLE RULES
- SET PRINTRULES ON or OFF

The effects of these statements are global throughout the DBEnvironment; that is, they affect all connected users. The ENABLE RULES and DISABLE RULES statements turn on and off the operation of rules in a DBEnvironment. Use DISABLE RULES to perform load operations when you do not wish rules to be activated, or for testing the operation of rules and procedures. Use the SET PRINTRULES ON statement to turn on display of the rule’s name as it fires. Use SET PRINTRULES OFF to stop the display of rule names.

By default, PRINTRULES is set OFF, and rule firing is enabled.

Updating System Catalog Statistics

The UPDATE STATISTICS statement is used to update the system catalog to reflect the current status of the DBEnvironment. The UPDATE STATISTICS statement operates on one table at a time for the table data, its indexes, the DBEFileSet containing the table, and all DBEFiles in the DBEFileSet.

Since ALLBASE/SQL uses data from the system catalog to optimize queries, the DBA should update system catalog statistics after any of the following:

- Numerous inserts, updates, or deletes
- Creating or dropping database objects
- Restructuring databases

The columns for the following system views are updated for the table specified in the UPDATE STATISTICS statement:

- The DBEFUPAGES column in the SYSTEM.DBEFILE view is updated for every DBEFile in the DBEFileSet where the table resides. It indicates the number of used pages in each DBEFile.
- The DBEFSUPAGES column in the SYSTEM.DBEFILESET view is updated for the DBEFileSet where the table resides. It indicates the number of used pages in the DBEFileSet.
- The AVGLEN column in the SYSTEM.COLUMN view is updated for all columns in the table. It indicates for each column the average length of the values in that column.
- The NPAGES and CCOUNT columns in the SYSTEM.INDEX and SYSTEM.CONSTRAINTINDEX views are updated for each index (including constraint indexes) created on the table. They indicate how many pages each index occupies, and how well the data is clustered for each index, respectively.
- The NPPAGES, AVGLEN, MAXLEN, NFULL, and NOVERFLOW columns in the SYSTEM.HASH view are updated if the table is a hash table. NPPAGES indicates how many primary pages are in use in the table; AVGLEN and MAXLEN indicate the average and maximum chain length, that is, the number of overflow pages it is necessary to traverse before finding a row. NFULL is the number of primary pages that are more than half full, and NOVERFLOW is the total number of overflow pages.
- The NPAGES, NROWS, AVGLEN, and USTIME columns in the SYSTEM.TABLE view are updated for that table. NPAGES indicates how many pages the data in the table occupies. NROWS indicates the number of rows in the table. AVGLEN specifies the average row length in the table. USTIME indicates the last time an UPDATE STATISTICS was performed on the table.

The table owner or a user with DBA authority can update statistics on a table. Users with DBA authority can update statistics on system views.

The following statement updates statistics for the PurchDB.Parts table:

```
isql=> UPDATE STATISTICS FOR TABLE PurchDB.Parts;
```

Note Updating statistics locks system catalog pages, and it also can invalidate sections stored for preprocessed statements. Since there can be an impact on performance while the statement is executing, you should use the UPDATE STATISTICS statement during off hours, or when accessing the DBEnvironment in single-user mode.

Changing System Table Lock Types

Two ALLBASE/SQL special names are supported as owners of the system base tables. STOREDSECT owns the tables used to store compiled sections and views (the **section tables**); HPRDBSS owns all other system tables. By issuing a query on the SYSTEM.TABLE view, you can see which system tables are owned by HPRDBSS and STOREDSECT. The RTYPE column indicates each table's lock type (granularity). (Refer to the "System Catalog" chapter in this manual for a complete description of the SYSTEM.TABLE view.)

If you are a DBA, you can change the lock type of any system base table or user table by means of the ALTER TABLE statement. A sophisticated understanding of locking strategy in general (and for the particular DBEnvironment) is required. For details, please refer to the *ALLBASE/SQL Reference Manual* chapter, "Concurrency Control through Locks and Isolation Levels."

As a DBA, you can use the UPDATE STATISTICS statement to insure that system and user table information is current. Then use SQLMON to detect concurrency problems and, if necessary, alter table types to change lock granularity. The SQLMON help Facility and the *ALLBASE/SQL Performance and Monitoring Guidelines* provide additional information.

Note Locking is a complex subject. It has far reaching effects on a DBEnvironment and possibly on system performance. For example, one possible effect of setting table lock types to PUBLICROW (row level locking) is that memory requirements may increase.

Managing DBEFiles and DBEFileSets

DBEFiles should be added to a DBEFileSet when you need more space in the DBEFileSet for the current tables and indexes or when you are going to create another table or index in the DBEFileSet.

Do the following before you add a new DBEFile:

- determine the available space in the DBEFileSet
- calculate the number of DBEFile pages needed
- determine the DBEFile type needed

Use SQLMON to monitor the space available in a DBEFileset. The Static DBEFile screen displays the number of pages in use and the maximum number of pages for each DBEFile and DBEFileset.

If your table and index data are separated into TABLE and INDEX DBEFiles, you must make sure that each type has enough room for pending inserts and updates. The Static Size screen in SQLMON lists the number of pages occupied by and the type (TABLE, INDEX, or MIXED) of each DBEFile.

Adding a New DBEFile

If you have determined that your DBEFileSet needs another DBEFile, use the following steps to create the additional DBEFile and add it to the DBEFileSet:

- Determine the DBEFile type. Use a TABLE DBEFile if you are adding rows to a table. Use both TABLE and INDEX DBEFiles if you are adding rows to a table with an index. Use MIXED DBEFiles if you are not separating table and index data.
- Use the formulas in the “Physical Design” chapter for calculating the number of DBEFile pages needed to store table and index data.
- Create the DBEFile using the CREATE DBEFILE statement. You can use the DEVICE clause to indicate the device on which the DBEFile will reside, as described in *ALLBASE/SQL Reference Manual*.
- Add the DBEFile to the DBEFileSet using the ADD DBEFILE statement.

Note If you are adding a DBEFile to the SYSTEM DBEFileSet, your transaction must be the only active transaction. If there are other active transactions, your transaction will wait until they complete.

- Make the changes permanent by using the COMMIT WORK statement.

Note If you use the ROLLBACK WORK statement, or if there is a system failure after you create the DBEFile and before you commit the transaction, a physical file will remain on the operating system without a corresponding entry in the system catalog. You should use the SQLUtil PURGEFILE command to remove this file before attempting to create the DBEFile again.

Changing DBEFile Type

DBEFiles are of type TABLE, INDEX, or MIXED. You can change a file from one to the other if necessary.

From TABLE or INDEX to MIXED

If you find you are not using indexes very often on some tables, you may want to consolidate the tables and indexes into DBEFiles of type MIXED. Mixed DBEFiles use space more efficiently than separate table and index DBEFiles. To change to type MIXED, you do not need to unload and empty the tables. You simply use the ALTER DBEFILE statement to change all DBEFiles to type MIXED. The SQL ALTER DBEFILE statement is a simple maintenance operation:

```
isql=> ALTER DBEFILE SomeDBEFile SET TYPE = MIXED;
```

From MIXED to TABLE or INDEX

To change DBEFile type from MIXED to either INDEX or TABLE, you must use a complex operation. For instance, if you want to separate table and index data that is currently stored in MIXED DBEFiles in order to place tables and indexes on different disk drives, use the ALTER DBEFILE statement in conjunction with UNLOAD and LOAD, as follows:

- Unload all tables in the DBEFileSet with the INTERNAL option. Before unloading, you may wish to drop indexes on the tables so as to improve performance during the unload operation. Do not drop the tables.
- Delete all rows from all tables in the DBEFileSet.
- Use the ALTER DBEFile statement to change some of the DBEFiles to type INDEX and others to type TABLE.
- Load the tables using the INTERNAL option. Recreate indexes if necessary.
- Use the SQLUtil MOVEFILE command to locate the DBEFiles on separate devices.

Keep in mind that you should know how much space is required for tables and indexes so that the appropriate number of DBEFiles are altered to type TABLE and to type INDEX.

Dropping a DBEFile

DBEFiles should be dropped when rows have been deleted from tables and space is no longer being used. The most significant implication of empty DBEFiles is wasted disk space. You may also experience slight performance degradation during serial table reads because all DBEFile pages are read during a serial table read.

Before a DBEFile can be removed from a DBEFileSet, it must be empty. To empty a DBEFile, you must drop all tables associated with the DBEFileSet that have data in them. Alternatively, you can delete all the rows in the table without dropping the table itself. This preserves the table definition, but has the drawback of requiring enough log space for all the data being deleted. If you want to preserve the data, you must unload the tables before deleting the rows. A DBEFile must be removed from the DBEFileSet with the REMOVE statement, as follows:

```
isql=> REMOVE DBEFILE WareDataF1 FROM DBEFILESET WarehFS;
```

Note	If you are removing a DBEFile from the SYSTEM DBEFileSet, your transaction must be the only active transaction. If there are other active transactions, your transaction will wait until they complete.
-------------	---

A DBEFile is dropped with the DROP DBEFILE statement. The following example drops the WareDataF1 DBEFile:

```
isql=> DROP DBEFILE WareDataF1;
```

Once you drop the DBEFile, it cannot be used to store data. However, it still resides on the system as an MPE/iX file.

Maintaining Tables

Table maintenance can involve the following:

- changing a table's locking behavior
- dropping tables
- adding and deleting columns
- "emptying" a table by removing its rows
- merging a table with another table
- dividing a table into two tables
- renaming tables or columns

Note If your tables use integrity constraints, rules, and procedures, the operations described in the next few sections can have unexpected consequences. For more information, see the section "Maintaining Sets of Interrelated Objects" later in this chapter.

Changing a Table's Locking Behavior

You can use the ALTER TABLE statement to change a table's type. You can also optionally reset the implicit grant of authority to the special user PUBLIC for tables that were created PUBLIC, PUBLICROW, or PUBLICREAD. When you use the ALTER TABLE statement, you change the table type permanently. To change it back to its original state, you must issue another ALTER TABLE statement. (To change locking behavior for the duration of a transaction, use the LOCK TABLE statement.)

Dropping Tables

Dropping a table from a database is a simple operation. Simply use the DROP TABLE statement:

```
isql=> DROP TABLE PurchDB.Parts;
```

Dropping tables has the following effects in addition to the deletion of the table itself:

- All indexes defined on the table are dropped.
- All views defined on the table, including views on multiple tables of which the table is one, are dropped.
- All authorities granted to users on the table and associated views are revoked.
- All rows in the system catalog pertaining to the table and its columns, indexes, rules, views, constraints, and authorizations are deleted.
- All sections that reference the table are invalidated.

You cannot drop a table that was defined with a primary key if another table references it. You must first drop the referencing table or else drop the constraint. Dropping views invalidates all sections that reference those views; dropping indexes invalidates sections that reference the tables on which the indexes were created.

When tables and indexes are dropped, the pages that held the data for those objects are not free to store other data until the transaction is terminated with a COMMIT WORK or a ROLLBACK WORK statement.

If you want to delete all of the rows in a table, but want to maintain the table definition, issue the TRUNCATE TABLE statement:

```
isql=> TRUNCATE TABLE PurchDB.Parts;
```

Adding Columns

Adding columns is a simple maintenance operation that does not require removing the existing data from a table. To add one or more columns to a table, use the ALTER TABLE statement. Refer to the *ALLBASE/SQL Reference Manual* for syntax. Adding columns has the following effects:

- Adding a column to a table will invalidate stored sections of applications that access the table. In addition, applications that access the table must be updated and re-preprocessed if they are to reference the new column.
- Applications that use a star (*) in their SELECT statement will need to be revised so that their host variable declarations will accommodate the new columns. They will also need to be re-preprocessed.
- If you have created a view on the table using a star (*) in the SELECT clause, the new columns will *not* appear in the view. Views that use the table as a base table must be deleted and recreated if they are to reference the new columns.
- A column added with the ALTER TABLE statement always contains either null values or default values in existing rows; therefore, a column cannot be added with the NOT NULL attribute unless you specify a default value.

The following statement adds an integer column called NewColumn1 and a character column called NewColumn2 to the PurchDB.Parts table:

```
isql=> ALTER TABLE PurchDB.Parts  
> ADD (NewColumn1 INTEGER, NewColumn2 CHAR(50));
```

Consider creating an index on one or more of the new columns if they are going to be used in query predicates. Use the UPDATE statement to insert values into the columns in existing rows.

Optionally, a data integrity constraint can be specified while adding a column in the ALTER TABLE statement:

```
isql=> ALTER TABLE RecDB.Clubs  
> ADD President CHAR(40)  
> REFERENCES RecDB.Members (MemberName);
```

Deleting Columns

You can delete columns from a table with either complex or simple operations:

- Using UNLOAD and LOAD (complex operation)
- Using an INSERT INTO statement (complex operation)
- Using a view (simple operation)

The following example shows how to delete columns from a table using the UNLOAD and LOAD commands:

- OldTable currently has five columns. ColumnTwo and ColumnFour are to be deleted. OldTable is unloaded with the following command:

```
isql=> UNLOAD TO INTERNAL SomeFile  
> FROM "SELECT ColumnOne, ColumnThree, ColumnFive  
> FROM OldTable  
> ORDER BY ColumnOne, ColumnThree, ColumnFive";
```

Note the ORDER BY clause, which orders the columns by index key to help clustering when the table is loaded.

- OldTable is dropped with the following statement:

```
isql=> DROP TABLE OldTable;  
isql=> COMMIT WORK;
```

- NewTable is created with the following statement:

```
isql=> CREATE TABLE NewTable (ColumnOne INTEGER,  
> ColumnThree INTEGER, ColumnFive INTEGER) IN SomeDBEFileSet;  
isql=> COMMIT WORK;
```

- NewTable is loaded with the following command:

```
isql=> LOAD FROM INTERNAL SomeFile TO NewTable;  
isql=> COMMIT WORK;
```

Remember, when a table is dropped all associated views and indexes are dropped and sections referencing the table are invalidated.

Columns can also be deleted using the form of the INSERT statement that uses a SELECT statement. The steps are as follows:

- Create the new table:

```
isql=> CREATE TABLE NewTable (ColumnOne INTEGER, ColumnThree INTEGER,  
> ColumnFive INTEGER) IN SomeDBEFileSet;  
isql=> COMMIT WORK;
```

- Insert the data from the old table:

```
isql=> INSERT INTO NewTable SELECT ColumnOne,  
> ColumnThree, ColumnFive FROM OldTable;  
isql=> COMMIT WORK;
```

- Drop the old table:

```
isql=> DROP TABLE OldTable;  
isql=> COMMIT WORK;
```

Removing Rows from a Table

You can use TRUNCATE TABLE to delete all the rows of a table, leaving the table's structure intact. You may, for example, wish to remove all the data from an old table and then reload the table with similar, new data.

The table definition is not removed or modified. All indexes, views, constraints, rules, default values, and authorizations defined for the table are unchanged.

Before you use TRUNCATE TABLE, be sure that the DDL (data definition language) flag is set to YES. If it is not, use the ALTDBE command (SQLUTIL) to set it.

For example, to delete all the rows from the table PurchDB.Parts, you would enter these statements:

```
isql=> TRUNCATE TABLE PurchDB.Parts;  
isql=> COMMIT WORK;
```

You can then reload PurchDB.Parts with the following command:

```
isql=> LOAD FROM INTERNAL SomeFile TO PurchDB.Parts;  
isql=> COMMIT WORK;
```

For more information on the TRUNCATE TABLE statement, refer to *ALLBASE/SQL Reference Manual*.

Merging Tables

You can merge columns in a complex operation. The process is much the same as for deletion, as shown in the above example. Unload the table with a SELECT statement. Drop the old table, and create and load a new one with the desired column structure. However, if you are changing the column name, size, or data type, you must unload and load using the EXTERNAL option.

The following SELECT specifies a join operation for an UNLOAD:

```
isql=> UNLOAD TO INTERNAL SomeFile  
> FROM "SELECT ColumnOne, ColumnTwo, ColumnB, ColumnC  
> FROM TableOne, TableTwo  
> WHERE TableOne.ColumnOne = TableTwo.ColumnA";
```

If you have applications that use TableOne and TableTwo, you may not want to drop them because the applications would have to be modified and re-preprocessed. The same table merge can be accomplished without re-preprocessing by creating a view that joins the two tables:

```
isql=> CREATE VIEW NewTable (ColumnOne,ColumnTwo,  
> ColumnThree,ColumnFour)  
> AS SELECT ColumnOne, ColumnTwo, ColumnB, ColumnC  
> FROM TableOne, TableTwo  
> WHERE TableOne.ColumnOne = TableTwo.ColumnA;  
isql=> COMMIT WORK;
```

The form of the INSERT statement that uses a SELECT statement can also be used to merge two tables. Use the second format of the INSERT statement to merge tables as follows:

```
isql=> INSERT INTO NewTable SELECT ColumnOne, ColumnTwo,  
> ColumnB, ColumnC FROM TableOne, TableTwo  
> WHERE TableOne.ColumnOne = TableTwo.ColumnA;  
isql=> COMMIT WORK;
```

After creating the new table, you can drop the old one.

After you merge tables in the manner just described, application programs accessing the old tables will have to be modified and re-preprocessed to access the new table if the old tables are dropped. As an alternative, you can create two new views, each of which would have the name and column description of one of the original tables. After creating these views, the old application programs would work as before after their sections are revalidated.

Dividing Tables

To divide a table into two tables, you must use a complex operation. For example, you can unload the tables using two separate UNLOAD commands with appropriate SELECT statements in the FROM clause. The following scenario divides OldTable into two new tables:

- OldTable consists of six columns. It will be divided into TableOne and TableTwo, each of which will consist of three columns. The following UNLOAD command captures data for TableOne:

```
isql=> UNLOAD TO EXTERNAL SomeFile1
> FROM "SELECT ColumnOne, ColumnTwo, ColumnThree
> FROM OldTable";
```

- The following UNLOAD command captures data for TableTwo:

```
isql=> UNLOAD TO EXTERNAL SomeFile2
> FROM "SELECT ColumnFour, ColumnFive, ColumnSix
> FROM OldTable";
```

- Next, you can drop OldTable and create a view with the characteristics of OldTable (including name) to avoid the necessity of re-coding applications that access OldTable.

- TableOne and TableTwo are created with the following statements:

```
isql=> CREATE TABLE TableOne (ColumnOne INTEGER,
> ColumnTwo INTEGER, ColumnThree INTEGER) IN SomeDBEFileSet;
isql=> CREATE TABLE TableTwo (ColumnFour INTEGER,
> ColumnFive INTEGER, ColumnSix INTEGER) IN SomeDBEFileSet;
isql=> COMMIT WORK;
```

Use the LOAD command to load the tables from the proper files. As long as the column sizes and data types of the old table are compatible with those of the new table, you can use the INTERNAL option of the UNLOAD command. If you are going to use an incompatible data type or size, you must unload and reload the data with the EXTERNAL option. For more information on the LOAD and UNLOAD commands, refer to the *ISQL Reference Manual for ALLBASE/SQL and IMAGE/SQL*.

Whenever tables are modified, the sections that reference them are marked as invalid. To avoid having to re-preprocess each program that accesses a modified table, you can create views that simulate the original tables. If the views have the same owner, name, and column structure, the sections will remain valid.

Note If you divide a table in two and then create a view that has the same column definition as the original table, you *cannot* use the view to update both underlying tables. Instead, updates must be done on the individual tables.

Renaming Tables or Columns

You can rename tables or columns using the RENAME TABLE or RENAME COLUMN statements. All indexes, columns, default columns, constraints, referential authorization, rules, and user authorities tables dependent on a renamed table or column will be renamed. All views dependent on a renamed table or column will be dropped. A RENAME statement is not allowed for IMAGE/SQL tables or tables with check constraints.

Dropping and Recreating Hash Structures

You may wish to modify a hash structure for any of the reasons you need to modify an ordinary table. In addition, you may need to recreate the hash structure when it is no longer large enough for the data in the table.

The SQLMON Static Hash screen contains information about each hash structure you have defined. By examining the `OVERFLOW CHAIN LENG`, `AVGOVERFLOW`, and `MAXOVERFLOW` fields on the Static Hash screen, you can determine the efficiency of hash structures.

The `AVGOVERFLOW` value is the average number of page accesses required to retrieve a particular row. `MAXOVERFLOW` is the maximum number of accesses; therefore, it is a worst case value. For most hash structures, the normal value of `AVGOVERFLOW` is between 1.05 and 1.10. This means that approximately 5 to 10% of the primary pages have one overflow page. A value of 1.5 means that half the primary pages have one overflow page. Use this information to decide whether unloading and reloading the hash structure might improve performance.

To alter the hash structure, simply unload the data, then drop the table and recreate it with the desired modifications, including, if necessary, a new number of primary pages. If you add to the number of primary pages, make sure there are enough empty `TABLE` or `MIXED DBEFiles` in the appropriate `DBEFileSet` with the required pages available.

Maintaining Indexes

From time to time, an index becomes inefficient and needs to be modified through maintenance operations. You can maintain indexes of different kinds by using both simple and complex operations.

Monitoring Index Space

Indexes on tables that have undergone multiple updates and deletes may use unnecessary space. As rows are added to indexes, the indexes increase in number of pages and number of levels. As those rows are updated and (especially) deleted, the pages become sparsely populated, reducing the amount of used space per index page. To compress the indexes, delete them and recreate them. Remember, to make the most of clustering indexes, the table must be unloaded and sorted on the index, and then reloaded after the index is deleted and created again. To determine the amount of space used by an index, check the `INDEX PAGES` field on the Static Size screen in SQLMON.

Tables that make heavy use of indexes should be located on a different disk drive than their indexes. In order to be able to do this you must have the table and index data stored in separate `DBEFiles`. This is most easily accomplished using `DBEFile` types of `INDEX` and `TABLE` to restrict the contents of the `DBEFiles`.

Tables that are used together frequently should be located on separate drives. Again, the tables must be stored in separate `DBEFiles` for this to be possible. The easiest way to ensure that tables are stored in separate `DBEFiles` is to create them in separate `DBEFileSets`. Then you can use `MOVEFILE` to move all `DBEFiles` in one `DBEFileSet` to a different device.

Monitoring the Cluster Count

The **cluster count** of an index indicates how many times ALLBASE/SQL has to access a different data page to retrieve the next row during an index scan. The greater the cluster count, the greater the potential for I/O when a query that uses that index is processed.

The CCOUNT column in the SYSTEM.INDEX view shows the cluster count of each index at the time of the last UPDATE STATISTICS statement. For a given data page, when the value for each key is different from all previous values on the page, the cluster count is incremented for the index. After a significant amount of table modification, the cluster count of an index may increase, which can hinder performance.

The CCOUNT is used by ALLBASE/SQL to determine whether or not to use an index to retrieve data. The higher the CCOUNT, the less likely ALLBASE/SQL will perform the query using an index scan. It will instead perform the query using a serial scan or a different index.

Although indexes are dynamically updated, information in the system views is not. It is important to execute the UPDATE STATISTICS statement after significant modifications have been made to any table or index in order to update the information, such as CCOUNT, used by ALLBASE/SQL.

To monitor the cluster count with SQLMON, invoke the Static Cluster screen and compare the value of the CCOUNT field with the values of the TABLE PAGES and TOT TUPLES fields. Performance is best when CCOUNT is equal to TABLE PAGES. As CCOUNT approaches TOT TUPLES, performance degrades.

The cluster count of an index is one measure of its efficiency. All indexes, clustering and nonclustering, have a cluster count. For a discussion of the differences between clustering and nonclustering indexes, refer to the “Logical Design” chapter.

Dropping and Recreating an Index

Indexes are dynamically updated when INSERT, UPDATE, or DELETE operations are performed on a table. To minimize overhead during large inserts, updates, or deletes, you can delete indexes with large keys and recreate them after the changes are made. This depends, of course, on table size versus insert, update, or delete size. For example, if you have one million rows in the table and you are updating one hundred of them, deleting the indexes will cause more overhead than leaving them intact.

As rows are inserted into a table, the indexes expand in number of pages and in number of levels. As rows are updated or deleted, the number of index pages may decrease, but the number of levels remains the same. During deletes, the number of occupied index pages will decrease as pages are emptied, but those pages not completely emptied will be sparsely populated causing an unnecessarily high number of index levels. For performance reasons and storage efficiency, you should drop and recreate indexes after multiple updates and deletes to a table.

Temporary indexes can be used by applications to improve performance. Applications that run on a periodic basis (once a month, for example) can create indexes to use while they are running and drop the indexes before terminating. (The DDL Enabled flag in the DBECon file must be set to YES in order to create and drop temporary indexes.) This way application data access is optimized at run time and overhead is minimized for updates when the application is not being run.

In all cases, you need to weigh the size of the table against the amount of work that is required to drop and recreate the index. In general, if your tables are extremely large, you should only drop and create indexes if you are experiencing poor performance or if you need to load a large amount of data. For smaller tables, you may want to recreate indexes periodically to ensure optimal performance.

Reloading a Table to Improve Index Performance

Another approach to index maintenance is to unload and reload tables in key order, which can improve performance. Use the following steps:

- Unload the table using the `INTERNAL` option. Use a `SELECT` with an `ORDER BY` on the index key in the `UNLOAD` command.
- Delete all rows in the table by omitting the `WHERE` clause in the `DELETE` statement.
- Drop all indexes on the table.
- Load the table using the `INTERNAL` option.
- Recreate the indexes.
- Do an `UPDATE STATISTICS` on the table to update system catalog information about the table and the index.
- `COMMIT WORK`.

Altering the Index Key

You can change an index key at any time without disturbing the table data by dropping the old index and creating a new index with a different key. However, when you change a clustering index key, the rows already in the table are not rearranged for clustering. New rows inserted into the table after the new index is created are clustered according to the new key. To change the key in a clustering index and rearrange the rows currently in the table, you must follow the steps illustrated above. In step one, be sure to use an `ORDER BY` clause that reflects the new clustering key.

Maintaining Constraints

During the life of a database, you may add or drop constraints on tables.

Adding Constraints

To add one or more constraints use the `ALTER TABLE` statement on an existing table. Refer to the *ALLBASE/SQL Reference Manual* for syntax. Adding a constraint may require the following considerations:

- You may want to later control the level at which constraint errors are checked with the `SET CONSTRAINTS` statement.
- You may need to add constraints to tables or columns.
- You should name the constraint for easy reference in case you later need to drop it.

The following statement adds a constraint to table `PurchDB.Parts`:

```
isql=> ALTER TABLE PurchDB.Vendors
> ADD CONSTRAINT CHECK (VendorNumber > 0) CONSTRAINT VndNum;
```

The added check constraint named VndNum ensures that PartNumber will be greater than zero.

Dropping Constraints

To drop one or more constraints, use the ALTER TABLE statement on an existing table. Refer to the *ALLBASE/SQL Reference Manual* for syntax. Dropping a constraint requires the following considerations:

- In order to drop a constraint, you must know its name. The *ConstraintID* is the name you optionally gave the constraint when it was defined. If you did not name the constraint, it has a system-defined name. Table and view constraint names are stored in SYSTEM.CONSTRAINT.
- You cannot drop a unique or primary key constraint if there exists a referential constraint referring to that unique or primary key. The referential constraint must be dropped first.

The following statement drops a constraint:

```
isql=> ALTER TABLE PurchDB.Vendors DROP CONSTRAINT VndNum;
```

The constraint named VndNum that was added above is dropped from table PurchDB.Parts.

Maintaining Rules and Procedures

The following tasks are used for the maintenance of rules and procedures in a DBEnvironment:

- Granting and Revoking Procedure Authorities
- Examining the Inventory of Rules and Procedures
- Dropping and Recreating Rules and Procedures
- Validating Procedures and Procedure Sections

Granting and Revoking Procedure Authorities

You must grant EXECUTE authority to users who need to execute procedures with the EXECUTE PROCEDURE statement. The creator of a rule that executes a procedure must also have EXECUTE authority for the procedure at the time the CREATE RULE statement is issued. The user of a statement that fires the procedure through a rule need not have EXECUTE authority for the procedure; EXECUTE authority is implicit in tying the procedure to an INSERT, UPDATE, or DELETE, for which the user must have authority.

The following example grants EXECUTE authority to the Managers group:

```
isql=> grant execute on PurchDB.ReportMonitor to Managers;
```

You can use the REVOKE statement to remove EXECUTE authority, as in the following:

```
isql=> revoke execute on PurchDB.ReportMonitor from Managers;
```

In addition, when you create a procedure, you can specify the IN *DBEFileSetName* clause to indicate where the procedure's sections are to be stored. Any user storing sections in the

specified DBFileSet must have been granted SECTIONSPACE authority to do so, as in the following grant to a group:

```
isql=> grant sectionspace on dbfileset DBFileSet1 to Group1;
```

You can use the REVOKE statement to remove SECTIONSPACE authority, as in the following:

```
isql=> revoke sectionspace on dbfileset DBFileSet1 from Group1;
```

Examining the Inventory of Rules and Procedures

Rules, procedures, and their definitions are stored in the system catalog. You can display the current list of procedures with the following query:

```
isql=> select * from system.procedure;  
isql=>
```

Use a query like the following to display the definition of a particular procedure (use the real procedure name in place of PROCNAME and the real owner name in place of OWNER):

```
isql=> select segnum, definestring  
> from system.proceduredef  
> where name = 'PROCNAME' and  
> owner = 'OWNER' order by segnum;
```

To display the current list of rules in the DBEnvironment, use the following query:

```
isql=> select * from system.rule;
```

To display the definition of a particular rule, use the following query (use the real rule name in place of RULENAME and the real owner name in place of OWNER):

```
isql=> select segnum, rulestring  
> from system.ruledef  
> where rulename = 'RULENAME' and  
> owner = 'OWNER' order by segnum;  
isql=>
```

Similar queries can be performed on the SYSTEM.RULECOLUMN, SYSTEM.PARAMETER, SYSTEM.PARAMDEFAULT, and SYSTEM.PROCAUTH views to obtain more detailed information.

Dropping and Recreating Rules and Procedures

As business rules change, the rules and procedures defined in the DBEnvironment can be modified. Use the DROP RULE statement to remove a rule that is no longer needed, and use the DROP PROCEDURE statement to remove a procedure that is no longer needed. If you use the PRESERVE option with the DROP PROCEDURE statement, the EXECUTE authorities associated with the procedure remain in the system catalog. Rules that invoke a particular procedure are *not* dropped when the procedure is dropped. However, stored sections that depend on rules which invoke the procedure are marked invalid when you drop the procedure. Creating or dropping rules has the effect of invalidating all sections that depend on the table on which the rule is based.

Validating Procedure Sections

When you create a procedure, a section is created for each SQL statement in the procedure *except*:

BEGIN WORK	OPEN CURSOR	WHENEVER
CLOSE CURSOR	ROLLBACK WORK	
COMMIT WORK	SAVEPOINT	

When procedure sections become invalid, ALLBASE/SQL will attempt to revalidate each section as it executes. You can also use the VALIDATE statement to revalidate all the sections in a procedure at one time. When a procedure is dropped, recreated, and reinvoked, ALLBASE/SQL must revalidate any invalid sections that execute the procedure or invalid sections containing rules that may invoke the procedure.

Maintaining Sets of Interrelated Objects

If your databases contain tables that are interrelated through rules and procedures or integrity constraints, the job of maintaining them may be more difficult. As you begin to restructure a database, be sure to study the implications of loading and unloading, dropping and recreating objects that have relationships to other objects. The following are a few common situations that can lead to unexpected results:

- Dropping a unique constraint also drops the B-tree index that supports the constraint and deprives the optimizer of one access path to the data in the table.
- Attempting to drop a table that is referenced by another table results in an error. You must drop the constraint that links the two tables, or else drop the referencing table first. To drop the constraint, use the DROP CONSTRAINT clause in the ALTER TABLE statement, specifying the referencing table.
- Dropping a table results in dropping of constraints and rules built on it. If you are restructuring, and you wish to drop and recreate the table, you must also recreate the rules and constraints.
- Dropping a procedure does *not* drop rules that invoke the procedure, but it does invalidate stored sections that depend on invoking the procedure from a rule. The loss of the procedure will be reported as an error when there is an attempt to revalidate those sections. If you wish to drop rules associated with a particular procedure, you should do so explicitly with DROP RULE statements.
- A complex chaining set of rules and procedures will be disrupted by the dropping of any rule or procedure in the chain. If you are using rule chaining, it is your responsibility to control the inventory of rules and procedures carefully.

Maintaining Applications

The DBA is involved in maintaining applications as follows:

- Monitoring changes that invalidate sections and determining when to re-preprocess or update applications based on those changes.
- Monitoring system catalog space for modules.
- Maintaining module related authorities.
- Sharing modules between DBEnvironments.
- Dropping modules.

Invalidation and Revalidation of Sections

Before changes are made to a DBEnvironment, the impact of those changes on preprocessed statements should be weighed. The following information will help you be aware of which changes affect preprocessed statements.

In full preprocessing mode, the preprocessor stores a section for each embedded statement *except*:

BEGIN DECLARE SECTION	BEGIN WORK	CLOSE CURSOR
COMMIT WORK	CONNECT	CREATE SCHEMA
END DECLARE SECTION	EXECUTE	EXECUTE IMMEDIATE
INCLUDE	OPEN CURSOR	PREPARE
RELEASE	ROLLBACK WORK	SAVEPOINT
START DBE	STOP DBE	TERMINATE USER

The statements listed above either require no authorization to execute or are executed based on information contained in the compilable preprocessor output files.

In interactive mode, ALLBASE/SQL stores a section for the following SQL statements:

```
PREPARE
CREATE VIEW
```

When a section is stored, ALLBASE/SQL actually stores what are known as an input tree and a run tree. The **input tree** consists of the uncompiled statement. The **run tree** is the compiled, optimized, executable form of the statement. If a section is **valid** at run time, ALLBASE/SQL executes the appropriate run tree when the SQL statement is encountered in the application program or procedure. If a section is **invalid**, ALLBASE/SQL determines whether the objects referenced in the sections exist and whether current authorization criteria are satisfied. If an invalid section can be validated, ALLBASE/SQL dynamically recompiles the input tree to create an executable run tree and executes the statement. If a section cannot be validated, the statement is not executed, and an error condition is returned to the program.

Information in the System Catalog on Validity of Sections

The SYSTEM.SECTION view contains information about stored sections. The TYPE column defines the type of SQL statement in the section:

- A section for executing the SELECT statement associated with a DECLARE CURSOR statement is identified by a 1 in the TYPE column.
- A section for executing the SELECT statement associated with a CREATE VIEW statement is identified by a 2 in the TYPE column.

- Sections for all other statements for which the preprocessor stores a section are identified by a 0 in the TYPE column.

The STYPE column defines the section type:

- A section that is part of a module is identified by a 0 in the STYPE column.
- A section that is part of a procedure is identified by a 1 in the STYPE column.

The VALID column tells whether the section is valid or invalid. If a section is marked invalid, it is identified by a 0 in the VALID column. If a section is valid, it is identified by a 1. Refer to the “System Catalog” chapter for a description of all the columns in SYSTEM.SECTION.

The example below illustrates the kind of information in the SYSTEM.SECTION view:

```
SELECT Name,Owner,Section,Type,Stype,Valid FROM System.Section;
```

NAME	OWNER	SECTION	TYPE	STYPE	VALID	
TABLE	SYSTEM		0	2	0	1
COLUMN	SYSTEM		0	2	0	1
INDEX	SYSTEM		0	2	0	1
SECTION	SYSTEM		0	2	0	1
DBEFILESET	SYSTEM		0	2	0	1
DBEFILE	SYSTEM		0	2	0	1
SPECAUTH	SYSTEM		0	2	0	1
TABAUTH	SYSTEM		0	2	0	1
COLAUTH	SYSTEM		0	2	0	1
MODAUTH	SYSTEM		0	0	0	1
GROUP	SYSTEM		0	2	0	1
PARTINFO	PURCHDB		0	2	0	0
VENDORSTATISTICS	PURCHDB		0	2	0	1
CEXP11	KAREN@RIZZO		1	1	0	1
CEXP11	KAREN@RIZZO		2	0	0	1
FAILURELIST	MANUFDB		1	0	1	1

Number of rows selected is 16.

U[p], d[own], l[eft], r[ight], t[op], b[ottom], pr[int] <n>,or e[nd]>

The first eleven rows in this query result describe some of the sections stored for the system views. The next two rows describe two views in the sample database: PurchDB.PartInfo and PurchDB.VendorStatistics. Views are always stored as invalid sections, because the run tree is always generated at run time when the view is queried.

The remaining rows describe sections associated with preprocessed programs and procedures. Module CEXP11 contains two sections, one for executing the SELECT statement associated with a DECLARE CURSOR statement and one for executing a FETCH statement. Procedure ManufDB.FailureList contains one section, for an INSERT statement. For more information on preprocessing and cursors, refer to the appropriate *ALLBASE/SQL Application Programming Guide* and the *ALLBASE/SQL Advanced Application Programming Guide*. For more on procedures, see the *ALLBASE/SQL Reference Manual* chapter “Constraints, Procedures, and Rules.”

Monitoring File Space for Modules and Sections

In order to monitor file space used for program modules and other stored sections, you need to perform the following tasks:

- determine the available file space periodically
- calculate the number of pages needed to store new modules

Note that through the ALTER TABLE, CREATE TABLE, CREATE PROCEDURE, CREATE RULE, CREATE VIEW, DECLARE CURSOR, and PREPARE statements you can specify a DBEFileSet for storing sections, table or long column data. If a DBEFileSet is not specified, the default DBEFileSet is used instead. Refer to the *ALLBASE/SQL Reference Manual* syntax for these statements and for the SET DEFAULT DBEFILESET, GRANT and REVOKE statements for complete information.

Causes for Invalidation of Sections

Sections are stored in **modules** or procedures. ALLBASE/SQL generates a program module when an embedded SQL program is preprocessed. Any changes to an object accessed by a section will cause that section to be invalidated.

For example, if you drop a table, all sections that assume the existence of that table will be invalidated. To enable ALLBASE/SQL to revalidate the section at run time, the table must be recreated before the section is executed. Likewise, all sections are marked invalid during migration of the DBEnvironment. ALLBASE/SQL automatically attempts to revalidate the sections at run time.

The following statements, if they operate on an object accessed by a given section, will cause that section to be invalidated:

ADD DBEFILE	ALTER TABLE	CREATE INDEX
CREATE RULE	DROP DBEFILE	DROP GROUP
DROP INDEX	DROP MODULE	DROP PROCEDURE
DROP RULE	DROP TABLE	DROP VIEW
REMOVE FROM GROUP	REVOKE	TRANSFER OWNERSHIP
UPDATE STATISTICS		

At run time, ALLBASE/SQL will automatically revalidate most of the sections invalidated by any of the statements listed above. If the sections cannot be revalidated by ALLBASE/SQL, the source code must be modified to reflect the changes in the DBEnvironment.

Avoiding the Need for Re-Preprocessing

ALLBASE/SQL will not automatically re-preprocess a program that has undergone source code modification. The program must be fully preprocessed with the C, COBOL, FORTRAN, or Pascal preprocessor. The following statements may require source code changes; each statement is followed by a suggestion on how to avoid changing the code and re-preprocessing the program:

- ALTER TABLE: Using this statement invalidates stored sections that access the table. ALLBASE/SQL will revalidate the section if the statements that accessed the table are still valid following the alteration.
- DROP (anything accessed by the section): ALLBASE/SQL will revalidate the section if the object is recreated before the program is executed. (This does not apply to indexes, which are not accessed directly by a section.)

- **REVOKE** (those authorities granted to the module owner): ALLBASE/SQL will revalidate the section if the required authorities are granted to the module owner before the program is executed. This will not require source code changes, but may require that the program be re-preprocessed with a different module owner.
- **TRANSFER OWNERSHIP** (for all objects in the program except modules): ALLBASE/SQL will revalidate the section if ownership is transferred back to the original owner, or if the current owner is granted all the required authorities.

The first time an invalidated section is executed (when you run a program or execute a procedure), there may be a decrease in performance while ALLBASE/SQL revalidates the section. If you want to validate sections before executing them, you can use the **VALIDATE** statement.

Determining Available Space for Sections

To determine the space available in a DBEFileSet for stored sections, run **SQLMON** and go to Static Size screen. Examine the value of the **TABLE PAGES** field for every table whose owner is **STOREDSECT**.

Determining Number of Sections in the DBEnvironment

You can also determine how many sections are currently stored in the DBEnvironment by querying the **SYSTEM.TABLE** view. First, update statistics on the **SYSTEM.SECTION** view so that the **NROWS** column in the **SYSTEM.TABLE** is updated to show the current number of rows in the **SECTION** table. The **NROWS** column for **SYSTEM.SECTION** shows the number of sections in the DBEnvironment since the last **UPDATE STATISTICS** statement.

The query in the example below shows 44 sections stored in the DBEnvironment with a total of 2 pages occupied by **SYSTEM.SECTION**:

```
isql=> SELECT Name, NRows, NPages FROM System.Table WHERE Name='SECTION';
SELECT Name, NRows, NPages FROM System.Table WHERE Name='SECTION';
-----
NAME                |NROWS      |NPAGES
-----
SECTION             |          44|         2
-----
Number of rows selected is 1
U[p], d[own], l[eft], r[ight], t[op], b[ottom], pr[int] <n>,or e[nd]>
```

Module Related Authorities

Authorities govern who can preprocess, execute, and maintain an application that accesses a DBEnvironment.

To preprocess an application for the first time, you need **CONNECT** authority for the DBEnvironment in which the module is to be stored. To preprocess an existing application, you need **OWNER** authority for that module and **CONNECT** authority for its DBEnvironment.

At run time, the **OWNER** of the program must have authority to execute all SQL statements in the application if the program is to successfully execute. Dynamic statements are an exception. The individual running the program must have the authority to execute a statement that is dynamically preprocessed.

An individual lacking OWNER or DBA authority must have RUN authority for the module and CONNECT authority for the DBEnvironment to be able to run a program that accesses the DBEnvironment. The DBA or module owner can grant RUN authority to users. Only users with DBA authority can grant CONNECT authority.

To maintain a program (modifying code or updating RUN authority) you need OWNER authority for the module or DBA authority. Ownership cannot be transferred, but users with DBA authority can modify or preprocess the program or grant related authorities.

Sharing Modules Between DBEnvironments

Program development usually entails quite a bit of preprocessing and bug fixing before a program is ready for production. Therefore, developing an application in a production DBEnvironment is not a good idea for the following reasons:

- Preprocessing a program may reduce concurrency in a multiuser DBEnvironment by holding locks on the system catalog until preprocessing is finished.
- ALLBASE/SQL treats a preprocessing session as a single transaction which can fill buffers and log files quickly.
- Preprocessing programs not fully debugged can cause problems for other users in the DBEnvironment.

The use of a separate DBEnvironment for development is recommended. When a program is ready to be moved to a production DBEnvironment, you can either re-preprocess the source in the new DBEnvironment or install the module using the ISQL INSTALL command.

When a C, COBOL, FORTRAN, or Pascal program is preprocessed, the preprocessor creates a file in your current working directory called **SQLMOD**, which contains a copy of the module that can be installed in another DBEnvironment.

The INSTALL command installs a module in another DBEnvironment. To use the INSTALL command, you need to have CONNECT or DBA authority for the DBEnvironment that will contain the new module. The following example illustrates the use of the INSTALL command:

```
isql=> CONNECT TO 'PartsDBE';
isql=> INSTALL SQLMOD.COBOL.ALLBASE;

Name of module in this file: PGMR1@ALLBASE.SOMEPROG
Number of sections installed: 6
COMMIT WORK to save to DBEnvironment.

isql=> COMMIT WORK;
```

ISQL copies the installable module from the file named SQLMOD.COBOL.ALLBASE. During installation, ALLBASE/SQL marks each section in the module valid or invalid, depending on the current objects and authorities in PartsDBE.

If your DBEnvironments have different names, you can avoid re-preprocessing a separate program with a different CONNECT statement in each by back referencing the DBEnvironment in the program. Then you can use the INSTALL command to install the module in the desired DBEnvironments. A COBOL program back referencing a DBEnvironment will contain the following CONNECT command:

```
EXEC SQL CONNECT TO '*DBE' END-EXEC.
```

The appropriate DBEnvironment is identified before run time with a file equation. The following file equation identifies *SomeDBE.SomeGrp.SomeAcct* as the DBEnvironment the program accesses at run time:

```
:FILE DBE=SomeDBE.SomeGrp.SomeAcct
```

Dropping Modules

Before a program can be preprocessed a second time, the previously stored module must be dropped. To do this, you can use the DROP MODULE statement in ISQL, or you can use the DROP option of the preprocessor command:

```
isql=> DROP MODULE CEXP01D;
```

or

```
: RUN PSQLPAS.PUB.SYS;INFO="PartsDBE (MODULE(CEXP01D) DROP)"
```

The DROP MODULE statement assumes that all related RUN authorities are to be dropped along with the module. Therefore, when using the DROP MODULE statement, use the PRESERVE option to preserve all related RUN authorities for the new version of the module:

```
isql=> DROP MODULE CEXP01D PRESERVE;
```

The preprocessor, on the other hand, assumes that all related RUN authorities are to be PRESERVED unless revoked with the REVOKE option. Thus you do not need to specify PRESERVE with the preprocessor:

```
: RUN PSQLPAS.PUB.SYS;INFO="PartsDBE (MODULE(CEXP01D) DROP)"
```

The DROP MODULE statement is also useful in conjunction with revised programs whose modules must be installed in a DBEnvironment different from that on which preprocessing occurred. Before using the INSTALL command to store the new module, you drop the existing module using the DROP MODULE statement, preserving or dropping related RUN authorities as required.

Maintaining a Nonstop Production System

A collection of ALLBASE/SQL features can be used to satisfy needs for nonstop, continuously available operations. These features consist of ALLBASE/SQL statements and SQLUtil commands that perform the tasks of database creation, maintenance, and recovery. The statements and commands involved are described in this section under the following topics:

- DBEFiles in different groups and volumes
- Moving DBEFiles to different groups
- Detaching and attaching database files
- CHECKPOINT statement host variable
- Console message file
- User-initiated change to a new log file
- System catalog information

DBEFiles in Different Groups and Volumes

A DBEFile can be defined on a particular device by using the `DEVICE` option to specify either the volume name or the volume identifier. For example, to create a DBEFile on device 3, the following command can be used:

```
CREATE DBEFILE PARTSDBE,  
  WITH PAGES = 4,  
  NAME = 'PARTSDBE',  
  TYPE = TABLE,  
  DEVICE = 'number3';
```

A DBEFile can also be defined in a particular group of an account by qualifying the volume name or the volume identifier. All DBEFiles for a DBEnvironment must be created in groups within the same account. If the DBEFile name is not qualified by a group name, the file is created in the same group as the DBECon file.

For example, suppose that a DBEnvironment was created in an `EMPLOYEES` group. (The DBECon file would also be created in the `EMPLOYEES` group.) To create a DBEFile, `SWING`, in the `PARTTIME` group, the following command might be used:

```
CREATE DBEFILE SWING,  
  WITH PAGES = 4,  
  
  NAME = 'SWING.PARTTIME',  
  TYPE = TABLE;  
ON DEVICE = 'VOL1';
```

This command would create the DBEFile `SWING.PARTTIME` in the `PARTTIME` group on `VOL1`.

Moving DBEFiles to Different Groups

You can move a DBEFile to a different group with the SQLUtil command `MOVEFILE`. If you had created the `SWING` DBEFILE in the `EMPLOYEES` group and wanted to later move it to the `PARTTIME` group, the `MOVEFILE` command would be used as follows:

```
isql=>sqlutil  
>>movefile  
DBEnvironmentName: PARTSDBE  
Current FileName: SWING  
New FileName: SWING.PARTTIME  
Current Device: DISK  
New Device (opt): 2  
Access Mode (mapped/pseudo) (opt): RETURN
```

The `MOVEFILE` command must be executed from the group containing the file to be moved. Thus, the above command must be executed from the `EMPLOYEES` group. The DBECon file can be moved to a different device, but not to a different group.

Detaching and Attaching Database Files

You can detach a DBEFile or DBEFileSet from a DBEnvironment. Detaching a DBEFileSet is equivalent to detaching all of the DBEFiles in the DBEFileSet. Once a DBEFile is detached, data in the DBEFile is inaccessible until it is attached again. The SQLUtil commands, `DETACHFILE` and `ATTACHFILE` are used to detach and attach files.

Detaching DBEFiles from a database is useful for restricting access to parts of a database while the remaining files are online and operational. Detaching files is also useful during file level recovery processing when DBEFiles must be detached before such processing.

To detach a DBEFile or DBEFileSet, the SQLUtil command DETACHFILE is used as shown in the following example of detaching the WareFS DBEFileSet:

```
isql=>sqlutil
>> detachfile
DBEnvironment Name: PartsDBE
Maintenance Word: MaintenanceWord

Enter DBEFileset Name (return to finish): WareFS
Enter DBEFileset Name (return to finish):
Enter DBEFile name (return to finish):
Do you wish to proceed (y/n)? y
```

To attach a file, the SQLUtil command ATTACHFILE is used as shown in the following example of attaching the WareFS DBEFileSet:

```
isql=>sqlutil
>> attachfile
DBEnvironment Name: PartsDBE
Maintenance Word: MaintenanceWord

Enter DBEFileset Name (return to finish): WareFS
Enter DBEFileset Name (return to finish):
Enter DBEFile name (return to finish):
Do you wish to proceed (y/n)? y
```

Using a Host Variable with the CHECKPOINT Statement

The CHECKPOINT statement returns a host variable that contains the approximate number of free blocks (NFB) available in the log file.

The syntax of the CHECKPOINT statement is as follows:

```
CHECKPOINT [ :HostVariable
            :LocalVariable
            :ProcedureParameter]
```

Using Console Message Files

Occurrences of certain events causes messages to be written to a console. In an operatorless environment, these messages can be redirected to a file that can later be queried.

All messages are written to the console if you enter nothing or CONSOLE. If you want messages to also be written to a file, you must enter a file name other than CONSOLE.

The setvar command is used to create a console file as follows:

```
setvar HPSQLconsole console_file_name
setvar HPSQLconsole cnslfile.group.account
```

If no console file name is given, no error is issued and no file is created. The messages are written to the console instead.

The console file is created in the group where the DBECon file resides.

The messages written to the console file are 80 character ASCII records. They can be read with any editor. The text of the message is the same whether it is written to the console or the console file.

Messages are written to the console file when the following events occur:

- A log file is full and the system initiates a switch to a new log file.
- A log file is ready for backup.
- A read or write error occurs on a log file.

Note Messages are color coded for use with OpenView on Emerald systems.

Making Changes to a New Log File

The CHANGELOG command forces a change to a new log file instead of having to wait for the system to switch logs when the present log file becomes full. ALLBASE/SQL changes to a new log file only when the current log is full. However, if you want to force a change to a new log even if the log file is only 90% full, for example, you can perform a backup on the log file at a certain time of day.

The CHANGELOG command returns the sequence number of the current and new log file, as follows:

```
isql=>sqlutil
>>changelog

DBEnvironmentName: PartsDBE
Maintenance Word: MaintenanceWord
Change to a new log (y/n) y

Changed log from Sequence Number 2 to Sequence Number 3.
```

Now, the log file with the sequence number 2 can be backed up.

Checking the System Catalog

The system catalog views, SYSTEM.DBFile and SYSTEM.DBFILESET, have an additional column called ATTACHED which indicates whether the DBFile is attached.

The DBFile view specifies the group name as well as the file name for the DBFile in the FileID field. The group name of a DBFile can optionally be specified as different than that of the DBECon file when a DBFile is created.

Maintaining Security

As new users are added to your system, you will need to grant authorities to them as needed. You must also add new users to appropriate groups. Conversely, when users leave your system, their authorities should be revoked, and they should be removed from all authorization groups.

Use the GRANT and REVOKE statements to add or remove authorities. These statements perform simple maintenance operations that do not affect table data.

Whenever possible, it is recommended that you grant authorities to groups rather than individuals. This simplifies the task of maintenance, since you only need to add or delete a name rather than granting or revoking an entire set of authorities. You can use simple maintenance operations to change the characteristics of authorization groups. Use the ADD and REMOVE statements to add or remove an individual or group. Use the GRANT and REVOKE statements to change the authorizations given to the group.

Disabling Data Definition

A strategy for improving performance is to disable all data definition operations while your applications are running. You do this by using SQLUtil to set the DDL Enabled parameter in the DBECon file to NO. This is a simple matter if you do all data definition in a development DBEnvironment. In such a case, you can disable DDL in the production DBEnvironment only. Disabling data definition

- Allows ALLBASE/SQL to retain sections in memory between transactions. This means that an application program that re-executes the same sections again and again does not require ALLBASE/SQL to read the sections in from disk each time. This can significantly improve performance.

Note ALLBASE/SQL can retain sections in memory between transactions. However, if DDL is enabled, ALLBASE/SQL checks its system catalog once per transaction to see if the cached section is still valid. When DDL is disabled, ALLBASE/SQL does not have to check. Disabling DDL still results in best performance.

- Makes it impossible to perform data definition and to lock system catalog pages exclusively.
- Does *not* inhibit the revalidating of sections. If ALLBASE/SQL encounters invalid sections, they will be revalidated as necessary.
- Disables the following statements:

ADD DBEFILE	ADD GROUP	ADD TO GROUP
ALTER DBEFILE	ALTER TABLE	CREATE DBEFILE
CREATE DBEFILESET	CREATE GROUP	CREATE INDEX
CREATE TABLE	CREATE VIEW	DROP DBEFILE
DROP DBEFILESET	DROP GROUP	DROP INDEX
DROP TABLE	DROP VIEW	GRANT
REMOVE FROM GROUP	REVOKE	START DBE NEWLOG
TRANSFER OWNERSHIP	UPDATE STATISTICS	

Inhibiting DDL has little effect on performance for applications that do not re-use the same sections.

Judging Maintenance Expenses

Many database maintenance statements (CREATE, DROP, ALTER, and so on) lock system catalog resources. To maximize concurrency and improve performance, restructuring and creation should take place in single-user mode during off hours when the DBEnvironment is not usually being accessed.

Transactions that update the system catalog should be kept as short as possible. Commands that create, grant, add, drop, revoke, or update statistics write to the system catalog, which is locked exclusive at the page level for updates. For example, to improve concurrency, you can divide a transaction that creates and loads a table into two transactions. The first transaction creates the table and then commits work, thus freeing system catalog pages. The second transaction loads the table and then commits work.

A chart that shows which SQL statements hold which locks on various system tables is in the appendix, “Locks Held on the System Catalog by SQL Statements.” Keep in mind that lock granularity can be set to table, page, or row level. See the “Changing System Table Lock Types” section in this chapter for more details. If you think you are having locking problems with the system catalog, refer to that appendix as you review your transactions.

Cleaning Up after Abnormal Termination

In the event of abnormal termination, a DBEnvironment may be left with DBEFiles open or with locks still in place for specific pages. ALLBASE/SQL provides a monitor process that regularly checks for abnormal termination and cleans up in such cases. If you have trouble re-connecting to a DBEnvironment, wait a few seconds for the monitor to do its work, then try again. The monitor process starts automatically the first time a user connects to a DBEnvironment. Whenever cleanup occurs, messages describing the actions taken by the monitor are sent to the console.

When a multiuser DBCore session terminates abnormally, ALLBASE/SQL creates two files containing information about the error condition. These files can be useful when support engineers debug the problem. The files are created in the same group and account as the DBECon file. If this cannot be done for security reasons, they are created in the user's current group and account.

The files are as follows:

DBDCxxxx
DBDSxxxx

where *xxxx* is a unique name chosen by the system. *DBDC0000* through *DBDC9999* and *DBDS0000* through *DBDS9999* should be considered reserved filenames.

You can use an ordinary PRINT command to view file DBDSxxxx. In order to view file DBDCxxxx, use the special utility program CBFORMAT.PUB.SYS. After prompting for a file name, CBFORMAT opens the privileged file, formats each record in hexadecimal and ASCII, and prints the formatted records on the standard output.

After displaying the formatted records, CBFORMAT asks whether you wish to delete the file:

Purge this file: (YES,NO)

Respond *YES* to purge the file.

DBDCxxxx and DBDSxxxx files can accumulate if you do not format and purge them periodically.

System Catalog

For each DBEnvironment you configure, ALLBASE/SQL automatically builds and maintains a database known as the **system catalog**, which contains information about:

- Tables and views in the DBEnvironment
- Columns in the tables
- Indexes on the tables
- Constraints, procedures, and rules
- DBFiles and DBFileSets
- Users and their corresponding authorities
- Programs that access the DBEnvironment
- Current DBEnvironment usage and internal usage

This information is actually stored in a set of base tables owned by the special users HPRDBSS and STOREDSECT.

The DBA does not share ownership of the system base tables. The base tables of the system catalog are used by ALLBASE/SQL for tasks such as checking user authority, determining optimal access paths to data, and locating DBEnvironment objects. To provide user access to the information in the system catalog, ALLBASE/SQL creates a set of **system catalog views** on the base tables when the DBEnvironment is configured.

Views owned by SYSTEM and CATALOG

Three distinct groups of system catalog views are created by ALLBASE/SQL:

- Views owned by special user SYSTEM
- Views owned by special user CATALOG
- Pseudotables owned by special user SYSTEM

The SYSTEM views contain information about database objects owned by all users, and access is restricted to users with DBA authority or SELECT authority on the SYSTEM views. The DBA can grant SELECT on these views to other users. SYSTEM views are used to perform administration tasks such as monitoring system usage to improve performance and monitoring physical storage to determine when to add DBFiles.

The CATALOG views contain information about database objects owned by a particular user. These views permit users without DBA authority or SELECT authority on the SYSTEM views to examine system catalog information for objects to which they have access.

In addition to system and catalog views, ALLBASE/SQL generates a set of **pseudotables**. These contain statistical data, including counts and other runtime information about system usage. Pseudotables are generated by SQLCore each time they are accessed. They reside in shared memory, and, although formatted to appear like tables, they are not physically stored in the system catalog. Pseudotables are owned by SYSTEM, and are accessible only to users with DBA authority.

Summary of System Catalog Views by Function

Table 8-1 shows the system catalog views arranged according to their function.

Table 8-1. System Catalog Views by Function

Function	Name of View	Description
User Authority	SYSTEM.COLAUTH, CATALOG.COLAUTH	UPDATE authority on specific columns in a table
	SYSTEM.GROUP, CATALOG.GROUP	Authorization groups
	SYSTEM.MODAUTH, CATALOG.MODAUTH	RUN authority on programs
	SYSTEM.PROCAUTH, CATALOG.PROCAUTH	EXECUTE authority on procedures
	SYSTEM.SPACEAUTH, CATALOG.SPACEAUTH	TABLESPACE and SECTIONSPACE authority for a DBEFileSet
	SYSTEM.SPECAUTH, CATALOG.SPECAUTH	SPECIAL authorities
	SYSTEM.INSTALLAUTH, CATALOG.INSTALLAUTH	INSTALL authority
Object Definitions	SYSTEM.TABAUTH, CATALOG.TABAUTH	Authorities for operations on tables and views
	SYSTEM.CHECKDEF, CATALOG.CHECKDEF	Check constraint definitions
	SYSTEM.COLDEFAULT, CATALOG.COLDEFAULT	Defaults for each column
	SYSTEM.COLUMN, CATALOG.COLUMN	Columns in tables and views
	SYSTEM.CONSTRAINT, CATALOG.CONSTRAINT	Constraints defined on tables and views
	SYSTEM.CONSTRAINTCOL, CATALOG.CONSTRAINTCOL	Columns with a unique, referential,, check constraint
	SYSTEM.CONSTRAINTINDEX, CATALOG.CONSTRAINTINDEX	Indexes in constraints
SYSTEM.HASH, CATALOG.HASH	Hash structures	

Table 8-1. System Catalog Views by Function (continued)

Function	Name of View	Description
	SYSTEM.INDEX, CATALOG.INDEX	Indexes on tables
	SYSTEM.PARAMDEFAULT, CATALOG.PARAMDEFAULT	Defaults for parameters in procedures
	SYSTEM.PARAMETER, CATALOG.PARAMETER	Parameters in procedures
	SYSTEM.PROCEDURE, CATALOG.PROCEDURE	Procedures
	SYSTEM.PROCEDUREDEF, CATALOG.PROCEDUREDEF	Procedure definitions
	SYSTEM.PROCRESULT, CATALOG.PROCRESULT	Procedure results
	SYSTEM.RULE , CATALOG.RULE	Rules
	SYSTEM.RULECOLUMN , CATALOG.RULECOLUMN	Columns listed for a rule triggered by the UPDATE statement type
	SYSTEM.RULEDEF, CATALOG.RULEDEF	Rule definitions
	SYSTEM.SECTION, CATALOG.SECTION	Sections and views
	SYSTEM.TABLE, CATALOG.TABLE	Tables and views
	SYSTEM.VIEWDEF, CATALOG.VIEWDEF	View definitions
	SYSTEM.PARTITION	Partition information
	SYSTEM.IMAGEKEY, CATALOG.IMAGEKEY	Master and detail dataset keys associated with TurboIMAGE databases attached to DBE
	SYSTEM.TPINDEX, CATALOG.TPINDEX	Third-party indexes used in TurboIMAGE database attached to DBE
Storage Management	SYSTEM.DBFILE	DBEFiles
	SYSTEM.DBFILESET	DBEFileSets
	SYSTEM.SPACEDEFAULT	Default TABLESPACE and SECTIONSPACE DBEFileSets
	SYSTEM.TEMPSPACE	TempSpace names

Table 8-1. System Catalog Views by Function (continued)

Function	Name of View	Description
System Usage and Statistics (pseudotables)	SYSTEM.ACCOUNT	I/O resource currently in use
	SYSTEM.CALL	Current DBCore calls
	SYSTEM.COUNTER	Internal system counters
	SYSTEM.TRANSACTION	Current transactions
	SYSTEM.USER	Users currently accessing the DBEnvironment

Using the System Catalog

ALLBASE/SQL creates and initializes the system catalog during DBEnvironment configuration. Thereafter, you query the system views and pseudotables to obtain required information. All view names must be fully qualified when querying the system catalog.

As a DBA, you have limited capabilities to access and alter the base system tables owned by HPRDBSS and STOREDSECT. For example, you cannot read or write directly to these tables. You cannot issue any of the following statements on the system catalog base tables:

```
ALTER TABLE (except SetTypeSpecification)
CREATE INDEX
DELETE
DROP INDEX
DROP TABLE
GRANT
INSERT
LOCK TABLE
REVOKE
TRANSFER OWNERSHIP
UPDATE
```

However, you can issue some statements on the base system tables, which you can use to tune the DBEnvironment for concurrency and performance. For example, you can

- Use ALTER TABLE to change the base system table locking. For example, the DBA may decide to change the HPRDBSS.SECTION table from PUBLIC to PUBLICROW to help resolve concurrency problems associated with section validation.
- Use UPDATE STATISTICS on the base system tables to provide up-to-date statistics for the optimizer in optimizing queries on the system catalog views.

Through the SYSTEM.TABLE view, you can see that the HPRDBSS and STOREDSECT tables exist. However, you cannot see any other information, such as columns and indexes, about the HPRDBSS and STOREDSECT tables. The definitions of the system catalog views and pseudotables are complete in the system catalog.

System Catalog Views

System views are like all other views in that statements normally not allowed on views cannot be executed on the system views. In addition, the following statements, normally allowed on views, are not allowed on system views:

```
DELETE
DROP VIEW
GRANT (except SELECT)
INSERT
REVOKE (except SELECT)
TRANSFER OWNERSHIP
UPDATE
```

You can access the system views with SELECT statements like the following:

```
isql=> SELECT * FROM System.DBFile;
isql=> SELECT * FROM System.Table
> WHERE Owner='SYSTEM';
isql=> SELECT IndexName, TableName FROM System.Index
> WHERE Cluster=1;
```

Because the owner of the system views is SYSTEM, you must fully qualify the names of all system views.

Partition values are stored in the system catalog. You can verify the attributes of table partitions by examining the SYSTEM.TABLE and the SYSTEM.PARTITION views.

Using UPDATE STATISTICS on System Views

The UPDATE STATISTICS statement, which is not normally allowed on a view, is accepted on system views. A user with DBA authority can perform an UPDATE STATISTICS on a system view. ALLBASE/SQL makes an exception for the system views so that the DBA may monitor the system catalog for storage use and performance tuning. The following system views are updated by the UPDATE STATISTICS statement:

```
SYSTEM.COLUMN          SYSTEM.CONSTRAINTINDEX  SYSTEM.DBFILE
SYSTEM.DBFILESET       SYSTEM.HASH              SYSTEM.INDEX
SYSTEM.TABLE
```

You cannot use UPDATE STATISTICS on CATALOG views, and UPDATE STATISTICS does *not* update the following pseudotables:

```
SYSTEM.ACCOUNT          SYSTEM.CALL              SYSTEM.COUNTER
SYSTEM.TRANSACTION      SYSTEM.USER
```

Initially, only users with DBA authority have SELECT authority on views owned by SYSTEM. SELECT authority can be granted to non-DBA users. Views can be created on system views to grant partial access to non-DBA users. If you do not want to restrict access to system views, you can grant SELECT authority to the special DBEUserID PUBLIC. However, you also have the option of keeping the contents of the DBEnvironment confidential.

