The Advance of HP SQL - Towards the OLTP Market

Alberto Lutgardo

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

Abstract

One of the attributes of an Online Transaction Processing system is to provide to the user a high level of throughput. The purpose of this paper is to explain to the user how to increase the throughput of a given application by the use of the Isolation Level concept, and how this concept was implemented in the HP SQL product. This paper shows the relationship between the concept of Isolation Level and the concept of consistency levels.

Isolation Level is a property of data base access from the point of view of concurrency control while the level of consistency is defined as the effects the user experiences due to the concurrency control. There are three Isolation Levels that stand among all. The Repeatable Read is the HP SQL default. It implies that all locks acquired by a transaction are held until transaction commits, rolls back, or it is terminated by the system due to deadlock. Cursor Stability holds the read locks only until after a subsequent read occurs. Read Uncommitted does not acquire locks for read operations; in other words, it only provides physical consistency. Write locks are the same for all isolation levels and are held until transaction commits or rolls back.

1. Introduction

HP SQL is a transaction oriented system; therefore, it will provide a high level of transaction throughput. This is vital for a product to be competitive in the OLTP market. An increase of concurrent access to common data increases the transaction throughput in the system. There are two ways to increase concurrency to common data. The first is to relax the locking scheme by using weaker locks, and the other one is to release

the locks on common data as soon as possible. The HP SQL product has implemented the concept of Isolation Level to achieve this purpose. The HP SQL product provides the option of Isolation Levels to allow the user to control the concurrency of his/her applications.

This paper assumes that the reader is familiar with the concepts of concurrency control as defined in [2] and also familiar with HP SQL concurrency control mechanism as described in [5].

The concept of Isolation Level in the HP SQL product applies only to PUBLIC tables. A PUBLIC TABLE is a table that allows multiple writers and multiple readers concurrently. Cursor Stability is one of the isolation levels implemented in the HP SQL product that releases read locks before the transaction is committed. Cursor Stability releases read locks between SQL FETCH commands. The feature is provided for applications that scan a table in one direction.

The organization of the rest of the paper is as follow: Section 2 provides the HP SQL definition for Isolation Level. Section 3 describes the implementation of Isolation Level within the HP SQL product. Section 4 describes an application that takes advantage of the Isolation Level concept to increase the throughput of the system. Section 5 describes some performance results. Section 6 offers some conclusions, and it describes new features that HP SQL is planning to provide to the user in the near future to be even more competitive in the OLTP environment.

2. Isolation Level Definition

The concept of Isolation Level has been reported as a level of consistency in [2,3,4]. Level of consistency is defined as the kind of anomalies that can occur in the system during the execution of concurrent transactions. Some of these anomalies are:

- "dirty read": a transaction T1 can read a tuple that has been just created by another transaction T2 which is aborted after T1 has read the tuple.
- "non-repeatable read": transaction T1 is not guaranteed to get the same tuple if it re-reads it.
- "phantom read" transaction T1 is not guaranteed that it will get the same set of tuples matching a given search condition if it re-reads the table using the same search condition

The level of consistency for 0, 2, and 4 are defined in [4]. The user might experience all three anomalies when he uses consistency level 0, with consistency level 2 the user might experience non-repeatable reads and phantom reads, and with consistency level 4 the user will not experience any anomaly.

HP SQL defines Isolation Level as a property of an access to a table, defined in the data base, through a cursor from the concurrency point of view. A cursor in the HP SQL is an address of some specific tuple in a table. A cursor is associated with a type of access to a table. The HP SQL product provides internally index access and sequential access. A cursor can be defined explicitly by the user through the SQL DECLARE command or implicitly by the HP SQL product. There are eight isolation levels of which The HP SQL

product currently support five. The relationship between Isolation Level and level of consistency is shown in Figure 1.

A description of each Isolation Level and an example of how to use it using the "accounts" PUBLIC table are provided below. Let us assume that the "accounts" table has four columns: account number, account name, debit, and credit. Each example shows the type of locks that the HP SQL product acquires on behalf of the transaction. However, in order to describe the concurrency between transactions, two concepts must be introduced first. READ ONLY TRANSACTION is an HP SQL feature under investigation in which updates are not allowed, while in a READ-WRITE TRANSACTION, retrievals and update are allowed. Only the READ-WRITE transaction is currently supported by the HP SQL product.

