Unit – III

Transactions

Transaction Concepts - ACID Properties - Schedules - Serializability - Concurrency Control - Need for Concurrency - Locking Protocols - Two Phase Locking - Deadlock - Transaction Recovery - Save Points - Isolation Levels - SQL Facilities for Concurrency and Recovery.

Introduction

- **Single-User System:** Only one user can use the system at a time.
- **Multiuser System:** Many users can access the system concurrently.

Concurrency

- **Interleaved Processing:** Concurrent execution of processes is interleaved in a single CPU.
- **Parallel Processing:** Processes are concurrently executed in multiple CPUs.

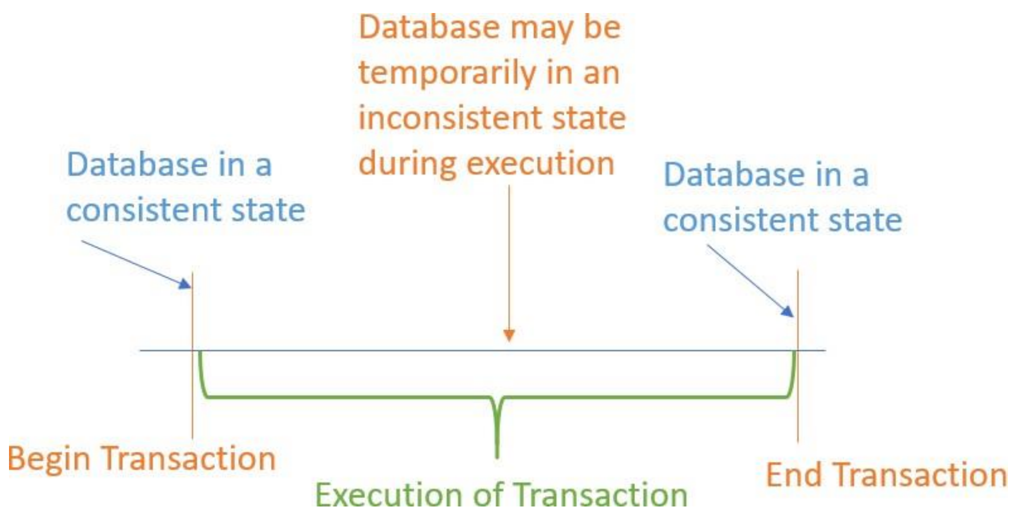
Transaction Concepts

Write short notes on transaction concepts. (Nov/Dec 2014)

- A **transaction** can be defined as a group of tasks. It is a logical unit of work on the database processing that includes one or more access operations (read - retrieval, write - insert or update, delete).

(Or)

- Collections of operations that form a single logical unit of work are called **transactions**.
- A database system must ensure proper execution of transactions despite failures—either the entire transaction executes, or none of it does.
- Usually, a transaction is initiated by a user program written in a high-level data- manipulation language (typically SQL), or programming language (for example, C++, or Java), with embedded database accesses in JDBC or ODBC.
- A transaction is delimited by statements (or function calls) of the form **begin transaction** and **end transaction**.
- The transaction consists of all operations executed between the **begin transaction** and **end transaction**.


Database may be temporarily in an inconsistent state during execution
Database in a consistent state
Database in a consistent state
Begin Transaction
Execution of Transaction
End Transaction

<u>A Simple Transaction Model / Simple Model of a Database</u> (for purposes of transactions):

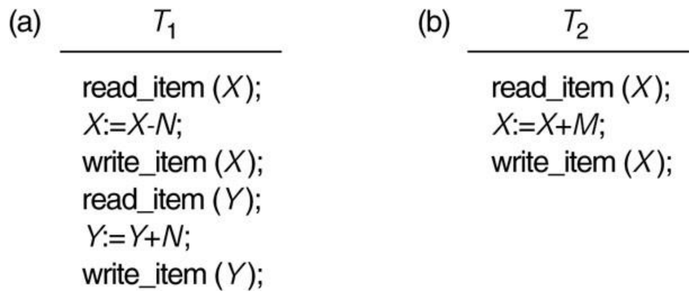
- A database - collection of named data items
- Granularity of data - a field, a record or a whole disk block (Concepts are independent of granularity)
- Basic operations are read and write.
    - read_item(X): Reads a database item named X into a program variable. To simplify our notation, we assume that *the program variable is also named X.*
    - write_item(X): Writes the value of program variable X into the database item named X.

Read Operation:

- Basic unit of data transfer from the disk to the computer main memory is one block.
- In general, a data item (what is read or written) will be the field of some record in the database, although it may be a larger unit such as a record or even a whole block.
- read_item(X) command includes the following steps:
    - ✓ 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 buffer to the program variable named X.

Write Operation:

- write_item(X) command includes the following steps:
    - ✓ 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 in time).



```
(a)   T1                    (b)   T2

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

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

- Let *Ti* be a transaction that transfers \$50 from account *A* to account *B*. This transaction can be defined as:

  Ti : read(*A*);

  $A := A - 50;$

  write(*A*);

  read(*B*);

B:=*B*+50;

write(*B*).

Storage Structure

- To understand how to ensure the atomicity and durability properties of a transaction, we must gain a better understanding of how the various data items in the database may be stored and accessed.

Volatile Storage

- Information residing in volatile storage does not usually survive system crashes.
- Examples of such storage are main memory and cache memory.

Nonvolatile Storage

- Information residing in nonvolatile storage survives system crashes.
- Examples of nonvolatile storage include secondary storage devices such as magnetic disk and flash storage, used for online storage, and tertiary storage devices such as optical media, and magnetic tapes, used for archival storage.

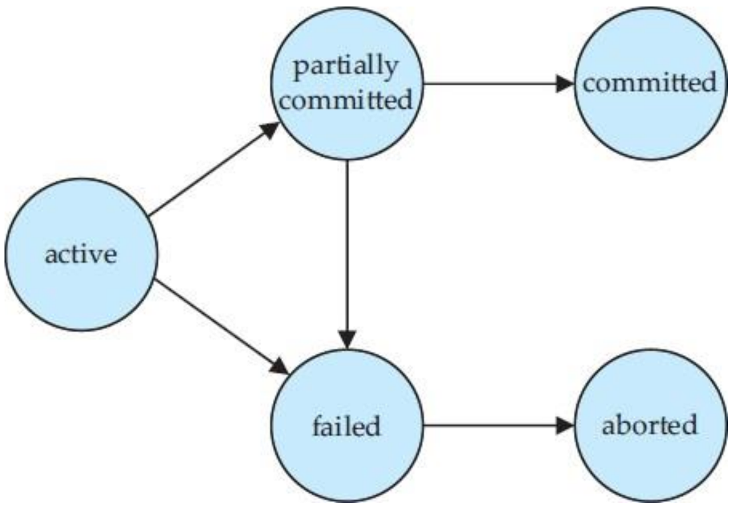
Stable Storage

- Information residing in stable storage is *never* lost (*never* should be taken with a grain of salt, since theoretically *never* cannot be guaranteed).

State Diagram of a Transaction

Write short notes on states of a transaction.

- A transaction in a database can be in one of the following states:
  - ✓ Active
  - ✓ Partially Committed
  - ✓ Failed
  - ✓ Aborted
  - ✓ Committed



(Or)

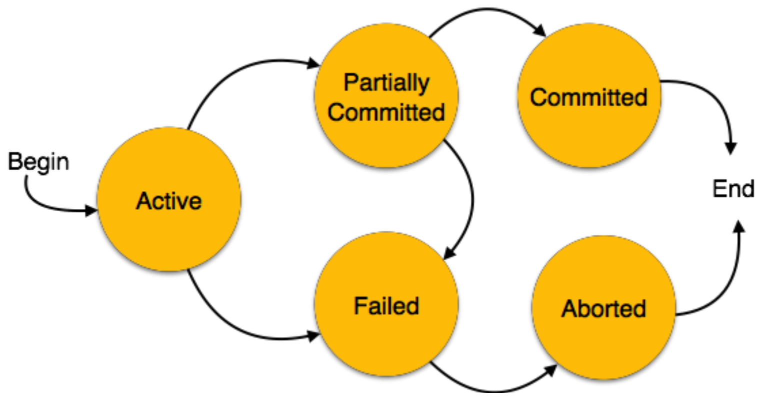


Figure: State Diagram of a Transaction

Active State

- The active state is the first state of every transaction.
- In this state, the transaction is being executed.
- For example: Insertion or deletion or updating a record is done here. But all the records are still not saved to the database.

Partially Committed

- In the partially committed state, a transaction executes its final operation, but the data is still not saved to the database.
- In the total mark calculation example, a final display of the total marks step is executed in this state.

Committed

- A transaction is said to be in a committed state if it executes all its operations successfully.
- In this state, all the effects are now permanently saved on the database system.

Failed State

- If any of the checks made by the database recovery system fails, then the transaction is said to be in the failed state.
- In the example of total mark calculation, if the database is not able to fire a query to fetch the marks, then the transaction will fail to execute.

Aborted

- If any of the checks fail and the transaction has reached a failed state then the database recovery system will make sure that the database is in its previous consistent state.
- If not then it will abort or roll back the transaction to bring the database into a consistent state.
- If the transaction fails in the middle of the transaction then before executing the transaction, all the executed transactions are rolled back to its consistent state.
- After aborting the transaction, the database recovery module will select one of the two operations:
  - ✓ Re-start the Transaction
  - ✓ Kill the Transaction

ACID Properties

Explain with an example the properties that must be satisfied by a transaction. (April/May 2018)

- The transaction has the four properties. These are used to maintain consistency in a database, before and after the transaction.

Properties of Transaction

1. Atomicity
2. Consistency
3. Isolation
4. Durability

Atomicity

- It states that all operations of the transaction take place at once if not, the transaction is aborted.
- There is no midway, i.e., the transaction cannot occur partially. Each transaction is treated as one unit and either run to completion or is not executed at all.
- Atomicity involves the following two operations:
  - ✓ **Abort:** If a transaction aborts then all the changes made are not visible.
  - ✓ **Commit:** If a transaction commits then all the changes made are visible.

Example:

- Let's assume that following transaction T consisting of T1 and T2. A consists of Rs 600 and B consists of Rs 300. Transfer Rs 100 from account A to account B.

T1	T2

Read(A)
A:= A-100
Write(A)
	Read(B)
	Y:= Y+100
	Write(B)

- After completion of the transaction, A consists of Rs 500 and B consists of Rs 400.
- If the transaction T fails after the completion of transaction T1 but before completion of transaction T2, then the amount will be deducted from A but not added to B.
- This shows the inconsistent database state.
- In order to ensure correctness of database state, the transaction must be executed in entirety.

Consistency

- The integrity constraints are maintained so that the database is consistent before and after the transaction.
- The execution of a transaction will leave a database in either its prior stable state or a new stable state.
- The consistent property of database states that every transaction sees a consistent database instance.
- The transaction is used to transform the database from one consistent state to another consistent state.

- For example: The total amount must be maintained before or after the transaction.
   1. **Total before T occurs = 600 + 300 = 900**
   2. **Total after T occurs = 500 + 400 = 900**
- Therefore, the database is consistent. In the case when T1 is completed but T2 fails, then inconsistency will occur.

Isolation

- It shows that the data which is used at the time of execution of a transaction cannot be used by the second transaction until the first one is completed.
- In isolation, if the transaction T1 is being executed and using the data item X, then that data item can't be accessed by any other transaction T2 until the transaction T1 ends.
- The concurrency control subsystem of the DBMS enforced the isolation property.

Durability

- The durability property is used to indicate the performance of the database's consistent state. It states that the transaction made the permanent changes.
- They cannot be lost by the erroneous operation of a faulty transaction or by the system failure.
- When a transaction is completed, then the database reaches a state known as the consistent state. That consistent state cannot be lost, even in the event of a system's failure.
- The recovery subsystem of the DBMS has the responsibility of Durability property.

Schedules

Explain in detail about schedules with an example.

- A series of operation from one transaction to another transaction is known as schedule.
- It is used to preserve the order of the operation in each of the individual transaction.
- If two transactions are executed at the same time, the result of one transaction may affect the output of other.

Example

Initial Product Quantity is 10
Transaction 1: Update Product Quantity to 50
Transaction 2: Read Product Quantity

- If Transaction 2 is executed before Transaction 1, outdated information about the product quantity will be read. Hence, schedules are required.

Equivalence Schedules

- Parallel execution in a database is inevitable.
- But, Parallel execution is permitted when there is an equivalence relation amongst the simultaneously executing transactions.
- This equivalence is of 3 Types.

Result Equivalence

- If two schedules display the same result after execution, it is called result equivalent schedule.

- They may offer the same result for some value and different results for another set of values.
- For example, one transaction updates the product quantity, while other updates customer details.

View Equivalence

- View Equivalence occurs when the transaction in both the schedule performs a similar action.
- For example, one transaction inserts product details in the product table, while another transaction inserts product details in the archive table.
- The transaction is the same, but the tables are different.

Conflict Equivalence

- In this case, two transactions update/view the same set of data.
- There is a conflict amongst transaction as the order of execution will affect the output.

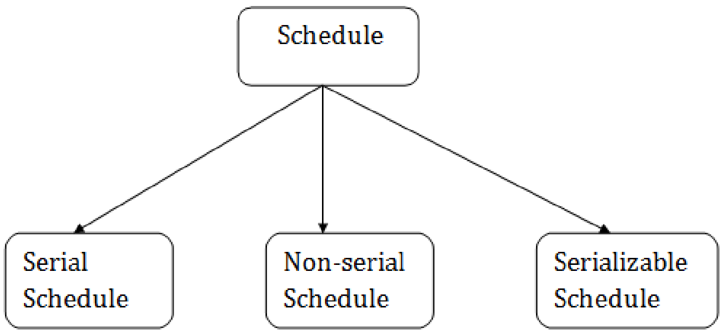
Two schedules would be conflicting if they have the following properties −

- Both belong to separate transactions.
- Both access the same data item.
- At least one of them is "write" operation.

Two schedules having multiple transactions with conflicting operations are said to be conflict equivalent if and only if −

- Both the schedules contain the same set of Transactions.
- The order of conflicting pairs of operation is maintained in both the schedules.

Types of Schedule

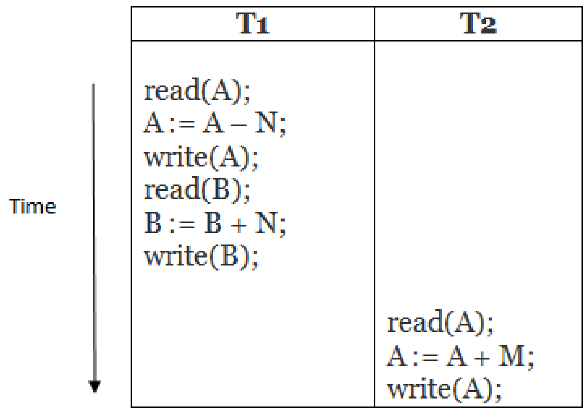


Serial Schedule

- The serial schedule is a type of schedule where one transaction is executed completely before starting another transaction.
- In the serial schedule, when the first transaction completes its cycle, then the next transaction is executed.
- For example: Suppose there are two transactions T1 and T2 which have some operations.
- If it has no interleaving of operations, then there are the following two possible outcomes:
  - ✓ Execute all the operations of T1 which was followed by all the operations of T2.
  - ✓ Execute all the operations of T1 which was followed by all the operations of T2.
- In the given (a) figure, Schedule A shows the serial schedule where T1 followed by T2.

- In the given (b) figure, Schedule B shows the serial schedule where T2 followed by T1.

(a)



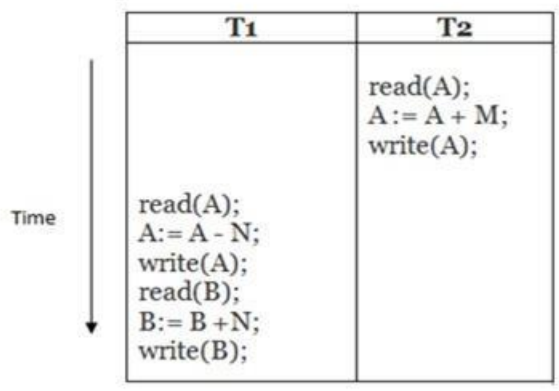
T1	T2

read(A);
A := A – N;
write(A);
read(B);
B := B + N;
write(B);	

	read(A);
	A := A + M;
	write(A);

Schedule A

(b)



T1	T2

	read(A);
	A:= A + M;
	write(A);

read(A);
A:= A - N;
write(A);
read(B);
B:= B +N;
write(B);	

Schedule B

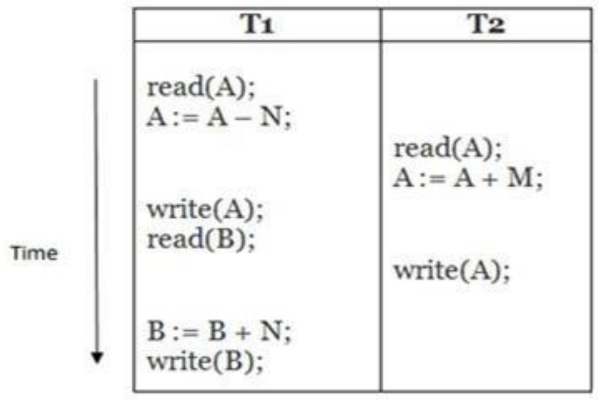
Here,

- Schedule A and Schedule B are serial schedule.

Non-serial Schedule

- If interleaving of operations is allowed, then there will be non-serial schedule.
- It contains many possible orders in which the system can execute the individual operations of the transactions.
- In the given figure (c) and (d), Schedule C and Schedule D are the non-serial schedules.
- It has interleaving of operations.

(c)



T1	T2

read(A);
A := A – N;	

	read(A);
	A := A + M;

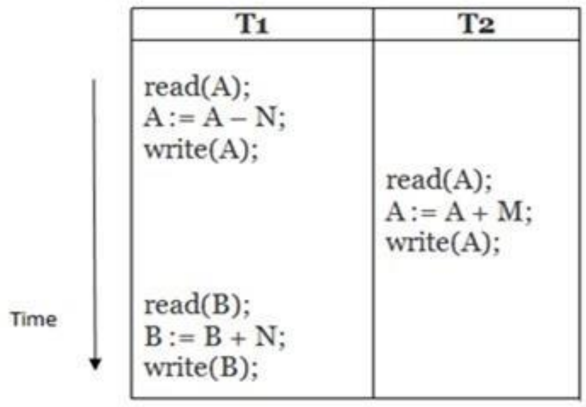
write(A);
read(B);	

	write(A);

B := B + N;
write(B);	

Schedule C

(d)



T1	T2

read(A);
A := A – N;
write(A);	

	read(A);
	A := A + M;
	write(A);

read(B);
B := B + N;
write(B);	

Schedule D

Here,

- Schedule C and Schedule D are Non-serial schedule.

Serializable Schedule

- The serializability of schedules is used to find non-serial schedules that allow the transaction to execute concurrently without interfering with one another.
- It identifies which schedules are correct when executions of the transaction have interleaving of their operations.
- A non-serial schedule will be serializable if its result is equal to the result of its transactions executed serially.

Serializability

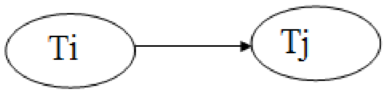
Explain Serializability in detail. (Or) Discuss View Serializability and Conflict Serializability. (Nov/Dec 2015, April/May 2018)

- **Serializability** is a **concurrency scheme** where the concurrent transaction is equivalent to one that executes the transactions serially.
- A schedule is a list of transactions.
- The objective of a **concurrency control** protocol is to schedule transactions in such a way as to avoid any interference between them.
- **Schedule** is a sequence of the operations by a set of concurrent transactions that preserves the order of the operations in each of the individual transactions.
- **Serial schedule** is a schedule where the operations of each transaction are executed consecutively without any interleaved operations from other transactions.
- In a serial schedule, the transactions are performed in serial order, ie., if T1 and T2 are transactions, serial order would be T1 followed by T2 or T2 followed by T1.
- **Non serial schedule** is a schedule where the operations from a set of concurrent transactions are interleaved.
- In non-serial schedule, if the schedule is not proper, then the problems can arise like multiple update, uncommitted dependency and incorrect analysis.
- The **objective of serializability** is to find non serial schedules that allow transactions to execute concurrently without interfering with one another, and there by produce a database state that could be produced by a serial execution.

Testing of Serializability

- Serialization Graph is used to test the Serializability of a schedule.
- Assume a schedule S. For S, we construct a graph known as precedence graph.
- This graph has a pair G = (V, E), where V consists a set of vertices, and E consists a set of edges. The set of vertices is used to contain all the transactions participating in the schedule.
- The set of edges is used to contain all edges Ti ->Tj for which one of the three conditions holds:
  1. Create a node Ti → Tj if Ti executes write (Q) before Tj executes read (Q).
  2. Create a node Ti → Tj if Ti executes read (Q) before Tj executes write (Q).
  3. Create a node Ti → Tj if Ti executes write (Q) before Tj executes write (Q).

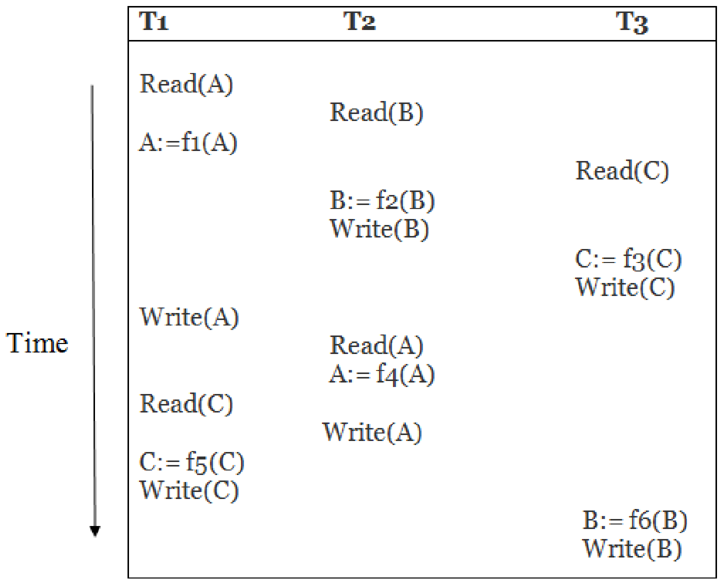
Precedence Graph for Schedule S:



- If a precedence graph contains a single edge Ti → Tj, then all the instructions of Ti are executed before the first instruction of Tj is executed.

- If a precedence graph for schedule S contains a cycle, then S is non-serializable. If the precedence graph has no cycle, then S is known as serializable.

For Example:



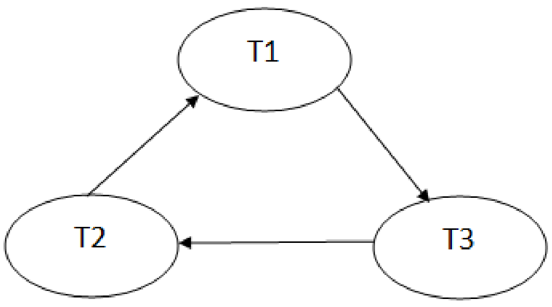
T1	T2	T3
Read(A)		
	Read(B)	
A:=f1(A)		
		Read(C)
	B:= f2(B) Write(B)	
		C:= f3(C) Write(C)
Write(A)		
	Read(A) A:= f4(A)	
Read(C)		
	Write(A)	
C:= f5(C) Write(C)		
		B:= f6(B) Write(B)

Schedule S1

Explanation:

- **Read(A):** In T1, no subsequent writes to A, so no new edges
- **Read(B):** In T2, no subsequent writes to B, so no new edges
- **Read(C):** In T3, no subsequent writes to C, so no new edges
- **Write(B):** B is subsequently read by T3, so add edge T2→T3
- **Write(C):** C is subsequently read by T1, so add edge T3→T1
- **Write(A):** A is subsequently read by T2, so add edge T1→T2
- **Write(A):** In T2, no subsequent reads to A, so no new edges
- **Write(C):** In T1, no subsequent reads to C, so no new edges
- **Write(B):** In T3, no subsequent reads to B, so no new edges

Precedence Graph for Schedule S1:



- The precedence graph for schedule S1 contains a cycle that's why Schedule S1 is non- serializable.

Time

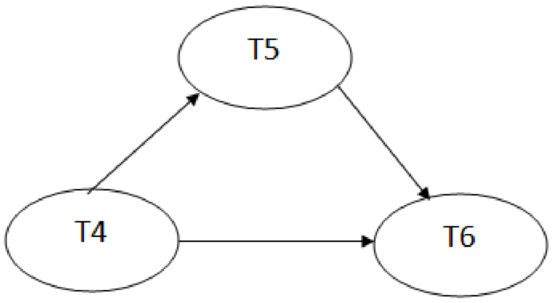
T4	T5	T6
Read(A)		
A:= f1(A)		
Read(C)		
Write(A)		
A:= f2(C)		
	Read(B)	
Write(C)		
	Read(A)	
		Read(C)
	B:= f3(B)	
	Write(B)	
		C:= f4(C)
		Read(B)
		Write(C)
	A:=f5(A)	
	Write(A)	
		B:= f6(B)
		Write(B)

Schedule S2

Explanation:

- **Read(A):** In T4, no subsequent writes to A, so no new edges
- **Read(C):** In T4, no subsequent writes to C, so no new edges
- **Write(A):** A is subsequently read by T5, so add edge T4 → T5
- **Read(B):** In T5,no subsequent writes to B, so no new edges
- **Write(C):** C is subsequently read by T6, so add edge T4→T6
- **Write(B):** A is subsequently read by T6, so add edge T5→T6
- **Write(C):** In T6, no subsequent reads to C, so no new edges
- **Write(A):** In T5, no subsequent reads to A, so no new edges
- **Write(B):** In T6, no subsequent reads to B, so no new edges

Precedence graph for schedule S2:



- The precedence graph for schedule S2 contains no cycle that's why ScheduleS2 is serializable.

Types of Serializability

1. Conflict Serializability

- Conflict serializability defines two instructions of two different transactions accessing the same data item to perform a read/write operation.
- It deals with detecting the instructions that are conflicting in any way and specifying the order in which the instructions should execute in case there is any conflict.

- A conflict serializability arises when one of the instruction is a write operation. The following rules

are important in Conflict Serializability,

- If two transactions are both read operation, then they are not in conflict.
- If one transaction wants to perform a read operation and other transaction wants to perform a write operation, then they are in conflict and cannot be swapped.
- If both the transactions are for write operation, then they are in conflict, but can be allowed to take place in any order, because the transactions do not read the value updated by each other.

2. View Serializability

- View serializability is another type of serializability.
- It can be derived by creating another schedule out of an existing schedule and involves the same set of transactions**.**

Example:

- Let us assume two transactions T1 and T2 that are being serialized to create two different schedules S1 and S2, where T1 and T2 want to access the same data item.
- Now there can be three scenarios,
  - ✓ If in S1, T1 reads the initial value of data item, then in S2, T1 should read the initial value of that same data item.
  - ✓ If in S2, T1 writes a value in the data item which is read by T2, and then in S2, T1 should write the value in the data item before T2 reads it.
- If in S1, T1 performs the final write operation on that data item, then in S2, T1 should perform the final write operation on that data item.
- If a concurrent schedule is view equivalent to a serial schedule of same transaction then it is said to be View serializable.

Conflict Serializable Schedule

- A schedule is called conflict serializability if after swapping of non-conflicting operations, it can transform into a serial schedule.
- The schedule will be a conflict serializable if it is conflict equivalent to a serial schedule.

Conflicting Operations

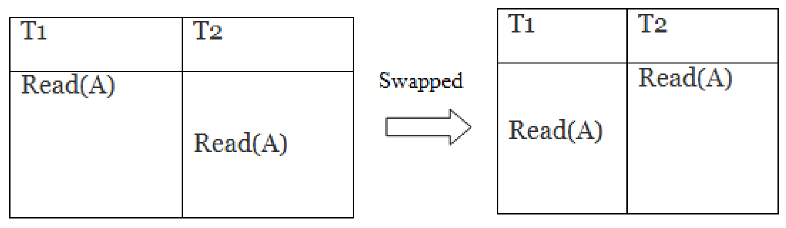
The two operations become conflicting if all conditions satisfy:

1. Both belong to separate transactions.
2. They have the same data item.
3. They contain at least one write operation.

Example:

- Swapping is possible only if S1 and S2 are logically equal.

1. T1: Read(A) T2: Read(A)



T1	T2
Read(A)	
	Read(A)

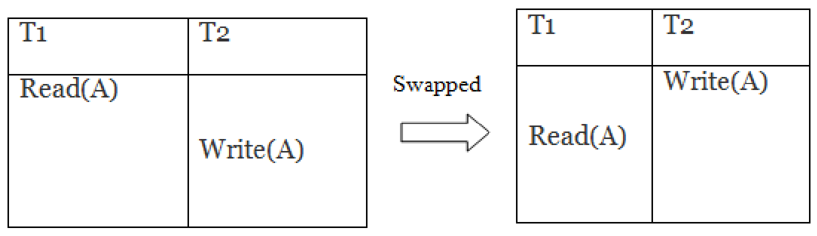
Schedule S1

T1	T2
	Read(A)
Read(A)	

Schedule S2

- Here, S1 = S2. That means it is non-conflict.

2. T1: Read(A) T2: Write(A)



T1	T2
Read(A)	
	Write(A)

Schedule S1

T1	T2
	Write(A)
Read(A)	

Schedule S2

- Here, S1 ≠ S2. That means it is conflict.

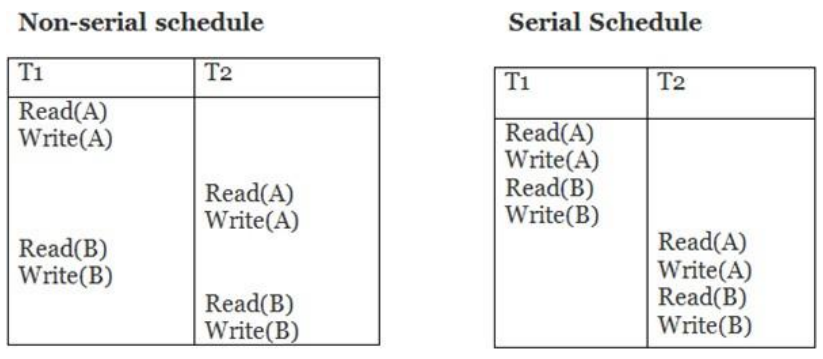
Conflict Equivalent

- In the conflict equivalent, one can be transformed to another by swapping non-conflicting operations.
- In the given example, S2 is conflict equivalent to S1 (S1 can be converted to S2 by swapping non-conflicting operations).

Two schedules are said to be conflict equivalent if and only if:

1. They contain the same set of the transaction.
2. If each pair of conflict operations are ordered in the same way.

Example:



Non-serial schedule

T1	T2

Read(A)
Write(A)	

	Read(A)
	Write(A)

Read(B)
Write(B)	

	Read(B)
	Write(B)

Schedule S1

Serial Schedule

T1	T2

Read(A)
Write(A)
Read(B)
Write(B)	

	Read(A)
	Write(A)
	Read(B)
	Write(B)

Schedule S2

- Schedule S2 is a serial schedule because, in this, all operations of T1 are performed before starting any operation of T2.
- Schedule S1 can be transformed into a serial schedule by swapping non-conflicting operations of S1.
- After swapping of non-conflict operations, the schedule S1 becomes:

T1	**T2**
Read(A)	
Write(A)	
Read(B)	
Write(B)	
	Read(A)
	Write(A)
	Read(B)
	Write(B)

- Since, S1 is conflict serializable.

View Serializability

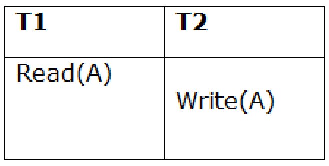
- A schedule will view serializable if it is view equivalent to a serial schedule.
- If a schedule is conflict serializable, then it will be view serializable.
- The view serializable which does not conflict serializable contains blind writes.

View Equivalent

- Two schedules S1 and S2 are said to be view equivalent if they satisfy the following conditions:

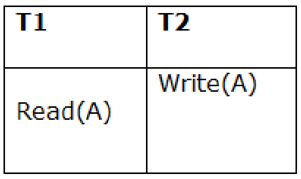
1. Initial Read

- An initial read of both schedules must be the same.
- Suppose two schedule S1 and S2.
- In schedule S1, if a transaction T1 is reading the data item A, then in S2, transaction T1 should also read A.



T1	T2
Read(A)	
	Write(A)

Schedule S1



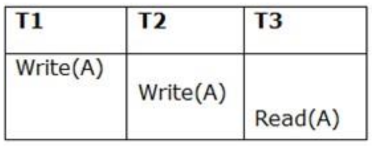
T1	T2
	Write(A)
Read(A)	

Schedule S2

- Above two schedules are view equivalent because Initial read operation in S1 is done by T1 and in S2 it is also done by T1.

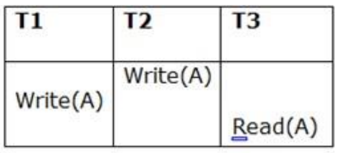
2. Updated Read

- In schedule S1, if Ti is reading A which is updated by Tj then in S2 also, Ti should read A which is updated by Tj.



T1	T2	T3
Write(A)		
	Write(A)	
		Read(A)

Schedule S1



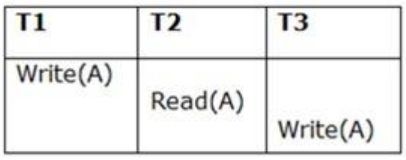
T1	T2	T3
	Write(A)	
Write(A)		
		Read(A)

Schedule S2

- Abovetwoschedulesarenotviewequalbecause,inS1,T3isreadingAupdatedbyT2and in S2, T3 is reading A updated by T1.

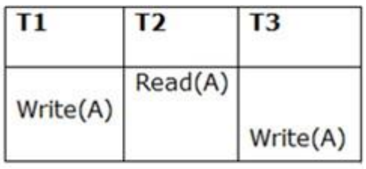
3. Final Write

- A final write must be the same between both the schedules. In schedule S1, if a transaction T1 updates A at last then in S2, final writes operations should also be done by T1.



T1	T2	T3
Write(A)		
	Read(A)	
		Write(A)

Schedule S1

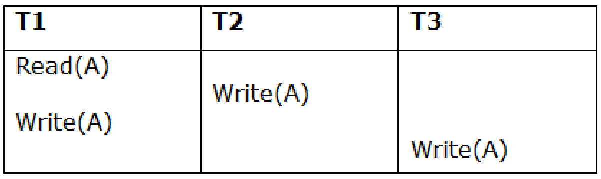


T1	T2	T3
	Read(A)	
Write(A)		
		Write(A)

Schedule S2

- AbovetwoschedulesisviewequalbecauseFinalwriteoperationinS1isdonebyT3andin S2, the final write operation is also done by T3.

Example:



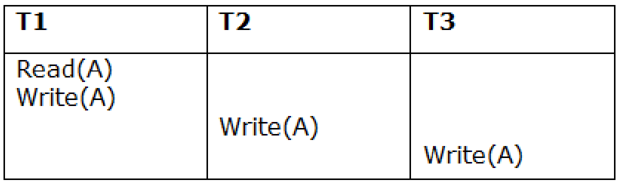
T1	T2	T3
Read(A)		
	Write(A)	
Write(A)		
		Write(A)

Schedule S

- With3transactions,thetotalnumberofpossibleschedule 1. = 3! = 6

2. S1=<T1T2T3>
3. S2=<T1T3T2>
4. S3=<T2T3T1>
5. S4=<T2T1T3>
6. S5=<T3T1T2>
7. S6=<T3T2T1>

Taking first schedule S1:



T1	T2	T3

Read(A)
Write(A)		

	Write(A)	
		Write(A)

Schedule S1

Step 1: Final Updation on Data Items

- In both schedules S and S1, there is no read except the initial read that's why we don't need to check that condition.

Step 2: Initial Read

- The initial read operation in S is done by T1 and in S1, it is also done by T1.

Step 3: Final Write

- The final write operation in S is done by T3 and in S1, it is also done by T3. So, S and S1 are view Equivalent.
- The first schedule S1 satisfies all three conditions, so we don't need to check another schedule.
- Hence, view equivalent serial schedule is:

T1 → T2 → T3

Recoverability of Schedule

- Sometimes a transaction may not execute completely due to a software issue, system crash or hardware failure.
- In that case, the failed transaction has to be rollback.
- But some other transaction may also have used value produced by the failed transaction.
- So we also have to rollback those transactions.

T1	**T1's buffer space**	**T2**	**T2's buffer space**	**Database**
				A = 6500
Read(A);	A = 6500			A = 6500
A = A - 500;	A = 6000			A = 6500
Write(A);	A = 6000			A = 6000
		Read(A);	A = 6000	A = 6000
		A =A + 1000;	A = 7000	A = 6000
		Write(A);	A = 7000	A = 7000
		Commit;		
Failure Point				
Commit;				

- The above table 1 shows a schedule which has two transactions.
- T1 reads and writes the value of A and that value is read and written by T2.
- T2 commits but later on, T1 fails. Due to the failure, we have to rollback T1.
- T2 should also be rollback because it reads the value written by T1, but T2 can't be rollback because it already committed.
- So this type of schedule is known as irrecoverable schedule.

Irrecoverable Schedule

- The schedule will be irrecoverable if Tj reads the updated value of Ti and Tj committed before Ti commit.

T1	**T1's buffer space**	**T2**	**T2's buffer space**	**Database**
				A = 6500
Read(A);	A = 6500			A = 6500
A = A - 500;	A = 6000			A = 6500
Write(A);	A = 6000			A = 6000
		Read(A);	A = 6000	A = 6000
		A =A + 1000;	A = 7000	A = 6000
		Write(A);	A = 7000	A = 7000
Failure Point				
Commit;				
		Commit;		

- The above table 2 shows a schedule with two transactions.
- Transaction T1 reads and writes A, and that value is read and written by transaction T2.
- But later on, T1 fails. Due to this, we have to rollback T1.
- T2 should be rollback because T2 has read the value written by T1.
- As it has not committed before T1 commits so we can rollback transaction T2 as well.
- So it is recoverable with cascade rollback.

Recoverable with Cascading Rollback

- The schedule will be recoverable with cascading rollback if Tj reads the updated value of Ti. Commit of Tj is delayed till commit of Ti.

T1	**T1's buffer space**	**T2**	**T2's buffer space**	**Database**
				A = 6500
Read(A);	A = 6500			A = 6500
A = A - 500;	A = 6000			A = 6500
Write(A);	A = 6000			A = 6000
Commit;		Read(A);	A = 6000	A = 6000
		A =A + 1000;	A = 7000	A = 6000
		Write(A);	A = 7000	A = 7000
		Commit;		

- The above Table 3 shows a schedule with two transactions.
- Transaction T1 reads and write A and commits, and that value is read and written by T2.
- So this is a cascade less recoverable schedule.

<u>Concurrency Control</u>

What is Concurrency? Explain it in terms of locking mechanism and two phase commit protocols. (Nov/Dec 2014) (Or) What is Concurrency Control? How is it implemented in DBMS? Illustrate with a suitable example. (Nov/Dec 2015) (Or) State and explain the lock based concurrency control with suitable example. (Nov/Dec 2017)

- Concurrency control manages the transactions simultaneously without letting them interfere with each another.
- The **main objective of concurrency** is to **allow multiple users to perform** different operations at the same time.

(Or)

- A mechanism which ensures that simultaneous execution of more than one transaction does not lead to any database inconsistencies is called **concurrency control mechanism**.
- Using more than one transaction concurrently improves the performance of system.

- If we are not able to perform the operations concurrently, then there can be serious problems such as loss of data integrity and consistency.
- Concurrency control increases the throughput because of handling multiple transactions simultaneously.
- It reduces waiting time of transaction.

Purpose of Concurrency Control

- The fundamental goal of database concurrency control is to ensure that concurrent execution of transactions does not result in a loss of database consistency.
- The concept of serializability can be used to achieve this goal, since all serializable schedules preserve consistency of the database.
- To enforce Isolation (through mutual exclusion) among conflicting transactions.
- To preserve database consistency through consistency preserving execution of transactions.
- To resolve read-write and write-write conflicts.

Example:

- In concurrent execution environment if T1 conflicts with T2 over a data item A, then the existing concurrency control decides if T1 or T2 should get the A and if the other transaction is rolled-back or waits.
- Different types of protocols/schemes used to control concurrent execution of transactions are:
  - ✓ Lock based Protocols
  - ✓ Timestamp based Protocols
  - ✓ Graph based Protocols

<u>Need for Concurrency</u>

Explain why Concurrency Control is needed?

- The purposes of concurrency control are,
  - ✓ To ensure isolation
  - ✓ To resolve read-write or write-write conflicts
  - ✓ To preserve consistency of database

The Lost Update Problem

- This occurs when two transactions that access the same database items have their operations interleaved in a way that makes the value of some database item incorrect.

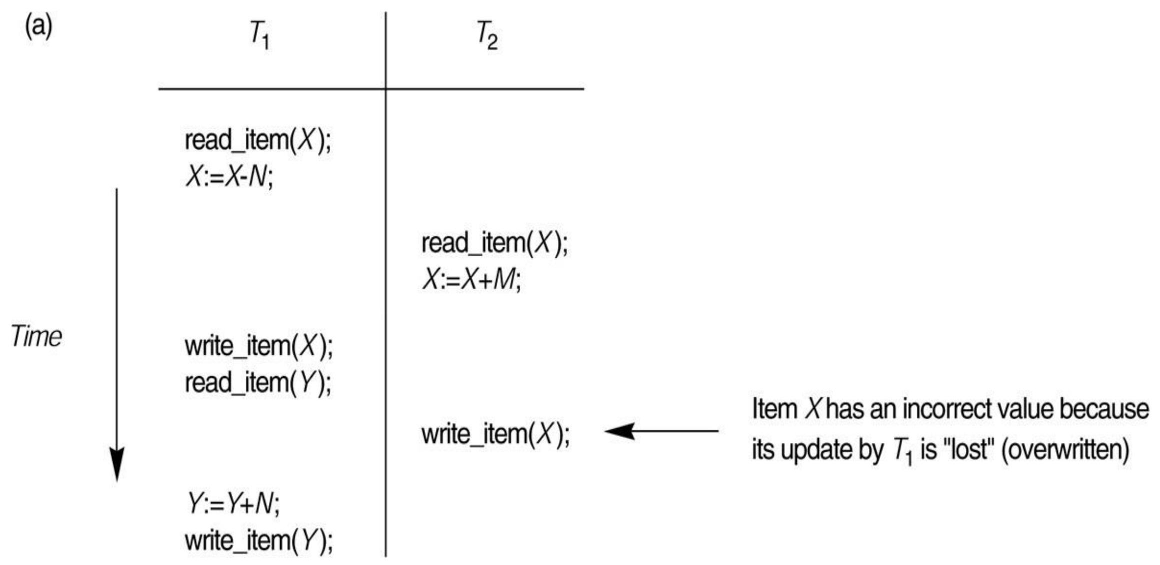
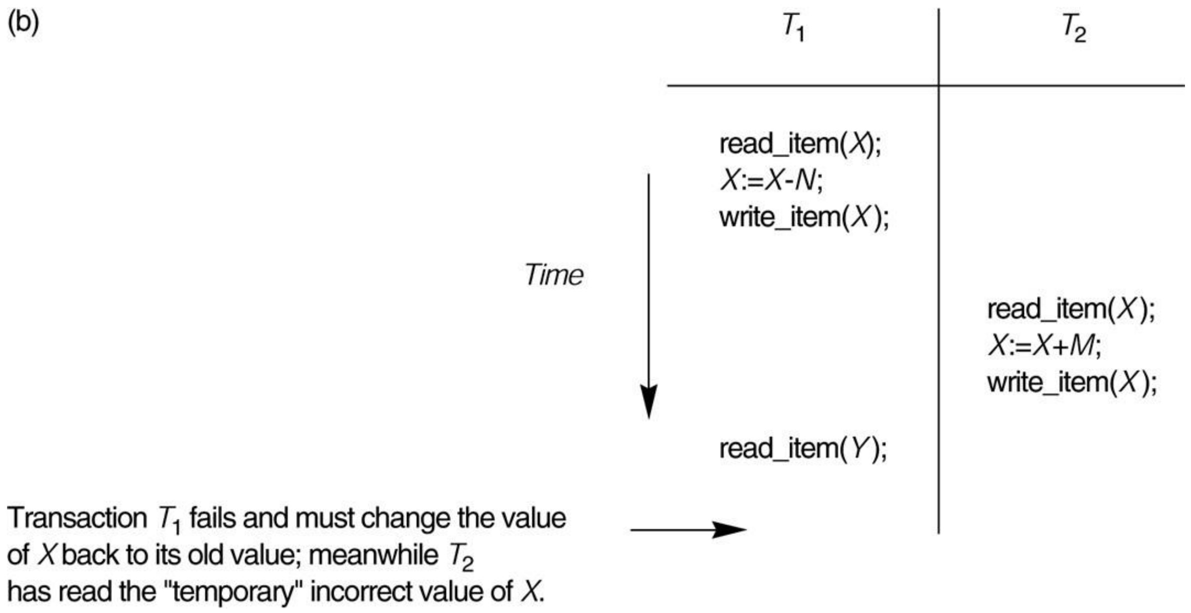


Figure: (a) The Lost Update Problem

The Temporary Update (or Dirty Read) Problem

- This occurs when one transaction updates a database item and then the transaction fails for some reason.
- The updated item is accessed by another transaction before it is changed back to its original value.

Figure: (b) The Temporary Update Problem


(b)
T_1
T_2
read_item(X);
X:=X-N;
write_item(X);
Time
read_item(X);
X:=X+M;
write_item(X);
read_item(Y);
Transaction T_1 fails and must change the value of X back to its old value; meanwhile T_2 has read the "temporary" incorrect value of X.

The Incorrect Summary Problem

- If one transaction is calculating an aggregate summary function on a number of records while other transactions are updating some of these records, the aggregate function may calculate some values before they are updated and others after they are updated.

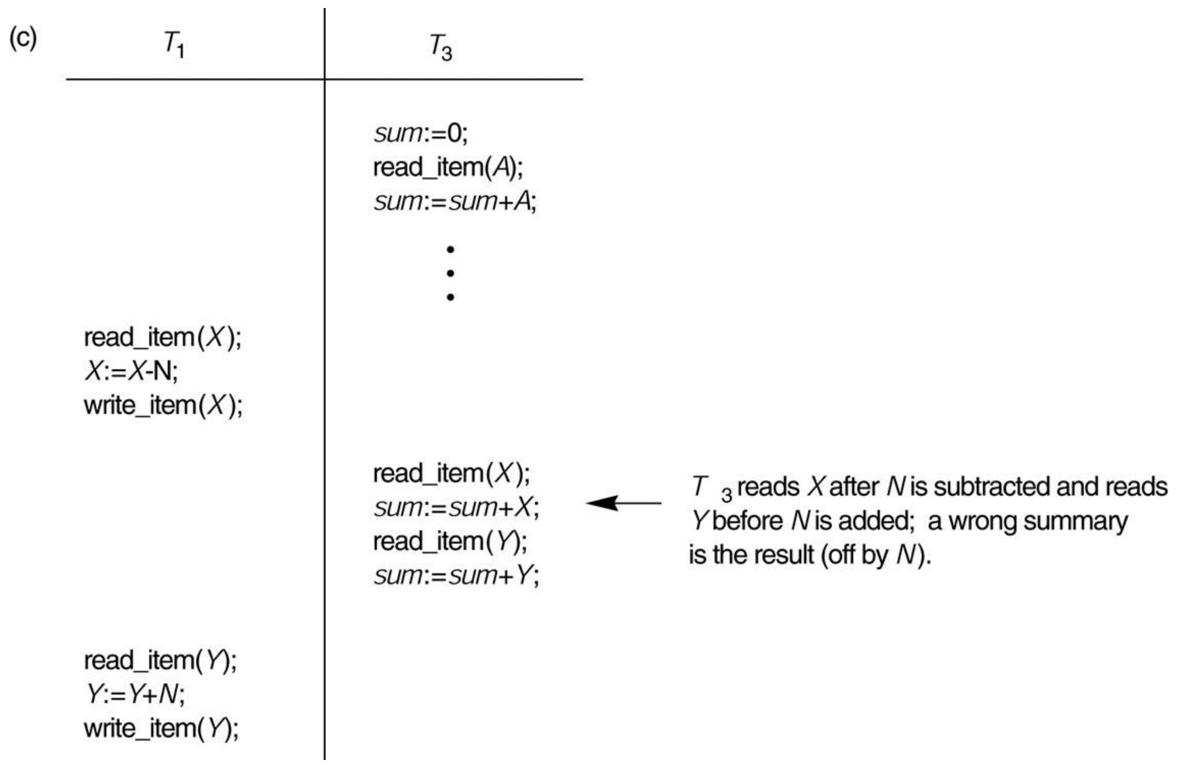


Figure: (c) The Incorrect Summary Problem

Locking Protocols

Explain in detail about various locking protocols.

Locking

- **Locking** is a procedure used to control concurrent access to data when one transaction is accessing the database, a lock may deny access to other transactions to prevent incorrect results.
- A transaction must obtain a read or write lock on a data item before it can perform a read or write operation.
- The read lock is also called a shared lock. The write lock is also known as an exclusive lock. The lock depending on its types gives or denies access to other operations on the same data item.

The basic rules **for locking are,**

- If a transaction has a read lock on a data item, it can read the item but not update it.
- If a transition has a read lock on a data item, other transactions can obtain a read lock on the data item, but no write locks.
- If a transaction has a write lock on a data item, it can both read and update the data item.
- If a transaction has a write lock on a data item, then other transactions cannot obtain either a read lock or a write lock on the data item.

The locking worksas,

- All transactions that needs to access a data item must first acquire a read lock or write lock on the data item depending on whether it is a ready only operation or not.
- If the data item for which the lock is requested is not already locked, the transaction is granted the requested lock,
- If the item is currently lock, the DBMS determines what kind of lock is the current one. The DBMS also finds out what lock is requested.
- If a read lock is requested on an item that is already under a read lock, then the requested will be granted.
- If a read lock or a write lock is requested on an item that is already under a write lock, then the request is denied and the transaction must wait until the lock is released.
- A transaction continues to hold the lock until it explicitly releases it either during execution or when itterminates.
- The effects of a write operation will be visible to other transactions only after the write lock is released.

<u>Locking Mechanisms</u>

Explain various locking mechanisms in detail.

<u>1. Lock-Based Protocols</u>

- A lock is a mechanism to control concurrent access to a data item.
- It assures that one process should not retrieve or update a record which another process is updating.
- For example, in traffic, there are signals which indicate stop and go.
- When one signal is allowed to pass at a time, then other signals are locked.
- Similarly, in database transaction only one transaction is performed at a time and other transactions are locked.
- If the locking is not done properly, then it will display the inconsistent and corrupt data.
- It manages the order between the conflicting pairs among transactions at the time of execution.

There are two lock modes,

- Binary Lock
- Shared Lock / Exclusive Lock

Binary Lock

- A lock on a data item can be in two states; it is either locked or unlocked.

Shared (S) Lock Mode

- Shared Locks are represented by S.
- The data items can only read without performing modification to it from the database.
- S – lock is requested using lock – s instruction.

Exclusive (X) Lock Mode

- Exclusive Locks are represented by X.
- The data items can be read as well as written.

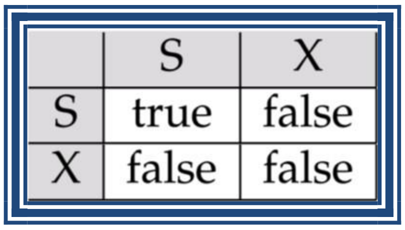
- X – lock is requested using lock – X instruction.

Lock Compatibility Matrix

- Lock Compatibility Matrix controls whether multiple transactions can acquire locks on the same resource at the same time.

	Shared	**Exclusive**
Shared	True	False
Exclusive	False	False

(Or)



	S	X
S	true	false
X	false	false

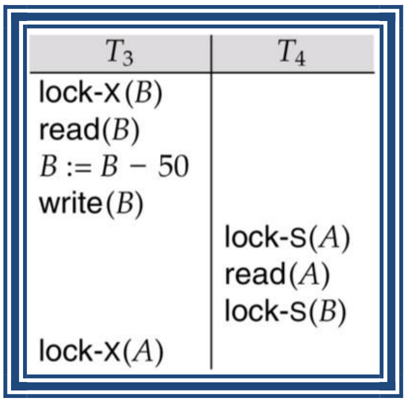
- If a resource is already locked by another transaction, then a new lock request can be granted only if the mode of the requested lock is compatible with the mode of the existing lock.
- Any number of transactions can hold shared locks on an item, but if any transaction holds an exclusive lock on item, no other transaction may hold any lock on the item.
- Example of a transaction performing locking:

  T$_2$: **lock-S**(*A*);
  read(*A*);
  unlock(*A*);
  lock-S(*B*);
  read(*B*);
  unlock(*B*);
  display(*A*+*B*)

- Locking as above is not sufficient to guarantee serializability — if *A* and *B* get updated in- between the read of *A* and *B*, the displayed sum would be wrong.
- A **locking protocol** is a set of rules followed by all transactions while requesting and releasing locks. Locking protocols restrict the set of possible schedules.

Pitfalls of Lock-Based Protocols

- Consider the partial schedule



T_3	T_4
lock-X(*B*)	
read(*B*)	
B := *B* − 50	
write(*B*)	
	lock-S(*A*)
	read(*A*)
	lock-S(*B*)
lock-X(*A*)	

- Neither T_3 nor T_4 can make progress — executing lock-**S**(B) causes T_4 to wait for T_3 to release its lock on B, while executing **lock-X**(A) causes T_3 to wait for T_4 to release its lock on A.
- Such a situation is called a **deadlock**.
  - ✓ To handle a deadlock one of T_3 or T_4 must be rolled back and its locks released.
- The potential for deadlock exists in most locking protocols. Deadlocks are a necessary evil.
- **Starvation** is also possible if concurrency control manager is badly designed. For example:
  - ✓ A transaction may be waiting for an X-lock on an item, while a sequence of other transactions request and are granted an S-lock on the same item.
  - ✓ The same transaction is repeatedly rolled back due to deadlocks.
- Concurrency control manager can be designed to prevent starvation.

2. Timestamp Based Protocol

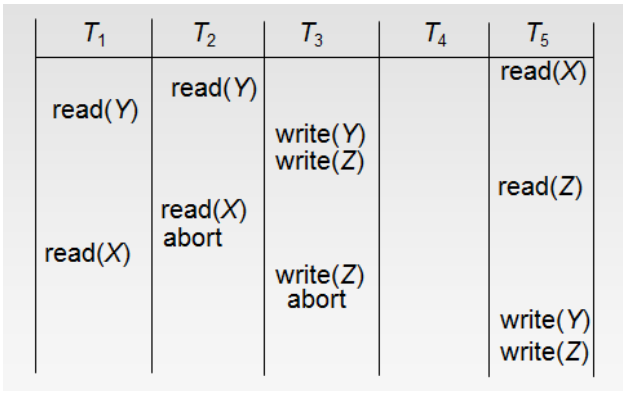
- Timestamp Based Protocol helps DBMS to identify the transactions.
- It is a unique identifier. Each transaction is issued a timestamp when it enters into the system.
- Timestamp protocol determines the serializability order.
- It is most commonly used concurrency protocol.
- It uses either system time or logical counter as a timestamp.
- It starts working as soon as a transaction is created.

- Each transaction is issued a timestamp when it enters the system.
- If an old transaction T_i has time-stamp TS(T_i), a new transaction T_j is assigned time-stamp TS(T_j) such that TS(T_i) <TS(T_j).
- The protocol manages concurrent execution such that the time-stamps determine the serializability order.
- In order to assure such behavior, the protocol maintains for each data Q two timestamp values:
  - ✓ **W-timestamp** (Q) is the largest time-stamp of any transaction that executed **write**(Q) successfully.
  - ✓ **R-timestamp** (Q) is the largest time-stamp of any transaction that executed **read**(Q) successfully.
- The timestamp ordering protocol ensures that any conflicting **read** and **write** operations are executed in timestamp order.
- Suppose a transaction T_i issues a **read**(Q)
  - ✓ If TS(T_i) ≤ **W**-timestamp (Q), then T_i needs to read a value of Q that was already overwritten.
    - ▪ Hence, the **read** operation is rejected, and T_i is rolled back.

- ✓ If TS(T_i) ≥ **W**-timestamp (Q), then the **read** operation is executed, and R- timestamp(Q) is set to **max**(R-timestamp(Q), TS(T_i)).
- Suppose that transaction T_i issues **write**(Q).
  - ✓ If TS(T_i) < R-timestamp (Q), then the value of Q that T_i is producing was needed previously, and the system assumed that that value would never be produced.
    - Hence, the **write** operation is rejected, and T_i is rolled back.
  - ✓ If TS(T_i) < W-timestamp (Q), then T_i is attempting to write an obsolete value of Q.
    - Hence, this **write** operation is rejected, and T_i is rolled back.
  - ✓ Otherwise, the **write** operation is executed, and W-timestamp (Q) is set to TS(T_i).

Example use of the Protocol

- A partial schedule for several data items for transactions with timestamps 1, 2, 3, 4, 5.



T_1	T_2	T_3	T_4	T_5
				read(X)
	read(Y)			
read(Y)				

		write(Y)
		write(Z)		

				read(Z)
	read(X) abort			
read(X)				
		write(Z) abort		

				write(Y)
				write(Z)

<u>3. Timestamp Ordering Protocol</u>

- The timestamp ordering Protocol ensures serializability among transactions in their conflicting read and write operations.
- This is the responsibility of the protocol system that the conflicting pair of tasks should be executed according to the timestamp values of the transactions.
- The timestamp of transaction T_i is denoted as TS(T_i).
- Read time-stamp of data-item X is denoted by R-timestamp(X).
- Write time-stamp of data-item X is denoted by W-timestamp(X). Timestamp

ordering protocol works as follows −

- **If a transaction Ti issues a read(X) operation −**
  - If TS(Ti) < W-timestamp(X)
    - Operation rejected.
  - If TS(Ti) >= W-timestamp(X)
    - Operation executed.
  - All data-item timestamps updated.
- **If a transaction Ti issues a write(X) operation −**
  - If TS(Ti) <R-timestamp(X)
    - Operation rejected.
  - If TS(Ti) < W-timestamp(X)
    - Operation rejected and Ti rolled back.
  - Otherwise, operation executed.

Following are the Timestamp Ordering Algorithms,

1. Basic Timestamp Ordering

- It compares the timestamp of T with Read_TS(X) and Write_TS(X) to ensure that the transaction execution is not violated.
- If the transaction execution order is violated, transaction T is aborted and resubmitted to the system as a new transaction with a new timestamp.

2. Strict Timestamp Ordering

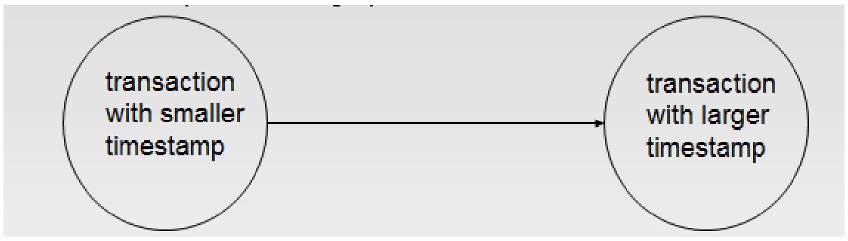
- It ensures that the schedules are both strict for easy recoverability and conflict serializability.

3. Thomas's Write Rule

- Modified version of the timestamp-ordering protocol in which obsolete **write** operations may be ignored under certain circumstances.
- It does not enforce conflict serializability.
- This rule states if TS(Ti) < W-timestamp(X), then the operation is rejected and T_i is rolled back.
- It rejects some write operations, by modifying the checks for the write_item(X) operation as follows,
  - ✓ If Read_TS(X) > TS(T) (read timestamp is greater than timestamp transaction), then abort and rollback transaction T and reject the operation.
  - ✓ If Write_TS(X) > TS(T) (write timestamp is greater than timestamp transaction), then do not execute the write operation but continue processing. Because some transaction with a timestamp is greater than TS(T) and after T in the timestamp has already written the value of X.
  - ✓ If neither the condition transaction 1 nor the condition in transaction 2 occurs, then execute the Write_item(X) operation of transaction T and set Write_TS(X) to TS(T).
- Otherwise this protocol is the same as the timestamp ordering protocol.
- Thomas' Write Rule allows greater potential concurrency.
  - ✓ Allows Time-stamp ordering rules allows to make the schedule view serializable.
- Instead of making T_i rolled back, the 'write' operation itself is ignored.

Correctness of Timestamp-Ordering Protocol

- The timestamp-ordering protocol guarantees serializability since all the arcs in the precedence graph are of the form:


transaction with smaller timestamp
transaction with larger timestamp

- Thus, there will be no cycles in the precedence graph
  - ✓ Timestamp protocol ensures freedom from deadlock as no transaction ever waits.
  - ✓ But the schedule may not be cascade-free, and may not even be recoverable.

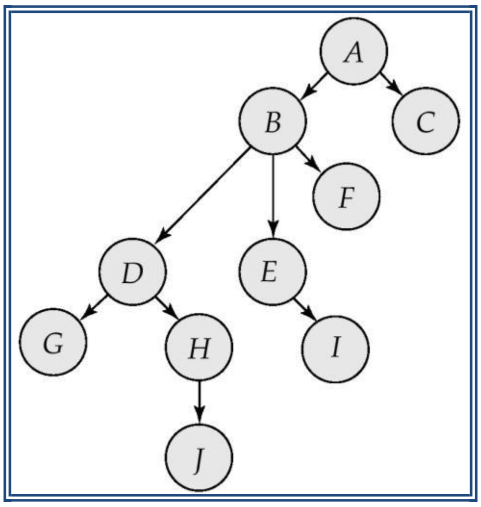
Recoverability and Cascade Freedom

- Problem with timestamp-ordering protocol:
  - ✓ Suppose T_i aborts, but T_j has read a data item written by T_i.
  - ✓ Then T_j must abort; if T_j had been allowed to commit earlier, the schedule is not recoverable.
  - ✓ Further, any transaction that has read a data item written by T_j must abort.
  - ✓ This can lead to cascading rollback --- that is, a chain of rollbacks.
- **Solution 1:**
  - ✓ A transaction is structured such that its writes are all performed at the end of its processing.
  - ✓ All writes of a transaction form an atomic action; no transaction may execute while a transaction is being written.
  - ✓ A transaction that aborts is restarted with a new timestamp.
- **Solution 2:** Limited form of locking: wait for data to be committed before reading it.
- **Solution 3**: Use commit dependencies to ensure recoverability.

4. <u>Graph based Protocols</u>

- Graph-based protocols are an alternative to two-phase locking
- Impose a partial ordering $\rightarrow$ on the set **D** = $\{d_1, d_2, ..., d_h\}$ of all data items.
  - ✓ If $d_i \rightarrow d_j$ then any transaction accessing both d_i and d_j must access d_i before accessing d_j.
  - ✓ Implies that the set **D** may now be viewed as a directed acyclic graph, called a *database graph*.
- The *tree-protocol* is a simple kind of graph protocol.

Tree Protocol



1. Only exclusive locks are allowed.
2. The first lock by T_i may be on any data item. Subsequently, a data Q can be locked by T_i only if the parent of Q is currently locked by T_i.
3. Data items may be unlocked at any time.

4. A data item that has been locked and unlocked by T_i cannot subsequently be relocked by T_i

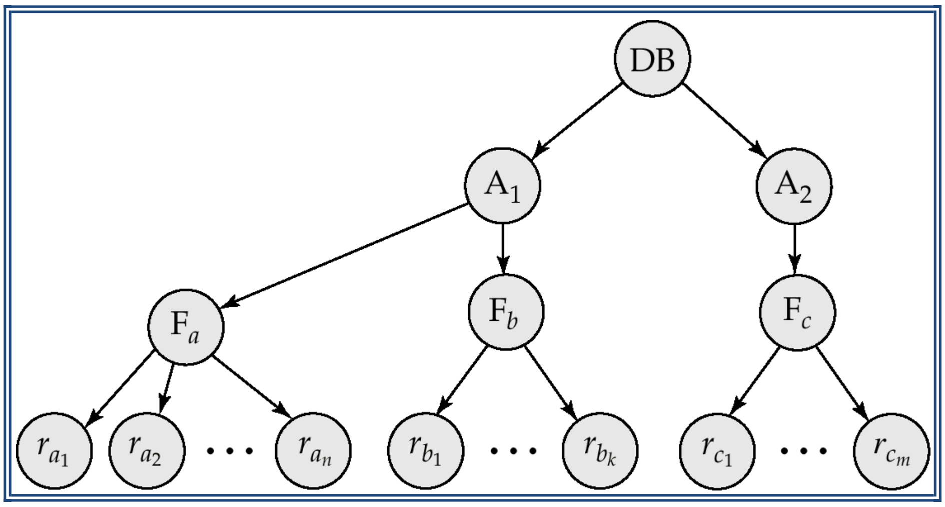
- The tree protocol ensures conflict serializability as well as freedom from deadlock.
- Unlocking may occur earlier in the tree-locking protocol than in the two-phase locking protocol.
  - ✓ shorter waiting times, and increase in concurrency
  - ✓ protocol is deadlock-free, no rollbacks are required
- **Drawbacks**
  - ✓ Protocol does not guarantee recoverability or cascade freedom
  - ✓ Need to introduce commit dependencies to ensure recoverability
  - ✓ Transactions may have to lock data items that they do not access.
  - ✓ increased locking overhead, and additional waiting time
  - ✓ potential decrease inconcurrency
  - ✓ Schedules not possible under two-phase locking are possible under tree protocol, and vice versa.

Multiple Granularity

Write short notes on Multiple Granularity.

- Allow data items to be of various sizes and define a hierarchy of data granularities, where the small granularities are nested within larger ones
- Can be represented graphically as a tree (but don't confuse with tree-locking protocol)
- When a transaction locks a node in the tree *explicitly*, it *implicitly* locks the entire node's descendents in the same mode.
- Granularity of locking (level in tree where locking is done):
  - ✓ **Fine Granularity** (lower in tree): high concurrency, high locking overhead.
  - ✓ **Coarse Granularity** (higher in tree): low locking overhead, low concurrency.

Example of Granularity Hierarchy



The levels, starting from the coarsest (top) level are,

- database
- area

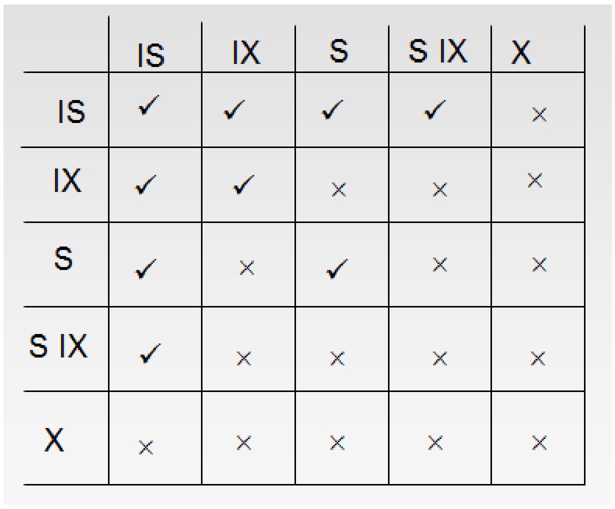
- file
- record

Intention Lock Modes

- In addition to S and X lock modes, there are three additional lock modes with multiple granularity:
  - ✓ ***Intention-Exclusive*** (IX): indicates explicit locking at a lower level with exclusive or shared locks
  - ✓ ***Shared and Intention-Exclusive*** (SIX): the subtree rooted by that node is locked explicitly in shared mode and explicit locking is being done at a lower level with exclusive-mode locks.
- Intention locks allow a higher level node to be locked in S or X mode without having to check all descendent nodes.

Compatibility Matrix with Intention Lock Modes

- The compatibility matrix for all lock modes is:



	IS	IX	S	S IX	X
IS	✓	✓	✓	✓	×
IX	✓	✓	×	×	×
S	✓	×	✓	×	×
S IX	✓	×	×	×	×
X	×	×	×	×	×

Multiple Granularity Locking Scheme

- Transaction T_i can lock a node Q, using the following rules:
  - ✓ The lock compatibility matrix must be observed.
  - ✓ The root of the tree must be locked first, and may be locked in any mode.
  - ✓ A node Q can be locked by T_i in S or IS mode only if the parent of Q is currently locked by T_i in either IX or IS mode.
  - ✓ A node Q can be locked by T_i in X, SIX, or IX mode only if the parent of Q is currently locked by T_i in either IX or SIX mode.
  - ✓ T_i can lock a node only if it has not previously unlocked any node (that is, T_i is two- phase).
  - ✓ T_i can unlock a node Q only if none of the children of Q are currently locked by T_i.
- Observe that locks are acquired in root-to-leaf order, whereas they are released in leaf-to- root order.

Two Phase Locking

Explain the two phase commit and three phase commit protocols. (April/May 2015) (Or) Explain about Locking Protocols. (May/June 2016) (Or) Briefly explain about two phase commit. (May/June 2016) (Or) Illustrate two phase locking protocol with an example. (Nov/Dec 2016) (Or) What is Concurrency? Explain the two phase locking with an example. (April/May 2018)

- Two phase commit is important whenever a given transaction can interact with several independent ―resource managers‖, each managing its own set of recoverable resources and maintaining its own recovery log.
- The two phase commit protocol guarantees that if a portion of a transaction operation cannot be committed; all changes made at the other sites participating in the transaction will be undone to maintain a consistent database state.
- For example, transaction running on an IBM mainframe that updates both an IMS database and DB2 database.
- If the transaction completes successfully, then allow updates to both IMS data and DB2 data, must be committed equally. If it fails then all of its updates must be rolled back.
- Transaction has completed its database processing successfully, the system broad instruction it issues COMMIT, not ROLLBACK on receiving that commit request, the coordinator goes through the following two-phase process:

Phase 1: Preparation

- The coordinator sends a PREPARE TO COMMIT message to all subordinates.
- The Subordinates receive the message. Write the transaction log, using the write ahead protocol and send an acknowledgement (YES/PREPARED TO COMMIT or NO/NOT PREPARED) message to coordinator.
- The coordinator makes sure that all nodes are ready to commit, or it aborts the action.
- If all nodes are PREPARED TO COMMIT, the transaction goes to phase-2 if one or more nodes reply NO or NOT PREPARED, the coordinator broadcasts an ABORT Message to all subordinates

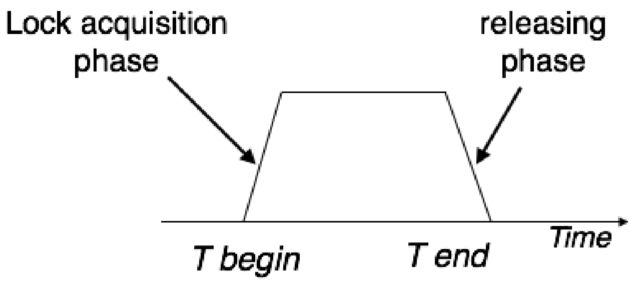
Phase 2: The Final Commit

- The coordinator broadcast a COMMIT message to all subordinates and waits for the replies
- Each subordinate receives the COMMIT message, then updates the database using the read or write operation in subordinator to coordinator.
- The subordinates reply with COMMITTED or NOT COMMITTED message to the coordinator
- If one or more subordinates did not commit, the coordinator sends an ABORT message thereby forcing them to UNDO all changes.
- The objective of the two phase commit is to ensure that all nodes commit their part of the transaction is aborted.
- If one of the nodes fails to commit, the information necessary to recover the database is in the transaction log, and the database can be recovered with do-undo-redo protocol.

The Two-Phase Locking Protocol

What is Two-Phase Locking (2PL)? Explain two phase locking protocol in detail.

- Two-Phase Locking (2PL) is a concurrency control method which divides the execution phase of a transaction into three parts.
- It ensures conflict serializable schedules.
- If read and write operations introduce the first unlock operation in the transaction, then it is said to be Two-Phase Locking Protocol.



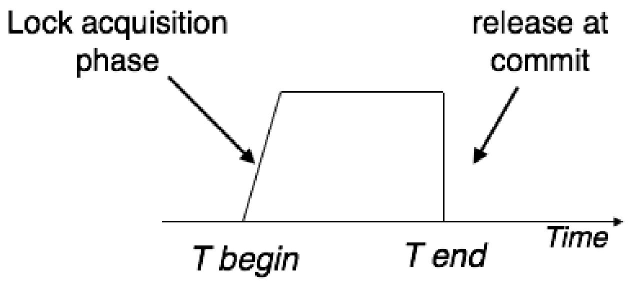
- This protocol can be divided into two phases,
  - ✓ In **Growing Phase**, a transaction obtains locks, but may not release any lock.
  - ✓ In **Shrinking Phase**, a transaction may release locks, but may not obtain any lock.
- Two-Phase Locking does not ensure freedom from deadlocks.

Types of Two – Phase Locking Protocol

- Following are the types of two – phase locking protocol:
  - ✓ Strict Two – Phase Locking Protocol
  - ✓ Rigorous Two – Phase Locking Protocol
  - ✓ Conservative Two – Phase Locking Protocol

Strict Two-Phase Locking Protocol

- Strict Two-Phase Locking Protocol avoids cascaded rollbacks.
- This protocol not only requires two-phase locking but also all exclusive-locks should be held until the transaction commits or aborts.
- It is not deadlock free.
- It ensures that if data is being modified by one transaction, then other transaction cannot read it until first transaction commits.
- Most of the database systems implement rigorous two – phase locking protocol.



Rigorous Two-Phase Locking

- Rigorous Two – Phase Locking Protocol avoids cascading rollbacks.
- This protocol requires that all the share and exclusive locks to be held until the transaction commits.

Conservative Two-Phase Locking Protocol

- Conservative Two – Phase Locking Protocol is also called as Static Two – Phase Locking Protocol.
- This protocol is almost free from deadlocks as all required items are listed in advanced.
- It requires locking of all data items to access before the transaction starts.
- This is a protocol which ensures conflict-serializable schedules.

Lock Conversions

- It is a mechanism in two phase locking mechanism which allows conversion of shared lock to exclusive lock or vice versa.
- Two-phase locking with lock conversions:

First Phase:

- can acquire a lock-S on item
- can acquire a lock-X on item
- can convert a lock-S to a lock-X (upgrade)

Second Phase:

- can release a lock-S
- can release a lock-X
- can convert a lock-X to a lock-S (downgrade)

- This protocol assures serializability. But still relies on the programmer to insert the various locking instructions.

Automatic Acquisition of Locks

- A transaction T_i issues the standard read/write instruction, without explicit locking calls.
- The operation **read**(D) is processed as:

```
If T_i has a lock on D
then
read(D)
else begin
if necessary wait until no other
transaction has a lock-X on D grant
T_i a lock-S on D; read(D)
end
```

- **write**(D) is processed as:

```
if T_i
has a lock-X on D then
write(D)
else begin
if necessary wait until no other trans. has any lock on D, if T_i has a
lock-S on D
then
```

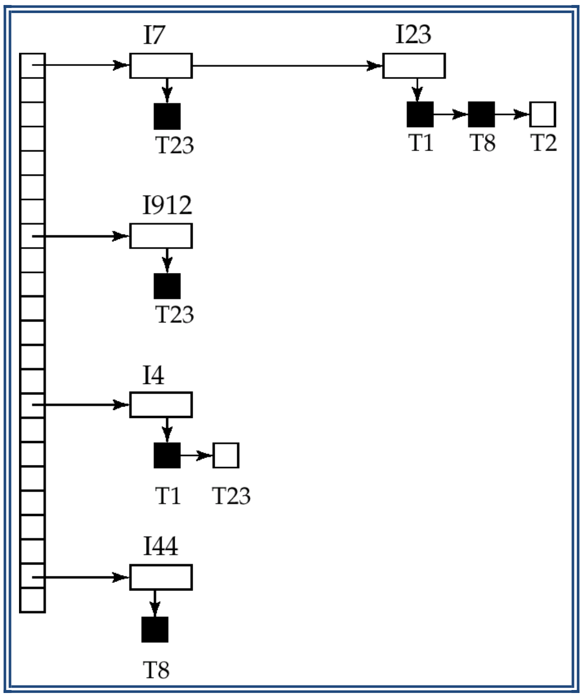
```
upgrade lock on D to lock-X else
grant Ti a lock-X on D
write(D)
end;
```

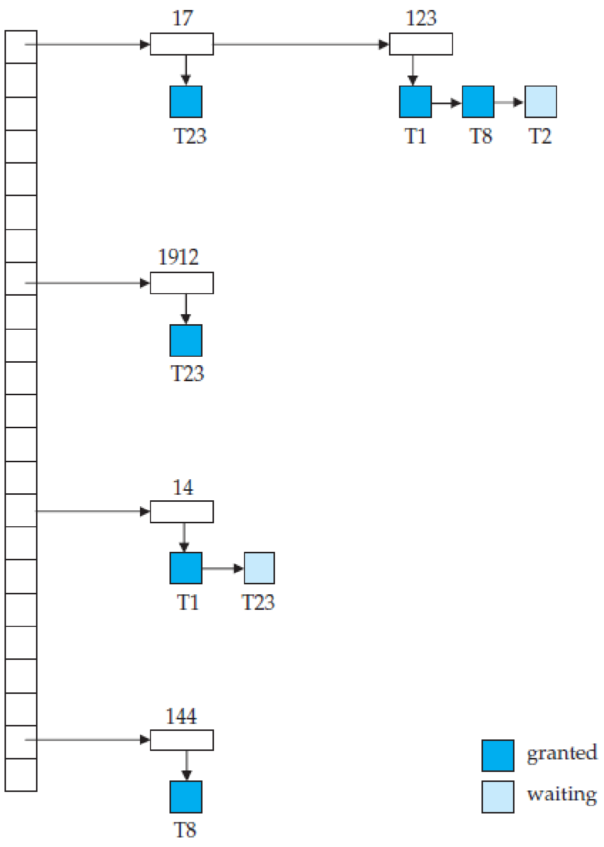
- All locks are released after commit or abort

Implementation of Locking

- A **lock manager** can be implemented as a separate process to which transactions send lock and unlock requests
- The lock manager replies to a lock request by sending a lock grant messages (or a message asking the transaction to roll back, in case of a deadlock)
- The requesting transaction waits until its request is answered
- The lock manager maintains a data-structure called a **lock table** to record granted locks and pending requests
- The lock table is usually implemented as an in-memory hash table indexed on the name of the data item being locked.

Lock Table





- Black rectangles indicate granted locks, white ones indicate waiting requests.
- Lock table also records the type of lock granted or requested.
- New request is added to the end of the queue of requests for the data item, and granted if it is compatible with all earlier locks.

- Unlock requests result in the request being deleted, and later requests are checked to see if they can now be granted.
- If transaction aborts, all waiting or granted requests of the transaction are deleted.
    - ✓ Lock manager may keep a list of locks held by each transaction, to implement this efficiently.

Deadlock

Write short notes on deadlock. (Nov/Dec 2014) (Or) What is Deadlock? How does it occur? How transactions be written to avoid deadlock, guarantee correct execution? (Nov/Dec 2105) (Or) Outline deadlock handling mechanisms. (Nov/Dec 2016) (Or) When does deadlock occurs? Explain two-phase commit protocol with example. (Nov/Dec 2017) (Or) Explain the methods used to handle deadlock. (Nov/Dec 2018)

What is Deadlock?

- A deadlock is a condition that occurs when two or more different database tasks are waiting for each other and none of the task is willing to give up the resources that other task needs.
- It is an unwanted situation that may result when two or more transactions are each waiting for locks held by the other to be released.
- In deadlock situation, no task ever gets finished and is in waiting state forever.
- Deadlocks are not good for the system.

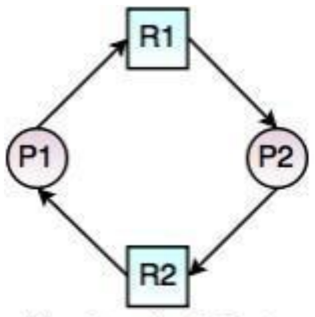


Fig. Deadlock State

In the above diagram,

- Process P1 holds Resource R2 and waits for resource R1, while Process P2 holds resource R1 and waits for Resource R2. So, the above process is in deadlock state.
- There is the only way to break a deadlock, is to abort one or more transactions.
- Once, a transaction is aborted and rolled back, all the locks held by that transaction are released and can continue their execution.
- So, the DBMS should automatically restart the aborted transactions.

Deadlock Conditions

Following are the deadlock conditions,

1. Mutual Exclusion
2. Hold and Wait
3. No Preemption
4. Circular Wait

- A deadlock may occur, if all the above conditions hold true.
  - ✓ **In Mutual exclusion states** that at least one resource cannot be used by more than one process at a time. The resources cannot be shared between processes.
  - ✓ **Hold and Wait states** that a process is holding a resource, requesting for additional resources which are being held by other processes in the system.
  - ✓ **No Preemption states** that a resource cannot be forcibly taken from a process. Only a process can release a resource that is being held by it.
  - ✓ **Circular Wait states** that one process is waiting for a resource which is being held by second process and the second process is waiting for the third process and so on and the last process is waiting for the first process. It makes a circular chain of waiting.
- There are three general techniques for handling deadlock:
  - ✓ Timeouts
  - ✓ Deadlock Prevention
  - ✓ Deadlock Detection
  - ✓ Recovery

Timeouts

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

Deadlock Prevention

- Deadlock Prevention ensures that the system never enters a deadlock state. Following are the

requirements to free the deadlock:

- **No Mutual Exclusion:**
  - ✓ No Mutual Exclusion means removing all the resources that are sharable.
- **No Hold and Wait:**
  - ✓ Removing hold and wait condition can be done if a process acquires all the resources that are needed before starting out.
- **Allow Preemption:**
  - ✓ Allowing preemption is as good as removing mutual exclusion.
  - ✓ The only need is to restore the state of the resource for the preempted process rather than letting it in at the same time as the preemptor.
- **Removing Circular Wait:**
  - ✓ The circular wait can be removed only if the resources are maintained in a hierarchy and process can hold the resources in increasing the order of precedence.
- There are deadlock prevention schemes that use timestamp ordering mechanism of transactions in order to predetermine a deadlock situation.

Wait-Die Scheme

- In this scheme, if a transaction requests to lock a resource (data item), which is already held with a conflicting lock by another transaction, then one of the two possibilities may occur −
    - ✓ If $TS(T_i) < TS(T_j)$ − that is T_i, which is requesting a conflicting lock, is older than T_j − then T_i is allowed to wait until the data-item is available.
    - ✓ If $TS(T_i) > TS(t_j)$ − that is T_i is younger than T_j − then T_i dies. T_i is restarted later with a random delay but with the same timestamp.
- This scheme allows the older transaction to wait but kills the younger one.

Wound-Wait Scheme

- In this scheme, if a transaction requests to lock a resource (data item), which is already held with conflicting lock by some another transaction, one of the two possibilities may occur −
    - ✓ If $TS(T_i) < TS(T_j)$, then T_i forces T_j to be rolled back − that is T_i wounds T_j. T_j is restarted later with a random delay but with the same timestamp.
    - ✓ If $TS(T_i) > TS(T_j)$, then T_i is forced to wait until the resource is available.
- This scheme, allows the younger transaction to wait; but when an older transaction requests an item held by a younger one, the older transaction forces the younger one to abort and release the item.
- In both the cases, the transaction that enters the system at a later stage is aborted.
- Both in *wait-die* and in *wound-wait* schemes, a rolled back transactions is restarted with its original timestamp. Older transactions thus have precedence over newer ones, and starvation is hence avoided.
- Timeout-Based Schemes :
    - ✓ A transaction waits for a lock only for a specified amount of time. After that, the wait times out and the transaction is rolled back.
    - ✓ Thus deadlocks are not possible.
    - ✓ Simple to implement; but starvation is possible. Also difficult to determine good value of the timeout interval.

<u>**Deadlock Avoidance**</u>

- Deadlock Avoidance helps in avoiding the rolling back conflicting transactions.
- It is not good or practical approach to abort a transaction when a deadlock occurs.
- Instead, deadlock avoidance mechanisms can be used to detect any deadlock situation in advance.
- Deadlock occurs when each transaction T in a set of two or more transactions is waiting for some item that is locked by some other transaction T in the set.

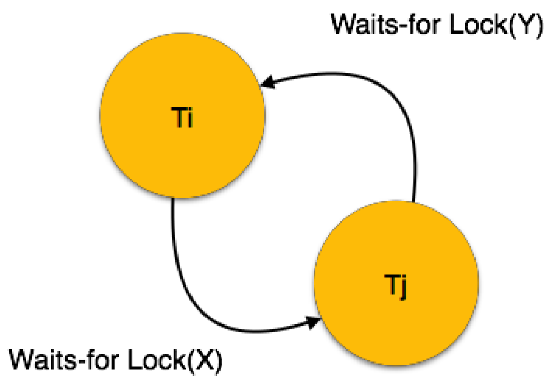
(Or)

- System is deadlocked if there is a set of transactions such that every transaction in the set is waiting for another transaction in the set.
- There is only one way to break deadlock: abort one or more of the transactions. This usually involves undoing all the changes made by the aborted transactions (S).

- Methods like "wait-for graph" are available but they are suitable for only those systems where transactions are lightweight having fewer instances of resource.
- In a bulky system, deadlock prevention techniques may work well.

Wait-for Graph

- This is a simple method available to track if any deadlock situation may arise.
- For each transaction entering into the system, a node is created.
- When a transaction T_i requests for a lock on an item, say X, which is held by some other transaction T_j, a directed edge is created from T_i to T_j.
- If T_j releases item X, the edge between them is dropped and T_i locks the data item.
- The system maintains this wait-for graph for every transaction waiting for some data items held by others.
- The system keeps checking if there's any cycle in the graph.



Here, we can use any of the two following approaches –

- First, do not allow any request for an item, which is already locked by another transaction. This is not always feasible and may cause starvation, where a transaction indefinitely waits for a data item and can never acquire it.
- The second option is to roll back one of the transactions.
- It is not always feasible to roll back the younger transaction, as it may be important than the older one.
- With the help of some relative algorithm, a transaction is chosen, which is to be aborted.
- This transaction is known as the **victim** and the process is known as **victim selection.** **<u>Deadlock Recovery</u>**

- When deadlock is detected :
  - Some transaction will have to rolled back (made a victim) to break deadlock. Select that transaction as victim that will incur minimum cost.
  - Rollback -- determine how far to roll back transaction
    - Total rollback: Abort the transaction and then restart it.
    - More effective to roll back transaction only as far as necessary to break deadlock.
  - Starvation happens if same transaction is always chosen as victim. Include the number of rollbacks in the cost factor to avoid starvation.

Transaction Recovery

Explain in detail about transaction recovery.

- A transaction is the basic logical unit of execution in an information system.
- A transaction is a sequence of operations that must be executed as a whole, taking a consistent (& correct) database state into another consistent (& correct) database state.
- Data **recovery** is the process of restoring data that has been lost, accidentally deleted, corrupted or made inaccessible.

(Or)

- Data **recovery** typically refers to the restoration of data to a desktop, laptop, server or external storage system from a backup.
- The techniques used to **recover** the lost data due to system crash, **transaction** errors, viruses, catastrophic failure, incorrect commands execution etc. are called as database **recovery** techniques.

What is Recovery?

- Recovery is the process of restoring a database to the correct state in the event of a failure.
- It ensures that the database is reliable and remains in consistent state in case of a failure.

Failure Classification

- We generalize a failure into various categories, as follows
- There are some common causes of failures such as,
  1. System Crash
  2. Transaction Failure
  3. Network Failure
  4. Disk Failure
  5. Media Failure
- Each transaction has ACID property. If we fail to maintain the ACID properties, it is the failure of the database system.

1. System Crash

- System crash occurs when there is a hardware or software failure or external factors like a power failure.
- The data in the secondary memory is not affected when system crashes because the database has lots of integrity. Checkpoint prevents the loss of data from secondary memory.

2. Transaction Failure

- A transaction has to abort when it fails to execute or when it reaches a point from where it can‘t go any further.
- This is called transaction failure where only a few transactions or processes are hurt.
- Reasons for a transaction failure could be −
- **Logical errors** − Where a transaction cannot complete because it has some code error or any internal error condition.

- **System errors** − Where the database system itself terminates an active transaction because the DBMS is not able to execute it, or it has to stop because of some system condition. For example, in case of deadlock or resource unavailability, the system aborts an active transaction.
- This failure occurs when there are system errors like deadlock or unavailability of system resources to execute the transaction.

3. Network Failure

- A network failure occurs when a client – server configuration or distributed database system are connected by communication networks.

4. Disk Failure

- Disk Failure occurs when there are issues with hard disks like formation of bad sectors, disk head crash, unavailability of disk etc.

5. Media Failure

- Media failure is the most dangerous failure because, it takes more time to recover than any other kind of failures.
- A disk controller or disk head crash is a typical example of media failure.
- Natural disasters like floods, earthquakes, power failures, etc. damage the data.

Storage Structure / Storage of Data

- A DBMS stores the data on external storage because the amount of data is very huge and must persist across program executions.
- Data storage is the memory structure in the system.
- The storage structure/storage of data can be divided into three categories −
  - Volatile Memory
  - ✓ Non – Volatile Memory
  - Stable Memory

Volatile Memory

- Volatile memory can store only a small amount of data. For eg. Main memory, cache memory etc.
- Volatile memory is the primary memory device in the system and placed along with the CPU.
- In volatile memory, if the system crashes, then the data will be lost.
- RAM is a primary storage device which stores a disk buffer, active logs and other related data of a database.
- Primary memory is always faster than secondary memory.
- When we fire a query, the database fetches a data from the primary memory and then moves to the secondary memory to fetch the record.
- If the primary memory crashes, then the whole data in the primary memory is lost and cannot be recovered.
- To avoid data loss, create a copy of primary memory in the database with all the logs and buffers, create checkpoints at several places so the data is copied to the database.

Non - VolatileMemory

- Non – volatile memory is the secondary memory.
- These memories are huge in size, but slow in processing. For eg. Flash memory, hard disk, magnetic tapes etc.
- If the secondary memory crashes, whole data in the primary memory is lost and cannot be recovered.

To avoid data loss in the secondary memory, there are three methods used to back it up:

1. Remote backup creates a database copy and stores it in the remote network.
   - ✓ The database is updated with the current database and sync with data and other details.
   - ✓ The remote backup is also called as an offline backup because it can be updated manually.
   - ✓ If the current database fails, then the system automatically switches to the remote database and starts functioning. The user will not know that there was a failure.
2. The database is copied to secondary memory devices like Flash memory, hard disk, magnetic tapes, etc. and kept in a secured place.
   - ✓ If the system crashes or any failure occurs, the data would be copied from these tapes to bring the database up.
3. The huge amount of data is an overhead to backup the whole database.
   - ✓ To overcome this problem the log files are backed up at regular intervals. The log file includes all the information about the transaction being made.
   - ✓ These files are backed up at regular intervals and the database is backed up once in a week.

Stable Memory

- Stable memory is the third form of the memory structure and same as non-volatile memory.
- In stable memory, copies of the same non – volatile memories are stored in different places, because if the system crashes and data loss occurs, the data can be recovered from other copies.

<u>Log-Based Recovery</u>

- Logs are the sequence of records that maintain the records of actions performed by a transaction.
- In Log – Based Recovery, log of each transaction is maintained in some stable storage. If any failure occurs, it can be recovered from there to recover the database.
- The log contains the information about the transaction being executed, values that have been modified and transaction state.
- All these information will be stored in the order of execution.

Log-based recovery works as follows −

- The log file is kept on a stable storage media.
- When a transaction enters the system and starts execution, it writes a log about it.

  <T_n, Start>

- When the transaction modifies an item X, it write logs as follows –

  <T_n, X, V_1, V_2>

- It reads T_n has changed the value of X, from V_1 to V_2.
- When the transaction finishes, it logs –

  <T_n, commit>

- The database can be modified using two approaches. (Or) There are two methods of creating the log files and updating the database,
  1. Deferred Database Modification
  2. Immediate Database Modification

Deferred Database Modification

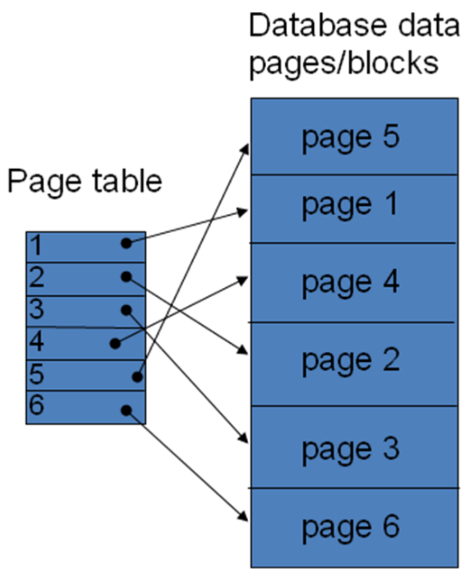
- All the logs for the transaction are created and stored into stable storage system.
- In the above example, three log records are created and stored it in some storage system, the database will be updated with those steps.

Immediate Database Modification

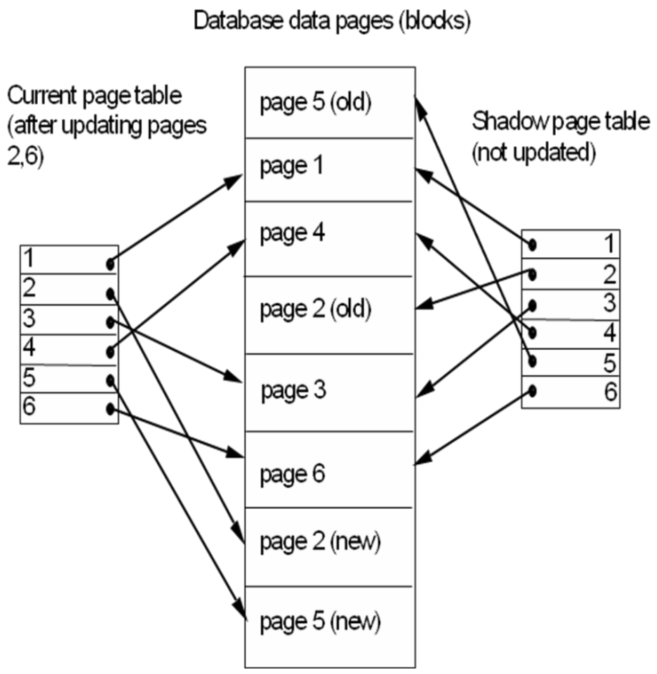
- After creating each log record, the database is modified for each step of log entry immediately.
- In the above example, the database is modified at each step of log entry that means after first log entry, transaction will hit the database to fetch the record, then the second log will be entered followed by updating the employee's address, then the third log followed by committing the database changes.

Shadow Paging Technique

- Data isn't updated _in place'
- The database is considered to be made up of a number of n fixed-size disk blocks or pages, for recovery purposes.
- A page table with n entries is constructed where the ith page table entry points to the ith database page on disk.
- Current page table points to most recent current database pages on disk



- When a transaction begins executing,
  - the current page table is copied into a shadow page table
  - shadow page table is then saved
  - shadow page table is never modified during transaction execution
  - ✓ writes operations—new copy of database page is created and current page table entry modified to point to new disk page/block


Database data pages (blocks)
Current page table (after updating pages 2,6)
Shadow page table (not updated)
page 5 (old)
page 1
page 4
page 2 (old)
page 3
page 6
page 2 (new)
page 5 (new)
1
2
3
4
5
6

- To recover from a failure
  - ✓ the state of the database before transaction execution is available through the shadow page table
  - ✓ free modified pages
  - ✓ discard currrent page table
  - ✓ that state is recovered by reinstating the shadow page table to become the current page table once more
- Commiting a transaction
  - ✓ discard previous shadow page
  - ✓ free old page tables that it references
- Garbage Collection

Recovery with Concurrent Transaction

- When more than one transaction is executed in parallel, the logs are interleaved.
- At that time of recovery, it would become difficult for the recovery system to return all logs to a previous point and then start recovering.
- To overcome this situation 'Checkpoint' is used.

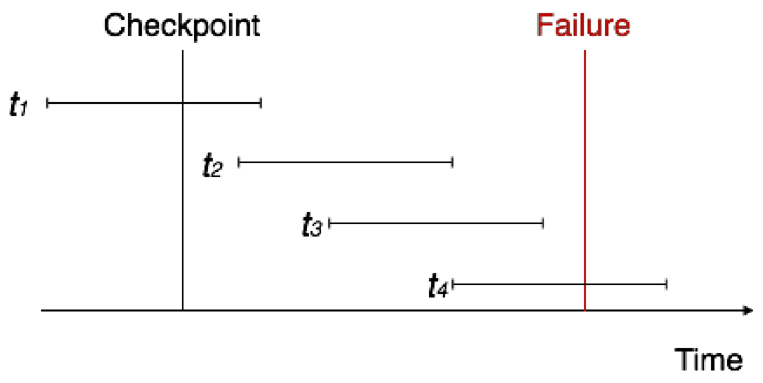
Checkpoint

- Checkpoint acts like a benchmark.
- Checkpoints are also called as **Syncpoints** or **Savepoints**.

- It is a mechanism where all the previous logs are removed from the system and stored permanently in a storage system.
- It declares a point before which the database management system was in consistent state and all the transactions were committed.
- It is a point of synchronization between the database and the transaction log file.
- It involves operations like writing log records in main memory to secondary storage, writing the modified blocks in the database buffers to secondary storage and writing a checkpoint record to the log file.
- The checkpoint record contains the identifiers of all transactions that are active at the time of the checkpoint.

Recovery

- When concurrent transactions crash and recover, the checkpoint is added to the transaction and recovery system recovers the database from failure in following manner,
- When a system with concurrent transactions crashes and recovers, it behaves in the following manner –
  - The recovery system reads the logs backwards from the end to the last checkpoint.
  - It maintains two lists, an undo-list and a redo-list.
  - If the recovery system sees a log with <T_n, Start> and <T_n, Commit> or just <T_n, Commit>, it puts the transaction in the redo-list.
  - If the recovery system sees a log with <T_n, Start> but no commit or abort log found, it puts the transaction in undo-list.
- All the transactions in the undo log are undone and their logs are removed.
- All the transactions in the redo log and their previous logs are removed and then redone before saving their logs.



Save Points

Write short notes on save points.

- A savepoint is a way of implementing subtransactions (also known as nested transactions) within a relational **database management system** by indicating a **point** within a transaction that can be "rolled back to" without affecting any work done in the transaction before the savepoint was created.
- Multiple savepoints can exist within a single transaction.
- Savepoints are useful for implementing complex error recovery in database applications.

- If an error occurs in the midst of a multiple-statement transaction, the application may be able to recover from the error (by rolling back to a savepoint) without needing to abort the entire transaction.
- A savepoint can be declared by issuing a **SAVEPOINT *name*** statement.
- All changes made after a savepoint has been declared can be undone by issuing a **ROLLBACK TO SAVEPOINT *name*** command.
- Issuing **RELEASE SAVEPOINT *name*** will cause the named savepoint to be discarded, but will not otherwise affect anything.
- Issuing the commands ROLLBACK or COMMIT will also discard any savepoints created since the start of the main transaction.

COMMIT Command

- COMMIT command saves all the work done.
- It ends the current transaction and makes permanent changes during the transaction.

Syntax:

commit;

SAVEPOINT Command

- SAVEPOINT command is used for saving all the current point in the processing of a transaction.
- It marks and saves the current point in the processing of a transaction.

Syntax:

SAVEPOINT <savepoint_name>

Example:

SAVEPOINT no_update;

- It is used to temporarily save a transaction, so that you can rollback to that point whenever necessary.

ROLLBACK Command

- ROLLBACK command restores database to original since the last COMMIT.
- It is used to restores the database to last committed state.

Syntax:

ROLLBACK TO SAVEPOINT <savepoint_name>;

Example:

ROLLBACK TO SAVEPOINT no_update;

<u>Isolation Levels</u>

Discuss isolation levels in detail.

- Isolation level is nothing but locking the row while performing some task, so that other transaction cannot access or will wait for the current transaction to finish its job.
- Serializability is a useful concept because it allows programmers to ignore issues related to concurrency when they code transactions.
- If every transaction has the property that it maintains database consistency if executed alone, then serializability ensures that concurrent executions maintain consistency.

- The SQL standard also allows a transaction to specify that it may be executed in such a way that it becomes nonserializable with respect to other transactions.

Let's write a transaction without Isolation level.

```
BEGIN TRANSACTION MyTransaction
BEGIN TRY
UPDATE Account SET Debit=100 WHERE Name='John Cena'
UPDATE ContactInformation SET Mobile='1234567890' WHERE Name='The Rock'
COMMIT TRANSACTION MyTransaction
PRINT 'TRANSACTION SUCCESS'
END TRY
BEGIN CATCH
ROLLBACK TRANSACTION MyTransaction
PRINT 'TRANSACTIONFAILED'
END CATCH
```

- SQL Server provides 5 Isolation levels to implement with SQL Transaction to maintain data concurrency in the database. The isolation levels specified by the SQL standard are as follows:

Read uncommitted

- It allows uncommitted data to be read. It is the lowest isolation level allowed by SQL.
- All the isolation levels above additionally disallow **dirty writes**, that is, they disallow writes to a data item that has already been written by another transaction that has not yet committed or aborted.

Example

```
SET TRANSACTION ISOLATION LEVEL
READ UNCOMMITTED
BEGIN TRANSACTION MyTransaction
BEGIN TRY
UPDATE Account SET Debit=100 WHERE Name='John Cena'
UPDATE ContactInformationSET Mobile='1234567890'WHERE Name='TheRock'
COMMIT TRANSACTION MyTransaction
PRINT 'TRANSACTION SUCCESS'
END TRY
BEGIN CATCH
ROLLBACK TRANSACTION MyTransaction
PRINT 'TRANSACTION FAILED'
END CATCH
```

Read committed

- It allows only committed data to be read, but does not require repeatable reads.
- For instance, between two reads of a data item by the transaction, another transaction may have updated the data item and committed.

Example

SET TRANSACTION ISOLATION LEVEL
READ COMMITTED
BEGIN TRANSACTION MyTransaction
BEGIN TRY
UPDATE Account **SET Debit=100 WHERE** Name='John Cena'
UPDATE ContactInformation**SET** Mobile='1234567890'**WHERE** Name='TheRock'
COMMIT TRANSACTION MyTransaction
PRINT 'TRANSACTION SUCCESS'
END TRY
BEGIN CATCH
ROLLBACK TRANSACTION MyTransaction
PRINT 'TRANSACTION FAILED'
END CATCH

Repeatable read

- It allows only committed data to be read and further requires that, between two reads of a data item by a transaction, no other transaction is allowed to update it.
- However, the transaction may not be serializable with respect to other transactions.
- For instance, when it is searching for data satisfying some conditions, a transaction may find some of the data inserted by a committed transaction, but may not find other data inserted by the same transaction.

Example

SET TRANSACTION ISOLATION LEVEL
REPEATABLE READ

Serializable

- It usually ensures serializable execution.
- However, as we shall explain shortly, some database systems implement this isolation level in a manner that may, in certain cases, allow nonserializable executions.

Example

SET TRANSACTION ISOLATION LEVEL
SERIALIZABLE

Snapshot

- Snapshot Isolation (SI) is an optimistic isolation level.
- Allows for processes to read older versions of committed data if the current version is locked.
- Difference between snapshot and read committed has to do with how old the older versions have to be.
- It‘s possible to have two transactions executing simultaneously that give us a result that is not possible in any serial execution.

Example

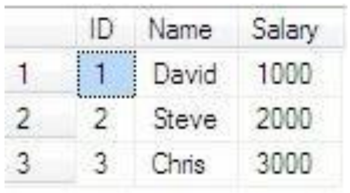
SET TRANSACTION ISOLATION LEVEL SNAPSHOT

SQL Facilities for Concurrency and Recovery

- It is possible to achieve concurrency control and recovery using SQL statements. Following are the different types of isolations available in SQL Server.
- READ COMMITTED
- READ UNCOMMITTED
- REPEATABLE READ
- SERIALIZABLE
- SNAPSHOT

Let us discuss about each isolation level in details. Before this, execute following script to create table and insert some data that we are going to use in examples for each isolation

```
IF OBJECT_ID('Emp') is not null begin
DROP TABLE Emp
end
create table Emp(ID int,Name Varchar(50),Salary Int) insert into
Emp(ID,Name,Salary) values ( 1,'David',1000) insert into
Emp(ID,Name,Salary) values ( 2,'Steve',2000) insertinto
Emp(ID,Name,Salary)values(3,'Chris',3000)
```



	ID	Name	Salary
1	1	David	1000
2	2	Steve	2000
3	3	Chris	3000

Note: Before executing each example in this article, reset the Emp table values by executing the above script.

Read Committed

- In select query it will take only commited values of table.
- If any transaction is opened and incompleted on table in others sessions then select query will wait till no transactions are pending on same table.
- Read Committed is the default transaction isolation level.

Read committed example 1:

Session 1

```
begin tran
update emp set Salary=999 where ID=1 waitfor delay
'00:00:15'
commit
```

Session 2

```
settransactionisolationlevelreadcommitted select
Salary from Emp where ID=1
```

- Run both sessions side by side.

Output

```
999
```

- In second session, it returns the result only after execution of complete transaction in first session because of the lock on Emp table.
- We have used wait command to delay 15 seconds after updating the Emp table in transaction.

Read committed example 2

Session 1

```
begin tran
select * from Emp waitfor
delay '00:00:15' commit
```

Session 2

```
settransactionisolationlevelreadcommitted select *
fromEmp
```

- Run both sessions side by side.

Output

```
1000
```

- In session 2, there won't be any delay in execution because in session 1 Emp table is used under transaction but it is not used update or delete command hence Emp table is not locked.

Read committed example 3

Session 1

```
begin tran
select * from emp waitfor
delay '00:00:15'
update emp set Salary=999 where ID=1 commit
```

Session 2

```
settransactionisolationlevelreadcommitted select
Salary from Emp where ID=1
```

- Run both sessions side by side.

Output

```
1000
```

- In session 2, there won't be any delay in execution because when session 2 is executed Emp table in session 1 is not locked (used only select command, locking on Emp table occurs after wait delay command).

Read Uncommitted

- If any table is updated (insert or update or delete) under a transaction and same transaction is not completed that is not committed or roll backed then uncommitted values will display (Dirty Read) in select query of "Read Uncommitted" isolation transaction sessions.
- There won't be any delay in select query execution because this transaction level does not wait for committed values on table.

Read uncommitted example 1

Session 1

```
begin tran
update emp set Salary=999 where ID=1 waitfor delay
'00:00:15'
rollback
```

Session 2

```
set transaction isolation level read uncommitted select Salary
from Emp where ID=1
```

- Run both sessions at a time one by one.

Output

999

- Select query in Session2 executes after update Emp table in transaction and before transaction rolled back.
- Hence 999 is returned instead of 1000.
- If you want to maintain Isolation level "Read Committed" but you want dirty read values for specific tables then use **with (nolock)** in select query for same tables as shown below.

```
set transaction isolation level read committed select *
from Emp with(nolock)
```

Repeatable Read

- select query data of table that is used under transaction of isolation level "Repeatable Read" cannot be modified from any other sessions till transcation is completed.

Repeatable Read Example 1

Session 1

```
set transaction isolation level repeatable read begin tran
select * from emp where ID in(1,2) waitfor
delay'00:00:15'
select * from Emp where ID in (1,2)
rollback
```

Session 2

update emp set Salary=999 where ID=1

- Run both sessions side by side.

Output

- Update command in session 2 will wait till session 1 transaction is completed because emp table row with ID=1 has locked in session 1 transaction.

Repeatable Read Example 2

Session 1

set transaction isolation level repeatable read begin tran
select * from emp waitfor
delay '00:00:15' select * from
Emp rollback

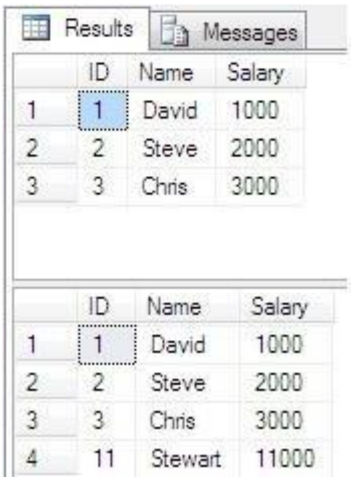
Session 2

insert into Emp(ID,Name,Salary) values(
11,'Stewart',11000)

- Run both sessions side by side.

Output

Result in Session 1.



	ID	Name	Salary
1	1	David	1000
2	2	Steve	2000
3	3	Chris	3000

	ID	Name	Salary
1	1	David	1000
2	2	Steve	2000
3	3	Chris	3000
4	11	Stewart	11000

- Session 2 will execute without any delay because it has insert query for new entry. This isolation level allows inserting new data but does not allow modifying data that is used in select query executed in transaction.
- You can notice two results displayed in Session 1 have different number of row count (1 row extra in second result set).

Repeatable Read Example 3

Session 1

set transaction isolation level repeatable read begin tran
select * from emp where ID in(1,2)

waitfor delay'00:00:15'

select * from Emp where ID in (1,2)

rollback

Session 2

update emp set Salary=999 where ID=3 Run both

sessions at a time one by one.

Output

- Session 2 will execute without any delay because row with ID=3 is not locked, that is only 2 records whose IDs are 1, 2 are locked in Session 1.

Serializable

- Serializable Isolation is similar to Repeatable Read Isolation but the difference is it prevents Phantom Read.
- This works based on range lock. If table has index then it locks records based on index range used in WHERE clause (like where ID between 1 and 3).
- If table doesn't have index then it locks complete table.

Serializable Example 1

- Assume table does not have index column.

Session 1

set transaction isolation level serializable begin tran

select * from emp waitfor

delay '00:00:15' select * from

Emp rollback

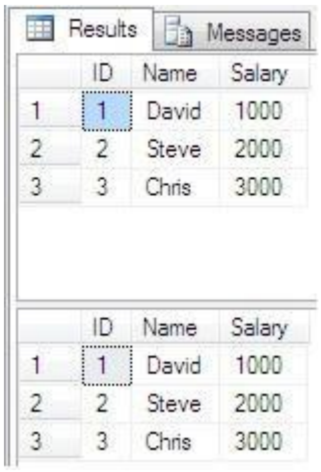
Session 2

insert into Emp(ID,Name,Salary) values (11,'Stewart',11000)

- Run both sessions side by side.

Output

Result in Session 1.



	ID	Name	Salary
1	1	David	1000
2	2	Steve	2000
3	3	Chris	3000

	ID	Name	Salary
1	1	David	1000
2	2	Steve	2000
3	3	Chris	3000

- Complete Emp table will be locked during the transaction in Session 1.

- Unlike "Repeatable Read", insert query in Session 2 will wait till session 1 execution is completed.
- Hence Phantom read is prevented and both queries in session 1 will display same number of rows.
- To compare same scenario with "Repeatable Read" read **Repeatable Read Example 2**. **Serializable**

Example 2

- Assume table has primary key on column "ID".
- In our example script, primary key is not added. Add primary key on column Emp.ID before executing below examples.

Session 1

set transaction isolation level serializable begin tran

select * from emp where ID between 1 and 3 waitfor

delay '00:00:15'

select * from Emp where ID between 1 and 3 rollback

Session 2

insert into Emp(ID,Name,Salary) values (11,'Stewart',11000)

- Run both sessions side by side.

Output

- Since Session 1 is filtering IDs between 1 and 3, only those records whose IDs range between 1 and 3 will be locked and these records cannot be modified and no new records with ID range between 1 to 3 will be inserted.
- In this example, new record with ID=11 will be inserted in Session 2 without any delay. Snapshot
- Snapshot isolation is similar to Serializable isolation.
- The difference is Snapshot does not hold lock on table during the transaction so table can be modified in other sessions.
- Snapshot isolation maintains versioning in Tempdb for old data in case of any data modification occurs in other sessions then existing transaction displays the old data from Tempdb.

Snapshot Example 1

Session 1

settransactionisolationlevelsnapshot begin tran

select * from Emp waitfor

delay '00:00:15' select * from

Emp rollback

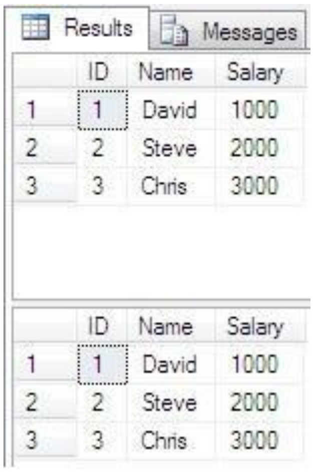
Session 2

