Concurrency Control In HP SQL

Ragaa K. Ishak

Distributed Data Management Lab
Hewlett-Packard
19447 Pruneridge Avenue
Cupertino, CA 95014

ABSTRACT

Hewlett-Packard's proprietary relational database management product HP SQL provides data integrity, and high concurrency between users. In any system that allows a high number of concurrent users, there is always the possibility of two or more users trying to update the same data simultaneously. HP SQL deals with the issue of concurrency and data integrity by using locks to restrict a user's access to data. The use of locks, which is handled by a lock manager, is fundamental to HP SQL's ability to ensure data integrity.

This paper describes the functionality and internal design of the locking mechanism in HP SQL. This paper is aimed at readers who want to write better applications using the full strength of HP SQL and readers who want to write diagnostic tools. This knowledge is crucial for designing high throughput applications and useful tools.

This paper assumes that the reader is familiar with HP SQL, however starters may still find it informative. The reader is assumed to be familiar with HP SQL concepts and specifically with the HP SQL structures such as tables, rows, indexes, DBEfiles, and DBEfilesets.

1. Introduction

HP SQL can support up to 240 concurrent transactions. Concurrency in HP SQL is controlled by using locks. Implicit as well as explicit locks are supported by HP SQL. Different types of locks are used depending on the type of table created, lock commands, and options specified when a transaction begins. Locks are set on user tables, as well as internal tables. The locks involved in SQL operations are described in section 3. Compatibility rules and conversion rules tell us what operations can be concurrent. The compatibility rules for locks are discussed in section 4. The conversion rules for locks are discussed in section A look is taken behind the scenes

at how concurrency is managed. The internal design of locking inside HP SQL is described in section 6. The different locks set during the different SQL commands are overviewed in section 7. Finally, the presented information will be used in drawing some conclusions pertaining to performance in section 8.

2. Overview of HP SQL Structure, Role of DBCORE

HP SQL consists of two major subsystems, SQLCORE and DBCORE. Discussion of other components of HP SQL is outside the scope of this paper. HP SQL uses a lock manager to synchronize access to data and manage locks. The lock manager in HP SQL is part of DBCORE.

DBCORE performs the basic DBMS functions of data definitions, data access, transaction management, concurrency control, logging and recovery, and accounting. In this paper, we are concerned with the concurrency control services that DBCORE provides with HP SQL.

SQLCORE translates SQL commands into an access plan. The access plan contains a sequence of DBCORE calls to define, store, and retrieve data. SQLCORE uses an optimizer to generate the most effective plan to retrieve data. The access plan consists of one or more DBCORE calls.

3. Lock Types

In this section, the lock types supported by HP SQL are described. Lock granularity, and intention locks are introduced.

a. Lock Granularity

To synchronize access to data, READ locks and WRITE locks are set. The type of table created determines the locking unit during read and write operations. For example a table of type PRIVATE is locked exclusively at the table level whenever accessed for read or write. A table of type PUBLICREAD is locked at the table level when doing a write operation and also locked at the table level when doing a read operation. A table of type PUBLIC is usually locked at the page level when doing read or write operations. READ locks can be shared but WRITE locks are exclusive. In summary, the locking unit on a user table is a page or the table itself.

Throughout this paper a lockable unit will also be referred to as an *object*. An object, a locking unit, has a *granularity* level. A table in HP SQL consists of one or several pages. The granularity of a page, then, is finer than the granularity of a table. In the same way, the granularity of a row is finer than the granularity of a page.

b. Intention Locks

When an object needs to be locked, several locks are set. When a lock is set on a relatively finer granularity object say a page, then an intention lock is set on the coarser granularity objects containing this object such as the table containing this page.

To illustrate the need for intention locks, consider the following scenario. Suppose that a given transaction T wishes to process some large table A, and also T requires A to be stable, i.e. it cannot tolerate changes in A by a concurrent transaction. Suppose also that T itself does not wish to make changes in A either. T can achieve the desired stability in A by obtaining a share lock on A. On receipt of T's Share lock request, HP SQL must be able to tell whether any other transaction is making changes to A, i.e., whether any other transaction already has an eXclusive lock on any page in A. If so, then T's request cannot be granted at this time. How can HP SQL detect this conflict ?

