Hrushikesh

Q. DBA and explain the role of a DBA.

Ans:

Database Administrator:

A Database Administrator, Database Analyst or Database Developer is the person responsible for managing all the activities related to the database system.. As most companies continue to experience inevitable growth of their databases, these positions are probably the most solid within the IT industry.
The DBA has many different responsibilities, but the overall goal of the DBA is to keep the server up at all times and to provide users with access to the required information when they need it. The DBA makes sure that the database is protected and that any chance of data loss is minimized.

DBA Responsibilities:

1. Installing and Upgrading an SQL Server

The DBA is responsible for installing SQL Server or upgrading an existing SQL Server. In the case of upgrading SQL Server, the DBA is responsible for ensuring that if the upgrade is not successful, the SQL Server can be rolled back to an earlier release until the upgrade issues can be resolved.

2. Monitoring the Database Server's Health and Tuning Accordingly

Monitoring the health of the database server means making sure that the following is done:

- The server is running with optimal performance.
- The error log or event log is monitored for database errors.
- Databases have routine maintenance performed on them, and the overall system has periodic maintenance performed by the system administrator.

3. Using Storage Properly:

SQL Server 2000 enables you to automatically grow the size of your databases and transaction logs, or you can choose to select a fixed size for the database and transaction log.

4. Performing Backup and Recovery Duties:

Backup and recovery are the DBA's most critical tasks; they include the following aspects:

- Establishing standards and schedules for database backups
- Developing recovery procedures for each database
- Making sure that the backup schedules meet the recovery requirements

5. Managing Database Users and Security:

The DBA is also responsible for assigning users to databases and determining the proper security level for each user.

6. Working with Developers:

It is important for the DBA to work closely with development teams to assist in overall database design, such as creating normalized databases, helping developers tune queries, assigning proper indexes, and aiding developers in the creation of triggers and stored procedures.

7. Establishing and Enforcing Standards :

The DBA should establish naming conventions and standards for the SQL Server and databases and make sure that everyone sticks to them.

8. Transferring Data:

The DBA is responsible for importing and exporting data to and from the SQL Server.

9. Scheduling Events :

The database administrator is responsible for setting up and scheduling various events using Windows NT and SQL Server to aid in performing many tasks such as backups and replication.

Q. Explain Components of an ER Diagram

Ans:

An ER Diagram consists of the following components:

- Entity
- Attributes
- Relationships

Entity

Entities are represented by means of rectangles. Rectangles are named with the entity set they represent.

An entity can be characterized into two types:

Strong entity: This type of entity has a primary key attribute which uniquely identifies each record in a table. In the ER diagram, a strong entity is usually represented by a single rectangle.

Weak entity:An entity does not have a primary key attribute and depends on another strong entity via foreign key attribute. In the ER diagram, a weak entity is usually represented by a double rectangle.

Attributes

Attributes are the properties of entities. Attributes are represented by means of ellipses. Every ellipse represents one attribute and is directly connected to its entity (rectangle).

Key attribute The attribute which **uniquely identifies each entity** in the entity set is called key attribute.For example, Roll_No will be unique for each student.

Composite An attribute **composed of many other attributes** is called a composite attribute. For example, the Address attribute of the student Entity type consists of Street, City, State, and Country.

Multivalued An attribute consisting **more than one value** for a given entity. For example, Phone_No (can be more than one for a given student).

Derived An attribute which can be **derived from other attributes** of the entity type is known as derived attribute. e.g.; Age (can be derived from DOB).

Relationship

Relationships are represented by diamond-shaped boxes. Name of the relationship is written inside the diamond-box. All the entities (rectangles) participating in a relationship, are connected to it by a line.

Binary Relationship and Cardinality:

A relationship where two entities are participating is called a binary relationship. Cardinality is the number of instances of an entity from a relation that can be associated with the relation.

One-to-one − When each entity in each entity set can take part only once in the relationship**,** the cardinality is one to one.

One-to-many − When more than one instance of an entity is associated with a relationship, it is marked as '1:N'.

Many-to-one − When entities in one entity set can take part only once in the relationship set and entities in another entity set can take part more than once in the relationship set**,** cardinality is many to one.

Many-to-many − When entities in all entity sets can take part more than once in the relationship cardinality is many to many.

Q Explain ACID Properties in DBMS

Ans:

A transaction is a single logical unit of work which accesses and possibly modifies the contents of a database. Transactions access data using read and write operations.

In order to maintain consistency in a database, before and after the transaction, certain properties are followed. These are called ACID properties.

Atomicity

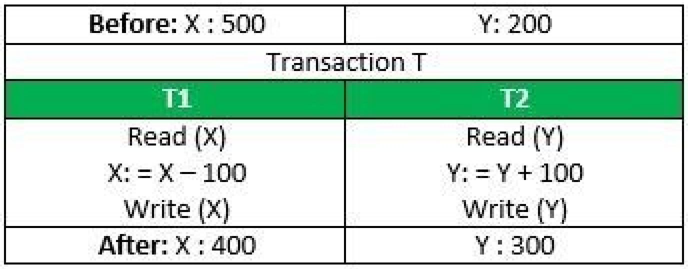
By this, we mean that either the entire transaction takes place at once or doesn't happen at all. There is no midway i.e. transactions do not occur partially. Each transaction is considered as one unit and either runs to completion or is not executed at all. It involves the following two operations.

—**Abort**: If a transaction aborts, changes made to the database are not visible.

—**Commit**: If a transaction commits, changes made are visible.

Atomicity is also known as the 'All or nothing rule'.

Consider the following transaction **T** consisting of **T1** and **T2**: Transfer of 100 from account **X** to account **Y**.



Before: X : 500	Y: 200
Transaction T	
T1	T2
Read (X) X: = X – 100 Write (X)	Read (Y) Y: = Y + 100 Write (Y)
After: X : 400	Y : 300

If the transaction fails after completion of **T1** but before completion of **T2**.(say, after **write(X)** but before **write(Y)**), then the amount has been deducted from **X** but not added to **Y**. This results in an inconsistent database state. Therefore, the transaction must be executed in entirety in order to ensure correctness of database state.

Consistency

This means that integrity constraints must be maintained so that the database is consistent before and after the transaction. It refers to the correctness of a database. Referring to the example above,

The total amount before and after the transaction must be maintained.

Total **before T** occurs = **500 + 200 = 700**.

Total **after T occurs** = **400 + 300 = 700**.

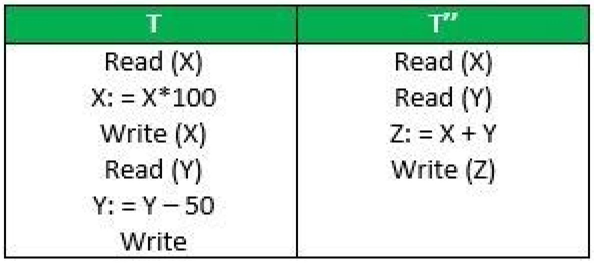
Therefore, the database is **consistent**. Inconsistency occurs in case **T1** completes but **T2** fails. As a result T is incomplete.

Isolation

This property ensures that multiple transactions can occur concurrently without leading to the inconsistency of database state. Transactions occur independently without interference. Changes occurring in a particular transaction will not be visible to any other transaction until that particular change in that transaction is written to memory or has been committed. This property ensures that the execution of transactions concurrently will result in a state that is equivalent to a state achieved serially in some order.

Let **X**= 500, **Y** = 500.

Consider two transactions **T** and **T".**



T	T''
Read (X)	Read (X)
X: = X*100	Read (Y)
Write (X)	Z: = X + Y
Read (Y)	Write (Z)
Y: = Y – 50	
Write	

Suppose **T** has been executed till **Read (Y)** and then **T''** starts. As a result , interleaving of operations takes place due to which **T''** reads correct value of **X** but incorrect value of **Y** and sum computed by

T'': (X+Y = 50, 000+500=50, 500)

is thus not consistent with the sum at end of transaction:

T: (X+Y = 50, 000 + 450 = 50, 450).

This results in database inconsistency, due to a loss of 50 units. Hence, transactions must take place in isolation and changes should be visible only after they have been made to the main memory.

Durability:

This property ensures that once the transaction has completed execution, the updates and modifications to the database are stored in and written to disk and they persist even if a system failure occurs. These updates now become permanent and are stored in non-volatile memory. The effects of the transaction, thus, are never lost.

Q Explain Database Language

Ans:

A DBMS has appropriate languages and interfaces to express database queries and updates.

Database languages can be used to read, store and update the data in the database.

1. Data Definition Language

- DDL stands for Data Definition Language. It is used to define database structure or pattern.
- It is used to create schema, tables, indexes, constraints, etc. in the database.
- Using the DDL statements, you can create the skeleton of the database.

- Data definition language is used to store the information of metadata like the number of tables and schemas, their names, indexes, columns in each table, constraints, etc.

Here are some tasks that come under DDL:

- Create: It is used to create objects in the database.
- Alter: It is used to alter the structure of the database.
- Drop: It is used to delete objects from the database.
- Truncate: It is used to remove all records from a table.
- Rename: It is used to rename an object.
- Comment: It is used to comment on the data dictionary.

These commands are used to update the database schema that's why they come under Data definition language.

2. Data Manipulation Language

DML stands for Data Manipulation Language. It is used for accessing and manipulating data in a database. It handles user requests.

Here are some tasks that come under DML:

- Select: It is used to retrieve data from a database.
- Insert: It is used to insert data into a table.
- Update: It is used to update existing data within a table.
- Delete: It is used to delete all records from a table.
- Merge: It performs UPSERT operation, i.e., insert or update operations.

3. Data Control Language

DCL stands for Data Control Language. It is used to retrieve the stored or saved data.

The DCL execution is transactional. It also has rollback parameters.

(But in Oracle database, the execution of data control language does not have the feature of rolling back.)

Here are some tasks that come under DCL:

- Grant: It is used to give user access privileges to a database.
- Revoke: It is used to take back permissions from the user.

There are the following operations which have the authorization of Revoke:

CONNECT, INSERT, USAGE, EXECUTE, DELETE, UPDATE and SELECT.

4. Transaction Control Language

TCL is used to run the changes made by the DML statement. TCL can be grouped into a logical transaction.

Here are some tasks that come under TCL:

- Commit: It is used to save the transaction on the database.
- Rollback: It is used to restore the database to original since the last Commit.

Q Differences between DBMS and File system:

Ans:

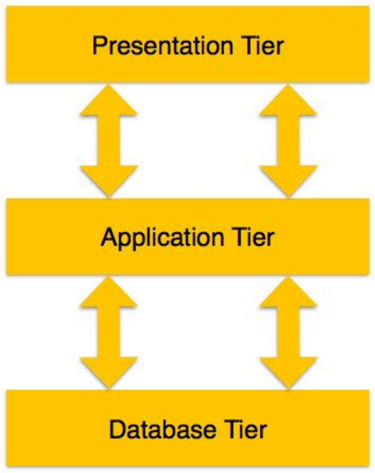
S.NO.	FILE SYSTEM	DBMS
1.	**File system is a software that manages and organizes the files in a storage medium within a computer.**	**DBMS is a software for managing the database.**
2.	**Redundant data can be present in a file system.**	**In DBMS there is no redundant data.**
3.	**It doesn't provide backup and recovery of data if it is lost.**	**It provides backup and recovery of data even if it is lost.**
4.	**There is no efficient query processing in the file system.**	**Efficient query processing is there in DBMS.**
5.	**There is less data consistency in the file system.**	**There is more data consistency because of the process of normalization.**
6.	**It is less complex as compared to DBMS.**	**It has more complexity in handling as compared to file systems.**

7.	**File systems provide less security in comparison to DBMS.**	**DBMS has more security mechanisms as compared to file systems.**
8.	**It is less expensive than DBMS.**	**It has a comparatively higher cost than a file system.**

Q 3-Tier Architecture in DBMS

Ans:

A 3-tier architecture separates its tiers from each other based on the complexity of the users and how they use the data present in the database. It is the most widely used architecture to design a DBMS.



