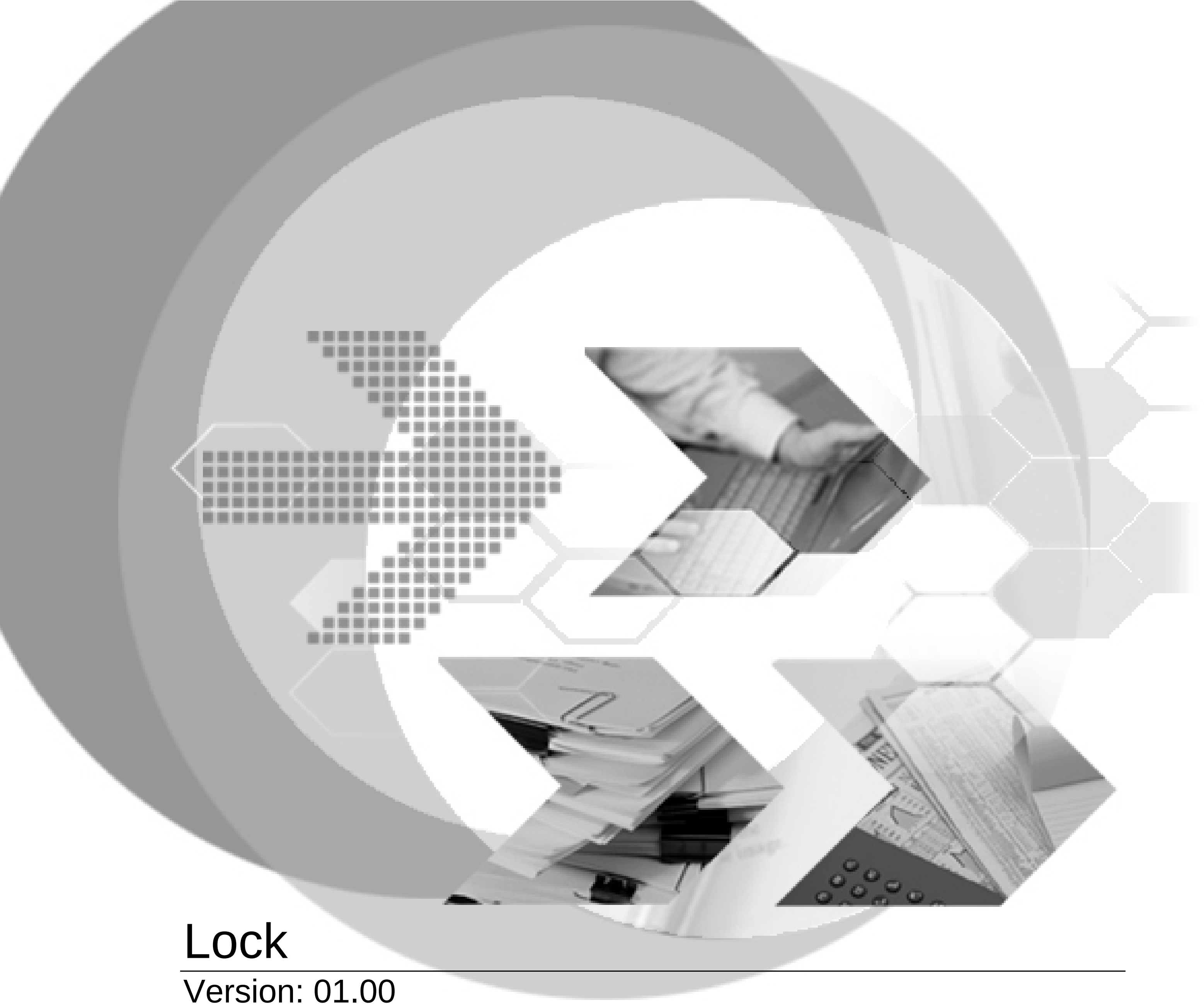


Lock

Version: 01.00

Document NO: 43/DBM43-T01202006-01-LOIG
Author: DBMaker Production Team Aug 11, 2005
LingAn Computer Engineering CO.

Print Date: Aug 08, 2005

Table of Content

1. Lock Concept

In general, a multi-user database system uses several forms of locking to synchronize the access of concurrent transactions. Before accessing the data objects, such as tables and tuples, a transaction must lock those data objects.

DBMaster locking is fully automatic and does not require any user action. Implicit locking occurs in all SQL statements; the users do not need to explicitly lock any data objects in the database.

- DBMaster will lock objects automatically during operation
- Lock is released when transaction commits or aborts
- When rolling back a savepoint, DBMaster also release the locks required after this savepoint
- Basic Lock Type

Share Locks (S) and Exclusive Locks (X).

1.1 Shared And Exclusive Locks

In general, two types of locking are used to allow multiple-read with single-write operations in a multi-user database.

