Promineo Tech Lesson - Week 7

promineotech.openclass.ai/resource/lesson-659c2dc3fe775297cad773cd

MySQL Part I

This lesson contains Questions 1-10 (Part 1) of the OpenClass MySQL Lesson, and contains five (5) coding questions, each followed by a mastery question.

Vocabulary

- **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.

Relational Database Model

The **Relational Database Model** is a conceptual framework for organizing and managing data in a database. It was introduced by Edgar F. Codd in 1970 and has since become the most widely used data model in the field of Database Management Systems (DBMS).

The **Relational Database Model** organizes data into tables, which consist of **rows** (also known as **tuples**) and **columns**. Each **table** represents a specific **entity** or **relationship** in the real world. The **rows** represent *individual instances of the entity*, and the **columns** represent *attributes or properties of that entity*.

Key characteristics of the Relational Database Model include:

- **Tables**: The basic building blocks of the model. Each table has a unique name and a defined set of columns. Each column represents a specific attribute of the entity represented by the table.
- **Rows**: Also referred to as tuples or records, rows represent individual instances or occurrences of the entity. Each row contains values corresponding to the attributes defined by the table's columns.
- **Primary Key**: Each table in a relational database has a primary key, which uniquely identifies each row in the table. It ensures the uniqueness and integrity of data within the table.
- **Foreign Key**: A foreign key establishes relationships between tables. It is a column in one table that refers to the primary key of another table. This relationship enables data integrity and supports the concept of data normalization.
- **Relationships**: Relationships between tables represent associations or connections between entities. Common types of relationships include one-to-one, one-to-many, and many-to-many. Relationships help maintain data consistency and enable efficient querying across multiple tables.

- **Normalization**: The process of organizing and structuring data in a database to eliminate redundancy and dependency issues. Normalization minimizes data duplication, improves data integrity, and reduces data inconsistencies.
- **SQL** (Structured Query Language): The standardized language used to interact with relational databases. SQL provides a set of commands for querying, inserting, updating, and deleting data in the tables.

The **Relational Database Model** offers several advantages, including data integrity, flexibility, and scalability. It allows for efficient data retrieval, supports complex queries, and provides mechanisms for data consistency and security. The model's structure and adherence to normalization principles make it well-suited for a wide range of applications and industries.

Relational Database Management Systems (RDBMS) such as MySQL, Oracle, and PostgreSQL are widely used to implement and manage relational databases based on this model.

SQL Overview

SQL stands for **Structured Query Language** . SQL is a language which is used to communicate and manage databases and schema. SQL is the language that is used to communicate with a database server. There are many database servers in existence today, including but not limited to: **MySQL**, **Postgres**, **Oracle**, **Oracle Rdb**, **Sybase**, **MongoDB**, and more. There are three main subsystems within a DBMS, the **DDL** or Data Definition Language, the **DML** or Data Manipulation Language, and the Security Subsystem, which grants privileges, and prevents unauthorized access to data, it is important to make sure a user has privilege to be able to retrieve, modify or delete data.

When information is stored on disk, there may be many requests to read, store or modify data, and a database allows us to manage the concurrent access to that data. Additionally, it's important to have a secure location to store data, and that data needs to be accessible in a timely manner.

When looking at real word uses for databases, here are some examples. A database is essential to provide the backend to web sites, they provide storage for data that is used and analyzed in other applications, they allow report generation, and of course, databases are important for live queries, or requests for data (imagine your bank account balance).

RDBMS stands for **Relational Database Management System** (**RDBMS**), and is a term which describes a particular way that data is organized, stored and retrieved. **SQL** is specifically an **RDBMS** language, so here are some of the more interesting concepts with regard to relational databases.

- A **Database Server** can contain many databases
- **Databases** are collections of tables
- **Tables** are two-dimensional, with rows and columns
- **RDBMSs** using **SQL** allows mathematical and summary operations (aggregates, operators, etc.)
- **RDBMSs** using **SQL** are excellent at being able to manage data, and combine data from several tables.

