**Name -Monika Gariya**

**Email-** [**monikagariya2023@gmail.com**](mailto:monikagariya2023@gmail.com)

**Data Engineering Batch 1**

**Date – 24-01-2024**

**Topic – Partition by Clause,Order of execution,Nested Subquary,RegExp,Star and Snowflake schema**

**Partition By Clause**

A **PARTITION BY**clause is used to partition rows of table into groups. It is useful when we have to perform a calculation on individual rows of a group using other rows of that group.

It is always used inside OVER() clause.

The partition formed by partition clause are also known as **Window**.

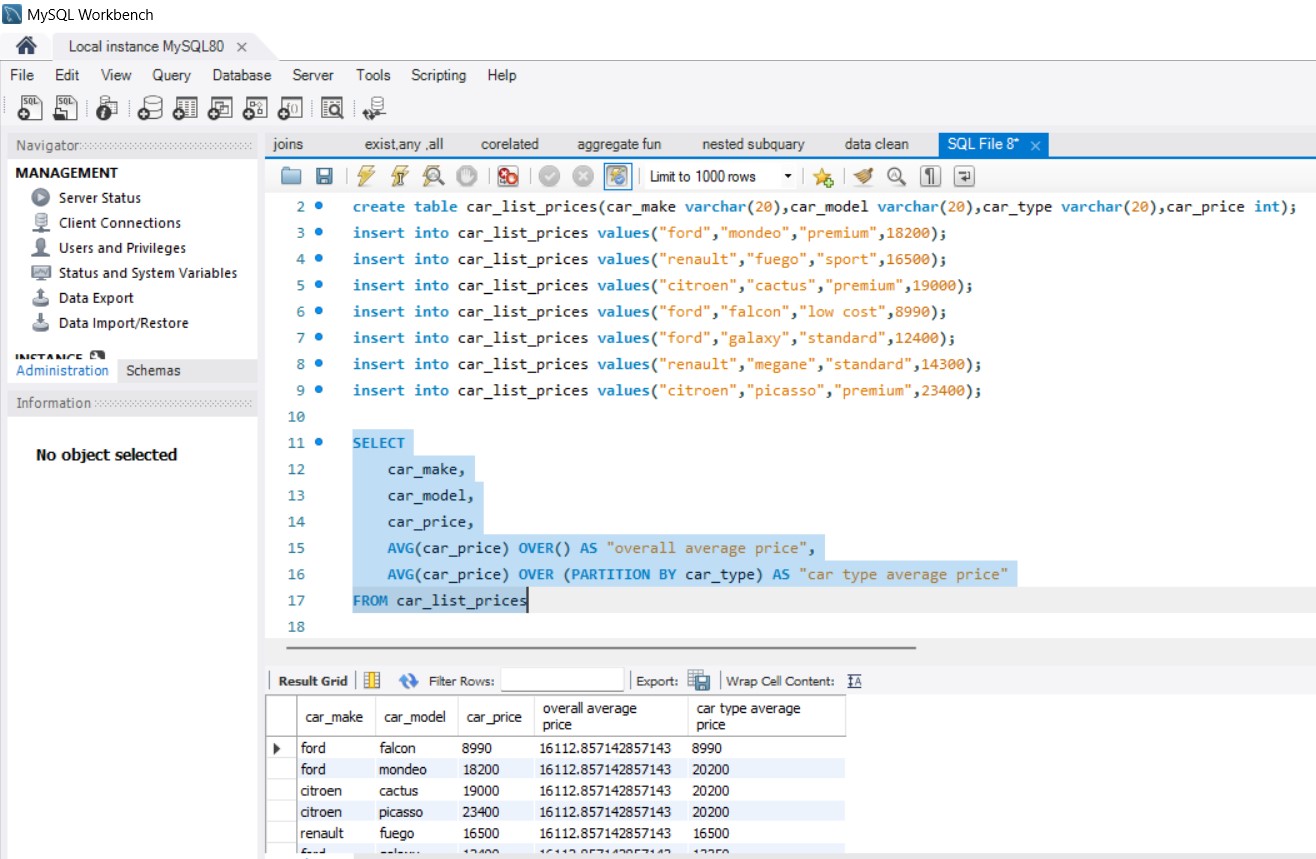
This clause works on windows functions only. Like- RANK (), LEAD (), LAG () etc.