Locking of the System Catalog

Many ALLBASE/SQL statements have the effect of obtaining locks on parts of the system catalog. In particular, UPDATE STATISTICS acquires many locks that can affect concurrency. This overhead can affect performance considerably. Refer to the appendix, “Locks Held on the System Catalog by SQL Statements,” for a complete list of locks obtained on the system catalog by ALLBASE/SQL statements.

The rest of this chapter contains the column definition for each system view, plus a brief description of the purpose of the view, and a sample SELECT. The examples contain the DBEUserIDs, tables, indexes, views, groups, and other objects from the sample DBEnvironment. The columns of each view are listed in the order in which they are created.

Storedsect.System

STOREDSECT.SYSTEM contains the stored sections for the system DBFileSet.

Even with DBA authority, you have limited access to STOREDSECT.SYSTEM. You cannot access the table directly, but you can use ALTER TABLE or UPDATE STATISTICS on it, as explained in the section “Using the System Catalog” earlier in this chapter.

You can see an entry for STOREDSECT.SYSTEM in SYSTEM.TABLE.

Storedsect.*DBEFileSetName*

The tables named `STOREDSECT.DBEFileSetName` are the stored section tables for *DBEFileSetName*.

These tables are created when you issue the statement `GRANT SECTIONSPACE ON DBEFileSet`.

Even with DBA authority, you have limited access to the `STOREDSECT.DBEFileSetName` tables. You cannot access these tables directly, but you can use `ALTER TABLE` or `UPDATE STATISTICS` on them, as explained in the section “Using the System Catalog” earlier in this chapter.

You can see entries for the `STOREDSECT.DBEFileSetName` tables in `SYSTEM.TABLE`.

System.Account

SYSTEM.ACCOUNT is a pseudotable that contains information about I/O resource usage by users currently accessing the DBEnvironment. This table is initially empty, and ALLBASE/SQL updates it whenever a user queries the DBEnvironment, including accessing a system catalog view.

SYSTEM.ACCOUNT supports an unlimited number of users, transactions, and sessions.

Table 8-2. System.Account

Column Name	Type	Length	Description
USERID	CHAR	20	DBEUserID
CID	INTEGER	4	Unique connection identifier
SID	INTEGER	4	Session identifier. If the same USERID has multiple connections to the same DBEnvironment from the same application (including ISQL), all connections have the same SID. If the user has multiple connections to the same DBEnvironment from separate applications, each application (or ISQL session) has a different SID. For more information about multiple DBEnvironment connections, see the section “Using Multiple Connections and Transactions with Timeouts” in the “Using ALLBASE/SQL” chapter of the <i>ALLBASE/SQL Reference Manual</i> .
NPA	INTEGER	4	Number of page accesses
NLB	INTEGER	4	Number of log bytes written
NTP	INTEGER	4	Number of temporary pages allocated
NPP	INTEGER	4	Number of permanent pages allocated

Example

```
SELECT * FROM System.Account;
```

```
-----+-----+-----+-----+-----
USERID          |CID      |SID      |NPA      |NLB
-----+-----+-----+-----+-----
JOHN@BROCK     |         |108|     |108|     |23|     |0
```

```
Number of rows selected is 1
```

```
U[p], d[own], l[eft], r[ight], t[op], b[ottom], pr[int] <n>,or e[nd]> e
```

```
+-----+-----
|NTP      |NPP
+-----+-----
|         |0|     |0
```

```
Number of rows selected is 1
```

```
U[p], d[own], l[eft], r[ight], t[op], b[ottom], pr[int] <n>,or e[nd]> e
```

System.Call

SYSTEM.CALL is a pseudotable that contains information about current internal (DBCORE) calls, such as whether a user's process is running, waiting for a lock, waiting for a page to be freed, and so forth.

You can use SYSTEM.CALL to determine which users are accessing the DBEnvironment. SYSTEM.CALL supports an unlimited number of users, transactions, and sessions.

Table 8-3. System.Call

Column Name	Type	Length	Description
USERID	CHAR	20	DBEUserID
CID	INTEGER	4	Unique connection identifier
SID	INTEGER	4	Session identifier. If the same USERID has multiple connections to the same DBEnvironment from the same application (including ISQL), all connections have the same SID. If the user has multiple connections to the same DBEnvironment from separate applications, each application (or ISQL session) will have a different SID. For more information about multiple DBEnvironment connections, see the section "Using Multiple Connections and Transactions with Timeouts" in the "Using ALLBASE/SQL" chapter of the <i>ALLBASE/SQL Reference Manual</i> .
FUNCTION	INTEGER	4	Internal function code: 1 start session 2 terminate session 3 begin transaction 4 end transaction 5 abort transaction 6 status 7 relation lock 8 checkpoint 9 open index scan 10 open thread scan 11 open relation scan 12 next 13 close scan 14 fetch TID 15 fetch first 16 insert 17 delete 20 define DBEFileSet 19 drop DBEFileSet 20 associate DBEFileSet 21 disassociate DBEFileSet 22 update

Table 8-3. System.Call (continued)

Column Name	Type	Length	Description
FUNCTION (continued)	INTEGER	4	Description 23 define table 24 drop table 25 alter table 26 create index 27 drop index 28 set index clustering 29 set index uniqueness 30 read counters 31 sort 32 get statistics 33 read account 34 create DBEFile 35 drop DBEFile 43 display parameter 44 display directory data 45 start server 46 terminate server 47 terminate DBCORE 48 start DBCORE 49 open list scan 50 savepoint 51 restore 52 begin archive 53 end archive 54 abort archive 55 change DBEFile type 56 trace on 57 trace off 58 log memo 59 define parent relationship 60 define child relationship 61 drop parent/child relationship 65 relation to relation 66 rename DBEFile 67 update session info 68 modify scan 69 verify predicate 70 extend DBEFile 71 retrieve single tuple 72 define scratch space 73 drop scratch space 74 add a new log 75 purge an existing log 76 display log info 77 get backup flag status 78 reset backup flag 79 resume recovery 80 terminate recovery 81 reset backup flag 82 version id

System.Call

Table 8-3. System.Call (continued)

Column Name	Type	Length	Description
FUNCTION (continued)	INTEGER	4	Description 83 delete a thread 84 quiesce the database 85 unquiesce the database 86 check index 87 open log scan 88 transmit log 89 apply log 90 close log scan 91 get synchronization checkpoint information 92 modify synchronization checkpoint information 93 alter integrity deferral 94 migrate MARSCH for dynamic space expansion 95 bind parent/child relationship 96 set timeout 97 increment table version 98 get table version number 99 put section to offline heap 100 get section from offline heap 101 purge section from offline heap 102 open recovery scan 103 close recovery scan 104 fetch from recovery scan 105 get tran info 106 log persistent information 107 forget record 108 detach transaction 109 alter table type 110 add to columns 111 fix DBCore structures through SQLMigrate 112 switchlog 113 parallel scan 114 truncate table 115 alter transaction attributes 116 detach 117 attach 118 open status scan 119 fetch from status scan 120 close status scan
XID	INTEGER	4	Unique transaction identifier
STATUS	CHAR	20	Internal status: 30 Running 31 Waiting on LOCK 32 Waiting on LATCH 33 Waiting for PAGE 35 Waiting for SERVICE 36 Waiting (Other) 37 Throttle Wait

Example

```
SELECT * FROM System.Call;
```

```
-----+-----+-----+-----+-----+-----  
USERID      |CID      |SID      |FUNCTION    |XID      |STATUS  
-----+-----+-----+-----+-----+-----  
JOHN@BROCK  |    108  |    108  |             |6|-2091903712|Running  
-----+-----+-----+-----+-----+-----
```

```
Number of rows selected is 1
```

```
U[p], d[own], l[eft], r[ight], t[op], b[ottom], pr[int] <n>, or e[nd]> e
```

System.CheckDef

SYSTEM.CHECKDEF contains check constraint definitions. Initially, the table is empty. ALLBASE/SQL updates this table when processing a CREATE TABLE, ALTER TABLE, DROP TABLE, or TRANSFER OWNERSHIP statement involving a check constraint.

The text of the table check constraint comprising the search condition is stored in this table for each check constraint along with the constraint owner and constraint name. All other information about this constraint is in SYSTEM.CONSTRAINT.

SQLGEN also uses this table for recreating a CREATE TABLE statement containing check constraints.

Table 8-4. System.CheckDef

Column Name	Type	Length	Description
CONSTRAINTNAME	CHAR	20	Name of the constraint
OWNER	CHAR	20	Owner of the constraint
COLUMNNAME	CHAR	20	Column to which the constraint applies
SEGNUM	INTEGER	4	Segment number
SEGLN	INTEGER	4	Length of segment in bytes
CONDITIONSTRING	CHAR	64	Check constraint string segment

Example

```
select constraintname, owner, columnname from system.checkdef;
-----+-----+-----
CONSTRAINTNAME |OWNER | COLUMNNAME
-----+-----+-----
SQLCON_0009000200 |PURCHDB |
-----

Number of rows selected is 1
U[p], d[own], l[left], r[ight], t[op], b[ottom], pr[int] <n>,or e[nd]>
```

```
select segnum, seglen, conditionstring from system.checkdef;
-----+-----+-----
SEGNUM |SEGLN |CONDITIONSTRING
-----+-----+-----
1 | 14 |salesprice > 0
-----

Number of rows selected is 1
U[p], d[own], l[left], r[ight], t[op], b[ottom], pr[int] <n>,or e[nd]>
```

System.Colauth

SYSTEM.COLAUTH contains records of authorities granted on specific columns in a table or view; it does not contain records of UPDATE authorities granted on an entire table or view. UPDATE and REFERENCES authorities are the only authorities which can be granted on specific columns. UPDATE and REFERENCES authorities granted on a table or view basis instead of a column-by-column basis are recorded in SYSTEM.TABAUTH (refer to SYSTEM.TABAUTH).

ALLBASE/SQL uses SYSTEM.COLAUTH to verify a user's UPDATE authority prior to executing an UPDATE statement. If the user's UPDATE authority was not found in SYSTEM.TABAUTH or SYSTEM.COLAUTH, ALLBASE/SQL will issue an error message.

ALLBASE/SQL uses SYSTEM.COLAUTH to verify a user's REFERENCES authority prior to executing a CREATE TABLE statement containing referential constraint definitions. If the user's REFERENCES authority was not found in SYSTEM.TABAUTH or SYSTEM.COLAUTH, ALLBASE/SQL will issue an error message.

SYSTEM.COLAUTH is initially empty, and is revised whenever an UPDATE authority on specific columns is granted or revoked. Each row specifies a particular table (or view) column on which UPDATE authority has been granted. If no column is entered, the authority is granted for all of the columns in the table or view. For example, if you grant UPDATE authority to PETER@CRANE:

```
isql=> GRANT UPDATE (BinNumber, QtyOnHand, LastCountDat)
> ON Inventory TO PETER@CRANE;
```

SYSTEM.TABAUTH would contain a row with a DBEUserID of PETER@CRANE, a TableName of Inventory, a C in the UPDATE column, and a 3 in the NCOL column. SYSTEM.COLAUTH would contain three entries for DBEUserID PETER@CRANE; one for each of the three listed columns.

If UPDATE authority was granted without specifying specific columns, SYSTEM.TABAUTH would contain a row with a Y in the update column and SYSTEM.COLAUTH would not contain any rows.

This view, along with SYSTEM.MODAUTH, SYSTEM.PROCAUTH, SYSTEM.SPECAUTH, and SYSTEM.TABAUTH, contains the security scheme for the DBEnvironment.

When you create a PUBLIC or PUBLICREAD table, ALLBASE/SQL implicitly grants table authorities to the special user PUBLIC. In this case, the GRANTOR column contains the table owner name and the GRANTABLE column contains an N to indicate that privileges cannot be granted.

System.Colauth**Table 8-5. System.Colauth**

Column Name	Type	Length	Description
USERID	CHAR	20	Authorized DBEUserID or authorization group
TABLENAME	CHAR	20	Name of the table or view on which the user or authorization group has update authority
OWNER	CHAR	20	Owner of the table or view on which the user or authorization group has update authority
COLNAME	CHAR	20	Name of a table or view column on which the user or authorization group has update authority
TYPE	CHAR	2	Type of authority the user or group has: U UPDATE authority R REFERENCES authority
GRANTOR	CHAR	20	Name of the grantor of the privilege described in this row
GRANTABLE	CHAR	2	GRANTABLE privilege on the column: Y for yes, the user can grant this privilege to others N for no, the user cannot grant this privilege to others

Example

```
SELECT * FROM System.Colauth;
```

USERID	TABLENAME	OWNER	COLNAME	TYPE
KELLY@COTA	INVENTORY	PURCHDB	BINNUMBER	U
KELLY@COTA	INVENTORY	PURCHDB	QTYONHAND	U
KELLY@COTA	INVENTORY	PURCHDB	LASTCOUNTDAT	U
PETER@CRANE	INVENTORY	PURCHDB	BINNUMBER	U
PETER@CRANE	INVENTORY	PURCHDB	QTYONHAND	U
PETER@CRANE	INVENTORY	PURCHDB	LASTCOUNTDAT	U
KAREN@RIZZO	VENDORS	PURCHDB	PHONENUMBER	U
KAREN@RIZZO	VENDORS	PURCHDB	VENDORS TREET	U
KAREN@RIZZO	VENDORS	PURCHDB	VENDORCITY	U
KAREN@RIZZO	VENDORS	PURCHDB	VENDORSTATE	U
KAREN@RIZZO	VENDORS	PURCHDB	VENDORZIPCOD	U
JIM@FRANCIS	VENDORS	PURCHDB	PHONENUMBER	U
JIM@FRANCIS	VENDORS	PURCHDB	VENDORS TREET	U
JIM@FRANCIS	VENDORS	PURCHDB	VENDORCITY	U
JIM@FRANCIS	VENDORS	PURCHDB	VENDORSTATE	U
JIM@FRANCIS	VENDORS	PURCHDB	VENDORZIPCOD	U

```
Number of rows selected is 16
```

```
U[p], d[own], l[eft], r[ight], t[op], b[ottom], pr[int] <n>, or e[nd]> r
```

GRANTOR	GRANTABLE
CLEM@DBMS	N
CLEM@DBMS	N
CLEM@DBMS	N
CLEM@DBMS	N
CLEM@DBMS	N
CLEM@DBMS	N
CLEM@DBMS	N
CLEM@DBMS	N
CLEM@DBMS	N
CLEM@DBMS	N
CLEM@DBMS	N
CLEM@DBMS	N
CLEM@DBMS	N
CLEM@DBMS	N
CLEM@DBMS	N
CLEM@DBMS	N
CLEM@DBMS	N

```
Number of rows selected is 16
```

```
U[p], d[own], l[eft], r[ight], t[op], b[ottom], pr[int] <n>, or e[nd]> e
```

System.Coldefault

SYSTEM.COLDEFAULT contains detailed information about the default constant values which have been specified for columns, as indicated in SYSTEM.COLUMN. While SYSTEM.COLUMN contains a row for each column in each table and view, SYSTEM.COLDEFAULT only contains a row for each column for which a literal default value has been specified. Default values of NULL, USER, CURRENT_DATE, CURRENT_TIME, or CURRENT_DATETIME do not appear in SYSTEM.COLDEFAULT; they are indicated in SYSTEM.COLUMN by code numbers.

This table is initially empty, and is updated whenever ALLBASE/SQL processes a CREATE TABLE, DROP TABLE, or TRANSFER OWNERSHIP command in which the column definitions use the DEFAULT *Constant* clause. The source string containing the default value specification is stored in segments of up to 64 characters. Only BINARY or CHARACTER string columns may require more than one 64-byte segment. All other data types can fit in a 64 byte field.

Table 8-6. System.Coldefault

Column Name	Type	Length	Description
COLNAME	CHAR	20	Name of the column with a default
TABlename	CHAR	20	Name of the table containing this column
OWNER	CHAR	20	Owner of the table
SEGNUM	SMALLINT	2	Segment number
SEGLen	SMALLINT	2	Length of segment in bytes
DEFAULTVAL	CHAR	64	Literal value string segment

Example

```
SELECT * FROM System.Column;
```

```
-----+-----+-----+-----+
COLNAME      | TABlename    | OWNER      | SEGNUM|
-----+-----+-----+-----+
COMPANY      | PARTSOURCE   | PURCHDB    |      1|
```

```
Number of rows selected is 1
```

```
U[p], d[own], l[eft], r[ight], t[op], b[ottom], pr[int] <n>,or e[nd]> r
```

```
-----+-----+-----+-----+
SEGLen| DEFAULTVAL
-----+-----+-----+-----+
```

```
22|Integrated Peripherals
```

```
Number of rows selected is 1
```

```
U[p], d[own], l[eft], r[ight], t[op], b[ottom], pr[int] <n>,or e[nd]> e
```

System.Column

SYSTEM.COLUMN contains detailed information about the columns in the tables described in SYSTEM.TABLE (refer to “SYSTEM.TABLE”). While SYSTEM.TABLE contains a row for every table and view in the DBEnvironment, SYSTEM.COLUMN contains a row for each column in each of those tables and views.

Initially, only the columns of the system views are described. ALLBASE/SQL updates this table when processing an ALTER TABLE, CREATE TABLE, CREATE VIEW, DROP TABLE, DROP VIEW, TRANSFER OWNERSHIP, or UPDATE STATISTICS statement.

Note that the value for PRECISION is only used for the decimal and float data types and SCALE is only used for the decimal data type. For decimal columns, PRECISION has to be a value between 1 and 15, and the value for SCALE must be between 0 and the corresponding value for PRECISION. For floating point columns, PRECISION has to be 53 when LENGTH is 8 and 24 when length is 4.

Note that the DBEFILESET column will only contain a DBEFileSet name for LONG columns, which can reside in a separate DBEFileSet than the table.

Table 8-7. System.Column

Column Name	Type	Length	Description
COLNAME	CHAR	20	Name of the column being described
TABlename	CHAR	20	Name of the table or view containing this column
OWNER	CHAR	20	Owner of the table or view
COLNUM	INTEGER	4	Number of the column in the table or view. Columns are numbered 1, 2, . . . n, and n is kept in the NUMC column of SYSTEM.TABLE
LENGTH	INTEGER	4	Either Maximum length of the column if TYPECODE is 3 (VARCHAR) Or Number of bytes in the column for all other data types
AVGLEN	INTEGER	4	Average column length; initially 0. This value is needed by ALLBASE/SQL.
TYPECODE	SMALLINT	2	Data type of the column: 0 INTEGER or SMALLINT (these two are distinguished by the LENGTH field) 1 BINARY 2 CHAR (ASCII only) 3 VARCHAR (ASCII only) 4 FLOAT or REAL (these two are distinguished by the LENGTH field) 5 DECIMAL 6 TID (for ALLBASE/SQL use only) 7 reserved 8 NATIVE CHAR 9 NATIVE VARCHAR

System.Column

Table 8-7. System.Column (continued)

Column Name	Type	Length	Description
TYPECODE (continued)	SMALLINT	2	Data type of the column: 10 DATE 11 TIME 12 DATETIME 13 INTERVAL 14 VARBINARY 15 LONG BINARY 16 LONG VARBINARY 19 CASE INSENSITIVE CHAR 20 CASE INSENSITIVE VARCHAR 21 CASE INSENSITIVE NATIVE CHAR 22 CASE INSENSITIVE NATIVE VARCHAR
NULLS	SMALLINT	2	Null value indicator: 0 if the column cannot contain null values 1 if the column can contain null values
PRECISION	SMALLINT	2	Number of significant decimal or binary digits in the number (excluding the sign and the decimal point)
SCALE	SMALLINT	2	Number of digits after the decimal point
LANGUAGEID	SMALLINT	2	Code for the language of this column. Run NLUTIL.PUB.SYS to display a complete list of native languages and codes for your system. A value of -1 means NOT APPLICABLE (for numeric type columns or columns in views)
DEFAULTTYPE	SMALLINT	2	Default value type indicator: 0 no default clause specified 1 DEFAULT NULL 2 DEFAULT USER 3 DEFAULT <i>Constant</i> 4 DEFAULT CURRENT_DATE 5 DEFAULT CURRENT_TIME 6 DEFAULT CURRENT_DATETIME
DBEFILESET	CHAR	20	Name of the DBEFileSet holding LONG column data

Example

```
SELECT * FROM System.Column;
```

COLNAME	TABLENAME	OWNER	COLNUM
PARTNUMBER	PARTS	PURCHDB	1
PARTNAME	PARTS	PURCHDB	2
SALESPRICE	PARTS	PURCHDB	3
PARTNUMBER	SUPPLYPRICE	PURCHDB	1
VENDORNUMBER	SUPPLYPRICE	PURCHDB	2
VENDORPARTNUMBER	SUPPLYPRICE	PURCHDB	3
UNITPRICE	SUPPLYPRICE	PURCHDB	4
DELIVERYDAYS	SUPPLYPRICE	PURCHDB	5
DISCOUNTQTY	SUPPLYPRICE	PURCHDB	6
VENDORNUMBER	VENDORSTATISTICS	PURCHDB	1
VENDORNAME	VENDORSTATISTICS	PURCHDB	2
ORDERDATE	VENDORSTATISTICS	PURCHDB	3
ORDERQUANTITY	VENDORSTATISTICS	PURCHDB	4
TOTALPRICE	VENDORSTATISTICS	PURCHDB	5
ORDERNUMBER	ORDERS	PURCHDB	1
VENDORNUMBER	ORDERS	PURCHDB	2

First 16 rows have been selected.

```
U[p], d[own], l[eft], r[ight], t[op], b[ottom], pr[int] <n>, or e[nd]> r
```

LENGTH	AVGLEN	TYPECODE	NULLS	PRECISION	SCALE	LANGUAGEID
16	16	2	0	0	0	-1
30	30	2	1	0	0	0
8	4	4	1	0	0	-1
16	16	2	0	0	0	-1
4	4	0	0	0	0	-1
16	16	2	0	0	0	-1
8	8	4	1	0	0	-1
2	2	0	1	0	0	-1
2	2	2	1	0	0	-1
4	4	0	0	0	0	-1
30	30	2	0	0	0	0
8	8	2	1	0	0	0
2	2	0	1	0	0	-1
8	8	4	1	0	0	-1
4	0	0	0	0	0	-1
4	0	0	0	0	0	-1

First 16 rows have been selected.

```
U[p], d[own], l[eft], r[ight], t[op], b[ottom], pr[int] <n>, or e[nd]> r
```

System.Column

DEFAULTTYPE	DBEFLESET
	0
	0
	0
	0
	0
	0
	0
	0
	0
	0
	0
	0
	0
	0
	0
	0

First 16 rows have been selected.

U[p], d[own], l[eft], r[ight], t[op], b[ottom], pr[int] <n>, or e[nd] > e

System.Constraint

SYSTEM.CONSTRAINT contains detailed information about the integrity constraints contained in the database.

Initially, the table is empty. ALLBASE/SQL updates this table when processing a CREATE TABLE, ALTER TABLE, CREATE VIEW, DROP TABLE, DROP VIEW, or TRANSFER OWNERSHIP statement involving an integrity constraint.

This table is accessed whenever an INSERT, format II INSERT, UPDATE, or DELETE is performed on a table to determine if any integrity constraints need to be enforced.

Further checking into SYSTEM.CONSTRAINTCOL is made if a table is found in SYSTEM.CONSTRAINT, to enforce referential constraints (UNIQUE, PRIMARY KEY, FOREIGN KEY) on a column-dependent basis.

Table 8-8. System.Constraint

Column Name	Type	Length	Description
CONSTRAINTNAME	CHAR	20	Name of the constraint
OWNER	CHAR	20	Owner of the table or view
TABLENAME	CHAR	20	Name of the table or view
TYPE	CHAR	2	Type of constraint: C table check constraint P primary key R referential constraint (foreign key) U unique constraint (key other than primary) V view check constraint
INDEXTYPE	CHAR	2	Type of index used: C clustered index H hash structure N none specified
NUMC	INTEGER	4	Number of columns in the constraint if TYPE is one of P, R, or U.
REFOwner	CHAR	20	Owner of the key column referenced in a FOREIGN KEY constraint
REFCONSTRAINT	CHAR	20	Constraint name of the key column referenced in a FOREIGN KEY constraint
DBEFILESET	CHAR	20	Name of the DBEFileSet containing the stored section for a table check constraint (blank if not applicable)

System.Constraint

Example

```
SELECT * FROM System.Constraint;
```

CONSTRAINTNAME	OWNER	TABLENAME	TYPE	INDEXTYPE
SQLCON_00000001P00	PURCHDB	PARTS1	P	N
CLUBS_PK	RECDB	CLUBS	P	N
MEMBERS_PK	RECDB	MEMBERS	P	N
MEMBERS_FK	RECDB	MEMBERS	R	N
EVENTS_FK	RECDB	EVENTS	R	N

Number of rows selected is 5

U[p], d[own], l[eft], r[ight], t[op], b[ottom], pr[int] <n>, or e[nd]> r

NUMC	REFOwner	REFCONSTRAINT	DBEFILESET
1			
1			
2			
1	RECDB	CLUBS_PK	
2	RECDB	MEMBERS_PK	

Number of rows selected is 5

U[p], d[own], l[eft], r[ight], t[op], b[ottom], pr[int] <n>, or e[nd]> r

System.Constraintcol

SYSTEM.CONSTRAINTCOL contains information about the columns used in PRIMARY KEY, UNIQUE, and FOREIGN KEY integrity constraints.

Initially, the table is empty since the system catalog does not use integrity constraints. ALLBASE/SQL updates this table when processing a CREATE TABLE, DROP TABLE, or TRANSFER OWNERSHIP statement involving an integrity constraint.

This table is accessed whenever an INSERT, type II INSERT, UPDATE, or DELETE is performed on a table in the CONSTRAINT table to determine if the constraint needs to be enforced. It is accessed whenever a REVOKE, REMOVE FROM GROUP, or DROP GROUP statement is performed to determine whether the constraint columns depend on the current authorization scheme.

Table 8-9. System.ConstraintCol

Column Name	Type	Length	Description
CONSTRAINTNAME	CHAR	20	Name of the constraint containing this column
OWNER	CHAR	20	Owner of the constraint
COLUMNNAME	CHAR	20	Name of a column used in the constraint
POSITION	INTEGER	4	Position of the occurrence of the column in the constraint, numbered 1, 2, . . . n, where n is the value of NUMC in the SYSTEM.CONSTRAINT tuple for this constraint.

Example

```
SELECT * FROM System.ConstraintCol;
```

```
-----+-----+-----+-----
CONSTRAINTNAME | OWNER          | COLUMNNAME      | POSITION
-----+-----+-----+-----
SQLCON_00000001P00 | PURCHDB       | PARTNUM         | 1
CLUBS_PK         | RECDB         | CLUBNAME        | 1
MEMBERS_PK      | RECDB         | MEMBERNAME     | 1
MEMBERS_PK      | RECDB         | CLUB            | 2
MEMBERS_FK      | RECDB         | CLUB            | 1
EVENTS_FK       | RECDB         | COORDINATOR     | 1
EVENTS_FK       | RECDB         | SPONSORCLUB    | 2
-----+-----+-----+-----
```

Number of rows selected is 7

U[p], d[own], l[eft], r[ight], t[op], b[ottom], pr[int] <n>, or e[nd]> r

System.ConstraintIndex

SYSTEM.CONSTRAINTINDEX contains an entry for each integrity constraint. This view is initially empty, but is updated whenever ALLBASE/SQL processes a CREATE TABLE, DROP TABLE, ALTER TABLE TRANSFER OWNERSHIP (of a table), or UPDATE STATISTICS statement involving an integrity constraint.

Table 8-10. System.ConstraintIndex

Column Name	Type	Length	Description
CONSTRAINTNAME	CHAR	20	Name of the unique or referential constraint
TABLERNAME	CHAR	20	Name of the table on which the index is defined
OWNER	CHAR	20	Owner of the table on which the index is defined
NUMC	INTEGER	4	Number of columns in the index
COLNUMS	BINARY	32	A vector of 16 SYSTEM.COLUMN entries, each of type SMALLINT, identifying the column numbers the index is defined over. In ISQL, each SMALLINT (two-byte) entry is displayed as a field of 4 hexadecimal digits.
NPAGES	INTEGER	4	Number of data pages containing the constraint index
CCOUNT	INTEGER	4	Cluster count; indicates how well the data of the index are sorted: 0 before first UPDATE STATISTICS statement is processed n efficiency of clustering: best clustering if n=NPAGES of table indexed; worst if n=NROWS of table indexed
CTIME	CHAR	16	Time the index was created
UNIQUE	SMALLINT	2	Uniqueness indicator: 0 if duplicates are allowed, that is, the index is not unique 1 if duplicates are not allowed, that is the index is unique
CLUSTER	SMALLINT	2	Clustering indicator: 0 if the index is not a clustering index 1 if the index is the clustering index for the table
COLDIRS	BINARY	32	A vector of 16 direction entries, each of type SMALLINT, indicating the direction of the corresponding column in the index definition. In ISQL, each SMALLINT (two-byte) entry is displayed as a field of 4 hexadecimal digits. The following values indicate a specific direction: 5 ASC (Ascending) 6 DESC (Descending)

System.Counter

SYSTEM.COUNTER is a pseudotable that contains internal system counters. While the statistics are primarily for SQLCore's internal use, you can use these statistics to monitor DBEnvironment use and make appropriate changes to your DBEnvironment and/or application programs.

For example, you can use the DEADLOCK counter to determine if the number of deadlocks is excessive, and the CHECKPTS counter to determine if you need to change the size of the log file.

Values in SYSTEM.COUNTER are continually incremented from the time of a START DBE until you issue the RESET statement, which sets all counter values to 0. The counters are automatically reset each time the DBE is started.

Some of the rows in this system view are reserved. They are designated as 'UNUSED' when you select all entries from the table, and can be deleted from your query with the following statement:

```
isql=> SELECT * FROM System.Counter
> WHERE NAME <> 'UNUSED';
```

SYSTEM.COUNTER supports an unlimited number of users, transactions, and sessions.

Table 8-11. System.Counter

Column Name	Type	Length	Description
NAME	CHAR	8	Name of counter
VALUE	INTEGER	4	Counter value (time values are in milliseconds)
DESCRIPTION	CHAR	20	Description of the counter

System.Counter

Example

```
SELECT * FROM System.Counter;
```

NAME	VALUE	DESCRIPTION
TRANSBEG	6	Transactions begun
TRANSEND	4	Transactions ended
TRANSABT	0	Transactions abort
CHECKPTS	0	Checkpoints
LOCKREQS	86	Lock requests
SHLTREQS	69	Shared latch reqs
EXLTREQS	537	Excl latch reqs
DEADLOCK	0	Deadlocks
PAGEACCS	152	Page accesses
PAGEREAD	30	Page reads
PAGEWRTE	0	Page writes
LGBLKRD	17	Log block reads
LGBLKWR	0	Log block writes
LGRECWR	0	Log record writes
ERRORLG1	0	Errors on log ds 1
ERRORLG2	0	Errors on log ds 2

First 16 rows have been selected.

U[p], d[own], l[eft], r[ight], t[op], b[ottom], pr[int] <n>,or e[nd]> b

NAME	VALUE	DESCRIPTION
UNUSED	0	-----
UNUSED	0	-----
TIMEOUTS	0	Expired timeouts
UNUSED	0	-----
UNUSED	0	-----
UNUSED	0	-----
UNUSED	0	-----
UNUSED	0	-----
RESETS	1	Number of Resets
DBCALL	117	DBCORE calls
SHLTWAIT	0	Share latch waits
EXLTWAIT	0	Excl latch waits
LOCKWAIT	0	Lock waits
IO	79	I/O s
SERVREQS	0	Service requests

Number of rows selected is 31

U[p], d[own], l[eft], r[ight], t[op], b[ottom], pr[int] <n>,or e[nd]> e

System.DBFile

SYSTEM.DBFILE contains information about the DBFiles in the DBEnvironment. Initially it contains the DBFiles created when the DBEnvironment is configured. A row is added, updated, or deleted whenever ALLBASE/SQL processes an ADD DBFILE, ALTER DBFILE, CREATE DBFILE, DROP DBFILE, or REMOVE DBFILE statement.

SYSTEM.DBFILE and SYSTEM.DBFILESET contain all the information about where tables are stored.

You can use the following query to determine which DBFileSet contains a certain DBFile:

```
isql=> SELECT DBEFName, DBEFSName FROM System.DBFile
> WHERE DBEFName='PURCHDF1';
```

To determine the DBFiles in a DBFileSet, use the following query:

```
isql=> SELECT * FROM System.DBFile
> WHERE DBEFSName='PURCHFS';
```

Table 8-12. System.DBFile

Column Name	Type	Length	Description
DBEFNAME	CHAR	20	Name of the DBFile
DBEFTYPE	SMALLINT	2	Type: 90 for mixed 91 for index 92 for data
FILEID	CHAR	44	System identifier for the file being used for the DBFile
DBEFNPAGES	INTEGER	4	Number of pages in the DBFile
DBEFUPAGES	INTEGER	4	Number of pages in the DBFile containing table and index data, excluding page table pages
DBEFINCRSZ	INTEGER	4	Number of pages the DBFile will be expanded each time
DBEFIPAGES	INTEGER	4	Number of pages in the DBFile at file creation time
DBEFMPAGES	INTEGER	4	Maximum number of pages the DBFile can be expanded
CTIME	CHAR	16	Time of creation: yyyyymmddhhmmsstt
DBEFSNAME	CHAR	20	Name of the DBFileSet the DBFile is associated with (if any)
DBEFNUMBER	INTEGER	4	Number of the DBFileSet the DBFile is associated with
ATTACHED	CHARACTER	2	Whether the DBFile is attached to the DBEnvironment (Y for yes, N for no)

System.DBFile

Example

```
SELECT * FROM System.DBFile;
```

DBEFNAME	DBEFTYPE	FILEID
PARTSDBEO	90	PartsFO
PURCHDATAF1	92	PurchDF1
PURCHINDEXF1	91	PurchXF1
WAREHDATAF1	92	WarehDF1
WAREHINDEXF1	91	WarehXF1
ORDERDATAF1	92	OrderDF1
ORDERINDEXF1	91	OrderXF1
FILEDATA	92	FileData
RECDATAF1	90	RecDF1
NEWFILE	90	NewFile
NEWFILE2	90	NewFile2

```
Number of rows selected is 11
```

```
U[p], d[own], l[eft], r[ight], t[op], b[ottom], pr[int] <n>,or e[nd]> r
```

DBEFNPAGES	DBEFUPAGES	DBEFINCRSZ	DBEFIPAGES	DBEFMPAGES
150	0	0	150	150
50	2	0	50	50
50	4	0	50	50
50	4	0	50	50
50	4	0	50	50
50	3	0	50	50
50	3	0	50	50
50	0	0	50	50
50	7	0	50	50
150	0	0	150	150
100	0	0	100	100

```
Number of rows selected is 11
```

```
U[p], d[own], l[eft], r[ight], t[op], b[ottom], pr[int] <n>,or e[nd]> r
```

CTIME	DBEFSNAME
1990090614175200	SYSTEM
1990090614192820	PURCHFS
1990090614193170	PURCHFS
1990090614194600	WAREHFS
1990090614195400	WAREHFS
1990090614200640	ORDERFS
1990090614200990	ORDERFS
1991020613192200	SYSTEM
1993072014054600	RECFS
1994021415230500	SYSTEM
1994021415252200	

```
Number of rows selected is 11
```

```
U[p], d[own], l[eft], r[ight], t[op], b[ottom], pr[int] <n>,or e[nd]> e
```

DBEFNUMBER	ATTACHED
0	Y
1	Y
2	Y
3	Y
4	Y
5	Y
6	Y
7	Y
8	Y
9	Y
10	Y

Number of rows selected is 11

U[p], d[own], l[eft], r[ight], t[op], b[ottom], pr[int] <n>, or e[nd] > e

System.DBFileSet

SYSTEM.DBFILESET contains information about DBFileSets. This table initially contains a row for each DBFileSet created when the DBEnvironment is configured. A row is added, updated, or deleted whenever ALLBASE/SQL processes an ADD DBFILE, CREATE DBFILESET, REMOVE DBFILE, or DROP DBFILESET statement.

SYSTEM.DBFILESET and SYSTEM.DBFILE contain all the information about where tables and indexes are stored.

Table 8-13. System.DBFileSet

Column Name	Type	Length	Description
DBEFSNAME	CHAR	20	Name of the DBFileSet
DBEFSNDBEFILES	INTEGER	4	Number of DBFiles in the DBFileSet
DBEFSNPAGES	INTEGER	4	Number of pages in the DBFileSet (the sum of the pages in all associated DBFiles)
DBEFSUPAGES	INTEGER	4	Number of pages in the DBFileSet containing table and index data, excluding page table pages (the sum of the used pages in all associated DBFiles)
DBEFSPTPAGES	INTEGER	4	Number of page table pages in the DBFileSet
DBEFSMPAGES	INTEGER	4	Total maximum number of pages the DBFileSet may be expanded (the sum of all DBEFMPAGES in System.DBFile for the associated DBFile)
CTIME	CHAR	16	Time of creation: yyymmddhhmmsstt
ATTACHED	CHAR	2	Attached: Y for yes, N for no

Example

```

SELECT * FROM System.DBFileSet ;
-----+-----+-----+-----+
DBEFSNAME      |DBEFSNDBEFILES|DBEFSNPAGES|DBEFSUPAGES
-----+-----+-----+-----+
SYSTEM         |          2|      100|          8
PURCHFS        |          2|      100|          6
WAREHFS        |          2|      100|          4
ORDERFS        |          2|      100|          5
-----+-----+-----+-----+

Number of rows selected is 4
U[p], d[own], l[eft], r[ight], t[op], b[ottom], pr[int] <n>,or e[nd]> r

+-----+-----+-----+-----+
|DBEFSPTPAGES|DBEFSMPAGES|CTIME          |ATTACHED
+-----+-----+-----+-----+
|          4|          100|1990090614175200|Y
|          2|          100|1990090614192370|Y
|          2|          100|1990090614194160|Y
|          2|          100|1990090614200170|Y
-----+-----+-----+-----+

Number of rows selected is 4
U[p], d[own], l[eft], r[ight], t[op], b[ottom], pr[int] <n>,or e[nd]> e

```

System.Group

SYSTEM.GROUP contains a row for each user or group that belongs to an authorization group. Each user, whether an individual user or another authorization group, has an entry for each authorization group to which he or she is a member. This view is initially empty, but is updated whenever ALLBASE/SQL processes a CREATE GROUP, ADD TO GROUP, TRANSFER OWNERSHIP (of a group), REMOVE FROM GROUP, or DROP GROUP statement.

SYSTEM.GROUP can be used with SYSTEM.COLAUTH, SYSTEM.MODAUTH, SYSTEM.SPECAUTH, and SYSTEM.TABAUTH to determine authorities granted to authorization groups.

You can use this table to determine all the members of a particular authorization group or how many authorization groups to which a particular user belongs. For example, if you want to know all the users in the Receiving authorization group, enter:

```
isql=> SELECT UserId FROM System.Group
> WHERE GroupId='RECEIVING';
```

Table 8-14. System.Group

Column Name	Type	Length	Description
USERID	CHAR	20	DBEUserID or authorization group that is a member of the GROUPID
GROUPID	CHAR	20	Authorization group name
OWNER	CHAR	20	Owner of the authorization group
NMEMBERS	INTEGER	4	Number of members in the authorization group

Example

```
SELECT * FROM System.Group;
```

USERID	GROUPID	OWNER	NMEMBERS
PURCHMANAGERS	PURCHMANAGERS	JOHN@BROCK	3
MARGY@RYAN	PURCHMANAGERS	JOHN@BROCK	0
RON@HART	PURCHMANAGERS	JOHN@BROCK	0
SHARON@MULDOON	PURCHMANAGERS	JOHN@BROCK	0
PURCHDBMAINT	PURCHDBMAINT	JOHN@BROCK	4
ANNIE@MELCHOIR	PURCHDBMAINT	JOHN@BROCK	0
DOUG@DOLAN	PURCHDBMAINT	JOHN@BROCK	0
DAVID@BLOOM	PURCHDBMAINT	JOHN@BROCK	0
PURCHASING	PURCHASING	JOHN@BROCK	5
AJ@SERED	PURCHASING	JOHN@BROCK	0
JORGE@GUERRERO	PURCHASING	JOHN@BROCK	0
RAGAA@QUILLEN	PURCHASING	JOHN@BROCK	0
GREG@SAWYER	PURCHASING	JOHN@BROCK	0
KAREN@RIZZO	PURCHASING	JOHN@BROCK	0
RECEIVING	RECEIVING	JOHN@BROCK	4
AL@DAL	RECEIVING	JOHN@BROCK	0

First 16 rows have been selected

```
U[p], d[own], l[eft], r[ight], t[op], b[ottom], pr[int] <n>, or e[nd]> e
```

System.Hash

SYSTEM.HASH contains an entry for each hash structure you create. This view is initially empty, but is updated whenever ALLBASE/SQL processes a hash-related CREATE TABLE, DROP TABLE, TRANSFER OWNERSHIP, or UPDATE STATISTICS statement.

Table 8-15. System.Hash

Column Name	Type	Length	Description
TABlename	CHAR	20	Name of the table on which the hash is defined
OWNER	CHAR	20	Owner of the table on which the hash structure is defined
NUMC	INTEGER	4	Number of columns in the hash key
COLNUMS	BINARY	32	A vector of 16 SYSTEM.COLUMN entries, each of type SMALLINT, identifying the column numbers the hash is defined over. In ISQL, each SMALLINT (two-byte) entry is displayed as a field of 4 hexadecimal digits.
PRIMPAGES	INTEGER	4	Number of primary pages allocated for the hash structure
NPPAGES	INTEGER	4	Number of primary pages in use, excluding page table pages
AVGLEN	FLOAT	8	Average chain length
MAXLEN	INTEGER	4	Maximum chain length
NFULL	INTEGER	4	Number of primary pages that are more than half full
NOVERFLOW	INTEGER	4	Number of primary pages with overflow pages
UNIQUE	SMALLINT	2	Uniqueness indicator; currently always 1

System.Imagekey

SYSTEM.IMAGEKEY contains an entry for each IMAGE key on each dataset in each IMAGE database attached to this DBE. In ALLBASE/SQL terms the IMAGE keys are viewed as indexes, and the datasets on which they are defined are viewed as tables. The IMAGE database in which these datasets reside is listed as the owner of these IMAGE tables.

Table 8-16. System.Imagekey

Column Name	Type	Length	Description
INDEXNAME	CHAR	20	Index name give to the IMAGE key
TABlename	CHAR	20	Name of the table (dataset) on which the IMAGE key is defined
OWNER	CHAR	20	Owner of the table (dataset) on which the IMAGE key is defined. This is always the name of the IMAGE database where the dataset is stored.
UNIQUE	SMALLINT	2	Uniqueness indicator: 0 if duplicates are allowed, that is, the index is not unique 1 if duplicates are not allowed, that is the index is unique
NUMC	INTEGER	4	Number of columns in the IMAGE key (always 1 for IMAGE keys)
COLNUMS	BINARY	64	A vector of 16 SYSTEM.COLUMN entries, each of type SMALLINT, identifying the column numbers the IMAGE key is defined over
NDISTINCT	INTEGER	4	Number of distinct values
PRIMARIES	INTEGER	4	Number of primary slots used. (not currently used)
SCCCOUNT	INTEGER	4	Synonym chain cluster count; indicates the number of page changes necessary to read through the entire synonym chain. (not currently used)
DCCCOUNT	INTEGER	4	Detail chain cluster count; indicates the average number of page changes necessary to read through a detail chain. (not currently used)

System.Index

SYSTEM.INDEX contains an entry for each index created by a user on a table. This view is initially empty, but is updated whenever ALLBASE/SQL processes a CREATE INDEX, DROP INDEX, DROP TABLE, TRANSFER OWNERSHIP (of a table), or UPDATE STATISTICS statement.

Table 8-17. System.Index

Column Name	Type	Length	Description
INDEXNAME	CHAR	20	Name of the index
TABlename	CHAR	20	Name of the table on which the index or hash structure is defined
OWNER	CHAR	20	Owner of the table on which the index is defined
UNIQUE	SMALLINT	2	Uniqueness indicator: 0 if duplicates are allowed, that is, the index is not unique 1 if duplicates are not allowed, that is the index is unique
CLUSTER	SMALLINT	2	Clustering indicator: 0 if the index is not a clustering index 1 if the index is the clustering index for the table
NUMC	INTEGER	4	Number of columns in the index or hash key
COLNUMS	BINARY	64	A vector of column numbers, each of type SMALLINT, identifying the columns the index is defined over. In ISQL, each SMALLINIT (two-byte) entry is displayed as a field of 4 hexadecimal digits.
NPAGES	INTEGER	4	Number of index pages containing the index
NLEVELS	INTEGER	4	Number of B-tree levels
NLEAVES	INTEGER	4	Number of B-tree leaf pages
NDISTINCT	INTEGER	4	Number of distinct keys
NFIRST	INTEGER	4	Number of distinct first column values of the B-tree key
NPERKEY	INTEGER	4	Number of pages per B-tree key

Table 8-17. System.Index (continued)

Column Name	Type	Length	Description
CCOUNT	INTEGER	4	Cluster count; indicates how well the data of the index is sorted 0 before first UPDATE STATISTICS statement is processed n efficiency of clustering: best clustering if n=NPAGES of table indexed; worst if n=NROWS of table indexed
CTIME	CHAR	16	Time of creation: yyyyymmddhhsstt
COLDIRS	BINARY	64	A vector of direction entries, each of type SMALLINT indicating the direction of the corresponding column in the index definition. In ISQL, each SMALLINT (two-byte) entry is displayed as a field of 4 hexadecimal digits. 5 ASC (Ascending) 6 DESC (Descending)

Example

```
SELECT * FROM System.Index;
```

```
-----+-----+-----+-----+-----+
INDEXNAME      | TABLENAME      | OWNER      | UNIQUE | CLUST
-----+-----+-----+-----+-----+
PARTNUMINDEX   | PARTS           | PURCHDB    | 1      | 1
PARTTONUMINDEX| SUPPLYPRICE     | PURCHDB    | 0      | 0
PARTTOVENDINDEX| SUPPLYPRICE     | PURCHDB    | 0      | 0
VENDPARTINDEX  | SUPPLYPRICE     | PURCHDB    | 1      | 1
VENDORNUMINDEX| VENDORS         | PURCHDB    | 1      | 1
ORDERNUMINDEX  | ORDERS          | PURCHDB    | 1      | 1
ORDERVENDINDEX| ORDERS          | PURCHDB    | 0      | 0
ORDERITEMINDEX | ORDERITEMS      | PURCHDB    | 1      | 1
INVPARTNUMINDEX| INVENTORY       | PURCHDB    | 1      | 1
-----+-----+-----+-----+-----+
```

Number of rows selected is 9

```
U[p], d[own], l[eft], r[ight], t[op], b[ottom], pr[int] <n>, or e[nd] > r
```

```
-----+-----+-----+-----+-----+
| OWNER      | UNIQUE | CLUSTER | NUMC   | COLNUMS
-----+-----+-----+-----+-----+
| PURCHDB    | 1      | 0       |        | 1|000100000000000000000000000000
| PURCHDB    | 0      | 1       |        | 1|000100000000000000000000000000
| PURCHDB    | 0      | 0       |        | 1|000200000000000000000000000000
| PURCHDB    | 1      | 0       |        | 1|000300000000000000000000000000
| PURCHDB    | 1      | 0       |        | 1|000100000000000000000000000000
| PURCHDB    | 1      | 1       |        | 1|000100000000000000000000000000
| PURCHDB    | 0      | 0       |        | 1|000200000000000000000000000000
| PURCHDB    | 1      | 1       |        | 2|000100020000000000000000000000
| PURCHDB    | 1      | 0       |        | 1|000100000000000000000000000000
-----+-----+-----+-----+-----+
```

Number of rows selected is 9

```
U[p], d[own], l[eft], r[ight], t[op], b[ottom], pr[int] <n>, or e[nd] > r
```

System.Installauth

SYSTEM.INSTALLAUTH lists all users and authorization groups that have been granted INSTALL authority. ALLBASE/SQL updates SYSTEM.INSTALLAUTH when processing a GRANT INSTALL or REVOKE INSTALL statement, or when dropping a group having INSTALL authority.

SYSTEM.INSTALLAUTH, SYSTEM.COLAUTH, SYSTEM.MODAUTH, SYSTEM.PROCAUTH, SYSTEM.SPACEAUTH, SYSTEM.SPACEDEFAULT, SYSTEM.SPECAUTH, and SYSTEM.TABAUTH, contain the security scheme for the DBEnvironment.

CATALOG.INSTALLAUTH is identical in format to SYSTEM.INSTALLAUTH; it permits users without DBA authority or SELECT authority on SYSTEM.INSTALLAUTH to examine rows to which they have access.

Table 8-18. System.Installauth

Column Name	Type	Length	Description
USERID	CHAR	20	Authorized DBEUserID or authorization group
OWNER	CHAR	20	An owner name the USERID is authorized to use. If blank, the USERID is authorized to use any owner name.

Example

```
SELECT * FROM System.Installauth;
-----+-----
USERID          |OWNER
-----+-----
GEORGE@DBMS    |
CLEM@DBMS      |JOHN@BROCK
CLEM@DBMS      |SUSAN@SMITH
-----+-----

Number of rows selected is 3
U[p], d[own], l[eft], r[ight], t[op], b[ottom], pr[int] <n>, or e[nd]> e
```


System.Modauth

SYSTEM.MODAUTH contains RUN authorities for application programs. This view, along with the SYSTEM.COLAUTH, SYSTEM.PROCAUTH, SYSTEM.SPECAUTH, and SYSTEM.TABAUTH views, contains the authorities for the DBEnvironment.

Table 8-19. System.Modauth

Column Name	Type	Length	Description
USERID	CHAR	20	Authorized DBEUserID or authorization group
NAME	CHAR	20	Name of the module for which the user has RUN authority
OWNER	CHAR	20	Owner of the module for which the user has RUN authority

Example

```
SELECT * FROM System.Modauth;
```

```
-----+-----+-----
USERID      | NAME      | OWNER
-----+-----+-----
JIM@FRANCIS | CEXPO6    | JOHN@BROCK
KAREN@RIZZO | CEXPO6    | JOHN@BROCK
-----+-----+-----
```

```
Number of rows selected is 2
```

```
U[p], d[own], l[eft], r[ight], t[op], b[ottom], pr[int] <n>, or e[nd]> e
```

System.Paramdefault

SYSTEM.PARAMDEFAULT contains information about the default values of parameters that may be passed to and from procedures.

SYSTEM.PARAMDEFAULT is initially empty, and it is updated whenever ALLBASE/SQL processes a CREATE PROCEDURE, DROP PROCEDURE, or TRANSFER OWNERSHIP statement for a procedure that includes default parameter values.

If a TRANSFER OWNERSHIP is done on the procedure, the owner field in this view is updated with the name of the new owner.

For literal default values, the source string containing the default value specification is stored in segments of up to 64 bytes.

Only BINARY or CHARACTER string columns may require more than one 64 byte segment. All other data types can fit in a 64 byte field.

Table 8-20. System.Paramdefault

Column Name	Type	Length	Description
NAME	CHAR	20	Name of the parameter with a default
PROCNAME	CHAR	20	Name of the procedure containing this parameter
OWNER	CHAR	20	Owner of the procedure
SEGNUM	SMALLINT	2	Segment number
SEGLen	SMALLINT	2	Length of segment in bytes
DEFAULTVAL	CHAR	64	Literal value string segment

Example

```
SELECT Name, ProcName, Owner FROM System.Paramdefault;
-----+-----+-----
NAME      | PROCNAME      | OWNER
-----+-----+-----
SHIFTNAME | PROCESS12     | PURCHDB

-----
Number of rows selected is 1
U[p], d[own], l[eft], r[ight], t[op], b[ottom], pr[int] <n>, or e[nd]> e
```

```
SELECT SegNum, SegLen, DefaultVal FROM System.Paramdefault;
-----+-----+-----
SEGNUM| SEGLen| DEFAULTVAL
-----+-----+-----
1|     3| Day

-----
Number of rows selected is 1
U[p], d[own], l[eft], r[ight], t[op], b[ottom], pr[int] <n>, or e[nd]> e
```

System.Parameter

SYSTEM.PARAMETER contains information about parameters that may be passed to and from procedures. SYSTEM.PARAMETER is initially empty, and it is updated whenever ALLBASE/SQL processes a CREATE PROCEDURE, DROP PROCEDURE, or TRANSFER OWNERSHIP statement for a procedure that includes parameters.

If a TRANSFER OWNERSHIP is done on the procedure, the owner field in this view is updated with the name of the new owner.

Table 8-21. System.Parameter

Name	Type	Length	Description
NAME	CHAR	20	Name of the parameter
PROCNAME	CHAR	20	Name of the procedure
OWNER	CHAR	20	Owner of the procedure
NUM	INTEGER	4	Number of the parameter within the procedure
LENGTH	INTEGER	4	Either Maximum length of the column if TYPECODE is 3 (VARCHAR), 9 (NATIVE VARCHAR), or 14 (VARBINARY) Or Number of bytes in the column for all other data types
TYPECODE	SMALLINT	2	Data type of the column: 0 INTEGER or SMALLINT (set by the LENGTH field) 1 BINARY 2 CHAR (ASCII only) 3 VARCHAR (ASCII only) 4 FLOAT or REAL (set by the LENGTH field) 5 DECIMAL 6 TID 7 reserved 8 NATIVE CHAR 9 NATIVE VARCHAR 10 DATE 11 TIME 12 DATETIME 13 INTERVAL 14 VARBINARY
NULLS	SMALLINT	2	Null value indicator: 0 if the column cannot contain null values 1 if the column can contain null values

System.Parameter

Table 8-21. System.Parameter (continued)

Name	Type	Length	Description
PRECISION	SMALLINT	2	Number of significant decimal or binary digits in the number (excluding the sign and the decimal point)
SCALE	SMALLINT	2	Number of digits after the decimal point
LANGUAGEID	SMALLINT	2	Code for the language of this column. Run NLUTIL.PUB.SYS to display a complete list of native languages and codes for your system. A value of -1 means <i>not applicable</i> (for non-character type columns)
DEFAULTTYPE	SMALLINT	2	Default value type indicator: 0 no default clause specified 1 DEFAULT NULL 2 DEFAULT USER 3 DEFAULT <i>Constant</i> 4 DEFAULT CURRENT_DATE 5 DEFAULT CURRENT_TIME 6 DEFAULT CURRENT_DATETIME
OUTPUT	SMALLINT	2	Parameter type: 0 input 1 input/output 2 output only

Example

```
select * from system.parameter;
```

```
-----+-----+-----+-----
NAME          | PROCNAME      | OWNER      | NUM
-----+-----+-----+-----
OPERATOR      | PROCESS12     | MANUFDB    | 1
SHIFT         | PROCESS12     | MANUFDB    | 2
FAILURETYPE   | PROCESS12     | MANUFDB    | 3
PARTNUMBER    | DISCOUNTPART  | PURCHDB    | 1
NAME          | REPORTMONITOR | PURCHDB    | 1
OWNER         | REPORTMONITOR | PURCHDB    | 2
TYPE          | REPORTMONITOR | PURCHDB    | 3
VENDORNUMBER  | DELVENDOR     | PURCHDB    | 1
VENDORNUMBER  | CHECKVENDOR   | PURCHDB    | 1
PARTNUMBER    | PROCESS15     | PETERW@WEYGANT | 1
PERCENTAGE    | DISCOUNTALL  | PURCHDB    | 1
NROWS        | ENTERTEST     | PETERW@WEYGANT | 1
-----+-----+-----+-----
```

```
Number of rows selected is 12
```

```
U[p], d[own], l[eft], r[ight], t[op], b[ottom], pr[int] <n>, or e[nd] >
```

```
select length, typecode,nulls,precision from system.parameter;
```

LENGTH	TYPECODE	NULLS	PRECISION
20	2	0	0
20	2	0	0
10	2	0	0
16	2	1	0
20	2	0	0
20	2	0	0
10	2	0	0
4	0	0	10
4	0	0	10
16	2	0	0
4	5	1	4
4	0	1	10

Number of rows selected is 12

U[p], d[own], l[eft], r[ight], t[op], b[ottom], pr[int] <n>, or e[nd] >

```
select scale, languageid, defaulttype, output from system.parameter;
```

SCALE	LANGUAGEID	DEFAULTTYPE	OUTPUT
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
2	0	0	0
0	0	0	0

Number of rows selected is 12

U[p], d[own], l[eft], r[ight], t[op], b[ottom], pr[int] <n>, or e[nd] >

System.Partition

The SYSTEM.PARTITION view is used to describe each partition. The base table (which is HPRDBSS.PARTITION) is locked with an exclusive lock if you execute a CREATE PARTITION or DROP PARTITION statement. HPRDBSS.PARTITION is locked with a share lock if you execute a CREATE TABLE statement that specifies a partition or an ALTER TABLE statement that modifies a partition.

The SYSTEM.PARTITION table initially specifies the DEFAULT partition for all tables, if they are created when the DBEnvironment is configured. A row is added, updated, or deleted from SYSTEM.PARTITION whenever one of the following occurs:

- A CREATE PARTITION or DROP PARTITION statement is processed.
- A START DBE NEWLOG statement defining or altering the DEFAULT partition is processed.
- A CREATE TABLE or ALTER TABLE statement specifying a partition is processed.

The PARTITION table format is shown in Table 8-22.

Table 8-22. SYSTEM.PARTITION

Column Name	Type	Length	Description
PARTITIONNAME	CHAR	20	Name of the Partition
PARTITIONID	INTEGER	4	Number of the partition this table is assigned to. 0 if default, -1 if none. This column is not exposed in the system view, and is used to minimize table accesses.

System.Plan

SYSTEM.PLAN is a pseudotable which displays the access plan generated by the optimizer for a SELECT, UPDATE or DELETE statement processed by the GENPLAN statement. Information is displayed for only a single statement at a time.

To display an access plan, you must first process a statement of the above type with the GENPLAN statement as in the following example:

```
isql=> GENPLAN FOR SELECT * FROM Purchdb.Parts;
```

To display the access plan, issue the following statement within the same transaction:

```
isql=> SELECT * FROM System.Plan;
```

Table 8-23. System.Plan

Column Name	Type	Length	Description
QUERYBLOCK	INTEGER	4	Queryblock in which operation is executed
STEP	INTEGER	4	Sequence within the query block in which operation is executed at run time
LEVEL	INTEGER	4	Level of operation within the run tree
OPERATION	CHAR	20	Type of Operation: merge join nestedloop join sort project filter distinct distinct sort group by or union serial scan index scan TID scan hash scan block scan (block number)
TABlename	CHAR	20	Table upon which operation is executed
OWNER	CHAR	20	Owner of the table
INDEXNAME	CHAR	20	Name of index used for operation

System.Plan

Example

```
isql=> GENPLAN FOR
> SELECT *
> FROM Purchdb.Parts
> WHERE Partnumber =
> (SELECT Partnumber
> FROM PurchDB.SupplyPrice sp, PurchDB.Vendors v
> WHERE v.VendorName = 'Pro-Litho Inc.'
> AND sp.UnitPrice <= 200.00
> AND sp.VendorNumber = v.VendorNumber);
```

```
isql=> SELECT * FROM System.Plan;
```

```
SELECT * FROM System.Plan;
```

QUERYBLOCK	STEP	LEVEL	OPERATION	TABLENAME
	1	1	3 serial scan	VENDORS
	1	2	3 serial scan	SUPPLYPRICE
	1	3	2 nestedloop join	
	2	1	1 index scan	PARTS

Number of rows selected is 4

U[p], d[own], l[left], r[ight], t[op], b[ottom], pr[int] <n>, or e[nd] > r

OWNER	INDEXNAME
PURCHDB	
PURCHDB	
PURCHDB	PARTNUMINDEX

Number of rows selected is 4

U[p], d[own], l[left], r[ight], t[op], b[ottom], pr[int] <n>, or e[nd] > e

System.Procauth

SYSTEM.PROCAUTH contains EXECUTE authorities for procedures. This view, along with the SYSTEM.COLAUTH, SYSTEM.MODAUTH, SYSTEM.SPECAUTH, and SYSTEM.TABAUTH views, contains the security scheme for the DBEnvironment.

SYSTEM.PROCAUTH is initially empty, and it is updated whenever ALLBASE/SQL processes a GRANT EXECUTE, REVOKE EXECUTE, a TRANSFER OWNERSHIP of a procedure, or a DROP PROCEDURE (without the PRESERVE option).

Table 8-24. System.Procauth

Column Name	Type	Length	Description
USERID	CHAR	20	Authorized DBEUserID or authorization group
NAME	CHAR	20	Name of the procedure for which the user has EXECUTE authority
OWNER	CHAR	20	Owner of the procedure for which the user has EXECUTE authority

Example

```
SELECT * FROM System.Procauth;
```

```
-----+-----+-----
USERID      | NAME          | OWNER
-----+-----+-----
JIM@FRANCIS | REPORTMONITOR | PURCHDB
KAREN@RIZZO | PROCESS12     | JOHN@BROCK
-----+-----+-----

Number of rows selected is 2
U[p], d[own], l[eft], r[ight], t[op], b[ottom], pr[int] <n>, or e[nd]> e
```

System.Procedure

SYSTEM.PROCEDURE contains information about each procedure created in the DBEnvironment.

SYSTEM.PROCEDURE is initially empty, and it is updated whenever ALLBASE/SQL processes a CREATE PROCEDURE, DROP PROCEDURE, or TRANSFER OWNERSHIP statement on a procedure.

If a TRANSFER OWNERSHIP is done on the procedure, the owner field in this view is updated with the name of the new owner.

Table 8-25. System.Procedure

Name	Type	Length	Description
NAME	CHAR	20	Name of the procedure
OWNER	CHAR	20	Owner of the procedure
NUMP	INTEGER	4	Number of parameters (0 to 1023) to the procedure
NUMR	SMALLINT	2	Number of result columns (0 to 1024) for a procedure defined with a WITH RESULT clause
MULTIRESET	SMALLINT	2	Number of SELECT statements with no INTO clause in the procedure (0 if there are none)
CTIME	CHAR	16	Time of creation: yyymmddhhmmsstt
LANGUAGEID	SMALLINT	2	Code for the language of this procedure. Run NLUTIL.PUB.SYS to display a complete list of native languages and codes for your system.
DBEFILESET	CHAR	20	Name of the DBEFileSet containing the procedure's definition and stored sections

Example

```
select * from system.procedure;
```

```
-----+-----+-----+-----
NAME          |OWNER          |NUMP          |NUMR
-----+-----+-----+-----
PROCESS12     |MANUFDB       |              |3| 2
DISCOUNTPART|PURCHDB       |              |1| 3
```

```
-----
Number of rows selected is 2
```

```
U[p], d[own], l[eft], r[ight], t[op], b[ottom], pr[int] <n>, or e[nd] >
```

```
select multiresult, ctime, languageid from system.procedure;
```

```
-----+-----+-----+-----
MULTIRESULT|CTIME          |LANGUAGEID|DBEFILESET
-----+-----+-----+-----
          3|1991121010220700|          0|SYSTEM
          1|1991121011442200|          0|PURCHFS
```

```
-----
Number of rows selected is 2
```

```
U[p], d[own], l[eft], r[ight], t[op], b[ottom], pr[int] <n>, or e[nd] >
```

System.ProcedureDef

The SYSTEM.PROCEDUREDEF view displays information about procedure definitions, including the text of the procedure itself. The procedure text is stored in a series of segments of up to 64 bytes, starting with the initial BEGIN and ending with the final semicolon. The procedure definition does not include parameter definitions, which are stored separately in SYSTEM.PARAMETER.

SYSTEM.PROCEDUREDEF is initially empty, and it is updated whenever ALLBASE/SQL processes a CREATE PROCEDURE statement, a DROP PROCEDURE statement, or a TRANSFER OWNERSHIP statement on a procedure.

When the procedure is dropped, the rows making up the byte string of the procedure definition are deleted. If a TRANSFER OWNERSHIP is done on the procedure, the owner field in this table is updated with the name of the new owner.

Table 8-26. System.ProcedureDef

Column Name	Type	Length	Description
NAME	CHAR	20	Name of the procedure
OWNER	CHAR	20	Owner of the procedure
QUALIFIER	CHAR	20	Owner name to be used to qualify any unqualified objects referenced in the procedure definition
SEGNUM	INTEGER	4	Segment number
SEGLEN	INTEGER	4	Length of segment in bytes (up to 64)
DEFINESTRING	CHAR	64	Procedure definition byte string segment

Example

```
SELECT * FROM System.ProcedureDef WHERE NAME = 'REPORTMONITOR';
-----+-----+-----+-----
NAME          |OWNER          |QUALIFIER      |SEGNUM
-----+-----+-----+-----
REPORTMONITOR |PURCHDB        |PURCHDB        |1
REPORTMONITOR |PURCHDB        |PURCHDB        |2
-----+-----+-----+-----

First 2 rows have been selected.
U[p], d[own], l[eft], r[ight], t[op], b[ottom], pr[int] <n>,or e[nd]> r

+-----+-----+-----+-----
|SEGLEN|DEFINESTRING
+-----+-----+-----+-----
| 55| begin insert into PurchDB.ReportMonitor values (:Type,
| 45| CURRENT_DATETIME, USER, :Name, :Owner); end;
-----+-----+-----+-----

First 2 rows have been selected.
U[p], d[own], l[eft], r[ight], t[op], b[ottom], pr[int] <n>,or e[nd]> r
```

System.ProcResult

SYSTEM.PROCRESULT contains information about the result columns in procedure result sets for procedures returning results of a single format. Such procedures are created using the WITH RESULT clause.

