**Module: 4 DBMS**

**Introduction to SQL**

**Que: 1 What is SQL, and why is it essential in database management?**

Ans: SQL, or Structured Query Language, is a specialized programming language used to manage and manipulate data in relational databases. It's the standard language for interacting with Relational Database Management Systems (RDBMS) like MySQL, PostgreSQL, Oracle, and Microsoft SQL Server.

**Why is SQL essential in database management?**

SQL is absolutely crucial in database management for several key reasons:

1. **Universal Language for Relational Databases:** SQL is the industry standard. This means that once you learn SQL, you can apply your knowledge to virtually any relational database system, making it a highly transferable and in-demand skill. This standardization ensures consistency and interoperability across different platforms.
2. **Data Definition (DDL):** SQL provides commands (Data Definition Language - DDL) to define and manage the structure of your database. You can:

* **CREATE** databases, tables, views, and indexes.
* **ALTER** existing database objects to change their structure.
* **DROP** database objects when they are no longer needed.

1. **Data Manipulation (DML):** SQL allows you to interact with the data itself (Data Manipulation Language - DML). You can:

* **INSERT** new data records into tables.
* **UPDATE** existing records.
* **DELETE** records from tables.
* **SELECT** data to retrieve specific information. This is perhaps the most frequently used aspect of SQL, allowing for powerful querying and analysis.

1. **Data Querying and Analysis:** SQL excels at retrieving specific data from large datasets. Its declarative nature means you tell the database what you want, not how to get it. This simplifies complex queries and allows users to:

* Filter data based on various criteria.
* Sort results.
* Aggregate data (e.g., sums, counts, averages).
* Join data from multiple tables to get a comprehensive view. This is fundamental to relational databases, allowing you to link related information across different tables (e.g., connecting customer information with their orders).

1. **Data Integrity and Security (DCL & TCL):** SQL helps maintain the accuracy, consistency, and security of data:

* Data Control Language (DCL): Commands like GRANT and REVOKE allow database administrators to manage user access permissions, ensuring that only authorized individuals or applications can view, modify, or delete data.
* Transaction Control Language (TCL): Commands like COMMIT and ROLLBACK ensure data consistency during transactions, preventing data corruption in case of errors or system failures. SQL enforces constraints (like primary keys and foreign keys) to ensure data validity and relationships between tables.

1. **Efficiency and Scalability:** SQL databases are designed to handle complex queries and large volumes of data efficiently. They can scale to manage growing data needs, from small applications to enterprise-level systems. Query optimization techniques and indexing further enhance performance.
2. **Integration:** SQL integrates seamlessly with various programming languages (e.g., Java, Python, C#), tools, and platforms, making it a versatile solution for building high-performance data processing applications and integrating with business intelligence systems.

In essence, SQL provides the fundamental framework and tools for organizing, accessing, and managing the vast amounts of structured data that drive modern applications and businesses. Without SQL, managing and extracting value from relational databases would be incredibly complex, inefficient, and prone to errors.

**Que: 2 Explain the difference between DBMS and RDBMS.**

**Ans:** A **Database Management System (DBMS)** and a **Relational Database Management System (RDBMS)** are both software systems designed to manage databases, but they differ significantly in structure, capabilities, and use cases. Here's a detailed comparison:

**Data Storage and Structure**

* **DBMS**: Stores data as files, often in hierarchical or navigational formats. It lacks a structured relationship between data elements.
* **RDBMS**: Stores data in tables consisting of rows and columns, adhering to the relational model. This structure allows for defining relationships between tables using keys.

### Data Relationships

* **DBMS**: Does not inherently support relationships between data elements.
* **RDBMS**: Supports relationships between tables through keys (primary and foreign keys), enabling complex data interconnections.

### User Support

* **DBMS**: Typically supports a single user at a time.
* **RDBMS**: Designed to support multiple users simultaneously, facilitating concurrent data access and manipulation.

