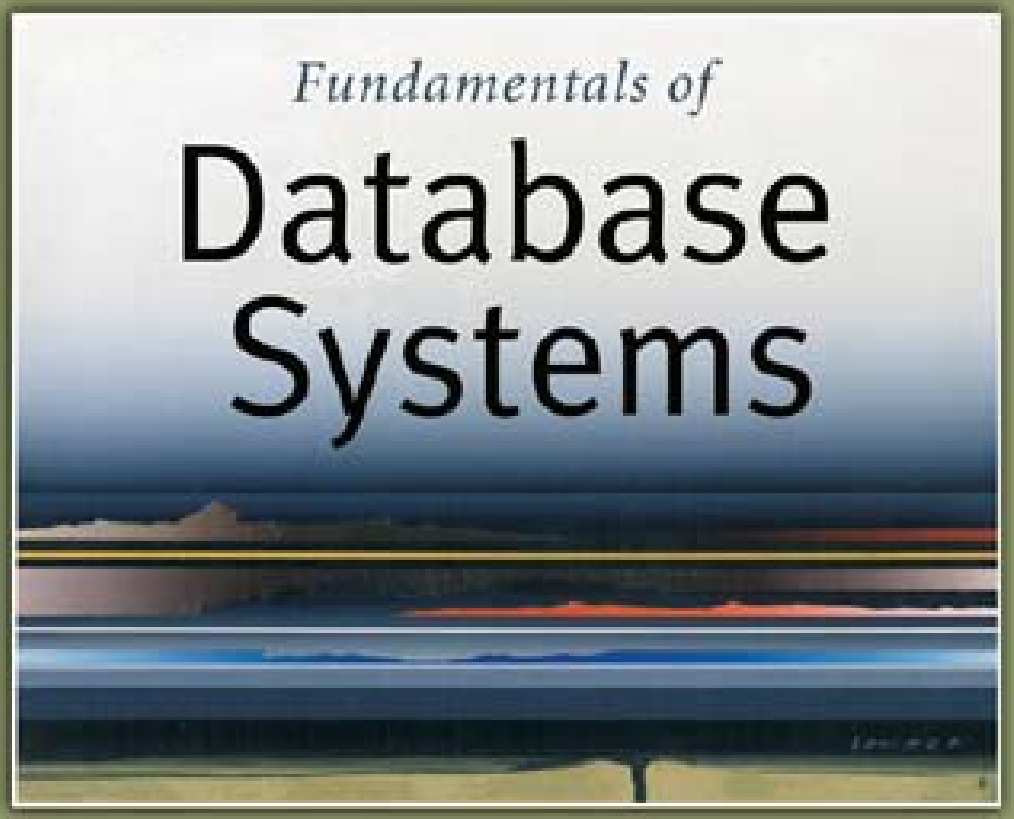


5th Edition

Elmasri / Navathe

Chapter 17

Introduction to Transaction Processing Concepts and Theory

Multi-user processing and concurrency

- ‘Simultaneous’ processing on a single processor is an illusion.
- When several users are using a system concurrently, each process gets a ‘time slot’.
- Sometimes the processes swap in and out, with bits of one being done, followed by bits of another.
- Simultaneous processing can be done using multiple processors.