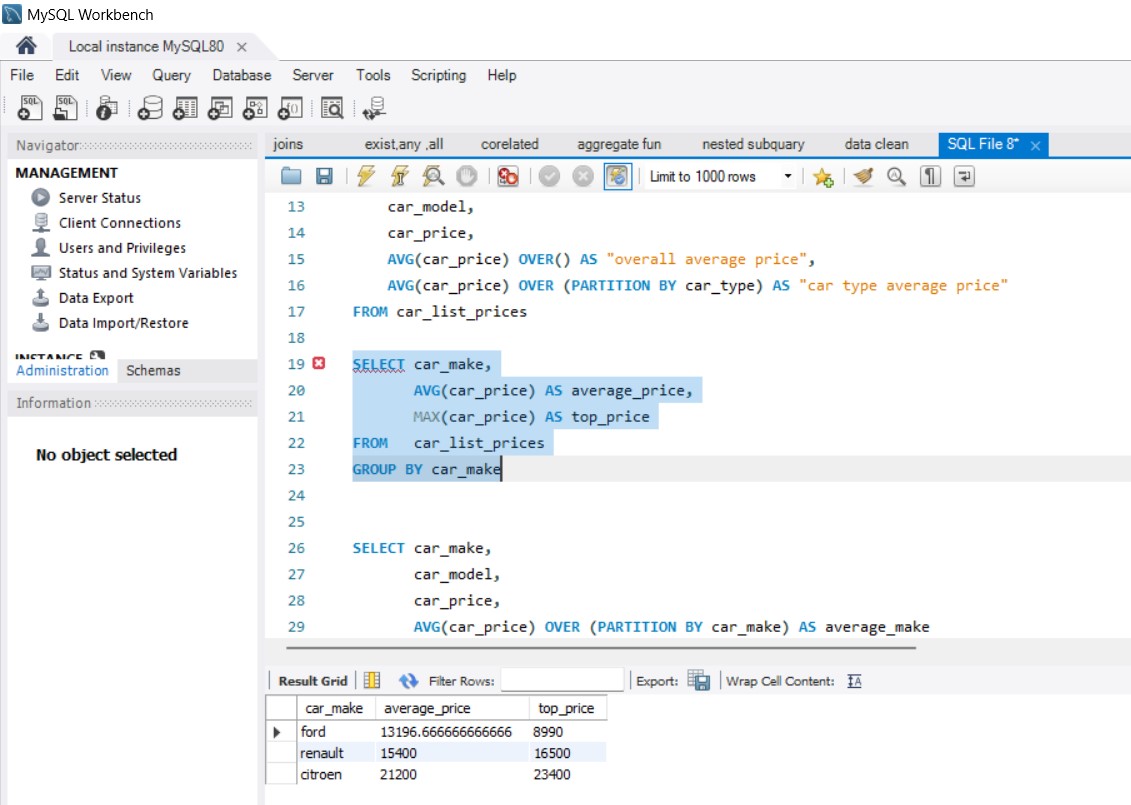
If this clause is omitted in OVER () clause, then whole table is considered as a single partition.

