- **what four main types of action involve databases? Briefly discuss each?**

The four main types of action that involve databases are:

1. Create: Creating a database involves designing its structure, defining its tables, and specifying its relationships between tables. The creation process requires careful planning to ensure that the database can efficiently store and retrieve the required data.
2. Read: Reading data from a database involves querying the database to retrieve specific information. This process involves searching for data in one or more tables and retrieving the matching records. The read operation can be as simple as a basic search, or it can involve complex queries that require joining multiple tables.
3. Update: Updating a database involves modifying existing data or adding new data. This process requires the user to have appropriate permissions to modify the data in the database. Updates can be as simple as changing a single field in a record, or they can involve more complex operations, such as adding or deleting records or modifying multiple tables at once.
4. Delete: Deleting data from a database involves removing records or entire tables from the database. This operation requires careful consideration, as it can permanently remove important data from the database. The delete operation is typically restricted to users with appropriate permissions to prevent accidental deletion of data.

In summary, creating, reading, updating, and deleting (CRUD) are the four main types of actions that involve databases. These actions are critical to the management and manipulation of data in a database system.

- **Explain the Qruey Processing stages**

Query processing is the process of converting a user's query into a series of steps that can be executed by the database management system (DBMS) to retrieve the required data. There are several stages involved in the query processing process, which are as follows:

1. Parsing: The first stage of query processing is parsing. In this stage, the DBMS analyzes the syntax of the query and checks it for any errors. The query is broken down into its constituent parts, such as keywords, clauses, and operators, which are then used to generate a query execution plan.
2. Optimization: Once the query has been parsed, the DBMS will try to optimize it to execute it in the most efficient way possible. The query optimizer examines various

possible execution plans, and selects the one that will result in the least amount of I/O, CPU usage, and execution time.

3. Semantic Analysis: In this stage, the DBMS checks the validity of the query, making sure that it is semantically correct and that it can be executed against the database schema. The DBMS also checks if the user has the necessary permissions to execute the query.
4. Execution: Once the query has been parsed, optimized, and validated, it is executed against the database. The DBMS retrieves the data from the relevant tables and returns the result to the user.
5. Result Processing: The final stage of query processing is result processing. In this stage, the DBMS formats the result and presents it to the user. The DBMS also releases any resources that were allocated during the query execution process.

In summary, query processing involves several stages, including parsing, optimization, semantic analysis, execution, and result processing. These stages are critical to the efficient execution of queries against a database.

- **Define the following terms: data, database, DBMS, database system, user view, DBA, end user, meta-data, and transaction-processing.**

1. Data: Data refers to any information that can be stored and processed by a computer system. It can take various forms, such as text, numbers, images, audio, or video.
2. Database: A database is a collection of related data that is organized in a structured way and stored in a computer system. Databases are designed to allow efficient storage, retrieval, and manipulation of data.
3. DBMS: A database management system (DBMS) is a software system that enables users to manage and manipulate databases. It provides tools for creating, modifying, querying, and administering databases.
4. Database system: A database system is a combination of hardware, software, and data that is used to manage and manipulate databases. It includes the DBMS, the database, and any associated software and hardware.
5. User view: A user view is a subset of a database that is customized for a particular user or group of users. It provides a simplified and customized view of the data that is relevant to the user's needs.
6. DBA: A database administrator (DBA) is a person responsible for managing and maintaining a database system. Their responsibilities include database design, security, backup and recovery, performance tuning, and user management.
7. End user: An end user is a person who uses a database system to access and manipulate data. They may be employees of an organization or external users such as customers or suppliers.
8. Meta-data: Meta-data refers to data about data. It includes information such as data types, data formats, data definitions, and relationships between data elements. Meta-data is used by DBMS to manage and manipulate data.
9. Transaction-processing: Transaction-processing refers to a set of operations that are executed as a single unit of work. A transaction may involve multiple database operations, such as reading, updating, or deleting data. Transaction-processing ensures that all operations are completed successfully or rolled back in case of failure to maintain data consistency and integrity.

