PER SCHOLAS

Lesson - 304.4 - SQL Clauses

SQL Clauses

Learning Objectives:

By the end of this lesson, learners should be able to:

- Describe the Order By, Group By, Having, and Limit clauses.
- Explain the Like and Between Operators.
- Demonstrate the Order By, Group By, Having, and Limit clauses.

Table of Contents

Prerequisite

Note: We will utilize the **classicmodels** database for demonstrations and examples in this presentation.

Overview of ORDER BY Clause

The **ORDER BY** clause is used to sort the records in your result set.

Syntax:

```
SELECT expressions
FROM tables
[WHERE conditions]
ORDER BY
   column1 [ASC|DESC],
   column2 [ASC|DESC],
   ...;
```

When executing the **SELECT** statement with an **ORDER BY** clause, SQL always evaluates the **ORDER BY** clause after the **FROM** and **SELECT**:

ASC:

Optional. ASC sorts the result set in ascending order by *expression* (default, if no modifier is provider).

DESC:

Optional. DESC sorts the result set in descending order by *expression*.

Overview of ORDER BY Clause

Below is the ordering of the **Select** Statement:

```
SELECT (Return Expressions)   [required]
FROM (specify tables)         [required]
WHERE (Row Filter)            [optional]
GROUP BY                      [optional]
HAVING (Group Filter)         [optional]
ORDER BY                      [optional]
```

Below is the execution process of a Select statement into RDBMS:

Example 1: ORDER BY Clause

In this example, we will use the **customers** table from the classicmodels database for the demonstration.

The following query uses the **ORDER BY** clause to sort the customers by their last names in ascending order.

```
SELECT
     contactLastname,
     contactFirstname
FROM      customers
ORDER BY contactLastname;
```

contactLastname	contactFirstname
Accorti	Paolo
Altagar,G M	Raanan
Andersen	Mel
Anton	Carmen
Ashworth	Rachel
Barajas	Miguel
Benitez	Violeta
Bennett	Helen
Berglund	Christina
Bergulfsen	Jonas
Bertrand	Marie
Brown	Julie
Brown	Ann

If you want to sort customers by the **last name** in descending order, you can use the **DESC** attribute after the ***contactLastname*** column in the **ORDER BY** clause as shown in the following query:

```
SELECT   contactLastname, contactFirstname
FROMcustomers
ORDER BY contactLastname DESC;
```

contactLastname	contactFirstname
Young	Jeff
Young	Julie
Young	Mary
Young	Dorothy
Yoshido	Juri
Walker	Brydey
Victorino	Wendy
Urs	Braun

Example 2: ORDER BY Clause

Sort the result by multiple columns.

If you want to sort the customers by the ***last name*** in descending order, and then by the ***first name*** in ascending order, you will specify both **DESC** and **ASC** attributes in these respective columns as follows:

```
SELECT
    contactLastname,
    contactFirstname
FROM
    customers
ORDER BY
     contactLastname DESC ,
     contactFirstname ASC;
```

Output

contactLastname	contactFirstname
Young	Dorothy
Young	Jeff
Young	Julie
Young	Mary
Yoshido	Juri
Walker	Brydey
Victorino	Wendy
Urs	Braun
Tseng	Jerry
Tonini	Daniel
Thompson	Steve
Thompson	Valarie
Taylor	Leslie

In this example, the ORDER BY clause sorts the result set by the last name in descending order first, and then sorts the sorted result set by the first name in ascending order to make the final result set.

Example 3: ORDER BY Clause

Sorting by relative position.