- **Share Locks (S)**— A transaction involving a read operation on a data object. To support a higher degree of data concurrency, several transactions can acquire share locks on the same data object at the same time.
- **Exclusive Locks (X)**— A transaction involving an update operation on a data object. This transaction is the only one that can access the object until the exclusive lock is released.

2. Lock Granularity

There are three granularity levels for data locks in DBMaster: relation (table), page, and tuple (row). A relation contains several pages, and a page contains several tuples.

A lock applied on a higher level carries through to lower levels. For example, if a user gets an exclusive lock (X lock) on a relation, all pages and tuples that are included in this relation will have the X lock applied to them. Therefore, no user can access any tuple or page from this relation. However, if a user gets an X lock on a tuple, another user can get an X lock on another tuple simultaneously. There is no interference between two objects at the same level when using the X lock. Figure 1- shows the lock granularity (levels) in DBMaster.

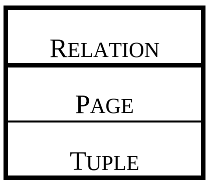


Figure 1-1: Lock granularity

Using a higher lock granularity results in a lower degree of data concurrency, in contrast, the higher lock granularity uses fewer system resources (such as shared memory). Selecting the lock granularity level is a trade-off between concurrency and resources. In DBMaster, the default lock granularity level is page, but if a different lock granularity is required, it can be specified when creating a table.

Affection:

- Decision of lock mode is a trade-off between concurrency degree and resource.
- Rowlock gains the highest concurrency, but spends the most resource.
- One special lock level ‘system’ denotes the lock on catalogs such as table, index, file, and tablespace. The system lock maintains the concurrency of DDL.

2.1 Lock Mode

The *lock mode* of a table identifies the type of lock that DBMaster automatically places on objects when accessing the database. DBMaster supports three lock mode levels: TABLE, PAGE, and ROW. The PAGE lock mode is used by default if the lock mode is not specified when a table is created. If the lock mode is set to a higher level (such as TABLE), the level of concurrency on database accesses will be lower, but the required lock resources (shared memory) will also be smaller. If the lock mode is set to a lower level (such as ROW), the level of concurrency on database accesses will be higher, but the required lock resources (shared memory) will be larger. In other words, if a user inserts or modifies rows in a table with the lock mode set to TABLE, no one else will be able to access the table. The reason for this is that an exclusive lock is taken on the entire table.

- **Example**

To specify the lock mode on a table:

```
dmSQL> CREATE TABLE employee (nation CHAR(20) DEFAULT ‘R.O.C’,
                                  ID INTEGER NOT NULL,
                                name CHAR (30) NOT NULL,
                            JoinDate DATE DEFAULT CURDATE (),
                              height FLOAT,
                              degree VARCHAR(200)) IN ts1
                    LOCK MODE ROW;
```

2.2 Changing the Lock Mode

To gain a higher level of concurrency on simultaneous connections to a database, set the lock mode to a lower level (such as a **ROW** lock). However, doing this causes DBMaster to expend more resources; deciding which lock mode to use on a table always involves a trade-off.

- **Example**

To change the lock mode for the **employee** table:

```
dmSQL> ALTER TABLE employee SET LOCK MODE ROW;
```

3. Lock Types

The main lock modes (types) supported in DBMaster are shared (S) and exclusive (X) locks. More than one user can have an S lock on a data object simultaneously, but only one user can have an X lock on a data object. In addition to S and X locks, another lock mode called an *intention lock* is supported.

When a data object is locked, the system will automatically assign an intention lock to the next higher granularity object. For example, an S lock specified on a tuple will generate an intention S (IS) lock on the page which includes this tuple, and an IS lock on the relation which the tuple belongs to.

The supported intention lock modes are:

- **IS—**Indicates that the S lock is specified at a lower granularity.
- **IX—**Indicates that the X lock is specified at a lower granularity.
- **SIX—**Indicates that an S lock is specified at the current granularity and an X lock is specified at a lower granularity. This is a combination of S and IX locks.

The result from the compatibility of each of the lock modes is listed in Table 1-1. T represents true, which means the matrix for each of the two lock modes are compatible and can exist on a data object simultaneously. F represents false, which means the matrix for each of the two lock modes are not compatible and cannot exist simultaneously.

If lock requests on a data object conflicts with an existing lock on that object, this request will not execute until the existing lock is released, or until the waiting time for the lock request times out. If the error message 'Lock timeout' is returned to the user, the waiting time for the lock has expired. The default waiting time is 5 seconds. However, users can specify a different waiting time by setting the value of the **DB_LTimO** keyword in the **dmconfig.ini** file to another value according to their individual requirements.