- **What are the responsibilities of the DBA and the database designers?**

The responsibilities of the DBA (database administrator) and the database designers are different but complementary.

The DBA is responsible for the overall management and maintenance of the database system. Their responsibilities may include:

1. Installing, configuring, and upgrading the database software.
2. Creating and managing user accounts and access permissions.
3. Defining and enforcing security policies and procedures.
4. Monitoring the performance and capacity of the database system.
5. Planning and implementing backup and recovery procedures.
6. Troubleshooting and resolving issues with the database system.
7. Ensuring compliance with data protection and privacy regulations.
8. Providing technical support to end users.

On the other hand, database designers are responsible for designing the database schema and ensuring that it meets the requirements of the organization. Their responsibilities may include:

1. Gathering and analyzing requirements from stakeholders.
2. Designing the database schema and defining tables, columns, and relationships.
3. Defining data validation and integrity constraints.
4. Normalizing the database to eliminate data redundancy and improve efficiency.
5. Optimizing database performance by creating indexes and tuning queries.
6. Documenting the database schema and data dictionaries.
7. Collaborating with developers to ensure that the database schema is integrated with the application.
8. Providing guidance and training to end users on how to use the database.

In summary, the DBA is responsible for the overall management and maintenance of the database system, while the database designers are responsible for designing the database schema to meet the organization's requirements. Both roles are critical to the efficient and effective use of the database system.

- **What are ACID properties? What is deadlock?**

ACID properties are a set of four key properties that guarantee the reliability and consistency of transactions in a database management system. The acronym ACID stands for:

1. Atomicity: A transaction is atomic, which means that it is an indivisible unit of work that either completes successfully or fails completely. If a transaction fails, any changes made to the database are rolled back to their previous state.
2. Consistency: A transaction brings the database from one consistent state to another consistent state. This means that the data in the database is valid and conforms to any defined constraints, such as data types, relationships, and business rules.
3. Isolation: Each transaction is isolated from other transactions, which means that its operations are not visible to other transactions until it is completed. This prevents conflicts and ensures that the database remains consistent.

4. Durability: Once a transaction is committed, its changes are permanent and will survive any subsequent system failures or crashes. This means that the database can recover to a consistent state after a failure.

Deadlock is a situation in which two or more transactions are waiting for each other to release resources, such as locks, that are required to complete their operations. This can result in a circular wait, where each transaction is waiting for a resource that is held by another transaction. Deadlocks can cause transactions to be blocked indefinitely, which can lead to performance issues and data inconsistency. To prevent deadlocks, database systems use techniques such as lock timeouts, deadlock detection, and deadlock prevention strategies, such as resource allocation ordering or graph-based algorithms.

- **Explain Locking Properties in detail**

Locking is a technique used by database systems to manage concurrent access to shared resources, such as data records, pages, or tables, to ensure data consistency and integrity. Locks are used to prevent conflicts between transactions that access the same resources simultaneously. Locking properties refer to the behavior of locks and how they affect the performance and concurrency of database systems. The following are the most common locking properties:

1. Shared lock (S-lock): A shared lock is a type of lock that allows multiple transactions to read data simultaneously but prohibits any transaction from modifying the data until the lock is released. Shared locks are also known as read locks because they allow read-only access to the data. Multiple transactions can acquire shared locks on the same resource simultaneously.
2. Exclusive lock (X-lock): An exclusive lock is a type of lock that prohibits any other transaction from accessing the locked resource until the lock is released. An exclusive lock is also known as a write lock because it allows write access to the data. Only one transaction can acquire an exclusive lock on a resource at a time.
3. Intent lock (I-lock): An intent lock is a type of lock that is used to indicate the intention of a transaction to acquire a lock on a resource. An intent lock is a higher-level lock that is acquired on a parent resource before acquiring lower-level locks on child resources. For example, a transaction may acquire an intent lock on a table before acquiring shared or exclusive locks on its data records.
4. Update lock (U-lock): An update lock is a type of lock that is a combination of a shared lock and an exclusive lock. An update lock allows a transaction to read data and update it later without the risk of conflicting with other transactions that also read the same data. An update lock is acquired when a transaction intends to update a record, but it does not want to block other transactions that only intend to read the record.
5. Deadlock: A deadlock occurs when two or more transactions are waiting for each other to release locks that are needed to proceed, resulting in a circular wait. Deadlocks can occur in any locking scheme, but they are more likely to occur in systems that use a strict two-phase locking protocol. To prevent deadlocks, database systems use techniques such as lock timeouts, deadlock detection, and deadlock prevention strategies, such as resource allocation ordering or graph-based algorithms.

