Introduction

– We will discuss a number of concurrency control techniques that are used to ensure the isolation property of concurrently executing transactions.

– Most of these techniques use protocols –i.e. sets of rules- to ensure serializability of schedules.

– Classification of Techniques
  • Concurrency Based on Locking Techniques
  • Concurrency Based on Timestamps
  • Multiversion Concurrency Control Protocols.
  • Optimistic Concurrency Control.
Locking Techniques

• Lock
  – Variable associated with each data item.
  – Describes status of item regarding the operations that can be performed on it
  – Locks are used as a means of synchronizing the access by concurrent transactions to the database items.
    • A lock is a mechanism to control concurrent access to a data item.
  – Several types of lock
    • Binary locks, Shared/Exclusive locks and certify locks.
Binary locks

A binary lock can have two states/values: locked and unlocked (1 and 0).

A distinct lock is associated with each database item X.

Current value or state associated with X is referred as LOCK(X).

- If Lock(X)=1, then item X cannot be accessed by a database operation that requests the item
- If Lock(X)=0, then item can be accessed when requested

Two operations are associated with binary locks

- lock_item().
- unlock_item().
Locking Techniques con’t

• Binary locks (cont’d)

  • lock_item()
    – A transaction requests access to an item X by first issuing a lock_item(X) operation.
    – If LOCK(X) is 1, the transaction is forced to wait.
    – If LOCK(X) is 0, it is set to 1 and the transaction is allowed to access an item X.

  • unlock_item()
    – When the transaction is through using the item it issues an unlock_item(X) operation.
    – Sets LOCK(X) to 0 so that X may be accessed by other transactions.

  Binary lock enforces mutual exclusion on the data item.
  – lock_item() and unlock_item() must be implemented as indivisible units - knows as critical sections in OS: no interleaving-.
Locking Techniques con’t

• Binary locks (cont’d)
  – To implement a binary Lock.

• Each lock can be a record with two fields: <data item name, locking transaction> plus a queue for transactions that are waiting to access the item.

  – The wait command within the lock_item(X) operation is usually implemented by putting the transaction on a queue for item X.

• The system needs to maintain only these records for the items that are currently locked in a lock_table, which could be organized as a hash file.

• Items not in the lock table are considered to be unlocked.

  – DBMS has a lock manager subsystem to keep track of and control access to locks.
Locking Techniques con’t

• Binary locks (cont’d)
  – If the binary locking scheme is used, every transaction must obey the following rules.

1. A transaction T must issue the operation \texttt{lock\_item(X)} before any read\_item(X) or write\_item(X) operations are performed in T.
2. A transaction T must issue the operation \texttt{unlock\_item(X)} after all read\_item(X) and write\_item(X) operations are completed in T.
3. A transaction T will not issue a lock\_item(X) operation if it already holds the lock on item X.
4. A transaction T will not issue an unlock\_item(X) operation unless it already holds the lock on item X.

• Between the lock\_item(X) and unlock\_item(X) operations in transaction T, T is said to \textbf{hold the lock} on item
  – At most one transaction can hold the lock on a particular item.
Locking Techniques con’t

- Shared/Exclusive (Read/Write) locks.
  - Binary locking too restrictive:
    - Should allow several transactions to access the same item X if they all access X for reading purposes only.
    - If a transaction is to write an item X, it must have exclusive access to X.
  - Three possible states for LOCK(X) associated with an item X.
    - “read-locked”, “write-locked” or “unlocked.”
    - A read-locked item is also called share-locked.
    - A write-locked item is called exclusive-locked because a single transaction exclusively holds the lock on the item.
  - Three locking operations
    - read_lock(X), write_lock(X) and unlock(X);
    - Each operation indivisible – no interleaving-
Locking Techniques con’t

- Shared/Exclusive (Read/Write) locks (cont’d).
  - To implement Shared/Exclusive locks operations: Lock table.
    - A record of the lock table: <data item name, LOCK, no_of_reads, locking_transaction(s)>
      - LOCK(X)=write-locked, the value of locking_transaction(s) is a single transaction that holds the exclusive (write) lock on X.
      - LOCK(X)=read-locked, the value of locking_transaction(s) is a list of one or more transactions that hold the shared (read) lock on X.
Locking Techniques con’t

- Shared/Exclusive (Read/Write) locks (cont’d).