SYSTEM.PROCRESULT is initially empty, and is updated whenever ALLBASE/SQL processes a CREATE PROCEDURE statement including the WITH RESULT clause, or a DROP PROCEDURE or TRANSFER OWNERSHIP statement specifying such a procedure.

If a TRANSFER OWNERSHIP is done on the procedure, the owner field in this view is updated with the name of the new owner.

Table 8-27. System.ProcResult

Name	Type	Length	Description
PROCEDURE	CHAR	20	Name of the procedure
OWNER	CHAR	20	Owner of the procedure
RESULTNUM	INTEGER	4	Number of the procedure result column
LENGTH	INTEGER	4	Maximum length of the result column if TYPECODE is 3 (VARCHAR), 9 (NATIVE VARCHAR), or 14 (VARBINARY) or number of bytes in the result column for all other data types
TYPECODE	SMALLINT	2	Data type of the result column: 0 for INTEGER or SMALLINT (these two are distinguished by the LENGTH field) 1 for BINARY 2 for CHAR (ASCII only) 3 for VARCHAR (ASCII only) 4 for FLOAT or REAL (these two are distinguished by the LENGTH field) 5 for DECIMAL 6 for TID (for ALLBASE/SQL use only) 7 reserved 8 for NATIVE CHAR 9 for NATIVE VARCHAR 10 for DATE 11 for TIME 12 for DATETIME 13 for INTERVAL 14 for VARBINARY 15 for LONG BINARY 16 for LONG VARBINARY
NULLS	SMALLINT	2	Nullability: 0 if the result column cannot contain null values 1 if the result column can contain null values

System.ProcResult

Table 8-27. System.ProcResult (continued)

Name	Type	Length	Description
PRECISION	SMALLINT	2	Number of significant decimal or binary digits in the number (excluding the sign and the decimal point)
SCALE	SMALLINT	2	Number of digits after the decimal point
LANGUAGEID	SMALLINT	2	Code for the language of this column. Run NLUTIL.PUB.SYS to display a complete list of native languages and codes for your system. A value of -1 means <i>not applicable</i> (for non-character type columns)

Example

```
select procedure, owner, resultnum, length from system.procrresult;
-----+-----+-----+-----
PROCEDURE      |OWNER      |RESULTNUM |LENGTH
-----+-----+-----+-----
PROCESS12     |MANUFDB   |          1|      30
PROCESS12     |MANUFDB   |          2|      20
-----+-----+-----+-----

Number of rows selected is 2
U[p], d[own], l[left], r[right], t[op], b[ottom], pr[int] <n>, or e[nd] >

select typecode, nulls, precision, scale, languageid from system.procrresult;
-----+-----+-----+-----+-----
TYPECODE|NULLS |PRECISION|SCALE |LANGUAGEID
-----+-----+-----+-----+-----
        0|    0    |        5|    0|        -1
        2|    1|    0|    0|         0
-----+-----+-----+-----+-----

Number of rows selected is 2
U[p], d[own], l[left], r[right], t[op], b[ottom], pr[int] <n>, or e[nd] >
```

System.Rule

SYSTEM.RULE contains detailed information about the rules defined for database tables.

Initially, the table is empty. ALLBASE/SQL updates this table when processing a CREATE RULE, DROP RULE, or TRANSFER OWNERSHIP statement that affects a rule.

This table is accessed whenever an INSERT, UPDATE, or DELETE is performed on a table to determine whether any rules need to be enforced. If there is a rule for the table that is enforced at UPDATE time, the SYSTEM.RULECOLUMN table is also checked.

Table 8-28. System.Rule

Column Name	Type	Length	Description
RULENAME	CHAR	20	Name of the rule
OWNER	CHAR	20	Owner of the target table and rule
TABLENAME	CHAR	20	Name of the target table
OLDNAME	CHAR	20	Old Correlation Name
NEWNAME	CHAR	20	New Correlation Name
DELETE	CHAR	2	DELETE applicable for this rule Y Yes N No
INSERT	CHAR	2	INSERT applicable for this rule Y Yes N No
UPDATE	CHAR	2	UPDATE applicable for this rule Y Yes on all columns N No on any columns C Yes on specific columns only
NUMC	INTEGER	4	Number of columns applicable for UPDATE on this rule
CTIME	CHAR	16	Time of creation: yyymmddhhmmsstt
DBEFILESET	CHAR	20	Name of the DBEFileSet containing the rule's definition and stored sections

System.Rule

Example

```
select rulename, owner, tablename, oldname from system.rule;
```

RULENAME	OWNER	TABLENAME	OLDNAME
INSERTREPORT	PURCHDB	REPORTS	OLD
DELETEREPORT	PURCHDB	REPORTS	OLD
UPDATEREPORT	PURCHDB	REPORTS	OLD
CHECKVENDOR	PURCHDB	VENDORS	OLD

First 4 rows have been selected.

```
U[p], d[own], l[left], r[ight], t[op], b[ottom], pr[int] <n>,or e[nd]> r
```

```
select newname, delete, insert, update,numc, ctime from system.rule;
```

NEWNAME	DELETE	INSERT	UPDATE	NUMC	CTIME
NEW	N	Y	N		0 1991121013511100
NEW	Y	N	N		0 1991121013511200
NEW	N	N	C		2 1991121013511200
NEW	Y	N	N		0 1991121216034600

First 4 rows have been selected.

```
U[p], d[own], l[left], r[ight], t[op], b[ottom], pr[int] <n>,or e[nd]> r
```

```
select dbfilesset from system.rule;
```

```
DBFILESSET
```

```
PURCHFS  
PURCHFS  
PURCHFS  
PURCHFS
```

First 4 rows have been selected.

```
U[p], d[own], l[left], r[ight], t[op], b[ottom], pr[int] <n>,or e[nd]> r
```


System.RuleColumn

SYSTEM.RULECOLUMN records the columns listed for the UPDATE statement type in the rule.

Initially, the table is empty, since the system catalog does not use rules. ALLBASE/SQL updates the table when processing a CREATE RULE, DROP RULE, or TRANSFER OWNERSHIP involving a rule.

RULENAME and OWNER uniquely identify a particular rule. The table it is defined upon may be deduced by searching the SYSTEM.RULE table for a match on these two columns. The SYSTEM.RULE view contains the TABLENAME.

Table 8-29. System.RuleColumn

Column Name	Type	Length	Description
RULENAME	CHAR	20	Name of the rule
OWNER	CHAR	20	Owner of the rule
COLUMNNAME	CHAR	20	Name of the column for which an UPDATE results in firing the rule

Example

```
select * from system.rulecolumn;
```

```
-----+-----+-----
RULENAME      |OWNER      |COLUMNNAME
-----+-----+-----
UPDATEREPORT  |PURCHDB    |REPORTOWNER
UPDATEREPORT  |PURCHDB    |REPORTNAME
```

```
-----
Number of rows selected is 2
```

```
U[p], d[own], l[eft], r[ight], t[op], b[ottom], pr[int] <n>, or e[nd] >
```

System.RuleDef

SYSTEM.RULEDEF contains character strings that can be used by SQLGEN to generate the rules associated with a table when recreating a DBEnvironment. SYSTEM.RULEDEF stores the part of the rule string that begins with the WHERE clause and continues to the end of the string (not including the final semicolon).

SYSTEM.RULEDEF is initially empty, and it is updated whenever ALLBASE/SQL processes a CREATE RULE, DROP RULE, or TRANSFER OWNERSHIP statement on a rule.

When the rule is dropped, the rows making up the character string of the rule definition are deleted. If a TRANSFER OWNERSHIP is done on the rule, the owner field in this table is updated with the name of the new owner.

Table 8-30. System.RuleDef

Column Name	Type	Length	Description
RULENAME	CHAR	20	Name of the rule
OWNER	CHAR	20	Owner of the rule
QUALIFIER	CHAR	20	Owner name to be used to qualify any unqualified objects referenced in the rule definition.
SEGNUM	INTEGER	4	Segment Number
SEGLen	INTEGER	4	Length of segment in bytes (up to 64)
RULESTRING	CHAR	64	RULE character string segment

Example

```
select * from system.ruledef;
-----+-----+-----+-----+
RULENAME      |OWNER      |QUALIFIER    |SEGNUM      |
-----+-----+-----+-----+
INSERTREPORT  |PURCHDB    |PETERW      |            |1|
INSERTREPORT  |PURCHDB    |PETERW      |            |2|
DELETEREPORT  |PURCHDB    |PETERW      |            |1|
DELETEREPORT  |PURCHDB    |PETERW      |            |2|
UPDATEREPORT  |PURCHDB    |PETERW      |            |1|
UPDATEREPORT  |PURCHDB    |PETERW      |            |2|
-----+-----+-----+-----+
Number of rows selected is 6
U[p], d[own], l[eft], r[ight], t[op], b[ottom], pr[int] <n>, or e[nd] > e
```

```
isql=> select seglen, rulestring from system.ruledef;
```

```
select seglen, rulestring from system.ruledef;
```

```
-----+-----  
SEGLEN      | RULESTRING  
-----+-----  
          57| execute procedure PurchDB.ReportMonitor (NEW.ReportName,  
          26| NEW.ReportOwner, 'INSERT')  
          57| execute procedure PurchDB.ReportMonitor (OLD.ReportName,  
          26| OLD.ReportOwner, 'DELETE')  
          57| execute procedure PurchDB.ReportMonitor (NEW.ReportName,  
          26| NEW.ReportOwner, 'UPDATE')  
-----+-----
```

Number of rows selected is 6

U[p], d[own], l[eft], r[ight], t[op], b[ottom], pr[int] <n>, or e[nd] > e

System.Section

SYSTEM.SECTION contains information about modules, sections, and views. It initially contains the definitions of the system views, and is updated whenever ALLBASE/SQL processes a statement that creates, modifies, or drops an object on which a section depends.

You can use the following query on this table to determine which sections are invalid:

```
isql=> SELECT OWNER, NAME, STYPE FROM System.Section  
> WHERE Valid=0;
```

Table 8-31. System.Section

Column Name	Type	Length	Description
NAME	CHAR	20	Name of the module, procedure, or view
OWNER	CHAR	20	Owner of the section or view
DBEFILESET	CHAR	20	Name of the DBEFileSet containing the section or view
SECTION	INTEGER	4	Section number: 0 for views 1 to <i>n</i> for sections (numbers are sequentially assigned)
CTIME	CHAR	16	Time of creation: yyymmddhhmmsstt
TYPE	INTEGER	2	Type of SQL statement represented in the section: 0 for sections that are neither views nor cursors 1 for cursors 2 for views
STYPE	INTEGER	2	Type of section: 0 for module section 1 for procedure section
VALID	INTEGER	2	Validity indicator: 0 for view or invalid section 1 for procedure
OPTFLAG	INTEGER	2	Setopt indicator: 0 optimization plan not specified by SETOPT 1 optimization plan specified by SETOPT (See System.Setoptinfo)

Example

```
SELECT * FROM System.Section;
```

NAME	OWNER	DBFILESET	SECTION
TABLE	SYSTEM	SYSTEM	0
COLUMN	SYSTEM	SYSTEM	0
INDEX	SYSTEM	SYSTEM	0
SECTION	SYSTEM	SYSTEM	0
DBFILESET	SYSTEM	SYSTEM	0
DBFILE	SYSTEM	SYSTEM	0
SPECAUTH	SYSTEM	SYSTEM	0
TABAUTH	SYSTEM	SYSTEM	0
COLAUTH	SYSTEM	SYSTEM	0
MODAUTH	SYSTEM	SYSTEM	0
GROUP	SYSTEM	SYSTEM	0
PARTINFO	PURCHDB	SYSTEM	0
VENDORSTATISTICS	PURCHDB	SYSTEM	0
CEXP06	JOHN@BROCK	SYSTEM	1
CEXP06	JOHN@BROCK	SYSTEM	2
CEXP06	JOHN@BROCK	SYSTEM	3

First 16 rows have been selected.

```
U[p], d[own], l[eft], r[ight], t[op], b[ottom], pr[int] <n>,or e[nd]> r
```

CTIME	TYPE	STYPE	VALID
1985090614181790	2	0	0
1985090614182250	2	0	0
1985090614182800	2	0	0
1985090614183320	2	0	0
1985090614183950	2	0	0
1985090614184680	2	0	0
1985090614185320	2	0	0
1985090614190000	2	0	0
1985090614190470	2	0	0
1985090614191190	2	0	0
1985090614191690	2	0	0
1985100915341710	2	0	0
1985100915342620	2	0	0
1985101211291030	0	0	0
1985101211291510	0	0	0
1985101211292020	0	0	0

First 16 rows have been selected.

```
U[p], d[own], l[eft], r[ight], t[op], b[ottom], pr[int] <n>,or e[nd]> e
```

System.Setoptinfo

The SYSTEM.SETOPTINFO view displays the access optimization plan of a section as specified by the SETOPT statement. The information is stored in a series of segments of up to 64 bytes. The access plan in SYSTEM.SETOPTINFO is used by the section if the OPTFLAG column of SYSTEM.SECTION is set to 1. SYSTEM.SETOPTINFO is initially empty, and it is updated whenever ALLBASE/SQL processes a SETOPT statement.

Table 8-32. System.Setoptinfo

Column Name	Type	Length	Description
NAME	CHAR	20	Name of the section
OWNER	CHAR	20	Owner of the section
SECTION	INTEGER	4	Section number
TYPE	INTEGER	2	Type of section: 0 for module section 1 for procedure section
SEGNUM	INTEGER	4	Segment number
SEGLEN	INTEGER	4	Segment length (Up to 64)
SETOPTSTRING	CHAR	64	SETOPT byte string segment

Example

```
SELECT * FROM System.Setoptinfo;
```

```
-----+-----+-----+-----+
NAME          |OWNER          |SECTION      |TYPE          |
-----+-----+-----+-----+
CEX9          |PROGM1        |             |1|           |0|
CEX9          |PROGM1        |             |2|           |0|
CEX9          |PROGM1        |             |3|           |0|
CEX9          |PROGM1        |             |4|           |0|
CEX9          |PROGM1        |             |5|           |0|
CEX9          |PROGM1        |             |6|           |0|
```

First 6 rows have been selected.

```
U[p], d[own], l[eft], r[ight], t[op], b[ottom], pr[int] <n>,or e[nd]> r
```

```
-----+-----+-----+-----+
SEGNUM        |SEGLEN        |SETOPTSTRING
-----+-----+-----+
1|            |25|setopt general indexscan;
1|            |25|setopt general indexscan;
1|            |25|setopt general indexscan;
1|            |25|setopt general indexscan;
1|            |25|setopt general indexscan;
1|            |25|setopt general indexscan;
```

Number of rows selected is 6

```
U[p], d[own], l[eft], r[ight], t[op], b[ottom], pr[int] <n>,or e[nd]> e
```

System.Spaceauth

SYSTEM.SPACEAUTH lists all users and authorization groups for which TABLESPACE or SECTIONSPACE authority has been granted.

Initially SYSTEM.SPACEAUTH contains a row having the column values PUBLIC, SYSTEM, Y, Y. That is, all users have TABLESPACE and SECTIONSPACE authority in the SYSTEM DBEFileSet. A DBA can modify these authorities with further revokes and grants as desired.

ALLBASE/SQL accesses SYSTEM.SPACEAUTH whenever a user attempts to put a table or long column or a section (generated by a PREPARE or DECLARE CURSOR statement) into an explicit DBEFileSet. The owner of the table or section must have TABLESPACE or SECTIONSPACE authority, respectively, on the DBEFileSet.

Table 8-33. System.Spaceauth

Column Name	Type	Length	Description
USERID	CHAR	20	Authorized DBEUserID or authorization group
DBEFILESET	CHAR	20	DBEFileSet name for SECTIONSPACE or TABLESPACE authority
TABLESPACE	CHAR	2	Resource authority: Y (The user has authority to store table and long column data in this DBEFileSet.) N (The user cannot store table and long column data in this DBEFileSet.)
SECTIONSPACE	CHAR	2	Resource authority: Y (The user has authority to store sections in this DBEFileSet.) N (The user cannot store sections in this DBEFileSet.)

Example

```

SELECT * FROM System.Spaceauth;
-----+-----+-----+-----
USERID          |DBEFILESET|TABLESPACE|SECTIONSPACE
-----+-----+-----+-----
PUBLIC          |SYSTEM    |Y         |Y
PUBLIC          |PURCHFS   |Y         |Y
PUBLIC          |WAREHFS   |Y         |Y
PUBLIC          |ORDERSFS  |Y         |Y
PUBLIC          |FILESFS   |Y         |Y
PUBLIC          |RECF      |Y         |Y
-----+-----+-----+-----
Number of rows selected is 6
U[p], d[own], l[eft], r[ight], t[op], b[ottom], pr[int] <n>, or e[nd]> e

```

System.Spacedefault

SYSTEM.SPACEDEFAULT contains the default TABLESPACE and SECTIONSPACE DBEFileSets for PUBLIC.

Initially SYSTEM.SPACEDEFAULT contains two rows, one having the column values PUBLIC, SYSTEM, 1 and the other having the column values PUBLIC, SYSTEM, 2. Thus SYSTEM is the default DBEFileSet for table, long column, and section data for PUBLIC.

ALLBASE/SQL accesses SYSTEM.SPACEDEFAULT whenever a table, long column, rule, check constraint, procedure, or section (generated by a PREPARE, DECLARE CURSOR, or CREATE VIEW statement) is created without the IN *DBEFileSetName* clause.

Table 8-34. System.Spacedefault

Column Name	Type	Length	Description
USERID	CHAR	20	Authorized DBEUserID or authorization group
DBEFILESET	CHAR	20	Name of a DBEFileSet
SPACETYPE	SMALLINT	2	Type of default DBEFileSet: 1 (SECTIONSPACE) 2 (TABLESPACE)

Example

```
SELECT * FROM System.Spacedefault;
```

```
-----+-----+-----  
USERID      |DBEFILESET      |SPACETYPE  
-----+-----+-----  
PUBLIC      |SYSTEM          |          1  
PUBLIC      |SYSTEM          |          2
```

```
-----  
Number of rows selected is 2
```

```
U[p], d[own], l[eft], r[ight], t[op], b[ottom], pr[int] <n>,or e[nd]> e
```


System.Specauth

SYSTEM.SPECAUTH lists all users and authorization groups and the special authorities they have been granted. Because a user must be granted the special authority CONNECT before being able to access the DBEnvironment, every user or group that has CONNECT authority to the DBEnvironment is listed.

ALLBASE/SQL updates SYSTEM.SPECAUTH when processing a GRANT or REVOKE statement specifying a special authority, or when dropping a group with special authority.

SYSTEM.SPECAUTH, along with SYSTEM.COLAUTH, SYSTEM.INSTALLAUTH, SYSTEM.MODAUTH, SYSTEM.PROCAUTH, SYSTEM.SPACEAUTH, SYSTEM.SPACDEFAULT, and SYSTEM.TABAUTH, contains the security scheme for the DBEnvironment.

CATALOG.SPECAUTH is identical in format to SYSTEM.SPECAUTH; it permits users without DBA authority or SELECT authority on SYSTEM.SPECAUTH to examine rows to which they have access.

Table 8-35. System.Specauth

Column Name	Type	Length	Description
USERID	CHAR	20	Authorized DBEUserID or authorization group
DBA	CHAR	2	DBA authority: Y for yes N for no
RESOURCE	CHAR	2	Resource authority: Y for yes N for no
CONNECT	CHAR	2	Connect Authority: Y for yes N for no
MONITOR	CHAR	2	Connect Authority: Y for yes N for no

System.Specauth

Example

```
SELECT * FROM System.Specauth;
```

USERID	DBA	RESOURCE	CONNECT	MONITOR
HPRDBSS	N	N	N	N
TEMP	N	N	N	N
PUBLIC	N	N	N	N
SYSTEM	N	N	N	N
JOHN@BROCK	Y	N	N	N
PURCHDBMAINT	N	Y	N	N
PURCH	N	N	Y	N
MICHELE@DING	Y	N	N	N
GEORGE@DBMS	N	N	Y	Y

Number of rows selected is 8

U[p], d[own], l[eft], r[ight], t[op], b[ottom], pr[int] <n>, or e[nd]> e

System.Tabauth

SYSTEM.TABAUTH contains the authorities for operations on tables and views. If a user has been granted authority to operate on only specific columns instead of the entire table or view, you must also use SYSTEM.COLAUTH to determine the actual columns.

This view initially contains entries for the CATALOG views. The view is updated whenever ALLBASE/SQL processes a GRANT or REVOKE statement containing a table or view related authority, and when PUBLIC or PUBLICREAD tables are created. It is also updated when a DROP TABLE, DROP VIEW, or table-related TRANSFER OWNERSHIP statement is processed.

SYSTEM.TABAUTH, along with SYSTEM.COLAUTH, SYSTEM.MODAUTH, SYSTEM.PROCAUTH, and SYSTEM.SPECAUTH, contains the security scheme for the DBEnvironment. ALLBASE/SQL verifies a user's authority in SYSTEM.TABAUTH if the appropriate authority was not contained in SYSTEM.SPECAUTH. If the UPDATE or REFERENCES column contains a C, ALLBASE/SQL verifies the user's UPDATE authority in SYSTEM.COLAUTH.

When you create a PUBLIC or PUBLICREAD table, ALLBASE/SQL implicitly grants table authorities to the special user PUBLIC. In this case, the GRANTOR column contains the table owner name and the GRANTABLE column contains an N to indicate that privileges cannot be granted.

Table 8-36. System.Tabauth

Column Name	Type	Length	Description
USERID	CHAR	20	Authorized DBEUserID or authorization group
NAME	CHAR	20	Name of the table or view on which the user has one or more authorities
OWNER	CHAR	20	Owner of the table or view on which the user has one or more authorities
NCOL	INTEGER	4	Number of columns the user has update authority on (0 if the UPDATE column contains a Y)
NRCOL	INTEGER	4	Number of columns the user has references authority on (0 if the REFERENCES column contains a Y or N)
SELECT	CHAR	2	SELECT authority on the table or view: Y for yes N for no
INSERT	CHAR	2	INSERT authority on the table or view: Y for yes N for no
UPDATE	CHAR	2	UPDATE authority on the table or view: Y for yes on all columns N for no on all columns C for yes on specific columns only

System.Tabauth

Table 8-36. System.Tabauth (continued)

Column Name	Type	Length	Description
DELETE	CHAR	2	DELETE authority on the table or view: Y for yes N for no
ALTER	CHAR	2	ALTER authority on the table (for adding columns): Y for yes N for no
INDEX	CHAR	2	INDEX authority on the table (for creating indexes): Y for yes N for no or for a view
REFERENCES	CHAR	2	REFERENCES authority on the table (for creating referential constraints that refer to this table): Y for yes on all columns N for no on all columns or for a view C for yes on specific columns only
GRANTOR	CHAR	20	Name of the grantor of the privileges described in this row or blank if column privileges have differing grantabilities
GRANTABLE	CHAR	2	GRANTABLE privilege on the table or view: Y for yes, the user can grant these privileges to others N for no, the user cannot grant these privileges to others blank if column privileges have differing grantabilities

Example

```
SELECT * FROM System.Tabauth;
```

```
-----+-----+-----+-----+-----+-----+-----+-----+-----+
USERID          |NAME          |OWNER          |NCOL          |
-----+-----+-----+-----+-----+-----+-----+-----+-----+
PUBLIC          |PARTS         |PURCHDB        |              | 0
PUBLIC          |INVENTORY     |PURCHDB        |              | 0
PUBLIC          |SUPPLYPRICE   |PURCHDB        |              | 0
PUBLIC          |VENDORS       |PURCHDB        |              | 0
PUBLIC          |ORDERS        |PURCHDB        |              | 0
PUBLIC          |ORDERITEMS    |PURCHDB        |              | 0
PUBLIC          |MESSAGE       |PURCHDB        |              | 0
-----+-----+-----+-----+-----+-----+-----+-----+-----+
```

```
Number of rows selected is 7
```

```
U[p], d[own], l[eft], r[ight], t[op], b[ottom], pr[int] <n>,or e[nd]> r
```

```
-----+-----+-----+-----+-----+-----+-----+-----+-----+
|NRCOL      |SELECT|INSERT|UPDATE|DELETE|ALTER|INDEX|REFERENCES|GRANTOR
-----+-----+-----+-----+-----+-----+-----+-----+-----+
|           |O|Y   |Y     |Y     |Y     |Y     |Y     |N         |PURCHDB
|           |O|Y   |Y     |Y     |Y     |Y     |Y     |N         |PURCHDB
|           |O|Y   |Y     |Y     |Y     |Y     |Y     |N         |PURCHDB
|           |O|Y   |Y     |Y     |Y     |Y     |Y     |N         |PURCHDB
|           |O|Y   |Y     |Y     |Y     |Y     |Y     |N         |PURCHDB
|           |O|Y   |Y     |Y     |Y     |Y     |Y     |N         |PURCHDB
|           |O|Y   |Y     |Y     |Y     |Y     |Y     |N         |PURCHDB
|           |O|Y   |Y     |Y     |Y     |Y     |Y     |N         |PURCHDB
-----+-----+-----+-----+-----+-----+-----+-----+-----+
```

```
Number of rows selected is 7
```

```
U[p], d[own], l[eft], r[ight], t[op], b[ottom], pr[int] <n>,or e[nd]> r
```

```
+-----+
|GRANTABLE
+-----+
|N
|N
|N
|N
|N
|N
|N
|N
|N
-----+-----+-----+-----+-----+-----+-----+-----+-----+

```

```
First 7 rows have been selected.
```

```
U[p], d[own], l[eft], r[ight], t[op], b[ottom], pr[int] <n>,or e[nd]> e
```

System.Table

SYSTEM.TABLE contains a record of each table and view in the DBEnvironment including one for itself and one for each of the other system views. The columns for the tables and views contained in this table are described in SYSTEM.COLUMN.

Table 8-37. System.Table

Column Name	Type	Length	Description
NAME	CHAR	20	Name of the table or view
OWNER	CHAR	20	Owner of the table or view
DBEFILESET	CHAR	20	Name of the DBEFileSet containing the table or view
TYPE	SMALLINT	2	Type of object: 0 for table 1 for view
RTYPE	SMALLINT	2	Locking mode for the table: 0 for view 1 for PUBLICREAD table 2 for PRIVATE table 3 for PUBLIC table 4 for temporary table 5 for PUBLICROW table
NUMC	INTEGER	4	Number of columns in the table or view
NUMI	INTEGER	4	The number of indexes (including HASH and constraint structures) on a table; 0 for a view
NUMIC	INTEGER	4	The number of constraints on the table or view
NUMR		4	The number or rules on the table or view
NPAGES	INTEGER	4	Number of data pages containing the table; 0 for all views except system views
NROWS	INTEGER	4	Number of rows in a table; 0 for all views except system views
AVGLEN	INTEGER	4	Average row length of the table; 0 for all views except system views
CTIME	CHAR	16	Time of creation: yyyyymmddhhmmsstt
USTIME	CHAR	16	Time of the most recent execution of the UPDATE STATISTICS statement
LANGUAGEID	SMALLINT	2	Code for the language of this table. Run NLUTIL.PUB.SYS to display a complete list of native languages and codes for your system. A value of -1 means NOT APPLICABLE (that is, the entry is a view, not a table)

Table 8-37. System.Table (continued)

Column Name	Type	Length	Description
PARTITION	CHAR	20	Name of the partition containing the table: NONE for a view DEFAULT if not specified

Example

```
SELECT * FROM System.Table;
```

```
-----+-----+-----+-----+
NAME          |OWNER          |DBEFILESET    |TYPE
-----+-----+-----+-----+
COUNTER       |SYSTEM        |SYSTEM        |0
USER          |SYSTEM        |SYSTEM        |0
TRANSACTION   |SYSTEM        |SYSTEM        |0
CALL          |SYSTEM        |SYSTEM        |0
ACCOUNT       |SYSTEM        |SYSTEM        |0
TABLE         |SYSTEM        |SYSTEM        |1
COLUMN        |SYSTEM        |SYSTEM        |1
INDEX         |SYSTEM        |SYSTEM        |1
SECTION       |SYSTEM        |SYSTEM        |1
DBEFILESET    |SYSTEM        |SYSTEM        |1
DBEFILE       |SYSTEM        |SYSTEM        |1
SPECAUTH      |SYSTEM        |SYSTEM        |1
TABAUTH       |SYSTEM        |SYSTEM        |1
COLAUTH       |SYSTEM        |SYSTEM        |1
MODAUTH       |SYSTEM        |SYSTEM        |1
GROUP         |SYSTEM        |SYSTEM        |1
-----+-----+-----+-----+
```

First 16 rows have been selected.

```
U[p], d[own], l[eft], r[ight], t[op], b[ottom], pr[int] <n>,or e[nd]> r
```

```
-----+-----+-----+-----+-----+-----+
|RTYPE|NUMC  |NUMI  |NUMIC |NUMR  |NPAGES
-----+-----+-----+-----+-----+
| 3|    3|    3|    0|    0|    0|    0
| 3|    3|    2|    0|    0|    0|    0
| 3|    4|    4|    0|    0|    0|    0
| 3|    5|    5|    0|    0|    0|    0
| 3|    6|    6|    0|    0|    0|    0
| 0|   12|   12|    0|    0|    0|    2
| 0|   10|   10|    0|    0|    0|    9
| 0|   10|   10|    0|    0|    0|    1
| 0|    7|    7|    0|    0|    0|    2
| 0|    5|    5|    0|    0|    0|    1
| 0|    7|    7|    0|    0|    0|    1
| 0|    4|    4|    0|    0|    0|    1
| 0|   10|   10|    0|    0|    0|    1
| 0|    4|    4|    0|    0|    0|    0
| 0|    3|    3|    0|    0|    0|    0
| 0|    4|    4|    0|    0|    0|    1
-----+-----+-----+-----+-----+-----+
```

First 16 rows have been selected.

```
U[p], d[own], l[eft], r[ight], t[op], b[ottom], pr[int] <n>,or e[nd]> r
```

System.Table

NRROWS	AVGLEN	CTIME	USTIME	LANGUAGEID
0	32	1985090614175200	Not done yet	0
0	24	1985090614175200	Not done yet	0
0	32	1985090614175200	Not done yet	0
0	52	1985090614175200	Not done yet	0
0	40	1985090614175200	Not done yet	0
39	140	1985090614181790	1985101613380630	-1
258	108	1985090614182250	1985101613351330	-1
9	160	1985090614182800	1985101613362650	-1
44	128	1985090614183320	1985101613365940	-1
4	68	1985090614183950	1985101613355530	-1
9	122	1985090614184680	1985101613354100	-1
10	26	1985090614185320	1985101613373740	-1
7	84	1985090614190000	1985101613375100	-1
0	0	1985090614190470	1985101613345170	-1
0	0	1985090614191190	1985101613364530	-1
38	64	1985090614191690	1985101613361280	-1

First 16 rows have been selected.

U[p], d[own], l[left], r[right], t[op], b[ottom], pr[int] <n>,or e[nd]> e

PARTITION
NONE
NONE
NONE
NONE
NONE
NONE
NONE
NONE
NONE
NONE
NONE
NONE
NONE
NONE
NONE
NONE
NONE
NONE
NONE
NONE

First 16 rows have been selected.

U[p], d[own], l[left], r[right], t[op], b[ottom], pr[int] <n>,or e[nd]> e

System.TempSpace

SYSTEM.TEMPSPACE contains information about TempSpace names. This view is initially empty, but is updated whenever ALLBASE/SQL processes a CREATE TEMPSPACE or DROP TEMPSPACE. You can use this view to determine the defined TempSpaces.

```
isql=> SELECT * from SYSTEM.TEMPSPACE;
```

Table 8-38. System.TempSpace

Column Name	Type	Length	Description
TEMPSNAME	CHAR	20	Name of the TempSpace
LOCATION	CHAR	8	System identifier for the group being used for TempSpace files
MAXFILEPAGES	INTEGER	4	Maximum number of pages per file opened in the TempSpace
CTIME	CHAR	16	Time of creation: yyymmddhhmmsstt

Example

```
select * from system.tempspace;
-----+-----
TEMPSNAME      |LOCATION
-----+-----
MJTMPSP        |TEMPGRP
-----+-----
Number of rows selected is 1
U[p], d[own], l[eft], r[ight], t[op], b[ottom], pr[int] <n>, or e[nd] > r

+-----+-----+
|MAXFILEPAGES|CTIME
+-----+-----+
|           128|1990041216384500
-----+-----

Number of rows selected is 1
U[p], d[own], l[eft], r[ight], t[op], b[ottom], pr[int] <n>, or e[nd] > e
-----+-----
```

System.TPIndex

SYSTEM.TPINDEX contains an entry for each third-party index associated with the IMAGE datasets associated with this DBEnvironment. Third-party indexes cannot be used in association with normal ALLBASE/SQL tables. In ALLBASE/SQL terms the datasets are viewed as tables within the DBEnvironment.

Table 8-39. System.TPIndex

Column Name	Type	Length	Description
INDEXNAME	CHAR	20	Name of the index
TABlename	CHAR	20	Name of the table on which the index or hash structure is defined
OWNER	CHAR	20	Owner of the table on which the index is defined
UNIQUE	SMALLINT	2	Uniqueness indicator: 0 if duplicates are allowed, that is, the index is not unique 1 if duplicates are not allowed, that is the index is unique
CLUSTER	SMALLINT	2	Clustering indicator: 0 if the index is not a clustering index 1 if the index is the clustering index for the table
NUMC	INTEGER	4	Number of columns in the index or hash key
COLNUMS	BINARY	64	A vector of column numbers, each of type SMALLINT, identifying the columns the index is defined over. In ISQL, each SMALLINIT (two-byte) entry is displayed as a field of 4 hexadecimal digits.
NPAGES	INTEGER	4	Number of index pages containing the index
NLEVELS	INTEGER	4	Number of B-tree levels
NLEAVES	INTEGER	4	Number of B-tree leaf pages
NDISTINCT	INTEGER	4	Number of distinct keys
NFIRST	INTEGER	4	Number of distinct first column values of the B-tree key
NPERKEY	INTEGER	4	Number of pages per B-tree key

Table 8-39. System.TPIndex (continued)

Column Name	Type	Length	Description
CCOUNT	INTEGER	4	Cluster count; indicates how well the data of the index is sorted 0 before first UPDATE STATISTICS statement is processed n efficiency of clustering: best clustering if n=NPAGES of table indexed; worst if n=NROWS of table indexed
CTIME	CHAR	16	Time of creation: yyyyymmddhhsstt
COLDIRS	BINARY	64	A vector of direction entries, each of type SMALLINT indicating the direction of the corresponding column in the index definition. In ISQL, each SMALLINT (two-byte) entry is displayed as a field of 4 hexadecimal digits. 5 ASC (Ascending) 6 DESC (Descending)

System.Transaction

SYSTEM.TRANSACTION is a pseudotable that records the transactions of users currently accessing the DBEnvironment.

SYSTEM.TRANSACTION supports an unlimited number of users, transactions, and sessions.

Table 8-40. System.Transaction

Column Name	Type	Length	Description
USERID	CHAR	20	DBEUserID of the user who started the transaction
CID	INTEGER	4	Unique connection identifier
SID	INTEGER	4	Session identifier. If the same USERID has multiple connections to the same DBEnvironment from the same application (including ISQL), all connections have the same SID. If the user has multiple connections to the same DBEnvironment from separate applications, each application (or ISQL session) will have a different SID. For more information about multiple DBEnvironment connections, see the section “Using Multiple Connections and Transactions with Timeouts” in the “Using ALLBASE/SQL” chapter of the <i>ALLBASE/SQL Reference Manual</i> .
XID	INTEGER	4	Unique transaction identifier
PRIORITY	INTEGER	4	Transaction priority: 0 (highest) - 255 (lowest)
ISOLATION LEVEL	CHAR	2	Transaction isolation level: RR (Repeatable Read, Serializable) CS (Cursor Stability) RU (Read Uncommitted) RC (Read Committed)
LABEL	CHAR	8	Transaction label string

Example

```
SELECT * FROM System.Transaction;
```

```
-----+-----+-----+-----+-----
USERID          |CID      |SID      |XID      |PRIORITY
-----+-----+-----+-----+-----
JOHN@BROCK     |      108|      108|    11320|      127
-----+-----+-----+-----+-----
```

Number of rows selected is 1

U[p], d[own], l[eft], r[ight], t[op], b[ottom], pr[int] <n>, or e[nd]> r

```
-----+-----
ISOLATION LEVEL |LABEL
-----+-----
RC              +Xact2
-----+-----
```

Number of rows selected is 1

System.User

SYSTEM.USER is a pseudotable that contains a row for each user currently accessing the DBEnvironment.

Table 8-41. System.User

Column Name	Type	Length	Description
USERID	CHAR	20	DBEUserID
CID	INTEGER	4	Unique connection identifier
SID	INTEGER	4	Session identifier. If the same USERID has multiple connections to the same DBEnvironment from the same application (including ISQL), all connections have the same SID. If the user has multiple connections to the same DBEnvironment from separate applications, each application (or ISQL session) will have a different SID. For more information about multiple DBEnvironment connections, see the section "Using Multiple Connections and Transactions with Timeouts" in the "Using ALLBASE/SQL" chapter of the <i>ALLBASE/SQL Reference Manual</i> .

Example

```
SELECT * FROM System.User;
```

```
-----+-----+-----  
USERID      |CID      |SID  
-----+-----+-----  
JOHN@BROCK  |      108|      108  
-----+-----+-----
```

```
Number of rows selected is 1
```

```
U[p], d[own], l[eft], r[ight], t[op], b[ottom], pr[int] <n>, or e[nd]> e
```

System.ViewDef

The SYSTEM.VIEWDEF view displays information about view definitions, including the SELECT part of each CREATE VIEW statement used to create a view. The SELECT is stored in a series of segments of up to 64 bytes.

SYSTEM.VIEWDEF is initially empty, and it is updated whenever ALLBASE/SQL processes a CREATE VIEW statement, a DROP VIEW statement, or a TRANSFER OWNERSHIP statement on a view. System views are not included in SYSTEM.VIEWDEF.

When the view is dropped, the rows making up the byte string of the SELECT portion are deleted. If a TRANSFER OWNERSHIP is done on the view, the owner field in this table is updated with the name of the new owner.

Table 8-42. System.ViewDef

Column Name	Type	Length	Description
VIEWNAME	CHAR	20	Name of the view
OWNER	CHAR	20	Owner of the view
QUALIFIER	CHAR	20	Owner name to be used to qualify any unqualified objects referenced in the SELECT byte string
SEGNUM	INTEGER	4	Segment Number
SEGLEN	INTEGER	4	Length of segment in bytes (up to 64)
SELECTSTRING	CHAR	64	SELECT byte string segment

Example

```
SELECT * FROM System.ViewDef WHERE VIEWNAME = 'VENDORSTATISTICS';
```

```
-----+-----+-----+-----+
VIEWNAME      |OWNER      |QUALIFIER   |SEGNUM
-----+-----+-----+-----+
VENDORSTATISTICS |PURCHDB   |PURCHDB    |1
VENDORSTATISTICS |PURCHDB   |PURCHDB    |2
VENDORSTATISTICS |PURCHDB   |PURCHDB    |3
VENDORSTATISTICS |PURCHDB   |PURCHDB    |4
VENDORSTATISTICS |PURCHDB   |PURCHDB    |5
-----+-----+-----+-----+
```

First 5 rows have been selected.

```
U[p], d[own], l[eft], r[ight], t[op], b[ottom], pr[int] <n>,or e[nd]> r
```

```
-----+-----+-----+-----+
|SEGLEN|SELECTSTRING
-----+-----+-----+-----+
| 64| SELECT PurchDB.Vendors.VendorNumber,PurchDB.Vendors.VendorName
| 51| , OrderDate,OrderQty,OrderQty*PurchasePrice FROM
| 58| PurchDB.Vendors,PurchDB.Orders,PurchDB.OrderItems WHERE
| 63| PurchDB.Vendors.VendorNumber=PurchDB.Orders.VendorNumber AND
| 60| PurchDB.Orders.OrderNumber=PurchDB.OrderItems.OrderNumber;
-----+-----+-----+-----+
```

First 5 rows have been selected.

```
U[p], d[own], l[eft], r[ight], t[op], b[ottom], pr[int] <n>,or e[nd]> r
```

ALLBASE/SQL Limits

The tables below present logical and physical system limits for ALLBASE/SQL. The values show the maximums permitted by the ALLBASE/SQL software. In practice, lower maximums may be imposed by the operating system or hardware environment in which ALLBASE/SQL is installed. In addition, ALLBASE/SQL uses some space per page for internal data structures. This leaves less than a complete page available to the user for tuple data and header storage. Refer to the “Logical Design” and “Physical Design” chapters for methods of determining the amount of space available to the user.

Page size is 4096 bytes, except for log pages, which are 512 bytes.

Table A-1. System Control Limits

Name of Limit	Value
Multiuser DBEnvironments per system	No limit
Single-user DBEnvironments per system	No limit
Number of concurrent sessions	Limited by Runtime Control Block Allocation
Number of concurrent transactions	2 to 240 (when 240 are active, additional transactions are placed in a wait queue)
Number of concurrent connections per user	1 to 32
Number of concurrent users	No limit
Timeout value	1 to 2,147,483,647 seconds or 1 to 35,791,394 minutes

Table A-2. Logical Data Limits

Name of Limit	Value
Tables per DBE	$2^{31} - 1$
Maximum columns per table	1023
Bytes per row	$(NumberOfColumns+1)*2+SumOfColumnLengths \leq 4000$ (for the VARCHAR data type, <i>ColumnLength</i> equals <i>ColumnValueLength</i>)
Bytes per column	3996
Bytes per long column	$2^{31} - 1$
Views per DBE	$2^{31} - 1$
Maximum columns per view	1023
Maximum parameters per procedure	1023
Maximum result columns per procedure	1024

Table A-2. Logical Data Limits (continued)

Name of Limit	Value
Indexes per DBEnvironment	$2^{31} - 1$
Maximum columns per B-tree index or hash	16
Maximum columns per virtual index (PCR)	15
Maximum B-tree or hash key size	$(NumberOfKeyColumns + 2)*2 + SumofColumnLengths + 8 \leq 1024$
Maximum virtual index (PCR) key size	$(NumberOfKeyColumns + 3)*2 + SumofColumnLengths + 10 \leq 1024$
Maximum sort columns	1023
Maximum sort tuple size	$(NumberOfSortColumns + 1)*2 + SumOfSortColumnLengths \leq 4000$ (for the VARCHAR data type, <i>ColumnLength</i> equals <i>ColumnValueLength</i>)

Table A-3. Space Management Limits

Name of Limit	Value
Tables per statement (including tables underlying views)	31
Query blocks per query	16
Data and index page size	4096 bytes
Log page size	512 bytes
Pages per DBEFile (except DBEFile0)	2 to $2^{19} - 1$
Pages per DBEFile0 (one per DBEnvironment)	150 to $2^{19} - 1$
DBEFiles per DBEnvironment	32767
Pages per DBEFileSet (when there is only one)	$(32767)(2^{19} - 1)$
Pages per DBEnvironment (all DBEFiles combined)	$(32767)(2^{19} - 1)$
Log size (pages)	250 to $2^{22} - 1$
Maximum size of a single log file	4 gigabytes
Maximum number of log files	34
Maximum size of log file space	136 gigabytes
Number of data buffer pages	15 to 50000
Number of log buffer pages	24 to 1024
Number of runtime control block pages	17 to 2000
Total of data buffer pages and runtime control block pages	256 Mbytes
Number of shared memory objects	72
ISQL command size	32K bytes
Maximum levels of nesting for ISQL STARTs per user	10
SQL command size	32K bytes
Number of pages per file opened in a TempSpace (MaxFilePages)	128 to 524284

Authorities Required by ALLBASE/SQL Statements

This appendix lists the authorities required for each ALLBASE/SQL statement. Note that you do not have to have *all* of the authorities that are marked to perform an activity, you only need one of them. For more information, refer to the “Database Creation and Security” chapter and to the “DBEnvironment Configuration and Security” chapter. The *ALLBASE/SQL Reference Manual* contains complete descriptions of each statement.

Table B-1. Authorities Required By ALLBASE/SQL Statements

Statement	Authority Required
ADD DBEFILE	DBA
ADD TO GROUP	DBA or OWNER
ADVANCE	None
ALTER DBEFILE	DBA
ALTER TABLE	DBA, OWNER, or ALTER
BEGIN	None
BEGIN ARCHIVE	DBA
BEGIN DECLARE SECTION	None
BEGIN WORK	None
CHECKPOINT	DBA
CLOSE	None
COMMIT ARCHIVE	DBA
COMMIT WORK	None
CONNECT	DBA or CONNECT
CREATE DBEFILE	DBA
CREATE DBEFILESET	DBA
CREATE GROUP	DBA or RESOURCE
CREATE INDEX	DBA, OWNER, or INDEX
CREATE PARTITION	DBA
CREATE PROCEDURE	DBA or RESOURCE (and appropriate authorities on objects referenced by the procedure)
CREATE RULE	OWNER for the table and OWNER or EXECUTE for the procedure; or DBA
CREATE SCHEMA	DBA or RESOURCE
CREATE TABLE	DBA or RESOURCE
CREATE TEMPSPACE	DBA or RESOURCE
CREATE VIEW	DBA, OWNER, or SELECT (on relevant columns)

Table B-1. Authorities Required By ALLBASE/SQL Statements (continued)

Statement	Authority Required
DECLARE CURSOR	For a select cursor: DBA, OWNER, or SELECT (on relevant columns) and UPDATE (for an updatable cursor). For a procedure cursor: DBA or OWNER or EXECUTE (for the procedure).
DECLARE variable	None
DELETE	DBA, OWNER, or DELETE
DELETE WHERE CURRENT	DBA, OWNER, or DELETE
DESCRIBE	Same as for a SELECT or EXECUTE PROCEDURE that is described; otherwise, none
DISABLE AUDIT LOGGING	DBA
DISABLE RULES	DBA
DISCONNECT	None
DROP DBEFILE	DBA
DROP DBEFILESET	DBA
DROP GROUP	DBA or OWNER
DROP INDEX	DBA, OWNER, or INDEX
DROP MODULE	DBA or OWNER
DROP PARTITION	DBA
DROP PROCEDURE	DBA or OWNER
DROP RULE	DBA or OWNER
DROP TABLE	DBA or OWNER
DROP TEMPSPACE	DBA or OWNER
DROP VIEW	DBA or OWNER
ENABLE AUDIT LOGGING	DBA
ENABLE RULES	DBA
END DECLARE SECTION	None
EXECUTE (Interactive)	DBA, RESOURCE, or OWNER
EXECUTE (Programmatic)	Same as for statement being executed
EXECUTE IMMEDIATE	Same as for statement being executed
EXECUTE PROCEDURE	OWNER or EXECUTE for the procedure; or DBA
Execute an SQL program	RUN
FETCH	None
GENPLAN	Same as for statement being analyzed
GOTO	None
GRANT (Special)	DBA
GRANT (Table, View, RUN, EXECUTE)	DBA, OWNER, or user with grantable privilege (on tables and views)
GRANT WITH GRANT OPTION	Direct DBA, direct OWNER, or user with grantable privilege
GRANT BY	DBA
IF	None
INCLUDE	None
INSERT	DBA, OWNER, or INSERT

Table B-1. Authorities Required By ALLBASE/SQL Statements (continued)

Statement	Authority Required
Labeled Statement	None
LOCK TABLE	DBA, OWNER, or SELECT
OPEN	For a select cursor: DBA, OWNER, or SELECT (on relevant columns) and UPDATE (for an updatable cursor). For a procedure cursor: DBA or OWNER or EXECUTE (for the procedure).
PREPARE PRINT RAISE ERROR REFETCH RELEASE REMOVE DBEFILE REMOVE FROM GROUP RESET RETURN REVOKE (Special) REVOKE (Table, View, RUN, EXECUTE) ROLLBACK WORK	Same as for statement being prepared None None None None DBA DBA or OWNER DBA DBA DBA, OWNER, or grantable privilege; only DBA if table or view and if no CASCADE and chain of grants exists None
SAVEPOINT SELECT SET CONNECTION SET CONSTRAINTS SET DEFAULT DBEFILESET SET DML ATOMICITY SET MULTITRANSACTION SETOPT SET PRINTRULES SET SESSION SET TRANSACTION SET USER TIMEOUT SQLEXPLAIN START DBE START DBE NEW START DBE NEWLOG STOP DBE	None DBA, OWNER, or SELECT None None DBA None None None None DBA None None DBA DBA DBA and DBECreator status DBA
TERMINATE USER	DBA
TRANSFER OWNERSHIP	DBA or OWNER
TRUNCATE TABLE	DBA or OWNER
UPDATE	DBA, OWNER, or UPDATE
UPDATE STATISTICS	DBA or OWNER
UPDATE WHERE CURRENT	DBA, OWNER, or UPDATE

Table B-1. Authorities Required By ALLBASE/SQL Statements (continued)

Statement	Authority Required
VALIDATE	DBA or OWNER; RUN on module, or EXECUTE on procedure
WHENEVER	None
WHILE	None

SQL Syntax Summary

ADD DBEFILE

ADD DBEFILE *DBEFileName* TO DBEFILESET *DBEFileSetName*

ADD TO GROUP

ADD $\left\{ \begin{array}{l} \textit{DBEUserID} \\ \textit{GroupName} \\ \textit{ClassName} \end{array} \right\} [, \dots]$ TO GROUP *TargetGroupName*

ADVANCE

ADVANCE *CursorName* $\left[\text{USING [SQL] DESCRIPTOR} \left\{ \begin{array}{l} \textit{SQLDA} \\ \textit{AreaName} \end{array} \right\} \right]$

ALTER DBEFILE

ALTER DBEFILE *DBEFileName* SET TYPE = $\left\{ \begin{array}{l} \textit{TABLE} \\ \textit{INDEX} \\ \textit{MIXED} \end{array} \right\}$

ALTER TABLE

ALTER TABLE $[\textit{Owner} .] \textit{TableName}$ $\left\{ \begin{array}{l} \textit{AddColumnSpecification} \\ \textit{AddConstraintSpecification} \\ \textit{DropConstraintSpecification} \\ \textit{SetTypeSpecification} \\ \textit{SetPartitionSpecification} \end{array} \right\}$

AddColumnSpecification

ADD $\left\{ \left(\textit{ColumnDefinition} [, \dots] \right) \right\} [\text{CLUSTERING ON CONSTRAINT} [\textit{ConstraintID}]]$

AddConstraintSpecification

$$\text{ADD CONSTRAINT } \left\{ \begin{array}{l} \textit{UniqueConstraint} \\ \textit{ReferentialConstraint} \\ \textit{CheckConstraint} \end{array} \right\} [, \dots])$$

[CLUSTERING ON CONSTRAINT [*ConstraintID1*]]

DropConstraintSpecification

$$\text{DROP CONSTRAINT } \left\{ \begin{array}{l} (\textit{ConstraintID} [, \dots]) \\ \textit{ConstraintID} \end{array} \right\}$$

SetTypeSpecification

$$\text{SET TYPE } \left\{ \begin{array}{l} \text{PRIVATE} \\ \text{PUBLICREAD} \\ \text{PUBLIC} \\ \text{PUBLICROW} \end{array} \right\} \left[\begin{array}{l} \text{RESET AUTHORITY} \\ \text{PRESERVE AUTHORITY} \end{array} \right]$$

SetPartitionSpecification

$$\text{SET PARTITION } \left\{ \begin{array}{l} \textit{PartitionName} \\ \text{DEFAULT} \\ \text{NONE} \end{array} \right\}$$

Assignment (=)

$$\left\{ \begin{array}{l} : \textit{LocalVariable} \\ : \textit{ProcedureParameter} \end{array} \right\} = \textit{Expression};$$

BEGIN

BEGIN [*statement*;] [...] END;

BEGIN ARCHIVE

BEGIN ARCHIVE

BEGIN DECLARE SECTION

BEGIN DECLARE SECTION

BEGIN WORK

$$\text{BEGIN WORK } [\textit{Priority}] \left[\begin{array}{l} \text{RR} \\ \text{CS} \\ \text{RC} \\ \text{RU} \end{array} \right] \left[\text{LABEL } \left\{ \begin{array}{l} \text{'LabelString'} \\ : \textit{HostVariable} \end{array} \right\} \right] \left[\left[\begin{array}{l} \text{PARALLEL} \\ \text{NO} \end{array} \right] \text{FILL} \right]$$

CHECKPOINT

CHECKPOINT [*:HostVariable*
 :LocalVariable
 :ProcedureParameter]

CLOSE

CLOSE *CursorName* [USING { [SQL] DESCRIPTOR { SQLDA
 AreaName }
 :HostVariable [[INDICATOR] *:Indicator*] [, ...] }]

COMMIT ARCHIVE

COMMIT ARCHIVE

COMMIT WORK

COMMIT WORK [RELEASE]

CONNECT

CONNECT TO { *'DBEnvironmentName'*
 :HostVariable1 } [AS { *'ConnectionName'*
 :HostVariable2 }]
[USER { *'UserID'*
 :HostVariable3 } [USING *:HostVariable4*]]

CREATE DBEFIELD

CREATE DBEFIELD *DBEFieldName* WITH PAGES = *DBEFileSize*, NAME = *'SystemFileName'*
[, INCREMENT = *DBEFileIncrSize* [, MAXPAGES = *DBEFileMaxSize*]]
[, TYPE = { TABLE
 INDEX
 MIXED }] [, DEVICE = *volume*;]

CREATE DBEFILESET

CREATE DBEFILESET *DBEFileSetName*

CREATE GROUP

CREATE GROUP [*Owner.*] *GroupName*

CREATE INDEX

```
CREATE [ UNIQUE ] [ CLUSTERING ] INDEX [ Owner. ] IndexName ON
[ Owner. ] TableName ( { ColumnName [ ASC
DESC ] } [ , ... ] )
```

CREATE PARTITION

```
CREATE PARTITION PartitionName WITH ID = PartitionNumber
```

CREATE PROCEDURE

```
CREATE PROCEDURE [ Owner. ] ProcedureName [ LANG = ProcLangName ]
[ ( ParameterDeclaration [ , ParameterDeclaration ] [ ... ] ) ]
[ WITH RESULT ResultDeclaration [ , ResultDeclaration ] [ ... ] ]
AS BEGIN [ ProcedureStatement ] [ ... ] END [ IN DBEFileSetName ]
```

ParameterDeclaration

```
ParameterName ParameterType [ LANG = ParameterLanguage ]
[ DEFAULT DefaultValue ] [ NOT NULL ] [ OUTPUT [ ONLY ] ]
```

ResultDeclaration

```
ResultType [ LANG = ResultLanguage ] [ NOT NULL ]
```

CREATE RULE

```
CREATE RULE [ Owner. ] RuleName
AFTER StatementType [ , ... ] { ON
OF
FROM
INTO } [ Owner ] . TableName
[ REFERENCING { OLD AS OldCorrelationName } [ ... ] ] [ WHERE FiringCondition ]
EXECUTE PROCEDURE [ OwnerName. ] ProcedureName [ ( ParameterValue [ , ... ] ) ]
[ IN DBEFileSetName ]
```

CREATE SCHEMA

```
CREATE SCHEMA AUTHORIZATION AuthorizationName
[ TableDefinition
ViewDefinition
IndexDefinition
ProcedureDefinition
RuleDefinition
CreateGroup
AddToGroup
GrantStatement ] [ ... ]
```

CREATE TABLE

```
CREATE [ PRIVATE  
      PUBLICREAD  
      PUBLIC  
      PUBLICROW ] TABLE [ Owner. ] TableName  
[ LANG = TableLanguageName ]  
( { ColumnDefinition  
  UniqueConstraint  
  ReferentialConstraint  
  CheckConstraint } [ , ... ] )  
[ UNIQUE HASH ON ( HashColumnName [ , ... ] ) PAGES = PrimaryPages ]  
[ HASH ON CONSTRAINT [ ConstraintID ] PAGES = PrimaryPages ]  
[ CLUSTERING ON CONSTRAINT [ ConstraintID ] ]  
[ IN PARTITION { PartitionName  
               DEFAULT  
               NONE } ]  
[ IN DBEFileSetName1 ]
```

Column Definition

```
ColumnName { ColumnDataType  
            LongColumnType [ IN DBEFileSetName2 ] }  
[ LANG = ColumnLanguageName ]  
[ [ NOT ] CASE SENSITIVE ]  
[ DEFAULT { Constant  
           USER  
           NULL  
           CurrentFunction } ]  
[ NOT NULL [ { UNIQUE  
             PRIMARY KEY } [ CONSTRAINT ConstraintID ] ] ]  
[ REFERENCES RefTableName [ ( RefColumnName ) ] [ CONSTRAINT ConstraintID ] ]  
[ ... ] [ ... ]  
[ CHECK ( SearchCondition ) [ CONSTRAINT ConstraintID ] ]  
[ IN DBEFileSetName3 ]
```

Unique Constraint (Table Level)

```
{ UNIQUE  
  PRIMARY KEY } ( ColumnName [ , ... ] ) [ CONSTRAINT ConstraintID ]
```

Referential Constraint (Table Level)

```
FOREIGN KEY ( FKColumnName [ , ... ] )  
REFERENCES RefTableName [ ( RefColumnName [ , ... ] ) ] [ CONSTRAINT ConstraintID ]
```

Check Constraint (Table Level)

```
CHECK (SearchCondition) [CONSTRAINT ConstraintID] [IN DBEFileSetName3]
```

CREATE TEMPSPACE

```
CREATE TEMPSPACE TempSpaceName  
WITH [MAXFILEPAGES = MaxTempFileSize,] LOCATION = 'PhysicalLocation'
```

CREATE VIEW

```
CREATE VIEW [Owner.] ViewName [(ColumnName [, ... ])]  
AS QueryExpression [IN DBEFileSetName]  
[WITH CHECK OPTION [CONSTRAINT ConstraintID]]
```

DECLARE CURSOR

```
DECLARE CursorName [IN DBEFileSetName] CURSOR FOR  
{ { QueryExpression } [FOR UPDATE OF { ColumnName } [, ... ] ] }  
{ { SelectStatementName } [FOR READ ONLY ] }  
{ ExecuteProcedureStatement }  
{ ExecuteStatementName }
```

DECLARE Variable

```
DECLARE { LocalVariable } [, ... ] VariableType [LANG = VariableLangName]  
[ DEFAULT { Constant } ] [ NOT NULL ]  
[ { CurrentFunction } ]
```

DELETE

```
DELETE [WITH AUTOCOMMIT] FROM { [Owner.] TableName } [WHERE SearchCondition]  
[ [Owner.] ViewName ]
```

DELETE WHERE CURRENT

```
DELETE FROM { [Owner.] TableName } WHERE CURRENT OF CursorName  
[ [Owner.] ViewName ]
```

DESCRIBE

```
DESCRIBE [ OUTPUT ]  
[ INPUT ] StatementName { INTO [ [SQL] DESCRIPTOR ] } { SQLDA }  
[ RESULT ] [ USING [SQL] DESCRIPTOR ] { AreaName }
```

DISABLE AUDIT LOGGING

DISABLE AUDIT LOGGING

DISABLE RULES

DISABLE RULES

DISCONNECT

DISCONNECT { ' *ConnectionName*'
' *DBEnvironmentName*'
: *HostVariable*
ALL
CURRENT }

DROP DBEFILE

DROP DBEFILE *DBEFileName*

DROP DBEFILESET

DROP DBEFILESET *DBEFileSetName*

DROP GROUP

DROP GROUP *GroupName*

DROP INDEX

DROP INDEX [*Owner.*] *IndexName* [FROM [*Owner.*] *TableName*]

DROP MODULE

DROP MODULE [*Owner.*] *ModuleName* [PRESERVE]

DROP PARTITION

DROP PARTITION *PartitionName*

DROP PROCEDURE

DROP PROCEDURE [*Owner.*] *ProcedureName* [PRESERVE]

DROP RULE

DROP RULE [*Owner.*] *RuleName* [FROM TABLE [*Owner.*] *TableName*]

DROP TABLE

DROP TABLE [*Owner.*] *TableName*

DROP TEMPSPACE

DROP TEMPSPACE *TempSpaceName*

DROP VIEW

DROP VIEW [*Owner.*] *ViewName*

ENABLE AUDIT LOGGING

ENABLE AUDIT LOGGING

ENABLE RULES

ENABLE RULES

END DECLARE SECTION

END DECLARE SECTION

EXECUTE

EXECUTE { *StatementName*
[*Owner.*] *ModuleName* [(*SectionNumber*)] }

USING { [SQL] DESCRIPTOR { [INPUT] { SQLDA
 AreaName1 }
 [AND OUTPUT { SQLDA
 AreaName2 }] }
 OUTPUT { SQLDA
 AreaName } } }
[INPUT] *HostVariableSpecification1*
[AND OUTPUT *HostVariableSpecification2*]
OUTPUT *HostVariableSpecification*
: *Buffer* [, : *StartIndex* [, : *NumberOfRows*]] }

HostVariableSpecification

: *HostVariableName* [[INDICATOR] : *IndicatorVariable*] [, ...]

EXECUTE IMMEDIATE

EXECUTE IMMEDIATE { 'String'
:HostVariable }

EXECUTE PROCEDURE

EXECUTE PROCEDURE [:ReturnStatusVariable =] [Owner.] ProcedureName
[([ActualParameter] [, [ActualParameter]] [...])]

ActualParameter

[ParameterName =] ParameterValue [OUTPUT [ONLY]]

FETCH

[BULK] FETCH CursorName { INTO HostVariableSpecification
USING { [SQL] DESCRIPTOR { SQLDA
AreaName } }
HostVariableSpecification } }

BULK HostVariableSpecification

:Buffer [, :StartIndex [, :NumberOfRows]]

Non-BULK HostVariableSpecification

{ :HostVariable [[INDICATOR] :Indicator] } [, ...]

GENPLAN

GENPLAN [WITH (HostVariableDefinition)] FOR
{ SQLStatement
MODULE SECTION [Owner.] ModuleName(SectionName)
PROCEDURE SECTION [Owner.] ProcedureName(SectionName) }

GOTO

{ GOTO } { Label }
{ GO TO } { Integer }

GRANT

$$\text{GRANT } \left\{ \begin{array}{l} \text{ALL [PRIVILEGES]} \\ \left\{ \begin{array}{l} \text{SELECT} \\ \text{INSERT} \\ \text{DELETE} \\ \text{ALTER} \\ \text{INDEX} \\ \text{UPDATE [(\{ ColumnName \} [, \dots])]} \\ \text{REFERENCES [(\{ ColumnName \} [, \dots])]} \end{array} \right\} |, \dots | \end{array} \right\}$$
$$\text{ON } \left\{ \begin{array}{l} [Owner.] TableName \\ [Owner.] ViewName \end{array} \right\} \text{ TO } \left\{ \begin{array}{l} DBEUserID \\ GroupName \\ ClassName \\ \text{PUBLIC} \end{array} \right\} [, \dots] \text{ [WITH GRANT OPTION]}$$
$$\left[\text{BY } \left\{ \begin{array}{l} DBEUserID \\ ClassName \end{array} \right\} \right]$$

Grant RUN or EXECUTE Authority

$$\text{GRANT } \left\{ \begin{array}{l} \text{RUN ON [Owner.] ModuleName} \\ \text{EXECUTE ON PROCEDURE [Owner.] ProcedureName} \end{array} \right\} \text{ TO}$$
$$\left\{ \begin{array}{l} \left\{ \begin{array}{l} DBEUserID \\ GroupName \\ ClassName \end{array} \right\} [, \dots] \\ \text{PUBLIC} \end{array} \right\}$$

Grant CONNECT, DBA, INSTALL, MONITOR, or RESOURCE Authority

$$\text{GRANT } \left\{ \begin{array}{l} \text{CONNECT} \\ \text{DBA} \\ \text{INSTALL [AS OwnerID]} \\ \text{MONITOR} \\ \text{RESOURCE} \end{array} \right\} \text{ TO } \left\{ \begin{array}{l} DBEUserID \\ GroupName \\ ClassName \end{array} \right\} [, \dots]$$

Grant DBEFileSet Authority

$$\text{GRANT } \left\{ \begin{array}{l} \text{SECTIONSPACE} \\ \text{TABLESPACE} \end{array} \right\} [, \dots] \text{ ON DBEFILESET } DBEFileSetName \text{ TO}$$
$$\left\{ \begin{array}{l} DBEUserID \\ GroupName \\ ClassName \\ \text{PUBLIC} \end{array} \right\} [, \dots]$$

IF

```
IF Condition THEN [ Statement; [ ... ] ]
[ ELSEIF Condition THEN [ Statement; [ ... ] ] ]
[ ELSE [ Statement; [ ... ] ] ] ENDIF;
```

INCLUDE

INCLUDE { SQLCA [[IS] EXTERNAL] }
 { SQLDA }

INSERT - 1

[BULK] INSERT INTO { [*Owner.*] *TableName* }
 { [*Owner.*] *ViewName* }
[({ *ColumnName* } [, ...])]
VALUES ({ *SingleRowValues* }
 { *BulkValues* }
 { ? })

SingleRowValues

{ NULL
 USER
 : *HostVariable* [[INDICATOR] : *IndicatorVariable*]
 ?
 : *LocalVariable*
 : *ProcedureParameter*
 : : *Built-inVariable*
 ConversionFunction
 CurrentFunction
 [+] { *Integer* }
 [-] { *Float* }
 { *Decimal* }
 ' *CharacterString* '
 0x *HexadecimalString*
 ' *LongColumnIOString* ' }

[, ...]

