Promineo Tech Lesson - Week 7

promineotech.openclass.ai/resource/lesson-659c2dc3fe775297cad773d6

MySQL Part 2

This lesson contains Questions 11-20 (Part 2) of the OpenClass MySQL Lesson, and contains five (5) coding questions, each followed by a mastery question.

Some **Review** Sections are included here from the **MySQL Part 1** lesson.

Each OpenClass question has a database attached to it. The only statement required in the Solution Box is the query requested.

11. SELECT Statement - city & country using `COUNT(*)`

Retrieve the count of the rows of the city table and the country table only counting the city names that start with "Ab". Use the `JOIN` keyword, and join on the key column in common between the two tables.

SELECT COUNT(*) FROM city JOIN country USING (country_id) WHERE city LIKE 'Ab%';

Expected STDOUT

COUNT(*)
2

Your STDOUT

COUNT(*)
2

SELECT COUNT(*) FROM city

INNER JOIN country USING (country_id)

WHERE city LIKE 'Ab%';

12. SELECT Statement - city & country using `COUNT(*)` and a column alias

Retrieve the count of the rows, with a column alias of "Count", of the city table and the country table only counting the city names that start with "Ab". Use the `JOIN` keyword, and join on the key column in common between the two tables.

SELECT COUNT(*) AS 'Count' FROM city JOIN country USING (country_id) WHERE city LIKE 'Ab%';

Expected STDOUT

Count
2

Your STDOUT

Count
2

Instructor Solution

SELECT COUNT(*) AS "Count" FROM city

INNER JOIN country USING (country_id)

WHERE city LIKE 'Ab%';

More Aggregate Functions

Remember, aggregate functions are allowed in two places in SQL, in a `SELECT` clause or in a `HAVING` clause. Here is a more in-depth look at additional SQL aggregate functions. To use many of the functions provided in the SQL language, it is important to utilize another SQL clause, the `GROUP BY` clause. In our above examples of `COUNT()`, we did not use a `GROUP BY`; however, the `GROUP BY` clause is often used with aggregate functions to allow SQL to group the result set by one or more columns.

The SQL `GROUP BY` clause is available for use in a `SELECT` statement. This clause collects data or results across multiple records and groups the results by one or more columns. The `GROUP BY` clause will return one row per group. Another way to describe this is that `GROUP BY` groups a result into subsets that have matching values for one or more columns.

- `COUNT()` -- returns the number of elements in the set provided
- `MAX()` -- returns the maximum value of the set of values provided

- `MIN()` -- returns the minimum value of the set of values provided
- `AVG()` -- returns the average of the set of values provided
- `SUM()` -- returns the sum of all values in the set of values provided

`AVG()` Example

In the **Sakila** database, there is a payment table. The columns in the payment table are as follows:

```
Table Name                      Column Names
-------------------------------------------------------------------------------
payment                         payment_id, customer_id (FK), staff_id (FK),
                                rental_id (FK), amount, payment_date, last_update
-------------------------------------------------------------------------------
```

To ***find the average amount spent on a video rental (e.g. using the `amount` column) for each customer***, the `SELECT` statement will use an aggregate function, `AVG()` on the `amount` column. What does the following query return?

```
SELECT AVG(amount) FROM payment;
```

The above query returns the average of every payment that has ever been recorded in this table. Remember, originally we asked to ***find the average amount spent on a video rental (e.g. using the `amount` column) for each customer***. Notice that the above query does not mention the customer at all, just an average. Try the following:

```
SELECT AVG(amount), customer_id FROM payment;
```

Note that the above query also does not do what we think it should. The above query will return the average amount spent on all of the rows, and it will return a random customer_id (probably the first one), but because we did not include a `GROUP BY` clause, we are not getting the average spent **per customer**.

