college CD

Database concepts (CBD)

Class 02

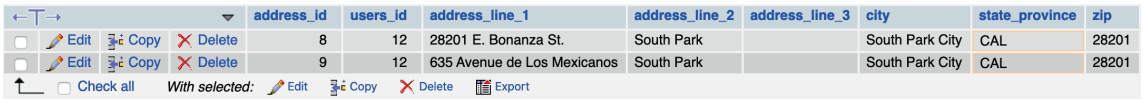
MySQL (SUITE)

Refining queries (suite)

On class 1, we have seen how to refine a query using the ***WHERE*** command to filter the results. In the example below, we are searching the addresses table for entries in the state of California (CAL).

```
SELECT * FROM `addresses` WHERE `state_province` = "CAL"
```

Note:
The value after the equal sign must use double quotes if the value is a string and none if the value is a number.



	address_id	users_id	address_line_1	address_line_2	address_line_3	city	state_province	zip
Edit Copy Delete	8	12	28201 E. Bonanza St.	South Park		South Park City	CAL	28201
Edit Copy Delete	9	12	635 Avenue de Los Mexicanos	South Park		South Park City	CAL	28201

Check all With selected: Edit Copy Delete Export

Refining queries using multiple rules

To refine the result of a query even more, to narrow it down further more, it is possible to apply multiple filters, to add new rules using the commands ***AND*** and **OR**.

```
SELECT * FROM `addresses` WHERE `state_province` = "CAL" AND `address_id` = 8
```

```
SELECT * FROM `addresses` WHERE `state_province` = "CAL"
OR `state_province` = "UK"
```

Refining queries using comparisons

It is possible to refine the results of a SQL search using comparisons. For instance, in order to search the products table, we may want to retrieve the entries which names have to do or sound like light using the command LIKE and the wildcard ("%").

```
SELECT * FROM `products` WHERE `product_name` LIKE "%light%"
```

Note:
Using the wildcard before and/or after will include all words containing "light" (depending where the wildcard is used) such as "lightsaber" that would have been ignored otherwise.

Refining queries using exclusions

It is possible to refine the results of SQL queries by excluding certain entries. In the example below, we are searching the *addresses* table to show all entries except those containing *South Park* in the *adress_line_2* field.

```
SELECT * FROM `addresses` WHERE NOT `address_line_2` = "South Park"
```

Working with numbers

SQL makes it possible to perform various calculations using different functions.

COUNT()

COUNT() function makes it possible to count the number of entries based on a table's given field. In the example below, we are asking to count how many values the field *username* contains in the *users* table.

```
SELECT COUNT(`username`) FROM `users`
```

Of course, the count function may be used with any other rules used to narrow down the results. In the example below, we count the number of email addresses containing the word "southpark" within the *user_email* field.

```
SELECT COUNT(`username`) FROM `users` WHERE `user_email`
LIKE "%southpark%"
```

AVG()

AVG() function calculates the average. In the example below, we calculate the average price of the products from the *product_price* field of the *products* table.

```
SELECT AVG(`product_price`) FROM `products`
```

SUM()

SUM() function makes an addition. In the example above, we count the total number of items contained in the *product_stock* field from the *products* table.

```
SELECT SUM(`product_stock`) FROM `products`
```

Modifying a database in MySQL

Adding to a database

So far, we have been learning how to select certain element from databases tables. Let's now see how to add element to a MySQL database's tables.

Inserting a new row (entry) in a table

In order to insert a new row in a table, you first need to get familiar with the table so you know how many entries and how many fields there are., if the primary key (ID) is set to *AUTO_INCREMENT*, etc.

Syntax

INSERT INTO `table's name` (`table's field 1`,`table's field 2`) VALUES (`value 1`,`value 2`)

The command *INSERT* will be used. To specify where the insertion will occur, the command *INTO* followed by *table* will be added, followed by a parenthesis containing the names of the fields. Finally, the VALUES command will be used to specify the values associated to the fields between parenthesis.

INSERT INTO `users` (`username`,`password`,`user_email`,`first_name`,`last_name`)
VALUES ("NewBrian", "12345", "nb@fakemail.com", "Brian", "New")

Note :
Since the ID field (primary key) was set to AUTO_INCREMENT, it will be generated automatically, so we've got here only 5 fields instead of 6.

Updating an entry

In order to make modification to an entry, it is needed to to use the UPDATE command to select the proper table, the SET command to indicate which field needs to be updated and, finally, supply the new information.

UPDATE `users` **SET** `username` **=** "quentin",`user_email` **=** "qw@mail.com" **WHERE** `username` **=** "quentin92"

In the above example, we update the *users* table by modifying the *username* field's value for *quentin* and we change the email for the entry where the existing username is *quentin92*.

Deleting an entry

In order to delete an entry, the command DELETE will be used. But, this command comes with great risks. A simple mistake may damage or erase your database. It is a good idea to make a backup of the database before doing such a thing.

In the example below, we are deleting from the table *users* the entry which contains in the field *first_name* the value *Sibu*.

```
DELETE FROM `users` WHERE `first_name` = "Sibu"
```

IMPORTANT

To avoid dramatic mistakes, it's always a better idea to use a primary key's number instead of any other value as a rule used with WHERE.

Example :

```
DELETE FROM `users` WHERE `id` = 3
```