Multi-user processing and concurrency

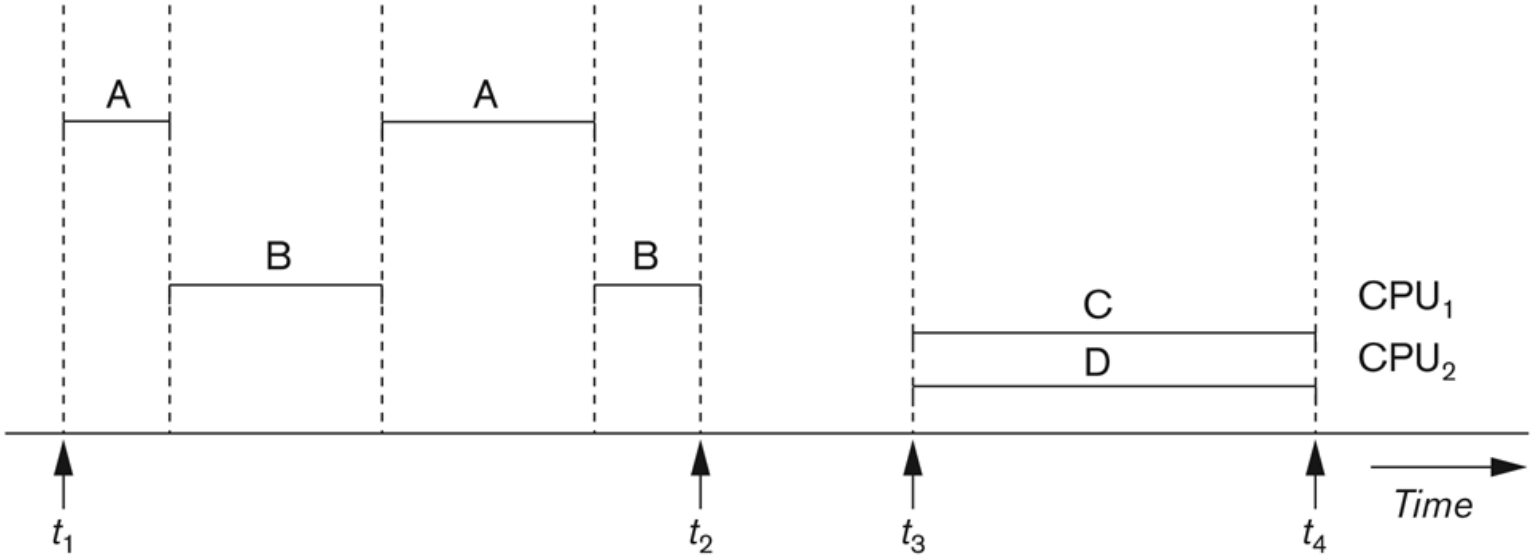


Figure 17.1
Interleaved processing versus parallel processing of concurrent transactions.

- In this case, A and B are being swapped in and out of memory for a single CPU.
- C and D are being processed simultaneously by two separate processors.

Transactions

- A transaction includes one or more database access operations.
- These can include insert, delete, update or select.
  - Any of the first three immediately place the session into transaction mode.
  - Select can be a read-only transaction.
- To illustrate concepts:
  - T is a transaction
  - T_n is transaction number n.

Definitions

- We consider the database to be a set of named data items.
- The size of a data item is called its granularity.
  - It can be field-level, row-level, block-level or table-level.
- We denote a retrieval as:
  - Read_item(X), where X is a database item and a program variable name.
- We denote a write as:
  - write_item(X) where X is a database item and a program variable name.

read_item(X)

- Finds the address of the disk block that contains item X.
- Copy that disk block into a buffer in main memory
  - (if that disk block is not already in some main memory buffer)
- Copy item X from the buffer to the program variable named X.

write_item(X)

- Find the address of the disk block that contains item X.
- Copy that disk block into a buffer in main memory.
  - If that disk block is not already in some main memory buffer.
- Copy item X from the program variable named X into its correct location in the buffer.
- Store the updated block from the buffer back to disk.
  - Either immediately, or at some later point.

Simultaneous transactions

(a)

T_1

read_item(X);
$X := X - N$;
write_item(X);
read_item(Y);
$Y := Y + N$;
write_item(Y);

(b)

T_2

read_item(X);
$X := X + M$;
write_item(X);

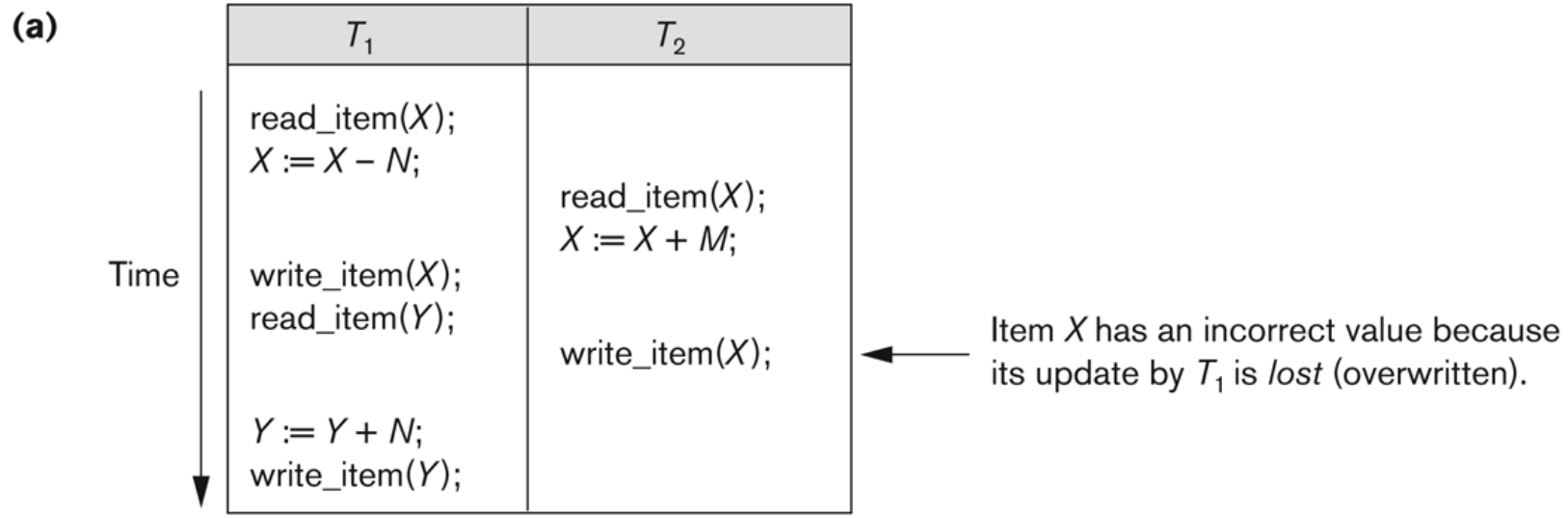
Figure 17.2
Two sample transactions.
(a) Transaction T_1.
(b) Transaction T_2.