### Security and Integrity

* **DBMS**: Offers minimal security features and lacks robust data integrity constraints.
* **RDBMS**: Provides advanced security measures and enforces data integrity through constraints like primary keys, foreign keys, and unique constraints.

### Scalability and Performance

* **DBMS**: Suitable for small-scale applications with limited data requirements. Performance may degrade with increased data volume.
* **RDBMS**: Optimized for handling large volumes of data efficiently. Supports scalability and complex queries, ensuring high performance even with extensive datasets.

### Normalization

* **DBMS**: Does not support normalization, leading to potential data redundancy.
* **RDBMS**: Supports normalization processes to eliminate data redundancy and ensure data consistency.

### Architecture

* **DBMS**: Does not typically support client-server architecture.
* **RDBMS**: Supports client-server architecture, enabling distributed database management and remote access.

### Examples

* **DBMS**: Examples include XML, Windows Registry, and FoxPro.
* **RDBMS**: Examples include MySQL, PostgreSQL, Oracle, SQL Server, and Microsoft Access.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | | Feature | DBMS | RDBMS | | Data Storage | Files (hierarchical/navigational) | Tables (rows and columns) | | Data Relationships | Not supported | Supported via keys | | User Support | Single user | Multiple users | | Security & Integrity | Minimal | Advanced (constraints, ACID compliance) | | Scalability | Limited | High | | Normalization | Not supported | Supported | | Architecture | Standalone | Client-server | | Examples | XML, Windows Registry, FoxPro | MySQL, PostgreSQL, Oracle, SQL Server | |

**Que: 3 Describe the role of SQL in managing relational databases.**

**Ans:** SQL (Structured Query Language) plays an absolutely central and indispensable role in managing relational databases. It's the primary language through which users and applications interact with a Relational Database Management System (RDBMS). Think of it as the command center for all operations within a relational database.

Here's a breakdown of SQL's key roles in managing relational databases:

**1. Data Definition (DDL - Data Definition Language)**

SQL provides commands to define and manage the *structure* of the database and its objects. This is like designing the blueprint for your data storage.

* **CREATE**: Used to build new database objects.
  + **CREATE DATABASE:** To create a new database.
  + **CREATE TABLE:** To define new tables, specifying column names, data types, and constraints (e.g., **PRIMARY KEY, FOREIGN KEY, NOT NULL, UNIQUE**).
  + **CREATE VIEW:** To create virtual tables based on the result set of a SELECT query, offering a simplified or restricted view of the underlying data.
  + **CREATE INDEX:** To create indexes on columns, which significantly speed up data retrieval operations.
* **ALTER**: Used to modify the structure of existing database objects.
  + **ALTER TABLE**: To add, modify, or drop columns, or to add/remove constraints from a table.
* **DROP**: Used to delete existing database objects.
  + **DROP TABLE**: To remove a table and all its data.
  + **DROP DATABASE**: To delete an entire database.

**2. Data Manipulation (DML - Data Manipulation Language)**

Once the database structure is in place, SQL provides commands to manage the *data itself* within those structures. This is where you put data in, change it, and take it out.

* **INSERT**: Used to add new rows (records) of data into a table.
  + **INSERT INTO Customers (CustomerID, Name) VALUES (1, 'Alice');**
* **UPDATE**: Used to modify existing data in one or more rows of a table.
  + **UPDATE Products SET Price = 25.00 WHERE ProductID = 101;**
* **DELETE**: Used to remove one or more rows from a table.
  + **DELETE FROM Orders WHERE OrderID = 500;**
* **SELECT (Data Query Language - DQL)**: While often grouped with DML, SELECT is so fundamental it sometimes gets its own category (DQL). It's used to retrieve data from the database. This is SQL's most powerful and frequently used capability.
  + **SELECT \* FROM Employees;** (Retrieve all data from the Employees table)
  + **SELECT Name, Email FROM Customers WHERE City = 'New York';** (Retrieve specific columns for customers in New York)
  + **SELECT ProductName, SUM(Quantity) FROM OrderDetails GROUP BY ProductName HAVING SUM(Quantity) > 100;** (Complex queries with aggregation and filtering)
  + **SELECT Orders.OrderID, Customers.Name FROM Orders JOIN Customers ON Orders.CustomerID = Customers.CustomerID;** (Combine data from multiple related tables using JOINs)

**3. Data Control (DCL - Data Control Language)**

SQL handles security and permissions within the database, determining who can access what and what they can do with it.