You can also use the **ORDER BY** Clause to sort by relative position in the result set, where the first field in the result set is 1, and the next field is 2, and so on:

```
SELECT  contactLastname, contactFirstname, city
FROM customers
ORDER BY 2 DESC;
```

contactLastname	contactFirstname	city
Piestrzeniewicz	Zbyszek	Warszawa
Choi	Yu	NYC
Tamuri	Yoshi	Vancouver
Huang	Wing	New Bedford
Brown	William	Newark
Victorino	Wendy	Singapore
MacKinlay	Wales	Auckland
Benitez	Violeta	New Bedford
Oeztan	Veysel	Bergen
Thompson	Valarie	San Diego
Franco	Valarie	Boston
Snowden	Tony	Auckland
Smith	Thomas	London

Hands-On LAB

Complete the lab **GLab - 304.4.1 - ORDER BY Clause**, you can find this Lab on Canva under the **Assignment** section.

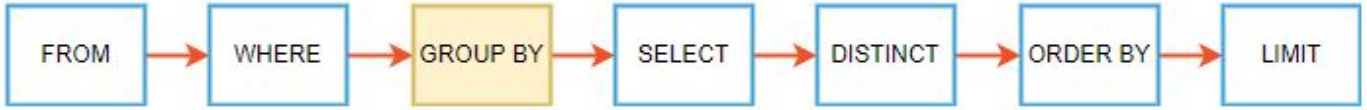
If you have any technical questions while performing the lab activity, ask your instructors for assistance.

GROUP BY Clause

- The **GROUP BY** clause groups a set of rows into a set of summary rows by values of columns. The **GROUP BY** clause returns one row for each group.
- We often use the **GROUP BY** clause with aggregate functions such as **SUM(), AVG(), MAX(), MIN(), COUNT(), and etc.**
  - **Remember:** When we use **GROUP BY** clause in the ***SELECT*** statement without using aggregate functions, it would behave like **DISTINCT** clause.
- The **GROUP BY** clause is an *optional* clause of the SELECT statement.
- GROUP BY clause syntax:

```
SELECT c1, c2,..., cn,
    aggregate_function(ci)
FROM table
WHERE where_conditions
GROUP BY c1 , c2,...,cn;
```

SQL evaluates the **GROUP BY** clause after the **FROM** and **WHERE** clauses and before the **HAVING**, **SELECT, DISTINCT, ORDER BY,** and **LIMIT** clauses:



Example: GROUP BY Without Aggregate Function

The following queries will return the same result. When we use the **GROUP BY** clause in the ***SELECT*** statement without using aggregate functions, it would behave like the **DISTINCT** clause.

```
SELECT Country
FROM customers
GROUP BY country;
```

```
SELECT  distinct(Country)
FROM customers ;
```

Result Grid | Filter R

Country
France
USA
Australia
Norway
Poland
Germany
Spain
Sweden
Denmark
Singapore
Portugal
Japan
Finland

Result Grid | Filter R

Country
France
USA
Australia
Norway
Poland
Germany
Spain
Sweden
Denmark
Singapore
Portugal
Japan
Finland

Example: GROUP BY Clause

- Suppose you want to group values of the order's status into subgroups. To do so, use the GROUP BY clause with the status column as the following query:
  - SELECT **status** FROM **orders** GROUP BY **status**;

status
Cancelled
Disputed
In Process
On Hold
Resolved
Shipped

- If you want to know the number of orders in each status, you can use the **COUNT()** function with the GROUP BY clause as follows:
  - SELECT **status**, **COUNT(*)** FROM orders GROUP BY **status**;

status	COUNT(*)
Cancelled	6
Disputed	3
In Process	6
On Hold	4
Resolved	4
Shipped	303

continue…

Example: GROUP BY Clause (continued)

- The following query returns the order numbers and the total amount of each order.

```
SELECT orderNumber, SUM(quantityOrdered * priceEach) AS total FROM orderdetails GROUP BY orderNumber;
```

orderNumber	total
10100	10223.83
10101	10549.01
10102	5494.78
10103	50218.95
10104	40206.20
10105	53959.21
10106	52151.81
10107	22292.62