	IS	S	IX	SIX	X
IS	T	T	T	T	F
S	T	T	F	F	F
IX	T	F	T	F	F
SIX	T	F	F	F	F
X	F	F	F	F	F

Table 1-1: Compatibility matrix for lock modes

- **Example 1**

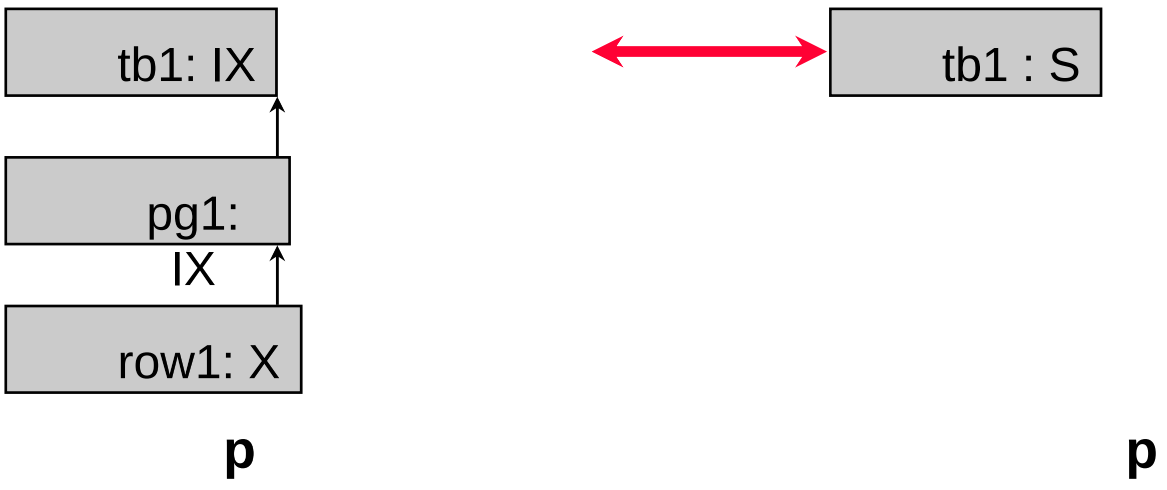
The following shows how to set the waiting time to 8 seconds:

```
DB_LTimO = 8;
```

- **Example 2**

Two processes p1, p2:

```
P1: update tb1 where c1 = 1
P2: select * from t1
```



p2 will be waiting status for the p1 release lock.

3.1 Locking Tables

Although DBMaster automatically handles the lock mechanism whenever a database is accessed, a table may be manually locked for subsequent SELECT or UPDATE statements. Locking a table while a user is viewing or modifying it will prevent updates by other people.

DBMaster supports some options for locking tables, such as *shared locks* for viewing data or *exclusive locks* for modifying data, and the WAIT or NO WAIT option which is used when obtaining a lock.

- **Example**

To lock the **employee** table for later selections and not wait if it cannot get the table lock right away:

```
dmSQL> LOCK TABLE employee IN SHARE MODE NO WAIT;
```

4. Deadlock

When two or more transactions are waiting for the release of data locked by other transactions before it can proceed, a deadlock occurs.

- **Example 1**

T1 is waiting for T2 to release the share lock of X, while T2 is waiting for T1 to release the share lock of Y. Therefore, deadlock occurs and the system will wait indefinitely:

```
   T1                  T2
-------------        -------------
share_lock(Y);
read(Y);
                    share_lock(X);
                    read(X);
exclusive_lock(X);
(T1 waits for T2)     exclusive_lock(Y);
                    (T2 waits for T1)
```

- **Example 2**

Table t1 (c1 int, c2 char (10)), no index

Two processes p1, p2:

```
          p1                        p2
==================== ====================
T1: select * from t1
T2:                      select * from t1
T3: delete from t1(wait)
T4:                      delete from t1 (wait)
                    ERROR (11421): deadlock
```

4.1 Dealing with Deadlock

There are four methods to Prevent/Avoid Deadlock:

- Set table's lock mode to row
- Reduce unnecessary indexes or index columns
- Access tables in sequence*e.g., two tables t1, t2, always update t1 before t2*
- Shorten the transaction

***Note:** a multi-process application must process time out or deadlock error handling.*

- **By analyzing the "wait for" graph, DBMaster can automatically detect a deadlock situation. If a deadlock is detected, a victim transaction will be aborted to solve the deadlock problem. The victim transaction is the last transaction DBMaker will sacrifice**

Example

DBMaster detects a deadlock when transaction T2 issues an X lock on Y. Transaction T2 will be aborted to resolve the deadlock problem and the user executing transaction T2 will receive the error message, "transaction aborted due to deadlock":

```
     T1                    T2
--------------          ------------
share_lock(Y);
read(Y);
                     share_lock(X);
                     read(X);
exclusive_lock(X);
(T1 waits for T2)       exclusive_lock(Y);
                     (T2 waits for T1)

                     T2 aborted by DBMaster
```

4.2 Lock Escalation

DBMaster can escalate some lower locks to a higher lock to speed up performance and reduce consumed lock resource. When the locks belong to the same higher level object and the locks number reach the specific threshold, DBMaster will escalate them to a higher lock granularity automatically.