It is undesirable to have to examine every page in the large table A to see whether any one of them is X locked by any transaction, or to have to examine all existing eXclusive locks to see whether any one of them is for a page in A. Instead, intent locking is used. The need for intent locking is even more obvious when locks are set at the row level. *Intention* locks are used to speed up compatibility checking and to reduce deadlocks.

1) To lock a page in table T for read (page P)
   - lock table T with mode (IS)
   - lock page P with mode (S)

2) To lock a page in table T for write (page P)
   - lock table T with mode (IX)
   - lock page P with mode (X)

3) To lock a page in table T for read/potential write (page P)
   - lock table T with mode (IX)
   - lock page P with mode (SIX)

4) To lock row for read (row T)
   - lock table containing T with mode (IS)
   - lock page containing T with mode (IS)
   - lock row T with mode (S)

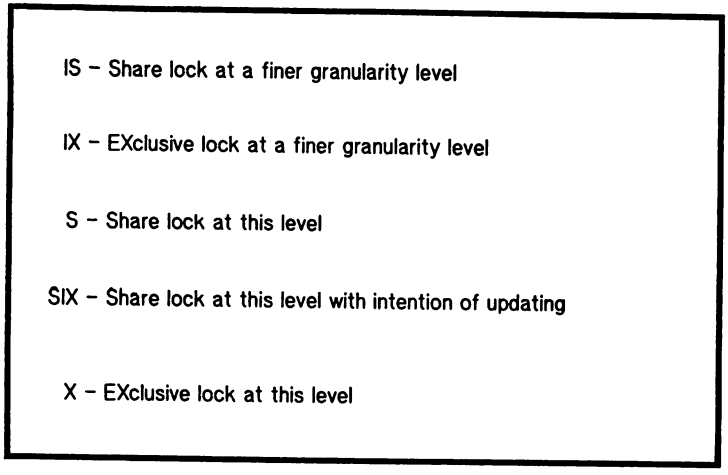
5) To lock row for write (row T)
   - lock table containing T with mode (IX)
   - lock page containing T with mode (IX)
   - lock row T with mode (X)

Intention Locks Generated

Figure 1

Figure 1 lists all the locks that are generated on an object during different types of operations. Note that row level locks are not currently supported on user tables.

c. Lock Summary



HP SQL Locks

Figure 2

Figure 2 lists all the locks supported by HP SQL. In summary, HP SQL uses five types of locks: Share (S), eXclusive (X), Intent Share (IS), Intent eXclusive (IX), and Share Intent eXclusive (SIX). An SIX, set by a transaction on an object allows multiple readers at finer granularities of this object, and disallows writers except the first transaction. More on the usage of SIX locks can be found in the section "compatibility between different locks". Note that the locks IS and IX cannot be set by the user. IS and IX are set internally by HP SQL to facilitate lock management. Row level locks are set on internal tables as described in a later section.

4. HP SQL Locking Modes

The user can control the concurrency level in HP SQL in three ways. First, when the table is created, a table type can be specified. Second, when a transaction is started, an isolation level can be specified. Third, in the course of a transaction, an explicit lock command can be specified. Implicit locks take effect when no explicit lock commands are specified.

a. Table Types

One of three following options can be specified when a table is created:

I. CREATE TABLE PRIVATE which implies one user at a time.

II. CREATE TABLE PUBLICREAD which implies one writer or multiple readers allowed to access the table concurrently.

III. CREATE TABLE PUBLIC which implies multiple writers and multiple readers allowed to access the table concurrently.

Table Type	Read Locks	Write Locks
Private	Exclusive Table (X)	Exclusive Table (X)
Publicread	Share Table (S)	Exclusive Table (X)
Public	Share Table (IS) Share page (S)	Intent Exclusive Table (IX) Exclusive page (X)

Implicit Lock Summary Table

Figure 3

Figure 3 shows the implicit locks that are generated for the different types of user tables. The locks shown in the table are the locks generated on the user table itself. The locks shown are for read and write operations. Note that for a PUBLICREAD table, a Share lock is set on the table for read operations. Since a write lock on such a table is eXclusive, then individual page Share locks are not necessary. The combination of Share locks and eXclusive locks at the page level is not allowed.

In the case of PUBLIC tables, the locks shown during read operations assume an index scan. During an index scan, the address of the requested data is obtained from the index. The pages on which the data reside are accessed directly and Share locks need to be set only on these pages. If the index is not used, then the whole table is scanned to find the data. As a result, a share lock is set at the table level.