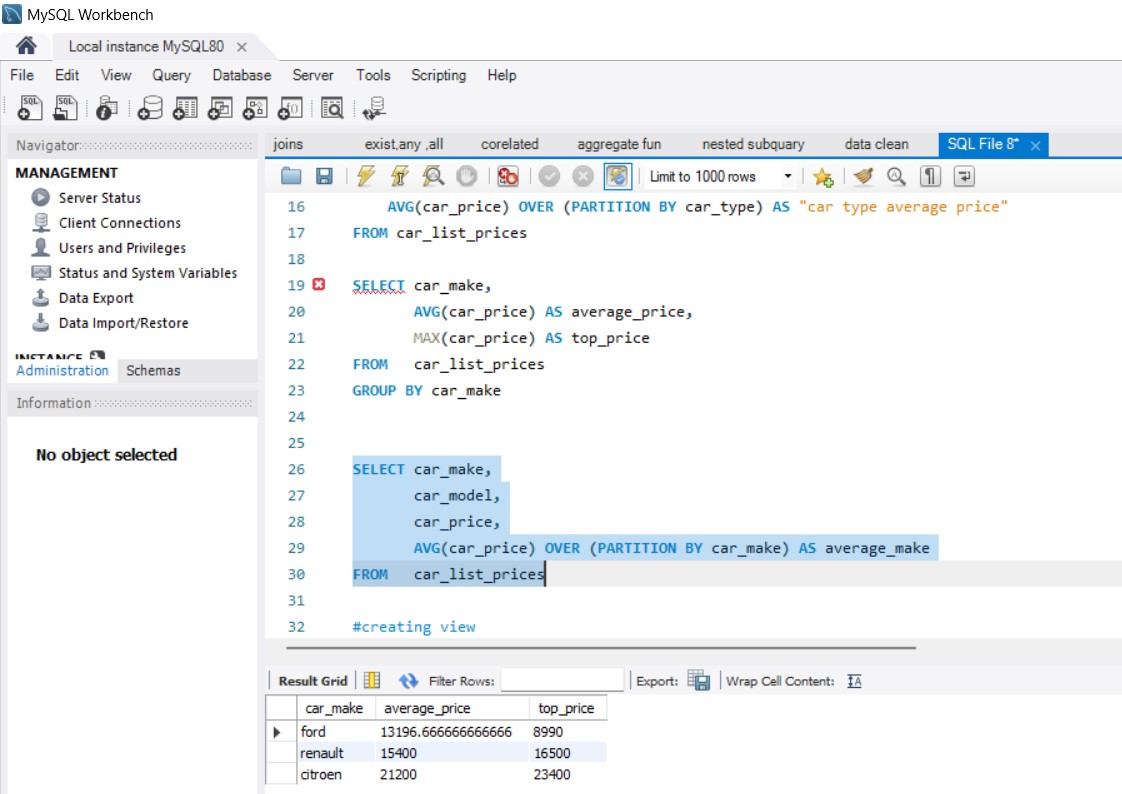
OVER CLAUSE

**OVER Clause:**

* The **OVER** clause is used to define the window or set of rows for a window function. It specifies the partitioning and ordering of the rows within the window.







**ORDER OF EXECUTION IN SQL**

SQL queries adhere to a specific order when evaluating clauses, similar to how mathematical operations adhere to PEMDAS or BODMAS.