- **Database (Data) Tier** − At this tier, the database resides along with its query processing languages. We also have the relations that define the data and their constraints at this level.
- **Application (Middle) Tier** − At this tier reside the application server and the programs that access the database. For a user, this application tier presents an abstracted view of the database. End-users are unaware of any existence of the database beyond the application. At the other end, the database tier is not aware of any other user beyond the application tier. Hence, the application layer sits in the middle and acts as a mediator between the end-user and the database.
- **User (Presentation) Tier** − End-users operate on this tier and they know nothing about any existence of the database beyond this layer. At this layer, multiple

views of the database can be provided by the application. All views are generated by applications that reside in the application tier.

Multiple-tier database architecture is highly modifiable, as almost all its components are independent and can be changed independently.

Q What is 3 layer database abstraction

Ans:

Database systems comprise complex data structures. Thus, to make the system efficient for retrieval of data and reduce the complexity of the users, developers use the method of Data Abstraction.

There are mainly three levels of data abstraction:

1. Internal Level: Actual PHYSICAL storage structure and access paths.
2. Conceptual or Logical Level: Structure and constraints for the entire database
3. External or View level: Describes various user views

Internal Level/Schema

The internal schema defines the physical storage structure of the database. The internal schema is a very low-level representation of the entire database. It contains multiple occurrences of multiple types of internal record. In the ANSI term, it is also called "stored record'.

Conceptual Schema/Level

The conceptual schema describes the Database structure of the whole database for the community of users. This schema hides information about the physical storage structures and focuses on describing data types, entities, relationships, etc.

This logical level comes between the user level and physical storage view. However, there is only a single conceptual view of a single database.

External Schema/Level

An external schema describes the part of the database which a specific user is interested in. It hides the unrelated details of the database from the user. There may be "n" number of external views for each database.

Each external view is defined using an external schema, which consists of definitions of various types of external record of that specific view.

An external view is just the content of the database as it is seen by some specific particular user. For example, a user from the sales department will see only sales related data.

Q Different types of database users.

Ans:

These are seven types of database users in DBMS.

Database Administrator (DBA) :

Database Administrator (DBA) is a person/team who defines the schema and also controls the 3 levels of the database.

The DBA will then create a new account id and password for the user if he/she needs to access the database.

DBA is also responsible for providing security to the database and he allows only the authorized users to access/modify the database.

- DBA also monitors the recovery and backup and provides technical support.
- The DBA has a DBA account in the DBMS which is called a system or superuser account.
- DBA repairs damage caused due to hardware and/or software failures.

Naive / Parametric End Users :

Parametric End Users are the unsophisticated who don't have any DBMS knowledge but they frequently use the database applications in their daily life to get the desired results.

For examples, Railway's ticket booking users are naive users. Clerks in any bank is a naive users because they don't have any DBMS knowledge but they still use the database and perform their given task.

System Analyst :

System Analyst is a user who analyzes the requirements of parametric end users. They check whether all the requirements of end users are satisfied.

Sophisticated Users :

Sophisticated users can be engineers, scientists, business analysts, who are familiar with the database. They can develop their own database applications according to their requirement. They don't write the program code but they interact with the database by writing SQL queries directly through the query processor.

Database Designers :

Database Designers are the users who design the structure of database which includes tables, indexes, views, constraints, triggers, stored procedures. He/she controls what data must be stored and how the data items are related.

Application Program :

Application Program are the back end programmers who write the code for the application programs.They are the computer professionals. These programs could be written in Programming languages such as Visual Basic, Developer, C, FORTRAN, COBOL etc.

Casual Users / Temporary Users :

Casual Users are the users who occasionally use/access the database but each time when they access the database they require the new information, for example, Middle or higher level manager.

Q Structure of Database Management System

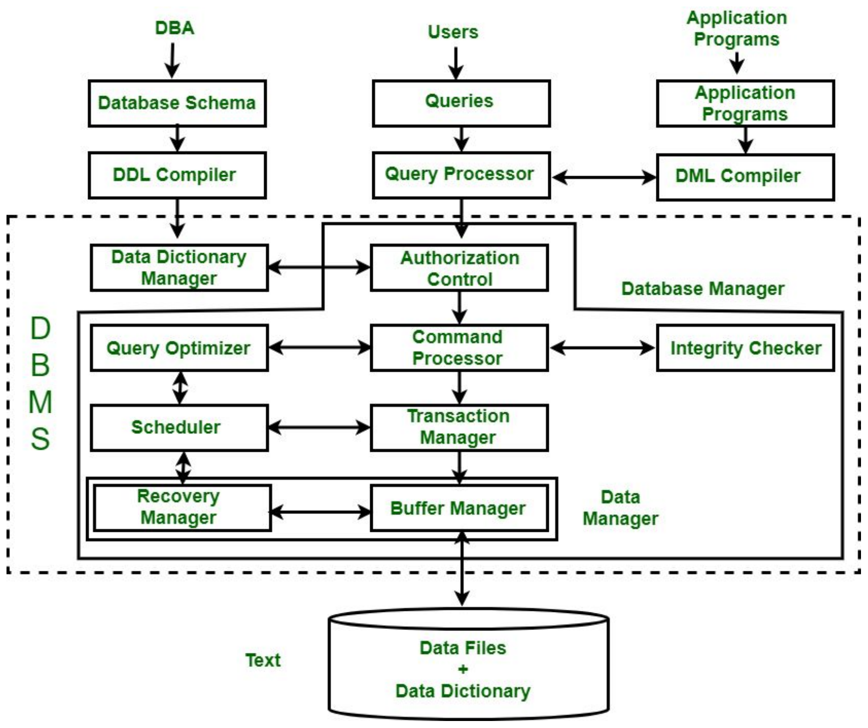
Ans:

Database Management System (DBMS) is a software that allows access to data stored in a database and provides an easy and effective method of –

- Defining the information.
- Storing the information.
- Manipulating the information.
- Protecting the information from system crashes or data theft.
- Differentiating access permissions for different users.

The database system is divided into three components: Query Processor, Storage Manager, and Disk Storage. These are explained as follows below.

Hrushikesh



1. Query Processor :

It interprets the requests (queries) received from the end user via an application program into instructions. It also executes the user request which is received from the DML compiler.

Query Processor contains the following components –

- DML Compiler – It processes the DML statements into low level instruction (machine language), so that they can be executed.
- DDL Interpreter – It processes the DDL statements into a set of tables containing meta data (data about data).
- Embedded DML Pre-compiler – It processes DML statements embedded in an application program into procedural calls.
- Query Optimizer – It executes the instruction generated by DML Compiler.

2. Storage Manager :

Storage Manager is a program that provides an interface between the data stored in the database and the queries received. It is also known as Database Control System. It maintains the consistency and integrity of the database by applying the constraints and executes the DCL statements. It is responsible for updating, storing, deleting, and retrieving data in the database.

It contains the following components –

- Authorization Manager – It ensures role-based access control, i.e,. checks whether the particular person is privileged to perform the requested operation or not.

- Integrity Manager – It checks the integrity constraints when the database is modified.
- Transaction Manager – It controls concurrent access by performing the operations in a scheduled way that it receives the transaction. Thus, it ensures that the database remains in the consistent state before and after the execution of a transaction.
- File Manager – It manages the file space and the data structure used to represent information in the database.
- Buffer Manager – It is responsible for cache memory and the transfer of data between the secondary storage and main memory.

3. Disk Storage :

It contains the following components –

- Data Files – It stores the data.
- Data Dictionary – It contains the information about the structure of any database object. It is the repository of information that governs the metadata.
- Indices – It provides faster retrieval of data items.

Q Difference between Schema and Instance in DBMS

Ans:

Instances are the collection of information stored at a particular moment. The instances can be changed by certain CRUD operations like addition, deletion of data. It may be noted that any search query will not make any kind of changes in the instances.

Example **–** Let’s say a table teacher in our database whose name is School, suppose the table has 50 records so the instance of the database has 50 records for now and tomorrow we are going to add another fifty records so tomorrow the instance has a total of 100 records. This is called an instance.

Schema is the overall description of the database. The basic structure of how the data will be stored in the database is called schema.

Example – Let’s say a table teacher in our database name school, the teacher table require the name, dob, doj in their table so we design a structure as :

Teacher table

name: String

doj: date

dob: date

Above given is the schema of the table teacher.

Schema is of two types: Logical Schema, and Physical Schema.

Hrushikesh

Difference between Schema and Instance :

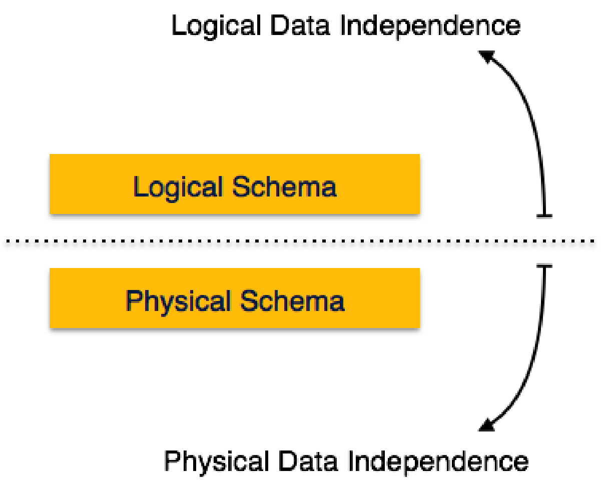
SCHEMA	INSTANCE
It is the overall description of the database.	**It is the collection of information stored in a database at a particular moment.**
Schema is the same for the whole database.	**Data in instances can be changed using addition, deletion, and update.**
Does not change Frequently.	**Changes Frequently.**
Defines the basic structure of the database i.e how the data will be stored in the database.	**It is the set of Information stored at a particular time.**

Q What is Data independence:

Ans:

Data Independence

A database system normally contains a lot of data in addition to users' data. For example, it stores data about data, known as metadata, to locate and retrieve data easily. It is rather difficult to modify or update a set of metadata once it is stored in the database. But as a DBMS expands, it needs to change over time to satisfy the requirements of the users. If the entire data is dependent, it would become a tedious and highly complex job.



Metadata itself follows a layered architecture, so that when we change data at one layer, it does not affect the data at another level. This data is independent but mapped to each other.

Logical Data Independence

Logical data is data about databases, that is, it stores information about how data is managed inside. For example, a table (relation) stored in the database and all its constraints, applied on that relation.

Logical data independence is a kind of mechanism, which liberalizes itself from actual data stored on the disk. If we do some changes on table format, it should not change the data residing on the disk.

Physical Data Independence

All the schemas are logical, and the actual data is stored in bit format on the disk. Physical data independence is the power to change the physical data without impacting the schema or logical data.

For example, in case we want to change or upgrade the storage system itself − suppose we want to replace hard-disks with SSD − it should not have any impact on the logical data or schemas.