SQL commands

Though SQL is case-insensitive, to be clear in this example, the SQL keywords will be in ALL CAPS.

```
CREATE DATABASE employees;
SHOW DATABASES;
--  database name: employees
USE employees;
SHOW TABLES IN employees;
-- table name is:  employees
SHOW COLUMNS in employees;
-- show columns and their data types
DESCRIBE employees;
```

SQL Variable Types

- **Numeric**: **INTEGER**, **SMALLINT**, **BIGINT**, **NUMERIC (w,d)** and **DECIMAL(w,d)** -- numbers with width **w** and **d** decimal places, **REAL**, **DOUBLE PRECISION**, **FLOAT(p)** where **p** represents the number of binary digits of precision in a floating point number.
- **Character**: **CHAR(L)** where **L** is the length of a fixed-length character, **VARCHAR(L)** that supports a max length of **L**
- **Binary**: **BIT(L)**, **BLOB(L)**
- **Temporal**: **DATE**, **TIME**, **TIMESTAMP**

SQL CREATE TABLE Example

```
CREATE TABLE employees (
        id INT,
        birth_date DATE,
        first_name VARCHAR(15),
        last_name VARCHAR(20),
        gender ENUM('M', 'F',  'GF', 'NB'),
        hire_date DATE
 );
```

SQL INSERT INTO Example

```
INSERT INTO employees (id, birth_date, first_name, last_name, gender, hire_date)
              VALUES (250001, "2003-12-29", "John", "Jones", "M", "2023-06-01");
```

SQL SELECT Example

```
SELECT * FROM  employees;
```

Reference: MySQL Documentation

SQL SELECT Syntax

SELECT

In the `SELECT` statement, many of these clauses are not required. Please refer to the documentation link below for more details.

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

Reference: MySQL SELECT Syntax

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 *previousTableName_id* would be 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 statement required in the Solution Box is the query requested.

Reference: Sakila Database

SQL Sakila Example

To retrieve information from a database using SQL, the **SELECT** statement is used.

This example is going to use Sakila and referencing the **city** table. The fields in the **city** table are: **city_id**, **city**, **country_id**, **last_update**

```
SELECT * from city WHERE city LIKE 'Ab%';
```

SQL WHERE Clause

The WHERE clause in a SELECT (or other) SQL statements allows us to restrict the information that is returned from a database. For example, say we want all of the cities that are stored in the database that start with the letters "Ab" (like above). The WHERE clause will compare the *city* field in the *city* table, to see if it starts with "Ab" using the LIKE operator and a wildcard operator '%'. There are a number of different options for the WHERE clause. Here are a few additional options for use in a WHERE clause..

- Logical Operators: `< > <= >= = <>`
- `BETWEEN`: inclusive range
- `IN`: testing group membership
- `NOT`: used for negation
- `LIKE`: allows comparison with wildcards `%`, `_`

- Logical `AND` and `OR`

SQL LIMIT Clause

To retrieve only a few records from a table in a database, use the LIMIT Clause. If the question asks for the first five records, then `LIMIT 5` would be added to the end of the query. The following example would retrieve the first 5 cities stored in the city table.

```
SELECT * FROM city LIMIT 5;
```

Imagine that you would only want to retrieve the city names (which are stored in the column city). The SELECT statement would look like this:

```
SELECT city FROM city LIMIT 5;
```

Aggregate Functions

SQL provides a number of functions that can be used within an SQL Query. Aggregate functions are allowed in two places in SQL. IN a SELECT clause or in the HAVING clause. Below we will ask for a total number of rows in a table. The appropriate function to use there will be `COUNT()`. Additional Aggregates include `MAX()`, `MIN()`, `AVG()`, and `SUM()`. Of note is the need to have a `GROUP BY` clause with the use of some of these functions.

Aliases

SQL aliases are used to give a table, or a column in a table, a temporary name. The goal of **SQL aliases** is to make a query more readable. In particular, **aliases** are often used to make column names more readable. An alias will only exist for the duration of that query, and is created by using the SQL `AS` keyword. This keyword is used for both **table aliases** and **column aliases**.

Use the SQL keyword `AS` to allow a different way to reference a table, or to display a different title for a column.

Compare the two `SELECT` statements below using **column aliases**, in the first query, the columns will be printed with the headers `first_name` and `last_name`. In the second query, they headers will be `First Name` and `Last Name`. **Column aliases** enhance readability of the results.

```
-- This one will display the column headers as first_name and last_name
SELECT first_name, last_name  FROM actor;
-- This one will display the column headers as First Name and Last Name
SELECT first_name AS "First Name", last_name AS "Last Name" FROM actor;
```

It is now time to use the SQL `SELECT` statement. This lesson is a little different than the previous lessons. Each question will ask you to use one SQL statement. There are 20 SQL Coding Questions, all using the Sakila database. Refer back to the information shared to assist you in this lesson.