**1)FROM -** The FROM clauses are executed first to determine the data of interest.

**2)WHERE  -** The WHERE clause is executed to filter out records that do not meet the constraints.

**3)GROUP BY -** The GROUP BY clause is executed to group the data based on the values in one or more columns.

**4)** **HAVING**- The HAVING clause is executed to remove the created grouped records that don’t meet the constraints.

**5)SELECT -** The SELECT clause is executed to derive all desired columns and expressions.

**6)ORDER BY**- The ORDER BY clause is executed to sort the derived values in ascending or descending order.

**7)LIMIT/OFFSET**- The LIMIT and/or OFFSET clauses are executed to keep or skip a specified number of rows.

TABLE 1 -ORDERS

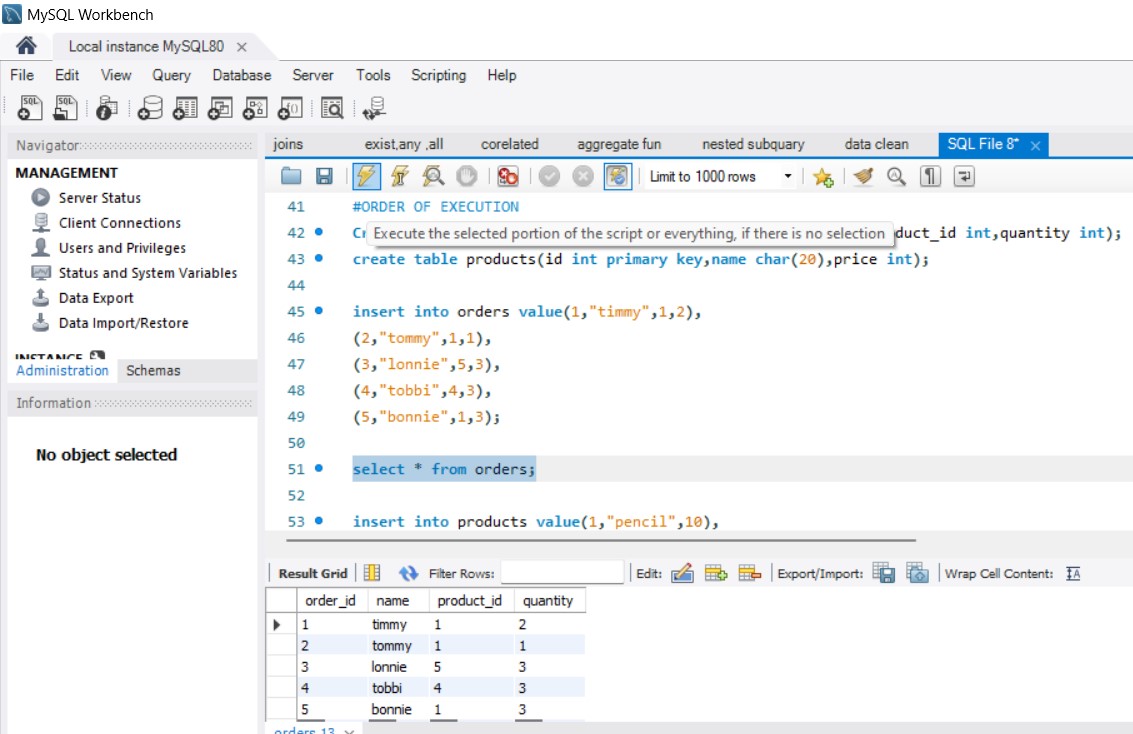
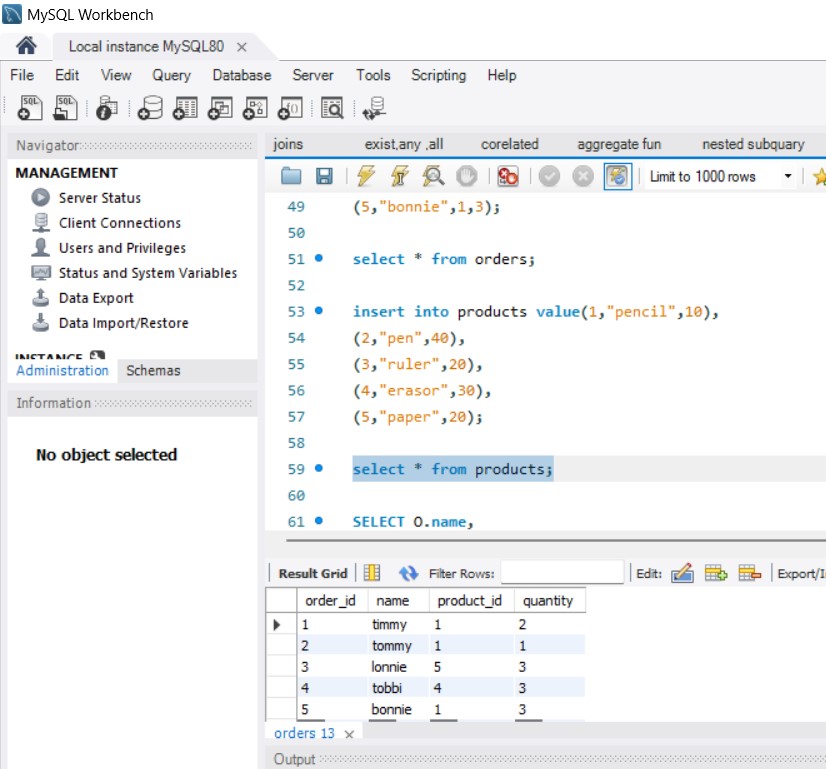
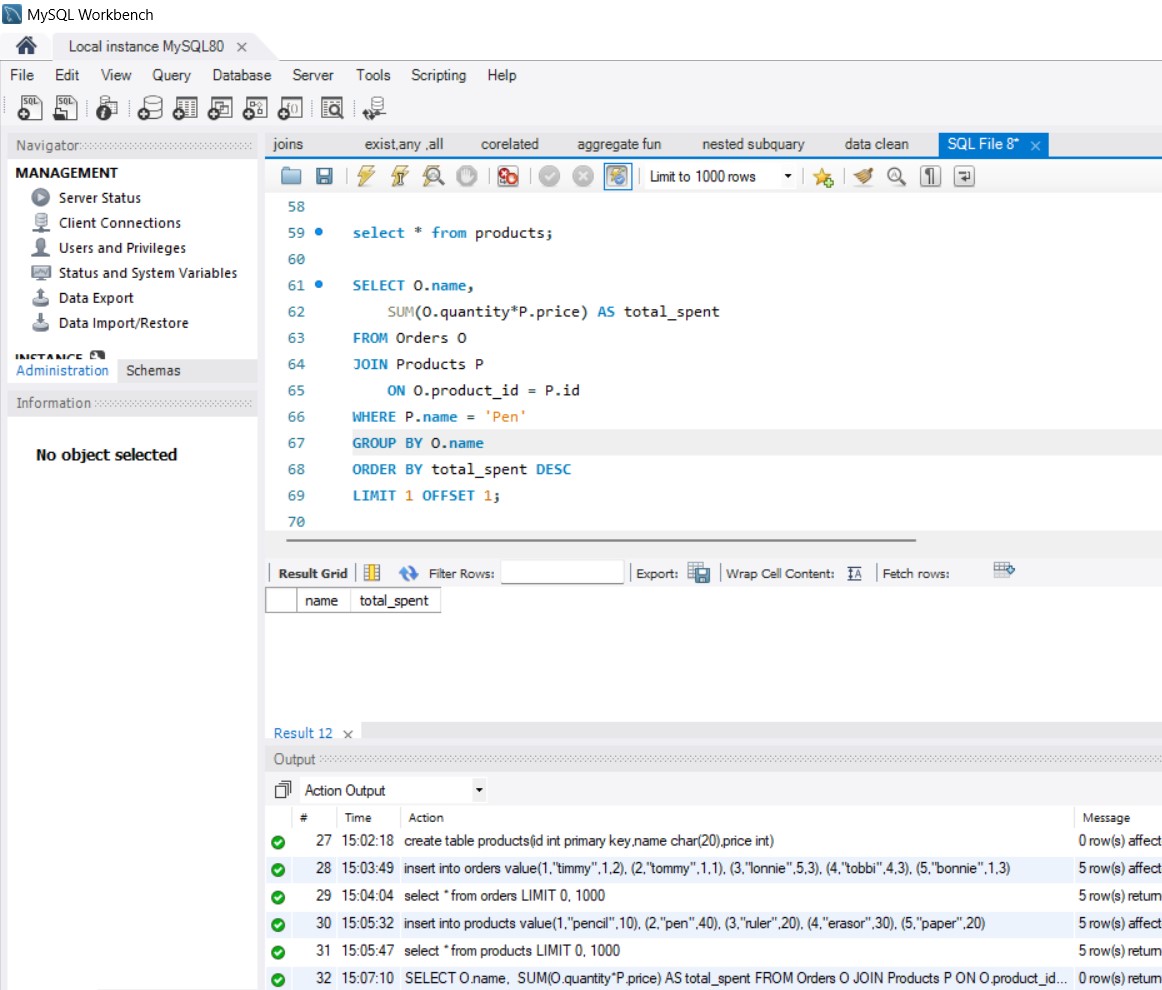


