

## Section-A

a). **What is data model? List the types of data model used.**

**Ans).** Database model defines the logical design and structure of a database and defines how data will be stored, accessed and updated in a database management system.

Data models can be categorized into two types:

1. High level data models (e.g.: implementation DM or Relational data model)
2. Low level data models

While the **Relational Model** is the most widely used database model, there are other models too:

- Hierarchical Model
- Network Model
- Entity-relationship Model
- Relational Model

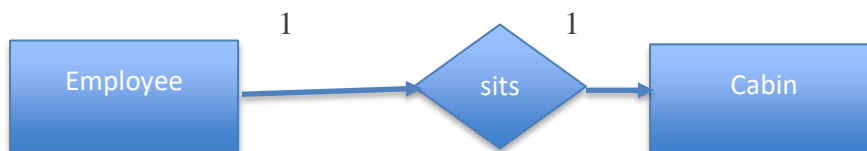
b). **Give the example of one to and one to many relationship.**

**Ans).**

### 1. One-One Relationship (1-1 Relationship)

One-to-One (1-1) relationship is defined as when entity in A is associated with at most one entity in B and entity in B is associated with at most one entity in A.

e.g. A single employee sits in its own single cabin



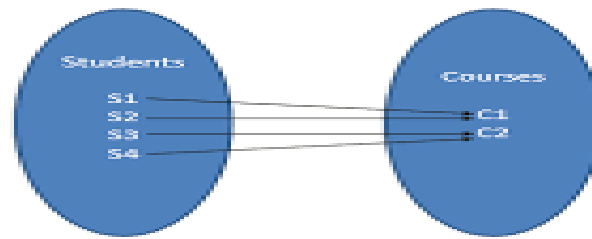
### One-Many Relationship (1-M Relationship)

one-many(1-M) relationship is defined as when An entity in A is associated with any number of entity in B ..

e.g. school has student



## One To Many Relationship



c). with an example show that how referential integrity can be implemented.

Ans) **CREATE TABLE** Persons(PersonID **int PRIMARY KEY**,  
PersonName **int NOT NULL**,Address **varchar(10)**)

**CREATE TABLE** Orders(OrderID **int PRIMARY KEY**,  
OrderNumber **int NOT NULL**,  
PersonID **int, FOREIGN KEY (PersonID) REFERENCES Persons(PersonID)**);

d). write the purpose of trigger.

Ans). In a **DBMS**, a **trigger** is a **SQL** procedure that initiates an action (i.e., fires an action) when an event (INSERT, DELETE or UPDATE) occurs. Since **triggers** are event-driven specialized procedures, they are stored in and managed by the **DBMS**. ... In addition, **triggers** can also execute stored procedures.

### Benefits of Triggers

Triggers can be written for the following purposes –

- Generating some derived column values automatically
- Enforcing referential integrity
- Event logging and storing information on table access
- Auditing
- Synchronous replication of tables
- Imposing security authorizations
- Preventing invalid transactions

e). What is normalization?

**Ans).** Normalization is a database design technique which organizes tables in a manner that reduces redundancy and dependency of data. Redundancy in relation may cause insertion, deletion and updation anomalies. So, it helps to minimize the redundancy in relations. **Normal forms** are used to eliminate or reduce redundancy in database tables.

It divides larger tables to smaller tables and links them using relationships.

**f).** Define the term ACID properties.

**Ans.** DBMS follows the concept of ACID properties: -

1. Atomicity : Atomicity states that either all the operation of the transactions are executed or none of them will execute.
2. Consistency : It should maintain the update value because values changes when various transactions take place.
3. Isolation : It state that only one transaction takes place at a time so that there is no error in values.
4. Durability : It state that there is no impact on our database once the transaction is completes and saved to database.

**g).** State the properties of transaction.

**Ans.** Same as above.

**h).** What is serializability? How it is tested?

**Ans.** Serializability is the classical concurrency scheme. When multiple transactions are running concurrently then there is a possibility that the database may be left in an inconsistent state. Serializability is a concept that helps us to check which **schedules** are serializable. A serializable schedule is the one that always leaves the database in consistent state.

It ensures that a schedule for executing concurrent transactions is equivalent to one that executes the transactions serially in some order. It assumes that all accesses to the database are done using read and write operations.

The Serializability of a schedule is tested using a Serialization graph. Assume a schedule S1. For S1, a graph called Precedence Graph is constructed. This graph consists of a pair  $G = (V, E)$ , where E is a set of the edges, and V is a set of all vertices.

**i).** Why is concurrency control needed?

**Ans.** Need of concurrency control: -

1. To apply Isolation through mutual exclusion between conflicting transactions.
2. To resolve read-write and write-write conflict issues.
3. To preserve database consistency through constantly preserving execution obstructions.

**j).** Define timestamp.

**Ans.** Time stamp is a ordering protocol to decide the order between the transaction before they enter into the system. So that in case of conflict during execution we can resolve the conflict during ordering.

