Agenda

- What is CRUD?
- Sakila Database Walkthrough
- CRUD
    - Create
    - Read
        - Selecting Distinct Values
        - Select statement to print a constant value
        - Operations on Columns
        - Inserting Data from Another Table
        - WHERE Clause
        - AND, OR, NOT
        - IN Operator

> *Remaining topics will be covered in next lecture.*

What is CRUD

Today we are going to start the journey of learning MySQL queries by learning about CRUD Operations. Let's say there is a table in which we are storing information about students. What all can we do in that table or its entries?

Primarily, on any entity stored in a table, there are 4 operations possible:

1. Create (or inserting a new entry)
2. Read (fetching some entries)
3. Update (updating information about an entry already stored)
4. Delete (deleting an entry)

Today we are going to discuss about these operations in detail. Understand that read queries can get a lot more complex, involving aggregate functions, subqueries etc.

We will be starting with learning about Create, then go to Read, then Update and finally Delete. So let's get started. For this class as well as most of the classes ahead, we will be using Sakila database, which is an official sample database provided by MySQL.

Sakila Database Overview

Let us give you all a brief idea about what Sakila database represents so that it is easy to relate to the conversations that we shall have around this over the coming weeks. Sakila database represents a digital video rental store, assume an old movie rental store before Netflix etc. came. It's designed with functionality that would allow for all the operations of such a business, including transactions renting films, managing inventory, and storing customer and staff information. Example: it has tables regarding films, actors, customers, staff, stores, payments etc. You will get more familiar with this in the coming notes, don't worry!

> *Note: Please download these following in the same order as mentioned here:*

MYSQL Community Server Download Link: https://dev.mysql.com/downloads/mysql/ **MYSQL workbench Download Link:** https://dev.mysql.com/downloads/workbench/ **Sakila Download**

Link: https://dev.mysql.com/doc/index-other.html **How to add Sakila Database:** https://drive.google.com/file/d/1eiHtEwGr6r0qWlVpjzYefgP-rPG6DSbv/view?usp=sharing
Overall Doc containing steps for MYSQL setup: https://drive.google.com/file/d/1gJ2W4HFY6YxYMX1xtjOyKOefW93WoO-y/view

---

Create new entries using Insert

Now let's start with the first set of operation for the day: The Create Operation. As the name suggests, this operation is used to create new entries in a table. Let's say we want to add a new film to the database. How do we do that?

`INSERT` statement in MySQL is used to insert new entries in a table. Let's see how we can use it to insert a new film in the `film` table of Sakila database.

```
INSERT INTO film (title, description, release_year, language_id, rental_duration,
rental_rate, length, replacement_cost, rating, special_features)
VALUES ('The Dark Knight', 'Batman fights the Joker', 2008, 1, 3, 4.99, 152, 19.99,
'PG-13', 'Trailers'),
       ('The Dark Knight Rises', 'Batman fights Bane', 2012, 1, 3, 4.99, 165, 19.99,
'PG-13', 'Trailers'),
       ('The Dark Knight Returns', 'Batman fights Superman', 2016, 1, 3, 4.99, 152,
19.99, 'PG-13', 'Trailers');
```

> *Note: MySQL queries are not case sensitive.*

Let's dive through the syntax of the query. First we have the `INSERT INTO` clause, which is used to specify the table in which we want to insert the new entry. Then we have the column names in the brackets, which are the columns in which we want to insert the values. Then we have the `VALUES` clause, which is used to specify the values that we want to insert in the columns. The values are specified in the same order as the columns are specified in the `INSERT INTO` clause. So the first value in the `VALUES` clause will be inserted in the first column specified in the `INSERT INTO` clause, and so on.

