Chapter one

Transaction Management and Concurrency Control

1.1 Transaction

Transaction is the action or series of actions that carried out by users or application, which reads or updates the contents of the database. It is a logical unit of works on a database.

A **transaction** is a collection of operations that performs a single logical function in a database application. Each transaction is a unit of both atomicity and consistency, Transforms database from one consistent state to another, although consistency may be violated during transaction. Thus, we require that transactions do not violate any database consistency constraints. That is, if the database was consistent when a transaction started, the database must be consistent when the transaction successfully terminates. However, during the execution of a transaction, it may be necessary temporarily to allow inconsistency, since either the debit of **A** or the credit of **B** must be done before the other. This temporary inconsistency, although necessary, may lead to difficulty if a failure occurs.
It is the programmer's responsibility to define properly the various transactions, so that each preserves the consistency of the database. For example, the transaction to transfer funds from the account of department *A* to the account of department *B* could be defined to be composed of two separate programs: one that debits account *A*, and another that credits account *B*. The execution of these two programs one after the other will indeed preserve consistency. However, each program by itself does not transform the database from a consistent state to a new consistent state. Thus, those programs are not transactions.

Transaction processing systems are systems with large databases and hundreds of concurrent users executing database transactions. Examples of such systems include airline reservations, banking, credit card processing, online retail purchasing, stock markets, supermarket checkouts, and many other applications. These systems require high availability and fast response time for hundreds of concurrent users. We define the concept of a transaction, which is used to represent a logical unit of database processing that must be completed in its entirety to ensure correctness. A transaction is typically implemented by a computer program, which includes database commands such as retrievals, insertions, deletions, and updates.

1.2 Transaction Support

The basic definition of an SQL transaction is similar to our already defined concept of a transaction. That is, it is a logical unit of work and is guaranteed to be atomic. A single SQL statement is always considered to be atomic—either it completes execution without an error or it fails and leaves the database unchanged.

With SQL, there is no explicit Begin_Transaction statement. Transaction initiation is done implicitly when particular SQL statements are encountered. However, every transaction must have an explicit end statement, which is either a COMMIT or a ROLLBACK. Every transaction has certain characteristics attributed to it. These characteristics are specified by a SET TRANSACTION statement in SQL. The characteristics are the *access mode*, the *diagnostic area size*, and the *isolation level*.

The **access mode** can be specified as READ ONLY or READ WRITE. The default is READ WRITE, unless the isolation level of READ UNCOMMITTED is specified (see below), in which case READ ONLY is assumed. A mode of READ WRITE allows select, update, insert, delete, and create commands to be executed. A mode of READ ONLY, as the name implies, is simply for data retrieval.

The **diagnostic area size** option, DIAGNOSTIC SIZE *n*, specifies an integer value *n*, which indicates the number of conditions that can be held simultaneously in the diagnostic area. These conditions supply feedback information (errors or exceptions) to the user or program on the *n* most recently executed SQL statement.

The **isolation level** option is specified using the statement ISOLATION LEVEL <isolation>, where the value for <isolation> can be READ UNCOMMITTED, READ COMMITTED, REPEATABLE READ, or SERIALIZABLE. The default isolation level is SERIALIZABLE, although some systems use READ COMMITTED as their default. The use of the term SERIALIZABLE here is based on not allowing violations that cause dirty read, unrepeatable read, and phantoms, and it is thus not identical to the way serializability. If a transaction executes at a lower isolation level than SERIALIZABLE, then one or more of the following three violations may occur:

1. **Dirty read.** A transaction *T*1 may read the update of a transaction *T*2, which has not yet committed. If *T*2 fails and is aborted, then *T*1 would have read a value that does not exist and is incorrect.
2. **Nonrepeatable read.** A transaction *T*1 may read a given value from a table. If another transaction *T*2 later updates that value and *T*1 reads that value again, *T*1 will see a different value.
3. **Phantoms.** A transaction *T*1 may read a set of rows from a table, perhaps based on some condition specified in the SQL WHERE-clause. Now suppose that a transaction *T*2 inserts a new row *r* that also satisfies the WHERE-clause condition used in *T*1, into the table used by *T*1. The record *r* is called a **phantom record** because it was not there when *T*1 starts but is there when
*T*1 ends. *T*1 may or may not see the phantom, a row that previously did not exist. If the equivalent serial order is *T*1 followed by *T*2, then the record *r* should not be seen; but if it is *T*2 followed by *T*1, then the phantom record should be in the result given to *T*1. If the system cannot ensure the correct behavior, then it does not deal with the phantom record problem.

Table 1.1 Possible Violations Based on Isolation Levels as Defined in SQL

	Type of Violation		
Isolation Level	**Dirty Read**	**Nonrepeatable Read**	**Phantom**
READ UNCOMMITTED	Yes	Yes	Yes
READ COMMITTED	No	Yes	Yes
REPEATABLE READ	No	No	Yes
SERIALIZABLE	No	No	No

Table 1.1 summarizes the possible violations for the different isolation levels. An entry of *Yes* indicates that a violation is possible and an entry of *No* indicates that it is not possible. READ UNCOMMITTED is the most forgiving, and SERIALIZABLE is the most restrictive in that it avoids all three of the problems mentioned above.

A sample SQL transaction might look like the following:

```
EXEC SQL WHENEVER SQLERROR GOTO UNDO;
EXEC SQL SET TRANSACTION
        READ WRITE
        DIAGNOSTIC SIZE 5
        ISOLATION LEVEL SERIALIZABLE;
EXEC SQL INSERT INTO EMPLOYEE (Fname, Lname, Ssn, Dno, Salary)
          VALUES ('Robert', 'Smith', '991004321', 2, 35000);
EXEC SQL UPDATE EMPLOYEE
       SET Salary = Salary * 1.1 WHERE Dno = 2;
 EXEC SQL COMMIT;
 GOTO THE_END;
 UNDO: EXEC SQL ROLLBACK;
 THE_END: ... ;
```

The above transaction consists of first inserting a new row in the EMPLOYEE table and then updating the salary of all employees who work in department 2. If an error occurs on any of the SQL statements, the entire transaction is rolled back. This implies that any updated salary (by this transaction) would be restored to its previous value and that the newly inserted row would be removed. As we have seen, SQL provides a number of transaction-oriented features. The DBA or database programmers can take advantage of these options to try improving transaction performance by relaxing serializability if that is acceptable for their applications. Snapshot isolation will ensure that the phantom record problem does not occur,

since the database transaction, or in some cases the database statement, will only see the records that were committed in the database at the time the transaction starts. Any insertions, deletions, or updates that occur after the transaction starts will not be seen by the transaction.

1.3 Properties of Transaction

- Four basic (ACID) properties of the transaction
  - ✓ **Atomicity:** All or Nothing property.
  - ✓ **Consistency:** Must transform database from one consistent state to another.
  - ✓ **Isolation:** Partial effects of incomplete transactions should not be visible to other transactions.
  - ✓ **Durability:** Effects of a committed transaction are permanent and must not be lost because of later failure.

1.4 Concurrency Control

Processes of managing simultaneous operations on the database without having them interfere with one another. Prevents interference when two or more users are accessing database simultaneously and at least one is updating data. Although two transactions may be correct in themselves, interleaving of operations may produce an incorrect result.

Needs for Concurrency Control

- Three examples of potential problems caused by concurrency:
  - Lost update problem.
  - Uncommitted dependency problem.
  - Inconsistent analysis problem.

1.4.1 Lost update Problem

- Successfully completed update is overridden by another user.
- T_1 withdrawing £10 from an account with bal_x, initially £100.
- T_2 depositing £100 into same account.
- Serially, final balance would be £190.
- Loss of T_2's update avoided by preventing T_1 from reading bal_x until after update.

Time	T_1	T_2	bal_x
t_1		begin_transaction	100
t_2	begin_transaction	read(bal_x)	100
t_3	read(bal_x)	$bal_x = bal_x + 100$	100
t_4	$bal_x = bal_x - 10$	write(bal_x)	200
t_5	write(bal_x)	commit	90
t_6	commit		90

1.4.2 Uncommitted dependency problem

- Occurs when one transaction can see intermediate results of another transaction before it has committed.
- T_4 updates bal_x to £200 but it aborts, so bal_x should be back at original value of £100.
- T_3 has read new value of bal_x (£200) and uses value as basis of £10 reduction, giving a new balance of £190, instead of £90.
- Problem avoided by preventing T_3 from reading bal_x until after T_4 commits or aborts.

Time	T_3	T_4	bal_x
t_1		begin_transaction	100
t_2		read(bal_x)	100
t_3		$bal_x = bal_x + 100$	100
t_4	begin_transaction	write(bal_x)	200
t_5	read(bal_x)	⋮	200
t_6	$bal_x = bal_x - 10$	rollback	100
t_7	write(bal_x)		190
t_8	commit		190

1.4.4 Inconsistent Analysis Problem

- Occurs when transaction reads several values but second transaction updates some of them during execution of first.
- Sometimes referred to as *dirty read* or *unrepeatable read*.
- T_6 is totaling balances of account x (£100), account y (£50), and account z (£25).
- Meantime, T_5 has transferred £10 from bal_x to bal_z, so T_6 now has wrong result (£10 too high).

- Problem avoided by preventing T_6 from reading bal_x and bal_z until after T_5 completed updates.

Time	T_5	T_6	bal_x	bal_y	bal_z	sum
t_1		begin_transaction	100	50	25	
t_2	begin_transaction	sum = 0	100	50	25	0
t_3	read(bal_x)	read(bal_x)	100	50	25	0
t_4	$bal_x = bal_x - 10$	sum = sum + bal_x	100	50	25	100
t_5	write(bal_x)	read(bal_y)	90	50	25	100
t_6	read(bal_z)	sum = sum + bal_y	90	50	25	150
t_7	$bal_z = bal_z + 10$		90	50	25	150
t_8	write(bal_z)		90	50	35	150
t_9	commit	read(bal_z)	90	50	35	150
t_{10}		sum = sum + bal_z	90	50	35	185
t_{11}		commit	90	50	35	185

1.5 Concept of Serializability

- ✓ Objective of a concurrency control protocol is to schedule transactions in such a way as to avoid any interference.
- ✓ Could run transactions serially, but this limits degree of concurrency or parallelism in system. (Most programs block for I/O and most systems have DMA-separate module for I/O)
- ✓ Serializability identifies those executions of transactions guaranteed to ensure consistency.

Schedule: Sequence of reads/writes by set of concurrent transactions.