The definition of each Isolation Level is based on the set of tuples that the transaction retrieves through a cursor and the protection that this transaction provides to the set from the concurrency point of view. A cursor is associated with the type of scan done to a table. There two types of scans, index scan and relation scan.

Isolation Level	Level of Consistency
Repeatable Read with Intent Update	4
Repeatable Read	
Phantom Read with Intent Update	3
Phantom Read	
Cursor Stability with Intent Update	2
Cursor Stability	
Read Committed	1
Read Uncommitted	0

Figure 1

2.1 Repeatable Read

The set of tuples obtained through a cursor, matching the search condition, within a transaction can not be updated by another transaction; however, it can be read by another READ-WRITE transaction. No data can be added to the table that matches the search condition of the cursor through another transaction. Therefore, the same set of tuples are generated if the data is re-read using the same cursor.

```
EXEC SQL BEGIN WORK RR;

EXEC SQL SELECT * FROM accounts WHERE
                    credit > 1500.00;

EXEC SQL UPDATE accounts SET credit = credit*1.1
                    WHERE debit = 0;

EXEC SQL COMMIT WORK;
```

In order to explain the type of locks that the HP SQL product acquires on behalf of the SQL SELECT command, we can loosely say that the SQL SELECT command is divided internally into a) open scan, b) several internal FETCH commands, and c) close scan.

- Index scan: Subshare lock on the "accounts" table is acquired by the HP SQL product at open time. When the internal FETCH command is executed, a share lock is acquired by the HP SQL product for index and data pages. Once the SQL UPDATE command is executed, the subshare lock in the "accounts" table is promoted from subshare to subexclusive. Besides, for each updated page, there is a promotion of the lock from share to exclusive.

- Relation scan: Share lock is acquired by the HP SQL product on the "accounts" table at open time. When the internal FETCH command is executed, no locks are acquired by HP SQL product. Once the SQL UPDATE command is executed, the share lock on the "accounts" table is promoted to share subexclusive.

- Concurrency: Other READ/WRITE transactions can be executed concurrently with the above transaction between the SQL SELECT and the SQL UPDATE commands. Deadlocks can occur in this scheme during lock promotion.

- Usability: This Isolation Level should be used if the set of tuples to be retrieved will *not* be updated within the transaction. The user should either specify or let it default to the RR option in the SQL BEGIN WORK command. If the user only wants to retrieve and not update the data using the SQL FETCH command, the user *should not* use the FOR UPDATE clause in the SQL DECLARE command.

2.2 Repeatable Read with Intent Update

The set of tuples obtained through a cursor, matching the search condition, within a transaction can not be updated by another transaction. Also, this set of tuples can not be read by another transaction unless it is a READ ONLY TRANSACTION and the data has not yet been modified. READ ONLY TRANSACTION is an HP SQL feature under investigation in which updates are not allowed within the transaction. No data can be added to the table that matches the search condition of the cursor through another transaction. Therefore, the same set of tuples are generated if the data is re-read using the same cursor.

```
EXEC SQL BEGIN WORK RR;

EXEC SQL DECLARE c1 CURSOR FOR SELECT * FROM accounts WHERE
                    credit > 1500.00 FOR UPDATE OF credit;

EXEC SQL OPEN c1;

EXEC SQL FETCH c1 INTO :hostvar;

EXEC SQL UPDATE accounts SET credit = :credit*0.1 WHERE CURRENT OF c1;

EXEC SQL CLOSE c1;

EXEC SQL COMMIT WORK;
```

- Index scan: Subexclusive lock on the "accounts" table is acquired by the HP SQL product at open time. When the SQL FETCH command is executed, share locks are acquired for non-leaf index pages, and share subexclusive locks are acquired for leaf index pages and data pages.

- Relation scan: Share subexclusive lock is acquired by the HP SQL product on the "accounts" table at open time. When the SQL FETCH Statement is executed, share subexclusive locks are acquired by the HP SQL product on data pages that match the search condition (credit > 1500.00)

- Concurrency: Only READ ONLY transactions can execute concurrently with the above transaction between the SQL FETCH and SQL UPDATE commands. READ-WRITE transactions can execute concurrently with the above transaction if the READ-WRITE transaction is doing only retrieval on a different subset of tuples than the above transaction and it is using index scan. Once the SQL UPDATE command is executed, the lock on each updated page is promoted from share subexclusive to exclusive. Promotion of a lock from share subexclusive to exclusive does not generate a deadlock.

