LOCKS

LOCKS

- In a multiprogramming environment where multiple transactions can be executed simultaneously, it is highly important to control the concurrency of transactions.
- Concurrency control protocols are there to ensure atomicity, isolation, and serializability of concurrent transactions.
- Concurrency control protocols can be broadly divided into two categories:
  - Lock based protocols
  - Time stamp based protocols

LOCKS

- Database systems equipped with lock-based protocols use a mechanism by which any transaction cannot read or write data until it acquires an appropriate lock on it.
- A lock is a variable associated with a data item that describes the status of the item with respect to possible operations that can be applied to it.
- Generally, there is one lock for each data item in the database.
- Locks are used as a means of synchronizing the access by concurrent transactions to the database item.
- The oracle engine automatically locks table data while executing SQL statements like SELECT/INSERT/UPDATE/DELETE. This type of locking is called **IMPLICIT LOCKING**.

LOCKS

TYPES:

Several types of locks are used in concurrency control. They are:

- Binary Lock
- Shared/Exclusive Lock

LOCKS

Binary Lock:

- It is simple but restrictive and so are not used in practice.
- A binary lock can have two states or values: **locked** and **unlocked**.
- A distinct lock is associated with each database item *A*.
- If the value of the **lock on *A* is 1**, item *A* **cannot be accessed** by a database operation that requests the item.
- If the value of the **lock on *A* is 0** then item **can be accessed** when requested.

LOCKS

- There are two operations, lock item and unlock item are used with binary locking A transaction requests access to an item *A* by first issuing a lock *item (A)* operation.
- If **LOCK (A) = 1**, the **transaction is forced to wait**. If **LOCK (A) = 0** it is set to 1 and the **transaction is allowed to access item *A***.
- The lock manager module of the DBMS can enforce these rules.

Disadvantages of Binary Locks

Binary locking scheme is too restrictive for database items, because **at most one transaction can hold a lock on a given item**. So, binary locking system cannot be used for practical purpose.

LOCKS

Shared/Exclusive (for Read/Write) Locks

Shared lock:

- Shared locks are placed on resources whenever a read operation (SELECT) is performed.
- Multiple shared locks can be simultaneously set on a resource.
- It can be shared between the transactions because when the transaction holds a lock, then it can't update the data on the data item.
- It is also known as a Read-only lock.

Syntax:

```
Lock table table_name in SHARE MODE;
Unlock table table_name;
```

LOCKS

Exclusive Lock:

- Exclusive locks are placed on resources whenever a write operation (INSERT, UPDATE And DELETE) are performed.
- In the exclusive lock, the data item can be both reads as well as written by the transaction.
- Only one exclusive lock can be placed on a resource at a time.
- This lock is exclusive, and in this lock, multiple transactions do not modify the same data simultaneously.

Syntax:

```
Lock table table_name in EXCLUSIVE MODE;

Unlock table table_name;
```

LOCKS

Locking operations

- There are three locking operations called,
  - read_lock(A), write_lock(A) and unlock(A)
- They are represented as,
  - Lock_S(A), Lock_X(A), unlock(A)

  Here, S indicates Shared lock, X indicates Exclusive Lock
- A lock associated with an item A, LOCK (A), now has three possible states: "read-locked", "write-locked," or "unlocked."
- A read-locked item is also called share-locked item because other transactions are allowed to read the item, whereas a write-locked item is caused exclusive-locked, because a single transaction exclusively holds the lock on the item.

LOCKS

Compatibility of Locks:

STATE OF THE LOCK	SHARED	EXCLUSIVE
UNLOCK	YES	YES
SHARED	YES	NO
EXCLUSIVE	NO	NO

LOCKS

EXAMPLE:

As an illustration consider the simplified banking system. Let A and B be two accounts that are accessed by transactions TI and T2. Transaction TI transfers Rs.50 from account B to account A and is defined as:

TRANSACTION T1
LOCK_X(B)
READ(B)
B=B-50
WRITE(B)
UNLOCK(B)
LOCK_X(A)
READ(A)
A=A+50
WRITE(A)
UNLOCK(A)

LOCKS

Transaction T2 displays the total amount of money in accounts A and B that is, the sum A+B and is defined as:

TRANSACTION T2
LOCK_S(A)
READ(A)
UNLOCK(A)
LOCK_S(B)
READ(B)
UNLOCK(B)
DISPLAY(A+B)

LOCKS

Suppose that the values of accounts A and B are Rs.100 and Rs.200, respectively. If these two transactions are executed serially, either in the order T1 and T2 or the order T2, T1 then transaction T2 will display the value Rs.300.