TABLE – 2 PRODUCTS

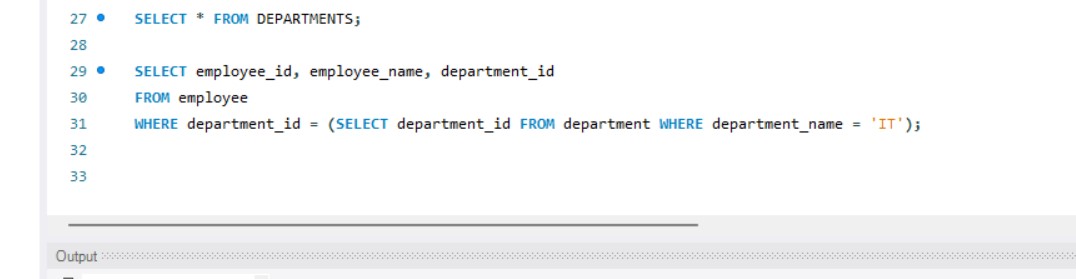


APPLYING ORDER OF EXECUTION.



**NESTED SUBQUARY**

A subquery can be nested inside other subqueries. SQL has an ability to nest queries within one another. A subquery is a SELECT statement that is nested within another SELECT statement and which return intermediate results. SQL executes innermost subquery first, then next level.



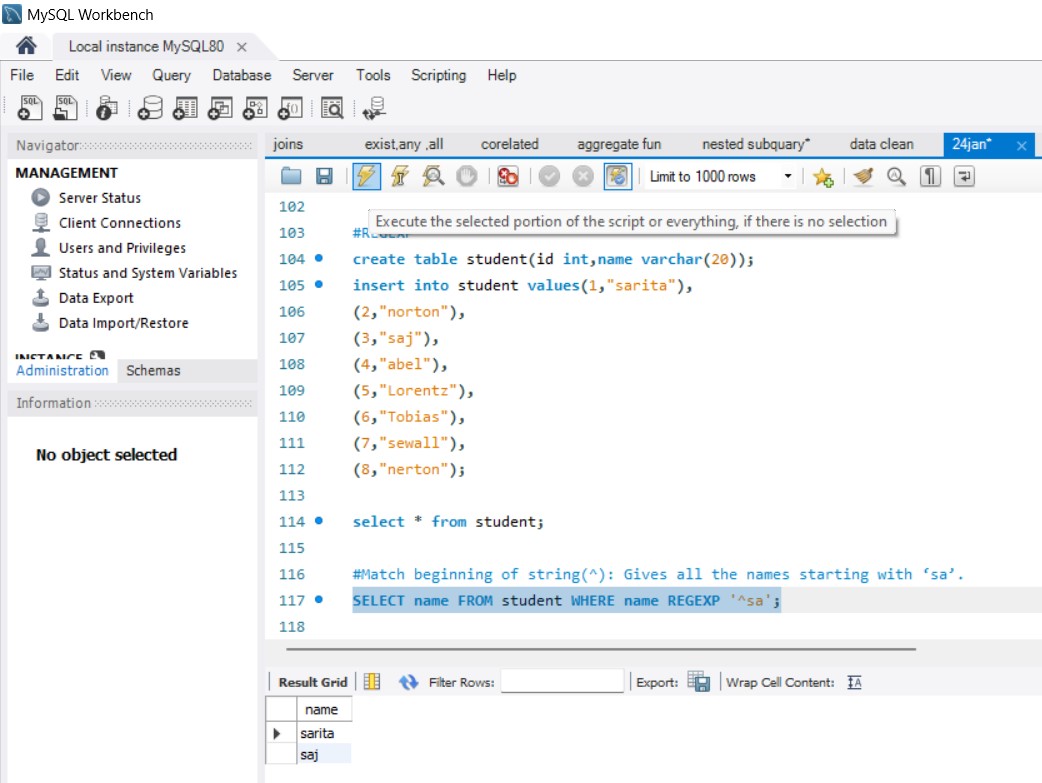
**RegEXP**

MySQL supports another type of pattern matching operation based on the regular expressions and the REGEXP operator.