In addition to the locks mentioned in the table for read and write, SIX (Share with Intent eXclusive) locks can be placed at the page or table level. When read with intent to update operations are requested, SIX locks are set in HP SQL. For example the SQL command pair "DECLARE CURSOR ..FOR UPDATE, OPEN CURSOR.." is an intent to update operation. The SIX lock is also set at the table level with the command LOCK TABLE ..IN SHARE UPDATE MODE.

b. Explicit Locks

Explicit lock commands can be specified in HP SQL. The advantage of specifying a lock command on a table is to reduce the number of locks generated, and therefore reduce the overhead of managing these locks. An explicit lock command such as LOCK TABLE ..IN EXCLUSIVE MODE will place one lock on the table, thus eliminating multiple locks on several pages and eliminating intention locks. The LOCK TABLE ..IN EXCLUSIVE MODE reduces the concurrency on the table to one transaction but gives this transaction the exclusive access to the table, therefore inproving the performance of this transaction. An explicit lock on a table can also be used to prevent deadlocks. A table can be locked explicitly as EXCLUSIVE, SHARE, or SHARE UPDATE.

c. Isolation Levels

The user can also control concurrency by specifying an *isolation level* when issuing the BEGIN WORK command. An isolation level is a consistency level for the data seen by a transaction. An isolation level is the degree of isolation a particular database access has from other attempts to access the database. The isolation levels currently supported by HP SQL are *repeatable read* and *cursor stability*. With the repeatable read level of consistency, also the default, HP SQL ensures that if data are repeatedly read in a transaction, the same data will be seen. To achieve this consistency level, READ locks, as well as WRITE locks of course, are kept until the end of the transaction. Data seen by transaction T1 cannot be updated by transaction T2 until T1 is finished.

With the cursor stability level of consistency, data seen is guaranteed to stay the same only as long the data is being addressed. If data is read again in the same transaction, seeing the same data is not guaranteed. With cursor stability, more concurrency can be achieved, since transactions keep their READ locks for shorter periods of time. Note that, the cursor stability consistency level is also achieved with the repeatable read consistency level, but the repeatable read consistency level is not achieved with cursor stability.

A higher performance is always very desirable in a DBMS. One of the major factors contributing to performance is the concurrency level. A higher concurrency level can be achieved with a lower consistency level, i.e., isolation level. As a result, there is a great potential of a performance increase, when cursor stability is used over repeatable read, in a multi user environment. In a single user system, cursor stability reduces performance.

Isolation levels, lower than repeatable read and cursor stability can be used to increase concurrency. *Read committed* and *read uncommitted* are lower isolation levels, currently under investigations for future releases of HP SQL. With the read committed isolation level, data seen is guaranteed to be committed, but not the same while it is being addressed. READ locks are kept, only while the data is being fetched from the database. With the read committed isolation level, more concurrency can be achieved than in cursor stability, since transactions keep their READ locks for shorter periods of time.

With the read uncommitted isolation level, the DBMS ensures that no updates are lost, i.e., no one transaction can write over uncommitted changes of another transaction. However, one transaction can read uncommitted changes by another transaction. At this level, reading committed data, cursor stability, and consequently repeatable reads are not guaranteed in a transaction. On the other hand, a read only operation should minimally wait for a lock and minimally effect waiters.

d. Commit/Keep Cursor

Commit/Keep Cursor is an enhancement under investigation for future releases of HP SQL. Currently all locks, READ and WRITE, are released, and the cursors set to null when a transaction ends, i.e., COMMIT WORK or ROLLBACK WORK are issued. With the Commit/Keep cursor option, implicit locks set to guarantee the consistency levels repeatable read and cursor stability may be retained (Kept) across COMMIT WORK commands. A cursor position is kept across COMMIT WORK commands.

5. Compatibility Between Different Locks

The compatibility between the locks tells us what operations on common data can be concurrent. Figure 4 shows the compatibility matrix between the five different locks that can be placed on objects in the database. As the table shows, some locks are compatible, and some are not. For example, two Intent eXclusive(IX) locks are compatible. An Intent eXclusive (IX) lock is placed on a user table of type public, when the user is updating a page with an eXclusive lock. Another user may update another page with an eXclusive lock in the table and share the Intent eXclusive (IX) lock on the table. A Share (S) lock compatible with a (S) lock means that two user transactions can read the same pages in a table at the same time, and so on...