IMPORTANT INFORMATION

In these OpenClass SQL assignments, the order of the columns and tables matters. The reason we mention this, is that OpenClass compares results exactly. Any of the following could cause your Coding Question to fail:

- Columns retrieved in the wrong order -- The data may be the same, but the result will not match.
- Joining tables in opposite order, depending on the retrieved columns -- The data may be the same, but the result will not match.
- Using `count(*)` instead of `COUNT(*)` -- The same as far as SQL knows, BUT the result will not match because the column name does not match.
- Using `ROUND(AVG(amount), 2)` instead of `ROUND(AVG(amount),2)` -- Notice the space (" ") between the comma and the 2.

The Convention Used in the following Coding Questions

- *All SQL keywords in the following examples are CAPITALIZED.*
- *All database names and columns are lowercase.*
- *No spaces will added unless requested in the question.*
- *The order of tables in a query, or with a JOIN will be the order of mention in the question.*
- *The order of columns retrieved will also be the order of mention in the question.*

1. SELECT Statement

Retrieve all information in the first 5 rows from the `film_text` table.

Sample Test Case #1

Expected STDOUT

film_id	title	description
1	ACADEMY DINOSAUR	A Epic Drama of a Feminist And a Mad Scientist who must Battle a Teacher in The Canadian Rockies

2	ACE GOLDFINGER	A Astounding Epistle of a Database Administrator And a Explorer who must Find a Car in Ancient China
3	ADAPTATION HOLES	A Astounding Reflection of a Lumberjack And a Car who must Sink a Lumberjack in A Baloon Factory
4	AFFAIR PREJUDICE	A Fanciful Documentary of a Frisbee And a Lumberjack who must Chase a Monkey in A Shark Tank
5	AFRICAN EGG	A Fast-Paced Documentary of a Pastry Chef And a Dentist who must Pursue a Forensic Psychologist in The Gulf of Mexico

2. SELECT Statement with a WHERE Clause

Retrieve all information in the rows from the `film_text` table where the `title` starts with `Z`.

Sample Test Case #1

Expected STDOUT

film_id	title	description
998	ZHIVAGO CORE	A Fateful Yarn of a Composer And a Man who must Face a Boy in The Canadian Rockies
999	ZOOLANDER FICTION	A Fateful Reflection of a Waitress And a Boat who must Discover a Sumo Wrestler in Ancient China
1000	ZORRO ARK	A Intrepid Panorama of a Mad Scientist And a Boy who must Redeem a Boy in A Monastery

3. SELECT Statement with an Aggregate Function

How many rows are in the actor table?

Use `COUNT(*)`

Sample Test Case #1

Expected STDOUT

COUNT(*)
200

4. SELECT Statement with an Aggregate Function and Column Alias

How many rows are in the actor table? This time use the **column alias** "Count of Actors"!

The keyword AS allows an aggregate function to be displayed as any text in quotes.

Sample Test Case #1

Expected STDOUT

Count of Actors
200

5. SELECT Statement

Retrieve all information in the rows from the actor table that have an actor_id less than 10.

Sample Test Case #1

Expected STDOUT

actor_id	first_name	last_name	last_update
1	PENELOPE	GUINESS	2006-02-15 09:34:33
2	NICK	WAHLBERG	2006-02-15 09:34:33
3	ED	CHASE	2006-02-15 09:34:33
4	JENNIFER	DAVIS	2006-02-15 09:34:33
5	JOHNNY	LOLLOBRIGIDA	2006-02-15 09:34:33
6	BETTE	NICHOLSON	2006-02-15 09:34:33
7	GRACE	MOSTEL	2006-02-15 09:34:33
8	MATTHEW	JOHANSSON	2006-02-15 09:34:33
9	JOE	SWANK	2006-02-15 09:34:33

6. SELECT Statement

Retrieve only the actor_id, first_name and last_name from the rows from the actor table that have an actor_id less than 10.

REMEMBER: Retrieve the columns in the order listed in the question

Sample Test Case #1

Expected STDOUT

actor_id	first_name	last_name
1	PENELOPE	GUINESS
2	NICK	WAHLBERG
3	ED	CHASE
4	JENNIFER	DAVIS
5	JOHNNY	LOLLOBRIGIDA
6	BETTE	NICHOLSON
7	GRACE	MOSTEL
8	MATTHEW	JOHANSSON
9	JOE	SWANK

7. SELECT Statement with Column Aliases

Retrieve only the actor_id, first_name and last_name from the rows from the actor table that have an actor_id less than 10. This time use the following **column aliases**.