  - In the shared/exclusive locking scheme, every transaction must obey the following rules.

  1. A transaction $T$ must issue the operation `read_lock(X)` or `write_lock(X)` before any `read_item(X)` operation is performed in $T$.

  2. A transaction $T$ must issue the operation `write_lock(X)` before any `write_item(X)` operation is performed in $T$.

  3. A transaction $T$ must issue the operation `unlock(X)` after all `read_item(X)` and `write_item(X)` operations are completed in $T$.

  4. A transaction $T$ will not issue a `read_lock(X)` operation if it already holds a read (shared) lock or a write (exclusive) lock on item $X$.

  5. A transaction $T$ will not issue a `write_lock(X)` operation if it already holds a read (shared) lock or write (exclusive) lock on item $X$.

  6. A transaction $T$ will not issue an `unlock(X)` operation unless it already holds a read (shared) lock or a write (exclusive) lock on item $X$. 
Locking Techniques con’t

- Shared/Exclusive (Read/Write) locks (cont’d).
  - Lock conversion.
    - Desirable to relax conditions 4 and 5 to allow lock conversion.
      - Under certain conditions, a transaction that holds a lock on item X is allowed to convert the lock from one locked state to another.
  - Upgrade
    - It is possible for a transaction T to issue a read_lock(X) and then later on to upgrade the lock by issuing a write_lock(X) operation.
    - If T is the only transaction holding a read lock on X at the time it issues the write_lock(X) operation, the lock can be upgraded; otherwise, the transaction must wait.
  - Downgrade
    - It is possible for a transaction T to issue a write_lock(X) and then later on to downgrade the lock by issuing a read_lock(X) operation.
    - If lock conversion –i.e. upgrading/downgrading- used, change lock table structure.
      - Locking_transaction(s) field must include transaction identifier(s).
**Locking Techniques con’t**

- Binary locks or Read/Write locks does not guarantee serializability.

<table>
<thead>
<tr>
<th>T1</th>
<th>T2</th>
</tr>
</thead>
<tbody>
<tr>
<td>read_lock(Y);</td>
<td>read_lock(X);</td>
</tr>
<tr>
<td>read_item(Y);</td>
<td>read_item(X);</td>
</tr>
<tr>
<td>unlock(Y);</td>
<td>unlock(X);</td>
</tr>
<tr>
<td>write_lock(X);</td>
<td>write_lock(Y);</td>
</tr>
<tr>
<td>read_item(X);</td>
<td>read_item(Y);</td>
</tr>
<tr>
<td>X:=X+Y;</td>
<td>Y:=X+Y;</td>
</tr>
<tr>
<td>write_item(X);</td>
<td>write_item(Y);</td>
</tr>
<tr>
<td>unlock(X);</td>
<td>unlock(Y);</td>
</tr>
</tbody>
</table>

- Initial values: X=20, Y=30
  - Serial schedule T1 followed by T2 → Result: X=50, Y=80
  - Serial schedule T2 followed by T1 → Result: X=70, Y=50
Locking Techniques con’t

- Let’s run $T_1$ and $T_2$ in interleaved fashion: Schedule $S$

<table>
<thead>
<tr>
<th>T1</th>
<th>T2</th>
</tr>
</thead>
<tbody>
<tr>
<td>read_lock($Y$);</td>
<td>read_lock($X$);</td>
</tr>
<tr>
<td>read_item($Y$);</td>
<td>read_item($X$);</td>
</tr>
<tr>
<td>unlock($Y$);</td>
<td>unlock($X$);</td>
</tr>
<tr>
<td>write_lock($X$);</td>
<td>write_lock($Y$);</td>
</tr>
<tr>
<td>read_item($X$); $X := X + Y$;</td>
<td>read_item($Y$); $Y := X + Y$;</td>
</tr>
<tr>
<td>write_item($X$); $unlock(X)$;</td>
<td>write_item($Y$); $unlock(Y)$;</td>
</tr>
<tr>
<td>unlocked too early!</td>
<td></td>
</tr>
</tbody>
</table>

Non-serializable!

Result: $X=50$, $Y=50$

- To guarantee serializability, we need additional protocols on the positioning of locking and unlocking operations on every transaction.
Two-Phase Locking Protocol (2PL)

- All locking operations -read_lock, write_lock- precede the first unlock operation in the transaction.
  - Divide a transaction in two phases: expanding & shrinking phases.