```
insert into Emp(ID,Name,Salary) values ( 11,'Stewart',11000) update Emp
set Salary=4444 where ID=4
select * from Emp
```

Run both sessions side by side.

Output

ResultinSession1.

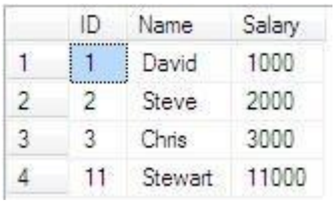


Results | Messages

	ID	Name	Salary
1	1	David	1000
2	2	Steve	2000
3	3	Chris	3000

	ID	Name	Salary
1	1	David	1000
2	2	Steve	2000
3	3	Chris	3000

ResultinSession2.



	ID	Name	Salary
1	1	David	1000
2	2	Steve	2000
3	3	Chris	3000
4	11	Stewart	11000

- Session 2 queries will be executed in parallel as transaction in session 1 won't lock the table Emp.

Unit – II

Database Design

Entity-Relationship Model - E-R Diagrams - Enhanced-ER Model - ER-to-Relational Mapping - Functional Dependencies - Non-loss Decomposition - First, Second, Third Normal Forms, Dependency Preservation - Boyce / Codd Normal Form - Multi-valued Dependencies and Fourth Normal Form - Join Dependencies and Fifth Normal Form.

Entity-Relationship Model

Explain in detail about Entity Relationship Model.

- The entity-relationship (E-R) data model uses a collection of basic objects, called *entities* and *relationships* among these objects.
- An entity is a –thing‖ or –object‖ in the real world that is distinguishable from other objects.
- For example, each person is an entity and bank accounts can be considered as entities.
- The E-R models employ the basic concepts such as,
    - ✓ Entity Sets
    - ✓ Relationship Sets
    - ✓ Attributes

Entity Set

- The set of all entities of the same type are termed as an **entity set.**
- Entities are described in a database by a set of **attributes**.
- The extra attribute *ID* is used to identify an instructor uniquely.

Types of Entity Set

- **Strong Entity Set**
    - ✓ An entity set that has a primary key is called strong entity set.
- **Weak Entity Set**
    - ✓ An entity set that does not have a primary key is called weak entity set.
- **Identifying Or Owner Entity set**
    - ✓ Weak entity set must be associated with another entity set called identifying or owner entity set.
    - ✓ The weak entity set is said to be existence dependent on identifying entity set.

Relationship Set

- A **relationship** is an association among several entities.
- The set of all relationships of the same type are termed as a **relationship set**.

Attributes

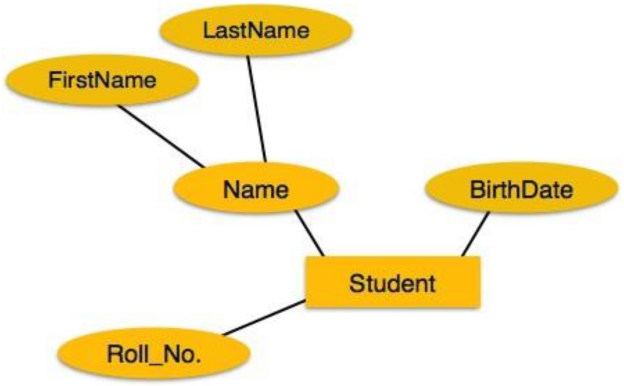
- An attribute of an entity set is a function that maps from the entity set into a domain.
- For each attributes, there is a set of permitted value set of that attribute.

Types of Attributes

- Simple and Composite
- Single Value and Multi-valued
- Derived
- Descriptive

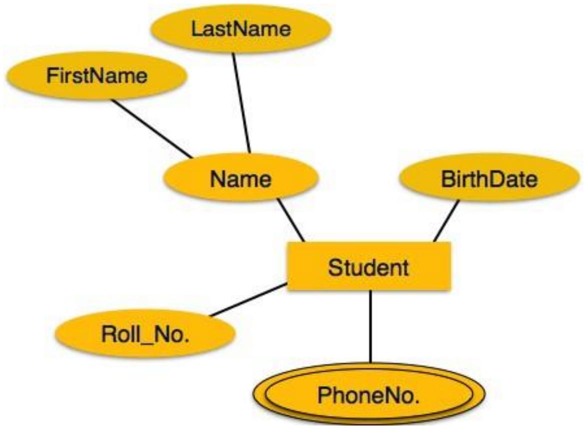
Simple & Composite Attributes

- **Simple Attribute**
  - ✓ Attributes that cannot be divided into subparts are called simple attributes.
  - ✓ Ex: S.no is a simple attributes.
- **Composite Attribute**
  - ✓ Attribute that can be divided into subparts is called as composite attributes.



Single Valued & Multi-valued Attribute

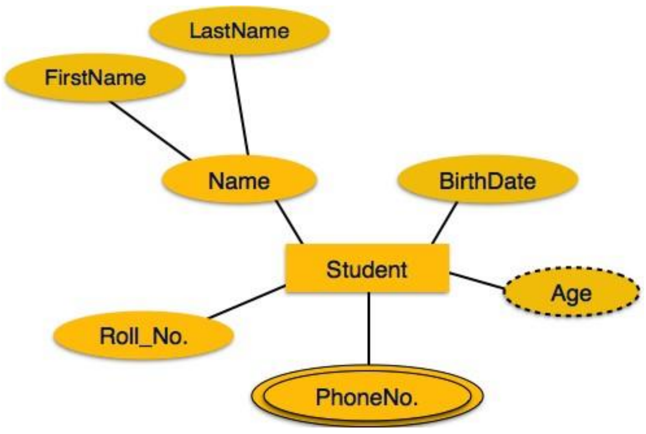
- **Single Valued Attribute**
  - ✓ The attribute that have single value for a particular entity is called single valued attribute.
  - ✓ Ex: student_id – refers to only one student.
- **Multi-valued Attributes**
  - ✓ The attribute that has a set of values for a specific entity is called multi-valued attributes.
  - ✓ Ex: phone_number



Derived Attribute

- The value of this type of attribute can be derived from the values of other related attributes or entities.

Example:



Descriptive Attribute

- The attribute present in the relationship is called as descriptive attribute.

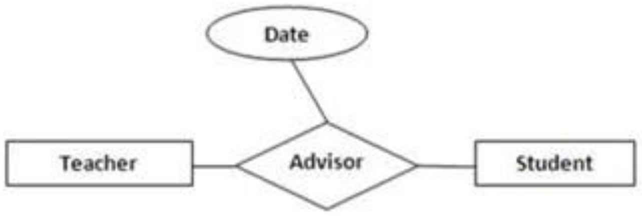


Figure: ER Diagram with Descriptive Attribute

- **NULL Value:**
  - ✓ The attribute takes a NULL value, when an entity does not have a value for it.

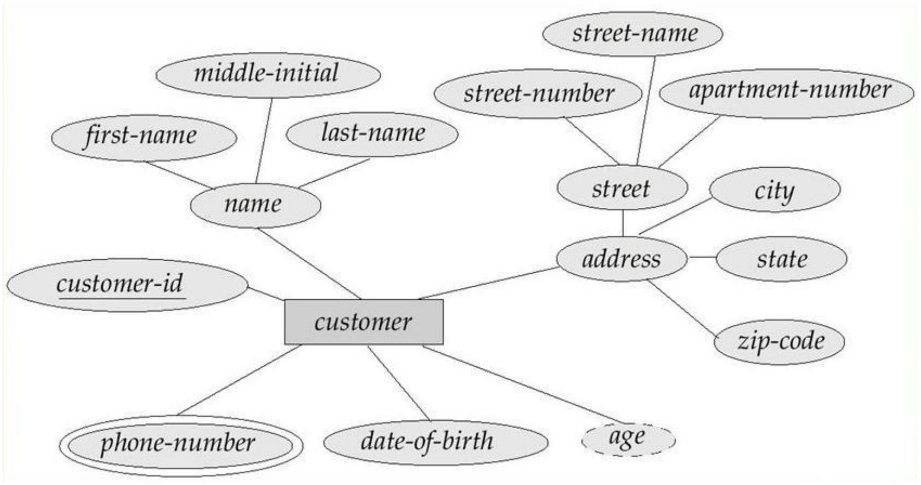


Figure: ER diagram with Composite, Multi-valued and Derived Attributes

In the above ER diagram,

- ✓ C_name & address - Composite Attribute
- ✓ Street - Component Attribute
- ✓ Phone-number - Multi-valued Attribute
- ✓ Age - Derived Attribute

Constraints

Explain in detail about constraints.

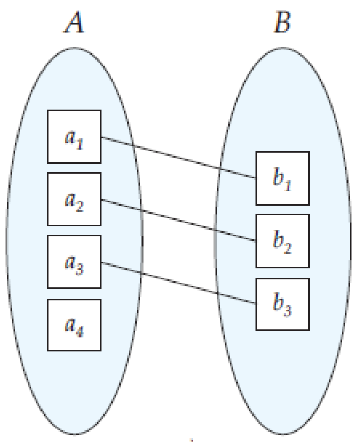
- An E-R enterprise schema may define certain constraints to which the contents of a database must conform.
- That limits the possible combinations of entities that may participate in the corresponding relationship set.
  - ✓ Mapping Cardinalities (Cardinality Constraints)
  - ✓ Participation Constraints

Mapping Cardinalities (May/June 2013)

- **Mapping cardinalities**, or cardinality ratios, express the number of entities to which another entity can be associated via a relationship set.
- Mapping cardinalities are most useful in describing binary relationship sets, although they can contribute to the description of relationship sets that involve more than two entity sets.
- A mapping cardinality is a data constraint that specifies how many entities an entity can be related to in a relationship set.
- Consider a binary relationship set R between entity sets A and B. There are four possible mapping cardinalities in this case:
- Types of mapping cardinalities are,
  - ✓ One to One
  - ✓ One to Many
  - ✓ Many to One
  - ✓ Many to Many

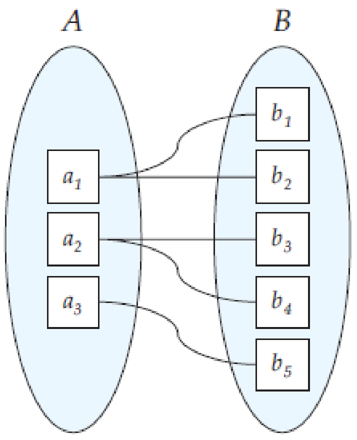
One-to-One

- An entity in *A* is associated (related) with *at most* one entity in *B*, and an entity in *B* is associated with *at most* one entity in *A*.



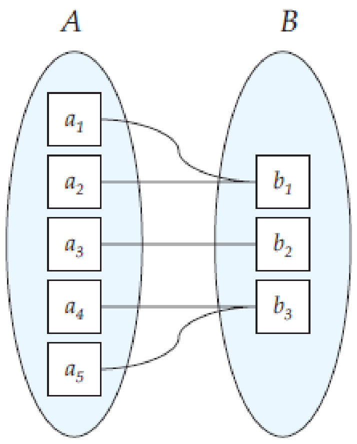
One-to-Many

- An entity in *A* is associated with any number (zero or more) of entities in *B*.
- And an entity in *B*, can be associated with *at most* one entity in *A*.



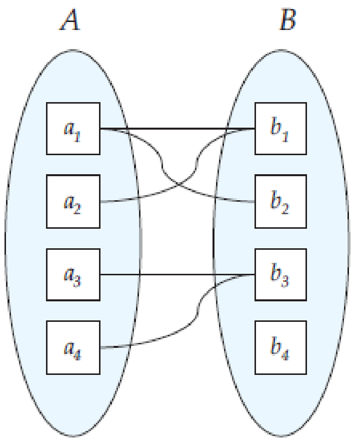
Many-to-One

- An entity in *A* is associated with *at most* one entity in *B*.
- And an entity in *B*, can be associated with any number (zero or more) of entities in *A*.



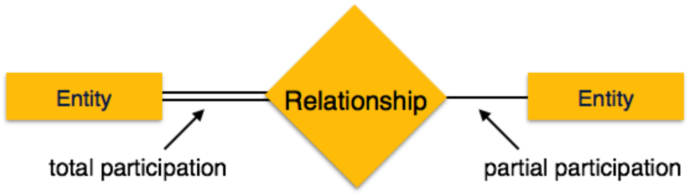
Many-to-Many

- An entity in *A* is associated with any number (zero or more) of entities in *B*, and an entity in *B* is associated with any number (zero or more) of entities in *A*.



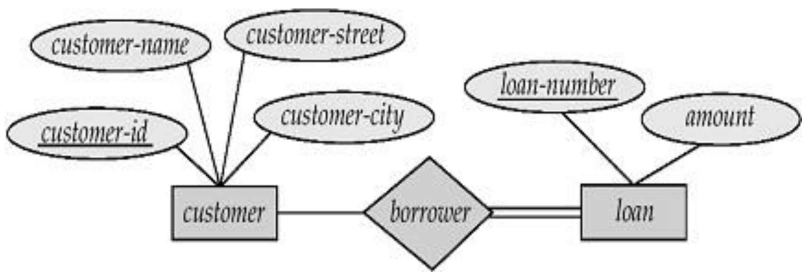
<u>Participation Constraints</u>

- It specifies whether the existence of an entity depends on its being related to another entity via the relationship type.



Total Participation

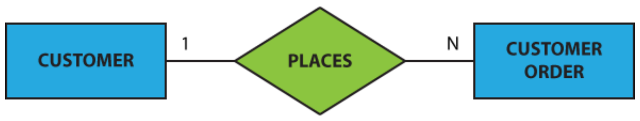
- The participation of an entity set E in a relationship set R is said to be total, if every entity in E participates in at least one relationship in R.



- Double line indicates that from loan to borrower, each loan must have at least one associated customer.

Partial Participation

- If only some entities in E participate in relationship in R, then the participation of entity E in relationship R is said to be partial.



Weak Entity Set

- Entity types that do not have key attributes of their own are called weak entity types.
- A weak entity type always has a total participation constraint (existence dependency) with respect to its identifying relationship, because a weak entity cannot be identified without an owner entity type.
- A weak entity set is indicated in E-R diagrams by a doubly outlined rectangular box and the corresponding identifying relationship by a doubly outlined diamond.

Strong Entity Set

- Entity types that have key attributes of their own are called strong entity types.
- A strong entity set is indicated in E-R diagrams by rectangular box and its relationship by a diamond.

<u>Diagrams</u>

- E-R diagram is used to express the overall logical structure (schema) of a database graphically by an *entity-relationship (E-R) diagram*.
- E-R diagrams are simple and clear.

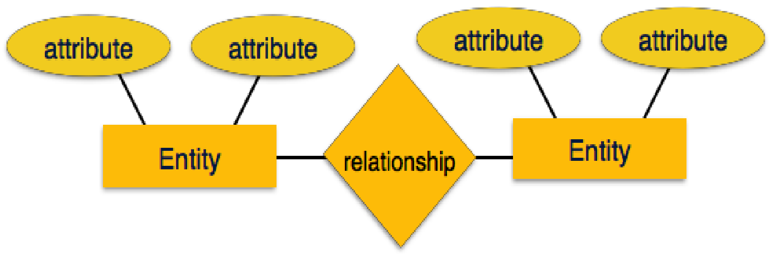
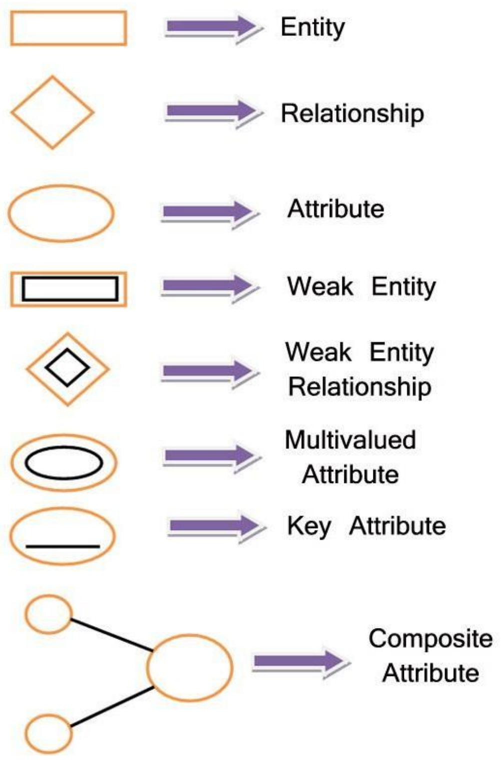


Figure: Sample ER Diagram

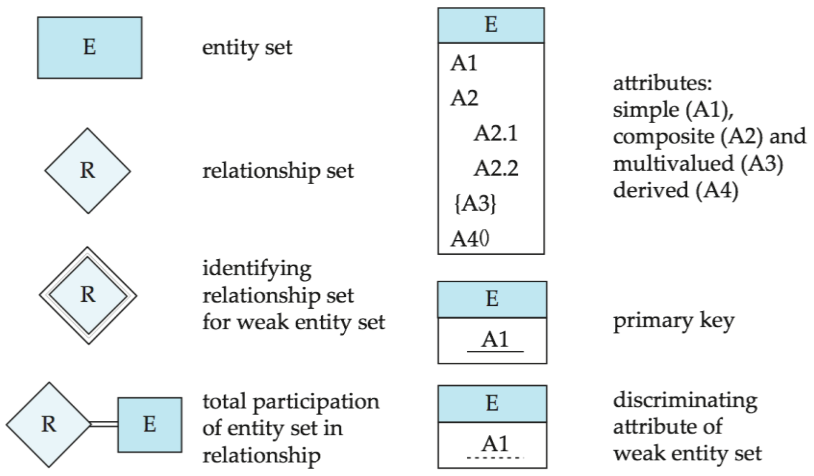
<u>ER Diagram Representation Symbols</u>

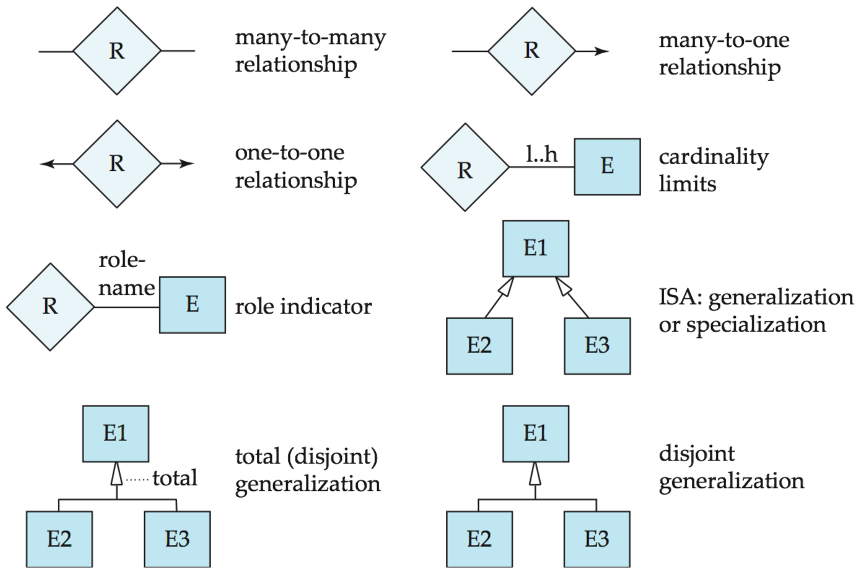
Write short notes on ER diagram representation symbols.



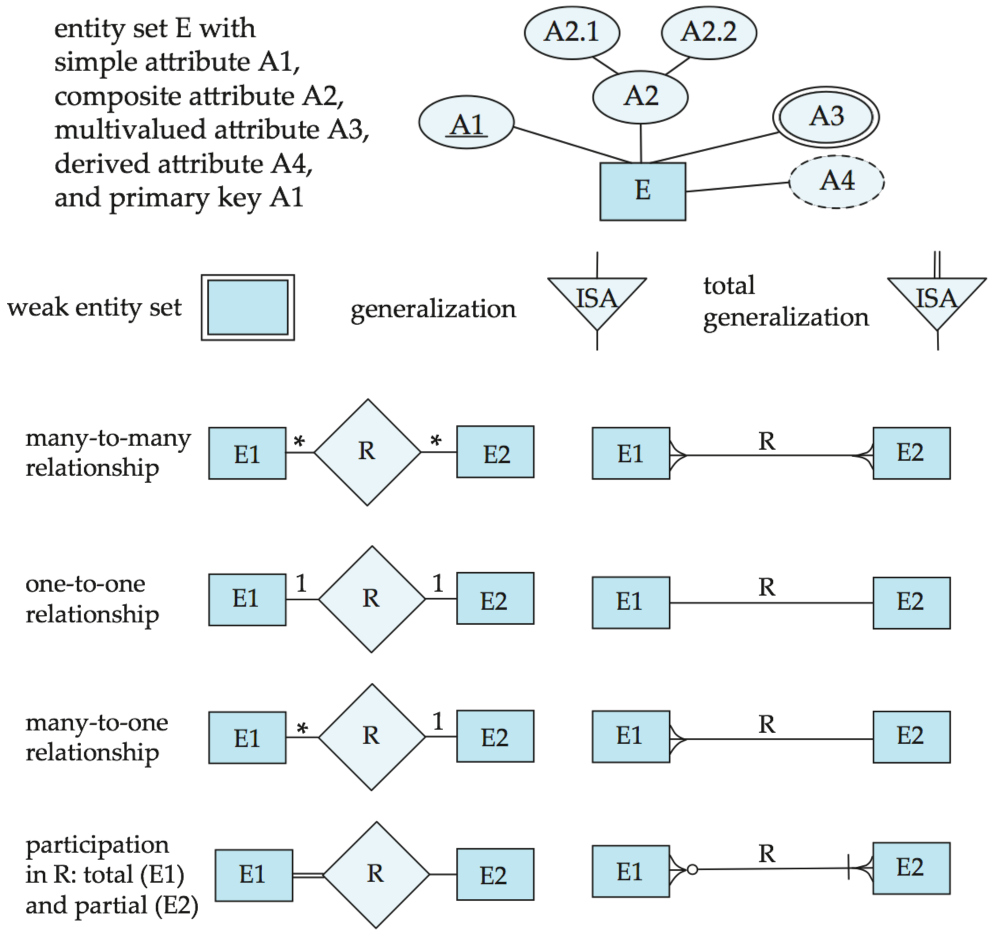
(Or)

Symbols Used in E-R Notation


E
entity set
R
relationship set
R
identifying relationship set for weak entity set
R
E
total participation of entity set in relationship
E
A1
A2
A2.1
A2.2
{A3}
A4()
attributes: simple (A1), composite (A2) and multivalued (A3) derived (A4)
E
A1
primary key
E
A1
discriminating attribute of weak entity set



Alternative ER Notations


entity set E with simple attribute A1, composite attribute A2, multivalued attribute A3, derived attribute A4, and primary key A1
A2.1
A2.2
A2
A1
A3
E
A4
weak entity set
generalization
ISA
total generalization
ISA
many-to-many relationship
E1
*
R
*
E2
E1
R
E2
one-to-one relationship
E1
1
R
1
E2
E1
R
E2
many-to-one relationship
E1
*
R
1
E2
E1
R
E2
participation in R: total (E1) and partial (E2)
E1
R
E2
E1
R
E2

Basic Structure

An E-R diagram consists of the following major components:

- **Rectangles** divided into **two parts** represent entity sets.
- The first part, which in this textbook is shaded blue, contains the name of the entity set.
- The second part contains the names of all the attributes of the entity set.

Rectangle	-	to represent an entity set
Ellipses	-	to represent an attribute
Diamonds	-	to represent relationship sets

- Lines - to link attributes to entity sets and entity sets to relationship sets
- Double ellipses - to represent multi-valued attributes
- Dashed ellipses - to represent derived attributes
- Double rectangle - to represent weak entity sets
- Double line - to represent total participation of an entity in a relationship set.
- **Double diamonds** - represent identifying relationship sets linked to weak entity sets
- **Undivided rectangles** - represent the attributes of a relationship set. Attributes that are part of the primary key are underlined.

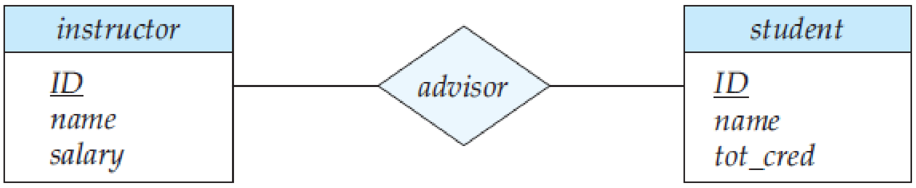


Figure 1: E-R Diagram corresponding to Instructors and Students.

- Consider the E-R diagram in Figure 1, which consists of two entity sets, *instructor* and *student* related through a binary relationship set *advisor*.
- The attributes associated with *instructor* are *ID, name*, and *salary*.
- The attributes associated with *student* are *ID, name*, and *tot cred*. In Figure 1, attributes of an entity set that are members of the primary key are underlined.
- If a relationship set has some attributes associated with it, then we enclose the attributes in a rectangle and link the rectangle with a dashed line to the diamond representing that relationship set.

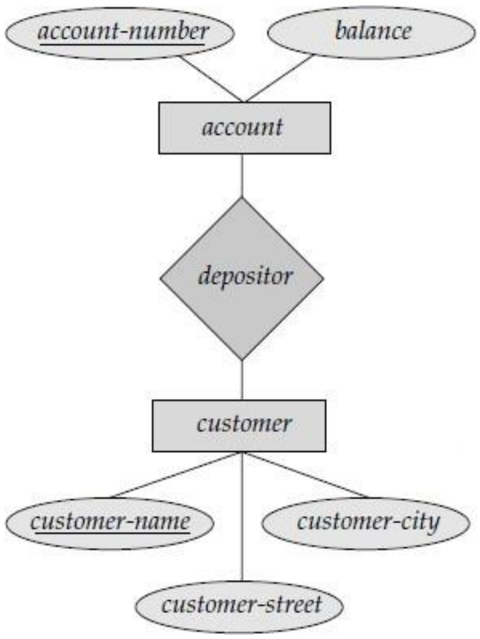


Figure 2: An ER Diagram for Banking Enterprise

- Entity sets are represented by a rectangular box with the entity set name in the header and the attributes listed below it.
- Relationship sets are represented by a diamond connecting a pair of related entity sets.
- The name of the relationship is placed inside the diamond.

- The above E-R diagram indicates that there are two entity sets, *instructor* and *department*, with attributes as outlined earlier.
- The diagram also shows a relationship *member* between *instructor* and *department*.
- In addition to entities and relationships, the E-R model represents certain constraints to which the contents of a database must conform.
- One important constraint is **mapping cardinalities**, which express the number of entities to which another entity can be associated via a relationship set.

<u>Enhanced-ER Model</u>

Discuss enhanced ER model in detail.

- **Enhanced entity-relationship** models, also known as extended **entity-relationship models**, are advanced database **diagrams** very similar to regular **ER diagrams**.
- **Enhanced** ERDs are high-level **models** that represent the requirements and complexities of complex databases.
- It is a diagrammatic technique for displaying the Sub Class and Super Class; Specialization and Generalization; Union or Category; Aggregation etc.

Features of EER Model

- EER creates a design more accurate to database schemas.
- It reflects the data properties and constraints more precisely.
- It includes all modeling concepts of the ER model.
- Diagrammatic technique helps for displaying the EER schema.
- It includes the concept of specialization and generalization.
- It is used to represent a collection of objects that is union of objects of different of different entity types.

A. Sub Class and Super Class

- Sub class and Super class relationship leads the concept of Inheritance.
- The relationship between sub class and super class is denoted with ⓓ symbol.

1. Super Class

- Super class is an entity type that has a relationship with one or more subtypes.
- An entity cannot exist in database merely by being member of any super class.
- **For example:** Shape super class is having sub groups as Square, Circle, Triangle.

2. Sub Class

- Sub class is a group of entities with unique attributes.
- Sub class inherits properties and attributes from its super class.
- **For example:** Square, Circle, Triangle are the sub class of Shape super class.

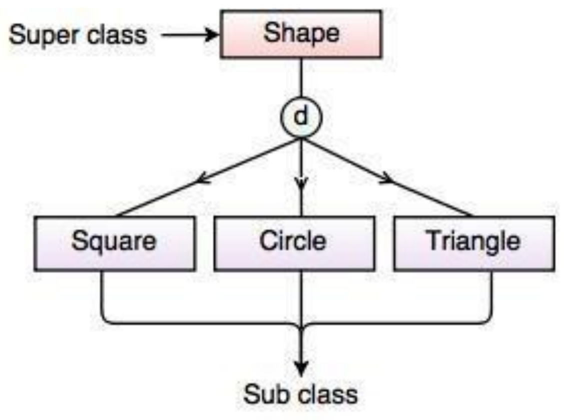


Fig. Super class/Sub class Relationship

B. Specialization and Generalization

1. Generalization

- Generalization is the process of generalizing the entities which contain the properties of all the generalized entities.
- It is a bottom approach, in which two lower level entities combine to form a higher level entity.
- Generalization is the reverse process of Specialization.
- It defines a general entity type from a set of specialized entity type.
- It minimizes the difference between the entities by identifying the common features. For example:

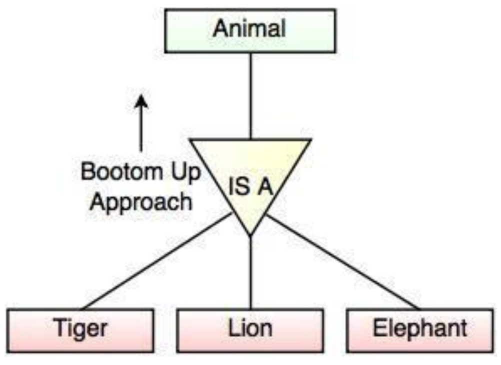


Fig. Generalization

- In the above example, Tiger, Lion, Elephant can all be generalized as Animals.

2. Specialization

- Specialization is a process that defines a group entity which is divided into sub groups based on their characteristic.
- It is a top down approach, in which one higher entity can be broken down into two lower level entities.
- It maximizes the difference between the members of an entity by identifying the unique characteristic or attributes of each member.
- It defines one or more sub class for the super class and also forms the superclass/subclass relationship.

For example

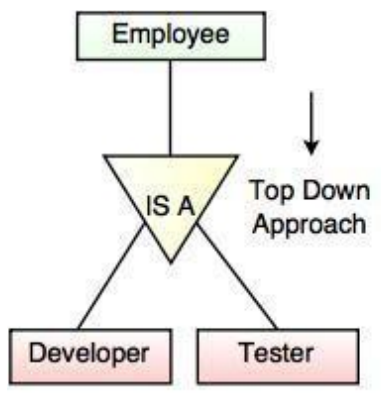


Fig. Specialization

In the above example, Employee can be specialized as Developer or Tester, based on what role they play in an Organization.

C. Category or Union

- Category represents a single super class or sub class relationship with more than one super class.
- It can be a total or partial participation.
- **For example** Car booking, Car owner can be a person, a bank (holds a possession on a Car) or a company.
- Category (sub class) → Owner is a subset of the union of the three super classes → Company, Bank and Person.
- A Category member must exist in at least one of its super classes.

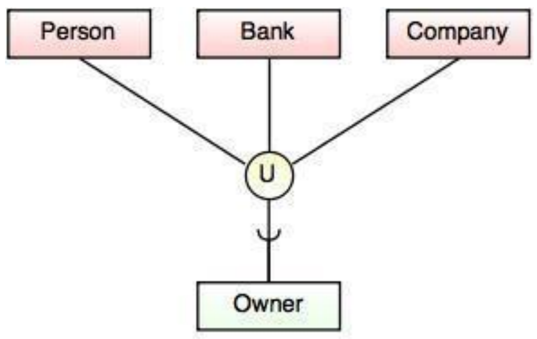


Fig. Categories (Union Type)

D. Aggregation

- Aggregation is a process that represents a relationship between a whole object and its component parts.
- It abstracts a relationship between objects and viewing the relationship as an object.
- It is a process when two entities are treated as a single entity.

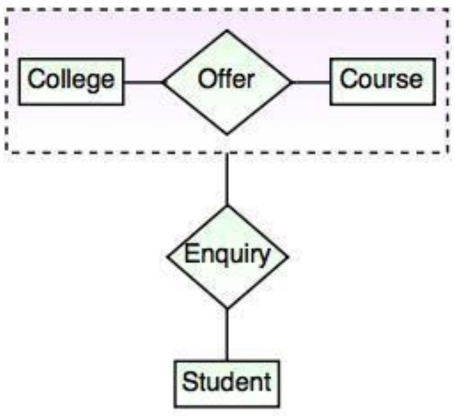


Fig. Aggregation

- In the above example, the relation between College and Course is acting as an Entity in Relation with Student.

Disjointness / Overlap Constraint

- Specifies that the subclass of the specialization must be disjoint, which means that an entity can be a member of, at most, one subclass of the specialization.
- The d in the specialization circle stands for disjoint.
- If the subclasses are not constrained to be disjoint, they overlap.
- Overlap means that an entity can be a member of more than one subclass of the specialization.
- Overlap constraint is shown by placing an o in the specialization circle.

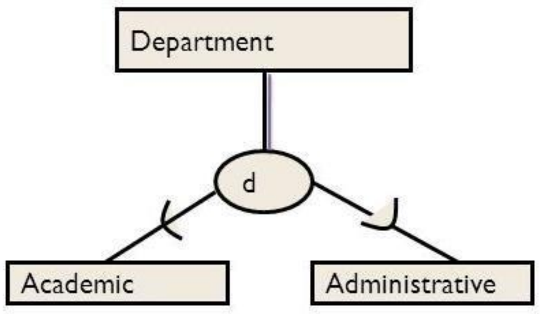
Completeness Constraint

- The completeness constraint may be either total or partial.
- A total specialization constraint specifies that every entity in the superclass must be a member of at least one subclass of the specialization.
- Total specialization is shown by using a double line to connect the super class to the circle.
- A single line is used to display a partial specialization, meaning that an entity does not have to belong to any of the subclasses.

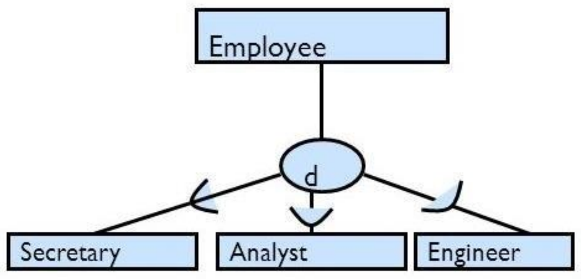
Disjointness vs. Completeness

- Disjoint constraints and completeness constraints are independent. The following possible constraints on specializations are possible:

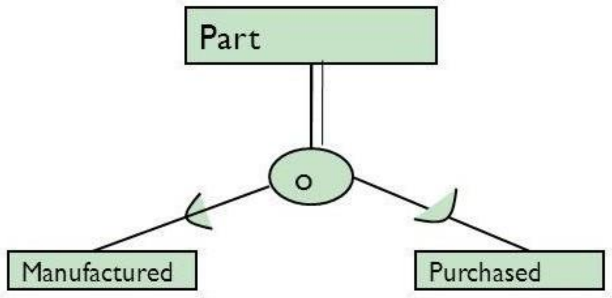
a) Disjoint, total



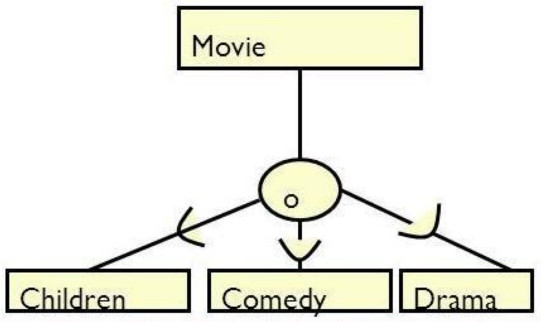
b) Disjoint, partial



c) Overlapping, total



d) Overlapping, partial



<u>ER-to-Relational Mapping</u>

Describe ER to Relational mapping in detail.

- ER diagrams can be mapped to relational schema, that is, it is possible to create relational schema using ER diagram.
- We cannot import all the ER constraints into relational model, but an approximate schema can be generated.
- There are several processes and algorithms available to convert ER Diagrams into Relational Schema. Some of them are automated and some of them are manual.

ER-to-Relational Mapping Algorithm

- Let us use the COMPANY database example to illustrate the mapping procedure.

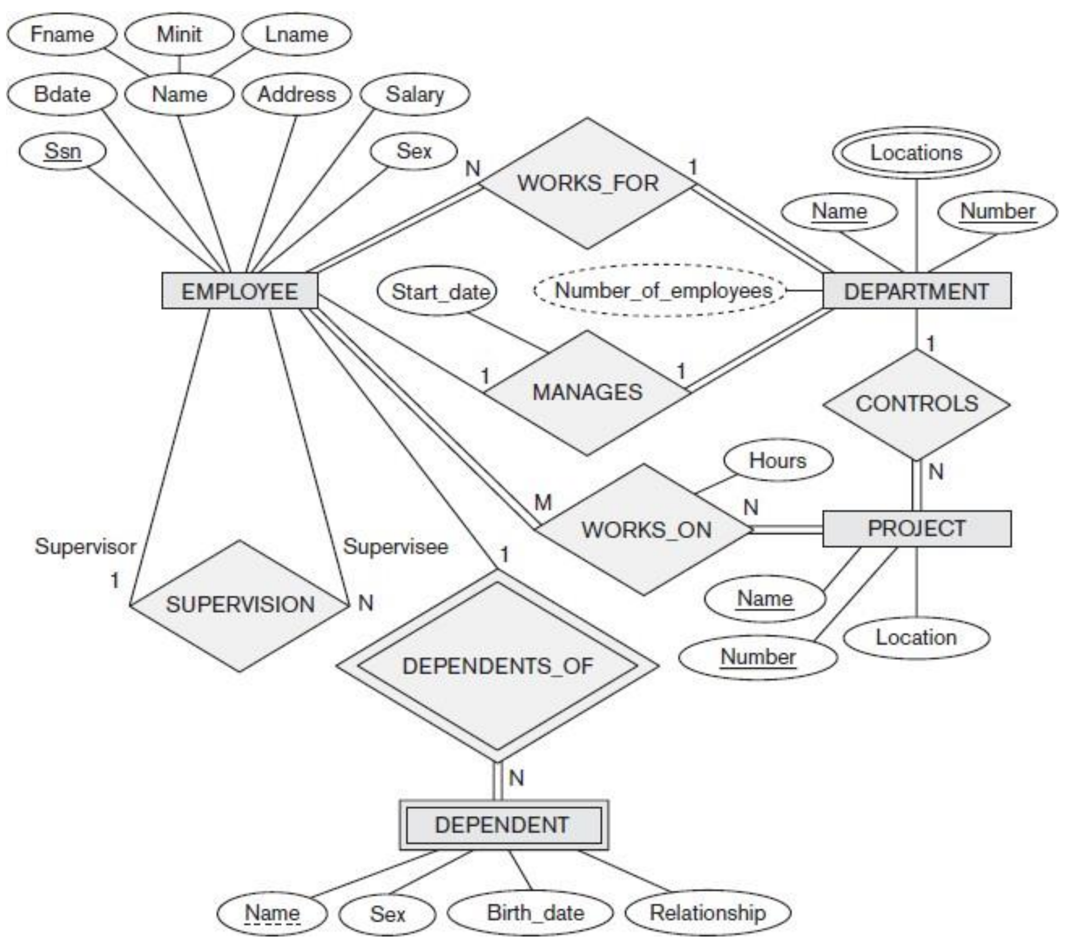


Figure 1: ER conceptual schema diagram for the COMPANY database

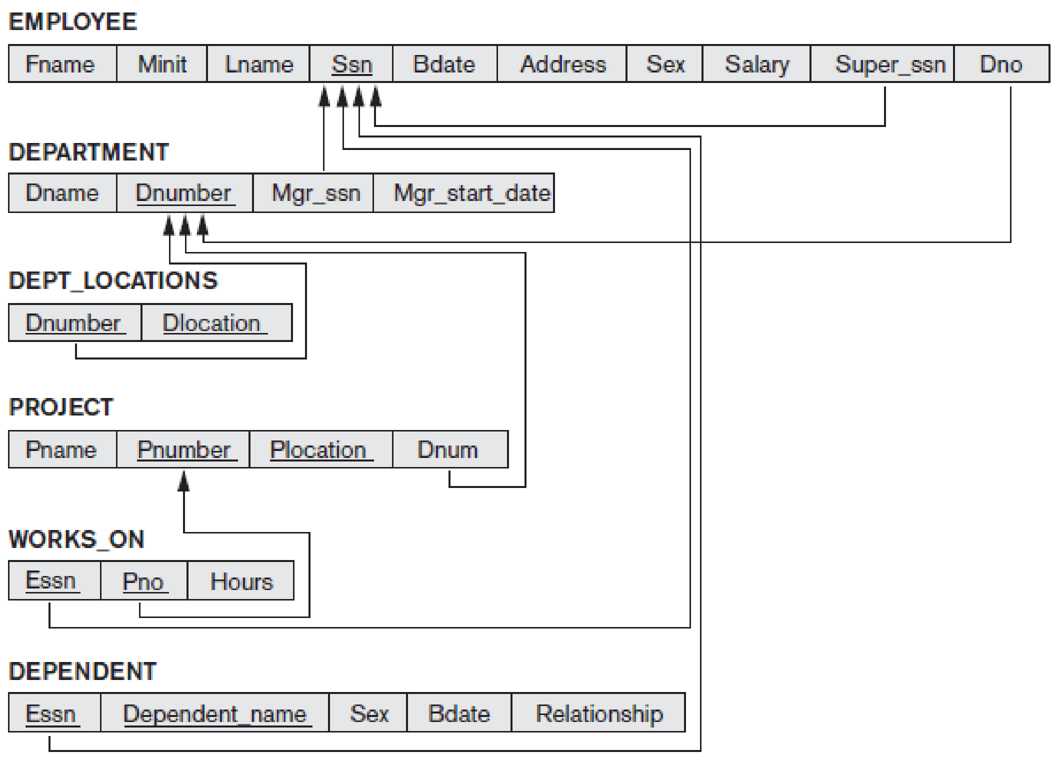


Figure 2: Result of mapping the COMPANY ER schema into a relational database schema

- The COMPANY ER schema is shown again in Figure 1, and the corresponding COMPANY relational database schema is shown in Figure 2 to illustrate the mapping steps.

Step 1: Mapping of Regular Entity Types

- For each regular (strong) entity type *E* in the ER schema, create a relation *R* that includes all the simple attributes of *E*.
- Include only the simple component attributes of a composite attribute. Choose one of the key attributes of *E* as the primary key for *R*.
- If the chosen key of *E* is a composite, then the set of simple attributes that form it will together form the primary key of *R*.
- If multiple keys were identified for *E* during the conceptual design, the information describing the attributes that form each additional key is kept in order to specify secondary (unique) keys of relation *R*.

Step 2: Mapping of Weak Entity Types

- For each weak entity type *W* in the ER schema with owner entity type *E*, create a relation *R* and include all simple attributes (or simple components of composite attributes) of *W* as attributes of *R*.
- In addition, include as foreign key attributes of *R*, the primary key attribute(s) of the relation(s) that correspond to the owner entity type(s); this takes care of mapping the identifying relationship type of *W*.
- The primary key of *R* is the combination of the primary key(s) of the owner(s) and the partial key of the weak entity type *W*, if any.
- If there is a weak entity type *E*2 whose owner is also a weak entity type *E*1, then *E*1 should be mapped before *E*2 to determine its primary key first.

Step 3: Mapping of Binary 1:1 Relationship Types

- For each binary 1:1 relationship type *R* in the ER schema, identify the relations *S* and *T* that correspond to the entity types participating in *R*.
- There are three possible approaches:
  - ✓ The foreign key approach
  - ✓ The merged relationship approach
  - ✓ The cross-reference or relationship relation approach

Foreign Key Approach:

- Choose one of the relations—*S*, say—and include as a foreign key in *S* the primary key of *T*.
- It is better to choose an entity type with *total participation* in *R* in the role of *S*.
- Include all the simple attributes (or simple components of composite attributes) of the 1:1 relationship type *R* as attributes of *S*.

Merged Relation Approach:

- An alternative mapping of a 1:1 relationship type is to merge the two entity types and the relationship into a single relation.
- This is possible when *both participations are total*, as this would indicate that the two tables will have the exact same number of tuples at all times.

Cross-reference or relationship relation approach:

- The third option is to set up a third relation *R* for the purpose of cross-referencing the primary keys of the two relations *S* and *T* representing the entity types.
- As we will see, this approach is required for binary M:N relationships.
- The relation *R* is called a **relationship relation** (or sometimes a **lookup table**), because each tuple in *R* represents a relationship instance that relates one tuple from *S* with one tuple from *T*.
- The relation *R* will include the primary key attributes of *S* and *T* as foreign keys to *S* and *T*.
- The primary key of *R* will be one of the two foreign keys, and the other foreign key will be a unique key of *R*.

Step 4: Mapping of Binary 1:N Relationship Types

- For each regular binary 1:N relationship type *R*, identify the relation *S* that represents the participating entity type at the *N-side* of the relationship type.
- Include as foreign key in *S* the primary key of the relation *T* that represents the other entity type participating in *R*; we do this because each entity instance on the N-side is related to at most one entity instance on the 1-side of the relationship type.
- Include any simple attributes (or simple components of composite attributes) of the 1:N relationship type as attributes of *S*.

Step 5: Mapping of Binary M:N Relationship Types

- For each binary M:N relationship type *R*, create a new relation *S* to represent *R*.
- Include as foreign key attributes in *S* the primary keys of the relations that represent the participating entity types; their *combination* will form the primary key of *S*.Also include any simple attributes of the M:N relationship type (or simple components of composite attributes) as attributes of *S*.
- Notice that we cannot represent an M:N relationship type by a single foreign key attribute in one of the participating relations (as we did for 1:1 or 1:N relationship types) because of the M:N cardinality ratio; we must create a separate *relationship relation S*.

Step 6: Mapping of Multi-valued Attributes

- For each multi-valued attribute *A*, create a new relation *R*.
- This relation *R* will include an attribute corresponding to *A*, plus the primary key attribute *K*—as a foreign key in *R*—of the relation that represents the entity type or relationship type that has *A* as a multi-valued attribute.
- The primary key of *R* is the combination of *A* and *K*. If the multi-valued attribute is composite, we include its simple components.

Step 7: Mapping of *N*-ary Relationship Types

- For each *n*-ary relationship type *R*, where *n* > 2, create a new relation *S* to represent *R*.
- Include as foreign key attributes in *S* the primary keys of the relations that represent the participating entity types.

- Also include any simple attributes of the *n*-ary relationship type (or Step 7: Mapping of N-ary Relationship Types)

Step 8: Options for Mapping Specialization or Generalization

- Convert each specialization with *m* subclasses {*S*1, *S*2, ..., *Sm*} and (generalized) superclass *C*, where the attributes of *C* are {*k*, *a*1, ...*an*} and *k* is the (primary) key, into relation schemas using one of the following options:
- **Option 8A: Multiple relations—superclass and subclasses**
  - ✓ Create a relation *L* for *C* with attributes Attrs(*L*) = {*k*, *a*1, ..., *an*} and PK(*L*) = *k*. Create a relation *Li* for each subclass *Si*, 1 ≤*i* ≤*m*, with the attributes Attrs(*Li*) = {*k*} ∪ {attributes of *Si*} and PK(*Li*) = *k*.