- The read-set of a transaction is the set of all items that the transaction reads
  - the read-set of T_1 is {X,Y} and of T_2 is {X}
- The write-set of a transaction is the set of all items that the transaction writes.
  - The write-set of T1 is {X,Y} and of T2 is {X}

Lost Update Problem

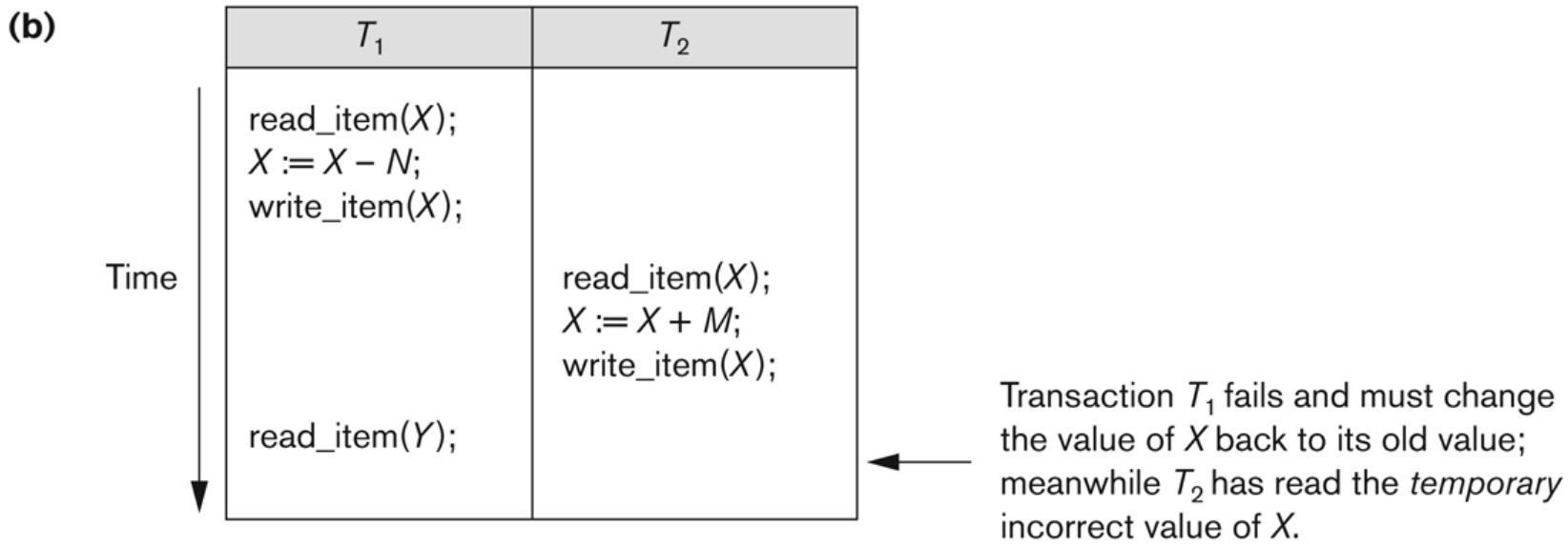
Figure 17.3

Some problems that occur when concurrent execution is uncontrolled. (a) The lost update problem. (b) The temporary update problem. (c) The incorrect summary problem.



- This problem occurs when Transaction 2's update is overwritten by Transaction 1's update.

Temporary Update Problem



- This happens when Transaction 2 reads a value for X that is not permanent and updates it on that basis. Transaction 1, meanwhile, has abandoned (rolled back) its original update.

Incorrect Summary Problem

(c)

T_1	T_3
	$sum := 0$; read_item(A); $sum := sum + A$; • • •

read_item(X);
$X := X - N$;
write_item(X);	

	read_item(X);
	$sum := sum + X$;
	read_item(Y);
	$sum := sum + Y$;

read_item(Y);
$Y := Y + N$;
write_item(Y);	

T_3 reads X after N is subtracted and reads Y before N is added; a wrong summary is the result (off by N).

