**IMPORTANT DIFFERENCES**

**CHAR vs. VARCHAR:**

* **CHAR**: Fixed-length storage, faster but wastes space if data length varies.
* **VARCHAR**: Variable-length storage, more space-efficient but slightly slower.

**------------------------------------------------------------------------------------------------**

**INNER JOIN vs. OUTER JOIN:**

* **INNER JOIN**: Returns only matching rows from both tables.
* **OUTER JOIN**: Returns all rows from one table and matching rows from the other (LEFT, RIGHT, or FULL).

**------------------------------------------------------------------------------------------------**

**PRIMARY KEY VS. UNIQUE KEY:**

* **Primary Key**: Uniquely identifies each row, cannot be NULL, only one per table.
* **Unique Key**: Ensures uniqueness in a column, can have multiple unique keys, allows NULL values.

**------------------------------------------------------------------------------------------------**

**PRIMARY KEY VS FOREIGN KEY:**

* **Primary Key :** A primary key is a unique identifier for each record in a database table. It ensures that each row in the table is uniquely identifiable.
* **Foreign Key :** A foreign key is a column or a set of columns in one table that refers to the primary key in another table. It establishes a relationship between the two tables.

**------------------------------------------------------------------------------------------------**

**WHERE vs. HAVING:**

* **WHERE**: Filters rows before grouping (used with SELECT, UPDATE, DELETE).
* **HAVING**: Filters groups after aggregation (used with GROUP BY).

**------------------------------------------------------------------------------------------------**

**TRUNCATE vs. DELETE:**

* **TRUNCATE**: Removes all rows, faster, cannot be rolled back, resets auto-increment.
* **DELETE**: Removes specific rows, slower, can be rolled back, doesn’t reset auto-increment.

**------------------------------------------------------------------------------------------------**

**STORED PROCEDURE VS. FUNCTION:**

* **Stored Procedure**: Can perform complex operations, return multiple values, and modify data.
* **Function**: Returns a single value, used for calculations, cannot modify data.

**------------------------------------------------------------------------------------------------**

**UNION vs. UNION ALL:**

* **UNION**: Combines results from multiple SELECTs, removes duplicates.
* **UNION ALL**: Combines results from multiple SELECTs, keeps duplicates.

**------------------------------------------------------------------------------------------------**

**TEMPORARY TABLE VS. VIEW:**

* **Temporary Table**: Stores intermediate results, exists only for the duration of the session.
* **View**: A saved query, acts as a virtual table, doesn't store data physically.

**------------------------------------------------------------------------------------------------**

**SQL vs. NoSQL:**

* **SQL**: Structured data, uses relational databases, fixed schemas, ACID compliance.
* **NoSQL**: Unstructured or semi-structured data, uses non-relational databases, flexible schemas, BASE model (Basically Available, Soft state, Eventual consistency).

**------------------------------------------------------------------------------------------------**

**RELATIONAL VS. NON-RELATIONAL DATABASES**

**RELATIONAL DATABASES:**

* **Structure:** Data is organized into tables with rows and columns, where each row represents a record and each column represents a field within that record.
* **Schema:** Fixed schema; you must define the structure (tables, columns, data types) before adding data.
* **Relationships:** Uses primary and foreign keys to define relationships between tables.
* **Examples:** MySQL, PostgreSQL, Oracle, SQL Server.
* **Use Cases:** Ideal for complex queries, structured data, and ACID-compliant transactions (e.g., financial systems, CRM, ERP).

**NON-RELATIONAL DATABASES:**

* **Structure:** Data can be stored in various formats such as documents, key-value pairs, wide-columns, or graphs.
* **Schema:** Flexible schema; data can be added without a predefined structure.
* **Relationships:** Typically doesn’t enforce relationships between data entities (although some like graph databases do).
* **Examples:** MongoDB (document), Cassandra (wide-column), Redis (key-value), Neo4j (graph).
* **Use Cases:** Suitable for unstructured or semi-structured data, horizontal scaling, and rapid development (e.g., social networks, big data, real-time analytics).

**------------------------------------------------------------------------------------------------**

**STRUCTURAL VS. NON-STRUCTURAL DATABASES**

**STRUCTURAL DATABASES:**

* **Definition:** Another way to refer to relational databases where data is structured in a defined format (tables with rows and columns).
* **Schema:** Requires a fixed schema before data entry.
* **Data Format:** Strict format, making it easier to enforce data integrity and consistency.
* **Examples:** MySQL, SQL Server, PostgreSQL.
* **Use Cases:** Applications requiring well-organized data with strict relationships (e.g., banking systems, inventory management).

**NON-STRUCTURAL DATABASES:**

* **Definition:** A term often used interchangeably with non-relational databases, where data doesn't adhere to a fixed structure.
* **Schema:** No fixed schema, allowing for flexible data formats and easier scaling.
* **Data Format:** Data can be stored in various formats like JSON, BSON, or simple key-value pairs.
* **Examples:** MongoDB (JSON documents), Cassandra (wide-columns), Neo4j (graphs).
* **Use Cases:** Ideal for applications where data structure can evolve, or where large-scale, distributed data processing is needed (e.g., IoT, content management, social networks).