- actor_id --> "Actor Id"
- first_name --> "First Name"
- last_name --> "Last Name"

REMEMBER: Retrieve the columns in the order listed in the question

Sample Test Case #1

Expected STDOUT

Actor Id	First Name	Last Name
1	PENELOPE	GUINESS
2	NICK	WAHLBERG
3	ED	CHASE
4	JENNIFER	DAVIS
5	JOHNNY	LOLLOBRIGIDA
6	BETTE	NICHOLSON
7	GRACE	MOSTEL

8	MATTHEW	JOHANSSON
9	JOE	SWANK

SQL JOIN Information

When using SQL, the way to connect information from two tables is to use the `JOIN` keyword. A `JOIN` in SQL is a relational concept. When information is stored in a RDBMS, the concept of `KEY` is often used to connect information in tables.

This example is going to use Sakila and referencing the following two tables: **city** and **country**

- The fields in the **city** table are: **city_id**, **city**, **country_id**, **last_update**
- The fields in the **country** table are: **country_id**, **country**, **last_update**

Notice: the `city` table does not store the corresponding `country` information within the `city` table. Instead, there is a `country` table which contains the name of the country, and a `KEY`. That `PRIMARY KEY` in the `country` table with the name `country_id` is then stored *in the `city` table as a `FOREIGN KEY` with the name `country_id`*.

To actually print out the `city` name with the corresponding `country` name, a `JOIN` is required within the `SELECT` statement to connect the two tables.

You previously saw this example as a brief introduction into the `JOIN` concept, but here it is again to demonstrate the concepts of `PRIMARY` and `FOREIGN KEY`, and how two tables are connected to `SELECT` the information requested within an RDBMS. In the following example, notice these details:

- **Table aliases**: `ci` and `co` -- these are used as a short cut way to tell SQL which table we are retrieving the information from.
- `INNER JOIN` clause with `USING` -- since the two columns are named the same, this clause works.
- `WHERE` clause that specifies only to retrieve the countries that start with 'Ab';

SQL JOIN Example

This example uses **table aliases**:

```
SELECT ci.city, co.country FROM city ci
INNER JOIN country co USING (country_id)
WHERE ci.city LIKE 'Ab%';
```

A similar way to do this query by using the actual table names as the **table alias**:

```
SELECT city.city, country.country FROM city
INNER JOIN country  USING (country_id)
WHERE city.city LIKE 'Ab%';
```

The result of both of these queries will be as follows:

```
city                    country
Abha                    Saudi Arabia
Abu Dhabi               United Arab Emirates
```

`USING` vs `ON` in a `JOIN` clause

The usage of these two keywords depends on the column names in the two tables being joined. If the column names are the same, then `USING` can be used as seen in the above examples. If the column names are different, then the `ON` keyword must be used. Here is an example of the `ON` keyword, which will retrieve the same results as the above queries.

```
SELECT city.city, country.country FROM city
INNER JOIN country  ON (city.country_id = country.country_id)
WHERE city.city LIKE 'Ab%';
```

IMPORTANT Reminder: In OpenClass SQL Coding Questions, the order does matter

- *Please `SELECT` any columns in the order that they are listed in the question.*
- *The **first** table mentioned in the question will be used in the `FROM` clause.*
- *`JOIN` all other tables in the order that they are mentioned in the question.*

8. SELECT Statement

Retrieve all of the information from the first 5 rows of the city table and the country table, using the `JOIN` keyword, and joining on the key column that is in common between the two tables.

`SELECT` any information `FROM city` and then `JOIN country`

Sample Test Case #1

Expected STDOUT

city_id	city	country_id	last_update	country	last_update

1	A Coruña (La Coruña)	87	2006-02-15 09:45:25	Spain	2006-02-15 09:44:00
2	Abha	82	2006-02-15 09:45:25	Saudi Arabia	2006-02-15 09:44:00
3	Abu Dhabi	101	2006-02-15 09:45:25	United Arab Emirates	2006-02-15 09:44:00
4	Acuña	60	2006-02-15 09:45:25	Mexico	2006-02-15 09:44:00
5	Adana	97	2006-02-15 09:45:25	Turkey	2006-02-15 09:44:00

9. SELECT Statement - city & country

Retrieve only the city name and the country name from the first 5 rows of the city table and the country table, using the `JOIN` keyword, and joining on the key column that is in common between the two tables.

Sample Test Case #1

Expected STDOUT

city	country
A Coruña (La Coruña)	Spain
Abha	Saudi Arabia
Abu Dhabi	United Arab Emirates
Acuña	Mexico
Adana	Turkey

10. SELECT Statement - city & country

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

Sample Test Case #1

Expected STDOUT

city	country

Abha	Saudi Arabia
Abu Dhabi	United Arab Emirates