Guided LAB - 304.5.1 - Aggregate Functions

Objective

In this lab, you will demonstrate the aggregate function of SQL.

Learning Objective:

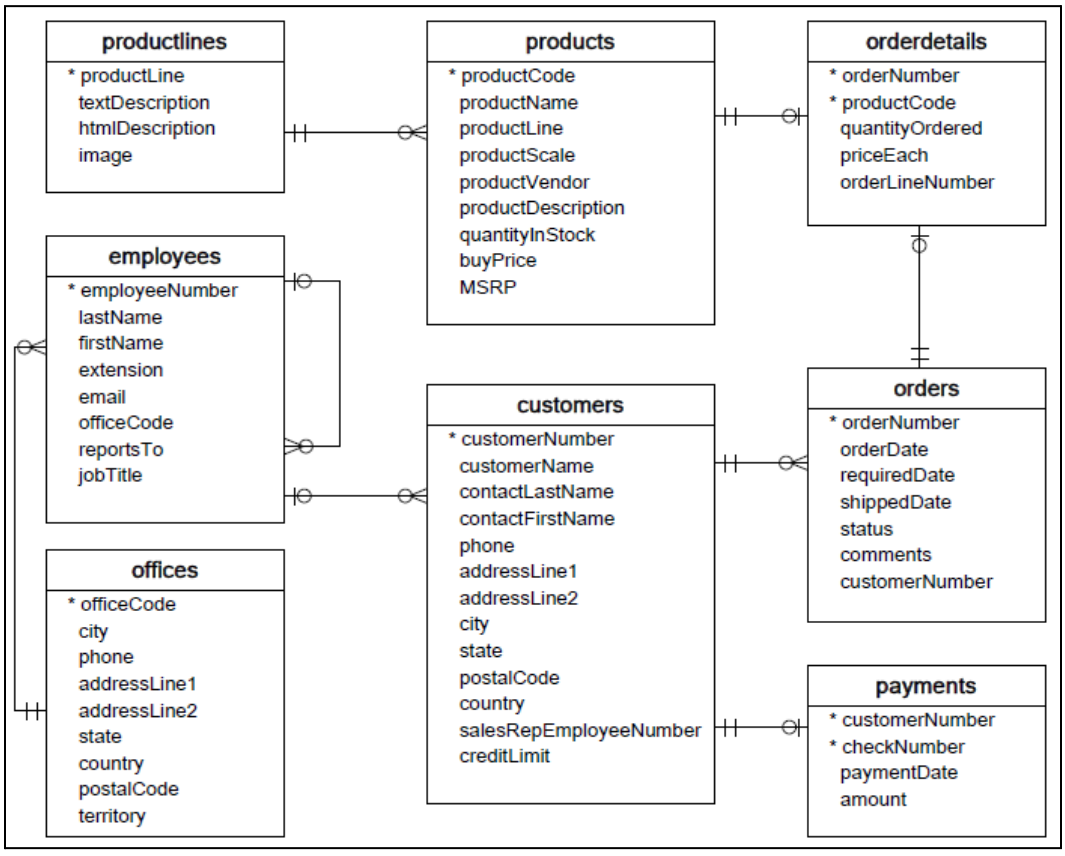
By the end of this lab, learners will be able to:

- Define the Aggregate Function.
- Use Aggregate Function in MySQL.

Prerequisites:

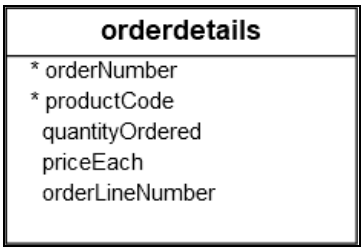
For this lab, you must have a "**classicmodesl"** database. If you do not have a '**classicmodels '** database setup, [click here to download the database script file.]