  - ✓ This option works for any specialization (total or partial, disjoint or overlapping).
- **Option 8B: Multiple relations—subclass relations only**
  - ✓ Create a relation *Li* for each subclass *Si*, 1 ≤*i* ≤*m*, with the attributes Attrs(*Li*) = {attributes of *Si*} ∪ {*k*, *a*1, ..., *an*} and PK(*Li*) = *k*.
  - ✓ This option only works for a specialization whose subclasses are *total* (every entity in the superclass must belong to (at least) one of the subclasses).
  - ✓ Additionally, it is only recommended if the specialization has the *disjointedness constraint*. If the specialization is *overlapping*, the same entity may be duplicated in several relations.
- **Option 8C: Single relation with one type attribute.**
  - ✓ Create a single relation *L* with attributes Attrs(*L*) = {*k*, *a*1, ..., *an*} ∪ {attributes of *S*1} ∪ ... ∪ {attributes of *Sm*} ∪ {*t*} and PK(*L*) = *k*.
  - ✓ The attribute *t* is called a **type** (or) **discriminating**) attribute whose value indicates the subclass to which each tuple belongs, if any.
  - ✓ This option works only for a specialization whose subclasses are *disjoint*, and has the potential for generating many NULL values if many specific attributes exist in the subclasses.
- **Option 8D: Single relation with multiple type attributes.**
  - ✓ Create a single relation schema *L* with attributes Attrs(*L*) = {*k*, *a*1, ..., *an*} ∪ {attributes of *S*1} ∪ ... ∪ {attributes of *Sm*} ∪ {*t*1, *t*2, ..., *tm*} and PK(*L*) = *k*.
  - ✓ Each *ti*, 1 ≤ *i* ≤ *m*, is a **Boolean type attribute** indicating whether a tuple belongs to subclass *Si*.
  - ✓ This option is used for a specialization whose subclasses are *overlapping* (but will also work for a disjoint specialization).

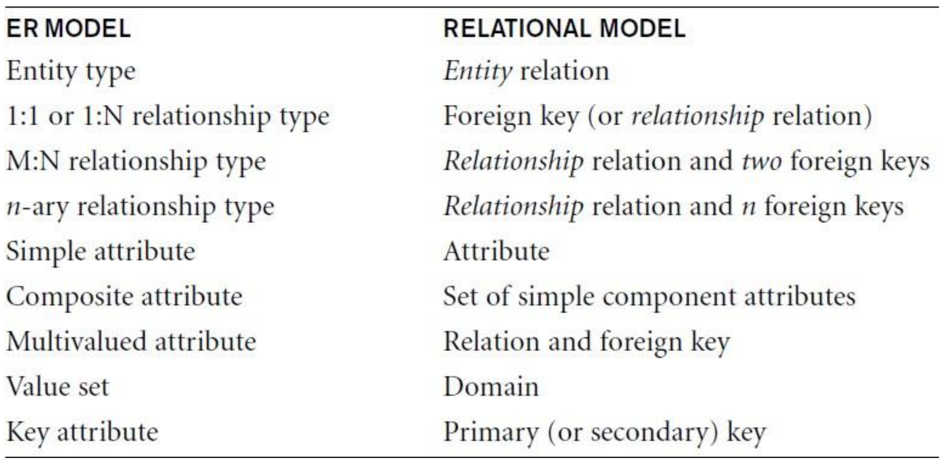
Step 9: Mapping of Shared Subclasses (Multiple Inheritance)

- A shared subclass is a subclass of several superclasses, indicating multiple inheritance.
- These classes must all have the same key attribute; otherwise, the shared subclass would be modeled as a category (union type) as we discussed in Section 8.4.
- We can apply any of the options discussed in step 8 to a shared subclass, subject to the restrictions discussed in step 8 of the mapping algorithm.

Step 10: Mapping of Union Types (Categories)

- For mapping a category whose defining superclasses have different keys, it is customary to specify a new key attribute, called a **surrogate key**, when creating a relation to correspond to the category.
- The keys of the defining classes are different, so we cannot use any one of them exclusively to identify all entities in the category.

Correspondence between ER and Relational Models



ER MODEL	RELATIONAL MODEL
Entity type	*Entity* relation
1:1 or 1:N relationship type	Foreign key (or *relationship* relation)
M:N relationship type	*Relationship* relation and *two* foreign keys
n-ary relationship type	*Relationship* relation and *n* foreign keys
Simple attribute	Attribute
Composite attribute	Set of simple component attributes
Multivalued attribute	Relation and foreign key
Value set	Domain
Key attribute	Primary (or secondary) key

Functional Dependencies

What is Functional Dependency? (Or) Explain the concept of functional dependency in detail.

- **Functional Dependency** is a relationship that exists between multiple attributes of a relation.
- This concept is given by **E. F.Codd.**
- Functional dependency represents a formalism on the infrastructure of relation.
- It is a type of constraint existing between various attributes of a relation.
- It is used to define various normal forms.
- These dependencies are restrictions imposed on the data in database.
- If P is a relation with A and B attributes, a functional dependency between these two attributes is represented as $\{A \rightarrow B\}$.
- It specifies that,

A	It is a determinant set.
B	It is a dependent attribute.
$\{A \rightarrow B\}$	A functionally determines B. B is a functionally dependent on A.

- Each value of A is associated precisely with one B value. A functional dependency is trivial if B is a subset of A.
- 'A' Functionality determines 'B' $\{A \rightarrow B\}$ (Left hand side attributes determine the values of Right hand side attributes).

(Or)

- Functional dependency (FD) is a set of constraints between two attributes in a relation. Functional dependency says that if two tuples have same values for attributes A1, A2,..., An, then those two tuples must have to have same values for attributes B1, B2, ..., Bn.
- Functional dependency is represented by an arrow sign (→) that is, X→Y, where X functionally determines Y. The left-hand side attributes determine the values of attributes on the right-hand side.

Design goals

- Avoid redundant data.
- Ensure that relationships among attributes are represented.
- Facilitate the checking of updates for violation of database integrity constraints.

For example: <Employee> Table

EmpId	EmpName

- In the above <Employee> table, EmpName (employee name) is functionally dependent on EmpId (employee id) because the EmpId is unique for individual names.
- The EmpId identifies the employee specifically, but EmpName cannot distinguish the EmpId because more than one employee could have the same name.
- The functional dependency between attributes eliminates the repetition of information.
- It is related to a candidate key, which uniquely identifies a tuple and determines the value of all other attributes in the relation.

Types of Dependency

- ***Full Functional Dependencies***
  - ✓ In a relation R, X and Y are attributes. X functionally determines Y ($X \to Y$). subset of X should not functionally determine Y

    Student_no → Marks ← course_no

    In the above example marks is fully functionally dependent on student_no and course_no together and not on subset of student_no, course_no.
- ***Partial Dependencies***
  - ✓ Attribute Y is partially dependent on attribute X only if it is dependent on a subset of attribute X.

    For example course_name, Instructor_name are partially dependent on composite attributes (student_no, course_no} beacuase course_no only defines course_name, Instructor_name.
- The main characteristics of functional dependencies
  - ✓ Have a one-to-one relationship between attributes on the left-and right-hand side of a dependency is nontrivial.
  - ✓ A dependency is trivial if and only if, the right-hand side is a subset of the left side.

<u>Properties of Functional Dependencies (Or) Inference rules for Functional Dependencies</u>

Armstrong's Axioms

- Armstrong's Axioms is a set of rules.
- It provides a simple technique for reasoning about functional dependencies.
- It was developed by William W. Armstrong in 1974.
- It is used to infer all the functional dependencies on a relational database.
- If F is a set of functional dependencies then the closure of F, denoted as F^+, is the set of all functional dependencies logically implied by F.
- Armstrong's Axioms are a set of rules, that when applied repeatedly, generates a closure of functional dependencies.

Various Axioms Rules

A. Primary Rules

Rule 1	**Reflexivity Rule** If A is a set of attributes and B is a subset of A, then A holds B. { A → B }
Rule 2	**Augmentation Rule** If A hold B and C is a set of attributes, then AC holds BC. {AC → BC} It means that attribute in dependencies does not change the basic dependencies.
Rule 3	**Transitivity Rule** If A holds B and B holds C, then A holds C. If {A → B} and {B → C}, then {A →C} A holds B {A → B} means that A functionally determines B.

B. Secondary Rules

Rule 1	**Union** If A holds B and A holds C, then A holds BC. If{A → B} and {A → C}, then {A → BC}
Rule 2	**Decomposition** If A holds BC and A holds B, then A holds C. If{A → BC} and {A → B}, then {A → C}
Rule 3	**Pseudo Transitivity** If A holds B and BC holds D, then AC holds D. If{A → B} and {BC → D}, then {AC → D}

Sometimes Functional Dependency Sets are not able to reduce if the set has following properties,

- The Right-hand side set of functional dependency holds only one attribute.
- The Left-hand side set of functional dependency cannot be reduced, it changes the entire content of the set.
- Reducing any functional dependency may change the content of the set.
- A set of functional dependencies with the above three properties are also called as **Canonical or Minimal.**

Trivial Functional Dependency

Trivial	If A holds B {A → B}, where B is a subset of A, then it is called a **Trivial Functional Dependency**. Trivial always holds Functional Dependency.
Non-Trivial	If A holds B {A → B}, where B is not a subset A, then it is called as a **Non-Trivial Functional Dependency.**
Completely Non-Trivial	If A holds B {A → B}, where A intersect Y = Φ, it is called as a **Completely Non-Trivial Functional Dependency.**

Advantages of Functional Dependency

- Functional Dependency avoids data redundancy where same data should not be repeated at multiple locations in same database.
- It maintains the quality of data in database.
- It allows clearly defined meanings and constraints of databases.
- It helps in identifying bad designs.
- It expresses the facts about the database design.

Example 1:

Consider relation E = (P, Q, R, S, T, U) having set of Functional Dependencies (FD).

P → Q ***P → R***
QR → S ***Q → T***
QR → U ***PR → U***

Calculate some members of Axioms are as follows,

1. P → T
2. PR → S
3. QR → SU
4. PR → SU

Solution:

1. P → T

In the above FD set, P → Q and Q → T

So, Using Transitive Rule: If {A → B} and {B → C}, then {A → C}

∴ If P → Q and Q → T, then **P → T.**

P → T

2. PR → S

In the above FD set, P → Q

As, QR → S

So, Using Pseudo Transitivity Rule: If {A → B} and {BC → D}, then {AC → D}

∴ If P → Q and QR → S, **then PR → S.**

PR → S

3. QR → SU

In above FD set, QR → S and QR → U

So, Using Union Rule: If {A → B} and {A → C}, then {A → BC}

∴ If QR → S and QR → U, **then QR → SU.**

QR → SU

4. PR → SU

So, Using Pseudo Transitivity Rule: If {A → B} and {BC → D}, then {AC → D}

∴ If PR → S and PR → U, **then PR → SU.**

PR → SU

<u>Example 2:</u>

Let us consider the example of schema *R* = (*A*, *B*, *C*, *G*, *H*, *I*) and the set *F* of functional dependencies {*A*→ *B*, *A*→ *C*, *CG* → *H*, *CG* → *I* , *B* → *H*}.

We list several members of *F*+ here:

- *A* → *H*. Since *A* → *B* and *B* → *H* hold, we apply the transitivity rule. Observe that it was much easier to use Armstrong's axioms to show that *A* → *H* holds than it was to argue directly from the definitions, as we did earlier in this section.
- *CG* → *HI*. Since *CG* → *H* and *CG* → *I*, the union rule implies that *CG* →*HI*.
- *AG* → *I*. Since *A*→*C* and *CG* → *I*, the pseudotransitivity rule implies that *AG* → *I* holds.
- Another way of finding that *AG* →*I* holds is as follows: We use the augmentation rule on *A*→*C* to infer *AG* → *CG*. Applying the transitivity rule to this dependency and *CG* →*I*, we infer *AG* →*I*.

<u>Non-loss Decomposition</u>

Explain in detail about non-loss decomposition and functional dependencies.

What is Decomposition?

- Decomposition is the process of breaking down in parts or elements.
- It replaces a relation with a collection of smaller relations.
- It breaks the table into multiple tables in a database.
- It should always be lossless, because it confirms that the information in the original relation can be accurately reconstructed based on the decomposed relations.
- If there is no proper decomposition of the relation, then it may lead to problems like loss of information.

Properties of Decomposition

The following are the properties of decomposition,

1. Lossless Decomposition
2. Dependency Preservation
3. Lack of Data Redundancy

1. Lossless Decomposition

- Decomposition must be lossless. It means that the information should not get lost from the relation that is decomposed.
- It gives a guarantee that the join will result in the same relation as it was decomposed.

Example:

- Let's take 'E' is the Relational Schema, With instance 'e'; is decomposed into: E1, E2, E3, ... En; With instance: e1, e2, e3, ... en, If e1 ⋈ e2 ⋈ e3 . . . . ⋈ en, then it is called as **'Lossless Join Decomposition'.**
- In the above example, it means that, if natural joins of all the decomposition give the original relation, then it is said to be lossless join decomposition.

Example: <Employee_Department> Table

Eid	Ename	Age	City	Salary	Deptid	DeptName
E001	ABC	29	Pune	20000	D001	Finance
E002	PQR	30	Pune	30000	D002	Production
E003	LMN	25	Mumbai	5000	D003	Sales
E004	XYZ	24	Mumbai	4000	D004	Marketing
E005	STU	32	Bangalore	25000	D005	Human Resource

- Decompose the above relation into two relations to check whether decomposition is lossless or lossy.
- Now, we have decomposed the relation that is Employee and Department.

Relation 1: <Employee> Table

Eid	Ename	Age	City	Salary
E001	ABC	29	Pune	20000
E002	PQR	30	Pune	30000
E003	LMN	25	Mumbai	5000
E004	XYZ	24	Mumbai	4000
E005	STU	32	Bangalore	25000

Employee Schema contains (Eid, Ename, Age, City, Salary).

Relation 2: <Department> Table

Deptid	Eid	DeptName
D001	E001	Finance
D002	E002	Production
D003	E003	Sales
D004	E004	Marketing
D005	E005	Human Resource

- Department Schema contains (Deptid, Eid, DeptName).
- So, the above decomposition is a Lossless Join Decomposition, because the two relations contains one common field that is 'Eid' and therefore join is possible.
- Now apply natural join on the decomposed relations.

Employee ⋈ Department

Eid	Ename	Age	City	Salary	Deptid	DeptName
E001	ABC	29	Pune	20000	D001	Finance
E002	PQR	30	Pune	30000	D002	Production
E003	LMN	25	Mumbai	5000	D003	Sales
E004	XYZ	24	Mumbai	4000	D004	Marketing

E005	STU	32	Bangalore	25000	D005	Human Resource

Hence, the decomposition is Lossless Join Decomposition.

- If the <Employee> table contains (Eid, Ename, Age, City, Salary) and <Department> table contains (Deptid and DeptName), then it is not possible to join the two tables or relations, because there is no common column between them. And it becomes **Lossy Join Decomposition.**

2. Dependency Preservation

- Dependency is an important constraint on the database.
- Every dependency must be satisfied by at least one decomposed table.
- If {A → B} holds, then two sets are functional dependent. And, it becomes more useful for checking the dependency easily if both sets in a same relation.
- This decomposition property can only be done by maintaining the functional dependency.
- In this property, it allows to check the updates without computing the natural join of the database structure.

3. Lack of Data Redundancy

- Lack of Data Redundancy is also known as a **Repetition of Information.**
- The proper decomposition should not suffer from any data redundancy.
- The careless decomposition may cause a problem with the data.
- The lack of data redundancy property may be achieved by Normalization process.

<u>**Normalization**</u>

Explain in detail about normalization. (Or) What are Normal Forms? Explain the types of normal forms with an example. (Nov/Dec 2014) (Or) State the need for normalization of a database and explain the various forms with suitable examples. (April/May 2015) (Or) Explain first normal form, second normal form, third normal form and BCNF with an example. (Nov/Dec 2016) (Or) What is database Normalization? Explain first normal form, second normal form, third normal form with an example. (April/May 2018) (Or) Give an example of a relation that is in 3NF but not in BCNF. How will you convert that relation into BCNF. (Nov/Dec 2018)

- **Normalization** is a process of organizing the data in the database.
- It is a systematic approach used to minimize the redundancy from a relation or set of relations.
- It is used to avoid / eliminate data redundancy, insertion anomaly, update anomaly & deletion anomaly.
- It was developed by **E. F. Codd.**

(Or)

- ―Normalization is a process of designing a consistent database by minimizing redundancy and ensuring data integrity through decomposition which is lossless.‖
- The goal is to generate a set of relation schemas that allows us to store information without unnecessary redundancy, yet also allows us to retrieve information easily.
- The approach is to design schemas that are in an appropriate *normal form*.

- To determine whether a relation schema is in one of the desirable normal forms, we need additional information about the real-world enterprise that we are modeling with the database.
- The most common approach is to use **functional dependencies.**
- It is a multi-step process that puts data into tabular form, removing duplicated data from the relation tables.
- It is also called as Canonical Synthesis.
- Normalization is used for mainly two purposes,
  - ✓ Eliminating reduntant (useless) data.
  - ✓ Ensuring data dependencies make sense i.e data is logically stored.

Features of Normalization

- Normalization avoids the data redundancy.
- It is a formal process of developing data structures.
- It promotes the data integrity.
- It ensures data dependencies make sense that means data is logically stored.
- It eliminates the undesirable characteristics like Insertion, Updation and Deletion Anomalies.

Anomalies in DBMS

- There are three types of anomalies that occur when the database is not normalized. These are – Insertion, update and deletion anomaly.

emp_id	**emp_name**	**emp_address**	**emp_dept**
101	Rick	Delhi	D001
101	Rick	Delhi	D002
123	Maggie	Agra	D890
166	Glenn	Chennai	D900
166	Glenn	Chennai	D004

Update Anomaly:

- In the above table we have two rows for employee Rick as he belongs to two departments of the company.
- If we want to update the address of Rick then we have to update the same in two rows or the data will become inconsistent.
- If somehow, the correct address gets updated in one department but not in other then as per the database, Rick would be having two different addresses, which is not correct and would lead to inconsistent data.

Insert Anomaly:

- Suppose a new employee joins the company, who is under training and currently not assigned to any department then we would not be able to insert the data into the table if emp_dept field doesn't allow nulls.

Delete Anomaly:

- Suppose, if at a point of time the company closes the department D890 then deleting the rows that are having emp_dept as D890 would also delete the information of employee Maggie since she is assigned only to this department.

First, Second, Third Normal Forms

Types of Normalization

To overcome these anomalies we need to normalize the data.

Normal Form	Description
1NF	A relation is in 1NF if it contains an atomic value.
2NF	A relation will be in 2NF if it is in 1NF and all non-key attributes are fully functional dependent on the primary key.
3NF	A relation will be in 3NF if it is in 2NF and no transition dependency exists.
4NF	A relation will be in 4NF if it is in Boyce Codd normal form and has no multi-valued dependency.
5NF	A relation is in 5NF if it is in 4NF and not contains any join dependency and joining should be lossless.

(Or)

Following are the types of Normalization:

- First Normal Form
- Second Normal Form
- Third Normal Form
- Fourth Normal Form
- Fifth Normal Form
- BCNF (Boyce – Codd Normal Form)
- DKNF (Domain Key Normal Form)

1. First Normal Form (1NF)

- First Normal Form (1NF) is a simple form of Normalization.
- It simplifies each attribute in a relation.
- In 1NF, there should not be any repeating group of data.
- Each set of column must have a unique value.
- It contains atomic values because the table cannot hold multiple values.

Rules of First Normal Form

- For a table to be in the First Normal Form, it should follow the following 4 rules:
  - ✓ It should only have single (atomic) valued attributes/columns.
  - ✓ Values stored in a column should be of the same domain
  - ✓ All the columns in a table should have unique names.
  - ✓ And the order in which data is stored, does not matter.

Example: Employee Table

ECode	Employee_Name	Department_Name

1	ABC	Sales, Production
2	PQR	Human Resource
3	XYZ	Quality Assurance, Marketing

Employee Table using 1NF

ECode	Employee_Name	Department_Name
1	ABC	Sales
1	ABC	Production
2	PQR	Human Resource
3	XYZ	Quality Assurance
3	XYZ	Marketing

2. Second Normal Form (2NF)

- In 2NF, the table is required in 1NF.
- The main rule of 2NF is, *'No non-prime attribute is dependent on the proper subset of any candidate key of the table.'*
- An attribute which is not part of candidate key is known as non-prime attribute.

Example: Employee Table using 1NF

ECode	Employee_Name	Employee_Age
1	ABC	38
1	ABC	38
2	PQR	38
3	XYZ	40
3	XYZ	40

Candidate Key: ECode, Employee_Name

Non prime Attribute: Employee_Age

- The above table is in 1NF. Each attribute has atomic values.
- However, it is not in 2NF because non prime attribute Employee_Age is dependent on ECode alone, which is a proper subset of candidate key.
- This violates the rule for 2NF as the rule says 'No non-prime attribute is dependent on the proper subset of any candidate key of the table'.

2NF (Second Normal Form):

Employee1 Table

ECode	Employee_Age
1	38
2	38
3	40

Employee 2Table

ECode	Employee_Name
1	ABC
1	ABC
2	PQR
3	XYZ
3	XYZ

- Now, the above tables comply with the Second Normal Form (2NF).

3. Third Normal Form (3NF)

- Third Normal Form (3NF) is used to minimize the transitive redundancy.
- In 3NF, the table is required in 2NF.
- While using the 2NF table, there should not be any transitive partial dependency.
- 3NF reduces the duplication of data and also achieves the data integrity.

(Or)

- A table is said to be in the Third Normal Form when,
  - ✓ It is in the Second Normal form.
  - ✓ And, it doesn't have Transitive Dependency.

Example : <Employee> Table

EId	Ename	DOB	City	State	Zip
001	ABC	10/05/1990	Pune	Maharashtra	411038
002	XYZ	11/05/1988	Mumbai	Maharashtra	400007

- In the above <Employee> table, EId is a primary key but City, State depends upon Zip code.
- The dependency between Zip and other fields is called Transitive Dependency.
- Therefore we apply 3NF. So, we need to move the city and state to the new <Employee_Table2> table, with Zip as a Primary key.

<Employee_Table1> Table

EId	Ename	DOB	Zip
001	ABC	10/05/1990	411038
002	XYZ	11/05/1988	400007

<Employee_Table2> Table

City	State	Zip
Pune	Maharashtra	411038
Mumbai	Maharashtra	400007

- The advantage of removing transitive dependency is, it reduces the amount of data dependencies and achieves the data integrity.
- In the above example, using with the 3NF, there is no redundancy of data while inserting the new records.

- The City, State and Zip code will be stored in the separate table. And therefore the updation becomes more easier because of no data redundancy.

4. BCNF (Boyce – Code Normal Form)

- BCNF which stands for Boyce – Code Normal From is developed by Raymond F. Boyce and E. F. Codd in 1974.
- BCNF is a higher version of 3NF.
- It deals with the certain type of anomaly which is not handled by 3NF.
- A table complies with BCNF if it is in 3NF and any attribute is fully functionally dependent that is A → B. (Attribute 'A' is determinant).
- If every determinant is a candidate key, then it is said to be BCNF.
- Candidate key has the ability to become a primary key. It is a column in a table.

(Or)

- **Boyce and Codd Normal Form** is a higher version of the Third Normal form.
- This form deals with certain type of anomaly that is not handled by 3NF.
- A 3NF table which does not have multiple overlapping candidate keys is said to be in BCNF.
- For a table to be in BCNF, following conditions must be satisfied:
  - ✓ R must be in 3rd Normal Form
  - ✓ and, for each functional dependency (X → Y), X should be a super Key.

Example : <EmployeeMain> Table

Empid	Ename	DeptName	DepType
E001	ABC	Production	D001
E002	XYZ	Sales	D002

The functional dependencies are:

Empid → EmpName

DeptName → DeptType

Candidate Key:

Empid

DeptName

- The above table is not in BCNF as neither Empid nor DeptName alone are keys.
- We can break the table in three tables to make it comply with BCNF.

<Employee> Table

Empid	EmpName
E001	ABC
E002	XYZ

<Department> Table

DeptName	DeptType
Production	D001
Sales	D002

<Emp_Dept> Table

Empid	DeptName
E001	Production
E002	Sales

Now, the functional dependencies are:

Empid → EmpName

DeptName → DeptType

Candidate Key:

<Employee> Table : Empid

<Department> Table : DeptType

<Emp_Dept> Table : Empid, DeptType

- So, now both the functional dependencies left side part is a key, so it is in the BCNF.

5. Fourth Normal Form (4NF)

- Fourth Normal Form (4NF) does not have non-trivial multi-valued dependencies other than a candidate key.
- 4NF builds on the first three normal forms (1NF, 2NF and 3NF) and the BCNF.
- It does not contain more than one multi-valued dependency.
- This normal form is rarely used outside of academic circles.

Rules for 4th Normal Form

For a table to satisfy the Fourth Normal Form, it should satisfy the following two conditions:

- It should be in the **Boyce-Codd Normal Form**.
- And, the table should not have any **Multi-valued Dependency**.

For Example:

- A table contains a list of three things that is 'Student', 'Teacher', 'Book'. Teacher is in charge of Student and recommended book for each student.
- These three elements (Student, Teacher and Book) are independent of one another.
- Changing the student's recommended book, for instance, has no effect on the student itself. This is an example of multi-valued dependency, where an item depends on more than one value. In this example, the student depends on both teacher and book.
- Therefore, 4NF states that a table should not have more than one dependency.

6. Fifth Normal Form (5NF)

- 5NF is also knows as Project-Join Normal Form (PJ/NF).
- It is designed for reducing the redundancy in relational databases.
- 5NF requires semantically related multiple relationships, which are rare.
- In 5NF, if an attribute is multivalued attribute, then it must be taken out as a separate entity.
- While performing 5NF, the table must be in 4NF.

7. DKNF (Domain Key Normal Form)

- DKNF stands for Domain Key Normal Form requires the database that contains no constraints other than domain constraints and key constraints.
- In DKNF, it is easy to build a database.
- It avoids general constraints in the database which are not clear domain or key constraints.
- The 3NF, 4NF, 5NF and BCNF are special cases of the DKNF.
- It is achieved when every constraint on the relation is a logical consequence of the definition.

<u>Dependency Preservation</u>

Explain in detail about dependency preservation.

- F' is a set of functional **dependencies** on schema R, but in general, However, it may be that .
- If this is so, then every functional **dependency** in F is implied by F', and if F' is satisfied, then F must also be satisfied.
- A decomposition having the property that is a **dependency-preserving** decomposition.

Dependency Preservation

- A Decomposition D = { R1, R2, R3….Rn } of R is dependency preserving with respect to a set F of Functional dependency if

$(F1 \cup F2 \cup \dots \cup Fm)^+ = F^+$.

Consider a relation R

R ---> F{...with some functional dependency(FD)....}

R is decomposed or divided into R1 with FD { f1 } and R2 with { f2 }, then there can be three cases:

f1 U f2 = F -----> Decomposition is dependency preserving.

f1 U f2 is a subset of F -----> Not Dependency preserving.

f1 U f2 is a super set of F -----> This case is not possible.

Problem:

Let a relation R (A, B, C, D) and functional dependency {AB –> C, C –> D, D –> A}.

Relation R is decomposed into R1(A, B, C) and R2(C, D). Check whether decomposition is dependency preserving or not.

Solution:

R1(A, B, C) and R2(C, D)

Let us find closure of F1 and F2

To find closure of F1, consider all combination of ABC. i.e., find closure of A, B, C, AB, BC and AC

Note ABC is not considered as it is always ABC

closure(A) = { A } // Trivial

closure(B) = { B } // Trivial

closure(C) = {C, A, D} but D can't be in closure as D is not present R1.

= {C, A}

C--> A // Removing C from right side as it is trivial attribute

closure(AB) = {A, B, C, D}
= {A, B, C}
AB --> C // Removing AB from right side as these are trivial attributes
closure(BC) = {B, C, D, A}
= {A, B, C}
BC --> A // Removing BC from right side as these are trivial attributes
closure(AC) = {A, C, D}
AC --> D // Removing AC from right side as these are trivial attributes
F1 {C--> A, AB --> C, BC --> A}.
Similarly F2 { C--> D }
In the original Relation Dependency { AB --> C , C --> D , D --> A}.
AB --> C is present in F1.
C --> D is present in F2.
D --> A is not preserved.
F1 U F2 is a subset of F. So **given decomposition is not dependency preserving**.

Dependency-Preserving Decomposition

- The **dependency preservation decomposition** is another property of decomposed relational database schema D in which each functional dependency X -> Y specified in F either appeared directly in one of the relation schemas R_i in the decomposed D or could be inferred from the dependencies that appear in some Ri.
- Decomposition D = { R_1 , R_2, R_3,,.., ,R_m} of R is said to be dependency-preserving with respect to F if the union of the projections of F on each R_i , in D is equivalent to F.
- In other words, R ⊂ join of R_1, R_1 over X.
- The dependencies are preserved because each dependency in F represents a constraint on the database.
- If decomposition is not dependency-preserving, some dependency is lost in the decomposition.

Example:
Let a relation R(A,B,C,D) and set a FDs F = { A -> B , A -> C , C -> D} are given.
A relation R is decomposed into -
R_1 = (A, B, C) with FDs F_1 = {A -> B, A -> C}, and
R_2 = (C, D) with FDs F_2 = {C -> D}.
F' = $F_1 \cup F_2$ = {A -> B, A -> C, C -> D}
so, F' = F.
And so, $F'^{+} = F^{+}$.
Thus, the decomposition is dependency preserving decomposition.

Multi-valued Dependencies and Fourth Normal Form

What is Multi-valued Dependency?

A table is said to have multi-valued dependency, if the following conditions are true,

1. For a dependency A → B, if for a single value of A, multiple value of B exists, then the table may have multi-valued dependency.
2. Also, a table should have at-least 3 columns for it to have a multi-valued dependency.
3. And, for a relation R(A,B,C), if there is a multi-valued dependency between, A and B, then B and C should be independent of each other.

If all these conditions are true for any relation(table), it is said to have multi-valued dependency.

- To deal with the problem of BCNF, R. Fagin introduced the idea of multi-valued dependency (MVD) and the fourth normal form (4NF).
- A multi-valued dependency (MVD) is a functional dependency where the dependency may be to a set and not just a single value.
- It is defined as X →→ Y in relation R (X, Y, Z), if each X value is associated with a set of Y values in a way that does not depend on the Z values.
- Here X and Y are both subsets of R. The notation X→→ Y is used to indicate that a set of attributes of Y shows a multi-valued dependency (MVD) on a set of attributes of X.

Join Dependencies and Fifth Normal Form

- The anomalies of MVDs and are eliminated by join dependency (JD) and 5NF.
- A join dependency (JD) can be said to exist if the join of R_1 and R_2 over C is equal to relation R.
- Where, R_1 and R_2 are the decompositions R_1(A, B, C), and R_2 (C,D) of a given relations R (A, B, C, D). Alternatively, R_1 and R_2 is a lossless decomposition of R.
- In other words, *(A, B, C, D), (C, D) will be a join dependency of R if the join of the join's attributes is equal to relation R. Here, *(R_1, R_2, R_3, ...) indicates that relations R_1, R_2, R_3 and so on are a join dependency (JD) of R.
- Therefore, a necessary condition for a relation R to satisfy a JD *(R_1, R_2,...., R_n) is that R.

Denormalization

What is Denormalization?

- Denormalization is the process of increasing the redundancy in the database.
- It is the opposite process of normalization.
- It is mostly done for improving the performance.
- It is a strategy that database managers use to increase the performance of a database structure.
- Denormalization adds redundant data normalized database for reducing the problems with database queries which combine data from the various tables into a single table.
- The process of adding redundant data to get rid of complex join, in order to optimize database performance. This is done to speed up database access by moving from higher to lower form of normalization.

- Data is included in one table from another in order to eliminate the second table which reduces the number of JOINS in a query and thus achieves performance.

Difference between Lossless Join Decomposition and Dependency Preservation Decomposition

Lossless Join Decomposition

- The lossless join property is a feature of decomposition supported by normalization. It is the ability to ensure that any instance of the original relation can be identified from corresponding instances in the smaller relations.
  R : relation, F : set of functional dependencies on R,
  X,Y : decomposition of R
- A decomposition {R1, R2,…, Rn} of a relation R is called a lossless decomposition for R if the natural join of R1, R2,…, Rn produces exactly the relation R.
- A decomposition is lossless if we can recover:
  R(A, B, C) -> Decompose -> R1(A, B) R2(A, C) -> Recover -> R'(A, B, C)
  Thus,R' = R
- Decomposition is lossles if :
  X ∩ Y -> X, that is: all attributes common to both X and Y functionally determine ALL the attributes in X.
  X ∩ Y -> Y, that is: all attributes common to both X and Y functionally determine ALL the attributes in Y
  If X ∩ Y forms a superkey of either X or Y, the decomposition of R is a lossless decomposition.

<u>Dependency Preserving Decomposition</u>

- A decomposition D = {R1, R2, ..., Rn} of R is dependency-preserving with respect to F if the union of the projections of F on each Ri in D is equivalent to F;
  if (F1∪ F2 ∪ …∪ Fn)+ = F +
- Example-
  R= (A, B, C)
  F = {A ->B, B->C}
  Key = {A}
  Ris not in BCNF
  Decomposition R1 = (A, B), R2 = (B, C)
  R1 and R2 are in BCNF, Lossless-join decomposition, Dependency preserving
- Each Functional Dependency specified in F either appears directly in one of the relations in the decomposition.
- It is not necessary that all dependencies from the relation R appear in some relation Ri.
- It is sufficient that the union of the dependencies on all the relations Ri be equivalent to the dependencies on R.
- is lost in the decomposition.

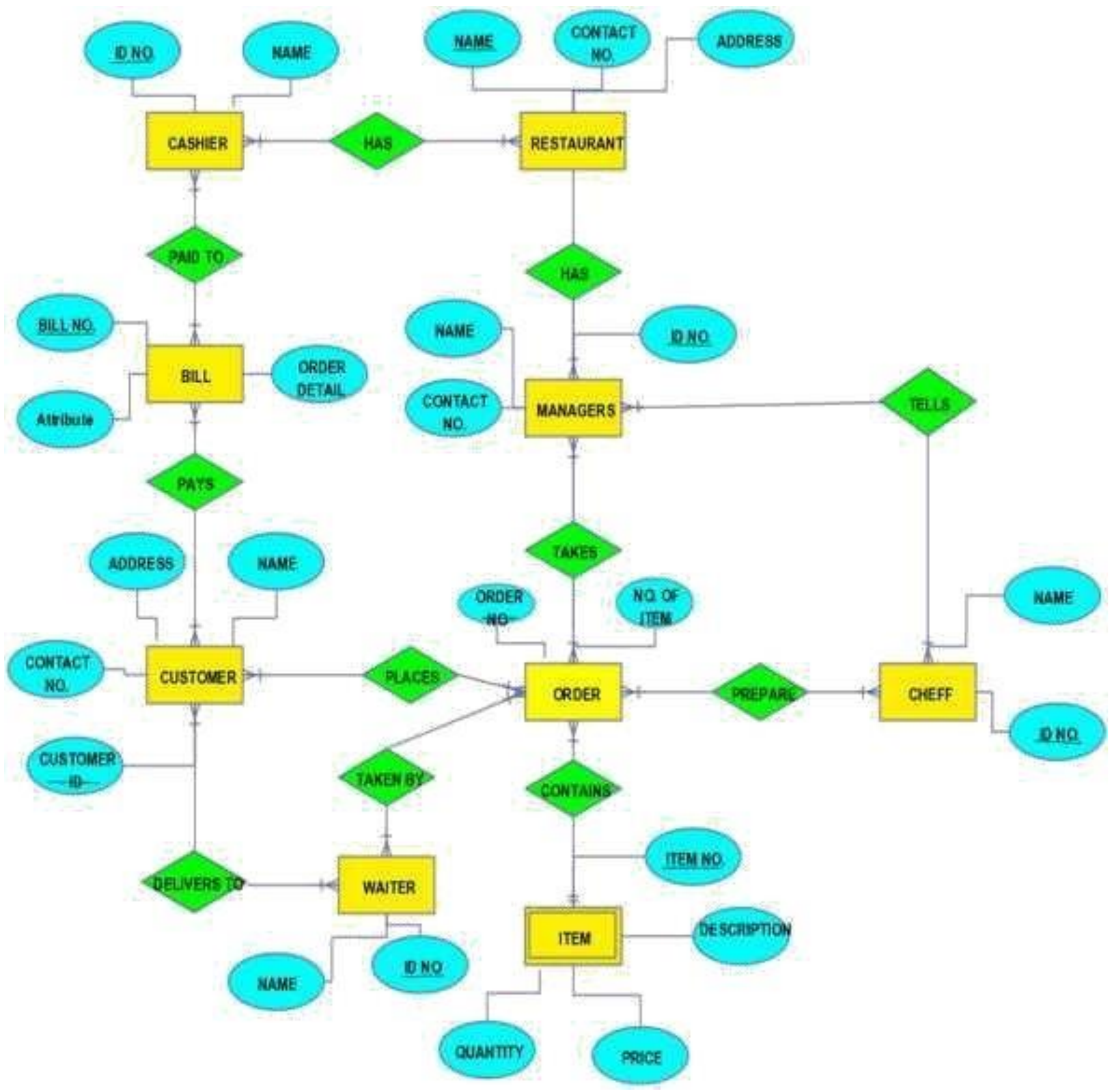
Differences Between E-R Model and Relational Model

1. Difference between E-R Model and Relational Model in DBMS The basic difference between E-R Model and Relational Model is that E-R model specifically deals with entities and their relations. On the other hand, the Relational Model deals with Tables and relation between the data of those tables.
2. An E-R Model describes the data with entity set, relationship set and attributes. However, the Relational model describes the data with the tuples, attributes and domain of the attribute.
3. One can easily understand the relationship among the data in E-R Model as compared to Relational Model.
4. E-R Model has Mapping Cardinality as a constraint whereas Relational Model does not have such constraint.

Example ER Diagrams

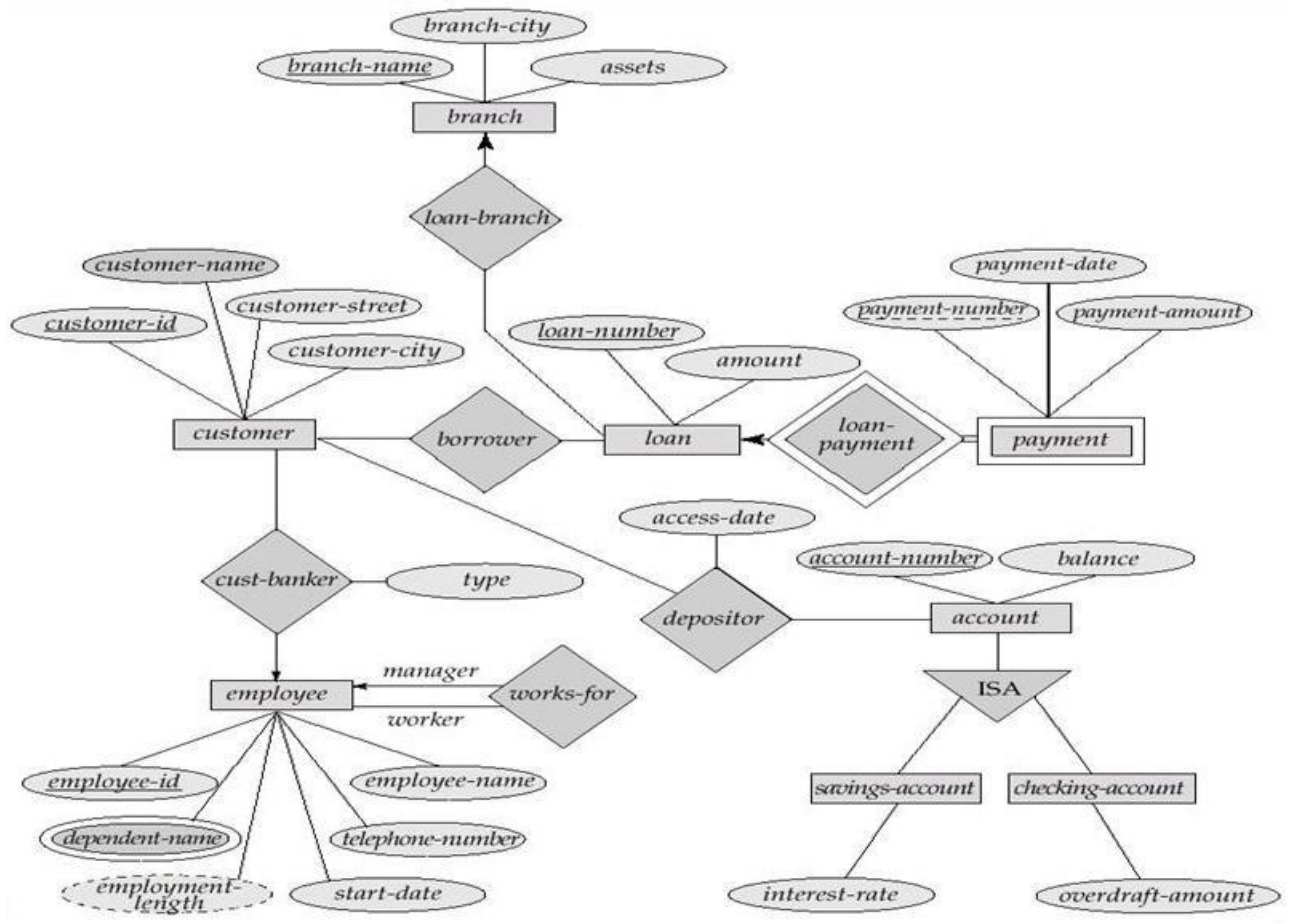
E-R Diagram for Restaurant Menu Ordering System

Draw E-R diagram for restaurant menu ordering System which will facilitate the food items ordering and services within a restaurant. The entire restaurant scenario is detailed as follows. The customer is able to view the food items menu, call the waiter place orders and obtain the final bill through the computer kept in their table. The waiters through their wireless tablet pc are able to initialize a table for customers control the table functions to assist customers, orders send orders to food preparation staff (chef) and finalize the customer bill. The food preparation staff with their touch display interfaces to the system, are able to view orders sent to the kitchen by waiters. During preparation they are able to let the waiter know the status of each item, and can send notification when items are completed. The system should have full accountability and logging facilities, and should support supervisor actions to account for exceptional circumstances, such as a meal being refunded or walked out on? (April/May 2015)

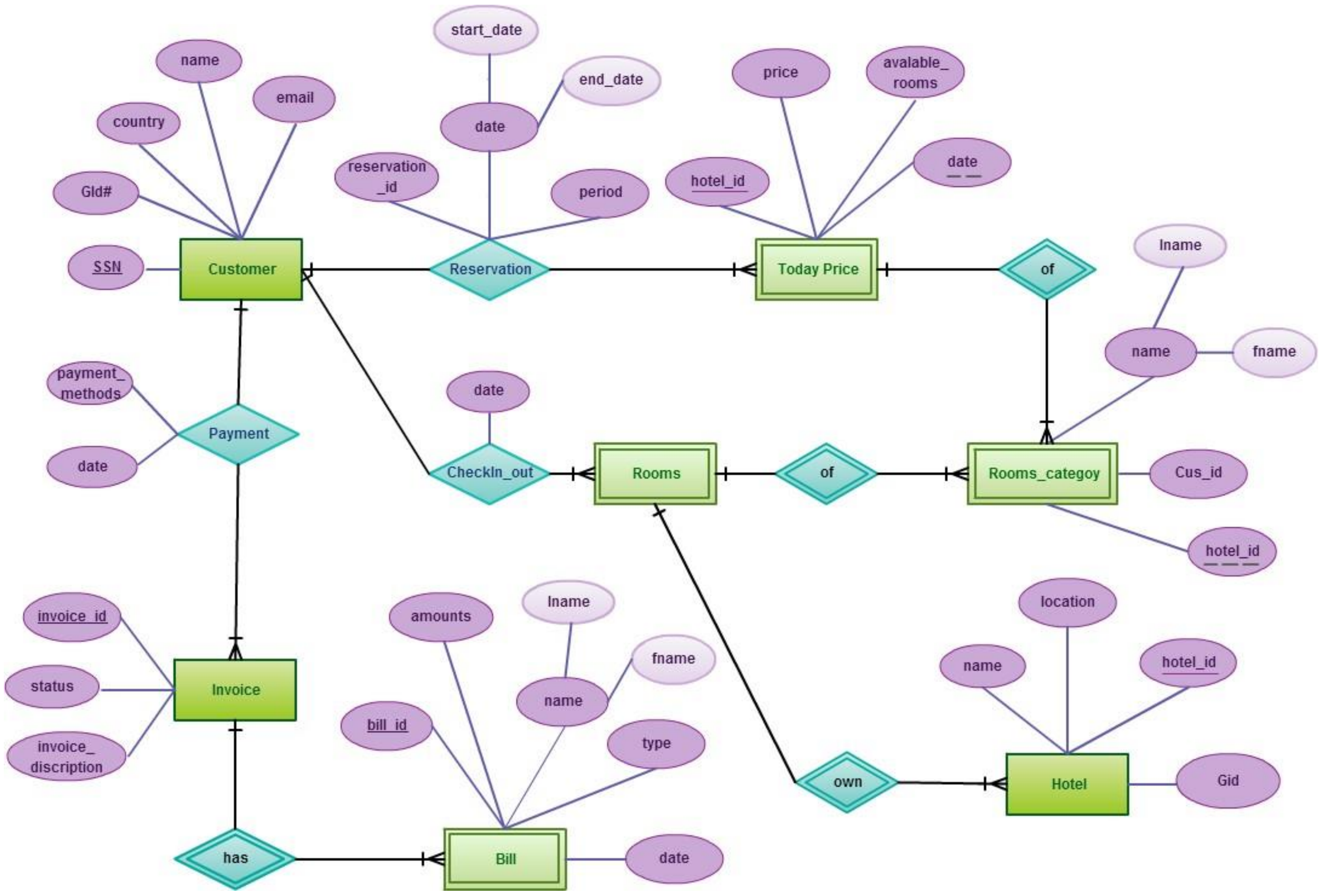


E-R Diagram for Banking Enterprise

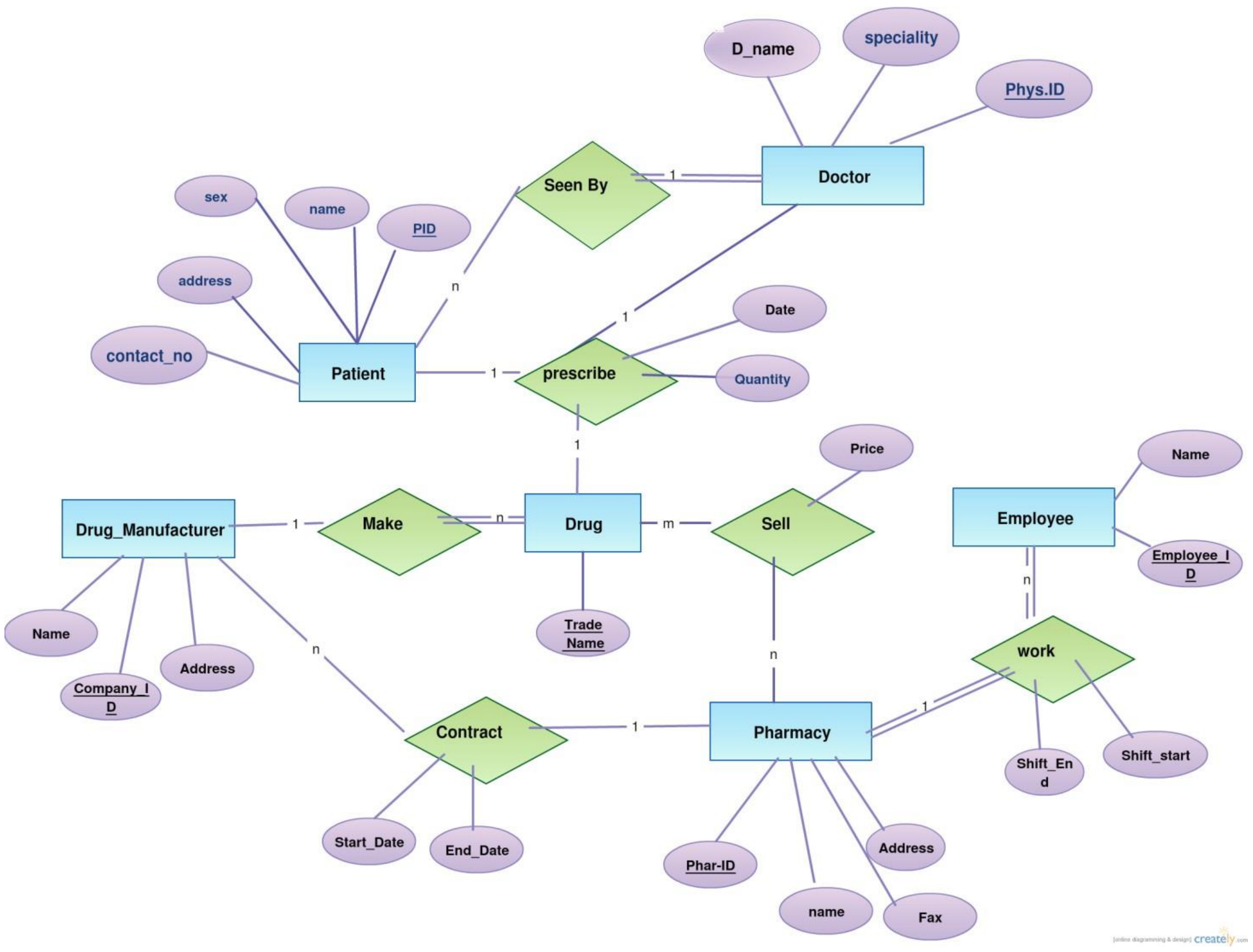
Write short notes on ER diagram for banking enterprise. (Nov/Dec 2014) (Nov/Dec 2017)



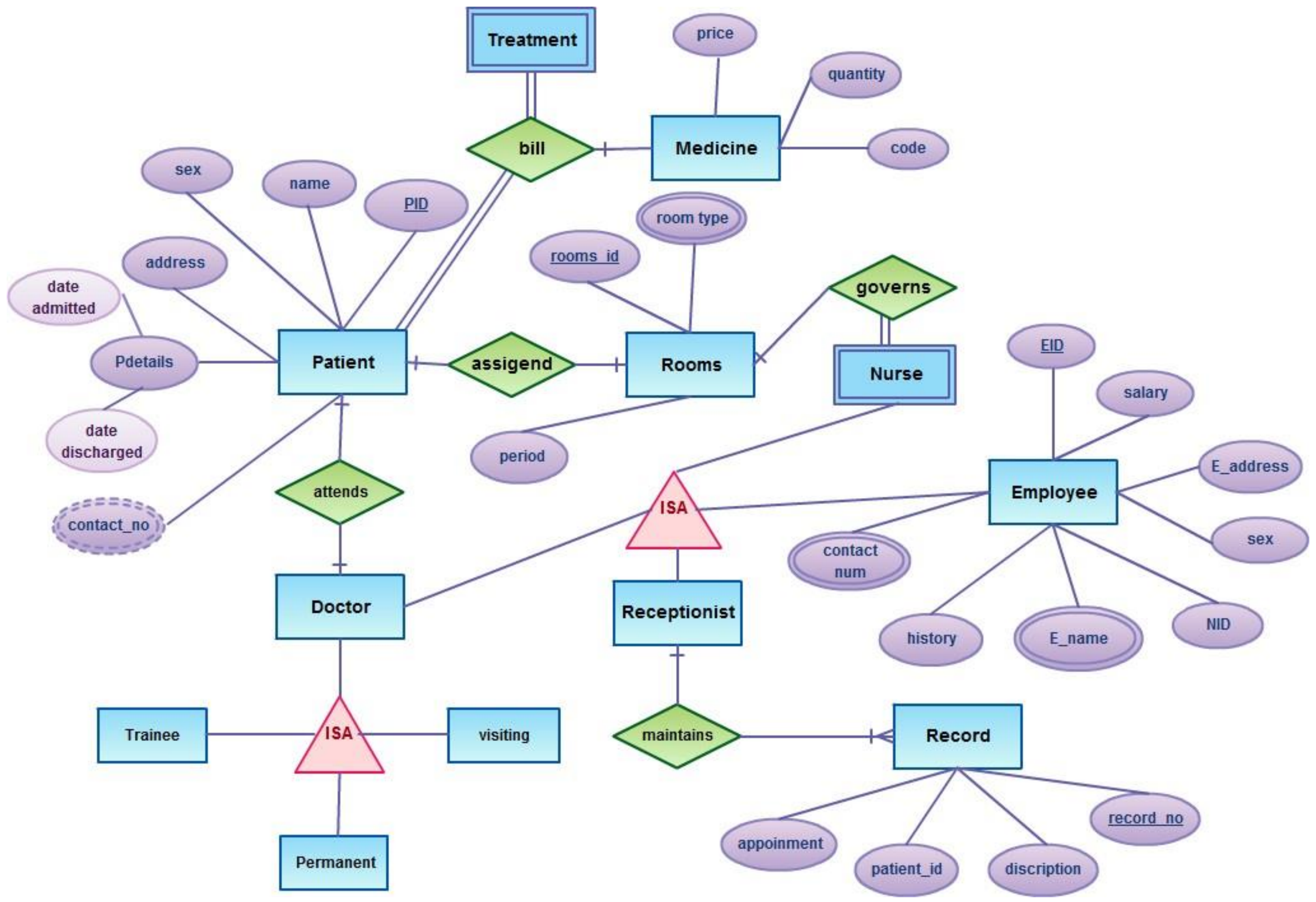
E-R Diagram for Hotel Management System



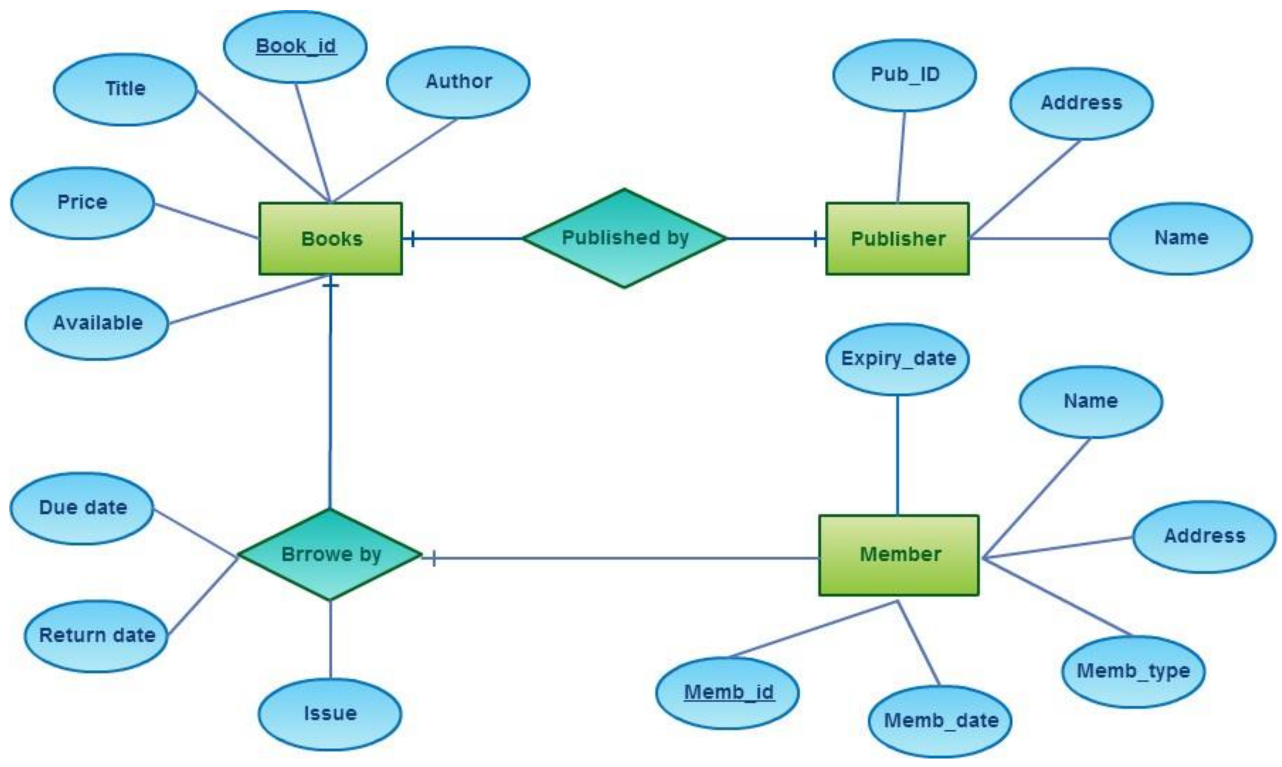
E-R Diagram for Pharmacy Store Information



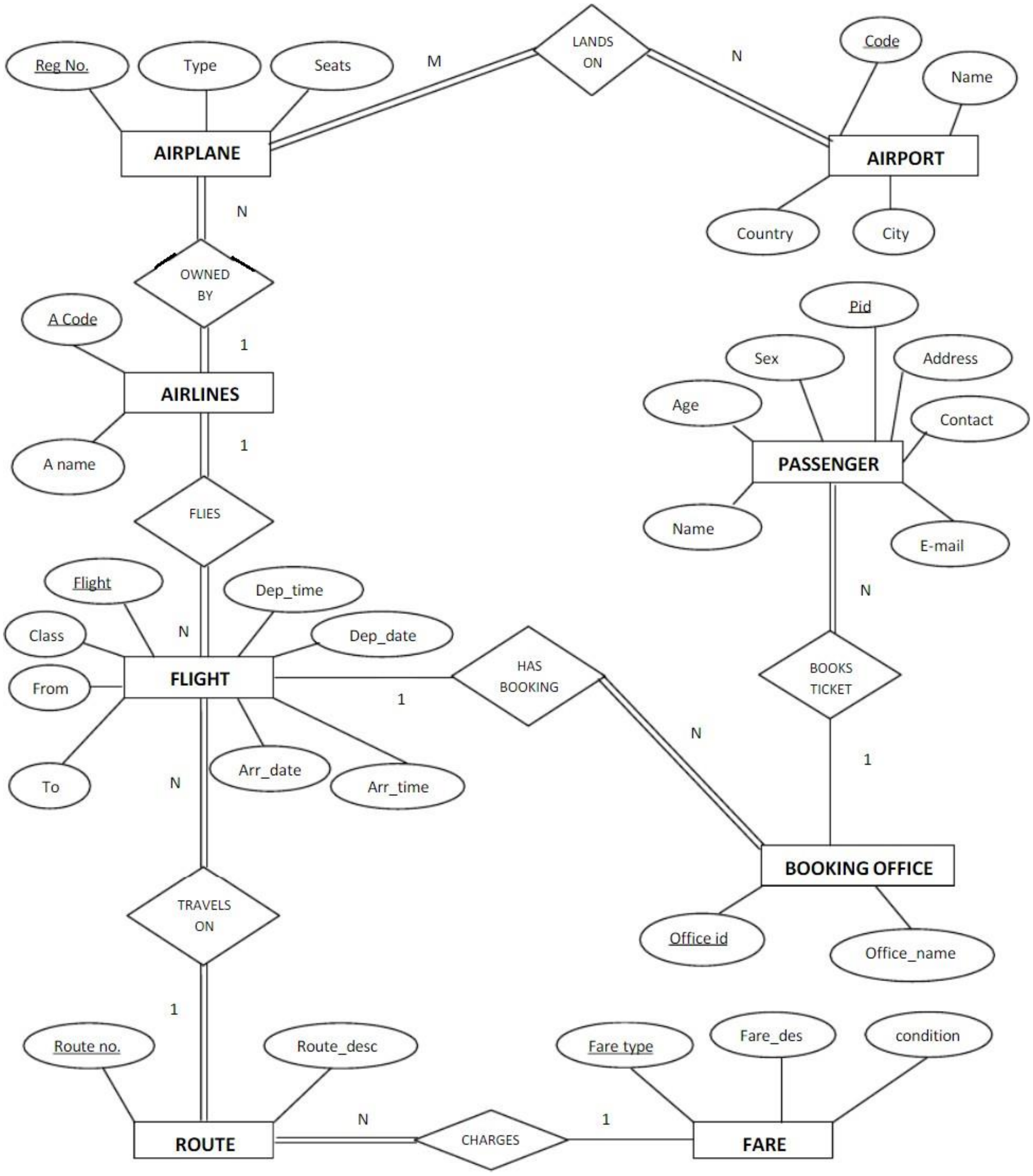
E-R Diagram for Hospital Management System



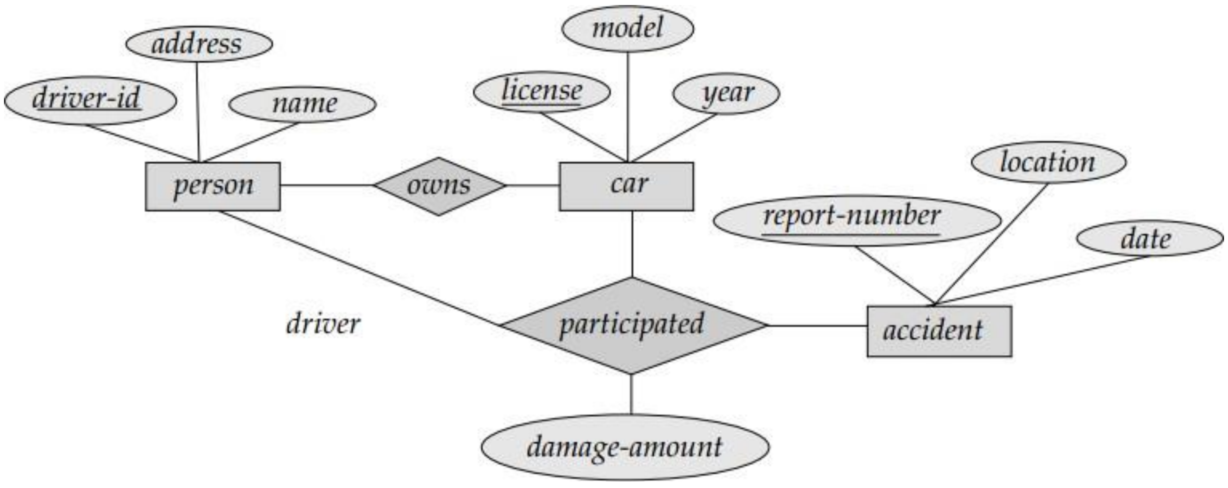
E-R Diagram for Library Management System



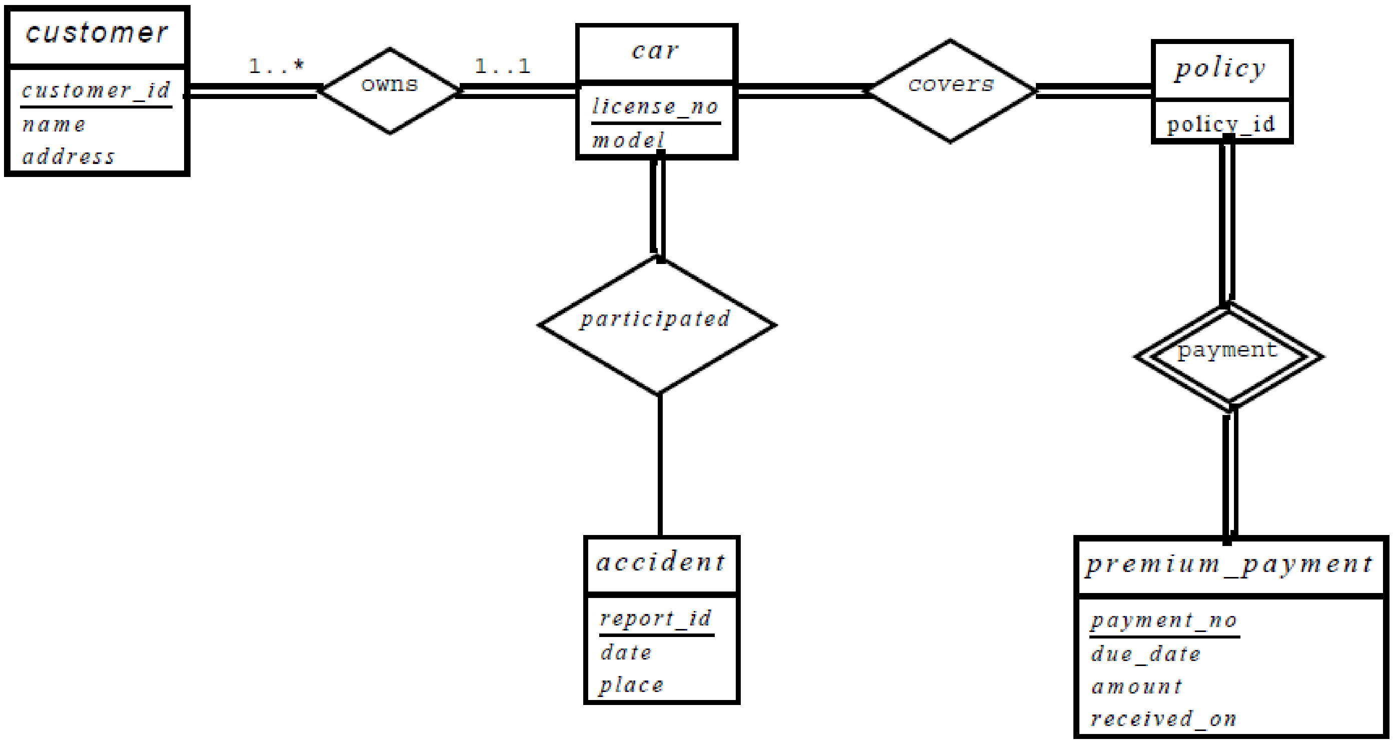
E-R Diagram for Airline Reservation System



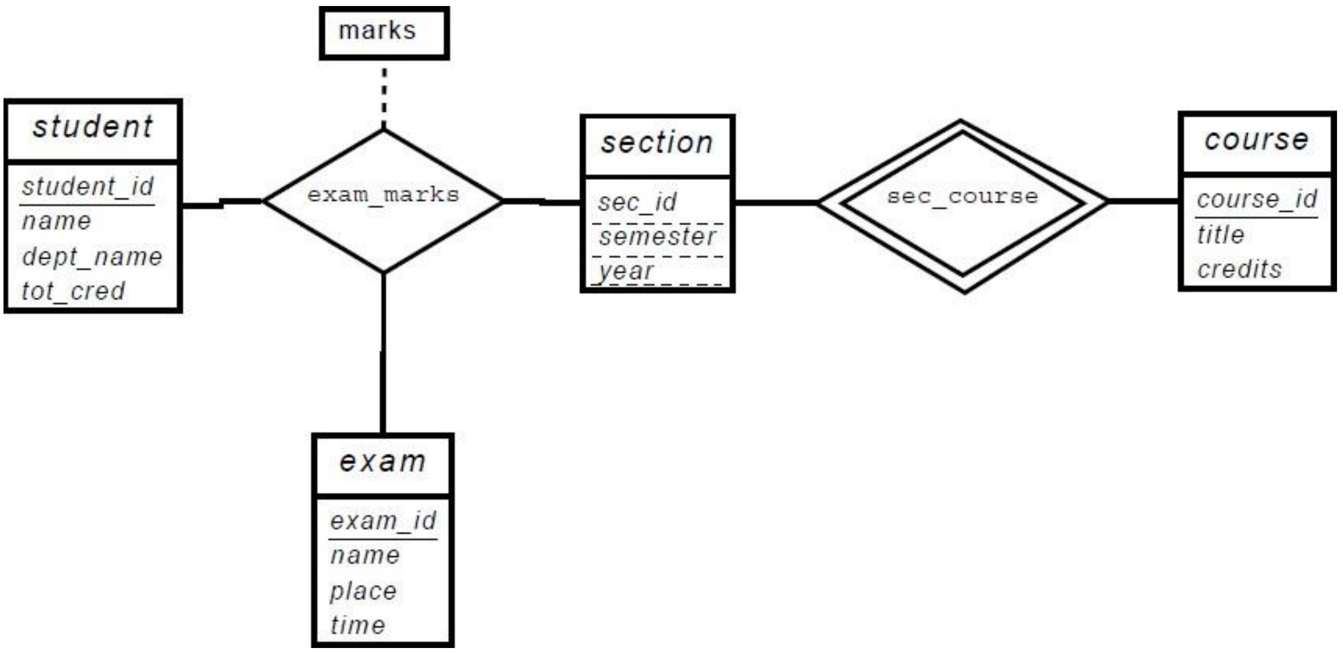
<u>E-R Diagram for Car Insurance Company</u> (Nov/Dec 2016) (Nov/Dec 2018)



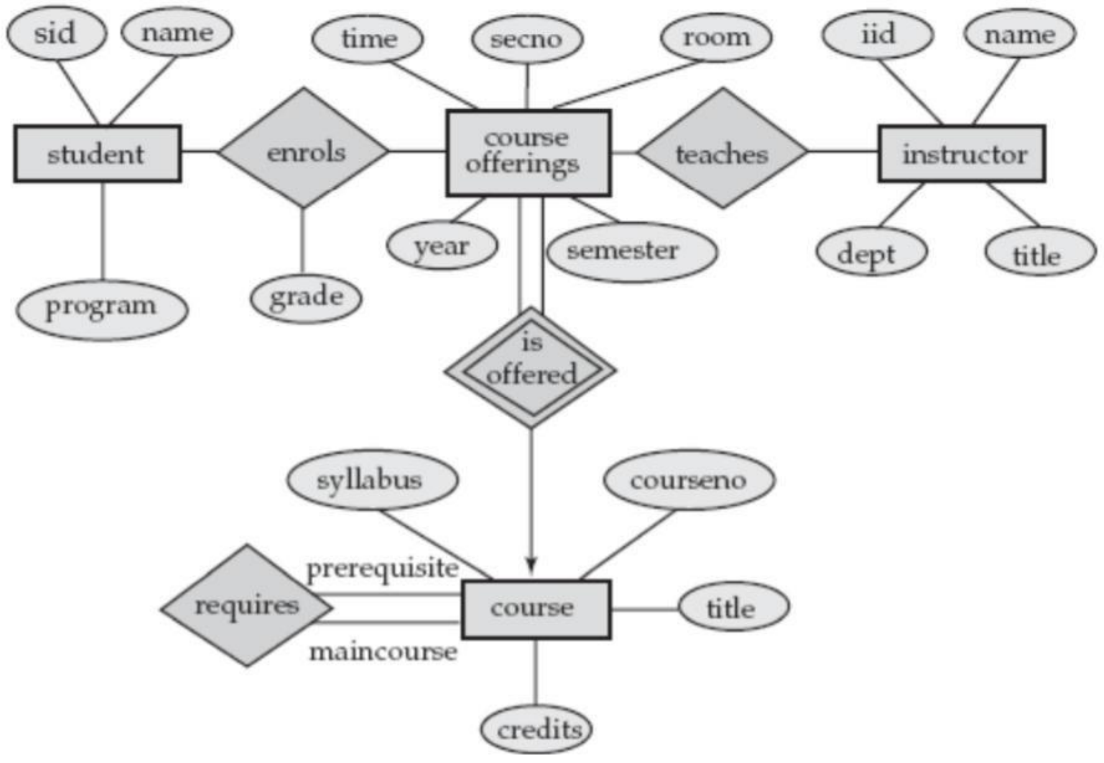
(Or)



<u>E-R Diagram for Marks Database</u>

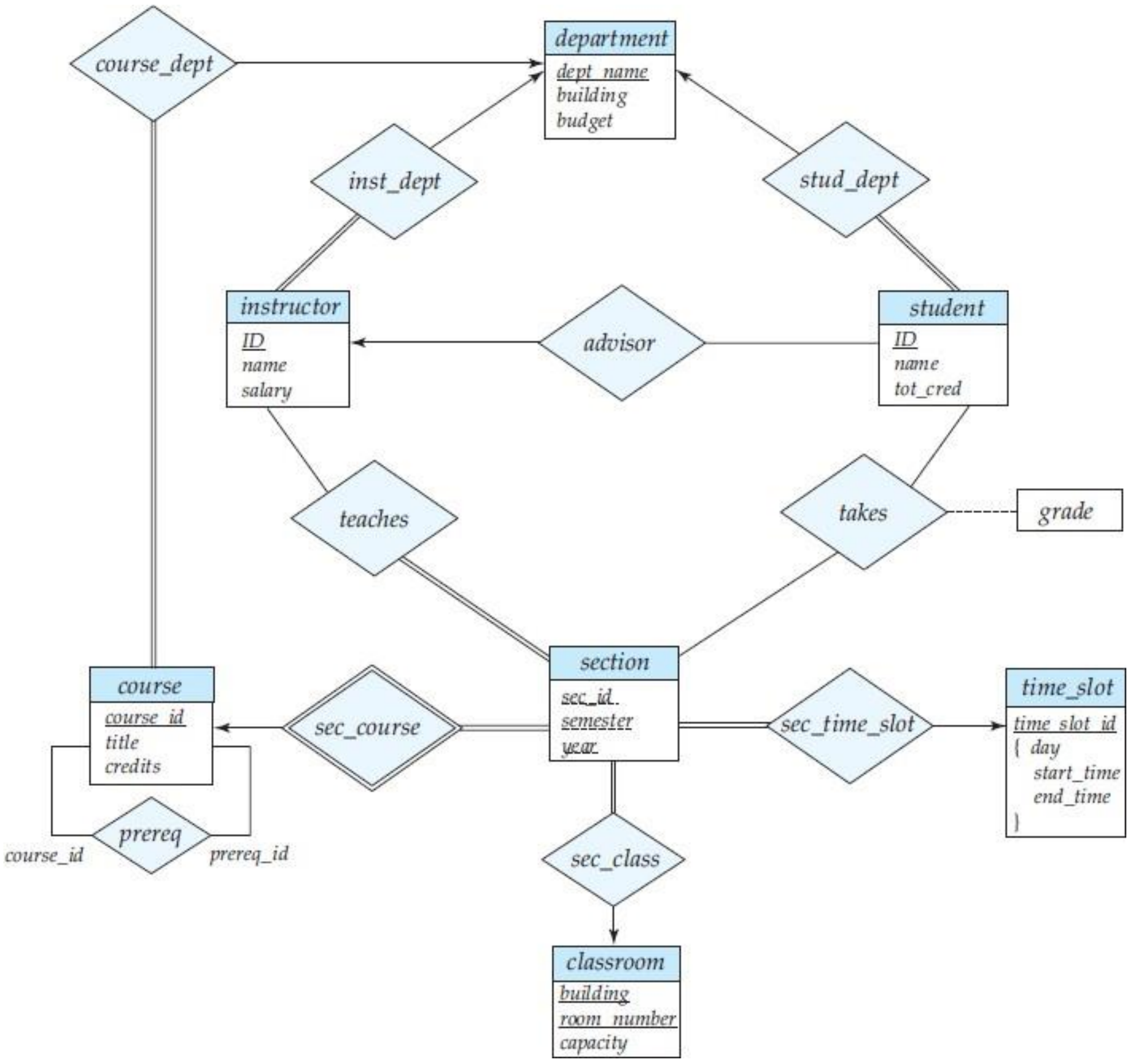


<u>E-R Diagram for University Database</u> (April/May 2018)



(Or)

<u>E-R Diagram for the University Enterprise</u>



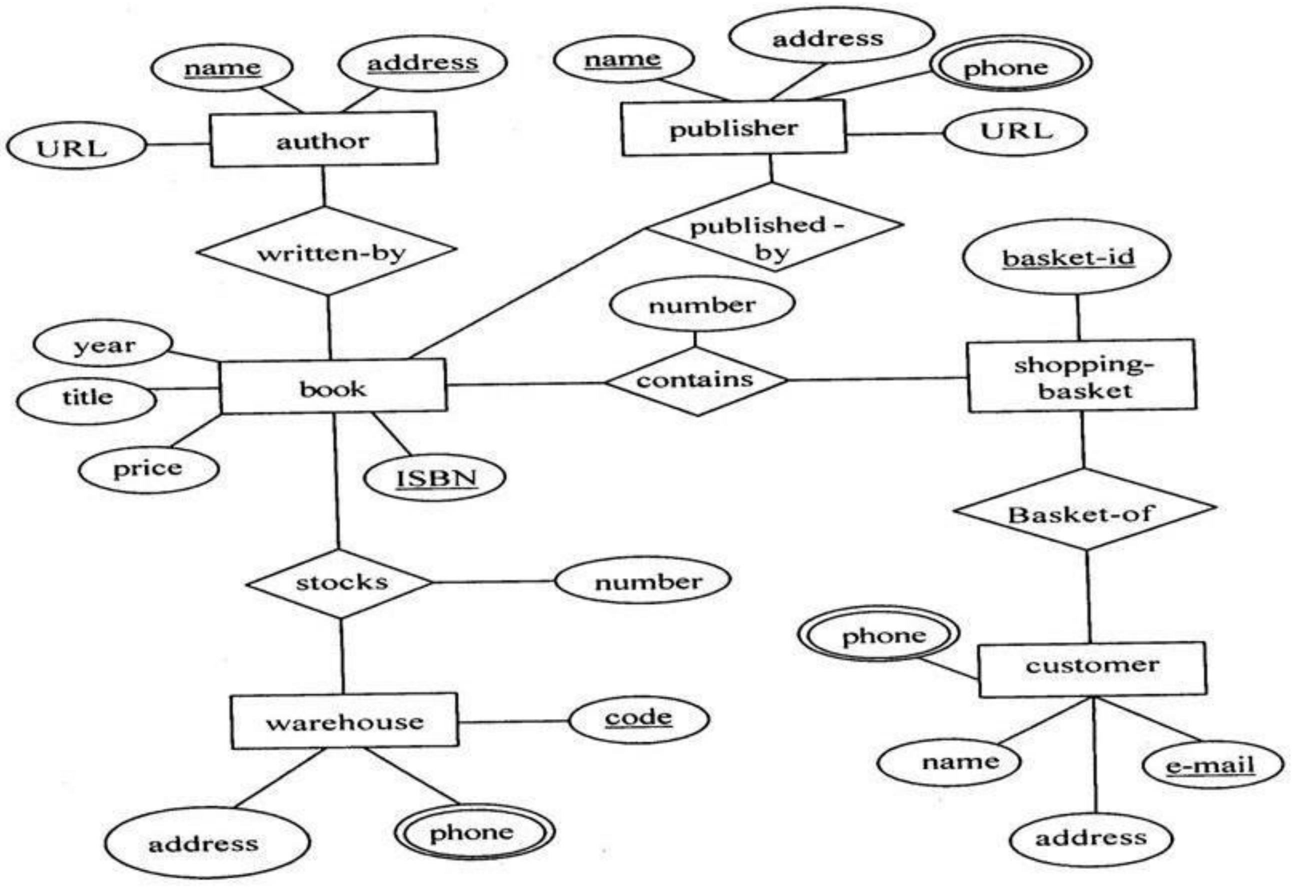
<u>E-R Diagram for Online Bookstore</u>

Draw an E-R diagram, which models an online bookstore.

a. List the entity sets and their primary keys.

b. Suppose the bookstore adds Blu-ray discs and downloadable video to its collection. The same item may be present in one or both formats, with differing prices. Extend the E-R diagram to model this addition, ignoring the effect on shopping baskets.

c. Now extend the E-R diagram, using generalization, to model the case where a shopping basket may contain any combination of books, Blu-ray discs, or downloadable video.


name
address
URL
author
written-by
name
address
phone
publisher
URL
published - by
number
basket-id
year
title
price
book
ISBN
contains
shopping-basket
Basket-of
stocks
number
phone
customer
warehouse
code
name
e-mail
address
phone
address

(Or)

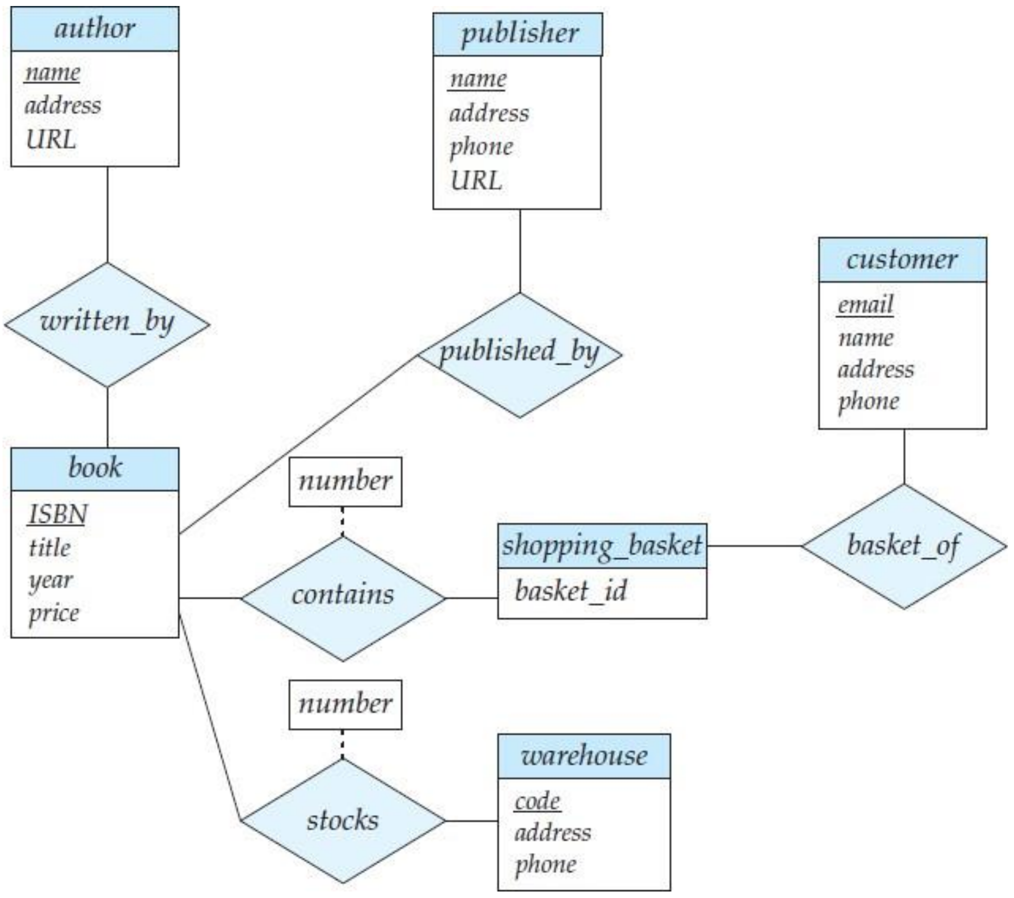
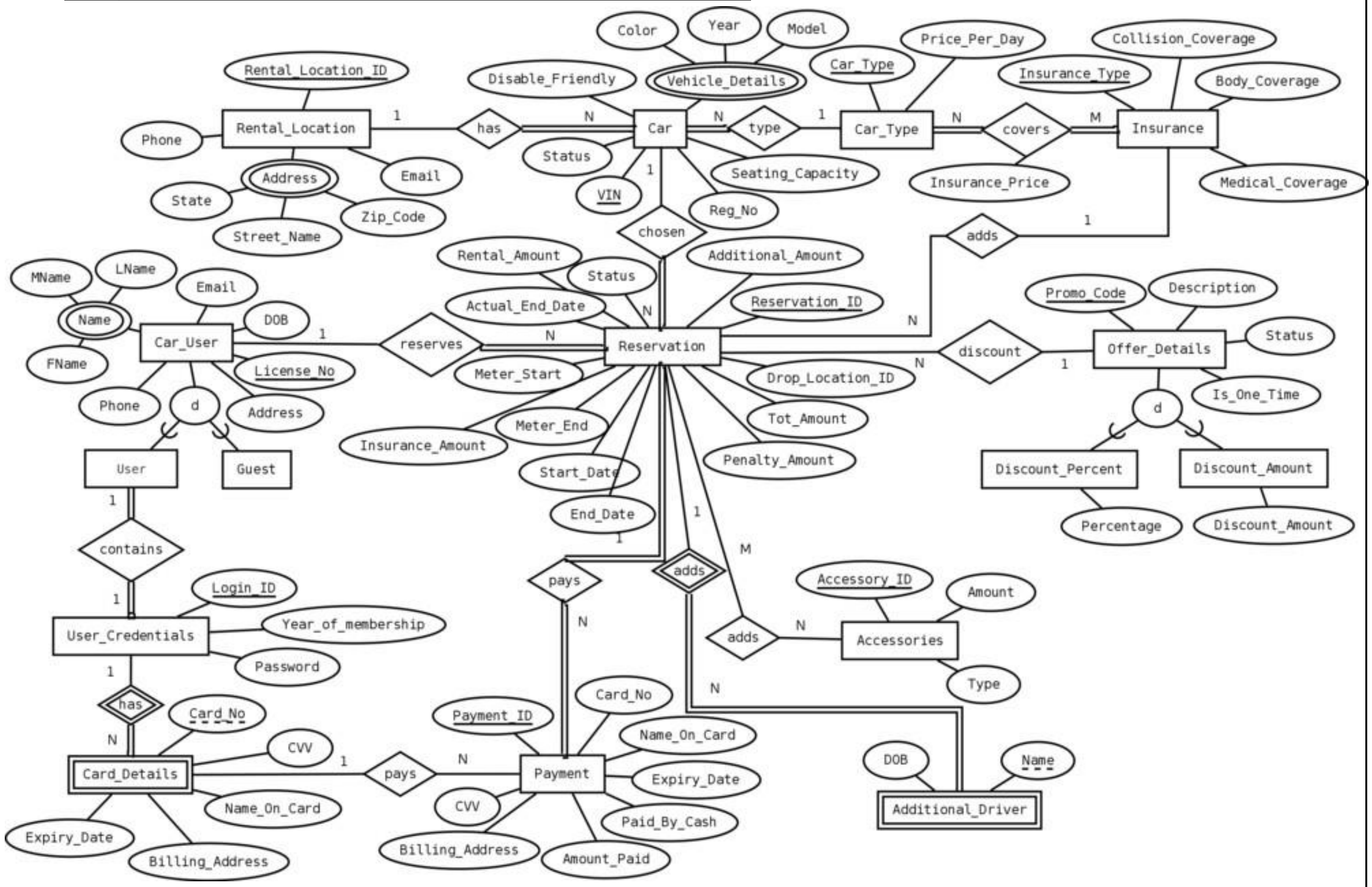

author
name
address
URL
publisher
name
address
phone
URL
customer
email
name
address
phone
written_by
published_by
book
ISBN
title
year
price
number
contains
shopping_basket
basket_id
basket_of
number
stocks
warehouse
code
address
phone

Diagram for Car Rental Company (Nov/Dec 2015)



Unit – I Relational Databases

Purpose of Database System - Views of Data - Data Models - Database System Architecture - Introduction to Relational Databases - Relational Model - Keys - Relational Algebra - SQL Fundamentals - Advanced SQL Features - Embedded SQL- Dynamic SQL.

Introduction

- Databases and database technology have a major impact on the growing use of computers.
- Databases play a critical role in almost all areas where computers are used, including business, electronic commerce, engineering, medicine, genetics, law, education and library science.

Database

What is Database?

- A database is a collection of data elements (facts) stored in a computer in a systematic way.

(Or)

- The collection of data, usually referred to as the **database**, contains information relevant to an enterprise.
- The computer program used to manage and query a database is known as a database management system (DBMS).
- A **Database System (DBS)** is a DBMS together with the data and applications.
- DBMS: A software package / system that can be used to store, manage and retrieve data form databases.

Database Management System (DBMS)

What is DBMS?

- A **database-management system (DBMS)** is a collection of interrelated data and a set of programs to access those data.
- The primary goal of a DBMS is to provide a way to store and retrieve database information that is both *convenient* and *efficient*.
- Database systems are designed to manage large bodies of information.
- Management of data involves both defining structures for storage of information and providing mechanisms for the manipulation of information.

Features of Database:

- It is a persistent (stored) collection of related data.
- The data is input (stored) only once.
- The data is organized (in some fashion).
- The data is accessible and can be queried (effectively and efficiently).

Functions of DBMS

- A DBMS makes it possible for users to create, edit and update data in database files.
- More specifically, a DBMS provides the following functions:
  - ✓ **Concurrency**: concurrent access (meaning 'at the same time') to the same database by multiple users
  - ✓ **Security**: security rules to determine access rights of users
  - ✓ **Backup and Recovery:** processes to back-up the data regularly and recover data if a problem occurs
  - ✓ **Integrity**: database structure and rules improve the integrity of the data
  - ✓ **Data Descriptions**: a data dictionary provides a description of the data

Database-System Applications

Discuss various database system applications.

Databases are widely used. Some of the database applications are:

- ***Enterprise Information***
  - ✓ ***Sales***
    - For customer, product, and purchase information.
  - ✓ ***Accounting***
    - For payments, receipts, account balances, assets and other accounting information.
  - ✓ ***Human Resources***
    - For information about employees, salaries, payroll taxes, and benefits, and for generation of paychecks.
  - ✓ ***Manufacturing***
    - For management of the supply chain and for tracking production of items in factories, inventories of items in warehouses and stores, and orders for items.
- ***Banking and Finance***
  - ✓ ***Banking***
    - For customer information, accounts, loans, and banking transactions.
  - ✓ ***Credit Card Transactions***
    - For purchases on credit cards and generation of monthly statements.
  - ✓ ***Finance***
    - For storing information about holdings, sales, and purchases of financial instruments such as stocks and bonds; also for storing real-time market data to enable online trading by customers and automated trading by the firm.
- ***Universities***
  - For student information, course registrations, and grades.
- ***Airlines***
  - For reservations and schedule information. Airlines were among the first to use databases in a geographically distributed manner.

- ***Telecommunication***
    - For keeping records of calls made, generating monthly bills, maintaining balances on prepaid calling cards, and storing information about the communication networks.

<u>Purpose of Database System</u>

Explain in detail about the purpose of database systems.

- Database systems arose in response to early methods of computerized management of commercial data.
- A **database management system (DBMS)** is a collection of programs that enables users to create and maintain a database.
- The DBMS is a *general-purpose software system* that facilitates the processes of *defining, constructing, manipulating,* and *sharing* databases among various users and applications.
    - ✓ **Defining** a Database
        - It involves specifying the data types, structures, and constraints of the data to be stored in the database.
        - The database definition or descriptive information is also stored by the DBMS in the form of a database catalog or dictionary; it is called as a **meta-data**.
    - ✓ **Constructing** the Database
        - It is the process of storing the data on some storage medium that is controlled by the DBMS.
    - ✓ **Manipulating** a Database
        - It includes functions such as querying the database to retrieve specific data, updating the database to reflect changes in the miniworld, and generating reports from the data.
    - ✓ **Sharing** a database
        - It allows multiple users and programs to access the database simultaneously.
- An **application program** accesses the database by sending queries or requests for data to the DBMS.
- A **query** typically causes some data to be retrieved; a **transaction** may cause some data to be read and some data to be written into the database.
- Some other important functions provided by the DBMS include *protecting* the database and *maintaining* it over a long period of time.
- **Protection** includes,
    - ✓ *System protection* against hardware or software malfunction (or crashes)
    - ✓ *Security protection* against unauthorized or malicious access
- A typical large database may have a life cycle of many years, so the DBMS must be able to **maintain** the database system by allowing the system to evolve as requirements change over time.

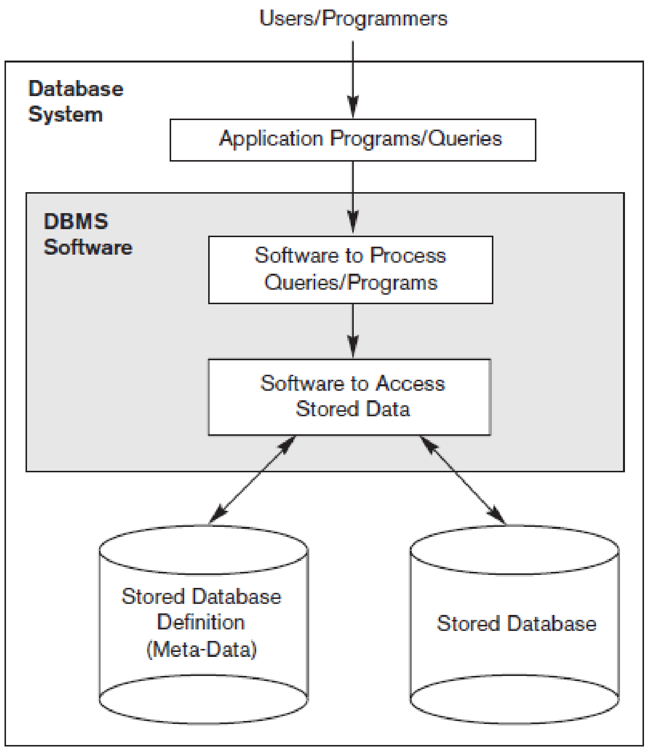

Users/Programmers
Database System
Application Programs/Queries
DBMS Software
Software to Process Queries/Programs
Software to Access Stored Data
Stored Database Definition (Meta-Data)
Stored Database

Figure: A Simplified Database System Environment

An Example

- Let us consider university database, which keeps information about all instructors, students, departments and course offerings.
- Onewaytokeeptheinformationonacomputeristostoreitinoperatingsystemfiles.
- To allow users to manipulate the information, the system has a number of application programs that manipulate the files, including programs to:
  - ✓ Add new students, instructors, and courses
  - ✓ Register students for courses and generate class rosters
  - ✓ Assign grades to students, compute grade point averages (GPA), and generate transcripts
- System programmers wrote these application programs to meet the needs of the university.

File Processing System

Explain in detail about file processing system.

- It is supported by a conventional operating system.
- The system stores permanent records in various files, and it needs different application programs to extract records from, and add records to, the appropriate files.
- Before database management systems (DBMS‘s) were introduced, organizations usually stored information in such systems.

Disadvantages of File System over DBMS

- File Processing System has a number of major disadvantages:
  - ✓ Data Redundancy andInconsistency
  - ✓ Difficulty in AccessingData
  - ✓ Data Isolation

- ✓ Integrity Problems
- ✓ Atomicity Problems
- ✓ Concurrent-Access Anomalies
- ✓ Security Problems

Data Redundancy and Inconsistency

- Same information may be duplicated in several places.
- All copies may not be updated properly.
- Files that represent the same data may become inconsistent.

Difficulty in Accessing Data

- File processing environments do not allow needed data to be retrieved in a convenient and efficient manner.
- May have to write a new application program to satisfy an unusual request.
- E.g. find all customers with the same postal code.
- Could generate this data manually, but a long job.

Data Isolation

- Because data are scattered in various files, and files may be in different formats, writing new application programs to retrieve the appropriate data is difficult.

Integrity Problems

- The data values stored in the database must satisfy certain types of consistency constraints.
- The constraints are enforced by adding appropriate code in programs.
- When new constraints are added it is difficult to change programs to enforce them.

Atomicity Problem

- If any failure occurs the data is to be restored to the consistent state that existed prior to failure.
- It must be atomic happen entirely or not at all.
- It is difficult to ensure atomicity in a conventional file processing system.

Concurrent Access Anomalies

- For the overall performance of the system and faster response, many systems allow multiple users to update the data simultaneously.
- In such an environment, interaction of concurrent updates is possible and may result in inconsistent data.

Security Problems

- Not every user of database system is allowed to access data.
- Ie., Every user of the system should be able to access only the data they are permitted to see.
- Enforcing security constraint is difficult in file processing system.

File Systems vs Database Systems

- DBMS are expensive to create in terms of software, hardware, and time invested.
- The solution is called maintaining data in flat files. So what is bad about flat files?

- ✓ Uncontrolled Redundancy
- ✓ Inconsistent Data
- ✓ Inflexibility
- ✓ Limited Data Sharing
- ✓ Poor Enforcement of Standards
- ✓ Low Programmer Productivity
- ✓ Excessive Program Maintenance
- ✓ Excessive Data Maintenance

File System

- Data is stored in Different Files in forms of Records
- The programs are written time to time as per the requirement to manipulate the data within files.
  - ✓ A program to debit and credit an account
  - ✓ A program to find the balance of an account
  - ✓ A program to generate monthly statements

Advantages of DBMS

- Improved security
- Improved data integrity
- Data consistency
- Improved data accessibility and responsiveness
- Increased concurrency
- Improved backup and recovery services

Disadvantages of DBMS

- Cost of DBMS's
- Complexity and Size
- Higher impact of a failure
- Performance

<u>Characteristics of the Database Approach</u>

Write down the characteristics of database approach.

- The main characteristics of the database approach versus the file-processing approach are the following:
  - ✓ Self-describing nature of a database system
  - ✓ Insulation between programs and data, and data abstraction
  - ✓ Support of multiple views of the data
  - ✓ Sharing of data and multiuser transaction processing

Self-Describing Nature of a Database System

- A fundamental characteristic of the database approach is that the database system contains not only the database itself but also a complete definition or description of the database structure and constraints.

- This definition is stored in the DBMS catalog, which contains information such as the structure of each file, the type and storage format of each data item, and various constraints on the data.
- The information stored in the catalog is called **meta-data**, and it describes the structure of the primary database.

Insulation between Programs and Data and Data Abstraction

- In traditional file processing, the structure of data files is embedded in the application programs, so any changes to the structure of a file may require *changing all programs* that access that file.
- By contrast, DBMS access programs do not require such changes in most cases.
- The structure of data files is stored in the DBMS catalog separately from the access programs.

Support of Multiple Views of the Data

- A database typically has many users, each of whom may require a different perspective or **view** of the database.
- A view may be a subset of the database or it may contain **virtual data** that is derived from the database files but is not explicitly stored.
- A multiuser DBMS whose users have a variety of distinct applications must provide facilities for defining multiple views.

Sharing of Data and Multiuser Transaction Processing

- A multiuser DBMS, as its name implies, must allow multiple users to access the database at the same time.
- This is essential if data for multiple applications is to be integrated and maintained in a single database.
- The DBMS must include **concurrency control** software to ensure that several users trying to update the same data do so in a controlled manner so that the result of the updates is correct.
- A transaction is an *executing program* or *process* that includes one or more database accesses, such as reading or updating of database records.

<u>Database Terminologies</u>

List out some of the terminologies used in database.

Some of the terminologies used in databases are, **Database**

- ✓ It is a collection (or list) of information.
- ✓ A database is comprised of one or more lists (called tables) of data organized by columns, rows and cells.

Tables

- ✓ The view displays the database as a combination of rows (records) and columns (fields).

- ✓ The cells contain the bits and pieces of data for each record in each field.
- ✓ The first row of a table is reserved for the field names.

Key

- ✓ A key is a logical value to access record in a table.
- ✓ A key that uniquely identifies a record is called as primary key.

Field Names

- ✓ Identify the different categories in a database.
- ✓ The top row is reserved for field names.
- ✓ Examples of field names are First name, last name, address, city, state, zip, phone number.

Fields

- ✓ It defines the categories in a database.
- ✓ Fields are displayed in columns.
- ✓ For Example, in a database, the zip field contains all the zip codes from each of the records.
- ✓ These are the bits and pieces of data.

Domain

- ✓ Domain refers to the possible values each field can contain.
- ✓ For example (marital status fields may contain either married or unmarried values.)

View

- ✓ It is a virtual table made up of a subset of the actual tables.

Records

- ✓ These are related information that is separated by columns or fields.
- ✓ A name and address are considered one record in the database.
- ✓ A second Name and address are a different record.

Constraints

- ✓ Constraints are the logic rules that are used to ensure data consistency or avoid certain unacceptable operations on the data.

Cells

- ✓ The intersection of columns and rows that contain the data for each record

Index

- ✓ It is the part of the physical structure.

Data

- ✓ All the records of information in a database including the field names.
- ✓ Data + Field Names = Records
- ✓ All Records = a Database

Information

- ✓ Information is data that is processed to have a meaning.

NULL Value

- ✓ A field is said to be contain a null value when it contains nothing at all.

Data Integrity

- ✓ It describes the accuracy, validity and consistency of data.

Database Normalization

- ✓ It is a technique that helps to reduce the occurrence of data anomalies and poor data integrity.

Objects

- ✓ Enables you to find, view, display and print data differently, based on your needs.
- ✓ The most commonly used objects are tables, queries, forms and reports.
    - Tables show all records in a spreadsheet format.
    - Queries allow you to ask questions of the one or more tables and show only the information you ask for.
    - Forms display one record at a time.
    - Reports give and organize why of presenting information.

Views of Data

Briefly explain about views of data. (May/June 2016)

- A database system is a collection of interrelated data and a set of programs that allow users to access and modify these data.
- A major purpose of a database system is to provide users with an *abstract* view of the data.
- That is, the system hides certain details of how the data are stored and maintained.

Data Abstraction

- **Data abstraction** generally refers to the suppression of details of data organization and storage, and the highlighting of the essential features for an improved understanding of data.
- For the system to be usable, it must retrieve data efficiently.
- The need for efficiency has led designers to use complex data structures to represent data in the database.
- Since many database-system users are not computer trained, developers hide the complexity from users through several levels of abstraction, to simplify user's interactions with the system:

☐ **Physical Level**

- ✓ The lowest level of abstraction describes *how* the data are actually stored.
- ✓ The physical level describes complex low-level data structures in detail.

☐ **Logical Level**

- ✓ The next-higher level of abstraction describes *what* data are stored in the database, and what relationships exist among those data.
- ✓ The logical level thus describes the entire database in terms of a small number of relatively simple structures.

- ✓ Although implementation of the simple structures at the logical level may involve complex physical-level structures, the user of the logical level does not need to be aware of this complexity. This is referred to as **physical data independence**.
- ✓ Database administrators, who must decide what information to keep in the database, use the logical level of abstraction.

type *instructor* = **record**
  ID : **char** (5);
  name : **char** (20);
  dept name : **char** (20);
  salary : **numeric** (8,2);
end;

- ✓ This code defines a new record type called *instructor* with four fields.
- ✓ Each field has a name and a type associated with it.

☐ **View Level**

- ✓ The highest level of abstraction describes only part of the entire database.
- ✓ Even though the logical level uses simpler structures, complexity remains because of the variety of information stored in a large database.
- ✓ Many users of the database system do not need all this information; instead, they need to access only a part of the database.
- ✓ The view level of abstraction exists to simplify their interaction with the system.
- ✓ The system may provide many views for the same database.

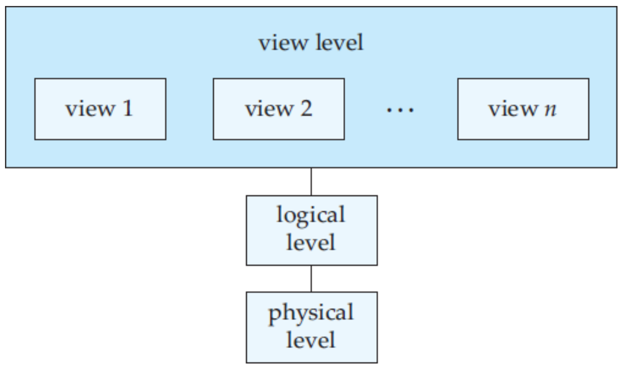


Figure: The Three Levels of Data Abstraction

- A university organization may have several such record types, including
  - ✓ *department*, with fields *dept name*, *building*, and *budget*
  - ✓ *course*, with fields *course id*, *title*, *dept name*, and *credits*
  - ✓ *student*, with fields *ID*, *name*, *dept name*, and *tot cred*
- At the physical level, an *instructor*, *department*, or *student* record can be described as a block of consecutive storage locations. The compiler hides this level of detail from programmers. Similarly, the database system hides many of the lowest-level storage details from database programmers.

- At the logical level, each such record is described by a type definition, as in the previous code segment, and the interrelationship of these record types is defined as well. Programmers using a programming language work at this level of abstraction.
- Finally, at the view level, computer users see a set of application programs that hide details of the data types. At the view level, several views of the database are defined, and a database user sees some or all of these views.
- In addition to hiding details of the logical level of the database, the views also provide a security mechanism to prevent users from accessing certain parts of the database.

Instances and Schemas

Write short notes on instance and schema.

- Databases change over time as information is inserted and deleted.
- The collection of information stored in the database at a particular moment is called an **instance** of the database.
- The overall design of the database is called the database **schema**.
- The concept of database schemas and instances can be understood by analogy to a program written in a programming language.
- A database schema corresponds to the variable declarations (along with associated type definitions) in a program.
- Each variable has a particular value at a given instant.
- The values of the variables in a program at a point in time correspond to an *instance* of a database schema.
- Database systems have several schemas, partitioned according to the levels of abstraction.
- The **physical schema** describes the database design at the physical level, while the **logical schema** describes the database design at the logical level.
- A database may also have several schemas at the view level, sometimes called **subschemas**, that describe different views of the database.

Data Models

Write short notes on data model and its types. (Nov/Dec 2014)

- A collection of conceptual tools for describing data, data relationships, data semantics and consistency constraints.
- A data model provides a way to describe the design of a database at the physical, logical, and view levels.

(Or)

- A **data model** is a collection of concepts that can be used to describe the structure of a database.
- It also includes a set of **basic operations** for specifying retrievals and updates on the database.
- The basic operations provided by the data model, include concepts in the data model to specify the **dynamic aspect** or **behavior** of a database application.

Categories of Data Models

- Data models can be categorized according to the types of concepts they use to describe the database structure.
- **High-level** or **conceptual data models** provide concepts that are close to the way many users perceive Data.
- **Low-level** or **physical data models** provide concepts that describe the details of howdata is stored on the computer storage media.
- **Representational** (or **implementation**) **data models**, which provide concepts that maybe easily understood by end users.
- Representational data models hide many details of data storage on disk but can be implemented on a computer system directly.

(Or)

The data models can be classified into four different categories:

Relational Model

- The relational model uses a collection of tables to represent both data and the relationships among those data.
- Each table has multiple columns, and each column has a unique name. Tables are also known as **relations**.
- The relational model is an example of a record-based model.
- Record-based models are so named because the database is structured in fixed-format records of several types.
- Each table contains records of a particular type.
- Each record type defines a fixed number of fields, or attributes.
- The columns of the table correspond to the attributes of the record type.
- The relational data model is the most widely used data model, and a vast majorityof current database systems are based on the relational model.

Entity-Relationship Model

- The entity-relationship (E-R) data model uses a collection of basic objects, called *entities*, and *relationships* among theseobjects.
- An entity is a ―thing‖ or ―object‖ in the real world that is distinguishable from other objects.
- The entity-relationship model is widely used in database design.

Object-Based Data Model

- Object-oriented programming (especially in Java, C++, or C#) has become the dominant software-development methodology.
- This led to the development of an object-oriented data model that can be seen as extending the E-R model with notions of encapsulation, methods (functions), and object identity.
- The object-relational data model combines features of the object-oriented data model and relational data model.

Semistructured Data Model

- The semistructured data model permits the specification of data where individual data items of the same type may have different sets of attributes.
- This is in contrast to the data models mentioned earlier, where every data item of a particular type must have the same set of attributes.
- The **Extensible Markup Language (XML)** is widely used to represent semistructured data.

Database Languages

Explain in detail about database languages.

- A database system provides a **data-definition language** to specify the database schema and a **data-manipulation language** to express database queries and updates.

Data-Manipulation Language

- A **data-manipulation language (DML)** is a language that enables users to access or manipulate data as organized by the appropriate data model.
- The types of access are:
  - ✓ Retrieval of information stored in the database
  - ✓ Insertion of new information into the database
  - ✓ Deletion of information from the database
  - ✓ Modification of information stored in the database
- There are basically two types:
  - ✓ **Procedural DMLs** require a user to specify *what* data are needed and *how* to get those data.
  - ✓ **Declarative DMLs** (also referred to as **nonprocedural DMLs**) require a userto specify *what* data are needed *without* specifying how to get those data.
- Declarative DMLs are usually easier to learn and use than are procedural DMLs.
- A **query** is a statement requesting the retrieval of information.
- The portion of a DML that involves information retrieval is called a **query language**.
- There are a number of database query languages in use, either commercially or experimentally.
- We study the most widely used query language, SQL.

Data-Definition Language

- We specify a database schema by a set of definitions expressed by a special language called a **data-definition language** (**DDL**).
- The DDL is also used to specify additional properties of the data.
- We specify the storage structure and access methods used by the database system by a set of statements in a special type of DDL called a **data storage and definition** language.
- These statements define the implementation details of the database schemas, which are usually hidden from the users.
- The data values stored in the database must satisfy certain **consistency constraints**.

- For example, suppose the university requires that the account balance of a department must never be negative.
- The DDL provides facilities to specify such constraints.
- In general, a constraint can be an arbitrary predicate pertaining to the database.

Domain Constraints

- A domain of possible values must be associated with every attribute (for example, integer types, character types, date/time types).
- Declaring an attribute to be of a particular domain acts as a constraint on the values that it can take.
- Domain constraints are the most elementary form of integrity constraint.
- They are tested easily by the system whenever a new data item is entered into the database.

Referential Integrity

- There are cases where we wish to ensure that a value that appears in one relation for a given set of attributes also appears in a certain set of attributes in another relation (referential integrity).
- For example, the department listed for each course must be one that actually exists.
- More precisely, the *dept name* value in a *course* record must appear in the *dept name* attribute of some record of the *department* relation.
- Database modifications can cause violations of referential integrity.
- When a referential-integrity constraint is violated, the normal procedure is to reject the action that caused the violation.

Assertions

- An assertion is any condition that the database must always satisfy.
- Domain constraints and referential-integrity constraints are special forms of assertions.
- However, there are many constraints that we cannot express by using only these special forms.
- For example, ―Every department must have at least five courses offered every semester‖ must be expressed as an assertion.
- When an assertion is created, the system tests it for validity.
- If the assertion is valid, then any future modification to the database is allowed only if it does not cause that assertion to be violated.

Authorization

- We may want to differentiate among the users as far as the type of access they are permitted on various data values in the database.
- These differentiations are expressed in terms of **authorization**.
    - ✓ **Read Authorization** - It allows reading, but not modification of data.
    - ✓ **Insert Authorization** - It allows insertion of new data, but not modification of existing data
    - ✓ **Update Authorization**- It allows modification, but not deletion of data.

- ✓ **Delete Authorization** - It allows deletion of data.

Database Designers

- **Database designers** are responsible for identifying the data to be stored in the database and for choosing appropriate structures to represent and store this data.
- These tasks are mostly undertaken before the database is actually implemented and populated with data.
- It is the responsibility of database designers to communicate with all prospective database users in order to understand their requirements and to create a design that meets these requirements.

Database Design for a University Organization

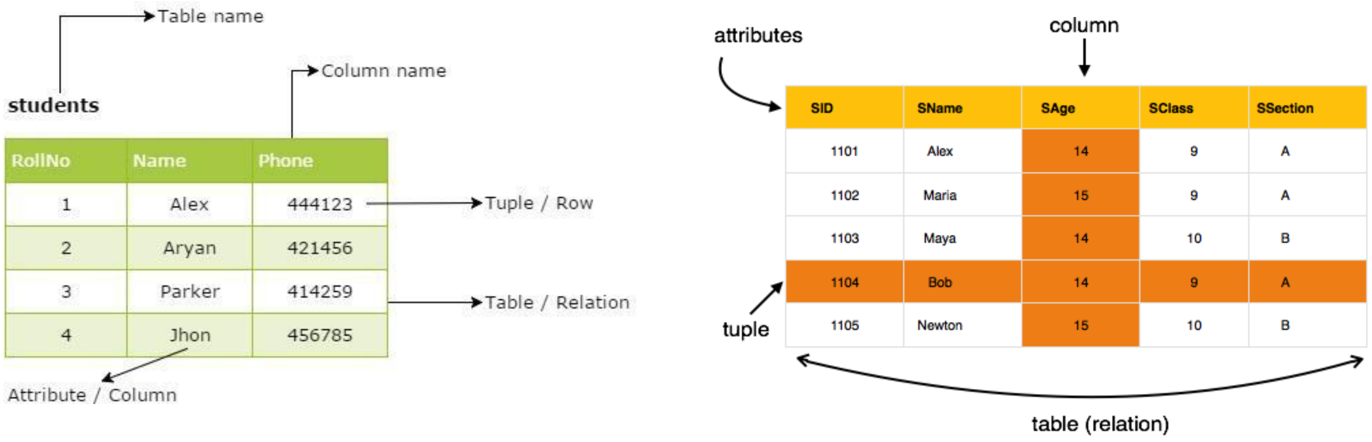
- Let us examine how a database for a university could be designed.
- The university is organized into departments. Each department is identified by a unique name (*dept name*), is located in a particular *building*, and has a *budget*.
- Each department has a list of courses it offers. Each course has associated with it a *course id*, *title*, *dept name*, and *credits*, and may also have associated *prerequisites*.
- Instructors are identified by their unique *ID*. Each instructor has *name*, associated department (*dept name*), and*salary*.
- Students are identified by their unique *ID*. Each student has a *name*, an associated major department (*dept name*), and *tot cred* (total credit hours the student earned thus far).
- The university maintains a list of classrooms, specifying the name of the *building*, *room number*, and room *capacity*.
- The university maintains a list of all classes (sections) taught. Each section is identified by a *course id*, *sec id*, *year*, and *semester*, and has associated with it a *semester*, *year*, *building*, *room number*, and *time slot id* (the time slot when the class meets).
- The department has a list of teaching assignments specifying, for each instructor, the sections the instructor is teaching.
- The university has a list of all student course registrations, specifying, for each student, the courses and the associated sections that the student has taken (registered for).

Record Based Data Models

Relational Data Model

- The relational model uses a collection of tables to represent both data and the relationships among those data.
- Each table has multiple columns, and each column has a unique name.
- The data is arranged in a relation which is visually represented in a two dimensional table.
- The data is inserted into the table in the form of tuples (which are nothing but rows).
- A tuple is formed by one or more than one attributes, which are used as basic building blocks in the formation of various expressions that are used to derive meaningful information.

- The relational model is implemented in databasewhere,
    - ✓ A relation is represented by atable.
    - ✓ A tuple is represented by a row.
    - ✓ An attribute is represented by a column of the table.
    - ✓ Attributenameisthenameofthecolumnsuchas_identifier‘,_name‘,_city‘etc.,
    - ✓ Attribute value contains the value for column in the row.
- It is example for record based model because the database is structured in fixed format records of several types.



RollNo	Name	Phone
1	Alex	444123
2	Aryan	421456
3	Parker	414259
4	Jhon	456785

SID	SName	SAge	SClass	SSection
1101	Alex	14	9	A
1102	Maria	15	9	A
1103	Maya	14	10	B
1104	Bob	14	9	A
1105	Newton	15	10	B

Figure: Relational Model

Network Data Model

- In network model the data are represented by collection of records and their relationship is represented by links.
- Network database consists of collection of records connected to one another through links.
- Each record is a collection of fields or attributes & each of which contain only one data value.
- A link is an associated between two records.

Example:

Customer record is defined as,

```
Type customer = record
        Customer_name:string;
        Customer_street:string;
        Customer_city:string;
End
```

- Account records is defined as,

```
Type account = record
        Acc_number : string;
        Balance :integer;
End
```

- In network model the two records are represented as,

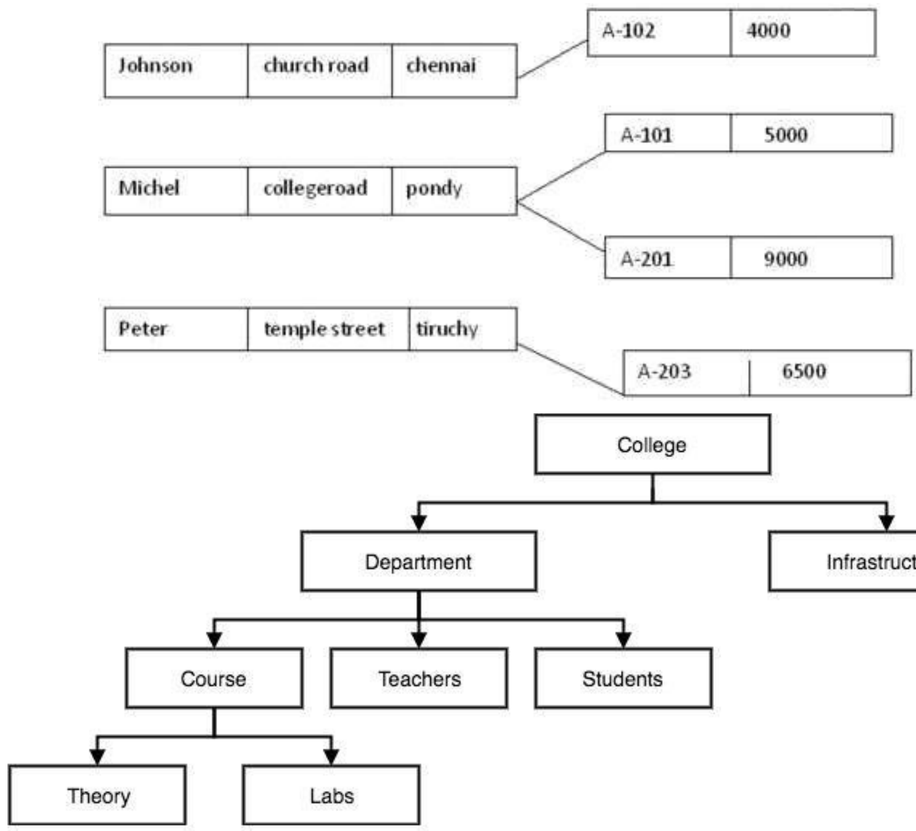


Figure: Network Model

Hierarchical Data Model

- Hierarchical model consists of a collection of records that are connected to each other through links records are organized as a collection of trees.
- The hierarchical database model looks like an organizational chart or a family tree. It has a single root segment (Employee) connected to lower level segments (Compensation,Job Assignments and Benefits).
- Each subordinate segment in turn, may connect to other subordinate segments.
- Here, compensation connects to Performance Ratings and Salary History.
- Benefits connect to Pension, Life Insurance and Health.
- Each subordinate segment is the child of the segment directly above it.

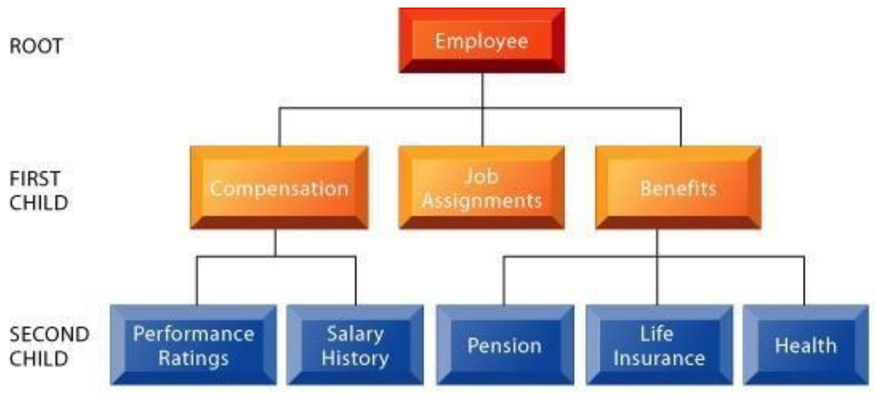


Figure: A Hierarchical data model for Human Resource System

Object-Oriented Data Model / Object Based Data Model

- The data is stored in the form of objects, which are structures called *classes* that display the data within.

- The fields are instances of these classes.
- Object oriented data model is extending the E-R model with notions of encapsulation, methods and object identity.
- Object oriented data model also supports a rich type system including structured and collection types.
- Object relational data model, a data model that combines features of the object-oriented data model and relational data model.

Semi-structured Data Model

- **Semi structured data models** permit the specification of data where individual data items of the same type may have different sets of attributes.
- This is in contrast with the data models mentioned earlier, where every data item of a particular type must have the same set of attributes.
- **The extensible markup language (XML)** is widely used to represent semi-structured data.

Database System Architecture

With help of a neat block diagram explain the basic architecture of a database management system. (Nov/Dec 2015) (Or) Briefly explain about database system architecture. (May/June 2016) (Or) State and explain the architecture of DBMS. (Nov/Dec 2017)

- The architecture of a database system is greatly influenced by the underlying computer system on which the database system runs.
- Database systems can be centralized, or client-server, where one server machine executes work on behalf of multiple client machines.

Components of DBMS

Explain the components of database in detail.

- Database system is partitioned into modules that deal with each of the responsibilities of the overall system.
- The functional components of the database system are,
  - ✓ Storage Manager
  - ✓ Query Processor

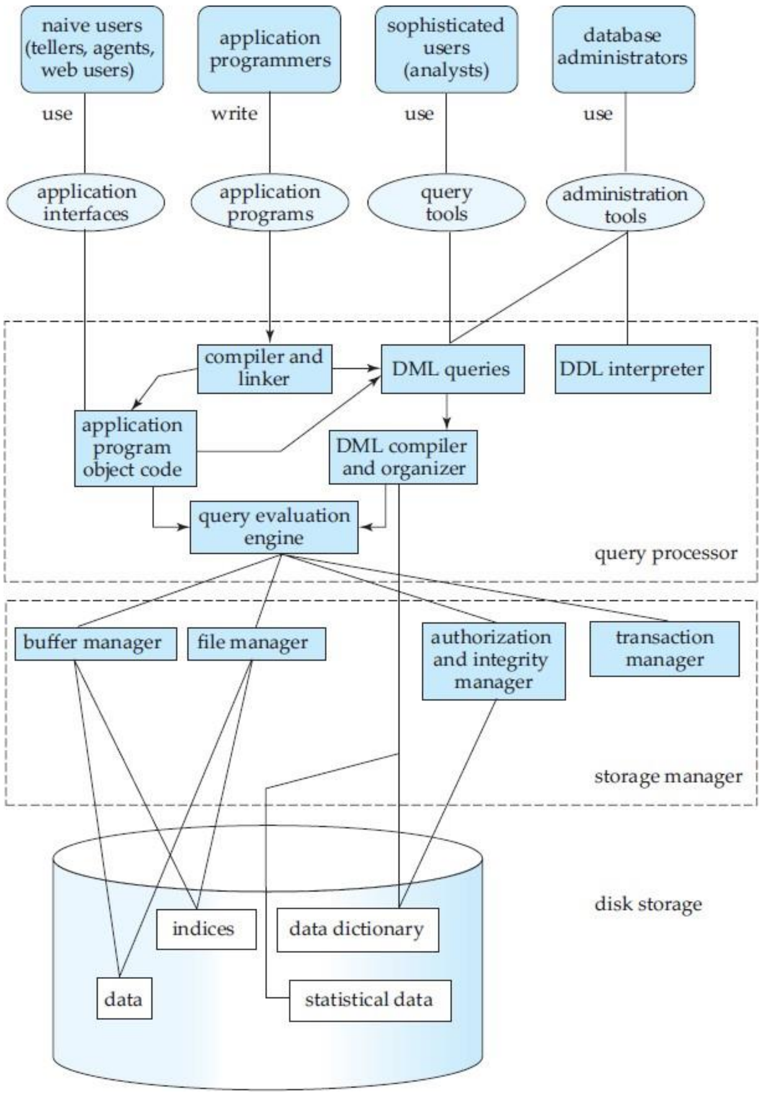


Figure: Database System Structure

Storage Manager

- It is a component of database system that provides the interface between the low- level data stored in the database and theapplication programs and queries submitted to thesystem.
- It is responsible for the interaction with the file manager.
- Therawdataarestoredonthediskusingthefilesystemprovidedbytheoperatingsystem.
- The storage manager translates the various DML statements into low-level file-system commands.
- The components of storage manager are,
    - ✓ Authentication & IntegrityManager
    - ✓ File Manager
    - ✓ Buffer Manager
    - ✓ Transaction Manager

Authorization & Integrity Manager

- It tests for satisfaction of integrity constraints and checks the authority of users to access data.

File Manager

- It manages the allocation of space on disk storage and the data structures used to represent information stored on disk.

Buffer Manager

- It is responsibility for fetching data from disk storage to main memory & deciding what data to cache in main memory.

Transaction Manager

- It ensures that database remains in a consistent state despite system failure and the concurrent transaction executions proceed without conflicting.
- It consists of the concurrency control manager and the recovery manager.
- The Four Properties of Transactions are,
  - ✓ **Atomicity**
    - This means that either all of the instructions within the transaction will be reflected in the database, or none of them will be reflected.
  - ✓ **Consistency**
    - If we execute a particular transaction in isolation or together with other transaction, (i.e. imagine in a multi-programming environment), the transaction will yield the same expected result.
  - ✓ **Isolation**
    - In case multiple transactions are executing concurrently and trying to access a sharable resource at the same time, the system should create an ordering in their execution so that they should not create any anomaly in the value stored at the sharable resource.
  - ✓ **Durability**
    - It states that once a transaction has been completed, the changes it has made should be permanent.
- The storage manager implements several data structure such as part of the physical system implementation:,
  - ✓ Data Files – Stores the database itself.
  - ✓ Data Dictionary - Stores the metadata about the structure of the database (i.e) Schema of the database.
  - ✓ Indices - It provides fast access to data items. The Database provides pointers to those data items that hold a particular index value.

<u>Query Processor</u>

- It helps the database system to simplify and facilitate access to data components of query processor.
- The Components of query processor includes:
  - ✓ DDL Interpreter
  - ✓ DML Compiler

- ✓ Query Optimization
- ✓ Query Evaluation Engine

DDL Interpreter

- Interprets DDL statements and records the definitions in data dictionary.

DML Compiler

- Translates DML statements into an evaluation plan consisting of low-level instruction that the query evaluation engine understands.
- It also performs query optimization (i.e) it picks the lowest cost evaluation plan from among the alternates.

Query Evaluation Engine

- Executes low-level instructions generated by the DML compiler.
- Databasesystems can be centralized as client –server.
- Based on this database applications are portioned into
  - ✓ Two Tier Architecture
  - ✓ Three tier Architecture

Two-tier Architecture

- Application resides at the client machine where it invokes database system functionality at the server machine through query language statements.
- E.g. client programs using ODBC/JDBC to communicate with a database.

Three-tier Architecture

- Client machine acts as merely a front end and does not contain any directdatabase calls.
- The client end communicates with an application server through forms interface and the application server in turn communicates with the databasesystem to access data.
- E.g. web-based applications and applications built using ―middleware‖.

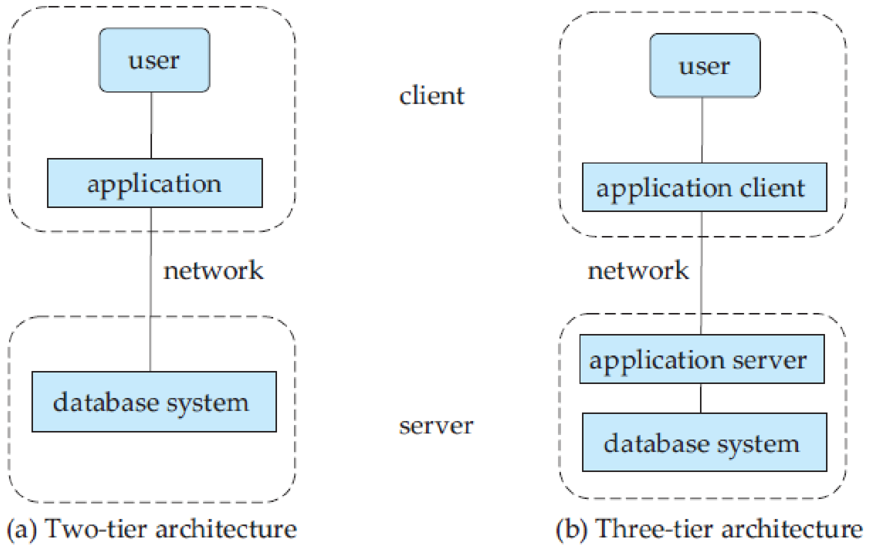


Figure: Two-tier and Three-tier Architectures

<u>Transaction Management</u>

Write short notes on transaction management.

- 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.
- 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.
- Clearly, it is essential that either both the credit and debit occur, or that neither occur.
- That is, the funds transfer must happen in its entirety or not at all. This all-or-none requirement is called **atomicity**.
- In addition, it is essential that the execution of the funds transfer preserve the consistency of the database.
- That is, the value of the sum of the balances of *A* and *B* must be preserved.
- This correctness requirement is called **consistency**.
- Finally, after the successful execution of a funds transfer, the new values of the balances of accounts *A* and *B* must persist, despite the possibility of system failure. This persistence requirement is called **durability**.

Recovery Manager

- It is the responsibility of the database system itself especially, the recovery manager to ensuring the atomicity and durability properties.

Failure Recovery

- It detects system failures and restores the database to the state that existed prior to the occurrence of the failure.

Concurrency-Control Manager

- It is responsible to control the interaction among the concurrent transactions, to ensure the consistency of the database.
- The **transaction manager** consists of the concurrency-control manager and the recovery manager.

<u>Database Users and Administrators</u>

- A primary goal of a database system is to retrieve information from and store new information into the database.
- There are four different types of database-system users, differentiated by the way they expect to interact with the system.
- Different types of user interfaces have been designed for the different types of users.

Naive Users

- They are unsophisticated users who interact with the system by invoking one of the application programs that have been written previously.

Application Programmers

- They are computer professionals who write application programs.
- Application programmers can choose from many tools to develop user interfaces.
- **Rapid application development (RAD)** tools are tools that enable an application programmer to construct forms and reports with minimal programming effort.

Sophisticated Users

- They interact with the system without writing programs.
- Instead, they form their requests either using a database query language or by using tools such as data analysis software.

Specialized Users

- They are sophisticated users who write specialized database applications that do not fit into the traditional data-processing framework.
- Among these applications are computer-aided design systems, knowledgebase and expert systems, systems that store data with complex data types (for example, graphics data and audio data), and environment-modeling systems.

Database Administrators

- In a database environment, the primary resource is the database itself, and the secondary resource is the DBMS and related software.
- Administering these resources is the responsibility of the **database administrator (DBA)**.
- The DBA is responsible for authorizing access to the database, coordinating and monitoring its use, and acquiring software and hardware resources as needed.

(Or)

- DBMS's is to have central control of both the data and the programs that access those data.
- A person who has such central control over the system is called a **database administrator (DBA)**.
- The functions of a DBA include:
- **Schema Definition**
  - ✓ The DBA creates the original database schema by executing a set of data definition statements in the DDL.
- **Storage Structure and Access-method Definition**
- **Schema and Physical-organization Modification**
  - ✓ The DBA carries out changes to the schema and physical organization to reflect the changing needs of the organization, or to alter the physical organization to improve performance.
- **Granting of Authorization for Data Access**

- ✓ By granting different types of authorization, the database administrator can regulate which parts of the database various users can access.
- ✓ The authorization information is kept in a special system structure that the database system consults whenever someone attempts to access the data in the system.

Routine Maintenance

- Examples of the database administrator's routine maintenance activities are:
  - ✓ Periodically backing up the database, either onto tapes or onto remote servers, to prevent loss of data in case of disasters such as flooding.
  - ✓ Ensuring that enough free disk space is available for normal operations, and upgrading disk space as required.
  - ✓ Monitoring jobs running on the database and ensuring that performance is not degraded by very expensive tasks submitted by some users.

<u>Introduction to Relational Databases</u>

Explain relational DBMS in detail.

- A Relational Database management System (RDBMS) is a database management system based on relational model introduced by E.F Codd.
- In relational model, data is represented in terms of tuples (rows).
- RDBMS is used to manageRelational database.
- Relational database is a collection of organized set of tables from which data can be accessed easily.
- Relational Database is most commonly used database.
- It consists of number of tables and each table has its own primary key.
- RDBMSs are a common choice for the storage of information in new databases used for financial records, manufacturing and logistical information, personnel data and other applications since the 1980s.
- A data model is a collection of conceptual tools for describing data, data relationships, data semantics and consistency constraints.
- A relational database is based on the relational model which uses a collection of tables to represent both data and the relationships among those data.
- It also includes a DML and DDL.
- A software system used to maintain relational databases is a relational database management system (RDBMS).
- Virtually all relational database systems use SQL (Structured Query Language) for querying and maintaining the database.

<u>Relational Model</u>

- The relational model is today the primary data model for commercial data processing applications.
- This model organizes data into one or more tables (or "relations") of columns and rows, with a unique key identifying each row.

- Rows are also called records or tuples. Columns are also called attributes.
- Generally, each table/relation represents one "entity type" (such as customer or product).
- The rows represent instances of that type of entity (such as "Lee" or "chair") and the columns representing values attributed to that instance (such as address or price).

Keys

Write short notes on keys in DBMS.

- It is defined as one or more columns in a database table that is used to sort and/or identify rows in a table.
- e.g. if you were sorting people by the field salary then the salary field is the key.
- It also establishes relationship among tables.

Types of keys in DBMS

- **Primary Key** – A primary is a column or set of columns in a table that uniquely identifies tuples (rows) in that table. The primary key cannot be null (blank). The primary key is indexed.
- **Super Key** – A super key is a set of one of more columns (attributes) to uniquely identify rows in a table.
- **Candidate Key** – A super key with no redundant attribute is known as candidate key
- **Alternate Key** – Out of all candidate keys, only one gets selected as primary key, remaining keys are known as alternate or secondary keys.
- **Composite Key** – A key that consists of more than one attribute to uniquely identify rows (also known as records & tuples) in a table is called composite key.
- **Foreign Key** – Foreign keys are the columns of a table that points to the primary key of another table. They act as a cross-reference between tables.

Relational Database Characteristics

- Data in the relational database must be represented in tables, with values in columns within rows.
- Data within a column must be accessible by specifying the table name, the column name, and the value of the primary key of the row.
- The DBMS must support missing and inapplicable information in a systematic way, distinct from regular values and independent of data type.
- The DBMS must support an active on-line catalogue.
- The DBMS must support at least one language that can be used independently and from within programs, and supports data definition operations, data manipulation, constraints and transaction management.
- Views must be updatable by the system
- The DBMS must support insert, update, and delete operations on sets.

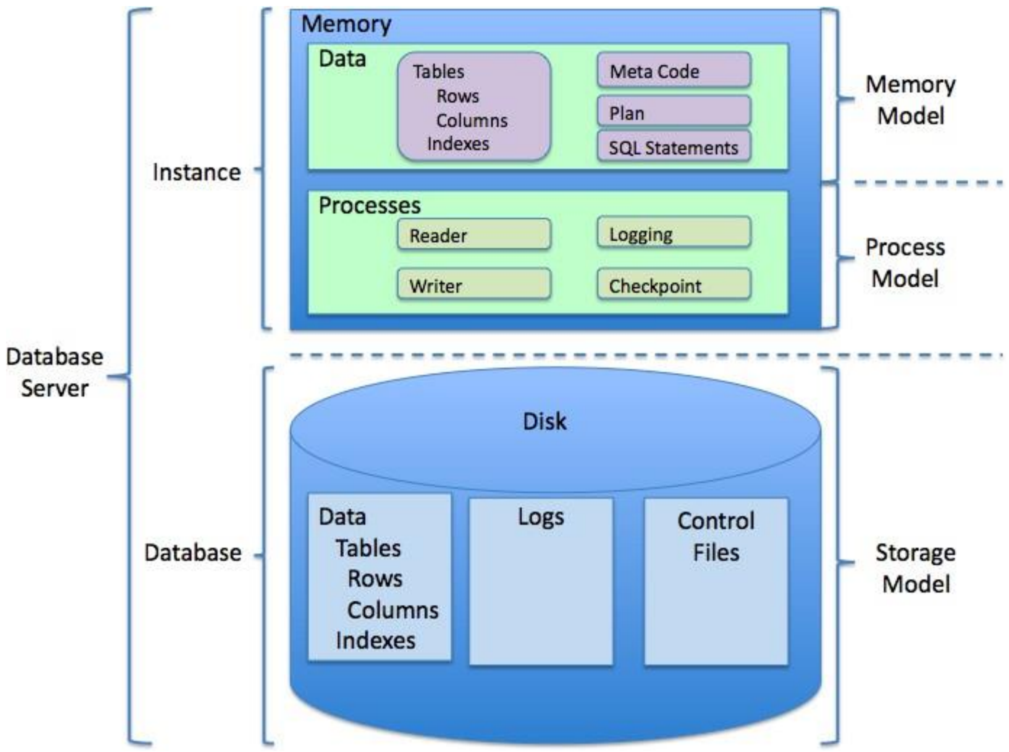


Figure: Relational DBMS (RDBMS)

- The DBMS must support physical and logical data independence.
- Integrity constraints must be stored within the catalogue, separate from the application.
- The DBMS must support distribution independence.
- The existing application should run when the existing data is redistributed or when the DBMS is redistributed.
- If the DBMS provides a low level interface (row at a time), that interface cannot bypass the integrity constraints.

<u>CODD'S RULE</u>

Explain Codd's rule in detail.

- Dr Edgar F. Codd did some extensive research in Relational Model of database systems and came up with twelve rules of his own which according to him, a database must obey in order to be a true relational database.
- These rules can be applied on a database system that is capable of managing is stored data using only its relational capabilities. This is a foundation rule, which provides a base to implyother rules on it.

Rule Zero

- This rule states that for a system to qualify as an **RDBMS**, it must be able to manage database entirely through the relational capabilities.

Rule 1: Information Rule

- This rule states that all information (data), which is stored in the database, must be a value of some table cell.
- Everything in a database must be stored in table formats. This information can be user data or meta-data.

Rule 2: Guaranteed Access Rule

- This rule states that every single data element (value) is guaranteed to be accessible logically with combination of table-name, primary-key (row value) and attribute-name (column value).
- No other means, such as pointers, can be used to access data.

Rule 3: Systematic Treatment of NULL Values

- This rule states the NULL values in the database must be given a systematic treatment.
- As a NULL may have several meanings, i.e. NULL can be interpreted as one the following: data is missing, data is not known, data is not applicable etc.

Rule 4: Active Online Catalog

- This rule states that the structure description of whole database must be stored in an online catalog, i.e. data dictionary, which can be accessed by the authorized users.
- Users can use the same query language to access the catalog which they use to access the database itself.

Rule 5: Comprehensive Data Sub-language Rule

- This rule states that a database must have a support for a language which has linear syntax which is capable of data definition, data manipulation and transaction management operations.
- Database can be accessed by means of this language only, either directly or by means of some application.
- If the database can be accessed or manipulated in some way without any help of this language, it is then a violation.

Rule 6: View Updating Rule

- This rule states that all views of database, which can theoretically be updated, must also be updatable by the system.

Rule 7: High-level Insert, Update and Delete Rule

- This rule states the database must employ support high-level insertion, updation and deletion.
- This must not be limited to a single row that is, it must also support union, intersection and minus operations to yield sets of data records.

Rule 8: Physical Data Independence

- This rule states that the application should not have any concern about how the data is physically stored.
- Also, any change in its physical structure must not have any impact on application.

Rule 9: Logical Data Independence

- This rule states that the logical data must be independent of its user's view (application). Any change in logical data must not imply any change in the application using it.
- For example, if two tables are merged or one is split into two different tables, there should be no impact the change on user application. This is one of the most difficult rules to apply.

Rule 10: Integrity Independence

- This rule states that the database must be independent of the application using it.
- All its integrity constraints can be independently modified without the need of any change in the application.
- This rule makes database independent of the front-end application and its interface.

Relational Query Languages

- A **query language** is a language in which a user requests information from the database.
- These languages are usually on a level higher than that of a standard programming language.
- Query languages can be categorized as either procedural or nonprocedural.
  - ✓ In a **procedural language**, the user instructs the system to perform a sequence of operations on the database to compute the desired result.
  - ✓ In a **nonprocedural language**, the user describes the desired information without giving a specific procedure for obtaining that information.
- There are a number of ―pure‖ query languages:
  - ✓ The relational algebra is procedural, whereas the tuple relational calculus and domain relational calculus are nonprocedural.

Relational Algebra

Explain the concept of relational algebra in detail. (Or) Explain select, project and Cartesian product operations in relational algebra with an example. (Nov/Dec 2016, April/May 2018)

- The relational algebra is a theoretical procedural query language which takes an instance of relations and does operations on one or more relations to describe another relation without altering the original relation(s).
- The relational algebra defines a set of operations on relations, paralleling the usual algebraic operations such as addition, subtraction or multiplication, which operate on numbers.
- Just as algebraic operations on numbers, the relational algebra consists of a set of operations that take one or two relations as input and produce a new relation as their result.

Operations of Relational Algebra

Unary Operations

- Select [ρ]
- Project [∏]
- Rename [ϭ]

Binary Operations

- Union [U]
- Set Difference [**-**]
- Cartesian Product [**X**]

Unary Operations

Select Operation (σ)

- It selects tuples that satisfy the given predicate from a relation.

Notation – $\sigma_p(r)$

- Where **σ** stands for selection predicate and **r** stands for relation. *p* is prepositional logic formula which may use connectors like **and, or** and **not**.
- These terms may use relational operators like – =, ≠, ≥, < , >, ≤.

Example

$\sigma_{subject = "database"}$**(Books)**

Output

- Selects tuples from books where subject is 'database'.

$\sigma_{subject = "database" \text{ and } price = "450"}$**(Books)**

Output

- Selects tuples from books where subject is 'database' and 'price' is 450.

$\sigma_{subject = "database" \text{ and } price = "450" \text{ or } year > "2010"}$**(Books)**

Output

- Selects tuples from books where subject is 'database' and 'price' is 450 or those books published after 2010.

Project Operation (∏)

- It projects column(s) that satisfy a given predicate.

Notation – $\prod_{A1, A2, An}(r)$

- Where A_1, A_2, A_n are attribute names of relation **r**.
- Duplicate rows are automatically eliminated, as relation is a set.

Example

$\prod_{subject, author}$ **(Books)**

- Selects and projects columns named as subject and author from the relation Books.

Union Operation (∪)

- It performs binary union between two given relations and is defined as –

$r \cup s = \{ t \mid t \in r \text{ or } t \in s\}$

Notation – r Us

- Where **r** and **s** are either database relations or relation result set (temporary relation).
- For a union operation to be valid, the following conditions must hold –
  - ✓ **r** and **s** must have the same number of attributes.
  - ✓ Attribute domains must be compatible.
  - ✓ Duplicate tuples are automatically eliminated.

$\prod_{author}$ **(Books)** ∪ $\prod_{author}$ **(Articles)**

Output

- Projects the names of the authors who have either written a book or an article or both.

Binary Operations

Set Difference (−)

- The result of set difference operation is tuples, which are present in one relation but are not in the second relation.

Notation − **r** − **s**

- It finds all the tuples that are present in **r** but not in **s**.

  $\prod_{author}$ **(Books) −** $\prod_{author}$ **(Articles)**

Output

- Provides the name of authors who have written books but not articles.

Cartesian Product (X)

- It combines information of two different relations into one.

Notation − r X s

- where **r** and **s** are relations and their output will be defined as − r X s = { q t | q ∈ r and t ∈ s}

  $\sigma_{author = 'tutorialspoint'}$**(Books X Articles)**

Output

- Yields a relation, which shows all the books and articles written by tutorialspoint.

Rename Operation (ρ)

- The results of relational algebra are also relations but without any name.
- The rename operation allows us to rename the output relation. 'rename' operation is denoted with small Greek letter **rho** ρ.

Notation − ρ_x (E)

- Where the result of expression **E** is saved with name of **x**. Additional

operations are−

- Set Intersection
- Assignment
- Natural join

Relational Calculus

- In contrast to Relational Algebra, Relational Calculus is a non-procedural query language, that is, it tells what to do but never explains how to do it.
- Relational calculus exists in two forms −
  - ✓ Tuple Relational Calculus(TRC)
  - ✓ Domain Relational Calculus (DRC)

Tuple Relational Calculus (TRC)

- Filtering variable ranges over tuples

Notation − {T |Condition}

- Returns all tuples T that satisfies a condition.

Example

{ T.name | Author(T) AND T.article = 'database' }

Output

- Returns tuples with 'name' from Author who has written article on 'database'.
- TRC can be quantified. We can use Existential (∃) and Universal Quantifiers (∀).

Example

{ R| ∃ T ∊ Authors(T.article='database' AND R.name=T.name)}

Output

- The above query will yield the same result as the previous one.

Domain Relational Calculus (DRC)

- In DRC, the filtering variable uses the domain of attributes instead of entire tuple values (as done in TRC, mentioned above).

Notation – {a_1, a_2, a_3, ..., a_n | P (a_1, a_2, a_3, ... ,a_n)}

- Where a1, a2 are attributes and **P** stands for formulae built by inner attributes.

Example

{< article, page, subject > | ∊ TutorialsPoint ∧ subject = 'database'}

Output

- Yields Article, Page, and Subject from the relation TutorialsPoint, where subject is database.
- Just like TRC, DRC can also be written using existential and universal quantifiers. DRC also involves relational operators.
- The expression power of Tuple Relation Calculus and Domain Relation Calculus is equivalent to Relational Algebra.

SQL Fundamentals

What is SQL? (Or) Explain detail about the fundamentals of SQL.

- SQL stands for Structured Query Language.
- SQL is a standard language for accessing and manipulating databases.
- The tasks related to relational data management— creating tables, querying the database for information, modifying the data in the database, deleting them, granting access to users, and so on.

(Or)

- SQL stands for Structured Query Language.
- It is a programming language which stores, manipulates and retrieves the stored data in RDBMS.
- SQL syntax is not case sensitive.
- SQL is standardized by both ANSI and ISO.
- It is a standard language for accessing and manipulating databases.

Characteristics of SQL

- SQL is extremely flexible.
- SQL uses a free form syntax that gives the ability to user to structure the SQL statements in a best suited way.
- It is a high level language.
- It receives natural extensions to its functional capabilities.
- It can execute queries against the database.

Advantages of SQL

- SQL provides a greater degree of abstraction than procedural language.
- It is coded without embedded data-navigational instructions.
- It enables the end users to deal with a number of database management systems where it is available.
- It retrieves quickly and efficiently huge amount of records from a database.
- No coding required while using standard SQL.

Roles of SQL

- SQL retrieves data from the database. It is **an interactive query language.**
- It can be used along with programming language to access data from database. It is **a database programming language.**
- It can be used to monitor and control data access by various users. It is **a database administration language.**
- It can be used as **an Internet data access language.**

SQL Datatypes

The SQL standard supports a variety of built-in domain types, including:

- **char(*n*)**: A fixed-length character string with user-specified length *n*. The full form, **character** can be used instead.
- **varchar(*n*)**: A variable-length character string with user-specified maximum length n. The full form, **character varying**, is equivalent.
- **int**: An integer (a finite subset of the integers that is machine dependent). The full form, **integer**, is equivalent.
- **smallint**: A small integer (a machine-dependent subset of the integer domain type).
- **numeric(*p*,*d*)**:
    - ✓ A fixed-point number with user-specified precision.
    - ✓ The number consists of *p* digits (plus a sign), and *d* of the *p* digits are to the right of the decimal point.
    - ✓ Thus, **numeric**(3,1) allows 44.5 to be stored exactly, but neither 444.5 or 0.32 can be stored exactly in a field of this type.
- **real, double precision**: Floating-point and double precision floating-point numbers with machine-dependent precision.
- **float(*n*)**: A floating-point number, with precision of at least n digits.

- **date**: a calender date containing a (four-digit) year, month, and day of the month.
- **time**:
  - ✓ The time of day, in hours, minutes and seconds.
  - ✓ A variant, **time**(*p*), can be used to specify the number of fractional digits for seconds (the default being 0).
  - ✓ It is also possible to store time zone information along with the time.
- **timestamp**: A combination of **date** and **time**. A variant, **timestamp**(*p*), can be used to specify the number of fractional digits for seconds (the default here being 6).
- Date and time values can be specified like this:

  date _2001-01-24'
  time '09:30:00'
  timestamp _2001-04-25 10:29:01.45'
- Dates must be specified in the format year followed by month followed by day, as shown.

SQL Languages

State and explain the command DDL, DML, DCL with suitable example. (Nov/Dec 2017)

SQL Command Types

- SQL commands can be divided into two main sub-languages.
- **Data Definition Language (DDL)**
  - ✓ It contains the commands used to create and destroy databases and database objects.
- **Data Manipulation Language (DML)**
  - ✓ After the database structure is defined with DDL, database administrators and users can use the DML commands.
  - ✓ It is used to insert, retrieve and modify the data contained within it.
- **Data Control Language (DCL)**
  - ✓ It is used to control the access privilege to the database.
  - ✓ DCL provides two commands such as grant and revoke.
- **Transaction Control Language (TCL)**
  - ✓ It is used to control and manage transactions to maintain the integrity of data within SQL statements.
  - ✓ TCL provides command such as commit, rollback, etc.
- **View Definition**: The SQL DDL includes commands for defining views.

DDL Commands

- The Data Definition Language is used to create and destroy databases and database objects.
- These commands are primarily used by database administrators during the setup and removal phases of a database project.
- The four basic DDL commands are,

Create Command (Database)

- It allows you to create and manage many independent databases.

Syntax

Create database <database name>

Example

Create database employee

Create Command (Table)

- It is used to create a table.

Syntax

create table <table name> (columnname1 datatype(size), columnname2 datatype(size)…);

Example

SQL>**create table** employee (ename **varchar**(10), eid **number**(5), address **varchar2**(10), salary **number**(5), designation varchar2(10));

Use Command

- It allows you to specify the database you want to work with within your DBMS.

Syntax

Use <database name>

Example

Use Employee

Alter Command

- It is used to add a new column or modify existing column definitions.

Syntax

alter table <tablename> **add** (new columnname1 datatype(size), new columnname2 datatype(size)…);

alter table <table name> **modify** (column definition);

Example

SQL>**alter table** employee **modify**(eid **number**(7));

SQL>**alter table** employee **add** (age **number**(2));

Drop Command

- It is used to delete a table.

Syntax

drop table <tablename>;

Example

SQL>**drop table** employee;

Notes: This command will delete the contents as well as structure.

Truncate Command

- It is used to delete the records but retain the structure.

Syntax

truncate table <tablename>;

Example

SQL>**truncate table** employee;

To view the table structure

Syntax

desc <tablename>;

Example

SQL>**desc** employee;

DML Commands

- It is used to retrieve, insert and modify database information.
- These commands are used by all database users during the routine operation of the database.

Insert Command

- It is used to insert a new record in the database.

Syntax

insert into <tablename> **values** (a list of data values);

Example

SQL>**create table** employee (ename varchar2(10), eid number (5), salary number(5));

SQL>**insert into** employee **values**(_ABC‘,50,1000); **Select**

Command

- The SELECT command is the most commonly used command in SQL.
- It allows database users to retrieve the specific information they desire from an operational database.

Syntax

Select * from <table name>

Example

Select * from employee

Update Command

- It is used to modify (update) the information contained within a table, either in bulk or individually.

Syntax

update <tablename> **set** field=value,…… **where** <condition>;

Example

SQL>**update** employee **set** eid=100 **where** ename = _ABC‘;

Delete Command

Rows can be deleted using delete command

Syntax

delete from <tablename> **where** <condition>;

Example

SQL>**delete** from employee **where** eid=100;

DCL Commands

- It is used to control privilege in Database.
- To perform any operation in the database, such as for creating tables, sequences or views we need privileges.

Grant Command

- It gives user access privileges to database.

Syntax

Grant Select/insert/delete/update/alter/all privileges on tablename to authenticate;

Example

Grant select, update on student to vikram;

Revoke Command

- It takes back permissions from user.

Syntax

Revoke Select/insert/delete/update/alter/all privileges on tablename from authenticate;

Example

Revoke select, update on student from vikram;

<u>TCL Commands</u>

- Used to manage transactions in database
- To manage the changes made by DML statements.
- Allows statements to be grouped together into logical transactions.

Commit command

- Commit command is used to permanently save any transaction into database.

Syntax

Commit;

Rollback Command

- It restores the database to last commited state.
- It is also use with savepoint command to jump to a savepoint in a transaction.

Syntax

rollback to savepoint_name;

Example

rollback to Temp;

Savepoint Command

- It is used to temporarily save a transaction so that you can rollback to that point whenever necessary.

Syntax

Savepoint savepoint_name;

Example

Savepoint Temp

<u>Some of the Most Important SQL Commands</u>

- **SELECT** - extracts data from a database
- **UPDATE** - updates data in a database
- **DELETE** - deletes data from a database
- **INSERT INTO** - inserts new data into a database
- **CREATE DATABASE** - creates a new database

- **ALTER DATABASE** - modifies a database
- **CREATE TABLE** - creates a new table
- **ALTER TABLE** - modifies a table
- **DROP TABLE** - deletes a table
- **CREATE INDEX** - creates an index (search key)
- **DROP INDEX** - deletes an index

Advanced SQL Features

Write short notes on advanced SQL features.

- The structure of an SQL expression consists of three clauses: select, from and where
    - ✓ The **select clause** corresponds to the projection operation of the relational algebra. It is used to list the attributes desired in the result of a query.
    - ✓ The **from clause** corresponds to the Cartesian product operation of the relational algebra. It lists the relations to be scanned in the evaluation of the expression.
    - ✓ The **where clause** corresponds to the selection predicate of the relational algebra. It consists of a predicate involving attributes of the relations that appear in the from clause.
- A SQL has the form,

  select $A_1, A_2, \ldots. A_n$

  from $r_1, r_2, \ldots.\ r_m$

  where p

  A_i - an attribute r_i -

  relation

  p - predicate
- The query is equivalent to the relational algebra expression. $A_1, A_2, \ldots.\ A_n$

  (σ_p ($r_1 \times r_2 \times \ldots\ldots. \times r_m$))
- If the where clause is omitted, the predicate p is true.

Tuple Variables

- **Tuple variables** are defined in the from clause via the use of the **as clause**.
- For example, find the customer names and their loan numbers for all customers having a loan at some branch in the banking database.

  SQL>**select** B.customer_name,B.loan_no,L.amount **from** borrower as B,loan **as** L **where**

  L. loan_no = B. loan_no;

String Operations

- SQL includes a string matching operator for comparisons on character strings. Patterns are described using 2 special characters.
    - ✓ **Percent (%):** The % character, matches any substring.
    - ✓ **Underscore (-):** The-character matches any character.

Example:

Findthenamesofcustomerswherethe1st2charactersare‗Ba'.

SQL>**select** customer_name **from** customer **where** customer_name **like** ‗Ba%'; Find the names of customer where the 2nd character is ‗*n*' or ‗*a*'.

SQL>**select** customer_name **from** customer **where** customer_name **like** ‗_*n*%' or ‗_*a*%'; Patternsarecase sensitive,i.e.,uppercasecharactersdonotmatchlowercasecharacters,andvice versa.

SQL supports a variety of string operations such as

- ✓ Concatenation using ‗| |' or strcat ( )
- ✓ Converting the string into upper or lower case upper or lower ( )
- ✓ Findstringlength(strlen()),extractingsubstring (substr ( )), etc.

Order by Clause

- Theorderbyclausecausesthetuplesintheresultofaquerytoappearinsortedorder.

SQL>**select** *
from employee
order by salary;

SQL>**select** *
from employee
order by salary **desc**, eid **asc**;

<u>Aggregate Functions</u>

Explain the aggregate functions in SQL with an example. (April/May 2018)

- Aggregate functions are functions that take a collection of values as input and return a single value as output. SQLoffers5builtinaggregatefuntions: avg

	-	average value
min	-	minimum value
max	-	maximum value
sum	-	sum of values
count	-	number ofvalues.

Examples

1. Find the average account balance at the perryridge branch:

   SQL>**select avg** (balance) **from** account **where** branch-name = ―Perryridge‖;

2. Findthenumberoftuplesinthecustomerrelation

   SQL>**select count** (*) **from** customer;

Aggregate functions with group by clause

- Group by clause is used to group the rows based on certain criteria. Group by is used in conjunction with aggregate functions like sum, avg, min, max, count, etc.

Example: Find the average account balance at each branch.

SQL>**select** branch_name, **avg** (balance) **from** account **group by** branch_name;

Aggregate functions with having clause

- Find the name of all branches where the average account balance is more than $2,000. SQL>**select** branch_name, **avg** (balance) **from** account **group by** branch_name**having avg** (balance) > 2000;

Null Values

- SQL allows the use of null values to indicate absence of information about the value of an attribute.
- Null signifies an unknown value or that a value does not exist.
- The predicate is null can be used to check for null values.

Example:

- Find all loan numbers which appear in the loan relation with null values for amount. SQL>**select** loan_no **from** loan **where** amount **is null**;
- The result of any arithmetic expression is null if any of the input values is null.

Example:

5 + null returnsnull.

Consider the unknown (null) value used in boolean expressions as,

- OR - (unknown or true) = true, (unknown or false) = unknown (unknown or unknown) = unknown.
- AND - (true and unknown) = unknown, (false and unknown) = false, (unknown and unknown) = unknown.
- NOT - (not unknown) = unknown. For

example, find total of all loan amounts.

SQL>**select sum** (amount) **from** loan;

Above statement ignores all null amounts. The result is null if there is no non- null amount.

- All aggregate operations except count (*) ignore tuples with null values on the aggregated attributes.

<u>**Nested Subqueries**</u>

- SQL provides a mechanism for the nesting of subqueries.
- A subquery is a select-from-where expression that is nested within another query.
- A common use of subquery is to perform tests for set membership, set comparisons and set cardinality.

<u>**Set Membership**</u>

- SQL uses in and not in constructs to set membership tests.

In

- The ***in*** connective tests for set membership, where the set is a collection of values produced by a select clause.

Example: Find all customers who have both an account and a loan at the bank.

SQL>**select distinct** customer-name **from** depositor
where customer-name **in** (**select** customer-name **from** borrower);

Not In

- The ***not in*** connective tests for the absence of set membership.

Example: Find all customers who have a loan at the bank but do not have an account at the bank.

SQL>**select distinct** customer-name **from** borrower
where customer-name **not in** (**select** customer name **from** depositor);

<u>Set Comparison</u>

- Nested queries are used to compare sets. SQL uses various comparison operators such as <, >, < =, > =, < >, any, all, and, some, etc. to compare sets.

Example:

Find the names of all branches that have assets greater than those of atleast one branch located in —Chennai‖.

SQL>**select distinct** T.branchname **from** branch **as** T, branch
as S **where** T. assets > S. assets
and S. branch-city = —Chennai‖; Same
query using > **some** clause.

SQL>**select** branch-name from branch **where** assets> **some** (**select** assets **from** branch
where branch-city = —Chennai‖);

- SQL also allows < **some**, < = **some**, > = **some**, = **some**, and < > **some** comparisons, = **some** is identical to in, and < > **some** is identical to not in. The keyword **any** is synonym to **some** in SQL.
- SQL also allows < **all**, > **all**, <= **all**, >= **all**, = **all**, and <> **all** comparisons **< > all** is identical to **not in**.

Example:

Find the names all branches that have an-assets value greater than that of each branch in —Chennai‖.

SQL>**select** branch-name **from** branch **where** assets > **all** (select assets **from** branch **where**
branch-city = —Chennai‖);

<u>Test for Empty Relations</u>

- SQL includes a feature for testing whether a subquery has any tuples in its result.
- The **exists** construct returns the value true if the argument subquery is non-empty.

Example:

Find all customers who have both an account and a loan at the bank.

SQL>**select** customer-name **from** borrower **where** exists (**select** * **from** depositor **where**
depositor.customer-name = borrower.customer-name);

- Similar to **exists** we can use **not exists** also. Find all customers who have an account at all branches located in —Chennai‖.

SQL>**select distinct** S. customer-name **from** depositor **as** S
where not exists ((select branch-name **from** branch

where branch-city = ―Chennai‖)**except**
(**select** R. branch_name **from** depositor **as** T, account **as** R
where T. account_no = R. account_no **and**
S. customer-name = T. customer-name));

<u>Test for absence of duplicate tuples</u>

- The **unique** construct tests whether a subquery has any duplicate tuples in its result.

Example:

Find all customers who have at most one account at the ―Chennai branch‖.

SQL>**select** T. customer_name **from** depositor **as** T
where unique (**select** R. customer-name
from account, depositor as R
where T.customer_name=R.customer-name **and** R.
account-no=account.account_no **and**
account.branch_name=―Chennai‖);

- **not unique** construct is used for test the existence of duplicate tuples in the same manner.

<u>Complex Queries</u>

- Complex queries are often hard or impossible to write as a single SQL block.
- There are two ways for composing multiple SQL blocks to express a complex query.
    - ✓ Derived Relations
    - ✓ With Clause

Derived Relations

- SQL allows a subquery expression to be used in the **from** clause.
- If we use such an expression, then we must give the result relation a name, and we can rename the attributes. For renaming **as** clause is used.
- For example, find the average account balance of those branches where the average account balance is greater than $ 2000.

SQL>**select** branch-name, avg-balance **from** (select branch-name,
avg (balance) from account **group** by branch-name) **as**
branch-avg (branch-name, avg-balance) **where** avg-balance
>2000;

- Here subquery result is named branch-avg with the attributes branch-name and avg- balance.

With Clause

- The with clause provides a way of defining a temporary view, whose definition available only to the query in which the **with** clause occurs.
- Consider the following query, which selects **accounts with the maximum balance**.
- If there are many accounts with the same maximum balance, all of them are selected. with max_balance
(value) as
select max (balance)

from account
select account_no from account, max_balance where
account.balance = max_balance.value;

Views

Write short notes on views.

- A view is an object that gives the user a logical view of data from an underlying table or tables (relation or relations).
- It is not desirable for all users to see the entire logical model.
- Security considerations may require that certain data be hidden from users.
- Any relation that is not part of the logical model, but is made visible to a user as a **virtual relation**, is called as **view**.
- Views may be created for the following reasons:
  - ✓ To provide DataSecurity
  - ✓ Query Simplicity
  - ✓ Structural Simplicity (because view contains only limited number of columns and rows).

(1) Creation of Views

Syntax

create view view_name **as** < query expression >;

Example:

Create a view customer_details from customer relation with customer name and customer-id. create view customer_details as
select customer_name, customer_id from customer;

(2) Assigning Names to Columns

- We can assign names for the various columns in the view.
- This may be entirely different from what has been used in the main relation. For example,
  SQL>**create view** customer_details (cust_name,customer_no)
  as select customer_name, customer_id **from** customer;

(3) Selecting data from a view

SQL>**select** * **from** customer_details;

(4) Updation of a view

- Views can also be used for data manipulation i.e., the user can perform insert, update, and the delete operations on the view.
- The views on which data manipulation can be done are called **Updatable views**, the views that do not allow data manipulation are called **Readonly views**.
- When you give a view name in the update, insert, or delete statement, the modification to the data will be passed to the underlying (main) relation.

For the view to be updatable, it should meet following criteria:

- The view must be created on a single table.
- Primary key column of the table should be included in the view.
- Aggregate functions cannot be used in the select statement.
- The select statement used for creating a view should not include distinct, group by or having clause.
- Select statement used for creating a view should not include subqueries.
- It must not use constant, strings or value expression like total/6.

(5) Destroying a View

A view can be dropped by using the drop view command.

Syntax

drop view view_name;

Example

SQL>**drop view** customer_details;

<u>Joins</u>

Explain in detail about join operation in SQL.

- A join is a query using which we can query data more than one table.
- Joins are the basic of multi-table query processing in SQL.
- A join is a query that extracts corresponding rows from two or more tables, views or snapshots.
- If the two tables used in the join have the same column name, then the column names should be prefixed with table name followed by a period.
- SELECT statement of a multi-table query must contain a filter condition that specify the column match. The **where** clause is used to specify the selection condition and the join condition. In the where clause the logical operators can also be used.

Types of Joins

Joins are classified into four types namely:

- Inner Join
- Outer Join
- Natural Join

Inner Join

- Inner join returns the matching rows from the tables that are being joined.
- Consider followingtwo relations:
  - ✓ Student(sname, place)
  - ✓ Student_marks(sname, dept, mark)

Student	sname	place
	Prajan	Chennai
	Anand	Kolkata
	Kumar	Delhi
	Ravi	Mumbai

Student_marks	sname	dept	mark
	Prajan	CS	700
	Anand	IT	650
	Vasu	CS	680
	Ravi	IT	600

Example 1

SQL>**select** Student.sname, Student_marks, mark **from** Student **inner join** Student_marks **on** Student.sname=Student_marks.sname; The output of the above query is,

sname	mark
Prajan	700
Anand	650
Ravi	600

Example 2

SQL>**select** * **from** Student **inner join** Student_marks **on** Student.sname=Student_marks.sname; The result of the above query is,

sname	place	sname	dept	mark
Prajan	Chennai	Prajan	CSE	700
Anand	Kolkata	Anand	IT	650
Ravi	Mumbai	Ravi	IT	600

- For example 2 the result consists of the attributes of the left-hand-side relation followed by the attributes of the right-hand-side relation.
- Thus, the sname attribute appears twice in result, first is from student table and second is from student_marks table.

Outer Join

- When tables are joined using inner join, rows which contain matching values in the join predicate are returned.
- Sometimes you may want both matching and non-matching rows returned for the tables that are being joined. This kind of operation is known as an outer join.
- An outer join is an extended form of the inner join.

- In this, the rows in one table having no matching rows in the other table will also appear in the result table with nulls.

Types of Outer Join

The Outer Join can be any one of the following:

- ✓ Left Outer
- ✓ Right Outer

1. Left Outer join

- The left outer join returns matching rows from the tables being joined and also non- matching rows from the left table in the result and places null values in the attributes that come from the right table.

Example 3

SQL> **select** Student.sname, Student_marks.mark **from** Student **left outer join** Student_marks **on** Student.sname = Student_marks.sname; The result of above query is

sname	mark
Prajan	700
Anand	650
Ravi	600
Kumar	null

Left outer join operation is computed as follows:

- First compute the result of inner join as before.
- Then, for every tuple _t‘ in the left hand side relation, Student that does not match any tuple in the right-hand –side relation Student_marks in the inner join , add a tuple _r‘ to the result of the join:
- The attributes of tuple _r‘ that are derived from the left-hand-side relation are filled with from tuple _t‘, remaining attributes of _r‘ are filled with null values as shown in example 3.

2. Right outer join

- The right outer join operation returns matching rows from the tables being joined, and also non-matching rows from the right table in the table in the result and places null values in the attributes that comes from the left table.

Example 4

SQL>**select** Student.sname,Student.place,Student_marks.mark **from** Student **right outer join** Student_marks **on** Student.sname = Student_marks.sname;

The result of above query is,

sname	place	mark
Prajan	Chennai	700
Anand	Kolkota	650
Ravi	Mumbai	600
Vasu	null	680

Embedded SQL

Explain in detail about Embedded SQL.

- Embedded SQLis a method of combining thecomputingpower of aprogramming language and the database manipulation capabilities of SQL.
- A language in which SQL queries are embedded is referred to as a *host* language, and the SQL structures permitted in the host language constitute *embedded* SQL.
- Programs written in the host language can use the embedded SQL syntax to access and update data stored in a database.
- An embedded SQL program must be processed by a special preprocessor (SQL Preprocessor) prior tocompilation.
- The preprocessor replaces embedded SQL requests with host-language declarations and procedure calls that allow runtime execution of the database accesses.
- The output from the preprocessor is then compiled by the host language compiler.
- This allows programmers to embed SQL statements in programs written in any number of languages such as C/C++, Java, COBOL and FORTRAN.
- To identify embedded SQL requests to the preprocessor, we use the EXEC SQL statement; it has the form:

  EXEC SQL <embedded SQL statement >;
- The exact syntax for embedded SQL requests depends on the language in which SQL is embedded.

Embedded SQL in C Program Examples

Example 1

/* Variable Declaration in Language C */

- Variables inside **DECLARE** are shared and can appear (while prefixed by a colon) in SQL statements
- **SQLCODE** is used to communicate errors/exceptions between the database and the program

```
int loop;
EXEC SQL BEGIN DECLARE SECTION;
        varchar dname[16], fname[16], …; char
        ssn[10], bdate[11], …;
        int dno, dnumber, SQLCODE, …; EXEC
SQL END DECLARE SECTION;
```

Example 2

/* Conditional and Looping Statements in Language C */

```
loop = 1; while
(loop) {
        prompt(—EnterSSN:—,ssn);
        EXEC SQL
                select FNAME, LNAME, ADDRESS, SALARY into :fname, :lname, :address,
                :salary from EMPLOYEE where SSN==:ssn; if
                (SQLCODE == 0) printf(fname, …);
                else printf(—SSN does not exist: —, ssn);
                prompt(—More SSN? (1=yes, 0=no): —,loop);
        END-EXEC
}
```

Dynamic SQL

- **Dynamic SQL** is a programming methodology for generating and running SQL statements at runtime.
- It is useful when writing general-purpose and flexible programs like dynamic query systems, when writing programs that must run database definition language (DDL) statements, or when you do not know at compile time the full text of a SQL statement or the number or data types of its input and output variables.

Difference between Static SQL and Dynamic SQL

S. No.	Static SQL	Dynamic SQL
1.	In static SQL how database will be accessed is predetermined in the embedded SQL statement.	In dynamic SQL, how database will be accessed is determined at run time.
2.	It is less flexible and more efficient.	It is more flexible and less efficient.
3.	SQL statements are compiled at compile time.	SQL statements are compiled at runtime.
4.	Parsing, validation, optimization, and generation of application plan are done at compile time.	Parsing, validation, optimization, and generation of application plan are done at run time.
5.	It is generally used for situations where data is distributed uniformly.	It is generally used for situations where data is distributed non-uniformly.
6.	EXECUTE IMMEDIATE, EXECUTE and PREPARE statements are not used.	EXECUTE IMMEDIATE, EXECUTE and PREPARE statements are used.