LongColumnIOString

< { *FileName* [. *Group* [. *Account*]] }
 { %*HeapAddress* : *LengthofHeap* }
 { { > } { *FileName* [. *Group* [. *Account*]] } }
 { { >> } { *CharString*\$ } }
 { { >! } { *CharString*\$ *CharString* } }
 { >%\$ }

BulkValues

: *Buffer* [, : *StartIndex* [, : *NumberOfRows*]]

Dynamic Parameter Substitution

(? [, ...])

INSERT - 2

INSERT INTO $\left\{ \begin{array}{l} [Owner.] Table\ Name \\ [Owner.] View\ Name \end{array} \right\} [(Column\ Name [, \dots])] Query\ Expression$

Labeled Statement

Label: Statement

LOCK TABLE

LOCK TABLE $[Owner.] Table\ Name$ IN $\left\{ \begin{array}{l} SHARE [UPDATE] \\ EXCLUSIVE \end{array} \right\} MODE$

LOG COMMENT

LOG COMMENT $\left\{ \begin{array}{l} ' String ' \\ : Host\ Variable \\ : Procedure\ Parameter \\ : Procedure\ Local\ Variable \\ ? \end{array} \right\}$

OPEN

OPEN *CursorName* $\left[\begin{array}{l} KEEP\ CURSOR \left[\begin{array}{l} WITH\ LOCKS \\ WITH\ NOLOCKS \end{array} \right] \end{array} \right]$
 $\left[\begin{array}{l} USING \left\{ \begin{array}{l} [SQL] DESCRIPTOR \left\{ \begin{array}{l} SQLDA \\ Area\ Name \end{array} \right\} \\ Host\ Variable\ Name [[INDICATOR] : Indicator\ Variable] [, \dots] \end{array} \right\} \end{array} \right]$

PREPARE

PREPARE $[REPEAT] \left\{ \begin{array}{l} Statement\ Name \\ [Owner.] Module\ Name [(Section\ Number)] \end{array} \right\}$
 $[IN\ DBE\ File\ Set\ Name] FROM \left\{ \begin{array}{l} ' String ' \\ : Host\ Variable \end{array} \right\}$

PRINT

PRINT $\left\{ \begin{array}{l} ' Constant ' \\ : Local\ Variable \\ : Parameter \\ : Built-in\ Variable \end{array} \right\};$

RAISE ERROR

RAISE ERROR [*ErrorNumber*] [MESSAGE *ErrorText*]

REFETCH

REFETCH *CursorName* INTO { :*HostVariable* [[INDICATOR]:*Indicator*] } [, ...]

RELEASE

RELEASE

REMOVE DBEFILE

REMOVE DBEFILE *DBEFileName* FROM DBEFILESET *DBEFileSetName*

REMOVE FROM GROUP

REMOVE { *DBEUserID*
GroupName
ClassName } [, ...] FROM GROUP [*Owner.*] *TargetGroupName*

RENAME COLUMN

RENAME COLUMN [*Owner.*] *TableName.ColumnName* TO *NewColumnName*

RENAME TABLE

RENAME TABLE [*Owner.*] *TableName* TO *NewTableName*

RESET

RESET { SYSTEM.ACCOUNT [FOR USER { *
DBEUserID }]]
SYSTEM.COUNTER }

RETURN

RETURN [*ReturnStatus*] ;

REVOKE

Revoke Table or View Authority

$$\text{REVOKE } \left\{ \begin{array}{l} \text{ALL [PRIVILEGES]} \\ \left[\begin{array}{l} \text{SELECT} \\ \text{INSERT} \\ \text{DELETE} \\ \text{ALTER} \\ \text{INDEX} \\ \text{UPDATE [(\{ ColumnName \} [, \dots])} \\ \text{REFERENCES [(\{ ColumnName \} [, \dots])} \end{array} \right] \end{array} \right\} |, \dots |$$
$$\text{ON } \left\{ \begin{array}{l} [\text{Owner.}] \text{TableName} \\ [\text{Owner.}] \text{ViewName} \end{array} \right\} \text{FROM } \left\{ \begin{array}{l} \text{DBEUserID} \\ \text{GroupName} \\ \text{ClassName} \\ \text{PUBLIC} \end{array} \right\} [, \dots] [\text{CASCADE}]$$

Revoke RUN or EXECUTE or Authority

$$\text{REVOKE } \left[\begin{array}{l} \text{RUN ON [Owner.] ModuleName} \\ \text{EXECUTE ON PROCEDURE [Owner.] ProcedureName} \end{array} \right] \text{FROM}$$
$$\left\{ \begin{array}{l} \left\{ \begin{array}{l} \text{DBEUserID} \\ \text{GroupName} \\ \text{ClassName} \end{array} \right\} [, \dots] \\ \text{PUBLIC} \end{array} \right\}$$

Revoke CONNECT, DBA, INSTALL, MONITOR, or RESOURCE Authority

$$\text{REVOKE } \left\{ \begin{array}{l} \text{CONNECT} \\ \text{DBA} \\ \text{INSTALL [AS OwnerID]} \\ \text{MONITOR} \\ \text{RESOURCE} \end{array} \right\} \text{FROM } \left\{ \begin{array}{l} \text{DBEUserID} \\ \text{GroupName} \\ \text{ClassName} \end{array} \right\} [, \dots]$$

SQL Syntax—Revoke DBEFileSet Authority

$$\text{REVOKE } \left\{ \begin{array}{l} \text{SECTIONSPACE} \\ \text{TABLESPACE} \end{array} \right\} |, \dots | \text{ ON DBEFILESET } \text{DBEFileSetName} \text{ FROM}$$
$$\left\{ \begin{array}{l} \left\{ \begin{array}{l} \text{DBEUserID} \\ \text{GroupName} \\ \text{ClassName} \end{array} \right\} [, \dots] \\ \text{PUBLIC} \end{array} \right\}$$

ROLLBACK WORK

$$\text{ROLLBACK WORK} \left[\begin{array}{l} \text{TO} \left\{ \begin{array}{l} \textit{SavePointNumber} \\ \textit{HostVariable} \\ \textit{LocalVariable} \\ \textit{ProcedureParameter} \end{array} \right\} \\ \text{RELEASE} \end{array} \right]$$

SAVEPOINT

$$\text{SAVEPOINT} \left[\begin{array}{l} \textit{HostVariable} \\ \textit{LocalVariable} \\ \textit{ProcedureParameter} \end{array} \right]$$

SELECT

Select Statement Level

$$[\text{BULK}] \textit{QueryExpression} \left[\text{ORDER BY} \left\{ \textit{ColumnID} \left[\begin{array}{l} \text{ASC} \\ \text{DESC} \end{array} \right] \right\} [, \dots] \right]$$

Subquery Level

(QueryExpression)

Query Expression Level

$$\left\{ \begin{array}{l} \textit{QueryBlock} \\ \textit{QueryExpression} \end{array} \right\} \left[\text{UNION} [\text{ALL}] \left\{ \begin{array}{l} \textit{QueryBlock} \\ \textit{QueryExpression} \end{array} \right\} \right] [\dots]$$

Query Block Level

$$\begin{array}{l} \text{SELECT} \left[\begin{array}{l} \text{ALL} \\ \text{DISTINCT} \end{array} \right] \textit{SelectList} \left[\text{INTO} \textit{HostVariableSpecification} \right] \\ \text{FROM} \textit{FromSpec} [, \dots] \\ \left[\text{WHERE} \textit{SearchCondition1} \right] \\ \left[\text{GROUP BY} \textit{GroupColumnList} \right] \\ \left[\text{HAVING} \textit{SearchCondition2} \right] \end{array}$$

SelectList

$$\left\{ \begin{array}{l} * \\ [\textit{Owner} .] \textit{Table} . * \\ \textit{CorrelationName} . * \\ \textit{Expression} \\ [[\textit{Owner} .] \textit{Table} .] \textit{ColumnName} \\ \textit{CorrelationName} . \textit{ColumnName} \end{array} \right\} [, \dots]$$

HostVariableSpecification—With BULK Option

:Buffer [, *:StartIndex* [, *:NumberOfRows*]]

HostVariableSpecification—Without BULK Option

{ *:HostVariable* [[INDICATOR] : *Indicator*] } [, ...]

FromSpec

$$\left\{ \begin{array}{l} \textit{TableSpec} \\ (\textit{FromSpec}) \\ \textit{FromSpec} \text{ NATURAL } \left[\begin{array}{l} \text{INNER} \\ \text{LEFT } [\text{OUTER}] \\ \text{RIGHT } [\text{OUTER}] \end{array} \right] \text{ JOIN } \left\{ \begin{array}{l} \textit{TableSpec} \\ (\textit{FromSpec}) \end{array} \right\} \\ \textit{FromSpec} \left[\begin{array}{l} \text{INNER} \\ \text{LEFT } [\text{OUTER}] \\ \text{RIGHT } [\text{OUTER}] \end{array} \right] \text{ JOIN } \left\{ \begin{array}{l} \textit{TableSpec} \\ (\textit{FromSpec}) \end{array} \right\} \left\{ \begin{array}{l} \text{ON } \textit{SearchCondition3} \\ \text{USING } (\textit{ColumnList}) \end{array} \right\} \end{array} \right\}$$

TableSpec

[*Owner.*] *TableName* [*CorrelationName*]

SET CONNECTION

SET CONNECTION { ' *ConnectionName* ' }
 { *:HostVariable* }

SET CONSTRAINTS

SET *ConstraintType* [, ...] CONSTRAINTS { DEFERRED }
 { IMMEDIATE }

SET DEFAULT DBFILESET

SET DEFAULT { SECTIONSPACE } TO DBFILESET *DBFileSetName* FOR PUBLIC
 { TABLESPACE }

SET DML ATOMICITY

SET DML ATOMICITY AT { ROW } LEVEL
 { STATEMENT }

SET MULTITRANSACTION

SET MULTITRANSACTION { ON }
 { OFF }

SETOPT

$$\text{SETOPT} \left\{ \begin{array}{l} \text{CLEAR} \\ \text{GENERAL} \left\{ \begin{array}{l} \textit{ScanAccess} \\ \textit{JoinAlgorithm} \end{array} \right\} [, \dots] \\ \text{BEGIN} \left\{ \text{GENERAL} \left\{ \begin{array}{l} \textit{ScanAccess} \\ \textit{JoinAlgorithm} \end{array} \right\} \right\} [; \dots] \text{END} \end{array} \right\}$$

Scan Access

$$[\text{NO}] \left\{ \begin{array}{l} \text{SERIALSCAN} \\ \text{INDEXSCAN} \\ \text{HASHSCAN} \\ \text{SORTINDEX} \end{array} \right\}$$

Join Algorithm

$$[\text{NO}] \left\{ \begin{array}{l} \text{NESTEDLOOP} \\ \text{NLJ} \\ \text{SORTMERGE} \\ \text{SMJ} \end{array} \right\}$$

SET PRINTRULES

$$\text{SET PRINTRULES} \left[\begin{array}{l} \text{ON} \\ \text{OFF} \end{array} \right]$$

SET SESSION

```

SET SESSION {
    ISOLATION LEVEL {
        RR
        CS
        RC
        RU
        REPEATABLE READ
        SERIALIZABLE
        CURSOR STABILITY
        READ COMMITTED
        READ UNCOMMITTED
        :HostVariable1
    }
    PRIORITY { Priority
               :HostVariable2
    }
    LABEL { 'LabelString'
            :HostVariable3
    }
    ConstraintType [ , ... ] CONSTRAINTS { DEFERRED
                                           IMMEDIATE
    }
    DML ATOMICITY AT { STATEMENT
                       ROW
                     } LEVEL
    ON { TIMEOUT
         DEADLOCK
    } ROLLBACK { QUERY
                TRANSACTION
    }
    USER TIMEOUT [ TO ] {
        DEFAULT
        MAXIMUM
        TimeoutValue [ { SECONDS
                       MINUTES
                     } ]
        :HostVariable4 [ { SECONDS
                        MINUTES
                      } ]
    }
    TERMINATION AT { SESSION
                    TRANSACTION
                    QUERY
                    RESTRICTED
                  } LEVEL
    [ { PARALLEL
        NO
      } ] FILL
} [ , ... ]

```

SET TRANSACTION

SET TRANSACTION {

 ISOLATION LEVEL {

 RR

 CS

 RC

 RU

 REPEATABLE READ

 SERIALIZABLE

 CURSOR STABILITY

 READ COMMITTED

 READ UNCOMMITTED

 :HostVariable1

 }

 PRIORITY { Priority

 :HostVariable2

 }

 LABEL { 'LabelString'

 :HostVariable3

 }

 ConstraintType [, ...] CONSTRAINTS { DEFERRED

 IMMEDIATE

 }

 DML ATOMICITY AT { STATEMENT

 ROW

 } LEVEL

 ON { TIMEOUT

 DEADLOCK

 } ROLLBACK { QUERY

 TRANSACTION

 }

 USER TIMEOUT [TO] {

 DEFAULT

 MAXIMUM

 TimeoutValue [{ SECONDS

 MINUTES

 }]

 :HostVariable4 [{ SECONDS

 MINUTES

 }]

 }

 TERMINATION AT { SESSION

 TRANSACTION

 QUERY

 RESTRICTED

 } LEVEL

 } [, ...]

SET USER TIMEOUT

SET USER TIMEOUT [TO] {

 { TimeoutValue

 :HostVariable

 } [SECONDS

 MINUTES

 }

 DEFAULT

 MAXIMUM

SQL EXPLAIN

SQL EXPLAIN :HostVariable

START DBE

```
START DBE 'DBEnvironmentName' [AS ConnectionName'] [MULTI]
[
  BUFFER = (DataBufferPages, LogBufferPages)
  TRANSACTION = MaxTransactions
  MAXIMUM TIMEOUT = { Timeout Value [SECONDS]
                     NONE [MINUTES] }
  DEFAULT TIMEOUT = { Timeout Value [SECONDS]
                     MAXIMUM [MINUTES] }
  RUN BLOCK = ControlBlockPages
] |, ... |
```

START DBE NEW

```
START DBE 'DBEnvironmentName' [AS 'ConnectionName'] [MULTI] NEW
[
  { DUAL } | ... | LOG
  { AUDIT }
  BUFFER = (DataBufferPages, LogBufferPages)
  LANG = LanguageName
  TRANSACTION = MaxTransactions
  MAXIMUM TIMEOUT = { Timeout Value [SECONDS]
                     NONE [MINUTES] }
  DEFAULT TIMEOUT = { Timeout Value [SECONDS]
                     MAXIMUM [MINUTES] }
  RUN BLOCK = ControlBlockPages
  DEFAULT PARTITION = { DefaultPartitionNumber }
                     NONE
  COMMENT PARTITION = { CommentPartitionNumber }
                     DEFAULT
                     NONE
  MAXPARTITIONS = MaximumNumberOfPartitions
  AUDIT NAME = 'AuditName'
  { COMMENT
    DATA
    DEFINITION
    STORAGE } | ... | AUDIT ELEMENTS
  { AUTHORIZATION
    SECTION
    ALL }
  DBEFileDefinition
  DBELogDefinition
] |, ... |
```

DBEFile0Definition

```
DBEFILE0 DBEFILE DBEFile0ID
WITH PAGES = DBEFile0Size,
NAME = 'SystemFileName1'
```

DBELogDefinition

```
LOG DBEFILE DBELog1ID [AND DBELog2ID]
WITH PAGES = DBELogSize,
NAME = 'SystemFileName2' [AND 'SystemFileName3' ]
```

START DBE NEWLOG

```
START DBE 'DBEnvironmentName' [AS 'ConnectionName' ] [MULTI] NEWLOG
{
  { ARCHIVE
    DUAL      } | ... | LOG
  { AUDIT
    NONE      }
  BUFFER = (DataBufferPages, LogBufferPages)
  TRANSACTION = MaxTransactions
  MAXIMUM TIMEOUT = { TimeoutValue [SECONDS]
                     NONE           [MINUTES] }
  DEFAULT TIMEOUT = { TimeoutValue [SECONDS]
                     MAXIMUM       [MINUTES] }
  RUN BLOCK = ControlBlockPages
  DEFAULT PARTITION = { DefaultPartitionNumber
                      NONE
  } | ... | NewLogDefinition
  COMMENT PARTITION = { CommentPartitionNumber
                      DEFAULT
                      NONE
  }
  MAXPARTITIONS = MaximumNumberOfPartitions
  AUDIT NAME = 'AuditName'
  {
    COMMENT
    DATA
    DEFINITION
    STORAGE
    AUTHORIZATION
    SECTION
    ALL
  } | ... | AUDIT ELEMENTS
}
```

NewLogDefinition

```
LOG DBEFILE DBELog1ID [AND DBELog2ID]
WITH PAGES = DBELogSize,
NAME = 'SystemFileName1' [AND 'SystemFileName2' ]
```

STOP DBE

STOP DBE

TERMINATE QUERY

TERMINATE QUERY FOR $\left\{ \begin{array}{l} \text{CID } \textit{ConnectionID} \\ \text{XID } \textit{TransactionID} \end{array} \right\}$

TERMINATE TRANSACTION

TERMINATE TRANSACTION FOR $\left\{ \begin{array}{l} \text{CID } \textit{ConnectionID} \\ \text{XID } \textit{TransactionID} \end{array} \right\}$

TERMINATE USER

TERMINATE USER $\left\{ \begin{array}{l} \textit{DBEUserID} \\ \textit{SessionID} \\ \text{CID } \textit{ConnectionID} \end{array} \right\}$

TRANSFER OWNERSHIP

TRANSFER OWNERSHIP OF $\left\{ \begin{array}{l} [\text{TABLE}] [\textit{Owner.}] \textit{TableName} \\ [\text{VIEW}] [\textit{Owner.}] \textit{ViewName} \\ \text{PROCEDURE } [\textit{Owner.}] \textit{ProcedureName} \\ \text{GROUP } \textit{GroupName} \end{array} \right\}$ TO $\textit{NewOwnerName}$

TRUNCATE TABLE

TRUNCATE TABLE $[\textit{Owner.}] \textit{TableName}$

UPDATE

UPDATE $\left\{ \begin{array}{l} [\textit{Owner.}] \textit{TableName} \\ [\textit{Owner.}] \textit{ViewName} \end{array} \right\}$
SET $\left\{ \textit{ColumnName} = \left\{ \begin{array}{l} \textit{Expression} \\ \text{' LongColumnIOString' } \\ \text{NULL} \end{array} \right\} \right\} [, \dots]$
[WHERE $\textit{SearchCondition}$]

LongColumnIOString

$$\left\{ \left[\left\langle \left\{ \begin{array}{l} \text{FileName} [. \text{Group} [. \text{Account}]] \\ \% \text{HeapAddress} : \text{LengthofHeap} \end{array} \right\} \right] \right] \right\} \left| \dots \right| \left\{ \left[\left\langle \left\{ \begin{array}{l} \text{FileName} [. \text{Group} [. \text{Account}]] \\ \text{CharString} \$ \\ \text{CharString} \$ \text{CharString} \end{array} \right\} \right] \right] \right\} \left[\begin{array}{l} > \\ > ! \\ > \% \$ \end{array} \right]$$

UPDATE STATISTICS

UPDATE STATISTICS FOR TABLE $\left\{ \begin{array}{l} [\text{Owner} .] \text{TableName} \\ \text{SYSTEM} . \text{SystemViewName} \end{array} \right\}$

UPDATE WHERE CURRENT

UPDATE $\left\{ \begin{array}{l} [\text{Owner} .] \text{TableName} \\ [\text{Owner} .] \text{ViewName} \end{array} \right\}$
SET $\left\{ \begin{array}{l} \text{ColumnName} = \left\{ \begin{array}{l} \text{Expression} \\ ' \text{LongColumnIOString}' \\ \text{NULL} \end{array} \right\} \end{array} \right\} [, \dots]$
WHERE CURRENT OF *CursorName*

LongColumnIOString

$$\left\{ \left[\left\langle \left\{ \begin{array}{l} \text{FileName} [. \text{Group} [. \text{Account}]] \\ \% \text{HeapAddress} : \text{LengthofHeap} \end{array} \right\} \right] \right] \right\} \left| \dots \right| \left\{ \left[\left\langle \left\{ \begin{array}{l} \text{FileName} [. \text{Group} [. \text{Account}]] \\ \text{CharString} \$ \\ \text{CharString} \$ \text{CharString} \end{array} \right\} \right] \right] \right\} \left[\begin{array}{l} > \\ > ! \\ > \% \$ \end{array} \right]$$

VALIDATE

VALIDATE $\left[\begin{array}{l} \text{FORCE} \\ \text{DROP SETOPTINFO} \end{array} \right]$
 $\left\{ \begin{array}{l} \text{MODULE} \left\{ \left\{ [\text{Owner} .] \text{ModuleName} \right\} [, \dots] \right. \\ \left. \left\{ \text{SECTION} [\text{Owner} .] \text{ModuleName} (\text{Section Number}) \right\} [, \dots] \right\} \\ \text{PROCEDURE} \left\{ \left\{ [\text{Owner} .] \text{ProcedureName} \right\} [, \dots] \right. \\ \left. \left\{ \text{SECTION} [\text{Owner} .] \text{ProcedureName} (\text{Section Number}) \right\} [, \dots] \right\} \\ \text{ALL} \left\{ \begin{array}{l} \text{MODULES} \\ \text{PROCEDURES} \end{array} \right\} [\text{WITH AUTOCOMMIT}] \end{array} \right\}$

WHENEVER

$$\text{WHENEVER} \left\{ \begin{array}{l} \text{SQLERROR} \\ \text{SQLWARNING} \\ \text{NOT FOUND} \end{array} \right\} \left\{ \begin{array}{l} \text{STOP} \\ \text{CONTINUE} \\ \text{GOTO } [:] \textit{Label} \\ \text{GO TO } [:] \textit{Label} \end{array} \right\}$$

WHILE

WHILE *Condition* DO [*Statement*; [...]]ENDWHILE;

ISQL Syntax Summary

ISQL is the interactive interface to ALLBASE/SQL. Some, but not all, ALLBASE/SQL statements can be entered interactively as ISQL commands.

CHANGE

C[HANGE] *Delimiter OldString Delimiter NewString Delimiter* [©]

DO

DO $\left[\begin{array}{l} \textit{CommandNumber} \\ \textit{CommandString} \end{array} \right]$

EDIT

ED[IT]

END

EN[D]

ERASE

ER[ASE] *FileName*

EXIT

EX[IT]

EXTRACT

EXTRACT $\left\{ \begin{array}{l} \text{MODULE [} \textit{Owner.} \text{] } \textit{ModuleName} \text{ [, ...]} \\ \text{SECTION [} \textit{Owner.} \text{] } \textit{ModuleName} \text{(} \textit{SectionNumber} \text{) [, ...]} \\ \text{ALL MODULES} \end{array} \right\}$
 [NO SETOPTINFO] INTO *FileName*

HELP

$$\text{HE[LP]} \left\{ \begin{array}{l} \text{\textcircled{C}} \\ \text{SQLStatement} \\ \text{ISQLCommand} \end{array} \right\} \left[\begin{array}{l} \text{D[ESCRIPTION]} \\ \text{S[YNTAX]} \\ \text{E[AMPLE]} \end{array} \right]$$

HOLD

$$\text{HO[LD]} \left\{ \begin{array}{l} \text{SQLStatement} \\ \text{ISQLCommand} \end{array} \right\} \left[\text{EscapeCharacter}; \left\{ \begin{array}{l} \text{SQLStatement} \\ \text{ISQLCommand} \end{array} \right\} \right] [\dots]$$

INFO

$$\text{IN[FO]} \left\{ \begin{array}{l} [\text{Owner.}] \text{TableName} \\ [\text{Owner.}] \text{ViewName} \end{array} \right\}$$

INPUT

$$\text{INP[UT]} \left\{ \begin{array}{l} [\text{Owner.}] \text{TableName} \\ [\text{Owner.}] \text{ViewName} \end{array} \right\} (\text{ColumnName} [, \text{ColumnName}] [\dots]) \\ \left\{ (\text{Value} [, \text{Value}] [\dots]) \left[\begin{array}{l} \text{ROLLBACK WORK} \\ \text{COMMIT WORK} \end{array} \right] \right\} [\dots] \text{E[ND]}$$

INSTALL

$$\text{IN[STALL]} \text{FileName} [\text{DROP}] [\text{IN DBEFileSetName}] [\text{NO OPTINFO}]$$

LIST FILE

$$\text{LI[ST]F[ILE]} \text{FileName}$$

LIST HISTORY

$$\text{LI[ST]H[ISTORY]} \left\{ \begin{array}{l} \text{CommandNumber} \\ \text{\textcircled{C}} \end{array} \right\}$$

LIST INSTALL

$$\text{LI[ST]I[NSTALL]} \text{FileName}$$

LIST SET

LI[ST]S[ET] { *Option* }
 @

LOAD

LO[AD] [P[ARTIAL]] FROM { E[XTERNAL] } *InputFileName* [AT *StartingRow*]
 I[NTERNAL]
[FOR *NumberOfRows*] TO { [Owner.] *TableName* } [*ExternalInputSpec*]
 [Owner.] *ViewName* } [USING *DescriptionFileName*]
{ Y[ES] *PatternLocation Pattern* }
{ N[O]

ExternalInputSpec

{ *ColumnName StartingLocation Length* [*NullIndicator*] } [...] E[ND]
 [*FormatType*]

RECALL

REC[ALL] { C[URRENT]
 F[ILE] *FileName*
 H[ISTORY] *CommandNumber* }

REDO

RED[O] [*CommandNumber*]
 [*CommandString*]

Subcommands

B Break
D Delete
E Exit
H Help
I Insert
L List
R Replace
X Execute
+[*n*] Forward *n*
-[*n*] Backward *n*
Return Next Line

RENAME

REN[AME] *OldFileName NewFileName*

SELECTSTATEMENT

SelectStatement; [PA[USE];] [*BrowseOption*;] [...] E[ND]

SET

SE[T] *Option OptionValue*

Options and Values

AUTO[COMMIT] ON | OFF
AUTO[LOCK] ON | OFF
AUTO[SAVE] *NumberOfRows*
C[ONTINUE] ON | OFF
CONV[ERT] ASCII | EBCDIC | OFF
EC[HO] ON | OFF
ECHO_[ALL] ON | OFF
EDITOR *EditorName*
ES[CAPE] *Character*
EXIT[_ON_DBERR] ON | OFF
EXIT_ON_DBWARN ON | OFF
FL[AGGER] *FlaggerName*
F[RACTION] *Length*
N[ULL] [*Character*]
OU[TPUT] *FileName*
OW[NER] *OwnerName*
LOAD_B[UFFER] *BufferSize*
PA[GEWIDTH] *PageWidth*
PR[OMPT] *PromptString*

SQLGEN

SQLG[EN]

SQLUTIL

SQLU[TIL]

START

STA[RT] [*CommandFileName*] [(*Value* [, *Value*] [...])]

STORE

STO[RE] *FileName* [R[EPLACE]]

SYSTEM

{ SY[STEM] } [*MPE/iXCommand*]
:

UNLOAD

U[NLOAD] TO { E[XTERNAL] } *OutputFileName*
 { I[NTERNAL] }
FROM { [*Owner.*] *TableName* } *ExternalOutputSpec*
 { [*Owner.*] *ViewName* }
 { "*SelectCommand*" }

ExternalOutputSpec

DescriptionFileName { *OutputLength* [*FractionLength*] } [...]
 [*NullIndicator*]

Locks Held on the System Catalog by SQL Statements

This appendix shows the locks that you can obtain on the tables in the system catalog. This appendix may be useful in determining the source of repeated locking problems. For more information, refer to the chapter “Concurrency Control Through Locks and Isolation Levels” in the *ALLBASE/SQL Reference Manual*.

If an SQL statement listed in Table E-2 (which appears later in this chapter) is embedded in a procedure or a preprocessed application and a section is stored for the statement, system catalog pages will also be locked as follows:

- At INSTALL time (in ISQL), preprocess time, or CREATE PROCEDURE time (ISQL): exclusive page locks on SYSTEM.SECTION.
- At run time or EXECUTE PROCEDURE time: shared page locks on SYSTEM.SECTION. If the section is found to be invalid at run time, all pages accessed for a PREPARE statement could be locked in addition to the pages normally locked for the statement contained in the section.
- At VALIDATE time: exclusive page locks on SYSTEM.SECTION pages containing invalid sections.

As an example, consider an UPDATE statement embedded in an application program. When you preprocess the application, a page in the SYSTEM.SECTION table is locked exclusively as the new section is added. When you run the application, the EXECUTE statement implicitly runs as the stored section executes. EXECUTE obtains share locks on pages in SYSTEM.SECTION. If the section becomes invalid and then you run the application again, the PREPARE statement implicitly runs, obtaining an exclusive lock on pages in SYSTEM.SECTION; then the EXECUTE statement implicitly runs as the stored section executes.

Some of the system catalog views overlap with each other, because they are based on the same underlying table. The following groups of system views overlap in such a way that a lock on one member of the group effectively is a lock on all the members of the group:

- Views containing definitions, based on the table HPRDBSS.VIEWDEF:
 - SYSTEM.VIEWDEF
 - SYSTEM.CHECKDEF
 - SYSTEM.RULEDEF
 - SYSTEM.PROCEDUREDEF
- Views with index information, based on the table HPRDBSS.INDEX:
 - SYSTEM.INDEX
 - SYSTEM.HASH
 - SYSTEM.CONSTRAINTINDEX
- Views containing defaults, based on the table HPRDBSS.COLDEFAULT:
 - SYSTEM.COLDEFAULT
 - SYSTEM.PARAMDEFAULT

- Views containing file definitions, based on the table HPRDBSS.DBEFIELD:
 - SYSTEM.DBEFIELD
 - SYSTEM.TEMPSPACE
- Views containing authorizations, based on the table HPRDBSS.MODAUTH:
 - SYSTEM.MODAUTH
 - SYSTEM.PROCAUTH

Table E-1 lists the base tables from which the system catalog views are derived.

Table E-1. Mapping the System Views to the Base System Tables

View Name	Table Name
SYSTEM.CHECKDEF	HPRDBSS.VIEWDEF
SYSTEM.COLAUTH	HPRDBSS.COLAUTH
SYSTEM.COLDEFAULT	HPRDBSS.COLDEFAULT
SYSTEM.COLUMN	HPRDBSS.COLUMN
SYSTEM.CONSTRAINT	HPRDBSS.CONSTRAINT
SYSTEM.CONSTRAINTCOL	HPRDBSS.CONSTRAINTCOL
SYSTEM.CONSTRAINTINDEX	HPRDBSS.INDEX
SYSTEM.DBEFIELD	HPRDBSS.DBEFIELD
SYSTEM.DBEFIELDSET	HPRDBSS.DBEFIELDSET
SYSTEM.GROUP	HPRDBSS.GROUP
SYSTEM.HASH	HPRDBSS.INDEX
SYSTEM.IMAGEKEY	HPRDBSS.INDEX
SYSTEM.INDEX	HPRDBSS.INDEX
SYSTEM.INSTALLAUTH	HPRDBSS.MODAUTH
SYSTEM.MODAUTH	HPRDBSS.MODAUTH
SYSTEM.PARAMDEFAULT	HPRDBSS.COLDEFAULT
SYSTEM.PARAMETER	HPRDBSS.PARAMETER
SYSTEM.PARTITION	HPRDBSS.PARTITION
SYSTEM.PROCAUTH	HPRDBSS.MODAUTH
SYSTEM.PROCEDURE	HPRDBSS.PROCEDURE
SYSTEM.PROCEDUREDEF	HPRDBSS.VIEWDEF
SYSTEM.PROCRESULT	HPRDBSS.PROCRESULT
SYSTEM.RULE	HPRDBSS.RULE
SYSTEM.RULECOLUMN	HPRDBSS.RULECOLUMN
SYSTEM.RULEDEF	HPRDBSS.VIEWDEF
SYSTEM.SECTION	HPRDBSS.SECTION
SYSTEM.SETOPTINFO	HPRDBSS.SETOPTINFO
SYSTEM.SPACEAUTH	HPRDBSS.SPACEAUTH
SYSTEM.SPACEDEFAULT	HPRDBSS.SPACEDEFAULT
SYSTEM.SPECAUTH	HPRDBSS.SPECAUTH
SYSTEM.TABAUTH	HPRDBSS.TABAUTH
SYSTEM.TABLE	HPRDBSS.TABLE
SYSTEM.TEMPSPACE	HPRDBSS.DBEFIELD
SYSTEM.TPINDEX	HPRDBSS.INDEX
SYSTEM.VIEWDEF	HPRDBSS.VIEWDEF

Locks on the system catalog are held to the end of the transaction, no matter what isolation level is used, to ensure the integrity of database objects while they are being used.

Whenever the HPRDBSS.SECTION table is locked, similar locks are placed on a STOREDSECT.DBFileSetName table.

Some statements by their nature incorporate one or more other SQL statements. When a statement from the following list includes other SQL statements, locks may also be obtained for each statement incorporated:

- CREATE PROCEDURE (most SQL statements)
- CREATE SCHEMA (data definition statements)
- CREATE VIEW (SELECT statement)
- DECLARE CURSOR (SELECT statement or EXECUTE PROCEDURE statement)
- EXECUTE IMMEDIATE (most SQL statements)
- EXECUTE (most SQL statements, when not valid)
- GENPLAN (SELECT, UPDATE, or DELETE statements)
- PREPARE (most SQL statements)
- VALIDATE (SQL statements contained in stored sections or procedures)

The following group of statements used only within procedures do *not* obtain locks:

- Assignment
- BEGIN ... END
- DECLARE Variable
- GOTO
- IF ... THEN ... ELSE ... ENDIF
- Labeled Statement
- PRINT
- RETURN
- WHILE ... DO ... ENDWHILE

Note The information in this appendix is general in nature, and it shows worst case locking for a particular SQL statement. Not all locks are necessarily requested in all instances.

Table E-2. Locks Held on the System Catalog by SQL Statements

SQL Statement	System Table	Type of Lock
ADD DBEFILE	HPRDBSS.DBEFILE HPRDBSS.DBEFILESET HPRDBSS.SPECAUTH	Exclusive Exclusive Shared
ADD TO GROUP	HPRDBSS.GROUP HPRDBSS.SPECAUTH	Exclusive Shared
ADVANCE	HPRDBSS.SECTION Same as statement in procedure if not valid HPRDBSS.SPECAUTH	Shared Same as statement in procedure if not valid Shared
ALTER DBEFILE	HPRDBSS.DBEFILE HPRDBSS.DBEFILESET HPRDBSS.SPECAUTH	Exclusive Shared Shared
ALTER TABLE	HPRDBSS.CHECKDEF HPRDBSS.COLUMN HPRDBSS.COLDEFAULT HPRDBSS.CONSTRAINT HPRDBSS.CONSTRAINTCOL HPRDBSS.CONSTRAINTINDEX HPRDBSS.DBEFILESET HPRDBSS.SECTION HPRDBSS.SPECAUTH HPRDBSS.TABLE	Exclusive Exclusive Exclusive Exclusive Exclusive Exclusive Shared Exclusive Shared Exclusive
BEGIN ARCHIVE	HPRDBSS.SPECAUTH	Shared
BEGIN DECLARE SECTION		
BEGIN WORK		
CHECKPOINT	HPRDBSS.SPECAUTH	Shared
CLOSE (select cursor)	HPRDBSS.SECTION	Shared
CLOSE (procedure cursor)	HPRDBSS.SECTION Same as statement in procedure if not valid	Shared Same as statement in procedure if not valid
COMMIT ARCHIVE	HPRDBSS.SPECAUTH	Shared
COMMIT WORK		
CONNECT	HPRDBSS.GROUP HPRDBSS.SPECAUTH	Shared Shared
CREATE DBEFILE	HPRDBSS.DBEFILE HPRDBSS.SPECAUTH HPRDBSS.TEMPSPACE	Exclusive Shared Shared
CREATE DBEFILESET	HPRDBSS.DBEFILESET HPRDBSS.GROUP HPRDBSS.SPECAUTH	Exclusive Shared Shared

**Table E-2.
Locks Held on the System Catalog by SQL Statements (continued)**

SQL Statement	System Table	Type of Lock
CREATE GROUP	HPRDBSS.COLAUTH	Shared
	HPRDBSS.GROUP	Exclusive
	HPRDBSS.MODAUTH	Shared
	HPRDBSS.SECTION	Shared
	HPRDBSS.SPECAUTH	Shared
	HPRDBSS.TABAUTH	Shared
	HPRDBSS.TABLE	Shared
CREATE INDEX	HPRDBSS.COLUMN	Shared
	HPRDBSS.GROUP	Shared
	HPRDBSS.INDEX	Exclusive
	HPRDBSS.SECTION	Exclusive
	HPRDBSS.TABAUTH	Shared
	HPRDBSS.TABLE	Exclusive
CREATE PROCEDURE	HPRDBSS.GROUP	Shared
	HPRDBSS.PROCEDURE	Exclusive
	HPRDBSS.PROCEDUREDEF	Exclusive
	HPRDBSS.PARAMDEFAULT	Exclusive
	HPRDBSS.PARAMETER	Exclusive
	HPRDBSS.PROCRESULT	Exclusive
	HPRDBSS.SECTION	Exclusive
	HPRDBSS.SPECAUTH	Shared
Views accessed by SQL statements in the procedure	Same as for SQL statements in the procedure	
CREATE RULE	HPRDBSS.COLUMN	Shared
	HPRDBSS.GROUP	Shared
	HPRDBSS.RULE	Exclusive
	HPRDBSS.RULECOLUMN	Exclusive
	HPRDBSS.RULEDEF	Exclusive
	HPRDBSS.SPECAUTH	Shared
	HPRDBSS.TABLE	Exclusive
CREATE SCHEMA	HPRDBSS.COLAUTH	Shared
	HPRDBSS.GROUP	Shared
	HPRDBSS.SECTION	Shared
	HPRDBSS.SPECAUTH	Shared
	HPRDBSS.TABAUTH	Shared
	HPRDBSS.TABLE	Shared
	Views accessed by data definition statements in the schema	Same as for data definition statements in the schema

**Table E-2.
Locks Held on the System Catalog by SQL Statements (continued)**

SQL Statement	System Table	Type of Lock
CREATE TABLE	HPRDBSS.CHECKDEF	Exclusive
	HPRDBSS.COLAUTH	Shared
	HPRDBSS.COLDEFAULT	Exclusive
	HPRDBSS.COLUMN	Exclusive
	HPRDBSS.CONSTRAINT	Exclusive
	HPRDBSS.CONSTRAINTCOL	Exclusive
	HPRDBSS.CONSTRAINTINDEX	Exclusive
	HPRDBSS.DBFILESET	Exclusive
	HPRDBSS.GROUP	Shared
	HPRDBSS.HASH	Exclusive
	HPRDBSS.SECTION	Exclusive
	HPRDBSS.SPECAUTH	Shared
	HPRDBSS.TBAUTH	Exclusive (for PUBLIC or PUBLICREAD tables)
	HPRDBSS.TABLE	Exclusive and Shared
CREATE TEMPSPACE	HPRDBSS.DBFILESET	Exclusive
	HPRDBSS.GROUP	Shared
	HPRDBSS.SPECAUTH	Shared
	HPRDBSS.TEMPSPACE	Exclusive
CREATE VIEW	HPRDBSS.COLUMN	Exclusive
	HPRDBSS.GROUP	Shared
	HPRDBSS.SECTION	Exclusive
	HPRDBSS.SPECAUTH	Shared
	HPRDBSS.TABLE	Exclusive
	HPRDBSS.VIEWDEF	Exclusive
Same as for SELECT	Same as for SELECT	
DECLARE CURSOR	Same as for SELECT or EXECUTE PROCEDURE	Same as for SELECT or EXECUTE PROCEDURE
DELETE	HPRDBSS.COLUMN	Shared
	HPRDBSS.CONSTRAINT	Shared
	HPRDBSS.CONSTRAINTINDEX	Shared
	HPRDBSS.DBFILESET	Shared
	HPRDBSS.GROUP	Shared
	HPRDBSS.HASH	Shared
	HPRDBSS.INDEX	Shared
	HPRDBSS.MODAUTH	Shared
	HPRDBSS.PROCEDURE	Shared
	HPRDBSS.RULE	Shared
	HPRDBSS.SECTION	Shared
	HPRDBSS.SPECAUTH	Shared
	HPRDBSS.TBAUTH	Shared

**Table E-2.
Locks Held on the System Catalog by SQL Statements (continued)**

SQL Statement	System Table	Type of Lock
DELETE WHERE CURRENT	HPRDBSS.COLUMN HPRDBSS.CONSTRAINT HPRDBSS.CONSTRAINTINDEX HPRDBSS.DBFILESET HPRDBSS.GROUP HPRDBSS.INDEX HPRDBSS.MODAUTH HPRDBSS.PROCEDURE HPRDBSS.RULE HPRDBSS.SECTION HPRDBSS.SPECAUTH HPRDBSS.TBAUTH	Shared Shared Shared Shared Shared Shared Shared Shared Shared Shared Shared Shared
DESCRIBE	HPRDBSS.DBFILESET HPRDBSS.GROUP HPRDBSS.MODAUTH HPRDBSS.SECTION HPRDBSS.SPECAUTH Same as for statement being DESCRIBED	Shared Shared Shared Shared Shared Same as for statement being DESCRIBED
DROP DBEFILE	HPRDBSS.DBEFILE HPRDBSS.GROUP HPRDBSS.SECTION HPRDBSS.SPECAUTH HPRDBSS.TEMPSPACE	Exclusive Shared Exclusive Shared Exclusive
DROP DBEFILESET	HPRDBSS.DBFILESET HPRDBSS.GROUP HPRDBSS.SECTION HPRDBSS.SPECAUTH	Exclusive Shared Exclusive Shared
DROP GROUP	HPRDBSS.COLAUTH HPRDBSS.CONSTRAINTCOL HPRDBSS.GROUP HPRDBSS.MODAUTH HPRDBSS.SECTION HPRDBSS.SPECAUTH HPRDBSS.TBAUTH HPRDBSS.TABLE	Exclusive Exclusive Exclusive Exclusive Shared Exclusive Exclusive Shared

**Table E-2.
Locks Held on the System Catalog by SQL Statements (continued)**

SQL Statement	System Table	Type of Lock
DROP INDEX	HPRDBSS.GROUP	Shared
	HPRDBSS.INDEX	Exclusive
	HPRDBSS.SECTION	Exclusive
	HPRDBSS.SPECAUTH	Shared
	HPRDBSS.TABAUTH	Shared
	HPRDBSS.TABLE	Exclusive
DROP MODULE	HPRDBSS.GROUP	Shared
	HPRDBSS.MODAUTH	Exclusive
	HPRDBSS.SECTION	Exclusive
DROP PROCEDURE	HPRDBSS.GROUP	Shared
	HPRDBSS.MODAUTH	Exclusive
	HPRDBSS.PARAMETER	Exclusive
	HPRDBSS.PARAMDEFAULT	Exclusive
	HPRDBSS.PROCEDURE	Exclusive
	HPRDBSS.PROCEDUREDEF	Exclusive
	HPRDBSS.PROCRESULT	Exclusive
	HPRDBSS.SECTION	Exclusive
HPRDBSS.SPECAUTH	Shared	
DROP RULE	HPRDBSS.GROUP	Shared
	HPRDBSS.RULE	Exclusive
	HPRDBSS.RULECOLUMN	Exclusive
	HPRDBSS.RULEDEF	Exclusive
	HPRDBSS.SECTION	Exclusive
	HPRDBSS.SPECAUTH	Shared
HPRDBSS.TABLE	Exclusive	
DROP TABLE	HPRDBSS.COLAUTH	Exclusive
	HPRDBSS.COLUMN	Exclusive
	HPRDBSS.CONSTRAINT	Exclusive and Shared
	HPRDBSS.CONSTRAINTCOL	Exclusive
	HPRDBSS.CONSTRAINTINDEX	Exclusive
	HPRDBSS.GROUP	Shared
	HPRDBSS.HASH	Exclusive
	HPRDBSS.INDEX	Exclusive
	HPRDBSS.RULE	Exclusive
	HPRDBSS.RULECOLUMN	Exclusive
	HPRDBSS.RULEDEF	Exclusive
	HPRDBSS.SECTION	Exclusive
	HPRDBSS.SPECAUTH	Shared
	HPRDBSS.TABAUTH	Exclusive
	HPRDBSS.TABLE	Exclusive
HPRDBSS.VIEWDEF	Exclusive	

**Table E-2.
Locks Held on the System Catalog by SQL Statements (continued)**

SQL Statement	System Table	Type of Lock
DROP TEMPSPACE	HPRDBSS.DBFILESET HPRDBSS.SPECAUTH HPRDBSS.TEMPSPACE	Exclusive Shared Exclusive
DROP VIEW	HPRDBSS.COLAUTH HPRDBSS.COLUMN HPRDBSS.GROUP HPRDBSS.SECTION HPRDBSS.SPECAUTH HPRDBSS.TABAUTH HPRDBSS.TABLE HPRDBSS.VIEWDEF	Exclusive Exclusive Shared Exclusive Shared Exclusive Exclusive Exclusive
END DECLARE SECTION		
EXECUTE	HPRDBSS.GROUP HPRDBSS.MODAUTH HPRDBSS.SECTION HPRDBSS.SPECAUTH Views accessed by SQL statements executed if not valid	Shared Shared Shared Shared Same as for statements executed if not valid
EXECUTE IMMEDIATE	Views accessed by SQL statements executed	Same as for statements executed
FETCH		
EXECUTE PROCEDURE	HPRDBSS.MODAUTH HPRDBSS.PARAMDEFAULT HPRDBSS.PARAMETER HPRDBSS.PROCEDURE HPRDBSS.PROCRESULT HPRDBSS.SECTION Same as statement in procedure if not valid	Shared Shared Shared Shared Shared Shared Same as statement in procedure if not valid
GENPLAN	HPRDBSS.PLAN Same as SELECT	Exclusive Same as SELECT
GRANT Table Authority GRANT UPDATE GRANT REFERENCES	HPRDBSS.COLAUTH HPRDBSS.GROUP HPRDBSS.SPECAUTH HPRDBSS.TABAUTH HPRDBSS.TABLE	Exclusive Shared Shared Exclusive Shared
GRANT RUN GRANT EXECUTE GRANT INSTALL	HPRDBSS.GROUP HPRDBSS.MODAUTH HPRDBSS.SPECAUTH	Shared Exclusive Shared

**Table E-2.
Locks Held on the System Catalog by SQL Statements (continued)**

SQL Statement	System Table	Type of Lock
GRANT Special Authorities	HPRDBSS.SPECAUTH	Exclusive
	HPRDBSS.GROUP	Shared
GRANT SECTIONSPACE GRANT TABLESPACE	HPRDBSS.SPACEAUTH	Exclusive
	HPRDBSS.SPECAUTH	Shared
	HPRDBSS.GROUP	Shared
INCLUDE		
INSERT	HPRDBSS.COLUMN	Shared
	HPRDBSS.COLDEFAULT	Shared
	HPRDBSS.CONSTRAINT	Shared
	HPRDBSS.CONSTRAINTINDEX	Shared
	HPRDBSS.DBFILESET	Shared
	HPRDBSS.GROUP	Shared
	HPRDBSS.HASH	Shared
	HPRDBSS.INDEX	Shared
	HPRDBSS.MODAUTH	Shared
	HPRDBSS.PROCEDURE	Shared
	HPRDBSS.RULE	Shared
	HPRDBSS.SECTION	Shared
	HPRDBSS.SPECAUTH	Shared
	HPRDBSS.TBAUTH	Shared
	HPRDBSS.TABLE	Shared
LOCK TABLE	HPRDBSS.GROUP	Shared
	HPRDBSS.SPECAUTH	Shared
	HPRDBSS.TBAUTH	Shared
	HPRDBSS.TABLE	Shared
OPEN	HPRDBSS.GROUP	Shared
	HPRDBSS.MODAUTH	Shared
	HPRDBSS.SECTION	Shared
	HPRDBSS.SPECAUTH	Shared
	Same as for SELECT or EXECUTE PROCEDURE in DECLARE CURSOR statement	Same as for SELECT or EXECUTE PROCEDURE in DECLARE CURSOR statement
PREPARE (Permanent sections created with ISQL)	HPRDBSS.DBFILESET	Shared
	HPRDBSS.GROUP	Shared
	HPRDBSS.SECTION	Exclusive
	HPRDBSS.SPECAUTH	Shared
Views accessed for SQL statements prepared	Same as for statements prepared	
PREPARE (Temporary sections created in applications)	HPRDBSS.DBFILESET	Shared
	HPRDBSS.GROUP	Shared
	HPRDBSS.SPECAUTH	Shared
	Views accessed for statements PREPARED	Same as for statements prepared

**Table E-2.
Locks Held on the System Catalog by SQL Statements (continued)**

SQL Statement	System Table	Type of Lock
RAISE ERROR		
REFETCH		
RELEASE		
REMOVE DBEFILE	HPRDBSS.DBEFILE HPRDBSS.DBEFILESET HPRDBSS.SPECAUTH HPRDBSS.TEMPSPACE	Exclusive Exclusive Shared Exclusive
REMOVE FROM GROUP	HPRDBSS.CONSTRAINTCOL HPRDBSS.GROUP HPRDBSS.SECTION HPRDBSS.SPECAUTH	Shared Exclusive Exclusive Exclusive
RESET	HPRDBSS.SPECAUTH	Shared
REVOKE Table Authority REVOKE UPDATE REVOKE REFERENCES	HPRDBSS.COLAUTH HPRDBSS.CONSTRAINTCOL HPRDBSS.GROUP HPRDBSS.SECTION HPRDBSS.TABAUTH HPRDBSS.TABLE	Exclusive Shared Shared Exclusive Exclusive Shared
REVOKE EXECUTE REVOKE RUN	HPRDBSS.GROUP HPRDBSS.MODAUTH HPRDBSS.SECTION HPRDBSS.SPECAUTH	Shared Exclusive Exclusive Shared
REVOKE Special Authorities	HPRDBSS.CONSTRAINTCOL HPRDBSS.GROUP HPRDBSS.SECTION HPRDBSS.SPECAUTH	Shared Shared Exclusive Exclusive
ROLLBACK WORK		
SAVEPOINT		
SELECT	HPRDBSS.COLUMN HPRDBSS.DBEFILESET HPRDBSS.GROUP HPRDBSS.SECTION HPRDBSS.SPECAUTH HPRDBSS.TABAUTH HPRDBSS.TABLE If table has index: HPRDBSS.CONSTRAINTINDEX HPRDBSS.HASH HPRDBSS.INDEX	Shared Shared Shared Shared in user programs; none in ISQL Shared Shared Shared Shared Shared Shared
SQL EXPLAIN		

**Table E-2.
Locks Held on the System Catalog by SQL Statements (continued)**

SQL Statement	System Table	Type of Lock
START DBE	HPRDBSS.SPECAUTH	Shared (Exclusive the first time the statement is used after the DBECreator has changed following a restore)
	HPRDBSS.GROUP	Shared
START DBE NEW		System catalog is being created
START DBE NEWLOG	HPRDBSS.SPECAUTH	Shared
START DBE NEWLOG (After migration)	HPRDBSS.SPECAUTH	Shared
	HPRDBSS.COLUMN	Exclusive
	HPRDBSS.SECTION	Exclusive
	HPRDBSS.TABLE	Exclusive
STOP DBE	HPRDBSS.SPECAUTH	Shared
TERMINATE USER	HPRDBSS.SPECAUTH	Shared
TRANSFER OWNERSHIP (PROCEDURE)	HPRDBSS.GROUP	Shared
	HPRDBSS.PARAMDEFAULT	Exclusive
	HPRDBSS.PARAMETER	Exclusive
	HPRDBSS.MODAUTH	Exclusive
	HPRDBSS.PROCEDURE	Exclusive
	HPRDBSS.PROCEDUREDEF	Exclusive
	HPRDBSS.PROCRESULT	Exclusive
	HPRDBSS.SECTION	Exclusive
HPRDBSS.SPECAUTH	Exclusive	
TRANSFER OWNERSHIP (TABLE)	HPRDBSS.CHECKDEF	Exclusive
	HPRDBSS.CONSTRAINT	Exclusive
	HPRDBSS.CONSTRAINTCOL	Exclusive
	HPRDBSS.COLAUTH	Exclusive
	HPRDBSS.COLUMN	Exclusive
	HPRDBSS.CONSTRAINTINDEX	Exclusive
	HPRDBSS.GROUP	Shared
	HPRDBSS.HASH	Exclusive
	HPRDBSS.INDEX	Exclusive
	HPRDBSS.RULE	Exclusive
	HPRDBSS.RULECOLUMN	Exclusive
	HPRDBSS.RULEDEF	Exclusive
	HPRDBSS.SPECAUTH	Shared
	HPRDBSS.TABAUTH	Exclusive
HPRDBSS.TABLE	Exclusive	
TRANSFER OWNERSHIP (VIEW)	HPRDBSS.CONSTRAINT	Exclusive
	HPRDBSS.COLAUTH	Exclusive
	HPRDBSS.COLUMN	Exclusive
	HPRDBSS.SPECAUTH	Shared
	HPRDBSS.TABAUTH	Exclusive
	HPRDBSS.TABLE	Exclusive
HPRDBSS.VIEWDEF	Exclusive	

**Table E-2.
Locks Held on the System Catalog by SQL Statements (continued)**

SQL Statement	System Table	Type of Lock
TRANSFER OWNERSHIP (GROUP)	HPRDBSS.GROUP	Exclusive
	HPRDBSS.SPECAUTH	Shared
UPDATE	HPRDBSS.COLAUTH	Shared
	HPRDBSS.COLUMN	Shared
	HPRDBSS.CONSTRAINT	Shared
	HPRDBSS.CONSTRAINTINDEX	Shared
	HPRDBSS.DBFILESET	Shared
	HPRDBSS.GROUP	Shared
	HPRDBSS.HASH	Shared
	HPRDBSS.INDEX	Shared
	HPRDBSS.MODAUTH	Shared
	HPRDBSS.PROCEDURE	Shared
	HPRDBSS.RULE	Shared
	HPRDBSS.RULECOLUMN	Shared
	HPRDBSS.SECTION	Shared
	HPRDBSS.SPECAUTH	Shared
	HPRDBSS.TABAUTH	Shared
HPRDBSS.TABLE	Shared	
UPDATE STATISTICS	HPRDBSS.COLUMN	Exclusive
	HPRDBSS.CONSTRAINT	Shared
	HPRDBSS.CONSTRAINTINDEX	Exclusive
	HPRDBSS.DBFILE	Exclusive
	HPRDBSS.DBFILESET	Exclusive
	HPRDBSS.HASH	Exclusive
	HPRDBSS.INDEX	Exclusive
	HPRDBSS.TABLE	Exclusive
HPRDBSS.TEMPSPACE	Exclusive	
UPDATE WHERE CURRENT	HPRDBSS.COLAUTH	Shared
	HPRDBSS.COLUMN	Shared
	HPRDBSS.CONSTRAINT	Shared
	HPRDBSS.DBFILESET	Shared
	HPRDBSS.GROUP	Shared
	HPRDBSS.INDEX	Shared
	HPRDBSS.MODAUTH	Shared
	HPRDBSS.PROCEDURE	Shared
	HPRDBSS.RULE	Shared
	HPRDBSS.RULECOLUMN	Shared
	HPRDBSS.SECTION	Shared
	HPRDBSS.SPECAUTH	Shared
	HPRDBSS.TABAUTH	Shared
	HPRDBSS.TABLE	Shared

**Table E-2.
Locks Held on the System Catalog by SQL Statements (continued)**

SQL Statement	System Table	Type of Lock
VALIDATE MODULE or VALIDATE PROCEDURE	HPRDBSS.MODAUTH HPRDBSS.PROCEDURE HPRDBSS.SECTION Views accessed for the SQL statements in the sections being validated	Shared Exclusive Exclusive Same as for sections being validated
WHENEVER		

SQLUtil

The following pages contain SQLUtil command syntax. A discussion of how to use SQLUtil is in the “DBA Tasks and Tools” chapter.

Note As in other ALLBASE/SQL utilities, you can use a semicolon following any SQLUtil command. The semicolon is not required, however.

ADDLOG

Adds a new log file to the DBEnvironment.

Scope

SQLUtil Only

SQLUtil Syntax

```
>> ADDLOG
DBEnvironment Name: DBEnvironmentName
Maintenance Word: MaintenanceWord
Enter Log File Name(s) Separated by a Blank? LogFileName1 [LogFileName2]
New Log File Size? FileSize
Add Log File (y/n)? { Y[es] }
                   { N[o] }
```

Parameters

DBEnvironmentName is the name of a DBEnvironment.

Maintenance Word is the maintenance word.

LogFileName1,
LogFileName2 is the physical (MPE/iX) name of the log file that is to be added.
LogFileName2 is included only if you are using dual logging.

FileSize is the size of the new log file.

Description

- This command starts a DBEnvironment session, and lets you add one new log file. The DBEnvironment can be in use when this command is executed.
- The new file is added in the same account as the DBECon file. You can specify a different group name. If you specify an account name, it must be the same as that of the DBECon file.
- Each time ALLBASE/SQL switches to a new log file, a checkpoint is taken. Therefore, the size and number of log files can control the frequency of checkpoints. If you repeatedly see a *Log Full* message, add a new log file of the appropriate size.
- In nonarchive logging mode, add as many log files as you need (one at a time, with separate ADDLOG commands) for a total size that can accommodate your largest transaction carried out by the maximum number of concurrent users.
- In archive logging mode, add as many log files as you need (one at a time, with separate ADDLOG commands) to permit an ongoing cycle of STORELOG operations which make log files available.
- ADDLOG prompts for the names of files to be added. You may enter a single log file name or, in the case of dual logging, you may enter two file names. ADDLOG will not accept more than two file names on a single line.

- The maximum number of log files in a DBEnvironment is 34. Since one log is created when you start the DBEnvironment, the maximum number of log files you can add to a DBEnvironment is 33.

Authorization

You must be the DBECreator or provide the correct maintenance word to use this command.

Example

```
>> ADDLOG
DBEnvironment Name: PARTSDBE.SOMEGRP.SOMEACCT
Maintenance Word: Return
Enter Log File Name(s) Separated by a Blank? LGM1.SOMEGRP.SOMEACCT
New Log File Size? 300
Add Log File (y/n)? y

Log file 'LGM1.SOMEGRP.SOMEACCT' was Added.
Log Identifier Is: 2
```

ALTDDBE

Changes the startup parameters for a DBEnvironment.

Scope

SQLUtil Only

SQLUtil Syntax

```
>> ALTDDBE
DBEnvironment Name: DBEnvironmentName
Maintenance Word: MaintenanceWord
AutoStart Mode (on/off) (opt): AutoStartMode
User Mode (single/multi) (opt): UserMode
DDL Enabled (y/n) (opt): DDLEnabled
No. of Runtime Control Block Pages (opt): ControlBlockPages
No. of Data Buffer Pages (opt): DataBufferPages
Data Buffer Pages Memory Resident (y/n) (opt): MemoryResidentEnabled
No. of Log Buffer Pages (opt): LogBufferPages
Max. Transactions (opt): MaxTransactions

Maximum Timeout (opt):  $\left[ \begin{array}{l} \textit{MaximumTimeout} \left[ \begin{array}{l} \text{SECONDS} \\ \text{MINUTES} \end{array} \right] \\ \text{NONE} \end{array} \right]$ 

Default Timeout (opt):  $\left[ \begin{array}{l} \textit{DefaultTimeout} \left[ \begin{array}{l} \text{SECONDS} \\ \text{MINUTES} \end{array} \right] \\ \text{MAXIMUM} \end{array} \right]$ 

Authorize Once per Session (on/off) (opt): AuthorizeOnce
Alter DBEnvironment Startup Parameters (y/n)?  $\left\{ \begin{array}{l} \text{Y}[\text{es}] \\ \text{N}[\text{o}] \end{array} \right\}$ 
```

Note You must stop the DBEnvironment before you can issue the ALTDDBE command.

Parameters

<i>DBEnvironmentName</i>	is the name of the DBEnvironment whose startup parameters are to be altered.
<i>MaintenanceWord</i>	is the maintenance word of the DBEnvironment.
<i>AutoStartMode</i>	is ON to enable auto starting of a multiuser DBEnvironment session when a CONNECT command is given. It is OFF to disable this feature. The default is ON.
<i>UserMode</i>	is SINGLE to specify that the DBEnvironment should be started in single-user mode when AutoStart is enabled. It is MULTI to specify multiuser operation. The default is determined by the value you supply in the START DBE NEW command. If you do not supply a value, the default is SINGLE.

<i>DDLEnabled</i>	is YES to enable data definition language (DDL), or NO to disable DDL. The default is YES.
<i>ControlBlockPages</i>	is the number of 4096-byte pages to allocate to the pool of shared memory used by DBCore services in a multi-user DBEnvironment.
<i>DataBufferPages</i>	is the number of 4096-byte data buffer pages for holding data from tables and indexes during program execution.
<i>MemoryResidentEnabled</i>	is YES to specify that data buffer pages will remain resident in memory. It is NO to specify that data buffer pages will not be memory resident (they will be swapped in and out of memory as needed by ALLBASE/SQL). The default is NO.
<i>LogBufferPages</i>	is the number of 512-byte log buffers for holding log records during program execution.
<i>MaxTransactions</i>	is the maximum number of concurrent transactions to be supported.
<i>MaximumTimeout</i>	is an integer literal greater than zero. If the <i>MaximumTimeout</i> is not qualified by MINUTES, SECONDS is assumed. If <i>MaximumTimeout</i> is set to NONE, no timeouts are assumed. The default is NONE.
<i>DefaultTimeout</i>	is an integer literal greater than zero. If the <i>DefaultTimeout</i> is not qualified by MINUTES, SECONDS is assumed. If <i>DefaultTimeout</i> is set to MAXIMUM, its value is the same as <i>MaximumTimeout</i> . The default value of <i>DefaultTimeout</i> is also <i>MaximumTimeout</i> .
<i>AuthorizeOnce</i>	is set ON to permit users of ISV (independent software vendor) toolsets to obtain improved performance with dynamic queries. If you have preprocessed an application with the (DYNAMIC) option, or if you are using an ISV toolset, set this flag ON if you want authorization checks on your dynamic queries to be performed only the first time the query is executed during the user session. Set OFF if you want ALLBASE/SQL to do authorization checks on a dynamic query each time it is executed. The default is OFF.

Description

- The ALTDBE command updates the parameters required for DBEnvironment startup. You are prompted for new values for the parameters one by one; entering a retains the old value.
- Startup parameters have been initially defined by a START DBE NEW command which creates the DBEnvironment and stores these parameters in the DBECon file. These parameters will be used when the DBEnvironment is accessed by either a START DBE or a CONNECT command.
- Setting the *MemoryResidentEnabled* flag to YES is useful for database servers where very little is going on besides database work, and the database buffers occupy a small fraction of real memory in the system.

ALTDBE

- You get better performance from pseudo-mapped DBEFiles if you set the *MemoryResidentEnabled* flag to YES.
- This command can only be executed when the DBEnvironment is not in use.
- You cannot change archive mode with ALTDBE. Refer to the section “Choosing an Approach to Backup and Recovery” in the “Backup and Recovery” chapter.

Note

Scripts using the ALTDBE command that were prepared for use with releases of ALLBASE/SQL prior to F.0 will not work with this release unless you edit them to include responses for the memory resident data buffer flag, default and maximum user timeout, and the authorize once per session flag. Scripts using ALTDBE that were prepared for use with releases of ALLBASE/SQL prior to E.1 will not work unless you edit them to remove the response for archive mode and include responses for the memory resident data buffer flag, default and maximum user timeout, and the authorize once per session flag.

Authorization

You must provide the correct maintenance word or be the DBECreator to use this command.