	IS	IX	S	SIX	X
IS	Y	Y	Y	Y	N
IX	Y	Y	N	N	N
S	Y	N	Y	N	N
SIX	Y	N	N	N	N
X	N	N	N	N	N

Lock Compatibility Matrix

Figure 4

a. SIX Lock Implications

Figure 4 shows that an SIX lock is compatible only with an IS lock. This compatibility implies that if an SIX lock is set on a user table, then only readers, with Share locks, are allowed at the page level. Remember, that a Share lock at the page level generates an IS lock at the table level. No writers are allowed to this table at any level. So, in effect, an SIX lock is weaker than an eXclusive lock at a certain level, because it allows readers at finer granularities. An X lock is, of course, exclusive. On the other hand, an SIX lock is stronger than an IX lock because an IX lock set at a certain level, allows writers in addition to readers, at finer granularities. If an SIX lock is set by a transaction, on a page in a user table, then it is equivalent to an eXclusive lock on the page. Remember, that currently, row level locks are not allowed on user tables. If row level locks were allowed on user tables, readers will be allowed at the row level when an SIX is set at the page level. To summarize the effects of an SIX lock, we can assert that it is stronger than an IX lock and weaker than an eXclusive lock at a certain level. An SIX lock at the finest granularity, such as row, is meaningless.

When an SIX lock is set at the page level, an IX lock is set at the table level. If an SIX lock is set, then when the transaction updates, the SIX lock is upgraded to eXclusive.

b. Wait For Lock Protocol

If a lock request is not compatible with an existing lock on the same object then the user transaction will wait until the lock can be granted.

One important fact to be aware of, is that even though the compatibility rules allow two concurrent share locks on the same object, a second share lock might not be instantly granted for another transaction. If a Share lock is granted, an eXclusive lock request is waiting first in line, a second Share lock request will wait second in line.

The other fact is that, if a first transaction T1 is granted a Share lock, a second transaction T2 is waiting for an eXclusive lock, and the first transaction decides to convert from Share to eXclusive, then the converter is given priority for the eXclusive lock. If a Share lock is already granted, then the converter will have to wait. A reader who intends to update may request an SIX lock instead of Share to disallow granting Share locks on the same object, and eliminate the wait for lock conversion.

The application developer should keep the above protocols in mind while designing an application.

5. Conversion Between Different Locks

When changing the type of operation on an object such as from read to write, a lock might need to be converted. There is also the case where an explicit lock needs to be specified to overrule the implicit locks.

	New Mode				
Old	IS	IX	S	SIX	X
IS	IS	IX	S	SIX	X
IX	IX	IX	SIX	SIX	X
S	S	IX	S	SIX	X
SIX	SIX	SIX	SIX	SIX	X
X	X	X	X	X	X

Lock Conversion Matrix

Figure 5

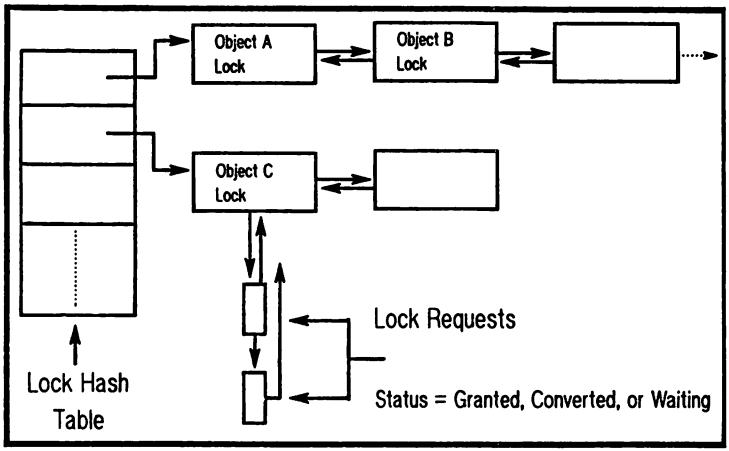
If a lock is already granted, then it can only be upgraded to a stronger lock. Figure 5 shows the lock conversion matrix. Note for example that an IS lock can be converted to an eXclusive lock, but not vice versa. As a result of the conversion rules, an explicit lock command can overrule the locks set on the table only if it is stronger. For example if a table is created as private and the command LOCK TABLE SHARE is issued, a read operation still effects an eXclusive lock on the table.

