sql-for-beginners

August 21, 2025

1 SQL for Beginners

By Justin Barry https://watsontechworld.com

Welcome to SQL for Beginners. SQL is a wonderful language in my opinion. It is one of the biggest languages in the world, and is a language designed specifically for tabular databases; that is, databases that use tables, like spreadsheets do.

2 Note: this book is a work in progress. It might not be finished until the end of 2025 or early 2026. But enjoy!

[1]:
```
# Note I might use this book as the basis for a course
```

[ ]:
```

```

Note about different database systems: Different database systems such as MySQL / MariaDB, PostgreSQL, SQLite have some subtle differences in how they are implemented. I will try to show you SQL code that will work in all of the systems. This book will use MySQL / MariaDB, but with a little editing, you should be able to use it in PostgreSQL, SQLite, or almost any other database system that uses SQL

[ ]:
```

```

3 Why this book?

This book is meant to be an introduction to SQL with MySQL and to hopefully help motivate you to try SQL and learn by doing.

Honestly, this is a book for me as much as it is a book for you. In part, I'm writing an SQL book I would have liked to read when I was first learning SQL.

[ ]:
```

```

4 Why even learn SQL? Why not just use Microsoft Excel, Google Sheets, LibreOffice Calc, or some other spreadsheet software?

This is a good question. In a nutshell, here are some top benefits of using SQL instead of a spreadsheet: * SQL scales far better for large datasets. For example, you can go from dozens of rows to millions of rows without needing to change much * SQL offers great optimizations * SQL can be used with other programming languages such as Python, JavaScript, and almost any other modern programming language * SQL allows you to build modern apps and websites * Learning SQL will allow you to learn backend web development and full stack web development

[ ]:

5 What are some top softwares that use SQL?

- WordPress (which uses MySQL by default)

[ ]:

6 Database definition?????

6.1 Get definition

This book is not meant to be a complicated computer science book on the theory of databases. In a nutshell, a database is just a collection of data. You could even consider a library with many books as a database. A database is a collection of information. But specifically with a computer, ?????????

[ ]:

[ ]:

7 SQL vs Spreadsheets. Which to use?

There is no cut and dried answer. But generally, spreadsheets are good if you have a few hundred to a few thousand rows, based on my own experience. But once you get past 100,000 rows, SQL generally becomes a far better use case

8 Spreadsheet benefits

- probably easier to use for very easy tracking, such as tracking exercise or spending for a small number of rows
- Spreadsheet softwares may have a lower learning curve

9 Data is data. How you should view or use it depends on the situation. You can sometimes interchangably go between SQL databases and spreadsheets

And actually, sometimes you'll use both. I often will import data from a CSV file and then insert that data into a database. Or I do a search for something in a database, and then output that to a CSV file.

[ ]:

10 SQL benefits

- Far more powerful than spreadsheet softwares
- Scales far better
- Can be more powerful
- Can be far faster than a spreadsheet
- Allows you to interact with a database interactively

[ ]:

11 Setting Up MySQL / MariaDB

In Ubuntu, you can install mysql or mariadb along with the necessary server with the following: (just first decide if you will install mysql or mariadb. If you can't decide, you can just go with mysql-server

`sudo apt install mysql-server` `sudo apt install mariadb-server`

[ ]:

12 How to Create a MySQL Database

Taken from my article at https://linuxwebdevelopment.com/how-to-create-a-new-database-in-mysql/

Choose the name you want for the databse, for example you might want the name `exampledotcomdatabase`

```
CREATE DATABASE exampledotcomdatabase CHARACTER SET utf8;
```

[ ]:

13 Importing a MySQL Database

Taken from my article at: https://linuxwebdevelopment.com/import-mysql-database/

Make sure you first have a username (for example `sampleuser`) and you have already defined a database, for example `sampledatabase`

```
mysql -u sampleuser -p sampledatabase < databasebackup.sql
```

[ ]:

14 How To Create A New Username In MySQL

Taken from my article at: https://linuxwebdevelopment.com/how-to-create-a-new-username-in-mysql/

First you need to have a database already set up for which to give the username permissions to. For example, you might have a database named `exampledotcomdatabase`