Expanding (growing, first) Phase

- New locks on items can be acquired but none can be released.
  - Upgrading of locks must be done in expanding phase.

Shrinking (second) Phase

- Existing locks can be released but no new locks can be acquired.
  - Downgrading of locks must be done in shrinking phase.

locking mechanism with 2PL guarantees Serializability.

- If every transaction in a schedule follows 2PL, the schedule is guaranteed serializable.
Locking Techniques con’t

- Both T1’ and T2’ follow the 2PL protocol
- Any schedule including T1’ and T2’ is guaranteed to be serializable

2PL limits the amount of concurrency
- T may not be able to release an item X after it is through using it if T must lock an additional item Y later; or conversely.
Locking Techniques con’t

• There are a number of variations of 2PL.
• **Basic 2PL** - Technique just described is known as basic 2PL-
  – Not deadlock-free & cascading rollback is also possible.
• **Conservative (Static) 2PL**
  – To lock all the items it accesses before the transaction begins execution.
    • Predeclaring read-set and write-set..
  – Do not lock any of the item if any of the predeclared items needed cannot be locked.
  – Once the transaction starts it is in the shrinking phase
  – Deadlock-free protocol
• **Strict 2PL (Guarantees Strict Schedule)** — most popular variation of 2PL-
  – Transaction does not release any of its exclusive (write) locks until after it commits or aborts.
    • Hence, no other transaction can read or write an item that is written by T unless T has committed.
  – Not deadlock-free protocol.
Locking Techniques con’t

• Rigorous 2PL (Guarantees Strict Schedule)
  – Transaction T does not release any of its locks (exclusive or shared) until after it commits or aborts.
  – Transaction is in its expanding phase until it ends.
  – Easier to implement than strict 2PL.

• Concurrency control subsystem responsible on generating read_lock & write_lock requests.
  – A transaction T issues the standard read_item(X) (or write_item(X)) without explicit locking calls.
  – Hence, the system calls the read_lock(X) (or write_item(X)) operation on behalf of T – with upgrading if possible.

• Limitation
  – 2PL guarantees serializability but does not permit all possible schedules.
    • Some serializable schedules will be prohibited by the protocol.
  – The use of locks can cause two additional problems: deadlock and starvation.
Deadlock Prevention Protocols

1. Using conservative 2PL protocol
   - The protocol has been discussed and was shown it is deadlock-free.
   - Not practical because of the need of the predeclared read and write sets.

2. Ordering DB items
   - The protocol needs to order the DB items and enforce the transaction to lock its items in that order.
   - Impractical because it requires that the programmer be aware of the chosen order of the items

3. Using the concept of transaction timestamps
   - It is used to decide if the transaction involved in a deadlock situation should wait, abort or preempt another transaction.
   - The timestamp of a transaction T is TS(T) which is a unique identifier assigned to the transaction T and is based on the order in which the transaction T is started.
   - If T1 started before T2, then TS(T1)<TS(T2); T1 older and T2 younger
Deadlock Prevention Protocols cont’d

3. Using the concept of transaction timestamps (cont’d)
   - Two schemas are used: Wait-Die and Wound-Wait
   - Wait-Die
     - Suppose that Ti tries to lock X but it is not able to because X is blocked by Tj with a conflicting lock
     - If $TS(Ti)<TS(Tj)$, then Ti is allowed to wait; otherwise abort Ti (Ti dieds) and restart it later with the same timestamp.
   - Wound-Wait
     - Suppose that Ti tries to lock X but it is not able to because X is blocked by Tj with a conflicting lock
     - If $TS(Ti)<TS(Tj)$, then aborts Tj (Ti wounds Tj) and restart it later with the same timestamp. Otherwise Ti is allowed to wait.
   - The Wait-Die aborts Ti if it is the younger.
   - The Wound-Wait aborts Tj if it is the younger.
   - Both may cause some transactions to be aborted and restarted even they may never actually cause a deadlock.
4. Nowaiting protocols (NW)
   - If a transaction is unable to obtain a lock, it is immediately aborted and then restarted after a certain time delay without checking if a deadlock will actually occur or not.
   - The protocol can cause transactions to abort and restart needlessly.