Locking properties are critical to the performance and concurrency of database systems. The choice of a locking scheme depends on the specific requirements of the application, the level of concurrency, and the performance goals. A good locking scheme should balance the need for concurrency with the need for data consistency and integrity.

- **Define the following terms:**
- **a. data model**
- **b. database schema**
- **c. data independence**
- **d. DDL, DML**

a. Data model: A data model is a conceptual representation of the data in a database that describes the relationships between data elements, the constraints that govern those relationships, and the operations that can be performed on the data. Data models can be hierarchical, network, relational, or object-oriented.

b. Database schema: A database schema is a logical blueprint that describes the structure and organization of a database, including the tables, columns, keys, relationships, and constraints. The schema defines the types of data that can be stored in the database, how the data is organized, and the rules that govern its manipulation.

c. Data independence: Data independence is the ability to modify the structure of a database without affecting the applications that use it. There are two types of data independence: physical data independence and logical data independence. Physical data independence allows changes to the physical storage structure of the database without affecting the logical view of the data. Logical data independence allows changes to the logical structure of the database, such as adding or removing tables or columns, without affecting the application programs that use the data.

d. DDL and DML: DDL (Data Definition Language) is a set of SQL statements used to define the structure and schema of a database, including creating tables, defining columns, specifying keys, and setting constraints. DML (Data Manipulation Language) is a set of SQL statements used to manipulate the data in a database, including inserting, updating, and deleting data from tables.

- **What is the difference between the two-tier and three-tier client/server architectures**

The main difference between two-tier and three-tier client/server architectures is the number of layers or tiers in the architecture.

In a two-tier client/server architecture, there are two tiers or layers: the client layer and the server layer. The client layer typically includes the user interface, application logic, and some data processing functions, while the server layer includes the database management system and the data storage. The client layer communicates directly with the server layer, and the two layers work together to perform the required tasks. This architecture is simpler and more straightforward than the three-tier architecture and is suitable for small-scale applications that do not require complex processing.

In a three-tier client/server architecture, there are three tiers or layers: the presentation layer, the application layer, and the data layer. The presentation layer is responsible for the user interface and provides a graphical user interface (GUI) for the user to interact with. The application layer contains the business logic and application processing, which are independent of the user interface and the database management system. The data layer contains the database management system and the data storage. The three layers work together to perform the required tasks. This architecture is more complex and flexible than the two-tier architecture and is suitable for large-scale applications that require scalability, security, and distributed processing.

In summary, the main differences between two-tier and three-tier client/server architectures are the number of tiers or layers and the level of complexity and flexibility. Two-tier architecture is simpler and more suitable for small-scale applications, while three-tier architecture is more complex and suitable for large-scale applications.

- **Discuss insertion, deletion, and modification anomalies. Why are they considered bad? Illustrate with examples.**

Insertion, deletion, and modification anomalies are common issues that can occur in a database when it is not properly designed. These anomalies can lead to inconsistencies in the data and can make it difficult to update, delete or insert data into the database.

Insertion anomaly: An insertion anomaly occurs when data cannot be inserted into the database without including additional, unnecessary data. For example, consider a database that has a table of employees and their departments. If a new employee joins the company but there is no corresponding department in the database, the employee's data cannot be inserted until a new department is created. This creates an insertion anomaly because the creation of a new employee record is dependent on the creation of a new department record.

Deletion anomaly: A deletion anomaly occurs when deleting data from the database unintentionally deletes other data as well. For example, consider a database that has a table of employees and their departments. If an employee is the only employee in their department, and that employee is deleted from the database, the department information is also deleted, even though there may be other employees in that department. This creates a deletion anomaly because deleting an employee record has unintended consequences on other records.