Serial schedule: Schedule where operations of each transaction are executed consecutively without any interleaved operations from other transactions.

No guarantee that results of all serial executions of a given set of transactions will be identical.

Nonserial schedule: Schedule where operations from set of concurrent transactions are interleaved. Objective of serializability is to find nonserial schedules that allow transactions to execute concurrently without interfering with one another.

In other words, want to find nonserial schedules that are equivalent to *some* serial schedule. Such a schedule is called *serializable*.

❖ In serializability, ordering of read/writes is important:
(a) If two transactions only read a data item, they do not conflict and order is not important.
(b) If two transactions either read or write completely separate data items, they do not conflict and order is not important.

(c) If one transaction writes a data item and another reads or writes same data item, order of execution is important.

Conflict Schedules. Two schedules are said to be **conflict equivalent** if the relative order of any two *conflicting operations* is the same in both schedules.

Time	T_7	T_8	T_7	T_8	T_7	T_8
t_1	begin_transaction		begin_transaction		begin_transaction	
t_2	read(bal_x)		read(bal_x)		read(bal_x)	
t_3	write(bal_x)		write(bal_x)		write(bal_x)	
t_4		begin_transaction		begin_transaction	read(bal_y)	
t_5		read(bal_x)		read(bal_x)	write(bal_y)	
t_6		write(bal_x)	read(bal_y)		commit	
t_7	read(bal_y)			write(bal_x)		begin_transaction
t_8	write(bal_y)		write(bal_y)			read(bal_x)
t_9	commit		commit			write(bal_x)
t_{10}		read(bal_y)		read(bal_y)		read(bal_y)
t_{11}		write(bal_y)		write(bal_y)		write(bal_y)
t_{12}		commit		commit		commit
	(a)		(b)		(c)	

Serializability premises

- Conflict serializable schedule orders any conflicting operations in same way as some serial execution.
- Under *constrained write rule* (transaction updates data item based on its old value, which is first read),
- use *precedence graph* to test for serializability.

Precedence Graph

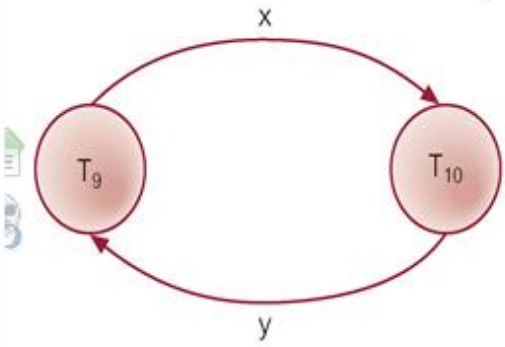
Create:

- node for each transaction;
- a directed edge $T_i \rightarrow T_j$, if T_j reads the value of an item written by T_I;
- a directed edge $T_i \rightarrow T_j$, if T_j writes a value into an item after it has been read by T_i.
- a directed edge $T_i \rightarrow T_j$, if T_j writes a value into an item after it has been written by T_i.
- If precedence graph contains cycle, schedule is not conflict serializable.

Example - Non-conflict serializable schedule

- T_9 is transferring £100 from one account with balance bal_x to another account with balance bal_y.
- T_{10} is increasing balance of these two accounts by 10%.
- Precedence graph has a cycle and so is not serializable.

Time	T_9	T_{10}
t_1	begin_transaction	
t_2	read(bal_x)	
t_3	$bal_x = bal_x + 100$	
t_4	write(bal_x)	begin_transaction
t_5		read(bal_x)
t_6		$bal_x = bal_x * 1.1$
t_7		write(bal_x)
t_8		read(bal_y)
t_9		$bal_y = bal_y * 1.1$
t_{10}		write(bal_y)
t_{11}	read(bal_y)	commit
t_{12}	$bal_y = bal_y - 100$	
t_{13}	write(bal_y)	
t_{14}	commit	



1.6 Recoverability

- ✓ Serializability identifies schedules that maintain database consistency, assuming no transaction fails.
- ✓ Could also examine recoverability of transactions within schedule.
- ✓ If transaction fails, atomicity requires effects of transaction to be undone.
- ✓ Durability states that once transaction commits, its changes cannot be undone (without running another, compensating, transaction).

Recoverable schedule

A schedule where, for each pair of transactions T_i and T_j, if T_j reads a data item previously written by T_i, then the commit operation of T_i precedes the commit operation of T_j.

1.7 Concurrency Control Techniques

- ❖ Two basic pessimistic concurrency control techniques:
  - ✓ **Locking,**
  - ✓ **Timestamping.**
- ❖ Both are conservative(pessimistic) approaches: delay transactions in case they conflict with other transactions.

- ❖ Optimistic methods assume conflict is rare and only check for conflicts at commit time.

1.7.1 Locking

Transaction uses locks to deny access to other transactions and so prevent incorrect updates.

- Most widely used approach to ensure serializability.
- Generally, a transaction must claim a *shared* (*read*) or *exclusive* (*write*) lock on a data item before read or write.
- Lock prevents another transaction from modifying item or even reading it, in the case of a write lock.

Locking - Basic Rules

- If transaction has shared lock on item, can read but not update item.
- If transaction has exclusive lock on item, can both read and update item.
- Reads cannot conflict, so more than one transaction can hold shared locks simultaneously on same item.
- Exclusive lock gives transaction exclusive access to that item.
- Some systems allow transaction to upgrade read lock to an exclusive lock, or downgrade exclusive lock to a shared lock.

Example - Incorrect Locking Schedule

- For two transactions above, a valid schedule using these rules is:

S = {write_lock(T_9, bal_x), read(T_9, bal_x), write(T_9, bal_x), unlock(T_9, bal_x), write_lock(T_{10}, bal_x), read(T_{10}, bal_x), write(T_{10}, bal_x), unlock(T_{10}, bal_x), write_lock(T_{10}, bal_y), read(T_{10}, bal_y), write(T_{10}, bal_y), unlock(T_{10}, bal_y), commit(T_{10}), write_lock(T_9, bal_y), read(T_9, bal_y), write(T_9, bal_y), unlock(T_9, bal_y), commit(T_9) }

- If at start, bal_x = 100, bal_y = 400, result should be:
  - bal_x = 220, bal_y = 330, if T_9 executes before T_{10}, or
  - bal_x = 210, bal_y = 340, if T_{10} executes before T_9.
- However, result gives bal_x = 220 and bal_y = 340.
- S is not a serializable schedule.

- Problem is that transactions release locks too soon, resulting in loss of total isolation and atomicity.
- To guarantee serializability, need an additional protocol concerning the positioning of lock and unlock operations in every transaction.

Two-Phase Locking (2PL)

Transaction follows 2PL protocol if all locking operations precede first unlock operation in the transaction.

- ❖ Two phases for transaction:
  - **Growing phase -** acquires all locks but cannot release any locks.
  - **Shrinking phase**-releases locks but cannot acquire any new locks.
  - Which phases downgrading and upgrading allowed?

Preventing Lost Update Problem using 2PL

Time	T_1	T_2	bal_x
t_1		begin_transaction	100
t_2	begin_transaction	write_lock(bal_x)	100
t_3	write_lock(bal_x)	read(bal_x)	100
t_4	WAIT	$bal_x = bal_x + 100$	100
t_5	WAIT	write(bal_x)	200
t_6	WAIT	commit/unlock(bal_x)	200
t_7	read(bal_x)		200
t_8	$bal_x = bal_x - 10$		200
t_9	write(bal_x)		190
t_{10}	commit/unlock(bal_x)		190

Preventing Uncommitted Dependency Problem using 2PL

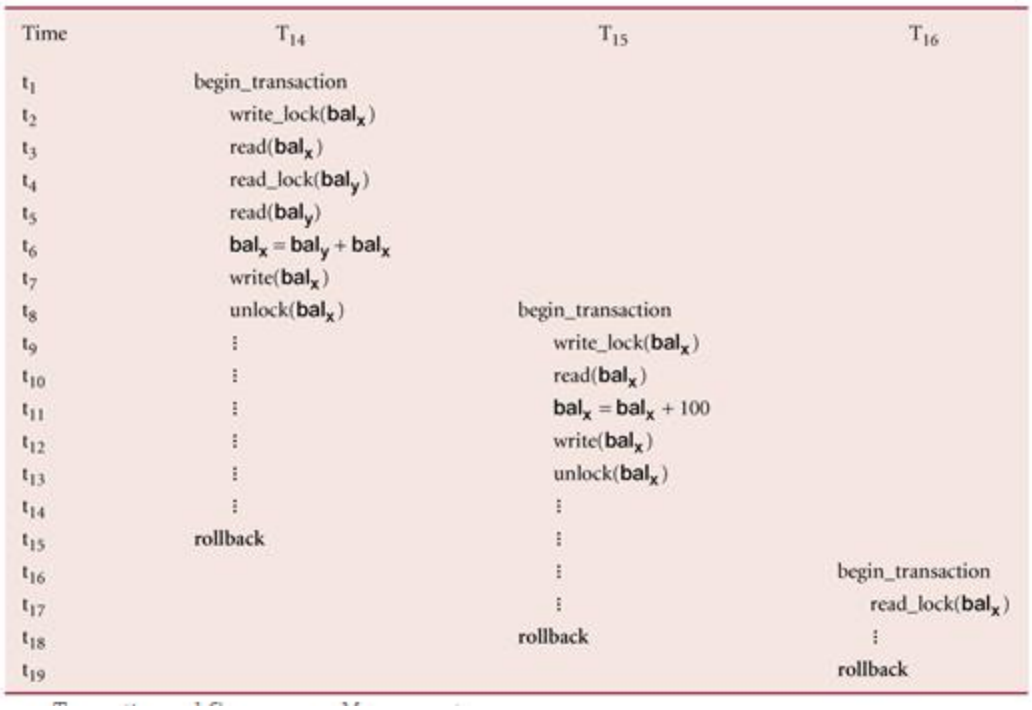
Time	T_3	T_4	bal_x
t_1		begin_transaction	100
t_2		write_lock(bal_x)	100
t_3		read(bal_x)	100
t_4	begin_transaction	$bal_x = bal_x + 100$	100
t_5	write_lock(bal_x)	write(bal_x)	200
t_6	WAIT	rollback/unlock(bal_x)	100
t_7	read(bal_x)		100
t_8	$bal_x = bal_x - 10$		100
t_9	write(bal_x)		90
t_{10}	commit/unlock(bal_x)		90

Preventing Inconsistent Analysis Problem using 2PL

Time	T_5	T_6	bal_x	bal_y	bal_z	sum
t_1		begin_transaction	100	50	25	
t_2	begin_transaction	sum = 0	100	50	25	0
t_3	write_lock(bal_x)		100	50	25	0
t_4	read(bal_x)	read_lock(bal_x)	100	50	25	0
t_5	$bal_x = bal_x - 10$	WAIT	100	50	25	0
t_6	write(bal_x)	WAIT	90	50	25	0
t_7	write_lock(bal_z)	WAIT	90	50	25	0
t_8	read(bal_z)	WAIT	90	50	25	0
t_9	$bal_z = bal_z + 10$	WAIT	90	50	25	0
t_{10}	write(bal_z)	WAIT	90	50	35	0
t_{11}	commit/unlock(bal_x, bal_z)	WAIT	90	50	35	0
t_{12}		read(bal_x)	90	50	35	0
t_{13}		sum = sum + bal_x	90	50	35	90
t_{14}		read_lock(bal_y)	90	50	35	90
t_{15}		read(bal_y)	90	50	35	90
t_{16}		sum = sum + bal_y	90	50	35	140
t_{17}		read_lock(bal_z)	90	50	35	140
t_{18}		read(bal_z)	90	50	35	140
t_{19}		sum = sum + bal_z	90	50	35	175
t_{20}		commit/unlock(bal_x, bal_y, bal_z)	90	50	35	175

Cascading Rollback

- If *every* transaction in a schedule follows 2PL, schedule is serializable.
- However, problems can occur with interpretation of when locks can be released.