User can set the lock escalation threshold or turn off lock escalation.

- DB_LETPT: page to table lock escalation threshold, default 50 (32767 means no escalation)
- DB_LETRP: row to page lock escalation threshold, default 15 (larger than 255 means no escalation)

4.3 Lock Waiting Rule

- **DB_LTimO:** lock time out value

0: no wait

-1: wait without time out

1~65535: wait seconds, default 5

- **First in first out**

u1 S, u2 X, u3 S => get lock sequence is u1, u2, u3.

FIFO can avoid starvation. In the above example, three transactions were waiting for the lock of the same object. If supposed sequence of asking for lock is u1,u2, u3,then get lock sequence is u1,u2,u3. So u1 will get lock firstly,u2 and u3 will continue waiting.

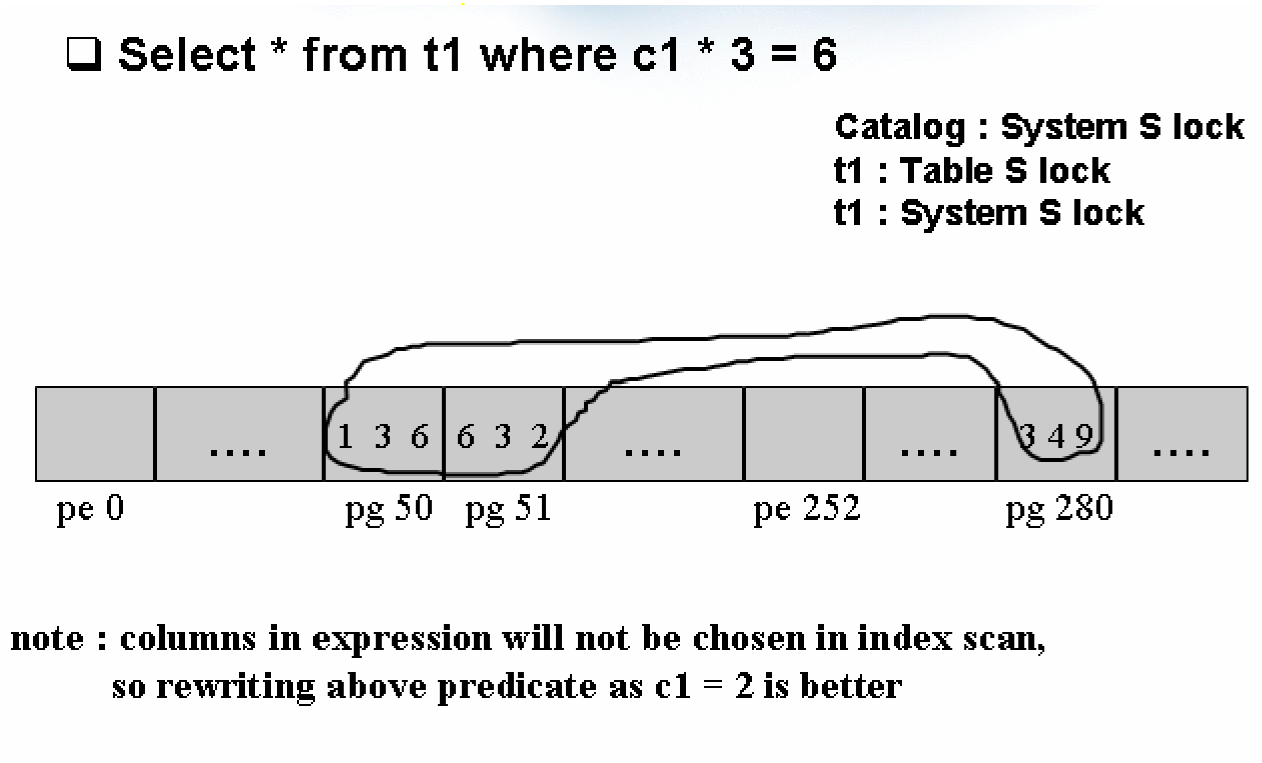
5. Lock On Different DMLs

The chapter will introduce lock on different DMLs. To understand following examples, you need know some basic conditions, such as page50, page51 and page 280 are data pages of t1 for storing table data, lock mode of table t1 is row lock. These conditions are fit to all following examples.