If however, these transactions are executed concurrently, as shown in Schedule 1 (NEXT SLIDE). In this case, transaction T2 displays Rs.250, which is incorrect.

The reason for this mistake is that the transaction TI unlocked data item B too early, as a result of which T2 shows an inconsistent state.

LOCKS

SCHEDULE S1

TRANSACTION T1	TRANSACTION T2	CONCURRENCY CONTROL MANAGER
LOCK_X(B)		GRANT X(B,T1)
READ(B)		
B=B-50		
WRITE(B)		
UNLOCK(B)		
	LOCK_S(A)	GRANT S(A,T2)
	READ(A)	
	UNLOCK(A)	
	LOCK_S(B)	GRANT S(B,T2)
	READ(B)	
	UNLOCK(B)	
	DISPLAY(A+B)	
LOCK_X(A)		GRANT X(A,T1)
READ(A)		
A=A+50		
WRITE(A)		
UNLOCK(A)		

LOCKS

Solution of Inconsistency Problem:

Suppose now that unlocking is delayed to the end of the transaction. The transaction T1 and T2 with unlocking delayed and is defined as

TRANSACTION T1
LOCK_X(B)
READ(B)
B=B-50
WRITE(B)
LOCK_X(A)
READ(A)
A=A+50
WRITE(A)
UNLOCK(B)
UNLOCK(A)

TRANSACTION T2
LOCK_S(A)
READ(A)
LOCK_S(B)
READ(B)
UNLOCK(A)
UNLOCK(B)
DISPLAY(A+B)

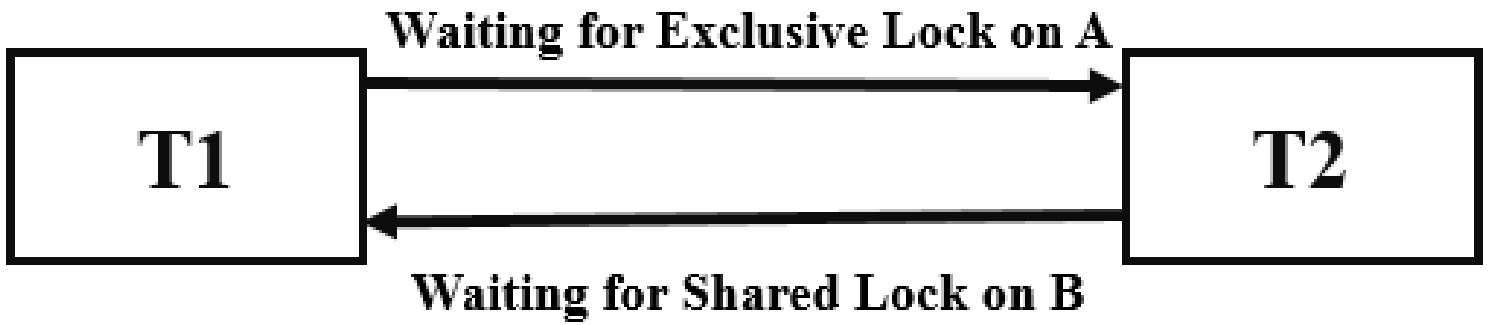
LOCKS

The sequence of reads and writes in schedule S1 which leads to an incorrect total of Rs.250 being displayed, is no longer possible with T1 and T2 as shown in Schedule S2:

TRANSACTION T1	TRANSACTION T2
LOCK_X(B)	
READ(B)	
B=B-50	
WRITE(B)	
	LOCK_S(A)
	READ(A)
	LOCK_S(B)
	WAIT…
	WAIT…
LOCK_X(A)	
WAIT…	

LOCKS

- The use of locking can lead to an undesirable situation.
- Consider the schedule S2, T1 is holding an exclusive mode lock on B and T2 is requesting a shared mode lock on B i.e.T2 is waiting for T1 to unlock B.
- Similarly, T2 is holding a shared mode lock on A and T1 is requesting an exclusive mode lock on A, thus T1 is waiting for T2 to unlock A.



LOCKS

- Thus we have arrived at a state where neither of these transactions can ever proceed with its normal execution. This situation is called **DEADLOCK**.
- The solution of inconsistency leads to deadlock problem.
- If we do not use locking or unlock data items as soon as possible after reading or writing them, we may get inconsistent states.
- On the other hand, if we do not unlock a data item before requesting a lock on another data item deadlocks may occur.
- There are ways to avoid deadlock in some situations. Deadlocks are definitely preferable to inconsistent states, since they can be handled by rolling back of transactions.