

Introductory MySQL Commands

Principles of Databases (CS 365)

UTF-8 Character Set Conflicts

UTF-8 Character Set Conflicts

- Use UTF-8 character sets whenever possible

MySQL Configuration File

MySQL Configuration File

- On macOS, add **my.cnf** to the **/etc** folder.

MySQL Configuration File

- On macOS, add **my.cnf** to the **/etc** folder.
- In Windows **my.cnf** may be called **my.ini** and could be in one of many places. Read the official documentation from **dev.mysql.com** at <https://dev.mysql.com/doc/refman/8.0/en/option-files.html>

Logging in

The following command says, “Log in to MySQL as user (**-u**) root and tell the CLI to request my password (**-p**).

```
mysql -u root -p
```

Logging in

You can also close the space between **-u** and root, as follows:

```
mysql -uroot -p
```


Logging in

You can also append the password to the **-p** option. (No space character.)
For example, if my password were **password**, I could log in as follows:

```
mysql -u root -ppassword
```

or

```
mysql -uroot -ppassword
```

Logging in

Appending the password to the **-p** option is insecure, as the password would sit as a plain text entry in your CLI's history file.

In **bash**, for example, you'd find the password in **.bash_history**. You could clear it (and the rest of your history) with the **-c** flag to the **history** command:

```
history -c
```

Logging in

The more secure option is to have MySQL request your password via your CLI.

```
mysql -u root -p
```

Exiting MySQL

Similar to exiting your CLI, exiting MySQL is simply...

EXIT

Warnings

If an error is generated, you can see the latest warning with

SHOW WARNINGS;

Checking the Status of the Database

You can view some important information, such as current user and database, IP address, and character set configurations, using the **STATUS** command:

STATUS

Creating a Database

Let's create a database called **users** with a default and collation character set of UTF-8.

```
CREATE DATABASE `users` DEFAULT CHARACTER SET utf8mb4 COLLATE utf8mb4_bin;
```

Note: This doesn't place focus on the new database; it simply creates it.

Creating a Database | Placing Focus

To work with a database, you need to focus on it by using the **USE** command. Let's focus on the **users** database:

```
USE users
```

If you now run **STATUS**, you'll see, **Current database: users** below **Connection id**.

Add a User to the Database with a Password

Let's create a user called **the-user** whose password is **the-password**.

```
CREATE USER 'the-user'@'localhost' IDENTIFIED BY 'the-Passw0rd!';
```

Provide a User Access to the Database

Let's now grant **the-user** *all* privileges to *all* the tables under the **users** database

```
GRANT ALL PRIVILEGES ON users.* to 'the-user'@'localhost';
```

Logging into the Database with the New User

Exit the database (**exit**), then log back in as the new user:

```
mysql -u the-user -p
```

Show Databases

You can see the databases to which you have access with the **SHOW** command:

```
SHOW DATABASES;
```

Create a Table

```
CREATE TABLE students (  
    first_name VARCHAR(20) NOT NULL,  
    last_name VARCHAR(20) NOT NULL  
);
```

Note: Both are set to **NOT NULL**, meaning that an entry into the **students** table can only happen when both values are present. What happens when you try to defeat the **NOT NULL** rule?

Show Tables

Show the tables in the current database:

```
SHOW TABLES;
```

Flush the Contents of a Table

To empty the contents of a table is to flush them. Flushing means that MySQL will drop the tables, then recreate them without any entries.

```
TRUNCATE TABLE students;
```

Drop/Delete a Table

Let's delete the **students** table.

```
DROP TABLE students;
```

Note: This isn't the same as **TRUNCATE**, which flushes the tuples in the table, but doesn't delete the table.

Insert a Single Record in a Table (CREATE)

```
INSERT INTO students  
  (first_name, last_name)  
VALUES  
  ("Fred", 'Flinstone');
```

Note: I can wrap values in inch marks (") or foot marks ('), as long as they're balanced.

