### **1. What is SQL? What are its main purposes/applications?**

* **Answer:** SQL (Structured Query Language) is a standard language used to **communicate with and manage relational databases**.
  + **Main Purposes/Applications:**
    - **Querying data:** Retrieving specific information (SELECT).
    - **Manipulating data:** Adding, modifying, or deleting records (INSERT, UPDATE, DELETE).
    - **Defining data structures:** Creating, altering, or dropping database objects like tables (CREATE, ALTER, DROP).
    - **Controlling data access:** Managing user permissions (GRANT, REVOKE).

### **2. DBMS vs. RDBMS**

| **Feature** | **DBMS (Database Management System)** | **RDBMS (Relational Database Management System)** |
| --- | --- | --- |
| **Data Model** | Can use various models (hierarchical, network, file-based, etc.). | Uses the **relational model**, organizing data in tables (relations). |
| **Data Structure** | Data may be stored in files or unstructured ways. | Data is stored in **structured tables** with rows and columns. |
| **Relationships** | May not explicitly enforce relationships between data. | **Enforces relationships** between tables using keys. |
| **Data Integrity** | Less stringent enforcement of data integrity. | **Strong enforcement** of data integrity through constraints. |
| **Complexity** | Simpler, often for smaller, less complex data needs. | More complex, designed for large, intricate datasets. |
| **Normalization** | May not support or require normalization. | Supports and encourages data **normalization**. |
| **Examples** | File systems, XML databases. | MySQL, PostgreSQL, Oracle, SQL Server. |

### **3. PRIMARY KEY and FOREIGN KEY**

* **PRIMARY KEY:**
  + **Uniquely identifies** each row in a table.
  + Must contain **unique values** and **cannot be NULL**.
  + A table can have only **one** primary key.
  + **Purpose:** Ensures data integrity by guaranteeing uniqueness and providing a stable identifier for each record.
* **FOREIGN KEY:**
  + A column (or set of columns) in one table that refers to the **PRIMARY KEY** in another table.
  + Establishes a **link or relationship** between two tables.
  + Can contain duplicate values and **NULL values** (unless specified otherwise).
  + **Purpose:** Enforces **referential integrity**, meaning that a foreign key value must exist as a primary key value in the referenced table, preventing orphaned records.

### **4. Constraint and Its Types**

* **Constraint:** A rule enforced on data columns in a table to limit the type of data that can be entered. They ensure the **accuracy and reliability** of the data.
* **Types of Constraints:**
  + **NOT NULL**: Ensures a column cannot have a NULL value.
  + **UNIQUE**: Ensures all values in a column are different (can have one NULL).
  + **PRIMARY KEY**: A combination of NOT NULL and UNIQUE. Uniquely identifies each row.
  + **FOREIGN KEY**: Links two tables, enforcing referential integrity.
  + **CHECK**: Ensures all values in a column satisfy a specific condition (e.g., age > 18).
  + **DEFAULT**: Provides a default value for a column when no value is specified.

### **5. What are NULL values in SQL? How do you handle them?**

* **Answer:**
  + NULL represents a **missing or unknown value** in a column. It is *not* equivalent to zero or an empty string.
  + **Handling:**
    - Use IS NULL to check for NULL values (e.g., WHERE Email IS NULL).
    - Use IS NOT NULL to check for non-NULL values (e.g., WHERE Email IS NOT NULL).
    - NULL values can affect aggregate functions (COUNT(), SUM(), AVG() often ignore NULLs).
    - Use functions like COALESCE() (standard SQL) or IFNULL() (MySQL) / NVL() (Oracle) to substitute NULL with a default value.

### **6. What are the different types of relationships in a database?**

* **Answer:**
  + **One-to-One (1:1):** Each record in Table A relates to exactly one record in Table B, and vice-versa. (Less common, often indicates denormalization or splitting a table for security/performance).
    - *Example:* Employee (details) and Employee\_Contact\_Info (private details).
  + **One-to-Many (1:N):** One record in Table A can relate to multiple records in Table B, but each record in Table B relates to only one record in Table1 A. (Most common).
    - *Example:* Departments and Employees (one department has many employees).
  + **Many-to-Many (N:M):** Multiple records in Table A can relate to multiple records in Table B, and vice-versa. This relationship is typically resolved using a **junction table** (or associative table).
    - *Example:* Students and Courses (one student can take many courses, one course can have many students).