The database Schema



1. Using MOD() function.

\We will use the orderDetails table from the sample database for the demonstration:



The following statement finds whether the quantity of products, which customers ordered, is odd or even.

```
SELECT orderNumber, SUM(quantityOrdered) as Qty,
    IF(MOD(SUM(quantityOrdered),2),'Odd', 'Even') as oddOrEven
FROM    orderdetails
GROUP BY    orderNumber
ORDER BY    orderNumber;
```

In this example:

- First, we used the SUM() function to get the total quantity of products by the sales order.
- Then we used the MOD() function to find the remainder of the total quantity divided by two. This results in zero or one, depending on where the total quantity is (even or odd).
- Finally, we used the IF() function to display the Odd and Even string based on the result of the MOD() function.

Here is the output:

orderNumber	Qty	oddOrEven
10100	151	Odd
10101	142	Even
10102	80	Even
10103	541	Odd
10104	443	Odd
10105	545	Odd
10106	675	Odd
10107	229	Odd
10108	561	Odd

2. TRUNCATE() Function

Let's review some examples of using the TRUNCATE() function, using SQL TRUNCATE() with a positive number of decimal places.

```
SELECT TRUNCATE(1.555,1);
```

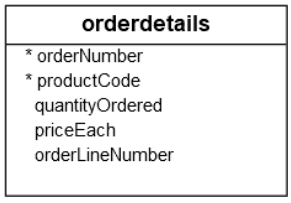
Here is the **output**:

	TRUNCATE(1.555,1)
▶	1.5

Because the number of decimal places argument is 1, the TRUNCATE() function keeps only one decimal place in the return value.

3. ROUND() function

We will use the orderDetails table from the sample database for the demonstration.



The following query finds the average order line item values by product codes:

```
SELECT productCode, AVG(quantityOrdered * priceEach) as avg_order_value
FROM orderDetails
GROUP BY productCode;
```

Here is the output:

	productCode	avg_order_value
▶	S12_1108	7065.031852
	S10_1949	6786.355714
	S10_4698	6095.928571
	S12_1099	5982.647407
	S12_3891	5649.741481
	S18_1749	5621.424000
	S18_3232	5223.395849

The average order values of products are not quite readable because they contain many numbers after the decimal points.

For the average values, the number after the decimal point may not be important. Therefore, you can use the ROUND() function to round them to zero decimal places, as shown in the following query:

```
SELECT     productCode,
  ROUND(AVG(quantityOrdered * priceEach)) as avg_order_item_value
FROM     orderDetails
GROUP BY     productCode;
```

The following picture shows the output:

	productCode	avg_order_value
▶	S12_1108	7065
	S10_1949	6786
	S10_4698	6096
	S12_1099	5983
	S12_3891	5650
	S18_1749	5621
	S18_3232	5223
	S18_1662	5177

SQL TRUNCATE() vs. ROUND()

The following example uses both TRUNCATE() and ROUND() function for comparison:

```
SELECT    TRUNCATE(1.999,1),   ROUND(1.999,1);
```

Here is the query output:

	TRUNCATE(1.999,1)	ROUND(1.999,1)
▶	1.9	2.0

As clearly shown in the output, the TRUNCATE() function only trims the decimal places while the ROUND() function performs the rounding.

4. REPLACE() Function

The **syntax** of using the REPLACE() function in an UPDATE statement is as follows:

```
UPDATE tbl_name SET
    field_name = REPLACE(field_name, string_to_find, string_to_replace)
WHERE    conditions;
```

Note that when searching for text to replace, SQL uses the case-sensitive match to perform a search for a string to be replaced.

For example, if you want to correct the spelling mistake in the products table in the sample database, you can use the REPLACE() function as follows:

```
UPDATE products
SET productDescription = REPLACE(productDescription,'abuot','about');
```

The above query will find all occurrences of a spelling mistake 'abuot,' and replaces it by the correct word '*about*' in the productDescription column of the products table.