Then you will need to choose a username, for example `exampleusername` and you will need to give it a password, which is in the part below of IDENTIFIED BY 'abcdefg12345'. So you could have a password of `abcdefg12345` (probably not a good password)

```
CREATE USER 'exampleusername'@'localhost' IDENTIFIED BY 'abcdefg12345';
GRANT ALL PRIVILEGES ON exampledotcomdatabase . * TO 'exampleusername'@'localhost';
FLUSH PRIVILEGES;
exit;
```

[ ]:

15 How to Delete / Drop a MySQL Database (Use with Caution)

Deleting a database is very simple in MySQL. You just do: `DROP DATABASE database_name;`

or `DROP DATABASE IF EXISTS database_name;`

[2]:
```
# https://dev.mysql.com/doc/refman/8.4/en/drop-database.html
```

For example, to drop the `exampledotcomdatabase` database, you could do:

```
DROP DATABASE IF EXISTS exampledotcomdatabase;
```

[ ]:

16 The SELECT Keyword

17 Selecting or retrieving data from an SQL database with SELECT

[ ]:
```
Every table by default will be empty. You can still select from an empty␣
↪database, you will just get null.

For example, if I make an empty table
```

```
CREATE TABLE example_table (
  column_1 TEXT,
  column_2 TEXT,
  column_3 INT,
  column_4 VARCHAR(2),
);

```

Selecting or retrieving data is done with the `SELECT` keyword. Select is how we retrieve rows from a database.

The general format is

`SELECT column_names FROM table_name;`

Watch my video on SQL SELECT below SQL SELECT Statement Tutorial https://youtu.be/iN8JMOtTG5E

Example:

`SELECT * from salary;`

might give:

```
+--------+---------------+--------+------+-----------+-------------+--------------------------
| name   | yearly_salary | gender | age  | age_group | temp_column | career
+--------+---------------+--------+------+-----------+-------------+--------------------------
| Brad   |        150000 | m      |   35 | 30-39     | NULL        | doctor
| Sana   |       1450000 | f      |   28 | 20-29     | NULL        | idol / singer
| Luke   |         80000 | m      |   39 | 30-39     | NULL        | data analyst
| Tony   |         45000 | m      |   42 | 40-49     | NULL        | teacher
| Mike   |         44630 | m      |   25 | 20-29     | NULL        | account clerk
| Leia   |        112000 | f      |   45 | 40-49     | NULL        | chemical engineer
| Nick   |        250000 | m      |   57 | 50-59     | NULL        | CEO
| Cheryl |         68000 | f      |   23 | 20-29     | NULL        | software engineering inte
| Wayne  |         75000 | m      |   75 | 70-79     | NULL        | author
| Julie  |         80000 | f      |   40 | 40-49     | NULL        | author
| John   |         30000 | m      |   37 | 30-39     | NULL        | pizza delivery driver
| Jean   |         30000 | f      |   27 | 20-29     | NULL        | waitress
| Sonja  |        150000 | f      |   45 | 40-49     | NULL        | professor
| Sarah  |         40000 | f      |   38 | 30-39     | NULL        | Uber driver
| Mary   |        300000 | f      |   33 | 30-39     | NULL        | lawyer
| Olivia |         80000 | f      |   65 | 60-69     | NULL        | day trader
| Pilar  |         55000 | m      |   47 | 40-49     | NULL        | entrepreneur / self-emplo
+--------+---------------+--------+------+-----------+-------------+--------------------------
17 rows in set (0.01 sec)
```

Another example:

`SELECT  name, yearly_salary, gender, age FROM salary;`

might give:

```
+--------+---------------+--------+------+
| name   | yearly_salary | gender | age  |
+--------+---------------+--------+------+
| Brad   |        150000 | m      |   35 |
| Sana   |       1450000 | f      |   28 |
| Luke   |         80000 | m      |   39 |
| Tony   |         45000 | m      |   42 |
| Mike   |         44630 | m      |   25 |
| Leia   |        112000 | f      |   45 |
| Nick   |        250000 | m      |   57 |
| Cheryl |         68000 | f      |   23 |
| Wayne  |         75000 | m      |   75 |
| Julie  |         80000 | f      |   40 |
| John   |         30000 | m      |   37 |
| Jean   |         30000 | f      |   27 |
| Sonja  |        150000 | f      |   45 |
| Sarah  |         40000 | f      |   38 |
| Mary   |        300000 | f      |   33 |
| Olivia |         80000 | f      |   65 |
| Pilar  |         55000 | m      |   47 |
+--------+---------------+--------+------+
17 rows in set (0.00 sec)
```