### **7. SQL Commands in Detail (DQL, DML, DDL, DCL, TCL)**

SQL commands are categorized into five main types:

* **DQL (Data Query Language):**
  + Used for **retrieving data**.
  + **Command:** SELECT (e.g., SELECT \* FROM Employees;)
* **DML (Data Manipulation Language):**
  + Used for **modifying data**.
  + **Commands:** INSERT (add rows), UPDATE (modify existing data), DELETE (remove rows).
* **DDL (Data Definition Language):**
  + Used for **defining, modifying, and dropping database objects** (like tables, indexes).
  + **Commands:** CREATE, ALTER, DROP, TRUNCATE, RENAME.
* **DCL (Data Control Language):**
  + Used for **controlling access to data and the database**.
  + **Commands:** GRANT (give permissions), REVOKE (remove permissions).
* **TCL (Transaction Control Language):**
  + Used to **manage transactions**.
  + **Commands:** COMMIT (save changes), ROLLBACK (undo changes), SAVEPOINT (set a rollback point).

### **8. DELETE vs. DROP vs. TRUNCATE**

| **Feature** | **DELETE** | **DROP** | **TRUNCATE** |
| --- | --- | --- | --- |
| **What it does** | Removes **rows** from a table. | Deletes the **entire table structure and data**. | Removes **all rows** from a table. |
| **SQL Command Type** | DML (Data Manipulation Language) | DDL (Data Definition Language) | DDL (Data Definition Language) |
| **Rollback** | **Can be rolled back** (within a transaction). | **Cannot be rolled back**. | **Cannot be rolled back**. |
| **WHERE Clause** | Can use a WHERE clause to specify rows. | No WHERE clause. | No WHERE clause. |
| **Performance** | Slower (logs each row deletion). | **Fastest** (minimal logging, removes object). | Faster than DELETE (resets high-water mark, minimal logging). |
| **Auto-increment Reset** | Resets only if no rows are left and the table is empty and re-created (DB dependent). | **Resets** the auto-increment counter. | **Resets** the auto-increment counter. |
| **Table Structure** | Keeps the table structure. | **Removes** the table structure. | Keeps the table structure. |
| **Triggers** | Fires DELETE triggers. | Does **not** fire triggers. | Does **not** fire triggers. |

## **9. GROUP BY and ORDER BY**

* **GROUP BY:**
  + **Purpose:** Groups rows with the same values in specified columns into **summary rows**.
  + **Usage:** Often used with **aggregate functions** (COUNT(), SUM(), AVG(), MIN(), MAX()) to perform calculations on each group.
  + **Placement:** Comes *after* the WHERE clause and *before* HAVING and ORDER BY.
  + **Example:** SELECT Department, AVG(Salary) FROM Employees GROUP BY Department;
* **ORDER BY:**
  + **Purpose:** Sorts the result set of a query in ascending (ASC, default) or descending (DESC) order based on one or more columns.
  + **Usage:** Determines the **presentation order** of the results.
  + **Placement:** Always the *last* clause in a SELECT statement (excluding LIMIT/OFFSET).
  + **Example:** SELECT Name, Salary FROM Employees ORDER BY Salary DESC;

### **10. JOINS**

* **JOIN**: Used to combine rows from two or more tables based on a related column between them.
* **Types of JOINs:**
  + **INNER JOIN (or JOIN):** Returns rows when there's a **match** in both tables.
  + LEFT JOIN (or LEFT OUTER JOIN): Returns all rows from the left table and matching rows from the right table.2 NULLs for non-matching right rows.
  + **RIGHT JOIN (or RIGHT OUTER JOIN):** Returns **all rows from the right table** and matching rows from the left table. NULLs for non-matching left rows.
  + **FULL JOIN (or FULL OUTER JOIN):** Returns **all rows when there's a match in either table**. NULLs where there's no match.
  + **SELF JOIN:** A table joined with itself (using aliases) to compare rows within the same table.
  + **CROSS JOIN:** Returns the **Cartesian product** (every row from first table combined with every row from second table).