6. Concurrency Management

In this section, the design of the lock manager in HP SQL is presented.

a. Objects, Locks, And Lock Requests

Figure 6 shows the relationships between objects, locks, and lock requests. When the database is first started with the SQL commands START DBE NEW, START DBE, or CONNECT with autostart option on, a lock hash table is created. The size of the hash table is dependent on the number of transactions specified when the database is started. A lock is implemented as a control block. The hash table keeps lists of locks. Each object, i.e., a table, a page, a row, has a unique identification number in the database. This unique identification number is used to hash the object to be locked, and assign a lock to it. Each request to lock an object is a lock request. A lock request is also implemented as a control block. A lock request is associated with one and only one lock. However, there can be multiple requests to lock the same object.

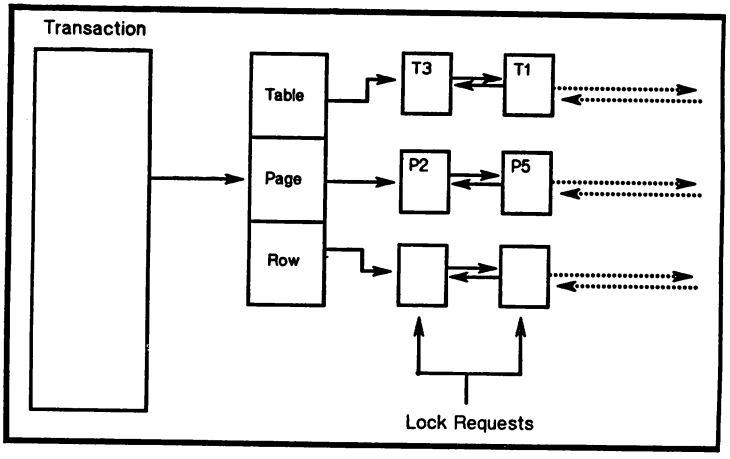


Lock Management

Figure 6

b. Transaction And Lock Requests

Figure 7 shows the relationship of a transaction to its lock requests. For each transaction, three lists of lock requests are kept. The first list is a list of lock requests for locks on tables. The second is a list of lock requests for locks on pages. The third is a list of lock requests for locks on rows. The status of a lock request can be either granted, waiting, or converting. For each scan, i.e., "FETCH" command, a list of READ lock requests granted is kept. In the case where cursor stability option is specified, this list is used to release the locks acquired during a previous "FETCH".



Transaction Lock Requests

Figure 7

c. Processing Of Lock Requests

I. When a lock is requested on an object, the lock requests issued by this transaction are checked first in the sequence table, page, and row.

II. If some lock request by this transaction already exists for the object, then the old lock request is converted according to conversion rules.

III. If the lock request for this transaction is not found, then compute a hash location for the object to be locked and the hash table is checked for it.

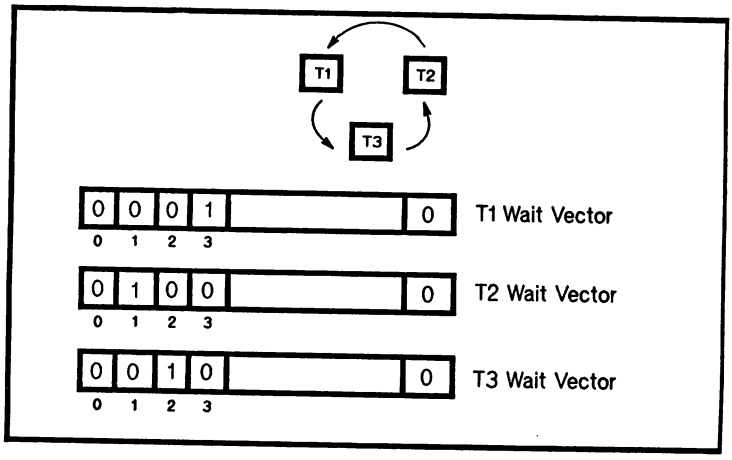
IV. If a lock for this object is not found in the hash table, a new lock is allocated and a lock is requested on the object.

V. If a lock is found, then it is determined if the lock requested is compatible with the existing granted lock. If compatible then it is granted, else place the new lock request in wait status.