5. DATEDIFF() function

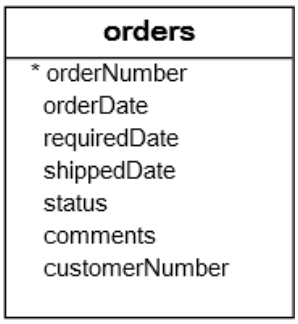
1. Let's take a look at some examples of using the DATEDIFF() function.

```
SELECT DATEDIFF('2011-08-17','2011-08-17');
#Result  :   0 day
```

```
SELECT DATEDIFF('2011-08-17','2011-08-08');
#Result:  9 days
```

```
SELECT DATEDIFF('2011-08-08','2011-08-17');
#Result:  -9 days
```

2. See the following orders table in the sample database.



a) To calculate the number of days between the required date and shipped date of the orders, we can use the DATEDIFF() function as follows:

```
SELECT orderNumber, DATEDIFF(requiredDate, shippedDate) as  daysLeft
FROM     orders
ORDER BY  daysLeft DESC;
```

Result:

orderNumber	daysLeft
10409	11
10410	10
10105	9
10135	9
10190	9
10201	9
10254	9

b) The following statement gets all orders whose status is “In Process,” and calculates the number of days between the ordered date and the required date:

```
SELECT orderNumber, DATEDIFF(requiredDate, orderDate) as remaining_days
FROM    orders
WHERE     status = 'In Process'
ORDER BY remaining_days;
```

Result:

orderNumber	remaining_days
10423	6
10425	7
10421	8
10424	8
10420	9
10422	12

c) For calculating an interval in week or month, you can divide the returned value of the DATEDIFF() function by 7 or 30 as the following query:

```
SELECT
   orderNumber,
   ROUND(DATEDIFF(requiredDate, orderDate) / 7, 2),
   ROUND(DATEDIFF(requiredDate, orderDate) / 30,2)
FROM    orders
WHERE   status = 'In Process';
```

Result

orderNumber	round(DATEDIFF(requiredDate, orderDate) / 7,2)	round(DATEDIFF(requiredDate, orderDate) / 30,2)
10420	1.29	0.30
10421	1.14	0.27
10422	1.71	0.40
10423	0.86	0.20
10424	1.14	0.27
10425	1.00	0.23

Note that the ROUND() function is used to round the results.

6. DATE_FORMAT() Function

Specifier	Meaning
%a	Three-characters abbreviated weekday names e.g., Mon, Tue, Wed, etc.
%b	Three-characters abbreviated month name e.g., Jan, Feb, Mar, etc.
%c	Month in numeric e.g., 1, 2, 3…12
%D	Day of the month with English suffix e.g., 0th, 1st, 2nd, etc.
%d	Day of the month with leading zero if it is 1 number e.g., 00, 01,02, …31
%e	Day of the month without leading zero e.g., 1,2,…31
%f	Microseconds in the range of 000000..999999
%H	Hour in 24-hour format with leading zero e.g., 00..23
%h	Hour in 12-hour format with leading zero e.g., 01, 02…12
%I	Same as %h
%i	Minutes with leading zero e.g., 00, 01,…59
%j	Day of year with leading zero e.g., 001,002,…366
%k	Hour in 24-hour format without leading zero e.g., 0,1,2…23
%l	Hour in 12-hour format without leading zero e.g., 1,2…12
%M	Full month name e.g., January, February,…December
%m	Month name with leading zero e.g., 00,01,02,…12
%p	AM or PM, depending on other time specifiers
%r	Time in 12-hour format hh:mm:ss AM or PM
%S	Seconds with leading zero 00,01,…59
%s	Same as %S
%T	Time in 24-hour format hh:mm:ss
%U	Week number with leading zero when the first day of week is Sunday e.g., 00,01,02…53
%u	Week number with leading zero when the first day of week is Monday e.g., 00,01,02…53
%V	Same as %U; it is used with %X
%v	Same as %u; it is used with %x