* **GRANT**: Used to give specific privileges to users or roles (e.g., SELECT on a table, INSERT into another).
  + **GRANT SELECT ON Products TO 'SalesUser';**
* **REVOKE**: Used to remove previously granted privileges.
  + **REVOKE INSERT ON Orders FROM 'GuestUser';**

**4. Transaction Control (TCL - Transaction Control Language)**

SQL allows for managing transactions, which are sequences of operations performed as a single logical unit. This ensures data consistency and integrity, especially in multi-user environments.

* **COMMIT**: Saves all changes made during the current transaction permanently to the database.
* **ROLLBACK**: Undoes all changes made during the current transaction, restoring the database to its state before the transaction began.
* **SAVEPOINT**: Sets a point within a transaction to which you can later roll back.

**5. Ensuring Data Integrity**

SQL is critical for maintaining the accuracy and consistency of data within the relational model:

* **Constraints**: SQL allows you to define various constraints (e.g., PRIMARY KEY, FOREIGN KEY, UNIQUE, NOT NULL, CHECK) directly within table definitions. These constraints automatically enforce business rules and relationships, preventing invalid data from being entered.
* **Referential Integrity**: Through FOREIGN KEY constraints, SQL ensures that relationships between tables are maintained (e.g., you can't delete a customer if there are still orders associated with them, unless specific cascading rules are defined).

**6. Performance Optimization**

While not strictly "management" of data in terms of its content, SQL also plays a role in optimizing how the database performs:

* **Indexes**: Creating indexes with SQL (using CREATE INDEX) can dramatically speed up data retrieval for frequently queried columns.
* **Query Optimization**: The way SQL queries are written can significantly impact performance. Database administrators and developers use their knowledge of SQL and database internals to write efficient queries.

In summary, SQL is the fundamental language for relational databases because it provides a comprehensive set of commands and features to:

* **Define** the structure of the database.
* **Manipulate** the data stored within it.
* **Control** access and security.
* **Manage transactions** for data consistency.
* **Enforce data integrity** rules.

Without SQL, the power and structure of relational databases would be largely untapped, making data management chaotic and inefficient.

**Que: 4 What are the key features of SQL?**

SQL's power and widespread adoption stem from its robust set of features, making it the de facto language for managing relational databases. Here are the key features of SQL:

1. **Declarative Language:**
   * **Focus on "What," not "How":** Unlike procedural languages where you specify step-by-step instructions, SQL is declarative. You tell the database *what* data you want to retrieve or *what* changes you want to make, and the RDBMS figures out the most efficient way to perform the operation. This simplifies complex operations for users.
2. **Standardization (ANSI/ISO Standard):**
   * SQL is an official standard (ANSI and ISO). While different RDBMS vendors (MySQL, Oracle, SQL Server, PostgreSQL) have their own extensions, the core SQL syntax and functionalities are largely consistent. This means knowledge of SQL is highly transferable across various database systems.
3. **Comprehensive Data Management Capabilities:**
   * **Data Definition Language (DDL):** For defining the database structure.
     + CREATE (databases, tables, views, indexes, stored procedures, etc.)
     + ALTER (modify existing structures)
     + DROP (delete existing structures)
     + TRUNCATE (remove all rows from a table, but keep the structure)
     + RENAME (change the name of objects)
   * **Data Manipulation Language (DML):** For managing the data within the defined structures.
     + SELECT (retrieve data – also known as Data Query Language, DQL)
     + INSERT (add new records)
     + UPDATE (modify existing records)
     + DELETE (remove records)
   * **Data Control Language (DCL):** For managing permissions and access.
     + GRANT (give users/roles permissions)
     + REVOKE (remove permissions)
   * **Transaction Control Language (TCL):** For managing database transactions to ensure data integrity.
     + COMMIT (save changes)
     + ROLLBACK (undo changes)
     + SAVEPOINT (set a point within a transaction to roll back to)
4. **Data Integrity Enforcement:**
   * **Constraints:** SQL allows defining rules that data must adhere to, ensuring accuracy and consistency.
     + PRIMARY KEY: Uniquely identifies each row in a table.
     + FOREIGN KEY: Enforces referential integrity between tables.
     + UNIQUE: Ensures all values in a column are distinct.
     + NOT NULL: Ensures a column cannot contain NULL values.
     + CHECK: Ensures all values in a column satisfy a specific condition.
   * **ACID Properties:** RDBMS systems, managed by SQL, are designed to adhere to ACID properties (Atomicity, Consistency, Isolation, Durability) for transactions, guaranteeing reliable data handling.
5. **Powerful Querying and Analysis:**
   * **Filtering (WHERE):** Retrieve specific data based on conditions.
   * **Sorting (ORDER BY):** Arrange results in a specific order.
   * **Grouping (GROUP BY):** Aggregate data based on common values.
   * **Aggregation Functions (SUM, COUNT, AVG, MIN, MAX):** Perform calculations on groups of data.
   * **Joining Tables (JOIN):** Combine data from multiple related tables (e.g., INNER JOIN, LEFT JOIN, RIGHT JOIN, FULL OUTER JOIN). This is a cornerstone of the relational model.
   * **Subqueries:** Nest queries within other queries for complex data retrieval.
   * **Set Operations (UNION, INTERSECT, EXCEPT):** Combine results from multiple SELECT statements.
6. **Security and Access Control:**
   * SQL provides robust mechanisms to control who can access what data and what operations they can perform (e.g., read-only access for some users, full modification rights for others).
7. **Client-Server Architecture:**
   * SQL databases typically operate in a client-server model, where SQL commands are sent from client applications to the database server, which processes them and returns results.
8. **Scalability and Performance:**
   * SQL-based RDBMS are designed to handle large volumes of data and a high number of concurrent users. Features like indexing (created via SQL) and query optimization (managed by the RDBMS based on SQL queries) contribute to high performance.
9. **Embedded SQL and API Integration:**
   * SQL commands can be embedded within other programming languages (like Python, Java, C#, PHP) using various APIs (e.g., JDBC for Java, ODBC for C++, psycopg2 for Python) to build dynamic, database-driven applications.
10. **English-like Syntax:**
    * SQL uses a relatively simple, intuitive syntax with keywords that resemble English, making it easier to learn and understand compared to many other programming languages.

These features collectively make SQL an indispensable tool for anyone working with structured data, from developers and database administrators to data analysts and business intelligence professionals.

**SQL Syntax**

**Que: 1 What are the basic components of SQL syntax?**

The basic components of SQL syntax are the building blocks that allow you to construct queries and commands to interact with a relational database. While SQL can get very complex with advanced clauses and functions, these fundamental components are present in almost every SQL statement.

Here are the basic components of SQL syntax, often categorized by their function:

### 1. Clauses

Clauses are the keywords that introduce or define specific parts of an SQL statement. They typically start with a keyword.

* **SELECT**: The most fundamental clause, used to specify which columns you want to retrieve from the database. It's almost always the first clause in a query.
  + Example: SELECT CustomerName, City
* **FROM**: Specifies the table (or tables) from which to retrieve data.
  + Example: FROM Customers
* **WHERE**: Filters records based on a specified condition. Only rows that meet the condition are included in the result.
  + Example: WHERE City = 'New York'
* **GROUP BY**: Groups rows that have the same values in specified columns into summary rows, often used with aggregate functions.
  + Example: GROUP BY Country
* **HAVING**: Filters groups based on a specified condition. It's used with GROUP BY, similar to how WHERE filters individual rows.
  + Example: HAVING COUNT(OrderID) > 5
* **ORDER BY**: Sorts the result set by one or more columns in ascending (ASC) or descending (DESC) order.
  + Example: ORDER BY CustomerName ASC
* **LIMIT / TOP / ROWNUM**: Restricts the number of rows returned by the query. (Syntax varies by RDBMS: LIMIT in MySQL/PostgreSQL, TOP in SQL Server, ROWNUM in Oracle).
  + Example (MySQL): LIMIT 10

### 2. Keywords

Keywords are reserved words that perform specific actions in SQL. Many clauses are keywords themselves.