When using the aggregate functions, it is required to use a `GROUP BY` clause to get the correct grouping of information. The following query will indeed return the average amount spent per customer_id. The `LIMIT` clause was added, or the query would have returned a row for every customer stored in the database.

```
SELECT AVG(amount), customer_id FROM payment GROUP BY customer_id LIMIT 5;
```

13. SELECT Statement - payment using `AVG()` & `GROUP BY`

Retrieve the `customer_id`, and the average of the `amount` paid for a rental in the `payment` table per customer. Use the `AVG()` aggregate function and the `GROUP BY` clause. Limit your results to the first 5 rows.

SELECT customer_id, AVG(amount) FROM payment GROUP BY customer_id LIMIT 5;

Expected STDOUT

customer_id	AVG(amount)
1	3.7087499999999975
2	4.767777777777775
3	5.212222222222222
4	3.7172727272727273
5	3.80578947368421

Your STDOUT

customer_id	AVG(amount)
1	3.7087499999999975
2	4.767777777777775
3	5.212222222222222
4	3.7172727272727273
5	3.80578947368421

Instructor Solution

SELECT customer_id, AVG(amount) FROM payment

GROUP BY customer_id LIMIT 5;

`ROUND()` Function

Notice in the above question, that the average is printed out with many decimal places. There is a function that is provided in SQL that will round the result to the number of requested decimal places. That function is `ROUND()`, and it takes two parameters. The first parameter is the number to be rounded, and the second parameter is the number of decimal places requested.

```
SELECT ROUND(AVG(amount),2) FROM payment;
```

14. SELECT Statement - payment using `AVG()` , `GROUP BY` and `ROUND()`

Retrieve the `customer_id`, and the average of the `amount` paid for a rental in the `payment` table per customer, rounded to two (2) decimal places. Limit your results to the first 5 rows.

If you are struggling with the `ROUND()` function, refer to the example above.

SELECT customer_id, ROUND(AVG(amount),2) FROM payment GROUP BY customer_id LIMIT 5;

Expected STDOUT

customer_id	ROUND(AVG(amount),2)
1	3.71
2	4.77
3	5.21
4	3.72
5	3.81

Your STDOUT

customer_id	ROUND(AVG(amount),2)
1	3.71
2	4.77
3	5.21
4	3.72
5	3.81

Instructor Solution

SELECT customer_id, ROUND(AVG(amount),2) FROM payment

GROUP BY customer_id LIMIT 5;

15. SELECT Statement - payment using `AVG()` , `GROUP BY`, `ROUND()` and Column Aliases

Retrieve the `customer_id`, and the average of the `amount` paid for a rental in the `payment` table per customer, rounded to two (2) decimal places. Limit your results to the first 5 rows. Use the following aliases in your query:

- customer_id --> "Id"
- rounded average --> "Average Spent"

SELECT customer_id AS "Id", ROUND(AVG(amount),2) AS "Average Spent" FROM payment GROUP BY customer_id LIMIT 5;

Expected STDOUT

Id	Average Spent
1	3.71
2	4.77
3	5.21
4	3.72
5	3.81

Your STDOUT

Id	Average Spent
1	3.71
2	4.77
3	5.21
4	3.72
5	3.81

Instructor Solution

SELECT customer_id AS "Id", ROUND(AVG(amount),2) AS "Average Spent"

FROM payment

GROUP BY customer_id LIMIT 5;

16. SELECT Statement -- `JOIN` two tables, payment and customer

Retrieve the `customer_id`, the customer's first and last names, and the average of the `amount` paid for a rental in the `payment` table per customer, rounded to two (2) decimal places. Limit your results to the first 5 rows.

NOTE: the column headers have to be **exact** for the answers to match. For example `round()` does not match `ROUND()`. If your answer looks correct, but does not match, please check the column headers.

