1. What is DataBase and why do we need it?

A database is a structured collection of data that is organized and stored in a computer system. It allows users to easily access, manage, and manipulate large volumes of data efficiently. Databases are essential for storing, retrieving, and managing information in various applications, from simple to complex systems.

Data Organization: Databases provide a structured way to organize and store data. Instead of storing data in unstructured files or spreadsheets, databases use tables with rows and columns, making it easier to organize and manage data effectively.

Data Retrieval: Databases allow users to retrieve specific data quickly and efficiently using queries. Users can retrieve data based on specific criteria or conditions, making it easier to access the information they need when they need it.

Data Integrity: Databases ensure data integrity by enforcing rules and constraints on the stored data. This helps maintain the accuracy and consistency of the data, reducing the risk of errors or inconsistencies.

2. What are DDL DML AND DCL COMMANDS

Category	Description	Examples
DDL (Data Definition Language)	Used to define, modify, and manage database objects and their structure.	`CREATE`, `ALTER`, `DROP`, `TRUNCATE`, `RENAME`
DML (Data Manipulation Language)	Used to manipulate data stored in database objects.	`INSERT`, `UPDATE`, `DELETE`, `SELECT`, `MERGE`
DCL (Data Control Language)	Used to control access to data and database objects.	`GRANT`, `REVOKE`, `DENY`

3. Difference bw delete , truncate and drop

Feature	DELETE	TRUNCATE	DROP
Operation	Deletes specific rows from a table based on a condition specified in the WHERE clause.	Removes all rows from a table.	Deletes an entire table along with its structure, including columns, indexes, constraints, and data.
Rollback	Can be rolled back using transaction logs, allowing for a rollback if necessary.	Cannot be rolled back (in most RDBMS).	Cannot be rolled back.
Speed	Slower than TRUNCATE.	Faster than DELETE.	Not applicable.
Example	`DELETE FROM Employees WHERE Age < 30;`	`TRUNCATE TABLE Employees;`	`DROP TABLE Employees;`

4. Difference bw primary key and foreign key

Feature	Primary Key	Foreign Key
Definition	A column or set of columns that uniquely identifies each record in a table.	A column or set of columns that establishes a link or relationship between two tables.
Uniqueness	Must be unique within the table.	May contain duplicate values within the table.
Purpose	Ensures data integrity and enforces entity integrity.	Establishes referential integrity between related tables.
Columns Allowed	Only one primary key is allowed per table.	Multiple foreign keys can exist in a table.
Creation Constraint	Must be defined when creating the table.	Can be added after table creation.
Notation in ER Diagram	Represented by underlining the attribute(s).	Represented by arrows pointing to the referenced table.
Example	`` `CREATE TABLE Students (StudentID INT PRIMARY KEY, Name VARCHAR(50));` ``	`` `CREATE TABLE Grades (GradeID INT, StudentID INT, FOREIGN KEY (StudentID) REFERENCES Students(StudentID));` ``

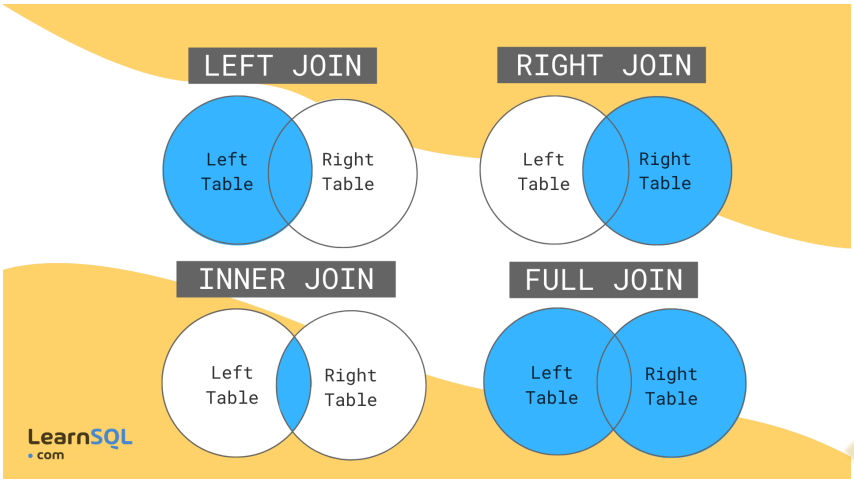
5. What are the different types of relationships in the DBMS?

One-to-One Relationship:
Imagine you have two people, John and Jane. Each of them has only one passport, and each passport belongs to only one person. That's a one-to-one relationship.

One-to-Many Relationship:
Now, think about a teacher and students in a classroom. One teacher can have many students, but each student only has one teacher. That's a one-to-many relationship.

Many-to-Many Relationship:
Lastly, picture a scenario where students can choose multiple courses, and each course can have many students enrolled. It's like a big network where students can be in multiple courses, and each course has many students. That's a many-to-many relationship.

6. Different types of join



Inner Join:
Think of two tables, one containing information about students and their math grades, and another containing information about students and their English grades. An inner join would combine these tables by matching up rows with the same student ID from both tables, showing only the students who have grades in both subjects.

Left Join:
Now, consider the same tables again. A left join would take all the rows from the table with math grades and match them with any corresponding rows from the table with English grades based on the student ID. If a student has grades only in math and not in English, their information would still be shown, with the English grade shown as NULL.

Right Join:
This is similar to the left join, but flipped. All rows from the table with English grades are taken, and if there are corresponding rows in the table with math grades, they are matched up based on the student ID. If not, the math grade is shown as NULL.

Full Outer Join:
Now, imagine you want to see all students' grades in both subjects, whether they have grades in one subject or both. A full outer join would take all rows from both tables and match them up based on the student ID. If a student has grades in only one subject, their information is still shown, with the missing grade shown as NULL.

7. Explain Different types of Constraints in sql

Constraint Type	Description
Primary Key	Ensures each row in a table has a unique identifier, typically used on columns with unique values.
Foreign Key	Establishes a relationship between two tables, ensuring values in one table match the primary key in another table.
Unique	Ensures all values in a column or group of columns are unique, similar to primary key but not necessarily identifying individual records.
Check	Verifies that all values in a column meet specific conditions, allowing you to enforce rules or restrictions on data.
Not Null	Ensures a column does not contain any NULL values, requiring each row to have a value for the specified column.

8. What is the difference between having and where clause?

WHERE Clause:

The WHERE clause is used to filter rows from the original table based on specified conditions.

It is applied before any aggregate functions (like SUM, COUNT, AVG) are calculated.

It filters rows based on individual values in columns.

HAVING Clause:

The HAVING clause is used to filter groups of rows returned by a GROUP BY clause.

It is applied after the data has been grouped using GROUP BY.

It filters groups of rows based on the result of aggregate functions (like SUM, COUNT, AVG).

In simple terms:

Use WHERE when you want to filter individual rows based on conditions.

Use HAVING when you want to filter groups of rows based on conditions after they have been grouped together using GROUP BY.

For example:

If you want to find students who have scored above 80, you would use WHERE.

If you want to find departments with an average salary greater than 50000, you would use HAVING because you first need to group salaries by department and then filter departments based on the average salary.

9. What is the main difference between UNION and UNION ALL?

Feature	UNION	UNION ALL
Deduplication	Removes duplicate rows from the result set.	Retains all rows, including duplicates, in the result set.
Performance	Generally slower due to the additional step of removing duplicates.	Generally faster because it does not remove duplicates.
Syntax	Uses `UNION` keyword.	Uses `UNION ALL` keyword.
Example	`SELECT column1 FROM table1 UNION SELECT column1 FROM table2;`	`SELECT column1 FROM table1 UNION ALL SELECT column1 FROM table2;`

10. Define normalisation , why do we use it and what are types of normalisation

Normalization is the process of organizing data in a database efficiently. It involves breaking down a database into smaller, manageable tables and defining relationships between them.

We use normalization to:

Reduce Redundancy: By organizing data into separate tables and linking them using relationships, we can avoid storing the same data multiple times, reducing redundancy.

Improve Data Integrity: Normalization helps maintain data integrity by minimizing the chances of anomalies such as insertion, update, or deletion anomalies.

Types Of Normalisation

First Normal Form (1NF):

Ensures that each column in a table contains atomic (indivisible) values and that there are no repeating groups of columns.

For example, if a table has multiple values in a single column separated by commas, it violates 1NF.

Second Normal Form (2NF):

Builds on 1NF and ensures that all non-key attributes are fully functionally dependent on the entire primary key.

In simple terms, it removes partial dependencies, where non-key attributes depend on only part of the primary key.

Third Normal Form (3NF):
Builds on 2NF and ensures that there are no transitive dependencies, where non-key attributes depend on other non-key attributes.
In other words, it removes dependencies between non-key attributes.

11. What is denormalisation and when to use it?

Denormalization is the process of adding redundancy to a database to improve query performance. We use it when we need faster queries or simpler data retrieval, especially in data warehouse environments or when read operations are more frequent than write operations.

12. What is transaction in database

A transaction in a database is a series of actions that are treated as a single unit of work. It ensures that either all of the actions within the transaction are completed successfully or none of them are. Transactions help maintain data integrity by allowing changes to be made in a consistent and reliable manner. In simple terms, it's like a banking transaction where money is transferred from one account to another: either the entire transfer is successful, or it's not.

13. What is indexing and why it is important

Indexing in databases is like creating a table of contents in a book. It's a way of organizing data to make it faster to search and retrieve information. Just as a book's index helps you find specific topics quickly, database indexes help speed up queries by allowing the database to locate specific rows or entries without having to search through every single one.

In simple terms, indexing is important because it:

1. **Speeds up data retrieval**: By creating indexes on columns frequently used in queries, the database can quickly locate and retrieve the relevant data, making queries faster.

2. **Improves query performance**: Indexes help reduce the amount of data that needs to be scanned, resulting in faster query execution times.

3. **Enhances overall database performance**: Faster query response times improve the overall performance of the database, leading to better user experiences and increased productivity.

In summary, indexing helps databases find and retrieve data quickly, making data access faster and more efficient.

14. Diff bw clustered and non-clustered index

Feature	Clustered Index	Non-Clustered Index
Organization	Dictates the physical order of data rows in the table.	Does not dictate the physical order of data rows.
Structure	The table itself is the index structure.	A separate index structure is created.
Number per table	Only one clustered index per table.	Multiple non-clustered indexes per table.
Performance	Generally faster for retrieval of large ranges of data or when performing sequential scans.	Generally faster for retrieval of individual rows or when performing specific searches.
Example	Primary key constraint often creates a clustered index.	Secondary index created separately from primary key.

15. What is view in a database

A view in a database is like a virtual table created by combining data from one or more tables. It's a way to simplify complex queries and present data in a more organized and meaningful way.

In simple terms, a view is like a window into your database that shows you specific data without you having to see all the underlying complexity. It's useful for simplifying data access, improving query performance, and ensuring consistent data presentation across multiple users or applications.

16. What are acid properties

ACID properties are the four key principles that ensure the reliability and consistency of transactions in a database:

1. **Atomicity**: This means that a transaction is treated as a single unit of work. It either completes in its entirety, or none of its changes are applied. It's like a light switch: when you flip it, either the light turns on completely or stays off; there's no in-between state.

2. **Consistency**: This ensures that the database remains in a consistent state before and after a transaction. All constraints, rules, and relationships defined in the database must be maintained. Think of it as a balance

scale: the sum of changes made by a transaction must keep the database in balance.

3. **Isolation**: Transactions should be isolated from each other, meaning the operations of one transaction should not interfere with those of another. Each transaction should proceed as if it's the only one running. It's like students taking an exam: they should focus on their own paper without being influenced by others.

4. **Durability**: Once a transaction is committed, its changes should be permanent and survive system failures. Even if the system crashes, the changes made by committed transactions should still be stored in the database. It's similar to saving a document: once you hit the save button, you expect your changes to be there even if your computer unexpectedly shuts down.

In summary, ACID properties ensure that database transactions are reliable, consistent, and durable, providing a solid foundation for data integrity and reliability.

17. What are triggers

Triggers in a database are like automatic reactions to certain events. They are special types of stored procedures that are automatically executed ("triggered") when specific actions occur in the database, such as inserting, updating, or deleting data in a table.

Triggers are like "if-then" statements: if a particular event happens (e.g., a new record is inserted into a table), then the trigger performs a predefined action (e.g., sends an email notification). Triggers are useful for enforcing business rules, maintaining data integrity, and automating tasks without manual intervention.

18. What is lock in database?

A lock in a database is like a temporary hold placed on a piece of data to prevent other transactions from modifying it simultaneously. It ensures data consistency and prevents conflicts that can arise when multiple transactions access the same data concurrently.