5.1 Lock On Select (Table Scan)

- **Example**

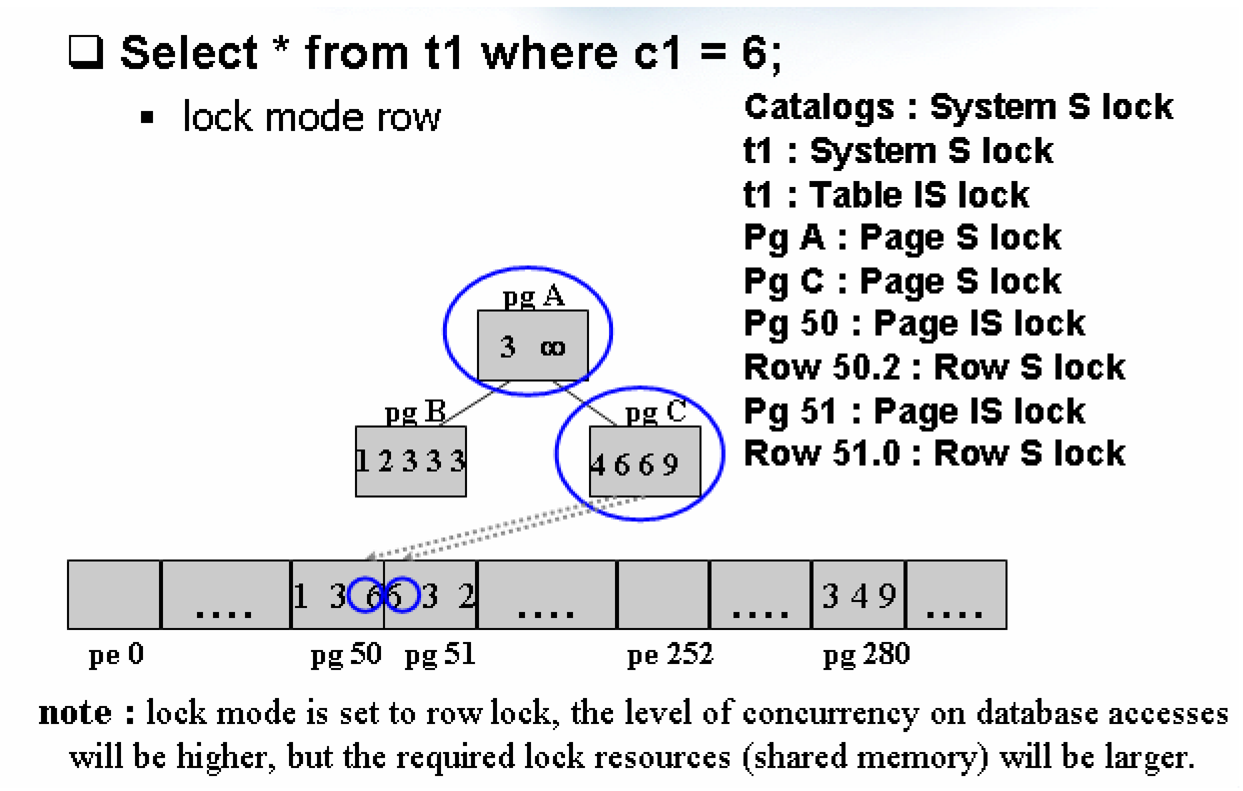
*Select * from t1 where c1 * 3 = 6*


❑ Select * from t1 where c1 * 3 = 6
Catalog : System S lock
t1 : Table S lock
t1 : System S lock
....
1 3 6
6 3 2
....
....
3 4 9
....
pe 0
pg 50
pg 51
pe 252
pg 280
note : columns in expression will not be chosen in index scan,
so rewriting above predicate as c1 = 2 is better

5.2 Lock On Select (Index Scan)

- **Example**

*Select * from t1 where c1 = 6;*



❑ Select * from t1 where c1 = 6;

