Promineo Tech Lesson - Week 8

promineotech.openclass.ai/resource/lesson-659c2dc3fe775297cad773cb

MySQL Part 3

This lesson contains Questions 21-30 (Part 3) of the OpenClass MySQL Lesson, and contains five (5) coding questions, each followed by a mastery question.

Vocabulary Reminder

- **Database**: A structured collection of data stored and organized for efficient retrieval and manipulation.
- **Schema**: A blueprint or structure that defines the logical organization and relationships of database objects, such as tables, views, and constraints.
- **DBMS** (Database Management System): Software that manages the storage, retrieval, and manipulation of data in a database.
- **RDBMS** (Relational Database Management System): An RDBMS organizes data based on the relational model, consisting of tables with rows and columns.
- **SQL** (Structured Query Language): is a language that is used by an RDBMS to interact with and manage relational data. SQL is a Standardized Language.
- **Query**: A request for data retrieval or manipulation from a database using a structured query language (SQL).
- **Primary Key**: A unique identifier for each record (row) in a table, used to ensure data integrity and facilitate record retrieval.
- **Foreign Key:** A field in one table that refers to the primary key of another table, establishing a relationship between the two tables.
- **Table**: A collection of related data organized in rows and columns in a relational database.
- **Entity**: A distinct object or concept in the real world that is represented in a database table.
- **Index**: A data structure that improves the speed of data retrieval operations by enabling efficient searching and sorting.
- **Attribute**: A characteristic or property of an entity that is stored as a column in a database table.
- **Transaction**: A logical unit of work that consists of one or more database operations, which must be performed atomically and consistently.
- **Normalization**: The process of organizing data in a database to eliminate redundancy and dependency issues.
- **ACID** (Atomicity, Consistency, Isolation, Durability): A set of properties that ensure reliability and consistency in database transactions.
- **Data Integrity**: The accuracy, consistency, and reliability of data stored in a database.
- **Query Optimization**: The process of selecting the most efficient execution plan for a database query to improve performance.
- **Data Warehousing**: The process of collecting, organizing, and storing large volumes of data from various sources for analysis and reporting.
- **Data Mining**: The process of discovering patterns, relationships, and insights from large datasets using statistical and machine learning techniques.
- **Backup and Recovery**: The process of creating backups of database data and implementing strategies to restore data in case of system failures or data loss.
- **CRUD** (Create, Read, Update, and Delete): The Operations that can be performed on a DBMS or RDBMS.
- **DML** (Data Manipulation Language): The language keywords that help manage and manipulate data in the database.
- ***Examples of DML**: `SELECT`, `INSERT`, `UPDATE`, and `DELETE`*
- **DDL** (Data Definition Language): The language keywords that help to define the structure or schema of the database
- ***Examples of DDL**: `CREATE`, `ALTER`, and `DROP`*

CRUD Operations in SQL

When we look at the **CRUD** operations on the data in the database, **CRUD** operations are all **DML** statements, as follows:

- **Create**: SQL `INSERT` statement
- **Read**: SQL `SELECT` statement
- **Update**: SQL `UPDATE` statement
- **Delete**: SQL `DELETE` statement

SQL Syntax

`INSERT`

```
INSERT [INTO] tbl_name
       [ (col_name [, col_name] ...)]
       {  VALUES (value_list) [, value_list) ] ... };
 value:
       { expr | DEFAULT}
 value_list:
       value [, value] ...
 assignment:
       col_name =
                   value
              | [row_alias.]col_name
              | [tbl_name.]col_name
              | [row_alias.]col_alias
 assignment_list:
       assignment [, assignment] ...
```

SELECT

```
SELECT select_expr [, select_expr] . . .
     [FROM table_references]
     [JOIN table_references { ON (col_name) | USING (col_name  = col_name) } ]
     [WHERE where_condition]
     [GROUP BY {col_name | expr | position}, ... ]
     [ORDER BY  {col_name | expr | position}
           [ASC | DESC], ... ]
     [LIMIT row_count];
```

UPDATE

```
UPDATE table_reference
 SET assignment_list
           [WHERE where_condition]
           [ORDER BY ...]
           [LIMIT row_count];
 value:
       { expr | DEFAULT}
 assignment:
       col_name = value
 assignment_list:
       assignment [, assignment] ...
```

DELETE

```
DELETE FROM tbl_name
      [WHERE where_condition]
      [ORDER BY ...]
      [LIMIT row_count];
```

References

Sakila Database

For all of our SQL lessons, we are going to use the **Sakila** Database. Each id column which is named *tableName_id* is a `PRIMARY KEY` in that table. Notice that some of those columns are used in subsequent tables as well, in those tables, that first *tableName_id* would be stored as a `FOREIGN KEY` in the subsequent table.