Modification anomaly: A modification anomaly occurs when modifying data in the database leads to inconsistencies in the data. For example, consider a database that has a table of customers and their orders. If a customer changes their address, then every order that they have placed will need to be updated with the new address. This creates a modification anomaly because a change in one record requires the modification of multiple other records.

These anomalies are considered bad because they can lead to data inconsistencies and errors, which can cause incorrect results in data analysis, reporting, and decision-making. They also make it difficult to maintain data integrity and can result in poor performance and inefficiencies in data processing.

To avoid these anomalies, it is important to properly design the database and normalize the data to eliminate redundant and unnecessary data. Normalization involves breaking up tables into smaller, more manageable tables and establishing relationships between them, so that each table only contains data that pertains to a single entity or concept. By doing so, insertion, deletion, and modification anomalies can be minimized, if not eliminated entirely.

- **Why should NULLS in a relation be avoided as much as possible? Discuss the problem of [unauthentic/fake] spurious tuples and how we may prevent it.**

NULL values in a relation should be avoided as much as possible because they can lead to several problems. One of the main problems with NULLs is that they can cause confusion and inconsistencies in the data. For example, if a column in a table allows NULL values, it can be difficult to determine whether a NULL value represents a missing value or a value that has not been specified yet. This can make it difficult to query the data and can lead to incorrect results.

Another problem with NULLs is that they can cause problems when performing calculations or aggregations on the data. For example, if a column contains NULL values, any calculations or aggregations involving that column may result in NULL values, even if some of the values in the column are not NULL. This can lead to incorrect results and can make it difficult to perform data analysis and reporting.

Spurious tuples are tuples that do not represent real-world entities or relationships, but rather are created as a result of combining tables improperly. For example, suppose we have two tables, one containing information about students and another containing information about courses. If we combine these tables without properly establishing relationships between them, we may end up with spurious tuples that do not represent real-world entities, such as a student who is not enrolled in any course.

To prevent spurious tuples, it is important to properly design the database and establish relationships between tables. This involves identifying the entities and relationships in the data and creating tables that represent these entities and relationships. It also involves ensuring that each table contains only data that pertains to a single entity or concept and that the relationships between tables are established through the use of primary and foreign keys.

Overall, to avoid NULL values and spurious tuples, it is important to properly design and normalize the database. By doing so, we can ensure that the data is consistent, accurate, and reliable, and that it can be easily queried, analyzed, and reported.

- **What is the use of Rollback Statement and how to manage it while data recovery**

The rollback statement is used in database management systems to undo a transaction that has not been completed successfully. When a transaction is executed in a database, it may make changes to the data that are not permanent until the transaction is committed. If the transaction fails for some reason, the database may need to be rolled back to its previous state before the transaction was executed.

The rollback statement is used to undo any changes that were made during the failed transaction and return the database to its previous state. This helps to ensure data integrity and consistency by preventing incomplete or incorrect data from being stored in the database.

When managing data recovery, it is important to ensure that the rollback statement is used appropriately to avoid data loss or corruption. One approach to managing data recovery is to use backup and recovery procedures that include regular backups of the database and transaction logs. This allows the database to be restored to a previous point in time if necessary.

In addition, it is important to implement a system for monitoring and managing database transactions to ensure that they are executed properly and that any failed transactions are rolled back appropriately. This may involve setting up alerts or notifications to alert database administrators of failed transactions or other issues that may impact data integrity.

Overall, managing the rollback statement is an important part of ensuring data integrity and consistency in a database management system. By using appropriate backup and recovery procedures and monitoring database transactions, administrators can help to prevent data loss or corruption and ensure that the database remains reliable and accurate.

- **How to manage users by assigning different privileges?**

Managing users by assigning different privileges involves controlling access to the database and determining what actions each user is allowed to perform. This helps to ensure that the database is secure and that data is only accessed and modified by authorized users.