5. Cautious waiting protocol
   - It has been proposed to reduce the number of needless aborts/restarts.
   - Suppose that Ti tries to lock X but it is not able to because X is locked by Tj with conflicting lock.

     If Tj is not blocked (not waiting for some other locked items)
     Then Ti is blocked and allowed to wait
     Otherwise abort Ti
Deadlock Detection Protocols

• The concept is not to enforce any restrictions on executing the transactions, but check if a deadlock actually exists.

• It is a more practical set of protocols.

• It is beneficial if the transactions are short or the load is light (conflicts are not expected to highly exist)

• Wait-for graph is a simple way to detect the state of deadlock.
  – It is used to check the existing of a deadlock.
    • One node for each currently executing transaction.
    • If Ti is waiting to lock an item X that is currently locked by Tj, a directed edge from Ti to Tj is created.
    • If Tj releases the lock of item that Ti was waiting for, the directed edge is dropped from the graph.
  – If the graph has a cycle, the state of deadlock exists.
  – The system must abort some of the transactions when a deadlock had been detected.
Deadlock solution: Timeouts

- It is simple and practical due to its low overhead.
- The idea is to have a system-defined timeout period and a mechanism to abort the transaction that waits for a period longer than the predefined timeout.
- The system assumes that the transaction expired the timeout may be in deadlock.
**Starvation**

- The problem occurs when a transaction cannot proceed for an indefinite period of time while other transactions in the system continue normally.
- It may occur if the waiting scheme for locked items is unfair.
  1. Using a fair waiting scheme as first-come first-serve.
  2. Allowing priorities for transactions but increase the priority of a transaction the longer it waits, until it eventually gets the highest priority and proceeds.
- It also may occur if the victim selection algorithm used in deadlock detection selects the same transaction as a victim repeatedly.
  - The algorithm can use higher priorities for transactions that have been aborted multiple times.
- Wait-Die and Wound-Wait protocols avoid starvation.
**Timestamps**

- Timestamps concurrency control protocols using the timestamps to order the execution of transactions for an equivalent serial schedule to guarantee serializability.
- Timestamp TS(T) is a unique identifier created by the DBMS to identify the transaction.
- The timestamp values are assigned in the order in which the transactions are submitted to the system.
- The timestamp can be generated by using a counter that is incremented each time its value is assigned to a transaction or by using the current value of date/time value of the system clock.
- It is important to note that the timestamp concurrency protocols are deadlock-free because there is no lock.
Multiversion concurrency control protocols

- The idea is to keep the old values of a data item when it is updated.
- When a transaction writes an item, it writes a new version and the old is retained.
- When a transaction reads an item, an appropriate version is chosen to ensure serializability.
- A disadvantage of these protocols is that more storage is needed, but the old version may be needed for recovery or is a major part of the DB itself as in temporal database.
Multiversion concurrency control protocols cont’d

Multiversion Technique based on timestamp ordering

• There are several versions X1, X2,..Xk for each item X.
• for each version of Xi the following two timestamps are kept:
  ▪ Read_T(S(Xi)): read_T(S(Xi)=TS(T) where T is the youngest transaction that has read Xi successfully.
  ▪ Write_T(S(Xi)): write_T(S(Xi)=TS(T) with T the transaction that wrote Xi.
• To ensure serializability, the following rules are used:
  1. If T issues write_item(X) then
     {Get Xi that has the highest write_T(S(Xi)) that is less than or equals to TS(T).
      If read_T(S(Xi)>TS(T) then abort and rollback T
      Else create new version Xj for X and set read_T(S(Xj)=write_T(S(Xj)=TS(T)
     }
  2. If T issues read_item(X) then
     {Get Xi that has the highest write_T(S(Xi)) that is less than or equals to TS(T).
      return the value Xi to T and set read_T(S(Xi)=max(read_T(S(Xi),TS(T))
     }
MV2PL cont’d

- MV2PL provides two versions for each item, one version (X) must always have been written by some committed transaction and the second (X’) is created when transaction T acquires a write lock on the item.
- Other transactions can continue read the committed version of the item while T holds the write lock.
  - Not allowed under standard 2PL
- T can write the value X’ as needed without affecting the value of the committed version X.
- When T is ready to commit, it must obtain a certify lock on all items it currently hold write locks on before it can commit.
- Once the certify locks are acquired, the committed version of X is set to the value of X’ and X’ is discarded and then the certify locks are released.
- In MV2PL, reads can processed concurrently with single write operation.
- MV2PL avoids cascading rollback
  - Since transactions are only allowed to read the version of X that was written by committed transaction
- Deadlocks may occur in case of allowing lock upgrade and the previous discussed methods of deadlock handling can be used to solve the problem
Validation (Optimistic) concurrency control protocols