Unrepeatable Read

- If Transaction 1 reads an item twice and the item is changed by another transaction between the two reads.
  - E.G. While reserving seats on a flight / at a concert / film showing.

The need for recovery – types of failure

- A computer failure
- Transaction or system error.
  - E.g. divide by zero, logical programming errors, etc.
- Local errors or exception conditions detected by the transaction.
  - E.g. data not found, already there, logic problem (a team can't play itself).
- Concurrency control enforcement.
  - One transaction making another incorrect.
- Disk failure
- Physical problems or catastrophes.

Transaction states

- A transaction is an atomic unit of work that is either completed in its entirety, or not done at all.
- For recovery purposes, the system needs to track the transaction start, termination, commit or abort.
- States:
  - BEGIN_TRANSACTION
  - READ OR WRITE
  - END_TRANSACTION
  - COMMIT_TRANSACTION
  - ROLLBACK or ABORT

Transaction states

- BEGIN_TRANSACTION
  - Beginning of transaction execution
- READ OR WRITE
  - A read / write has executed, but may not be permanent.
- END_TRANSACTION
  - Reads and writes finished, but no permanency guaranteed.
- COMMIT_TRANSACTION
  - Successful end of transaction, making updates permanent.
- ROLLBACK or ABORT
  - Unsuccessful end, returning the database to the state it was in prior to the start of the transaction.

Transaction states

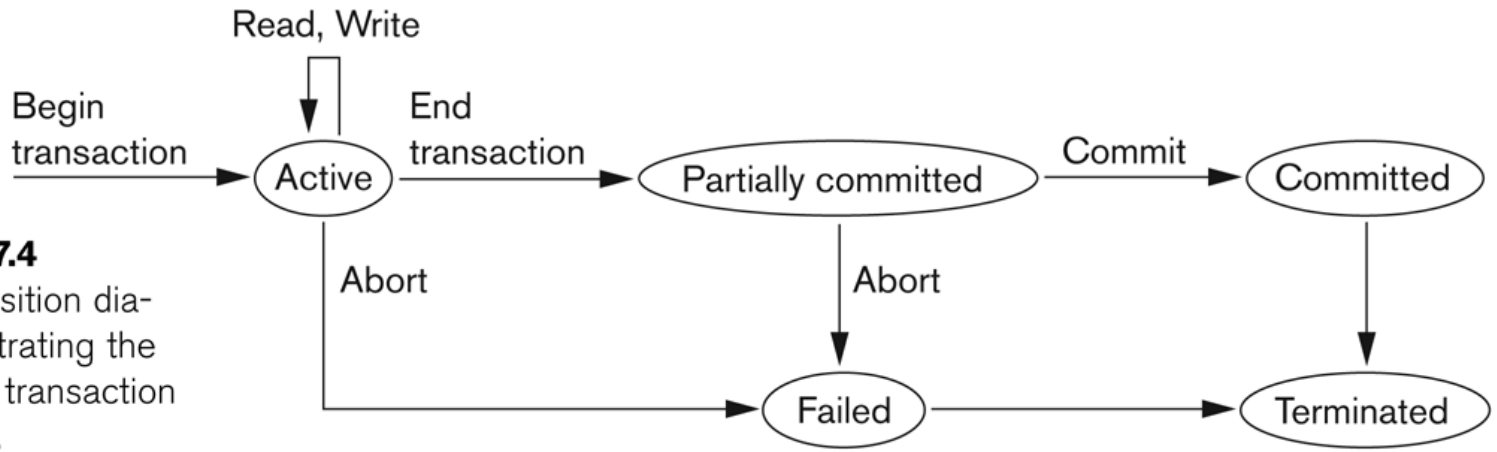


Figure 17.4
State transition diagram illustrating the states for transaction execution.

The system log

- This keeps log records about the state of the transaction:

1. [start_transaction, T]
2. [write_item, T, X, old_value, new_value]
3. [read_item, T, X]*
4. [commit, T]
5. [abort, T]

There is no necessity to record all reads.

- This log allows to undo (for an abort) or redo (for an unsaved commit) transactions.