Create - About column names in INSERT query

A few things to note here:

The column names are optional. If you don't specify the column names, then the values will be inserted in the columns in the order in which they were defined at the time of creating the table. Example: in the above query, if we don't specify the column names, then the values will be inserted in the order `film_id`, `title`, `description`, `release_year`, `language_id`, `original_language_id`, `rental_duration`, `rental_rate`, `length`, `replacement_cost`, `rating`, `special_features`, `last_update`. So the value `The Dark Knight` will be inserted in the `film_id` column, `Batman fights the Joker` will be inserted in the `title` column and so on.

- This is not a good practice, as it makes the query prone to errors. So always specify the column names.
- This makes writing queries tedious, as while writing query you have to keep a track of what column was where. And even a small miss can lead to a big error.
- If you don't specify column names, then you have to specify values for all the columns, including `film_id`, `original_language_id` and `last_update`, which we

may want to keep `NULL`.

Anyways, an example of a query without column names is as follows:

```
INSERT INTO film
VALUES (default, 'The Dark Knight', 'Batman fights the Joker', 2008, 1, NULL, 3, 4.99,
152, 19.99, 'PG-13', 'Trailers', default);
```

NULL is used to specify that the value of that column should be `NULL`, and `default` is used to specify that the value of that column should be the default value specified for that column. Example: `film_id` is an auto-increment column, so we don't need to specify its value. So we can specify `default` for that column, which will insert the next auto-increment value in that column.

So that's pretty much all that's there about Create operations. There is 1 more thing about insert, which is how to insert data from one table to another, but we will talk about that after talking about read.

Before we start with read operations, let us have 2 small Quiz questions for you.

Quiz 1

What is the correct syntax to insert a new record into a MySQL table?

Choices

- ☐ `INSERT INTO table_name VALUES (value1, value2, value3,...);`
- ☐ `INSERT INTO table_name (value1, value2, value3,...);`
- ☐ `INSERT VALUES (value1, value2, value3,...) INTO table_name;`
- ☐ `INSERT (value1, value2, value3,...) INTO table_name;`

Quiz 2

How do you insert a new record into a specific column (e.g., 'column1') in a table (e.g., 'table1')?

Choices

- ☐ `INSERT INTO table1 column1 VALUES (value1);`
- ☐ `INSERT INTO table1 (column1) VALUES (value1);`
- ☐ `INSERT VALUES (value1) INTO table1 (column1);`
- ☐ `INSERT (column1) VALUES (value1) INTO table1;`

Read

Now let's get to the most interesting, and also maybe most important part of today's session: Read operation. `SELECT` statement is used to read data from a table. `Select` command is similar to print statements in other languages. Let's see how we can use it to read data via different queries on the `film` table of Sakila database(Do writing this query once by yourself). A basic select query is as follows:

```
SELECT * FROM film;
```

However using above query isn't considered a very good idea. `Select *` have it's own downsides such as `Unnecessary I/O`, `Increased Network Traffic`, `Dependency on Order of Columns on ResultSet`, `More Application Memory`.

More on why using `Selec *` isn't good: https://dzone.com/articles/why-you-should-not-use-select-in-sql-query-1

Now try following query and guess the output before trying on workbench:

```
SELECT 'Hello world!';
```

Here, we are selecting all the columns from the `film` table. The `*` is used to select all the columns. This query will give you the value of each column in each row of the film table. If we want to select only specific columns, then we can specify the column names instead of `*`. Example:

```
SELECT title, description, release_year FROM film;
```

Here we are selecting only the `title`, `description` and `release_year` columns from the `film` table. Note that the column names are separated by commas. Also, the column names are case-insensitive, so `title` and `TITLE` are the same. For example, following query would have also given the same result:

```
SELECT TITLE, DESCRIPTION, RELEASE_YEAR FROM film;
```

Furthermore, if we want to have `title` column as 'film_name' and `film_id` as 'id' then we can use `as` keyword. This keyword is used to rename a column or table with an alias. It is temporary and only lasts until the duration of that particular query. For example:

```
SELECT title as film_name, film_id as id
FROM film;
```

Selecting Distinct Values

Now, let's learn some nuances around the `SELECT` statement.