To manage users and assign privileges, the following steps can be taken:

1. Create user accounts: Create a user account for each user who will be accessing the database. This involves assigning a username and password for each user.
2. Define roles: Define roles that correspond to different levels of access to the database. For example, you may create a "read-only" role that allows users to view data but not modify it, and a "read-write" role that allows users to view and modify data.
3. Assign privileges: Assign privileges to each user based on their role. For example, users in the "read-only" role may only be allowed to view data, while users in the "read-write" role may be allowed to view and modify data.

4. Test privileges: Test the privileges assigned to each user to ensure that they are working correctly. This involves logging in as each user and attempting to perform the actions they are authorized to perform.
5. Monitor activity: Monitor database activity to ensure that users are only accessing and modifying data that they are authorized to access. This may involve setting up alerts or notifications to alert database administrators of unauthorized access attempts or other suspicious activity.

Overall, managing users by assigning different privileges is an important part of database security. By carefully controlling access to the database and monitoring user activity, administrators can help to ensure that data is only accessed and modified by authorized users and that the database remains secure and reliable.

- **what is meant by the concurrent execution of database transactions in a multiuser system? Discuss why concurrency control is needed, and give informal examples.**

Concurrent execution of database transactions in a multiuser system refers to the ability of multiple users or applications to access and modify data in the database at the same time. In a multiuser system, it is common for multiple users or applications to be accessing and modifying data simultaneously, which can create issues with data consistency and integrity.

Concurrency control is needed to ensure that data is accessed and modified in a way that maintains consistency and integrity. Without concurrency control, multiple transactions may attempt to modify the same data simultaneously, leading to conflicts and data inconsistencies. For example, if two users attempt to update the same record in a database at the same time, one user's changes may overwrite the other user's changes, resulting in data loss or corruption.

To prevent these issues, concurrency control mechanisms are used to ensure that transactions are executed in a way that maintains data consistency and integrity. These mechanisms may include locking, timestamping, or optimistic concurrency control.

Informal examples of the need for concurrency control in a multiuser system include:

1. A banking system that allows multiple users to access their accounts simultaneously. Without concurrency control, it would be possible for two users to attempt to withdraw the same amount of money from their account at the same time, resulting in an overdraft or a loss of funds.
2. A social media platform that allows multiple users to post and interact with content simultaneously. Without concurrency control, it would be possible for two users to attempt to edit the same post at the same time, resulting in conflicting changes or data inconsistencies.

Overall, concurrency control is essential in a multiuser system to ensure that transactions are executed in a way that maintains data consistency and integrity, and prevents conflicts or data loss.

- **Discuss the actions taken by the read_item and write_item operations on a database.**

The `read_item` and `write_item` operations are two of the most fundamental operations in a database system. These operations are used to retrieve and store data in a database, respectively. Here's a brief discussion of each operation:

1. `read_item`:

The `read_item` operation is used to retrieve data from a database. When a user or application makes a request to read a specific item from a database, the database management system (DBMS) checks if the requested item is available in the database. If the item is present, the DBMS reads the data from the database and returns it to the user or application.

Here are the actions taken by the `read_item` operation:

- The user or application sends a read request to the DBMS, along with the identifier of the item to be read.
- The DBMS checks if the item is available in the database.
- If the item is found, the DBMS retrieves the data and returns it to the user or application.
- If the item is not found, the DBMS returns an error message to the user or application.

2. `write_item`:

The `write_item` operation is used to store data in a database. When a user or application makes a request to write a specific item to a database, the DBMS checks if the item already exists in the database. If the item is present, the DBMS updates the existing data with the new data. If the item is not present, the DBMS creates a new item with the provided data.

Here are the actions taken by the `write_item` operation:

- The user or application sends a write request to the DBMS, along with the identifier of the item and the data to be written.
- The DBMS checks if the item is already present in the database.
- If the item is present, the DBMS updates the data with the new data.
- If the item is not present, the DBMS creates a new item with the provided data and assigns it a new identifier.
- Once the write operation is complete, the DBMS returns a success message to the user or application.

It's worth noting that there are different types of databases, such as relational, NoSQL, and graph databases, and the exact actions taken by the `read_item` and `write_item` operations may vary depending on the database system used.