VI. If a lock needs to be converted to a stronger lock, it is added to the wait list for this lock. Requests for read and write are given the same priority in the wait list. The lock wait vectors of waiting transactions are updated.

d. Deadlock Detection

For each transaction, a wait vector is assigned. A bit in the vector is assigned for every transaction in the database. If a transaction waits for a lock held by another transaction then the corresponding bit of the other transaction is set in the first's wait vector. HP SQL checks for deadlocks whenever a lock request causes a wait. Figure 8 shows an example of wait vectors for three transactions in the database. T2 is waiting for T1. T1 is waiting for T3. T3 is waiting for T2.



Transaction Wait Vectors

Figure 8

The wait vector is updated for all waiting transactions each time the status of locks changes, e.g. a lock is released. A deadlock with part of the cycle outside the HP SQL system cannot be detected. This might be an obvious assertion, still, application complexity increases the potential of such scenarios.

There are various deadlock avoidence protocols that can be used. For example, a scheme is to request all locks in advance. Using this scheme is not possible where a high level of concurrency is required and one of the following conditions exists:

I. A transaction T may be unable to identify row R1, and hence unable to lock it, until it has examined R2.

II. The set of lockable objects is very large, consisting possibly of thousands of pages, but it also changes dynamically.

III. The precise number of lockable objects is not known in advance.

IV. Records are addressed not by name, but by content, so that it cannot be determined until execution time whether or not two distinct requests are for the same object.

7. Locking Activities During SQL commands:

In previous sections we learned about lock types, locks set during different operations, locks set on user tables, concurrent locks, and lock management internals. In this section, a closer look is taken at all the locks that are generated during different operations. To complete the picture, all the tables involved in the DBE are overviewed.

a. Tables In The DBE

There are three types of tables involved in a Data Base Environment (DBE):

I. The tables that the user creates and uses to store data. These tables are referred to as the user tables. User tables are locked at the page level or the table level.

II. The SQL catalog tables that contains information about the user's DBE, such as names of tables, column names, column types, authorization groups, etc.. The SQL catalog tables are locked at the page level.

III. The DBCORE internal tables contain low level basic information about the data.

When a Data Base Environment (DBE) is first created four DBCORE tables are also created. They contain information about tables, DBEfiles, DBEfilesets, and indexes in the DBE respectively. The information contained in these tables is the minimum information needed to store and retrieve data. For example, these tables do not contain names for tables or columns, but instead they contain identification numbers. SQLCORE translates table names into identification numbers and column names into positions within a row. The SQL catalog tables keep all the information necessary to provide the user with a high level interface to data. The DBCORE tables are locked at the row level. READ or WRITE locks are set on these tables at the row level during the various operations involving data access.

b. START DBE Command

When the START DBE NEW command is issued, the DBCORE tables are first created, then the SQL catalog tables are created. Each SQL catalog table is created as a separate transaction, surrounded by BEGIN WORK and COMMIT WORK commands internally. No locks are held after a START DBE command is issued.

With any START DBE command, the lock control blocks are allocated, formatted, and the lock hash table is initialized.

c. Data Definition Language Commands

A Data Definition Language (DDL) commands causes a change in the structure of data. With a change in the structure of the data, the DBCORE tables and the SQL catalog tables need to be updated. Exclusive WRITE locks need to be set on one or more internal tables. Since READ locks can be shared but WRITE locks are not, DDL commands tend to reduce the concurrency in the DBE considerably. To maximize the concurrency in a production environment, DDL commands need to be scheduled.

To improve performance, HP SQL supports an option where DDL commands can be disabled. When the user chooses the option of disabling DDL, HP SQL can keep executable sections in memory, eliminating the time spent in repeatedly bringing sections from the SQL catalog. When DDL commands are enabled, sections can be invalidated. A section to be executed needs to be brought from the catalog each time so that it can be revalidated if neccessary.

d. Data Manipulation Language Commands

Data Manipulation Language commands (DML) change the data itself but not the structure of the data. The DBCORE and SQL tables are locked in Share mode to prevent concurrent DDL, i.e. concurrent data restructuring. Share page locks are set on the SQL catalog tables and Share row level locks are set on the DBCORE tables.

e. UPDATE STATISTICS Command

An UPDATE STATISTICS command on a table causes a scan of the DBEfileset containing the table to find the pages belonging to it. The number of pages are counted. Each page belonging to the table is then scanned and the size of each row is checked. The average size of a row in the table is then computed. Similar index information is computed for each index on the table. To guarantee the consistency of data during this operation, exclusive locks are set on the DBCORE tables for table and index information. Exclusive locks are set on the SQL catalog tables because they are updated. No other concurrent operation on the table is allowed during an UPDATE STATISTICS command. To allow other transactions to access the table, a COMMIT WORK needs to be issued. SQLCORE uses the statistics on a table to generate optimized access plans to retrieve data.

f. CHECKPOINT Command.

