Assignment to Students

Unit – I

1. What is DBMS? Explain features of DBMS.

- A database management system (DBMS) is a software package designed to define, manipulate, retrieve and manage data in a database. A DBMS generally manipulates the data itself, the data format, field names, record structure and file structure. It also defines rules to validate and manipulate this data.

Main Features of a DBMS

Some of the main features of a database management system software include

· **Low Repetition and Redundancy**

In a database, the chances of data duplication are quite high as several users use one database. A DBMS reduces data repetition and redundancy by creating a single data repository that can be accessed by multiple users, even allowing easy data mapping while performing ETL.

· **Easy Maintenance of Large Databases**

Most organizational data is stored in large databases. A DBMS helps maintain these databases by enforcing user-defined validation and integrity constraints, such as user-based access.

· **Enhanced Security**

When handling large amounts of data, security becomes the top-most concern for all businesses. A database management software doesn't allow full access to anyone except the database administrator or the departmental head. Only they can modify the database and control user access, making the database more secure. All other users are restricted, depending on their access level.

.

Improved File Consistency

By implementing a database management system, organizations can create a standardized way to use files and ensure consistency of data with other systems and applications. Manipulating and streamlining advanced data management systems is essential. The application of an advanced database system allows using the same rules to all the data throughout the organization.

· **Multi-User Environment Support**

A database management software features and supports a multi-user environment, allowing several users to access and work on data concurrently. It also supports several views of the data. A view is a subsection of a database that's distinct and dedicated to specific operators of the system.

As a database is typically accessed by multiple operators simultaneously, these operators may need different database views. For example, operator A may want to print a bank statement, whereas Operator B would want to only check the bank balance. Although both are querying the same database, they will be presented with different views.

In addition to the above-mentioned features, it is also important to look for qualities of a good database system, such as it should represent logical structures of the problem, eliminate redundant data storage, and offer good data access.

2. Differentiate database and DBMS.

Catagories	Database	DBMS
Storage	Besides computers, databases can even be maintained in physical ledgers, books or papers.	In a database management system (DBMS), all the records are maintained only on a computer.
Data Retrieval	The retrieval of information from the databases can be done manually, through queries or by using programs (C, C++, Java etc.).	We can retrieve the data from the database management system through queries written in SQL.
Speed	As databases can be handled manually or via computers, when SQL is not used to retrieve information, it can be very slow.	As a computer system is involved in a database management system, the retrieval of information is very quick.
Access	The databases are not designed for a large number of people who can access data at the same time, rather it is designed for a very small number of people (preferably few people) who access data at different times.	The database management system is designed for a large number of people who can access the data at the same time.
Data Manipulation	In case of the databases, very less information can be modified at a time.	In the database management system (DBMS), a lot of information can be changed at one time (as it can have many users using it at the same time).
Backup and Recovery	The databases do not ensure that the data will be available after failure arises.	The database management system (DBMS) ensures that the data will always be available even after system failures.

3. Explain advantages and disadvantages of DBMS.

Advantages of DBMS

• DBMS offers a variety of techniques to store & retrieve data
• DBMS serves as an efficient handler to balance the needs of multiple applications
using the same data
• Application programmers never exposed to details of data representation and storage.
• A DBMS uses various powerful functions to store and retrieve data efficiently.
• Offers Data Integrity and Security
• The DBMS implies integrity constraints to get a high level of protection against prohibited access to data.
• A DBMS schedules concurrent access to the data in such a manner that only one user can access the same data at a time
• Reduced Application Development Time

Disadvantages of DBMS

• Cost of Hardware and Software of a DBMS is quite high which increases the budget of your organization.
• Most database management systems are often complex systems, so the training for users to use the DBMS is required.
• In some organizations, all data is integrated into a single database which can be damaged because of electric failure or database is corrupted on the storage media
• Use of the same program at a time by many users sometimes lead to the loss of some data.
• DBMS can't perform sophisticated calculations

4. Explain objective and importance of DBMS.

Objective of DBMS

1. Eliminate redundant data.
2. Make access to the data easy for the user.
3. Provide for mass storage of relevant data.
4. Protect the data from physical harm and un-authorized systems.
5. Allow for growth in the data base system.
6. Make the latest modifications to the data base available immediately.
7. Allow for multiple users to be active at one time.
8. Provide prompt response to user requests for data.

Importance of DBMS

• A database management system is important because it manages data efficiently
and allows users to perform multiple tasks with ease.
• A database management system stores, organizes and manages a large amount of
information within a single software application. Use of this system increases efficiency of business operations and reduces overall costs.
• Database management systems are important to businesses and organizations because they provide a highly efficient method for handling multiple types of data.

• Some of the data that are easily managed with this type of system include: employee records, student information, payroll, accounting, project management, inventory and library books. These systems are built to be extremely versatile.
• Without database management, tasks have to be done manually and take more time. Data can be categorized and structured to suit the needs of the company or organization.
• Data is entered into the system and accessed on a routine basis by assigned users.
Each user may have an assigned password to gain access to their part of the system. Multiple users can use the system at the same time in different ways.

5. List and explain application of DBMS.

Application of DBMS

SN	Sector	Use of DBMS
1	Banking	For customer information, account activities, payments, deposits, loans, etc.
2	Airlines	For reservations and schedule information.
3	Universities	For student information, course registrations, colleges and grades.
4	Telecommunication	It helps to keep call records, monthly bills, maintaining balances, etc.
5	Finance	For storing information about stock, sales, and purchases of financial instruments like stocks and bonds.
6	Sales	Use for storing customer, product & sales information.
7	Manufacturing	It is used for the management of supply chain and for tracking production of items. Inventories status in warehouses.
8	HR Management	For information about employees, salaries, payroll, deduction, generation of paychecks, etc.

6. Differentiate traditional flat file system and database.

DBMS vs. Flat File

SN	DBMS	Flat File Management System
1	Multi-user access	It does not support multi-user access
2	Design to fulfill the need for small and large businesses	It is only limited to smaller DBMS system.
3	Remove redundancy and Integrity	Redundancy and Integrity issues
4	Expensive. But in the long term Total Cost of Ownership is cheap	It's cheaper
5	Easy to implement complicated transactions	No support for complicated transactions

Unit –II

7. What do you mean by database design? Explain overall database designing process in detail.

Database Design is a collection of processes that facilitate the designing, development, implementation and maintenance of enterprise data management systems. Properly designed database are easy to maintain, improves data consistency and are cost effective in terms of disk storage space. The database designer decides how the data elements correlate and what data must be stored.

1. Requirements Gathering
Understanding what you want to do, and what you have is essential before you can dive into designing a database. We'll look at the steps in this article.
2. Conceptual Design
We specify the entities, columns, and their relationship. We may use an entity relationship (ER) diagram to visualize the database.
The output is: A conceptual schema (described using a conceptual data model like ER model).
3. Logical Design
It's concerned about data model mapping; mapping a conceptual schema (like ER model) into logical schema to provide a much detail description.
The output is: A logical schema (described using a logical data model specific to the DBMS like relational model).
4. Physical Design
It describes the details of how data is stored. You start by defining (already modeled) tables, how the data is stored, define relationships, … in DBMS.
This requires dealing with the DBMS, and could involve SQL.
The output is: An internal (physical) schema (described using a physical data model).

8. What is abstraction? List and explain different levels of abstraction.

Database systems are made-up of complex data structures. To ease the user interaction with database, the developers hide internal irrelevant details from users. This process of hiding irrelevant details from user is called data abstraction.

We have three levels of abstraction:
Physical level: This is the lowest level of data abstraction. It describes how data is actually stored in database. You can get the complex data structure details at this level.

Logical level: This is the middle level of 3-level data abstraction architecture. It describes what data is stored in database.

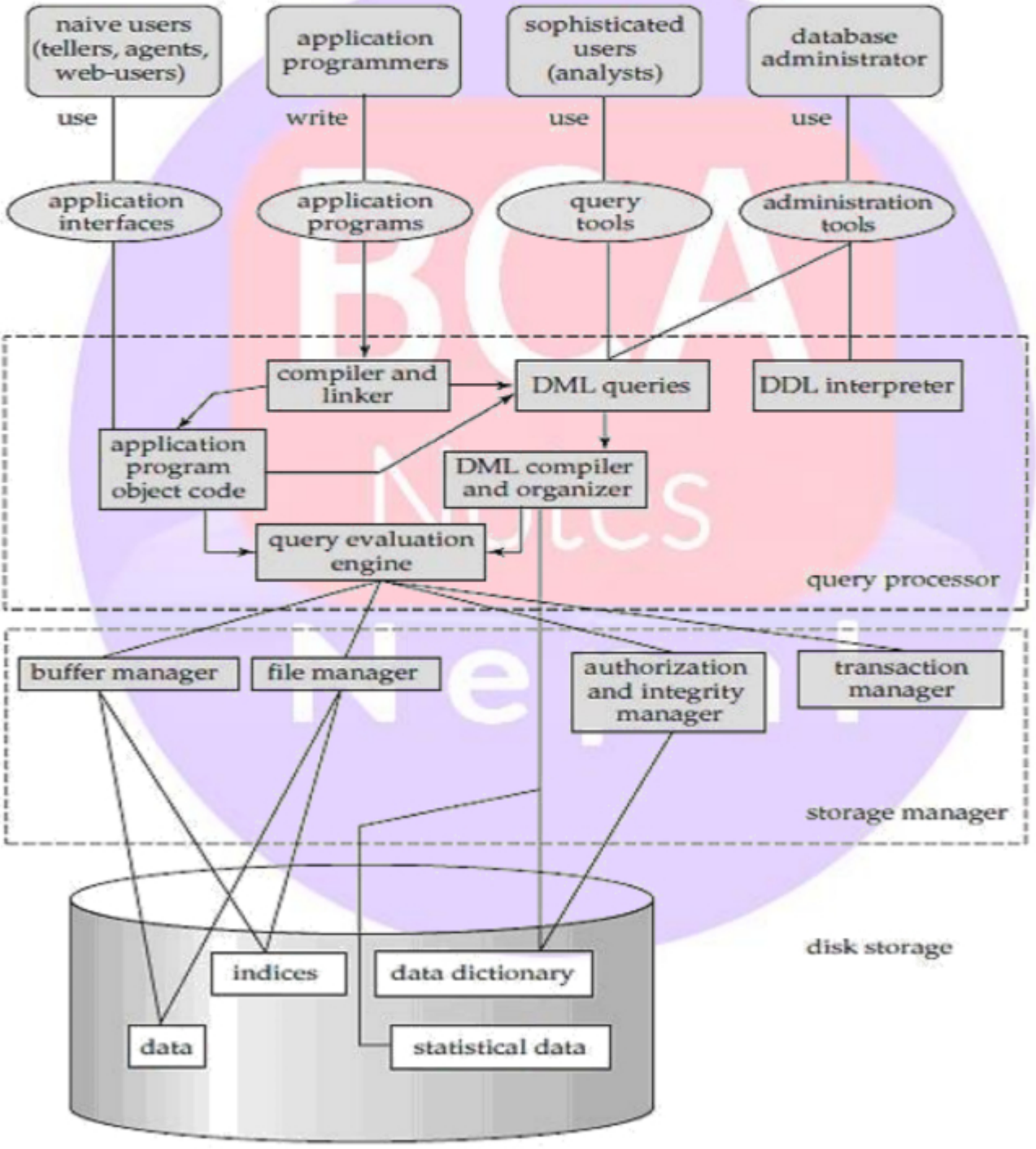
View level: Highest level of data abstraction. This level describes the user interaction with database system.

9. Explain structure of DBMS.

DBMS Structure

A database system is partitioned into modules that deal with each of the responsibilities of the overall system. The functional components of a database system can be broadly divided into the **storage manager** and the **query processor** components.

- ✓ The **storage manager** is important because databases typically require a large amount of storage space.
- ✓ The **query processor** is important because it helps the database system simplify and facilitate access to data.



Query Processor

✓ The query processor components include:

- **DDL interpreter**, which interprets DDL statements and records the definitions in the data dictionary.
- **DML compiler**, which translates DML statements in a query language into an evaluation plan consisting of low-level instructions that the query evaluation engine understands.

✓ A query can usually be translated into any of a number of alternative evaluation plans that all give the same result.

✓ The DML compiler also performs query optimization, that is, it picks the lowest cost evaluation plan from among the alternatives.

Storage Manager

✓ A storage manager is a program module that provides the interface between the low level data stored in the database and the application programs and queries submitted to the system. The storage manager is responsible for the interaction with the file manager.

✓ It is responsible for storing, retrieving, and updating data in the database.

The storage manager components include:

✓ **Authorization and integrity manager**, which tests for the satisfaction of integrity constraints and checks the authority of users to access data.

✓ **Transaction manager**, which ensures that the database remains in a consistent (correct) state despite system failures, and that concurrent transaction executions proceed without conflicting.

✓ **File manager**, which manages the allocation of space on disk storage and the data structures used to represent information stored on disk.

✓ **Buffer manager**, which is responsible for fetching data from disk storage into main memory, and deciding what data to cache in main memory. The buffer manager is a critical part of the database system, since it enables the database to handle data sizes that are much larger than the size of main memory.

Transaction Manager

✓ A transaction is a collection of operations that performs a single logical function in a database application.

✓ Each transaction is a unit of both atomicity and consistency. Thus, we require that transactions do not violate any database-consistency constraints.

✓ That is, if the database was consistent when a transaction started, the database must be consistent when the transaction successfully terminates.

✓ Transaction - manager ensures that the database remains in a consistent (correct) state despite system failures (e.g., power failures and operating system crashes) and transaction failures.

10. What do you mean by Data Independence? Explain physical and logical data independence in detail.

Physical Data Independence
Physical data independence helps you to separate conceptual levels from the internal/physical levels. It allows you to provide a logical description of the database without the need to specify physical structures. Compared to Logical Independence, it is easy to achieve physical data independence.