- Usability: This Isolation Level should be used if the set of tuples to be retrieved will be updated within the transaction. The user should either specify or let it default to the RR option in the SQL BEGIN WORK command and should use the FOR UPDATE clause in the SQL DECLARE command.

2.3 Phantom Read

The set of tuples obtained through a cursor, matching the search condition, within a transaction can not be updated by another transaction; however, it can be read by another transaction. Data can be added to the table that matches the search condition of the cursor through another transaction. *The HP SQL product does not yet support this feature.*

2.4 Phantom Read with Intent Update

The set of tuples obtained through a cursor, matching the search condition, within a transaction can not be updated by another transaction. Also, this set of tuples can not be read by another transaction unless it is a READ ONLY TRANSACTION (updates are not allowed within the transaction). Data can be added to the table that matches the search condition of the cursor through another transaction. *The HP SQL product does not yet support this feature.*

2.5 Cursor Stability

The set of tuples obtained through a cursor, matching the search condition, within a transaction can not be updated by another transaction while the cursor has addressability to the set of tuples; however, it can be read by another READ-WRITE transaction. Data can be added to the table that matches the search condition of the cursor through another transaction.

```
EXEC SQL BEGIN WORK CS;

EXEC SQL SELECT * FROM accounts WHERE
                    credit > 1500.00;

EXEC SQL UPDATE accounts SET credit = credit*1.1
                    WHERE debit = 0;

EXEC SQL COMMIT WORK;
```

In order to explain the type of locks that the HP SQL product acquires on behalf of the SQL SELECT command, we can loosely say that the SQL SELECT command is divided internally into a) open scan, b) several internal FETCH commands, and c) close scan.

- Index scan: Subshare lock on the "accounts" table is acquired by the HP SQL product at open time. When the internal FETCH command is executed, share locks are acquired for the leaf index pages and for data pages containing tuples that match the search condition. No locks are acquired for the non-leaf index pages, and non-exclusive locks are released between internal FETCH commands.

- Relation scan: Subshare lock is acquired by the HP SQL product on the "accounts" table at open time. When the internal FETCH command is executed, share locks are acquired by the HP SQL product for each page that contains tuples that match the search condition. Non-exclusive locks are released between internal FETCH commands.

- Concurrency: READ-WRITE transactions can be executed concurrently with the above transaction. Once the SQL UPDATE command is executed, for each page that is updated, the lock is promoted from share subexclusive to exclusive. Deadlocks can occur in this scheme during the promotion of a lock.

- Usability: This Isolation Level should be used if the set of tuples to be retrieved will *not* be updated within the transaction, and the user will not re-read the data again within the transaction. The user should specify the CS option in the SQL BEGIN WORK command and should *not* use the FOR UPDATE clause in the SQL DECLARE command.

- *Warning:* Since cursor stability holds locks only for the current FETCH command, phantom rows can show up for the above transaction when another transaction inserts or updates tuples containing the credit column that matches the search condition of the above SQL SELECT command. Therefore, if the user tries to re-execute the SQL SELECT command within the same transaction, the user might not get the same set of tuples. More important, a set of tuples previously read may be updated by another transaction and if it is re-read some of the tuples would contain new data. In other words, repeatable reads are not guaranteed by the HP SQL product when the user uses cursor stability. Furthermore, a loss of serializability can occur among transactions since locks are released before the transaction is committed. The execution of a set of transactions is said to be serializable if and only if it produces the same result as some serial execution of those same transactions. To guarantee serializability locks should only be released at the end of the transaction; otherwise, a loss of serializability occurs [2]. For example, a loss of serializability might occurs if transaction T2 is waiting for resource R1 held by transaction T1, and transaction T1 releases resource R1 before T1 commits, then it is possible for transaction T1 to wait for transaction T2 sometime before transaction T1 is committed. The user is also advised not to use the scanned data to update other tables of the data base since serializability among transactions is lost.