Time	T_{14}	T_{15}	T_{16}
t_1	begin_transaction		
t_2	write_lock(bal_x)		
t_3	read(bal_x)		
t_4	read_lock(bal_y)		
t_5	read(bal_y)		
t_6	$bal_x = bal_y + bal_x$		
t_7	write(bal_x)		
t_8	unlock(bal_x)	begin_transaction	
t_9	⋮	write_lock(bal_x)	
t_{10}	⋮	read(bal_x)	
t_{11}	⋮	$bal_x = bal_x + 100$	
t_{12}	⋮	write(bal_x)	
t_{13}	⋮	unlock(bal_x)	
t_{14}	⋮	⋮	
t_{15}	rollback	⋮	
t_{16}		⋮	begin_transaction
t_{17}		⋮	read_lock(bal_x)
t_{18}		rollback	⋮
t_{19}			rollback

- Transactions conform to 2PL.
- T_{14} aborts.
- Since T_{15} is dependent on T_{14}, T_{15} must also be rolled back. Since T_{16} is dependent on T_{15}, it too must be rolled back. This is called ***cascading rollback***.
- To prevent this with 2PL, leave release of *all* locks until end of transaction.

Deadlock

An impasse that may result when two (or more) transactions are each waiting for locks held by the other to be released.

Time	T_{17}	T_{18}
t_1	begin_transaction	
t_2	write_lock(bal_x)	begin_transaction
t_3	read(bal_x)	write_lock(bal_y)
t_4	$bal_x = bal_x - 10$	read(bal_y)
t_5	write(bal_x)	$bal_y = bal_y + 100$
t_6	write_lock(bal_y)	write(bal_y)
t_7	WAIT	write_lock(bal_x)
t_8	WAIT	WAIT
t_9	WAIT	WAIT
t_{10}	⋮	WAIT
t_{11}	⋮	⋮

- Only one way to break deadlock: abort one or more of the transactions.
- Deadlock should be transparent to user, so DBMS should restart transaction(s).
- Three general techniques for handling deadlock:
    - Timeouts.
    - Deadlock prevention.
    - Deadlock detection and recovery.

Timeouts

- Transaction that requests lock will only wait for a system-defined period of time.
- If lock has not been granted within this period, lock request times out.
- In this case, DBMS assumes transaction may be deadlocked, even though it may not be, and it aborts and automatically restarts the transaction.

Deadlock Prevention

- DBMS looks ahead to see if transaction would cause deadlock and never allows deadlock to occur.
- Could order transactions using transaction timestamps:
    - **Wait-Die** - only an older transaction can wait for younger one, otherwise transaction is aborted (*dies*) and restarted with same timestamp.(Why same timestamp?)

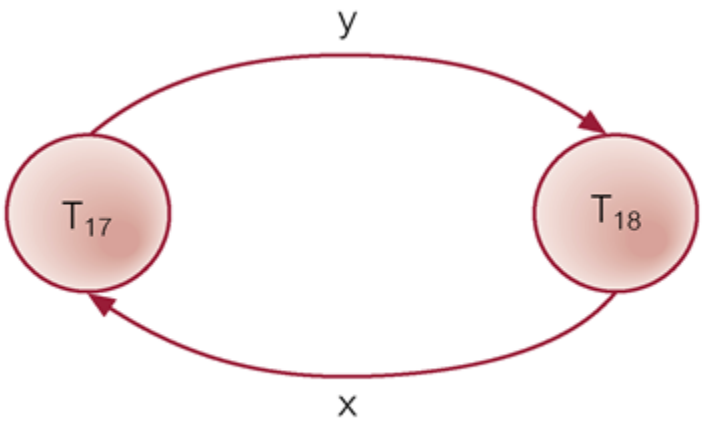
- If ts(T_i)<ts(T_j) then T_i waits else T_i dies
- Prefers younger transaction

- **Wound-Wait -** only a younger transaction can wait for an older one. If older transaction requests lock held by younger one, younger one is aborted (*wounded*).
  - "Wound": The younger transaction rolls back (if it cannot finish in small interval of time) and gives lock to the older one
  - If ts(T_i)<ts(T_j) then T_j is wounded(rolled back) else T_i waits
  - Prefers Older Transaction

Deadlock Detection and Recovery

- DBMS allows deadlock to occur but recognizes it and breaks it.
- Usually handled by construction of wait-for graph (WFG) showing transaction dependencies:
  - Create a node for each transaction.
  - Create edge T_i -> T_j, if T_i is waiting to lock item locked by T_j.
- Deadlock exists if and only if WFG contains cycle.
- WFG is created at regular intervals.

Example - Wait-For-Graph (WFG)



Recovery from Deadlock Detection

- Several issues:

- choice of deadlock victim;
- how far to roll a transaction back;
- Avoiding starvation on some transaction.

1.7.2 Timestamping

- Transactions ordered globally so that older transactions, transactions with *smaller* timestamps, get priority in the event of conflict.
- Conflict is resolved by rolling back and restarting transaction.
- No locks so no deadlock.

Timestamp: A unique identifier created by DBMS that indicates relative starting time of a transaction. Can be generated by using system clock at time transaction started, or by incrementing a logical counter every time a new transaction starts.

Timestamping – procedure

- Read/write proceeds only if *last update on that data item* was carried out by an older transaction.
- Otherwise, transaction requesting read/write is restarted and given a new timestamp.
- Also timestamps for data items:
  - **read-timestamp** - timestamp of last transaction to read item;
  - **write-timestamp** - timestamp of last transaction to write item.

Timestamping - Read(x)

- Consider a transaction T with timestamp ts(T):
- Check the last write on the Data Item ts(T) < WTS(x)
  - x already updated by younger (later) transaction.
  - Transaction must be aborted and restarted with a new timestamp.
  - Otherwise the Read continues and the RTS(X) will be set to the max of (RTS(x) and ts(T))

Timestamping - Write(x)

Check the last read and write

If ts(T) < RTS(x)

- X is already read by a younger transaction
- Hence error to update now and restarted with new time stamp

If ts(T) < WTS (x)

- x already written by younger transaction.
- Write can safely be ignored - *ignore obsolete write* rule. (Don't need to restart the transaction)

- Otherwise, operation is accepted and executed.
  - WTS(x) is set to ts(T)

Example – Basic Timestamp Ordering

Time	Op	T_{19}	T_{20}	T_{21}
t_1		begin_transaction		
t_2	read(bal_x)	read(bal_x)		
t_3	$bal_x = bal_x + 10$	$bal_x = bal_x + 10$		
t_4	write(bal_x)	write(bal_x)	begin_transaction	
t_5	read(bal_y)		read(bal_y)	
t_6	$bal_y = bal_y + 20$		$bal_y = bal_y + 20$	begin_transaction
t_7	read(bal_y)			read(bal_y)
t_8	write(bal_y)		write(bal_y)+	
t_9	$bal_y = bal_y + 30$			$bal_y = bal_y + 30$
t_{10}	write(bal_y)			write(bal_y)
t_{11}	$bal_z = 100$			$bal_z = 100$
t_{12}	write(bal_z)			write(bal_z)
t_{13}	$bal_z = 50$	$bal_z = 50$		commit
t_{14}	write(bal_z)	write(bal_z)‡	begin_transaction	
t_{15}	read(bal_y)	commit	read(bal_y)	
t_{16}	$bal_y = bal_y + 20$		$bal_y = bal_y + 20$	
t_{17}	write(bal_y)		write(bal_y)	
t_{18}			commit	

+ At time t_8, the write by transaction T_{20} violates the first timestamping write rule described above and therefore is aborted and restarted at time t_{14}.
‡ At time t_{14}, the write by transaction T_{19} can safely be ignored using the ignore obsolete write rule, as it would have been overwritten by the write of transaction T_{21} at time t_{12}.

Optimistic Techniques

- Based on assumption that conflict is rare and more efficient to let transactions proceed without delays to ensure serializability.
- At commit, check is made to determine whether conflict has occurred.
- If there is a conflict, transaction must be rolled back and restarted.

- Potentially allows greater concurrency than traditional protocols.
- **Three phases:**
- Read
- Validation
- Write

Optimistic Techniques - Read Phase

- Extends from start until immediately before commit.
- Transaction reads values from database and stores them in local variables. Updates are applied to a local copy of the data.

Optimistic Techniques - Validation Phase

- Follows the read phase.
- For read-only transaction, checks that data read are still current values. If no interference, transaction is committed, else aborted and restarted.
- For update transaction, checks transaction leaves database in a consistent state, with serializability maintained.

Optimistic Techniques - Write Phase

- Follows successful validation phase for update transactions.
- Updates made to local copy are applied to the database.

Granularity of Data Items

- Size of data items chosen as unit of protection by concurrency control protocol.
- Ranging from coarse to fine:
    - The entire database.
    - A file.
    - A page (or area or database spaced).
    - A record.
    - A field value of a record.

      Tradeoff:

- coarser, the lower the degree of concurrency;
- finer, more locking information that is needed to be stored.
- Best item size depends on the types of transactions.