Let's say we want to select all the distinct values of the `rating` column from the `film` table. How do we do that? We can use the `DISTINCT` keyword to select distinct values. Example:

```
SELECT DISTINCT rating FROM film;
```

This query will give you all the distinct values of the `rating` column from the `film` table. Note that the `DISTINCT` keyword, as all other keywords in MySQL, is case-insensitive, so `DISTINCT` and `distinct` are the same.

We can also use the `DISTINCT` keyword with multiple columns. Example:

```
SELECT DISTINCT rating, release_year FROM film;
```

This query will give you all the distinct set of values of the `rating` and `release_year` columns from the `film` table. Try writing this query once by yourself.

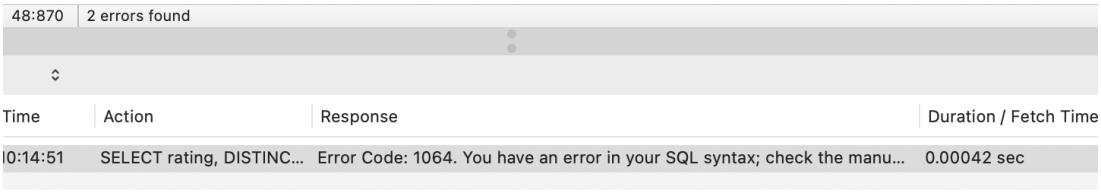
> *Note: DISTINCT keyword must be before all the column names, as it will find the unique values for the collection of column values. For example, if there are 2 column names, then it will find* ***distinct pairs*** *among the corresponding values from both the columns.*

Example:

```
 -- The Distinct keyword must only be used before the first column in the query
SELECT rating, DISTINCT release_year FROM film;
```

- The following picture shows the error which occurs when we don't use `DISTINCT` as first keyword after `Select`.
- This above query wants to print all ratings but distinct release_years which doesn't make sense since there can be mismatch in number of ratings and number of distinct years which will eventually cause an error.
- Here is an article from Scaler topics to read more: https://www.scaler.com/topics/distinct-in-sql/

```
SELECT rating, DISTINCT release_year FROM film;
```



Pseudo Code:

Let's talk about how this works. A lot of SQL queries can be easily understood by relating them to basic for loops, etc. Throughout this module, we will try to demonstrate the understanding of complex queries by providing corresponding pseudo code, as I attempt to do the same in a programming language. As all of you have already solved many DSA problems, this shall be much more easy and fun for you to learn.

So, let's try to understand the above query with a pseudo code. The pseudo code for the above query would be as follows:

```
answer = []

for each row in film:
    answer.append(row)
```

```
filtered_answer = []

for each row in answer:
    filtered_answer.append(row['rating'], row['release_year'])

unique_answer = set(filtered_answer)

return unique_answer
```

So, what you see is that DISTINCT keyword on multiple column gives you for all of the rows in the table, the distinct value of pair of these columns.

Select statement to print a constant value

In one of the above queries we have already seen that we can print constant values as well using `Select` command. Now let's see it's further uses.

Let's say we want to print a constant value in the output. Eg: The first program that almost every programmer writes: "Hello World". How do we do that? We can use the `SELECT` statement to print a constant value. Example:

```
SELECT 'Hello World';
```

That's it. No from, nothing. Just the value. You can also combine it with other columns. Example:

```
SELECT title, 'Hello World' FROM film;
```

Operations on Columns

Let's say we want to select the `title` and `length` columns from the `film` table. If you see, the value of length is currently in minutes, but we want to select the length in hours instead of minutes. How do we do that? We can use the `SELECT` statement to perform operations on columns. Example:

```
SELECT title, length/60 FROM film;
```

Later in the course we will learn about Built-In functions in SQL as well. You can use those functions as well to perform operations on columns. Example:

```
SELECT title, ROUND(length/60) FROM film;
```

ROUND function is used to round off a number to the nearest integer. So the above query will give you the title of the film, and the length of the film in hours, rounded off to the nearest integer.