[ ]:

17.1 Example SQL SELECT queries

– 1. Show all data from salary table `select * from salary;`

– 2. select name, gender, career, age, yearly_salary from salary table and sort by age `SELECT name, gender, career, age, yearly_salary FROM salary ORDER BY age;`

– 3. Do similarly, but instead sort by yearly salary ascending `SELECT name, gender, career, age, yearly_salary FROM salary ORDER BY yearly_salary;`

– 4. Now do similarly, but instead sort by yearly salary descending `SELECT name, gender, career, age, yearly_salary FROM salary ORDER BY yearly_salary DESC;`

– 5. Now do similarly, but instead sort by gender ascending and then yearly_salary descending `SELECT name, gender, career, age, yearly_salary FROM salary ORDER BY gender, yearly_salary DESC;`

[ ]:

– 1. Show all data from salary table `select * from salary;`

Might give you output like the following:

+——+——————————+————+——+——+————+ | name | career | yearly_salary | gender | age | age_group | +——+——————————+————+——+——+————+—

—+——+———+ | Brad | doctor | 150000 | m | 35 | 30-39 | | Sana | idol / singer | 1450000 | f | 28 | 20-29 | | Luke | data analyst | 80000 | m | 39 | 30-39 | | Tony | teacher | 45000 | m | 42 | 40-49 | | Mike | account clerk | 44630 | m | 25 | 20-29 | | Leia | chemical engineer | 112000 | f | 45 | 40-49 | | Nick | CEO | 250000 | m | 57 | 50-59 | | Cheryl | software engineering intern | 68000 | f | 23 | 20-29 | | Wayne | author | 75000 | m | 75 | 70-79 | | Julie | author | 80000 | f | 40 | 40-49 | | John | pizza delivery driver | 30000 | m | 37 | 30-39 | | Jean | waitress | 30000 | f | 27 | 20-29 | | Sonja | professor | 150000 | f | 45 | 40-49 | | Sarah | Uber driver | 40000 | f | 38 | 30-39 | | Mary | lawyer | 300000 | f | 33 | 30-39 | | Olivia | day trader | 80000 | f | 65 | 60-69 | | Pilar | entrepreneur / self-employed | 55000 | m | 47 | 40-49 | +——+——————————+————+——+——+———+ 17 rows in set (0.01 sec)

– 2. select name, gender, career, age, yearly_salary from salary table and sort by age `SELECT name, gender, career, age, yearly_salary FROM salary ORDER BY age;`

Might give you output like the following:

```
+--------+--------+------------------------------+------+---------------+
| name   | gender | career                       | age  | yearly_salary |
+--------+--------+------------------------------+------+---------------+
| Cheryl | f      | software engineering intern  |   23 |         68000 |
| Mike   | m      | account clerk                |   25 |         44630 |
| Jean   | f      | waitress                     |   27 |         30000 |
| Sana   | f      | idol / singer                |   28 |       1450000 |
| Mary   | f      | lawyer                       |   33 |        300000 |
| Brad   | m      | doctor                       |   35 |        150000 |
| John   | m      | pizza delivery driver        |   37 |         30000 |
| Sarah  | f      | Uber driver                  |   38 |         40000 |
| Luke   | m      | data analyst                 |   39 |         80000 |
| Julie  | f      | author                       |   40 |         80000 |
| Tony   | m      | teacher                      |   42 |         45000 |
| Leia   | f      | chemical engineer            |   45 |        112000 |
| Sonja  | f      | professor                    |   45 |        150000 |
| Pilar  | m      | entrepreneur / self-employed |   47 |         55000 |
| Nick   | m      | CEO                          |   57 |        250000 |
| Olivia | f      | day trader                   |   65 |         80000 |
| Wayne  | m      | author                       |   75 |         75000 |
+--------+--------+------------------------------+------+---------------+
17 rows in set (0.00 sec)
```