Insert Multiple Records into a Table (CREATE)

```
INSERT INTO students  
  (first_name, last_name)  
VALUES  
  ('Edward', 'Bobward'),  
  ('Ed', 'Bob'),  
  ('Frank', 'Einstein'),  
  ('Johnny', 'Rotten');
```

Read All Records from a Table (READ)

```
SELECT * FROM students;
```

Read All Records from a Table with a Matching Clause (READ)

Let's get all students whose first name is Frank.

```
SELECT * FROM students WHERE first_name = "Frank";
```

Pattern Matching

Let's get all students whose first name starts with "ed".

```
SELECT * FROM students WHERE first_name LIKE "Ed%";
```

or for a more case-insensitive search:

```
SELECT * FROM students WHERE UPPER(first_name) LIKE UPPER("ed%");
```

Pattern Matching

```
SELECT * FROM students WHERE last_name LIKE "%Bob";
```

Or, for a more case-insensitive search:

```
SELECT * FROM students WHERE UPPER(last_name) LIKE UPPER("%bob");
```

Pattern Matching

Get all track names that start with “Do” followed by 9 characters, listed alphabetically:

```
SELECT *  
FROM track  
WHERE track_name LIKE 'Do_____'  
ORDER BY track_name;
```

The underscore () character is a placeholder for any character.

Read All Records from a Table's Column (READ)

Let's get all **first_names** from the **students** table.

```
SELECT first_name FROM students;
```


Read All Records from a Table's Column (READ)

Or `last_names`.

```
SELECT last_name FROM students;
```

Read All Records from a Table in Reverse Order (READ)

```
SELECT last_name, first_name FROM students;
```

Describe the Fields/Columns in a Table

There are at least 3 different ways to describe the structure of a table.

SHOW COLUMNS FROM students;

DESC students;

DESCRIBE students;

Update (UPDATE)

Let's change Frank's first name to Albert:

```
UPDATE students SET first_name="Albert" WHERE first_name="Frank";
```

Remove (DELETE)

Let's remove Johnny, who's no longer a student:

```
DELETE FROM students WHERE first_name="Johnny";
```

Remove a Database and Its Users

There are multiple ways to delete a database. The most common and modern way is...

```
DROP DATABASE IF EXISTS users;
```

or

```
DROP DATABASE users;
```

Remove a Database and Its Users

SCHEMA is synonymous with **DATABASE**. Thus, you could also say...

```
DROP SCHEMA IF EXISTS users;
```

or

```
DROP SCHEMA users;
```

Remove a Database and Its Users

We'll now need to remove the user — whose username is **the-user** — from MySQL.

```
DROP USER IF EXISTS 'the-user'@'localhost';
```


Stand up a Database in Two Commands

Log in to MySQL...

```
mysql -u root -p
```

...then load **setup.sql**:

```
source setup.sql
```

Stand up a Database in Two Commands

You can also stand up the database in one command:

```
mysql -u root -p < setup.sql
```

The DELETE Statement

Remove all rows from a table.

```
DELETE FROM track;
```

The DELETE Statement

Let's delete Every Country's Sun

```
DELETE FROM album WHERE album_name = "Every Country's Sun";
```

or

```
DELETE FROM album  
WHERE (artist_id = 5 AND album_id = 2);
```

The latter makes use of the keys that we used to design the database. As such, it is more secure.