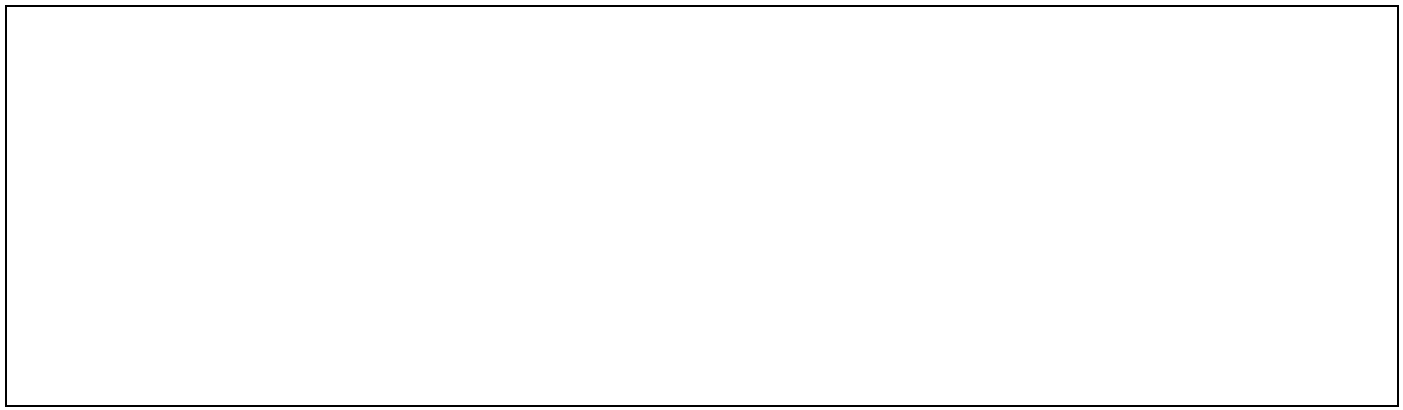
Commit point of a transaction

- A transaction T reaches its commit point when
  - all its operations that access the database have been executed successfully and
  - The effect of all the transaction operations on the database have been recorded in the log.

Desirable properties of a transaction

- ACID properties:
  - Atomicity. A transaction is an atomic unit of processing; either it is completed or not done at all.
  - Consistency preservation. Its complete execution should take the database from one consistent state to another.
  - Isolation. It should appear as though it is being executed in isolation from other transactions. There should be no interference from other transactions.
  - Durability (or permanency). The changes applied to the database by the transaction must persist in the database. They must not be lost because of any failure.

Locking and concurrency



Overview

- In multi-user systems, several users may update the same information concurrently – i.e. at the same time.
- Locking allows
  - A user to take hold of a block for updating.
  - No-one else can modify the same data.
- When a user modifies data through a transaction,
  - Data should be locked by that transaction until the transaction is committed or rolled back.
  - The lock should be held until the transaction is complete - this known as data concurrency.

Locking

- Although locks are vital to enforce database consistency, they can create performance problems.
- Every time one process issues a lock, another user may be shut out from processing the locked row or table.
- For this reason, there are several levels of locking.

Read consistency

- Locking can ensure that all processes can always access (read) the original data as they were at the time the query began (uncommitted modification).
  - This is known as ***read consistency***.

Locking in Oracle

- Oracle provides two different levels of locking: Row Level Lock and Table Level Lock.
- Row-Level Locking
  - With a row-level locking strategy, each row within a table can be locked individually.
  - Locked rows can be updated only by the locking process.
  - All other rows in the table are still available for ***updating*** by other processes.
  - Other processes continue to be able to ***read*** any row in the table, including the one that is actually being updated.
  - When other processes read updated rows, they see only the old version of the row prior to the lock being placed for update (via a rollback segment) until the changes are actually committed.
  - This is known as a **consistent read**.

Row level lock – how to:

- a data manipulation language **(DML) lock** is put on the row.
  - This stops other processes from updating (or locking) the row.
- a data definition language **(DDL) lock** is placed over the table to prevent structural alterations to the table.
    - For example, this type of lock keeps the DBA from being able to remove a table by issuing a DROP statement against the table.
  - This lock is released only when
    - the locking process successfully commits the transaction to the database (i.e., makes the updates to that transaction permanent)
  - or when
    - the process is rolled back.

Table-Level Locking

- With table-level locking, the entire table is locked as an entity.
- Once a process has locked a table, only that process can update (or lock) any row in the table.
- None of the rows in the table are available for updating by any other process.
- Other processes can read any row in the table, including the one that is being updated.

How does table-level locking work?

- The first DML operation that needs to update a row in a table obtains what's called a Row Share Exclusive lock over the entire table.
- All other query-only processes needing access to the table are informed that they must use the rollback information for the locking process.
- The lock is released only when the locking process successfully commits the transaction to the database or when the process is rolled back.

Releasing Locks

- Many users believe that they are the only users on the system - at least the only ones who count.
- Unfortunately, this type of attitude is what causes locking problems.
- We've often observed applications that were completely stalled because one user decided to go to lunch without having committed his or her changes.
- Remember that all locking (row or table) will prevent other users from updating information.
- Every application has a handful of central, core tables. Inadvertently locking such tables can affect many other people in a system.