Hierarchy of Granularity

- Could represent granularity of locks in a hierarchical structure.
- Root node represents entire database, level 1s represent files, etc.
- When node is locked, all its descendants are also locked.
- DBMS should check hierarchical path before granting lock.

1.8 Database Recovery

Process of restoring database to a correct state in the event of a failure.

- Need for Recovery Control
  - Two types of storage: volatile (main memory) and nonvolatile.
  - Volatile storage does not survive system crashes.
  - Stable storage represents information that has been replicated in several nonvolatile storage media with independent failure modes.

Types of Failures

- System crashes, resulting in loss of main memory.
- Media failures, resulting in loss of parts of secondary storage.
- Application software errors.
- Natural physical disasters.
- Carelessness or unintentional destruction of data or facilities.
- Sabotage.

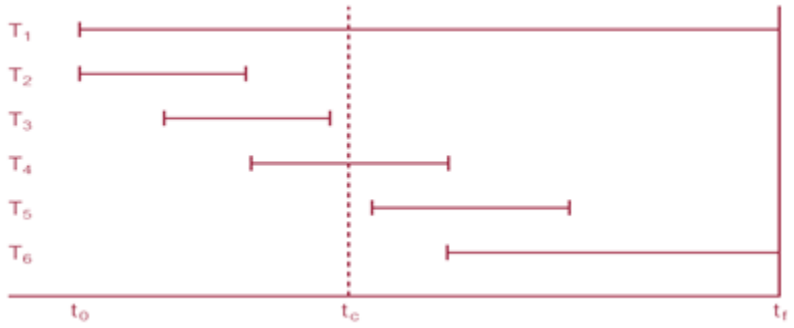
1.9 Transactions and Recovery

- ✓ Transactions represent basic unit of recovery.
- ✓ Recovery manager responsible for atomicity and durability.
- ✓ If failure occurs between commit and database buffers being flushed to secondary storage then, to ensure durability, recovery manager has to *redo* (*rollforward*) transaction's updates.

- ✓ If transaction had not committed at failure time, recovery manager has to *undo* (*rollback*) any effects of that transaction for atomicity.
- ✓ Partial undo - only one transaction has to be undone.
- ✓ Global undo - all transactions have to be undone.

Example

- DBMS starts at time t_0, but fails at time t_f. Assume data for transactions T_2 and T_3 have been written to secondary storage.
- T_1 and T_6 have to be undone. In absence of any other information, recovery manager has to redo T_2, T_3, T_4, and T_5.



1.10 Recovery Facilities

- ✓ **DBMS** should provide following facilities to assist with recovery:
- ✓ Backup mechanism, which makes periodic backup copies of database.
- ✓ Logging facilities, which keep track of current state of transactions and database changes.
- ✓ Checkpoint facility, which enables updates to database in progress to be made permanent.
- ✓ Recovery manager, which allows **DBMS** to restore database to consistent state following a failure.

Log File

- Contains information about all updates to database:
  - Transaction records.
  - Checkpoint records.
- Often used for other purposes (for example, auditing).
- Transaction records contain:
- Transaction identifier.
- Type of log record, (transaction start, insert, update, delete, abort, commit).

- Identifier of data item affected by database action (insert, delete, and update operations).
- Before-image of data item.
- After-image of data item.
- Log management information.

Checkpointing

Checkpoint: Point of synchronization between database and log file. All buffers are force-written to secondary storage.

- Checkpoint record is created containing identifiers of all active transactions.
- When failure occurs, redo all transactions that committed since the checkpoint and undo all transactions active at time of crash.
- In previous example, with checkpoint at time t_c, changes made by T_2 and T_3 have been written to secondary storage.

  Thus:

- only redo T_4 and T_5,
- undo transactions T_1 and T6.

1.11 Recovery Techniques

- If database has been damaged:
  - ✓ Need to restore last backup copy of database and reapply updates of committed transactions using log file.
- If database is only inconsistent:
  - ✓ Need to undo changes that caused inconsistency. May also need to redo some transactions to ensure updates reach secondary storage.
  - ✓ Do not need backup, but can restore database using before- and after-images in the log file.

Main Recovery Techniques

❖ Three main recovery techniques:
- Deferred Update
- Immediate Update
- Shadow Paging

1.11.1 Deferred Update

- Updates are not written to the database until after a transaction has reached its commit point.

- If transaction fails before commit, it will not have modified database and so no undoing of changes required.
- May be necessary to redo updates of committed transactions as their effect may not have reached database.

1.11.2 Immediate Update

- Updates are applied to database as they occur.
- Need to redo updates of committed transactions following a failure.
- May need to undo effects of transactions that had not committed at time of failure.
- Essential that log records are written before write to database. *Write-ahead log protocol*.
- If no "*transaction commit*" record in log, then that transaction was active at failure and must be undone.
- Undo operations are performed *in reverse order in which they were written to log*.

1.11.3 Shadow Paging

- Maintain two page tables during life of a transaction: *current* page and *shadow* page table.
- When transaction starts, two pages are the same.
- Shadow page table is never changed thereafter and is used to restore database in event of failure.
- During transaction, current page table records all updates to database.
- When transaction completes, current page table becomes shadow page table.

Chapter Two

Query Processing and Optimization

Objective

- Objectives of query processing and optimization.
- Static versus dynamic query optimization.
- How a query is decomposed and semantically analyzed.
- How to create a R.A.T. to represent a query.
- Rules of equivalence for RA operations.
- How to apply heuristic transformation rules to improve efficiency of a query.
- Types of database statistics required to estimate cost of operations.
- How pipelining can be used to improve efficiency of queries.
- Difference between materialization and pipelining.

2.1 Introduction

- In network and hierarchical DBMSs, low-level procedural query language is generally embedded in high-level programming language.
- Programmer's responsibility to select most appropriate execution strategy.
- With declarative languages such as SQL, user specifies what data is required rather than how it is to be retrieved.
- Relieves user of knowing what constitutes good execution strategy.
- Also gives DBMS more control over system performance.
- Two main techniques for query optimization:
    - heuristic rules that order operations in a query;
    - comparing different strategies based on relative costs, and selecting one that minimizes resource usage.
- Disk access tends to be dominant cost in query processing for centralized DBMS.

2.2 Query Processing

Activities involved in retrieving data from the database.

❖ Aims of QP:

- transform query written in high-level language (e.g. SQL), into correct and efficient execution strategy expressed in low-level language (implementing RA);
- execute strategy to retrieve required data.

2.3 Query Optimization

Activity of choosing an efficient execution strategy for processing query.

- As there are many equivalent transformations of same high-level query, aim of QO is to choose one that minimizes resource usage.
- Generally, reduce total execution time of query.
- May also reduce response time of query.

Example - Different Strategies

Find all Managers who work at a London branch.

SELECT *

FROM Staff s, Branch b

WHERE s.branchNo = b.branchNo AND

(s.position = 'Manager' AND b.city = 'London');

Three equivalent RA queries are:

(1) $\sigma_{(position='Manager') \wedge (city='London') \wedge (Staff.branchNo=Branch.branchNo)}$ **(Staff X Branch)**

(2) $\sigma_{(position='Manager') \wedge (city='London')}$**(**

Staff $_{Staff.branchNo=Branch.branchNo}$ **Branch)**

(3) $(\sigma_{position='Manager'}$**(Staff))** $_{Staff.branchNo=Branch.branchNo}$

$(\sigma_{city='London'}$ **(Branch))**

Assume:

- 1000 tuples in Staff; 50 tuples in Branch;
- 50 Managers; 5 London branches;
- no indexes or sort keys;

- results of any intermediate operations stored on disk;
- cost of the final write is ignored;
- tuples are accessed one at a time.

Analysis of each Query Expression

- ***Analysis 1:***
  i. *read each tuple from the two relations →n+m reads*
  ii. *create a table of the Cartesian product→ nXm writes*
  iii. *test each tuple of step 2→nXm read*
- ***Total No. of Disk access: →2(nXm) +n+m***
- ***Analysis 2:***
  i. *read each tuple from the two relations →n+m reads*
  ii. *create a table of the Join→ n writes*
  iii. *test each tuple of step 2→n read*
- ***Total No. of Disk access: →3(n) +m***
- ***Analysis 3:***
  i. *read each tuple from the two relations →n+m reads*
  ii. *create a table of the Join→ n writes*
  iii. *test each tuple of step 2→n read*
- ***Total No. of Disk access: →3(n) +m***

Example - Cost Comparison

- **Cost (in disk accesses) are:**

(1) (1000 + 50) + 2*(1000 * 50) = 101 050

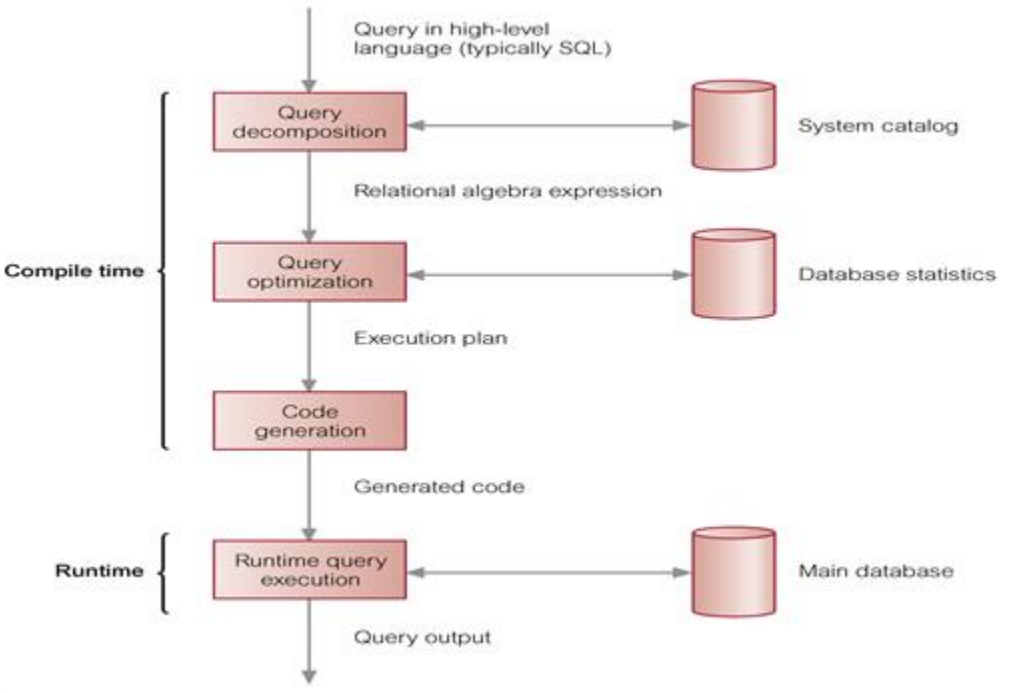
(2) 2*1000 + (1000 + 50) = 3 050

(3) 1000 + 2*50 + 5 + (50 + 5) = 1 160

- Cartesian product and join operations much more expensive than selection, and third option significantly reduces size of relations being joined together.