– 3. Do similarly, but instead sort by yearly salary ascending `SELECT name, gender, career, age, yearly_salary FROM salary ORDER BY yearly_salary;` Might give you output like the following:

```
+--------+--------+------------------------------+------+---------------+
| name   | gender | career                       | age  | yearly_salary |
+--------+--------+------------------------------+------+---------------+
| Jean   | f      | waitress                     |   27 |         30000 |
| John   | m      | pizza delivery driver        |   37 |         30000 |
| Sarah  | f      | Uber driver                  |   38 |         40000 |
| Mike   | m      | account clerk                |   25 |         44630 |
```

```
| Tony   | m      | teacher                       |   42 |         45000 |
| Pilar  | m      | entrepreneur / self-employed |   47 |         55000 |
| Cheryl | f      | software engineering intern  |   23 |         68000 |
| Wayne  | m      | author                        |   75 |         75000 |
| Julie  | f      | author                        |   40 |         80000 |
| Luke   | m      | data analyst                  |   39 |         80000 |
| Olivia | f      | day trader                    |   65 |         80000 |
| Leia   | f      | chemical engineer             |   45 |        112000 |
| Brad   | m      | doctor                        |   35 |        150000 |
| Sonja  | f      | professor                     |   45 |        150000 |
| Nick   | m      | CEO                           |   57 |        250000 |
| Mary   | f      | lawyer                        |   33 |        300000 |
| Sana   | f      | idol / singer                 |   28 |       1450000 |
+--------+--------+-------------------------------+------+---------------+
17 rows in set (0.00 sec)
```

– 4. Now do similarly, but instead sort by yearly salary descending `SELECT name, gender, career, age, yearly_salary FROM salary ORDER BY yearly_salary DESC;` Might give you output like the following:

```
+--------+--------+-------------------------------+------+---------------+
| name   | gender | career                        | age  | yearly_salary |
+--------+--------+-------------------------------+------+---------------+
| Sana   | f      | idol / singer                 |   28 |       1450000 |
| Mary   | f      | lawyer                        |   33 |        300000 |
| Nick   | m      | CEO                           |   57 |        250000 |
| Sonja  | f      | professor                     |   45 |        150000 |
| Brad   | m      | doctor                        |   35 |        150000 |
| Leia   | f      | chemical engineer             |   45 |        112000 |
| Luke   | m      | data analyst                  |   39 |         80000 |
| Olivia | f      | day trader                    |   65 |         80000 |
| Julie  | f      | author                        |   40 |         80000 |
| Wayne  | m      | author                        |   75 |         75000 |
| Cheryl | f      | software engineering intern  |   23 |         68000 |
| Pilar  | m      | entrepreneur / self-employed |   47 |         55000 |
| Tony   | m      | teacher                       |   42 |         45000 |
| Mike   | m      | account clerk                 |   25 |         44630 |
| Sarah  | f      | Uber driver                   |   38 |         40000 |
| John   | m      | pizza delivery driver         |   37 |         30000 |
| Jean   | f      | waitress                      |   27 |         30000 |
+--------+--------+-------------------------------+------+---------------+
17 rows in set (0.00 sec)
```

– 5. Now do similarly, but instead sort by gender ascending and then yearly_salary descending `SELECT name, gender, career, age, yearly_salary FROM salary ORDER BY gender, yearly_salary DESC;` Might give you output like the following:

```
+--------+--------+-------------------------------+------+---------------+
| name   | gender | career                        | age  | yearly_salary |
+--------+--------+-------------------------------+------+---------------+
```

```
| Sana   | f      | idol / singer                |   28 |       1450000 |
| Mary   | f      | lawyer                       |   33 |        300000 |
| Sonja  | f      | professor                    |   45 |        150000 |
| Leia   | f      | chemical engineer            |   45 |        112000 |
| Olivia | f      | day trader                   |   65 |         80000 |
| Julie  | f      | author                       |   40 |         80000 |
| Cheryl | f      | software engineering intern  |   23 |         68000 |
| Sarah  | f      | Uber driver                  |   38 |         40000 |
| Jean   | f      | waitress                     |   27 |         30000 |
| Nick   | m      | CEO                          |   57 |        250000 |
| Brad   | m      | doctor                       |   35 |        150000 |
| Luke   | m      | data analyst                 |   39 |         80000 |
| Wayne  | m      | author                       |   75 |         75000 |
| Pilar  | m      | entrepreneur / self-employed |   47 |         55000 |
| Tony   | m      | teacher                      |   42 |         45000 |
| Mike   | m      | account clerk                |   25 |         44630 |
| John   | m      | pizza delivery driver        |   37 |         30000 |
+--------+--------+------------------------------+------+---------------+
17 rows in set (0.00 sec)
```