- Many users, and some programmers, don't understand that terminating a process does not always release locks.
- Switching off your workstation before you go home does not always release locks.
- Locks are released only when changes are committed or rolled back.
- A user's action is the only thing that distinguishes between committing, aborting, and rolling back changes.
- Make it a priority to train your users to commit or roll back all outstanding changes before leaving their current screens.

Modes of Locking

- Oracle uses **two modes** of locking in a multi-user database:
- **Exclusive lock mode (X)** prevents the associates resource from being shared. This lock mode is obtained to modify data. The first transaction to lock a resource exclusively is the only transaction that can alter the resource until the exclusive lock is released.
- Both a row and a table can be locked exclusively.
- **Share lock mode (S)** allows the associated resource to be shared, depending on the operations involved. Multiple users reading data can share the data, holding share locks to prevent concurrent access by a writer (who needs an exclusive lock).
- Both a row and a table can be locked for sharing.

Exclusive Locks

SQL Statement	Lock mode
SELECT ... FROM *table* ...	No Lock
INSERT INTO *table* ...	Row Exclusive
UPDATE *table* ...	Row Exclusive
DELETE FROM *table* ...	Row Exclusive
LOCK TABLE *table* IN ROW EXCLUSIVE MODE	Row Exclusive
LOCK TABLE *table* IN SHARE ROW EXCLUSIVE MODES	Row Exclusive
LOCK TABLE *table* IN EXCLUSIVE MODE	Exclusive

Exclusive locks

- Exclusive locks occur on ROWS when:
  - The row is INSERTed
  - The row is DELETEd
  - The row is UPDATEd
  - The table is locked as follows:
    - LOCK TABLE *table* in ROW EXCLUSIVE MODE
    - LOCK TABLE *table* in SHARE ROW EXCLUSIVE MODE
- To get an exclusive lock on a whole table:
  - LOCK TABLE *table* IN SHARE ROW EXCLUSIVE

Share Locks

- Row Share locks happen when the following statements are issued:
  - **`SELECT ... FROM table  FOR UPDATE OF…`**
- or
  - **`LOCK TABLE table IN ROW SHARE MODE`**
- A Share lock happens when the following statement is issued:
  - **`LOCK TABLE table IN SHARE MODE`**

Row Share Table Locks

- A row share table lock indicates that the transaction holding the lock on the table
  - has locked rows in the table and
  - intends to update them.
- A row share table lock is the least restrictive mode of table lock, offering the highest degree of concurrency for a table.

Row Share Lock:

- Allowed Operations:
  - A row share table lock held by a transaction allows ***other*** transactions to:
    - SELECT (query the table)
    - INSERT,
    - UPDATE,
    - DELETE or
    - lock rows concurrently in the same table.
- Prohibited Operations:
  - prevents other transactions from exclusive write access to the same table.

When to Lock with ROW SHARE Mode:

1. Your transaction needs to prevent another transaction from acquiring an intervening
    1. share,
    2. share row
    3. exclusive table lock
    4. alteration
    5. Drop

for a table before the table can be updated in your transaction.

2. If another transaction acquires this lock on your row, your transaction cannot update the table until the locking transaction commits or rolls back.
3. Your transaction needs to prevent a table from being altered or dropped before the table can be modified later in your transaction.

Example

- We use the EMP table for the next examples.

EMPNO	ENAME	JOB
7369	Smith	CLERK
7499	Allen	SALESMAN
7521	Ward	SALESMAN
7566	Jones	MANAGER
7654	Martin	SALESMAN
7698	Blake	MANAGER
7782	Clark	MANAGER
7788	Scott	ANALYST
7839	King	PRESIDENT
7844	Turner	SALESMAN
7876	Adams	CLERK
7900	James	TEST
7902	Ford	ANALYST
7934	Miller	CLERK

Session 1	Session 2

`select job from emp`
`  where job = 'CLERK'`
`  for update of empno;`
OK
	`select job from emp`
	`  where job = 'CLERK'`
	`  for update of empno;`
	Waiting ....

	`select job from emp`
	`  where job = 'MANAGER'`
	`  for update of empno;`
	OK

	`lock table emp in share mode;` **OK**
	`lock table emp in exclusive mode;` **Waiting ....**

	`insert into emp (empno,ename)`
	`  values (9999,'Test');`
	OK

	`delete from emp where empno = 9999;` **OK**
	`delete from emp where empno = 7876;` **Waiting .... (Blocked by Session 1)**

	`update emp set job = 'CLIMBER'`
	`  where empno = 7876;`
	Waiting .... (Blocked by Session 1)

Row Exclusive Table Locks

- A row exclusive table lock indicates that the transaction holding the lock has made one or more updates to rows in the table.
- A row exclusive table lock is acquired automatically for a table modified by the following types of statements:
  - **INSERT INTO** *table* **... ;**
  - **UPDATE** *table* **... ;**
  - **DELETE FROM** *table* **... ;**
  - **LOCK TABLE** *table* **IN ROW EXCLUSIVE MODE;**
    - A row exclusive table lock is slightly more restrictive than a row share table lock.