With Physical independence, you can easily change the physical storage structures or devices with an effect on the conceptual schema. Any change done would be absorbed by the mapping between the conceptual and internal levels. Physical data independence is achieved by the presence of the internal level of the database and then the transformation from the conceptual level of the database to the internal level.

Examples of changes under Physical Data Independence
Due to Physical independence, any of the below change will not affect the conceptual layer.

- Using a new storage device like Hard Drive or Magnetic Tapes
- Modifying the file organization technique in the Database
- Switching to different data structures.
- Changing the access method.
- Modifying indexes.
- Changes to compression techniques or hashing algorithms.
- Change of Location of Database from say C drive to D Drive

Logical Data Independence
Logical Data Independence is the ability to change the conceptual scheme without changing

1 External views
2 External API or programs
Any change made will be absorbed by the mapping between external and conceptual levels.

When compared to Physical Data independence, it is challenging to achieve logical data independence.

Examples of changes under Logical Data Independence
Due to Logical independence, any of the below change will not affect the external layer.

- Add/Modify/Delete a new attribute, entity or relationship is possible without a rewrite of existing application programs
- Merging two records into one
- Breaking an existing record into two or more records

11. What is database architecture? Explain 1 tier, 2 tier and 3 tier database architecture.

A Database Architecture is a representation of DBMS design. It helps to design, develop, implement, and maintain the database management system. A DBMS architecture allows dividing the database system into individual components that can be independently modified, changed, replaced, and altered. It also helps to understand the components of a database.

1 Tier Architecture

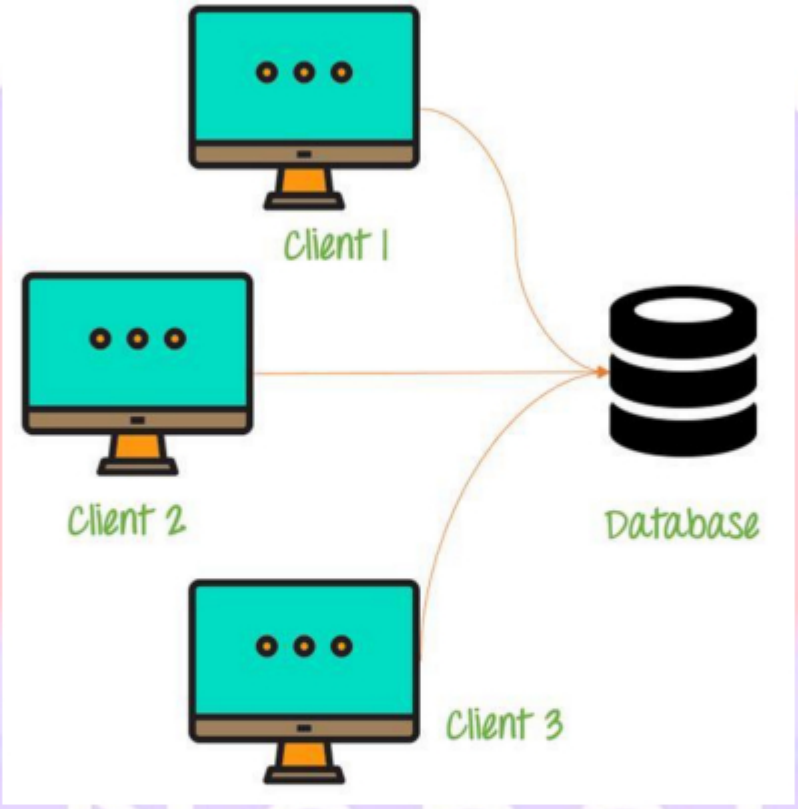
Single Tier Architecture

- ✓ The simplest of Database Architecture are 1 tier where the Client, Server, and Database all reside on the same machine.
- ✓ Anytime you install a DB in your system and access it to practise SQL queries it is 1 tier architecture. But such architecture is rarely used in production.

2 Tier Architecture

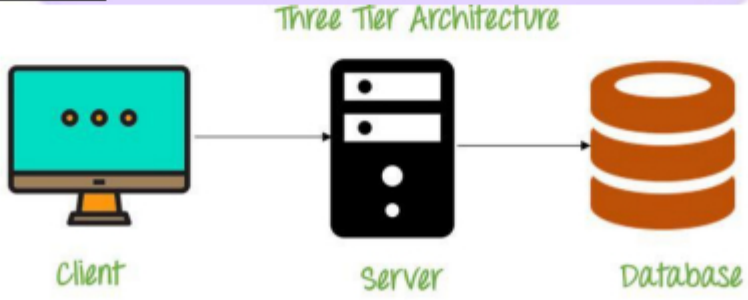
A two-tier architecture is a database architecture where:

1. Presentation layer runs on a client (PC, Mobile, Tablet, etc.)
2. Data is stored on a Server.



- ✓ An application interface which is called ODBC (Open Database Connectivity) an API which allows the client-side program to call the DBMS.
- ✓ Today most of the DBMS offers ODBC drivers for their DBMS. 2 tier architecture provides added security to the DBMS as it is not exposed to the end user directly.

3 Tier Architecture



3-tier schema is an extension of the 2-tier architecture. 3-tier architecture has following layers:

1. Presentation layer (your PC, Tablet, Mobile, etc.)
2. Application layer (server)
3. Database Server

This DBMS architecture contains an Application layer between the user and the DBMS, which is responsible for communicating the user's request to the DBMS system and send the response from the DBMS to the user.

The application layer (business logic layer) also processes functional logic, constraint, and rules before passing data to the user or down to the DBMS Three tier architecture is the most popular DBMS architecture.

The goal of Three-tier architecture is:

- To separate the user applications and physical database
- Proposed to support DBMS characteristics
- Program-data independence
- Support of multiple views of the data

Example of Three-tier Architecture is any large website on the internet.

12. Differentiate DDL and DML with examples.

DDL	DML
1. It stands for Data Definition Language. 2. It is used to create database schema and can be used to define some constraints as well. 3. It basically defines the column (Attributes) of the table. 4. It doesn't have any further classification. 5. Basic command present in DDL are CREATE, DROP, RENAME, ALTER etc. 6. DDL does not use WHERE clause in its statement.	1. It stands for Data Manipulation Language. 2. It is used to add, retrieve or update the data. 3. It add or update the row of the table. These rows are called as tuple. 4. It is further classified into Procedural and Non-Procedural DML. 5. BASIC command present in DML are UPDATE, INSERT, MERGE etc. 6. While DML uses WHERE clause in its statement.

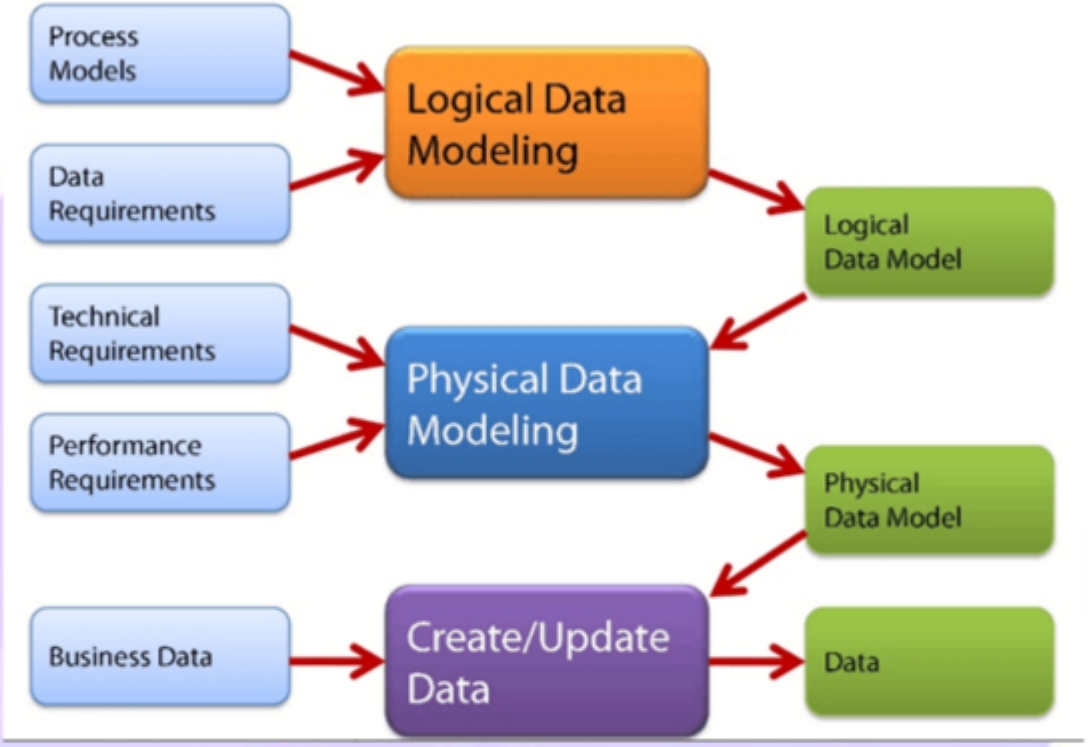
13. What do you mean by data model? Explain different data models in detail. With characteristics and merits and demerits of Data Model?

The Data Model is defined as an abstract model that organizes data description, data semantics, and consistency constraints of data. The data model emphasizes on what data is needed and how it should be organized instead of what operations will be performed on data. Data Model is like an architect's building plan, which helps to build conceptual models and set a relationship between data items.

There are mainly three different types of data models: conceptual data models, logical data models, and physical data models, and each one has a specific

purpose. The data models are used to represent the data and how it is stored in the database and to set the relationship between data items.

1. **Conceptual Data Model:** This Data Model defines **WHAT** the system contains. This model is typically created by Business stakeholders and Data Architects. The purpose is to organize, scope and define business concepts and rules.
2. **Logical Data Model:** Defines **HOW** the system should be implemented regardless of the DBMS. This model is typically created by Data Architects and Business Analysts. The purpose is to developed technical map of rules and data structures.
3. **Physical Data Model**: This Data Model describes **HOW** the system will be implemented using a specific DBMS system. This model is typically created by DBA and developers. The purpose is actual implementation of the database.



Types of Data Model

Conceptual Data Model

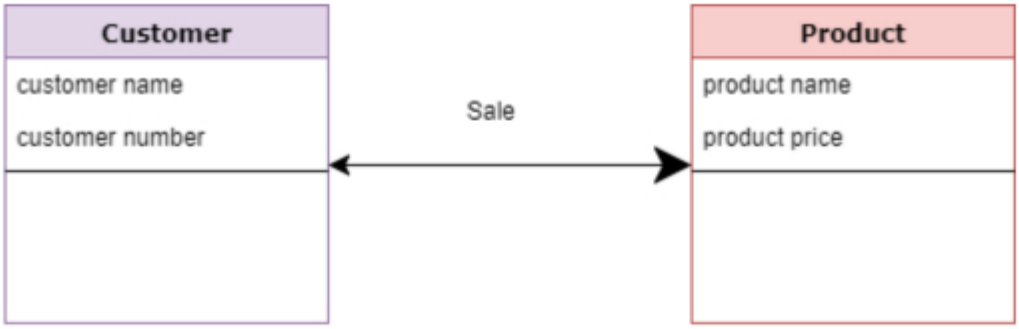
A **Conceptual Data Model** is an organized view of database concepts and their relationships. The purpose of creating a conceptual data model is to establish entities, their attributes, and relationships. In this data modeling level, there is hardly any detail available on the actual database structure. Business stakeholders and data architects typically create a conceptual data model.

The 3 basic tenants of Conceptual Data Model are

- **Entity**: A real-world thing
- **Attribute**: Characteristics or properties of an entity
- **Relationship**: Dependency or association between two entities

Data model example:

- Customer and Product are two entities. Customer number and name are attributes of the Customer entity
- Product name and price are attributes of product entity
- Sale is the relationship between the customer and product



Conceptual Data Model

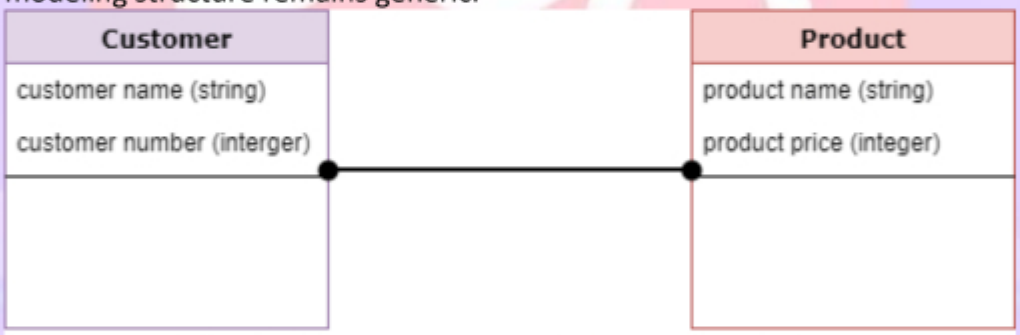
Characteristics of a conceptual data model

- Offers Organisation-wide coverage of the business concepts.
- This type of Data Models are designed and developed for a business audience.
- The conceptual model is developed independently of hardware specifications like data storage capacity, location or software specifications like DBMS vendor and technology. The focus is to represent data as a user will see it in the "real world."

Conceptual data models known as Domain models create a common vocabulary for all stakeholders by establishing basic concepts and scope.

Logical Data Model

The **Logical Data Model** is used to define the structure of data elements and to set relationships between them. The logical data model adds further information to the conceptual data model elements. The advantage of using a Logical data model is to provide a foundation to form the base for the Physical model. However, the modeling structure remains generic.



Logical Data Model

At this Data Modeling level, no primary or secondary key is defined. At this Data modeling level, you need to verify and adjust the connector details that were set earlier for relationships.

Characteristics of a Logical data model

- Describes data needs for a single project but could integrate with other logical data models based on the scope of the project.
- Designed and developed independently from the DBMS.
- Data attributes will have datatypes with exact precisions and length.
- Normalization processes to the model is applied typically till 3NF.