The ***Table*** and ***Column Names*** in this database are these:

```
Table Name                      Column Names
-------------------------------------------------------------------------------
actor                           actor_id, first_name, last_name last_update
-------------------------------------------------------------------------------
address                         address_id, address, address2,  district,
                                city_id (FK), postal_code, phone, location,
                                last_update
-------------------------------------------------------------------------------
category                        category_id, name, 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
-------------------------------------------------------------------------------
film                   film_id, title, description, release_year, language_id (FK),
                       original_language_id (FK), rental_duration, rental_rate,
                       length, replacement_cost, rating, special_features,
                       last_update
-------------------------------------------------------------------------------
film_actor                      actor_id (FK), film_id (FK), last_update
-------------------------------------------------------------------------------
film_category                   film_id (FK), category_id (FK), last_update
-------------------------------------------------------------------------------
film_text                       film_id (FK), title, description
-------------------------------------------------------------------------------
inventory                       inventory_id, film_id, store_id (FK), last_update
-------------------------------------------------------------------------------
language                        language_id, name, last_update
-------------------------------------------------------------------------------
payment                         payment_id, customer_id (FK), staff_id (FK),
                                rental_id (FK), amount, payment_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
-------------------------------------------------------------------------------
```

Each OpenClass question has a database attached to it. The only requirements necessary in the Solution Box are the SQL statements that accomplish what is being requested.

Reference: Sakila Database

21. `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"

Remember, when using an aggregate function, a `GROUP BY` clause may be important.

Sample Test Case #1

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

CREATE TABLE Syntax

When creating a database, this statement will allow a change to be made to the structure of the database or schema. This is a **DDL** statement.

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS] tbl_name
       (create_definition,...)
       [table_options]
       [partition_options]

create_definition: {
     col_name column_definition
     | {INDEX | KEY} [index_name] [index_type] (key_part,...)
         [index_option] ...
    | {FULLTEXT | SPATIAL} [INDEX | KEY] [index_name] (key_part,...)
         [index_option] ...
    | [CONSTRAINT [symbol]] PRIMARY KEY
         [index_type] (key_part,...)
         [index_option] ...
    | [CONSTRAINT [symbol]] UNIQUE [INDEX | KEY]
          [index_name] [index_type] (key_part,...)
          [index_option] ...
    | [CONSTRAINT [symbol]] FOREIGN KEY
          [index_name] (col_name,...)
          reference_definition
    | check_constraint_definition
    }


   column_definition: {
           data_type [NOT NULL | NULL] [DEFAULT {literal | (expr)} ]
               [AUTO_INCREMENT] [UNIQUE [KEY]] [[PRIMARY] KEY]
               [reference_definition]
           | data_type
               [VIRTUAL | STORED] [NOT NULL | NULL]
              [UNIQUE [KEY]] [[PRIMARY] KEY]
              [reference_definition]
    }
```

`CREATE TABLE employee` Example

```
CREATE TABLE employee (
           employee_id SMALLINT unsigned NOT NULL,
           first_name VARCHAR(45) NOT NULL,
           last_name VARCHAR(45) NOT NULL,
           address_id SMALLINT unsigned NOT NULL,
           email VARCHAR(50) DEFAULT NULL,
           store_id TINYINT  unsigned NOT NULL,
           employee_level  TEXT CHECK( employee_level IN ('FAIR','GOOD','GREAT', 'EXCELLENT') ) ,
           title VARCHAR(25) DEFAULT NULL,
           PRIMARY KEY (employee_id),
           FOREIGN KEY (address_id) REFERENCES address (address_id),
           FOREIGN KEY (store_id) REFERENCES store (store_id)

);
```

References:

- MySQL CREATE TABLE Documentation
- MySQL Data Types Documentation

22. `CREATE TABLE rewards`

Create a new table: rewards
Add the following fields:

- `rewards_id` SMALLINT
- `customer_id` SMALLINT
- `status` VARCHAR(20) DEFAULT 'MEMBER'
- `discount_percent` DOUBLE DEFAULT 0.0
- `year_joined` YEAR DEFAULT CURRENT_YEAR

All fields need to be `NOT NULL` or use the `DEFAULT` keyword
Add the following PRIMARY and FOREIGN KEYS

- `PRIMARY KEY (rewards_id)`
- `FOREIGN KEY (customer_id) REFERENCES customer(customer_id)`
- `FOREIGN KEY (status) REFERENCES reward_status (status)`

Test the creation by doing a retrieval of all data from this table -- which will show the column names.

`SELECT * FROM rewards;` will retrieve all of the data from this table (or in this case, show the column names).

Sample Test Case #1

Expected STDOUT

rewards_id	customer_id	status	discount_percent	year_joined

`status` Table

This table has been created in your database, and the table is called `status`. This is one way to store information in the database, and restrict values for a particular field.

There is a new table added into this database called `status`, which has the following columns and values:

```
Table Name            Column Names                  Allowed Values
-------------------------------------------------------------------------------------
status                  status           'MEMBER', 'SILVER', 'GOLD', 'PLATINUM'
-------------------------------------------------------------------------------------
```

*23. `INSERT INTO` **rewards** table*

Add the following 4 rows into the **rewards** table:

- (1, 1, 'PLATINUM', 0.20, '2000')
- (2, 2, 'GOLD', 0.15, '2010')
- (3, 3, 'SILVER', 0.10, '2015')
- (4, 4, 'MEMBER', 0.05, '2020')

Test those inserts by retrieving all information from the **rewards** table.

Sample Test Case #1

Expected STDOUT

rewards_id	customer_id	status	discount_percent	year_joined
1	1	PLATINUM	0.2	2000
2	2	GOLD	0.15	2010
3	3	SILVER	0.1	2015
4	4	MEMBER	0.05	2020

Sakila `customer` table with columns

```
Table Name                      Column Names
--------------------------------------------------------------------------------
customer            customer_id, store_id (FK), first_name, last_name,
                    email, address_id (FK), active, create_date, last_update
--------------------------------------------------------------------------------
rewards             rewards_id, customer_id, status, discount_percent, year_joined
--------------------------------------------------------------------------------
```

*24. `INSERT` a new customer in the **customer** table*

Insert customer "Mary Mallows" with the following data:

- `customer_id`: 1000
- `store_id`: 1
- `first_name`: "Mary"
- `last_name`: "Mallows"
- `email`: "mm@gmail.com"
- `address_id`: 1
- `active`: 1
- `create_date`: "2023-05-31 14:26:19"
- `last_update`: "2023-05-31 14:26:19"

Add the following row into the **rewards** table: (1000, 1000, 'PLATINUM', 0.20, '2023')

Test those inserts by retrieving this `customer_id` from **customer** and **rewards**

Use a `JOIN` in your `SELECT` statement.

SELECT * FROM customer

JOIN rewards USING (customer_id)

WHERE customer_id = 1000;

Sample Test Case #1

Expected STDOUT

customer_id	store_id	first_name	last_name	email	address_id	active	create_date	last_update	rewards_id	statu
1000	1	Mary	Mallows	mm@gmail.com	1	1	2023-05-31 14:26:19	2023-05-31 14:26:19	1000	PLAT

*25. `SELECT` customer "Mary Mallows" with **rewards** data*

Retrieve the first name & last name from the customer with the name "Mary Mallows" along with her **rewards** information.

`SELECT first_name, last_name, rewards.* FROM customer`
Then use a `JOIN` in your `SELECT` statement to get the rest of the information.

Sample Test Case #1

Expected STDOUT

first_name	last_name	rewards_id	customer_id	status	discount_percent	year_joined
Mary	Mallows	1000	1000	PLATINUM	0.2	2023

26. `UPDATE` customer "Mary Mallows"
"Mary Mallows" got married this week. You need to change the following:

- Her last name to "Smith".
- Her e-mail address to "ms@gmail.com".
- Don't forget to update the `last_update` column to "2023-06-19 23:00:00".

The `customer_id` of "Mary Mallows" is 1000.
SELECT the customer record showing the updated information.

Note: In real life, we would set `last_update` to `CURRENT_TIMESTAMP`, but for testing purposes, we will set it to the same value as the comparison!

Sample Test Case #1

Expected STDOUT

customer_id	store_id	first_name	last_name	email	address_id	active	create_date	last_update
1000	1	Mary	Smith	ms@gmail.com	1	1	2023-05-31 14:26:19	2023-06-19 23:00:00

27. `DELETE` customer "Mary Smith"
Do not forget *the WHERE clause in an SQL DELETE statement!*
SELECT the customer record from **customer** showing that it has been deleted.

Sample Test Case #1

Expected STDOUT

customer_id	store_id	first_name	last_name	email	address_id	active	create_date	last_update

`ORDER BY` Clause

When retrieving data from a relational database system, it is sometimes important to order that data in a particular way. The `ORDER BY` clause allows the data to be displayed in a particular order. The `ORDER BY` clause is an optional part in a `SELECT` statement, and is used to sort the result set in ascending (`ASC`) or descending (`DESC`) order by whatever `column_name`, expression or position is chosen. If omitted, the data will be displayed in the order that it is retrieved.

The `ORDER BY` keyword sorts the records in ascending order by default. To sort the records in descending order, use the `DESC` keyword. The `ASC` keyword is also used for clarity, even though records are sorted in ascending order by default.

`ORDER BY` Syntax