[ ]:

18 YouTube videos I've made on SQL

[2]:
```
# note to self. Use redirects instead of direct YouTube links for long term␣
↪stability of the links.
```

SQL SELECT Statement Tutorial https://youtu.be/iN8JMOtTG5E

SQL GROUP BY: Visual Guide & Examples https://youtu.be/Fud1Rfsl9dE

SQL ORDER BY: Tutorial for Beginners https://youtu.be/VKRnf60WZuo

SQL LIMIT: Quick Tutorial https://youtu.be/64YKfHVbBiE

[ ]:

19 INSERTING data / adding new data

[ ]:

[ ]:

[ ]:

[ ]:

20 DELETING data

[ ]:

21 UPDATING rows

The general format is

```
    UPDATE table_name
SET column_1 = 'some_value"
WHERE some_condition
```

[ ]:

[ ]:

[ ]:

22 JOINs between 2 or more tables

[ ]:

[ ]:

23 Creating a table with `CREATE TABLE`

```
Example:
CREATE TABLE salary (
  name TEXT NOT NULL,
  career TEXT,
  yearly_salary INT,
  gender VARCHAR(2),
  age INT,
  age_group TEXT
);
```

[ ]:

24 INSERT INTO

`INSERT INTO` is how we can populate a table with new data.

Every table starts out as empty. Below is an example of how we can add 17 rows of data to the `salary` table. First let's delete the table (if it exists), re-create it, and then add the rows

Delete the table if it exists `DROP TABLE IF EXISTS salary;`

Then re-create the table salary with the following definitions:

```
CREATE TABLE salary (
  name TEXT NOT NULL,
  career TEXT,
  yearly_salary INT,
  gender VARCHAR(2),
  age INT,
  age_group TEXT
);
```

[ ]:

[ ]:

```
-- insert some values

INSERT INTO salary VALUES ('Brad', 'doctor', 150000, 'm', 35, '30-39');
INSERT INTO salary VALUES ('Sana', 'idol / singer', 1450000, 'f', 28, '20-29');
INSERT INTO salary VALUES ('Luke', 'data analyst', 80000, 'm', 39, '30-39');
INSERT INTO salary VALUES ('Tony', 'teacher', 45000, 'm', 42, '40-49');
INSERT INTO salary VALUES ('Mike', 'account clerk', 44630, 'm', 25, '20-29');
INSERT INTO salary VALUES ('Leia', 'chemical engineer', 112000, 'f', 45, '40-49');
INSERT INTO salary VALUES ('Nick', 'CEO', 250000, 'm', 57, '50-59');
INSERT INTO salary VALUES ('Cheryl', 'software engineering intern', 68000, 'f', 23, '20-29');
INSERT INTO salary VALUES ('Wayne', 'author', 75000, 'm', 75, '70-79');
INSERT INTO salary VALUES ('Julie', 'author', 80000, 'f', 40, '40-49');
INSERT INTO salary VALUES ('John', 'pizza delivery driver', 30000, 'm', 37, '30-39');
INSERT INTO salary VALUES ('Jean', 'waitress', 30000, 'f', 27, '20-29');
INSERT INTO salary VALUES ('Sonja', 'professor', 150000, 'f', 45, '40-49');
INSERT INTO salary VALUES ('Sarah', 'Uber driver', 40000, 'f', 38, '30-39');
INSERT INTO salary VALUES ('Mary', 'lawyer', 300000, 'f', 33, '30-39');
INSERT INTO salary VALUES ('Olivia', 'day trader', 80000, 'f', 65, '60-69');
INSERT INTO salary VALUES ('Pilar', 'entrepreneur / self-employed', 55000, 'm', 47, '40-49');
```

[ ]: The above command should have

25 DROP TABLE for deleting a table (use with caution)

DROP TABLE table_name;

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

Select is how we retrieve rows from a database.