Example

```
>> altdbe

DBEnvironment Name: PartsDBE
Maintenance Word: MaintWd
AutoStart Mode (on/off) (opt): on
User Mode (opt): multi
DDL Enabled (y/n) (opt): yes
No. of Runtime Control Block Pages (opt): 37
No. of Data Buffer Pages (opt): 100
Data Buffer Pages Memory Resident (y/n) (opt): no
No. of Log Buffer Pages (opt): 24
Max. Transactions (opt): 5
Maximum Timeout (opt): 10
Default Timeout (opt): 5
Authorize Once per Session (opt): on
Alter DBEnvironment Startup Parameters (y/n)? yes

DBEnvironment startup parameters altered.
```

ATTACHFILE

Attaches previously detached DBEFileSets and DBEFiles to the DBEnvironment and makes them available for normal access.

Scope

SQLUtil Only

SQLUtil Syntax

```
>> ATTACHFILE
DBEnvironment Name: DBEnvironmentName
Maintenance Word: MaintenanceWord
Enter DBEFileSet name or Carriage Return to Finish: DBEFileSetName
Enter DBEFileSet name or Carriage Return to Finish:
Enter DBEFile name or Carriage Return to Finish: SystemFileName
Enter DBEFile name or Carriage Return to Finish:
Do you wish to attach (y/n) { y[es] }
                          { n[o] }
```

Parameters

<i>DBEnvironmentName</i>	is the name of the DBEnvironment associated with the DBEFileSets and DBEFiles to be attached.
<i>MaintenanceWord</i>	is the maintenance word.
<i>DBEFileSetName</i>	is the name of the DBEFileSet to be attached. You may enter a list of DBEFileSets, one name per line, with a carriage return to terminate the list. You may enter a carriage return by itself if no DBEFileSets are to be attached.
<i>SystemFileName</i>	is the physical name of the DBEFile to be attached. You may enter a list of DBEFiles, one name per line, with a null line to terminate the list, or a carriage return by itself if no DBEFiles are to be added.

Description

- Attaching a DBEFileset makes all the DBEFiles in the DBEFileSet available for database access.
- Attaching explicitly named DBEFiles makes those specific files available for database access.
- You can determine whether a DBEFile is ATTACHED or DETACHED by looking at the column “ATTACHED” in the system views SYSTEM.DBFILE or CATALOG.DBFILE, or the “Static DBEFile” screen of SQLMON.
- Also refer to the SQLUtil DETACHFILE command.

ATTACHFILE

Authorization

You must be the DBECreator, have SM capability, or you must supply the correct maintenance word to use this command.

Example

```
>> attachfile
DBEnvironment Name: PartsDBE
Enter DBEFileset name or Carriage Return to Finish): WarehFS
Enter DBEFileset name or Carriage Return to Finish):
Enter DBEFile name or Carriage Return to Finish: OrderDF1
Enter DBEFile name or Carriage Return to Finish:
Do you wish to attach (y/n)? y

File(s) Attached.
```


CHANGELOG

Causes the DBEnvironment to change to the next log file.

Scope

SQLUtil Only

SQLUtil Syntax

```
>> CHANGELOG
DBEnvironment Name: DBEnvironmentName
Maintenance Word: MaintenanceWord
Change to a new log (y/n)?  $\left\{ \begin{array}{l} \text{y[es]} \\ \text{n[o]} \end{array} \right\}$ 
```

Parameters

DBEnvironmentName is the name of the DBEnvironment with which the log files are associated.

MaintenanceWord is the maintenance word.

Description

- The CHANGELOG command causes ALLBASE/SQL to switch from the currently active log file to the next available log file, if one exists.
- Once you switch to the next available log file, you can back up the last used log file.
- The CHANGELOG command also displays the sequence numbers of the log file active just prior to the change as well as the log file active after the change.

Authorization

You must be the DBECreator, have SM capability, or supply the correct maintenance word to use this command.

Example

```
>> changelog
DBEnvironment Name: PartsDBE
Maintenance Word: MaintenanceWord
Change to new log (y/n)? y

Changed log from Sequence Number 2 to Sequence Number 3.
```

DETACHFILE

Detaches DBEFileSets and DBEFiles from the DBEnvironment and makes them unavailable for normal access.

Scope

SQLUtil Only

SQLUtil Syntax

```
>> DETACHFILE
DBEnvironment Name: DBEnvironmentName
Maintenance Word: MaintenanceWord
Enter DBEFileSet name or Carriage Return to Finish: DBEFileSetName
Enter DBEFileSet name or Carriage Return to Finish:
Enter DBEFile name or Carriage Return to Finish: SystemFileName
Enter DBEFile name or Carriage Return to Finish:
Do you wish to detach (y/n)?  $\left\{ \begin{array}{l} y[es] \\ n[o] \end{array} \right\}$ 
```

Parameters

<i>DBEnvironmentName</i>	is the name of the DBEnvironment from which the DBEFileSets or DBEFiles are to be detached.
<i>MaintenanceWord</i>	is the maintenance word.
<i>DBEFileSetName</i>	is the name of the DBEFileset to be detached. You may enter a list of DBEFileSets, one name per line, followed by a null line to terminate the list.
<i>SystemFileName</i>	is the physical name of the DBEFile to be detached. You may enter a list of DBEFiles, one name per line, followed by a null line to terminate the list.

Description

- DBEFileSets and DBEFiles may be detached for partial recovery operations, security reasons, or because they are infrequently used.
- If you detach a DBEFileset, all the DBEFiles in the DBEFileset are detached.
- You cannot detach the SYSTEM DBEFileSet, DBEFile0, or any other DBEFiles that you have added to the SYSTEM DBEFileSet to expand its size.
- You can determine if a DBEFile is attached or detached by looking at the column “ATTACHED” in the system views SYSTEM.DBEFIL and CATALOG.DBEFIL, or the “Static DBEFile” screen of SQLMON.
- Also see the SQLUtil ATTACHFILE command.

Authorization

You must be the DBECreator, have SM capability, or supply the correct maintenance word to use this command.

Example

```
>> detachfile
DBEnvironment Name: PartsDBE
Maintenance Word: MaintenanceWord
Enter DBEFileset name or Carriage Return to Finish: WarehFS
Enter DBEFileset name or Carriage Return to Finish:
Enter DBEFile name or Carriage Return to Finish: OrderDF1
Enter DBEFile name or Carriage Return to Finish:
Do you wish to proceed (y/n)? y
```

ENDRECOVERY

Ends a rollforward recovery process that was started with SETUPRECOVERY.

Scope

SQLUtil Only

SQLUtil Syntax

```
>> ENDRECOVERY
DBEnvironment Name: DBEnvironmentName
Maintenance Word: MaintenanceWord
End Recovery (y/n)? { Y[es] }
                   { N[o] }
```

Parameters

DBEnvironmentName is the name of a DBEnvironment.

MaintenanceWord is the maintenance word.

Description

- This command starts a DBEnvironment session, and terminates a rollforward recovery process after all relevant log files have been applied in sequence.
- Rollforward recovery uses four commands:
 - SETUPRECOVERY—initiates the process.
 - RESTORELOG—restores a log file from a backup.
 - RECOVERLOG—issued by you for each log file being applied to the DBEnvironment.
 - ENDRECOVERY—ends the rollforward recovery process.
- You normally apply SETUPRECOVERY once. You can specify a *RecoverTime* as the end point for recovery. You then use RESTORELOG to restore each backup log file to the system in sequence number order, and you use RECOVERLOG once for each log file that you have restored, to apply it to the DBEnvironment. ALLBASE/SQL will recover the transactions in the file up to the *RecoverTime* you specified, or up to the end of the file. You use ENDRECOVERY once after recovering all the log files you wish to apply to the DBEnvironment. Normally, you do *not* use ENDRECOVERY if you specified a *RecoverTime* in the SETUPRECOVERY command.
- ENDRECOVERY will result in an error if you have not recovered enough files to make the DBEnvironment consistent. If you used STOREONLINE to do an online backup of the DBEnvironment, you must recover starting from the *First Log Sequence Number Needed for Recovery* up to and including the file that was active at the time the last STOREONLINE command completed. If you did a static backup of the DBEnvironment, you can recover as little or as much of the log as you desire once you have restored the most recent copy of the DBEnvironment.

Authorization

You must be the DBECreator or supply the correct maintenance word to use this command.

Example

```
>> setuprecovery
DBEnvironment Name: PartsDBE
Maintenance Word: 
Recover to: (mm/dd/yy/hh/mm/ss) (opt): 
New Log Mode (Single/Dual) (opt): single
Enter New Log File Name(s) Separated by a Blank: newlg1
New Log File Size: 250
Setup Recovery (y/n)? y
```

```
Recovery Has Been Set Up.
Next Log Sequence Number is      1.
```

```
>> restorelog
DBEnvironment Name: PartsDBE
Maintenance Word: 
Input Device: TAPE
Local (y/n) (opt): y
Restore the Log File (y/n)? y
Log File 'lgn1' was Restored.
Rename 'lgn1' Log File To: newlog
Log File 'lgn1' was Renamed to 'newlog'.
```

```
>> recoverlog
DBEnvironment Name: PartsDBE
Maintenance Word: 
Next Log File To Recover: newlog
Recover Log (y/n)? y

Log File Recovered.
Next Possible Log Sequence Number is      2.
```

```
>> endrecovery
DBEnvironment Name: PartsDBE
Maintenance Word: 
End Recovery (y/n)? y

Recovery Has Terminated.
```

EXIT

Terminates SQLUtil execution.

Scope

SQLUtil Only

SQLUtil Syntax

```
>> E[XIT]
```

Description

- This command terminates the SQLUtil session and returns the user to the point from which SQLUtil was invoked.
- This command can be abbreviated to E.

Example

```
>> exit
```

HELP

Displays and describes all SQLUtil commands.

Scope

SQLUtil Only

SQLUtil Syntax

```
>> HELP
Command Name (opt): CommandName
```

Parameters

CommandName is the name of an SQLUtil command.

Description

- If you do not specify a command name, the HELP command lists the names of all valid SQLUtil commands.
- If a command name is specified, the correct syntax and an explanation for that command is displayed.
- You must enter // to leave the HELP command.

Example

```
>> HELP
```

The following SQLUtil commands are available:

ADDLOG	ALTDDBE	ENDRECOVERY	EXIT	HELP
MOVEFILE	MOVELOG	PURGEALL	PURGEDBE	PURGEFILE
PURGELOG	QUIT	RECOVERLOG	RESCUELOG	RESTORE
RESTORELOG	SET	SETDBEMAJNT	SETUPRECOVERY	SHOWACCESS
SHOWDBE	SHOWLOG	SHOWSET	STORE	STORELOG
STOREONLINE	ATTACHFILE	DETACHFILE	STOREINFO	CHANGELOG

For more information on any one of the functions, enter the command name at the prompt. Carriage return <cr> displays a brief description of each command. Type // to leave SQLUtil HELP.

Command Name (opt):

HELP

Command Summary:

ADDLOG	Adds a new log file to a DBEnvironment.
ALTDBE	Changes the startup parameters for a DBEnvironment.
ENDRECOVERY	Terminates rollforward recovery procedure.
EXIT	Terminates SQLUtil execution.
HELP	Displays and describes all SQLUtil commands.
MOVEFILE	Moves any DBEFile file across devices.
MOVELOG	Moves a log file across devices.
PURGEALL	Purges the DBEnvironment and all associated DBEFiles including the DBELog files.
PURGEDBE	Purges an existing DBEnvironment and all associated DBEFiles except the DBELog files.
PURGEFILE	Purges any DBEFile.
PURGELOG	Purges a log file from a DBEnvironment.
QUIT	Terminates SQLUtil execution.
RECOVERLOG	Recovers a log file in rollforward recovery.
RESCUELOG	Copies a log file to tape(s) or serial disk(s) when a database can not be restarted.
RESTORE	Copies a DBEnvironment to disk from tape(s) or serial disk(s) created by using the SQLUtil STORE command.
RESTORELOG	Copies a log file to disk from tape(s) or serial disk(s) created by using the SQLUTIL STORELOG command.
SET	Sets a parameter for a command option. Current parameters supported are ECHO_ALL, EXIT_ON_DBERR, and BACKUP.
SETDBEMAINT	Sets or alters the maintenance word of a DBEnvironment.
SETUPRECOVERY	Sets up a DBEnvironment to perform a rollforward recovery. This is the first step of rollforward recovery.
SHOWACCESS	Shows the access mode (mapped or pseudo-mapped) of DBEFiles.
SHOWDBE	Shows the information related to a DBEnvironment.
SHOWLOG	Shows the information related to log(s) files associated with a DBEnvironment.
SHOWSET	Displays the value of a command option parameter. Current parameters supported are ECHO_ALL, EXIT_ON_DBERR, and BACKUP.
STORE	Copies a DBEnvironment to tape(s).
STORELOG	Copies a single log file to tape(s).
STOREONLINE	Copies a DBEnvironment to tape(s) ONLINE.
ATTACHFILE	Attaches DBEfilesset/DBEfile name with DBEnvironment.
DETACHFILE	Detaches DBEfilesset/DBEfile name with DBEnvironment.
STOREINFO	Displays files stored on a backup device.
CHANGELOG	Causes the DBEnvironment to change to the next log file.

Command Name (opt): movefile

```
>> MOVEFILE
DBEnvironment Name: DBEnvironmentName
File Name: FileName
Current Device: CurrentDevice
New Device: [NewDevClass/Number]
Access Mode (mapped) (opt): [FileAccessMode]
```

Moves DBEFiles, or DBECon files across devices.

Command Name (opt): //

>>

MOVEFILE

Moves the DBECon file or any DBEFile across devices.

Scope

SQLUtil Only

SQLUtil Syntax

```
>> MOVEFILE
DBEnvironment Name: DBEnvironmentName
Current File Name: CurrentFileName
New File Name: NewFileName [.NewGroupName]
Current Device: CurrentDeviceNumber
New Device (opt): NewDeviceNumber
Access Mode (Mapped) (opt): FileAccessMode
```

Parameters

<i>DBEnvironmentName</i>	is the name of the DBEnvironment with which the file you are changing is associated.
<i>CurrentFileName</i>	is the name of the file to be moved. Only DBEFiles can be moved with the MOVEFILE command. DBECon files cannot be moved with the MOVEFILE command.
<i>NewFileName</i> [<i>.NewGroupName</i>]	is the new name of the file to be moved. The file can be moved to a new group by specifying the optional <i>NewGroupName</i> preceded by a period (.).
<i>CurrentDeviceNumber</i>	is the device number of the device on which the file currently resides. This value is displayed by MOVEFILE.
<i>NewDeviceNumber</i>	is the device number of the device to which the file should be moved. Use the MPE/iX SHOWDEV command to display a list of available devices. If nothing is entered at the prompt, the file is not moved.
<i>FileAccessMode</i>	is MAPPED to specify that the DBEFile is accessed as an MPE/iX mapped file. The default is to leave the access mode of the file unchanged. Pseudo-mapped is no longer supported.

Description

- The MOVEFILE command first prompts for the current name of the file to be moved. It then prompts for the new name for the file and allows you to specify an optional group name if you wish to place the file in a new group. MOVEFILE then displays the current device number of the device on which the specified file resides, then it prompts for a new device number. The file is moved to the device with the number the user specifies. You can specify the old device number if you do not want to change devices. You can use the MPE/iX LISTF,3 command for the specified file to display both the current device class

MOVEFILE

name and the current device number. If you press **Return** instead of entering a new device number, the file is *not* moved.

- The MOVEFILE command can be used to change a file's group but not the account, which always remains the same as the account of the DBECon file.
- The MOVEFILE command can be used to change a DBEFile's access mode to mapped (the default mode is mapped). If you press **Return** in response to the prompt for access mode, the access mode is left unchanged.

Note Pseudo-mapped access is no longer supported. If any files are currently pseudo-mapped, use the MOVEFILE command to convert them to MAPPED.

- You can display the access modes for the DBEFiles in a DBEnvironment by using the SHOWACCESS command.
- To change the access mode without physically moving the DBEFile, press **Return** in response to the New Device prompt, then enter the access mode. Now you can convert the DBEFiles to Mapped mode. If they already in mapped mode, press **Return** to move the files.
- To retain the current access mode, press **Return** in response to the prompt for Access Mode.
- You cannot convert the DBECon file to a pseudo-mapped file.
- The MOVEFILE command can only be executed when the DBEnvironment is not in use.

Authorization

You must either be the DBECreator or have SM capability to use this command.

Example

```
>> movefile
DBEnvironment Name: PartsDBE.SomeGrp.SomeAcct
Current File Name: PARTIDX1
New File Name: PARTIDX1.OtherGroup
Current Device: DISC
New Device (opt): 2
Access Mode (Mapped)(opt): Return

File moved.
```

MOVELOG

Moves a log file from one location to another, and lets you specify a new name.

Scope

SQLUtil Only

SQLUtil Syntax

```
>> MOVELOG
DBEnvironment Name: DBEnvironmentName
Current Log File Name: CurrentFileName
Current Device: CurrentDeviceClass
New Device (opt): NewDeviceNumber
New Log File Name: NewLogFileName
```

Parameters

<i>DBEnvironmentName</i>	is the name of the DBEnvironment associated with the file you are changing.
<i>CurrentFileName</i>	is the name of the log file that is to be moved.
<i>CurrentDeviceNumber</i>	is the device number of the device on which the file currently resides. This value is displayed by MOVELOG.
<i>NewDeviceNumber</i>	is the device number of the device to which the log file should be moved. Use the MPE/iX SHOWDEV command to display a list of available devices.
<i>NewLogFileName</i>	is the new name for the log file. This must be in the same account as the DBECon file.

Description

- The MOVELOG command first displays the current device class of the device on which the log file resides, then it prompts for a new device number. The file is moved to the device with the number the user specifies. Use the MPE/iX LISTF,3 command for the specified log file to display both the current device class name and the current device number.
- The MOVELOG command can be used to change a log file's name and group, but it cannot be used to change a log file's account, which must be the same as the account of the DBECon file.
- The MOVELOG command can only be executed when the DBEnvironment is not in use.

Authorization

You must either be the DBECreator or have SM capability to use this command.

MOVELOG

Example

```
>> movelog
DBEnvironment Name: PartsDBE.SomeGrp.SomeAcct
Current Log File Name: lgn1
Current Device: DISC
New Device (opt): 2
New Log File Name: lgn2

File moved.
```

PURGEALL

Purges an existing DBEnvironment—including the DBECon file, DBEFiles, and all associated log files.

Scope

SQLUtil Only

SQLUtil Syntax

```
>> PURGEALL
DBEnvironment Name: DBEnvironmentName
Purge DBEnvironment and Log Files (y/n)?  $\left\{ \begin{array}{l} \text{Y[es]} \\ \text{N[o]} \end{array} \right\}$ 
```

Parameters

DBEnvironmentName is the name of the DBEnvironment to be purged.

Description

- The PURGEALL command purges an existing DBEnvironment. When a DBEnvironment is purged with this command, the DBECon file, the log files, and all DBEFiles related to the DBEnvironment are purged. In order to prevent a user from accidentally purging the DBEnvironment, a YES response to the confirmation prompt is required.
- If you previously purged either the file DBEFile0 or the log file(s), the DBEnvironment cannot be purged with the PURGEALL command. In such a case, you would need to use PURGEFILE to remove each file separately.
- PURGEALL will not purge log files which were in existence prior to the most recent START DBE NEWLOG statement. To remove these log files you must use the SQLUtil PURGEFILE command.
- This command can only be executed when the DBEnvironment is not in use.
- This command differs from the PURGEDBE command, which does not purge log files.

Authorization

You must be the DBECreator or have SM capability to use the PURGEALL command.

Example

```
>> PURGEALL
DBEnvironment Name: PartsDBE
Purge DBEnvironment and Log Files (y/n)? yes

DBEnvironment purged.
```

PURGEDBE

Purges an existing DBEnvironment—including the DBECon file and all associated DBEFiles except the log files.

Scope

SQLUtil Only

SQLUtil Syntax

```
>> PURGEDBE
DBEnvironment Name: DBEnvironmentName
Purge DBEnvironment (y/n)?  $\left\{ \begin{array}{l} \text{Y[es]} \\ \text{N[o]} \end{array} \right\}$ 
```

Parameters

DBEnvironmentName is the name of the DBEnvironment to be purged.

Description

- The PURGEDBE command purges an existing DBEnvironment. When a DBEnvironment is purged, the DBECon file and all DBEFiles related to the DBEnvironment are purged. The log files are not purged.

Note PURGEDBE does not purge the log files because the log files may be used for a subsequent rollforward recovery. The user must use the PURGEFILE command to purge the log file(s).

- In order to prevent a user from accidentally purging the DBEnvironment, a YES response to the confirmation prompt is required.
- This command can only be executed when the DBEnvironment is not in use.
- If you previously purged either the file DBEFile0 or the log file(s), the DBEnvironment cannot be purged with the PURGEDBE command.

Authorization

You must be the DBECreator or have SM capability to use this command.

Example

```
>> purgedbe
DBEnvironment Name: PartsDBE
Purge DBEnvironment (y/n)? yes

DBEnvironment purged.
```

PURGEFILE

Purges any DBEFile or DBECon file.

Scope

SQLUtil Only

SQLUtil Syntax

```
>> PURGEFILE
DBEFile Name: DBEFileName
Purge DBEFile (y/n)? { Y[es] }
                     { N[o] }
```

Parameters

DBEFileName is the physical file name of the DBEFile or DBECon file to be purged.

Description

- The PURGEFILE command can be used to purge any DBEFile or DBECon file. The file will not be purged if it is in use.
- PURGEFILE should be used after using START DBE NEWLOG to purge any log files that existed prior to issuing the START DBE NEWLOG statement.
- After completing the ENDRECOVERY step of partial rollforward recovery, PURGEFILE should be used to purge the group which contained the temporary DBEnvironment of all log files used for the partial rollforward recovery process.
- Do not use PURGEFILE to try to purge a log file following a STORELOG command. Be sure to use PURGELOG instead.

Warning

The DROP DBEFILE command automatically removes the physical file associated with the DBEFile when it executes. Therefore, you should not purge the physical file associated with a DBEFile before using the DROP DBEFILE statement. If you do, an entry for the physical file which you will not be able to remove will still remain in the system catalog even though the physical file no longer exists.

- If you purged either the file DBEFile0 or the log file(s), you will not be able to purge the DBEnvironment using PURGEDBE.

PURGEFILE

Authorization

You must be the DBECreator or have SM capability to execute this command.

Example

```
>> purgefile  
DBEFile Name: PartsXF1  
Purge DBEFile (y/n)? yes  
  
File purged.
```


PURGELOG

Purges a log file that is not needed.

Scope

SQLUtil Only

SQLUtil Syntax

```
>> PURGELOG
DBEnvironment Name: DBEnvironmentName
Maintenance Word: MaintenanceWord
Log Identifier: LogIdentifier
Purge Log File (y/n)? { Y[es] }
                     { N[o] }
```

Parameters

DBEnvironmentName is the name of the DBEnvironment with which the file you are changing is associated.

Maintenance Word is the maintenance word.

LogIdentifier is the log identifier number assigned to the file you wish to purge.

Description

- This command lets you remove a log file provided it does not contain any active transactions and has either been backed up or else never used. In the case of dual logging, both physical files corresponding to the identifier number are deleted.
- If you try to purge a log file that is either in use or that has not been backed up, you will receive an error message.
- The DBEnvironment may be in use while this command is executed.

Authorization

You must be the DBECreator or have SM capability to purge a log file.

Example

```
>> purgelog
DBEnvironment Name: PartsDBE
Maintenance Word: 
Log Identifier: 2
Purge Log File (y/n)? y
```

QUIT

Terminates SQLUtil execution.

Scope

SQLUtil Only

SQLUtil Syntax

```
>> Q[UIT]
```

Description

- This command terminates the SQLUtil session and returns you to the point from which SQLUtil was invoked.
- This command can be abbreviated to Q.

Example

```
>> quit
```

RECOVERLOG

Applies the transactions from a specific log file or multiple log files during rollforward recovery.

Scope

SQLUtil Only

SQLUtil Syntax

```
>> RECOVERLOG
DBEnvironment Name: DBEnvironmentName
Maintenance Word: MaintenanceWord
Next Log File to Recover: LogFileName
Recover Log File (y/n)? { Y[es] }
                       { N[o] }
```

Parameters

DBEnvironmentName is the name of the DBEnvironment with which the file(s) you are recovering is associated.

MaintenanceWord is the maintenance word.

LogFileName is the name of the file being recovered.

Description

- This command starts a DBEnvironment session, then lets you apply individual log files for rollforward recovery after a DBEnvironment and its log files have been restored. Rollforward recovery uses four commands:
 - SETUPRECOVERY or SETUPRECOVERY PARTIAL, which initiates the process.
 - RESTORELOG, which restores a log file from a backup.
 - RECOVERLOG, which you issue for each log file being applied to the DBEnvironment.
 - ENDRECOVERY, which ends the rollforward recovery process.
- You normally apply SETUPRECOVERY or SETUPRECOVERY PARTIAL once. For SETUPRECOVERY you can specify a *RecoverTime* as the end point for recovery. For SETUPRECOVERY PARTIAL you must roll forward to the point of the failure. You then use RESTORELOG to restore each backup log file to the system in sequence number order, and you use RECOVERLOG once for each log file that you have restored, to apply it to the DBEnvironment. ALLBASE/SQL will recover the transactions in the file up to the *RecoverTime* you specified, or up to the end of the file.
- You use ENDRECOVERY once after recovering all the log files you wish to apply to the DBEnvironment. Normally, you do *not* use ENDRECOVERY if you specified a *RecoverTime* in the SETUPRECOVERY command.
- At the completion of each RECOVERLOG command, the next possible log sequence number is displayed.

RECOVERLOG

- If you specified a *RecoverTime* in the SETUPRECOVERY command, the following message appears when you reach the specified time stamp:

```
Time Stamp Reached.
```

When you see this message, recovery is complete. In this case, you do *not* use the ENDRECOVERY command.

- ENDRECOVERY will result in an error if you have not recovered enough files to make the DBEnvironment consistent. If you used STOREONLINE to do an online backup of the DBEnvironment, you must recover starting from the *First Log Sequence Number Needed for Recovery* up to and including the log file that was active at the time the last STOREONLINE command completed. If you did a static backup of the DBEnvironment, you can recover as little or as much of the log as you desire once you have restored the most recent copy of the DBEnvironment.

Authorization

To use this command, you must be the DBECreator or supply the correct maintenance word.

Example

```
>> setuprecovery
DBEnvironment Name: PartsDBE
Maintenance Word: 
Recover to: (mm/dd/yy/hh/mm/ss) (opt): 
New Log Mode (Single/Dual) (opt): single
Enter New Log File Name(s) Separated by a Blank: newlg1
New Log File Size: 250
Setup Recovery (y/n)? y

Recovery Has Been Set Up.
Next Log Sequence Number is      1.

>> restorelog
DBEnvironment Name: PartsDBE
Maintenance Word: 
Input Device: TAPE
Local (y/n) (opt): y
Restore the Log File (y/n)? y
Log File 'lgn1' was Restored.
Rename 'lgn1' Log File To: newlog
Log File 'lgn1' was Renamed to 'newlog'.

>> recoverlog
DBEnvironment Name: PartsDBE
Maintenance Word: 
Next Log File To Recover: newlog
Recover Log File (y/n)? y

Log File Recovered.
Next Possible Log Sequence Number is      2.

>> endrecovery
DBEnvironment Name: PartsDBE
Maintenance Word: 
End Recovery (y/n)? y

Recovery Has Terminated.
```

RESCUELOG

Stores a copy of a log file without accessing the DBECon file. You use this command to store a log file that cannot be stored using STORELOG.

Scope

SQLUtil Only

SQLUtil Syntax

```
>> RESCUELOG
Log File Name: LogFile
Size Of The Log File: LogFileSize
To File Name? DeviceName
Rescue Log File(y/n)? { Y[es] }
                       { N[o] }
```

Parameters

<i>LogFile</i>	is the name of the log file that is to be rescued (that is, stored).
<i>LogFileSize</i>	is the size in 512-byte pages of the log file that is to be rescued.
<i>DeviceName</i>	is the device name for the tape or disk device that will receive the store file.

Description

- This command lets you store a log file. It is normally used only with files that cannot be stored with STORELOG.
- Normally, this command would only be used following the corruption of DBEnvironment files through a media failure or some other serious problem. If you are unable to connect to the DBEnvironment, use a static SHOWLOG to display log files that have not been stored, then use RESCUELOG to store them.
- If more than one tape volume is required to store the file, a request will be displayed on the console. The next tape can then be mounted.

Authorization

Any user may execute this command.

Example

```
>> rescuelog
Log File Name: DBELOG6
Size Of The Log File: 250
To File Name? TAPE
Rescue Log File(y/n)? y

Log file 'DBELOG6' with Sequence Number          12 was rescued.
```

RESTORE

Copies an entire DBEnvironment to disk from tape files created by using the SQLUtil STORE or STOREONLINE command.

Scope

SQLUtil Only

SQLUtil Syntax

```
>> RESTORE
DBEnvironment Name: DBEnvironmentName
From File Name: StoreFileName
Restore DBEnvironment (y/n)?  $\left\{ \begin{array}{l} \text{Y}[\text{es}] \\ \text{N}[\text{o}] \end{array} \right\}$ 
```

Parameters

DBEnvironmentName is the name of the DBEnvironment to be restored.

StoreFileName is the formal file designator of the tape device or disk.

Description

- This command allows you to restore the DBECon file and all DBEFiles of a DBEnvironment that was saved with the SQLUtil STOREONLINE or STORE command.
- When using RESTORE PARTIAL you must be in the same group and account from which the DBEFiles were stored. This is usually the group and account containing the DBECon file and the SYSTEM DBEFileSet.
- The DBEnvironment is restored to its original group and account, and the DBECreator remains the same as when the DBEnvironment was originally created. If there is not an appropriate user, group, or account on the target system, an error will result.
- If the user is not logged on to the target group and account, an error will occur.
- If a DBEnvironment with the same name already exists in the same group and account, the user must use the PURGEDBE command to purge it before RESTORE will successfully execute.

Authorization

Anyone can restore a DBEnvironment.

Example

```
>> restore
DBEnvironment Name: PartsDBE.SomeGrp.SomeAcct
From File Name: TAPE
Restore DBEnvironment (y/n)? y

DBEnvironment restored.
```

RESTORELOG

Restores a log file that had been previously backed up with the STORELOG command.

Scope

SQLUtil Only

SQLUtil Syntax

```
>> RESTORELOG
DBEnvironment Name: DBEnvironmentName
Maintenance Word: MaintenanceWord
Input Device: DeviceName
Local (y/n) (opt): { Y[es] }
                  { N[o] }
Restore the Log File (y/n)? { Y[es] }
                           { N[o] }
Rename LogFileName Log File To: [NewName[.NewGroup[.NewAccount]]]
```

Parameters

DBEnvironmentName is the name of the DBEnvironment to which the log files are being restored.

MaintenanceWord is the maintenance word.

DeviceName is the name of the tape or other backup device from which you are restoring the log file.

Local specifies whether the log is to be restored to the local group and account or to the group and account from which it was originally stored. Reply Y for the local group and account.

NewName [*NewGroup*. [*NewAccount*]] is a new name you are specifying for a log file after you restore it. The default is the same as the file name specified. If you include the optional *NewAccount*, it must be the same as the account name of the DBECon file.

It is easier to restore log files in proper order if you specify a new name which contains the log sequence number of the log file as part of the new file name.

Description

- This command lets you restore one or more previously stored archive log files. If only one log file was stored with a given invocation of STORELOG, one log file will be restored. If multiple log files were stored with a single invocation of STORELOG, all log files stored with that invocation will be restored.
- ALLBASE/SQL prompts for a new file name for each file before restoring the file. To avoid name conflicts, rename using distinct file names that reflect the log sequence numbers.
- Rollforward recovery uses four commands:

RESTORELOG

- SETUPRECOVERY or SETUPRECOVERY PARTIAL—initiates the process.
 - RESTORELOG—restores one or more log files from a backup.
 - RECOVERLOG—issued by you for each log file being applied to the DBEnvironment.
 - ENDRECOVERY—ends the rollforward recovery process.
- You normally apply SETUPRECOVERY or SETUPRECOVERY PARTIAL once. When using SETUPRECOVERY you can specify a *RecoverTime* as the end point for recovery. For SETUPRECOVERY PARTIAL you must recover all the way up to the time of the failure. You then use RESTORELOG to restore each backup log file to the system in sequence number order, and you use RECOVERLOG once for each log file that you have restored, to apply it to the DBEnvironment. ALLBASE/SQL will recover the transactions in the file up to the *RecoverTime* you specified, or up to the end of the file.
 - Use ENDRECOVERY once after recovering all the log files you wish to apply to the DBEnvironment. Normally, you do *not* use ENDRECOVERY if you specified a *RecoverTime* in the SETUPRECOVERY command.
 - ENDRECOVERY will result in an error if you have not recovered enough files to make the DBEnvironment consistent. If you used STOREONLINE to do an online backup of the DBEnvironment, you must recover starting from the *First Log Sequence Number Needed for Recovery* up to and including the file that was active at the time the last STOREONLINE command completed. If you did a static backup of the DBEnvironment, you can recover as little or as much of the log as you desire once you have restored the most recent copy of the DBEnvironment.
 - Normally, this command is used when recovering a DBEnvironment, in which case the DBEnvironment is *not* in use while the command is executed.

Authorization

Any user may execute this command.

Examples

Recovery of singly stored log file

```
>> setuprecovery
DBEnvironment Name: PartsDBE
Maintenance Word: Return
Recover to: (mm/dd/yy/hh/mm/ss) (opt): Return
New Log Mode (Single/Dual) (opt): single
Enter New Log File Name(s) Separated by a Blank: newlg1
New Log File Size: 250
Setup Recovery (y/n)? y
```

```
Recovery Has Been Set Up.
Next Log Sequence Number is      1.
```

```
>> restorelog
DBEnvironment Name: PartsDBE
Maintenance Word: Return
Input Device: TAPE
Local (y/n) (opt): y
Restore the Log File (y/n)? y
Log File 'lgn1' was Restored.
Rename 'lgn1' Log File To: lsn001
```


Log File 'lgn1' was Renamed to 'lsn001'.

```
>> recoverlog
DBEnvironment Name: PartsDBE
Maintenance Word: 
Next Log File To Recover: lsn001
Recover Log File (y/n)? y

Log File Recovered.
Next Possible Log Sequence Number is      2.
```

```
>> endrecovery
DBEnvironment Name: PartsDBE
Maintenance Word: 
End Recovery (y/n)? y
```

Recovery Has Terminated.

Recovery of multiple log files stored with single invocation of STORELOG

```
>> setuprecovery
DBEnvironment Name: PartsDBE
Maintenance Word: 
Recover to: (mm/dd/yy/hh/mm/ss) (opt): 
New Log Mode (Single/Dual) (opt): single
Enter New Log File Name(s) Separated by a Blank: newlg1
New Log File Size: 250
Setup Recovery (y/n)? y
```

Recovery Has Been Set Up.
Next Log Sequence Number is 1.

```
>> restorelog
DBEnvironment Name: PartsDBE
Maintenance Word: 
Input Device: TAPE
Local (y/n) (opt): y
Rename 'lgn1' Log File To: lsn001
Rename 'lgn2' Log File To: lsn002
Restore the Log File (y/n)? y
Log File 'lsn001' was Restored.
Log File 'lsn002' was Restored.
```

```
>> recoverlog
DBEnvironment Name: PartsDBE
Maintenance Word: 
Next Log File To Recover: lsn001
Recover Log File (y/n)? y

Log File Recovered.
Next Possible Log Sequence Number is      2.
```

```
>> recoverlog
DBEnvironment Name: PartsDBE
Maintenance Word: 
Next Log File To Recover: lsn002
Recover Log File (y/n)? y
```

RESTORELOG

Log File Recovered.
Next Possible Log Sequence Number is 3.

>> endrecovery
DBEnvironment Name: PartsDBE
Maintenance Word:
End Recovery (y/n)? y

Recovery Has Terminated.

RESTORE PARTIAL

Copies a subset of a DBEnvironment to disk from tape files that were created using the SQLUtil STORE, STOREONLINE, STORE PARTIAL, or STOREONLINE PARTIAL commands.

Scope

SQLUtil Only

SQLUtil Syntax - RESTORE PARTIAL

```
>> RESTORE PARTIAL
DBEnvironment Name: DBEnvironmentName
From File Name: StoreFileName
Enter File to Restore or Carriage Return to Finish: SystemFileName
Enter File to Restore or Carriage Return to Finish:
```

Parameters

DBEnvironmentName is the name of the DBEnvironment to be restored.

StoreFileName is the name of the file or tape device where the stored DBEnvironment files reside.

SystemFileName is the physical name of the DBEFile to be restored. You may enter a list of file names, one name per line, with a null line to terminate the list. Use the STOREINFO command to get the names of the files stored.

Description

- This command allows you to restore all, or a subset of, the DBEFiles previously stored with the SQLUtil STORE, STOREONLINE, STORE PARTIAL or STOREONLINE PARTIAL command.
- DBEFiles must be restored individually, on a file by file basis; you cannot specify a DBEFileSET name to be restored.
- If you are not logged on to the target group and account, an error will occur.
- When using RESTORE PARTIAL, if a DBEFile with the same name already exists in the target group and account, use the PURGEFILE command to purge it before restoring it.

When using RESTORE PARTIAL you must be in the same group and account from which the DBEFiles were stored. This is usually the group and account containing the DBECon file and the SYSTEM DBEFileSet.

RESTORE PARTIAL

Authorization

Anyone can execute the RESTORE PARTIAL command.

Example

```
>> restore partial
```

```
DBEnvironmentName: PartsDBE.SomeGrp.SomeAcct
```

```
From File Name: TAPE
```

```
Enter File to Restore or Carriage Return to Finish: OrderDF1
```

```
Enter File to Restore or Carriage Return to Finish: OrderXF1
```

```
Enter File to Restore or Carriage Return to Finish:
```

```
DBEnvironment restored.
```

SET

The SET command lets you specify the following operating parameters for SQLUtil: BACKUP, EXIT_ON_DBERR, and ECHO_ALL.

Scope

SQLUtil only.

SQLUtil Syntax

$$\gg \text{ SET } \left[\begin{array}{l} \text{BACKUP } \left\{ \begin{array}{l} \text{STORESET} \\ \text{RESTORESET} \\ \text{MOSET} \end{array} \right\} \left[\begin{array}{l} \text{'OptionString'} \\ \text{DEFAULT} \end{array} \right] \\ \\ \text{ECHO_ALL } \left\{ \begin{array}{l} \text{ON} \\ \text{OFF} \end{array} \right\} \\ \\ \text{EXIT_ON_DBERR } \left\{ \begin{array}{l} \text{ON} \\ \text{OFF} \end{array} \right\} \end{array} \right]$$

Parameters

STORESET	Together with <i>'OptionString'</i> , specifies a set of devices for use in storing DBEnvironment files and logs.
RESTORESET	Together with <i>'OptionString'</i> , specifies a set of devices for use in restoring DBEnvironment files and logs.
MOSET	Together with <i>'OptionString'</i> , specifies a set of DASS devices for use in restoring DBEnvironment files and logs.
ECHO_ALL	Causes SQLUtil to echo all input. The initial default setting is OFF.
EXIT_ON_DBERR	Causes SQLUtil to terminate immediately when a DBERR is encountered. The initial default setting is OFF.

Description

- Choosing a BACKUP parameter lets you select TurboSTORE II backup options if you are using TurboSTORE II with your operating system.
- The STORESET, RESTORESET, and MOSET options are only available when you have installed TurboSTORE II software on your system.
- In the STORESET option, you specify multiple drives for use in backup operations. You can specify either sequential or parallel operation. The specification of a STORESET is shown in the *TurboSTORE II User's Guide*.
- In the RESTORESET option, you specify multiple drives for use in restore operations. You can specify a parallel RESTORESET, but only for tapes that were created using a parallel

SET

STORESET. The specification of a RESTORESET is shown in the *TurboSTORE II User's Guide*.

- The MOSET option is for use only with direct access secondary storage (DASS) devices using optical disk drives. The MOSET is for use in both STORE and RESTORE operations. The specification of an MOSET is shown in the *TurboSTORE II User's Guide*.
- If you specify a parallel STORE operation, the division of files to be backed up is left to TurboSTORE II. You can use a parallel RESTORE, but you must specify the same sequence of devices in the RESTORESET that you specified at STORE time in the STORESET.
- The SET BACKUP parameters STORESET and RESTORESET cannot be set if the MOSET parameter has already been set. Similarly, the MOSET parameter cannot be set if either STORESET or RESTORESET has been set.
- If no option is specified for STORESET, RESTORESET, or MOSET, then the previous option specified for that parameter will be reset.
- Entering the keyword DEFAULT as an option has the same effect as specifying no option.
- When the ECHO_ALL command option is ON, batch user input is echoed to the standard list. The initial default setting is OFF.
- Setting the EXIT_ON_DBERR to ON causes immediate termination of SQLUtil when an ALLBASE/SQL error is encountered.
- If EXIT_ON_DBERR is ON, the system JCW is set to FATAL if a DBERR is encountered or to 0 if no DBERRs are encountered.
- See the “DBA Tasks and Tools” chapter of this manual for information on checking JCWs when SQLUtil terminates.
- Issuing a SET command without parameters displays the current settings, exactly as in the SHOWSET command.

Authorization

Anyone can use this command.

Example

```
>> set echo_all on
>> set exit_on_dberr on
set exit_on_dberr on
```

The following sets a STORESET for sequential operation.

```
>> set backup storeset '(*TAPE1, *TAPE2, *TAPE3)'
```

The following sets a RESTORESET for parallel operation. (The previous STORE was done using the same list of devices.)

```
>> set backup restoreset '(*TAPE1),(*TAPE2),(*TAPE3)'
```

The following displays current settings.

```
>> set
      ECHO_ALL           :  ON
      EXIT_ON_DBERR     :  ON
BACKUP:
      STORESET          :  (*TAPE1, *TAPE2, *TAPE3)
      RESTORESET        :  (*TAPE1),(*TAPE2),(*TAPE3)
      MOSET             :
>>
```

SETDBEMAIN

Sets or alters the maintenance word of a DBEnvironment.

Scope

SQLUtil Only

SQLUtil Syntax

```
>> SETDBEMAIN
DBEnvironment Name: DBEnvironmentName
Current Maintenance Word: OldMaintenanceWord
New Maintenance Word: NewMaintenanceWord
Retype New Maintenance Word: NewMaintenanceWord
```

Parameters

DBEnvironmentName is the name of a DBEnvironment.

OldMaintenanceWord is the old maintenance word. This is written to the terminal screen.

NewMaintenanceWord is the new maintenance word. What you type is not echoed to the terminal screen. You are prompted to enter the new maintenance word a second time for verification.

Description

- This command allows you to set or change the DBEnvironment maintenance word. The current maintenance word will be displayed after the DBEnvironment name has been given. You are asked to enter the new maintenance word twice to ensure it is entered correctly.
- The DBEnvironment cannot be in use while this command is executed.

Authorization

You must be the DBECreator or have SM capability to use this command.

Example

```
>> setdbemaint
DBEnvironment Name: PartsDBE
Current Maintenance Word: OldMaintWd
New Maintenance Word: NewMaintWd
Retype New Maintenance Word: NewMaintWd

Maintenance word changed.
```


SETUPRECOVERY

Initiates a process of full rollforward recovery.

Scope

SQLUtil Only

SQLUtil Syntax

```
>> SETUPRECOVERY
DBEnvironment Name: DBEnvironmentName
Maintenance Word: MaintenanceWord
Recover To (mm/dd/y/hh/mm/ss) (opt): RecoverTime
New Log Mode (Single/Dual) (opt): NewLogMode
Enter New Log File Name(s) Separated By A Blank: NewLogFile1 [NewLogFile2]
New Log File Size (opt): NewLogSize
Setup Recovery (y/n)? { Y[es] }
                      { N[o] }
```

Parameters

<i>DBEnvironmentName</i>	is the name of the DBEnvironment you are recovering.
<i>MaintenanceWord</i>	is the maintenance word.
<i>RecoverTime</i>	is the time up to which you want to recover the DBEnvironment.
<i>NewLogMode</i>	is either Single or Dual. This applies to the new log, not to the one from which recovery is carried out.
<i>NewLogFile1</i> and <i>NewLogFile2</i>	are the names you are specifying for the new log. Use both names for dual logging. If you specify the optional account name, it must be the same as the account of the DBECon file. Files are created in the same group as the DBECon file unless you specify a different group within the DBECon file's account.
<i>NewLogSize</i>	is the size of the new log in 512-byte pages.

Description

- This command starts a DBEnvironment session and lets you begin the process of rollforward recovery after a DBEnvironment and its log files have been restored.
- Rollforward recovery uses four commands:
 - SETUPRECOVERY or SETUPRECOVERY PARTIAL—initiates the process.
 - RESTORELOG—restores one or more log files from a backup.
 - RECOVERLOG—issued by you for each log file being applied to the DBEnvironment.
 - ENDRECOVERY—ends the rollforward recovery process.
- SETUPRECOVERY should be done from the group and account from which the DBEnvironment was originally stored (usually the group and account containing the DBECon file and SYSTEM DBEFileSet).

SETUPRECOVERY

- You normally apply SETUPRECOVERY or SETUPRECOVERY PARTIAL once. For SETUPRECOVERY you can specify a *RecoverTime* as the end point for recovery. For SETUPRECOVERY PARTIAL you must recover all the way forward to the point of failure. You then use RESTORELOG to restore each backup log file to the system in sequence number order, and you use RECOVERLOG once for each log file that you have restored, to apply it to the DBEnvironment. ALLBASE/SQL will recover the transactions in the file up to the *RecoverTime* you specified or up to the end of the file.
- You use ENDRECOVERY once after recovering all the log files you wish to apply to the DBEnvironment. Normally, you do *not* use ENDRECOVERY if you specified a *RecoverTime* in the SETUPRECOVERY command.
- ENDRECOVERY will result in an error if you have not recovered enough files to make the DBEnvironment consistent. If you used STOREONLINE to do an online backup of the DBEnvironment, you must recover starting from the *First Log Sequence Number Needed for Recovery* up to and including the file that was active at the time the last STOREONLINE command completed. If you did a static backup of the DBEnvironment, you can recover as little or as much of the log as you desire once you have restored the most recent copy of the DBEnvironment.

Authorization

You must be the DBECreator or supply the correct maintenance word to use this command.

Example

```
>> setuprecovery
DBEnvironment Name: PartsDBE
Maintenance Word: 
Recover to: (mm/dd/yy/hh/mm/ss) (opt): 
New Log Mode (Single/Dual) (opt): single
Enter New Log File Name(s) Separated by a Blank: newlg1
New Log File Size: 250
Setup Recovery (y/n)? y
```

```
Recovery Has Been Set Up.
Next Log Sequence Number is      1.
```

```
>> restorelog
DBEnvironment Name: PartsDBE
Maintenance Word: 
Input Device: TAPE
Local (y/n) (opt): y
Restore the Log File (y/n)? y
Log File 'lgn1' was Restored.
Rename 'lgn1' Log File To: newlog
Log File 'lgn1' was Renamed to 'newlog'.
```

```
>> recoverlog
DBEnvironment Name: PartsDBE
Maintenance Word: 
Next Log File To Recover: newlog
Recover Log File (y/n)? y

Log File Recovered.
Next Possible Log Sequence Number is      2.
```

SETUPRECOVERY

```
>> endrecovery  
DBEnvironment Name: PartsDBE  
Maintenance Word: Return  
End Recovery (y/n)? y
```

Recovery Has Terminated.

SETUPRECOVERY PARTIAL

This command initiates a process of partial rollforward recovery. When SETUPRECOVERY PARTIAL is used, a temporary DBEnvironment is created for the rollforward recovery process.

Scope

SQLUtil Only

SQLUtil Syntax - SETUPRECOVERY PARTIAL

```
>> SETUPRECOVERY PARTIAL
New DBEnvironment Name: TempDBEnvironmentName
Maintenance Word: Maintenance Word
Enter DBFileName or Carriage Return to Finish: SystemFileName
Enter DBFileName or Carriage Return to Finish:
Setup Recovery (y/n)? { y[es] }
                     { n[o] }
```

Parameters

<i>TempDBEnvironmentName</i>	is the name of the temporary DBEnvironment you are using to perform a partial rollforward recovery.
<i>Maintenance Word</i>	is the maintenance word for the temporary DBEnvironment.
<i>SystemFileName</i>	is the physical name of the DBEFile to be recovered. You will be prompted for a list of file names one name per line. Enter a blank line with a carriage return to terminate the list.

Description

- The SETUPRECOVERY PARTIAL command starts a DBEnvironment session and lets you begin the process of partial rollforward recovery on an existing DBEnvironment. This allows you to do rollforward recovery on a subset of the DBEnvironment while the intact portion of the DBEnvironment remains in use.
- DBEFiles must be specified individually on a file by file basis; you cannot specify a DBEFileSet name to be restored.
- Partial rollforward recovery uses four commands:
 - SETUPRECOVERY PARTIAL—initiates the process.
 - RESTORELOG—restores one or more log files from a backup.
 - RECOVERLOG—issued by you for each log file being applied to the DBEnvironment.
 - ENDRECOVERY—ends the rollforward recovery process.
- You normally apply SETUPRECOVERY PARTIAL once. When using SETUPRECOVERY PARTIAL you must roll forward all the way to the point of the failure. You then use RESTORELOG to restore each backup log file to the system in sequence number order, and you use RECOVERLOG once for each log file that you have restored, to apply it to the DBEnvironment. ALLBASE/SQL will recover the transactions in the file up to the end of the file.

- You use ENDRECOVERY once after recovering all the log files you wish to apply to the DBEnvironment.
- ENDRECOVERY will result in an error if you have not recovered enough files to make the DBEnvironment consistent. If you did a static backup of the DBEnvironment, you must still recover up to the moment of the failure if doing a partial rollforward recovery. You must be sure to recover all files that will be acted upon by the rollforward process in order to maintain a consistent database. (This includes files that were undamaged by the failure, but will still be acted upon by recovery. All files touched by recovered transactions must be recovered from the last static backup.)
- When performing SETUPRECOVERY PARTIAL you should be in a group and account which is different from the group and account which contains the existing DBECon file and SYSTEM DBEFileSet because a new, temporary DBECon file and SYSTEM DBEFileSet are created for the partial rollforward recovery process. This also isolates the log files used for the partial recovery process so there can be no accidental interaction between the active log files and those used for the recovery process.
- When you are performing a SETUPRECOVERY PARTIAL, the DBEnvironmentName must be different from the original DBEnvironmentName.
- If a maintenance word is specified in a SETUPRECOVERY PARTIAL, it is stored in the temporary DBEnvironment. Later the maintenance word can be used to execute the RECOVERLOG and the ENDRECOVERY commands.
- The temporary DBEnvironment created for a partial rollforward process is purged when the recovery is completed as are the temporary DBECon file and temporary SYSTEM DBEFiles. However, the log files applied during the partial recovery process must be explicitly removed using the SQLUtil PURGEFILE command after the ENDRECOVERY command has been executed.

Authorization

You must be the DBECreator or supply the correct maintenance word to use this command.

Example of Partial Recovery

From the group and account containing the DBECon file and the SYSTEM DBEFileSet, use the SQLUtil DETACHFILE command to detach all DBEFiles that will be acted upon by the partial roll forward recovery process.

Move to a new group in the same account, which does not contain any DBECon file or SYSTEM DBEFileSet, to carry out the remainder of the partial rollforward recovery process.

```
>> setuprecovery partial
DBEnvironment Name: tmpdbe
Maintenance Word: Maintenance Word
Enter File to Restore or Carriage Return to Finish: OrderDF1.OtherGroup
Enter File to Restore or Carriage Return to Finish: OrderXF1.OtherGroup
Enter File to Restore or Carriage Return to Finish:
Setup Recovery (y/n): y

Recovery Has Been Setup.
Next Log Sequence Number is 2.
```

SETUPRECOVERY PARTIAL

```
>> restorelog
DBEnvironment Name: tmpdbe
Maintenance Word: MaintenanceWord
Input Device: TAPE
Local (y/n) (opt): y
Rename ptslogA1 Log File To: lsn002
Restore the Log File (y/n)? y
```

Log File 'lsn002' was Restored.

```
>> recoverlog
DBEnvironment Name: tmpdbe
Maintenance Word: MaintenanceWord
Next Log File To Recover: lsn002
Recover Log File (y/n)? y
```

Log File Recovered.

Next Possible Log Sequence Number is 3.

Repeat the restorelog/recoverlog sequence until all log files have been applied up to the time of the failure.

```
>> endrecovery
DBEnvironment Name: tmpdbe
Maintenance Word: MaintenanceWord
End Recovery (y/n)? y
```

Recovery Has Terminated.

Remove the log files that remain after the partial recovery process from the group which contained the temporary DBEnvironment. Return to the group which contains the original DBEnvironment and use the SQLUtil ATTACHFILE command to attach all DBEFiles that were detached prior to the partial rollforward recovery process.

SHOWACCESS

The SHOWACCESS command displays the file access mode for all the DBEFiles in a DBEnvironment.

Scope

SQLUtil Only

SQLUtil Syntax

```
>> SHOWACCESS
DBEnvironment Name: DBEnvironmentName
Maintenance Word: MaintenanceWord
```

Parameters

DBEnvironmentName is the name of a DBEnvironment containing DBEFiles whose access mode you wish to display.

MaintenanceWord is the maintenance word.

Description

- Use this command to display the access mode of the DBEFiles in a DBEnvironment with the MOVEFILE command.

Note Pseudo-mapped files are no longer supported. If any files are currently pseudo-mapped, use the SQLUtil MOVEFILE command to convert them to mapped files.

Authorization

Any user may execute this command.

Example

```
>> SHOWACCESS
DBEnvironment Name: PARTSDBE.SOMEGRP.SOMEACCT
Maintenance Word: 

PARTSFO      mapped
PURCHDF1     mapped
PURCHXF1     mapped
WAREHDF1     mapped
WAREHXF1     mapped
ORDERDF1     mapped
ORDERXF1     mapped
RECDF1       mapped
```

SHOWDBE

Shows the information related to a DBEnvironment.

Scope

SQLUtil Only

SQLUtil Syntax

```
>> SHOWDBE
DBEnvironment Name: DBEnvironmentName
Maintenance Word: MaintenanceWord
Output File Name (opt): OutputFileName
```

Parameters

DBEnvironmentName is the name of a DBEnvironment.

Maintenance Word is the maintenance word.

OutputFileName is the name of the output file to which the output from this command should be directed. The default is the terminal.

Description

- This command shows information about the specified DBEnvironment.
- Once this command is executed, the user will be prompted for subsequent SHOWDBE commands.
- The DBEnvironment may be in use while this command is executed.

Authorization

Any user may execute this command. However, the DBEnvironment maintenance word will not be displayed unless the user is the DBECreator, has SM capability, or supplies the correct maintenance word.

Example

```
>> showdbe
DBEnvironment Name: PartsDBE.SomeGrp.SomeAcct
Maintenance Word: 
Output File Name (opt): 
-> startparms

DBEnvironment Language: NATIVE-3000
AutoStart: ON
Audit Logging Is: ON
Audit Logging Name is: Wrapper1
Default Partition ID is: 1
Maximum Number of Partitions Is: 10
Comment Partition ID Is: 2
Audit Elements Are: CHKPT, DATA, CMNT
User Mode: MULTI
DBEFile0 Name: PartsF0
DDL Enabled: YES
No. of Runtime Control Block Pages: 128
No. of Data Buffer Pages: 200
Data Buffer Pages Memory Resident: NO
No. of Log Buffer Pages: 200
Max. Transactions: 100
Maximum Timeout: NONE
Default Timeout: MAXIMUM
Authorize Once per session: OFF

-> quit
```

SHOWDBE-ALL

Shows information about the DBEnvironment.

Scope

SQLUtil SHOWDBE Command

Syntax

-> ALL

Description

- This command shows information contained in the DBECon file.

Authorization

Any user may execute this command. However, the DBEnvironment maintenance word will not be displayed unless the user is the DBECreator, has SM capability, or supplies the correct maintenance word.

Example

The values specified in START DBE NEW and START DBE NEWLOG statements for audit databases are stored in the DBECon file. The SQLUtil SHOWDBE command can be used to verify the parameters of a DBEnvironment. For example, when you enter the SHOWDBE command after creation of a DBEnvironment, you might see the following:

```
isql=> sqlutil

>> showdbe
DBEnvironment Name: DBE1.SOMEGRP.SOMEACCT
Maintenance Word: (Return)
Output File Name (opt): (Return)
->
-> all

Maintenance word:
DBEnvironment Language: (NATIVE-3000)
AutoStart: ON
Audit Logging Is: ON
Audit Logging ID is: MYDBE1
Default Partition ID Is: 1
Maximum number of Partitions Is: 10
User Mode: MULTI
DBEFile0 Name: MyDBE1
DDL Enabled: YES

No. of Runtime Control Block Pages: 128
No. of Data Buffer Pages: 200
Data Buffer Pages Memory Resident: NO
No. of Log Buffer Pages: 200
Max. Transactions: 30
Maximum Timeout: NONE
Default Timeout: MAXIMUM
Authorize Once per Session: OFF
```

SHOWDBE-EXIT

Terminates SHOWDBE execution.

Scope

SQLUtil SHOWDBE Command

Syntax

```
-> E[XIT]
```

Description

- This command can be abbreviated to E.

Authorization

Anyone can use this command.

Example

```
>> showdbe
DBEnvironment Name: PartsDBE.SomeGrp.SomeAcct
Maintenance Word: 
Output File Name (opt): 
-> .
.
.
-> exit
```

SHOWDBE-HELP

Displays and describes all SHOWDBE commands.

Scope

SQLUtil SHOWDBE Command

SQLUtil Syntax

```
-> HELP  
Command Name (opt): CommandName
```

Parameters

CommandName is the name of a SHOWDBE command.

Description

- If you do not specify a *CommandName*, the HELP command lists the names of all valid SHOWDBE commands and their descriptions.
- If a *CommandName* is specified, the correct syntax and an explanation for that command is displayed.
- You must enter // to leave the SHOWDBE HELP command.

Authorization

Anyone can use this command.

Example

```
>> showdbe
DBEnvironment Name: PartsDBE.SomeGrp.SomeAcct
Maintenance Word: Return
Output File Name (opt): Return
-> help
```

The following functions are available:

```
ALL      EXIT      HELP      LANG      MAINT      QUIT      STARTPARMS
```

For more information on any one of the functions, enter the command name at the prompt. Carriage return <cr> displays a brief description of each command. Type '//' to leave SHOWDBE HELP.

```
Command Name (opt): Return
```

Command Summary:

```
ALL          Shows all the information of the DBEnvironment.
EXIT         Terminates SHOWDBE execution.
HELP        Displays and describes all SHOWDBE commands.
LANG        Shows the language of the DBEnvironment.
MAINT       Shows the DBEnvironment maintenance word.
STARTPARMS Shows the startup parameters of the DBEnvironment.
QUIT        Terminates SHOWDBE execution.
```

```
Command Name (opt): maint
```

```
-> MAINT
```

Shows the DBEnvironment maintenance word.

```
Command Name (opt): //
```

```
-> quit
```

SHOWDBE-LANG

Shows the DBEnvironment language.

Scope

SQLUtil SHOWDBE Command

SQLUtil Syntax

-> LANG

Description

- This command displays the name of the DBEnvironment language.

Authorization

Anyone can use this command.

Example

```
>> showdbe
DBEnvironment Name: PartsDBE.SomeGrp.SomeAcct
Maintenance Word: 
Output File Name (opt): 
-> lang

DBEnvironment Language: NATIVE-3000

-> quit
```

SHOWDBE-MAINT

Shows the DBEnvironment maintenance word.

Scope

SQLUtil SHOWDBE Command

SQLUtil Syntax

```
-> MAINT
```

Description

- This command displays the DBEnvironment maintenance word.

Authorization

You must be the DBECreator, have SM capability, or supply the correct maintenance word.

Example

```
>> showdbe
DBEnvironment Name: PartsDBE.SomeGrp.SomeAcct
Maintenance Word: 
Output File Name (opt): 
-> maint

Maintenance Word: MaintWd

-> quit
```

SHOWDBE-QUIT

Terminates SHOWDBE execution.

Scope

SQLUtil SHOWDBE Command

Syntax

-> Q[UIT]

Description

- This command can be abbreviated to Q.

Authorization

Anyone can use this command.

Example

```
>> showdbe
DBEnvironment Name: PartsDBE.SomeGrp.SomeAcct
Maintenance Word: MaintWd
Output File Name (opt): 
-> .
   .
   .
-> quit
```

SHOWDBE-STARTPARMS

Shows the startup parameters of the DBEnvironment.

Scope

SQLUtil SHOWDBE Command

SQLUtil Syntax

```
-> STARTPARMS
```

Description

- This command shows the startup parameters of the DBEnvironment.

Authorization

Any user may execute this command.

Example

```
>> showdbe
DBEnvironment Name: PartsDBE.SomeGrp.SomeAcct
Maintenance Word: MaintWd
Output File Name (opt): 
-> startparms

DBEnvironment Language: NATIVE-3000
AutoStart Mode: ON
User Mode: MULTI
DBFile0 Name: PartsF0
DDL Enabled: YES
No. of Runtime Control Block Pages: 37
No. of Data Buffer Pages: 100
Data Buffer Pages Memory Resident: NO
No. of Log Buffer Pages: 24
Max. Transactions: 5
Maximum Timeout: NONE
Default Timeout: MAXIMUM
Authorize Once per session: OFF

-> quit
```

SHOWLOG

Displays information about the log files associated with the DBEnvironment.

Scope

SQLUtil Only

SQLUtil Syntax

```
>> SHOWLOG
DBEnvironment Name: DBEnvironmentName
Maintenance Word: MaintenanceWord
Connect? (y/n) (opt): { Y[es] }
                     { N[o] }
```

Parameters

DBEnvironmentName is the name of a DBEnvironment.

MaintenanceWord is the maintenance word.

Description

- This command lets you display the log file status within the DBEnvironment.
- Use SHOWLOG to display general information about the log as a whole and specific information about individual log files.
- The normal response to the prompt Connect? is Y, since connecting to the DBEnvironment will result in display of current status information. If you are not connected, SQLUtil does a static SHOWLOG, which examines only the log entries in the DBECon file, and does not report the number of free log pages available or tell whether the file is ready for backup. Static SHOWLOG is used immediately after restoring a DBEnvironment following a media failure, or in other cases when you cannot CONNECT to the DBEnvironment. The default response to the Connect? prompt is Y.
- SHOWLOG displays the following general information about the log as a whole:

Archive Mode	ON or OFF. ON means that you are using archive logging.
Log Sequence Number Containing Most Recent Archive Checkpoint	Indicates the log file that was active at the time the last STOREONLINE command was issued. This information is useful to HP support engineering personnel when you are using archive logging. If archive mode is OFF, this number is 0.
Current Sequence Number	Indicates the log file that has most recently been written to or switched into.
First Log Sequence Number Needed for Recovery	Indicates the log file from which you should start the rollforward recovery process if it is needed. This information is useful only when you are using archive logging. If archive mode is OFF, this number is 0.

- | | |
|---------------------------|--|
| Log Mode | SINGLE or DUAL, indicating single log files or dual log files. |
| Number of Free Log Blocks | Indicates the total amount of log file space available in 512-byte blocks. |
- SHOWLOG displays the following specific information about each log file:

First Log Name, Second Log Name	Indicates the file name, group and account of the log file.
First Log File Status, Second Log File Status	Shows the usability status of each file.
Log File Size	Indicates the number of pages in the log file. If you choose an odd number of log file pages in the START DBE NEW, START DBE NEWLOG statements or in the ADDLOG command, the number is rounded up to an even number, which is displayed here.
Log Identifier	A number that identifies a specific log file that has been added to the DBEnvironment. If you are using dual logging, one identifier number identifies both files in the pair. Log identifier numbers are reused when log files are purged and others are added.
Log Sequence Number	A number that identifies a specific log file in an archive log sequence. Log sequence numbers are never reused.
Log Backup Status	Backup status of the file. The type of status shown depends on whether you have specified a dynamic or static SHOWLOG. These are shown below.
 - For a dynamic SHOWLOG, the types of backup status are as follows:

Not Ready for Backup	This is the current file, which is not full yet.
Ready for Backup	The file is full, and the DBE is now using a different file, so this one is ready to be stored with STORELOG.
Backup is Done	The file has already been backed up.
Backup is Not Required	The DBEnvironment is in nonarchive mode, so the file does not need to be backed up.
 - For a static SHOWLOG, the types of backup status are as follows:

Backup is Required	The file has not been backed up yet.
Backup is Done	The file has already been backed up.
Backup is Not Required	The DBEnvironment is in nonarchive mode, so the file does not need to be backed up.
 - For archive logging, SHOWLOG displays information about the log files needed for rollforward recovery. You begin rollforward recovery with a SETUPRECOVERY command, then you use RESTORELOG and RECOVERLOG for each log file that is to be applied, starting with the *First Log Sequence Number Needed for Recovery*.
 - The DBEnvironment can be in use when this command is executed.

SHOWLOG

Authorization

You must be the DBECreator, have SM capability, or supply the correct maintenance word to use this command.