2.4 Phases of Query Processing

- Query process has four main phases:
    - decomposition (consisting of parsing and validation);
    - optimization;
    - code generation;
    - execution.



Dynamic versus Static Optimization

- Two times when first three phases of QP can be carried out:
    - dynamically every time query is run;
    - statically when query is first submitted.
- Advantages of dynamic query optimization arise from fact that information is up to date.
- Disadvantages are that performance of query is affected, time may limit finding optimum strategy.
- Advantages of static query optimization are removal of runtime overhead, and more time to find optimum strategy.

- Disadvantages arise from fact that chosen execution strategy may no longer be optimal when query is run.
- Could use a hybrid approach to overcome this.

2.5 Query Decomposition

- Aims are to transform high-level query into RA query and check that query is syntactically and semantically correct.
- Typical stages are:
    - analysis,
    - normalization,
    - semantic analysis,
    - simplification,
    - query restructuring.

Analysis

- Analyze query lexically and syntactically using compiler techniques.
- Verify relations and attributes exist.
- Verify operations are appropriate for object type.

Analysis – Example

SELECT staff_no

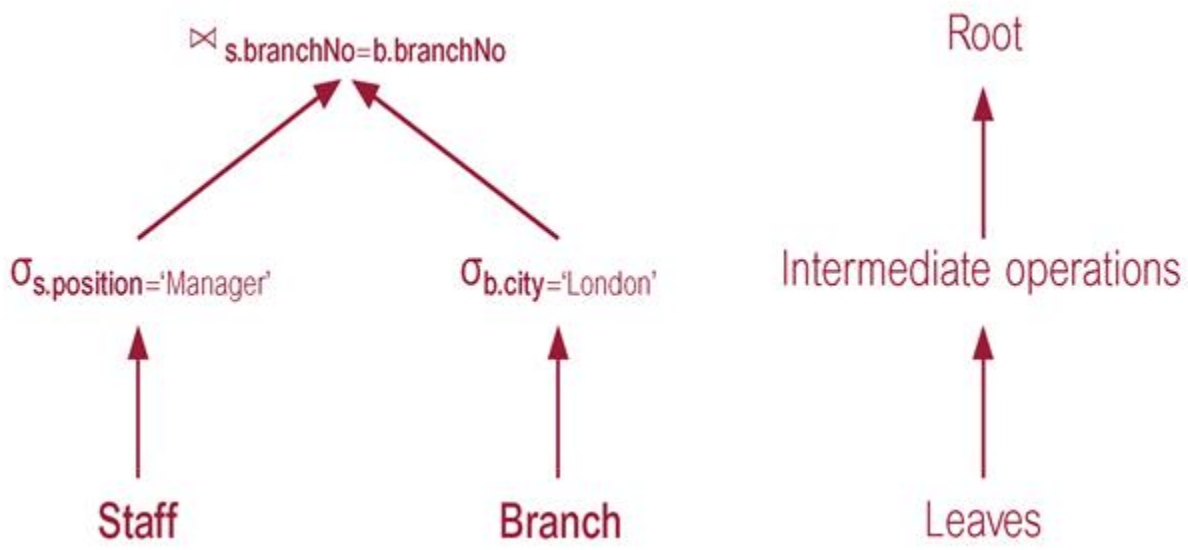
FROM Staff

WHERE position > 10;

- This query would be rejected on two grounds:
    - staff_no is not defined for Staff relation (should be staffNo).
    - Comparison '>10' is incompatible with type position, which is variable character string.
- Finally, query transformed into some internal representation more suitable for processing.
- Some kind of query tree is typically chosen, constructed as follows:
    - Leaf node created for each base relation.

- Non-leaf node created for each intermediate relation produced by RA operation.
- Root of tree represents query result.
- Sequence is directed from leaves to root.

Example - R.A.T



Normalization

- Converts query into a normalized form for easier manipulation.
- Predicate can be converted into one of two forms:

Conjunctive normal form:

(position = 'Manager' ∨ salary > 20000) ∧ (branchNo = 'B003')

Disjunctive normal form:

(position = 'Manager' ∧ branchNo = 'B003') ∨

(salary > 20000 ∧ branchNo = 'B003')

Semantic Analysis

- Rejects normalized queries that are incorrectly formulated or contradictory.

- Query is incorrectly formulated if components do not contribute to generation of result.
- Query is contradictory if its predicate cannot be satisfied by any tuple.
- Algorithms to determine correctness exist only for queries that do not contain disjunction and negation.
- For these queries, could construct:
- A relation connection graph.
- Normalized attribute connection graph.
- Relation connection graph
- Create node for each relation and node for result. Create edges between two nodes that represent a join, and edges between nodes that represent projection.
- If not connected, query is incorrectly formulated.

Example - Checking Semantic Correctness

SELECT p.propertyNo, p.street

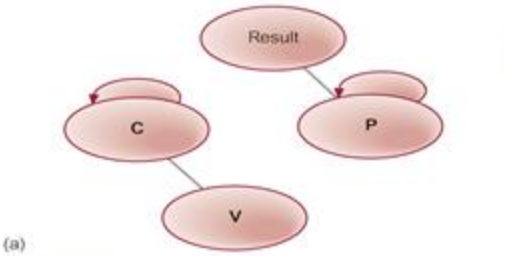
FROM Client c, Viewing v, PropertyForRent p

WHERE c.clientNo = v.clientNo AND

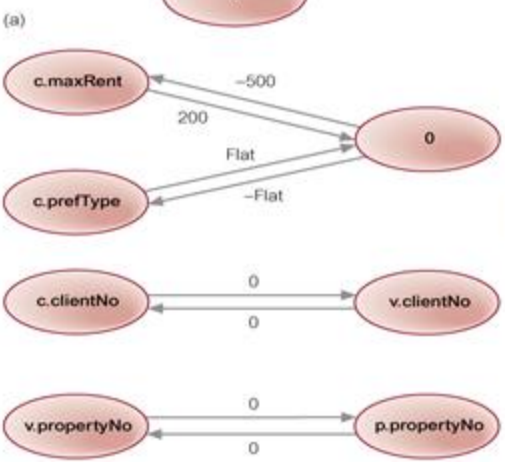
c.maxRent >= 500 AND

c.prefType = ‘Flat’ AND p.ownerNo = ‘CO93’;

- Relation connection graph not fully connected, so query is not correctly formulated.
- Have omitted the join condition (v.propertyNo = p.propertyNo) .



Relation Connection graph



Normalized attribute connection graph

SELECT p.propertyNo, p.street

FROM Client c, Viewing v, PropertyForRent p

WHERE c.maxRent > 500 AND

c.clientNo = v.clientNo AND

v.propertyNo = p.propertyNo AND

c.prefType = ‘Flat’ AND c.maxRent < 200;

- Normalized attribute connection graph has cycle between nodes c.maxRent and 0 with negative valuation sum, so query is contradictory.

Simplification

- Detects redundant qualifications,
- eliminates common sub-expressions,
- transforms query to semantically equivalent but more easily and efficiently computed form.

- Typically, access restrictions, view definitions, and integrity constraints are considered.
- Assuming user has appropriate access privileges, first apply well-known idempotency rules of boolean algebra.

- Examples: for a two predicates p and q , P V~P= True, p ∧ (qv~q)=P …etc.
- In relational Algebra, we have Transformation rules to do so

2.6 Transformation Rules for RA Operations

Conjunctive Selection operations can cascade into individual Selection operations (and vice versa).

$\sigma_{p \wedge q \wedge r}(R) = \sigma_p(\sigma_q(\sigma_r(R)))$

- Sometimes referred to as cascade of Selection.

$\sigma_{branchNo='B003' \wedge salary>15000}(Staff) = \sigma_{branchNo='B003'}(\sigma_{salary>15000}(Staff))$

Commutativity of Selection.

$\sigma_p(\sigma_q(R)) = \sigma_q(\sigma_p(R))$

- For example:

$\sigma_{branchNo='B003'}(\sigma_{salary>15000}(Staff)) = \sigma_{salary>15000}(\sigma_{branchNo='B003'}(Staff))$

In a sequence of Projection operations, only the last in the sequence is required.

$\Pi_L \Pi_M \ldots \Pi_N(R) = \Pi_L(R)$, provided that **L** is in **M and M** is in **N**

- **For example:**

$\Pi_{lName} \Pi_{branchNo, lName}(Staff) = \Pi_{lName}(Staff)$

Commutativity of Selection and Projection.

- If predicate p involves only attributes in projection list, Selection and Projection operations commute:

$\Pi_{A_i, \ldots, A_m}(\sigma_p(R)) = \sigma_p(\Pi_{A_i, \ldots, A_m}(R))$ **where** $p \in \{A_1, A_2, \ldots, A_m\}$

- **For example:**

$\Pi_{fName, lName}(\sigma_{lName='Beech'}(Staff)) = \sigma_{lName='Beech'}(\Pi_{fName,lName}(Staff))$

Commutativity of Theta join (and Cartesian product).

$R \bowtie_p S = S \bowtie_p R$

$R \times S = S \times R$

- **Rule also applies to Equijoin and Natural join. For example:**

  Staff $\bowtie_{staff.branchNo=branch.branchNo}$ **Branch =**

  Branch $\bowtie_{staff.branchNo=branch.branchNo}$ **Staff**

Commutativity of Selection and Theta join (or Cartesian product).

- If selection predicate involves only attributes of one of join relations, Selection and Join (or Cartesian product) operations commute:

  $\sigma_p(R \bowtie_r S) = (\sigma_p(R)) \bowtie_r S$

  $\sigma_p(R \times S) = (\sigma_p(R)) \times S$

  where $p \in \{A_1, A_2, \ldots, A_n\}$ which are the attributes of R.