Physical Data Model

A **Physical Data Model** describes a database-specific implementation of the data model. It offers database abstraction and helps generate the schema. This is because of the richness of meta-data offered by a Physical Data Model. The physical data model also helps in visualizing database structure by replicating database column keys, constraints, indexes, triggers, and other RDBMS features.

Physical Data Model

Characteristics of a physical data model:

- The physical data model describes data need for a single project or application though it maybe integrated with other physical data models based on project scope.
- Data Model contains relationships between tables that which addresses cardinality and nullability of the relationships.
- Developed for a specific version of a DBMS, location, data storage or technology to be used in the project.
- Columns should have exact datatypes, lengths assigned and default values.
- Primary and Foreign keys, views, indexes, access profiles, and authorizations, etc. are defined.

(Merits)Advantages of Data model:

- The main goal of a designing data model is to make certain that data objects offered by the functional team are represented accurately.
- The data model should be detailed enough to be used for building the physical database.
- The information in the data model can be used for defining the relationship between tables, primary and foreign keys, and stored procedures.
- Data Model helps business to communicate the within and across organizations.
- Data model helps to documents data mappings in ETL process
- Help to recognize correct sources of data to populate the model

(Demerits)Disadvantages of Data model:

- To develop Data model one should know physical data stored characteristics.
- This is a navigational system produces complex application development, management. Thus, it requires a knowledge of the biographical truth.
- Even smaller change made in structure require modification in the entire application.
- There is no set data manipulation language in DBMS.

14. What is QBE? Explain working and advantages of QBE.
15. Define entity and attributes. Differentiate strong and weak entity set.

Strong Entity:

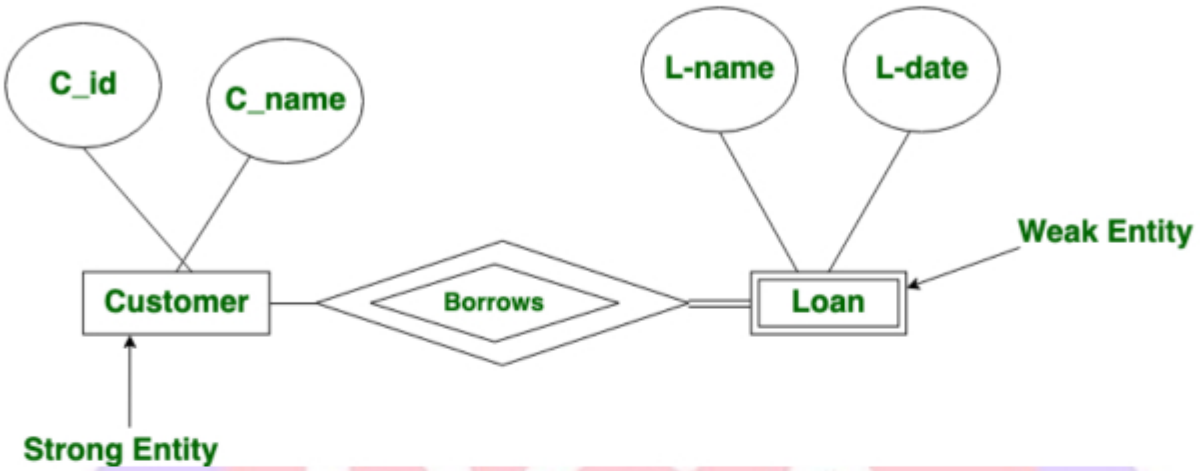
A strong entity is not dependent of any other entity in the schema. A strong entity will always have a primary key. Strong entities are represented by a single rectangle. The relationship of two strong entities is represented by a single diamond.

Various strong entities, when combined together, create a strong entity set.

Weak Entity:

A weak entity is dependent on a strong entity to ensure the its existence. Unlike a strong entity, a weak entity does not have any primary key. It

instead has a partial discriminator key. A weak entity is represented by a double rectangle.
The relation between one strong and one weak entity is represented by a double diamond.



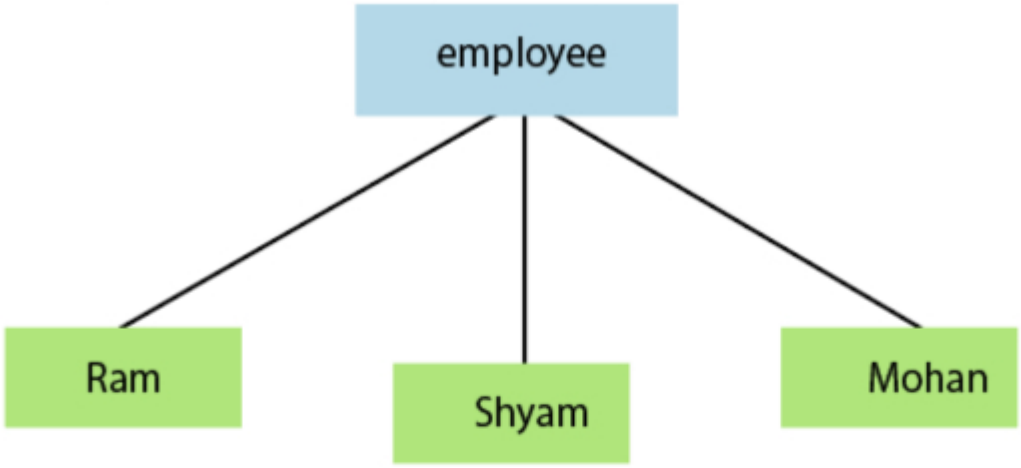
Strong Entity	Weak Entity
1. Strong entity always has primary key. 2. Strong entity is not dependent of any other entity. 3. Strong entity is represented by single rectangle. 4. Two strong entity's relationship is represented by single diamond. 5. Strong entity have either total participation or not.	1. While weak entity has partial discriminator key. 2. Weak entity is depend on strong entity. 3. Weak entity is represented by double rectangle. 4. While the relation between one strong and one weak entity is represented by double diamond. 5. While weak entity always has total participation.

16. Explain Specialization,aggregation and generalization with examples.

DBMS Specialization

It is opposite or inverse of generalization. A specialization is a top-down approach in which an entity of higher-level entity is broken down into two or more entities of lower level. In specialization, a higher-level entity set may not have any lower-level entity set. It is always applied to a single entity and results in the formation of multiple new entities. It increases the size of schema due to the increase in the number of entities.
In the below example, it can be seen that the **employee** is a high-level entity which is divided into three sub-entities (Ram, Shyam, and Mohan). The sub-entities are the

names of the employees (relating to the high-level entity). Therefore, splitting a high-level entity into a low-level entity is called **specialization**.



Specialisation

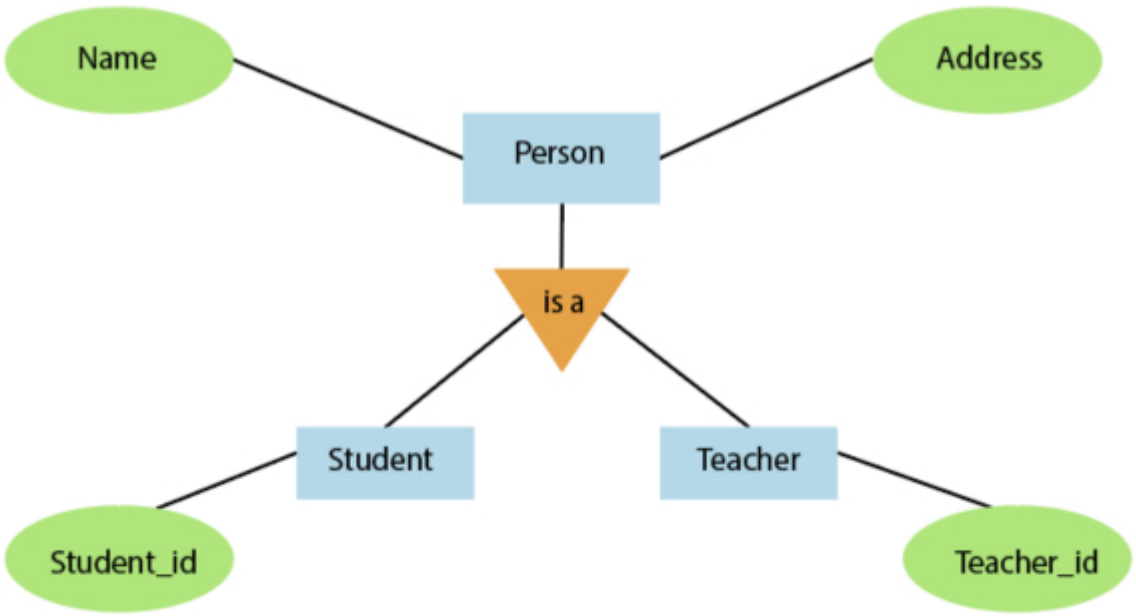
DBMS Generalization

Generalization is a bottom-up approach in which the common attributes of two or more lower-level entities combines to form a new higher-level entity. In generalization, the generalized entity of higher level can also combine with entities of the lower-level to make further higher-level entity.

It is like a superclass and subclass system, but the only difference is that it uses the bottom-up approach. It helps in reducing the schema size. It is always applied to the group of entities and result in the formation of a single entity.

According to the below diagram, there are two entities, namely **teacher** and **student**. The teacher entity contains attributes such as **teacher_id, name, and address** and student entity include **student_id, name, and address**. Both entities can be combined to create a higher-level entity **person**. The address and name are common to both the entities. The teacher entity has its attribute teacher_id, and the student has its attribute student_id. The entities teacher and student are generalized further into the person entity.

A lower-level entity is called a subclass, and the higher-level entity is called a superclass. So, the **person** entity is the superclass of two subclasses **teacher** and **student**.



Generalisation

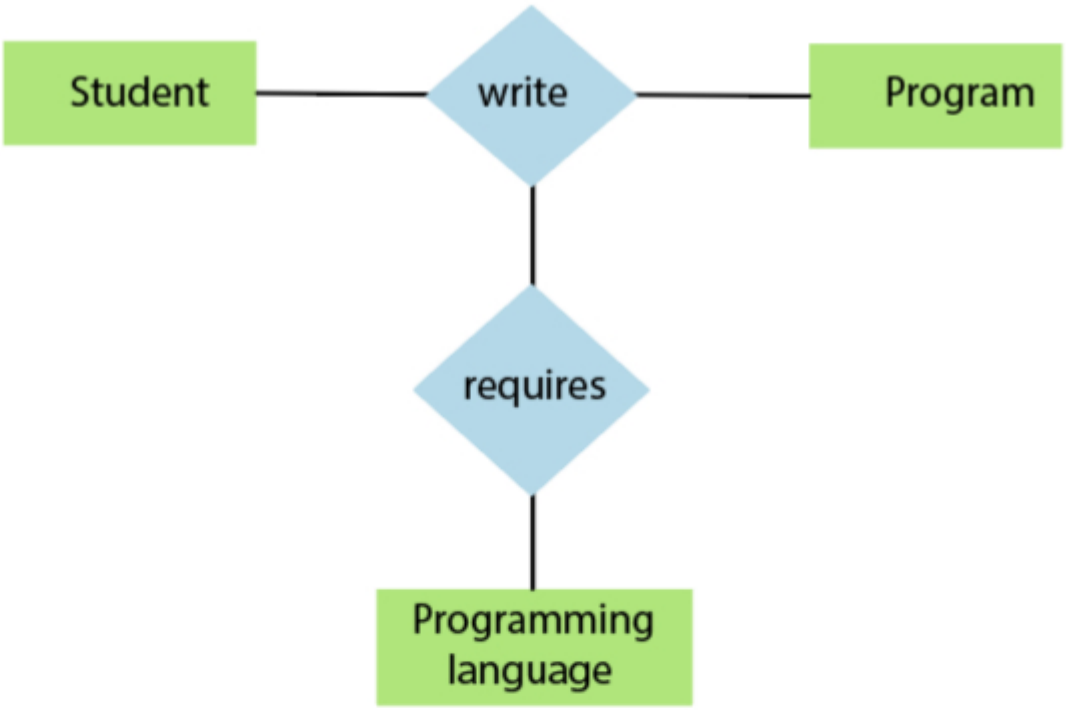
DBMS Aggregation

Through the Entity-Relationship model, we cannot express a relationship set with another relationship set. Thus we use the concept of aggregation to express it. Aggregation is an abstraction in which the relationship between two entities is treated as a higher-level entity. It allows you to indicate that a relationship set participates in another relationship set.

Example:

The **student** and **program** are two entity set and related through a relationship set **write**. Suppose, a student who writes any program must require a programming language installed in their system. So, there will be another relationship set **requires**. You need to connect the relationship set **requires** with an entity set **programming language** and relationship set **write.** But, we can connect only entity set to a relationship set.

One relationship set cannot be connected with another relationship set; for this, we need aggregation. Here, the **write** (relationship set) will treat like a **higher-level entity** and can be associated with a relationship set **requires**. So, aggregation is needed when you express a relationship set with another relationship set.