- This technique is known as validation or optimistic or certification technique.
- No checking is done while the transaction is executing.
- The idea is to do all the checks at once to allow the transaction proceeds with a minimum overhead until the validation phase.
- The technique is optimistic in sense of expecting less interference and hence little number of transactions will be avoided.
- There are three phases in this technique:
  1. **Read phase**: transaction can read values of committed data items from the DB. Update are applied only to local copies of the data item kept in the transaction workspace.
  2. **Validation phase**: checking is performed to ensure serializability if the updates are applied to the DB.
  3. **Write phase**: if the validation phase is successful, the updates are applied to the DB, otherwise the updates are discarded and the transaction is restarted.
When validating Ti, the first condition is checked first to each Tj, only if condition (1) is false is condition (2) is checked, and only if (2) is false is condition (3) is checked.

- If any one of these three conditions holds, Ti is validated successfully.
- If none of the three conditions holds, the validation of transaction Ti fails and it is aborted and restarted later.
Granularity and MGL

- A database item could be one of the following:
  - A database record.
  - A field of a value of a database.
  - A disk block.
  - A whole file.
  - The whole database

- The size of the data items is often called the data item granularity.
- The granularity can affect the performance of concurrency and recovery.
- Fine granularity refers to small sizes, whereas coarse granularity refers to large item size.
- Large data size will provide lower degree of concurrency.
- Smaller data size will provide of locks which will waste the storage and also a higher overhead is needed to handle the locks.
- The optimal size of the data item depends on the types of the transactions on hand
Granularity and MGL cont’d

Multiple Granularity level locking (MGL)

- The idea is to allow the DB system to support multiple levels of granularity to fulfill the optimal size needed by different transactions.
- The following figure illustrates the hierarchy of granularity in DB.

```
+----------+-
| db       |
+----------+-
|           |
+----------+-
| f1       |
+----------+-
| p11      |
| r111 ... |
+----------+-
| p12      |
| r121 ... |
+----------+-
| ...      |
+----------+-
| p1n      |
| r1nj     |
+----------+-
|           |
+----------+-
| f2       |
+----------+-
| p21      |
| r211 ... |
+----------+-
| p22      |
| r221 ... |
+----------+-
| ...      |
+----------+-
| p2m      |
| r2mk     |
```

- Multiple granularity level 2PL protocol (MGL) needs these new locks:
  1. Intention-shared (IS) indicates that a shared lock(s) will be requested on some descendent node(s).
  2. Intention-exclusive (IX) indicates that exclusive lock(s) will be requested on some descendent node(s).
  3. Shared-intention-exclusive (SIX) indicates that the current node is locked in shared mode but an exclusive lock(s) will be requested on some descendent node(s).
Granularity and MGL cont’d

• The compatibility table is as follows

<table>
<thead>
<tr>
<th>T requests</th>
<th>IS</th>
<th>IX</th>
<th>S</th>
<th>SIX</th>
<th>X</th>
</tr>
</thead>
<tbody>
<tr>
<td>IS</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>IX</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>S</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>SIX</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>X</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
</tbody>
</table>

• The MGL protocol consists of the following rules
  1. The lock compatibility table must adhere to.
  2. The root of the tree must be locked first in any mode.
  3. A node N can be locked by T in S or IS mode only if its parent is already locked by T in either IS or IX mode.
  4. A node can be locked by T in X, IX, or SIX mode only if its parent is already locked by T in either IX or SIX mode.
  5. A transaction T can lock a node only if it has not unlocked any node (to enforce 2PL).
  6. A transaction T can unlock a node N only if none of the children of the node are currently locked by T.
Other issues

• Insertion in a DB
  – The new inserted item must not be accessed until after the operation is completed.
  – In locking techniques, this can be done by using exclusive lock for the item.
  – In timestamp techniques, this can be done by setting the read and write timestamps of the new item to the timestamp of the creating transaction.