- If selection predicate is conjunctive predicate having form $(p \wedge q)$, where p only involves attributes of R, and q only attributes of S, Selection and Theta join operations commute as:

  $\sigma_{p \wedge q}(R \bowtie_r S) = (\sigma_p(R)) \bowtie_r (\sigma_q(S))$

  $\sigma_{p \wedge q}(R \times S) = (\sigma_p(R)) \times (\sigma_q(S))$

- **For example:**

$\sigma_{position='Manager' \wedge city='London'}$**(Staff)** $\bowtie_{Staff.branchNo=Branch.branchNo}$ **Branch) =**

($\sigma_{position='Manager'}$**(Staff))** $\bowtie_{Staff.branchNo=Branch.branchNo}$ **(**$\sigma_{city='London'}$ **(Branch))**

Commutativity of Projection and Theta join (or Cartesian product).

- **If projection list is of form $L = L_1 \cup L_2$, where L_1 only has attributes of R, and L_2 only has attributes of S, provided join condition only contains attributes of L, Projection and Theta join commute:**

  $\Pi_{L1 \cup L2}(R \bowtie_r S) = (\Pi_{L1}(R)) \bowtie_r (\Pi_{L2}(S))$

- **If join condition contains additional attributes not in L ($M = M_1 \cup M_2$ where M_1 only has attributes of R, and M_2 only has attributes of S), a final projection operation is required:**

$\Pi_{L1 \cup L2}(R \bowtie_r S) = \Pi_{L1 \cup L2}((\Pi_{L1 \cup M1}(R)) \bowtie_r (\Pi_{L2 \cup M2}(S)))$

- **For example:**

$\Pi_{position,city,branchNo}(Staff \bowtie_{Staff.branchNo=Branch.branchNo} Branch) =$

$(\Pi_{position,\ branchNo}(Staff)) \bowtie_{Staff.branchNo=Branch.branchNo} ($

$\Pi_{city,\ branchNo}(Branch))$

- **and using the latter rule:**

$\Pi_{position,\ city}(Staff \bowtie_{Staff.branchNo=Branch.branchNo} Branch) =$

$\Pi_{position,\ city}((\Pi_{position,\ branchNo}(Staff)) \bowtie_{Staff.branchNo=Branch.branchNo} (\Pi_{city,\ branchNo}(Branch)))$

Commutativity of Union and Intersection (but not set difference).

$R \cup S = S \cup R$

$R \cap S = S \cap R$

$R - S \neq S - R$

Commutativity of Selection and set operations (Union, Intersection, and Set difference).

$\sigma_p(R \cup S) = \sigma_p(R) \cup \sigma_p(S)$

$\sigma_p(R \cap S) = \sigma_p(R) \cap \sigma_p(S)$

$\sigma_p(R - S) = \sigma_p(R) - \sigma_p(S)$

Commutativity of Projection and Union.

$\Pi_L(R \cup S) = \Pi_L(S) \cup \Pi_L(R)$

Associativity of Union and Intersection (but not Set difference).

$(R \cup S) \cup T = S \cup (R \cup T)$

$(R \cap S) \cap T = S \cap (R \cap T)$

Associativity of Theta join (and Cartesian product).

- **Cartesian product and Natural join are always associative:**

$(R \bowtie S) \bowtie T = R \bowtie (S \bowtie T)$

$(R \times S) \times T = R \times (S \times T)$

- **If join condition q involves attributes only from S and T, then Theta join is associative:**

$(R \bowtie_p S) \bowtie_{q \wedge r} T = R \bowtie_{p \wedge r} (S \bowtie_q T)$

- **For example:**

$(Staff \bowtie_{Staff.staffNo=PropertyForRent.staffNo} PropertyForRent) \bowtie_{ownerNo=Owner.ownerNo \wedge staff.lName=Owner.lName} Owner =$

$Staff \bowtie_{staff.staffNo=PropertyForRent.staffNo \wedge staff.lName=lName} (PropertyForRent \bowtie_{ownerNo} Owner)$

Example Use of Transformation Rules

For prospective renters of flats, find properties that match requirements and owned by CO93.

```
SELECT p.propertyNo, p.street
FROM Client c, Viewing v, PropertyForRent p
WHERE   c.prefType = 'Flat' AND
                    c.clientNo = v.clientNo AND
                    v.propertyNo = p.propertyNo AND
                    c.maxRent >= p.rent AND
                    c.prefType = p.type AND
                    p.ownerNo = 'CO93';
```

Heuristical Processing Strategies

- Perform Selection operations as early as possible.

- Keep predicates on same relation together. Conjunctive selects

  →individual selects

- Push Selections to the respective tables
- Use associativity of binary operations to rearrange leaf nodes so leaf nodes with most restrictive Selection operations executed first.
- Combine Cartesian product with subsequent selection whose predicate represents join condition into a Join operation.
- Perform Projection as early as possible.
- Keep projection attributes on same relation together.
- Push Projection to the respective tables.

Cost Estimation for RA Operations

- Many different ways of implementing RA operations.
- Aim of QO is to choose most efficient one.
- Use formulae that estimate costs for a number of options, and select one with lowest cost.
- Consider only cost of disk access, which is usually dominant cost in QP.
- Many estimates are based on cardinality of the relation, so need to be able to estimate this.

2.7 Database Statistics

- Success of estimation depends on amount and currency of statistical information DBMS holds.
- Keeping statistics current can be problematic.
- If statistics updated every time tuple is changed, this would impact performance.
- DBMS could update statistics on a periodic basis, for example nightly, or whenever the system is idle.

Typical Statistics for Relation R

nTuples(R) - number of tuples in **R**.

bFactor(R) - blocking factor of R (Number of tuples in a block).

nBlocks(R) - number of blocks required to store R:

nBlocks(R) = [nTuples(R)/bFactor(R)]

Typical Statistics for Attribute A of Relation R

$nDistinct_A(R)$ - number of distinct values that

appear for attribute A in R.

$min_A(R)$,$max_A(R)$

- minimum and maximum possible values for attribute A in R.

$SC_A(R)$ - *selection cardinality* of attribute A in R.

Average number of tuples that satisfy an equality condition on attribute A.

2.8 Pipelining

- Materialization - output of one operation is stored in temporary relation for processing by next. (heuristic approach)
- Could also pipeline results of one operation to another without creating temporary relation.
- Known as pipelining or on-the-fly processing.
- Pipelining can save on cost of creating temporary relations and reading results back in again.
- Generally, pipeline is implemented as separate process or thread.

Chapter Three

Database Integrity, Security and Recovery

Objective

- The scope of database security.
- Why database security is a serious concern for an organization.
- The type of threats that can affect a database system.
- How to protect a computer system using computer-based controls.
- Techniques for securing a DBMS on the Web.

3.1 Database Security

- **Data** is a valuable resource that must be strictly controlled and managed, as with any corporate resource.
- Part or all of the corporate data may have strategic importance and therefore needs to be kept secure and confidential.
- Mechanisms that protect the database against intentional or accidental threats.
- Security considerations do not only apply to the data held in a database. Breaches of security may affect other parts of the system, which may in turn affect the database.
- Involves measures to avoid the following threats:
    - ✓ Theft and fraud
    - ✓ Loss of confidentiality (secrecy)
    - ✓ Loss of privacy
    - ✓ Loss of integrity
    - ✓ Loss of availability

Threat

- Any situation or event, whether intentional or unintentional, that will adversely affect a system and consequently an organization.

Levels of security measures

- **Physical Level**: concerned with securing the site containing the computer system should be physically secured. The backup systems should also be physically protected from access except for authorized users.

- **Human Level**: concerned with authorization of database users for access to the content at different levels and privileges.
- **Operating System**: concerned with the weakness and strength of the operating system security on data files. Weakness may serve as a means of unauthorized access to the database. This also includes protection of data in primary and secondary memory from unauthorized access.
- **Database System**: concerned with data access limit enforced by the database system. Access limit like password, isolated transaction and etc.
- **Communication Network** : Securing Data In transit.
- **Application Level**: Different Application Software need to have their own Security mechanism. –eg Authentication

3.2 Countermeasures – Computer-Based Controls

- Concerned with physical controls to administrative procedures and includes:
    - Authorization
    - Access controls
    - Views
    - Backup and recovery
    - Integrity
    - Encryption
    - RAID technology

Authorization

- The granting of a right or privilege, which enables a subject to legitimately have access to a system or a system's object.
- Authorization is a mechanism that determines whether a user is, who he or she claims to be.
- The granting of a right or privilege that enables a subject to have legitimate access to a system or a system's object
- The access allowed to a user could be for data manipulation or control

- Authorization controls can be built into the software, and govern not only what system or object a specified user can access, but also what the user may do with it
- Authorization controls are sometimes referred to as ***access controls***
- The process of authorization involves authentication of ***subjects*** (i.e. a user or program) requesting access to ***objects*** (i.e. a database table, view, procedure, trigger, or any other object that can be created within the system)

Forms of user authorization on data

- **User authorization on the data/extension**
- **Read Authorization**: the user with this privilege is allowed only to read the content of the data object.
- **Insert Authorization**: the user with this privilege is allowed only to insert new records or items to the data object.
- **Update Authorization**: users with this privilege are allowed to modify content of attributes but are not authorized to delete the records.
- **Delete Authorization**: users with this privilege are only allowed to delete a record and not anything else.

User authorization on the database schema

- **Index Authorization**: deals with permission to create as well as delete an index table for relation.
- **Resource Authorization**: deals with permission to add/create a new relation in the database.
- **Alteration Authorization**: deals with permission to add as well as delete attribute.
- **Drop Authorization**: deals with permission to delete and existing relation.

Roles of the DBA in DB security

- ***Account Creation:*** involves creating different accounts for different **users** as well as **user groups.**
- ***Security Level Assignment***: involves in assigning different users at different categories of access levels.
- ***Privilege Grant:*** involves giving different levels of privileges for different users and user groups.

- ***Privilege Revocation:*** involves denying or canceling previously granted privileges for users due to various reasons.
- ***Account Deletion:*** involves in deleting an existing account of users or user groups. Is similar with denying all privileges of users on the database.

  Access control

  - Based on the granting and revoking of privileges.
  - A privilege allows a user to create or access (that is read, write, or modify) some database object (such as a relation, view, and index) or to run certain DBMS utilities.
  - Privileges are granted to users to accomplish the tasks required for their jobs.

Security at different Levels of Data

- Almost all RDBMSs provide security at different levels and formats of data. This includes:
- **Relation Level:** permission to have access to a specific relation.
- **View Level:** permission to data included in the view and not in the named relations
- **Hybrid (Relation/View):** the case where only part of a single relation is made available to users through View.

Database Access Request

- Any database access request will have the following three major components
- **Requested Operation:** what kind of operation is requested by a specific query?
- **Requested Object:** on which resource or data of the database is the operation sought to be applied?
- **Requesting User:** who is the user requesting the operation on the specified object?
- The database should be able to check for all the three components before processing any request. The checking is performed by the security subsystem of the DBMS.