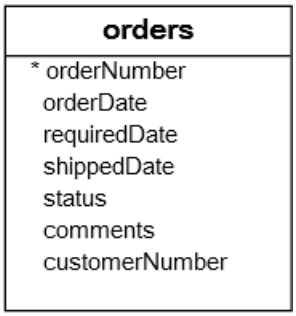
%W	Full name of weekday e.g., Sunday, Monday,…, Saturday
%w	Weekday in number (0=Sunday, 1= Monday,etc.)
%X	Year for the week in four digits where the first day of the week is Sunday; often used with %V
%x	Year for the week, where the first day of the week is Monday, four digits; used with %v
%Y	Four digits year e.g., 2000 and 2001.
%y	Two digits year e.g., 10,11,and 12.
%%	Add percentage (%) character to the output

The following are some commonly used date format strings:

DATE_FORMAT string	Formatted date
%Y-%m-%d	7/4/2013
%e/%c/%Y	4/7/2013
%c/%e/%Y	7/4/2013
%d/%m/%Y	4/7/2013
%m/%d/%Y	7/4/2013
%e/%c/%Y %H:%i	4/7/2013 11:20
%c/%e/%Y %H:%i	7/4/2013 11:20
%d/%m/%Y %H:%i	4/7/2013 11:20
%m/%d/%Y %H:%i	7/4/2013 11:20
%e/%c/%Y %T	4/7/2013 11:20
%c/%e/%Y %T	7/4/2013 11:20
%d/%m/%Y %T	4/7/2013 11:20
%m/%d/%Y %T	7/4/2013 11:20
%a %D %b %Y	Thu 4th Jul 2013
%a %D %b %Y %H:%i	Thu 4th Jul 2013 11:20

%a %D %b %Y %T	Thu 4th Jul 2013 11:20:05
%a %b %e %Y	Thu Jul 4 2013
%a %b %e %Y %H:%i	Thu Jul 4 2013 11:20
%W %D %M %Y	Thursday 4th July 2013
%W %D %M %Y %H:%i	Thursday 4th July 2013 11:20
%W %D %M %Y %T	Thursday 4th July 2013 11:20:05
%l:%i %p %b %e, %Y	7/4/2013 11:20
%M %e, %Y	4-Jul-13
%a, %d %b %Y %T	Thu, 04 Jul 2013 11:20:05

Let's take a look at the orders table in the sample database.



To select the order's data and format the date value, you can use the query statement:

```
SELECT
    orderNumber,
    DATE_FORMAT(orderdate, '%Y-%m-%d') orderDate,
    DATE_FORMAT(requireddate, '%a %D %b %Y') requireddate,
    DATE_FORMAT(shippedDate, '%W %D %M %Y') shippedDate
FROM    orders;
```

Result:

orderNumber	orderDate	requireddate	shippedDate
10100	2003-01-06	Mon 13th Jan 2003	Friday 10th January 2003
10101	2003-01-09	Sat 18th Jan 2003	Saturday 11th January 2003
10102	2003-01-10	Sat 18th Jan 2003	Tuesday 14th January 2003
10103	2003-01-29	Fri 7th Feb 2003	Sunday 2nd February 2003
10104	2003-01-31	Sun 9th Feb 2003	Saturday 1st February 2003
10105	2003-02-11	Fri 21st Feb 2003	Wednesday 12th February 2003
10106	2003-02-17	Mon 24th Feb 2003	Friday 21st February 2003

We formatted the order date, required date, and shipped date of each order based on different date formats specified by the format strings.

Let's use DATE_FORMAT() with the ORDER BY clause and review the following example:

```
SELECT     orderNumber,
    DATE_FORMAT(shippeddate, '%W %D %M %Y')  as 'Shipped date'
FROM    orders
ORDER BY shippeddate;
```

Result:

orderNumber	Shipped date
10100	Friday 10th January 2003
10101	Saturday 11th January 2003
10102	Tuesday 14th January 2003
10104	Saturday 1st February 2003
10103	Sunday 2nd February 2003
10105	Wednesday 12th February 2003
10106	Friday 21st February 2003

7. LPAD(*str, len, padstr*)

- The LPAD() function left-pads a string with another string to a certain length.
- LPAD() function returns the string ***str***, left-padded with the string ***padstr*** to a length of ***len*** characters. If ***str*** is longer than ***len***, the return value is shortened to ***len*** characters.