- **What is the system log used for? What are the typical kinds of records in a system log?**

A system log, also known as a system event log or system log file, is a record of events and messages generated by a computer system or application. System logs are used to provide information about the operation and health of a system, help diagnose problems, and provide a history of activity.

System logs typically contain various kinds of records that provide information about different aspects of the system. Here are some examples of the typical kinds of records found in a system log:

1. System events: These records contain information about events that occur on the system, such as system startups and shutdowns, software installations, hardware changes, and system errors.
2. Security events: These records contain information about security-related events, such as user logins and logouts, failed login attempts, and system access attempts.
3. Application events: These records contain information about events related to specific applications running on the system, such as errors and warnings generated by the application.
4. Performance metrics: These records contain information about the performance of the system, such as CPU usage, memory usage, and disk usage.
5. Audit logs: These records contain information about changes made to the system, such as changes to system configurations and file access.

System logs can be used by system administrators, developers, and security personnel to monitor and manage the system. By analyzing system logs, they can identify and troubleshoot problems, improve system performance, and enhance system security.

- **What is a serial schedule? What is a serializable schedule? Why is a serial schedule considered correct? Why a serializable schedule is considered correct?**

A serial schedule is a sequential execution of transactions in a database, where each transaction is executed one at a time without any concurrent execution of transactions. In other words, a serial schedule is a schedule in which transactions are executed one after the other, without any interleaving of their operations.

A serializable schedule is a schedule that is equivalent to some serial schedule, even though the transactions in the schedule are executed concurrently. In other words, a serializable schedule ensures the same final result as if the transactions were executed serially.

A serial schedule is considered correct because it guarantees that the database is always in a consistent state after each transaction. This is because transactions are executed in a serialized manner, which means that there are no conflicts between the operations of different transactions. As a result, there is no need to ensure any concurrency control mechanisms in a serial schedule, since the transactions are already executed one at a time.

A serializable schedule is also considered correct because it ensures that the database remains in a consistent state after each transaction, even if the transactions are executed concurrently. This is because a serializable schedule guarantees that the final result of the transactions is the same as if they were executed serially. In other words, a serializable schedule ensures that the concurrent execution of transactions does not violate the consistency of the database.

In summary, a serial schedule is considered correct because it guarantees consistency of the database without the need for concurrency control mechanisms, while a serializable schedule is considered correct because it ensures consistency even when transactions are executed concurrently.

- **Define first, second, and third normal forms when only primary keys are considered.**

First normal form (1NF) is a property of a relation in a database, which states that each attribute of the relation should contain only atomic (indivisible) values. In other words, each attribute should contain only a single value, not a collection of values or multiple values. When considering only primary keys, 1NF ensures that each primary key is unique and cannot be broken down into smaller components.

Second normal form (2NF) is a property of a relation in a database, which states that each non-primary key attribute of the relation should be functionally dependent on the entire primary key. In other words, each non-primary key attribute should be uniquely determined by the entire primary key, not just part of it. When considering only primary keys, 2NF ensures that each primary key uniquely identifies every row in the table, and that there are no partial dependencies where only part of the primary key is needed to identify a row.

Third normal form (3NF) is a property of a relation in a database, which states that each non-primary key attribute of the relation should not depend on any other non-primary key attribute. In other words, each non-primary key attribute should be determined solely by the primary key. When considering only primary keys, 3NF ensures that there are no transitive dependencies where a non-primary key attribute depends on another non-primary key attribute, instead of depending directly on the primary key.

In summary, when only primary keys are considered, 1NF ensures that each primary key is unique, 2NF ensures that each primary key uniquely identifies every row in the table, and that there are no partial dependencies, and 3NF ensures that there are no transitive dependencies where a non-primary key attribute depends on another non-primary key attribute.

- **What are transaction commit points, and why are they important?**

Transaction commit points are the points at which a transaction is considered to be successfully completed and changes made by the transaction are made permanent in the database. When a transaction reaches a commit point, it indicates that all of its operations have been successfully executed and that the database has been updated accordingly.