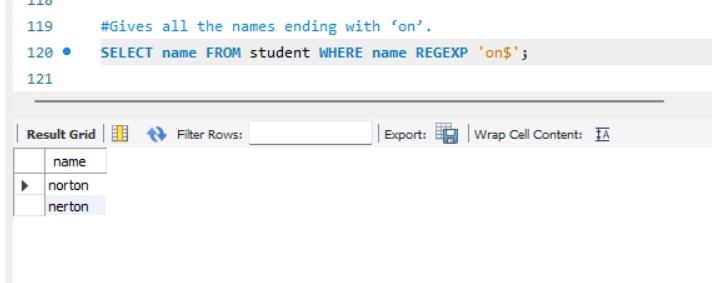
1. It provides a powerful and flexible pattern match that can help us implement power search utilities for our database systems.
2. REGEXP is the operator used when performing regular expression pattern matches. RLIKE is the synonym.
3. It also supports a number of metacharacters which allow more flexibility and control when performing pattern matching.
4. The backslash is used as an escape character. It’s only considered in the pattern match if double backslashes have used.
5. Not case sensitive.

| **Pattern** | **What the Pattern matches** |
| --- | --- |
| \* | Zero or more instances of string preceding it |
| + | One or more instances of strings preceding it |
| . | Any single character |
| ? | Match zero or one instances of the strings preceding it. |
| ^ | caret (^) matches Beginning of string |
| $ | End of string |
| [abc] | Any character listed between the square brackets |
| [^abc] | Any character not listed between the square brackets |
| [A-Z] | match any upper-case letter. |
| [a-z] | match any lower-case letter |
| [0-9] | match any digit from 0 through to 9. |
| [[:<:]] | matches the beginning of words. |
| [[:>:]] | matches the end of words. |
| [:class:] | matches a character class i.e. [: alpha:] to match letters, [:space:] to match white space, [: punct:] is match punctuations and [:upper:] for upper class letters. |
| p1|p2|p3 | Alternation; matches any of the patterns p1, p2, or p3 |
| {n} | n instances of preceding element |
| {m,n} | m through n instances of preceding element |

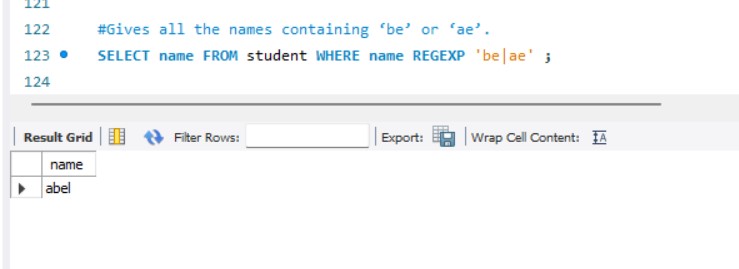
**Example 1**-Match beginning of string(^): Gives all the names starting with ‘sa’.



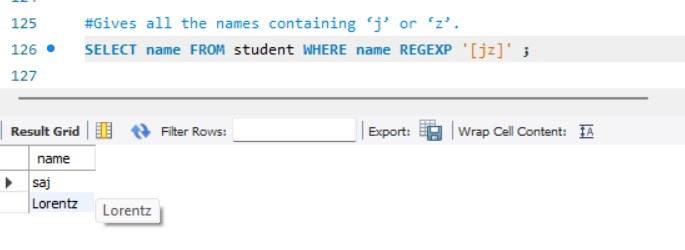
**Example 2-** Gives all the names ending with ‘on’.



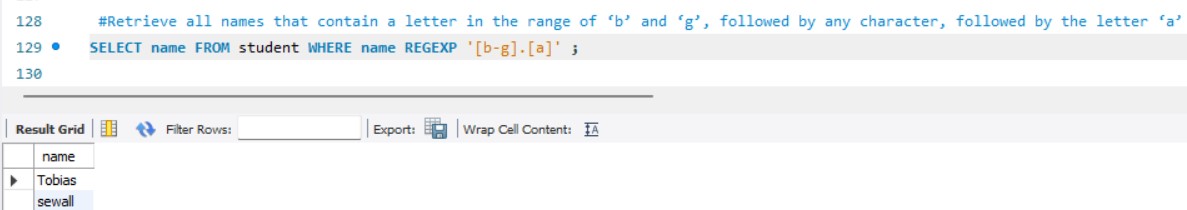
**Example 3 -** Gives all the names containing ‘be’ or ‘ae’.