* **Data Definition Language (DDL) Keywords:**
  + CREATE, ALTER, DROP, TABLE, DATABASE, INDEX, VIEW
* **Data Manipulation Language (DML) Keywords:**
  + INSERT INTO, VALUES, UPDATE, SET, DELETE FROM
* **Data Control Language (DCL) Keywords:**
  + GRANT, REVOKE
* **Transaction Control Language (TCL) Keywords:**
  + COMMIT, ROLLBACK, SAVEPOINT
* **Logical Operators:** AND, OR, NOT
* **Comparison Operators:** =, != (or <>), >, <, >=, <=
* **Pattern Matching:** LIKE (used with % and \_ wildcards)
* **Range:** BETWEEN
* **List:** IN
* **Null Check:** IS NULL, IS NOT NULL

### 3. Operators

Symbols or keywords used in the WHERE, HAVING, or SET clauses to perform comparisons, logical operations, or calculations.

* **Arithmetic Operators:** +, -, \*, /, % (modulus)
* **Comparison Operators:** =, != (or <>), >, <, >=, <=
* **Logical Operators:** AND, OR, NOT
* **Set Operators:** UNION, UNION ALL, INTERSECT, EXCEPT (or MINUS in Oracle)

### 4. Functions

Predefined routines that perform specific calculations or manipulations on data.

* **Aggregate Functions:** Operate on a set of rows and return a single summary value.
  + COUNT(): Number of rows.
  + SUM(): Sum of values.
  + AVG(): Average of values.
  + MIN(): Minimum value.
  + MAX(): Maximum value.
* **Scalar Functions:** Operate on a single value and return a single value.
  + UCASE() / UPPER(): Convert to uppercase.
  + LCASE() / LOWER(): Convert to lowercase.
  + LENGTH() / LEN(): Length of a string.
  + ROUND(): Round a number.
  + NOW() / GETDATE(): Current date and time.

### 5. Identifiers

Names given to database objects.

* **Table Names:** Customers, Products, Orders
* **Column Names:** CustomerID, ProductName, OrderDate, Price
* **Database Names:** my\_database
* **View Names, Index Names, Stored Procedure Names**, etc.

### 6. Expressions

Combinations of columns, values, operators, and functions that evaluate to a single value.

* Price \* Quantity (arithmetic expression)
* CustomerName + ' ' + City (string concatenation)
* OrderDate + INTERVAL '7 DAY' (date expression)

### 7. Data Types

Specify the type of data that can be stored in a column. This is fundamental during table creation.

* **Numeric:** INT, DECIMAL, FLOAT, DOUBLE, BIGINT
* **String:** VARCHAR, CHAR, TEXT
* **Date/Time:** DATE, TIME, DATETIME, TIMESTAMP
* **Boolean:** BOOLEAN (or TINYINT(1) in some systems)
* **Binary Large Object:** BLOB, VARBINARY

### 8. Comments

Used to add explanatory notes within SQL code, ignored by the database parser.