```
[ ORDER BY { column_name | expression | position}  [DESC | ASC] ... ]
```

28. `SELECT` from `film`

List the bottom 5 films with the lowest rental count, retrieving the title, and the rental count AS "rental_count". Order the results from lowest to highest rental count, and Limit the result to 5.

This query requires information from three (3) tables: `film`, `inventory` and `rental`.
Additionally, use the `GROUP BY title`, because of the aggregate function to achieve the COUNT, and an `ORDER BY` to correctly order your results.

Sample Test Case #1

Expected STDOUT

title	rental_count
HARDLY ROBBERS	4
MIXED DOORS	4
TRAIN BUNCH	4
BRAVEHEART HUMAN	5
BUNCH MINDS	5

`JOIN` Clause

A `JOIN` clause is part of the `SELECT` statement in SQL, and is used to combine rows from two (2) or more tables, based on a related column between the tables. The concept used here connects a `FOREIGN KEY` (FK) in one table to a `PRIMARY KEY` (PK) in another table.

When joining tables in a relational database management system, there are four distinct types of joins:

- `(INNER) JOIN`: An **inner join** returns records that have matching values in both tables.
- `RIGHT (OUTER) JOIN`: A **right (outer) join** returns **all records** from the **right** table, and the **matched records** from the **left** table.
- `LEFT (OUTER) JOIN`: A **left (outer) join** returns **all records** from the **left** table, and the **matched records** from the **right** table.
- `FULL (OUTER) JOIN`: A **full (outer) join** returns **all records** from **both** tables when there is a match in either the left or right table.

In most cases, a `SELECT` is looking for the rows that have matching values in both tables, so a `INNER JOIN` is used.

`USING` vs. `ON`

When joining tables, there are two types of syntax that can be used. If the **PK/FK** pair of columns have **exactly the same name in each table**, then the keyword `USING` can be used. If the **PK/FK** pair has a **different name in each table**, then the `ON` clause can be used. Imagine joining these two tables from the **Sakila** database.

```
Table Name                      Column Names
-------------------------------------------------------------------------------
film                  film_id, title, description, release_year, language_id (FK),
                      original_language_id (FK), rental_duration, rental_rate,
                      length, replacement_cost, rating, special_features,
                      last_update
-------------------------------------------------------------------------------
language              language_id (PK), name, last_update
-------------------------------------------------------------------------------
```

There are two Examples below, both joining the `film` table with the `language` table on the `language.language_id` **PK** column.

Example 1: Join `film` and `language` using `film.language_id` to `language.language_id`

Notice that in this example, the column names are exactly the same in each table, so the `USING` keyword can be used.

```
SELECT * from film
INNER JOIN language USING (language_id);
```

The `ON` keyword can also be used, and the following example will do the exact same thing as the above version:

```
SELECT * from film
INNER JOIN language ON language.language_id = film.language_id;
```

Example 2: Join `film` and `language` using `film.original_language_id` to `language.language_id`

Notice that in this example, the column names are **different** in each table, so the `ON` keyword **must** be used.

```
SELECT * from film
INNER JOIN language ON film.original_language_id = language.language_id;
```

This query can also be written using **table aliases**:

```
SELECT * from film f
INNER JOIN language l ON f.original_language_id = l.language_id;
```

29. `SELECT` from `film`

- List the top 5 film titles with the highest rental count
- Retrieve the rental count AS "rental_count"
- Remember that COUNT() is an aggregate.
- Order the results from highest to lowest rental count
- Limit the result to 5.

This query requires information from three (3) tables: `film`, `inventory` and `rental`.
Don't forget the `GROUP BY title`, because of the aggregate function to achieve the COUNT, and an `ORDER BY` to correctly order your results.

Sample Test Case #1

Expected STDOUT

title	rental_count
BUCKET BROTHERHOOD	34
ROCKETEER MOTHER	33
SCALAWAG DUCK	32
RIDGEMONT SUBMARINE	32
JUGGLER HARDLY	32

30. `SELECT` from `film`

- Given the top 5 films with the highest rental count, ordered from highest to lowest by "rental_count". *See question 29 solution.*
- Retrieve the amount of money that these 5 top films have made AS "rental_amount"

NOTE: The `JOIN` in question 29. will give you all of the information that you need.

This query requires information from three (3) tables: `film`, `inventory` and `rental`.
Use `SUM()` to calculate the amount of money made, which is found by multiplying `rental_duration` by `rental_rate`.

Sample Test Case #1

Expected STDOUT

title	rental_count	rental_amount
BUCKET BROTHERHOOD	34	1187.6199999999997
ROCKETEER MOTHER	33	98.00999999999998
SCALAWAG DUCK	32	958.0800000000008
RIDGEMONT SUBMARINE	32	95.03999999999998
JUGGLER HARDLY	32	126.71999999999991