- The following query extracts the year from the “orderDate.” It first uses year as an alias of the expression YEAR (orderDate), and then uses the year alias in the GROUP BY clause.

```
SELECT YEAR(orderDate) AS year, COUNT(orderNumber) FROM orders GROUP BY year;
```

year	COUNT(orderNumber)
2003	111
2004	151
2005	64

HAVING Clause

- The **HAVING** clause is used in the **SELECT** statement to specify filter conditions for a group of rows or aggregates.
- The **HAVING** clause is often used with the **GROUP BY** clause to filter groups based on a specified condition. If the **GROUP BY** clause is omitted, the **HAVING** clause behaves like the **WHERE** clause.
- SQL evaluates the **HAVING** clause after the **FROM, WHERE, SELECT, and GROUP BY clauses**, and before **ORDER BY** and **LIMIT clauses**.

- Syntax of the HAVING clause:

```
SELECT select_list FROM table_name
WHERE search_condition
GROUP BY group_by_expression
HAVING group_condition;
```

Notice that the HAVING clause applies a filter condition to each group of rows while the WHERE clause applies the filter condition to each individual row.

Example: Choosing Groups - HAVING

- The following query uses the GROUP BY clause to get order numbers, the number of items sold per order, and total sales for each from the orderdetails table:

```
SELECT ordernumber,
    SUM(quantityOrdered) AS itemsCount,
    SUM(priceeach*quantityOrdered) AS total
FROM orderdetails
   GROUP BY ordernumber;
```

ordernumber	itemsCount	total
10100	151	10223.83
10101	142	10549.01
10102	80	5494.78
10103	541	50218.95
10104	443	40206.20
10105	545	53959.21
10106	675	52151.81
10107	229	22292.62

- Now you can find which order has a total sale greater than 20,000 by using the **HAVING** clause as follows:

```
SELECT ordernumber,
      SUM(quantityOrdered) AS itemsCount,
      SUM(priceeach*quantityOrdered) AS total
FROM orderdetails
     GROUP BY ordernumber
     HAVING total > 20000;
```

ordernumber	itemsCount	total
10103	541	50218.95
10104	443	40206.20
10105	545	53959.21
10106	675	52151.81
10107	229	22292.62
10108	561	51001.22
10109	212	25833.14
10110	570	48425.69
10114	351	33383.14
10115	210	21665.98

Example: Choosing Groups - HAVING (continued)

- You can construct a complex condition in the HAVING clause by using logical operators such as **OR** and **AND**.
- The following example uses the **HAVING** clause to find orders that have total amounts greater than 1,000 and contain more than 600 items:

```
SELECT ordernumber, SUM(quantityOrdered) AS itemsCount,
    SUM(priceeach*quantityOrdered) AS total
FROM orderdetails
GROUP BY ordernumber
HAVING  total > 1000    AND   itemsCount > 600;
```

ordernumber	itemsCount	total
10106	675	52151.81
10126	617	57131.92
10135	607	55601.84
10165	670	67392.85
10168	642	50743.65
10204	619	58793.53
10207	615	59265.14
10212	612	59830.55
10222	717	56822.65

Important Facts About the Where and Having Clauses

We cannot use aggregate functions within the **WHERE** clause. The query below will generate an error:
ERROR CODE 1111. Invalid use of group function.

```
Select customerNumber, avg(amount) as avgAmount from payments p
Where avg(amount)>1000
group by customerNumber ;
```

The **WHERE** clause filters *individual rows,* and cannot be used with multi-row functions. Instead, we must use the **HAVING** clause, as shown in the query below:

```
Select customerNumber, avg(amount) as avgAmount
from payments p
group by customerNumber
having avg(amount)>1000;
```

This gives us the expected single row:

Result Grid | Filter Rows:

customerNumber	avgAmount
103	7438.120000
112	20359.785000
114	45146.267500
119	38983.226667
121	26056.197500
124	64909.804444
128	15872.804000
129	22236.853333
131	35879.980000
141	55056.844615
144	21840.325000
145	26861.625000
146	43435.116667