• Deletion in a DB
  – In locking techniques, this can be done by using exclusive lock before the item can be deleted.
  – In timestamp, the protocol must ensure that no later transaction has read or written the item before completing the deletion

• Phantom
  – The phantom problem can occur when a new record that is being inserted by some transaction T satisfies a condition that is set of records accessed by another transaction T’ must satisfy. It can be solved by using index locking.
  – Another technique called predicate locking which will lock access to all records that satisfy an arbitrary condition. The technique is not an easy to implement.
Overview of data concurrency and consistency in Oracle9i

– Data Concurrency: many users can access the data at the same time.
– Data consistency: each user sees a consistent view of the data.

• Oracle handles concurrency slightly different from the described protocols.

• Oracle concurrency uses a multiversion read consistency model, various types of locks, and transactions.

• Multiversion read consistency model allows to:
  – Guarantees that all the data returned by a single query comes from a single point in time –the time that the query began- (I.e.Statement-level read consistency).
  – Ensures that readers of DB data do not wait for writers or other readers of the same data.
  – Ensures that writers of DB data do not wait for readers of the same data.
  – Ensures that writers only wait for other writers if they attempt to update identical rows in concurrent transactions.
Overview of data concurrency and consistency in Oracle9i

To manage multiversion read consistency model

- When an update occurs, the original data values changed by the update are recorded in the DB undo records in rollback segments (or in undo segments (in tablespace))
- As long as this update remains part of an uncommitted transaction, any user, that later queries the modified data, views the original data values.
- Oracle uses the current information in the SGA and information in the undo records to construct a read-consistent view of a table's data for a query.
- Precisely, Oracle maintains a system change number (SCN) which is a logical timestamp that record the order in which operations occur. SCN is a serial unique number. Oracle assigns every committed a new SCN.
  - As a query enters the execution stage, the SCN is determined, 10023 for example. As data blocks are read on behalf of the query, only blocks written with the observed SCN (i.e. blocks with SCN less or equal to this SCN) are used. Blocks with changed data (more recent SCNs) are reconstructed from data in the rollback segments, and the reconstructed data is returned to the query.
- When a transaction is committed, its changes are made permanent; i.e. statements that start, after this transaction has committed, only see the changes made by this committed transaction. Oracle generates an SCN number and assigns it to the rollback segment and marks that the transaction is committed..
Overview of data concurrency and consistency in Oracle9i

- Oracle also uses locks to control concurrent data access.
- Three types of locks are used: DML, DDL, internal locks.
- DML locks guarantees data integrity and prevents destructive interference of simultaneous conflicting DML or DDL operations (i.e. a table cannot be dropped (Drop is DDL (Data Def Lang.)) if an uncommitted transaction contains an insert into this table). DML locks are released at the end of the transaction.

- Two different levels for DML locks
  - Rows locks
    - Requires no user action (i.e. automatically). A transaction acquires an exclusive DML lock for each individual row modified by one of the following statements: INSERT, UPDATE, DELETE, and SELECT…FOR UPDATE.
    - If a transaction obtains a row lock, the transaction also acquires a table lock for the corresponding table.
  - Table locks
    - A transaction acquires a table lock when a table is modified in the following DML statements: INSERT, UPDATE, DELETE, SELECT…FOR UPDATE, and LOCK TABLE.
    - A table lock can be held in any of several modes: row share (RS), row exclusive (RX), share (S), share row exclusive (SRX), and exclusive (X).
Overview of data concurrency and consistency in Oracle9i

- DDL (Data Dictionary Locks) protects the definition of a schema object (like definition of tables and views) while that object is acted upon or referred to by an ongoing DDL operation.
- Latches and internal locks protects internal DB and memory structures which are both inaccessible to users.
- Explicit (manual) data locking
  - Oracle always performs locking automatically (implicitly without user intervention). However, one can override the Oracle default locking mechanism.
  - Useful in situations like: Applications require that a transaction have exclusive access to a resource so that the transaction does not have to wait for other transactions to complete.
  - LOCK TABLE and SELECT...FOR UPDATE statements can be used to override the Oracle default locking.
- SELECT...FOR UPDATE statement
  - Lets you lock the selected rows so that other users cannot lock or update the rows until you end your transaction.
- Oracle detects deadlock and resolves it by rolling back one of the statements involved in the deadlock. The transaction, which its statement has been rolled back, should be rolled back explicitly.