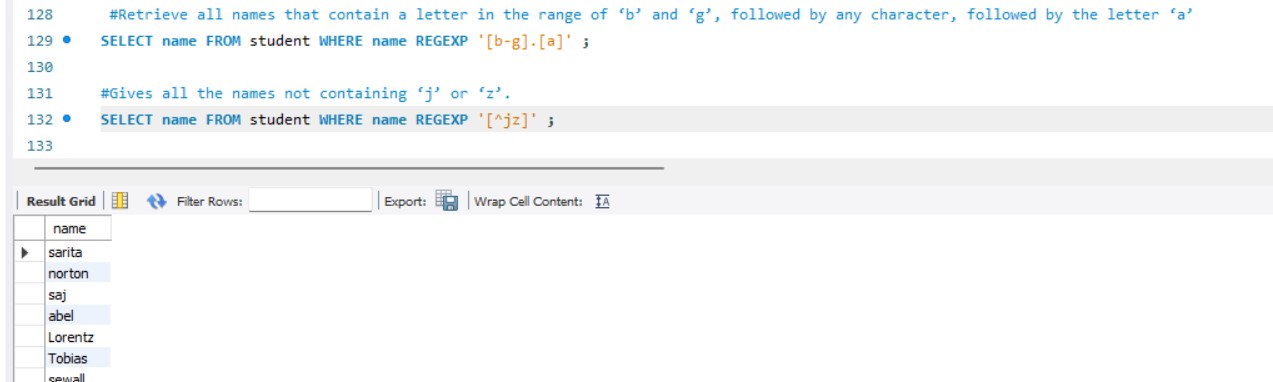
**Example 4**- Gives all the names containing ‘j’ or ‘z’.



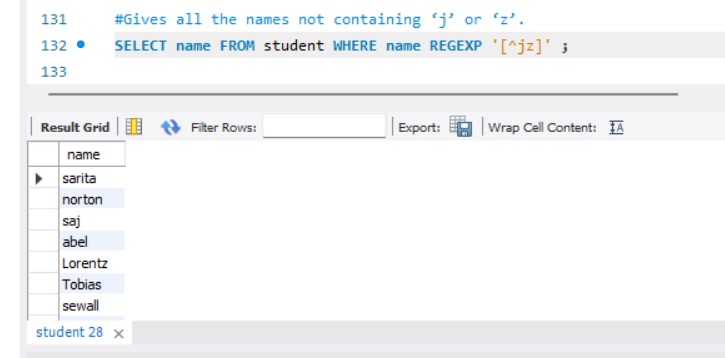
**Example 5-** Retrieve all names that contain a letter in the range of ‘b’ and ‘g’, followed by any character, followed by the letter ‘a’



**Example 6 -** Gives all the names not containing ‘j’ or ‘z

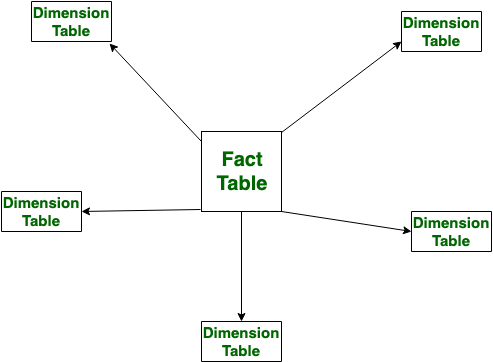


**Example 7-** Gives all the names not containing ‘j’ or ‘z’.



**STAR SCHEMA**

Star schema is the type of multidimensional model which is used for data warehouse. In star schema, the fact tables and the dimension tables are contained. In this schema fewer foreign-key join is used. This schema forms a star with fact table and dimension tables.



**SNOW FLAKE SCHEMA**

Snowflake Schema is also the type of multidimensional model which is used for [data warehouse](https://www.geeksforgeeks.org/data-warehousing/). In snowflake schema, the fact tables, dimension tables as well as sub dimension tables are contained. This schema forms a snowflake with fact tables, dimension tables as well as sub-dimension tables.