Access Control models

- Most DBMS provide an approach called Discretionary Access Control (DAC).
- SQL standard supports DAC through the GRANT and REVOKE commands.

- The GRANT command gives privileges to users, and the REVOKE command takes away privileges.
- DAC while effective has certain weaknesses. In particular an unauthorized user can trick an authorized user into disclosing sensitive data.
- An additional approach is required called Mandatory Access Control (MAC).
- MAC is based on system-wide policies that cannot be changed by individual users.
- Each database object is assigned a *security class* and each user is assigned a *clearance* for a security class, and *rules* are imposed on reading and writing of database objects by users.
- MAC determines whether a user can read or write an object based on rules that involve the security level of the object and the clearance of the user. These rules ensure that sensitive data can never be 'passed on' to another user without the necessary clearance.
- The SQL standard does *not* include support for MAC.

View

- Is the dynamic result of one or more relational operations operating on the base relations to produce another relation.
- A view is a virtual relation that does not actually exist in the database, but is produced upon request by a particular user, at the time of request.
- The view mechanism provides a powerful and flexible security mechanism by hiding parts of the database from certain users
- Using a view is more restrictive than simply having certain privileges granted to a user on the base relation(s)

Backup

- Process of periodically taking a copy of the database and log file (and possibly programs) to offline storage media.

Journaling

Process of keeping and maintaining a log file (or journal) of all changes made to database to enable effective recovery in event of failure.

- Restoring Database is done by restoring the database to the latest Back then applying the Log file

Integrity

- Prevents data from becoming invalid, and hence giving misleading or incorrect results. Different types of constraints
    - Primary key
    - Default
    - Foreign key
    - Unique key
    - Check

Encryption

- The encoding of the data by a special algorithm that renders the data unreadable by any program without the decryption key.

- Encryption can be used for both storing securely in a shared/Multiuser environment or for transmitting data securely
- Four main components
    - Encryption key
    - Encryption algorithm
    - Decryption key
    - Decryption algorithm

Types of cryptosystem

- Cryptosystems can be categorized into two
- ***Symmetric encryption*** – uses the same key for both encryption and decryption and relies on safe communication lines for exchanging the key.
- ***Asymmetric encryption*** – uses different keys for encryption and decryption
- Generally, symmetric algorithms are much faster to execute on a computer than those that are asymmetric. In the contrary, asymmetric algorithms are more secure than symmetric algorithms

RAID (Redundant Array of Independent Disks) Technology

- Hardware that the DBMS is running on must be *fault-tolerant*, meaning that the DBMS should continue to operate even if one of the hardware components fails.
- Suggests having redundant components that can be seamlessly integrated into the working system whenever there is one or more component failures.
- The main hardware components that should be fault-tolerant include disk drives, disk controllers, CPU, power supplies, and cooling fans.
- Disk drives are the most vulnerable components with the shortest times between failure of any of the hardware components.
- One solution is to provide a large disk array comprising an arrangement of several independent disks (with independent failure modes) that are organized to improve reliability and at the same time increase performance.
- Performance is increased through *data striping*: the data is segmented into equal-size partitions (the *striping unit*), which are transparently distributed across multiple disks.
- Reliability is improved through storing redundant information across the disks using a *parity* scheme or an *error-correcting* scheme.

3.2 Statistical Database Security

- Statistical databases contain information about individuals which may not be permitted to be seen by others as individual records.
- Such databases may contain information about various populations.
- Example: Medical Records, Personal Data like address, salary, etc
- Such kind of databases should have special security mechanisms so that confidential information about people will not be disclosed for many users.
- Only queries with statistical aggregate functions like Average, Sum, Min, Max, Standard Deviation, Mid, Count, etc should be executed.
- Queries retrieving confidential attributes should be prohibited.
- *Not to let the user make inference on the retrieved data, one can also implement constraint on the minimum number of records or tuples in the resulting relation by setting a threshold.*

3.3 DBMSs and Web Security

- Internet communication relies on TCP/IP as the underlying protocol. However, TCP/IP and HTTP were not designed with security in mind. Without special software, all Internet traffic travels ‘in the clear’ and anyone who monitors traffic can read it.

- Performance is increased through *data striping*: the data is segmented into equal-size partitions (the *striping unit*), which are transparently distributed across multiple disks.
- Reliability is improved through storing redundant information across the disks using a *parity* scheme or an *error-correcting* scheme.
- Must ensure while transmitting information over the Internet that:
- inaccessible to anyone but sender and receiver (privacy);
- not changed during transmission (integrity);
- receiver can be sure it came from sender (authenticity);
- sender can be sure receiver is genuine (non-fabrication);
- sender cannot deny he or she sent it (non-repudiation).
- Some Measures include:
    - ✓ Proxy servers
    - ✓ Firewalls
    - ✓ Message digest algorithms and digital signatures
    - ✓ Digital certificates
    - ✓ Kerberos
    - ✓ Secure sockets layer (SSL) and Secure HTTP (S-HTTP)
    - ✓ Secure Electronic Transactions (SET) and Secure Transaction Technology (STT)

3.4 Data Integrity

Accuracy of Data

• Quality of data entered determines the quality of generated information

Data Validation

• Process of ensuring that data entered into the database is valid

• Record validation rules

– Checks all fields before changes to a record are saved

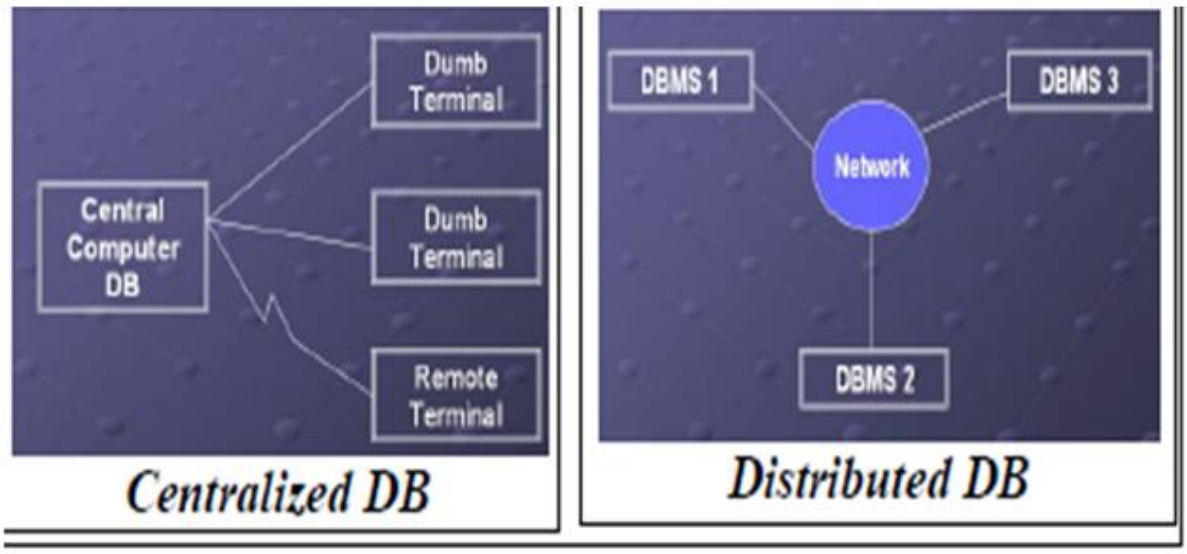
• Can be enforced on a per transaction basis so the entire transaction will fail if one part is invalid

Chapter Four Distributed Database Systems

4.1 Distributed Database Systems

- Database development facilitates the integration of data available in an organization and enforces security on data access.
- But it is not always the case that organizational data reside in one site.
- This demand databases at different sites to be integrated and synchronized with all the facilities of database approach.
- This leads to Distributed Database Systems.

Distributed Database is not a centralized database.



- In a distributed database system, the database is stored on several computers. The computers in a distributed system communicate with each other through various communication media, such as high speed buses or telephone line.
- A distributed database system consists of a collection of sites, each of which maintains a local database system and also participates in global transaction where different databases are integrated together.
- Even though integration of data implies centralized storage and control, in distributed database systems the intention is different. Data is stored in different database systems in a decentralized manner but act as if they are centralized through development of computer networks.

- A distributed database system consists of loosely coupled sites that share no physical component and database systems that run on each site are independent of each other.
- Transactions may access data at one or more sites
- Organization may implement their database system on a number of separate computer system rather than a single, centralized mainframe. Computer Systems may be located at each local branch office.
- The functionalities of a DDBMS will include: Extended Communication Services, Extended Data Dictionary, Distributed Query Processing, Extended Concurrency Control and Extended Recovery Services.
- Concepts in DDBMS
    - Replication: System maintains multiple copies of data, stored in different sites, for faster retrieval and fault tolerance.
    - Fragmentation: Relation is partitioned into several fragments stored in distinct sites
    - Data transparency: Degree to which system user may remain unaware of the details of how and where the data items are stored in a distributed system
- A distributed database system consists of a collection of sites, each of which maintains a local database system and also participates in global transaction where different databases are integrated together.
    - Local Transaction: transactions that access data only in that single site
    - Global Transaction: transactions that access data in several sites.

4.2 Advantages and disadvantages of DDBMS

1. Data sharing and distributed control:

- User at one site may be able access data that is available at another site.
- Each site can retain some degree of control over local data
- We will have local as well as global database administrator

2. Reliability and availability of data

- If one site fails the rest can continue operation as long as transaction does not demand data from the failed system and the data is not replicated in other sites

3. Speedup of query processing

- If a query involves data from several sites, it may be possible to split the query into sub-queries that can be executed at several sites which is parallel processing

Disadvantages of DDBMS

- Software development cost
- Greater potential for bugs (parallel processing may endanger correctness)
- Increased processing overhead (due to communication jargons)
- Communication problems

4.3 Homogeneous and Heterogeneous Distributed Databases

- In a homogeneous distributed database
  - All sites have identical software
  - Are aware of each other and agree to cooperate in processing user requests.
  - Each site surrenders part of its autonomy in terms of right to change schemas or software
  - Appears to user as a single system
- In a heterogeneous distributed database
  - Different sites may use different schemas and software
    - Difference in schema is a major problem for query processing
    - Difference in software is a major problem for transaction processing
  - Sites may not be aware of each other and may provide only limited facilities for cooperation in transaction processing

Chapter Five Object Oriented DBMS

5.1 Object Oriented Concepts

Database systems that were based on the object data model were known originally as object-oriented databases (OODBs) but are now referred to as **object databases** (**ODBs**).Traditional data models and systems, such as network, hierarchical, and relational have been quite successful in developing the database technologies required for many traditional business database applications.