Two ideas of time stamp are :-

1. Time stamp with transaction [TS(Ti)].
2. Time stamp of each data item.(W-TS(item),R-TS(item))

## **Section-B**

### **2a) QUERY**

1) Select e.employee\_name, e.city from employee e cross join works w where e.employee\_name = w.employee\_name and company\_name = "XYZ bank";

2) Select employee.employee\_name ,employee.city, employee.street from employee,works where employee.employee\_name = works.employee\_name and company\_name = "XYZ bank" and salary>10000;

3)      `Select employee.employee_name from employee, works , company where  
employee.employee_name = works.employee_name and works.company_name =  
company.company_name;`

## **2 b) Discuss about the deadlock prevention schemes**

Deadlocks are not healthy for a system. In case a system is stuck in a deadlock, the transactions involved in the deadlock are either rolled back or restarted.

### **Deadlock Prevention**

To prevent any deadlock situation in the system, the DBMS aggressively inspects all the operations, where transactions are about to execute. The DBMS inspects the operations and analyzes if they can create a deadlock situation. If it finds that a deadlock situation might occur, then that transaction is never allowed to be executed.

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.

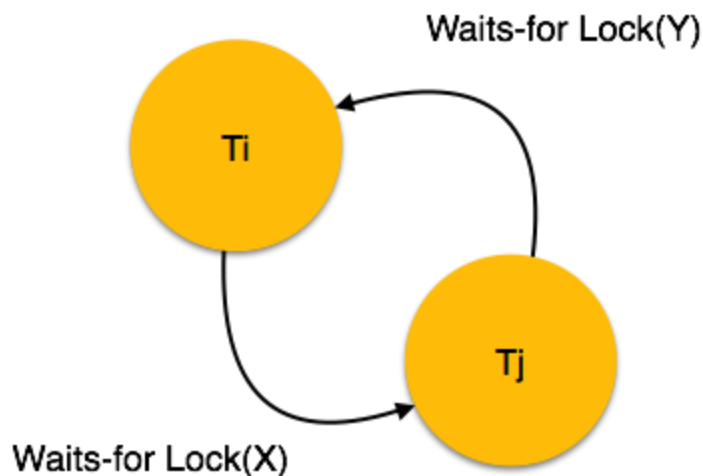
## Deadlock Avoidance

Aborting a transaction is not always a practical approach. Instead, deadlock avoidance mechanisms can be used to detect any deadlock situation in advance. 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

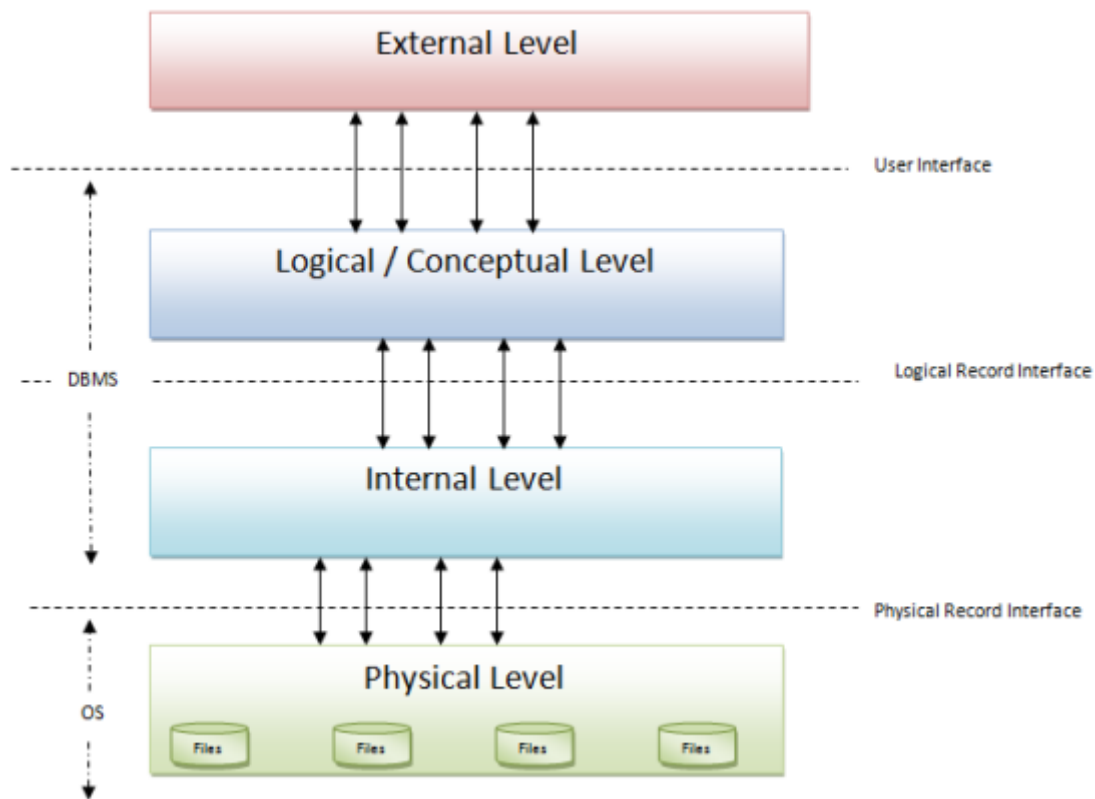
relative algorithm, a transaction is chosen, which is to be aborted. This transaction is known as the **victim** and the process is known as **victim selection**.