Inserting Data from Another Table

By the way, SELECT can also be used to insert data in a table. Let's say we want to insert all the films from the `film` table into the `film_copy` table. We can combine the `SELECT` and `INSERT INTO` statements to do that. Example:

```
INSERT INTO film_copy (title, description, release_year, language_id, rental_duration,
rental_rate, length, replacement_cost, rating, special_features)
SELECT title, description, release_year, language_id, rental_duration, rental_rate,
length, replacement_cost, rating, special_features
FROM film;
```

Here we are using the `SELECT` statement to select all the columns from the `film` table, and then using the `INSERT INTO` statement to insert the selected data into the `film_copy` table. Note that the column names in the `INSERT INTO` clause and the `SELECT` clause are the same, and the values are inserted in the same order as the columns are specified in the `INSERT INTO` clause. So, the first value in the `SELECT` clause will be inserted in the first column specified in the `INSERT INTO` clause, and so on.

Okay, let us verify how well you have learnt till now with a few quiz questions.

Quiz 3

What does the DISTINCT keyword do in a SELECT statement?

Choices

- [ ] It counts the number of unique records in a column.
- [ ] It finds the sum of all records in a column.
- [ ] It eliminates duplicate records in the output.
- [ ] It sorts the records in ascending order.

Quiz 4

If you want to retrieve all records from a 'customers' table, which statement would you use?

Choices

- [ ] SELECT * FROM customers;
- [ ] SELECT ALL FROM customers;
- [ ] RETRIEVE * FROM customers;
- [ ] GET * FROM customers;

Quiz 5

What is the result of the following SQL query: `SELECT DISTINCT column1 FROM table1;` ?

Choices

- [ ] It displays all values of column1, including duplicates.
- [ ] It displays unique non-null values of column1.
- [ ] It counts the total number of unique values in column1.
- [ ] It sorts all values in column1.

WHERE clause

Till now, we have been doing basic read operations. SELECT query with only FROM clause is rarely sufficient. Rarely do we want to return all rows. Often we need to have some kind of filtering logic etc. for the rows that should be returned. Let's learn how to do that.

Let's use Sakila database to understand this. Say we want to select all the films from the `film` table which have a rating of `PG-13`. How do we do that? We can use the `WHERE` clause to filter rows based on a condition. Example:

```
SELECT * FROM film WHERE rating = 'PG-13';
```

Here we are using the `WHERE` clause to filter rows based on the condition that the value of the `rating` column should be `PG-13`. Note that the `WHERE` clause is always used after the `FROM` clause. In terms of pseudocode, you can think of where clause to work as follows:

```
answer = []

for each row in film:
    if row.matches(conditions in where clause) # new line from above
        answer.append(row)

filtered_answer = []

for each row in answer:
    filtered_answer.append(row['rating'], row['release_year'])

unique_answer = set(filtered_answer) # assuming we also had DISTINCT

return unique_answer
```

If you see, where clause can be considered analgous to `if` in a programming language. With if also, there are many other operators that are used, right? Can you name which operators do we often use in programming languages with `if`?

AND, OR, NOT

We use things like `and`, `or`, `!` in programming languages to combine multiple conditions. Similarly, we can use `AND`, `OR`, `NOT` operators in SQL as well. Example: We want to get all the films from the `film` table which have a rating of `PG-13` and a release year of `2006`. We can use the `AND` operator to combine multiple conditions.

```
SELECT * FROM film WHERE rating = 'PG-13' AND release_year = 2006;
```

Similarly, we can use the `OR` operator to combine multiple conditions. Example: We want to get all the films from the `film` table which have a rating of `PG-13` or a release year of `2006`. We can use the `OR` operator to combine multiple conditions.

```
SELECT * FROM film WHERE rating = 'PG-13' OR release_year = 2006;
```