2.6 Cursor Stability with Intent Update

The set of tuples obtained through a cursor, matching the search condition, within a transaction can not be updated by another transaction while the cursor has addressability to the set of tuples. This set of tuples on which the cursor has addressability can not be read by another transaction, unless it is a READ ONLY TRANSACTION and the data has not been modified yet. Data can be added to the table that matches the search condition of the cursor.

```
EXEC SQL BEGIN WORK CS;

EXEC SQL DECLARE c1 CURSOR FOR SELECT * FROM accounts WHERE
                    credit > 1500.00 FOR UPDATE OF credit;
```

```
EXEC SQL OPEN c1;

EXEC SQL FETCH c1 INTO :hostvar;

EXEC SQL UPDATE accounts SET credit = :credit*0.1 WHERE CURRENT OF c1;

EXEC SQL CLOSE c1;

EXEC SQL COMMIT WORK;
```

- Index scan: Subexclusive lock on the "accounts" table is acquired by the HP SQL product at open time. When the SQL FETCH command is executed, a share subexclusive lock is acquired by the HP SQL product for the leaf index page, and for data pages that contain a tuple that matches the search condition. No locks are acquired for the non-leaf index pages, and non-exclusive locks are released between two SQL FETCH commands.

- Relation scan: Subexclusive lock is acquired by the HP SQL product on the "accounts" table at open time. When the SQL FETCH Statement is executed, share subexclusive locks are acquired by the HP SQL product for each page that contains tuples that match the search condition. Non-exclusive locks are released between two SQL FETCH commands.

- Concurrency: Only READ ONLY transactions can be executed concurrently with the above transaction between the SQL FETCH and SQL UPDATE commands. READ-WRITE transactions can execute concurrently with the above transaction if they are working in a different set of tuples. Once the SQL UPDATE command is executed, for each page that is updated, the lock is promoted from share subexclusive to exclusive. Since share subexclusive locks are compatible only with intent to share lock, there is not deadlock during the promotion of a lock.

- Usability: This Isolation Level should be used if the set of tuples to be retrieved will be updated within the transaction, and the user will not re-read the data again within the transaction. The user should specify the CS option in the SQL BEGIN WORK command and should use the FOR UPDATE clause in the SQL DECLARE command.

- *Warning:* The cursor stability warning applies to this Isolation Level.

2.7 Read Committed

The set of tuples obtained through a cursor, matching the search condition, within a transaction does not have any locks associated with it, when the data is presented to the user. Locks are acquired during the data retrieval to guarantee that only committed data is read.

```
EXEC SQL BEGIN WORK RC;
```

```
EXEC SQL DECLARE c1 CURSOR FOR SELECT * FROM accounts WHERE
                credit > 1500.00;

EXEC SQL OPEN c1;

EXEC SQL FETCH c1;

EXEC SQL CLOSE c1;

EXEC SQL COMMIT WORK;
```

- No locks are held when the data is returned to the user regardless of the type of scan. The Read committed option guarantees that the set of tuples is committed and it is not being updated at the instant that HP SQL retrieves the set of tuples. Subshare lock is used on the "accounts" table, and share locks are used on the leaf index pages and data pages; however, all locks are released before the retrieved data is exposed to the user.
- Concurrency: READ-WRITE transactions can be executed concurrently with the above transaction.
- Usability: This Isolation Level should be used if the set of tuples to be retrieved will not be updated within the transaction and the user will not re-read the data again within the transaction. The user should specify the RC option in the SQL BEGIN WORK command and should use the FOR UPDATE clause in the SQL DECLARE command.
- *Warning:* The cursor stability warning applies to this Isolation Level plus data being looked at may be updated. Data should not be updated based on reading with read committed.

2.8 Read Uncommitted

The set of tuples obtained through a cursor, matching the search condition, within a transaction does not have any locks associated with it. No locks are acquired during data retrieval; therefore, the uncommitted data may be read. *The HP SQL product does not yet support this feature.*

3. HP SQL Isolation Level Implementation

This section describes the implementation of the Isolation Level concept. The major challenge in the implementation of the Isolation Level concept was to answer the following questions:

- Which type of lock should be acquired?
- Which locks should be released before the transaction is committed?
- When the locks should be released?

In order to answer the above questions, the HP SQL Lock Manager was modified by building relationships between the cursor and the lock requests. In other words, the implementation of the Isolation Level concept was only possible with the help of the lock manager, since the control of concurrency among transactions is done by the lock manager.

3.1 Type of lock acquired by the HP SQL product

For each new lock requested, the lock manager allocates a lock control block on behalf of the transaction. The type of lock depends on the Isolation Level that the user has specified in the SQL BEGIN WORK command, and whether or not the user has specified the FOR UPDATE clause in the SQL DECLARE command. Intent to write locks are used for any update command (i.e. SQL DELETE and UPDATE commands). The HP SQL product, depending on the Isolation Level, might release non-exclusive locks before the transaction is committed.