The DELETE Statement

Let's delete *all* albums with an **album_id** of **1**.

```
DELETE FROM ALBUM WHERE album_id = 1;
```

The DELETE Statement

Let's delete an artist, their album(s), and those albums' tracks. First, let's choose a band, Melvins.

```
SELECT artist_id, artist_name, album_name  
FROM artist INNER JOIN album  
USING (artist_id)  
WHERE artist_name = "Melvins";
```

The DELETE Statement

Now we can delete everything related to The Melvins.

```
DELETE FROM artist, album, track USING artist, album, track
WHERE artist.artist_id = 4 AND
artist.artist_id = album.artist_id AND
artist.artist_id = track.artist_id AND
album.album_id = track.album_id;
```

Compare with...

```
DELETE FROM artist, album, track USING artist, album, track
WHERE artist.artist_id = 4 AND
artist.artist_id = album.artist_id AND
artist.artist_id = track.artist_id;
```

The DELETE Statement

And, we can now verify:

```
SELECT track_name  
FROM track  
WHERE artist_id = 4;
```


The SELECT Statement

Get all song titles, listed alphabetically:

```
SELECT * FROM track  
ORDER BY track_name;
```

The result is an alphabetical — or lexicographical — list of tuples in ascending order (**ASC**), which is the default. Thus, the previous command is equivalent to the following:

```
SELECT * FROM track  
ORDER BY track_name ASC;
```

The SELECT Statement

You may also list the result in descending (**DESC**) order:

```
SELECT * FROM track  
ORDER BY track_name DESC;
```

The SELECT Statement

Get and rename both attributes from the music schema's **artist** table:

```
SELECT artist_id AS ID, artist_name AS Artist  
FROM artist;
```

AS is optional in the renaming. You could also do:

```
SELECT artist_id ID, artist_name Artist  
FROM artist;
```

The SELECT Statement

If using multiple words in a string, you'll need to wrap the content in foot marks (' ').

```
SELECT artist_id AS 'Unique ID', artist_name AS Artist  
FROM artist;
```

You may also use backticks (` `):

```
SELECT artist_id AS `Unique ID`, artist_name AS Artist  
FROM artist;
```

The JOIN Statement

Get all artists and their albums. Note the different ways to do this.

```
SELECT artist_name AS Artist, album_name AS Album  
FROM artist  
JOIN album  
WHERE (artist.artist_id = album.artist_id);
```

```
SELECT artist_name AS Artist, album_name AS Album  
FROM artist  
INNER JOIN album  
USING (artist_id);
```

The JOIN Statement

```
SELECT artist_name AS Artist, album_name AS Album  
FROM artist  
JOIN album  
USING (artist_id);
```

```
SELECT artist_name AS Artist, album_name AS Album  
FROM artist  
INNER JOIN album ON  
(artist.artist_id = album.artist_id);
```

The JOIN Statement

```
SELECT artist_name AS Artist, album_name AS Album  
FROM artist, album  
WHERE (artist.artist_id = album.artist_id);
```

Working with AES_ENCRYPT

The default **block_encryption_mode** is **aes-128-ecb**, which does not require an initialization vector. We want the extra security attached to an initialization vector, so we need to set **block_encryption_mode** to **aes-256-cbc**.

First, let's check the current value of **block_encryption_mode**.

```
SHOW VARIABLES WHERE variable_name = "block_encryption_mode";
```

or

```
SELECT @@global.block_encryption_mode;
```


Working with AES_ENCRYPT

Let's now set it:

```
SET block_encryption_mode = 'aes-256-cbc';
```

Working with AES_ENCRYPT

According to the **AES_ENCRYPT** documentation, we'll need a key to unlock the encryption. We're advised to do this with a user-defined variable, as follows:

```
SET @init_vector = RANDOM_BYTES(16);
```

Working with AES_ENCRYPT

Let's test it:

```
SELECT @init_vector;
```

Working with AES_ENCRYPT

Now log in to MySQL and run **setup.sql** from the enclosed **aes-encrypt-example** folder:

```
source setup.sql;
```

Working with AES_ENCRYPT

View all the entries:

```
SELECT * FROM user;
```

Working with AES_DECRYPT

And, finally, run the following to view the unciphered passwords:

```
SELECT CAST(AES_DECRYPT(password, @key_str, @init_vector) AS CHAR) AS 'Plain Text Password' FROM user;
```