Examples

Dynamic SHOWLOG with archive logging

```
>> showlog
DBEnvironment Name: PartsDBE
Maintenance Word: 
Connect (y/n) (opt): y

Archive Mode: ON
Log Sequence Number Containing Most Recent Archive Checkpoint: 1
Current Log Sequence Number: 1
First Log Sequence Number Needed for Recovery: 1
Log Mode is: Dual
Number of Free Block(s): 340

First Log Name: PARTSLG1.TESTDBE.HOME
First Log File Status: Useable
Second Log Name: PARTSLG2.TESTDBE.HOME
Second Log File Status: Useable
Log File Size: 256
Log Identifier Is: 1
Log Sequence Number: 1
Log Backup Status: Not Ready for Backup
```

Static SHOWLOG with nonarchive logging

```
>> showlog
DBEnvironment Name: PartsDBE
Maintenance Word: 
Connect (y/n) (opt): n

Archive Mode: OFF
Log Sequence Number Containing Most Recent Archive Checkpoint: 0
Current Log Sequence Number: 1
First Log Sequence Number Needed for Recovery: 0
Log Mode is: Dual
Number of Free Log Blocks: N/A

First Log Name: PARTSLG1.TESTDBE.HOME
First Log File Status: Useable
Second Log Name: PARTSLG2.TESTDBE.HOME
Second Log File Status: Useable
Log File Size: 256
Log Identifier Is: 1
Log Sequence Number: 1
Log Backup Status: Backup Is Not Required
```

SHOWSET

Displays the setting of a parameter established with the SET command.

Scope

SQLUtil only.

SQLUtil Syntax

```
>> SHOWSET [BACKUP
            ECHO_ALL
            EXIT_ON_DBERR]
```

Parameters

BACKUP	is the name of the Backup setting established with the SET command. Possible parameters: <ul style="list-style-type: none"> ■ STORESET ■ RESTORESET ■ MOSET
ECHO_ALL	is the ECHO_ALL setting established with the SET command. Possible values are ON or OFF.
EXIT_ON_DBERR	is the EXIT_ON_DBERR setting established with the SET command. Possible values are ON or OFF.

Description

- The SHOWSET command displays the current value of the BACKUP, ECHO_ALL, or EXIT_ON_DBERR setting.
- If you issue the SHOWSET command without specifying an additional parameter, SQLUtil displays settings for all parameters that are set with the SET command.

Example

```
>> set echo_all on
>> showset echo_all
showset echo_all
      ECHO_ALL      : ON

>> set backup storeset '(*TAPE1, *TAPE2)'
>> showset backup
showset backup
      STORESET      option: (*TAPE1, *TAPE2)
      RESTORESET    option:
      MOSET          option:

>>
```

STORE

Makes a static backup of a DBEnvironment on tape.

Scope

SQLUtil

SQLUtil Syntax

```
>> STORE
```

```
WARNING: If you are using STORE to support RollForward Recovery
through archive logging, you must precede this command
with the BEGIN ARCHIVE and COMMIT ARCHIVE commands in
ISQL. (See the ALLBASE/SQL DBA Guide under Backup and
Recovery for details)
```

```
Do you wish to proceed (y/n)?: { Y[es] }
                               { N[o]  }
DBEnvironment Name: DBEnvironmentName
Maintenance Word:  MaintenanceWord
To File Name:      StoreFileName
```

Parameters

DBEnvironmentName is the name of the DBEnvironment to be stored.

MaintenanceWord is the maintenance word.

StoreFileName is the device name for the tape device that will receive the backup file.

Description

- STORE performs a *static* backup of a DBEnvironment. You use it when nonarchive logging is in use or when you are using archive logging without the TurboSTORE software. If you are logging in archive mode and TurboSTORE software is available, you normally use STOREONLINE to perform a backup.
- When using the STORE command you should be in the group and account which contains the DBECon file and the SYSTEM DBEFileSet. If you must later restore the DBEnvironment, you must be in the same group and account from which you originated the STORE.
- If you are using STORE for backups in archive mode, please refer to the section entitled “Static Backup Procedures in Archive Mode” in the “Backup and Recovery” chapter of this guide for a complete description of the procedure.
- This command allows the user to create a backup copy of the DBECon file and all DBEFiles of a DBEnvironment. Note that log files are not stored. If more than one volume is required to store the DBEnvironment, a request will be displayed on the console. The next tape can then be mounted. No reply is necessary for the continuation of the store on the new mount.
- The DBEnvironment cannot be in use while this command is executed.

Authorization

You must be the DBECreator, have SM capability, or supply the correct maintenance word to use this command.

Example

If you are doing a static store in conjunction with archive logging for later use in roll forward recovery, be sure to issue the `BEGIN ARCHIVE` and `COMMIT ARCHIVE` statements in ISQL immediately before doing the `STORE`.

Do not issue these two statements if you are using the `STOREONLINE` statement to store your DBEnvironment. Do not issue these two statements if you are using nonarchive logging and want to continue in nonarchive mode as these statements will automatically turn on archive logging.

```
ISQL=> BEGIN ARCHIVE;
```

```
ISQL=> COMMIT ARCHIVE;
```

Exit from ISQL and invoke SQLUtil.

```
>> store
WARNING: If you are using STORE to support RollForward Recovery
         through archive logging, you must precede this command
         with the BEGIN ARCHIVE and COMMIT ARCHIVE commands in
         ISQL. (See the ALLBASE/SQL DBA Guide under Backup and
         Recovery for details)
Do you wish to proceed (y/n)?: Y
DBEnvironment Name: PartsDBE
Maintenance Word: MaintWd
To File Name: TAPE

DBEnvironment stored.
```

STOREINFO

This command lists the physical name of all the files backed up on a specified tape.

Scope

SQLUtil Only

SQLUtil Syntax

```
>> STOREINFO
From File Name: StoreFileName
```

Parameters

StoreFileName is the name of the tape volume.

Description

- The STOREINFO command lists the names of the DBEFiles stored on the specified device.
- The STOREINFO command should be used from the same group and account from which the STORE or STOREONLINE command was issued. This is usually the group and account containing the DBECon file and the SYSTEM DBEFileSet.
- If the backup was created using a full STORE or STOREONLINE, the DBECon file will be shown as one of the listed files. If the backup was created using a STORE PARTIAL or STOREONLINE PARTIAL, the DBECon file is not stored and will not be shown in the list of stored files.
- If the DBEFiles have been moved to a group and account different from the one containing the DBECon file and the SYSTEM DBEFileSet, the fully qualified file name will be shown by STOREINFO. When you later restore the files, you must be in the group and account from which the files were stored (usually the group and account containing the DBECon file and SYSTEM DBEFileSet) and the file name used for the restore process must be exactly as shown by the STOREINFO command.

Authorization

Anyone can list the physical names of files on the archive.

Example

```
>> STOREINFO
From File Name: TAPE
DBEfile Name: PartsDBE
DBEfile Name: PartsFO
DBEfile Name: PurchDF1
DBEfile Name: PurchFX1
DBEfile Name: driveB/WarehDF1
DBEfile Name: driveC/WarehXF1
DBEfile Name: OrderDF1
DBEfile Name: FileData
DBEfile Name: RecDF1
```


STORELOG

Stores a copy of an archive log file that is ready for backup.

Scope

SQLUtil Only

SQLUtil Syntax

```
>> STORELOG
DBEnvironment Name: DBEnvironmentName
Maintenance Word: MaintenanceWord
Log Identifier (opt): LogIdentifier
To File Name? FileName
Use Static Store (y/n)? { Y[es] }
                       { N[o] }
Store Log File (y/n)? { Y[es] }
                      { N[o] }
```

Parameters

<i>DBEnvironmentName</i>	is the name of the DBEnvironment with which the file you are storing is associated.
<i>MaintenanceWord</i>	is the maintenance word.
<i>LogIdentifier</i>	is the log identifier number of the log file to be stored. Use the SHOWLOG command to display this number.
<i>FileName</i>	is the device name for the tape or disk device that will receive the store file.

Description

- This command lets you store an archive log file. The file to be stored must not be the current log file, and it must have the backup status of *Ready for Backup* in a dynamic SHOWLOG display.
- If you specify the log identifier number for a log file that is *Ready for Backup*, only the specified log file will be stored.
- If you omit the *LogIdentifier*, ALLBASE/SQL will store only the log file with the lowest sequence number that is storable—that is, has the lowest log sequence number and has *Ready for Backup* status in a dynamic SHOWLOG display.
- If you specify a log identifier number of 0, every log file which currently has the status *Ready for Backup* will be automatically stored to the same tape in order of increasing log sequence number. This allows you to store multiple log files to a single tape.
- If the *LogIdentifier* number you enter corresponds to a log file that is in use, the store will not take place.

STORELOG

- If more than one volume is required to store the log file, a request will be displayed on the console. The next tape can then be mounted. No reply is necessary for the continuation of the store on the new mount.
- For each successful STORELOG, label the backup tape with the log file name, sequence number, and the date and time of the backup. The sequence number will let you restore and recover log files in the correct order in the event that rollforward recovery is needed.
- If your system has the TURBOSTORE enhancements to the STORE utility, you should reply N to the prompt “Use Static Store (y/n)?” If you do not have TURBOSTORE, reply Y.
- The DBEnvironment may be in use while this command is executed. However, if your system does not have the TURBOSTORE enhancements to the STORE utility, the DBEnvironment must not be in use.

Authorization

You must be the DBECreator, have SM capability, or supply the correct maintenance word to use this command.

Example

STORELOG of single specified log file

```
>> storelog
DBEnvironment Name: PartsDBE
Maintenance Word: 
Log Identifier (opt): 2
To File Name? TAPE
Use Static Store (y/n)? n
Store Log File (y/n)? y

Log file 'lgn1' with Sequence Number      11 was stored.
```

STORELOG using a log identifier of 0 to store multiple log files to the same device

```
>> storelog
DBEnvironment Name: PartsDBE
Maintenance Word: 
Log Identifier (opt): 0
To File Name? TAPE
Use Static Store (y/n)? n
Store Log File (y/n)? y

Log file 'ptslogA2' with Sequence Number      5 was stored.
Log file 'ptslogA1' with Sequence Number      6 was stored.
Log file 'ptslogA3' with Sequence Number      7 was stored.
Log file 'ptslogA5' with Sequence Number      8 was stored.
```

STOREONLINE

Backs up a DBEnvironment to tape and enables archive logging if it is not already in effect.

Scope

SQLUtil

SQLUtil Syntax

```
>> STOREONLINE
DBEnvironment Name: DBEnvironmentName
Maintenance Word: MaintenanceWord
To File Name: StoreFileName
```

Parameters

DBEnvironmentName is the name of the DBEnvironment to be stored.

MaintenanceWord is the maintenance word.

StoreFileName is the device name for the tape device that will receive the backup.

Description

- You must have TurboSTORE software on your system to use STOREONLINE. If you do not have TurboSTORE, you must use the STORE command.
- STOREONLINE allows the user to create a backup copy of the DBECon file and all DBEFiles of a DBEnvironment. Note that log files are not stored. If more than one volume is required to store the DBEnvironment, a request will be displayed on the console. The next tape can then be mounted. No reply is necessary for the continuation of the store on the new mount.
- When you execute STOREONLINE you should be in the group and account which contains the DBECon file and the SYSTEM DBEFileSet. If you must later restore the DBEnvironment, you must be in the same group and account from which the DBEnvironment was stored.
- You issue the STOREONLINE command when the DBEnvironment is in normal use. If archive logging has not previously been in effect, this command enables it.
- STOREONLINE cannot be used if there are any pseudo-mapped files configured in your system. Before using STOREONLINE, convert all pseudo-mapped files back to ordinary mapped files using the MOVEFILE command.
- The following message appears after the STOREONLINE is complete:

```
DBEnvironment stored.
```

STOREONLINE

Caution

In order for an online backup to be of use for rollforward recovery, all logs containing transactions that were active during the execution of the STOREONLINE command must be available at recovery time. Without all the logs, the backup is worthless. This means that after the STOREONLINE command completes, you should do a STORELOG as soon as possible on any log files that contained active transactions during the time you did the STOREONLINE.

The range of sequence numbers of the log files containing transactions active during the STOREONLINE process is indicated at the completion of STOREONLINE. Both the beginning and ending archive log sequence numbers are displayed. You must store all log files in that range to be able to do roll forward recovery, later.

If there is a media failure involving the device containing the log files active during the STOREONLINE, and if the failure occurs before the files have been backed up, the DBEnvironment backup will not be usable. For the greatest protection, use dual logging with the dual files on different disk drives.

Authorization

You must be the DBECreator, have SM capability, or supply the correct maintenance word to use this command.

Example

```
>> storeonline
DBEnvironment Name: PartsDBE
Maintenance Word: MaintWd
To File Name: TAPE

DBEnvironment stored.

Begin Archive Log Sequence Number      2
End Archive Log Sequence Number        3
```

STOREONLINE PARTIAL

Backs up part of a DBEnvironment to tape and enables archive logging if it is not already in effect.

Scope

SQLUtil

SQLUtil Syntax - STOREONLINE PARTIAL

```
>> STOREONLINE PARTIAL
DBEnvironment Name: DBEnvironmentName
Maintenance Word: MaintenanceWord
To File Name: StoreFileName
Enter DBEFileset name or Carriage Return to Finish: DBEFilesetName
Enter DBEFileset name or Carriage Return to Finish:
Enter DBEFile name or Carriage Return to Finish: SystemFileName
Enter DBEFile name or Carriage Return to Finish:
```

Parameters

<i>DBEnvironmentName</i>	is the name of the DBEnvironment containing the subset to be stored.
<i>MaintenanceWord</i>	is the maintenance word.
<i>StoreFileName</i>	is always the device name for the tape that will receive the backup (backups cannot go to disk).
<i>DBEFilesetName</i>	is name of the DBEFileset to store. You may enter a list of DBEFileSets one name per line, followed by a blank line with a carriage return to terminate the list.
<i>SystemFileName</i>	is the physical name of the DBEFile to store. You may enter a list of DBEFiles one name per line, followed by a blank line with a carriage return to terminate the list.

Description

- You must have TurboSTORE software on your system to use STOREONLINE PARTIAL. If you do not have TurboSTORE, you must use the STORE PARTIAL command.
- STOREONLINE PARTIAL allows you to create a backup copy of a subset of the DBEFiles of a DBEnvironment. Note that DBECon file and the log files are not stored. If more than one volume is required to store the DBEnvironment, a request will be displayed on the console. The next tape can then be mounted. No reply is necessary for the continuation of the store on the new mount.
- When you execute STOREONLINE PARTIAL you should be in the same group and account which contains the DBECon file and SYSTEM DBEFileSet. If you must later restore the DBEnvironment, you must be in the same group and account from which it was stored.

STOREONLINE PARTIAL

- If a DBEFileset name is specified in STOREONLINE PARTIAL, all the DBEFiles in the DBEFileset are stored.
- To obtain the *SystemFileName* of a DBEFile, you can execute a statement that specifies the logical file name. Suppose a DBEFile was created with the following statement:

```
CREATE DBEFILE OrderDataF1
  WITH PAGES = 50,
  NAME = 'OrderDF1',
  TYPE = TABLE;
```

The STOREONLINE PARTIAL command requests the physical file name (*SystemFileName*), but if it is not remembered, the following statement using the logical file name will display the physical file name:

```
SELECT FILEID
FROM SYSTEM.DBEFILE
WHERE DBEFNAME = 'OrderDataF1';
```

The FILEID column gives the SystemFileName for the DBEFile.

- Issue the STOREONLINE PARTIAL command when the DBEnvironment is in normal use. If archive logging has not previously been in effect, this command enables it.
- STOREONLINE PARTIAL cannot be used if there are any pseudo-mapped files configured in your system which will take part in the store. Before using STOREONLINE PARTIAL, convert all pseudo-mapped files acted upon by STOREONLINE PARTIAL back to ordinary mapped files using the MOVEFILE command.
- The following message appears after the STOREONLINE PARTIAL is complete:

```
DBEnvironment stored.
```

Begin Archive Log Sequence Number	Number
End Archive Log Sequence Number	Number

Caution

In order for an online backup to be of use for rollforward recovery, all logs containing transactions that were active during the execution of the STOREONLINE PARTIAL command must be available at recovery time. Without all the logs, the backup is worthless. This means that after the STOREONLINE PARTIAL command completes, you should do a STORELOG as soon as possible on any log files that contained transactions active during the time you did the STOREONLINE.

The range of sequence numbers of the log files containing transactions active during the STOREONLINE PARTIAL process is indicated at the completion of STOREONLINE PARTIAL. Both beginning and ending archive log sequence numbers are displayed. You must store all log files in that range to be able to do roll forward recovery, later.

If there is a media failure involving the device containing the log files, and if the failure occurs before the files have been backed up, the DBEnvironment backup will not be usable. For the greatest protection, use dual logging with the dual files on different disk drives.

Authorization

You must be the DBECreator, have SM capability, or supply the correct maintenance word to use this command.

Example

```
>> storeonline partial
DBEnvironment Name: PartsDBE
Maintenance Word: MaintenanceWord
To File Name: TAPE
Enter DBEFileset name or Carriage Return to Finish: WarehFS
Enter DBEFileset name or Carriage Return to Finish:
Enter DBEFile name or Carriage Return to Finish: OrderDF1
Enter DBEFile name or Carriage Return to Finish:

DBEnvironment Stored

Begin Archive Log Sequence Number      2
End Archive Log Sequence Number         3
```

STORE PARTIAL

Makes a static backup of part of a DBEnvironment on tape.

Scope

SQLUtil only.

SQLUtil Syntax - STORE PARTIAL

```
>> STORE PARTIAL
WARNING:If you are using STORE to support rollforward recovery
through archive logging, you must precede this command
with the BEGIN ARCHIVE and COMMIT ARCHIVE commands.
(See the "Backup and Recovery" chapter of the
ALLBASE/SQL Database Administration Guide for details.)
Do you wish to proceed (y/n)?: { y[es] }
                               { n[o]  }
DBEnvironment Name: DBEnvironmentName
Maintenance Word:  MaintenanceWord
To File Name:      StoreFileName
Enter DBEFileset name or Carriage Return to Finish: DBEFileSetName
Enter DBEFileset name or Carriage Return to Finish:
Enter DBEFile name or Carriage Return to Finish: SystemFileName
Enter DBEFile name or Carriage Return to Finish:
Start backup (y/n)? { y[es] }
                   { n[o]  }
```

Parameters

<i>DBEnvironmentName</i>	is the name of the DBEnvironment containing the subset to be stored.
<i>Maintenance Word</i>	is the maintenance word.
<i>StoreFileName</i>	is the device name for the tape device that will receive the backup file.
<i>DBEFileSetName</i>	is the name of the DBEFileset to store. You may enter a list of DBEFileSets one name per line, followed with a blank line with a carriage return to terminate the list.
<i>SystemFileName</i>	is the physical name of the DBEFile to store. You may enter a list of DBEFiles one name per line, followed with a blank line with a carriage return to terminate the list.

Description

- STORE performs a *static* backup of a subset of a DBEnvironment. You use it when nonarchive logging is in use or when you are using archive logging without the TurboSTORE software. If you are logging in archive mode and TurboSTORE software is available, you normally use STOREONLINE PARTIAL to perform a partial backup.
- When using the STORE PARTIAL command you should be in the group and account which contains the DBECon file and the SYSTEM DBEFileSet. If you must later do a partial recovery of the DBEnvironment you must be in the same group and account from which the STORE PARTIAL was made.
- If you are using STORE for backups in archive mode, please refer to the section entitled “Static Backup Procedures in Archive Mode” in the “Backup and Recovery” chapter of this guide for a complete description of the procedure.
- This command allows the user to create a backup copy of a specified subset of the DBEFiles of a DBEnvironment. Note that the DBECon file and log files are not stored. If more than one volume is required to store the DBEnvironment files, a request will be displayed on the console. The next tape can then be mounted.
- If a DBEFileSet is specified in a PARTIAL backup, all the DBEFiles in the DBEFileset will be stored.
- To obtain the *SystemFileName* of a DBEFile, you can execute a statement that specifies the logical file name. Suppose a DBEFile was created with the following statement:

```
CREATE DBEFILE OrderDataF1
  WITH PAGES = 50,
  NAME = 'OrderDF1',
  TYPE = TABLE;
```

The STORE PARTIAL command requests the physical file name (*SystemFileName*), but if it is not remembered, the following statement using the logical file name will display the physical file name:

```
SELECT FILEID
FROM SYSTEM.DBEFILE
WHERE DBEFNAME = 'OrderDataF1';
```

The FILEID column gives the SystemFileName for the DBEFile. No reply is necessary for the continuation of the store on the new mount.

- The DBEnvironment cannot be in use while this command is executed.

Authorization

You must be the DBECreator, have SM capability, or supply the correct maintenance word to use this command.

STORE PARTIAL

Example

If you are doing a static STORE PARTIAL in conjunction with archive logging, for later use in rollforward recovery, be sure you issue the BEGIN ARCHIVE and COMMIT ARCHIVE statements in ISQL before doing the STORE PARTIAL.

Do not issue these two statements if you are using the STOREONLINE PARTIAL statement to store your DBEnvironment. Do not issue these two statements if you are using nonarchive logging and want to continue in nonarchive mode (these statements automatically turn on archive logging).

```
isql=>BEGIN ARCHIVE;
```

```
isql=>END ARCHIVE;
```

Exit ISQL and invoke SQLUtil

```
>> store partial
Do you wish to proceed (y/n)?: Y
DBEnvironment Name: PartsDBE
Maintenance Word: MaintenanceWord
To File Name: TAPE
Enter DBEFileset name or Carriage Return to Finish: WarehFS
Enter DBEFileset name or Carriage Return to Finish:
Enter DBEFile name or Carriage Return to Finish: OrderDF1
Enter DBEFile name or Carriage Return to Finish:

DBEnvironment stored.
```

SYSTEM

Escape temporarily to the operating system.

Scope

SQLUtil Only

SQLUtil Syntax

```
>> SYSTEM
```

Parameters

CommandName is the name of an operating system command.

Description

- Use the *RESUME* command to return to SQLUtil from MPE/iX.

Example

```
>> SYSTEM
: LISTF,3

ACCOUNT= SOMEACCT   GROUP= SOMEGRP

FILENAME CODE  -----LOGICAL RECORD-----  ----SPACE----
          SIZE  TYP          EOF          LIMIT R/B  SECTORS  #X MX
PARTSDBE PRIV  125W  FB           1           1  1         2  1  1
PARTSFO  PRIV  2048W FB          150          150  1        2416  5  5
PARTSLG1 PRIV  256W  FB           256          256  1         514  2  2
PARTSLG2 PRIV  256W  FB           256          256  1         514  2  2

: RESUME
>>
```

WRAPDBE

Associates a new DBEnvironment with the log files of an inaccessible DBEnvironment. It wraps the new DBEnvironment around orphaned log files.

Scope

SQLUtil Only

SQLUtil Syntax

```
>> WRAPDBE
WrapperDBEnvironment Name: WrapperDBEnvironmentName
Maintenance Word: MaintenanceWord
Wrapper Mode (log) (opt): Log
Enter Log File Name(s) Separated by a Blank? (RETURN to finish): LogFileNames1
Enter Log File Name(s) Separated by a Blank? (RETURN to finish):
Convert to Wrapper DBEnvironment? (y/n)? { Y[es] }
                                         { N[o] }
```

Parameters

<i>DBEnvironmentName</i>	is the name of a Wrapper DBEnvironment. Must be the name of an empty DBEnvironment previously created to wrap around the log files.
<i>Maintenance Word</i>	is the maintenance word. Must be the maintenance word of the <i>DBEnvironment</i> .
<i>Wrapper Mode</i>	is the mode. Log is the default and only mode presently allowed.
<i>LogFileNames1</i>	physical name of a log file that is to be wrapped.
<i>Convert to Wrapper DBEnvironment</i>	converts the empty DBEnvironment to a wrapper DBEnvironment or gives you the opportunity to exit this command without wrapping a DBEnvironment around the orphaned log file(s).

Description

- This command wraps a DBEnvironment around log files orphaned after a hard crash of a DBEnvironment.
- The DBEnvironment can be empty or it can have tables on which updates have been performed.
- No updates can be made to the DBEnvironment after it has been converted to a wrapper DBEnvironment with this command. Any updates must be performed before this command is issued.
- The log files around which the DBEnvironment is to be wrapped must be valid and must be marked as usable log files before this command is used. Refer to the section of this manual that describes wrapper DBEnvironment use.

- The log files entered should constitute a correct log file sequence list with no sequence numbers missing. If the DBEnvironment that crashed has dual log files, then log file names can be taken from either of the dual log files, as long as they constitute a proper sequence. The proper list of log files must be determined before using the WRAPDBE command.
- The DBEnvironment to be wrapped around the orphaned log files must be created before the WRAPDBE command is used. The DBEnvironment is created with the START DBE NEW statement. Refer to the “Setting up a Wrapper DBEnvironment” section of the “Backup and Recovery” chapter of this manual.
- WRAPDBE prompts for the names of files around which the WrapperDBE is to be wrapped. WRAPDBE does not accept more than one file name on a single line.
- Any log files associated with the wrapper DBEnvironment before it is converted are removed. Only the wrapped log files are associated with the DBEnvironment after the WRAPDBE command is executed.

For more information, refer to the “Backup and Recovery” chapter.

Authorization

You must be the superuser (HP-UX) or system administrator (MPE/iX) or DBECreator of the original inaccessible DBEnvironment to use this command.

Example

```
>> WRAPDBE
DBEnvironment Name: WRAPDBE1
Maintenance Word: Return
Wrapper Mode (log) (opt): log
Enter Log File Name Separated by a Blank? (RETURN to finish): LGM1
Enter Log File Name Separated by a Blank? (RETURN to finish):
Convert to Wrapper DBEnvironment (y/n)? y
```

SQLGEN

The following pages contain the SQLGEN command syntax. Refer to the “DBA Tasks and Tools” chapter for a discussion of how to use SQLGEN.

EDITOR

Invokes the current editor.

Scope

SQLGEN Only

SQLGEN Syntax

```
>> ED[EDITOR]
```

Description

- EDITOR is used to invoke the current editor. The default editor is EDITOR.PUB.SYS. (Use the SET EDITOR command to redefine the current editor.)

Example

```
>> EDITOR
HP32201A.07.17 EDIT/3000 WED, MAR 4, 1988, 8:39 AM
(C) HEWLETT-PACKARD CO. 1985
/text schema1
FILE UNNUMBERED
/list all, offline
***OFFLINE LISTING BEGUN***
/e
>>
```

EXIT

Terminates SQLGEN.

Scope

SQLGEN Only

SQLGEN Syntax

```
>> E[XIT]
```

Description

- EXIT is used to end SQLGEN execution. If the DBE session has not been terminated prior to using the EXIT command, then an automatic RELEASE occurs before exiting SQLGEN.

Example

```
>> EXIT
```

```
DBEnvironment has been RELEASED.
```

```
:
```

GENERATE ALL

Generates the SQL commands necessary to recreate an entire DBEnvironment.

Scope

SQLGEN Only

SQLGEN Syntax

```
>> [GEN[ERATE] ]ALL  
Schema file Name or '/' to STOP command >> SchemaFileName
```

Parameters

SchemaFileName is the name of the schema file where SQLGEN places the generated commands.

Description

- GENERATE ALL generates a set of commands that can be used to migrate all the objects of a DBEnvironment.
- GENERATE ALL does not generate the CREATE SCHEMA statement.
- GENERATE ALL calls all the GENERATE commands except GENERATE LOAD and GENERATE STATISTICS. All objects are selected for each GENERATE command.

The commands are called in the following order:

1. GENERATE DBE
2. GENERATE DBFILES
3. GENERATE SPECAUTH
4. GENERATE DEFAULTSPACE
5. GENERATE TEMPSPACES
6. GENERATE GROUPS
7. GENERATE PARTITION
8. GENERATE TABLES
9. GENERATE SPECAUTH
10. GENERATE TABAUTH
11. GENERATE INDEXES
12. GENERATE VIEWAUTH
13. GENERATE VIEWS
14. GENERATE PROCEDURES
15. GENERATE PROCAUTH
16. GENERATE RULES
17. GENERATE MODOPTINFO
18. GENERATE PROCOPTINFO
19. GENERATE MODAUTH
20. GENERATE INSTALLAUTH

- This is a description of each command:

GENERATE DBE	builds the commands necessary to start DBE new for a DBEnvironment.
GENERATE DBEFILES	builds ALLBASE/SQL commands to recreate one or more of the DBEFileSets and associated DBEFiles of the DBEnvironment.
GENERATE SPACEAUTH	builds ALLBASE/SQL commands to grant SECTIONSPACE and TABLESPACE authority.
GENERATE DEFAULTSPACE	builds ALLBASE/SQL commands to set the default SECTIONSPACE and TABLESPACE to certain DBEFileSet.
GENERATE TEMPSPACES	builds ALLBASE/SQL commands to recreate one or more of the TEMPSPACEs of the DBEnvironment.
GENERATE GROUPS	builds SQL commands to recreate all the groups of the DBEnvironment. It generates CREATE GROUP as well as commands to add users/groups to a group.
GENERATE PARTITION	builds ALLBASE/SQL commands to recreate one or more of the PARTITIONs of the DBEnvironment.
GENERATE TABLES	builds CREATE TABLE commands to recreate one or more tables of the DBEnvironment. You can also recreate associated indexes and table authorities for specified tables.
GENERATE SPECAUTH	builds ALLBASE/SQL commands to recreate DBA, CONNECT and RESOURCE authorities for a DBEnvironment.
GENERATE TABAUTH	builds ALLBASE/SQL commands to recreate authorities for one or more tables of the DBEnvironment.
GENERATE INDEXES	builds SQL commands to recreate one or more indexes of the DBEnvironment.
GENERATE VIEWAUTH	builds ALLBASE/SQL commands to recreate authorities for one or more views of the DBEnvironment.
GENERATE VIEWS	builds CREATE VIEW commands to recreate one or more views of the DBEnvironment.
GENERATE PROCEDURES	builds CREATE PROCEDURE commands to recreate one or more procedures of the DBEnvironment.
GENERATE PROCAUTH	builds GRANT EXECUTE commands to allow you to recreate authorities for one or more procedures stored in the DBEnvironment.
GENERATE RULES	builds SQL commands to recreate one or more rules of the DBEnvironment.
GENERATE MODOPTINFO	builds ALLBASE/SQL commands to generate modified access plans of sections belonging to certain modules.
GENERATE PROCOPTINFO	builds ALLBASE/SQL commands to generate modified access plans of sections belonging to certain procedures.
GENERATE MODAUTH	builds GRANT RUN commands to allow you to recreate authorities for one or more modules stored in the DBEnvironment.

GENERATE ALL

GENERATE builds GRANT INSTALL commands to recreate all Module Authorities in
INSTALLAUTH the DBEnvironment

Example

```
>> gen all

ALLBASE/SQL Command Generator for ALL

Schema File Name or '/' to STOP command >> partschm

Generating command to START DBE PARTSDBE

Generating command to CREATE DBEFILESET ORDERFS
Generating command to CREATE DBEFILESET PURCHF5
.
.
Generating command(s) for DBEFILE ORDERDATAF1
Generating command(s) for DBEFILE ORDERINDXF1
.
.
Generating CREATE GROUP AJ@SERED.ACCOUNTSPAYABLE
Generating CREATE GROUP AJ@SERED.DBEUSERS
.
.
Generating ADD JIM@FRANCIS TO GROUP ACCOUNTSPAYABLE
Generating ADD KAREN@RIZZO TO GROUP ACCOUNTSPAYABLE
.
.
Generating CREATE TABLE PURCHDB.INVENTORY
Generating CREATE TABLE PURCHDB.ORDERITEMS
.
.
Generating CREATE INDEX on PURCHDB.INVENTORY
Generating CREATE INDEX on PURCHDB.ORDERITEMS
.
.
Generating Authority for PURCHDB.INVENTORY
Generating Authority for PURCHDB.ORDERITEMS
.
.
Generating Special Authority for AJ@SERED
Generating Special Authority for JOHN@BROCK
.
.

>>
```

Schema File Produced

```

START DBE 'PARTSDBE' MULTI NEW
  DUAL LOG,
  BUFFER = (100, 24),
  TRANSACTION = 5,
  DBEFILEO DBEFILE PARTSDBEO
    WITH PAGES = 150,
    NAME = 'PARTSFO',
  LOG DBEFILE DBELOG1 AND DBELOG2
    WITH PAGES = 256,
    NAME = 'PARTSLG1'
      AND 'PARTSLG2';

CREATE DBEFILESET ORDERFS;

CREATE DBEFILESET PURCHFS;

CREATE DBEFILESET RECFS;

CREATE DBEFILESET WAREHFS;

CREATE DBEFILE ORDERDATAF1
  WITH PAGES = 50, NAME = 'ORDERDF1',
  TYPE = TABLE;

ADD DBEFILE ORDERDATAF1 TO DBEFILESET ORDERFS;

CREATE DBEFILE ORDERINDXF1
  WITH PAGES = 50, NAME = 'ORDERXF1',
  TYPE = INDEX;

ADD DBEFILE ORDERINDXF1 TO DBEFILESET ORDERFS;
      .
      .
      .
COMMIT WORK;

CREATE GROUP AJ@SERED.ACCOUNTSPAYABLE;

CREATE GROUP AJ@SERED.DBEUSERS;
      .
      .
      .

ADD JIM@FRANCIS TO GROUP ACCOUNTSPAYABLE;

ADD KAREN@RIZZO TO GROUP ACCOUNTSPAYABLE;
      .
      .
      .

```

GENERATE ALL

```
COMMIT WORK;

CREATE PUBLIC TABLE PURCHDB.INVENTORY
(PARTNUMBER          CHAR( 16)   NOT NULL,
 BINNUMBER           SMALLINT    NOT NULL,
 QTYONHAND           SMALLINT,
 LASTCOUNTDATE      CHAR( 8),
 COUNTCYCLE          SMALLINT,
 ADJUSTMENTQTY       SMALLINT,
 REORDERQTY          SMALLINT,
 REORDERPOINT        SMALLINT) IN WAREHFS;
.
.
```

```
COMMIT WORK;
```

```
CREATE UNIQUE INDEX INVPARTNUMINDEX
ON PURCHDB.INVENTORY
(PARTNUMBER);
.
.
```

```
COMMIT WORK;
```

```
GRANT SELECT,
      INSERT,
      DELETE,
      UPDATE
ON PURCHDB.INVENTORY TO DBEUSERS;
```

```
GRANT SELECT,
      INSERT,
      DELETE,
      UPDATE
ON PURCHDB.INVENTORY TO PURCHASING;
.
.
```

```
COMMIT WORK;
```

```
GRANT DBA TO AJ@SERED;
GRANT DBA TO JOHN@BROCK;
COMMIT WORK;
```

GENERATE DBE

Generates the SQL command to START DBE NEW for a DBEnvironment.

Scope

SQLGEN Only

SQLGEN Syntax

```
>> [ GEN[ERATE] ] DBE
  Schema File Name or '//' to STOP command >> SchemaFileName
```

Parameters

SchemaFileName is the name of the schema file where SQLGEN places the generated command.

Description

- The GENERATE DBE command builds the START DBE NEW command necessary to START the new DBEnvironment. Note that since the logical log DBEFile names are not stored in the system catalog, SQLGEN cannot know what these names are. Consequently, the names DBELOG1 and, if dual logging is in effect, DBELOG2 are used.
- No COMMIT WORK is generated for this command.

Example

```
>> gen dbe

ALLBASE/SQL Command Generator for DBE

Schema File Name or '//' to STOP command >> dbepart

Generating command to START DBE PARTSDBE

>>
```

Schema File Produced

```
START DBE 'PARTSDBE' MULTI NEW
DUAL LOG,
BUFFER = (100, 24),
TRANSACTION = 5,
DBEFILEO DBEFILE PARTSDBEO
  WITH PAGES = 150,
  NAME = 'PARTSFO',
LOG DBEFILE DBELOG1 AND DBELOG2
  WITH PAGES = 256,
  NAME = 'PARTSLG1'
  AND 'PARTSLG2';
```

GENERATE DBEFILES

Generates SQL commands to recreate one or more DBEFileSets and associated DBEFiles.

Scope

SQLGEN Only

SQLGEN Syntax

```
>> [GEN[ERATE] ]DBEFILE[S ]
Schema File Name or '/' to STOP command >> SchemaFileName
DBEFileSet Name >> DBEFileSetName
```

Parameters

SchemaFileName is the name of the schema file where SQLGEN places the generated commands.

DBEFileSetName is the name of the DBEFileSet for which you wish to GENERATE commands.

Description

- GENERATE DBEFILES builds commands to recreate DBEFileSets. It then builds commands to recreate DBEFiles and to associate these DBEFiles with their respective DBEFileSets.
- When @ is entered, commands for all DBEFileSets and DBEFiles are generated, including DBEFiles which have not been added to a DBEFileSet.
- When generating the SYSTEM DBEFileSet, commands to CREATE DBEFILESET and build DBEFILE0 are not generated since these commands are issued by START DBE NEW when the DBEnvironment is recreated.

Example

```
>> gen dbefile
ALLBASE/SQL Command Generator for DBEFiles

Schema File Name or '/' to STOP command >> dbefpart

Please enter DBEFileSet Names. Type @ for all, ? for a list of
DBEFileSet names, or RETURN to quit.

DBEFileSet Name >> WarehFS

Generating command to CREATE DBEFILESET WAREHFS

Generating command(s) for DBEFILE WAREHDATAF1
Generating command(s) for DBEFILE WAREHINDXF1

DBEFileSet Name >> Return
>>
```

Schema File Produced

```
CREATE DBFILESET WAREHFS;  
  
CREATE DBFILE WAREHDATAF1  
  WITH PAGES = 50, NAME = 'WAREHDF1',  
  TYPE = TABLE;  
  
ADD DBFILE WAREHDATAF1 TO DBFILESET WAREHFS;  
  
CREATE DBFILE WAREHINDXF1  
  WITH PAGES = 50, NAME = 'WAREHXF1',  
  TYPE = INDEX;  
  
ADD DBFILE WAREHINDXF1 TO DBFILESET WAREHFS;  
  
COMMIT WORK;
```

GENERATE DEFAULTSPACE

Generates SQL commands necessary to recreate the default DBEFileSet for section or table space.

Scope

SQLGEN Only

SQLGEN Syntax

```
>> [GEN[ERATE] ]DEFAULTSPACE  
Schema File Name or '/' to STOP command >> SchemaFileName
```

Parameters

SchemaFileName is the name of the schema file where SQLGEN places the generated commands.

Example

```
>> DEFAULTSPACE  
  
ALLBASE/SQL Command Generator for Default DBEFileset  
  
Schema File Name or '/' to STOP command >> DEFAULTS  
  
Generating command to SET DEFAULT TABLESPACE to DBEFileSet SYSTEM  
  
Generating command to SET DEFAULT SECTIONSPACE to DBEFileSet PURCHFS  
  
>>
```

Schema File Produced

```
SET DEFAULT TABLESPACE TO DBEFILESET SYSTEM  
FOR PUBLIC;  
  
SET DEFAULT SECTIONSPACE TO DBEFILESET PURCHFS  
FOR PUBLIC;  
  
COMMIT WORK;
```

GENERATE GROUPS

Generates SQL commands necessary to recreate the groups of a DBEnvironment.

Scope

SQLGEN Only

SQLGEN Syntax

```
>> [ GEN[ERATE] ] GROUP[S]
      Schema File Name or '/' to STOP command >> SchemaFileName
```

Parameters

SchemaFileName is the name of the schema file where SQLGEN places the generated commands.

Description

- GENERATE GROUPS builds the commands necessary to recreate all the groups of a DBEnvironment. It generates CREATE GROUP as well as commands to add users/groups to a group.

Example

```
>> generate group

ALLBASE/SQL Command Generator for Groups

Schema File Name or '/' to STOP command >> grpparts

Generating CREATE GROUP AJ@SERED.ACCOUNTSPAYABLE
      .
      .
Generating ADD JIM@FRANCIS TO GROUP ACCOUNTSPAYABLE
Generating ADD KAREN@RIZZO TO GROUP ACCOUNTSPAYABLE
      .
      .

>>
```

Schema File Produced

```
CREATE GROUP AJ@SERED.ACCOUNTSPAYABLE;

CREATE GROUP AJ@SERED.DBEUSERS;

CREATE GROUP AJ@SERED.PURCH;
      .
      .
      .

ADD JIM@FRANCIS TO GROUP ACCOUNTSPAYABLE;

ADD KAREN@RIZZO TO GROUP ACCOUNTSPAYABLE;
```

GENERATE GROUPS

```
ADD STACEY@WOLF TO GROUP ACCOUNTSPAYABLE;  
.  
.  
.  
.  
COMMIT WORK;
```

GENERATE INDEXES

Generates the SQL commands necessary to recreate indexes for one or more tables of the DBEnvironment.

Scope

SQLGEN Only

SQLGEN Syntax

```
>> [ GEN[ERATE] ] INDEX[ES ]
Schema File Name or '/' to STOP command >> SchemaFileName
Owner Name >> OwnerName
Do you wish to specify Table Names for each Owner (n/y)?  $\left\{ \begin{array}{l} \text{N}[\text{O}] \\ \text{Y}[\text{ES}] \end{array} \right\}$ 
Table Name for Owner OwnerName >> TableName
```

Parameters

<i>SchemaFileName</i>	is the name of the schema file where SQLGEN places the generated commands.
<i>OwnerName</i>	is the name of the Owner for whom you want to GENERATE INDEXES.
NO or YES	respond YES to specify certain tables for the named owner; respond NO to select all table names for the owner.
<i>TableName</i>	is the name of a table for which you wish to GENERATE INDEXES.

Example

```
>> generate indexes

ALLBASE/SQL Command Generator for INDEXES

Schema File Name or '/' to STOP command >> indxpart

Please enter Owner Names. Type @ for all, ? for a list of
Owner Names, or RETURN to quit.

Owner Name >> @
Do you wish to specify Table Names for each Owner (n/y)? n

Generating CREATE INDEX on PURCHDB.INVENTORY
Generating CREATE INDEX on PURCHDB.ORDERITEMS

>>
```

GENERATE INDEXES

Schema File Produced

```
CREATE UNIQUE INDEX INVPARTNUMINDEX
ON PURCHDB.INVENTORY
(PARTNUMBER);

CREATE CLUSTERING INDEX ORDERITEMINDEX
ON PURCHDB.ORDERITEMS
(ORDERNUMBER);

CREATE CLUSTERING INDEX ORDERNUMINDEX
ON PURCHDB.ORDERS
(ORDERNUMBER);
.
.
.

COMMIT WORK;
```

GENERATE INSTALLAUTH

Generates GRANT INSTALL commands necessary to recreate all the Module Authorities in the DBEnvironment.

Scope

SQLGEN Only

SQLGEN Syntax

```
>> [ GEN[ERATE] ]INSTALLAUTH
Schema File Name or '/' to STOP command >> SchemaFileName
User Name >> UserName
```

Parameters

SchemaFileName is the name of the schema file where SQLGEN places the generated command.

User Name is the user name for the modules for whom you are granting authority.

Description

- GENERATE INSTALLAUTH builds GRANT INSTALL commands to recreate all Module Authorities in the DBEnvironment.
- Authorities can be granted for the modules of one or more users of the DBEnvironment.

Example

```
>> generate installauth

ALLBASE/SQL Command Generator for INSTALL Authority

Schema File Name or '/' to STOP command >> INSTPART
Please enter User Names. Type @ for all, ? for a list of
User Names, or RETURN to quit.

User Name >>@
_

Generating Install Authorities for JIM@FRANCIS
Generating Install Authorities for KAREN@RIZZO
.
.
>>
```

Schema File Produced

```
GRANT INSTALL
  TO JIM@FRANCIS;

GRANT INSTALL
  TO KAREN@RIZZO;

COMMIT WORK;
```

GENERATE LOAD

Generates schema files to UNLOAD and LOAD one or more tables in a DBEnvironment.

Scope

SQLGEN Only

SQLGEN Syntax

```
>> [GEN[ERATE] ]LOAD
Unload Schema File Name or '/' to STOP command >> UnloadSchemaFile
Load Schema File Name or '/' to STOP command >> LoadSchemaFile
Prefix for Unloaded Data File Names >> UNLDPrefix
UPDATE STATISTICS after Loading (n/y)? { N[O] }
                                         { Y[ES] }
Internal or External Format (Int/Ext) >> { INT[ERNAL] }
                                         { EXT[ERNAL] }
NULL Indicator (?) >> NullIndicator
Prefix for DESCRIPTION File Names >> DESCPrefix
Owner Name >> OwnerName
Do you wish to specify Table Names for each Owner (n/y)? { N[O] }
                                                           { Y[ES] }
Table Name for Owner OwnerName >> TableName
```

Parameters

<i>UnloadSchemaFile</i>	is the name of the schema file where SQLGEN places the commands that UNLOAD the data into data files.
<i>LoadSchemaFile</i>	is the name of the schema file where SQLGEN places the commands that LOAD the data back into tables.
<i>UNLDPrefix</i>	is the prefix used to name files containing the unloaded data (max: 11 bytes).
NO or YES	respond NO if no UPDATE is desired; respond YES to UPDATE STATISTICS after the LOAD.
INTERNAL or EXTERNAL	indicates the format you want the UNLOAD/LOAD to use.
<i>NULLIndicator</i>	is the symbol you choose to use for the NULL indicator. The default is a question mark (external format only).
<i>DESCPrefix</i>	is the prefix used to name description files (external format only; max: 11 bytes).
<i>OwnerName</i>	is the name of the owner whose tables you wish to LOAD/UNLOAD.
NO or YES	respond NO to select all table names for the owner; respond YES to specify certain tables for the named owner.
<i>TableName</i>	is the name of a table you wish to UNLOAD/LOAD.

Description

- GENERATE LOAD builds two command files. One contains commands for unloading data from selected tables into files. The other contains commands for loading the data in the files back into tables.
- EXTERNAL or INTERNAL format can be selected and used. Optionally, you can UPDATE STATISTICS after the LOAD.
- LOAD EXTERNAL requires a prefix for naming description files and a NULL indicator.

Example

```
>> load

ALLBASE/SQL Command Generator for LOAD

Unload Schema File Name or '/' to STOP command >>puruld
Load Schema File Name or '/' to STOP command >> purld
Prefix for Unloaded Data File Names >> purch
UPDATE STATISTICS after Loading (n/y)? yes
Internal Format or External Format (Int/Ext) >> int

Please enter Owner Names. Type @ for all, ? for a list of
Owner Names, or RETURN to quit.

Owner Name >> purchdb
Do you wish to specify Table Names for each Owner(n/y)? n

Generating [Int] commands for PURCHDB.INVENTORY
Generating [Int] commands for PURCHDB.ORDERITEMS

Owner Name >> 
>>
```

UNLOAD Schema File Produced

```
UNLOAD TO INTERNAL purch1 FROM PURCHDB.INVENTORY;

UNLOAD TO INTERNAL purch2 FROM PURCHDB.ORDERITEMS;

UNLOAD TO INTERNAL purch3 FROM PURCHDB.ORDERS;

COMMIT WORK;
```

LOAD Schema File Produced

```
SET AUTOCOMMIT ON;

LOAD FROM INTERNAL purch1 TO PURCHDB.INVENTORY;

UPDATE STATISTICS FOR TABLE PURCHDB.INVENTORY;

LOAD FROM INTERNAL purch2 TO PURCHDB.ORDERITEMS;

UPDATE STATISTICS FOR TABLE PURCHDB.ORDERITEMS;

COMMIT WORK;
```

GENERATE MODAUTH

Generates SQL commands necessary to recreate module authorities for one or more owners of a DBEnvironment.

Scope

SQLGEN Only

SQLGEN Syntax

```
>> [GEN[ERATE] ]MODAUTH
Schema File Name or '/' to STOP command >> SchemaFileName
Owner Name >> OwnerName
```

Parameters

SchemaFileName is the name of the schema file where SQLGEN places the generated commands.

OwnerName is the owner of the modules for which you are granting authority.

Description

- GENERATE MODAUTH builds GRANT RUN commands to recreate authorities for modules stored in a DBEnvironment.
- Authorities can be granted for the modules of one or more owners of the DBEnvironment.

Example

```
>> generate modauth

ALLBASE/SQL Command Generator for Module Authority

Schema File Name or '/' to STOP command >> modapart

Please enter Owner Names. Type @ for all, ? for a list of
Owner Names, or RETURN to quit.

Owner Name >> @

Generating GRANT RUN on PURCHMANAGERS.REPORTPROG
      .
      .

>>
```

Schema File Produced

```
GRANT RUN ON PURCHMANAGERS.REPORTPROG  
TO PURCHDEMAINT;
```

```
GRANT RUN ON PURCHMANAGERS.UPDATEPROG  
TO PURCHDEMAINT;
```

```
.  
. .  
. . .
```

```
COMMIT WORK;
```

GENERATE MODOPTINFO

Generates SQL commands necessary to recreate modified access plans for module sections.

Scope

SQLGEN Only

SQLGEN Syntax

```
>> [GEN[ERATE] ]MODOPTINFO
Schema File Name or '/' to STOP command >> SchemaFileName
Owner Name >> OwnerName
Do you wish to specify Module Names for each Owner (n/y)?  $\left\{ \begin{array}{l} \text{N}[\text{O}] \\ \text{Y}[\text{ES}] \end{array} \right\}$ 
Module Name for Owner OwnerName >> ModuleName
```

Parameters

<i>SchemaFileName</i>	is the name of the schema file where SQLGEN places the generated commands.
<i>OwnerName</i>	is the name of the owner for whom you wish to generate modified access plans.
NO or YES	respond NO to specify all module names for the owner; respond YES to specify certain module names for the owner.
<i>ModuleName</i>	is the name of a module for which you wish to generate modified access plans.

Example

```
>> MODOPTINFO

ALLBASE/SQL Command Generator for Module Setoptinfo

Schema File Name or '/' to STOP command >> MOPTINFO

Please enter Owner Names. Type @ for all, ? for a list of
Owner Names, or RETURN to quit.

Owner Name >> PURCHDB
Do you wish to specify Module Names for each Owner (n/y)? n

Generating command to SETOPT for Module PURCHDB.MOD1

Generating command to SETOPT for Module PURCHDB.MOD2

Owner Name >> Return
```

Schema File Produced

```
SETOPT GENERAL INDEXSCAN;  
VALIDATE MODULE SECTION  
  PURCHDB.MOD1(1);
```

```
SETOPT GENERAL INDEXSCAN;  
VALIDATE MODULE SECTION  
  PURCHDB.MOD1(2);
```

```
SETOPT GENERAL INDEXSCAN;  
VALIDATE MODULE SECTION  
  PURCHDB.MOD2(1);
```

```
SETOPT CLEAR;
```

```
COMMIT WORK;
```

GENERATE PARTITION

Generates SQL commands necessary to recreate DBEnvironment partitions.

Scope

SQLGEN Only

SQLGEN Syntax

```
>> [GEN[ERATE] ]PARTITION
Schema File Name or '/' to STOP command >> SchemaFileName
PARTITION Name >> PartitionName
```

Parameters

SchemaFileName is the name of the schema file where SQLGEN places the generated commands.

PartitionName is the name of the partition to be recreated.

Example

```
>> PARTITION

ALLBASE/SQL Command Generator for Partitions

Schema File Name or '/' to STOP command >> PARTNS

Please enter PARTITION Names. Type @ for all, ? for a list of
PARTITION Names, or RETURN to quit.

PARTITION Name >> CAVENDOR

Generating command to CREATE PARTITION CAVENDOR

PARTITION Name >> Return
```

Schema File Produced

```
CREATE PARTITION CAVENDOR
WITH ID = 12;

COMMIT WORK;
```

GENERATE PROCAUTH

Generates SQL statements necessary to recreate authorities for one or more procedures in a DBEnvironment.

Scope

SQLGEN only

SQLGEN Syntax

```
>> [ GEN[ERATE] ] PROCAUTH
Schema File Name or '/' to STOP command >> SchemaFileName
Owner Name >> OwnerName
Do you wish to specify Procedure Names for each Owner (n/y)? { N[O] }
                                                             { Y[ES] }
Procedure Name for Owner OwnerName >> ProcedureName
```

Parameters

<i>SchemaFileName</i>	is the name of the schema file where SQLGEN places generated commands.
<i>OwnerName</i>	is the name of the owner of the procedure whose authorities you wish to generate.
NO or YES	respond NO to select all procedures for the owner; respond YES to select certain procedures for the owner.
<i>ProcedureName</i>	is the name of a procedure whose authorities you wish to generate.

Description

- GENERATE PROCAUTH builds GRANT EXECUTE ON PROCEDURE statements to recreate authorities for one or more procedures in a DBEnvironment.

GENERATE PROCAUTH

Example

```
>> gen procauth

ALLBASE/SQL Command Generator for Procedure Authority

Schema File Name or '//' to STOP command >> procpart

Please enter Owner Names. Type @ for all, ? for a list of
Owner Names, or RETURN to quit.

Owner Name >> PurchDB
Do you wish to specify Procedure Names for each Owner (n/y)? n

Generating GRANT EXECUTE ON PROCEDURE PURCHDB.PROCESS12
Generating GRANT EXECUTE ON PROCEDURE PURCHDB.REPORTMONITOR

Owner Name >> 

>>
```

Schema File Produced

```
/* This file was created with a user language environment of */
/* NATIVE-3000 */

GRANT EXECUTE ON PROCEDURE PURCHDB.PROCESS12
  TO MANAGERS;

GRANT EXECUTE ON PROCEDURE PURCHDB.REPORTMONITOR
  TO JACK@TEST;

COMMIT WORK;
```

GENERATE PROCEDURES

Generates SQL statements to recreate one or more procedures in a DBEnvironment.

Scope

SQLGEN only

SQLGEN Syntax

```
>> [ GEN[ERATE] ] PROCEDURE[S]
Schema File Name or '/' to STOP command >> SchemaFileName
Owner Name >> OwnerName
Do you wish to specify Procedure Names for each Owner (n/y)? { N[O] }
                                                                { Y[ES] }
Procedure Name for Owner OwnerName >> ProcedureName
```

Parameters

<i>SchemaFileName</i>	is the name of the schema file where SQLGEN places generated commands.
<i>OwnerName</i>	is the name of the owner whose procedures you wish to generate.
NO or YES	respond NO to select all procedures for the owner; respond YES to select certain procedures for the owner.
<i>ProcedureName</i>	is the name of a procedure you wish to generate.

Description

- GENERATE PROCEDURES builds CREATE PROCEDURE commands to recreate one or more procedures in the DBEnvironment.

GENERATE PROCEDURES

Example

```
>> gen procedures

ALLBASE/SQL Command Generator for Procedures

Schema File Name or '//' to STOP command >> partsch

Please enter Owner Names. Type @ for all, ? for a list of
Owner Names, or RETURN to quit.

Owner Name >> PurchDB
Do you wish to specify Procedure Names for each Owner (n/y)? n

Generating command to CREATE PROCEDURE PURCHDB.DISCOUNTPART
Generating command to CREATE PROCEDURE PURCHDB.REPORTMONITOR

Owner Name >> Return

>>
```

Schema File Produced

```
/* This file was created with a user language environment of */
/* NATIVE-3000 */

CREATE PROCEDURE PURCHDB.DISCOUNTPART
(PARTNUMBER CHAR 16)) AS
begin declare SalesPrice decimal(10,2); declare Discount
decimal(10,2); select SalesPrice into :SalesPrice from
PurchDB.Parts where PartNumber = :PartNumber; if :SalesPrice >
100. then :Discount = .80*:SalesPrice; insert into
PurchDB.Discounts values (:PartNumber, :Discount); endif; end;

CREATE PROCEDURE PURCHDB.REPORTMONITOR
(NAME CHAR( 20) NOT NULL,
OWNER CHAR( 20) NOT NULL,
TYPE CHAR( 10) NOT NULL) AS
begin insert into PurchDB.ReportMonitor values (:Type,
CURRENT_DATETIME, USER, :Name, :Owner); end;

COMMIT WORK;
```

GENERATE PROCOPTINFO

Generates SQL commands necessary to recreate modified access plans for procedure sections.

Scope

SQLGEN Only

SQLGEN Syntax

```
>> [ GEN[ERATE] ] PROCOPTINFO
Schema File Name or '/' to STOP command >> SchemaFileName
Owner Name >> OwnerName
Do you wish to specify Procedure Names for each Owner (n/y)?  $\left\{ \begin{array}{l} N[O] \\ Y[ES] \end{array} \right\}$ 
Procedure Name for Owner OwnerName >> ProcedureName
```

Parameters

<i>SchemaFileName</i>	is the name of the schema file where SQLGEN places the generated commands.
<i>OwnerName</i>	is the name of the owner for whom you wish to generate modified access plans.
NO or YES	respond NO to specify all procedure names for the owner; respond YES to specify certain procedure names for the owner.
<i>ProcedureName</i>	is the name of a procedure for which you wish to generate modified access plans.

Example

```
>> PROCOPTINFO

ALLBASE/SQL Command Generator for Procedure Setoptinfo

Schema File Name or '/' to STOP command >> POPTINFO

Please enter Owner Names. Type @ for all, ? for a list of
Owner Names, or RETURN to quit.

Owner Name >> PURCHDB
Do you wish to specify Procedure Names for each Owner (n/y)? n

Generating command to SETOPT for Procedure PURCHDB.PROC1

Generating command to SETOPT for Procedure PURCHDB.PROC2

Owner Name >> Return
```

GENERATE PROCOPTINFO

Schema File Produced

```
SETOPT GENERAL SERIALSCAN;  
VALIDATE PROCEDURE SECTION  
PURCHDB.PROC1(1);
```

```
SETOPT GENERAL SERIALSCAN;  
VALIDATE PROCEDURE SECTION  
PURCHDB.PROC2(1);
```

```
SETOPT CLEAR;
```

```
COMMIT WORK;
```

GENERATE RULES

Generates SQL commands to recreate one or more rules in a DBEnvironment.

Scope

SQLGEN only

SQLGEN Syntax

```
>> [ GEN[ERATE] ] RULE[S]
Schema File Name or '/' to STOP command >> SchemaFileName
Owner Name >> OwnerName
Do you wish to specify Rule Names for each Owner (n/y)? { N[O] }
                                                         { Y[ES] }
Rule Name for Owner OwnerName >> RuleName
```

Parameters

<i>SchemaFileName</i>	is the name of the schema file where SQLGEN places generated commands.
<i>OwnerName</i>	is the name of the owner whose rules you wish to generate.
NO or YES	respond NO to select all rules for the owner; respond YES to select certain rules for the owner.
<i>RuleName</i>	is the name of a rule you wish to generate.

Description

- GENERATE RULES builds CREATE RULE commands to recreate one or more rules in the DBEnvironment.

Example

```
>> generate rules

ALLBASE/SQL Command Generator for Rules

Schema File Name or '/' to STOP command >> partschema

Please enter Owner Names. Type @ for all, ? for a list of
Owner Names, or RETURN to quit.

Owner Name >> PurchDB
Do you wish to specify Rule Names for each Owner (n/y)? n

Generating command to CREATE RULE PURCHDB.INSERTREPORT
Generating command to CREATE RULE PURCHDB.UPDATEREPORT
Generating command to CREATE RULE PURCHDB.DELETEREPORT

Owner Name >> Return

>>
```

GENERATE RULES

Schema File Produced

```
/* This file was created with a user language environment of */  
/* NATIVE-3000 */
```

```
CREATE RULE PURCHDB.INSERTREPORT  
AFTER INSERT  
ON PURCHDB.REPORTS  
execute procedure PurchDB.ReportMonitor (NEW.ReportName,  
NEW.ReportOwner, 'INSERT');
```

```
CREATE RULE PURCHDB.UPDATEREPORT  
AFTER UPDATE(REPORTOWNER,REPORTNAME)  
ON PURCHDB.REPORTS  
execute procedure PurchDB.ReportMonitor (NEW.ReportName,  
NEW.ReportOwner, 'UPDATE');
```

```
CREATE RULE PURCHDB.DELETEREPORT  
AFTER UPDATE  
ON PURCHDB.REPORTS  
execute procedure PurchDB.ReportMonitor (OLD.ReportName,  
OLD.ReportOwner, 'DELETE');
```

```
COMMIT WORK;
```

GENERATE SPACEAUTH

Generates SQL commands necessary to recreate authorities for section or table space.

Scope

SQLGEN Only

SQLGEN Syntax

```
>> [ GEN[ERATE] ] SPACEAUTH
Schema File Name or '/' to STOP command >> SchemaFileName
DBEFileSetName >> DBEFileSetName
```

Parameters

SchemaFileName is the name of the schema file where SQLGEN places the generated commands.

DBEFileSetName is the name of the DBEFileSet for which section or table space authorities have been granted.

Example

```
>> SPACEAUTH

ALLBASE/SQL Command Generator for SPACEAUTH

Schema File Name or '/' to STOP command >> SPAUTH

Please enter DBEFileSet Names. Type @ for all, ? for a list of
DBEFileSet Names, or RETURN to quit.

DBEFileSet Name >> PURCHFS

Generating command to GRANT SPACE AUTHORITY ON DBEFileSet PURCHFS

DBEFileSet Name >> Return
```

Schema File Produced

```
GRANT TABLESPACE ON DBEFILESET PURCHFS
  TO PUBLIC;

GRANT SECTIONSPACE ON DBEFILESET PURCHFS
  TO PUBLIC;

COMMIT WORK;
```

GENERATE SPECAUTH

Generates SQL commands necessary to recreate all special authorities for a DBEnvironment.

Scope

SQLGEN Only

SQLGEN Syntax

```
>> [GEN[ERATE] ]SPECAUTH  
Schema File Name or '/' to STOP command >> SchemaFileName
```

Parameters

SchemaFileName is the name of the schema file where SQLGEN places the generated commands.

Description

- GENERATE SPECAUTH builds commands to recreate DBA, CONNECT, and RESOURCE authorities for a DBEnvironment.

Example

```
>> generate specauth  
  
ALLBASE/SQL Command Generator for Special Authority  
  
Schema File Name or '/' to STOP command >> speapart  
  
Generating Special Authority for AJ@SERED  
Generating Special Authority for JOHN@BROCK  
.  
.  
.  
  
>>
```

Schema File Produced

```
GRANT DBA TO AJ@SERED;  
  
GRANT DBA TO JOHN@BROCK;  
.  
.  
.  
COMMIT WORK;
```

GENERATE STATISTICS

Generates SQL commands necessary to UPDATE STATISTICS for one or more tables of a DBEnvironment.

Scope

SQLGEN Only

SQLGEN Syntax

```
>> [GEN[ERATE]]STAT[ISTICS]
Schema File Name or '/' to STOP command >> SchemaFileName
Owner Name >> OwnerName
Do you wish to specify Table Names for each Owner (n/y)?  $\left\{ \begin{array}{l} \text{N[O]} \\ \text{Y[ES]} \end{array} \right\}$ 
Table Name for Owner OwnerName >> TableName
```

Parameters

<i>SchemaFileName</i>	is the name of the schema file where SQLGEN places the generated commands.
<i>OwnerName</i>	is the name of the owner for whom you want to UPDATE STATISTICS.
NO or YES	respond NO to select all table names for the owner; respond YES to specify certain tables for the owner.
<i>TableName</i>	is the name of a table for which you wish to UPDATE STATISTICS.

Example

```
>> generate statistics

ALLBASE/SQL Command Generator for Statistics

Schema File Name or '/' to STOP command >> statpart

Please enter Owner Names. Type @ for all, ? for a list of
Owner Names, or RETURN to quit.

Owner Name >> purchdb
Do you wish to specify Table Names for each Owner (n/y)? n

Generating UPDATE STATISTICS for PURCHDB.INVENTORY
Generating UPDATE STATISTICS for PURCHDB.ORDERITEMS
.
.
.

Owner Name >> 

>>
```


GENERATE STATISTICS

Schema File Produced

```
UPDATE STATISTICS FOR TABLE PURCHDB.INVENTORY;  
UPDATE STATISTICS FOR TABLE PURCHDB.ORDERITEMS;  
UPDATE STATISTICS FOR TABLE PURCHDB.ORDERS;  
UPDATE STATISTICS FOR TABLE PURCHDB.PARTS;  
UPDATE STATISTICS FOR TABLE PURCHDB.SUPPLYPRICE;  
UPDATE STATISTICS FOR TABLE PURCHDB.VENDORS;  
COMMIT WORK;
```

GENERATE TABAUTH

Generates SQL commands necessary to recreate authorities for one or more tables of a DBEnvironment.

Scope

SQLGEN Only

SQLGEN Syntax

```
>> [GEN[ERATE]]TABAUTH
Schema File Name or '/' to STOP command >> SchemaFileName
Owner Name >> OwnerName
Do you wish to specify Table Names for each Owner (n/y)?  $\left\{ \begin{array}{l} \text{N}[\text{O}] \\ \text{Y}[\text{ES}] \end{array} \right\}$ 
Table Name for Owner OwnerName >> TableName
```

Parameters

<i>SchemaFileName</i>	is the name of the schema file where SQLGEN places generated commands.
<i>OwnerName</i>	is the name of the owner for whom you wish to generate table authorities.
NO or YES	respond NO to select all table names for the owner; respond YES to select certain table names for the owner.
<i>TableName</i>	is the name of a table for which you wish to generate authorities.