## **2 c) Explain the differences between physical level, conceptual level and view level of data abstraction.**

**Logical/ Conceptual level** - This is the next level of abstraction. It describes the actual data stored in the database in the form of tables and relates them by means of mapping. This level will not have any information on what a user views at external level. This level will have all the data in the database.

**Physical level** - This is the lowest level in data abstraction. This level describes how the data is actually stored in the physical memory like magnetic tapes, hard disks etc. In this level the file organization methods like hashing, sequential, B+ tree comes into picture. At this level, developer would know the requirement, size and accessing frequency of the records clearly. So designing this level will not be much complex for him.

**External level** - This is the highest level in data abstraction. At this level users see the data in the form of rows and columns. This level illustrates the users how the data is stored in terms of tables and relations. Users view full or partial data based on the business requirement. The users will have different views here, based on their levels of access rights. For example, student will not have access to see Lecturers salary details, one employee will not have access to see other employees details, unless he is a manager. At this level, one can access the data from database and perform some calculations based on the data. For example calculate the tax from the salary of employee, calculate CGPA of a Student, Calculate age of a person from his Date of Birth etc. These users can be real users or any programs.



## 2 d. Explain embedded SQL and dynamic SQL in detail.

### Embedded SQL

**Embedded SQL** are SQL statements in an application that do not change at run time and, therefore, can be hard-coded into the application. Embedded SQL is also known as Static SQL.

Embedded SQL is the one which combines the high-level Language with the DB language like SQL. It allows the application languages to communicate with DB and get requested result. The high-level languages which supports embedding SQLs within it are also known as host language. There are different host languages which support embedding SQL within it like C, C++, ADA, Pascal, FORTRAN, Java etc. When SQL is embedded within C or C++, then it is known as Pro\*C/C++ .

### Embedded SQL provides several advantages:

- Embedded SQL is easy to use because it is simply Transact-SQL with some added features that facilitate using it in an application.
- It is an ANSI/ISO-standard programming language.



- It requires less coding to achieve the same results as a call-level approach.
- Embedded SQL is essentially identical across different host languages. Programming conventions and syntax change very little. Therefore, to write applications in different languages, you need not learn new syntax.
- The precompiler can optimize execution time by generating stored procedures for the Embedded SQL statements.

#### **LIMITATION:**

They do not change at runtime thus are hard-coded into applications.

#### **Dynamic SQL**

**Dynamic SQL** is SQL statements that are constructed at runtime; for example, the application may allow users to enter their own query. **Dynamic SQL** is a programming technique that enables you to build SQL statements dynamically at runtime. You can create more general purpose, flexible applications by using dynamic SQL because the full text of a SQL statement may be unknown at compilation.

#### **ADVANTAGES:**

Dynamic SQL permits a client application to act interactively, passing different information at different times to the Open Server application, from the user. The Open Server application can then fill in the missing pieces in the SQL query with the data the user provides.

#### **LIMITATION:**

We cannot use some of the SQL statements Dynamically. Performance of these statements is poor as compared to Static SQL.

#### **2e. Describe Shadow Paging Recovery Technique.**

- Shadow paging is a technique for providing atomicity and durability in database systems.
- Shadow paging is a copy-on-write technique for avoiding in-place updates of pages. Instead, when a page is to be modified, a shadow page is allocated.
- Since the shadow page has no references (from other pages on disk), it can be modified liberally, without concern for consistency constraints, etc. When the page is ready to become durable, all pages that referred to the original are updated to refer to the new replacement page instead. Because the page is "activated" only when it is ready, it is atomic.

- This increases performance significantly by avoiding many writes on hotspots high up in the referential hierarchy (e.g.: a file system superblock) at the cost of high commit latency.

Shadow paging considers:

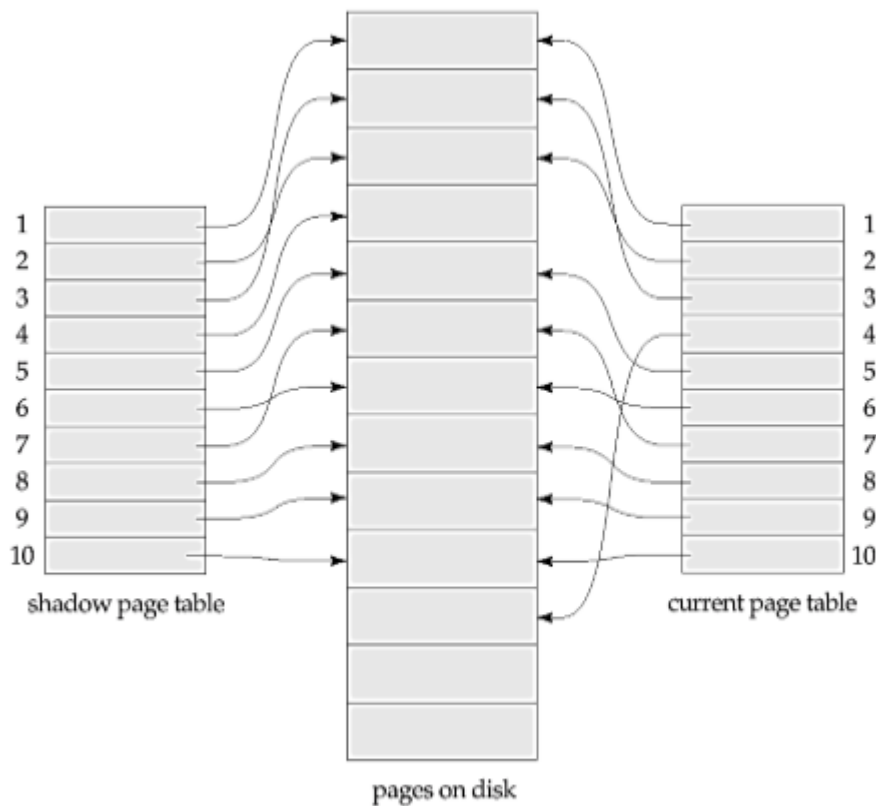
1. The database is partitioned into fixed-length blocks referred to as PAGES.
2. Page table has n entries – one for each database page.
3. Each contain pointer to a page on disk (1 to 1st page on database and so on...).

The idea is to maintain 2 pages tables during the life of transaction.

1. The current page tables
2. The shadow page table

When transaction starts, both page tables are identical

1. The shadow page table is never changed over the duration of the transaction.
2. The current page table may be changed when a transaction performs a write operation.
3. All input and output operations use the current page table to locate database pages on disk.



**Advantages:**

- No Overhead for writing log records.
- No Undo / No Redo algorithm.
- Recovery is faster.

**Disadvantages:**

- Data gets fragmented or scattered.
- After every transaction completion database page containing old version of modified data need to be garbage collected.
- Hard to extend algorithm to allow transaction to run concurrently.

**2f. Write Down in detail about Deadlock and Serializability.****DEADLOCK**

In a **database**, a **deadlock** is a situation in which two or more transactions are waiting for one another to give up locks. For example, Transaction A might hold a lock on some rows in the Accounts table and needs to update some rows in the Orders table to finish.

**Cause of Deadlock:**

The **Cause** of Every **Deadlock** in SQL Server

A **deadlock** happens when two (or more) transactions block each other by holding locks on resources that each of the transactions also need. For example: **Transaction** 1 holds a lock on Table A. **Transaction** 2 holds a lock on Table B.

**Deadlock Detection**

In a database, when a transaction waits indefinitely to obtain a lock, then the DBMS should detect whether the transaction is involved in a deadlock or not. The lock manager maintains a Wait for the graph to detect the deadlock cycle in the database.

## Example of Deadlock

A set of processes or threads is **deadlocked** when each process or thread is waiting for a resource to be freed which is controlled by another process. Here is an **example** of a situation where **deadlock** can occur. ... Traffic gridlock is an everyday **example** of a **deadlock** situation.

## Deadlock Prevention

Deadlock prevention mechanism proposes two schemes:

### Wait-Die Scheme –

In this scheme, If a transaction request for a resource that is locked by other transaction, then the DBMS simply checks the timestamp of both transactions and allows the older transaction to wait until the resource is available for execution. Suppose, there are two transactions T1 and T2 and Let timestamp of any transaction T be TS (T). Now, If there is a lock on T2 by some other transaction and T1 is requesting for resources held by T2, then DBMS performs following actions:

- Checks if  $TS(T1) < TS(T2)$  – if T1 is the older transaction and T2 has held some resource, then it allows T1 to wait until resource is available for execution. That means if a younger transaction has locked some resource and older transaction is waiting for it, then older transaction is allowed wait for it till it is available. If T1 is older transaction and has held some resource with it and if T2 is waiting for it, then T2 is killed and restarted latter with random delay but with the same timestamp. i.e. if the older transaction has held some resource and younger transaction waits for the resource, then younger transaction is killed and restarted with very minute delay with same timestamp.

This scheme allows the older transaction to wait but kills the younger one.

### Wound Wait Scheme –

In this scheme, if an older transaction requests for a resource held by younger transaction, then older transaction forces younger transaction to kill the transaction and release the resource. The younger transaction is restarted with minute delay but with same timestamp. If the younger transaction is requesting a resource which is held by older one, then younger transaction is asked to wait till older releases it.

## SERIALIZABILITY

. When multiple transactions are running concurrently then there is a possibility that the database may be left in an inconsistent state. Serializability is a concept that helps us to check which schedules are serializable. A serializable schedule is the one that always leaves the database in consistent state.