Subqueries in the HAVING Clause

The **HAVING** clause supports the use of subqueries. For example, our previous query is a bit fragile – as more amounts are added to the payment table, the average amount will change and our query will be invalid due to the hard-coded value. We can make it dynamic by using a subquery.

```
Select customerNumber, avg(amount)  as avgAmount
from payments p1
group by customerNumber
    having avg(amount) > (
        Select avg(amount) from payments p2
    ) order by 2 desc;
```

Output

Result Grid | Filter Rows:

customerNumber	avgAmount
239	80375.240000
321	66170.390000
124	64909.804444
450	59551.380000
141	55056.844615
298	54388.960000
187	49470.030000
157	49254.625000
167	48781.235000
211	45480.790000
286	45272.685000
114	45146.267500
386	45071.655000

Result 25

We will learn about Subqueries in more depth in the next lectures.

LIMIT Operator

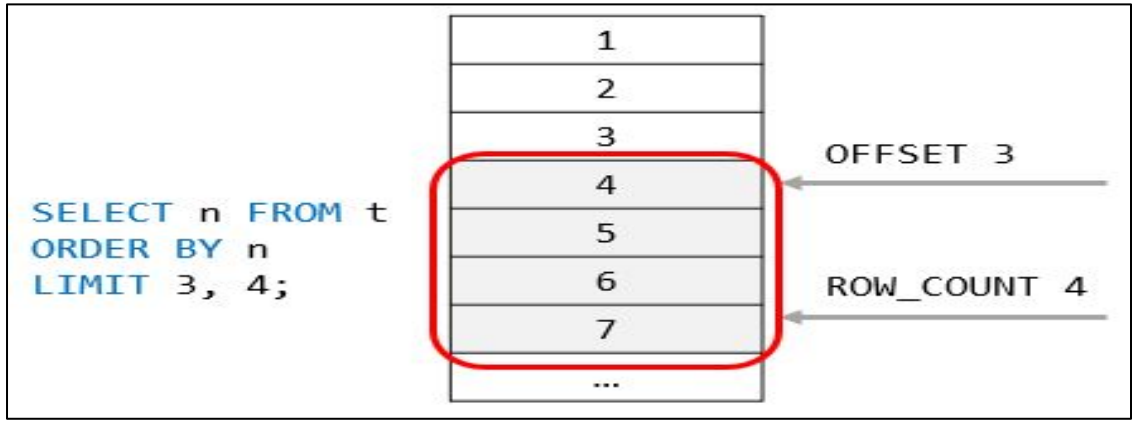
The **LIMIT** operator is used in the **SELECT** statement to constrain the number of rows to return. The LIMIT operator accepts one or two arguments. The values of both arguments must be zero, or positive integers.

Syntax: LIMIT Operator

```
SELECT  COLUMN_NAME   FROM  table_name
LIMIT [offset,] row_count;
```

In this syntax:

- **The offset specifies the offset of the first row to return. The offset of the first row is 0, not 1.**
- **The row_count specifies the maximum number of rows to return.**
- The following picture illustrates the LIMIT clause:



Example: LIMIT Operator

```
SELECT customerNumber, customerName, creditLimit
FROM customers
ORDER BY creditLimit DESC
LIMIT 5,2;
```

Output:

Result Grid | Filter Rows: | Edit:

customerNumber	customerName	creditLimit
146	Saveley & Henriot, Co.	123900.00
286	Marta's Replicas Co.	123700.00

LIMIT Operator and ORDER BY Clause

- By default, the **SELECT** statement returns rows in an unspecified order. When you add the **LIMIT** operator to the **SELECT** statement, the returned rows are unpredictable.
- Therefore, to ensure that the **LIMIT** operator returns an expected output, you should always use it with an **ORDER BY** clause.