- lock mode row

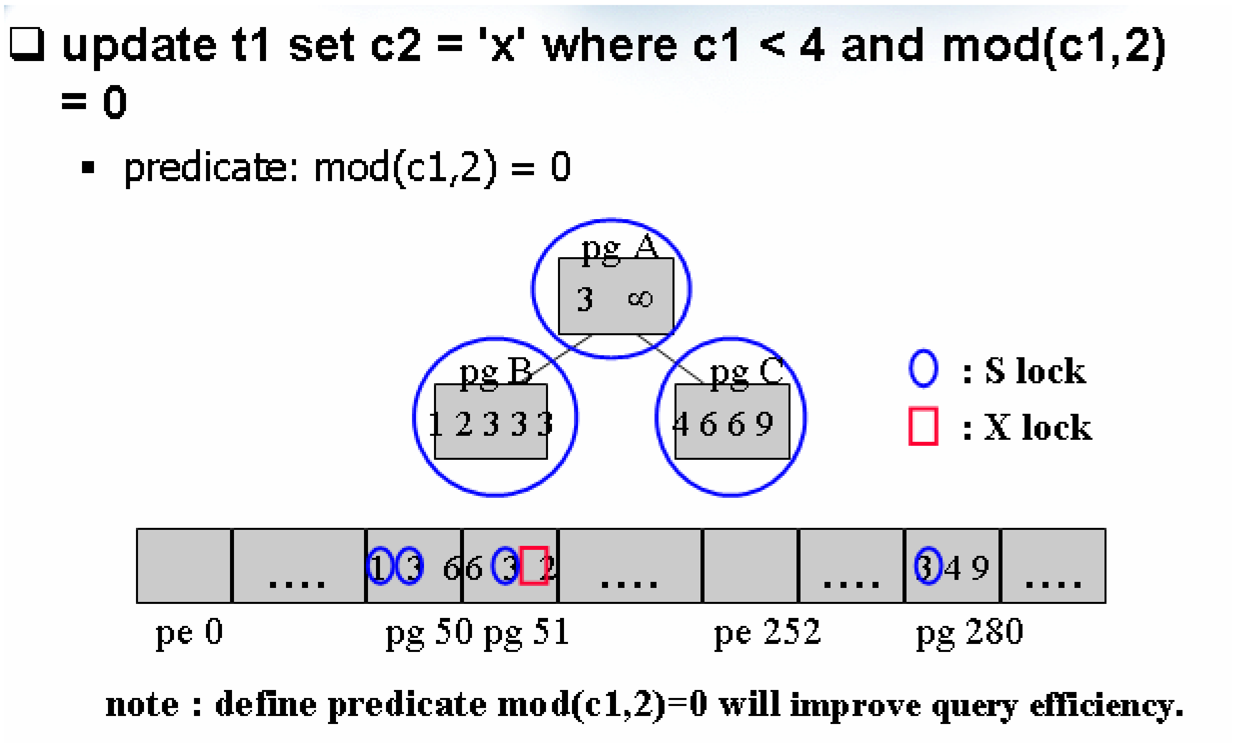
Catalogs : System S lock
t1 : System S lock
t1 : Table IS lock
Pg A : Page S lock
Pg C : Page S lock
Pg 50 : Page IS lock
Row 50.2 : Row S lock
Pg 51 : Page IS lock
Row 51.0 : Row S lock

note : lock mode is set to row lock, the level of concurrency on database accesses will be higher, but the required lock resources (shared memory) will be larger.