### **11. NESTED QUERY vs. CORRELATED QUERY**

| **Feature** | **Nested Query (Subquery / Inner Query)** | **Correlated Query** |
| --- | --- | --- |
| **Execution** | Inner query executes **first and once**. Its result is passed to the outer query. | Inner query executes **once for each row** processed by the outer query. |
| **Dependency** | Inner query is **independent** of the outer query. | Inner query is **dependent** on the outer query (references outer query columns). |
| **Speed** | Generally faster for simpler lookups. | Can be slower due to repeated execution. |
| **Data Flow** | Bottom-up (inner to outer). | Top-down (outer to inner, then back). |
| **Use Case** | When the inner query's result is a fixed set of values (e.g., IN, =, <). | When inner query's result depends on the specific row being processed by the outer query (e.g., EXISTS, row-by-row comparisons). |
| **Example** | SELECT Name FROM Employees WHERE DeptID IN (SELECT DeptID FROM Departments WHERE Location = 'NY'); | SELECT Name, Salary FROM Employees E WHERE Salary > (SELECT AVG(Salary) FROM Employees WHERE DeptID = E.DeptID); |

### **12. Pattern Matching (LIKE operator)**

* **Purpose:** Used to search for **patterns within string data** in a column.
* **Operator:** LIKE
* **Wildcard Characters:**
  + **% (Percent sign):** Represents **zero or more** characters.
    - 'A%': Starts with 'A'
    - '%A': Ends with 'A'
    - '%A%': Contains 'A' anywhere
  + **\_ (Underscore):** Represents a **single** character.
    - 'A\_C': 'ABC', 'AEC', etc. (A followed by any single character, then C)
* **Example:**
  + SELECT ProductName FROM Products WHERE ProductName LIKE 'Choc%';
  + SELECT FirstName FROM Customers WHERE FirstName LIKE '\_a%';

### **13. Explain Aggregate Functions in SQL. Give examples.**

* **Answer:** Aggregate functions perform calculations on a **set of values** (a column) and return a **single summary value**. They are often used with GROUP BY.
  + **Examples:**
    - COUNT(): Returns the number of rows. (COUNT(\*), COUNT(column\_name))
    - SUM(): Calculates the sum of values in a numeric column.
    - AVG(): Calculates the average of values in a numeric column.
    - MIN(): Returns the minimum value in a column.
    - MAX(): Returns the maximum value in a column.
  + **Note:** NULL values are generally ignored by aggregate functions (except COUNT(\*) which counts all rows including those with NULLs).

### **14. What is UNION and UNION ALL?**

| **Feature** | **UNION** | **UNION ALL** |
| --- | --- | --- |
| **Duplicates** | **Removes duplicate rows** from the final result set. | **Includes all rows**, including duplicates, from all queries. |
| **Performance** | Slower due to the extra sorting/hashing step to remove duplicates. | Faster as it simply appends result sets without checking for duplicates. |
| **Use Case** | When you need a distinct list of combined records. | When you need all combined records, regardless of duplicates, and performance is critical. |
| **Requirements** | Both SELECT statements must have the **same number of columns** and **compatible data types** in the same order. | Same requirements as UNION. |

### **15. What is UNION vs JOIN?**

| **Feature** | **UNION / UNION ALL** | **JOIN** |
| --- | --- | --- |
| **Purpose** | **Combines rows** from the result sets of two or more SELECT queries. | **Combines columns** from two or more tables based on a related column. |
| **Output** | Appends rows vertically. | Combines columns horizontally. |
| **Structure** | Queries must have the same number of columns and compatible data types. | Tables are linked by a common column, resulting in a wider table. |
| **Data Source** | Combines data from **similar structures** (e.g., two sales tables). | Combines data from **related but different entities** (e.g., customers and orders). |

## **Database Design & Architecture**

### **16. SQL vs. NoSQL**

| **Feature** | **SQL (Relational Databases)** | **NoSQL (Non-Relational Databases)** |
| --- | --- | --- |
| **Data Model** | Tabular, structured data with predefined schemas. | Dynamic schema, varied data models (document, key-value, graph, column-family). |
| **Scalability** | Primarily **vertical** (scale up: more resources). | Primarily **horizontal** (scale out: add more servers). |
| **ACID Properties** | Full **ACID compliance** (Atomicity, Consistency, Isolation, Durability) is typical. | May sacrifice full ACID for performance/scalability (often uses **Eventual Consistency**). |
| **Query Language** | **SQL** (Structured Query Language). | Varies (e.g., MongoDB Query Language), often API-driven. |
| **Schema** | **Rigid, fixed schema**. Schema changes can be complex. | **Flexible, schema-less**. Easy to adapt to changing data. |
| **Best For** | Complex transactions, financial systems, applications requiring strong data consistency and integrity. | Large-scale data, real-time web apps, big data, flexible data models, high write throughput. |
| **Examples** | MySQL, PostgreSQL, Oracle, SQL Server. | MongoDB, Cassandra, Redis, Neo4j, Couchbase. |