Example

```
>> generate tabauth

ALLBASE/SQL Command Generator for Table Authority

Schema File Name or '/' to STOP command >> tabapart

Please enter Owner Names. Type @ for all, ? for a list of
Owner Names, or RETURN to quit.

Owner Name >> @
Do you wish to specify Table Names for each Owner (n/y)? n

Generating Authority for PURCHDB.INVENTORY
Generating Authority for PURCHDB.ORDERITEMS
.
.
>>
```

GENERATE TABAUTH

Schema File Produced

```
GRANT SELECT,  
      INSERT,  
      DELETE,  
      UPDATE  
  ON PURCHDB.INVENTORY TO DBEUSERS;  
  
GRANT SELECT,  
      INSERT,  
      DELETE,  
      UPDATE  
  ON PURCHDB.INVENTORY TO PURCHASING;  
  
GRANT ALL  
  ON PURCHDB.INVENTORY TO PURCHDBMAINT;  
      .  
      .  
      .  
COMMIT WORK;
```

GENERATE TABLES

Generates SQL commands to recreate one or more tables of a DBEnvironment.

Scope

SQLGEN Only

SQLGEN Syntax

```
>> [ GEN[ERATE] ] TABLE[S]
Schema File Name or '// ' to STOP command >> SchemaFileName
Do you wish to generate associated Indexes (n/y)? { N[O] }
                                                    { Y[ES] }
Do you wish to generate associated Table Authority (n/y)? { N[O] }
                                                            { Y[ES] }
Owner Name >> OwnerName
Do you wish to specify Table Names for each Owner (n/y)? { N[O] }
                                                            { Y[ES] }
Table Name >> TableName
```

Parameters

<i>SchemaFileName</i>	is the name of the schema file where SQLGEN places generated commands.
NO or YES	respond NO if you do not want to generate associated indexes; respond YES if you want to generate associated indexes.
NO or YES	respond NO if you do not want to generate associated authorities; respond YES if you want to generate associated authorities.
<i>OwnerName</i>	is the name of the owner whose tables you wish to generate.
NO or YES	respond NO to select all table names for the owner; respond YES to select certain table names for the owner.
<i>TableName</i>	is the name of a table you wish to generate.

Description

- GENERATE TABLE builds CREATE TABLE commands to recreate one or more tables of the DBEnvironment. You can also recreate associated indexes and authorities for the specified tables.
- If a referential constraint exists, the referential constraint clause is omitted from the CREATE TABLE command and included in a subsequent ALTER TABLE command.

GENERATE TABLES

Example

```
>> generate tables

ALLBASE/SQL Command Generator for Tables

Schema File Name or '//' to STOP command >> partsch

Do you wish to generate associated Indexes (n/y)? y
Do you wish to generate associated Table Authority (n/y)? y

Please enter Owner Names. Type @ for all, ? for a list of
Owner Names, or RETURN to quit.

Owner Name >> purchdb
Do you wish to specify Table Names for each Owner (n/y)? y

Enter Table Names for Owner PURCHDB
Type @ for all, ? for a list of Table Names, or RETURN to quit.

Table Name for Owner PURCHDB >> inventory

Generating CREATE TABLE PURCHDB.INVENTORY

Generating CREATE INDEX on PURCHDB.INVENTORY

Generating Authority for PURCHDB.INVENTORY

Table Name for Owner PURCHDB >> 

Owner Name >> 

>>
```

Schema File Produced

```
CREATE PUBLIC TABLE PURCHDB.INVENTORY
(PARTNUMBER          CHAR( 16)  NOT NULL,
BINNUMBER            SMALLINT   NOT NULL,
QTYONHAND            SMALLINT,
LASTCOUNTDATE       CHAR( 8),
COUNTCYCLE          SMALLINT,
ADJUSTMENTQTY        SMALLINT,
REORDERQTY           SMALLINT,
REORDERPOINT         SMALLINT) IN WAREHFS;

CREATE UNIQUE INDEX INVPARTNUMINDEX
ON PURCHDB.INVENTORY
(PARTNUMBER);

GRANT SELECT,
      INSERT,
      DELETE,
      UPDATE
ON PURCHDB.INVENTORY TO DBEUSERS;

GRANT SELECT,
      INSERT,
      DELETE,
      UPDATE
ON PURCHDB.INVENTORY TO PURCHASING;

GRANT ALL
ON PURCHDB.INVENTORY TO PURCHDBMAINT;

GRANT SELECT
ON PURCHDB.INVENTORY TO PURCHMANAGERS;

GRANT SELECT,
      INSERT,
      DELETE,
      UPDATE
ON PURCHDB.INVENTORY TO WAREHOUSE;

REVOKE ALL
ON PURCHDB.INVENTORY FROM PUBLIC;

COMMIT WORK;
```

GENERATE TEMPSPACES

Generates SQL commands to recreate one or more tempspaces in a DBEnvironment.

Scope

SQLGEN Only

SQLGEN Syntax

```
>> [GEN[ERATE] ]TEMPSPACE[S]
Schema File Name or '/' to STOP command >> SchemaFileName
TEMPSPACE Name >> TempSpaceName
```

Parameters

SchemaFileName is the name of the schema file where SQLGEN places generated commands.

TempSpaceName is the name of the tempspace you wish to generate.

Description

- GENERATE TEMPSPACES builds CREATE TEMPSPACE commands to recreate one or more temporary spaces in the DBEnvironment.

Example

```
>> generate tempspace

ALLBASE/SQL Command Generator for Temp Space

Schema File Name or '/' to STOP command >> temps

Please enter TEMPSPACE Names. Type @ for all, ? for a list of
TEMPSPACE Names, or RETURN to quit.

TEMPSPACE Name >> @

Generating command to CREATE TEMPSPACE TEMPSPACE1

>>
```

Schema File Produced

```
/* This file was created with a user language environment of */
/* NATIVE-3000 */

CREATE TEMPSPACE TEMPSPACE1
  WITH MAXFILEPAGES = 1000,
  LOCATION = TEMP.SOMEACCT;

COMMIT WORK;
~
```

GENERATE VIEWAUTH

Generates SQL commands necessary to recreate authorities for one or more views of a DBEnvironment.

Scope

SQLGEN Only

SQLGEN Syntax

```
>> [GEN[ERATE]]VIEWAUTH
Schema File Name or '/' to STOP command >> SchemaFileName
Owner Name >> OwnerName
Do you wish to specify View Names for each Owner (n/y)? { N[O] }
{ Y[ES] }
View Name for Owner OwnerName >> ViewName
```

Parameters

<i>SchemaFileName</i>	is the name of the schema file where SQLGEN places the generated commands.
<i>OwnerName</i>	is the name of the owner for whom you wish to generate view authorities.
NO or YES	respond NO to specify all view names for the owner; respond YES to specify certain view names for the owner.
<i>ViewName</i>	is the name of a view for which you wish to generate authorities.

Example

```
>> generate viewauth

ALLBASE/SQL Command Generator for View Authority

Schema File Name or '/' to STOP command >> viewpart

Please enter Owner Names. Type @ for all, ? for a list of
Owner Names, or RETURN to quit.

Owner Name >> purchdb
Do you wish to specify View Names for each Owner (n/y)? n

Generating Authority for PURCHDB.PARTINFO
Generating Authority for PURCHDB.VENDORSTATISTICS

Owner Name >> 

>>
```


GENERATE VIEWAUTH

Schema File Produced

```
GRANT SELECT
  ON PURCHDB.PARTINFO TO DBEUSERS;

GRANT SELECT
  ON PURCHDB.PARTINFO TO PURCHDEMAINT;

GRANT SELECT
  ON PURCHDB.PARTINFO TO PURCHMANAGERS;
  .
  .
  .
COMMIT WORK;
```

GENERATE VIEWS

Generates SQL commands to recreate one or more views in a DBEnvironment.

Scope

SQLGEN Only

SQLGEN Syntax

```
>> [ GEN[ERATE] ]VIEW[S]
Schema File Name or '/' to STOP command >> SchemaFileName
Do you wish to generate associated View Authority (n/y)? { N[O] }
                                                    { Y[ES] }
Owner Name >> OwnerName
Do you wish to specify View Names for each Owner (n/y)? { N[O] }
                                                    { Y[ES] }
View Name for Owner OwnerName >> ViewName
```

Parameters

<i>SchemaFileName</i>	is the name of the schema file where SQLGEN places generated commands.
NO or YES	respond NO if you do not want to generate associated authorities; respond YES if you want to generate associated authorities.
<i>OwnerName</i>	is the name of the owner whose views you wish to generate.
NO or YES	respond NO to select all view names for the owner; respond YES to select certain view names for the owner.
<i>ViewName</i>	is the name of a view you wish to generate.

Description

- GENERATE VIEWS builds CREATE VIEW commands to recreate one or more views in the DBEnvironment. You can also recreate associated authorities for the specified views.

Example

```
>> generate views
ALLBASE/SQL Command Generator for Views
Schema File Name or '/' to STOP command >> partvsch
Do you wish to generate associated View Authority (n/y)? y
Please enter Owner Names. Type @ for all, ? for a list of
Owner Names, or RETURN to quit.
Owner Name >> purchdb
Do you wish to specify View Names for each Owner (n/y)? y
Enter View Names for Owner PURCHDB
```

GENERATE VIEWS

Type @ for all, ? for a list of View Names, or RETURN to quit.

View Name for Owner PURCHDB >> vendorstatistics

Generating CREATE VIEW PURCHDB.VENDORSTATISTICS

Generating Authority for PURCHDB.VENDORSTATISTICS

View Name for Owner PURCHDB >>

Owner Name >>

>>

Schema File Produced

```
SET OWNER PURCHDB;
CREATE VIEW PURCHDB.VENDORSTATISTICS
  (VENDORNUMBER,
   VENDORNAME,
   ORDERDATE,
   ORDERQUANTITY,
   TOTALPRICE) AS
SELECT PurchDB.Vendors.VendorNumber,PurchDB.Vendors.VendorName
, OrderDate, OrderQty, OrderQty*PurchasePrice FROM
PurchDB.Vendors, PurchDB.Orders, PurchDB.OrderItems WHERE
PurchDB.Vendors.VendorNumber = PurchDB.Orders.VendorNumber AND
PurchDB.Orders.OrderNumber = PurchDB.OrderItems.OrderNumber;

GRANT SELECT
  ON PURCHDB.VENDORSTATISTICS TO PURCHMANAGERS;

GRANT SELECT
  ON PURCHDB.VENDORSTATISTICS TO PURCHDBMAINT;

GRANT SELECT
  ON PURCHDB.VENDORSTATISTICS TO TOM@ASH;

GRANT SELECT,
  INSERT,
  DELETE,
  UPDATE
  ON PURCHDB.VENDORSTATISTICS TO DBEUSERS;
```

HELP

Displays and describes all SQLGEN commands.

Scope

SQLGEN Only

SQLGEN Syntax

$$\gg H[\text{ELP}] \left\{ \begin{array}{l} \textit{CommandName} \\ \textit{Keyword} \end{array} \right\}$$

Parameters

CommandName is the name of the command for which you want HELP.

Keyword is the name of a keyword for which you want HELP. Keywords include GENERAL, MAIN, OBJECT, and SUMMARY.

Description

- HELP with no parameters can be used to obtain general information about SQLGEN.
- HELP followed by a command name provides specific help for that command. In addition, help is also available for the following keywords:
 - GENERAL - provides general information about SQLGEN.
 - MAIN - provides command syntax and a summary of the HELP facility.
 - OBJECT - provides information about entering object names.
 - SUMMARY - provides a summary of all SQLGEN commands.

Example

```
>> help exit

SYNTAX      >> E[XIT]

OPERATION   EXIT terminates SQLGEN execution.  If the DBE session
            has not been ended before entering EXIT, an automatic
            RELEASE occurs before exiting SQLGEN.

EXAMPLE     >> EXIT

            DBEnvironment has been RELEASED.

            END OF PROGRAM
            :
```

RELEASE

Terminates the DBE session.

Scope

SQLGEN Only

SQLGEN Syntax

```
>>RELEASE
```

Description

RELEASE terminates the DBE session. You must use this command before issuing a STARTDBE for another DBEnvironment. When you exit SQLGEN, RELEASE is automatically invoked if the DBEnvironment session has not been terminated.

Example

```
>> release
```

```
The DBEnvironment has been RELEASED.
```

```
>>
```

SET ECHO_ALL OFF

Sets SQLGEN so as not to echo batch user input to the standard list.

Scope

SQLGEN Only

SQLGEN Syntax

```
>> SET ECHO_ALL OFF
```

Description

- After you issue this command, batch user input is not echoed to the standard list.

Example

```
>> SET ECHO_ALL OFF
```

SET ECHO_ALL ON

Sets SQLGEN to echo batch user input to the standard list.

Scope

SQLGEN Only

SQLGEN Syntax

```
>> SET ECHO_ALL ON
```

Description

- By default, or after you issue this command, batch user input is echoed to the standard list.

Example

```
>> SET ECHO_ALL ON
```

SET EDITOR

Defines the editor invoked when the EDITOR command is used.

Scope

SQLGEN Only

SQLGEN Syntax

```
>> SE[T]ED[ITOR]
```

Description

SET EDITOR is used to define the editor used by the EDITOR command (see Example 1 below). In addition to entering a SET EDITOR command, it is also possible to SET the EDITOR with a command argument when you run SQLGEN. The name of the editor must meet operating system naming conventions and must be preceded by 'ED=' in the INFO string (see example 2 below). The default editor is EDITOR.PUB.SYS.

Example 1

```
>> set editor
Current Editor:  EDITOR.PUB.SYS

Enter Editor Name, or '/' to keep current Editor
Editor >> TDP

Current Editor:  TDP.PUB.SYS

>>
```

Example 2

```
RUN SQLGEN.PUB.SYS;INFO="ED=TDP.PUB.SYS"
```

SET EXIT_ON_DBERR OFF

Causes SQLGEN in batch mode to continue when an SQL error is encountered.

Scope

SQLGEN Only

SQLGEN Syntax

```
>> SET EXIT_ON_DBERR OFF
```

Description

- By default, or after you issue this command, SQLGEN batch operation continues when an SQL error occurs.
- See the “DBA Tasks and Tools” chapter of this manual for information on checking JCWs when SQLGEN terminates.

Example

```
>> SET EXIT_ON_DBERR OFF
```

SET EXIT_ON_DBERR ON

Causes SQLGEN in batch operation to terminate immediately when an SQL error is encountered.

Scope

SQLGEN Only

SQLGEN Syntax

```
>> SET EXIT_ON_DBERR ON
```

Description

- After you issue this command, the system JCW is set to FATAL if an SQL error is encountered or to 0 if no SQL errors are encountered.
- See the “DBA Tasks and Tools” chapter of this manual for information on checking JCWs when SQLGEN terminates.
- By default, SQLGEN does not terminate when an SQL error is encountered.

Example

```
>> SET EXIT_ON_DBERR ON
```

SET SCHEMA

Defines the schema file name to be used by the generate commands.

Scope

SQLGEN Only

SQLGEN Syntax

```
>> SE[T]SCH[EMA ]  
Schema File Name or '/' to STOP command >> SchemaFileName
```

Parameters

SchemaFileName is the name of the schema file you wish to define.

Description

With SET SCHEMA you can define a schema file name before entering generate commands. The generate commands issued subsequently do not prompt for a schema file name. Instead they place commands into the schema file you have defined. This allows you to automatically place all generated commands in one schema file if you wish. Commands are appended to the end of the file as they are generated.

Example

```
>> set schema  
Current Schema File:  schema1  
  
Enter Schema File Name, RETURN to reset, or '/' to keep current name.  
  
Schema File Name or '/' to STOP command >> schema2  
  
Current Schema File:  schema2  
  
>>
```

Note

Because special schema files are required by the GENERATE LOAD command, you cannot use the SET SCHEMA command to define schema file names for the the GENERATE LOAD command.

STARTDBE

Starts the DBE session for SQLGEN execution.

Scope

SQLGEN Only

SQLGEN Syntax

```
>> START [DBE] [ DBEnvironmentName ]
```

Parameters

DBEnvironmentName is the name of the DBEnvironment to which you wish to connect.

Description

STARTDBE connects to the DBEnvironment and then checks to see if you have DBA authority for the DBEnvironment. If you do not, the DBEnvironment is released, since you cannot use SQLGEN to generate commands unless you have DBA authority. It is recommended that the DBEnvironment be shut down prior to issuing the STARTDBE command to ensure that system catalogs are not being modified during SQLGEN execution. Note that the DBEnvironment name can be entered as a parameter of the STARTDBE command or SQLGEN will prompt for the DBEnvironment name, as in the following example.

Example

```
>> startdbe  
DBEnvironment Name >> PartsDBE.SomeGrp.SomeAcct  
  
DBEnvironment successfully started.  
  
>>
```

:

Escape temporarily to the operating system and (optionally) execute a single operating system command.

Scope

SQLGEN Only

SQLGEN Syntax

```
>> : [CommandName];
```

Parameters

CommandName Name of an MPE operating system command.

Description

1. If you include a command name, control returns to SQLGEN as soon as the command has been executed.
2. If you omit the command name, use the *resume* command to return to SQLGEN.

Example

```
>> :LISTF,2;
```

```
ACCOUNT=  SOMEACCT      GROUP=  SOMEGRP
```

```
FILENAME  CODE  -----LOGICAL RECORD-----  ----SPACE----
          SIZE  TYP          EOF          LIMIT R/B  SECTORS  #X MX
PARTSDBE  PRIV  125W  FB           1           1  1         2  1  1
PARTSFO   PRIV  2048W FB          150          150  1        2416  5  5
PARTSLG1  PRIV  256W  FB           256          256  1         514  2  2
PARTSLG2  PRIV  256W  FB           256          256  1         514  2  2
```

```
>>
```

SQLMigrate

The following pages present SQLMigrate command syntax. A discussion of how to use SQLMigrate appears in the “DBA Tasks and Tools” chapter.

ADD DBEFILE

Associates a DBEFILE with the SYSTEM DBEFileSet.

Scope

SQLMigrate Only

SQLMigrate Syntax

```
ADD DBEFILE DBEFileName TO DBEFILESET SYSTEM;
```

Parameters

DBEFileName is the name of a DBEFile previously defined and created by the CREATE DBEFILE command.

Description

- When adding space to the SYSTEM DBEFileset, the first step is to CREATE a new DBEFile. The second step is to add the DBEFile to the SYSTEM DBEFileSet.
- Note that the DBEFile can be added only to the SYSTEM DBEFileSet.
- When this command is executed, you are modifying the DBEnvironment. Whenever you modify the DBEnvironment with the CREATE DBEFILE, ADD DBEFILE, or MIGRATE commands, a START DBE NEWLOG statement must be executed before you will be able to connect to the DBEnvironment. The MIGRATE command automatically executes a START DBE NEWLOG statement unless a SET NEWLOG OFF command has been issued.

Example

```
SQLMIGRATE=> create dbefile ThisDBEFile with pages=1000, name='ThisFile';  
SQLMIGRATE=> add dbefile ThisDBEFile to dbefileset SYSTEM;
```

CREATE DBEFILE

Defines and creates a DBEFile of TYPE MIXED.

Scope

SQLMigrate Only

SQLMigrate Syntax

```
CREATE DBEFILE DBEFileName WITH PAGES = DBEFileSize, NAME = 'SystemFileName';
```

Parameters

DBEFileName is the name to be assigned to the new DBEFile. Two DBEFiles in one DBEnvironment cannot have the same name. The maximum length for the DBEFile name is 20 bytes.

DBEFileSize specifies the number of 4096-byte pages in the new DBEFile.

SystemFileName identifies how the DBEFile is known to MPE. The file resides in the same group and account as the DBECon file.

Description

- When a DBEFile is created through SQLMigrate, the TYPE is MIXED. MIXED means that data and index pages can be stored in the new DBEFile.
- When adding space to the SYSTEM DBEFileSet, the first step is to CREATE a new DBEFile. The second step is to add the DBEFile to the SYSTEM DBEFileSet.
- When this command is executed, you are modifying the DBEnvironment. Whenever you modify the DBEnvironment with the CREATE DBEFILE, ADD DBEFILE, or MIGRATE commands, a START DBE NEWLOG statement must be executed before you will be able to connect to the DBEnvironment. The MIGRATE command automatically executes a START DBE NEWLOG statement unless a SET NEWLOG OFF command has been issued.
- Before using this command, you must issue either a PREVIEW or SHOW 'DBEnvironmentName' VERSION command to indicate which DBEnvironment is to receive the new DBEFile.
- After using this command, use the ADD DBEFILE command to add the newly created DBEFile to the SYSTEM DBEFileSet.

Example

```
SQLMIGRATE=> create dbefile ThisDBEfile with pages = 5, name = 'ThisFile';  
SQLMIGRATE=> add dbefile ThisDBEfile to dbefileset SYSTEM;
```

EXIT

Terminates SQLMigrate execution.

Scope

SQLMigrate Only

SQLMigrate Syntax

```
EXIT;
```

Description

- This command terminates the SQLMigrate session and returns you to the operating system.

Example

```
SQLMIGRATE=> exit;  
:
```

HELP

Displays SQLMigrate commands and descriptions.

Scope

SQLMigrate Only

SQLMigrate Syntax

```
HELP;
```

Description

- If you type HELP without parameters, a list of SQLMigrate commands is displayed.
- If you type HELP followed by a command name, then the syntax, an explanation of the command, and an example are returned.

Example

```
SQLMIGRATE=> help;
```

The SQLMIGRATE commands are:

ADD	MIGRATE	REPAIR
CREATE	PREVIEW	SET
EXIT	QUIT	SHOW

Help is available on any of the above commands by typing HELP followed by the command name (Example - HELP MIGRATE;).

For more information, please refer to the SQLMigrate section of the ALLBASE/SQL Database Administration Guide.

```
SQLMIGRATE=> help add;
```

```
ADD DBEFILE DBEfileName to DBEFILESET SYSTEM;
```

This command associates a DBEfile to the SYSTEM DBEfileSet.

When adding space to the SYSTEM DBEfileSet, the first step is to create a new DBEFILE. The second step is to ADD the DBEFILE to the SYSTEM DBEFILESET.

```
EXAMPLE: ADD DBEfile ThisDBEfile to DBEFILESET SYSTEM;
```

```
SQLMIGRATE=>
```

MIGRATE

Modifies a DBEnvironment to make it compatible with a different release of ALLBASE/SQL software.

Scope

SQLMigrate Only

SQLMigrate Syntax

```
MIGRATE 'DBEnvironmentName' { FORWARD [ TO 'Version' ]  
                                BACKWARD TO 'Version' };
```

Parameters

DBEnvironmentName identifies the DBEnvironment to be migrated.

Version indicates the version to which you want to migrate your DBEnvironment. *Version* is optional in forward migration. The default is to migrate the DBEnvironment to the most recent version supported by SQLMigrate. *Version* must be indicated for backward migration. Use the SHOW VERSIONS command to determine the value that may be entered as the version parameter to migrate a DBEnvironment to the desired release of ALLBASE/SQL software.

Description

- SQLMigrate cannot be used remotely.
- The following message is always displayed:

```
All stored sections have been invalidated.
```

```
Stored sections will be revalidated transparently when they are executed.  
Production should be faster if revalidation is done before that time.  
Revalidation can be accomplished by preprocessing the application programs  
that contain the stored sections.
```

- When a forward migration completes, SQLMigrate automatically issues a START DBE NEWLOG statement, unless NEWLOG has been set to OFF. You cannot connect to the DBEnvironment until the START DBE NEWLOG statement has been successfully performed.
- At the end of a backward migration, you must perform the START DBE NEWLOG statement with ISQL. SQLMigrate does not automatically issue the statement.

- System catalog views often change during migration. If authorities have been granted or user views have been defined on system catalog views, then create a schema file with SQLGEN before the migration. See the GENERATE VIEWAUTH and GENERATE VIEWS commands in the SQLGEN appendix of this manual for more information.
- For detailed instructions on how to perform a migration, see “Using SQLMigrate” in the “DBA Tasks and Tools” chapter of this manual.

Authorization

You must be the DBECreator or have SM capability to use the MIGRATE command.

Example 1

In the example below, a forward migration is shown.

```
SQLMIGRATE=> migrate 'PartsDBE' forward;

WARNING - Before continuing, please verify that a
          complete backup of the DBEnvironment exists. (DBWARN 20550)

Do you want to continue (yes,[no]) ? yes

MIGRATE BEGINNING (TUE, AUG 17, 1993, 10:16 AM)

All stored sections have been invalidated.

Stored sections will be revalidated transparently when they are executed.
Production should be faster if revalidation is done before that time.
Revalidation can be accomplished by preprocessing the application programs
that contain the stored sections.
```

At the end of this processing, the following message is issued:

```
MIGRATE SUCCEEDED with 1 WARNING(s) (TUE, AUG 17, 1993, 10:17 AM)
```

SQLMigrate automatically issues a START DBE NEWLOG command and echoes it to the screen:

```
START DBE NEWLOG BEGINNING (TUE, AUG 17, 1993, 10:17 AM)

START DBE 'PartsDBE' MULTI NEWLOG
DUAL LOG,
BUFFER = (100,24),
TRANSACTION = 5,
MAXIMUM TIMEOUT = NONE,
DEFAULT TIMEOUT = MAXIMUM,
RUN BLOCK = 37

LOG DBEFILE log1 AND log2 WITH PAGES = 256,
NAME = 'PartsLG1'
AND 'PartsLG2';

START DBE NEWLOG SUCCEEDED (TUE, AUG 17, 1993, 10:17 AM)

The DBEnvironment is ready to be accessed! If you desire archive mode
logging, you must run SQLUTIL and issue a STOREONLINE command.
SQLMIGRATE=>
```

MIGRATE

The previous message applies if you have TurboSTORE software on your system. If you do not have TurboSTORE, refer to the section “Static Backup Procedures in Archive Mode” in the “Backup and Recovery” chapter of this guide.

Example 2

In the example that follows a backward migration is shown.

```
SQLMIGRATE=> migrate 'PartsDBE' backward to 'F';  
  
WARNING - Before continuing, please verify that a complete backup of  
          the DBEnvironment exists. (DBWARN 20550)  
  
Do you want to continue (yes,[no]) ? yes  
  
MIGRATE BEGINNING (TUE, AUG 17, 1993, 10:27 AM)  
  
All stored sections have been invalidated.  
  
Stored sections will be revalidated transparently when they are executed.  
Production should be faster if revalidation is done before that time.  
Revalidation can be accomplished by preprocessing the application programs  
that contain the stored sections.  
  
WARNING - Views and stored sections created under later releases of  
          ALLBASE/SQL may not function correctly under the target version  
          of ALLBASE/SQL. (DBWARN 20580)  
  
Please drop all views created under later releases, and recreate them.  
  
Also drop all stored sections created under later releases, and either  
1) INSTALL backup installable module files that were created under the  
   target version of ALLBASE/SQL, or  
2) Preprocess programs again to obtain stored sections that are compatible  
   with the target version of ALLBASE/SQL.
```

At the end of this processing, the following message is issued:

```
MIGRATE SUCCEEDED with 2 WARNING(s) (TUE, AUG 17, 1993, 10:27 AM)
```

During backward migration, SQLMigrate does not automatically execute a START DBE NEWLOG statement.

```
Please note that you will not be able to use this DBEnvironment until  
you have run ISQL and issued a START DBE NEWLOG command. This action will  
create a new log file that is compatible with the target release. If you  
desire archive mode logging, you must run SQLUTIL and issue a STOREONLINE  
command after the START DBE NEWLOG command.
```

```
SQLMIGRATE=>
```

The previous message applies if you have TurboSTORE software on your system. If you do not have TurboSTORE, refer to the section, “Static Backup Procedures in Archive Mode,” in the “Backup and Recovery” chapter of this guide.

PREVIEW

Checks for errors that might occur during the migration of the DBEnvironment that would cause the migration to fail. PREVIEW checks for as many errors as it can before a migration, but it does not perform a “test migration.” It is still possible for the migration to fail after a preview, but usually this would only occur if the DBEnvironment is corrupt.

Scope

SQLMigrate Only

SQLMigrate Syntax

```
PREVIEW 'DBEnvironmentName' { FORWARD [ TO 'Version' ]
                              BACKWARD TO 'Version' };
```

Parameters

DBEnvironmentName identifies the DBEnvironment to be previewed.

Version indicates the version number to which you want to migrate your DBEnvironment. *Version* is optional in a forward preview. The default is to preview the DBEnvironment against the most recent version supported by SQLMigrate. *Version* must be indicated for a backward preview. Use the SHOW VERSIONS command to determine the value that may be entered as the version parameter to PREVIEW the migration of a DBEnvironment to the desired release of ALLBASE/SQL software.

Description

- The amount of free pages left in the SYSTEM DBEFileSet is calculated. If migration cannot be accomplished because there is not enough space, an error is returned along with an estimate of the additional number of pages needed.
- The PREVIEW BACKWARD command checks for database objects which contain functionality that is not supported under an old release. If they are detected, a warning message is issued that the database objects will be dropped during the migration process. You may want to save your table by issuing an ISQL UNLOAD command before migrating your DBEnvironment.
- The PREVIEW command is not a read only operation. If the PREVIEW command fails it may leave the DBEnvironment in an inconsistent state. Should the PREVIEW command fail, the DBEnvironment backup must be restored before the DBEnvironment can be accessed.

Authorization

You must be the DBECreator or have SM capability to use the PREVIEW command.

PREVIEW

Example 1

In the example below, a forward preview is shown. The warning is always given as a reminder to back up your DBEnvironment. The error indicates that there is not enough space for the migration to occur.

```
SQLMIGRATE=> preview 'PartsDBE' forward to 'G';

PREVIEW BEGINNING (TUE, AUG 17, 1993, 10:12 AM)

WARNING - Before continuing, please verify that a complete backup of
          the DBEnvironment exists. (DBWARN 20550)

Do you want to continue (yes,[no]) ? yes

ERROR - Insufficient space in the SYSTEM DBEFileset. (DBERR 22540)

DBEFile(s) containing 38 pages need to be added to the SYSTEM
DBEFileset before attempting the migration.

Use SQLMIGRATE to first CREATE a DBEFile, and then to ADD it to the
SYSTEM DBEFileSet. If you are concerned about the amount of additional
space being added, CREATE multiple small DBEFiles, rather than one large
one. After the migration is complete, perform an UPDATE STATISTICS on
SYSTEM.TABLE. Review SYSTEM.DBEFILE to determine the empty DBEFiles in
the SYSTEM DBEFileset, and remove them. Remember to leave about 20% of
the pages free for the temporary processing needs of ALLBASE/SQL.

PREVIEW FAILED with 1 ERROR(s) (TUE, AUG 17, 1993, 10:13 AM)

An attempt to MIGRATE this DBEnvironment will fail, and leave the
DBEnvironment in an inconsistent state. Please take corrective action
before attempting the migration.

SQLMIGRATE=>
```

At this point you will want to add space to the SYSTEM DBEFileSet. This procedure is shown here:

```
SQLMIGRATE=> CREATE DBEFILE partsfile3 WITH PAGES = 40, NAME ='prtfile3';

The DBEFile was SUCCESSFULLY CREATED in <PartsDBE>.

SQLMIGRATE=> ADD DBEFILE partsfile3 TO DBEFILESET SYSTEM;

The DBEFile was SUCCESSFULLY ADDED in <PartsDBE>.

SQLMIGRATE=>
```

Issue the PREVIEW command again to make sure that no other errors will be encountered during the migration:

```
SQLMIGRATE=> preview 'PartsDBE' forward to 'G';

PREVIEW BEGINNING (TUE, AUG 17, 1993, 10:13 AM)

WARNING - Before continuing, please verify that a complete backup of
          the DBEnvironment exists. (DBWARN 20550)

Do you want to continue (yes,[no]) ? yes

PREVIEW SUCCEEDED with 1 WARNING(s) (TUE, AUG 17, 1993, 10:13 AM)

The proposed migration should be successful.
```

SQLMIGRATE=>

Example 2

In the example below, a warning is issued that a particular table will be dropped because the column types were only supported under the current release of ALLBASE/SQL. This table was defined under the current release of ALLBASE/SQL and it took advantage of enhancements that exist only under the current release. Remember that the columns that are supported under the new release cannot be loaded into the old DBEnvironment. You should either omit them when the unload is performed or edit the unload file appropriately.

```
SQLMIGRATE=> preview 'PartsDBE' backward to 'F';

PREVIEW BEGINNING (TUE, AUG 17, 1993,  1:23 PM)

WARNING - Before continuing, please verify that a complete backup of
          the DBEnvironment exists. (DBWARN 20550)

Do you want to continue (yes,[no]) ? yes

WARNING - Table PURCHDB.PARTS contains the following columns which are not
          supported under the target version of ALLBASE/SQL (DBWARN
          22530):

          NONCASE

          This table will be dropped during the migration.

          Please NOTE that any other tables in your DBEnvironment that have
          REFERENTIAL CONSTRAINTS on this table will also be dropped during
          the migration. Please verify that a complete backup of this
          DBEnvironment exists to insure that table(s) are not lost.

WARNING - View PURCHDB.PARTINFO is defined on a table or view that will be
          dropped during the migration (DBWARN 22700). This view will
          also be dropped. The SELECTSTRING from SYSTEM.VIEWDEF for this
          view is:

          SELECT PurchDB.SupplyPrice.PartNumber, PurchDB.Parts.PartName,
          PurchDB.SupplyPrice.VendorNumber, PurchDB.Vendors.VendorName,
          PurchDB.SupplyPrice.VendPartNumber,
          PurchDB.SupplyPrice.UnitPrice, PurchDB.SupplyPrice.DiscountQty
          FROM PurchDB.Parts, PurchDB.SupplyPrice, PurchDB.Vendors WHERE
          PurchDB.SupplyPrice.PartNumber = PurchDB.Parts.PartNumber AND
          PurchDB.SupplyPrice.VendorNumber = PurchDB.Vendors.VendorNumber;

PREVIEW SUCCEEDED with 3 WARNING(s) (TUE, AUG 17, 1993,  1:23 PM)

The proposed migration should be successful.

SQLMIGRATE=>
```

QUIT

Terminates SQLMigrate execution.

Scope

SQLMigrate Only

SQLMigrate Syntax

```
QUIT;
```

Description

- This command terminates the SQLMigrate session and returns you to the operating system.

Example

```
SQLMIGRATE=> quit;  
:
```

REPAIR

Checks for and repairs migration corruption that may have been caused by a previous release of ALLBASE/SQL. You should only issue the REPAIR command when asked to do so by a qualified Hewlett-Packard representative.

Scope

SQLMigrate Only

SQLMigrate Syntax

```
REPAIR 'DBEnvironmentName';
```

Parameters

DBEnvironmentName identifies the DBEnvironment to be repaired.

Description

- This command will check for and repair migration corruption that may have been caused by a previous release of SQLMigrate. Migration corruption occurs when a release of SQLMigrate has a defect, so it does not modify a DBEnvironment correctly when the MIGRATE command is issued.
- The REPAIR command may only be used when no other user is accessing the DBEnvironment.

At the end of the REPAIR command, one of the following two messages will be issued:

```
REPAIR SUCCEEDED: No migration corruption has been detected.  
REPAIR SUCCEEDED: Migration corruption has been detected and repaired.
```

Example

```
SQLMIGRATE=> repair 'PartsDBE';  
  
REPAIRED: The INDEX base table has been corrected.  
REPAIRED: The VIEWDEF base table has been corrected.  
REPAIR SUCCEEDED: Migration corruption has been detected and repaired.  
  
SQLMIGRATE=>
```

SET

Sets operational options.

Scope

SQLMigrate Only

SQLMigrate Syntax

```
SET Option OptionValue;
```

Parameters

<i>Option OptionValue</i>	identifies the option and a value for it. The available option and values are:
AUTO_NO ON	Initial setting is OFF If ON, there is no need to answer interactive SQLMigrate questions. Sets all answers to NO.
AUTO_NO OFF	If OFF, all interactive SQLMigrate questions must be answered.
BACKUP_MADE ON	Initial setting is OFF If ON, there is no need to answer interactive SQLMigrate questions about backup.
BACKUP_MADE OFF	If OFF, all interactive SQLMigrate questions about backup must be answered.
ECHO_ALL ON	Initial setting is OFF. If ON, batch user input is echoed to the standard list.
ECHO_ALL OFF	If OFF, batch user input is not echoed.
EXIT_ON_DBERR ON	Initial setting is OFF. If ON, SQLMigrate sets the system JCW to FATAL and terminates immediately if an SQL is encountered. The system JCW is set to 0 if no SQL errors are encountered.
EXIT_ON_DBERR OFF	If OFF, SQLMigrate does not immediately terminate if an SQL error is encountered. The system JCW is not set.
NEWLOG ON	Initial setting is ON If ON, SQLMigrate automatically issues a START DBE NEWLOG statement after the MIGRATE FORWARD command has finished executing.
NEWLOG OFF	If OFF, you must exit from SQLMigrate, run ISQL, and issue a START DBE NEWLOG statement. You cannot use your DBEnvironment after it has been migrated until the START DBE NEWLOG statement executes successfully.

VERBOSE ON Initial setting is ON
 If ON, detailed error and warning messages are displayed.
 Suggested actions are often given along with the message.

VERBOSE OFF If OFF, error and warning messages are short.

Description

The options `AUTO_NO` and `BACKUP_MADE` can be used when using a batch job to migrate one or more DBEnvironments. When backups have been made and `BACKUP_MADE` is set to ON and `AUTO_NO` is set to ON, only unexpected situations will cause migration to fail. Review the job listing to make sure all DBEnvironments were migrated.

Example 1

In the following example, the message returned is detailed because `VERBOSE` is set to ON.

```
SQLMIGRATE=> set verbose on;

SQLMIGRATE=> preview 'PartsDBE' forward;

ERROR - Insufficient space in the SYSTEM DBEFileSet (DBERR 22540).

DBEFile(s) containing 5 pages need to be added to the SYSTEM
DBEFileSet before attempting migration.

If you are concerned about the amount of additional space being added,
CREATE multiple small DBEFiles, rather than one large one. After the
migration is complete, perform an UPDATE STATISTICS on SYSTEM.TABLE.
Review SYSTEM.DBFILE to determine the empty DBEFiles in the SYSTEM
DBEFileSet, and remove them. Remember to leave about 20% of the pages
free for the temporary processing needs of HP SQL.
```

Example 2

In the following example, the message is shorter because `VERBOSE` is set to OFF.

```
SQLMIGRATE=> set verbose off;

SQLMIGRATE=> migrate 'PartsDBE' forward;

ERROR - Insufficient space in the SYSTEM DBEFileSet (DBERR 22540).

DBEFile(s) containing 5 pages needs to be added to the SYSTEM
DBEFileSet before attempting the migration.

SQLMIGRATE=>
```

SHOW 'DBEnvironmentName' VERSION

Displays the current version number and startup parameters of the DBEnvironment specified in the command.

Scope

SQLMigrate Only

SQLMigrate Syntax

```
SHOW 'DBEnvironmentName' VERSION;
```

Parameters

DBEnvironmentName identifies the DBEnvironment to be previewed.

Example

In the example below, the version of the DBEnvironment is F:

```
SQLMIGRATE=> show 'PartsDBE' version;

VERSION          RELEASE

F                HP36216-02A.F

CURRENT STARTUP PARAMETERS:
SINGLE USER MODE
DUAL LOG,
BUFFER = (50,50)
TRANSACTION = 2
MAXIMUM TIMEOUT = 3600 SECONDS,
DEFAULT TIMEOUT = 30 SECONDS,
RUN BLOCK = 37

LOG DBEFILE DBELOG1 AND DBELOG2 WITH PAGES = 250,
NAME = 'DBEFile0'
AND 'DBEFile1';
SQLMIGRATE=>
```

SHOW VERSIONS

Displays the DBEnvironment versions and corresponding target releases of ALLBASE/SQL supported by the current release of SQLMigrate.

Scope

SQLMigrate Only

SQLMigrate Syntax

```
SHOW VERSIONS;
```

Description

This command lists the DBEnvironment version and corresponding ALLBASE/SQL release numbers which work with SQLMigrate.

Example

The following example shows supported target DBEnvironment versions and corresponding ALLBASE/SQL releases:

```
SQLMIGRATE=> show versions;
```

VERSION	RELEASE
E	HP36216-02A.E
F	HP36216-02A.F
G	HP36216-02A.G

```
SQLMIGRATE=>
```

:

Escape temporarily to the operating system and execute a single operating system command.

Scope

SQLMigrate Only

SQLMigrate Syntax

```
>> : CommandName;
```

Parameters

CommandName Name of an MPE operating system command.

Description

- Control returns to SQLMigrate as soon as the command has been executed.

Example

```
>> :LISTF,2;
```

```
ACCOUNT= SOMEACCT      GROUP= SOMEGRP
```

```
FILENAME  CODE  -----LOGICAL RECORD-----  ----SPACE----
          SIZE  TYP      EOF      LIMIT R/B  SECTORS  #X MX
PARTSDBE  PRIV   125W  FB        1         1   1         2   1  1
PARTSFO   PRIV  2048W  FB       150       150   1       2416  5  5
PARTSLG1  PRIV   256W  FB       256       256   1         514  2  2
PARTSLG2  PRIV   256W  FB       256       256   1         514  2  2
```

```
>>
```

SQLAudit

The following pages describe the syntax of the SQLAudit commands. You can find a discussion of how to use SQLAudit in the “DBA Tasks and Tools” chapter.

AUDIT

This command processes committed transactions for appropriate audit log records and places the results in a file for user viewing.

Scope

SQLAudit Only

SQLAudit Syntax

```
SQLAudit >> AUDIT
Beginning Audit Point File >> BEGINFILE
Ending Audit Point File >> ENDFILE
Results File to be generated >> RESULTFILE

Do you wish to specify Partition Numbers (n/y) >> { n[o] }
                                                    { y[es] }

Partition Number >> {
                    ?
                    @
                    PartNumber
                    DEF[ INITION ]
                    AUTH[ ORIZATION ]
                    STOR[ AGE ]
                    SECT[ ION ]
                    }
```

Parameters

<i>BeginFile</i>	is the name of the file containing the beginning audit point information. If only a carriage return is entered, SQLAudit tries to find the first committed transaction for each partition specified.
<i>EndFile</i>	is the name of the file containing the ending audit point information. If a carriage return is entered, SQLAudit uses the current audit point information for the DBEnvironment.
<i>ResultFile</i>	is the name of the file that will be generated. This file contains the output of the AUDIT command in a user-readable format. If a carriage return is entered, SQLAudit sends the generated results to the standard output. If the file specified already exists, you will be prompted to either purge, overwrite, or append the existing file. Enter <i>ResultFile</i> in all uppercase letters if you wish to see the file from within SQLAudit.
NO or YES	respond NO to select all partitions; respond YES if you wish to specify your own list of partitions.
?	displays a list of partitions having committed transactions.
@	selects all partitions.
<i>PartNumber</i>	is the number for a partition that you wish to audit.

Description

- The default is to process audit log records (for all partitions) generated between the beginning and ending audit points. This makes the list of partitions to be audited equal to the number of partitions that had transactions committed between the beginning and ending audit points.
- You may specify a set of partitions to be audited if you want to narrow down the number of records to process. When a list of partitions is specified, only transactions changing data in the given set of partitions are audited.
- When specifying a list of partitions, only one partition is allowed per input line. At any time you can enter an at sign (@) at the prompt to select all partitions. Entering a question mark displays a list of partitions that have committed transactions. Entering a carriage return ends the Partition Number prompt.
- Partitions must be specified by number for user defined partitions. Examples of user defined partitions are DEFAULT PARTITION, COMMENT PARTITION, and partitions created through the CREATE PARTITION command.
- System defined partitions such as DEFINITION, AUTHORIZATION, STORAGE, and SECTION (created through the START DBE command) are specified by name. These partitions are defined when the user specifies AUDIT ELEMENTS of DEFINITION, AUTHORIZATION, STORAGE, or SECTION in the START DBE command.
- If no beginning audit point is specified, SQLAudit attempts to process transactions starting from the beginning of log history. When this happens, if SQLAudit is unable to find the first transaction that has changed a given partition, a warning is returned and the partition is removed from the list of partitions to be processed.
- If no ending audit point is specified, SQLAudit determines the audit point information as of the last log record written and uses this for the ending audit point.
- If no result file is specified, SQLAudit automatically sends all generated results of the audit to standard output.
- If an error occurs while writing records to the result file, SQLAudit creates a file to hold the generated audit point information. This recovery file can be used just like any other audit point file (for example, as the beginning audit point file), except for use with the LOCK AUDITPOINT command. This audit point information can be particularly useful if errors such as FILE SYSTEM FULL are encountered while writing to the result file. The default name of this file is SQLAUREC but can be changed through the command SET RECOVERFILE. If SQLAudit needs to create this file, an error will occur if the file already exists.

Authorization

DBA authorization is required in order to use this command.

AUDIT

Example

```
SQLAudit >> AUDIT

Beginning Audit Point File >> STARTPT
Ending Audit Point File >> STOPPT
Result File to be generated >> RESULTS
Do you wish to specify Partition Numbers (n/y) >> yes

Please enter Partition Numbers or System Partition Names. Type @ for all,
? for a list of Partitions, or RETURN to finish. Valid System
Names are DEF[INITION], AUTH[ORIZATION], STOR[AGE], and SEC[TION].

Partition Number >> 1
Partition Number >> 2
Partition Number >>  

Generating Results ...

Records Audited: 10000   Records Generated: 10000
Records Audited: 20000   Records Generated: 20000
Records Audited: 24523   Records Generated: 24523

Finished Generating Results.

SQLAudit >>
```

The result file generated is in the format of ASCII records that can be viewed by the user. An example of such a file is shown below:

```
***** SQLAUDIT: GENERATING RESULTS *****
Creator: DBAUSER           Creation Time: 1993-05-11 14:22:16.531
BEGIN
INSERT (2) USER1.TABLE1 (123, 'test data', NULL, 1.23)
UPDATE (2) USER1.TABLE1 (123, 'test data', NULL, 1.23) ((3) 0x0000123C)
COMMIT User: USER1       Audit Name: MDBE1   Label: TRANS1
      Time: 1993-05-11 10:15:00.123
BEGIN
DELETE (2) USER1.TABLE1 (123, 'test data', 0x0000123C, 1.23)
COMMIT User: USER1       Audit Name: MDBE1   Label: TRANS2
      Time: 1993-05-11 10:15:01.455
End of File
```

EDITOR

Invokes the currently set editor.

Scope

SQLAudit only.

SQLAudit Syntax

```
SQLAudit >> ED[EDITOR]
```

Description

- This command invokes the currently set editor from within SQLAudit. The default editor is EDITOR.PUB.SYS. Use the SET EDITOR command to redefine the current editor.

Authorization

Anyone can issue the EDITOR command.

Example

```
SQLAudit >> editor
HP32201A.07.20 EDIT/3000 MON, JUN 14, 1993,  5:06 PM
(C) HEWLETT-PACKARD CO. 1990
/t results
/l all
1 *****  SQLAUDIT: GENERATED RESULTS *****
2 BEGIN
3 INSERT (2) ...
4 UPDATE (1) ...
.
.
/e
SQLAudit >>
```

EXIT

Exits from SQLAudit.

Scope

SQLAudit only.

SQLAudit Syntax

```
SQLAudit >> E[XIT]
```

Description

- This command terminates SQLAudit execution.
- This command is equivalent to the QUIT command.
- If you have not terminated the DBEnvironment session (through the SET DBENVIRONMENT OFF command), SQLAudit automatically terminates the DBE session before exiting.

Authorization

Anyone can issue the EXIT command.

Example

```
SQLAudit >> EXIT  
:
```

GET AUDITPOINT

Determines the current audit point information and places it in a file.

Scope

SQLAudit Only

SQLAudit Syntax

```
SQLAudit >> GE[T]AUDIT[POINT]
Audit Point File >> FileName
Lock Log for Audit Point (n/y) >> { n[o] }
                                     { y[es] }
Display Audit Point Information (n/y) >> { n[o] }
                                           { y[es] }
```

Parameters

<i>FileName</i>	is the name of the audit point file to be created. If the file already exists, you are prompted whether to purge and overwrite the file. Enter <i>FileName</i> in all uppercase letters if you wish to see the file from within SQLAudit.
NO or YES	respond NO if you do not want to lock the log files; respond YES if you do want to lock the log files with the current audit point information.
NO or YES	respond NO if you do not want to view the audit information; respond YES if you do want to view the current audit information.

Description

- This command is used to retrieve the current audit point information and place it into a file. This command can be used while the DBEnvironment is in use.
- Audit information is retrieved for all partitions in the DBEnvironment that have had transactions committed. Identifying information is also placed in the audit point file.
- When the file is created, the file permissions are set to restrict access to other users. In other words the file is created such that it is readable only by the user who created it. If the DBA who created the file wishes to allow others to access the file, it will be up to the DBA to change the files' permissions.
- If desired, you can lock the log files according to the current audit point information. If locked, all transactions committed after this command are protected from being overwritten until the lock is changed or removed. Log locks can be changed through the commands GET AUDITPOINT and LOCK AUDITPOINT. Log locks can be removed through the command UNLOCK AUDITPOINT, or using the SQL command START DBE NEWLOG.
- If you specify that the log files are to be locked, any previously defined log locks are replaced by the new audit point log lock. In other words, only one audit point can be locked in the log files at any one time.

GET AUDITPOINT

- You must be connected to the DBEnvironment to use this command. If you have not connected to the DBE (using the SET DBENVIRONMENT command), SQLAudit issues a warning and automatically issues the SET DBENVIRONMENT command.

Authorization

You must have DBA authorization in order to use this command.

Example

```
SQLAudit >> get auditpoint

Audit Point File >> STARTPT
Lock Log for Current Audit Point (n/y) >> yes
Display Current Audit Point Information (n/y) >> no

SQLAudit >>
```

Refer to the SHOW AUDITPOINT command for an example of the display of current audit point information.

HELP

Displays the help text of an SQLAudit command.

Scope

SQLAudit only.

SQLAudit Syntax

```
>> H[ELP] [CommandName or Keyword]
```

Parameter

CommandName is the name of the command whose syntax you want displayed.

Keyword is the name of the subject you want displayed. Valid keywords are MAIN, SUMMARY, and GENERAL.

Description

- If a *CommandName* or *Keyword* is included, SQLAudit displays the help text and returns you to the SQLAudit prompt.
- If HELP is entered by itself, SQLAudit displays the MAIN help screen and prompts you for the *CommandName* or *Keyword* for which you want additional help.
- Typing // at any time returns you to the SQLAudit prompt.

Authorization

Anyone can issue the HELP command.

Example

```
SQLAudit >> HELP GET AUDITPOINT

SYNTAX    GE[T] AUDIT[POINT]

OPERATION GET AUDITPOINT will determine the current audit point
          information from the DBEnvironment and place it in a file
          for the user. This audit point file can then be used by
          other SQLAudit commands (such as AUDIT or LOCK AUDITPOINT).

EXAMPLE   SQLAudit >> GET AUDITPOINT

          Audit Point File >> STARTPT
          Lock Log for Audit Point (n/y) >> NO
          Display Audit Point Information (n/y) >> NO

SQLAudit >>
```

LOCK AUDITPOINT

Locks the log files according to transaction information contained in an audit point file.

Scope

SQLAudit Only

SQLAudit Syntax

```
SQLAudit >> LO[CK]AUDIT[POINT]
Audit Point File >> FileName
Display Audit Point Information (n/y) >> { n[o]
                                           y[es] }
Lock Log for Audit Point (n/y) >> { n[o]
                                   y[es] }
```

Parameters

<i>FileName</i>	is the name of the file containing audit point information. This file must have been previously created by the command GET AUDITPOINT.
NO or YES	respond NO if you do not want to display audit point information; respond YES to confirm that you do want to display audit point information.
NO or YES	respond NO if you do not want to lock the log files; respond YES to confirm that you do want to lock the log files.

Description

- This command is used to lock the audit point (determined from the audit point file) in the DBEnvironment log files. When this happens, all transactions committed after the audit point are protected from being overwritten until the log lock is changed or removed. Log locks can be changed through the commands GET AUDITPOINT and LOCK AUDITPOINT. Log locks can be removed through the command UNLOCK AUDITPOINT, or using the SQL command START DBE NEWLOG.
- This command automatically replaces any currently held lock on the log files. Only one log lock can exist on the DBEnvironment log files at any one time.
- If the lock cannot be acquired (for example, if you have specified an audit point that is no longer valid), the old log lock remains.
- You must be connected to the DBEnvironment to use this command. If you have not connected to the DBE (using the SET DBENVIRONMENT command), SQLAudit issues a warning and automatically issues the SET DBENVIRONMENT command.

Authorization

You must have DBA authorization in order to use this command.

Example

```
SQLAudit >> lock auditpoint  
  
Audit Point File >> STARTPT  
Display Audit Point Information (n/y) >> no  
Lock Log for Audit Point (n/y) >> yes
```

MODIFY AUDITPOINT

Modifies the current audit point information for the DBEnvironment.

Scope

SQLAudit Only

SQLAudit Syntax

```
SQLAudit >> MOD[IFY] AUDIT[POINT]
Audit Point File >> FileName
Partition Number >> {
    ?
    @
    PartNumber
    DEF[INITION]
    AUTH[ORIZATION]
    STOR[AGE]
    SECT[ION]
}
Modify Current Audit Point Information (n/y) >> { n[o]
                                                y[es] }
```

Parameters

<i>FileName</i>	is the name of the audit point file containing unloaded SCR (or transaction) information. This file would have been created by running SQLAudit on the master DBEnvironment. If no audit point filename is entered, SQLAudit will delete transaction (SCR) information for the specified list of partitions.
<i>PartNumber</i>	is the number for a partition for which you wish to modify the transaction information.
NO or YES	respond NO if you do not want to modify the current audit point information in the DBE; respond YES if you do want to modify the current audit point information in the DBE.

Description

- This command should only be used for ALLBASE/REPLICATE DBEnvironments, and it should only be necessary during hard resynchronization of a given set of partitions. This command should not be necessary for audit only DBEnvironments that are not using ALLBASE/REPLICATE.
- This command is used to modify the current audit point information to reflect new transaction information for the specified set of partitions. This command can be used in two different ways. The first way it can be used is to replace the transaction information for the given set of partitions with new transaction information from the audit point file entered. The second way it can be used is to delete transaction information for a given set of partitions. This command can be used while the DBEnvironment is in use.

- If the user enters an audit point filename, SQLAudit will replace the DBEnvironment transaction (SCR) information with information from the audit point file (for the given set of partitions). This file is assumed to have been created by running SQLAudit against the master DBEnvironment (using the GET AUDITPOINT command). If no transaction information is found in the audit point file for a specified partition, the transaction information for the partition will be deleted from the DBEnvironment. The audit point information from the file will be used to list the current set of partitions.
- If the user did not enter an audit point filename, SQLAudit will delete all transaction information for the user specified list of partitions. If no audit point filename is entered (the user just pressed RETURN), SQLAudit will retrieve the current audit point information from the DBEnvironment. This audit point information will be used to list the current set of partitions.
- You must be connected to the DBEnvironment to use this command. If you have not connected to the DBE (using the SET DBENVIRONMENT command), SQLAudit will issue a warning and automatically issue the SET DBENVIRONMENT command on your behalf.

Authorization

You must have DBA authorization in order to use this command.

Example

```
SQLAudit >> modify auditpoint

Audit Point File >> MasterPt

System Partitions are DEF[INITIOW], AUTH[ORIZATIOW], STOR[AGE] or SECT[IOW].
Please enter Partition Numbers or System Partitions. Type @ for all,
? for a list of Partitions, or RETURN to quit.

Partition Number >> 1
Partition Number >> 2
Partition Number >>  
Modify Current Audit Point Information (n/y) >> yes

Current audit point information has been modified.

SQLAudit >>

SQLAudit >> modify auditpoint

Audit Point File >>

Audit point information will be deleted for the specified partitions.

System Partitions are DEF[INITIOW], AUTH[ORIZATIOW], STOR[AGE] or SECT[IOW].
Please enter Partition Numbers or System Partitions. Type @ for all,
? for a list of Partitions, or RETURN to quit.

Partition Number >> 3
Partition Number >>  
Modify Current Audit Point Information (n/y) >> yes

Current audit point information has been modified.

SQLAudit >>
```

QUIT

Quits and exits from SQLAudit.

Scope

SQLAudit only.

SQLAudit Syntax

```
SQLAudit >> Q[UIT]
```

Description

- This command terminates SQLAudit execution.
- This command is equivalent to the EXIT command.
- If you have not terminated the DBEnvironment session (through the SET DBENVIRONMENT OFF command), SQLAudit automatically terminates the DBE session before quitting.

Authorization

Anyone can issue the QUIT command.

Example

```
SQLAudit >> QUIT  
:
```

SET

Displays current values of SQLAudit options.

Scope

SQLAudit only.

SQLAudit Syntax

```
SQLAudit >> SE[T]
```

Description

- This command displays the current values for SQLAudit options.

Authorization

Anyone can issue the SET command.

Example

```
SQLAudit >> SET

ECHO_[ALL]          OFF
EDITOR              EDITOR.PUB.SYS
EXIT[_ON_DBERR]    OFF
DBEN[VIRONMENT]    OFF
REC[OVERFILE]      SQAUREC

SQLAudit >>
```

SET DBENVIRONMENT

Establishes or releases the connection to the DBEnvironment.

Scope

SQLAudit Only

SQLAudit Syntax

```
SQLAudit >> SE[T]DBEN[VIRONMENT] { DBEnvironmentName }
                                     { OFF }

SQLAudit >> SE[T]DBEN[VIRONMENT]
DBEnvironment Name >> { DBEnvironmentName }
                       { OFF }
```

Parameters

DBEnvironmentName specifies the name of the DBEnvironment with which you wish to establish a connection.

OFF terminates (or releases) your connection to the DBEnvironment.

Description

- This command is used to establish or release a connection with the DBEnvironment.
- *DBEnvironmentName* or OFF can be entered as part of the command line, or they can be entered in prompt mode.
- When OFF is specified, the current DBEnvironment (if set) is released, terminating the database session. You may issue the SET DBENVIRONMENT command to connect to a second DBEnvironment without issuing a SET DBEN OFF in between the two commands. In this case, SQLAUDIT automatically releases the first DBEnvironment before connecting to the second. When you exit SQLAudit without specifying a SET DBEN OFF, the current DBEnvironment is automatically released.

Authorization

Only CONNECT authority is required to issue this command. However, no auditing can be performed unless you also have DBA authority.

Example

```
SQLAudit >> set dben PartsDBE

SQLAudit >> set dbenvironment

DBEnvironment Name >> OFF

SQLAudit >>
```

SET ECHO_ALL

Enables or disables echoing of user input to the standard output file.

Scope

SQLAudit Only

SQLAudit Syntax

```
SQLAudit >> SE[T]ECHO_[ALL] { OFF }
                        { ON }
SQLAudit >> SE[T]ECHO_[ALL]
Option setting (OFF/ON) >> { OFF }
                        { ON }
```

Parameters

OFF or ON respond OFF to disable echoing your input to the standard output; respond ON to enable echoing your input to the standard output.

Description

- This command is used to enable or disable echoing of user input to the standard output file. This option is primarily useful in executing commands from a batch script or job stream.
- OFF or ON can be entered as part of the command line, or they can be entered in prompt mode.
- The default setting for this option is OFF (user input will not be echoed to the standard output file).

Authorization

No authorization is required to use this command.

Example

```
SQLAudit >> set echo_all on
SQLAudit >> set echo_all
Option setting (OFF/ON) >> off
SQLAudit >>
```

SET EDITOR

Defines the current editor.

Scope

SQLAudit Only

SQLAudit Syntax

```
SQLAudit >> SE[T]ED[ITOR] EditorName
```

```
SQLAudit >> SE[T]ED[ITOR]  
Editor Name or // to STOP command >> EditorName
```

Parameters

EditorName Name of the editor which you wish to use.

Description

- This command is used to define the current editor to be used by SQLAudit. The current editor can then be invoked by using the EDITOR command.
- The default setting for this option is EDITOR.PUB.SYS.

Authorization

No authorization is required to use this command.

Example

```
SQLAudit >> set editor tdp.pub.sys  
  
SQLAudit >> set editor  
  
Current Editor: tdp.pub.sys  
  
Enter Editor Name or '/' to keep current Editor  
  
Editor Name or '/' to STOP command >> qedit.pub.sys  
  
Current Editor: qedit.pub.sys  
  
SQLAudit >>
```

SET EXIT_ON_DBERR

Allows you to terminate or continue execution when a database error occurs.

Scope

SQLAudit only.

SQLAudit Syntax

```
SE[T]EXIT [_ON_DBERR] { OFF }
                        { ON  }
SE[T]EXIT [_ON_DBERR]
Option setting (OFF/ON) >> { OFF }
                             { ON  }
```

Description

- When EXIT_ON_DBERR is set to ON, SQLAudit terminates execution when a database error is encountered.
- When EXIT_ON_DBERR is set to OFF, SQLAudit continues execution when a database error is encountered.
- The default setting for this option is OFF (continue execution if a database error occurs).

Authorization

Anyone can issue the SET EXIT_ON_DBERR command.

Example

```
SQLAudit >> set exit_on_dberr on
```

```
SQLAudit >> set exit_on_dberr
```

```
Option Setting (OFF/ON) >> on
```

```
SQLAudit >>
```

SET RECOVERFILE

Defines the recovery file name which can be used by the AUDIT command.

Scope

SQLAudit Only

SQLAudit Syntax

```
SQLAudit >> SE[T]REC[OVERFILE] FileName
```

```
SQLAudit >> SE[T]REC[OVERFILE]  
Recovery File Name or '/' to STOP command >> FileName
```

Parameters

FileName is the name of the recovery file the AUDIT command is to use. Enter *FileName* in all uppercase letters, so that you can see the file within SQLAudit.

Description

- This command is used to define the name of the recovery file to be used by the AUDIT command. This file will only be created if the AUDIT command encounters an error while writing to the results file. When this file is created, SQLAudit places audit point information in the file to reflect the set of transactions that have been audited.
- If the file already exists at the time that the AUDIT command attempts to create it, an error occurs and the file is not created.
- The default file name setting for this option is `SQLAUREC`.

Authorization

No authorization is required to use this command.

Example

```
SQLAudit >> set recoverfile RECFILE.RECGRP  
  
SQLAudit >> set recoverfile  
  
Current Recovery File: RECFILE.RECGRP  
  
Enter Recovery File Name or '/' to keep current name  
  
Recovery File Name or '/' to STOP command >> SQLAUREC.RECGRP  
  
Current Recovery File: SQLAUREC.RECGRP  
  
SQLAudit >>
```

SHOW AUDITPOINT

Displays information contained in an audit point file.

Scope

SQLAudit Only

SQLAudit Syntax

```
SQLAudit >> SHOW AUDIT[POINT]
Audit Point File >> FileName
```

Parameters

FileName is the name of the file containing audit point information. This file must have been previously created by the command GET AUDITPOINT.

Description

- This command is used to display the audit point information contained within the indicated file.

Authorization

No authorization is required to use this command, though you must have read access to the audit point file.

Example

```
SQLAudit >> show auditpoint
```

```
Audit Point File >> ENDPT
```

```
Creator Name: dbauser
Transactions Lockpoint: Log Sequence No (184) Page No (837)
DBEnvironment Lockpoint: Log Sequence No (176) Page No (551)
No. of Partition Instances: 7
```

Audit Name	Partition Number	Global CommitID	Newlog Timestamp	Local CommitID
DBEONE	SECT	0000015E00000A21	2C31F1BE00055B15	0000015E00000A21
DBEONE	AUTH	0000000C00000085	2C31F1BE00055B15	0000000C00000085
DBEONE	STOR	0000004700000150	2C31F1BE00055B15	0000004700000150
DBEONE	DEF	000001620001A789	2C31F1BE00055B15	000001620001A789
DBEONE	3	000001620001D218	2C31F1BE00055B15	000001620001D218
DBEONE	4	000001620001D218	2C31F1BE00055B15	000001620001D218
DBEONE	5	000001620001D218	2C31F1BE00055B15	000001620001D218

```
SQLAudit >>
```

UNLOCK AUDITPOINT

Removes the current log lock on the DBEnvironment log files.

Scope

SQLAudit Only

SQLAudit Syntax

```
SQLAudit >> UNLOCK AUDIT[POINT]
Unlock Log Files (n/y) >> { n[o] }
                          { y[es] }
```

Parameters

NO or YES respond NO if you want to keep the log files locked; respond YES to confirm removing the lock on the log files.

Description

- This command is used to unlock or remove any currently held lock on the DBEnvironment log files. The lock must have been previously defined by the command GET AUDITPOINT or LOCK AUDITPOINT.
- When the log lock is removed (unlocked), transactions are no longer protected from being overwritten by database transactions.
- You must be connected to the DBEnvironment to use this command. If you have not connected to the DBE (using the SET DBENVIRONMENT command), SQLAudit will issue a warning and automatically issue the SET DBENVIRONMENT command on your behalf.