Q What is Data Dictionary

Ans:

Data Dictionary consists of database metadata. It has records about objects in the database.

Data Dictionary consists of the following information −

- Name of the tables in the database
- Constraints of a table i.e. keys, relationships, etc.
- Columns of the tables that related to each other
- Owner of the table
- Last accessed information of the object
- Last updated information of the object

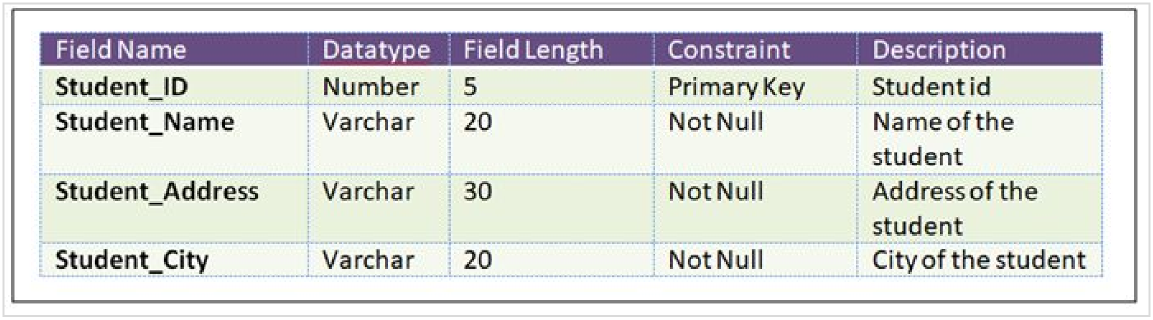
An example of Data Dictionary can be personal details of a student −

Example

<StudentPersonalDetails>

Student_ID	**Student_Name**	**Student_Address**	**Student_City**

The following is the data dictionary for the above fields −



Field Name	Datatype	Field Length	Constraint	Description
Student_ID	Number	5	Primary Key	Student id
Student_Name	Varchar	20	Not Null	Name of the student
Student_Address	Varchar	30	Not Null	Address of the student
Student_City	Varchar	20	Not Null	City of the student

Types of Data Dictionary

Here are the two types of data dictionary −

Active Data Dictionary

The DBMS software manages the active data dictionary automatically. The modification is an automatic task and most RDBMS have an active data dictionary. It is also known as an integrated data dictionary.

Passive Data Dictionary

Managed by the users and is modified manually when the database structure changes. Also known as a non-integrated data dictionary.

Q Difference between Procedural and Non-Procedural language:

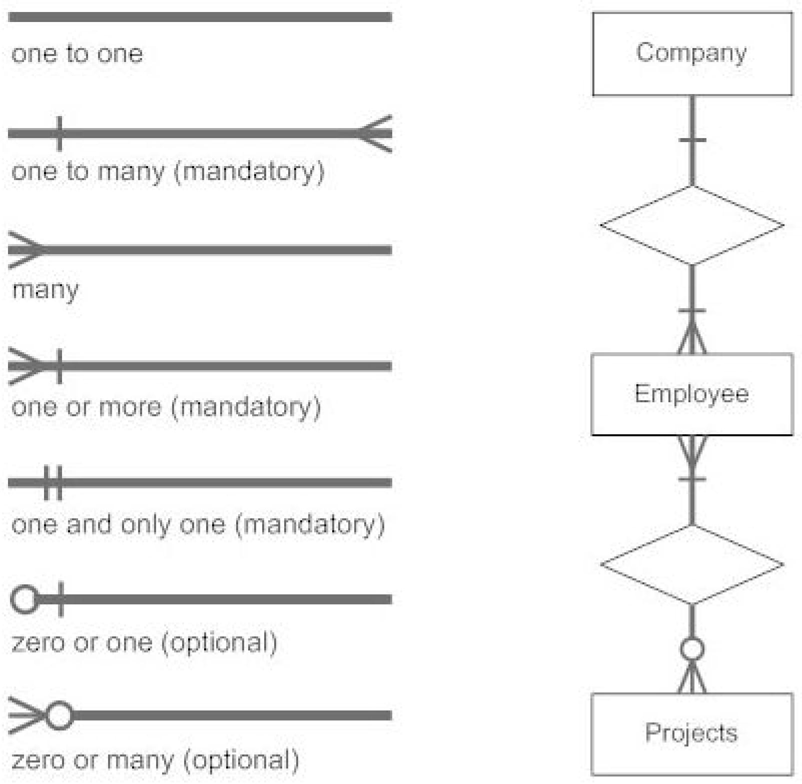
Ans

PROCEDURAL LANGUAGE	NON-PROCEDURAL LANGUAGE
It is a command-driven language.	It is a function-driven language
It works through the state of the machine.	It works through the mathematical functions.
Its semantics are quite tough.	Its semantics are very simple.
It returns only restricted data types and allowed values.	It can return any data type or value
Overall efficiency is very high.	Overall efficiency is low as compared to Procedural Language.
Size of the program written in Procedural language is large.	Size of the Non-Procedural language programs are small.
It is not suitable for time critical applications.	It is suitable for time critical applications.
Iterative loops and Recursive calls both are used in the Procedural languages.	Recursive calls are used in Non-Procedural languages.

Q Notation of ER diagram

Ans:

Databases can be represented using the notations. In ER diagrams, many notations are used to express the cardinality. These notations are as follows:



Q Advantages of ER Model

- **Conceptually it is very simple:** ER model is very simple because if we know the relationship between entities and attributes, then we can easily draw an ER diagram.
- **Better visual representation:** ER model is a diagrammatic representation of any logical structure of a database. By seeing ER diagrams, we can easily understand relationships among entities and relationships.
- **Effective communication tool:** It is an effective communication tool for database designers.
- **Highly integrated with relational model:** ER model can be easily converted into relational model by simply converting ER model into tables.
- **Easy conversion to any data model:** ER model can be easily converted into another data model like hierarchical data model, network data model and so on.

Q Disadvantages of ER Model

- **Limited constraints and specification**
- **Loss of information content:** Some information be lost or hidden in ER model
- **Limited relationship representation:** ER model represents limited relationship as compared to other data models like relational model etc.
- **No representation of data manipulation:** It is difficult to show data manipulation in ER models.

- **Popular for high level design:** ER model is very popular for designing high level design
- No industry standard for notation

Q Algorithm for ER Model to Relational Model mapping

Ans:

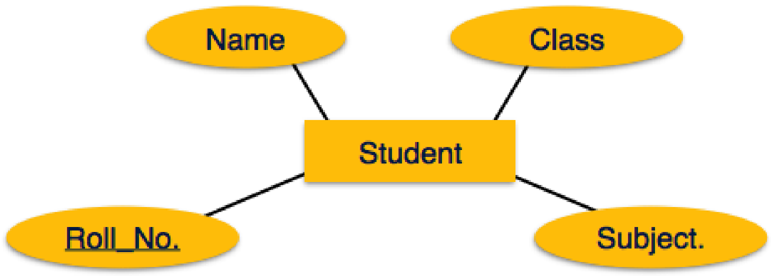
There are several processes and algorithms available to convert ER Diagrams into Relational Schema. Some of them are automated and some of them are manual. We may focus here on the mapping diagram contents to relational basics.

ER diagrams mainly comprise of −

- Entity and its attributes
- Relationship, which is association among entities.

Mapping Entity

An entity is a real-world object with some attributes.

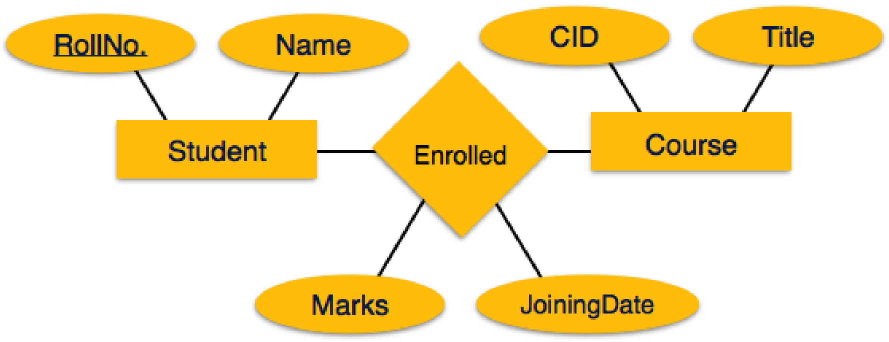


Mapping Process (Algorithm)

- Create a table for each entity.
- Entity's attributes should become fields of tables with their respective data types.
- Declare primary key.

Mapping Relationship

A relationship is an association among entities.

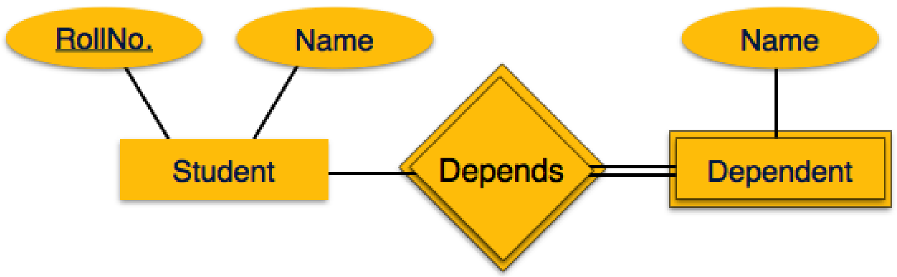


Mapping Process

- Create a table for a relationship.
- Add the primary keys of all participating Entities as fields of table with their respective data types.
- If a relationship has any attribute, add each attribute as a field of table.
- Declare a primary key composing all the primary keys of participating entities.
- Declare all foreign key constraints.

Mapping Weak Entity Sets

A weak entity set is one which does not have any primary key associated with it.

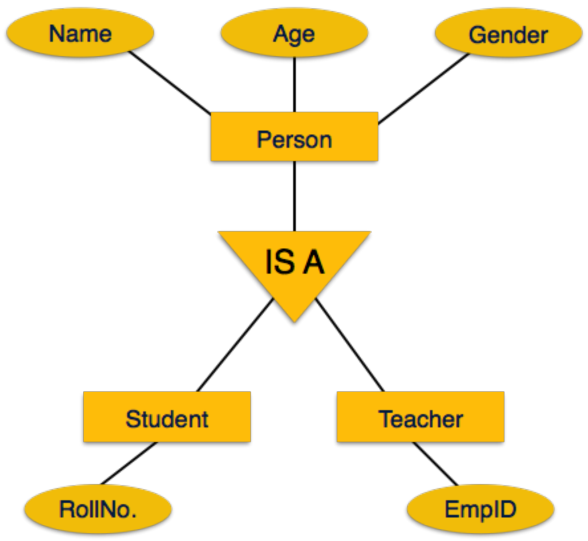


Mapping Process

- Create a table for weak entity sets.
- Add all its attributes to the table as a field.
- Add the primary key of identifying entity sets.
- Declare all foreign key constraints.

Mapping Hierarchical Entities

ER specialization or generalization comes in the form of hierarchical entity sets.



Mapping Process

- Create tables for all higher-level entities.

- Create tables for lower-level entities.
- Add primary keys of higher-level entities in the table of lower-level entities.
- In lower-level tables, add all other attributes of lower-level entities.
- Declare primary key for higher-level table and the primary key for lower-level table.
- Declare foreign key constraints.