Operations

- A row exclusive table lock held by a transaction ***allows other*** transactions to:
  - SELECT (query the table)
  - INSERT, UPDATE, DELETE
  - or lock rows concurrently in the same table.
- A row exclusive table lock held by a transaction ***prevents other*** transactions from :
  - LOCK TABLE *table* IN SHARE MODE;
  - LOCK TABLE *table* IN EXCLUSIVE MODE;
- *When to Lock with ROW EXCLUSIVE Mode:*
  - This is the **Default Locking Behaviour** of Oracle.

Example

Session 1	Session 2
update emp set ename = 'Zahn'; OK	lock table emp in exclusive mode; Waiting ....

Share Table Locks

- A share table lock is acquired automatically for the table specified in the following statement:
  - **LOCK TABLE table IN SHARE MODE;**
- *Permitted Operations:*
  - A share table lock held by a transaction ***allows other*** transactions only to
    - to SELECT (query the table)
    - to lock specific rows with SELECT ... FOR UPDATE
    - or to execute LOCK TABLE ... IN SHARE MODE statements successfully.
  - No updates are allowed by other transactions.
  - Multiple transactions can hold share table locks for the same table concurrently.
    - In this case, no transaction can update the table (even if a transaction holds row locks as the result of a SELECT statement with the FOR UPDATE clause). Therefore, a transaction that has a share table lock can update the table only if no other transactions also have a share table lock on the same table.

Share table locks

- *Prohibited Operations:*
- A share table lock held by a transaction prevents other transactions from modifying the same table and from executing the following statements:
  - LOCK TABLE table IN SHARE ROW EXCLUSIVE MODE;
  - LOCK TABLE table IN EXCLUSIVE MODE;
  - LOCK TABLE table IN ROW EXCLUSIVE MODE;
- *When to Lock with SHARE Mode*
- Your transaction only queries the table, and requires a consistent set of the table's data for the duration of the transaction.
- You can hold up other transactions that try to update the locked table, until all transactions that hold SHARE locks on the table either commit or roll back.
- Other transactions may acquire concurrent SHARE table locks on the same table, also allowing them the option of transaction-level read consistency.

Caution

- Your transaction may or may not update the table later in the same transaction.
- However, if multiple transactions concurrently hold share table locks for the same table, no transaction can update the table (even if row locks are held as the result of a SELECT... FOR UPDATE statement).
- Therefore, if concurrent share table locks on the same table are common, updates cannot proceed and **deadlocks are common**.
  - In this case, **use share row exclusive or exclusive table locks instead**.

Example 1

Session 1	Session 2
lock table emp in share mode; **OK**	select * from emp; **OK**
update emp set ename = 'Zahn' where empno = 7900; **commit;** **OK**	This and other Transactions have to wait until Session 1 commits the Transaction.
lock table emp in share mode; **OK**	This and other Transactions can get a Share Lock **(Lock Switch).**
This and other Transactions have to wait until Session 2 commits the Transaction.	lock table emp in share mode; **OK**
update emp set ename = 'Müller' where empno = 7900; **Waiting ....**	

Example 2

- For example,
  - assume that two tables, *emp* and *budget*, require a consistent set of data in a third table, *dept*.
  - For a given department number, you want to update the information in both of these tables, and ensure that no new members are added to the department between these two transactions.
- Although this scenario is quite rare, it can be accommodated by locking the dept table in SHARE MODE, as shown in the following example. Because the *dept* table **is rarely updated**, locking it probably does not cause many other transactions to wait long.

```
 LOCK TABLE dept IN SHARE MODE;  /* Other Transactions
have to wait */

UPDATE emp
  SET sal = sal * 1.1
  WHERE deptno IN
  (SELECT deptno FROM dept WHERE loc = 'DALLAS');

UPDATE budget
  SET Totsal = Totsal * 1.1
  WHERE deptno IN
  (SELECT deptno FROM dept WHERE Loc = 'DALLAS');

COMMIT; /* This releases the lock */
```

Exclusive Table Locks (X)

- An exclusive table lock is **the most restrictive mode** of table lock, allowing the transaction that holds the lock exclusive write access to the table. An exclusive table lock is acquired for a table as follows:
  - **LOCK TABLE table IN EXCLUSIVE MODE;**
- *Permitted Operations:*
  - Only one transaction can obtain an exclusive table lock for a table.
  - An exclusive table lock permits other transactions only to query the table.
- *Prohibited Operations:*
  - An exclusive table lock held by a transaction prohibits other transactions from performing any type of DML statement or placing any type of lock on the table.