## What is a serializable schedule?

A serializable schedule always leaves the database in consistent state. A serial schedule is always a serializable schedule because in serial schedule, a transaction only starts when the other transaction finished execution. However, a non-serial schedule needs to be checked for Serializability.

A non-serial schedule of n number of transactions is said to be serializable schedule, if it is equivalent to the serial schedule of those n transactions. A serial schedule doesn't allow concurrency, only one transaction executes at a time and the other starts when the already running transaction finished.

2 aspects of serializability are:

### Conflict serializability:

**Conflict Serializable:** A schedule is called **conflict serializable** if it can be transformed into a serial schedule by swapping non-**conflicting** operations. **Conflicting** operations: Two operations are said to be **conflicting** if all conditions satisfy: They belong to different transactions. They operate on the same data item.

A serial schedule T2 follows T1, by a series of swaps of non-conflicting instructions making the below Schedule conflict serializable.

T1	T2
Read(X)	
Write(X)	
	Read(X)
	Write(X)
Read(Y)	
Write(Y)	
	Read(Y)
	Write(Y)

### View Serializability:

- View equivalence is also based purely on reads and writes alone.
- A schedule S is view serializable if it is i.e. w equivalent to a serial schedule.
- Every conflict serializable schedule is also view serializable.
- Every view serializable schedule which is not conflict serializable has blind writes.

T3	T4	T6
Read(P)	Write(P)	
Write(P)		Write(P)

## Section-C

**3a. What are Relational Algebra operations supported in SQL? Write the SQL Statement for Each operation?**

Sol. Relational Algebra

Relational algebra is a procedural query language, which takes instances of relations as input and yields instances of relations as output. It uses operators to perform queries. An operator can be either unary or binary. They accept relations as their input and yield relations as their output. Relational algebra is performed recursively on a relation and intermediate results are also considered relations.

The fundamental operations of relational algebra are as follows –

- 1.Select
- 2.Project
- 3.Union
- 4.Set different

5.Cartesian product

6.Rename

### 1.Select Operation ( $\sigma$ )

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

Notation –  $\sigma p(r)$

Where  $\sigma$  stands for selection predicate and  $r$  stands for relation.  $p$  is propositional logic formula which may use connectors like and, or, and not. These terms may use relational operators like  $=$ ,  $\neq$ ,  $\geq$ ,  $<$ ,  $>$ ,  $\leq$ .

For example –

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

Output – Selects tuples from books where subject is 'database'.

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

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

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

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

## 2.Project Operation ( $\Pi$ )

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

Notation –  $\Pi[A_1, A_2, \dots, A_n](r)$

Where  $A_1, A_2, \dots, A_n$  are attribute names of relation  $r$ .

Duplicate rows are automatically eliminated, as relation is a set.

For example –

$\Pi[\text{subject, author}](\text{Books})$

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

## 3.Union Operation ( $\cup$ )

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 –  $r \cup s$

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.

$\Pi \text{ author (Books)} \cup \Pi \text{ author (Articles)}$

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

#### 4.Set Difference ( $-$ )

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

Notation –  $r - s$

Finds all the tuples that are present in  $r$  but not in  $s$ .

$\Pi \text{ author (Books)} - \Pi \text{ author (Articles)}$

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

### 5.Cartesian Product (X)

Combines information of two different relations into one.

Notation –  $r \times s$

Where r and s are relations and their output will be defined as –

$$r \times s = \{ \langle q, t \rangle \mid q \in r \text{ and } t \in s \}$$

$\sigma_{\text{author} = \text{'tutorialspoint'}}(\text{Books} \times \text{Articles})$

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

### 6.Rename Operation ( $\rho$ )

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  $\rho$ .