Q Difference between Strong and Weak Entity:

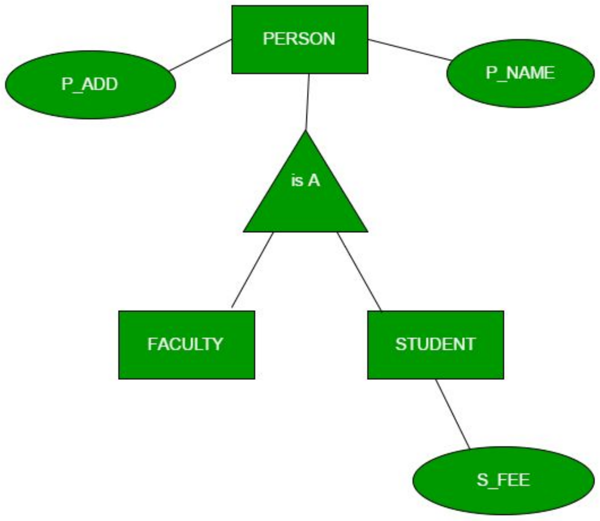
S.NO	STRONG ENTITY	WEAK ENTITY
1.	Strong entity always has a primary key.	While the weak entity has a partial discriminator key.
2.	Strong entity is not dependent on any other entity.	Weak entities depend on strong entities.
3.	Strong entity is represented by a single rectangle.	Weak entity is represented by a double rectangle.
4.	Two strong entity's relationships are represented by a single diamond.	While the relation between one strong and one weak entity is represented by a double diamond.
5.	Strong entities have either total participation or not.	While a weak entity always has total participation.

Q Explain Generalization, Specialization and Aggregation in ER Model

Ans:

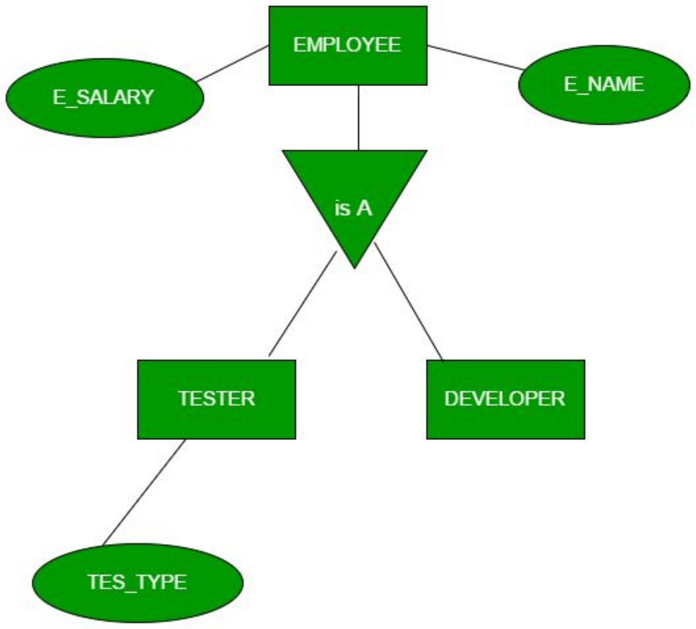
Generalization –

Generalization is the process of extracting common properties from a set of entities and creating a generalized entity from it. It is a bottom-up approach in which two or more entities can be generalized to a higher level entity if they have some attributes in common. For Example, STUDENT and FACULTY can be generalized to a higher level entity called PERSON as shown in Figure 1. In this case, common attributes like P_NAME, P_ADD become part of higher entity (PERSON) and specialized attributes like S_FEE become part of specialized entity (STUDENT).



Specialization –

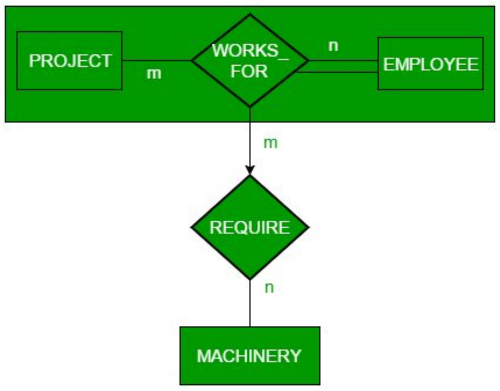
In specialization, an entity is divided into sub-entities based on their characteristics. It is a top-down approach where a higher level entity is specialized into two or more lower level entities. For Example, EMPLOYEE entity in an Employee management system can be specialized into DEVELOPER, TESTER etc. as shown in Figure 2. In this case, common attributes like E_NAME, E_SAL etc. become part of higher entities (EMPLOYEE) and specialized attributes like TES_TYPE become part of specialized entities (TESTER).



Specialization

Aggregation –

An ER diagram is not capable of representing the relationship between an entity and a relationship which may be required in some scenarios. In those cases, a relationship with its corresponding entities is aggregated into a higher level entity. For Example, Employees working for a project may require some machinery. So, REQUIRE relationship is needed between relationship WORKS_FOR and entity MACHINERY. Using aggregation, WORKS_FOR relationship with its entities EMPLOYEE and PROJECT is aggregated into a single entity and relationship REQUIRE is created between the aggregated entity and MACHINERY.



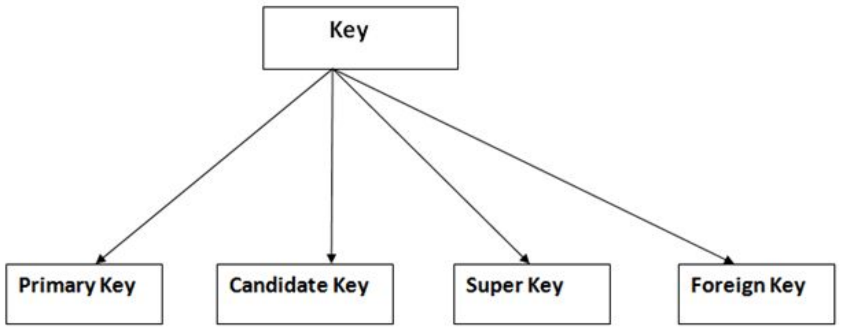
Aggregation

Hrushikesh

Q Explain various types of keys

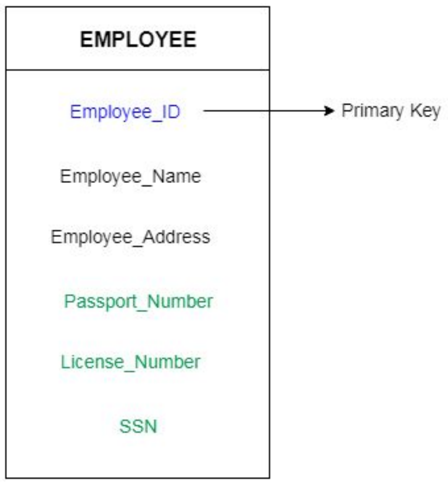
Ans:

Types of key:



1. Primary key

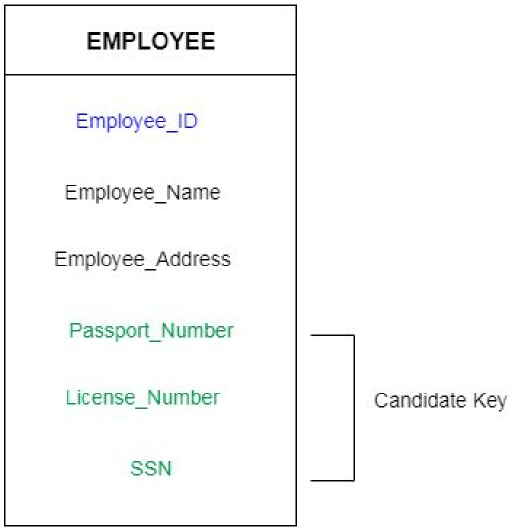
- It is the first key which is used to identify one and only one instance of an entity uniquely. An entity can contain multiple keys as we saw in the PERSON table. The key which is most suitable from those lists becomes a primary key.
- In the EMPLOYEE table, ID can be the primary key since it is unique for each employee. In the EMPLOYEE table, we can even select License_Number and Passport_Number as primary keys since they are also unique.
- For each entity, selection of the primary key is based on requirement and developers.



2. Candidate key

- A candidate key is an attribute or set of an attribute which can uniquely identify a tuple.
- The remaining attributes except for the primary key are considered as a candidate key. The candidate keys are as strong as the primary key.

For example: In the EMPLOYEE table, id is best suited for the primary key. Rest of the attributes like SSN, Passport_Number, and License_Number, etc. are considered as a candidate key.



3. Super Key

Super key is a set of attributes which can uniquely identify a tuple. Super key is a superset of a candidate key.

For example: In the above EMPLOYEE table, for(EMPLOEE_ID, EMPLOYEE_NAME) the name of two employees can be the same, but their EMPLYEE_ID can't be the same. Hence, this combination can also be a key.

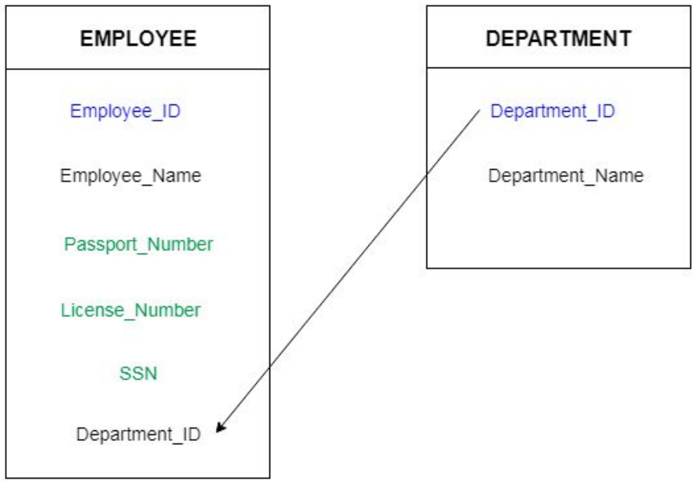
The super key would be EMPLOYEE-ID, (EMPLOYEE_ID, EMPLOYEE-NAME), etc.

4. Foreign key

- Foreign keys are the column of the table which is used to point to the primary key of another table.
- In a company, every employee works in a specific department, and employees and departments are two different entities. So we can't store the information of

the department in the employee table. That's why we link these two tables through the primary key of one table.

- We add the primary key of the DEPARTMENT table, Department_Id as a new attribute in the EMPLOYEE table.
- Now in the EMPLOYEE table, Department_Id is the foreign key, and both the tables are related.



Q Explain Constraints in DBMS

Ans:

Constraints are the rules that we can apply on the type of data in a table. That is, we can specify the limit on the type of data that can be stored in a particular column in a table using constraints.

The available constraints in SQL are:

NOT NULL: This constraint tells that we cannot store a null value in a column. That is, if a column is specified as NOT NULL then we will not be able to store null in this particular column any more.

For example, the below query creates a table Student with the fields ID and NAME as NOT NULL. That is, we are bound to specify values for these two fields every time we wish to insert a new row.

```
CREATE TABLE Student

(

ID int(6) NOT NULL,

NAME varchar(10) NOT NULL,

ADDRESS varchar(20)
```

);

UNIQUE: This constraint when specified with a column, tells that all the values in the column must be unique. That is, the values in any row of a column must not be repeated.

For example, the below query creates a tale Student where the field ID is specified as UNIQUE. i.e, no two students can have the same ID. Unique constraint in detail.

```
CREATE TABLE Student

(

ID int(6) NOT NULL UNIQUE,

NAME varchar(10),

ADDRESS varchar(20)

);
```

PRIMARY KEY: A primary key is a field which can uniquely identify each row in a table. And this constraint is used to specify a field in a table as the primary key.