Another reason for the creation of object-oriented databases is the vast increase in the use of object-oriented programming languages for developing software applications. Databases are fundamental components in many software systems, and traditional databases are sometimes difficult to use with software applications that are developed in an object-oriented programming language such as C++ or Java. Object databases are designed so they can be directly—or *seamlessly*—integrated with software that is developed using object-oriented programming languages.

An **object** typically has two components: state (value) and behavior (operations). It can have a *complex data structure* as well as *specific operations* defined by the programmer. Objects in an OOPL exist only during program execution; therefore, they are called *transient objects.* An OO database can extend the existence of objects so that they are stored permanently in a database, and hence the objects become *persistent objects* that exist beyond program termination and can be retrieved later and shared by other programs. In other words, OO databases store persistent objects permanently in secondary storage, and allow the sharing of these objects among multiple programs and applications. This requires the incorporation of other well-known features of database management systems, such as indexing mechanisms to efficiently locate the objects, concurrency control to allow object

Encapsulation: The concept of *encapsulation* is one of the main characteristics of OO languages and systems. It is also related to the concepts of *abstract data types* and *information hiding* in programming languages. In traditional database models and systems this concept was not applied, since it is customary to make the structure of database objects visible to users and external programs. In these traditional models, a number of generic database operations are applicable to objects *of all types*. For example, in the relational model, the operations for selecting, inserting, deleting, and modifying tuples are generic and may be applied to *any relation* in the database. The relation and its attributes are visible to users and to external programs that access the relation by using these operations. The concepts of encapsulation is applied to database objects in ODBs by defining the **behavior** of a type of object based on the **operations** that can be externally applied to objects of that type. Some operations may be used to create (insert) or destroy (delete) objects; other operations may update the object state; and others may be used to
retrieve parts of the object state or to apply some calculations. Still other operations may perform

a combination of retrieval, calculation, and update. In general, the **implementation** of an operation can be specified in a *general-purpose programming language* that provides flexibility and power in defining the operations.

Abstraction:

We now discuss four **abstraction concepts** that are used in semantic data models, such as the EER model, as well as in KR schemes:

1) classification and instantiation,

2) identification,

3) specialization and generalization, and

4) aggregation and association.

1 classification and instantiation

The process of **classification** involves systematically assigning similar objects/entities to object classes/entity types. **Instantiation** is the inverse of classification and refers to the generation and specific examination of distinct objects of a class. An object instance is related to its object class by the
IS-AN-INSTANCE-OF or **IS-A-MEMBER-OF** relationship. Although EER diagrams do not display instances, the UML diagrams allow a form of instantiation by permitting the display of individual objects.

2 identification

Identification is the abstraction process whereby classes and objects are made uniquely identifiable by means of some **identifier**. For example, a class name uniquely identifies a whole class within a schema. An additional mechanism is necessary for telling distinct object instances apart by means of object identifiers. Moreover, it is necessary to identify multiple manifestations in the database of the same real-world object. For example, we may have a tuple <‘Matthew Clarke’, ‘610618’, ‘376-9821’> in a PERSON relation and another tuple <‘301-54-0836’, ‘CS’, 3.8> in a STUDENT relation that happen to represent the same real-world entity. There is no way to identify the fact that these two database objects (tuples) represent the same real-world entity unless we make a provision *at design time* for appropriate cross-referencing to supply this identification. Hence, identification is needed at two levels:
- To distinguish among database objects and classes
- To identify database objects and to relate them to their real-world counterparts

3 specialization and generalization

Specialization is the process of classifying a class of objects into more specialized subclasses. **Generalization** is the inverse process of generalizing several classes into a higher-level abstract class that includes the objects in all these classes. Specialization is conceptual refinement, whereas generalization is conceptual synthesis. Subclasses are used in the EER model to represent specialization and generalization. We call the relationship between a subclass and its superclass an **IS-A-SUBCLASS-OF**
relationship, or simply an **IS-A** relationship.

4 Aggregation and Association

Aggregation is an abstraction concept for building composite objects from their component objects. There are three cases where this concept can be related to the EER model. The first case is the situation in which we aggregate attribute values of an object to form the whole object. The second case is when we represent an aggregation relationship as an ordinary relationship. The third case, which the EER
model does not provide for explicitly, involves the possibility of combining objects that are related by a particular relationship instance into a *higher-level aggregate object.* This is sometimes useful when the higher-level aggregate object is itself to be related to another object. We call the relationship between the primitive objects and their aggregate object **IS-A-PART-OF**; the inverse is called **IS-A-COMPONENT-OF**.UML provides for all three types of aggregation.

Chapter 6: Data warehousing and Data Mining Techniques

6.1 Data Warehousing

- Data warehouse is an integrated, subject-oriented, time-variant, non-volatile database that provides support for decision making.

- Integrated → centralized, consolidated database that integrates data derived from the entire organization.
    - Consolidates data from multiple and diverse sources with diverse formats.
    - Helps managers to better understand the company's operations.

- Subject-Oriented→ Data warehouse contains data organized by topics. Eg. Sales, marketing, finance, etc.
- Time variant: In contrast to the operational data that focus on current transactions, the warehouse data represent the flow of data through time.
    - Data warehouse contains data that reflect what happened last week, last month, past five years, and so on.
- Non volatile→ Once data enter the data warehouse, they are never removed. Because the data in the warehouse represent the company's entire history.

Differences between database and data warehouse

- Because data is added all the time, warehouse is growing.
- The data warehouse and operational environments are separated. Data warehouse receives its data from operational databases.
- Data warehouse environment is characterized by read-only transactions to very large data sets.
- Operational environment is characterized by numerous update transactions to a few data entities at a time.
- Data warehouse contains historical data over a long time horizon.

- Ultimately Information is created from data warehouses. Such Information becomes the basis for rational decision making.
- The data found in data warehouse is analyzed to discover previously unknown data characteristics, relationships, dependencies, or trends.

Data warehouse as *a subject-oriented, integrated, nonvolatile, time-variant collection of data in support of management's decisions*. Data warehouses provide access to data for complex analysis, knowledge discovery, and decision making. They support high-performance demands on an organization's data and information. Several types of applications—OLAP, DSS, and data mining applications—are supported. We define each of these next.
OLAP (online analytical processing) is a term used to describe the analysis of complex data

from the data warehouse. In the hands of skilled knowledge workers, OLAP tools use distributed computing capabilities for analyses that require more storage and processing power than can be economically and efficiently located on an individual desktop.
DSS (decision-support systems), also known as **EIS—executive information systems**; not to be confused with enterprise integration systems—support an organization's leading decision makers with higher-level data for complex and important decisions.

Data mining is used for *knowledge discovery*, the process of searching data for unanticipated new knowledge.
Traditional databases support **online transaction processing (OLTP)**, which includes insertions, updates, and deletions, while also supporting information query requirements. Traditional relational databases are optimized to process queries that may touch a small part of the database and transactions that deal with insertions or updates of a few tuples per relation to process. Thus, they cannot be
optimized for OLAP, DSS, or data mining. By contrast, data warehouses are designed precisely to support efficient extraction, processing, and presentation for analytic and decision-making purposes. In comparison to traditional databases, data warehouses generally contain very large amounts of data from multiple sources that may include databases from different data models and sometimes files acquired
from independent systems and platforms.

Data warehouses have the following distinctive characteristics:

- Multidimensional conceptual view
- Generic dimensionality
- Unlimited dimensions and aggregation levels
- Unrestricted cross-dimensional operations
- Dynamic sparse matrix handling
- Client-server architecture
- Multiuser support
- Accessibility
- Transparency
- Intuitive data manipulation
- Consistent reporting performance
- Flexible reporting

Because they encompass large volumes of data, data warehouses are generally an order of magnitude (sometimes two orders of magnitude) larger than the source databases. The sheer volume of data (likely to be in terabytes or even petabytes) is an issue that has been dealt with through enterprise-wide data warehouses, virtual data warehouses, and data marts:
- **Enterprise-wide data warehouses** are huge projects requiring massive investment of time and resources.
- **Virtual data warehouses** provide views of operational databases that are materialized for

efficient access.
■ **Data marts** generally are targeted to a subset of the organization, such as a department, and are more tightly focused.

6.2 Data Mining

Data mining refers to the mining or discovery of new information in terms of patterns or rules from vast amounts of data. To be practically useful, data mining must be carried out efficiently on large files and databases. Although some data mining features are being provided in RDBMSs, data mining is *not* well-integrated with database management systems.

Data mining is typically carried out with some end goals or applications. Broadly speaking, these goals fall into the following classes: prediction, identification, classification, and optimization.
■ **Prediction.** Data mining can show how certain attributes within the data will behave in the future. Examples of predictive data mining include the analysis of buying transactions to predict what consumers will buy under certain discounts, how much sales volume a store will generate in a given period, and whether deleting a product line will yield more profits. In such applications, business logic is used coupled with data mining. In a scientific context, certain seismic wave patterns may predict an earthquake with high probability.

■ **Identification.** Data patterns can be used to identify the existence of an item, an event, or an activity. For example, intruders trying to break a system may be identified by the programs executed, files accessed, and CPU time per session. In biological applications, existence of a gene may be identified by certain sequences of nucleotide symbols in the DNA sequence. The area known as *authentication* is a form of identification. It ascertains whether a user is indeed a specific user or one from an authorized class, and involves a comparison of parameters or images or signals against a database.
■ **Classification.** Data mining can partition the data so that different classes or categories can be identified based on combinations of parameters. For example, customers in a supermarket can be categorized into discount seeking shoppers, shoppers in a rush, loyal regular shoppers, shoppers attached to name brands, and infrequent shoppers. This classification may be used in different analyses of customer buying transactions as a post mining activity. Sometimes classification based on common domain knowledge is used as an input to decompose the mining problem and make it simpler. For instance, health foods, party foods, or school lunch foods are distinct categories in the supermarket business. It makes sense to analyze relationships within and across categories as separate problems. Such categorization may be used to encode the data appropriately before subjecting it to further data mining.
■ **Optimization.** One eventual goal of data mining may be to optimize the use of limited resources such as time, space, money, or materials and to maximize output variables such as sales or profits under a given set of constraints.
As such, this goal of data mining resembles the objective function used in operations research problems that deals with optimization under constraints.