Only use an EXCLUSIVE lock when…

- Your transaction requires immediate update access to the locked table, as other transactions cannot lock specific rows in the locked table.
  - Your transaction also ensures transaction-level read consistency for the locked table until the transaction is committed or rolled back.
- You are not concerned about low levels of data concurrency, making transactions that request exclusive table locks wait in line to update the table sequentially.

Example

Session 1	Session 2
lock table emp in exclusive mode; **OK** update emp set ename = 'Zahn' where empno = 7900; **commit;** **OK** lock table emp in exclusive mode; **OK**	select * from emp; **OK** This and other Transactions have to wait until Session 1 commits the Transaction. This and other Transactions cannot get any other Lock **(No Lock Switch).** lock table emp in share mode; **Waiting ....** lock table emp in exclusive mode; **Waiting ....** update emp set ename = 'Zahn' where empno = 7900; **Waiting ....**

Deadlocks

- A deadlock can occur when two or more users are waiting for data locked by each other.
- Deadlocks prevent some transactions from continuing to work.
- The next example illustrates two transactions in a deadlock.

Example

Session 1	Session 2	Time

update emp set
  sal = sal * 1.1
  where empno = 7369;
 	
1 row updated.
 	
update emp set
  sal = sal * 1.1
  where empno = 7934;
 	
ERROR at line 1:
ORA-00060: deadlock detected
while waiting for resource
	update emp set
	  mgr = 1342
	  where empno = 7934;
	
	1 row updated.
	
	update emp set
	  mgr = 1342
	  where empno = 7369;
	
	Waiting ...
		A
		
		B
		
		C

Example explanation

- At timepoint A
  - no problem exists as each transaction has a row lock on the row it attempts to update.
  - Each transaction proceeds without being terminated.
  - However, each tries next to update the row currently held by the other transaction.
- At timepoint B
  - a deadlock results, because neither transaction can obtain the resource it needs to proceed or terminate.
  - It is a deadlock because no matter how long each transaction waits, the conflicting locks are held.

How to avoid Deadlocks

- Application developers can eliminate all risk of deadlocks by ensuring that transactions requiring multiple resources **always lock them in the same order**.
- However, in complex applications, this is easier said than done, particularly if an ad hoc query tool is used.
- To be safe, you should adopt a strict locking order, but you must also **handle locking exceptions.**
- To handle locking exceptions:
  - Pause for three seconds, and then retry the statement.
- or
  - Roll back the transaction, wait 3 seconds and retry.

How our sample could change:

```
 procedure add_corderline
(onum in builder.corder.corderno%type,  scode  in  builder.stock.stock_code%type,
    qtyreq in builder.corderline.quantityrequired%type)
 as
  cur_q builder.stock.stock_level%type;
  lev   builder.stock.reorder_level%type;
  begin
    select stock_level, reorder_level  into cur_q, lev
    from builder.stock    where stock_code = scode
     FOR UPDATE OF stock_level;
    if (cur_q > qtyreq) then
      update builder.stock set stock_level = stock_level – qtyreq
       where stock_code = scode;
      if (cur_q - qtyreq) < lev then
          insert into builder.restock values (sysdate, scode);
      end if;
      insert into builder.corderline values (qtyreq,onum, scode);
    else
      insert into corderline_error values (sysdate, onum, scode,
       'Do not have enough to sell');
    end if;
  exception
    when others then
      rollback work;
      insert into corderline_error  values (sysdate, onum, scode, 'error inserting order
    detail');
  end;
```

Do we lock it here? If so, When is the work committed!?!

Where should you lock?

- Try to lock at the same level as the commit.
- This concentrates the mind on the locking function and makes it less likely that you'll forget to commit / rollback.
  - Use it in Execorder, rather than custorders.
  - Execorder is our transaction.

Other Processes can	No lock	Row Share Lock	Share Lock	Row Exclusive lock	Exclusive Lock
Select same row	√	√	√	√	√
Select another row	√	√	√	√	√
Insert another row	√	√	x	√	x
Delete another row	√	√	x	√	x
Update another row	√	√	x	√	x
Lock this row	√	x	x	x	x
Lock other rows	√	√	x	√	x
Lock table in share mode	√	√	x	x	x
Alter table	√	x	x	x	x
Drop table	√	x	x	x	x

References

- Examples taken from
- http://www.akadia.com/services/ora_locks_survival_guide.html