A table can have only one field as primary key.Below query will create a table named Student and specify the field ID as primary key.

```
CREATE TABLE Student

(

ID int(6) NOT NULL UNIQUE,

NAME varchar(10),

ADDRESS varchar(20),

PRIMARY KEY(ID)

);
```

FOREIGN KEY: A Foreign key is a field which can uniquely identify each row in another table. And this constraint is used to specify a field as Foreign key.

Syntax:

```
CREATE TABLE Orders

(

O_ID int NOT NULL,

ORDER_NO int NOT NULL,

C_ID int,
PRIMARY KEY (O_ID),

FOREIGN KEY (C_ID) REFERENCES Customers(C_ID)

)
```

CHECK: This constraint helps to validate the values of a column to meet a particular condition. That is, it helps to ensure that the value stored in a column meets a specific condition.

For example, the below query creates a table Student and specifies the condition for the field AGE as (AGE >= 18). That is, the user will not be allowed to enter any record in the table with AGE < 18.

```
CREATE TABLE Student

(

ID int(6) NOT NULL,

NAME varchar(10) NOT NULL,

AGE int NOT NULL CHECK (AGE >= 18)

);
```

DEFAULT: This constraint specifies a default value for the column when no value is specified by the user.

For example, the below query will create a table named Student and specify the default value for the field AGE as 18.

```
CREATE TABLE Student

(

ID int(6) NOT NULL,

NAME varchar(10) NOT NULL,

AGE int DEFAULT 18

);
```

Q Explain Extended Entity-Relationship (EE-R) Model

Ans:

EER is a high-level data model that incorporates the extensions to the original ER model. Enhanced ERD are high level models that represent the requirements and complexities of complex databases.

In addition to ER model concepts EE-R includes −

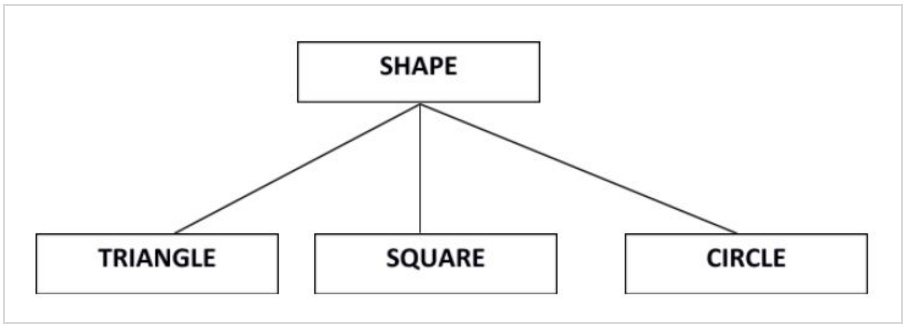
- Subclasses and Superclasses.
- Specialization and Generalization.
- Category or union type.
- Aggregation.

These concepts are used to create EE-R diagrams.

Subclasses and Super class

Super class is an entity that can be divided into further subtype.

For **example** − consider Shape super class.

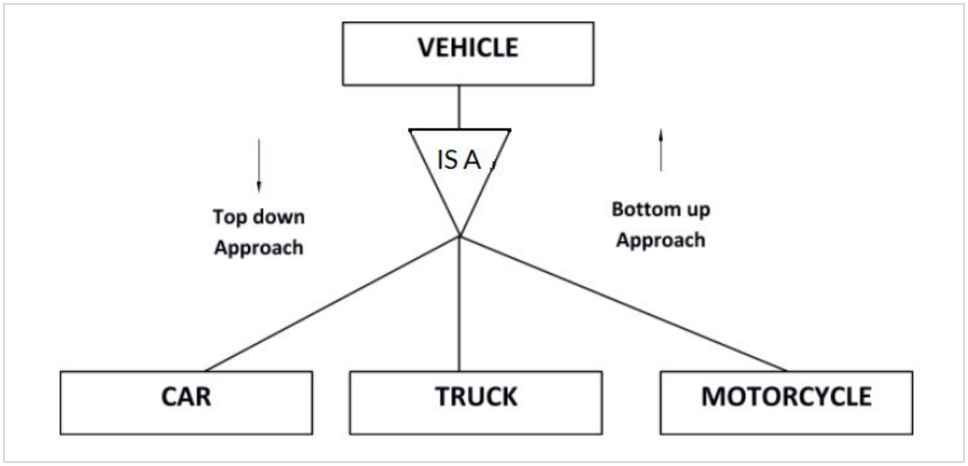


Super class shape has sub groups: Triangle, Square and Circle.

Subclasses are the group of entities with some unique attributes.Sub class inherits the properties and attributes from super class.

Specialization and Generalization

Generalization is a process of generalizing an entity which contains generalized attributes or properties of generalized entities.



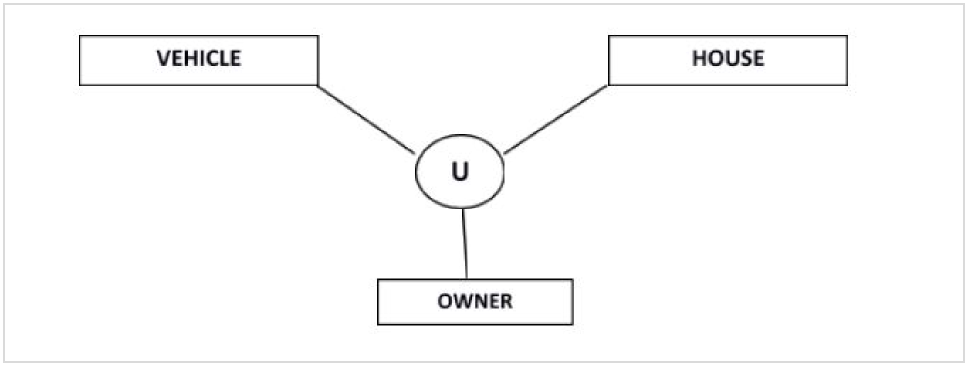
It is a Bottom up process i.e. consider we have 3 sub entities Car, Truck and Motorcycle. Now these three entities can be generalized into one superclass named as Vehicle.

Specialization is a process of identifying subsets of an entity that share some different characteristics. It is a top down approach in which one entity is broken down into low level entities.

In the above example Vehicle entity can be a Car, Truck or Motorcycle.

Category or Union

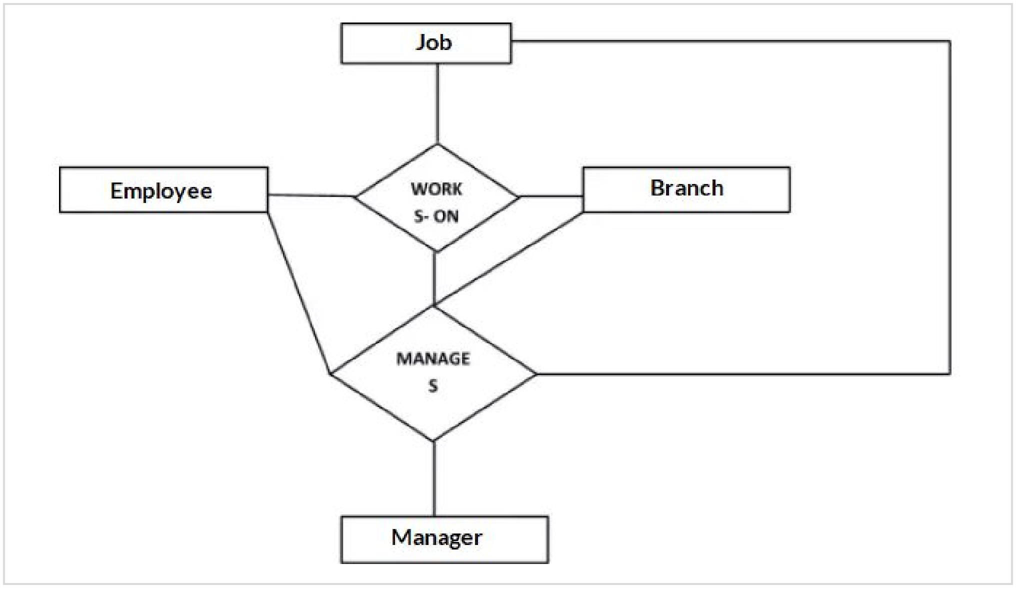
Relationship of one super or sub class with more than one superclass.



Owner is the subset of two super classes: Vehicle and House.

Aggregation

Represents relationship between a whole object and its component.



Consider a ternary relationship Works_On between Employee, Branch and Manager. Now the best way to model this situation is to use aggregation, So, the relationship-set, Works_On is a higher level entity-set. Such an entity-set is treated in the same manner as any other entity-set. We can create a binary relationship, Manager, between Works_On and Manager to represent who manages what tasks.

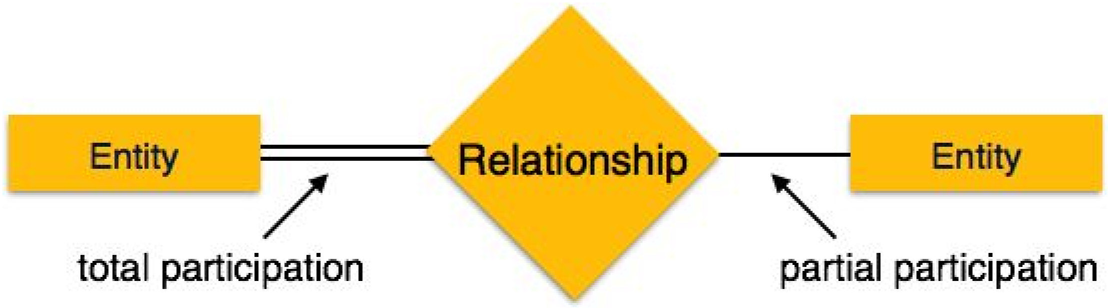
Q Explain Total Participation and Partial Participation with example.

Ans:

The participation constraint specifies whether the existence of an entity depends on its being related to another entity via the relationship type. This constraint specifies the minimum number of relationship instances that each entity can participate in.

There are two types of participation constraints: Total and Partial

- **Total Participation**
  Total Participation is when each entity in the entity set occurs in at least one relationship in that relationship set.
  For instance, consider the relationship borrower between customers and loans. A double line from loan to borrower, as shown in figure below indicates that each loan must have at least one associated customer.



- **Partial Participation**
  Partial Participation is when each entity in the entity set may not occur in at least one relationship in that relationship set. For instance, If a company policy states that employee (manager) must manage a department, However every employee may not manage a department, so the participation of EMPLOYEE in the MANAGES relationship type is partial, meaning that some or part of the set of employee entities are related to some department entity via MANAGES, but not necessarily all.
  Note: Partial Participation is represented by a single line connecting entities in relationship.

Q Explain Deadlock. Explain deadlock prevention, detection and recovery.

Ans:

A deadlock is a condition where two or more transactions are waiting indefinitely for one another to give up locks. Deadlock is said to be one of the most feared complications in DBMS as no task ever gets finished and is in a waiting state forever.

For example: In the student table, transaction T1 holds a lock on some rows and needs to update some rows in the grade table. Simultaneously, transaction T2 holds locks on some rows in the grade table and needs to update the rows in the Student table held by Transaction T1.

Now, the main problem arises. Now Transaction T1 is waiting for T2 to release its lock and similarly, transaction T2 is waiting for T1 to release its lock. All activities come to a halt state and remain at a standstill. It will remain in a standstill until the DBMS detects the deadlock and aborts one of the transactions.

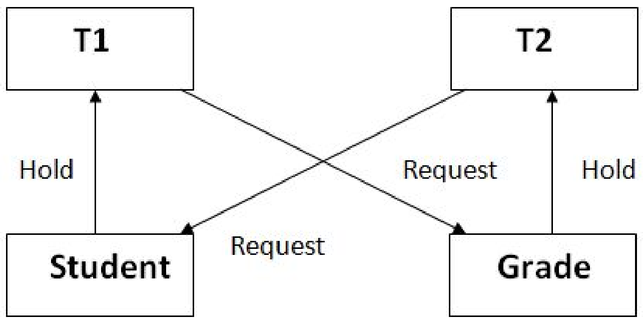


Figure: Deadlock in DBMS

Deadlock Avoidance

When a database is stuck in a deadlock state, then it is better to avoid the database rather than aborting or restating the database. This is a waste of time and resources.

Deadlock avoidance mechanism is used to detect any deadlock situation in advance. A method like "wait for graph" is used for detecting the deadlock situation but this method is suitable only for the smaller database. For the larger database, deadlock prevention methods can be used.

Deadlock Detection

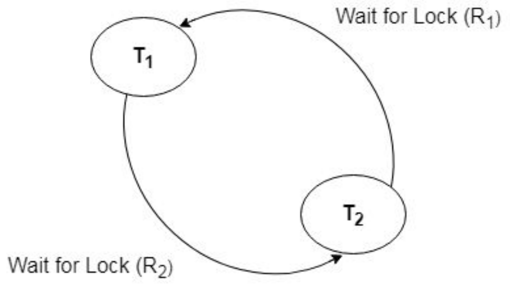
In a database, when a transaction waits indefinitely to obtain a lock, then the DBMS should detect whether the transaction is involved in a deadlock or not. The lock manager maintains a Wait for the graph to detect the deadlock cycle in the database.

- **Wait for Graph**

This is the suitable method for deadlock detection. In this method, a graph is created based on the transaction and their lock. If the created graph has a cycle or closed loop, then there is a deadlock.

The wait for the graph is maintained by the system for every transaction which is waiting for some data held by the others. The system keeps checking the graph if there is any cycle in the graph.

The wait for a graph for the above scenario is shown below:



Deadlock Prevention

- Deadlock prevention method is suitable for a large database. If the resources are allocated in such a way that deadlock never occurs, then the deadlock can be prevented.
- The Database management system analyzes the operations of the transaction whether they can create a deadlock situation or not. If they do, then the DBMS never allowed that transaction to be executed.

1. Wait-Die scheme

In this scheme, if a transaction requests for a resource which is already held with a conflicting lock by another transaction then the DBMS simply checks the timestamp of both transactions. It allows the older transaction to wait until the resource is available for execution.

Let's assume there are two transactions Ti and Tj and let TS(T) is a timestamp of any transaction T. If T2 holds a lock by some other transaction and T1 is requesting for resources held by T2 then the following actions are performed by DBMS:

1. Check if TS(Ti) < TS(Tj) - If Ti is the older transaction and Tj has held some resource, then Ti is allowed to wait until the data-item is available for execution. That means if the older transaction is waiting for a resource which is locked by the younger transaction, then the older transaction is allowed to wait for the resource until it is available.
2. Check if TS(T_i) < TS(Tj) - If Ti is an older transaction and has held some resource and if Tj is waiting for it, then Tj is killed and restarted later with the random delay but with the same timestamp.

2. **Wound wait scheme**

- In a wound wait scheme, if the older transaction requests for a resource which is held by the younger transaction, then the older transaction forces the younger one to kill the transaction and release the resource. After the minute delay, the younger transaction is restarted but with the same timestamp.
- If the older transaction has held a resource which is requested by the Younger transaction, then the younger transaction is asked to wait until the older releases it.

Deadlock Recovery

A traditional operating system such as Windows doesn't deal with deadlock recovery as it is a time and space consuming process. Real-time operating systems use Deadlock recovery.

1. **Recovery methodKilling the process:** killing all the processes involved in the deadlock. Killing process one by one. After killing each process check for deadlock again and keep repeating the process till the system recovers from the deadlock.
2. **Resource Preemption:** Resources are preempted from the processes involved in the deadlock, preempted resources are allocated to other processes so that there is a possibility of recovering the system from deadlock. In this case, the system goes into starvation.

Q What is normalization? Explain 1NF, 2NF, 3NF and BCNF giving examples.

Ans:

Database normalization is a technique of organizing the data in the database.
Normalization of data can be considered a process of analysing the given relation schemas based on their Functional Dependencies and primary keys to achieve the following properties:

i. Minimizing redundancy

ii. Minimizing the insertion, deletion, and update anomalies

iii. Ensuring data is stored in correct table

It can be considered as a filtering process to make the design have successively better quality. It is a multi-step process that puts data into tabular form by removing duplicated data from the relation tables. Without normalization it becomes difficult to handle and update databases without facing data loss.

The various forms of normalization are described below:

I. **First Normal Form (1NF):**

- First normal form (1NF) states that the domain of an attribute must include only atomic values and that the value of any attribute in a tuple must be a single value from the domain of that attribute.
- A relation is said to be in 1NF if it contains no non-atomic values and each row can provide a unique combination of values.
- 1NF disallows having a set of values, a tuple of values, or a combination of both as an attribute value for a single tuple.
- The only attribute values permitted by 1NF are single atomic values.
- Example: Un-Normalized Table-

Student	**Age**	**Subject**
Rooney	15	Java, C++
Kane	16	HTML, PHP

Normalized Table: Any Row must not have a column in which more than one value is saved, instead data is separated in multiple rows as shown below.

Student	**Age**	**Subject**
Rooney	15	JAVA
Rooney	15	C++
Kane	16	HTML
Kane	16	PHP

II. **Second Normal Form (2NF):**

- A relation is said to be in 2NF, if it is already in 1NF and each and every attribute fully depends on the primary key of the relation.
- There must not be any partial dependency of any column on the primary key.
- Second normal form (2NF) is based on the concept of full functional dependency. A functional dependency X -> Y is a full functional dependency if removal of any attribute A from X means that the dependency does not hold any more.
- A functional dependency X->Y is a partial dependency if some attribute A belongs to X can be removed from X and the dependency still holds.
- Example:

Student_Project Table

Stud_ID	**Proj_ID**	**Stud_Name**	**Proj_Name**
100	001	Rooney	Cloud
200	002	Kane	Servers

Stud_Name depends on Stud_ID and Proj_Name depends on Proj_ID

The above table can be normalized to 2NF as shown below.

Student Table in 2NF

Stud_ID	**Proj_ID**	**Stud_Name**
100	001	Rooney
200	001	Kane

Project Table in 2NF

Proj_ID	**Proj_Name**
001	001
002	Servers

III. **Third Normal Form (3NF):**

- A relation is said to be in 3NF, if it is already in 2NF and there exists no transitive dependency in that relation.
- If a table contains transitive dependency, then it is not in 3NF, and the table must be split to bring it into 3NF.
- What is a transitive dependency?
  A -> B [B depends on A] & B -> C [C depends on B]
  Then A -> C[C depends on A] can be derived.
- Example:Below table not in 3NF

Stud_ID	**Stud_Name**	**City**	**Zip**
100	Rooney	Manchester	4001
200	Kane	Stoke	4002

Stud_ID is the only prime key attribute. City can be identified by Stu_ID as well as Zip. Neither Zip is a superkey nor City is a prime attribute.

Stud_ID -> Zip -> City, so there exists transitive dependency. Hence 3NF table is below

Student_Detail

Stud_ID	**Stud_Name**	**Zip**
100	Rooney	4001
200	Kane	4002

Zip_Code

Zip	**City**
4001	Manchester
4002	Stoke

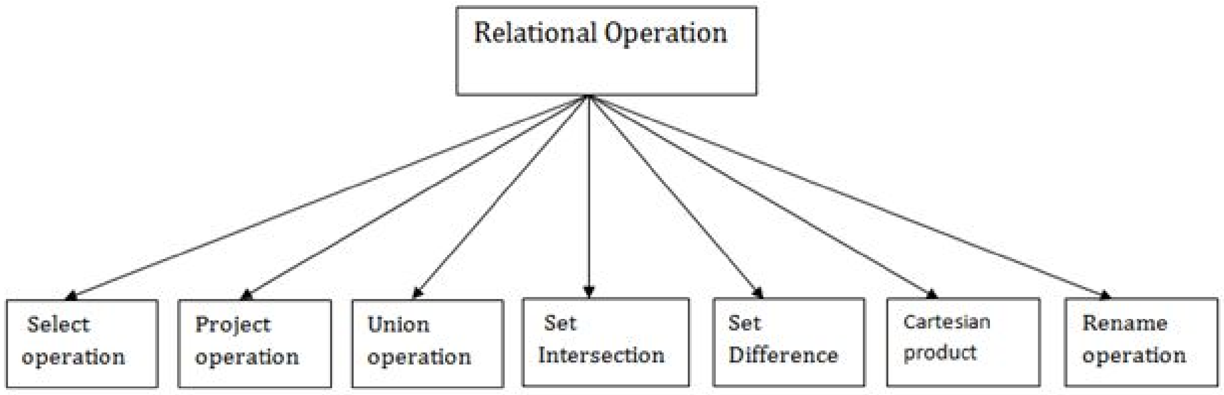
IV. **Boyce-Codd Normal Form (BCNF):**

- BCNF is an extension of Third Normal Form in strict way.

- A relationship is said to be in BCNF if it is already in 3NF and for any non-trivial functional dependency, X -> A, then X must be a super-key.
- A 3NF table which does not have multiple overlapping candidate keys is said to be in BCNF.
- Example:
  In the 3NF example,Stud_ID is super-key in Student_Detail relation and Zip is super-key in ZipCodes relation.
  So Stud_ID ->Stud_Name, Zip and
  Zip ->City
  Confirms that both relations are in BCNF.

Q Explain Relational Algebra operations with examples.

Ans:
Relational algebra is a procedural query language. It gives a step by step process to obtain the result of the query. It uses operators to perform queries.



1. **Select Operation (σ)**

It selects tuples that satisfy the given predicate from a relation.
Notation − σp(r)
Where σ stands for selection predicate and r stands for relation. p is a propositional logic formula which may use connectors like and, or, and not. These terms may use relational operators like − =, ≠, ≥, < , >, ≤.

For example −
σsubject = "database"(Books)

Output − Select tuples from books where the subject is 'database'.
σsubject = "database" and price = "450"(Books)

Output − Select tuples from books where subject is 'database' and 'price' is 450.
σsubject = "database" and price = "450" or year > "2010"(Books)

Output − Select tuples from books where the subject is 'database' and 'price' is 450 or those books published after 2010.

2. **Project Operation (∏)**

It projects column(s) that satisfy a given predicate.
Notation − ∏A1, A2, An (r)
Where A1, A2 , and are attribute names of relation r.
Duplicate rows are automatically eliminated, as relation is a set.

For example −
∏subject, author (Books)
Selects and projects columns named as subject and author from the relation Books.

3. **Union Operation (∪)**

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

r ∪ s = { t | t ∈ r or t ∈ s}
Notation − r U 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.

Example:
∏ author (Books) ∪ ∏ 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.
Example:
∏ author (Books) − ∏ 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 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}

Example:

σauthor = 'tutorialspoint'(Books X Articles)

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

6. **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 a small Greek letter rho ρ.

Notation − ρ x (E)

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

Additional operations are −

- Set intersection
- Assignment
- Natural join

7. **Set Intersection:**

Suppose there are two tuples R and S. The set intersection operation contains all tuples that are in both R & S.

It is denoted by intersection ∩.

Notation: R ∩ S

Example: Using the above DEPOSITOR table and BORROW table

Input:

∏ CUSTOMER_NAME (BORROW) ∩ ∏ CUSTOMER_NAME (DEPOSITOR)

Hrushikesh

Output:

CUSTOMER_NAME
Smith
Jones

Q Explain log based recovery

Ans

Log-Based Recovery

The log is a sequence of records. Log of each transaction is maintained in some stable storage so that if any failure occurs, then it can be recovered from there.

If any operation is performed on the database, then it will be recorded in the log.

But the process of storing the logs should be done before the actual transaction is applied in the database.

Let's assume there is a transaction to modify the City of a student. The following logs are written for this transaction.

When the transaction is initiated, then it writes 'start' log.

<Tn, Start>

When the transaction modifies the City from 'Noida' to 'Bangalore', then another log is written to the file.

<Tn, City, 'Noida', 'Bangalore' >

When the transaction is finished, then it writes another log to indicate the end of the transaction.

<Tn, Commit>

There are two approaches to modify the database:

1. Deferred database modification:

The deferred modification technique occurs if the transaction does not modify the database until it has committed.

In this method, all the logs are created and stored in the stable storage, and the database is updated when a transaction commits.

2. Immediate database modification:

The Immediate modification technique occurs if database modification occurs while the transaction is still active.

In this technique, the database is modified immediately after every operation. It follows an actual database modification.

Recovery using Log records
When the system is crashed, then the system consults the log to find which transactions need to be undone and which need to be redone.

If the log contains the record <Ti, Start> and <Ti, Commit> or <Ti, Commit>, then the Transaction Ti needs to be redone.
If log contains record<Tn, Start> but does not contain the record either <Ti, commit> or <Ti, abort>, then the Transaction Ti needs to be undone.

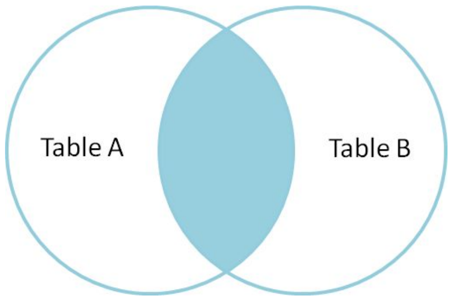
Q Explain SQL Join Statements

Ans:
A SQL Join statement is used to combine data or rows from two or more tables based on a common field between them. Different types of Joins are:

- INNER JOIN
- LEFT JOIN
- RIGHT JOIN
- FULL JOIN

INNER JOIN: The INNER JOIN keyword selects all rows from both the tables as long as the condition satisfies. This keyword will create the result-set by combining all rows from both the tables where the condition satisfies i.e value of the common field will be the same.
Syntax:
```
SELECT table1.column1,table1.column2,table2.column1,....
FROM table1
INNER JOIN table2
ON table1.matching_column = table2.matching_column;
```

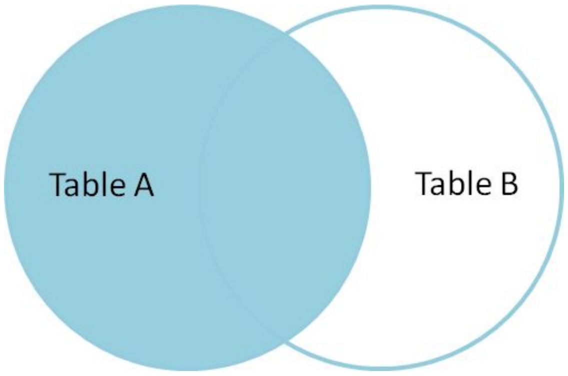
Note: We can also write JOIN instead of INNER JOIN. JOIN is the same as INNER JOIN.



LEFT JOIN: This join returns all the rows of the table on the left side of the join and matching rows for the table on the right side of the join. The rows for which there is no matching row on the right side, the result-set will contain null. LEFT JOIN is also known as LEFT OUTER JOIN.

Syntax:

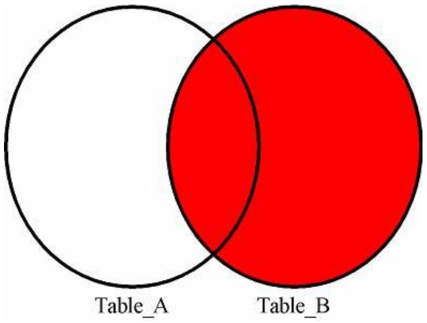
```
SELECT table1.column1,table1.column2,table2.column1,....
FROM table1
LEFT JOIN table2
ON table1.matching_column = table2.matching_column;
```



RIGHT JOIN: RIGHT JOIN is similar to LEFT JOIN. This join returns all the rows of the table on the right side of the join and matching rows for the table on the left side of the join. The rows for which there is no matching row on the left side, the result-set will contain null. RIGHT JOIN is also known as RIGHT OUTER JOIN.

Syntax:

```
SELECT table1.column1,table1.column2,table2.column1,....
FROM table1
RIGHT JOIN table2
ON table1.matching_column = table2.matching_column;
```

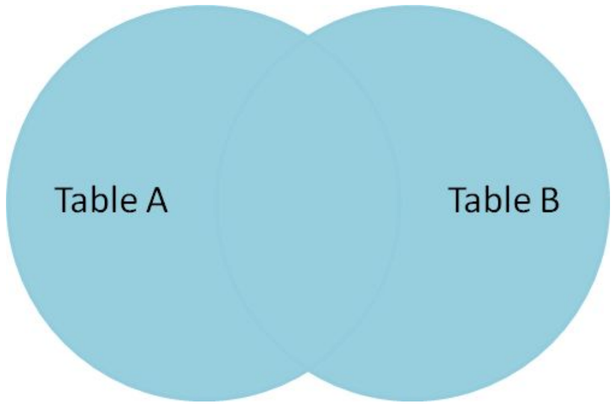


FULL JOIN: FULL JOIN creates the result-set by combining the result of both LEFT JOIN and RIGHT JOIN. The result-set will contain all the rows from both the tables. The rows for which there is no matching, the result-set will contain NULL values.

Syntax:

```
SELECT table1.column1,table1.column2,table2.column1,....
```

```
FROM table1
FULL JOIN table2
ON table1.matching_column = table2.matching_column;
```



Q Explain transaction control commands

Ans

The following commands are used to control transactions.

- COMMIT − to save the changes.
- ROLLBACK − to roll back the changes.
- SAVEPOINT − creates points within the groups of transactions in which to ROLLBACK.
- SET TRANSACTION − Places a name on a transaction.

Transactional Control Commands

Transactional control commands are only used with the DML Commands such as - INSERT, UPDATE and DELETE only. They cannot be used while creating tables or dropping them because these operations are automatically committed in the database.

The COMMIT Command

The COMMIT command is the transactional command used to save changes invoked by a transaction to the database.

The COMMIT command is the transactional command used to save changes invoked by a transaction to the database. The COMMIT command saves all the transactions to the database since the last COMMIT or ROLLBACK command.

The syntax for the COMMIT command is as follows.

```
COMMIT;
```

Example

```
SQL> DELETE FROM CUSTOMERS
```

```
  WHERE AGE = 25;
SQL> COMMIT;
```

The ROLLBACK Command

The ROLLBACK command is the transactional command used to undo transactions that have not already been saved to the database. This command can only be used to undo transactions since the last COMMIT or ROLLBACK command was issued.

The syntax for a ROLLBACK command is as follows −

ROLLBACK;

Example

```
SQL> DELETE FROM CUSTOMERS
  WHERE AGE = 25;
SQL> ROLLBACK;
```

The SAVEPOINT Command

A SAVEPOINT is a point in a transaction when you can roll the transaction back to a certain point without rolling back the entire transaction.

The syntax for a SAVEPOINT command is as shown below.

SAVEPOINT SAVEPOINT_NAME;

This command serves only in the creation of a SAVEPOINT among all the transactional statements. The ROLLBACK command is used to undo a group of transactions.

The syntax for rolling back to a SAVEPOINT is as shown below.

ROLLBACK TO SAVEPOINT_NAME;

Example

```
SQL> SAVEPOINT SP1;
Savepoint created.
SQL> DELETE FROM CUSTOMERS WHERE ID=1;
1 row deleted

SQL> ROLLBACK TO SP2;
Rollback complete.
```

The RELEASE SAVEPOINT Command
The RELEASE SAVEPOINT command is used to remove a SAVEPOINT that you have created.

The syntax for a RELEASE SAVEPOINT command is as follows.

```
RELEASE SAVEPOINT SAVEPOINT_NAME;
```

Once a SAVEPOINT has been released, you can no longer use the ROLLBACK command to undo transactions performed since the last SAVEPOINT.

The SET TRANSACTION Command

The SET TRANSACTION command can be used to initiate a database transaction. This command is used to specify characteristics for the transaction that follows. For example, you can specify a transaction to be read only or read write.

The syntax for a SET TRANSACTION command is as follows.

```
SET TRANSACTION [ READ WRITE | READ ONLY ];
```

Q Explain Triggers

Ans:
A trigger is a stored procedure in a database which automatically invokes whenever a special event in the database occurs. For example, a trigger can be invoked when a row is inserted into a specified table or when certain table columns are being updated.

Syntax:

```
create trigger [trigger_name]
[before | after]
{insert | update | delete}
on [table_name]
[for each row]
[trigger_body]
```

Explanation of syntax:
create trigger [trigger_name]: Creates or replaces an existing trigger with the trigger_name.
[before | after]: This specifies when the trigger will be executed.
{insert | update | delete}: This specifies the DML operation.

on [table_name]: This specifies the name of the table associated with the trigger.
[for each row]: This specifies a row-level trigger, i.e., the trigger will be executed for each row being affected.
[trigger_body]: This provides the operation to be performed as trigger is fired
BEFORE and AFTER of Trigger:

- BEFORE triggers run the trigger action before the triggering statement is run.
- AFTER triggers run the trigger action after the triggering statement is run.