* Single-line comment: -- This is a comment
* Multi-line comment: /\* This is a multi-line comment \*/

### Example of Basic SQL Syntax in a SELECT Statement:

**SELECT**

ProductName, -- Column Name (Identifier)

Price, -- Column Name (Identifier)

Quantity, -- Column Name (Identifier)

Price \* Quantity AS LineTotal -- Expression with arithmetic operator and alias

**FROM**

OrderDetails -- Table Name (Identifier)

**WHERE**

Quantity > 5 -- WHERE Clause with comparison operator

AND Price <= 100 -- Logical operator AND, comparison operator

**ORDER BY**

LineTotal DESC; -- ORDER BY Clause with sorting direction

**Que : 2 Write the general structure of an SQL SELECT statement.**

**Ans :** The general structure of an SQL SELECT statement is designed to retrieve data from one or more tables in a database. While not all clauses are mandatory, they can be combined to form a comprehensive query. Here's the standard syntax:

General Syntax of an SQL SELECT Statement

|  |
| --- |
| SELECT DISTINCT first\_name, last\_name, COUNT(order\_id) AS order\_count  FROM customers  JOIN orders ON customers.customer\_id = orders.customer\_id  WHERE order\_date >= '2025-01-01'  GROUP BY first\_name, last\_name  HAVING COUNT(order\_id) > 5  ORDER BY order\_count DESC  LIMIT 10; |

### Explanation of Each Clause

1. **SELECT**: Specifies the columns to retrieve. Use \* to select all columns.
2. **DISTINCT**: Optional. Eliminates duplicate rows from the result set.
3. **FROM**: Identifies the table(s) from which to retrieve data.
4. **AS alias**: Optional. Assigns a temporary name to a table or column for the duration of the query.
5. **JOIN**: Combines rows from two or more tables based on a related column. Common types include INNER JOIN, LEFT JOIN, RIGHT JOIN, and FULL JOIN.
6. **WHERE**: Filters rows based on a specified condition.
7. **GROUP BY**: Groups rows sharing a property so that an aggregate function can be applied to each group.
8. **HAVING**: Filters groups based on a specified condition, similar to WHERE but applied after GROUP BY.
9. **ORDER BY**: Sorts the result set by one or more columns, either ascending (ASC) or descending (DESC).
10. **LIMIT**: Restricts the number of rows returned.

**Que : 3 Explain the role of clauses in SQL statements.**

**Ans :** In SQL, **clauses** are the fundamental components that define the structure and behavior of a statement. Each clause serves a specific purpose, allowing you to perform various operations such as retrieving, filtering, grouping, and sorting data. Understanding the role of each clause is essential for constructing effective and efficient SQL queries.

## **Role of Clauses in SQL Statements**

### 1. SELECT ****Clause****

* **Purpose**: Specifies the columns to retrieve from the database.
* **Example**: SELECT first\_name, last\_name FROM employees;
* **Function**: Determines the data that will appear in the result set.

### 2. FROM ****Clause****

* **Purpose**: Indicates the table(s) from which to retrieve the data.
* **Example**: FROM employees;
* **Function**: Defines the data source for the query.

### 3. WHERE ****Clause****

* **Purpose**: Filters rows based on specified conditions.
* **Example**: WHERE department = 'Sales';
* **Function**: Limits the rows returned to those that meet the condition.

### 4. JOIN ****Clause****

* **Purpose**: Combines rows from two or more tables based on a related column.
* **Example**: JOIN departments ON employees.department\_id = departments.id;
* **Function**: Enables retrieval of related data spread across multiple tables.

### 5. GROUP BY ****Clause****

* **Purpose**: Groups rows that have the same values in specified columns.
* **Example**: GROUP BY department;
* **Function**: Allows aggregate functions like COUNT(), SUM(), AVG() to be applied to each group.

### 6. HAVING ****Clause****

* **Purpose**: Filters groups based on a specified condition.
* **Example**: HAVING COUNT(\*) > 5;
* **Function**: Similar to WHERE, but operates on groups created by GROUP BY.

### 7. ORDER BY ****Clause****

* **Purpose**: Sorts the result set by one or more columns.
* **Example**: ORDER BY last\_name ASC;
* **Function**: Determines the order in which rows appear in the result set. 8. LIMIT **Clause**
* **Purpose**: Restricts the number of rows returned by the query.
* **Example**: LIMIT 10;
* **Function**: Useful for pagination or limiting results to a specific number.