Authorization

You must have DBA authorization in order to use this command.

Example

```
SQLAudit >> unlock auditpoint
Unlock Log Files (n/y) >> yes
```

Index

Special characters

*

implications for ALTER TABLE, 7-11

A

accessing a DBEnvironment, 2-28

access mode

change for DBEFiles, F-7, F-17

display for DBEFiles, F-47

single user and multiuser, 1-6

access plan

displayed in SYSTEM.PLAN view, 8-51

ADD DBEFILE

example, 4-16

expanding DBEFileSet with, 7-8

in SQLMigrate, H-2

syntax, 4-15

when to use, 7-7

adding

column, 7-11

constraint, 7-17

ADDLOG

command creating log file, 4-12

command in SQLUtil, 4-12

syntax, F-2

ADD TO GROUP

example, 4-18

syntax, 4-19

AliasDB file, 1-35

ALLBASE/NET

connecting to remote databases, 4-28

ALLBASE/SQL command size, A-1

ALTDBE

syntax, F-4

ALTER authority

explained, 5-13

ALTER DBEFILE, 7-8

ALTER TABLE, 7-10

example, 7-11

when to use, 7-11

applications

dynamic recompile, 7-21

maintenance, 7-21

modifying, 7-21

stored sections in, 7-21

validation of sections, 7-21

archive checkpoint record

creating with COMMIT ARCHIVE, 6-15

defined, 1-8

archive logging

calculating log file size, 3-24

explained, 6-2

wrapperDBE, 6-28

archive mode

setting in DBECon file, 7-2

assigning a default DBEFileSet, 4-14

assigning pages

to data buffer, 3-30, 4-25

to DBEFiles, 3-13

to log buffer, 3-31, 4-25

to runtime control block buffer, 3-28

ATTACHFILE

attaching a file, 7-27

SQLUtil, 7-27

syntax, F-7

attribute

as a column in a table, 2-10

defined, 2-2

audit

elements changing, 4-11

elements parameter, 4-10

functionality, 4-10

information wrapperDBE, 6-28

logging in effect, 4-12

logging parameter, 4-10

log record, 4-10

name parameter, 4-10

parameter, 4-12

parameters for wrapperDBE, 6-31

syntax, I-2

tool wrapperDBE, 6-28

audit DBE

creating, 4-10, 4-11

example, 4-11

parameter, 4-10

audit tool

exit from SQLAudit, I-6

exit on error, I-19

generate audit information, I-2

get current audit point, I-7

help command, I-9

- invoke editor from SQLAudit, I-5
- lock current audit point, I-10
- modify current audit point, I-12
- quit SQLAudit, I-14
- set, I-15
- set connect to DBEnvironment, I-16
- set echo command, I-17
- set editor command, I-18
- set recoverfile command, I-20
- show current audit point, I-21
- unlock audit point, I-22
- authority
 - basic types, 2-22
 - defined, 2-22
 - EXECUTE in SYSTEM.PROCAUTH, 8-53
 - for SQL statements, B-1
 - grantable privilege, 5-17
 - granting, 4-22
 - in SYSTEM.TABAUTH, 8-71
 - module related, 7-24
 - OWNER, 2-24
 - REFERENCES in SYSTEM.COLAUTH, 8-15
 - revoking, 4-22
 - RUN in SYSTEM.MODAUTH, 8-45
 - space, 2-25
 - special, 2-23
 - special in SYSTEM.SPACEAUTH, 8-67
 - special in SYSTEM.SPACDEFAULT, 8-68
 - SPECIAL in SYSTEM.SPECAUTH, 8-69
 - table and view, 5-13
 - TABLE in SYSTEM.TABAUTH, 8-71
 - UPDATE in SYSTEM.COLAUTH, 8-15
 - verification, 4-23
- authorization group
 - as owner, 2-24
 - characteristics, 2-25
 - CONNECT authority for, 4-21
 - delegating management of, 4-21
 - determining membership, 2-26
 - entries in SYSTEM.GROUP, 8-35
 - indirect member, 2-26
 - invalid chain, 2-26
 - management, 4-18
 - resource authority for, 4-22
- authorize once per session
 - changing in SQLUtil, 7-2
 - in DBECon file, 7-2
 - setting with SQLUtil, F-4
 - startup value, 4-2
- autostart mode
 - and user mode, 4-24
 - changing in SQLUtil, 7-2, F-4
 - defined, 4-24
 - displaying with SQLUtil, F-50, F-57

- procedures, 4-24
- setting in DBECon file, 7-2

B

- backup
 - creating with SQLUtil STORE, F-62
 - creating with SQLUtil STOREONLINE, F-67
 - displaying type set in SQLUtil, F-61
 - of DBEnvironment, 6-11
 - online procedures with archive logging, 6-13
 - overview, 6-1
 - procedures with nonarchive logging, 6-11
 - SET BACKUP in SQLUtil, F-37
 - SQLUtil STORE, 6-11
 - SQLUtil STOREONLINE PARTIAL, F-69
 - SQLUtil STORE PARTIAL, F-72
- backward migration, 1-22
- base table
 - in the system catalog, 8-1
 - view underlying, 2-13
- BEGIN ARCHIVE
 - with archive log files, 1-8
- BINARY
 - defining columns as, 2-11
 - storage requirements, 3-4
- B-tree, 2-15, 3-6
- buffer
 - data, 3-27
 - log, 3-27
 - runtime control block, 3-27

C

- calculations
 - database storage, 3-1
 - directory overhead for tables, 3-6
 - disk space for indexes, 3-6
 - disk space for tables, 3-3
 - index key length, 2-17, 3-6
 - number of index leaf pages, 3-8
 - number of index non-leaf pages, 3-8
 - number of overhead index pages, 3-9
 - pages per table, 3-5
 - page table page for tables, 3-6
 - row length, 2-12, 3-4
 - rows per data page, 3-5
 - rows per index leaf page, 3-7
 - rows per index non-leaf page, 3-7
 - size of nonarchive log files, 3-24
 - total number of index pages, 3-9
- catalog
 - system, 1-5
- CATALOG
 - reserved user, 8-1
 - views owned by, 8-1
- CCOUNT

- in maintaining indexes, 7-16
- CHANGELOG
 - changing log file, 7-29
 - SQLUtil, 7-29
 - syntax, F-9
- changing
 - a table's lock mode, 7-10
 - audit elements, 4-11
 - log file with CHANGELOG, 7-29
 - system table lock type, 7-7
- CHAR
 - defining columns as, 2-11
 - storage requirements, 3-4
- check constraint
 - DBEFileSet specification, 5-6
 - entries in SYSTEM.CHECKDEF, 8-14
 - entries in SYSTEM.CONSTRAINT, 8-23
- checkpoint
 - and log file space, 6-4
 - archive checkpoint record, 1-8
 - using to monitor log file usage, 3-21
- CHECKPOINT
 - host variable, 7-28
 - parameter, 7-28
 - SQLUtil, 7-28
- class
 - as owner, 2-24
 - created implicitly, 2-27
 - creating, 5-14
 - defining, 2-27
 - differences from DBEUserID and group, 2-27
 - guidelines for creating, 2-28
- cleanup
 - with monitor, 7-31
- cluster count
 - in maintaining indexes, 7-16
- clustering index
 - changing a key, 7-17
 - creating, 2-18
 - defined, 2-17
 - design, 2-18
 - guidelines for, 2-18
 - when to create, 2-18
- collision of two keys, 2-19
- column
 - adding, 7-11
 - defining data types for, 2-10
 - definition, 2-10
 - definitions stored in the system catalog, 8-18, 8-19
 - deleting, 7-11
 - maximum length, 2-8
 - maximum number of, 2-8
 - minimum number, 5-4
 - naming, 2-10
 - null values for, 2-12
 - renaming, 7-14
 - size, 2-12
 - UPDATE authority in system catalog, 8-15
- command files
 - creating with SQLGEN, 1-15
 - in loading tables, 5-22
 - uses for, 1-13
- comment partition parameter, 4-10
- COMMIT ARCHIVE
 - with archive log files, 1-8
- complex operations
 - in maintenance, 7-1
- compound key
 - when to use, 2-17
- concurrency
 - in creating tables, 5-1
- concurrent
 - connections maximum, A-1
 - sessions maximum, A-1
 - transactions, 3-24
 - transactions maximum, A-1
- CONNECT authority
 - defined, 2-23
 - delegating, 4-21
 - for modules, 7-24
 - granting, 4-20
 - listed in SYSTEM.SPECAUTH, 8-69
 - revoking, 4-20
- connecting
 - example, 4-25
 - starting a DBE session by, 4-23
 - to the DBEnvironment, 1-6
- console log file
 - wrapperDBE, 6-30, 6-31
- console message file
 - messages, 7-28
- constraint
 - adding to table, 7-17
 - creating tables with, 5-5
 - dropping from table, 7-18
 - effects on loading tables, 5-22
 - entries in SYSTEM.CHECKDEF, 8-14
 - entries in SYSTEM.CONSTRAINT, 8-23
- CREATE DBEFILE
 - deviceid, 7-26
 - example, 4-15
 - group name, 7-26
 - in SQLMigrate, H-3
 - syntax, 4-14
- CREATE DBEFILESET
 - effect on SYSTEM.DBEFILE, 8-31
 - example, 4-13
 - syntax, 4-13
- CREATE GROUP

- example, 4-18
- syntax, 4-19
- CREATE INDEX
 - example, 5-11
 - syntax, 5-11
- CREATE SCHEMA
 - defining a database with, 5-20
- CREATE TABLE
 - basic syntax, 5-1
- CREATE TEMPSPACE
 - example, 3-26
 - to create sort space, 3-26
- CREATE VIEW
 - example, 5-10, 5-15
 - syntax, 5-9
- creating
 - audit DBE, 4-10, 4-11
 - indexes, 5-11
 - log file, 4-12
 - partition, 5-8
- creating objects
 - authorization groups, 4-18
 - databases, 5-1
 - DBEFiles, 4-14
 - DBEFileSets, 4-13
 - DBEnvironments, 4-2
 - defining a database schema, 5-20
 - expandable DBEFiles, 4-16
 - hash structures, 5-10
 - indexes, 5-11
 - log files, 4-4
 - procedures, 5-12
 - rules, 5-12
 - security scheme, 4-18
 - security schemes, 5-13
 - tables, 5-1
 - views, 5-9
- current language, 1-34

D

- data
 - access with views, 2-14
 - conversion and performance, 2-11
 - independence, 2-13
 - normalizing, 2-4
 - redundancy, 2-4
 - statistical, 8-5
- database
 - logical design, 2-1
 - objects, 2-24
 - physical design, 3-1
 - sample tables, 2-10
 - schema for, 5-20
 - storage calculation, 3-1
- database administrator (DBA)

- defined, 1-1
- database design
 - logical, 2-1
 - physical, 3-1
- data buffer
 - changing number of pages, F-4
 - displaying number of pages, F-57
 - pages displaying with SQLUtil, F-50
 - setting in DBECon file, 7-2
 - setting number of pages in SQLUtil, 7-2
 - shared memory for, 3-27
 - size, 3-30, A-1
- data definition
 - disabling, 7-30
 - disabling in DBECon file, 7-2
 - disabling in SQLUtil, 7-2
 - enabling in DBECon file, 7-2
 - overview, 2-1
- data types
 - BINARY, 2-11
 - CHAR, 2-11
 - DATE, 2-11
 - DATETIME, 2-11
 - DECIMAL, 2-11
 - FLOAT, 2-11
 - in defining columns, 2-10
 - INTEGER, 2-11
 - INTERVAL, 2-11
 - REAL, 2-11
 - SMALLINT, 2-11
 - storage requirements, 3-4
 - TIME, 2-11
 - VARBINARY, 2-11
 - VARCHAR, 2-11
- DATE
 - defining columns as, 2-11
 - storage requirements, 3-4
- DATETIME
 - defining columns as, 2-11
 - storage requirements, 3-4
- DBA authority, 2-29
 - defined, 2-23
 - for SYSTEM views, 8-1
 - statements requiring, 4-20
 - uses of, 4-19
- DBA (database administrator)
 - defined, 1-1
- DBA tasks
 - database creation and security, 5-1
 - logical design, 2-1
 - physical design, 3-1
- DBCore
 - and SYSTEM.CALL pseudotable, 8-10
 - control block buffer pages, 3-28
- DBE

- audit parameter, 4-10
- creating audit, 4-10
- DBECon file
 - adjusting values in, 7-2
 - backup, 6-11, 6-15
 - creating parameters with START DBE NEW, 4-2
 - DBEnvironment language, 4-2
 - DDL Enabled flag, 4-2, F-4
 - default options, 4-25
 - element in a DBEnvironment, 1-5
 - example, 4-3, 4-6
 - in a new DBEnvironment, 4-4
 - Memory Resident Data Buffer flag, 4-2
 - purging, F-21, F-22
 - SHOWDBE-ALL, F-50
 - timeout value, 4-2
 - wrapperDBE, 6-29, 6-31
- DBECon file parameters, 7-2
- DBECreator
 - and DBECon file, 4-2
 - and restoring, F-30
 - and restoring partial, F-35
 - and START DBE NEWLOG, 6-27
 - changing in SQLUtil, 7-2
 - defined, 1-1
 - setting in DBECon file, 7-2
 - specified in the DBECon file, 4-4
- DBEFile
 - access, F-10
 - access mode change, F-7, F-17
 - access mode display, F-47
 - adding to DBEFileSets, 3-13, 4-15
 - allocating expandable DBEFile space, 4-16
 - altering the type of, 7-8
 - calculating data rows per page, 3-5
 - calculating number of directory pages, 3-5
 - creating, 4-14
 - defined, 3-2
 - device change, F-7, F-9, F-17
 - dropping, 7-9
 - element in a DBEnvironment, 1-2, 1-4
 - entries in SYSTEM.DBEFILESET, 8-34
 - estimating size, 3-13
 - listed in SYSTEM.DBEFILE, 8-31
 - maximum size, A-1
 - naming, 4-14
 - on different disk drives, 3-12
 - pages, 2-12, 3-13, 4-14
 - per DBEnvironment, A-1
 - purging, F-21, F-22, F-23
 - size, 3-13
 - size in pages, 3-2
 - type, 3-2, 3-11, 4-14
 - type index, 3-11
 - type, mixed, 3-2, 3-11
 - types, 3-11
 - type table, 3-11
 - when to add, 7-7
- DBEFile0, 3-17
 - DBEFILE clause, 3-17
 - default name, 3-17
 - default size, 3-17
 - displaying name with SQLUtil, F-50, F-57
 - element in a DBEnvironment, 1-5
 - in a new DBEnvironment, 4-4
 - purging, F-21, F-22, F-23
 - setting in DBECon file, 7-2
- DBEFileSet, 3-2
 - access, F-10
 - adding DBEFiles to, 4-15
 - authority in SYSTEM.SPACEDEFAULT, 8-68
 - creating, 4-13
 - DBEFiles in, 3-2
 - default, 4-14
 - element in a DBEnvironment, 1-3
 - entries in SYSTEM.DBEFILESET, 8-34
 - for check constraints, 5-6
 - for long columns, 5-6
 - for tables, 5-6
 - granting authority, 4-23
 - listed in SYSTEM.DBEFILE, 8-31
 - maximum size, A-1
 - organizing tables in, 3-9
 - placing tables and indexes in, 3-20
 - revoking authority, 4-23
 - system, 3-17
 - SYSTEM, 3-9, 3-17, 4-4
 - tables, 3-10
- DBELOG1, 3-23
- DBELOG2, 3-23
- DBEnvironment, 2-28
 - access, 2-28
 - backup, 6-11
 - configuration, 3-21, 4-1, 4-2
 - creating separate, 2-29
 - elements, 1-4
 - elements created at configuration, 4-3
 - elements illustrated, 1-5
 - maintenance, 7-2
 - migration of, 1-12
 - naming, F-4
 - recovery, 6-15
 - recreating with SQLGEN, 1-15
 - sample, 2-29
 - starting, 4-25
 - starting with SQLGEN, G-55
 - startup parameters, F-57
 - stopping, 4-26

- storage requirements, 3-1
- DBE session, 4-23
 - starting, 4-24, 4-25, 4-26
 - terminating, 4-27
- DBEUserID, 4-23
 - as owner, 2-24, 5-3
 - reserved users, 8-1
- DBEUserIDs
 - reserved, 2-25
- DDL enabled flag
 - setting in DBECon file, 7-2
- DDL Enabled flag
 - changing, F-4
 - changing in SQLUtil, 7-2
 - disabling data definition, 7-30
 - startup value, 4-2
- DDL statements
 - wrapperDBE, 6-31
- deadlock
 - monitoring with SYSTEM.COUNTER, 8-29
- DECIMAL
 - defining columns as, 2-11
 - storage requirements, 3-4
- default
 - column values specifying, 5-5
 - DBEFileSet assigning, 4-14
 - partition parameter, 4-10
 - timeout setting in DBECon file, 7-2
 - user timeout changing in SQLUtil, 7-2
- deferred constraints
 - used when loading tables, 5-22
- definition
 - of audit functionality, 4-10
 - of objects in system catalog, 3-18
- DELETE authority
 - and views, 5-15
 - explained, 5-13
- deleting
 - column, 7-11
 - table, 7-11
- description file option
 - example, 5-21
- DETACHFILE
 - detaching a file, 7-27
 - SQLUtil, 7-27
 - syntax, F-10
- device change for DBEFiles, F-7, F-17
- deviceid
 - CREATE DBEFILE, 7-26
- direct grantable privilege
 - and WITH GRANT option, 5-17
- directory
 - calculations for overhead, 3-6
 - of log files, 6-3
- disk space
 - for DBEFile0, 3-17
 - for indexes, 3-6
 - for log file, 3-23
 - for system catalog, 3-17
 - for tables, 3-3
 - modules, 7-23
 - monitoring use, 7-7
- display DBE attributes
 - SHOWDBE, F-48
- double quotes, 5-7
- DROP DBEFILE, 7-9
 - before purging, F-23
 - example, 7-9
- DROP GROUP
 - syntax, 4-19
- DROP MODULE, 7-26
- dropping
 - constraint, 7-18
 - DBEFiles, 7-9
 - indexes, 7-16
 - modules, 7-26
 - partition, 5-8
 - rules and procedures, 7-19
 - sections, 3-18
 - side effects of dropping constraints, 7-20
 - tables, 7-10
- DROP TABLE, 7-10
- dual file
 - logging type, 6-3
- dual logging, 3-23
 - enabling, disabling, 6-27
 - specifying for DBE, 4-4
- dynamic statements, 7-24

E

- ECHO_ALL
 - SET in SQLGEN, G-49
 - SET in SQLUtil, F-37
- editor
 - syntax, I-5
- EDITOR
 - SQLGEN, G-2
- ENDRECOVERY
 - syntax, F-12
- entity
 - as a database table, 2-10
 - defined, 2-2
- error code
 - 4008, 3-29
- example
 - audit DBE, 4-11
 - partition, 5-8
- EXECUTE authority
 - entries in SYSTEM.PROCAUTH, 8-53
 - granting, 4-22

EXIT

- from SQLAudit, I-6
- from SQLGEN, G-3
- from SQLMigrate, H-4
- from SQLUtil, F-14

EXIT_ON_DBERR

- SET in SQLUtil, F-37

EXIT_ON_DBERR OFF

- SET in SQLGEN, G-52

exit on error

- SQLAudit, I-19

exit status

- checking, 1-33
- setting, 1-33

expandable DBEFiles

- creating, 4-16

expanding

- log file, 6-27
- system DBEFileSet, 3-18

EXTERNAL load

- loading tables, 5-21

F

first normal form

- defined, 2-4

FLOAT

- defining columns as, 2-11
- storage requirements, 3-4

formula

- directory overhead for tables, 3-6
- index key length, 3-6, A-1
- indexkey length, 2-17
- nonarchive log file size, 3-24
- number of index leaf pages, 3-8
- number of index non-leaf pages, 3-8
- number of overhead index leaf, 3-9
- pages per table, 3-5
- row length, 2-12, 3-4, A-1
- rows per data page, 3-5
- rows per index leaf page, 3-7
- rows per index non-leaf page, 3-7
- total number of index pages, 3-9

forward migration, 1-20

functional dependency, 2-6

G

GENERATE ALL, G-4

generate audit information

- audit tool, I-2

GENERATE DBE, G-9

GENERATE DBEFILES, G-10

GENERATE DEFAULTSPACE, G-12

GENERATE GROUPS, G-13

GENERATE INDEXES, G-15

GENERATE INSTALLAUTH, G-17

GENERATE LOAD, G-18

GENERATE MODAUTH, G-20

GENERATE MODOPTINFO, G-22

GENERATE PARTITION, G-24

GENERATE PROCAUTH, G-25

GENERATE PROCEDURES, G-27

GENERATE PROCOPTINFO, G-29

GENERATE RULES, G-31

GENERATE SPACEAUTH, G-33

GENERATE SPECAUTH, G-34

GENERATE STATISTICS, G-35

GENERATE TABAUTH, G-37

GENERATE TABLES, G-39

GENERATE TEMPSPACES, G-42

GENERATE VIEWAUTH, G-43

GENERATE VIEWS, G-45

GENPLAN

- and SYSTEM.PLAN, 8-51

get current audit point, I-7

GRANT

- EXECUTE authority, 4-22

- RUN authority, 4-22

- SELECT ON, 5-14

grantable privilege

- explained, 5-17

granting

- procedure authorities, 7-18

- SECTIONSPACE authority, 4-23

- TABLESPACE authority, 4-23

- to PUBLIC, 5-2

group

- assigning authorities to, 5-14

- created explicitly, 2-27

- entries in SYSTEM.GROUP, 8-35

- name CREATE DBEFILE, 7-26

H

hard crash, 1-8

- recovery from, 6-15

hash structure

- creating tables as, 5-10

- design, 2-19

- disk space for, 3-14

- entries in SYSTEM.HASH, 8-36

HELP

- in SQLMigrate, H-5

- SQLAudit, I-9

- SQLGEN, G-47

- SQLUtil, F-15

host variable

- CHECKPOINT, 7-28

HPRDBSS

- as owner of system tables, 7-7

- reserved user, 8-1

I

- identifier numbers
 - for log files, 6-4
- implicit locking
 - private, 5-2
 - public, 5-2
 - publicread, 5-2
- index
 - altering the key, 7-17
 - as type of DBEFile, 3-2
 - authority, 5-13
 - calculating key length, 3-6
 - calculating rows per leaf page, 3-7
 - calculating rows per non-leaf page, 3-7
 - cluster count, 7-16
 - clustering, 2-17
 - clustering design, 2-18
 - compound key, 2-16
 - creating, 5-11
 - DBEFile, 3-11
 - design, 2-15, 2-17
 - disk space for, 3-6
 - dropping, 7-10
 - dropping and recreating, 7-16
 - entries in SYSTEM.CONSTRAINTINDEX, 8-26
 - entries in SYSTEM.HASH, 8-36
 - entries in SYSTEM.INSTALLAUTH, 8-44
 - for joins, 2-17
 - key, 2-16, 7-17
 - key length, 2-16, 3-6, A-1
 - maximum key length, A-1
 - maximum per DBEnvironment, A-1
 - name, 5-11
 - number of leaf pages, 3-8
 - number of non-leaf pages, 3-8
 - number of overhead pages, 3-9
 - organizing in DBEFileSets, 3-9
 - simple key, 2-16
 - stored in mixed DBEFile, 3-11
 - temporary, 7-16
 - total number of pages, 3-9
 - unique, 2-17
 - unique design, 2-18
 - updating, 7-15
- INDEX authority
 - explained, 5-13
- indicator variable
 - for null values, 2-12
- in effect, audit logging, 4-12
- INFO command
 - displaying column definitions with, 5-7
- initial table loads
 - in database creation, 5-20

- input tree, 3-19, 7-21
- INSERT, 7-13
- INSERT authority
 - and views, 5-15
 - explained, 5-13
- INSTALL
 - example, 7-25
 - transferring modules with, 7-25
- installable module
 - SQLMOD, 7-25
- INTEGER
 - idefining columns as, 2-11
 - storage requirements, 3-4
- integrity constraint
 - creating tables with, 5-5
 - design, 2-21
 - disk space for, 3-17
 - entries in SYSTEM.CONSTRAINT, 8-23
 - entries in SYSTEM.CONSTRAINTCOL, 8-25
 - entries in SYSTEM.CONSTRAINTINDEX, 8-26
- intermediate levels of a tree, 2-15
- INTERNAL load
 - loading tables, 5-21
- INTERVAL
 - defining columns as, 2-11
 - storage requirements, 3-4
- invalidation
 - application program sections, 7-21, 7-22, 7-23
 - statements causing, 7-23
- invoke editor from SQLAudit
 - audit tool, I-5
- ISQL
 - command size, A-1
 - overview, 1-13
 - syntax summary, D-1

J

- JCW
 - GENERR, 1-18
 - GENWARN, 1-18
 - MIGERR, 1-23
 - MIGWARN, 1-23
 - SQLVERERR, 1-25
 - system JCW and SQLGEN, 1-18
 - system JCW and SQLMigrate, 1-23, H-14
 - system JCW and SQLUtil, 1-14, F-38
 - UTILERR, 1-14
 - UTILWARN, 1-14
- join, 2-8
 - maximum tables per, A-1
 - used to create a view, 5-9

K

key

- calculation of length, 3-6
- changing, in a clustering index, 7-17
- compound, 2-16, 2-17
- creating an index on primary, 2-17
- defined, 2-2
- determining for index, 2-16
- simple, 2-16

L

LANG clause

- in CREATE TABLE, 5-4

language

- current language, 1-34
- DBECon file setting, 7-2
- DBEnvironment and current, 4-6
- displaying the DBE's, F-54
- in defining tables or columns, 5-4
- native language support, 1-34

LANGUAGEID

- in SYSTEM.COLUMN, 8-19
- in SYSTEM.TABLE, 8-74

leaf level of a tree, 2-15

limits

- in ALLBASE/SQL, A-1
- table of maximum, A-1

list archived files

- STOREINFO, F-64

LOAD

- example, 5-21
- external, 7-14
- EXTERNAL option, 5-21
- GENERATE LOAD, G-18
- in deleting columns, 7-11
- in maintenance operations, 7-2
- INTERNAL option, 5-21
- loading tables from files, 5-20

lock current audit point, I-10

locking

- in table creation, 5-3
- of system catalog by UPDATE STATISTICS, 7-6
- on the system catalog, E-1
- PUBLIC, 2-22, 4-20
- publicread, 5-2
- strategy in CREATE TABLE, 5-1
- system catalog, 7-25

LOCK TABLE

- override automatic locking, 5-3

lock type of system tables

- changing, 7-7

log buffer

- changing number of pages, F-5

- displaying number of pages, F-50, F-57
- setting in DBECon file, 7-2
- setting number of pages in SQLUtil, 7-2
- shared memory for, 3-27
- size, 3-31, A-1
- written to log file, 3-31

LOG DBEFILE clause, 3-23

log file, 1-35

- adding with ADDLOG, 6-24
- adding with ADDLOG:syntax, F-2
- ADDLOG command, 4-12
- archive logging, 1-8
- backup, 6-15
- changing dual logging, 6-27
- changing the name, 6-27
- changing the size, 6-27
- creating, 4-4, 4-12
- creating a new, 6-27
- creating new with START DBE NEWLOG, 6-27
- default name, 3-23
- default size, 3-23
- defined, 1-7
- different from DBEFiles, 3-2, 3-20
- disk space for, 3-23
- displaying information with SHOWLOG, 4-9, 6-21, F-58
- dual, in wrapperDBE, 6-29
- dual logging, 3-23
- element in a DBEnvironment, 1-5
- estimating size for, 3-24
- inactive, in wrapperDBE, 6-32
- in a new DBEnvironment, 4-4
- listed in log directory, 6-3
- log records, 3-21
- moving with MOVELOG, 6-26
- not usable, 6-30
- number of, wrapperDBE, 6-32
- purging, F-21, F-22
- recovering with RECOVERLOG, 6-25
- rescuing with RESCUELOG, 6-25, F-29
- restoring with RESTORELOG, 6-25, F-31
- single, in wrapperDBE, 6-29
- size, 4-12, A-1
- size in pages, 3-20
- SQLUtil, 4-12
- storing with STORELOG, 6-24
- switching, 3-22, 6-3, 6-4
- usable, in wrapperDBE, 6-29
- wrapperDBE, 6-28, F-76
- writeahead logging, 1-7

logfile

- change for DBEnvironment, F-9
- name(s) in DBECon file, 7-2

- name stored in DBECon file, 7-2

LOG FULL

- understanding the condition, 6-3

logging

- archive explained, 6-2
- enabling archive logging, 4-5
- file names and numbers, 6-4
- file sequence numbers, 6-4
- LOG FULL condition, 6-3
- managing log files, 6-21
- nonarchive explained, 6-2
- single or dual log files, 6-3
- specifying dual logging, 4-4
- understanding log file types, 6-3
- using multiple files, 6-3
- using nonarchive logs, 6-4, 6-6

- logical design, 2-1

- log pages, 3-24

- log records, 3-21, 3-31

- audit, 4-10

- written to the log file, 1-8

LONG BINARY

- storage requirements, 3-4

long column

- DBEFileSet specification, 5-6

LONG VARBINARY

- storage requirements, 3-4

M

maintenance

- adding a column, 7-11
- adding a constraint to a table, 7-17
- adding DBEFiles to DBEFileSets, 7-8
- applications, 7-21
- changing a table's lock mode, 7-10
- changing DBEFile type, 7-8
- cleanup with monitor, 7-31
- DBECon File, 7-2
- DBEnvironment, 7-2
- deleting a column, 7-11
- deleting table rows, 7-10
- disk space, 3-9
- dividing tables, 7-14
- dropping a constraint from a table, 7-18
- dropping a DBEFile, 7-9
- dropping and recreating hash structures, 7-15
- dropping and recreating indexes, 7-16
- dropping and recreating procedures, 7-19
- dropping and recreating rules, 7-19
- dropping a table, 7-10
- dropping modules, 7-26
- enabling and disabling data definition, 7-30
- granting and revoking procedure authorities, 7-18
- indexes, 7-15

- merging tables, 7-13

- reloading tables, 7-17

- revalidating procedure sections, 7-20

- rules and procedures, 7-18

- security schemes, 7-30

- setting SQLUtil maintenance word, F-55

- system catalog statistics, 7-6

- tables, 7-10

- updating groups, 7-30

- using simple and complex operations, 7-1

maintenance word

- adjusting parameter in DBECon file, 7-2

- changing in SQLUtil, 7-2

- changing with SQLUtil, F-4

- setting, F-40, F-55

- setting in DBECon file, 7-2

maximum

- table of limits, A-1

- timeout value in DBECon file, 7-2

- transactions in DBECon file, 7-2

- transactions in SQLUtil, 7-2

- user timeout in SQLUtil, 7-2

maximum transactions

- changing with SQLUtil, F-5

- displaying with SQLUtil, F-50, F-57

- maxpartitions parameter, 4-10

media failure

- recovery from, 6-15

memory resident data buffer flag

- changing, 7-2

- setting, F-4

- startup value, 4-2

messages

- console message file, 7-28

MIGRATE

- in SQLMigrate, H-6

migration

- backward, 1-22

- forward, 1-20

- of DBEnvironments, 1-12

- steps in, 1-20

mixed

- as type of DBEFile, 3-2

- modify current audit point, I-12

module

- and SYSTEM.SECTION view, 8-64

- and SYSTEM.SETOPTINFO view, 8-66

- definition of, 7-23

- disk space for, 7-23

- dropping, 7-26

- installing, 7-25

- pages needed to store, 3-19

- validation of, 7-22, 7-23

monitor

- for cleanup, 7-31

- monitoring performance
 - with SQLMON, 1-24
- MOSET
 - in SET BACKUP, F-37
- MOVEFILE, 3-12
 - moving a file, 7-27
 - SQLUtil, 7-27
 - syntax, F-17
- MOVELOG
 - syntax, F-19
- moving a file
 - MOVEFILE, 7-27
- multiple files
 - in logging, 6-3
- multiuser mode
 - changing to, 7-2
 - DBE session, 1-6
 - setting in DBECon file, 7-2

N

- naming
 - double quotes, 4-13
- NATIVE-3000
 - defined, 1-34
- native language
 - current language, 1-34
 - DBEnvironment and current, 4-6
 - defaults, 1-34
 - displaying the DBE's, F-54
 - in creating tables, 5-4
 - in DBECon file, 4-6
 - in defining columns, 5-5
 - overview, 1-34
 - shown in SYSTEM.COLUMN, 8-19
 - shown in SYSTEM.TABLE, 8-74
- NETUsers file, 1-35
- NETUtil
 - purpose, 1-35
- nonarchive logging
 - and backing up database files, 6-11, 6-13
 - calculating log file size, 3-24
 - calculating number of pages, 3-24
 - explained, 6-2
 - size, 3-24
- nonreplicate DBE
 - SQLMigrate, 1-19
- normal forms
 - in database design, 2-4
- normalization, 2-5, 2-9
 - defined, 2-4
 - first normal form, 2-4
- NOT NULL, 2-12
- not usable log files
 - wrapperDBE, 6-30
- null indicator
 - in loading from files, 5-21

- in loading from files, 5-21
- null values, 2-12

O

- objects
 - statements that create, 3-18
- once per session flag in DBECon file, 7-2
- online backup
 - procedures in archive mode, 6-13
- orphaned log files
 - wrapperDBE, 6-28
- orphaned privilege
 - defined, 5-18
 - elimination of, 5-19
- overflow page, 2-19
- owner
 - and OWNER authority, 2-24
 - class, 2-24
 - creating classes, 5-14
 - defining classes, 2-27
 - in SYSTEM.TABLE, 5-7
 - of system section tables, 7-7
 - of system tables, 7-7
 - of tables, 5-3
- OWNER authority
 - defined, 2-23
 - for modules, 7-24

P

- pages
 - and clustering indexes, 2-18
 - assigning to DBEFiles, 3-13
 - calculating data rows per page, 3-5
 - calculating pages per table, 3-5
 - calculating rows per index leaf page, 3-7
 - calculating rows per index non-leaf page, 3-7
 - efficient use of, 3-4
 - in DBEFiles, 3-2
 - in log files, 3-20
 - in START DBE NEW, 3-17
 - number of index leaf pages, 3-8
 - number of index non-leaf pages, 3-8
 - number of overhead index pages, 3-9
 - temporary, 3-26
 - total number of index pages, 3-9
- page table page, 3-6
 - calculations for tables, 3-6
- parameter
 - audit, 4-12
 - audit elements, 4-10
 - audit log, 4-10
 - audit name, 4-10
 - CHECKPOINT, 7-28
 - comment partition, 4-10

- default partition, 4-10
- maxpartitions, 4-10
- replicate, 1-15
- parameters
 - changing with SQLUtil, F-4
 - displaying with SQLUtil, F-50
 - in DBECon file, 7-5
 - startup, 4-2
 - startup in SQLUtil, F-57
- parameters in procedures
 - default values in SYSTEM.PARAMDEFAULT, 8-46
 - entries in SYSTEM.PARAMETER, 8-47
- partition
 - creating, 5-8
 - dropping, 5-8
 - example, 5-8
 - table, 5-8, 8-5
- PCR (Parent-child relationship)
 - explained, 3-17
- performance
 - and deleting indexes, 7-16
 - effect on, from using mixed DBEFiles, 3-11
 - improving, by grouping tables, 3-9
 - improving, with clustering indexes, 2-18
 - improving with index on join column, 2-17
 - monitoring with SQLMON, 1-24
- physical design, 3-1
- physical object, 2-14
- preprocessing, 7-23, 7-25
 - drop, 7-23
- PREVIEW
 - in SQLMigrate, H-9
- primary key
 - creating an index on, 2-17
 - creating unique constraints with, 5-5
- printing
 - of rules, 7-5
- priority
 - in transactions, 8-80
- private, 5-2
- procedure
 - creating, 5-12
 - dropping and recreating, 7-19
 - entries for parameters in SYSTEM.PARAMETER, 8-47
 - entries in SYSTEM.PARAMDEFAULT, 8-46
 - entries in SYSTEM.PROCEDURE, 8-54
 - entries in SYSTEM.PROCRESULT, 8-57
 - examining in the system catalog, 7-19
- EXECUTE
 - authority in SYSTEM.PROCAUTH, 8-53
- GENERATE PROCAUTH command, G-25
- GENERATE PROCEDURES command, G-27

- maintaining, 7-18
- stored sections in, 7-21
- pseudo-mapped files
 - and online backups, 6-14
 - moving to mapped, F-17
 - not supported, F-17
- pseudotables
 - defined, 8-1
 - functions, 8-2
 - STOREDSECT.SYSTEM, 8-7
 - SYSTEM.ACCOUNT, 8-9
 - SYSTEM.CALL, 8-10
 - SYSTEM.COUNTER, 8-29
 - SYSTEM.PLAN, 8-51
 - SYSTEM.TRANSACTION, 8-80
 - SYSTEM.USER, 8-82
- PUBLIC
 - granting authorities to, 2-22
 - grant to, 5-2
 - special DBEUserID, 8-5
- PURGEALL
 - syntax, F-21
- PURGEDBE
 - syntax, F-22
- PURGEFILE, 6-27, 7-9
 - syntax, F-23
- PURGELOG
 - syntax, F-25

Q

- query
 - and SYSTEM.PLAN view, 8-51
- query optimization, 2-15
- quit
 - syntax, I-14
- QUIT
 - in SQLMigrate, H-12
 - in SQLUtil, F-26
- quit from SQLAudit
 - audit tool, I-14
- QUIT (SHOWDBE)
 - description, F-56

R

- REAL
 - defining columns as, 2-11
 - storage requirements, 3-4
- recompile dynamically, 7-21
- RECOVERLOG
 - syntax, F-27
- recovery
 - in archive mode, 6-15
 - in nonarchive mode, 6-12
 - overview, 6-1
 - rollback, 6-15

- recreating
 - indexes, 7-16
 - rules and procedures, 7-19
- REFERENCES authority
 - explained, 5-13
 - listed in SYSTEM.COLAUTH, 8-15
- REFERENCES clause
 - defining referential constraints with, 5-5
- referential constraint
 - creating tables with, 5-5
 - entries in SYSTEM.CONSTRAINT, 8-23
 - entries in SYSTEM.CONSTRAINTCOL, 8-25
 - entries in SYSTEM.CONSTRAINTINDEX, 8-26
- RELEASE, 4-23
 - SQLGEN, G-48
 - terminating a DBE session with, 4-26
- remote databases
 - accessing, 4-28
- REMOVE DBEFILE
 - example, 7-9
- REMOVE FROM GROUP
 - example, 4-22
 - syntax, 4-19
- removing all rows
 - TRUNCATE TABLE statement, 7-29
- renaming
 - columns, 7-14
 - tables, 7-14
- REPAIR
 - in SQLMigrate, H-13
- repeating data groups
 - removing in normalization, 2-5
- replicate
 - DBE using SQLmigrate, 1-19
 - parameter, 1-15
- re-preprocessing, 7-23
 - avoiding, 7-23
- RESCUELOG
 - syntax, F-29
- reserved users
 - CATALOG, 8-1
 - HPRDBSS, 8-1
 - PUBLIC, 2-22, 4-20
 - SYSTEM, 8-1
 - TEMP, 2-25
- resource authority
 - listed in SYSTEM.SPACEAUTH, 8-67
- RESOURCE authority, 4-21
 - defined, 2-23
- RESTORE
 - syntax, F-30
- RESTORELOG
 - syntax, F-31
- RESTORE PARTIAL
 - syntax, F-35
- RESTORESET
 - in SET BACKUP, F-37
- restoring
 - data with RESTORE, 6-12, 6-16, 6-17
 - logs with RESTORELOG, 6-16, 6-18
- revalidate
 - procedure sections, 7-20
- revalidating
 - application program sections, 7-21
 - sections, 7-23
- REVOKE, 7-23
 - CONNECT authority, 4-20
 - example, 4-22
 - EXECUTE authority, 4-22
 - RESOURCE authority, 4-21
 - SELECT ON, 5-14
- revoking
 - procedure authorities, 7-18
 - SECTIONSPACE authority, 4-23
 - TABLESPACE authority, 4-23
- rollback recovery
 - explained, 6-15
- rollforward
 - wrapperDBE, 6-28
- rollforward recovery
 - archive log files, 1-8
 - explained, 6-16, 6-17
- root level of a tree, 2-15
- row
 - calculating length, 3-4
 - calculating rows per page, 3-5
 - formula for length, 2-12
 - length, 3-4
 - maximum length, 2-8
- rule
 - columns in the system catalog, 8-61
 - creating, 5-12
 - determining behavior in a DBEnvironment, 7-5
 - dropping and recreating, 7-19
 - effects on loading tables, 5-22
 - entries in SYSTEM.RULE, 8-59
 - examining in the system catalog, 7-19
 - maintaining, 7-18
- RUN authority
 - for modules, 7-24
 - granting, 4-22
 - in system catalog, 8-45
- runtime control blocks
 - setting in DBECon file, 7-2
 - setting number in SQLUtil, 7-2
 - shared memory for, 3-27, 3-28
- run tree, 3-19, 7-21

S

sample DBEnvironment

- contents of, 2-29
- tables, 2-10

schema

- database, 5-20
- SQLGEN, 1-15

schema file

- creating with SQLGEN, G-54

search item, 2-16, 2-17

second normal form

- defined, 2-5

section

- and SYSTEM.SECTION view, 8-64
- and SYSTEM.SETOPTINFO view, 8-66
- and UPDATE STATISTICS, 7-6
- calculating space, 3-19
- default DBEFileSet, 4-14
- disk space for, 3-19
- invalidating, 7-10, 7-14, 7-21, 7-22, 7-23
- revalidating, 7-23
- statements that do not create, 7-21
- stored in catalog, 3-19
- types, 7-21
- valid, 7-21
- valid and invalid, 7-21
- validating and invalidating, 7-21

SECTIONSPACE authority

- defined, 2-25
- granting, 4-23
- revoking, 4-23

SECTIONSPACE default DBEFileSet, 4-14

security

- creating DBEnvironment security scheme, 4-18
- management, 2-22
- specifying when creating a table, 5-1

SELECT authority

- and views, 5-15
- explained, 5-13

selectlist, 5-9

semicolon

- used in SQLUtil, 1-13

sequence number

- for log files, 6-4
- wrapperDBE, 6-29, 6-32

sequential search, 2-16

set

- connect to DBEnvironment, I-16
- recoverfile, I-20
- SQLAudit, I-15

SET, H-14

SET BACKUP

- in SQLUtil, F-37

SET CONSTRAINTS DEFERRED

- used when loading tables, 5-22

SETDBEMAIN

- syntax, F-40

SET DBEnvironment

- SQLAudit, I-16

SET DML ATOMICITY

- used when loading tables, 5-22

SET ECHO_ALL

- in SQLAudit, I-17
- in SQLUtil, F-37

SET ECHO_ALL OFF

- syntax, G-49

SET ECHO_ALL ON

- syntax, G-50

SET EDITOR

- SQLAudit, I-18
- SQLGEN, G-51

SET_EXIT_DBERR

- syntax, I-19

SET EXIT_ON_DBERR OFF

- syntax, G-52

SET EXIT_ON_DBERR ON

- syntax, G-53

SETOPT

- and SYSTEM.SETOPTINFO view, 8-66

SET RECOVERFILE

- SQLAudit, I-20

SET SCHEMA

- SQLGEN, G-54

SETUPRECOVERY

- syntax, F-41

SETUPRECOVERY PARTIAL

- set up rollforward, F-44
- syntax, F-44

set up rollforward

- SETUPRECOVERY PARTIAL, F-44

shared memory

- data buffer, 3-30
- estimating requirements, 3-27
- for log buffer, 3-31
- requirements for buffers, 3-27
- runtime control block buffer, 3-28
- transactions, 3-32

SHOWACCESS

- syntax, F-47

SHOW AUDITPOINT

- syntax, I-21

SHOWDBE, 4-6

- ALL-syntax, F-50
- display DBE attributes, F-48
- EXIT-syntax, F-51
- HELP-syntax, F-52
- LANG-syntax, F-54
- MAINT-syntax, F-55

- QUIT-syntax, F-56
- STARTPARMS-syntax, F-57
- syntax, F-48
- wrapperDBE, 6-31, 6-33
- SHOW 'DBEnvironmentName' VERSION, H-16
 - in SQLMigrate, H-16
- SHOWLOG
 - syntax, F-58
 - using to monitor log file usage, 6-21
 - wrapperDBE, 6-29
- SHOWSET
 - in SQLUtil, F-61
- SHOW VERSIONS
 - in SQLMigrate, H-17
- simple key, 2-16
- simple operations
 - in maintenance, 7-1
- single file
 - logging type, 6-3
- single user mode
 - and starting a DBE session, 4-25
 - changing to, 7-2
 - DBE session, 1-6
 - setting in DBECon file, 7-2
- size of log file, 4-12
- SMALLINT
 - defining columns as, 2-11
 - storage requirements, 3-4
- soft crash, 1-8
 - and rollback recovery, 6-15
- sort ordering, 2-17
- special authority
 - DBA authority, 4-19
 - listed in SYSTEM.SPACEAUTH, 8-67
 - listed in SYSTEM.SPACEDEFAULT, 8-68
 - listed in SYSTEM.SPECAUTH, 8-69
- SQLAudit
 - overview, 1-27
- SQLAudit commands
 - audit[point], I-2
 - editor, I-5
 - exit, I-6
 - get audit[point], I-7
 - help, I-9
 - lock audit[point], I-10
 - modify audit[point], I-12
 - quit, I-14
 - set, I-15
 - set DBEnvironment, I-16
 - set echo, I-17
 - set editor, I-18
 - set_exit_dberr, I-19
 - set recoverfile, I-20
 - show audit[point], I-21
 - unlock audit[point], I-22
- SQLCore
 - and null values, 2-12
 - and pseudotables, 8-1
 - system statistics used by, 8-29
- SQLGEN
 - command summary, 1-16
 - introduction, 1-15
 - schema, 1-15
 - schema files, 1-16
 - starting up, 1-15
- SQLGEN commands
 - EDITOR, G-2
 - EXIT, G-3
 - GENERATE ALL, G-4
 - GENERATE DBE, G-9
 - GENERATE DBEFILES, G-10
 - GENERATE DEFAULTSPACE, G-12
 - GENERATE GROUPS, G-13
 - GENERATE INDEXES, G-15
 - GENERATE INSTALLAUTH, G-17
 - GENERATE LOAD, G-18
 - GENERATE MODAUTH, G-20
 - GENERATE MODOPTINFO, G-22
 - GENERATE PARTITION, G-24
 - GENERATE PROCAUTH, G-25
 - GENERATE PROCEDURES, G-27
 - GENERATE PROCOPTINFO, G-29
 - GENERATE RULES, G-31
 - GENERATE SPACEAUTH, G-33
 - GENERATE SPECAUTH, G-34
 - GENERATE STATISTICS, G-35
 - GENERATE TABAUTH, G-37
 - GENERATE TABLES, G-39
 - GENERATE TEMPSPACES, G-42
 - GENERATE VIEWAUTH, G-43
 - GENERATE VIEWS, G-45
 - HELP, G-47
 - RELEASE, G-48
 - SET ECHO_ALL OFF, G-49
 - SET ECHO_ALL ON, G-50
 - SET EDITOR, G-51
 - SET EXIT_ON_DBERR OFF, G-52
 - SET EXIT_ON_DBERR ON, G-53
 - SET SCHEMA, G-54
 - STARTDBE, G-55
- SQLINSTL
 - overview, 1-24
- SQLMigrate
 - nonreplicate DBE, 1-19
 - overview, 1-18
 - replicate DBE, 1-19
 - starting up, 1-19
- SQLMigrate commands
 - ADD DBEFile, H-2, H-3
 - EXIT, H-4

- HELP, H-5
- MIGRATE, H-6
- PREVIEW, H-9
- QUIT, H-12
- REPAIR, H-13
- SHOW 'DBEnvironmentName' VERSION, H-16
- SHOW VERSIONS, H-17
- SQLMON
 - available DBEFileset space, 7-7
 - cluster count, 7-16
 - examining DBEFilesets, 4-13
 - hash structures, 7-15
 - index space, 7-15
 - monitoring the log, 6-28
 - overview, 1-24
 - stored sections, 7-24
 - table information, 5-7
- SQL statements
 - authorities required for, B-1
 - locks obtained by, E-1
 - syntax summary, C-1
- SQLUtil
 - ATTACHFILE, 7-27
 - CHANGELOG, 7-29
 - CHECKPOINT, 7-28
 - DETACHFILE, 7-27
 - example, 4-6
 - log file, 4-12
 - MOVEFILE, 7-27
 - overview, 1-13
 - starting up, 1-14
- SQLUtil commands
 - ADDLOG, 4-12, F-2
 - ALTDDBE, F-4
 - ATTACHFILE, F-7
 - CHANGELOG, F-9
 - DETACHFILE, F-10
 - ENDRECOVERY, F-12
 - EXIT, F-14
 - HELP, F-15
 - MOVEFILE, F-17
 - MOVELOG, F-19
 - PURGEALL, F-21
 - PURGEDBE, F-22
 - PURGEFILE, F-23
 - PURGELOG, F-25
 - QUIT, F-26
 - RECOVERLOG, F-27
 - RESCUELOG, F-29
 - RESTORE, F-30
 - RESTORELOG, F-31
 - RESTORE PARTIAL, F-35
 - SET, F-37
 - SETDBEMAJINT, F-40
 - SETUPRECOVERY, F-41
 - SETUPRECOVERY PARTIAL, F-44
 - SHOWACCESS, F-47
 - SHOWDBE, F-48
 - SHOWDBE-ALL, F-50
 - SHOWDBE-EXIT, F-51
 - SHOWDBE-HELP, F-52
 - SHOWDBE-LANG, F-54
 - SHOWDBE-MAINT, F-55
 - SHOWDBE-QUIT, F-56
 - SHOWDBE-STARTPARMS, F-57
 - SHOWLOG, F-58
 - SHOWSET, F-61
 - STORE, F-62
 - STOREINFO, F-64
 - STORELOG, F-65
 - STOREONLINE, F-67
 - STOREONLINE PARTIAL, F-69
 - STORE PARTIAL, F-72
 - WRAPDBE, F-76
- SQLVer, 1-25
- START
 - in ISQL, 1-13
- START DBE, 4-23
 - example, 4-25
 - single user, 4-25
 - startup parameters, 4-25
 - syntax, 4-25, F-5
- STARTDBE
 - SQLGEN, G-55
- START DBE NEW
 - and dual log files, 3-23
 - configuring a DBE, 4-2
 - creating a log file, 4-4
 - elements created, 4-3
 - example, 3-17, 4-3
 - wrapperDBE, 6-31
- START DBE NEWLOG
 - changing dual logging, 6-27
 - creating a new log file, 6-27
 - example, 6-27
- starting DBE sessions
 - with CONNECT, 1-6
 - with START DBE, 1-6
- startup parameter, 4-2
 - adjusting in DBECon file, 7-2
 - changing with SQLUtil, F-4
 - DBEnvironment, F-57
 - default, 4-2
 - defaults, 7-2
 - in DBECon file, 4-2
- statements
 - dynamic, 7-24
- static backup
 - in archive mode, 6-14

- with nonarchive logging, 6-11
- statistical data
 - updating, 8-5
- STOP DBE, 4-23
 - terminating DBE sessions, 4-27
- storage requirements
 - for specific data types, 3-4
 - overview, 3-1
- STORE
 - SQLUtil, F-62
- STOREDSECT
 - as owner of system section tables, 7-7
 - reserved user, 8-1
- stored section
 - and validation, 7-21
 - when stored, 3-19
- STOREDSECT.SYSTEM
 - defined, 8-7
- STOREINFO
 - list archived files, F-64
 - syntax, F-64
- STORELOG
 - syntax, F-65
- STOREONLINE
 - SQLUtil, F-67
- STOREONLINE PARTIAL
 - SQLUtil, F-69
- STORE PARTIAL
 - SQLUtil, F-72
- STORESET
 - in SET BACKUP, F-37
- syntax
 - WRAPDBE command, F-76
- syntax summary
 - ISQL, D-1
 - SQL statements, C-1
- SYSTEM
 - reserved user, 8-1
- SYSTEM.ACCOUNT
 - column definition and description, 8-9
- SYSTEM.CALL
 - column definition and description, 8-10
- system catalog
 - after configuration, 4-4
 - allocating disk space, 3-17
 - CATALOG views, 8-1
 - complete information and tables, 8-1
 - disk space for, 3-17
 - element in a DBEnvironment, 1-5
 - examining, 4-7
 - example, 4-7
 - locks, 7-25
 - locks obtained by SQL statements, E-1
 - locks on, 7-31
 - monitoring space, 7-2
 - owners HPRDBSS, CATALOG, and SYSTEM, 8-1
 - pseudotable functions, 8-2
 - pseudotables, 8-1
 - purging, F-21, F-22
 - rule columns, 8-61
 - rules, 8-59
 - sizeo, 3-18
 - statements that add to, 3-18
 - SYSTEM views, 8-1
 - updating, 7-6
 - view constraint information store in, 5-9
 - view definitions stored in, 5-9
- SYSTEM.CHECKDEF
 - column definition and description, 8-14
 - view constraint information stored in, 5-9
- SYSTEM.COLAUTH
 - column definition and description, 8-15
- SYSTEM.COLDEFAULT
 - column definition and description, 8-18
- SYSTEM.COLUMN
 - affected by UPDATE STATISTICS, 7-6
 - column definition and description, 8-19
 - querying, 5-7
- SYSTEM.CONSTRAINT
 - column definition and description, 8-23
 - view constraint definitions stored in, 5-9
- SYSTEM.CONSTRAINTCOL
 - column definition and description, 8-25
- SYSTEM.CONSTRAINTINDEX
 - column definition and description, 8-26
- SYSTEM.COUNTER
 - column definition and description, 8-29
- SYSTEM.DBFILE
 - affected by UPDATE STATISTICS, 7-6
 - characteristics of files stored in, 4-15
 - column definition and description, 8-31
 - DBFileSet names stored in, 4-16
 - names of files stored in, 4-14
- SYSTEM.DBFILESET
 - affected by UPDATE STATISTICS, 7-6
 - column definition and description, 8-34
 - names stored in, 4-13
- SYSTEM DBFileSet
 - adding space to, 7-2
 - defined, 3-17
 - determining space in, 3-20
 - in a new DBEnvironment, 4-4
- SYSTEM.GROUP
 - column definition and description, 8-35
- SYSTEM.HASH
 - column definition and description, 8-36
- SYSTEM.IMAGEKEY
 - column definition and description, 8-38
- SYSTEM.INDEX

- affected by UPDATE STATISTICS, 7-6
- and cluster count, 7-16
- column definition and description, 8-40
- SYSTEM.INSTALLAUTH
 - column definition and description, 8-44
- SYSTEM.MODAUTH
 - column definition and description, 8-45
- SYSTEM.PARAMDEFAULT
 - column definition and description, 8-46
- SYSTEM.PARAMETER
 - column definition and description, 8-47
- SYSTEM.PLAN
 - column definition and description, 8-51
- SYSTEM.PROCAUTH
 - column definition and description, 8-53
- SYSTEM.PROCEDURE
 - column definition and description, 8-54
- SYSTEM.PROCEDUREDEF
 - column definition and description, 8-56
- SYSTEM.PROCRESULT
 - column definition and description, 8-57
- SYSTEM.RULE
 - column definition and description, 8-59
- SYSTEM.RULECOLUMN
 - column definition and description, 8-61
- SYSTEM.SECTION
 - column definition and description, 8-64
 - monitoring stored sections in, 7-22
 - view definitions stored in, 5-9
- SYSTEM.SETOPTINFO
 - column definition and description, 8-66
- SYSTEM.SPACEAUTH
 - column definition and description, 8-67
- SYSTEM.SPACEDEFAULT
 - column definition and description, 8-68
- SYSTEM.SPECAUTH
 - column definition and description, 8-69
- SYSTEM.TABAUTH
 - column definition and description, 8-71
- SYSTEM.TABLE
 - affected by UPDATE STATISTICS, 7-6
 - column definition and description, 8-74
 - examining table definitions in, 5-7
 - querying, 5-7
 - view definitions stored in, 5-9
- system table, 8-5
 - lock type changing, 7-7
- SYSTEM.TEMPSPACE
 - column definition and description, 8-77
- SYSTEM.TPINDEX
 - column definition and description, 8-78
- SYSTEM.TRANSACTION
 - column definition and description, 8-80
- SYSTEM.USER
 - column definition and description, 8-82

- monitoring, to show current users, 4-27
- SYSTEM.VIEWDEF
 - column definition and description, 8-83
- system views
 - containing security data, 5-19
 - in the system catalog, 8-1
 - listed, 4-7
 - restricting catalog access, 5-16

T

- table
 - adding a column to, 7-11
 - adding constraint to, 7-17
 - as type of DBEFile, 3-2
 - authorities, 5-13
 - changing lock mode, 7-10
 - creating, 5-1
 - DBEFile, 3-11
 - DBEFileSet, 3-10
 - DBEFileSet specification, 5-6
 - default DBEFileSet, 4-14
 - deleting a column to, 7-11
 - deleting all rows, 7-10
 - design, 2-7
 - disk space for, 3-3
 - dividing, 7-14
 - double quotes, 5-3
 - dropping, 7-10
 - dropping constraint from, 7-18
 - examining definitions, 5-7
 - for sample database, 2-10
 - implicit locking, 5-2
 - implied security, 5-2
 - in DBEFileSet, 3-3, 5-6
 - loading, 5-20
 - loading external files, 5-21
 - loading internal file, 5-21
 - locking, 5-2, 5-3
 - maximum column length, A-1
 - maximum columns per, A-1
 - maximum per DBEnvironment, A-1
 - merging columns, 7-13
 - name, 5-3
 - organizing in DBEFileSets, 3-9
 - owner, 5-3
 - partition, 5-8, 8-5
 - private, 5-2
 - public, 5-2
 - publicread, 5-2
 - renaming, 7-14
 - restrictions, 2-8
 - revoking authority for, 5-15
 - separate disk drives, 3-10
 - stored in mixed DBEFile, 3-11
 - system, 8-5

- unloading, 7-12
- TABLESPACE authority
 - defined, 2-25
 - granting, 4-23
 - revoking, 4-23
- TABLESPACE default DBEFileSet, 4-14
- tasks
 - adding a column, 7-11
 - adding a constraint to a table, 7-17
 - adding DBEFiles to DBEFileSets, 7-8
 - adding log files with ADDLOG, 6-24
 - backup and recovery, 6-1
 - backup in nonarchive mode, 6-11
 - changing a table's lock mode, 7-10
 - changing DBEFile type, 7-8
 - cleanup after abnormal termination, 7-31
 - creating a DBEnvironments, 4-2
 - creating a hash structure, 5-10
 - creating a new log file, 6-27
 - creating an index, 5-11
 - creating a procedure, 5-12
 - creating a rule, 5-12
 - creating a table, 5-1
 - creating a view, 5-9
 - creating DBEFiles and DBEFileSets, 4-13
 - creating DBEnvironment security scheme, 4-18
 - creating log files, 4-4
 - creating logical and physical objects, 1-2
 - creating the database security scheme, 5-13
 - defining a database schema, 5-20
 - deleting a column, 7-11
 - deleting table rows, 7-10
 - displaying log file information with SHOWLOG, 6-21
 - dividing tables, 7-14
 - dropping a constraint from a table, 7-18
 - dropping a DBEFile, 7-9
 - dropping and recreating hash structures, 7-15
 - dropping and recreating indexes, 7-16
 - dropping and recreating procedures, 7-19
 - dropping and recreating rules, 7-19
 - dropping a table, 7-10
 - dropping modules, 7-26
 - enabling and disabling data definition, 7-30
 - enabling and disabling rule operation, 7-5
 - examining the system catalog, 4-7
 - granting and revoking procedure authorities, 7-18
 - indexes, 7-15
 - loading tables, 5-20
 - maintaining applications, 7-21
 - maintaining rules and procedures, 7-18
 - maintenance, 7-1
 - managing log files, 6-21
 - merging tables, 7-13
 - moving log files with MOVELOG, 6-26
 - online backup in archive mode, 6-13
 - purging log files with PURGELOG, 6-26
 - recovering log files with RECOVERLOG, 6-25
 - recovery in nonarchive mode, 6-12
 - reloading tables, 7-17
 - rescuing log files with RESCUELOG, 6-25
 - restoring log files with RESTORELOG, 6-25
 - revalidating procedure sections, 7-20
 - revoking table authorities, 5-15
 - rollforward recovery in archive mode, 6-16, 6-17
 - starting and stopping DBE sessions, 1-6
 - static backup in archive mode, 6-14
 - storing log files with STORELOG, 6-24
 - updating groups, 7-30
 - updating system catalog statistics, 7-6
- temporary index
 - and performance, 7-16
- tempspace
 - creating, 3-26
- TERMINATE QUERY, 4-28
- TERMINATE TRANSACTION, 4-28
- TERMINATE USER, 4-27
- terminating a DBE session
 - using RELEASE, 4-26
 - using STOP DBE, 4-27
 - using TERMINATE USER, 4-27
- third normal form, 2-6
- TIME
 - defining columns as, 2-11
 - storage requirements, 3-4
- timeout value
 - and DBECon file, 4-2
 - changing in SQLUtil, 7-2
 - default in DBECon file, 7-2
 - maximum, A-1
 - maximum in DBECon file, 7-2
- transaction
 - changing maximum in SQLUtil, 7-2
 - concurrent, 3-24
 - defined, 1-10
 - estimating number, 3-32
 - implicitly begun, 3-21
 - priority, 8-80
 - setting maximum number of, 7-2
 - shared memory for block buffer, 3-27
 - SYSTEM.TRANSACTION, 8-80
- transfer ownership
 - and CONNECT authority, 4-21
- TRANSFER OWNERSHIP, 7-23
 - syntax, 4-19
- transitive dependency, 2-6

TRUNCATE TABLE, 7-10

removing all rows, 7-29

tuple header, 2-11

TurboSTORE II

in SET BACKUP, F-37

types of DBEFile

table and mixed, 3-11

U

unavailable DBECon file

wrapperDBE, 6-31

unique constraint

creating, 5-5

entries in SYSTEM.CONSTRAINT, 8-23

entries in SYSTEM.CONSTRAINTCOL, 8-25

entries in SYSTEM.CONSTRAINTINDEX,
8-26

unique index

defined, 2-17

design, 2-18

guidelines for, 2-18

UNLOAD

creating files from tables, 5-20

for restructuring, 7-12

GENERATE LOAD, G-18

in deleting columns, 7-11

in maintenance operations, 7-2

internal, 7-13

unlock audit point

audit tool, I-22

UPDATE authority

and views, 5-15

explained, 5-13

listed in SYSTEM.COLAUTH, 8-15

updates

wrapperDBE, 6-31, 6-33

UPDATE STATISTICS

example, 7-6

invalidating sections, 7-23

on system views, 8-5

overview, 7-6

updating cluster count, 7-16

when to use, 7-6

usable log files

wrapperDBE, 6-29

user mode

adjusting parameter in DBECon file, 7-2

changing with SQLUtil, 7-2, F-4

displaying with SQLUtil, F-50, F-57

setting in DBECon file, 7-2

user timeout value

changing in SQLUtil, 7-2

setting in DBECon file, 7-2

V

VALIDATE, 7-22, 7-23

revalidating procedure sections, 7-20

validating

application program sections, 7-21

modules, 7-24

VARBINARY

defining columns as, 2-11

storage requirements, 3-4

VARCHAR

defining columns as, 2-11

storage requirements, 3-4

version checker, 1-25

view

and SYSTEM.SECTION, 8-64

as a stored section, 5-9

authorities, 5-13

creating:view, 5-9

data access, 2-14

design, 2-13

dropping, 7-10

granting authorities for, 5-16

maximum base tables per, A-1

maximum columns per, A-1

maximum per DBEnvironment, A-1

name, 5-9

owner, 5-9

restructuring with, 7-13

revoking authority for, 5-15

security applications, 5-15

WITH CHECK OPTION, 5-9

virtual table, 2-13

W

WITH CHECK OPTION

view, 5-9

WITH GRANT OPTION clause

examples, 5-17

WRAPDBE

syntax, F-76

wrapperDBE, 6-29, 6-32

wrapperDBE

archive logging, 6-28

audit information, 6-28

audit parameters, 6-31

audit tool, 6-28

command mode, 6-32

console log file, 6-30, 6-31

DBECon file, 6-29, 6-31

DDL statements, 6-31

dual log files, 6-29

inactive log files, 6-32

log files, 6-28, F-76

log files, number of, 6-32

- not usable log files, 6-30
- orphaned log files, 6-28
- roll forward, 6-28
- sequence number, 6-29, 6-32
- SHOWDBE command, 6-31, 6-33
- SHOWLOG command, 6-29
- single log files, 6-29

- START DBE NEW, 6-31
- unavailable DBECon file, 6-31
- updates, 6-31, 6-33
- usable log files, 6-29
- WRAPDBE command, 6-29, 6-32
- writeahead logging
 - explained, 1-7