Example

```
SELECT
    customerNumber,
    customerName,
    creditLimit
FROM   customers
ORDER BY creditLimit DESC
LIMIT 5;
```

In this example:

- First, the ORDER BY clause sorts the customers by credits in high to low.
- Then, the LIMIT operator returns the first 5 rows.

Example: The LIMIT and ORDER BY Clause

The following query finds the customer who has the second-highest credit limit.

```
SELECT
    customerName,
    creditLimit
FROM    customers
ORDER BY     creditLimit DESC
LIMIT 1,1;
```

Output

	customerName	creditLimit
▶	Mini Gifts Distributors Ltd.	210500.00

LIKE Operator

- The **LIKE** operator is used in the **WHERE** clause with ***SELECT , DELETE,*** and ***UPDATE*** *statements* to filter data based on patterns or searches for a specified pattern in a column.
- There are two wildcards used in conjunction with the LIKE operator:
  - The percent sign **"%"** represents zero, one, or multiple characters.
  - The underscore " **_**" wildcard matches any single character.
- For example, **s%** matches any string starts with the character "*s*" (e.g., **s***un* or **s***ix*).
- The **se_** matches any string that starts with "*se*" and is followed by any character (e.g., **se***e* or **se***a*).
- **Syntax:**

```
SELECT column1, column2, ...
    FROM table_name
    WHERE columnN LIKE pattern;
```

LIKE Operator Wildcards

Like operator	Description
WHERE CustomerName LIKE 'a%'	Finds any values that starts with "a"
WHERE CustomerName LIKE '%a'	Finds any values that ends with "a"
WHERE CustomerName LIKE '%or%'	Finds any values that have "or" in any position
WHERE CustomerName LIKE '_r%'	Finds any values that have "r" in the second position
WHERE CustomerName LIKE 'a_%_%'	Finds any values that starts with "a" and are at least 3 characters in length
WHERE ContactName LIKE 'a%o'	Finds any values that starts with "a" and ends with "o"

Example: LIKE Operator Wildcards

The following SQL statement selects all customers with a CustomerName starting with "a":

```
SELECT * FROM Customers
WHERE CustomerName LIKE  'a%';
```

The following SQL statement selects all customers with a CustomerName that have “or" in any position:

```
SELECT * FROM Customers
WHERE CustomerName LIKE  '%or%';
```

continue….

Example: LIKE Operator Wildcards (continued)

The following SQL statement selects all customers with a `contactFirstName` that starts with **"a"** and ends with **"o"**:

```
SELECT * FROM Customers

WHERE contactFirstName  LIKE 'a%o';
```

In the below query we used the **LIKE** clause to find employees whose last names end with **"on"** (e.g., Patterson, Thompson)

```
SELECT employeeNumber, lastName, firstName FROM    employees
WHERE lastName LIKE '%on';
```

Practice Assignment

This assignment will be administered through HackerRank. Make sure to select the MySQL database in the drop down box above where you enter your code.

➔ **Click here for Hackerank link - Weather Observation Station 6.**

Note: Use your office hours to complete this assignment. If you have any technical questions while performing the practice assignment activity, ask your instructors for assistance.

BETWEEN Operator

- The **BETWEEN** operator selects values within a given range. The values can be numbers, text, or dates.
- The **BETWEEN** operator is inclusive: ***begin*** and ***end*** values are included.

BETWEEN clause Syntax:

```
SELECT column_name(s)
FROM table_name
WHERE column_name BETWEEN value1 AND value2;
```

Example 1: BETWEEN Operator

The following example uses the **BETWEEN** operator to find products whose buy prices are between $90 and $100.

```
SELECT   productCode, productName, buyPrice
FROM    products
WHERE  buyPrice BETWEEN 90 AND 100;
```

To find the products whose buy prices are NOT between $20 and $100, you can use the **NOT BETWEEN** operator as follows:

```
SELECT     productCode,  productName, buyPrice
FROM   products
WHERE   buyPrice NOT BETWEEN 20 AND 100;
```