### **17. VARCHAR vs. VARCHAR2**

| **Feature** | **VARCHAR (SQL Standard, various DBs)** | **VARCHAR2 (Oracle Specific)** |
| --- | --- | --- |
| **Standard** | SQL standard data type. | **Oracle proprietary** data type. |
| **Behavior** | Its exact behavior (e.g., empty string handling) can vary slightly across different database systems. | Guaranteed to store variable-length strings. **Empty strings are treated as NULL** in Oracle. |
| **Length Unit** | Specified in bytes (or characters in some DBs/settings). | Specified in bytes (or characters in newer Oracle versions/settings). |
| **Max Length** | Varies by database (e.g., MySQL 65,535 bytes, SQL Server 8000 bytes). | Typically **4000 bytes** (for non-LOB columns). |
| **Recommendation** | Use in non-Oracle databases (MySQL, PostgreSQL, SQL Server). | **Prefer VARCHAR2 in Oracle databases** for consistent and predictable behavior. |

### **18. What is Normalization? What are its benefits and different forms?**

* **Answer:**
  + **Normalization:** A systematic process of organizing the columns and tables of a relational database to minimize data redundancy and improve data integrity.3
  + **Benefits:**
    - Reduces data redundancy (less duplicated data).
    - Improved data integrity (prevents update, insert, and delete anomalies).
    - Saves storage space.
    - Makes the database easier to maintain and extend.
  + **Normal Forms (briefly mention the first few):**
    - **1NF (First Normal Form):** Each column contains atomic (indivisible) values, and there are no repeating groups of columns.
    - **2NF (Second Normal Form):** Is in 1NF and all non-key attributes are fully dependent on the primary key (no partial dependencies).
    - **3NF (Third Normal Form):** Is in 2NF and all non-key attributes are dependent on the primary key, and no transitive dependencies exist (i.e., no non-key attribute depends on another non-key attribute).

## **Advanced Concepts & Optimizations**

### **19. TRIGGERS**

* **Trigger:** A special type of stored procedure that **automatically executes** or "fires" when a specific event occurs in the database.
* **Events that can activate a trigger:**
  + **DML Events:** INSERT, UPDATE, DELETE (on a table).
  + **DDL Events:** CREATE, ALTER, DROP (on schema or database).
  + **Database Events:** LOGON, LOGOFF, STARTUP, SHUTDOWN.
* **When they fire:**
  + BEFORE (e.g., BEFORE INSERT): Executes before the DML operation. Useful for validation.
  + AFTER (e.g., AFTER UPDATE): Executes after the DML operation. Useful for auditing or cascading actions.
* **Purpose/Use Cases:** Auditing, enforcing complex business rules, data validation, cascading actions, maintaining complex referential integrity.
* **Considerations:** Can be difficult to debug and manage; overuse can impact performance.

### **20. ACID Properties (of Transactions)**

* **ACID:** An acronym representing a set of properties that guarantee **reliable processing of database transactions**.
* **Transaction:** A single logical unit of work performed on the data.
* **Properties:**
  + **Atomicity:** "**All or Nothing**." A transaction is treated as a single, indivisible unit. Either all operations within it complete successfully, or none do (it's rolled back).
  + **Consistency:** Ensures that a transaction brings the database from one **valid state to another valid state**. It upholds all defined rules and constraints.
  + Isolation: Ensures that concurrent transactions do not interfere with each other. Each transaction appears to execute in isolation.4
  + **Durability:** Guarantees that once a transaction has been **committed, its changes are permanent** and will survive any subsequent system failures.

### **21. ORDER OF EXECUTION in SQL**

This is the **logical order** in which SQL clauses are processed in a SELECT statement:

1. **FROM and JOINs**: Determine the base table(s) and how they are combined.
2. **WHERE**: Filters **rows** based on specified conditions.
3. **GROUP BY**: Groups the filtered rows into summary groups.
4. **HAVING**: Filters these **groups** based on specified conditions (often with aggregate functions).
5. **SELECT**: Specifies which columns (or expressions) to retrieve. Aggregate functions are calculated here.
6. **DISTINCT**: Removes duplicate rows from the result set.
7. **ORDER BY**: Sorts the final result set.
8. **LIMIT/OFFSET (or TOP/FETCH FIRST)**: Restricts the number of rows returned.