The general format is

```
SELECT column_names FROM table_name;
```

Example:

```
SELECT * from salary;
```

might give:

```
+--------+---------------+--------+------+-----------+-------------+--------------------------
| name   | yearly_salary | gender | age  | age_group | temp_column | career
+--------+---------------+--------+------+-----------+-------------+--------------------------
| Brad   |        150000 | m      |   35 | 30-39     | NULL        | doctor
| Sana   |       1450000 | f      |   28 | 20-29     | NULL        | idol / singer
| Luke   |         80000 | m      |   39 | 30-39     | NULL        | data analyst
| Tony   |         45000 | m      |   42 | 40-49     | NULL        | teacher
| Mike   |         44630 | m      |   25 | 20-29     | NULL        | account clerk
| Leia   |        112000 | f      |   45 | 40-49     | NULL        | chemical engineer
| Nick   |        250000 | m      |   57 | 50-59     | NULL        | CEO
| Cheryl |         68000 | f      |   23 | 20-29     | NULL        | software engineering inte
| Wayne  |         75000 | m      |   75 | 70-79     | NULL        | author
| Julie  |         80000 | f      |   40 | 40-49     | NULL        | author
| John   |         30000 | m      |   37 | 30-39     | NULL        | pizza delivery driver
| Jean   |         30000 | f      |   27 | 20-29     | NULL        | waitress
| Sonja  |        150000 | f      |   45 | 40-49     | NULL        | professor
| Sarah  |         40000 | f      |   38 | 30-39     | NULL        | Uber driver
| Mary   |        300000 | f      |   33 | 30-39     | NULL        | lawyer
| Olivia |         80000 | f      |   65 | 60-69     | NULL        | day trader
| Pilar  |         55000 | m      |   47 | 40-49     | NULL        | entrepreneur / self-emplo
+--------+---------------+--------+------+-----------+-------------+--------------------------
17 rows in set (0.01 sec)
```

Another example:

```
SELECT  name, yearly_salary, gender, age FROM salary;
```

might give:

```
+--------+---------------+--------+------+
| name   | yearly_salary | gender | age  |
+--------+---------------+--------+------+
| Brad   |        150000 | m      |   35 |
| Sana   |       1450000 | f      |   28 |
| Luke   |         80000 | m      |   39 |
| Tony   |         45000 | m      |   42 |
| Mike   |         44630 | m      |   25 |
| Leia   |        112000 | f      |   45 |
```

```
| Nick   |        250000 | m      |   57 |
| Cheryl |         68000 | f      |   23 |
| Wayne  |         75000 | m      |   75 |
| Julie  |         80000 | f      |   40 |
| John   |         30000 | m      |   37 |
| Jean   |         30000 | f      |   27 |
| Sonja  |        150000 | f      |   45 |
| Sarah  |         40000 | f      |   38 |
| Mary   |        300000 | f      |   33 |
| Olivia |         80000 | f      |   65 |
| Pilar  |         55000 | m      |   47 |
+--------+---------------+--------+------+
17 rows in set (0.00 sec)
```

26 GROUP BY

`GROUP BY` will put data into groups or buckets and then do aggregate functions.

[ ]:

[ ]:

27 ORDER BY

[3]:
```
# This will order by 1 or more columns
```

[ ]:

28 SQL LIMIT

The `LIMIT` keyword will restrict the number of rows that can appear in the output. For example, if you said: `SELECT * FROM salary LIMIT 5;` above would give a maximum of 5 rows (it would be 0 - 5 rows) thought.

```
-- watsontechworld.com

-- show databases;

use teaching_db;

show tables;

-- 1a. Show all data from salary table
select * from salary;

-- 2a. Show all data from salary table but limit to 5 rows
```

```
select * from salary LIMIT 5;

-- 3a. select name, gender, career, age, yearly_salary from salary table and sort by age
SELECT name, gender, career, age, yearly_salary FROM salary ORDER BY age;

-- 4a. select name, gender, career, age, yearly_salary from salary table and sort by age and li
SELECT name, gender, career, age, yearly_salary
FROM salary
ORDER BY age
LIMIT 10;

-- 5a. select name, gender, career, age, yearly_salary from salary table and sort by yearly sal
SELECT name, gender, career, age, yearly_salary
FROM salary
ORDER BY yearly_salary DESC;

-- 6a. select name, gender, career, age, yearly_salary from salary table and sort by yearly sal
SELECT name, gender, career, age, yearly_salary
FROM salary
ORDER BY yearly_salary DESC
LIMIT 7;


-- Using star database
--  HYG Database
-- https://www.astronexus.com/projects/hyg

-- columns
-- proper_star_name, parsecs_from_earth, light_years_from_earth, constellation_full_name, appar
-- proper_star_name - Full name of a star, if it exists (many stars have no proper name).
-- parsecs_from_earth - distance from earth in parsecs from earth
-- light_years_from_earth (converting from parsecs. 1 parsec = roughly 3.26156 light years)
-- constellation_full_name - full constellation name. The original data just has abbreviations
-- apparent_magnitude - Apparent magnitude (brightness as seen from Earth). The smaller the val

-- 1b. Show all data from star_data table. Don't limit the number of rows in MySQL Workbench
select * from star_data;

-- 2b. Show how many rows in star_data table (there are over 100,000 rows)
select COUNT(*) from star_data;

-- 3b. Note how adding limit N for N >=1 to above will not change the number of rows;
select COUNT(*) from star_data LIMIT 5;

-- 4b. Show all data from star_data table but limit to 100 rows
select * from star_data LIMIT 100;

-- 5b. Select proper_star_name, light_years_from_earth, constellation_full_name, apparent_magni
```

```
SELECT proper_star_name, light_years_from_earth, constellation_full_name, apparent_magnitude FR

-- 6b. Select proper_star_name, light_years_from_earth, constellation_full_name, apparent_magni
SELECT proper_star_name, light_years_from_earth, constellation_full_name, apparent_magnitude FR

-- 7b. Select proper_star_name, light_years_from_earth, constellation_full_name, apparent_magni
-- and sort by light_years_from_earth ascending
SELECT proper_star_name, light_years_from_earth, constellation_full_name, apparent_magnitude
FROM star_data
ORDER BY light_years_from_earth ASC;

-- 8b. Select proper_star_name, light_years_from_earth, constellation_full_name, apparent_magni
-- and sort by light_years_from_earth ascending and also limit to 100 rows
SELECT proper_star_name, light_years_from_earth, constellation_full_name, apparent_magnitude
FROM star_data
ORDER BY light_years_from_earth ASC
LIMIT 100;


-- 9b. Select proper_star_name, light_years_from_earth, constellation_full_name, apparent_magni
-- and make sure it must have a proper star name (it can't be NULL)
-- sort by light_years_from_earth ascending.
SELECT proper_star_name, light_years_from_earth, constellation_full_name, apparent_magnitude
FROM star_data
WHERE proper_star_name IS NOT NULL
ORDER BY light_years_from_earth ASC;

-- 10b. Select proper_star_name, light_years_from_earth, constellation_full_name, apparent_magn
-- and make sure it must have a proper star name (it can't be NULL)
-- sort by light_years_from_earth ascending
-- and limit it to 100 rows
SELECT proper_star_name, light_years_from_earth, constellation_full_name, apparent_magnitude
FROM star_data
WHERE proper_star_name IS NOT NULL
ORDER BY light_years_from_earth ASC
LIMIT 100;



-- 11b.  select constellation_full_name, average light years from earth grouped by constellatio
-- and make sure constellation name is not null
SELECT constellation_full_name, AVG(light_years_from_earth)
FROM star_data
WHERE constellation_full_name IS NOT NULL
GROUP BY constellation_full_name
ORDER BY AVG(light_years_from_earth);

-- 12b.  select constellation_full_name, average light years from earth grouped by constellatio
```

```
-- and make sure constellation name is not null and limit to 50 rows
SELECT constellation_full_name, AVG(light_years_from_earth)
FROM star_data
WHERE constellation_full_name IS NOT NULL
GROUP BY constellation_full_name
ORDER BY AVG(light_years_from_earth)
LIMIT 50;
```