Example 2: BETWEEN Operator

The following SQL statement selects all products with a price **BETWEEN** $10 and $20, and no products with a productLine of `'S10_1678'` or `'S10_1949'`

```
SELECT * FROM Products WHERE (buyPrice BETWEEN 10 AND 20)
AND NOT productLine IN ('S10_1678','S10_1949');
```

Output:

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

productCode	productName	productLine	productScale	productVendor	productDescription	quantityInStock	buyPrice
S24_2840	1958 Chevy Corvette Limited Edition	Classic Cars	1:24	Carousel DieCast Legends	The operating parts of this 1958 Chevy Corvett...	2542	15.91
S24_2972	1982 Lamborghini Diablo	Classic Cars	1:24	Second Gear Diecast	This replica features opening doors, superb det...	7723	16.24
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Example 3: Operator

```
SELECT * FROM Products
WHERE ProductName
BETWEEN 'Carnarvon Tigers' AND 'Mozzarella di Giovanni'
 ORDER BY ProductName;
```

Output:

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

productCode	productName	productLine	productScale	productVendor	productDescription	quantityInStock	buyP
S18_3259	Collectable Wooden Train	Trains	1:18	Carousel DieCast Legends	Hand crafted wooden toy train set is in about 1...	6450	67.56
S24_3949	Corsair F4U (Bird Cage)	Planes	1:24	Second Gear Diecast	Has retractable wheels and comes with a stand....	6812	29.34
S50_1392	Diamond T620 Semi-Skirted Tanker	Trucks and Buses	1:50	Highway 66 Mini Classics	This limited edition model is licensed and perfectl...	1016	68.29
S700_3167	F/A 18 Hornet 1/72	Planes	1:72	Motor City Art Classics	10" Wingspan with retractable landing gears.Co...	551	54.40
S700_2047	HMS Bounty	Ships	1:700	Unimax Art Galleries	Measures 30 inches Long x 27 1/2 inches High x...	3501	39.83
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Practice Assignment

Complete the assignment **PA 304.4.1 - Practice Assignment - Simple Queries.** You can find this assignment on Canva under the Assignment section.

Note: Use your office hours to complete this assignment. If you have any technical questions while performing the assignment activity, ask your instructors for assistance.

Knowledge Check

- How does the ORDER BY Clause work in SQL?
- When do we use a HAVING clause?
- Which is the default order of Sort in the ORDER BY Clause?
- What is the meaning of the GROUP BY Clause in Mysql?
- What SQL clause is used to restrict the rows returned by a query?
- What is the difference between percent sign (%) and the underscore (_) for pattern matching (e.g. in the LIKE operator)?

Summary

The MySQL ORDER BY clause is used to sort the records in a result set. The MySQL WHERE clause is used to filter the results from a SELECT, INSERT, UPDATE, or DELETE statement.

- The MySQL GROUP BY clause is used in a SELECT statements to collect data across multiple records and group the results by one or more columns.
- The MySQL HAVING clause is used in combination with the GROUP BY clause to restrict the groups of returned rows to only those whose the condition is TRUE.
- The MySQL LIMIT clause is used to retrieve records from one or more tables in MySQL and limit the number of records returned based on a limit value.
- The MySQL BETWEEN Condition is used to retrieve values within a range in a SELECT, INSERT, UPDATE, or DELETE statement.
- The MySQL LIKE condition allows wildcards to be used in the WHERE clause of a SELECT, INSERT, UPDATE, or DELETE statement. This allows you to perform pattern matching.
- MySQL system clauses are keywords or statements to handle information. It helps to operate a group of the data and apply it to require conditions. The clauses apply conditions or select patterns to get information. MySQL clauses are not used to insert new data. You retrieve data using several clauses. The table uses either single or multiple clauses.

References

https://www.techonthenet.com/mysql/group_by.php

https://www.techonthenet.com/mysql/between.php

Questions?