### **22. What is a VIEW in SQL? Why is it used?**

* **Answer:**
  + A VIEW is a **virtual table** based on the result-set of a SELECT query. It doesn't store data itself; it's like a stored query.
  + **Why used:**
    - **Security:** Restrict user access to specific rows and columns without giving direct table access.
    - **Simplification:** Simplify complex queries by pre-joining tables and applying conditions, making it easier for users to query.
    - **Consistency:** Present a consistent view of data even if underlying table structures change (as long as the VIEW definition is updated).
    - **Data Abstraction:** Hides the complexity of the underlying database schema from end-users.

### **23. What is an Index in SQL? What are its types and benefits?**

* **Answer:**
  + **Index:** A special lookup table that the database search engine can use to **speed up data retrieval** from a table. It's like an index in a book.
  + **Types (most common):**
    - **Clustered Index:** Determines the **physical order** of data rows in a table. A table can have only **one** clustered index. (e.g., typically on the Primary Key).
    - **Non-Clustered Index:** Stores the logical order of a column and contains pointers to the actual data rows. A table can have **multiple** non-clustered indexes.
  + **Benefits:**
    - **Faster query execution** for SELECT statements (especially with WHERE, ORDER BY, JOIN clauses).
    - Improves performance of JOIN and GROUP BY operations.
  + **Drawbacks:**
    - Requires additional storage space.
    - Can slow down INSERT, UPDATE, and DELETE operations as indexes need to be updated.

### **24. What is a Stored Procedure? What are its advantages?**

* **Answer:**
  + **Stored Procedure:** A **pre-compiled collection of SQL statements** (and often procedural logic) that are stored in the database. They can be executed by applications or users by calling their name.
  + **Advantages:**
    - **Performance:** Pre-compiled and cached, leading to faster execution.
    - **Reusability:** Can be called multiple times by different applications/users.
    - **Security:** Users can be granted permission to execute procedures without direct access to the underlying tables.
    - **Reduced Network Traffic:** Multiple SQL statements execute on the server, reducing client-server communication.
    - **Maintainability:** Centralized business logic makes updates easier.

**SQL Online Assessment**

1. If we divide 0 with a NULL, what will be the error/output

A) 0 B) NULL C) Division Error D) Query will not execute

1. If we divide a NULL with 0 (or any number), what will be the error/output

A) 0 B) *NULL* C) Division Error D) Query will not execute

1. If we divide a NULL with NULL, what will be the error/output

A) 0 B) NULL C) Division Error D) Query will not execute

1. If we divide any 0 with any number, what will be the error/output

A) 0 B) NULL C) Division Error D) Query will not execute

1. If we divide 0 with 0 (or any number), what will be the error/output

A) 0 B) NULL C) Division Error D) Query will not execute

**NOTE: Perform any operation (sum, subtract, div, multiply) with NULL value, output will be NULL.**

1. WHERE FirstName LIKE 'A%' - Which names does this query return? Select all that are applicable.

A) ARJUN B) TARA C) BHEEM D) ABHIMANYU

1. WHERE FirstName LIKE '\_*R%*' - Which names does this query return? Select all that are applicable.

A) AR B) KRISHNA C) ARJUN D) ROHINI

1. WHERE FirstName LIKE 'M%N' - Which names does this query return? Select all that are applicable.

A) NAKUL B) MADHAV C) SUNDAR D) MOON

1. From the given WHERE clauses, which will return only rows that have a NULL in a column?

A) WHERE column\_name <> NULL B) WHERE column\_name IS NULL C) WHERE column\_name = NULL D) WHERE column\_name NOT IN ('')

1. From the given WHERE clauses, which will return only rows that have a NOT NULL in a column?

A) WHERE column\_name = NULL B) WHERE column\_name IS NOT NULL C) WHERE column\_name IS NULL D) WHERE column\_name != NULL

1. Use of limit and offset in sql together in a sql query. Select all that are applicable.
2. A) SELECT \* FROM artists LIMIT 5 OFFSET 2; B) SELECT \* FROM artists 5, OFFSET 2; C) SELECT \* FROM artists OFFSET 2, 5; D) SELECT \* FROM artists 2, 5;
3. Which of the update queries listed below is/are valid?

A) UPDATE Supplier SET city='Pune' AND Phone='9123456' AND Fax='044-54321'

B) UPDATE Supplier SET city='Pune', Phone='9123456', Fax='044-54321'

C) Options A & B

D) None of the above