Click here for more details.

Example:

```
SELECT LPAD('hi',4,'??');        #Result    -> '??hi'
```

```
SELECT LPAD('hi',1,'??');      # Result -> 'h'
```

Let's take a look at the employees table in the 'classicmodels' database.

```
SELECT firstName, LPAD(firstName,10,'kk'), LPAD(firstName,5,'kk'),
LPAD(firstName,4,'kk') FROM classicmodels.employees;
```

firstName	LPAD(firstName,10,'kk')	LPAD(firstName,5,'kk')	LPAD(firstName,4,'kk')
Diane	kkkkkDiane	Diane	Dian
Mary	kkkkkkMary	kMary	Mary
Jeff	kkkkkkJeff	kJeff	Jeff
William	kkkWilliam	Willi	Will
Gerard	kkkkGerard	Gerar	Gera
Anthony	kkkAnthony	Antho	Anth
Leslie	kkkkLeslie	Lesli	Lesl
Leslie	kkkkLeslie	Lesli	Lesl
Julie	kkkkkJulie	Julie	Juli
Steve	kkkkkSteve	Steve	Stev
Foon Yue	kkFoon Yue	Foon	Foon
George	kkkkGeorge	Georg	Geor
Loui	kkkkkkLoui	kLoui	Loui
Gerard	kkkkGerard	Gerar	Gera
Pamela	kkkkPamela	Pamel	Pame
Larry	kkkkkLarry	Larry	Larr
Barry	kkkkkBarry	Barry	Barr
Andy	kkkkkkAndy	kAndy	Andy
Peter	kkkkkPeter	Peter	Pete
Tom	kkkkkkkTom	kkTom	kTom

7. SQL TRIM() Function

The data from the user's input is typically not what we expected. Sometimes, it is not well-formed (e.g., wrong cases; those containing leading and trailing spaces and other unwanted characters).

To keep the data in the correct format, before inserting or updating data in the database, you need to clean it up. One of the most important tasks in data cleansing is to remove unwanted leading and trailing characters.

SQL provides a very useful string function named TRIM to help you clean up the data. The following illustrates the syntax of the TRIM function.

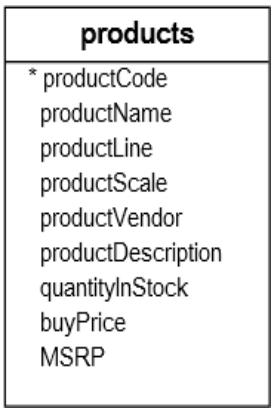
```
1 TRIM([{BOTH|LEADING|TRAILING} [removed_str]] FROM str);
```

The TRIM function provides several options

- You can use the LEADING, TRAILING, or BOTH option to explicitly instruct the TRIM function to remove leading, trailing, or both leading and trailing unwanted characters from a string.
- If you do not specify anything, the TRIM function uses the BOTH options by default.
- The [removed_str] is the string that you want to remove. By default, it is a space. It means that if you do not specify a particular string, the TRIM function removes spaces only.
- The TRIM function returns a string that has unwanted characters removed.
- The following statement removes both leading and trailing spaces from a string.

```
SELECT TRIM(' SQL TRIM Function ');
```

We will take the products table in the sample database for the demonstration.



SQL LTRIM and RTRIM function

If you want to remove only leading or trailing spaces, you can use other string functions such as LTRIM and RTRIM.