5.3 Lock On Update (Index Scan)

- **Example**

```
update t1 set c2 = 'x' where c1 < 4 and mod(c1,2) = 0
```



❑ update t1 set c2 = 'x' where c1 < 4 and mod(c1,2) = 0

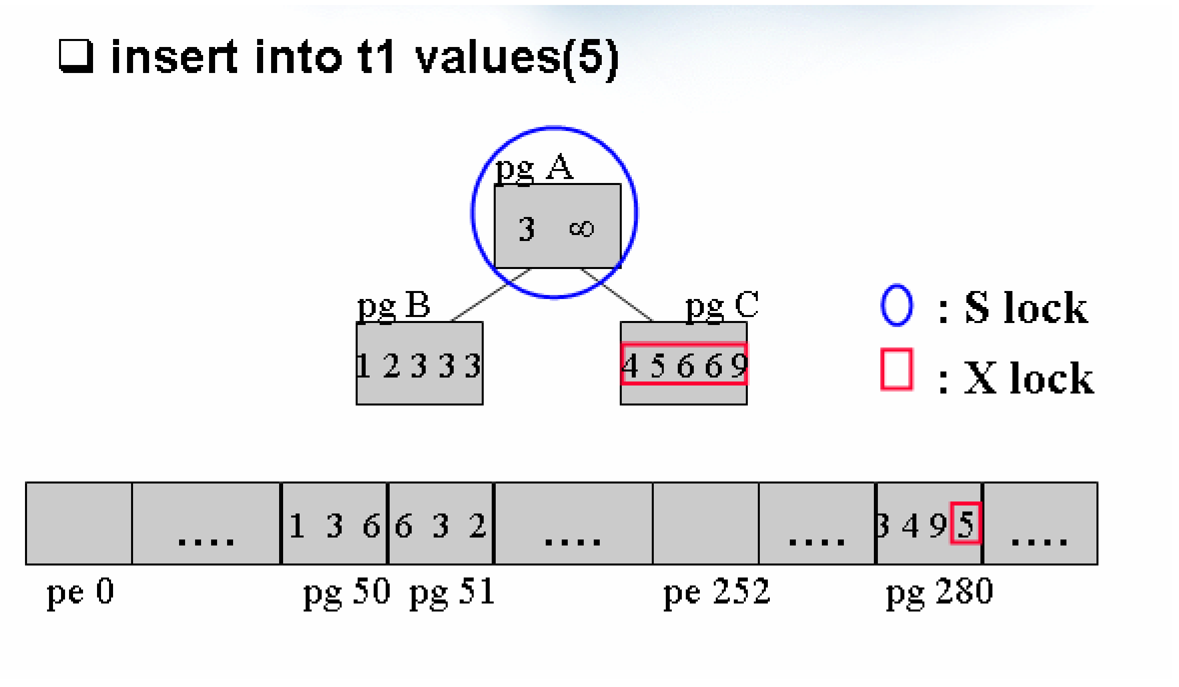
- predicate: mod(c1,2) = 0

note : define predicate mod(c1,2)=0 will improve query efficiency.

5.4 Lock On Insert

- **Example**

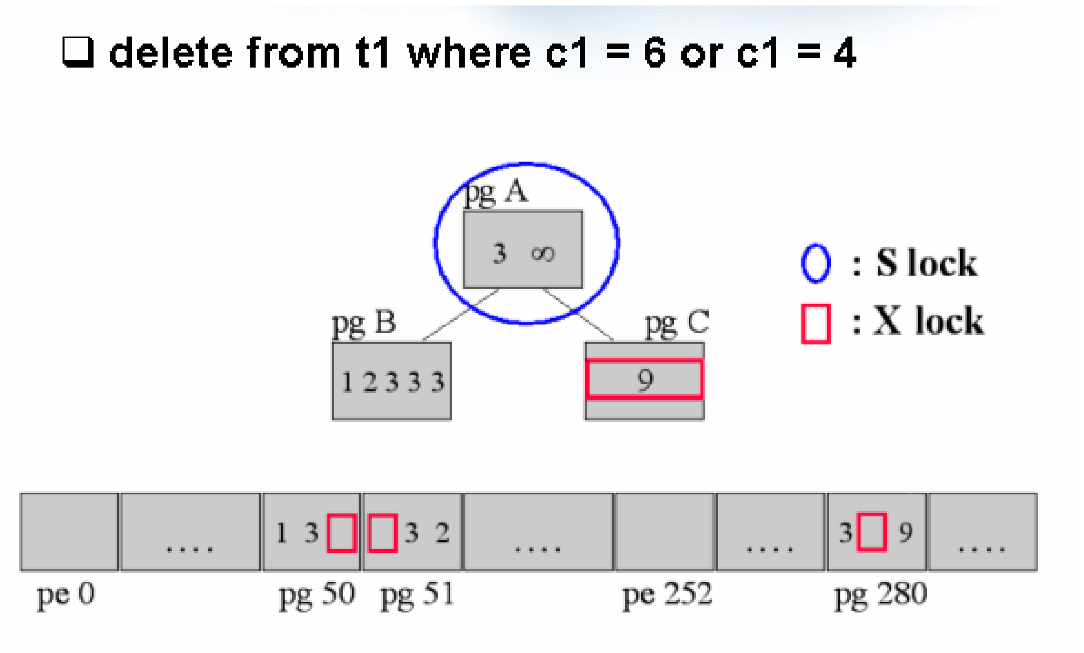
insert into t1 values(5)



5.5 Lock On Delete

- **Example**

delete from t1 where c1 = 6 or c1 = 4



5.6 DB link lock behavior

When executing SQL commands by DB Link, the behavior of lock on different DMLs is almost similar with the mentioned above. But they have two differences:

1. **Position of lock** – Position of Lock is on the Server side of the DB Link. So the local client couldn't know information of db link lock on the remote Server. If you want to see information of db link lock, you must link to the remote side (Server side). You can also query the SYSLOCK table of the remote side by DB link. The SYSLOCK table contains information on status for locks on objects.
2. **Release Locks -** If the local transaction that evokes the DB Link ended (commit / abort/ Close Links…etc), locks correlative with the remote side would be released.

6. Monitor Lock

Columns of SYSLOCK table:

- **LK_OBJECT_ID:** oid of locked object
- **TABLE_ID:** table oid of locked object
- **LK_GRAN:** lock granularity (TABLE, PAGE, TUPLE, SYSTEM)
- **HOLD_LK_CONNECTION:** connection id
- **LK_STATUS:** lock status (GRANTED, WAITING, CONVERT)
- **LK_CURRENT_MODE:** current mode of lock (S, X, IS, IX, SIX)
- **LK_NEW_MODE:** required mode of lock

- **Example**

```
dmSQL> select USER_NAME, TABLE_NAME, SYSLOCK.* from
        SYSLOCK,SYSTEM.SYSTABLE,SYSUSER where HOLD_LK_CONNECTION =
        CONNECTION_ID and LK_OBJECT_ID = TABLE_OID and TABLE_OWNER !=
        'SYSTEM' and LK_GRAN != 'SYSTEM';
```

7. Reducing Lock Contention

- Check lock contention frequency
- Downgrade lock level
- Set BROWSE mode

When accessing data in a database, DBMaster processes will lock the target objects (records, pages, tables) automatically. When two processes want to lock the same object, one must wait. If more than two processes wait for the other processes to release the lock, a deadlock occurs. When a deadlock occurs, DBMaster will sacrifice the last transaction that helped cause the deadlock by rolling it back. Deadlock reduces system performance. Monitor lock statistics to avoid a deadlock in DBMaster.