3.2 Which locks to release

In order to know which lock control blocks the Lock Manager has to release, the Lock Manager builds two linked lists to remember what was locked in the previous FETCH command, and what is being locked in the current FETCH command. Both linked lists are chained off the scan control block. A scan control block is created by the HP SQL product whenever a cursor is opened. Therefore, when the Lock Manager is requested to release locks, it gets the locks associated with the previous FETCH command by finding the appropiate linked list pointed by the scan control block.

3.3 When to release locks

A lock request keeps an Isolation Level flag and a counter in each lock control block in order to know when to release a lock. The Isolation Level flag is defined to allow the Lock Manager to release a lock before the transaction is committed. The counter is defined in order to know when to release the lock. If the Isolation Level flag is off, the lock is not released until the transaction is committed. The counter is incremented each time the page is locked on behalf of a cursor (scan control block) and it is decremented whenever the lock manager is requested to release the lock. The lock is released when the counter becomes zero, and the isolation flag is set to true.

4. Benchmark Example

In order to measure the performance gain of an application that uses the concept of Isolation Level, a banking transaction benchmark was designed and implemented. The benchmark simulates a banking system where tellers update bank accounts on-line (done by a foreground process). The generation of reports and bookkeeping work are done in background (done by three background processes).

The benchmark accesses the following tables defined within the data base.

- Accounts Table. It has four columns: account-number, account-name, debit, and credit. A unique index is defined on the account-number column.
- Tellers Table. It has three columns: teller-number, teller-name, and branch-number. A unique index is defined on the teller-number column.
- Branches Table. It has four columns: branch-number, branch-name, debit, and credit. A unique index is defined on the branch-number column.
- History Table. It has seven columns: voucher-number, account-number, teller-number, action, amount, date, and status. A unique index is defined on the voucher-number column and another unique index is defined on date column.
- Voucher Table. It has only one column named voucher-number.

Descriptions of the transactions for the foreground process and background processes are explained in the following paragraphs.

4.1 Transactions for the foreground process

The foreground process executes three types of transactions to simulate a banking system application. The first transaction executes an update of an account in the "accounts" table when money is withdrawn. The second transaction generates a small report on the status of each account. The third transaction increments the voucher number in the voucher table.

4.1.1 First transaction for the foreground process

The client enters the account number, and the teller number and the branch number are entered by the banking system where the client withdraws the money. The execution of the withdrawal transaction involves four steps.

- update to the "debit" field for a given account number in the "accounts" table.

- get a "branch_number" for a given "teller_number" in the "tellers" table.
- update "debit" field for a given "branch_number" in the "branches" table.
- insert a tuple in the "history" table to indicate that an action was done.

4.1.2 Second transaction for the foreground process

The second transaction generates a report for all accounts that are between a range of values provided by the banking system.

4.1.3 Third transaction for the foreground process

The third transaction gets the voucher number from the voucher table and increments it by one (1).

4.2 Transactions for the first background process

The first background process executes two types of transactions. The first transaction uses cursor stability to retrieve data from the "history" table. The second transaction updates the "debit" field on the "accounts" table, and updates the "status" field in the history table.

4.2.1 First transaction for the first background process

The first transaction retrieves tuples from the "history" table which have a value of zero (0) in the "status" field. In this example, a scan on the whole table is done; however, concurrency among transactions is allowed, since cursor stability is used. Cursor stability only holds locks between two SQL FETCH commands, and the BULK SELECT command is broken down internally as a set of calls. Loosely speaking, each call internally executes a FETCH command.

If repeatable read were used instead of cursor stability, serialization among transactions would have happened because a share lock on the "history" table would have prevented writers from accessing the table.

4.2.2 Second transaction for the first background process

The second transaction executes two updates. The first update is done to the "debit" field in the "accounts" table, and the other update is done to the "status" field in the "history" table.

4.3 Transactions for the second background process

The second background process executes two types of transactions. The first transaction uses cursor stability during the retrieval of tuples from the "history" table. The second transaction deletes tuples from the "history" table that match a given voucher number.

4.3.1 First transaction for the second background process

The first transaction scans the whole table to find tuples which have a status value of one (1) in the "history" table. Since cursor stability is used, concurrency among transactions is allowed during the scan of the "history" table. Cursor stability only holds locks between two SQL FETCH commands, and the BULK SELECT command is broken down internally as a set of calls. Loosely speaking, each call internally executes a FETCH command.