Similarly, we can use the `NOT` operator to negate a condition. Example: We want to get all the films from the `film` table which do not have a rating of `PG-13`. We can use the `NOT` operator to negate the condition.

```
SELECT * FROM film WHERE NOT rating = 'PG-13';
```

An advice on using these operators: If you are using multiple operators, it is always a good idea to use parentheses to make your query more readable. Else, it can be difficult to understand the order in which the operators will be evaluated. Example:

```
SELECT * FROM film WHERE rating = 'PG-13' OR release_year = 2006 AND rental_rate = 0.99;
```

Here, it is not clear whether the `AND` operator will be evaluated first or the `OR` operator. To make it clear, we can use parentheses. Example:

```
SELECT * FROM film WHERE rating = 'PG-13' OR (release_year = 2006 AND rental_rate = 0.99);
```

Till now, we have used only `=` for doing comparisons. Like traditional programming languages, MySQL also supports other comparison operators like `>`, `<`, `>=`, `<=`, `!=` etc. Just one special case, `!=` can also be written as `<>` in MySQL. Example:

```
SELECT * FROM film WHERE rating <> 'PG-13';
```

IN Operator

With comparison operators, we can only compare a column with a single value. What if we want to compare a column with multiple values? For example, we want to get all the films from the `film` table which have a rating of `PG-13` or `R`. One way to do that can be to combine multiple consitions using `OR`. A better way will be to use the `IN` operator to compare a column with multiple values. Example:

```
SELECT * FROM film WHERE rating IN ('PG-13', 'R');
```

Okay, now let's say we want to get those films that have ratings anything other than the above 2. Any guesses how we may do that?

Correct! We had earlier discussed about `NOT`. You can also use `NOT` before `IN` to negate the condition. Example:

```
SELECT * FROM film WHERE rating NOT IN ('PG-13', 'R');
```

Think of IN to be like any other operator, additionally, it allows comparison with multiple values.

ORDER BY Clause

Now let's discuss another important clause. ORDER BY clause allows to return values in a sorted order. Example:

```
SELECT * FROM film ORDER BY title;
```

The above query will return all the rows from the `film` table in ascending order of the `title` column. If you want to return the rows in descending order, you can use the `DESC` keyword. Example:

```
SELECT * FROM film ORDER BY title DESC;
```

You can also sort by multiple columns. Example:

```
SELECT * FROM film ORDER BY title, release_year;
```

The above query will return all the rows from the `film` table in ascending order of the `title` column and then in ascending order of the `release_year` column. Consider the second column as tie breaker. If 2 rows have same value of title, release year will be used to break tie between them. Example:

```
SELECT * FROM film ORDER BY title DESC, release_year DESC;
```

Above query will return all the rows from the `film` table in descending order of the `title` column and if tie on `title`, in descending order of the `release_year` column.

By the way, you can ORDER BY on a column which is not present in the SELECT clause. Example:

```
SELECT title FROM film ORDER BY release_year;
```

Let's also build the analogy of this with a pseudocode.

```
answer = []

for each row in film:
    if row.matches(conditions in where clause) # new line from above
        answer.append(row)

answer.sort(column_names in order by clause)

filtered_answer = []

for each row in answer:
    filtered_answer.append(row['rating'], row['release_year'])

return filtered_answer
```

If you see, the `ORDER BY` clause is applied after the `WHERE` clause. So, first the rows are filtered based on the `WHERE` clause and then they are sorted based on the `ORDER BY` clause. And only after that are the columns that have to be printed taken out. And that's why you can sort based on columns not even in the `SELECT` clause.

> *We will discuss about order by once again in CRUD 2 notes.*

Solution to Quizzes:

> *-- Quiz1: Option A (INSERT INTO table_name VALUES (value1, value2, value3,…);)*
> *Quiz2: Option B (INSERT INTO table1 (column1) VALUES (value1);) Quiz3: Option C*

> *(It eliminates duplicate records in the output.) Quiz4: Option A (SELECT * FROM customers;) Quiz5: Option B (It displays unique non-null values of column1.) --*