Use `AVG()` , `GROUP BY`, `ROUND()`, and `INNER JOIN`

SELECT customer_id, first_name, last_name, ROUND(AVG(amount),2) FROM PAYMENT

JOIN customer USING (customer_id)

GROUP BY customer_id

LIMIT 5;

Expected STDOUT

customer_id	first_name	last_name	ROUND(AVG(amount),2)
1	MARY	SMITH	3.71
2	PATRICIA	JOHNSON	4.77
3	LINDA	WILLIAMS	5.21
4	BARBARA	JONES	3.72
5	ELIZABETH	BROWN	3.81

Your STDOUT

customer_id	first_name	last_name	ROUND(AVG(amount),2)
1	MARY	SMITH	3.71
2	PATRICIA	JOHNSON	4.77
3	LINDA	WILLIAMS	5.21
4	BARBARA	JONES	3.72
5	ELIZABETH	BROWN	3.81

Instructor Solution

SELECT customer_id, first_name, last_name, ROUND(AVG(amount),2)

FROM payment

INNER JOIN customer USING (customer_id)

GROUP BY customer_id LIMIT 5;

17. SELECT Statement -- `JOIN` two tables, payment and customer -- Use column aliases

Retrieve the `customer_id`, the customer's first and last names, and the average of the `amount` paid for a rental in the `payment` table per customer, rounded to two (2) decimal places. Limit your results to the first 5 rows. Use the following column aliases in your query:

- customer_id --> "Id"
- first_name --> "First Name"
- last_name --> "Last Name"
- rounded average --> "Average Spent"

Use `AVG()` , `GROUP BY`, `ROUND()`, `INNER JOIN` and **column aliases**

SELECT customer_id AS "Id", first_name AS "First Name", last_name AS "Last Name", ROUND(AVG(amount),2) AS "Average

Spent" FROM payment

JOIN customer USING (customer_id)

GROUP BY customer_id

LIMIT 5;

Expected STDOUT

Id	First Name	Last Name	Average Spent
1	MARY	SMITH	3.71
2	PATRICIA	JOHNSON	4.77
3	LINDA	WILLIAMS	5.21
4	BARBARA	JONES	3.72
5	ELIZABETH	BROWN	3.81

Your STDOUT

Id	First Name	Last Name	Average Spent
1	MARY	SMITH	3.71
2	PATRICIA	JOHNSON	4.77
3	LINDA	WILLIAMS	5.21
4	BARBARA	JONES	3.72
5	ELIZABETH	BROWN	3.81

Instructor Solution

SELECT customer_id AS "Id", first_name AS "First Name",

last_name AS "Last Name", ROUND(AVG(amount),2) AS "Average Spent"

FROM payment

INNER JOIN customer USING (customer_id)

GROUP BY customer_id LIMIT 5;

Concatenation Operator

When retrieving columns, it sometimes makes sense to use the concatenation operator `||` to allow one column to be printed with the information from multiple columns. Here is an example:

```
  SELECT customer_id AS "Id", first_name || " " || last_name AS "Customer Name" FROM
customer;
```

18. SELECT Statement -- `JOIN` two tables, payment and customer -- Use column aliases & `||`

Retrieve the `customer_id`, the customer's first and last names, and the average of the `amount` paid for a rental in the `payment` table per customer, rounded to two (2) decimal places. Limit your results to the first 5 rows. Use the following column aliases in your query:

- customer_id --> "Id"
- customer's full name --> "Customer Name"
- rounded average --> "Average Spent"

Use `AVG()` , `GROUP BY`, `ROUND()`, `INNER JOIN`, **column aliases**, and `||`

SELECT customer_id AS "Id", first_name || " " || last_name AS "Customer Name",
ROUND(AVG(amount),2) AS "Average Spent"

FROM payment

JOIN customer USING (customer_id)

GROUP BY customer_id

LIMIT 5;

Expected STDOUT

Id	Customer Name	Average Spent
1	MARY SMITH	3.71
2	PATRICIA JOHNSON	4.77
3	LINDA WILLIAMS	5.21
4	BARBARA JONES	3.72
5	ELIZABETH BROWN	3.81

Your STDOUT

Id	Customer Name	Average Spent
1	MARY SMITH	3.71
2	PATRICIA JOHNSON	4.77
3	LINDA WILLIAMS	5.21
4	BARBARA JONES	3.72
5	ELIZABETH BROWN	3.81

