

DBMS Interview Questions

1.Explain about roll back and commit?

In MySQL (and databases in general), a "rollback" refers to the process of reverting a transaction's changes to the database to a previous consistent state. Here's how it works in MySQL:

1. **Transaction Concept:** In MySQL, a transaction is a sequence of one or more SQL operations treated as a single unit of work. Transactions allow multiple changes to be made to the database atomically (all or nothing).
2. **Commit and Rollback:** After performing a series of operations within a transaction, you have two options:
 - **Commit:** If you are satisfied with the changes and want them to be permanently applied to the database, you issue a COMMIT command. This makes all changes made by the transaction permanent.
 - **Rollback:** If something goes wrong or if you decide to discard the changes made by the transaction, you can issue a ROLLBACK command. This undoes all the changes made by the transaction since it started.
3. **Usage:** Rollbacks are typically used in scenarios where an error occurs during a transaction, and you want to undo all the changes made by that transaction to avoid leaving the database in an inconsistent state. For example, if you are transferring money between accounts and an error occurs midway, you would roll back the transaction to ensure the money isn't erroneously deducted from one account without being credited to the other.

Example of rollback and commit :

```
CREATE TABLE bank_accounts ( id INT AUTO_INCREMENT PRIMARY KEY, account_number VARCHAR(20) NOT NULL, balance DECIMAL(10, 2) NOT NULL );
```

id	account_number	balance
1	"1234567890"	1000.00
2	"9876543210"	500.00

Now, imagine we want to transfer 200.00 from account "1234567890" to account "9876543210". We'll perform this transfer within a transaction to ensure atomicity.

-- Start a transaction

START TRANSACTION;

-- Deduct 200.00 from account "1234567890"

UPDATE bank_accounts

SET balance = balance - 200.00

WHERE account_number = '1234567890';

-- Add 200.00 to account "9876543210"

UPDATE bank_accounts

SET balance = balance + 200.00

WHERE account_number = '9876543210';

-- Commit the transaction to make changes permanent

COMMIT;

In this case, if everything goes smoothly, the changes are committed, and the balances are updated accordingly:

id	account_number	balance
1	"1234567890"	800.00
2	"9876543210"	700.00

However, let's say an error occurs after deducting money from account "1234567890" but before adding it to account "9876543210" (e.g., network failure, system crash, etc.). In such a scenario, to ensure data consistency and integrity, we would roll back the transaction:

-- Start a transaction

START TRANSACTION;

-- Deduct 200.00 from account "1234567890"

UPDATE bank_accounts

SET balance = balance - 200.00

WHERE account_number = '1234567890';

-- Suppose an error occurs here before the next update

-- Roll back the transaction to undo changes

ROLLBACK;

After rolling back the transaction, the balances would revert to their original values

2.Difference between Drop, Delete and truncate?

TRUNCATE:

- **Operation:** TRUNCATE is a DDL (Data Definition Language) operation.
- **Function:** It removes all rows from a table quickly by deallocating the data pages used by the table.
- **Efficiency:** It is faster and uses fewer system and transaction log resources compared to DELETE.
- **Rollback:** TRUNCATE cannot be rolled back because it is not logged individually for each row deleted.

- **Conditions:** TRUNCATE does not allow WHERE clause or conditions; it removes all rows from the table.
- **Triggers:** Triggers associated with the table are not fired when using TRUNCATE.
- **Identity Columns:** Resets any identity (auto-increment) columns to their seed value.

DELETE:

- **Operation:** DELETE is a DML (Data Manipulation Language) operation.
- **Function:** Deletes rows one by one based on a condition specified with a WHERE clause.
- **Efficiency:** Generally slower compared to TRUNCATE, especially for large tables, because it logs each row deletion.
- **Rollback:** DELETE can be rolled back using a ROLLBACK statement if issued within a transaction.
- **Conditions:** Allows specifying conditions with a WHERE clause to selectively delete rows.
- **Triggers:** Triggers associated with the table are fired when rows are deleted using DELETE

DROP:

- **Operation:** DROP is a DDL (Data Definition Language) operation.
- **Function:** Removes an entire table, including its structure and data.
- **Efficiency:** Fast, similar to TRUNCATE, because it deallocates the data pages used by the table.
- **Rollback:** DROP cannot be rolled back. Once executed, the table and its data are permanently removed.
- **Conditions:** Does not apply conditions; drops the entire table.
- **Triggers:** Triggers associated with the table are not fired when using DROP.

3.what is a cursor explain?

In MySQL, a cursor is a database object that enables traversal over the rows of a result set from a query. It allows you to process individual rows returned by a SQL query one at a time, rather than handling the entire result set at once. Cursors are particularly useful when you need to perform operations on each row sequentially.

- Cursors are declared within the context of stored procedures or functions in MySQL.
- Before using a cursor, you need to declare it, open it, and fetch rows from it sequentially.
- Cursors are commonly used when you need to perform row-by-row operations or when the result set size is manageable but requires sequential processing.

-- Declare cursor

**DECLARE cursor_name CURSOR FOR SELECT column1, column2, ...
FROM table_name WHERE conditions;**

-- Open cursor

OPEN cursor_name;

-- Fetch rows read_loop:

LOOP FETCH cursor_name INTO var1, var2, ...;

IF done THEN LEAVE read_loop;

END IF;

-- Process fetched values here

-- Example:

SELECT var1, var2; END LOOP;

-- Close cursor CLOSE cursor_name;

4.what is a database?

A database is a structured collection of data that is organized and stored in a way that allows for efficient retrieval, management, and manipulation.

Key characteristics :

- Data in a database is organized into tables, which consist of rows (records) and columns (fields). This structured format allows for efficient storage and retrieval of information.
- Databases provide a level of abstraction between the physical storage of data (how it is stored on disk) and the logical view of data (how it is perceived and used by applications).
- Databases support a query language (e.g., SQL - Structured Query Language) that allows users and applications to interact with the database to retrieve, insert, update, and delete data.

5.How do you retrieve data from database?

Retrieving data from a database typically involves using SQL (Structured Query Language) queries to specify what data you want to retrieve and how you want it to be filtered, sorted, or formatted.

```
SELECT column1, column2, ...
```

```
FROM table_name
```

```
WHERE condition
```

```
ORDER BY column1 ASC/DESC;
```

6.what are different types of joins?

joins are used to combine rows from two or more tables based on a related column between them.

INNER JOIN:

- Returns records that have matching values in both tables.

```
SELECT *
```

```
FROM table1
```

```
INNER JOIN table2
```

```
ON table1.column_name = table2.column_name;
```

LEFT JOIN (or LEFT OUTER JOIN):

- Returns all records from the left table (table1), and the matched records from the right table (table2). If there is no match, NULL values are returned from the right side.

```
SELECT *  
FROM table1  
LEFT JOIN table2  
ON table1.column_name = table2.column_name;
```

RIGHT JOIN (or RIGHT OUTER JOIN):

- Returns all records from the right table (table2), and the matched records from the left table (table1). If there is no match, NULL values are returned from the left side.

```
SELECT *  
FROM table1  
RIGHT JOIN table2  
ON table1.column_name = table2.column_name;
```

FULL JOIN (or FULL OUTER JOIN):

- Returns all records when there is a match in either left (table1) or right (table2) table records. If there is no match, NULL values are returned on the side where there is no match.

```
SELECT *  
FROM table1  
FULL JOIN table2  
ON table1.column_name = table2.column_name;
```

7. Suppose There Are 6 Rows And 10 Columns What Will Be Output Of Select 1 From Table

The `SELECT 1 FROM table` statement in SQL selects a constant value 1 for each row in the table, effectively ignoring the actual columns in the table. This query is often used to check the existence of rows or to count rows without retrieving any actual data from the columns.

Given that there are 6 rows and 10 columns in the table, the output of `SELECT 1 FROM table` will be:

- A result set with 6 rows.
- Each row will contain the constant value 1.

8.Explain about normalization?

Normalization in MySQL (and relational databases in general) is the process of organizing data in a database to minimize redundancy and improve data integrity. It involves structuring a database in such a way that it reduces data duplication and ensures that data dependencies are logical.

Goals of normalization :

- Eliminate duplicate data to save storage space and maintain consistency.
- Ensure that data is accurate and consistent across the database.
- Make the database easier to update and maintain.

Normalization is achieved through a series of rules called "normal forms." Each normal form addresses specific issues related to redundancy and dependency. Here are the most commonly used normal forms:

First Normal Form (1NF):

- **Rule:** Ensure that each column contains only atomic (indivisible) values, and each column contains values of a single type.
- **Example:** If you have a table storing multiple phone numbers in a single column, split this data into separate rows or columns.

PersonID	Name	PhoneNumbers
1	John	123-456, 789-012
2	Alice	345-678

After 1NF:

PersonID	Name	PhoneNumber
1	John	123-456
1	John	789-012
2	Alice	345-678

Understand the key terms :

Non key columns : *These are columns in a table that are not part of the primary key.*

Composite primary key : *A primary key that consists of two or more columns. It is used when a single column is not sufficient to uniquely identify a row in the table.*

Second Normal Form (2NF):

- Rule: A table is in 2NF if it is in First Normal Form (1NF) and all non-key columns are fully dependent on the entire primary key, not just part of it.
 - In other words, there should be no partial dependencies. Partial dependency occurs when a non-key column is dependent on only a part of the composite primary key, not the whole primary key.

Example :

Initial Table (Not in 2NF):

Consider a table OrderDetails with the following columns:

OrderID	ProductID	ProductName	Quantity	OrderDate
1	101	Widget A	5	2023-01-01
1	102	Widget B	3	2023-01-01
2	101	Widget A	2	2023-02-01
2	103	Widget C	4	2023-02-01

- Composite Primary Key: (`OrderID`, `ProductID`)
- Non-Key Columns: `ProductName`, `Quantity`, `OrderDate`

In this table, ProductName is dependent only on ProductID, which means ProductName is partially dependent on the composite primary key (OrderID, ProductID). This violates 2NF.

Decomposing into 2NF:

To achieve 2NF, we need to remove partial dependencies by creating separate tables:

1. OrderDetails Table:

OrderID	ProductID	Quantity	OrderDate
1	101	5	2023-01-01
1	102	3	2023-01-01
2	101	2	2023-02-01
2	103	4	2023-02-01

- Composite Primary Key: (`OrderID`, `ProductID`)
- Non-Key Columns: `Quantity`, `OrderDate`

2. Products Table:

ProductID	ProductName
101	Widget A
102	Widget B
103	Widget C

- Primary Key: `ProductID`
- Non-Key Column: `ProductName`



Now, the OrderDetails table is in 2NF because all non-key columns (Quantity, OrderDate) are fully dependent on the entire composite primary key (OrderID, ProductID). The Products table holds the product information independently, ensuring no partial dependency.

A transitive dependency in the context of database normalization occurs when a non-key column depends on another non-key column rather than depending directly on the primary key.

Third Normal Form (3NF):

- **Rule:** A table is in 3NF if it is in Second Normal Form (2NF) and there are no transitive dependencies. This means that every non-key column must be directly dependent on the primary key and only the primary key.

Example of Transitive Dependency:

Consider a table EmployeeDetails with the following columns:

EmployeeID	EmployeeName	DepartmentID	DepartmentName
1	John	D1	HR
2	Alice	D2	IT
3	Bob	D1	HR

- Primary Key: `EmployeeID`
- Non-Key Columns: `EmployeeName`, `DepartmentID`, `DepartmentName`

Here, `DepartmentName` is dependent on `DepartmentID`, which is in turn dependent on `EmployeeID`. This creates a transitive dependency:

- `EmployeeID` → `DepartmentID` → `DepartmentName`

Removing Transitive Dependency:

To remove the transitive dependency and achieve 3NF, we need to separate the data into multiple tables so that non-key columns depend only on the primary key.

Decomposing into 3NF:


1. Employee Table:

EmployeeID	EmployeeName	DepartmentID
1	John	D1
2	Alice	D2
3	Bob	D1

- **Primary Key:** `EmployeeID`
- **Non-Key Columns:** `EmployeeName`, `DepartmentID`

2. Department Table:

DepartmentID	DepartmentName
D1	HR
D2	IT

- **Primary Key:** `DepartmentID`
- **Non-Key Column:** `DepartmentName` 

Now, in the Employee table, each non-key column (EmployeeName and DepartmentID) is directly dependent on the primary key (EmployeeID). The Department table holds the department information separately, ensuring that DepartmentName is directly dependent on DepartmentID.

BCNF :

- BCNF stands for Boyce-Codd Normal form. It is the stronger version of 3NF.
- It states that if a column or set of columns determine another column, then the determining column or columns must be a super key.

It follows two concepts :

- **Functional dependency** : Functional dependency states that if knowing the value of column X helps in determining column Y then $(X \rightarrow Y)$.

- **Superkey** : A set of columns which are used to uniquely identify each record or row in a table.
- To check if the table is in BCNF, we need to ensure that for every functional dependency $X \rightarrow Y$ and X should be a super key.

Example :

Let's start with a table Enrollments that records students enrolled in courses, along with the instructor teaching the course. The table has the following columns:

- StudentID
- CourseID
- InstructorID
- InstructorName

Functional Dependencies

Let's assume the following functional dependencies:

1. StudentID, CourseID \rightarrow InstructorID
2. InstructorID \rightarrow InstructorName
3. CourseID \rightarrow InstructorID

The candidate key for this table could be (StudentID, CourseID), because a student can enroll in multiple courses and each enrollment can be identified uniquely by these two columns.

Checking for BCNF

To check if the table is in BCNF, we need to ensure that for every functional dependency $X \rightarrow Y$, X should be a superkey.

1. Dependency 1: $\text{StudentID}, \text{CourseID} \rightarrow \text{InstructorID}$


- Here, `**StudentID**, **CourseID**` is a superkey because it uniquely identifies each row.
- This dependency is fine under BCNF.

2. Dependency 2: $\text{InstructorID} \rightarrow \text{InstructorName}$

- `**InstructorID**` is not a superkey because it does not uniquely identify each row in the original table.
- This violates BCNF.

3. Dependency 3: $\text{CourseID} \rightarrow \text{InstructorID}$

- `**CourseID**` is not a superkey because it does not uniquely identify each row.
- This also violates BCNF.

Since there are dependencies that violate BCNF  we need to decompose the table.

To achieve BCNF, we break down the table into smaller tables which are :

1.Instructors table

2.Courses

3.Enrollments

9.what is RDBMS?

RDBMS stands for **Relational Database Management System**. It is a type of database management system (DBMS) that stores data in a structured format, using rows and columns. The data is organized into tables, which allows for efficient retrieval and management. Here are the key features and concepts associated with RDBMS:

- Data is stored in tables (also called relations), where each table consists of rows (records) and columns (fields or attributes). Each table represents a specific entity, like customers, orders, or products.

- The schema defines the structure of the database, including the tables, columns, data types, and relationships between tables. It provides a blueprint for how data is organized.

10.difference between DBMS and RDBMS?

Feature	DBMS	RDBMS
Data Structure	Uses a file system to store data	Uses tables to store data
Relationship Handling	Does not support relationships between data	Supports relationships between tables using foreign keys
Normalization	Does not enforce normalization	Enforces normalization to reduce redundancy
Data Integrity	Limited data integrity constraints	Supports ACID properties for data integrity
Query Language	May use various query languages	Primarily uses SQL (Structured Query Language)
Data Redundancy	May have significant data redundancy	Minimizes data redundancy through normalization
Examples	File systems, XML databases, etc.	MySQL, PostgreSQL, Oracle, SQL Server, SQLite
Transaction Management	Limited transaction support	Robust transaction management with rollback and commit features
Security	Basic security features	Advanced security features including role-based access control
Scalability	Limited scalability	Highly scalable for large datasets and multiple users
Data Access	Data is accessed as files	Data is accessed through a relational schema
Data Model	Hierarchical, Network, or Object-oriented models	Relational model based on E.F. Codd's principles

11.what is mySQL?

SQL, or **Structured Query Language**, is a standardized programming language used for managing and manipulating relational databases. SQL is essential for interacting with databases, allowing users to perform a variety of operations on the data stored within them.

12. File system vs DBMS?

File system

A **file system** is a method and structure for storing and organizing files on a storage medium, such as a hard drive, SSD, or USB drive. It manages how data is stored, accessed, and managed on the storage device, ensuring that files are organized and retrievable.

- Organizes data into files, which can be documents, images, programs, etc.
- Files are stored in directories (or folders) which can be nested hierarchically.
- Provides methods for reading, writing, updating, and deleting files.
- Supports access permissions to control who can read or modify a file.
- Stores metadata about each file, such as name, size, type, creation date, and modification date.

Examples of File Systems:

- **NTFS** (New Technology File System) - used by Windows.
- **HFS+** (Hierarchical File System Plus) - used by older versions of macOS.
- **APFS** (Apple File System) - used by newer versions of macOS.
- **EXT4** (Fourth Extended File System) - used by Linux.

Database Management System (DBMS)

A **Database Management System (DBMS)** is software designed to store, retrieve, manage, and manipulate data in databases. It provides a systematic way to create, retrieve, update, and manage data, ensuring data integrity, security, and consistency.

- Organizes data into structured formats like tables, which consist of rows and columns.
- Supports various data types, including integers, strings, dates, and more.
- Provides SQL (Structured Query Language) for querying and manipulating data.
- Supports operations such as SELECT, INSERT, UPDATE, and DELETE.

Examples of DBMS:

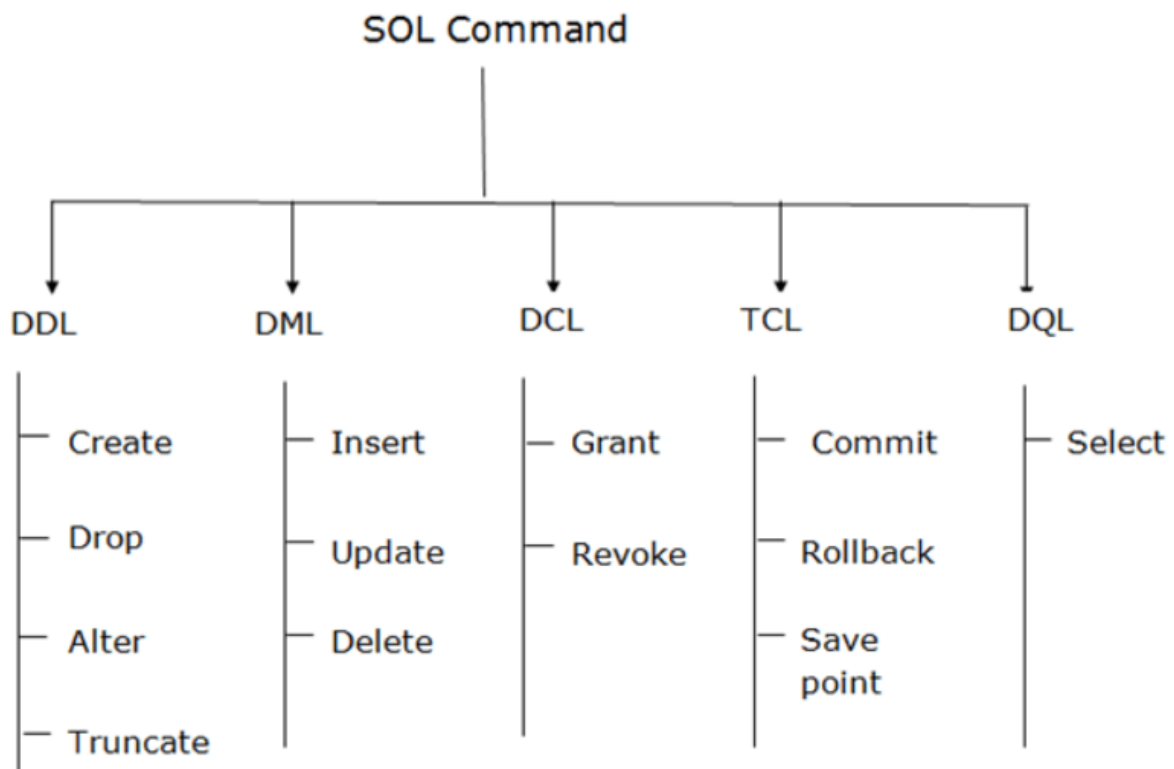
- **MySQL** - open-source RDBMS widely used for web applications.
- **PostgreSQL** - open-source RDBMS known for its advanced features.
- **Oracle Database** - commercial RDBMS used in large enterprises.

- **Microsoft SQL Server** - commercial RDBMS integrated with Microsoft products.
- **SQLite** - lightweight, file-based RDBMS used in embedded systems.

13.Types of SQL Commands

- SQL commands are instructions. It is used to communicate with the database. It is also used to perform specific tasks, functions, and queries of data.
- SQL can perform various tasks like create a table, add data to tables, drop the table, modify the table, set permission for users.

Types :



Data Definition Language :

- DDL changes the structure of the table like creating a table, deleting a table, altering a table, etc.
- All the command of DDL are auto-committed that means it permanently save all the changes in the database.

The CREATE statement is used to create database objects such as tables, indexes, views, and databases themselves.

```
CREATE TABLE Students (  
    StudentID INT PRIMARY KEY,  
    Name VARCHAR(100),  
    Age INT,  
    GPA DECIMAL(3, 2)  
);
```

The ALTER statement is used to modify existing database objects such as tables and indexes.

Adding column : ALTER TABLE Students ADD COLUMN Batch VARCHAR(10);

Modify column : ALTER TABLE Students MODIFY COLUMN Age INT NOT NULL;

Drop a column : ALTER TABLE Students DROP COLUMN GPA;

Rename a table : ALTER TABLE old_table_name RENAME TO new_table_name;

Rename a column : ALTER TABLE table_name RENAME COLUMN old_column_name TO new_column_name;

Drop command is used to delete database object

```
DROP TABLE Students;
```

The TRUNCATE statement is used to remove all records from a table quickly without deleting the table structure itself.

```
TRUNCATE TABLE Students;
```

Data Manipulation Language

- DML commands are used to modify the database. It is responsible for all form of changes in the database.
- The command of DML is not auto-committed that means it can't permanently save all the changes in the database. They can be rollback.

The INSERT statement is a SQL query. It is used to insert data into the row of a table.

```
INSERT INTO TABLE_NAME  
VALUES (value1, value2, value3, .... valueN);
```

UPDATE command is used to update or modify the value of a column in the table.

```
UPDATE table_name SET [column_name1= value1,...column_nameN = value  
N] [WHERE CONDITION]
```

DELETE: It is used to remove one or more row from a table.

```
DELETE FROM table_name [WHERE condition];
```

Data Control Language

DCL commands are used to grant and take back authority from any database user.

Grant: It is used to give user access privileges to a database.

1. GRANT SELECT, UPDATE ON MY_TABLE TO SOME_USER, ANOTHER_USER;

Revoke: It is used to take back permissions from the user.

1. REVOKE SELECT, UPDATE ON MY_TABLE FROM USER1, USER2;

Transaction Control Language

TCL commands can only use with DML commands like INSERT, DELETE and UPDATE only.

Commit: Commit command is used to save all the transactions to the database.

1. COMMIT;

Example:

1. DELETE FROM CUSTOMERS
2. WHERE AGE = 25;
3. COMMIT;

Rollback: Rollback command is used to undo transactions that have not already been saved to the database.

1. ROLLBACK;

Example:

1. DELETE FROM CUSTOMERS
2. WHERE AGE = 25;
3. ROLLBACK;

SAVEPOINT: It is used to roll the transaction back to a certain point without rolling back the entire transaction.

Syntax:

1. SAVEPOINT SAVEPOINT_NAME;

Data Query Language

DQL is used to fetch the data from the database.

It uses only one command:

- SELECT

a. SELECT: This is the same as the projection operation of relational algebra. It is used to select the attribute based on the condition described by WHERE clause.

Syntax:

SELECT expressions

FROM TABLES

WHERE conditions;

14.Explain about transaction and its states?

In SQL, a transaction is a sequence of one or more SQL operations (such as insert, update, delete, and select) executed as a single unit of work. A transaction ensures that the database remains in a consistent state

- **Active:** The transaction is currently executing one or more operations. It is in progress and has not yet been completed or committed.
- **Partially Committed:** The transaction has executed its final operation but has not yet been committed to the database. At this point, it is in the process of being committed.
- **Committed:** The transaction has been successfully completed, and all changes have been made permanent in the database. The system ensures that the changes are durable and will persist even in the case of a system failure.
- **Failed:** The transaction has encountered an error or has been explicitly rolled back before it reached the commit point. As a result, none of the changes made by the transaction are saved in the database.
- **Aborted (or Rolled Back):** The transaction has been terminated due to a failure or an explicit request for rollback. Any changes made by the transaction are undone, and the database is returned to its previous consistent state.

15.Explain all the ACID properties with an example?

1. Atomicity

Definition: Atomicity ensures that all operations within a transaction are completed successfully or none are. If any part of the transaction fails, the entire transaction is rolled back, and the database remains unchanged.

Example:

- Transaction: Transfer \$100 from Account A to Account B.

- Steps:
 1. Subtract \$100 from Account A.
 2. Add \$100 to Account B.

If the system crashes after step 1 but before step 2, Account A would have \$100 less, but Account B would not have \$100 more. Atomicity ensures that both steps are completed successfully or neither step is completed, so the system rolls back the subtraction from Account A, leaving both accounts unchanged.

2. Consistency

Definition: Consistency ensures that a transaction takes the database from one consistent state to another consistent state, maintaining all predefined rules and constraints.

Example:

- The bank's rule is that the total amount of money in both accounts combined should always be \$200.
- Initial State:
 - Account A: \$150
 - Account B: \$50
 - Total: $\$150 + \$50 = \$200$

After the transaction:

- Account A: \$50
- Account B: \$150
- Total: $\$50 + \$150 = \$200$

Consistency ensures that the total amount remains \$200 both before and after the transaction.

3. Isolation

Definition: Isolation ensures that transactions are executed in isolation from each other. The intermediate states of a transaction are not visible to other transactions until the transaction is committed.

Example:

- Two transactions:
 1. Transfer \$100 from Account A to Account B.
 2. Transfer \$50 from Account B to Account A.

If both transactions run concurrently without isolation, they might interfere with each other and lead to an inconsistent state. With isolation, one transaction will complete entirely before the other starts, ensuring the final state is consistent.

Scenario:

- Initial State:
 - Account A: \$200
 - Account B: \$100

Without Isolation:

- Transaction 1 subtracts \$100 from Account A (Account A: \$100, Account B: \$100).
- Transaction 2 adds \$50 to Account A (Account A: \$150, Account B: \$50).
- Transaction 1 adds \$100 to Account B (Account A: \$150, Account B: \$150).

With Isolation:

- Transaction 1 completes fully (Account A: \$100, Account B: \$200) before Transaction 2 starts.
- Transaction 2 then operates on the final state of Transaction 1, ensuring no intermediate states are visible.

4. Durability

Definition: Durability ensures that once a transaction has been committed, it will remain so, even in the event of a system crash. The changes are permanent.

Example:

- Transaction: Transfer \$100 from Account A to Account B.
- Steps:
 1. Subtract \$100 from Account A.
 2. Add \$100 to Account B.
 3. Commit the transaction.

After committing the transaction:

- Account A: \$50
- Account B: \$150

If the system crashes immediately after the commit, the changes will still be present in the database when the system restarts, ensuring the transaction's durability.

16.DBMS Architecture?

DBMS architecture depends upon how users are connected to the database to get their request done.

1-Tier Architecture

Definition: In a 1-Tier architecture, the database is directly accessible to the user. The user interacts with the database directly, and all the database operations are performed on the user's machine.

Characteristics:

- **Single Layer:** Both the database and the application reside on the same machine.
- **Direct Interaction:** Users interact directly with the database system.
- **Limited Scalability:** Suitable for single-user systems or small applications.

Example:

- **Standalone Applications:** Database management tools like Microsoft Access or SQLite where the database and the application are on the same system.

2-Tier Architecture

Definition: In a 2-Tier architecture, there are two layers: the client and the server. The client directly communicates with the server, and the server handles the database operations.

Characteristics:

- **Client-Server Model:** The client (application layer) interacts directly with the server (database layer).
- **Separation of Concerns:** The database is hosted on a server, and clients access it via a network.

- **Better Scalability:** More scalable than 1-Tier, supporting multiple users.

Example:

- **Client-Server Applications:** Applications where the client application communicates with a database server like MySQL, PostgreSQL, or SQL Server over a network.

3-Tier Architecture

Definition: In a 3-Tier architecture, there are three layers: the presentation layer (client), the application layer (middle tier or logic tier), and the database layer (data tier). Each layer has a specific role.

Characteristics:

- **Three Layers:**
 1. **Presentation Layer:** The user interface (UI) where users interact with the application.
 2. **Application Layer:** The business logic or middle tier that processes user requests and communicates between the presentation and database layers.
 3. **Database Layer:** The data storage where the database resides.
- **Modularity:** Each layer can be developed, deployed, and maintained independently.
- **Enhanced Scalability:** More scalable and suitable for larger applications with high user loads.
- **Security:** Better security as the application layer acts as a mediator, preventing direct access to the database.

Example:

- **Web Applications:** Typical web applications where the web browser (presentation layer) interacts with a web server (application layer) which, in turn, communicates with a database server (database layer).

Feature	1-Tier	2-Tier	3-Tier
Layers	Single layer	Two layers (Client-Server)	Three layers (Presentation, Application, Data)
Interaction	Direct user-database interaction	Client communicates directly with server	Client communicates with server through application layer
Scalability	Limited	Moderate	High
Security	Basic	Moderate	High
Development Complexity	Simple	Moderate	High
Use Case	Personal or small applications	Small to medium applications	Large enterprise applications
Example	Microsoft Access, SQLite	MySQL, PostgreSQL with client applications	Web applications like e-commerce platforms

17.what is a datamodel?

In a Database Management System (DBMS), a data model is a conceptual framework for organizing and structuring the data. It defines how data is connected, how data can be stored and retrieved, and how the relationships among data are maintained. Data models provide a systematic way to manage data and ensure its consistency, integrity, and security.

1. Hierarchical Data Model

Definition: In a hierarchical data model, data is organized in a tree-like structure. Each record has a single parent, but it can have multiple children.

Characteristics:

- **Tree Structure:** Data is structured in a hierarchy.
- **Parent-Child Relationship:** Each child record has only one parent.
- **Fast Access:** Efficient for certain types of queries that follow the hierarchical path.

Example:

- **File Systems:** Where directories have files and subdirectories.

2. Network Data Model

Definition: The network data model is an extension of the hierarchical model where a record can have multiple parent and child records, forming a graph structure.

Characteristics:

- **Graph Structure:** More complex relationships can be represented.
- **Many-to-Many Relationships:** Allows multiple relationships between records.
- **Flexible Navigation:** Can traverse the network in multiple ways.

Example:

- **Telecommunication Networks:** Where nodes represent switches or routers and edges represent connections.

3. Relational Data Model

Definition: In a relational data model, data is organized into tables (relations), where each table consists of rows (tuples) and columns (attributes). Each table represents an entity and relationships are established using keys.

Characteristics:

- **Tabular Structure:** Data is organized in tables.
- **Primary and Foreign Keys:** Used to establish relationships between tables.
- **SQL:** Structured Query Language (SQL) is used for querying and managing data.

Example:

- **Customer and Order Tables:** Customers have unique IDs (primary keys), and orders refer to customers using these IDs (foreign keys).

4. Entity-Relationship (ER) Model

Definition: The ER model is a high-level conceptual data model that defines data elements and their relationships. It uses entities, attributes, and relationships to model data.

Characteristics:

- **Entities:** Objects or concepts (e.g., Student, Course).
- **Attributes:** Properties of entities (e.g., Student Name, Course Title).
- **Relationships:** Connections between entities (e.g., Students Enroll in Courses).

Example:

- **University Database:** Entities like Students, Courses, and Instructors with relationships like Enrolls and Teaches.

Use Case:

- Designing databases at the conceptual level before implementation.

18.What is an ER model explain?

The Entity-Relationship (ER) model is a high-level conceptual data model used for database design. It provides a systematic way to visually represent the data and its relationships in a database. The ER model is instrumental in the initial stages of database design as it helps in understanding the data requirements and the structure of the database.

Key Components of the ER Model

1. **Entities**
2. **Attributes**
3. **Relationships**

1. Entities

Definition: An entity is an object or concept that can have data stored about it. Entities represent real-world objects or concepts.

Characteristics:

- Entities are typically nouns, like "Customer," "Order," "Product," etc.
- Each entity has a set of properties or attributes.

Example:

- In a university database, possible entities could be "Student," "Course," and "Instructor."

2. Attributes

Definition: Attributes are the properties or details of an entity. Each attribute has a value.

Types of Attributes:

- **Simple Attributes:** Indivisible values (e.g., first name, last name).
- **Composite Attributes:** Can be divided into smaller sub-parts (e.g., address can be subdivided into street, city, state, zip code).
- **Derived Attributes:** Values derived from other attributes (e.g., age can be derived from the date of birth).
- **Multivalued Attributes:** Attributes that can have multiple values (e.g., a person can have multiple phone numbers).

Example:

- For the "Student" entity, possible attributes are "StudentID," "Name," "DateOfBirth," "Email."

3. Relationships

Definition: Relationships define how entities are related to one another.

Types of Relationships:

- **One-to-One (1:1):** Each instance of entity A is related to one instance of entity B and vice versa (e.g., each person has one passport, and each passport is assigned to one person).
- **One-to-Many (1**

): Each instance of entity A can be related to multiple instances of entity B, but each instance of entity B is related to one instance of entity A (e.g., a customer can place multiple orders, but each order is placed by one customer).

- **Many-to-Many (M**

): Each instance of entity A can be related to multiple instances of entity B, and vice versa (e.g., students can enroll in multiple courses, and each course can have multiple students).

ER Diagram (ERD)

An ER Diagram is a graphical representation of the ER model. It uses various symbols to represent entities, attributes, and relationships.

Components:

- **Rectangles:** Represent entities.
- **Ellipses:** Represent attributes.
- **Diamonds:** Represent relationships.
- **Lines:** Connect entities to their attributes and entities to relationships.
- **Double Ellipses:** Represent multivalued attributes.
- **Dashed Ellipses:** Represent derived attributes.
- **Primary Key Attributes:** Underlined in the diagram.

19.Explain about Three schema architecture?

The three-schema architecture in a Database Management System (DBMS) is a framework that separates the database system into three levels: the internal schema, the conceptual schema, and the external schema. This separation helps to abstract the database details and promotes data independence, allowing changes at one level without affecting the other levels.

The Three Levels

1. **Internal Level**
2. **Conceptual Level**
3. **External Level**

1. Internal Level (Physical Schema)

Definition: The internal level describes how the data is physically stored in the database. It deals with the data storage and access methods, file organization, indexing, and other physical aspects.

Characteristics:

- Focuses on storage structures.
- Involves low-level details of data storage (e.g., data blocks, file systems).
- Manages performance optimization.

Example:

- Data is stored in blocks and indexed for faster retrieval.
- Use of B-trees or hash indexes to speed up access to data.

2. Conceptual Level (Logical Schema)

Definition: The conceptual level describes the structure of the entire database for a community of users. It represents the logical view of the entire database and hides the physical details from the users.

Characteristics:

- Defines what data is stored in the database and the relationships among those data.
- Provides a unified view of the entire database, abstracting away physical details.
- Ensures data integrity and security.

3. External Level (View Schema)

Definition: The external level consists of multiple user views or subschemas. Each user view represents a different perspective of the database tailored to the needs of different users or user groups.

Characteristics:

- Provides customized views of the database for different users.
- Enhances security by restricting access to only the relevant data for each user.
- Simplifies interaction with the database by presenting only the necessary data.

Example:

- A student view in a university database might show only the student's courses and grades.

- A faculty view might show courses they are teaching, and student information relevant to those courses

20.what is a schema?

- The data which is stored in the database at a particular moment of time is called an instance of the database.
- The overall design of a database is called schema.
- A database schema is the skeleton structure of the database. It represents the logical view of the entire database.
- A schema contains schema objects like table, foreign key, primary key, views, columns, data types, stored procedure, etc.
- A database schema can be represented by using the visual diagram. That diagram shows the database objects and relationship with each other.
- A database schema is designed by the database designers to help programmers whose software will interact with the database. The process of database creation is called data modeling.

21.What is Data independence?

Data independence can be explained using the three-schema architecture.

Data independence refers characteristic of being able to modify the schema at one level of the database system without altering the schema at the next higher level.

Logical Data Independence

- Logical data independence refers characteristic of being able to change the conceptual schema without having to change the external schema.
- Logical data independence is used to separate the external level from the conceptual view.
- If we do any changes in the conceptual view of the data, then the user view of the data would not be affected.

Physical Data Independence

- Physical data independence can be defined as the capacity to change the internal schema without having to change the conceptual schema.
- If we do any changes in the storage size of the database system server, then the Conceptual structure of the database will not be affected.
- Physical data independence is used to separate conceptual levels from the internal levels.

