**RENAME:**

*RENAME TABLE <old\_tablename> TO <new\_tablename>;*

*OR*

*ALTER TABLE <old\_tablename> RENAME TO <new\_tablename>;*

* These commands are used to rename the tables.

**COPYING TABLE:**

* *INSERT INTO <new\_table\_name> SELECT \* FROM <old\_table\_name>;*

*OR*

* *INSERT INTO <new\_table\_name> TABLE <old\_table\_name>;*
  + These commands are used to copy the data from one table to other table.
  + The new table should already be created. The schema should be same to use these commands.
  + SELECT \* FROM and TABLE serves the same we can use anything.

***INSERT INTO tbl\_temp2 (fld\_id) SELECT tbl\_temp1.fld\_order\_id FROM tbl\_temp1 WHERE tbl\_temp1.fld\_order\_id > 100;***

* + Using this command in this way we can also specify which data to copy from one table to another.
* *CREATE TABLE new\_table\_name LIKE old\_table\_name;*
  + This command is used to create a new table exactly like the given existing table.
* *SELECT \* INTO <new\_table\_name> TABLE <old\_table\_name>;*
  + This command is used in SQL SERVER, not in MySQL. This command creates the new table and will insert into that table.
  + This command won’t work if the table already exists.

**AGGREGATE FUNCTIONS:**

* **AVG()**: Calculates the average value of a numeric column.
* **COUNT()**: Counts the number of rows in a dataset.
* **MIN()**: Finds the minimum value in a column.
* **MAX()**: Finds the maximum value in a column.
* **SUM()**: Calculates the total sum of a numeric column.

These functions typically ignore NULL values, except for the COUNT() function, which counts all rows, including those with NULL values.

Aggregate functions are commonly used with the GROUP BY clause in SQL queries to group data into subsets and perform calculations on each subset.

**GROUP BY:**

* Organises data into groups based on one or more columns.
* Often used with aggregate functions such as SUM(), AVG(), COUNT(), etc.
* Produces a single row for each unique value (or combination of values) in the grouped columns.
* Use GROUP BY when you want to aggregate data by specific categories and get summaries like totals, averages, or counts.

**HAVING**

* Filters the grouped data after the GROUP BY clause has been applied.
* Works like WHERE, but for aggregated results.
* Removes rows that don’t meet the condition specified in HAVING.
* Use HAVING to filter groups based on aggregate values like averages, counts, or sums.
* HAVING is typically paired with GROUP BY, while WHERE filters before grouping, HAVING filters after grouping.

**OVER()**

The OVER() clause in SQL is used to perform operations across a set of rows that are related to the current query row. It is not used to group rows like the GROUP BY clause; instead, it provides the ability to perform calculations over a subset of rows without collapsing the rows into a single summary row.

* **RANK():** Assigns a rank to rows within a partition, with gaps for ties.
* **DENSE\_RANK():** Assigns a rank without gaps for ties.

The `OVER()` clause defines the set of rows that the window function should operate on. It can use various components to define the window:

* **PARTITION BY:** Divides the result set into partitions based on specified columns (like GROUP BY).
* **ORDER BY:** Orders the rows within each partition.
* **ROWS/RANGE:** Specifies a frame of rows within the partition to operate on.

**DAYOFWEEK()**

The DAYOFWEEK() function in SQL is used to return the day of the week for a given date. It returns an integer value representing the day of the week, where:

* 1 corresponds to Sunday
* 2 corresponds to Monday
* 3 corresponds to Tuesday
* … and so on, up to 7 for Saturday

**DATE\_SUB():**

The DATE\_SUB() function in SQL is used to subtract a specified time interval from a date. This function is commonly used in MySQL and is particularly useful when you need to perform date arithmetic, such as calculating dates before a given date or filtering records based on a date range.

**Syntax:** *DATE\_SUB(date, INTERVAL value unit)*

* date: The starting date from which the interval will be subtracted.
* value: The amount of time to subtract. This can be an integer representing the number of units.
* unit: The unit of time to subtract, such as DAY, MONTH, YEAR, HOUR, MINUTE, etc.

**JOINS:**

In SQL, JOINS are used to combine rows from two or more tables based on a related column between them. Joins allow you to retrieve data spread across multiple tables using a single query. Each type of join defines how SQL should return records when there is a match (or not) between the tables.

Types of Joins in SQL

1. INNER JOIN
2. LEFT JOIN (LEFT OUTER JOIN)
3. RIGHT JOIN (RIGHT OUTER JOIN)
4. FULL JOIN (FULL OUTER JOIN)
5. CROSS JOIN

**INNER JOIN**

* Returns only the rows that have matching values in both tables.
* Used when you only want to see data where there is a match in both tables.

**LEFT JOIN (LEFT OUTER JOIN)**

* Returns all rows from the left table and matching rows from the right table. If there’s no match, NULLs are returned for columns from the right table.
* Used when you want all records from the left table, regardless of whether there’s a match in the right table.

**RIGHT JOIN (RIGHT OUTER JOIN)**

* Returns all rows from the right table and matching rows from the left table. If there’s no match, NULLs are returned for columns from the left table.
* Used when you want all records from the right table, regardless of whether there’s a match in the left table.

**FULL JOIN (FULL OUTER JOIN)**

* Returns all rows when there’s a match in either the left or right table. Rows without a match will show NULLs for columns from the missing side.
* Used when you want to combine results of both LEFT JOIN and RIGHT JOIN.

**CROSS JOIN**

* Returns the Cartesian product of both tables, i.e., every row in the first table is combined with every row in the second table.
* Used with caution—typically used for generating combinations or testing.

**SELF JOIN:**

* A self join is a join where a table is joined with itself.
* It is used to compare rows within the same table or to create relationships between rows in the same table.
* This can be particularly useful when a table row needs to be compared with other rows in the same table.
* When performing a self join, the table is treated as if it were two separate tables by using table aliases.
* You essentially join the table to itself, typically using a JOIN clause.

**MYSQL DOCUMENTATION REFERENCE**

[MySQL :: MySQL 9.0 Reference Manual](https://dev.mysql.com/doc/refman/9.0/en/)