Instructor Solution

SELECT customer_id AS "Id", first_name || " " || last_name AS "Customer Name",

ROUND(AVG(amount),2) AS "Average Spent"

FROM payment

INNER JOIN customer USING (customer_id)

GROUP BY customer_id LIMIT 5;

These are the **Sakila** tables that might help you with the following two questions:

```
Table Name                      Column Names
-------------------------------------------------------------------------------
address                         address_id, address, address2,  district,
                                city_id (FK), postal_code, phone, location,
                                last_update
-------------------------------------------------------------------------------
city                            city_id, city,  country_id (FK), last_update
-------------------------------------------------------------------------------
country                         country_id, country, last_update
-------------------------------------------------------------------------------
customer                customer_id, store_id (FK), first_name, last_name,
                        email, address_id (FK), active, create_date, last_update
-------------------------------------------------------------------------------
rental                          rental_id, rental_date, inventory_id (FK),
                                customer_id (FK), return_date, staff_id (FK),
                                last_update
-------------------------------------------------------------------------------
staff                staff_id, first_name, last_name, address_id (FK), picture,
                     email, store_id (FK), active, username, password,
                     last_update
-------------------------------------------------------------------------------
store                           store_id, manager_staff_id (FK), address_id (FK),
                                      last_update
-------------------------------------------------------------------------------
```

19. Mastery #1: Complex SELECT Statement

Retrieve data from the **Sakila** database that solves the following request:

- List the `customer_id` & the count of films rented by the customer with a first_name of "AUSTIN".
- Create the following title for the count of films -- "Number of Rentals"
- Use "Id" as the header for the customer_id column

SELECT customer_id AS "Id", COUNT(rental_id) AS "Number of Rentals" FROM customer JOIN rental USING (customer_id) WHERE

first_name = "AUSTIN" GROUP BY customer_id;

Expected STDOUT

Id	Number of Rentals
599	19

Your STDOUT

Id	Number of Rentals
599	19

Instructor Solution

SELECT customer_id AS "Id", COUNT(rental_id) AS "Number of Rentals"

FROM rental

JOIN customer USING (customer_id)

WHERE first_name = "AUSTIN";

20. Mastery #2: Complex SELECT Statement

Retrieve data from the **Sakila** database that solves the following request:

- List all the staff members and their respective stores,
- For each `staff` member, print out the `staff_id`, `first_name`, and `last_name`
- For each `store`, print out the `store_id` as well as the location of the **store**, including the `address`, the `city` and the `country`
- Remember that each table has an `tableName_id` field, which can be used in the JOIN for each table.
- This question requires retrieving information from five (5) tables.
- Additionally, use the form `tableName.columnName` to retrieve the information from the correct location. For example, `city` could be `city.city` in the SELECT statement.

The `JOIN` statement is used to connect tables together.
Notice that both the staff and the store have an address, so use `JOIN address ON store.address_id = address.address_id` to get the correct address.

SELECT staff_id, first_name, last_name, store_id, address, city, country

FROM staff

JOIN store USING (store_id)

JOIN address ON store.address_id = address.address_id

JOIN city USING (city_id)

JOIN country USING (country_id)

GROUP BY staff_id;

Expected STDOUT

staff_id	first_name	last_name	store_id	address	city	country
1	Mike	Hillyer	2	28 MySQL Boulevard	Woodridge	Australia
2	Jon	Stephens	2	28 MySQL Boulevard	Woodridge	Australia

Your STDOUT

staff_id	first_name	last_name	store_id	address	city	country
1	Mike	Hillyer	2	28 MySQL Boulevard	Woodridge	Australia
2	Jon	Stephens	2	28 MySQL Boulevard	Woodridge	Australia

Instructor Solution

```
SELECT staff_id, first_name, last_name, store_id, address.address,

city.city, country.country

FROM staff

JOIN store USING (store_id)

JOIN address ON store.address_id = address.address_id

JOIN city USING (city_id)

JOIN country USING (country_id);
```