**Example SQL Query**

|  |
| --- |
| SELECT department, COUNT(\*) AS employee\_count  FROM employees  WHERE status = 'Active'  GROUP BY department  HAVING COUNT(\*) > 5  ORDER BY employee\_count DESC  LIMIT 5; |

**SQL Constraints**

**Que : 1 What are constraints in SQL? List and explain the different types of constraints**

**Ans :** In SQL, **constraints** are rules applied to table columns to ensure the integrity, accuracy, and reliability of the data. They define the permissible values and relationships within the database, helping maintain consistent and valid data.

**Types of SQL Constraints**

### ****NOT NULL****

* **Purpose**: Ensures that a column cannot have a NULL value.
* **Use Case**: Essential for columns that require a value for every row, such as ID or Name.

**Example**:

|  |
| --- |
| CREATE TABLE Students (  student\_id INT NOT NULL,  name VARCHAR(100) NOT NULL  ); |

### ****UNIQUE****

* **Purpose**: Ensures all values in a column are distinct across the table.
* **Use Case**: Useful for columns where duplicate values are not allowed, like email addresses.
* **Note**: Unlike PRIMARY KEY, UNIQUE allows NULL values but ensures that non-NULL values are unique.

**Example**:

|  |
| --- |
| CREATE TABLE Employees (  employee\_id INT,  email VARCHAR(100) UNIQUE  ); |

### ****PRIMARY KEY****

* **Purpose**: Uniquely identifies each record in a table. Combines NOT NULL and UNIQUE constraints.
* **Use Case**: Typically used for columns like ID that serve as the unique identifier for records.
* **Note**: A table can have only one PRIMARY KEY.

**Example**:

|  |
| --- |
| CREATE TABLE Products (  product\_id INT PRIMARY KEY,  name VARCHAR(100)  ); |

### ****FOREIGN KEY****

* **Purpose**: Establishes a link between two tables by ensuring that a column's value matches a valid value in another table's PRIMARY KEY or UNIQUE column.
* **Use Case**: Maintains referential integrity between related tables.

**Example**:

|  |
| --- |
| CREATE TABLE Orders (  order\_id INT PRIMARY KEY,  product\_id INT,  FOREIGN KEY (product\_id) REFERENCES Products(product\_id)  ); |

### ****CHECK****

* **Purpose**: Ensures that all values in a column satisfy a specific condition.
* **Use Case**: Validates data based on a specified rule, such as age being greater than 18.

**Example**:

|  |
| --- |
| CREATE TABLE Employees (  employee\_id INT PRIMARY KEY,  age INT CHECK (age >= 18)  ); |

### ****DEFAULT****

* **Purpose**: Provides a default value for a column when no value is specified during an insert operation.
* **Use Case**: Assigns a standard value to a column if the user does not provide one.

**Example**:

|  |
| --- |
| CREATE TABLE Employees (  employee\_id INT PRIMARY KEY,  hire\_date DATE DEFAULT CURRENT\_DATE  ); |

### ****INDEX**** (Optional)

* **Purpose**: Improves the speed of data retrieval operations on a table at the cost of additional space and maintenance time.
* **Use Case**: Enhances performance for queries that frequently search or sort by specific columns.

**Example**:

|  |
| --- |
| CREATE INDEX idx\_employee\_name ON Employees (name); |

**Que : 2 How do PRIMARY KEY and FOREIGN KEY constraints differ?**

**Ans : 2** In SQL, **Primary Key** and **Foreign Key** are fundamental constraints that serve distinct purposes in relational database design.

**Primary Key vs. Foreign Key**

|  |  |  |
| --- | --- | --- |
| Feature | Primary Key | Foreign Key |
| Purpose | Uniquely identifies each record within its own table. | Establishes a link between two tables, referencing a primary key in another table. |
| Uniqueness | Ensures all values in the column are unique. | Allows duplicate values; multiple rows can reference the same parent record. |
| Null ability | Does not allow NULL values. | Can contain NULL values, indicating optional relationships. |
| Occurrence | Only one primary key is allowed per table. | Multiple foreign keys can exist in a table, each referencing different parent tables. |
| Referential Role | Serves as the target for foreign key references. | Serves as the reference to a primary key in another table. |
| Data Integrity | Enforces entity integrity by ensuring each record is uniquely identifiable. | Enforces referential integrity by ensuring valid relationships between tables. |

Consider two tables: Customers and Orders.