[ ]:

29 Adding new column to the end and set default to NULL

```
ALTER TABLE salary
ADD COLUMN temp_column_2 text DEFAULT NULL;
```

now seeting new table

```
describe salary;
```

```
describe salary;
+---------------+------------+------+-----+---------+-------+
| Field         | Type       | Null | Key | Default | Extra |
+---------------+------------+------+-----+---------+-------+
| name          | text       | NO   |     | NULL    |       |
| career        | text       | YES  |     | NULL    |       |
| yearly_salary | int        | YES  |     | NULL    |       |
| gender        | varchar(2) | YES  |     | NULL    |       |
| age           | int        | YES  |     | NULL    |       |
| age_group     | text       | YES  |     | NULL    |       |
| temp_column   | text       | YES  |     | NULL    |       |
| temp_column_2 | text       | YES  |     | NULL    |       |
+---------------+------------+------+-----+---------+-------+
8 rows in set (0.00 sec)
```

[ ]:

[4]:
```
# note maybe default NULL isn't needed
```

30 Changing The Order of Columns in MySQL

https://dev.mysql.com/doc/refman/8.4/en/alter-table.html

ALTER TABLE tbl_name MODIFY [COLUMN] col_name column_definition [FIRST | AFTER col_name]

[5]:
```
# CHANGE EXAMPLE OUTPUT BELOW
```

```
For example, if I have table `salary` and its definition is the following:
salary;
```

```
+--------------+------------+------+-----+---------+-------+
| Field        | Type       | Null | Key | Default | Extra |
+--------------+------------+------+-----+---------+-------+
| name         | text       | NO   |     | NULL    |       |
| career       | text       | YES  |     | NULL    |       |
| yearly_salary | int        | YES  |     | NULL    |       |
| gender       | varchar(2) | YES  |     | NULL    |       |
| age          | int        | YES  |     | NULL    |       |
| age_group    | text       | YES  |     | NULL    |       |
+--------------+------------+------+-----+---------+-------+
```

Let's say, you wanted to put the career as the 2nd column from left to 2nd column from right. Here is how you can do it.

31 example

```
ALTER TABLE salary
MODIFY COLUMN career text
AFTER temp_column;
```

[ ]:

32 Appendix

[ ]:

33 MySQL Commands

33.1 showing databases

```
show databases
```

[ ]:

You might see something like this:

show databases; +——————–+ | Database | +——————–+ | information_schema | | mysql | | performance_schema | | sys | | teaching_db | | wordpress_db | +——————–+ 6 rows in set (0.01 sec)

34 Select or get into a specific database

```
use some_database_name
```

Let's say you want to use the teaching_db database. To start using that, in MySQL, you would type something like the following:

```
use teaching_db
```

You might see something like this:

Reading table information for completion of table and column names You can turn off this feature to get a quicker startup with -A Database changed

[ ]:

35 Showing tables in a database

To show all the tables in a database, once a database has been selected, you can use a similar command like the following:

```
show tables;
```

You might see something like:

show tables; +——————————+ | Tables_in_teaching_db | +——————————+ | salary | | star_data | +——————————+ 2 rows in set (0.01 sec)

[ ]:

36 Note on capitalization of keywords in SQL

Technically, keywords such as `SELECT`, `ORDER BY`, `GROUP BY`, `FROM`, and other SQL keywords actually don't need to be capitalized, and the system usually won't care if you do. But, it is good practice because: * It can make it easier to quickly see SQL keywords

[ ]:

37 pandas code for interaction with a database

df.to_sql() the .to_sql() method will "Write records stored in a DataFrame to a SQL database.", as mentioned in its documentation

[ ]:

[ ]:
```
# References
```

[ ]:
```
# https://dev.mysql.com/doc/refman/8.4/en/drop-database.html
```

[ ]:

[ ]:
```
# NOTE TO SELF. SHOW ERRORS BY INTENTIONALLY MAKING SOME
```

[ ]: