1. **What are the advantages of DBMS over traditional file based systems?**

There are several advantages of using a database management system (DBMS) over traditional file-based systems. Some of these advantages are:

1. Data consistency and integrity: A DBMS ensures that the data in the database is consistent and accurate. It enforces data integrity constraints, such as uniqueness and referential integrity, to maintain the integrity of the data.
2. Data sharing: A DBMS allows multiple users to access and share the same data concurrently. This is not possible in file-based systems where data is stored in files on a local disk and can only be accessed by one user at a time.
3. Data security: A DBMS provides better security and access control to the data. It allows the administrator to define access rights and permissions for different users, ensuring that only authorized users can access and modify the data.
4. Data independence: A DBMS provides data independence, which means that the application programs are independent of the underlying database structure. This makes it easier to modify the database structure without affecting the application programs.
5. Data backup and recovery: A DBMS provides a mechanism for backing up and restoring the database in case of system failures or disasters. This ensures that the data can be recovered even if the system crashes.
6. Improved performance: A DBMS provides optimized and efficient methods for accessing and manipulating data, resulting in faster data retrieval and processing. Additionally, a DBMS can handle large volumes of data efficiently, which may not be possible with a file-based system.

Overall, a DBMS provides a more efficient, secure, and scalable way to manage and store data than traditional file-based systems.

**2. What are primary and foreign keys?**

In a relational database, a primary key is a unique identifier for each record (row) in a table. It is a column or set of columns that uniquely identifies each row in the table, and it must have a unique value for each record. A primary key constraint ensures that the primary key values in a table are unique and not null.

A foreign key, on the other hand, is a column or set of columns in one table that refers to the primary key of another table. It establishes a relationship between two tables, where the foreign key in one table refers to the primary key in another table. The foreign key constraint ensures that the values in the foreign key column(s) of a table match the values in the primary key column(s) of the related table or are null.

For example, consider two tables: a "Customers" table and an "Orders" table. The "Customers" table might have a primary key column called "CustomerID", and the "Orders" table might have a foreign key column called "CustomerID" that refers to the "CustomerID" column in the "Customers" table. This establishes a relationship between the two tables, where each order is associated with a particular customer in the "Customers" table.

In summary, a primary key uniquely identifies each row in a table, while a foreign key establishes a relationship between two tables by referencing the primary key of another table.

**3. What is the difference between primary key and unique constraints?**

Both primary key and unique constraints ensure the uniqueness of data in a relational database, but they serve different purposes.

A primary key is a column or set of columns in a table that uniquely identifies each row in the table. It must have a unique value for each record in the table, and it cannot be null. A primary key constraint ensures that the primary key values in a table are unique and not null. Each table can have only one primary key.

A unique constraint, on the other hand, ensures that the values in a column or set of columns are unique. Like a primary key, a unique constraint does not allow duplicate values in the column(s) it applies to. However, unlike a primary key, a unique constraint can allow null values in the column(s) it applies to. A table can have multiple unique constraints.

So, the key difference between a primary key and a unique constraint is that a primary key uniquely identifies each row in a table and cannot have null values, whereas a unique constraint ensures uniqueness of values in a column(s) but allows null values.

In summary, both primary keys and unique constraints ensure data integrity by ensuring the uniqueness of values in a database, but a primary key is used to uniquely identify each row in a table and cannot have null values, while a unique constraint ensures the uniqueness of values in a column(s) but allows null values.

**4. What is database normalization and their types?**

Database normalization is a process of organizing the data in a database to eliminate redundancy and dependency, and to ensure data consistency and integrity. It involves breaking down a larger table into smaller tables and establishing relationships between them. Normalization reduces data duplication and ensures that each piece of data is stored only once.

There are several normal forms (NF) in database normalization, each building on the previous one. The most commonly used normal forms are:

* First Normal Form (1NF): A table is in 1NF if it has no repeating groups and every attribute (column) contains only atomic (indivisible) values.
* Second Normal Form (2NF): A table is in 2NF if it is in 1NF and every non-key attribute (column) is fully dependent on the primary key.
* Third Normal Form (3NF): A table is in 3NF if it is in 2NF and every non-key attribute (column) is non-transitively dependent on the primary key. This means that a non-key attribute cannot be derived from another non-key attribute.
* Boyce-Codd Normal Form (BCNF): A table is in BCNF if every determinant (a set of attributes that uniquely determines another attribute) is a candidate key. BCNF is a stricter form of 3NF.

There are additional normal forms beyond BCNF, such as fourth normal form (4NF) and fifth normal form (5NF), but they are less commonly used.

The process of normalization can involve dividing a larger table into smaller tables, creating relationships between them, and creating new tables to store data that does not fit well into the existing tables. The goal of normalization is to create a more efficient, flexible, and consistent database structure that is easier to maintain and update.

**5. What is SQL?**

SQL (Structured Query Language) is a standard programming language used to manage and manipulate relational databases. It is used to create, modify, and query databases, and to retrieve and manipulate data within them. SQL is a declarative language, which means that users specify what they want the database to do, and the SQL engine figures out how to do it.

SQL has a variety of commands, including:

* Data Definition Language (DDL) commands to create, alter, and drop database objects such as tables, indexes, and views.
* Data Manipulation Language (DML) commands to insert, update, and delete data in tables.
* Data Query Language (DQL) commands to retrieve data from tables based on specified conditions.
* Transaction Control Language (TCL) commands to manage transactions in the database, such as committing or rolling back changes.

SQL is used by a wide range of organizations and industries to manage and analyze large amounts of data. It is supported by most relational database management systems (RDBMS), such as MySQL, Oracle, Microsoft SQL Server, and PostgreSQL, and can also be used with non-relational databases through specialized SQL-based query languages.

**6. What are the differences between DDL, DML and DCL in SQL?**

In SQL, there are three types of commands that are used to manipulate databases: DDL (Data Definition Language), DML (Data Manipulation Language), and DCL (Data Control Language).

* DDL (Data Definition Language): DDL commands are used to define the structure of a database, such as creating, altering, and dropping tables, indexes, and other database objects. Some common DDL commands include CREATE, ALTER, and DROP. These commands are used to define the schema of a database, including the tables, columns, data types, constraints, and relationships.
* DML (Data Manipulation Language): DML commands are used to manipulate the data within the database. Some common DML commands include SELECT, INSERT, UPDATE, and DELETE. These commands are used to insert, update, retrieve, and delete data from the database. DML commands do not change the structure of the database; instead, they modify the data stored in the database.
* DCL (Data Control Language): DCL commands are used to control access to the database. Some common DCL commands include GRANT, REVOKE, and DENY. These commands are used to grant or revoke privileges to users and roles, such as the ability to execute DDL or DML commands or access specific tables or views. DCL commands are used to manage the security of the database.

In summary, DDL commands are used to define the schema of the database, DML commands are used to manipulate the data stored in the database, and DCL commands are used to control access to the database. Each of these commands plays an important role in managing and manipulating a relational database using SQL.

**7.What is the difference between having and where clause?**

In SQL, the HAVING and WHERE clauses are used to filter data based on specified conditions, but they are used in different contexts.

WHERE clause: The WHERE clause is used in a SELECT, UPDATE, or DELETE statement to filter data based on specified conditions. It is used to specify the search criteria for the rows that should be returned by the query. The WHERE clause filters rows based on the values in one or more columns, and only returns the rows that meet the specified conditions.

HAVING clause: The HAVING clause is used in a SELECT statement to filter data based on aggregate functions such as COUNT, SUM, AVG, MAX, or MIN. It is used to specify the search criteria for groups of rows that should be returned by the query. The HAVING clause filters groups based on the result of an aggregate function, and only returns the groups that meet the specified conditions.

In other words, the WHERE clause is used to filter rows based on specific column values, whereas the HAVING clause is used to filter groups based on the results of an aggregate function. The WHERE clause is used with a GROUP BY clause to group rows based on the values in one or more columns, and the HAVING clause is used to filter those groups based on the results of an aggregate function. The WHERE clause is applied before the GROUP BY clause, while the HAVING clause is applied after the GROUP BY clause.

**8.What is Join?**

In SQL, a JOIN is a way to combine data from two or more tables based on a related column between them. It allows you to retrieve data from multiple tables as if they were a single table. A JOIN statement is used to join two or more tables together based on a common column between them.

There are several types of JOINs in SQL:

* INNER JOIN: This type of JOIN returns only the rows from both tables that have matching values in the joined columns. It will exclude the rows where there is no matching value in both tables.
* LEFT JOIN (or LEFT OUTER JOIN): This type of JOIN returns all the rows from the left table and the matched rows from the right table. If there is no match in the right table, the result will contain NULL values.
* RIGHT JOIN (or RIGHT OUTER JOIN): This type of JOIN returns all the rows from the right table and the matched rows from the left table. If there is no match in the left table, the result will contain NULL values.
* FULL OUTER JOIN: This type of JOIN returns all the rows from both tables, with NULL values in the columns where there is no matching value in the other table.
* CROSS JOIN (or Cartesian product): This type of JOIN returns all the possible combinations of rows from both tables. It does not require a common column between the tables.

JOINs are commonly used in SQL to combine related data from multiple tables into a single result set. They are a powerful tool for querying data and retrieving information from complex database schemas.

**9.What is a view in SQL? How to create one?**

In SQL, a view is a virtual table that represents a subset of data from one or more tables in a database. Views are used to simplify queries and provide a layer of abstraction between the user and the underlying database schema. Views can be used to limit the data that is accessible to users, or to provide a customized perspective on the data that is more suitable for a particular application or user.

To create a view in SQL, you can use the CREATE VIEW statement. Here is the basic syntax:

Sql code:

CREATE VIEW view\_name AS

SELECT column1, column2, ...

FROM table\_name

WHERE condition;

In this statement, "view\_name" is the name of the view you want to create, "column1, column2, ..." are the columns you want to include in the view, "table\_name" is the name of the table you want to create the view from, and "condition" is an optional WHERE clause to filter the data.

For example, suppose you have a table called "customers" with columns "customer\_id", "first\_name", "last\_name", and "email". You could create a view that only includes the customer's first name and email address with the following statement:

Sql code:

CREATE VIEW customer\_emails AS

SELECT first\_name, email

FROM customers;

Once you have created a view, you can use it in the same way as you would use a regular table in SQL queries. You can select data from the view, join it with other tables, or use it as a source for INSERT, UPDATE, and DELETE statements. The data in a view is not stored separately from the underlying tables, so any changes to the tables will be reflected in the view as well.

**10.What is a Trigger and stored procedure and their difference?**

In SQL, a trigger and a stored procedure are two different types of database objects that can be used to automate tasks and enforce business rules. Here are the differences between them:

Trigger: A trigger is a special type of stored procedure that is automatically executed in response to certain events, such as the insertion, update, or deletion of data in a table. A trigger can be used to perform additional operations on the data, such as updating other tables, sending email notifications, or generating reports. A trigger is defined on a specific table or view and is automatically executed whenever the associated event occurs.

Stored Procedure: A stored procedure is a group of SQL statements that are saved in the database and can be executed on demand by a user or application. Stored procedures are often used to perform complex operations that involve multiple SQL statements or that require conditional logic or error handling. Stored procedures can also be used to improve performance by reducing network traffic and optimizing the use of database resources.

The main difference between a trigger and a stored procedure is that a trigger is automatically executed in response to a specific event, while a stored procedure is executed on demand by a user or application. Another difference is that a trigger is defined on a specific table or view, while a stored procedure can be executed on any table or view in the database.

In summary, a trigger is a special type of stored procedure that is automatically executed in response to certain events, while a stored procedure is a group of SQL statements that can be executed on demand by a user or application. Both triggers and stored procedures can be used to automate tasks and enforce business rules in a database, but they are used in different contexts and have different capabilities.

**11.What is a transaction? What are ACID properties?**

In SQL, a transaction is a sequence of one or more database operations that are treated as a single unit of work. A transaction can consist of multiple SQL statements, such as INSERT, UPDATE, and DELETE, that modify the database in some way. Transactions are used to ensure that database operations are performed atomically, meaning that they either all succeed or all fail, and that the database is left in a consistent state after the transaction is complete.

* ACID properties are a set of characteristics that define the reliability and consistency of transactions in a database. The four ACID properties are:
* Atomicity: This property ensures that a transaction is treated as a single, indivisible unit of work, and that either all the operations in the transaction are completed successfully, or none of them are. If a transaction is interrupted or fails for any reason, all changes made during the transaction are rolled back, leaving the database in its previous state.
* Consistency: This property ensures that a transaction preserves the integrity and validity of the data in the database. It ensures that the database is left in a consistent state after the transaction, meaning that all data constraints and rules are satisfied.
* Isolation: This property ensures that a transaction is isolated from other transactions being executed concurrently on the same database. It means that the intermediate state of a transaction is not visible to other transactions until it is completed, and that concurrent transactions do not interfere with each other.
* Durability: This property ensures that the changes made by a transaction are permanent and will survive any subsequent failures, such as power outages or system crashes. It means that once a transaction is committed, its changes are stored safely and can be recovered in the event of a failure.

Together, these ACID properties ensure that transactions in a database are reliable, consistent, and safe from failures and errors. They are a fundamental aspect of database design and are used to ensure that the data in the database is always accurate and consistent, even in the presence of failures or concurrent access by multiple users.

**12.What are indexes? And what are the types of indexes?**

In a database management system, an index is a data structure that provides fast access to specific rows in a table, based on the values of one or more columns. An index allows the database to locate the rows that match a certain search criteria much more quickly than scanning the entire table.

There are several types of indexes in a database:

* Clustered Index: This type of index determines the physical order of data in a table. A table can have only one clustered index, and it is usually created on the primary key of the table.
* Non-Clustered Index: This type of index stores the indexed columns and a pointer to the actual data rows in the table. A table can have multiple non-clustered indexes.
* Unique Index: This type of index ensures that the values in the indexed column(s) are unique. It is similar to a non-clustered index, but with the additional constraint of uniqueness.
* Composite Index: This type of index is created on multiple columns. It is useful when a query involves a WHERE clause that filters on multiple columns.
* Covering Index: This type of index includes all the columns required for a query, so that the database can satisfy the query from the index alone, without accessing the table itself.
* Spatial Index: This type of index is used to store and search for spatial data, such as maps or GPS coordinates.

Indexes are an important part of database design and optimization, as they can significantly improve the performance of database queries. However, creating too many indexes or using them improperly can also have a negative impact on performance, as they can increase the overhead of updates and inserts. Therefore, indexes should be chosen and designed carefully based on the specific needs of the database and the queries that will be run against it.

**13.What is embedded and dynamic SQL?**

Embedded SQL and Dynamic SQL are two different approaches to incorporating SQL statements into a program.

Embedded SQL refers to the practice of including SQL statements directly in a program written in another language, such as C, COBOL, or Java. In embedded SQL, the SQL statements are compiled and executed as part of the program, and the results are returned to the program for further processing. Embedded SQL is often used in large, complex systems that require high performance and tight integration between the database and application layers.

Dynamic SQL, on the other hand, refers to the practice of constructing SQL statements dynamically at runtime, based on the inputs or conditions of the program. Dynamic SQL allows programs to generate SQL statements on the fly, instead of hard-coding them into the program. This makes it easier to write flexible, reusable programs that can adapt to changing conditions or requirements. Dynamic SQL is often used in web applications, where users may want to query or manipulate data in a variety of ways.

One advantage of embedded SQL is that it can be optimized for performance, as the SQL statements can be compiled and executed directly by the database engine. However, embedded SQL can also be more complex and harder to maintain than dynamic SQL, as the SQL statements are tightly coupled to the program code.

Dynamic SQL, on the other hand, offers greater flexibility and ease of use, as SQL statements can be generated dynamically based on user input or other conditions. However, dynamic SQL can also be less efficient than embedded SQL, as the SQL statements must be parsed and executed at runtime, which can introduce overhead and slow down the program.