* **Customers Table**:
  + customer\_id (Primary Key)
  + name
  + email
* **Orders Table**:
  + order\_id (Primary Key)
  + order\_date
  + customer\_id (Foreign Key referencing Customers.customer\_id)

In this setup, customer\_id in the Customers table uniquely identifies each customer. In the Orders table, customer\_id is a foreign key that links each order to a specific customer, establishing a relationship between the two tables.

**Que : 3 What is the role of NOT NULL and UNIQUE constraints?**

Ans : In SQL, the **NOT NULL** and **UNIQUE** constraints are pivotal for maintaining data integrity and enforcing business rules within a database. Here's a detailed overview of their roles:

## **NOT NULL Constraint**

**Purpose**: Ensures that a column cannot have a NULL value, enforcing the presence of a value in that column for every record.

**Key Characteristics**:

* **Mandatory Values**: Prevents the insertion of records with missing or undefined data in specified columns.
* **Data Integrity**: Crucial for fields where a missing value would lead to data inconsistency or application errors.
* **Usage**: Commonly applied to columns like ID, Name, or any attribute that must always have a value. [Database Guide](https://database.guide/understanding-the-not-null-constraint-in-sql/?utm_source=chatgpt.com)[GeeksforGeeks+1Database Guide+1](https://www.geeksforgeeks.org/sql-not-null-constraint/?utm_source=chatgpt.com)

**Example**:

|  |
| --- |
| CREATE TABLE Employees ( |
| employee\_id INT PRIMARY KEY, |
| name VARCHAR(100) NOT NULL, |
| email VARCHAR(100) NOT NULL |
| ); |

In this example, both name and email columns are defined with the NOT NULL constraint, ensuring that every employee record includes these details.

## **UNIQUE Constraint**

**Purpose**: Ensures that all values in a column (or a combination of columns) are distinct across the table, preventing duplicate entries.[GeeksforGeeks+1GeeksforGeeks+1](https://www.geeksforgeeks.org/sql-not-null-constraint/?utm_source=chatgpt.com)

**Key Characteristics**:

* **Allows NULLs**: Unlike PRIMARY KEY, the UNIQUE constraint permits NULL values, but multiple NULL entries are allowed, depending on the database system.
* **Multiple Instances**: A table can have multiple UNIQUE constraints, each enforcing uniqueness on different columns or sets of columns.
* **Index Creation**: Typically, a unique index is automatically created to enforce the uniqueness requirement, which can also enhance query performance.

**Example**:

|  |
| --- |
| *CREATE TABLE Users (* |
| *user\_id INT PRIMARY KEY,* |
| *username VARCHAR(50) UNIQUE,* |
| *email VARCHAR(100) UNIQUE* |
| *);* |

In this example, both username and email columns are defined with the UNIQUE constraint, ensuring that each value in these columns is distinct across all user records.

## **Comparison Table**

|  |  |  |
| --- | --- | --- |
| Feature | NOT NULL | UNIQUE |
| **Purpose** | Ensures a column cannot have NULL values | Ensures all values in a column are distinct |
| **Allows NULL?** | ❌ No | ✅ Yes (but multiple NULLs may be allowed) |
| **Multiple per Table** | ✅ Yes (applied to different columns) | ✅ Yes (multiple unique constraints allowed) |
| **Index Creation** | ❌ No | ✅ Yes (typically creates a unique index) |
| **Use Case** | Mandatory fields like Name, Age | Fields like Email, Username |

By strategically applying these constraints, you can enforce data integrity and ensure that your database accurately reflects the business rules and requirements of your application.