Example

To start with, we will be using the CUSTOMERS table we had created and used in the previous chapters −

```
Select * from customers;
```

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
+----+----------+-----+-----------+----------+
```

The following program creates a row-level trigger for the customers table that would fire for INSERT or UPDATE or DELETE operations performed on the CUSTOMERS table. This trigger will display the salary difference between the old values and new values −

```
CREATE OR REPLACE TRIGGER display_salary_changes
BEFORE DELETE OR INSERT OR UPDATE ON customers
FOR EACH ROW
WHEN (NEW.ID > 0)
DECLARE
   sal_diff number;
BEGIN
   sal_diff := :NEW.salary  - :OLD.salary;
   dbms_output.put_line('Old salary: ' || :OLD.salary);
   dbms_output.put_line('New salary: ' || :NEW.salary);
   dbms_output.put_line('Salary difference: ' || sal_diff);
END;
/
```

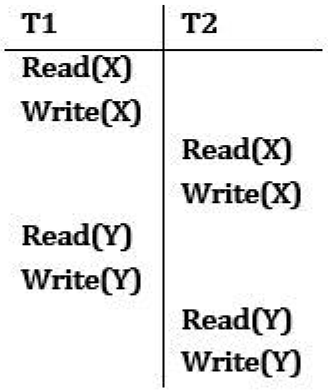
Q Describe conflict serializability and view serializability with examples.

Ans:
A schedule is serializable if it is equivalent to a serial schedule.

- **Conflict serializability:**
    - Instructions Ii and Ij, of transactions Ti and Tj respectively, conflict if and only if there exists some item P accessed by both Ii and Ij, and at least one of these instructions wrote P.
    - Consider the below operations-
      i. Ii = read(P), Ij = read(P). Ii and Ij don't conflict.
      ii. Ii = read(P), Ij = write(P). They conflict.
      iii. Ii = write(P), Ij = read(P). They conflict.
      iv. Ii = write(P), Ij = write(P). They conflict.
- A conflict between Ii and Ij forces a temporal order between them.
- If Ii and Ij are consecutive in a schedule and they do not conflict, their results would remain the same even if they had been interchanged in the schedule.
- If a schedule can be transformed into a schedule S by a series of swaps of non-conflicting instructions, then S and S` are equivalent.
- In other words a schedule Is conflict serializable if it is conflict equivalent to a serial schedule.
- Example of a schedule that is not conflict serializable:

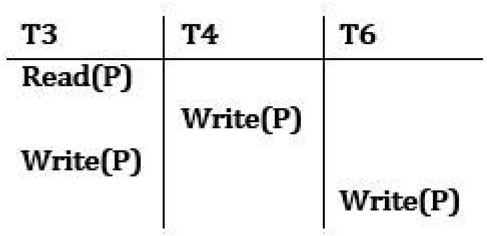
T3	**T4**
Read(P)	
	Write(P)
Write(P)	

- The instructions cannot be swapped in the above schedule to obtain either the serial schedule < T 3 , T 4 >, or the serial schedule < T 4 ,T 3 >.
- 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:**
  - S and S` are view equivalent if the following three conditions are met:
    i. For each data item P, if transaction Ti reads the initial value of P in schedule S, then transaction Ti must, in schedule S`, also read the initial value of P.
    ii. For each data item P, if transaction Ti executes read (P)in schedule S, and that value was produced by transaction Tj, then transaction Ti must in schedule S` also read the value of P that was produced by transaction Tj.
    iii. For each data item P, the transaction that performs the final write(P) operation in schedule S must perform the final write(P) operation in schedule S`.
  - View equivalence is also based purely on reads and writes alone.
  - A schedule S is view serializable if it is 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)

Q Explain Referential Integrity

Ans:

• A value appearing in a one table for a given set of attributes also appears for another set of attributes in another table. This is called referential integrity.

• A referential integrity constraint is specified between two tables and is used to maintain the consistency among tuples in the two relations.

• The tuple in one table refers only to an existing table in another relation.

Emp Table

Emp_id	**Emp_name**	**Did**
1	Sanjay	20
2	Simran	10
3	Jay	20
4	Neha	10

Department Table

Did	**Dept_name**
10	HR
20	TIS
30	L&D

• In the above example “Emp” table has “Did” as foreign key reference this is called Referential integrity.

• Here we are forcing the database to check the “Did” value key from the “Department” table while inserting any value of “Emp” table in Did column if there is no value existing in the department table of that “Did” then we can not insert that value in “Emp” table.

• This helps to maintain data consistency.

Referential integrity in SQL

• Foreign key is used to show relation into two tables in relational algebra.

• This helps in maintaining consistency in the database, as foreign keys cannot be inserted, deleted or updated.

Syntax:

```
          FOREIGN KEY
```

```
            REFERENCES [schema_name.] referenced_table_name [(ref_column)]

            [ON DELETE {NO ACTION

            |CASCADE

            |SET NULL

            |SET UPDATE}]

            [ON UPDATE {NO ACTION

            |CASCADE

            |SET NULL

            |SET DEFAULT}]
```

Example:

```
             Create table Emp(Emp_id integer,

              Emp_namevarchar(100) not null,

              Did as integer,

              Primary key(Emp_id),

              Foreign key (Did) references department

              On delete cascade

              On update cascade

             )

             Create table Department(Did integer,

             Dept_name_varchar (100) not null,

             Primary key (did)

             )
```

Q Explain Timestamp ordering protocol

Ans:

Hrushikesh

The Timestamp Ordering Protocol is used to order the transactions based on their Timestamps. The order of transaction is nothing but the ascending order of the transaction creation.
The priority of the older transaction is higher that's why it executes first. To determine the timestamp of the transaction, this protocol uses system time or logical counter.
The lock-based protocol is used to manage the order between conflicting pairs among transactions at the execution time. But Timestamp based protocols start working as soon as a transaction is created.
Let's assume there are two transactions T1 and T2. Suppose the transaction T1 has entered the system at 007 times and transaction T2 has entered the system at 009 times. T1 has the higher priority, so it executes first as it is entered the system first.
The timestamp ordering protocol also maintains the timestamp of last 'read' and 'write' operation on the data.
Basic Timestamp ordering protocol works as follows:

1. Check the following condition whenever a transaction Ti issues a Read (X) operation:
   - If W_TS(X) >TS(Ti) then the operation is rejected.
   - If W_TS(X) <= TS(Ti) then the operation is executed.
   - Timestamps of all the data items are updated.

2. Check the following condition whenever a transaction Ti issues a Write(X) operation:
   - If TS(Ti) < R_TS(X) then the operation is rejected.
   - If TS(Ti) < W_TS(X) then the operation is rejected and Ti is rolled back otherwise the operation is executed.

Where,

TS(TI) denotes the timestamp of the transaction Ti.

R_TS(X) denotes the Read time-stamp of data-item X.

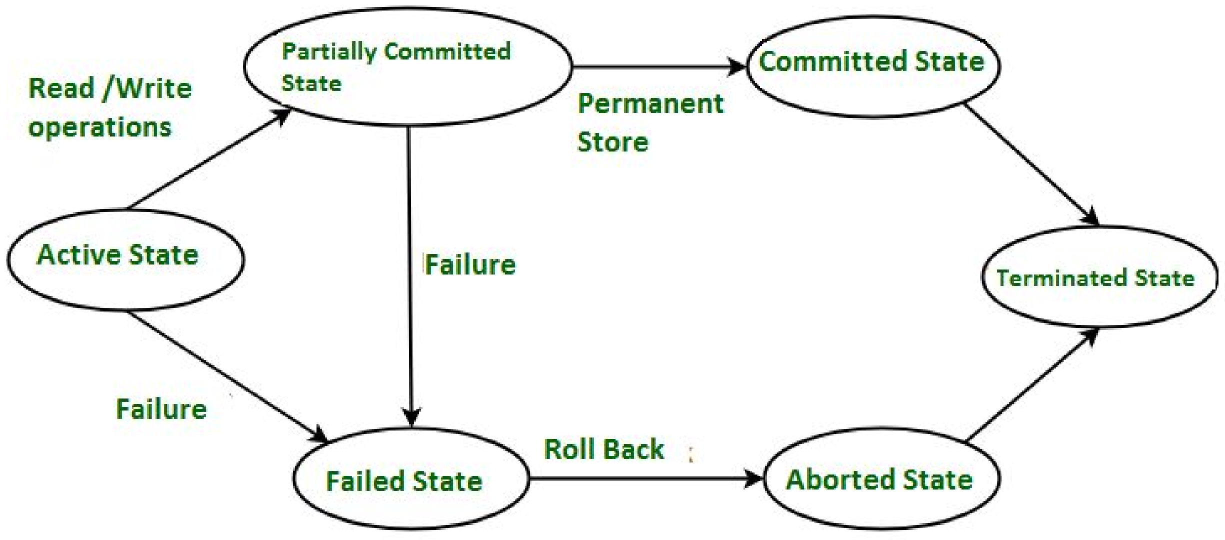
W_TS(X) denotes the Write time-stamp of data-item X.

Advantages and Disadvantages of TO protocol:
- TO protocol ensures serializability since the precedence graph is as follows:
- TS protocol ensures freedom from deadlock that means no transaction ever waits.
- But the schedule may not be recoverable and may not even be cascade- free.

Q Explain Transaction State Diagram

Ans:

States through which a transaction goes during its lifetime. These are the states which tell about the current state of the Transaction and also tell how we will further do processing we will do on the transactions. These states govern the rules which decide the fate of the transaction whether it will commit or abort.



Transaction States in DBMS

These are different types of Transaction States :

Active State –

When the instructions of the transaction are running then the transaction is in active state. If all the read and write operations are performed without any error then it goes to “partially committed state”, if any instruction fails it goes to “failed state”.

Partially Committed –

After completion of all the read and write operations the changes are made in the main memory or local buffer. If the changes are made permanent on the Database then state will change to “committed state” and in case of failure it will go to “failed state”.

Failed State –

When any instruction of the transaction fails it goes to “failed state” or if failure occurs in making permanent change of data on Database.

Aborted State –

After having any type of failure the transaction goes from "failed state" to "aborted state" and in before states the changes are only made to local buffer or main memory and hence these changes are deleted or rollback.

Committed Stage –
It is the stage when the changes are made permanent on the Database and the transaction is complete and therefore terminated in "terminated state".

Terminated State –
If there is any roll back or the transaction comes from a "committed state" then the system is consistent and ready for a new transaction and the old transaction is terminated.

Q Explain Functional Dependency

Ans:

The functional dependency is a relationship that exists between two attributes. It typically exists between the primary key and non-key attribute within a table.

1. X → Y

The left side of FD is known as a determinant, the right side of the production is known as a dependent.

For example:

Assume we have an employee table with attributes: Emp_Id, Emp_Name, Emp_Address.

The Emp_Id attribute can uniquely identify the Emp_Name attribute of the employee table because if we know the Emp_Id, we can tell that employee name associated with it.

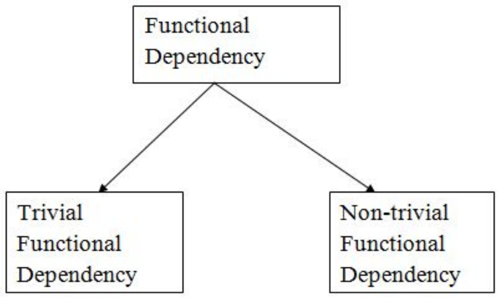
Functional dependency can be written as:

1. Emp_Id → Emp_Name

We can say that Emp_Name is functionally dependent on Emp_Id.

Types of Functional dependency

Hrushikesh


Functional Dependency
Trivial Functional Dependency
Non-trivial Functional Dependency

1. Trivial functional dependency

- A → B has trivial functional dependency if B is a subset of A.
- The following dependencies are also trivial like: A → A, B → B

Example:

1. Consider a table with two columns Employee_Id and Employee_Name.
2. {Employee_id, Employee_Name} → Employee_Id is a trivial functional dependency as
3. Employee_Id is a subset of {Employee_Id, Employee_Name}.
4. Also, Employee_Id → Employee_Id and Employee_Name → Employee_Name are trivial dependencies too.

2. Non-trivial functional dependency

- A → B has a non-trivial functional dependency if B is not a subset of A.
- When A intersection B is NULL, then A → B is called as complete non-trivial.

Example:

1. ID → Name,
2. Name → DOB