Select Statement

The Select Statement

Depending on the DBMS, the select statement can pack in it a lot of optional keywords. For example: for MySQL

```
SELECT
    [ALL | DISTINCT | DISTINCTROW ]
    [HIGH_PRIORITY]
    [STRAIGHT_JOIN]
    [SQL_SMALL_RESULT] [SQL_BIG_RESULT] [SQL_BUFFER_RESULT]
    [SQL_NO_CACHE] [SQL_CALC_FOUND_ROWS]
    select_expr [, select_expr] ...
    [into_option]
    [FROM table_references
      [PARTITION partition_list]]
    [WHERE where_condition]
    [GROUP BY {col_name | expr | position}, ... [WITH ROLLUP]]
    [HAVING where_condition]
    [WINDOW window_name AS (window_spec)
        [, window_name AS (window_spec)] ...]
    [ORDER BY {col_name | expr | position}
      [ASC | DESC], ... [WITH ROLLUP]]
    [LIMIT {[offset,] row_count | row_count OFFSET offset}]
    [into_option]
    [FOR {UPDATE | SHARE}
        [OF tbl_name [, tbl_name] ...]
        [NOWAIT | SKIP LOCKED]
      | LOCK IN SHARE MODE]
    [into_option]

into_option: {
    INTO OUTFILE 'file_name'
        [CHARACTER SET charset_name]
        export_options
  | INTO DUMPFILE 'file_name'
```

```
  | INTO var_name [, var_name] ...
}
```

I will only demonstrate a subset of those functionalities below.

Min, Max, Avg, Count, Sum

With these aggregator functions, you would apply them to a specific column or across all columns to find the specific aggregator value. For example, if you want to find the minimum value of a specific column `age` you would do:

```
SELECT MIN(age) FROM person;
```

All the other functions, max, avg, count, and sum works similarily, except computing different things.

Count(*) vs Count(column name)

Are there any differences between these two calling of the aggregator functions? Of course there is.

`count(*)` or any literal value inside the parenthesis will count the number of rows in the table, regardless whether or not there is a NULL value.

`count(column name)` on the other hand will only count the rows in the specified column while excluding NULL value.

Group by

Group by clause in your select statement will allow you to carry out the aggregator function on the specific column you are grouping the rows on. For example, if you have a class of students, and have their gender on a separate column. If you want to find the average age of the male and female students, you would group the rows of students by their gender, then apply the `avg` aggregator function on their age column:

```
SELECT avg(age) from students GROUP BY gender;
```

Your output may look like:

```
+------+
| age  |
+------+
|   64 |
```

```
|   69 |
+------+
```

Notice that you do not get to see which age is for which group, is the top one for Female or Male? In order to show that information, you would need to select on the column that you have grouped on:

```
SELECT avg(age), gender from students GROUP BY gender;
```

Then your output will look much more informative:

```
+------+--------+
| age  | gender |
+------+--------+
|   64 | M      |
|   69 | F      |
+------+--------+
```

Group by multiple columns

Just like in Pandas where you can group by with multiple columns, you can do it as well in SQL.

```
SELECT EmployeeID, ShipperID, COUNT(*)
FROM Orders
Group BY EmployeeID, ShipperID
Order BY ShipperID, Count(*) DESC
```

The output you might get is the following:

```
EmployeeID□ShipperID□COUNT(*)
4□□□□1□□□12
1□□□□1□□□8
2□□□□1□□□7
3□□□□1□□□7
6□□□□1□□□7
```

Just keep in mind that you would have to select out the columns you group on in order to show what you have grouped by in the output, otherwise it wouldn't be helpful information.

Order by

This is basically the sorting for your output. The rows that you have gotten, after group by if any, will sort base on the columns that you have specified.

Append the `DESC` keyword right after if you want your result to be in descending order, by default it is ascending.

Revision #1
Created 2023-06-01 13:30:42 UTC by Tamarine
Updated 2023-06-03 03:53:07 UTC by Tamarine