- **Example**

To view deadlock statistics:

```
dmSQL> select INFO, VALUE from SYSINFO where INFO = 'NUM_LOCK_REQUEST' or INFO =
          'NUM_DEADLOCK' or INFO = 'NUM_STARTED_TRANX';
INFO                                              VALUE
 ================================     ===========================
NUM_STARTED_TRANX                                                    9287
NUM_LOCK_REQUEST                                                      772967
NUM_DEADLOCK                                                            181

3 rows selected
```

NUM_LOCK_REQUEST—the number of times a lock was requested.

NUM_DEADLOCK—the number of times deadlock occurred.

NUM_STARTED_TRANX—the number of transactions that have been issued.

If the deadlock frequency is high, examine the schema design, SQL statements, and applications. Setting the table default lock mode lower, such as ROW lock, could reduce the lock contention, but it will require more lock resources.

Another method is to use the browse mode to read a table on a long query if the data does not need to remain consistent after the point in time that it was retrieved. This is useful when viewing the data or performing calculations using the data while not performing any updates. It provides a snapshot of the requested data at a particular point in time, but with the benefit of increased concurrency and fewer lock resources consumed, because locks are freed as soon as the data is read.

7.1 For Brows Mode

- **Select Command**

```
dmSQL> select * from t1 for browse;
```

DBMaker will not lock the result set of select statement.

```
dmSQL> select * from t1 for update;
```

In default, DBMaker will take S lock on the result set of select statement. If setting value of DB_forUX to 1, DBMaker will take X lock on the result set of select statement. DB_forUX is used in server side.

```
dmSQL> select * from t1;
```

In default, value of **DB_Brows** is 1 and DBMaker will not lock the result set of select statement. If setting value of **DB_Brows** to 0, DBMaker will take S lock on the result set of select statement. **DB_Brows** is used in client side.

- **DB_Brows**

This keyword specifies the lock behavior of a select statement. Setting the value to 0 denotes DBMaker will take S lock on the result set of select statement, and 1 denotes DBMaker will not lock the result set of select statement. This value is required while connecting to a database.

0: S lock on the result set

1: not lock the result set (dirty read)

default value: 1

valid range: 0,1

where to use: client side

8. Transaction isolation levels and lock behavior

DBMaster 4.2 only supports Read Uncommitted and Serializable isolation levels. Read Committed and Repeatable Read isolation levels will be supported in DBMaster 4.3. The lock behaviors in the two isolation levels in the DBMaster 4.2 are different. We discuss them in the later sections. (note that the table's lock mode is to "row" in the following discusses).

8.1 Read Uncommitted isolation level

8.1.1 WITHOUT INDEX

For select statement, transactions get records by using dirty read. So all the records in the result set will not be locked with any locks.

For insert statement, transactions lock the inserted records with X locks and lock the pages and the table with IX locks.

For delete and update statement, transactions lock the whole table with S lock first. Then lock the updated records with X locks and updated pages with IX locks. Finally, lock the table with IX lock and translate it to SIX lock. (S+IX)

8.1.2 WITH INDEX

For select statement, transactions get records by using dirty read. So all the records in the result set and index pages will not be locked with any locks.

For insert, delete and update statement, transactions lock the index non-leaf pages with S locks and leaf pages with X locks. Then lock the updated records with X locks and lock pages and table with IX locks.

8.2 Serializable isolation level

8.2.1 WITHOUT INDEX

For select statement, transactions lock the whole table with S lock.

For insert statement, transactions lock the inserted records with X locks and lock the pages and table with IX locks.

For delete and update statement, transactions lock the whole table with S lock first. Then lock the updated records with X locks and lock the updated pages with IX locks. Finally, lock the table with IX lock and translate it to SIX lock. (S+IX)

8.2.2 WITH INDEX

For select statement, transactions lock the records with S lock and lock the index pages (include non-leaf and leaf pages) with S locks.

For insert statement, transactions lock the inserted records with X locks and lock the pages and table with IX locks. Then lock the index non-leaf pages with S locks and lock leaf pages with X locks.

For delete and update statement, transactions lock the index non-leaf pages with S locks and leaf pages with X locks. Then lock the updated records with X locks and lock updated pages and table with IX locks.