Transaction commit points are important because they help to ensure the consistency and integrity of the database. By committing changes to the database only when a transaction has been successfully completed, it helps to prevent partial updates or inconsistent states. In other words, transaction commit points provide a way to ensure that changes made to the database are atomic, meaning they are either completely executed or completely rolled back.

Additionally, transaction commit points can also help to improve the performance of the database by allowing multiple transactions to be executed concurrently. When multiple transactions are executing at the same time, it is important to ensure that their changes to the database are isolated from each other and that they do not interfere with each other. By using transaction commit points, the database can ensure that each transaction is executed atomically and that changes made by one transaction do not affect the outcome of another transaction.

In summary, transaction commit points are important because they help to ensure the consistency and integrity of the database, enable atomic updates to the database, and allow multiple transactions to be executed concurrently without interfering with each other.

- **What are the difference types of constraints used in SQL?**

Constraints in SQL are rules that are applied to a table to ensure the integrity of the data being stored. The following are some of the different types of constraints used in SQL:

1. NOT NULL Constraint: This constraint ensures that a column cannot have a NULL value. It specifies that the column must have a value.
2. UNIQUE Constraint: This constraint ensures that a column or a group of columns in a table have unique values. It specifies that the values in the column(s) must be unique.
3. PRIMARY KEY Constraint: This constraint is used to uniquely identify each row in a table. It specifies that a column or a group of columns in a table are used to uniquely identify each row. This constraint is used to enforce entity integrity.
4. FOREIGN KEY Constraint: This constraint is used to establish a relationship between two tables. It specifies that the values in a column or a group of columns in one table are used to match the values in a column or a group of columns in another table.
5. CHECK Constraint: This constraint is used to check if the values in a column meet a specific condition. It specifies a condition that the values in the column must meet.
6. DEFAULT Constraint: This constraint is used to provide a default value for a column when no value is specified. It specifies a default value that will be used when no value is provided for the column.

In summary, the different types of constraints used in SQL are NOT NULL, UNIQUE, PRIMARY KEY, FOREIGN KEY, CHECK, and DEFAULT constraints.

- **What is a schedule? Define the concepts of recoverable, cascade less, and strict schedules, and compare them in terms of their recoverability**

In database systems, a schedule is a chronological sequence of transactions that operate on a set of data items. It specifies the order in which the transactions are executed and the data items that are accessed and updated by each transaction. A schedule can be represented as a series of read and write operations performed by the transactions.

Recoverable Schedule: A schedule is recoverable if, in case of a transaction failure, it is possible to restore the database to a state that existed before the failed transaction started. In other words, if a transaction T1 modifies a data item that is later read and used by another transaction T2, then T2 should not be allowed to commit until T1 has committed. If T1 fails before committing, then T2 must be rolled back to avoid reading inconsistent data.

Cascading Rollback (Cascadeless) Schedule: A schedule is cascadeless if, in case of a transaction failure, the effects of the failed transaction do not propagate to other transactions. In other words, if a transaction T1 modifies a data item that is later read and used by another transaction T2, then T2 should not be allowed to commit until T1 has committed. If T1 fails before committing, then T2 can still commit because it has not been affected by the failed transaction.

Strict Schedule: A schedule is strict if, in case of a transaction failure, no transaction can read or write data that has been modified by the failed transaction. In other words, if a transaction T1 modifies a data item that is later read and used by another transaction T2, then T2 should not be allowed to commit until T1 has committed. If T1 fails before committing, then T2 must also be rolled back because it has read data that was modified by the failed transaction.

In terms of recoverability, a strict schedule is the most recoverable, followed by a recoverable schedule, and then a cascadeless schedule. This is because a strict schedule ensures that no transaction reads or writes data that has been modified by a failed transaction, while a recoverable schedule ensures that the effects of a failed transaction can be undone without affecting other transactions, and a cascadeless schedule only ensures that the effects of a failed transaction do not propagate to other transactions. Therefore, a strict schedule provides the highest level of recoverability, but may result in more rollbacks, while a cascadeless schedule provides the lowest level of recoverability, but may result in fewer rollbacks.