A CHECKPOINT command does not acquire any locks. All updated data pages are written to disc so access to the data buffer pool is prevented. Concurrency is reduced considerably during this operation, while buffers are written to disc and log file space is reclaimed, but no locks are held.

A complete list of the type of locks set on the SQL tables is in the DataBase Administration manual.

9. Conclusions Pertaining To Performance.

Different applications have different requirements. As a result different techniques may be used. Following are some guidelines for performance improvement. If no specific application need is mentioned then the guideline applies for most applications.

I. Keep transactions short, especially if they use DDL commands. As mentioned in the previous section, DDL commands effect exclusive page locks on the SQL tables. Concurrency is then reduced considerably. If DDL commands can be scheduled, then use the DDL DISABLED option.

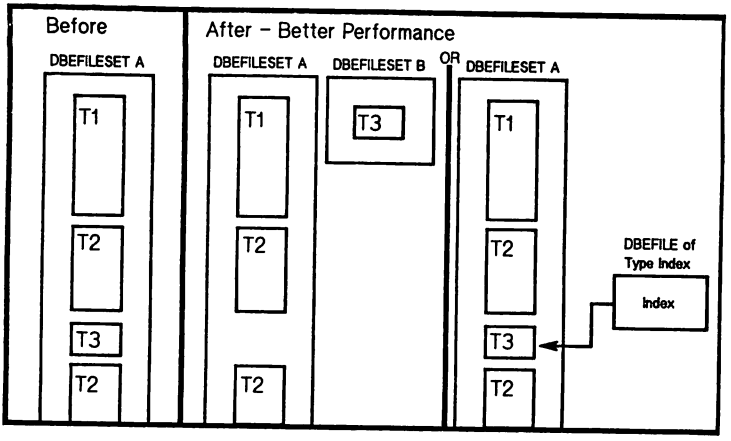
II. Specify SHARE UPDATE locks. This kind of lock increases concurrency on a table when a transaction needs to update many pages in the table and still allow other transactions to read.

III. Use the Cursor Stability option. Your READ locks are released as you read other pages so concurrency is increased. Your read performance decreases slightly to release no longer needed READ locks.

IV. The following guideline was learned from witnessing a benchmark application. The original performance was unsatisfactory. It turned out that the application was accessing a small table very often. Almost all transactions accessed this small table. The small table was stored in a DBEfileset with many other large tables. The left side of Figure 9 shows the "Before" placement of the small table. To access the table, a DBEfileset scan is performed to locate the page(s) owned by the small table. This is a slow process considering that there are alternatives to increase the concurrency in such a system. The right side in Figure 9 shows the "After" alternatives. The small table can be placed in a separate DBEfileset to speed up the process of finding it. The second alternative is to create an index for this table. The index contains the exact location (pages) owned by this table. For faster access, the index can be placed in a separate index type DBEfile.

V. Reduce I/O while holding locks. Terminal and disc I/O may consume unexpected amount of time.

VI. To increase the performance of an individual transaction, use locks at the table level. A lock at the table level reduces the overhead of intention locks, locks on each page accessed, and consequently lock management.



High Access Small Table Allocation

Figure 9

9. Summary

HP SQL is HP's relational database management system. HP SQL was designed to support a high level of concurrency and still maintain data integrity. As the knowledge of the concurrency mechanism in HP SQL increases, the throughput of applications developed can be maximized. To achieve this goal, the functionality and design of locking were discussed. The different locks involved in SQL operations were described. Compatibility as well as conversion rules and their effects were studied. The design of the lock manager, and the processing of lock requests were presented. The locking activities on all objects in a database were overviewed for the different SQL commands. Finally, some performance notes were summarized. This paper has hopefully presented clearly the functionality and design of concurrency control in HP SQL. HP SQL is currently available on HP-UX, MPE/XL, and MPE/V operating systems.

Acknowledgments

The author wish to thank Scott Walecka for his prompt and invaluable multiple reviews of this paper. The author would also like to thank Mary Loomis, Emmanuel Onuegbe, Frank Dean, Dave Wilde, and Alex Carlton for providing numerous comments to improve the presentation of this paper.