The following statement uses the LTRIM function to remove the leading spaces of a string.

```
SELECT LTRIM('  SQL LTRIM function');
```

	LTRIM(' MySQL LTRIM function')
▶	MySQL LTRIM function

The following statement uses the RTRIM function to remove the trailing spaces of a string.

```
SELECT RTRIM('SQL RTRIM function   ');
```

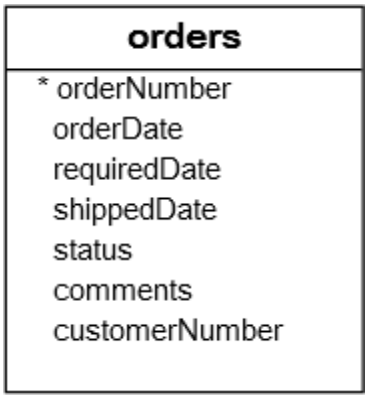
	RTRIM('MySQL RTRIM function ')
▶	MySQL RTRIM function

YEAR() Function

- The YEAR() function takes a date argument and returns the year of the date.
- The YEAR() function returns a year value in the range 1000 to 9999. If the date is zero, the YEAR() function returns 0.
- The following example returns the year January 1st, 2022, which is 2022.

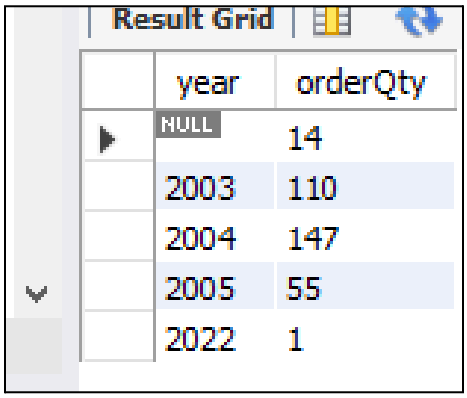
```
SELECT YEAR('2002-01-01');
```

- Let's consider the orders table in the **classicmodels** database.



The following query uses the **YEAR()** function to get the number of orders shipped per year.

```
SELECT YEAR(shippeddate) as year,  COUNT(ordernumber) as
orderQty
FROM    orders
GROUP BY YEAR(shippeddate)
ORDER BY YEAR(shippeddate);
```


Result Grid

year	orderQty
NULL	14
2003	110
2004	147
2005	55
2022	1

In this example, we use the YEAR() function to extract year information out of the shipped date and use the COUNT() function to count the number of delivered orders. The GROUP BY clause groups the number of orders by year.

DAY() Function

- The DAY() function returns the day of the month of a given date.
- The DAY() function accepts one argument, which is a date value for which you want to get the day of the month. If the date argument is zero e.g., '0000-00-00', the DAY() function returns 0. If the date is NULL, the DAY() function returns NULL.
- Note that the DAY() function is the synonym of the DAYOFMONTH() function.

The following example returns the day of the month of 2022-01-15:

```
SELECT DAY('2022-01-15');
```

Consider the following orders table.

orders
* orderNumber
orderDate
requiredDate
shippedDate
status
comments
customerNumber

The following statement uses the DAY() function to return the number of orders by day number in 2004.

```
SELECT  DAY(orderdate) as dayofmonth, COUNT(*)
FROM    orders WHERE    YEAR(orderdate) = 2004
GROUP BY dayofmonth
ORDER BY dayofmonth;
```

dayofmonth	COUNT(*)
1	6
2	5
3	4
4	5
5	6
6	7
7	3
8	4
9	3
10	6
11	6
12	4
13	2
14	3
15	1
16	4
17	2
18	4
19	2
20	4
21	7
22	1
23	1
24	3
25	6
26	3
27	2
28	4
29	2
31	1

Submission Instructions:

Canvas submission Instructions: Include the following deliverables in your submission -

- All queries should be written and submitted in a single SQL script file.
  - Example :**<your_name_labname>.sql**.
- **Do not add the questions in your SQL script file.**
- Submit your SQL script file using the **Start Assignment** button in the top-right corner of the assignment page in Canvas.