Notation –  $\rho_x(E)$

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

Additional operations are –

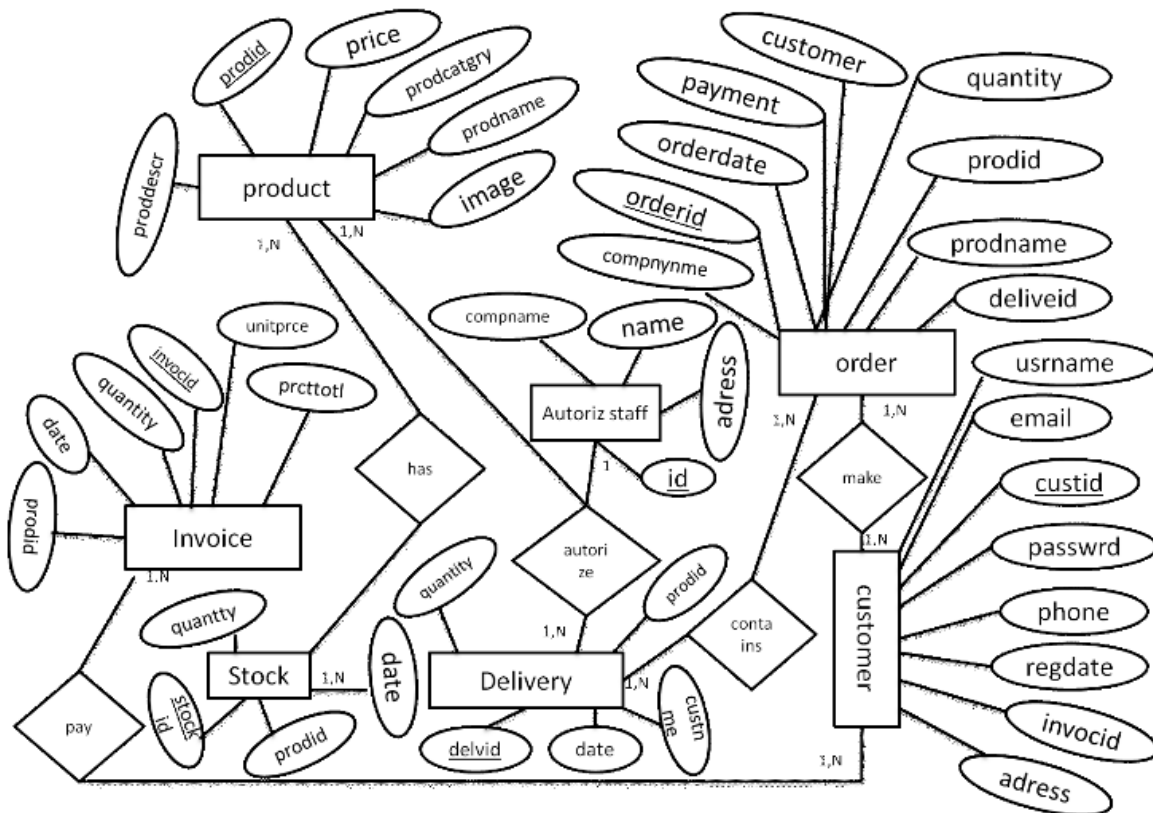
Set intersection

Assignment

Natural join

**3b. Draw E R diagram of small marketing company database, assuming your own data requirements?**

Ans:



#### **4a. Explain 1NF,2NF, 3NF, BCNF with examples?**

Sol

##### 1.1NF (First Normal Form) Rules

Each table cell should contain a single value.

Each record needs to be unique.

The below table in 1NF-

1NF Example

FULL NAMES	PHYSICAL ADDRESS	MOVIES RENTED	SALUTATION
Janet Jones	First Street Plot No 4	Pirates of the Caribbean	Ms.
Janet Jones	First Street Plot No 4	Clash of the Titans	Ms.
Robert Phil	3 <sup>rd</sup> Street 34	Forgetting Sarah Marshal	Mr.
Robert Phil	3 <sup>rd</sup> Street 34	Daddy's Little Girls	Mr.
Robert Phil	5 <sup>th</sup> Avenue	Clash of the Titans	Mr.

##### 2.2NF (Second Normal Form) Rules

Rule 1- Be in 1NF

Rule 2- Single Column Primary Key

It is clear that we can't move forward to make our simple database in 2nd Normalization form unless we partition the table above.

MEMBERSHIP ID	FULL NAMES	PHYSICAL ADDRESS	SALUTATION
1	Janet Jones	First Street Plot No 4	Ms.
2	Robert Phil	3 <sup>rd</sup> Street 34	Mr.
3	Robert Phil	5 <sup>th</sup> Avenue	Mr.

We have divided our 1NF table into two tables viz. Table 1 and Table2. Table 1 contains member information. Table 2 contains information on movies rented.

We have introduced a new column called Membership\_id which is the primary key for table 1. Records can be uniquely identified in Table 1 using membership id

MEMBERSHIP ID	MOVIES RENTED
1	Pirates of the Caribbean
1	Clash of the Titans
2	Forgetting Sarah Marshal
2	Daddy's Little Girls
3	Clash of the Titans

In Table 2, Membership\_ID is the Foreign Key

MEMBERSHIP ID	MOVIES RENTED
1	Pirates of the Caribbean
1	Clash of the Titans
2	Forgetting Sarah Marshal
2	Daddy's Little Girls
3	Clash of the Titans

### 3.3NF (Third Normal Form) Rules

Rule 1- Be in 2NF

Rule 2- Has no transitive functional dependencies

What are transitive functional dependencies?

A transitive functional dependency is when changing a non-key column, might cause any of the other non-key columns to change

MEMBERSHIP ID	FULL NAMES	PHYSICAL ADDRESS	SALUTATION
1	Janet Jones	First Street Plot No 4	Ms.
2	Robert Phil	3 <sup>rd</sup> Street 34	Mr.
3	Robert Phil	5 <sup>th</sup> Avenue	Mr.

*Change in Name* (circled around Robert Phil in row 3) → *May Change Salutation* (arrow pointing to Mr. in row 3)

Consider the table 1. Changing the non-key column Full Name may change Salutation.

To move our 2NF table into 3NF, we again need to again divide our table.

### 3NF Example-

MEMBERSHIP ID	FULL NAMES	PHYSICAL ADDRESS	SALUTATION ID
1	Janet Jones	First Street Plot No 4	2
2	Robert Phil	3 <sup>rd</sup> Street 34	1
3	Robert Phil	5 <sup>th</sup> Avenue	1

MEMBERSHIP ID	MOVIES RENTED
1	Pirates of the Caribbean
1	Clash of the Titans
2	Forgetting Sarah Marshal
2	Daddy's Little Girls
3	Clash of the Titans

SALUTATION ID	SALUTATION
1	Mr.
2	Ms.
3	Mrs.
4	Dr.

### 4.Boyce-Codd Normal Form (BCNF)

Even when a database is in 3rd Normal Form, still there would be anomalies resulted if it has more than one Candidate Key.

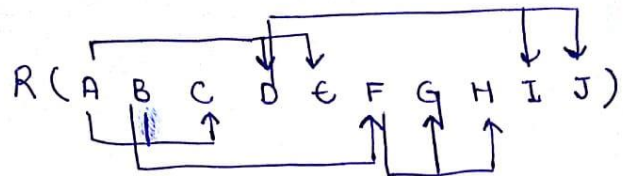
Sometimes is BCNF is also referred as 3.5 Normal Form.

**4b- Consiser Relational Schema R(A,B,C,D,E,F,G,H,I,J) and set of Functional Dependencies:**

**F={AB->C,A->DE,B->F,F->GH,D->IJ}**

**Determine keys and decompose R into 2NF.**

Sol<sup>n</sup>:



$\therefore$   $A, B$  are essential attributes.  
and  $(AB)^+ = R \therefore \underline{AB}$  is candidate Key  
Rel<sup>n</sup>  $R$  has Partial dependencies

i.e.  $A \rightarrow DE$   
and  $B \rightarrow F$

$\therefore$  Decomposing  $R$  into Relations —

$R_1(A, B, C)$

$R_2(A, D, E, I, J)$

$R_3(B, F, G, H)$

Now  $R_1, R_2$  and  $R_3$  does not have  
Partial dependencies  $\therefore$  They are in  
2NF.


### Ans5(i)-Lock-based Protocols

Database systems equipped with lock-based protocols use a mechanism by which any transaction cannot read or write data until it acquires an appropriate lock on it. Locks are of two kinds —

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

2-Shared/exclusive – This type of locking mechanism differentiates the locks based on their uses. If a lock is acquired on a data item to perform a write operation, it is an exclusive lock. Allowing more than one transaction to write on the same data item would lead the database into an inconsistent state. Read locks are shared because no data value is being changed.

Compatibility matrix of lock:

<i>State of the lock</i>	<i>Lock request type</i> 	
	<i>Shared</i>	<i>Exclusive</i>
<i>Shared</i>	<b>Yes</b>	<b>No</b>
<i>Exclusive</i>	<b>No</b>	<b>No</b>

Problems because of Locking are:

- i. Serializability Problem
- ii. Deadlock Problem

To guarantee serializability, we must follow some additional protocol *concerning the positioning of locking and unlocking operations* in every transaction. This is where the concept of Two-Phase Locking(2-PL) comes in the picture, 2-PL ensures serializability.

### Two Phase Locking –

A transaction is said to follow Two Phase Locking protocol if Locking and Unlocking can be done in two phases.

1. **Growing Phase:** New locks on data items may be acquired but none can be released.
2. **Shrinking Phase:** Existing locks may be released but no new locks can be acquired.

**Note** – If lock conversion is allowed, then upgrading of lock (from S(a) to X(a) ) is allowed in Growing Phase and downgrading of lock (from X(a) to S(a)) must be done in shrinking phase.

Let's see a transaction implementing 2-PL.





1	LOCK-S(A)	
2		LOCK-S(A)
3	LOCK-X(B)	
4	.....	.....
5	UNLOCK(A)	
6		LOCK-X(C)
7	UNLOCK(B)	
8		UNLOCK(A)
9		UNLOCK(C)
10	.....	.....

2-PL ensures serializability, but there are still some drawbacks of 2-PL.

- Cascading Rollback is possible under 2-PL.
- Deadlocks and Starvation is possible.

Categories of Two-Phase Locking (Strict, Rigorous & Conservative)

Now that we are familiar with what is Two Phase Locking (2-PL) and the basic rules which should be followed which ensures serializability. Moreover, we came across the problems with 2-PL, Cascading Aborts and Deadlocks. Now, we turn towards the enhancements made on 2-PL which tries to make the protocol nearly error free. Briefly we allow some modifications to 2-PL to improve it. There are three categories:

- Strict 2-PL
- Rigorous 2-PL

- Conservative 2-PL

Now recall the rules followed in Basic 2-PL, over that we make some extra modifications. Let's now see what are the modifications and what drawbacks they solve.

- Strict 2-PL –

This requires that in addition to the lock being 2-Phase all Exclusive(X) Locks held by the transaction be released until after the Transaction Commits.