If repeatable read were used instead of cursor stability, serialization among transactions would have happened because a share lock on the "history" table would have prevented writers from accessing the table.

4.3.2 Second transaction for the second background process

The second transaction deletes from the "history" table all tuples that match a given voucher number.

4.4 Transactions for the third background process

The third background process executes only one transaction using cursor stability. It produces a report from the "account" table. Cursor stability only holds locks between two SQL FETCH commands, and the BULK SELECT command is broken down internally as a set of calls. Loosely speaking, each call internally executes a FETCH command.

If repeatable read were used instead of cursor stability, serialization among transactions would have happened because a share lock on the "history" table would have prevented writers from accessing the table.

5. Performance Results

Two metrics were used to measure the performance of the HP SQL product. The first one was response time, and the second one was the number of transactions executed per hour.

The following table describes results demostrated by the benchmark when it was run on the on a HP-840 series, with HP-UX release 2.0, 24 Mbytes of memory, and 1 swap space. The benchmark used a 13 secnd delay, and only the third background process was run.

Using only Repeatable Read				
# of users	# of TranX	Ave. Elap(sec)	TranX/Hr	# of Deadlocks
1	50	36199.76	5	0
3	150	35427.99	15	0
5	250	37848.40	24	0

Using Cursor Stability and Repeatable Read with Intent Update				
# of users	# of TranX	Ave. Elap(sec)	TranX/Hr	# of Deadlocks
1	50	1241.38	145	0
7	350	1238.94	1017	0
13	650	1222.57	1914	0
17	850	1282.34	2667	0
23	1150	1280.15	3234	0

Figure 5

The above performance table shows us that the throughput is about 20 times higher when the benchmark uses cursor stability and repeatable read with intent update instead of repeatable read only. Better results have been obtained when the concept of disabling data definition was introduced. Disabling data definition implies that no data definition operations are allowed to execute concurrently with the normal data operations. Performance results using the disabling data definition feature can be found in [1].

6. Conclusions and Future Work

It is clear, from Section 5, that the HP SQL strategy, of allowing the user to specify the isolation level, allows the user to see a tremendous increase in throughput of the system. HP SQL allows the user to control the concurrency of his applications to generate fewer locks and better concurrency, because the user knows the semantics of the data. The Isolation Level concept was implemented to allow the user to control the concurrency of his application programs. The benchmark described in Section 4 was used to show that repeatable read with intent update and cursor stability have provided an increase of throughput 20 times better than the case in which only repeatable read is used.

Online transaction processing applications are characterized by large numbers of users concurrently accessing and updating very large data bases. HP SQL has implemented a flexible synchronization scheme that coordinates the accesses to the data base, and provides a good crash recovery mechanism that preserves the logical and physical consistency of the data base. Section 3 described the modification done to the Lock Manager to achieve a high level of concurrency without perturbing the logical and physical consistency of the data base.

In order to be even more competitive HP SQL is investigating, as a further concurrency enhancement, providing read uncommitted for users who only need physical consistency. Furthermore, the HP SQL product is investigating a feature that allows the user to commit a transaction, thereby releasing both read and write locks, but preserves the scan position for use in the subsequent transaction.

Acknowledgments:

Dora Lee and Edward Cheng were key players in the implementation of cursor stability. Birgit Luebke wrote the benchmark. Birgit Luebke with the help of Edward Cheng were able to collect the performance data. Discussions with Frank Dean and Emmanuel Onuegbe helped me to clarify the concept of Isolation Level. Alex Carlton provided valuable editorial help in the draft version of the paper. The author is also grateful to Sena Palanisami, Scott Walecka, and Sam Prather who supported this implementation.

References

[1] Edward Cheng. Performance in HP SQL. INTEREX, August 1988.

[2] C.J. Date An Introduction to Database Systems Volume II. Addison-Wesley Publishing Company.

[3] J.N. Gray, R.A. Lorie, G.R. Putzolu, I.L. Traiger. Granularity of Locks and Degrees of Consistency in a Shared Data Base. Modeling in Data Base Management Systems. G.M. Nijssen, (ed.) North Holland Publishing Company, 1976 (pages 365-392)

[4] Jim Melton ISO-ANSI (working draft) SQL2. Jim Melton (ed.), April 1988. (pages 23, 24, 190)

[5] Ragaa K. Ishak. Concurrency Control in HP SQL. INTEREX, May 1988.