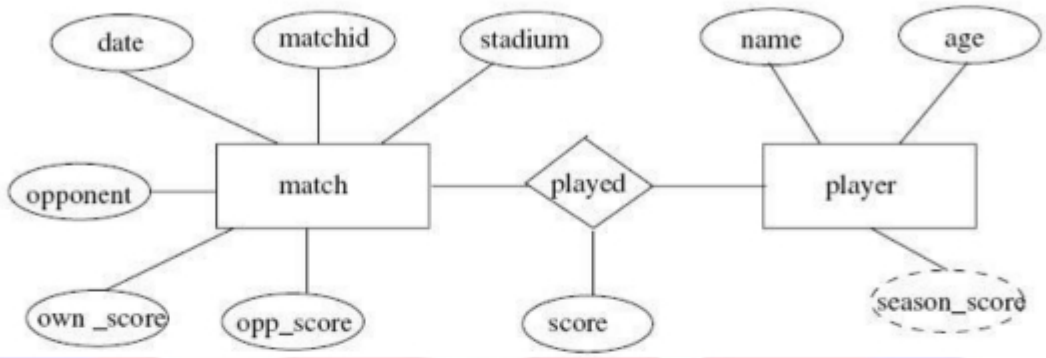
Aggregation

17. Explain the process of converting ER Diagrams to Tables.

18. Construct an E-R diagram for a car insurance company whose customers own one or more cars each. Each car has associated with it zero to any number of recorded accidents. Each insurance policy covers one or more cars, and has one or more premium payments associated with it. Each payment is for a particular period of time, and has an associated due date, and the date when the payment was received.
19. Design a database for an automobile company to provide to its dealers to assist them in maintaining customer records and dealer inventory and to assist sales staff in ordering cars. Each vehicle is identified by a vehicle identification number(VIN). Each individual vehicle is a particular model of a particular brand offered by the company (e.g., the XF is a model of the car brand Jaguar of Tata Motors). Each model can be offered with a variety of options, but an individual car may have only some (or none) of the available options. The database needs to store information about models, brands, and options, as well as information about individual dealers, customers, and cars. Your design should include an E-R diagram, a set of relational schemas, and a list of constraints, including primary-key and foreign-key constraints.
20. Design a database for an airline. The database must keep track of customers and their reservations, flights and their status, seat assignments on individual flights, and the schedule and routing of future flights. Your design should include an E-R

diagram, a set of relational schemas, and a list of constraints, including primary-key and foreign-key constraints.

21. Design an E-R diagram for keeping track of the exploits of your favourite sports team. You should store the matches played, the scores in each match, the players in each match, and individual player statistics for each match. Summary statistics should be modeled as derived attributes.
    - Ans



Unit – III

22. What is RDBMS? Explain the structure of RDBMS with example.
    A database management system (DBMS) that incorporates the relational-data model, normally including a Structured Query Language (SQL) application programming interface. It is a DBMS in which the database is organized and accessed according to the relationships between data items. In a relational database, relationships between data items are expressed by means of tables. Interdependencies among these tables are expressed by data values rather than by pointers. This allows a high degree of data independence.

23. Define database schema. Develop database schema for the following:
    a) Hotel Management System
    b) Library Management System
    c) Bus Ticketing System
24. What do you mean by key? Explain different types of keys with example. And define each keys?

KEYS in DBMS is an attribute or set of attributes which helps you to identify a row(tuple) in a relation(table). They allow you to find the relation between two tables. Keys help you uniquely identify a row in a table by a combination of one or more columns in that table. Key is also helpful for finding unique record or row from the table. Database key is also helpful for finding unique record or row from the table.

Example:

Employee ID	FirstName	LastName
11	Andrew	Johnson
22	Tom	Wood
33	Alex	Hale

In the above-given example, employee ID is a primary key because it uniquely identifies an employee record. In this table, no other employee can have the same employee ID.

Why we need a Key?

Here are some reasons for using sql key in the DBMS system.

- Keys help you to identify any row of data in a table. In a real-world application, a table could contain thousands of records. Moreover, the records could be duplicated. Keys ensure that you can uniquely identify a table record despite these challenges.
- Allows you to establish a relationship between and identify the relation between tables
- Help you to enforce identity and integrity in the relationship.

Types of Keys in Database Management System

There are mainly seven different types of Keys in DBMS and each key has it's different functionality:

- **Super Key -** A super key is a group of single or multiple keys which identifies rows in a table.
- **Primary Key -** is a column or group of columns in a table that uniquely identify every row in that table.
- **Candidate Key -** is a set of attributes that uniquely identify tuples in a table. Candidate Key is a super key with no repeated attributes.
- **Alternate Key -** is a column or group of columns in a table that uniquely identify every row in that table.

- **Foreign Key -** is a column that creates a relationship between two tables. The purpose of Foreign keys is to maintain data integrity and allow navigation between two different instances of an entity.
- **Compound Key -** has two or more attributes that allow you to uniquely recognize a specific record. It is possible that each column may not be unique by itself within the database.
- **Composite Key -** An artificial key which aims to uniquely identify each record is called a surrogate key. These kind of key are unique because they are created when you don't have any natural primary key.
- **Surrogate Key -** An artificial key which aims to uniquely identify each record is called a surrogate key. These kind of key are unique because they are created when you don't have any natural primary key.

What is the Super key?

A superkey is a group of single or multiple keys which identifies rows in a table. A Super key may have additional attributes that are not needed for unique identification.

Example:

EmpSSN	EmpNum	Empname
9812345098	AB05	Shown
9876512345	AB06	Roslyn
199937890	AB07	James

In the above-given example, EmpSSN and EmpNum name are superkeys.

What is a Primary Key?

PRIMARY KEY is a column or group of columns in a table that uniquely identify every row in that table. The Primary Key can't be a duplicate meaning the same value can't appear more than once in the table. A table cannot have more than one primary key.

Rules for defining Primary key:

- Two rows can't have the same primary key value

- It must for every row to have a primary key value.
- The primary key field cannot be null.
- The value in a primary key column can never be modified or updated if any foreign key refers to that primary key.

Example:

In the following example, <code>StudID</code> is a Primary Key.

StudID	Roll No	First Name	LastName	Email
1	11	Tom	Price	abc@gmail.com
2	12	Nick	Wright	xyz@gmail.com
3	13	Dana	Natan	mno@yahoo.com

What is the Alternate key?

ALTERNATE KEYS is a column or group of columns in a table that uniquely identify every row in that table. A table can have multiple choices for a primary key but only one can be set as the primary key. All the keys which are not primary key are called an Alternate Key.

Example:

In this table, StudID, Roll No, Email are qualified to become a primary key. But since StudID is the primary key, Roll No, Email becomes the alternative key.

StudID	Roll No	First Name	LastName	Email
1	11	Tom	Price	abc@gmail.com
2	12	Nick	Wright	xyz@gmail.com
3	13	Dana	Natan	mno@yahoo.com

What is a Candidate Key?

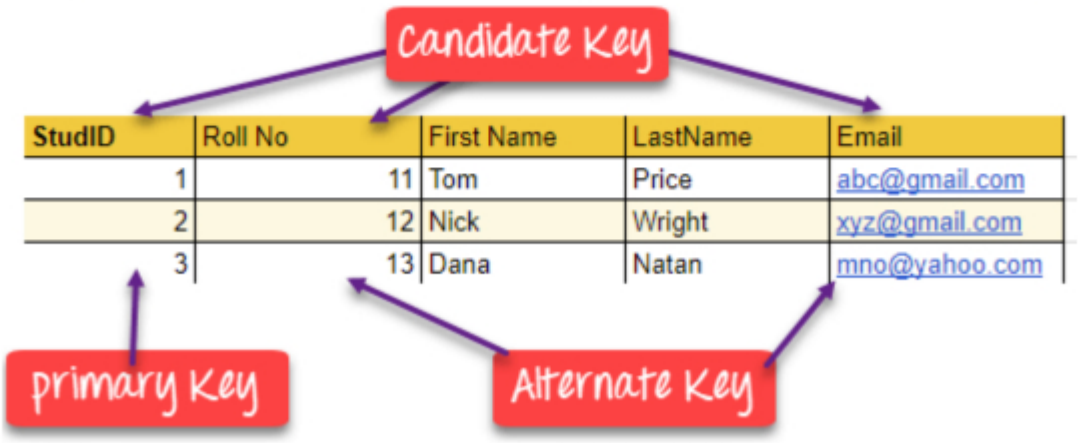
CANDIDATE KEY is a set of attributes that uniquely identify tuples in a table. Candidate Key is a super key with no repeated attributes. The Primary key should be selected from the candidate keys. Every table must have at least a single candidate key. A table can have multiple candidate keys but only a single primary key.

Properties of Candidate key:

- It must contain unique values
- Candidate key may have multiple attributes
- Must not contain null values
- It should contain minimum fields to ensure uniqueness
- Uniquely identify each record in a table

Example: In the given table Stud ID, Roll No, and email are candidate keys which help us to uniquely identify the student record in the table.

StudID	Roll No	First Name	LastName	Email
1	11	Tom	Price	abc@gmail.com
2	12	Nick	Wright	xyz@gmail.com
3	13	Dana	Natan	mno@yahoo.com



What is the Foreign key?

FOREIGN KEY is a column that creates a relationship between two tables. The purpose of Foreign keys is to maintain data integrity and allow navigation between two different instances of an entity. It acts as a cross-reference between two tables as it references the primary key of another table.

Example:

DeptCode	DeptName
001	Science
002	English
005	Computer

Teacher ID	Fname	Lname
B002	David	Warner
B017	Sara	Joseph
B009	Mike	Brunton

In this key in dbms example, we have two table, teach and department in a school. However, there is no way to see which search work in which department.

In this table, adding the foreign key in Deptcode to the Teacher name, we can create a relationship between the two tables.

Teacher ID	DeptCode	Fname	Lname
B002	002	David	Warner
B017	002	Sara	Joseph

B009	001	Mike	Brunton

This concept is also known as Referential Integrity.

What is the Compound key?

COMPOUND KEY has two or more attributes that allow you to uniquely recognize a specific record. It is possible that each column may not be unique by itself within the database. However, when combined with the other column or columns the combination of composite keys become unique. The purpose of the compound key in database is to uniquely identify each record in the table.

Example:

OrderNo	PorductID	Product Name	Quantity
B005	JAP102459	Mouse	5
B005	DKT321573	USB	10
B005	OMG446789	LCD Monitor	20
B004	DKT321573	USB	15
B002	OMG446789	Laser Printer	3

In this example, OrderNo and ProductID can't be a primary key as it does not uniquely identify a record. However, a compound key of Order ID and Product ID could be used as it uniquely identified each record.

What is the Composite key?

COMPOSITE KEY is a combination of two or more columns that uniquely identify rows in a table. The combination of columns guarantees uniqueness, though individually uniqueness is not guaranteed. Hence, they are combined to uniquely identify records in a table.

The difference between compound and the composite key is that any part of the compound key can be a foreign key, but the composite key may or maybe not a part of the foreign key.

What is a Surrogate key?

SURROGATE KEYS is An artificial key which aims to uniquely identify each record is called a surrogate key. This kind of partial key in dbms is unique because it is created when you don't have any natural primary key. They do not lend any meaning to the data in the table. Surrogate key is usually an integer. A surrogate key is a value generated right before the record is inserted into a table.

Fname	Lastname	Start Time	End Time
Anne	Smith	09:00	18:00
Jack	Francis	08:00	17:00
Anna	McLean	11:00	20:00
Shown	Willam	14:00	23:00

Above, given example, shown shift timings of the different employee. In this example, a surrogate key is needed to uniquely identify each employee.

Surrogate keys in sql are allowed when

- No property has the parameter of the primary key.
- In the table when the primary key is too big or complicated.

Difference Between Primary key & Foreign key

Primary key	Foreign Key
• Helps you to uniquely identify a record in the table. • Primary Key never accept null values.	• It is a field in the table that is the primary key of another table. • A foreign key may accept multiple null values.

• Primary key is a clustered index and data in the DBMS table are physically organized in the sequence of the clustered index. • You can have the single Primary key in a table.	• A foreign key cannot automatically create an index, clustered or non-clustered. However, you can manually create an index on the foreign key. • You can have multiple foreign keys in a table.

25. What is Relational Algebra? Explain different operations performed in relational algebra with examples.
RELATIONAL ALGEBRA is a widely used procedural query language. It collects instances of relations as input and gives occurrences of relations as output. It uses various operations to perform this action. SQL Relational algebra query operations are performed recursively on a relation. The output of these operations is a new relation, which might be formed from one or more input relations.

Basic SQL Relational Algebra Operations

Relational Algebra devided in various groups

Unary Relational Operations

- SELECT (symbol: σ)
- PROJECT (symbol: π)
- RENAME (symbol: ρ)

Relational Algebra Operations From Set Theory

- UNION (υ)
- INTERSECTION ( ),
- DIFFERENCE (-)
- CARTESIAN PRODUCT (x)

Binary Relational Operations

- JOIN
- DIVISION

Let's study them in detail with solutions:

SELECT (σ)

The SELECT operation is used for selecting a subset of the tuples according to a given selection condition. Sigma(σ)Symbol denotes it. It is used as an expression to choose tuples which meet the selection condition. Select operator selects tuples that satisfy a given predicate.

$\sigma_p(r)$

σ is the predicate

r stands for relation which is the name of the table

p is prepositional logic

Example 1

$\sigma_{topic = "Database"}(Tutorials)$

Output - Selects tuples from Tutorials where topic = 'Database'.

Example 2

$\sigma_{topic = "Database" \text{ and } author = "guru99"}(Tutorials)$

Output - Selects tuples from Tutorials where the topic is 'Database' and 'author' is guru99.

Example 3

$\sigma_{sales > 50000}(Customers)$

Output - Selects tuples from Customers where sales is greater than 50000

Projection(π)

The projection eliminates all attributes of the input relation but those mentioned in the projection list. The projection method defines a relation that contains a vertical subset of Relation.

This helps to extract the values of specified attributes to eliminates duplicate values. (pi) symbol is used to choose attributes from a relation. This operator helps you to keep specific columns from a relation and discards the other columns.

Example of Projection:

Consider the following table

CustomerID	CustomerName	Status
1	Google	Active
2	Amazon	Active
3	Apple	Inactive
4	Alibaba	Active

Here, the projection of CustomerName and status will give

$\Pi_{\text{CustomerName, Status}}(\text{Customers})$

CustomerName	Status
Google	Active
Amazon	Active
Apple	Inactive
Alibaba	Active

Rename (ρ)

Rename is a unary operation used for renaming attributes of a relation.

ρ (a/b)R will rename the attribute 'b' of relation by 'a'.

Union operation (∪)

UNION is symbolized by ∪ symbol. It includes all tuples that are in tables A or in B. It also eliminates duplicate tuples. So, set A UNION set B would be expressed as:

The result <- A ∪ B

For a union operation to be valid, the following conditions must hold -

- R and S must be the same number of attributes.
- Attribute domains need to be compatible.
- Duplicate tuples should be automatically removed.

Example

Consider the following tables.

Table A		Table B	
column 1	**column 2**	**column 1**	**column 2**
1	1	1	1
1	2	1	3

A ∪ B gives

Table A ∪ B	
column 1	**column 2**
1	1
1	2
1	3

Set Difference (-)

- Symbol denotes it. The result of A - B, is a relation which includes all tuples that are in A but not in B.

- The attribute name of A has to match with the attribute name in B.

- The two-operand relations A and B should be either compatible or Union compatible.
- It should be defined relation consisting of the tuples that are in relation A, but not in B.

Example

A - B

Table A - B

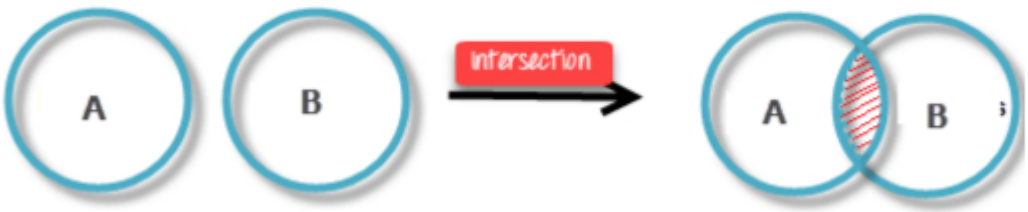
column 1	column 2
1	2

Intersection

An intersection is defined by the symbol ∩

A ∩ B

Defines a relation consisting of a set of all tuple that are in both A and B. However, A and B must be union-compatible.



Visual Definition of Intersection

Example:

A ∩ B

Table A ∩ B

column 1	column 2
1	1

Cartesian Product(X) in DBMS

Cartesian Product in DBMS is an operation used to merge columns from two relations. Generally, a cartesian product is never a meaningful operation when it performs alone. However, it becomes meaningful when it is followed by other operations. It is also called Cross Product or Cross Join.

Example – Cartesian product

$\sigma_{\text{column 2}} = \text{'1'} (A \times B)$

Output – The above example shows all rows from relation A and B whose column 2 has value 1

σ column 2 = '1' (A X B)	
column 1	**column 2**
1	1
1	1

Join Operations

Join operation is essentially a cartesian product followed by a selection criterion.

Join operation denoted by ⋈.

JOIN operation also allows joining variously related tuples from different relations.

Types of JOIN:

Various forms of join operation are:

Inner Joins:

- Theta join
- EQUI join
- Natural join

Outer join:

- Left Outer Join
- Right Outer Join
- Full Outer Join

Inner Join:

In an inner join, only those tuples that satisfy the matching criteria are included, while the rest are excluded. Let's study various types of Inner Joins:

Theta Join:

The general case of JOIN operation is called a Theta join. It is denoted by symbol **θ**

Example

$$A \bowtie_\theta B$$

Theta join can use any conditions in the selection criteria.

For example:

$$A \bowtie_{A.column\ 2\ >\ B.column\ 2} (B)$$

A ⋈ A.column 2 > B.column 2 (B)

column 1	column 2
1	2

EQUI join:

When a theta join uses only equivalence condition, it becomes a equi join.

For example:

$$A \bowtie_{A.column\ 2\ =\ B.column\ 2} (B)$$

A ⋈ A.column 2 = B.column 2 (B)

column 1	column 2
1	1

EQUI join is the most difficult operations to implement efficiently using SQL in an RDBMS and one reason why RDBMS have essential performance problems.

NATURAL JOIN (⋈)

Natural join can only be performed if there is a common attribute (column) between the relations. The name and type of the attribute must be same.

Example

Consider the following two tables

C	
Num	**Square**
2	4
3	9

D	
Num	**Cube**
2	8
3	27

C ⋈ D

C ⋈ D		

Num	Square	Cube
2	4	4
3	9	27

OUTER JOIN

In an outer join, along with tuples that satisfy the matching criteria, we also include some or all tuples that do not match the criteria.

Left Outer Join(A ⟕ B)

In the left outer join, operation allows keeping all tuple in the left relation. However, if there is no matching tuple is found in right relation, then the attributes of right relation in the join result are filled with null values.

Consider the following 2 Tables

A	
Num	**Square**
2	4
3	9
4	16

B	
Num	**Cube**
2	8
3	18
5	75

A ⋈ B

A ⋈ B		
Num	**Square**	**Cube**
2	4	4
3	9	9
4	16	-

Right Outer Join: (A ⟖ B)

In the right outer join, operation allows keeping all tuple in the right relation. However, if there is no matching tuple is found in the left relation, then the attributes of the left relation in the join result are filled with null values.

A ⟖ B

A ⋈ B		
Num	**Cube**	**Square**
2	8	4
3	18	9
5	75	-

Full Outer Join: (A ⟗ B)

In a full outer join, all tuples from both relations are included in the result, irrespective of the matching condition.

A ⟗ B

A ⋈ B		
Num	**Cube**	**Square**
2	4	8
3	9	18
4	16	-
5	-	75

26. Write relational algebra for the following relations:

Employee

Emp_Id	Name	Address	Salary	Dept_Id

Department

Dept_Id	Dept_Name	Floor

i. Select name and address of employees whose salary is between 10000 and 20000.
ii. Select employee id, employee name and department name of employees working in first floor.
iii. Select all records of department which are in second floor.
iv. Select name, address and department name of employees which are from Birtamode.
v. Select employee id and name of employees having salary more than 10000 and from Kathmandu.

Unit – IV

27. What is Normalization? Explain advantages and disadvantages of Normalization. ADVANTAGES OF NORMALIZATION and Types of normal form?

Normalization

- Normalization is the process of organizing the data in the database.
- Normalization is used to minimize the redundancy from a relation or set of relations. It is also used to eliminate the undesirable characteristics like Insertion, Update and Deletion Anomalies.
- Normalization divides the larger table into the smaller table and links them using relationship.
- The normal form is used to reduce redundancy from the database table.

Here we can see why normalization is an attractive prospect in RDBMS concepts.

1) A smaller database can be maintained as normalization eliminates the duplicate data. Overall size of the database is reduced as a result.

2) Better performance is ensured which can be linked to the above point. As databases become lesser in size, the passes through the data becomes faster and shorter thereby improving response time and speed.

3) Narrower tables are possible as normalized tables will be fine-tuned and will have lesser columns which allows for more data records per page.

4) Fewer indexes per table ensures faster maintenance tasks (index rebuilds).

5) Also realizes the option of joining only the tables that are needed.

DISADVANTAGES OF NORMALIZATION

1) More tables to join as by spreading out data into more tables, the need to join table's increases and the task becomes more tedious. The database becomes harder to realize as well.

2) Tables will contain codes rather than real data as the repeated data will be stored as lines of codes rather than the true data. Therefore, there is always a need to go to the lookup table.

3) Data model becomes extremely difficult to query against as the data model is optimized for applications, not for ad hoc querying. (Ad hoc query is a query that cannot be determined before the issuance of the query. It consists of an SQL that is constructed dynamically and is usually constructed by desktop friendly query tools.). Hence it is hard to model the database without knowing what the customer desires.

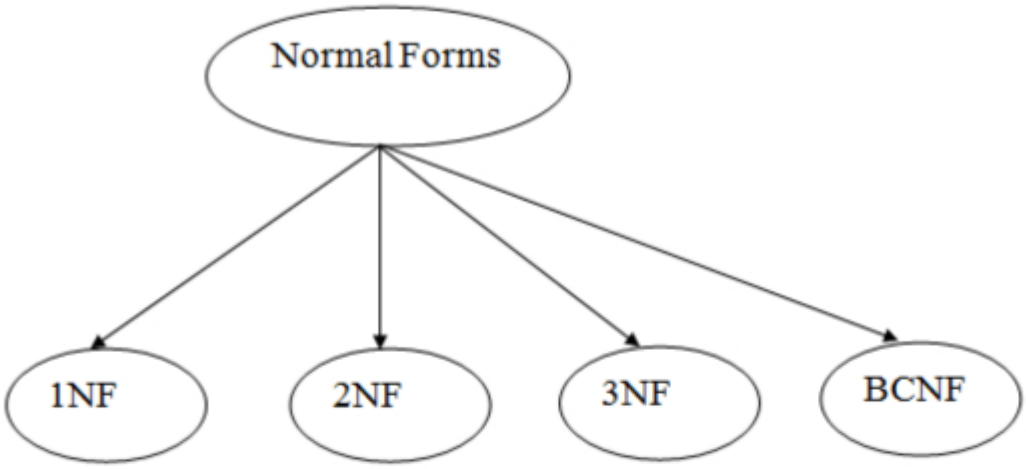
4) As the normal form type progresses, the performance becomes slower and slower.

5) Proper knowledge is required on the various normal forms to execute the normalization process efficiently. Careless use may lead to terrible design filled with major anomalies and data inconsistency.

Types of Normal Forms

There are the four types of normal forms:

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



28. What are differernt anomalies? Explain each of them with suitable examples.
29. What is functional dependency? Explain different types of functional dependencies with example.

What is Functional Dependency?

Functional Dependency (FD) is a constraint that determines the relation of one attribute to another attribute in a Database Management System (DBMS). Functional Dependency helps to maintain the quality of data in the database. It plays a vital role to find the difference between good and bad database design.

A functional dependency is denoted by an arrow "→". The functional dependency of X on Y is represented by X → Y. Let's understand Functional Dependency in DBMS with example.

Example:

Employee number	Employee Name	Salary	City
1	Dana	50000	San Francisco
2	Francis	38000	London
3	Andrew	25000	Tokyo

In this example, if we know the value of Employee number, we can obtain Employee Name, city, salary, etc. By this, we can say that the city, Employee Name, and salary are functionally depended on Employee number.

Rules of Functional Dependencies

Below are the Three most important rules for Functional Dependency in Database:

- Reflexive rule –. If X is a set of attributes and Y is_subset_of X, then X holds a value of Y.
- Augmentation rule: When x -> y holds, and c is attribute set, then ac -> bc also holds. That is adding attributes which do not change the basic dependencies.
- Transitivity rule: This rule is very much similar to the transitive rule in algebra if x -> y holds and y -> z holds, then x -> z also holds. X -> y is called as functionally that determines y.

Types of Functional Dependencies in DBMS

There are mainly four types of Functional Dependency in DBMS. Following are the types of Functional Dependencies in DBMS:

- **Multivalued Dependency**
- **Trivial Functional Dependency**
- **Non-Trivial Functional Dependency**
- **Transitive Dependency**

Multivalued Dependency in DBMS

Multivalued dependency occurs in the situation where there are multiple independent multivalued attributes in a single table. A multivalued dependency is a complete constraint between two sets of attributes in a relation. It requires that certain tuples be present in a relation. Consider the following Multivalued Dependency Example to understand.

Example:

Car_model	Maf_year	Color
H001	2017	Metallic

H001	2017	Green
H005	2018	Metallic
H005	2018	Blue
H010	2015	Metallic
H033	2012	Gray

In this example, maf_year and color are independent of each other but dependent on car_model. In this example, these two columns are said to be multivalue dependent on car_model.

This dependence can be represented like this:

car_model -> maf_year

car_model-> colour

Trivial Functional Dependency in DBMS

The Trivial dependency is a set of attributes which are called a trivial if the set of attributes are included in that attribute.

So, X -> Y is a trivial functional dependency if Y is a subset of X. Let's understand with a Trivial Functional Dependency Example.

For example:

Emp_id	**Emp_name**
AS555	Harry
AS811	George
AS999	Kevin

Consider this table of with two columns Emp_id and Emp_name.

{Emp_id, Emp_name} -> Emp_id is a trivial functional dependency as Emp_id is a subset of {Emp_id,Emp_name}.

Non Trivial Functional Dependency in DBMS

Functional dependency which also known as a nontrivial dependency occurs when A->B holds true where B is not a subset of A. In a relationship, if attribute B is not a subset of attribute A, then it is considered as a non-trivial dependency.

Company	CEO	Age
Microsoft	Satya Nadella	51
Google	Sundar Pichai	46
Apple	Tim Cook	57

Example:

(Company} -> {CEO} (if we know the Company, we knows the CEO name)

But CEO is not a subset of Company, and hence it's non-trivial functional dependency.

Transitive Dependency in DBMS

A Transitive Dependency is a type of functional dependency which happens when t is indirectly formed by two functional dependencies. Let's understand with the following Transitive Dependency Example.

Example:

Company	CEO	Age
Microsoft	Satya Nadella	51
Google	Sundar Pichai	46

{Company} -> {CEO} (if we know the compay, we know its CEO's name)

{CEO } -> {Age} If we know the CEO, we know the Age

Therefore according to the rule of rule of transitive dependency:

{ Company} -> {Age} should hold, that makes sense because if we know the company name, we can know his age.

Note: You need to remember that transitive dependency can only occur in a relation of three or more attributes.

What is Normalization?

Normalization is a method of organizing the data in the database which helps you to avoid data redundancy, insertion, update & deletion anomaly. It is a process of analyzing the relation schemas based on their different functional dependencies and primary key.

Normalization is inherent to relational database theory. It may have the effect of duplicating the same data within the database which may result in the creation of additional tables.

Advantages of Functional Dependency

- Functional Dependency avoids data redundancy. Therefore same data do not repeat at multiple locations in that database
- It helps you to maintain the quality of data in the database
- It helps you to defined meanings and constraints of databases
- It helps you to identify bad designs
- It helps you to find the facts regarding the database design

30. Explain 1NF, 2NF, 3NF, BCNF and 4NF with suitable example.
What is Normalization?
Normalization is a database design technique that reduces data redundancy and eliminates undesirable characteristics like Insertion, Update and Deletion Anomalies. Normalization rules divides larger tables into smaller tables and links them using relationships. The purpose of Normalization in SQL is to eliminate redundant (repetitive) data and ensure data is stored logically.
The inventor of the relational model Edgar Codd proposed the theory of normalization of data with the introduction of the First Normal Form, and he

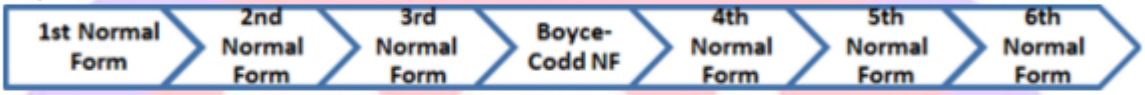
continued to extend theory with Second and Third Normal Form. Later he joined Raymond F. Boyce to develop the theory of Boyce-Codd Normal Form.

Database Normal Forms

Here is a list of Normal Forms

- 1NF (First Normal Form)
- 2NF (Second Normal Form)
- 3NF (Third Normal Form)
- BCNF (Boyce-Codd Normal Form)
- 4NF (Fourth Normal Form)
- 5NF (Fifth Normal Form)
- 6NF (Sixth Normal Form)

The Theory of Data Normalization in SQL server is still being developed further. For example, there are discussions even on 6th Normal Form. **However, in most practical applications, normalization achieves its best in 3rd Normal Form**. The evolution of SQL Normalization theories is illustrated below-



Database Normal Forms

Database Normalization With Examples

Database **Normalization Example** can be easily understood with the help of a case study. Assume, a video library maintains a database of movies rented out. Without any normalization in database, all information is stored in one table as shown below. Let's understand Normalization in database with tables example:

Here you see **Movies Rented column has multiple values.** Now let's move into 1st Normal Forms:

1NF (First Normal Form) Rules

- Each table cell should contain a single value.
- Each record needs to be unique.

The above table in 1NF-

1NF Example

Before we proceed let's understand a few things --

What is a KEY?

A KEY is a value used to identify a record in a table uniquely. A KEY could be a single column or combination of multiple columns

Note: Columns in a table that are NOT used to identify a record uniquely are called non-key columns.

What is a Primary Key?

A primary is a single column value used to identify a database record uniquely.

It has following attributes

- A primary key cannot be NULL

- A primary key value must be unique
- The primary key values should rarely be changed
- The primary key must be given a value when a new record is inserted.

What is Composite Key?

A composite key is a primary key composed of multiple columns used to identify a record uniquely

In our database, we have two people with the same name Robert Phil, but they live in different places.

Composite Key

Robert Phil	3rd Street 34	Daddy's Little Girls	Mr.
Robert Phil	5th Avenue	Clash of the Titans	Mr.

Names are common. Hence you need name as well Address to uniquely identify a record.

Hence, we require both Full Name and Address to identify a record uniquely. That is a composite key.

Let's move into second normal form 2NF

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.

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

Database - Foreign Key

In Table 2, Membership_ID is the Foreign Key

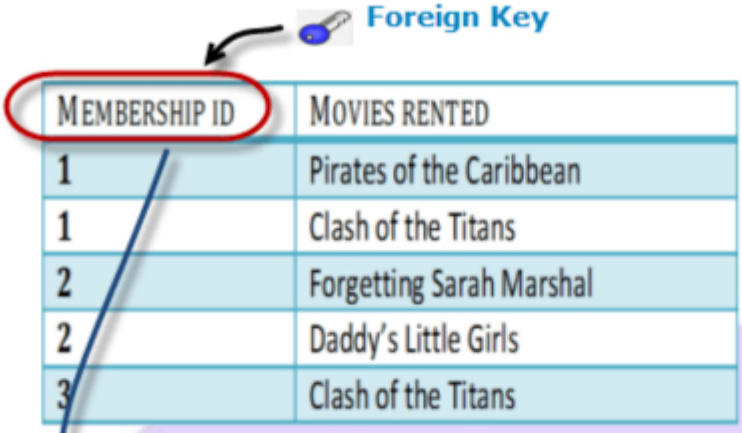
MEMBERSHIP ID	MOVIES RENTED
1	Pirates of the Caribbean
1	Clash of the Titans
2	Forgetting Sarah Marshal
2	Daddy's Little Girls
3	Clash of the Titans

Foreign Key

Foreign Key references the primary key of another Table! It helps connect your Tables

- A foreign key can have a different name from its primary key
- It ensures rows in one table have corresponding rows in another

- Unlike the Primary key, they do not have to be unique. Most often they aren't
- Foreign keys can be null even though primary keys can not



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

Foreign Key references Primary Key
Foreign Key can only have values present in primary key
It could have a name other than that of Primary Key

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

Why do you need a foreign key?
Suppose, a novice inserts a record in Table B such as

Insert a record in Table 2 where Member ID =101

MEMBERSHIP ID	MOVIES RENTED
101	Mission Impossible

But Membership ID 101 is not present in Table 1

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

Database will throw an ERROR. This helps in referential integrity

You will only be able to insert values into your foreign key that exist in the unique key in the parent table. This helps in referential integrity.
The above problem can be overcome by declaring membership id from Table2 as foreign key of membership id from Table1
Now, if somebody tries to insert a value in the membership id field that does not exist in the parent table, an error will be shown!
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

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

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

Let's move into 3NF

3NF (Third Normal Form) Rules

- Rule 1- Be in 2NF
- Rule 2- Has no transitive functional dependencies

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

3NF Example

Below is a 3NF example in SQL database:

We have again divided our tables and created a new table which stores Salutations.

There are no transitive functional dependencies, and hence our table is in 3NF

In Table 3 Salutation ID is primary key, and in Table 1 Salutation ID is foreign to primary key in Table 3

Now our little example is at a level that cannot further be decomposed to attain higher normal forms of normalization. In fact, it is already in higher normalization forms. Separate efforts for moving into next levels of normalizing data are normally needed in complex databases. However, we will be discussing next levels of normalizations in brief in the following.

BCNF (Boyce-Codd Normal Form)

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.**

4NF (Fourth Normal Form) Rules

If no database table instance contains two or more, independent and multivalued data describing the relevant entity, then it is in 4th Normal Form.

31. Convert the following:

1. **Convert the following Student table to 2NF.**

sid	name	course_id	course	phone_no
1	Rohit	101	BCA	9833888831
2	Ravi	102	BBA	3555939392
3	Shyam	103	MCA	4994994993
4	Hari	104	MBA	4884484844
5	Ram	105	BBS	4908590495

3. **Normalize the following Employee table upto 3NF.**

emp_id	name	address	contact	branch_code	branch
1	Rohit	Btm	9866666,985555	101	Dhulabari
2	Ravi	Btm	983333333	102	Surunga
3	Shyam	Ktm	98777777	102	Surunga
4	Hari	Brt	98334444,9874774	101	Dhulabari
5	Ram	Ktm	98123444	103	Bhadrapur

2. **Convert the following Staff Table to 3NF.**

staff_id	name	address	department	department _contact
1	Rohit	Btm	Accounting	9866666666
2	Ravi	Btm	HR	9877777777
3	Shyam	Ktm	Accounting	9866666666
4	Hari	Brt	HR	9877777777
5	Ram	Ktm	Marketing	9833333333

Unit V, VI & VII

32. Define SQL. Explain importance of SQL with examples.
33. What are aggregate functions in SQL? Explain each of them with suitable examples.

Aggregate functions in SQL

- `COUNT` counts how many rows are in a particular column.
- `SUM` adds together all the values in a particular column.
- `MIN` and `MAX` return the lowest and highest values in a particular column, respectively.
- `AVG` calculates the average of a group of selected values.

Why use aggregate functions.

From a business perspective, different organization levels have different information requirements. Top levels managers are usually interested in knowing whole figures and not necessary the individual details.

>Aggregate functions allow us to easily produce summarized data from our database.

For instance, from our myflix database , management may require following reports

- Least rented movies.
- Most rented movies.
- Average number that each movie is rented out in a month.

We easily produce above reports using aggregate functions.

Let's look into aggregate functions in detail.

COUNT Function

The COUNT function returns the total number of values in the specified field. It works on both numeric and non-numeric data types. **All aggregate functions by default exclude nulls values before working on the data.**

COUNT (*) is a special implementation of the COUNT function that returns the count of all the rows in a specified table. COUNT (*) also considers Nulls and duplicates.

The table shown below shows data in movierentals table

reference_ number	transaction_ date	return_date	membership_ number	movie_id	movie_ returned
11	20-06-2012	NULL	1	1	0
12	22-06-2012	25-06-2012	1	2	0
13	22-06-2012	25-06-2012	3	2	0
14	21-06-2012	24-06-2012	2	2	0
15	23-06-2012	NULL	3	3	0

Let's suppose that we want to get the number of times that the movie with id 2 has been rented out

```
SELECT COUNT(`movie_id`) FROM `movierentals` WHERE `movie_id` = 2;
```

Executing the above query in MySQL workbench against myflixdb gives us the following results.

COUNT('movie_id')
3

DISTINCT Keyword

The DISTINCT keyword that allows us to omit duplicates from our results. This is achieved by grouping similar values together .

To appreciate the concept of Distinct, lets execute a simple query

```
SELECT `movie_id` FROM `movierentals`;
```

movie_id
1
2
2
2
3

Now let's execute the same query with the distinct keyword -

```
SELECT DISTINCT `movie_id` FROM `movierentals`;
```

As shown below , distinct omits duplicate records from the results.

movie_id
1
2
3

MIN function

The MIN function **returns the smallest value in the specified table field**.

As an example, let's suppose we want to know the year in which the oldest movie in our library was released, we can use MySQL's MIN function to get the desired information.

The following query helps us achieve that

```
SELECT MIN(`year_released`) FROM `movies`;
```

Executing the above query in MySQL workbench against myflixdb gives us the following results.

MIN('year_released')
2005

MAX function

Just as the name suggests, the MAX function is the opposite of the MIN function. It **returns the largest value from the specified table field**.

Let's assume we want to get the year that the latest movie in our database was released. We can easily use the MAX function to achieve that.

The following example returns the latest movie year released.

```
SELECT MAX(`year_released`) FROM `movies`;
```

Executing the above query in MySQL workbench using myflixdb gives us the following results.

MAX('year_released')
2012

SUM function

Suppose we want a report that gives total amount of payments made so far. We can use the MySQL **SUM** function which **returns the sum of all the values in the specified column. SUM works on numeric fields only. Null values are excluded from the result returned.**

The following table shows the data in payments table-

payment_id	membership_number	payment_date	description	amount_paid	external_reference_number
1	1	23-07-2012	Movie rental payment	2500	11
2	1	25-07-2012	Movie rental payment	2000	12
3	3	30-07-2012	Movie rental payment	6000	NULL

The query shown below gets the all payments made and sums them up to return a single result.

```
SELECT SUM(`amount_paid`) FROM `payments`;
```

Executing the above query in MySQL workbench against the myflixdb gives the following results.

SUM('amount_paid')
10500

AVG function

MySQL AVG function **returns the average of the values in a specified column**. Just like the SUM function, it **works only on numeric data types**.

Suppose we want to find the average amount paid. We can use the following query -

```
SELECT AVG(`amount_paid`) FROM `payments`;
```

Executing the above query in MySQL workbench, gives us the following results.

AVG('amount_paid')
3500

34. What do you mean by constraint? Explain different types of constraints with examples.

Constraints enforce limits to the data or type of data that can be inserted/updated/deleted from a table. The whole purpose of constraints is to maintain the **data integrity** during an update/delete/insert into a table. In this tutorial we will learn several types of constraints that can be created in RDBMS.

Types of constraints

- NOT NULL
- UNIQUE
- DEFAULT
- CHECK
- Key Constraints – PRIMARY KEY, FOREIGN KEY
- Domain constraints
- Mapping constraints

NOT NULL:

NOT NULL constraint makes sure that a column does not hold NULL value. When we don't provide value for a particular column while inserting a record into a table, it takes NULL value by default. By specifying NULL constraint, we can be sure that a particular column(s) cannot have NULL values.

Example:

```
CREATE TABLE STUDENT(
ROLL_NO INT NOT NULL,
STU_NAME VARCHAR (35) NOT NULL,
STU_AGE INT NOT NULL,
STU_ADDRESS VARCHAR (235),
PRIMARY KEY (ROLL_NO)
);
```

UNIQUE:

UNIQUE Constraint enforces a column or set of columns to have unique values. If a column has a unique constraint, it means that particular column cannot have duplicate values in a table.

```
CREATE TABLE STUDENT(
ROLL_NO INT NOT NULL,
STU_NAME VARCHAR (35) NOT NULL UNIQUE,
STU_AGE INT NOT NULL,
STU_ADDRESS VARCHAR (35) UNIQUE,
PRIMARY KEY (ROLL_NO)
);
```

DEFAULT:

The DEFAULT constraint provides a default value to a column when there is no value provided while inserting a record into a table.

```
CREATE TABLE STUDENT(
ROLL_NO   INT  NOT NULL,
STU_NAME VARCHAR (35) NOT NULL,
STU_AGE INT NOT NULL,
EXAM_FEE INT  DEFAULT 10000,
STU_ADDRESS VARCHAR (35) ,
PRIMARY KEY (ROLL_NO)
);
```

CHECK:

This constraint is used for specifying range of values for a particular column of a table. When this constraint is being set on a column, it ensures that the specified column must have the value falling in the specified range.

```
CREATE TABLE STUDENT(
ROLL_NO   INT  NOT NULL CHECK(ROLL_NO >1000) ,
STU_NAME VARCHAR (35)  NOT NULL,
STU_AGE INT  NOT NULL,
```

```
EXAM_FEE INT DEFAULT 10000,
STU_ADDRESS VARCHAR (35) ,
PRIMARY KEY (ROLL_NO)
);
```

In the above example we have set the check constraint on ROLL_NO column of STUDENT table. Now, the ROLL_NO field must have the value greater than 1000.

Key constraints:

PRIMARY KEY:

Primary key uniquely identifies each record in a table. It must have unique values and cannot contain nulls. In the below example the ROLL_NO field is marked as primary key, that means the ROLL_NO field cannot have duplicate and null values.

```
CREATE TABLE STUDENT(
ROLL_NO   INT  NOT NULL,
STU_NAME VARCHAR (35)  NOT NULL UNIQUE,
STU_AGE INT NOT NULL,
STU_ADDRESS VARCHAR (35) UNIQUE,
PRIMARY KEY (ROLL_NO)
);
```

FOREIGN KEY:

Foreign keys are the columns of a table that points to the primary key of another table. They act as a cross-reference between tables.

Domain constraints:

Each table has certain set of columns and each column allows a same type of data, based on its data type. The column does not accept values of any other data type.
Domain_constraints are **user defined data type** and we can define them like this:

Domain Constraint = data type + Constraints (NOT NULL / UNIQUE / PRIMARY KEY / FOREIGN KEY / CHECK / DEFAULT)

Mapping Cardinality:
One to One: An entity of entity-set A can be associated with at most one entity of entity-set B and an entity in entity-set B can be associated with at most one entity of entity-set A.

One to Many: An entity of entity-set A can be associated with any number of entities of entity-set B and an entity in entity-set B can be associated with at most one entity of entity-set A.

Many to One: An entity of entity-set A can be associated with at most one entity of entity-set B and an entity in entity-set B can be associated with any number of entities of entity-set A.

Many to Many: An entity of entity-set A can be associated with any number of entities of entity-set B and an entity in entity-set B can be associated with any number of entities of entity-set A.

We can have these constraints in place while creating tables in database.

Example:

```
CREATE TABLE Customer (
customer_id int PRIMARY KEY NOT NULL,
first_name varchar(20),
last_name varchar(20)
);

CREATE TABLE Order (
order_id int PRIMARY KEY NOT NULL,
customer_id int,
order_details varchar(50),
constraint fk_Customers foreign key (customer_id)
       references dbo.Customer
);
```

Assuming, that a customer orders more than once, the above relation represents **one to many** relation. Similarly we can achieve other mapping constraints based on the requirements.

35. Differentiate ORDER BY and GROUP BY clause with suitable example.

1. Order By :

Order by keyword sort the result-set either in ascending or in descending order. This clause sorts the result-set in ascending order by default. In order to sort the result-set in descending order **DESC** keyword is used.

Order By Syntax –

```
SELECT column_1, column_2, column_3..........
FROM Table_Name
ORDER BY column_1, column_2, column_3..... ASC|DESC;


Table_Name: Name of the table.
ASC: keyword for ascending order
DESC: keyword for descending order
```

2. Group By :

Group by statement is used to group the rows that have the same value. It is often used with aggregate functions for example:AVG(), MAX(), COUNT(), MIN() etc. One thing to remember about the group by clause is that the tuples are grouped based on the similarity between the attribute values of tuples.

Group By Syntax –

Group By Syntax –

```
SELECT function_Name(column_1), column_2
FROM Table_Name
WHERE condition
GROUP BY column_1, column_2
ORDER BY column_1, column_2;
```

function_Name: Name of the aggregate function, for example:

```
SUM(), AVG(), COUNT() etc.

Table_Name: Name of the table.
```

Group By	Order By
• Group by statement is used to group the rows that have the same value. • It may be allowed in CREATE VIEW statement. • In select statement, it is always used before the order by keyword.	• Whereas Order by statement sort the result-set either in ascending or in descending order. • While it does not use in CREATE VIEW statement. • While in select statement, it is always used after the group by keyword.

• Attribute cannot be in the group by statement under aggregate function. • In group by clause, the tuples are grouped based on the similarity between the attribute values of tuples. • Group by controls the presentation of tuples(rows).	• Whereas in order by statement, attribute can be under aggregate function. • Whereas in order by clause, the result-set is sorted based on ascending or descending order. • While order by clause controls the presentation of columns.

36. What do you mean by JOIN? Explain different types of join with suitable examples.

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

Consider the two tables below: Student

ROLL_NO	NAME	ADDRESS	PHONE	Age
1	HARSH	DELHI	xxxxxxxxxx	18
2	PRATIK	BIHAR	xxxxxxxxxx	19
3	RIYANKA	SILIGURI	xxxxxxxxxx	20
4	DEEP	RAMNAGAR	xxxxxxxxxx	18
5	SAPTARHI	KOLKATA	xxxxxxxxxx	19
6	DHANRAJ	BARABAJAR	xxxxxxxxxx	20
7	ROHIT	BALURGHAT	xxxxxxxxxx	18
8	NIRAJ	ALIPUR	xxxxxxxxxx	19

Student Course

COURSE_ID	ROLL_NO
1	1
2	2
2	3
3	4
1	5
4	9
5	10
4	11

The simplest Join is INNER JOIN.

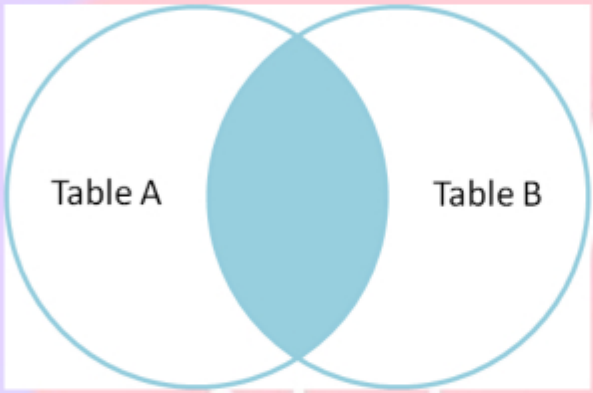
1. **INNER JOIN:** The INNER JOIN keyword selects all rows from both the tables as long as the condition satisfies. This keyword will create the result-set by combining all rows from both the tables where the condition satisfies i.e value of the common field will be same.

Syntax:

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


table1: First table.
table2: Second table
matching_column: Column common to both the tables.
```

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



Example Queries(INNER JOIN)

This query will show the names and age of students enrolled in different courses.

SELECT StudentCourse.COURSE_ID, Student.NAME, Student.AGE FROM Student
INNER JOIN StudentCourse
ON Student.ROLL_NO = StudentCourse.ROLL_NO;

OutPut

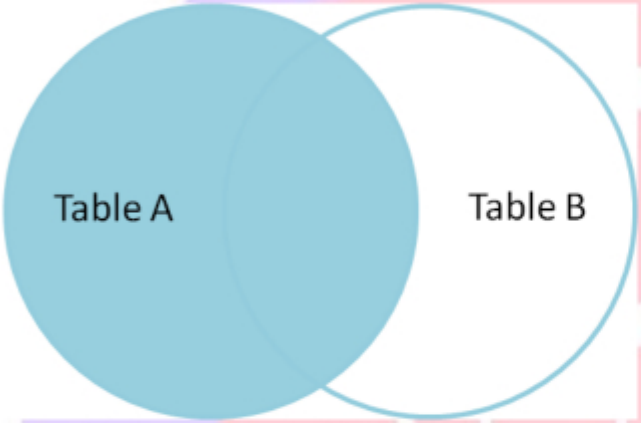
COURSE_ID	NAME	Age
1	HARSH	18
2	PRATIK	19
2	RIYANKA	20
3	DEEP	18
1	SAPTARHI	19

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 join. The rows for which there is no matching row on 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;


table1: First table.
table2: Second table
matching_column: Column common to both the tables.
```

Note: We can also use LEFT OUTER JOIN instead of LEFT JOIN, both are same.



Example Queries(LEFT JOIN):

```
SELECT Student.NAME,StudentCourse.COURSE_ID
FROM Student
LEFT JOIN StudentCourse
ON StudentCourse.ROLL_NO = Student.ROLL_NO;
```

Output

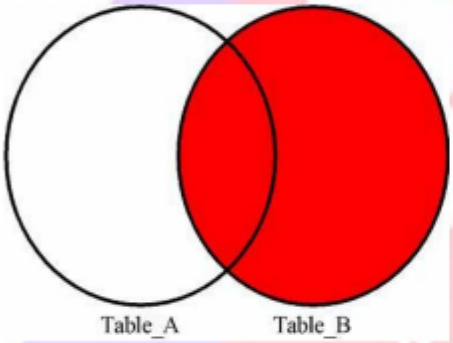
NAME	COURSE_ID
HARSH	1
PRATIK	2
RIYANKA	2
DEEP	3
SAPTARHI	1
DHANRAJ	*NULL*
ROHIT	*NULL*
NIRAJ	*NULL*

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 join. The rows for which there is no matching row on 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;


table1: First table.
table2: Second table
matching_column: Column common to both the tables.
```

Note: We can also use RIGHT OUTER JOIN instead of RIGHT JOIN, both are same.



Example Queries(RIGHT JOIN):

```
SELECT Student.NAME,StudentCourse.COURSE_ID
FROM Student
RIGHT JOIN StudentCourse
ON StudentCourse.ROLL_NO = Student.ROLL_NO;
```

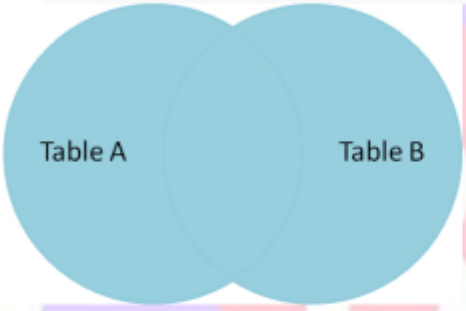
OutPut

NAME	COURSE_ID
HARSH	1
PRATIK	2
RIYANKA	2
DEEP	3
SAPTARHI	1
NULL	4
NULL	5
NULL	4

FULL JOIN: FULL JOIN creates the result-set by combining 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;


table1: First table.
table2: Second table
matching_column: Column common to both the tables.
```



Example Queries(FULL JOIN):

```
SELECT Student.NAME,StudentCourse.COURSE_ID
FROM Student
FULL JOIN StudentCourse
ON StudentCourse.ROLL_NO = Student.ROLL_NO;
```

Output

NAME	COURSE_ID
HARSH	1
PRATIK	2
RIYANKA	2
DEEP	3
SAPTARHI	1
DHANRAJ	*NULL*
ROHIT	*NULL*
NIRAJ	*NULL*
NULL	9
NULL	10
NULL	11

37. Explain importance of creating views in database. How views can be created in database?
38. Explain stored procedure with example.
39. How parameters can be created in stored procedure? Explain with example.
40. Explain trigger with example. What are the limitation of trigger?

Limitation of trigger:
Only one INSTEAD OF trigger is allowed for each event type (INSERT, UPDATE, or DELETE) per table.

If a table has a BEFORE trigger, it cannot have an INSTEAD OF trigger. The reverse is also true.

If a table has an INSTEAD OF X trigger, AFTER X triggers defined for that table will not fire (where X is an event type).

The __error table can only have one row. Delete operations are not currently allowed on the __error table.

VarChar fields are not supported in the __old and __new tables.

When using the Advantage Local Server, if a trigger fails for any reason, the database is left as is. This means any operations the trigger may have already performed will be persistent.

Nested and recursive triggers are limited to 64 levels of server re-entrance before an error is returned.

41. Explain the importance of creating index in database. Explain the process of creating and dropping index.

Unit – VIII

42. Explain query processing with suitable diagram. How query cost is measured?
43. Explain different algorithms used for selection operation.
44. Explain sorting operation with the help of external merge sort.
45. What do you mean by query optimization? Explain query optimization process.

Query optimization is a *feature* of many relational database management systems and other databases such as graph databases. The **query optimizer** attempts to determine the most efficient way to execute a given query by considering the possible query plans. [1]

teps for Query Optimization

Query optimization involves three steps, namely query tree generation, plan generation, and query plan code generation.

Step 1 − Query Tree Generation

A query tree is a tree data structure representing a relational algebra expression. The tables of the query are represented as leaf nodes. The relational algebra operations are represented as the internal nodes. The root represents the query as a whole.

During execution, an internal node is executed whenever its operand tables are available. The node is then replaced by the result table. This process continues for all internal nodes until the root node is executed and replaced by the result table.

For example, let us consider the following schemas −

EMPLOYEE

EmpID	EName	Salary	DeptNo	DateOfJoining

DEPARTMENT

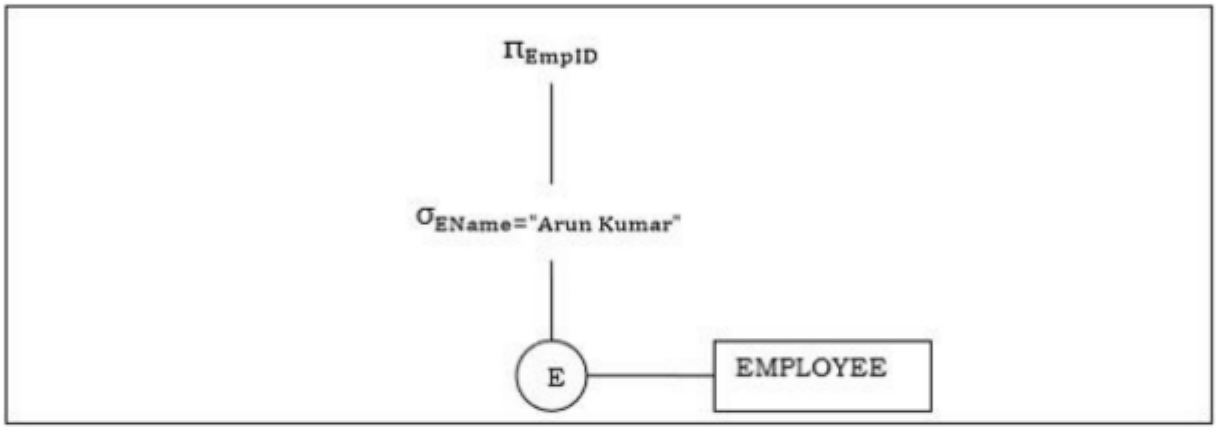
DNo	DName	Location

Example 1

Let us consider the query as the following.

$$\pi_{EmpID} (\sigma_{EName = \small "ArunKumar"} {(EMPLOYEE)})$$

The corresponding query tree will be −

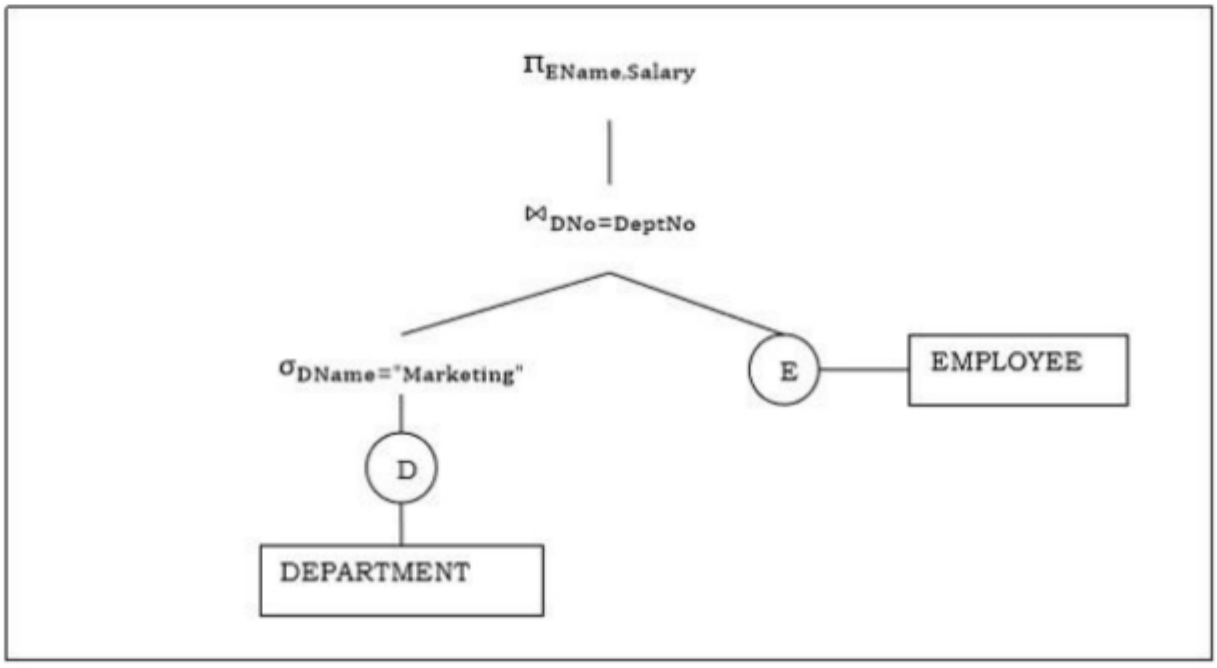


Example 2

Let us consider another query involving a join.

$\pi_{EName, Salary} (\sigma_{DName = \small "Marketing"} {(DEPARTMENT)})
\bowtie_{DNo=DeptNo}{(EMPLOYEE)}$

Following is the query tree for the above query.



Step 2 – Query Plan Generation

After the query tree is generated, a query plan is made. A query plan is an extended query tree that includes access paths for all operations in the query tree. Access paths specify how the relational operations in the tree should be performed. For example, a selection operation can have an access path that gives details about the use of B+ tree index for selection.

Besides, a query plan also states how the intermediate tables should be passed from one operator to the next, how temporary tables should be used and how operations should be pipelined/combined.

Step 3– Code Generation

Code generation is the final step in query optimization. It is the executable form of the query, whose form depends upon the type of the underlying operating system. Once the query code is generated, the Execution Manager runs it and produces the results.

46. Define query optimizer. Explain importance of query optimization.
    - A query optimizer is a critical database management system (DBMS) component that analyses Structured Query Language (SQL) queries and determines efficient execution mechanisms.
    - A query optimizer generates one or more query plans for each query, each of which may be a mechanism used to run a query. The most efficient query plan is selected and used to run the query.
    - Database users do not typically interact with a query optimizer, which works in the background.

Importance of Query Optimization

The goal of query optimization is to reduce the system resources required to fulfill a query, and ultimately provide the user with the correct result set faster.

First, it provides the user with faster results, which makes the application seem faster to the user.

Secondly, it allows the system to service more queries in the same amount of time, because each request takes less time than unoptimized queries.

Thirdly, query optimization ultimately reduces the amount of wear on the hardware (e.g. disk drives), and allows the server to run more efficiently (e.g. lower power consumption, less memory usage).

47. What do you mean by DBA? Explain different types of DBA.
    - A database administrator, frequently known just by the acronym DBA, is a role usually within the Information Technology department, charged with the creation, maintenance, backups, querying, tuning, user rights assignment and security of an organization's databases.
    - The role requires technical training and expertise in the specific RDBMS used by the organization, in addition to other skills such as analytical thinking and ability to concentrate on tasks, as well as experience working with databases in the real world. The DBA role is a critical member of the IT team.

Types of DBA

There are different kinds of DBA depending on the responsibility that he owns.

I. **Administrative DBA** – This DBA is mainly concerned with installing, and maintaining DBMS servers. His prime tasks are installing, backups, recovery, security, replications, memory management, configurations and tuning. He is mainly responsible for all administrative tasks of a database.

II. **Development DBA** – He is responsible for creating queries and procedure for the requirement. Basically his task is similar to any database developer.

Database Architect – Database architect is responsible for creating and maintaining the users, roles, access rights, tables, views, constraints and indexes. He is mainly responsible for designing the structure of the database depending on the requirement. These structures will be used by developers and development DBA to code.

III. **Data Warehouse DBA** –DBA should be able to maintain the data and procedures from various sources in the datawarehouse. These sources can be files, COBOL, or any other programs. Here data and programs will be from different sources. A good DBA should be able to keep the performance and function levels from these sources at same pace to make the datawarehouse to work.

IV. **Application DBA** –He acts like a bridge between the application program and the database. He makes sure all the application program is optimized to interact with the database. He ensures all the activities from installing, upgrading, and patching, maintaining, backup, recovery to executing the records works without any issues.

V. **OLAP DBA –** He is responsible for installing and maintaining the database in OLAP systems. He maintains only OLAP databases.

48. What are the different roles and responsibilities of DBA?
49. Define database security. What are the different database security issues?
50. What are the different types of database security? Differentiate MAC and DAC with example.
51. What do you mean by encryption and decryption? Explain public key and secret key cryptography with example.

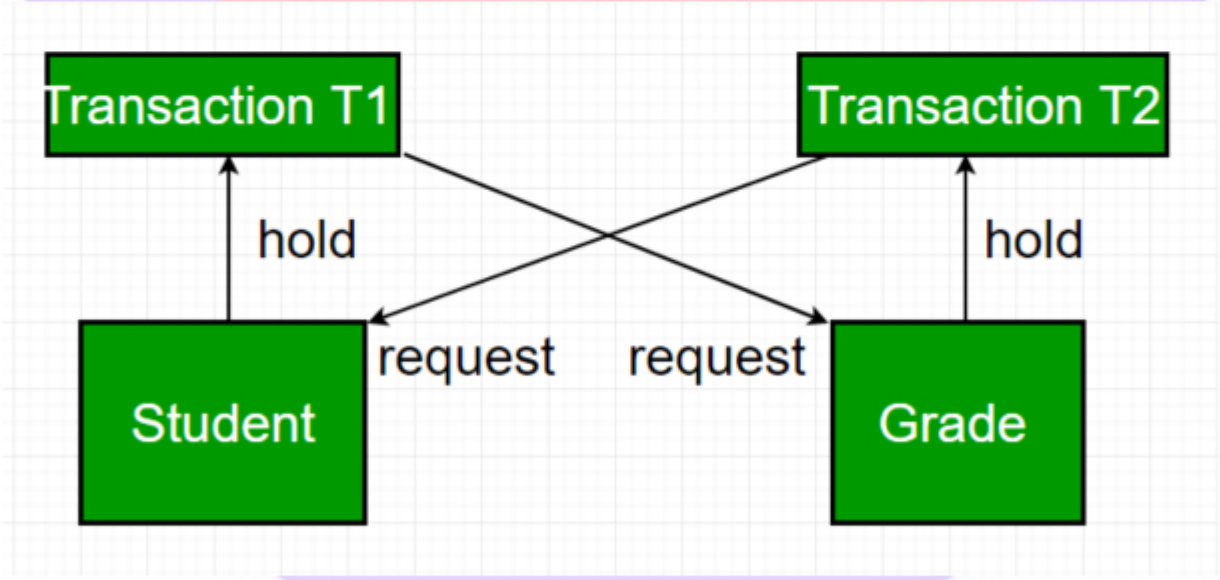
Unit – IX

52. Define transaction. Explain transaction model along with different states of transaction in detail.
53. Explain ACID properties of transaction with example.
54. What do you mean by serializability? Explain different types of serializability with example.
55. Describe concurrency control. What are the different types of concurrency control protocol? Explain lock based and time stamp based protocol in detail.
56. Explain deadlock in DBMS. How we can avoid deadlock in DBMS? Explain with example.

In a database, a deadlock is an unwanted situation in which 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 it brings the whole system to a Halt.

Example – let us understand the concept of Deadlock with an example :
Suppose, Transaction T1 holds a lock on some rows in the Students table and **needs to update** some rows in the Grades table. Simultaneously, Transaction **T2 holds** locks on those very rows (Which T1 needs to update) in the Grades table **but needs** to update the rows in the Student table **held by Transaction T1**.

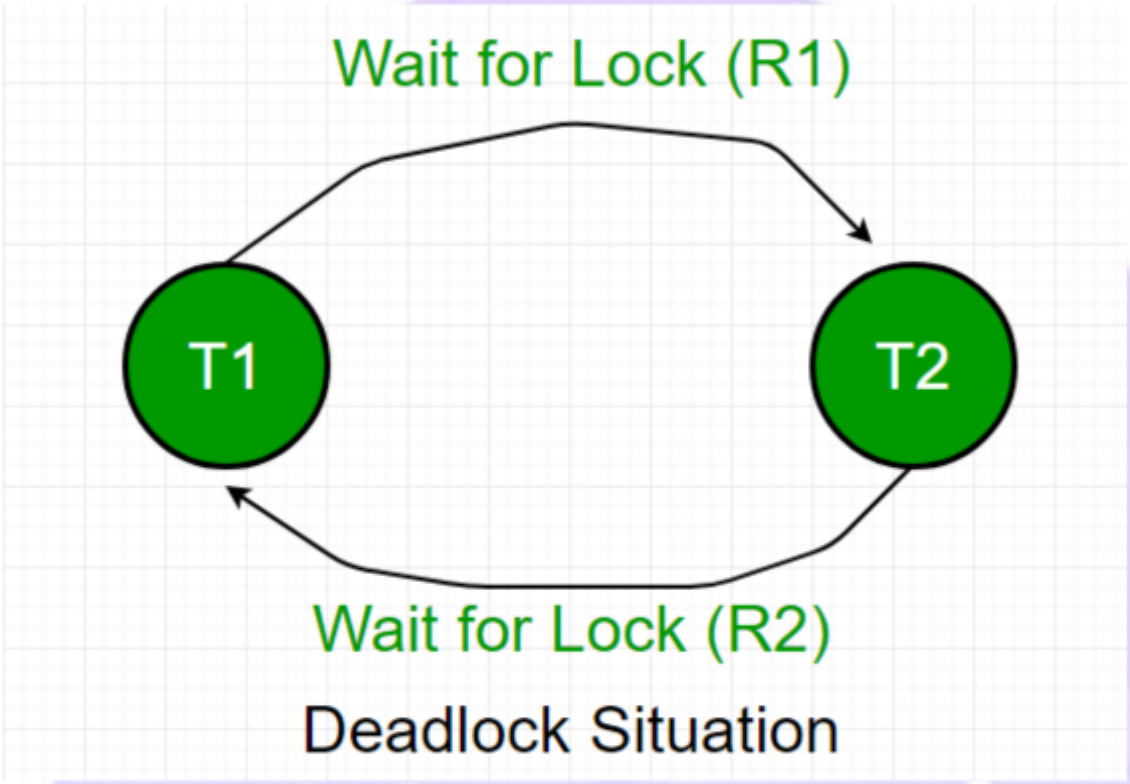
Now, the main problem arises. Transaction T1 will wait for transaction T2 to give up lock, and similarly, transaction T2 will wait for transaction T1 to give up the lock. As a consequence, All activity comes to a halt and remains at a standstill forever unless the DBMS detects the deadlock and aborts one of the transactions.



Deadlock in DBMS

Deadlock Avoidance –
When a database is stuck in a deadlock, It is always better to avoid the deadlock rather than restarting or aborting the database. Deadlock avoidance method is suitable for smaller databases whereas deadlock prevention method is suitable for larger databases.
One method of avoiding deadlock is using application-consistent logic. In the above given example, Transactions that access Students and Grades should always access the tables in the same order. In this way, in the scenario described above, Transaction T1 simply waits for transaction T2 to release the lock on Grades before it begins. When transaction T2 releases the lock, Transaction T1 can proceed freely.
Another method for avoiding deadlock is to apply both row-level locking mechanism and READ COMMITTED isolation level. However, It does not guarantee to remove deadlocks completely.

Deadlock Detection –
When a transaction waits indefinitely to obtain a lock, The database management system should detect whether the transaction is involved in a deadlock or not.
Wait-for-graph is one of the methods for detecting the deadlock situation. This method is suitable for smaller database. In this method a graph is drawn based on the transaction and their lock on the resource. If the graph created has a closed loop or a cycle, then there is a deadlock.
For the above mentioned scenario the Wait-For graph is drawn below



Deadlock prevention –
For large database, deadlock prevention method is suitable. A deadlock can be prevented if the resources are allocated in such a way that deadlock never occur. The DBMS analyzes the operations whether they can create deadlock situation or not, If they do, that transaction is never allowed to be executed.
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.