Following Strict 2-PL ensures that our schedule is:

- Recoverable

- Cascadeless

Hence it gives us freedom from Cascading Abort which was still there in Basic 2-PL and moreover guarantee Strict Schedules but still Deadlocks are possible!

**Example:**

T1

Lock-s(A)

Read(A)

Lock-x(B)

Unlock(A)

Read(B)

Write(B)

Commit

Unlock(B)

Exclusive locks are unlocked after commit. So it is strict 2PL

- Rigorous 2-PL –

This requires that in addition to the lock being 2-Phase all Exclusive(X) and Shared(S) Locks held by the transaction be released until after the Transaction Commits.

Following Rigorous 2-PL ensures that our schedule is:

- Recoverable
- Cascadeless

Hence it gives us freedom from Cascading Abort which was still there in Basic 2-PL and moreover guarantee Strict Schedules but still Deadlocks are possible!

Note the difference between Strict 2-PL and Rigorous 2-PL is that Rigorous is more restrictive, it requires both Exclusive and Shared locks to be held until after the Transaction commits and this is what makes the implementation of Rigorous 2-PL easier.

**Example:**

T1:

Lock-s(A)

Read(A)

Lock-x(B)

Read(B)

Write(B)

Commit

Unlock(B)

Unlock(A)

We have unlocked all the locks after commit. So, it is rigorous 2PL

Conservative 2-PL –

A.K.A Static 2-PL, this protocol requires the transaction to lock all the items it access before the Transaction begins execution by predeclaring its read-set and write-set. If any of the predeclared items needed cannot be locked, the transaction does not lock any of the items, instead it waits until all the items are available for locking.

Conservative 2-PL is Deadlock free and but it does not ensure Strict schedule. However, it is difficult to use in practice because of need to predeclare the read-set and the write-set which is not possible in many situations. In practice, the most popular variation of 2-PL is Strict 2-PL.

**Example:**

T1:

Lock-s(A)

Lock-x(B)

Read(B)

Write(B)

Read(A)

Commit

Unlock(B)

Unlock(A)

We have taken all the locks at first then start the transaction So, It is conservative 2PL

**Ans5(ii)-Timestamp based Concurrency Control**

Timestamp is a unique identifier created by the DBMS to identify a transaction. They are usually assigned in the order in which they are submitted to the system. Refer to the timestamp of a transaction T as TS(T).

- W-<sub>TS(X)</sub> is the largest timestamp of any transaction that executed write(X) successfully.

- R-<sub>TS(X)</sub> is the largest timestamp of any transaction that executed read(X) successfully.

## Basic Timestamp Ordering –

Every transaction is issued a timestamp based on when it enters the system. Suppose, if an old transaction  $T_i$  has timestamp  $TS(T_i)$ , a new transaction  $T_j$  is assigned timestamp  $TS(T_j)$  such that  $TS(T_i) < TS(T_j)$ . The protocol manages concurrent execution such that the timestamps determine the serializability order. The timestamp ordering protocol ensures that any conflicting read and write operations are executed in timestamp order. Whenever some Transaction  $T$  tries to issue a  $R\_item(X)$  or a  $W\_item(X)$ , the Basic TO algorithm compares the timestamp of  $T$  with  $R\_TS(X)$  &  $W\_TS(X)$  to ensure that the Timestamp order is not violated. This describes the Basic TO protocol in following two cases.

1. Whenever a Transaction  $T$  issues a  $W\_item(X)$  operation, check the following conditions:

If  $R\_TS(X) > TS(T)$  or if  $W\_TS(X) > TS(T)$ , then abort and rollback  $T$  and reject the operation. else,

Execute  $W\_item(X)$  operation of  $T$  and set  $W\_TS(X)$  to  $TS(T)$ .

2. Whenever a Transaction  $T$  issues a  $R\_item(X)$  operation, check the following conditions:

If  $W\_TS(X) > TS(T)$ , then abort and reject  $T$  and reject the operation, else

If  $W\_TS(X) \leq TS(T)$ , then execute the  $R\_item(X)$  operation of  $T$  and set  $R\_TS(X)$  to the larger of  $TS(T)$  and current  $R\_TS(X)$ .

Whenever the Basic TO algorithm detects two conflicting operations that occur in incorrect order, it rejects the later of the two operations by aborting the Transaction that issued it. Schedules produced by Basic TO are guaranteed to be conflict serializable. Already discussed that using Timestamp, can ensure that our schedule will be deadlock free.

One drawback of Basic TO protocol is that it Cascading Rollback is still possible. Suppose we have a Transaction  $T_1$  and  $T_2$  has used a value written by  $T_1$ . If  $T_1$  is aborted and resubmitted to the system then,  $T_2$  must also be aborted and rolled back. So the problem of Cascading aborts still prevails.

Let's list the Advantages and Disadvantages of Basic TO protocol:

1-Timestamp Ordering protocol ensures serializability.

2-Timestamp protocol ensures freedom from deadlock as no transaction ever waits.

3-But the schedule may not be cascade free, and may not even be recoverable.

