**Create a database named myDB**

CREATE DATABASE myDB;

**Create a table (employees) within the database myDB**

CREATE TABLE employees (

employee\_id INT,

first\_name VARCHAR(50),

last\_name VARCHAR(50),

hourly\_pay DECIMAL(5, 2),

hire\_date DATE

);

**To Select all the rows and columns from the table**

SELECT \* FROM employees;

**To rename the table**

RENAME TABLE employees TO workers;

**To alter (add the column or rename the column) the table**

ALTER TABLE employees

ADD phone\_number VARCHAR(15);

**To change the position of the column**

ALTER TABLE employees

MODIFY email VARCHAR (100)

AFTER last\_name;

**To drop a column**

ALTER TABLE employees

DROP COLUMN email;

**Insert rows into the table**

INSERT INTO employees

VALUES (1, "Eugene", "Krabs", 25.50, "2023-01-02"),

(2, "Squidward", "Tentacles", 15.00, "2023-01-03"),

(3, "Spongebob", "Squarepents", 12.50, "2023-01-04"),

(4, "Patrick", "Star", 12.50, "2023-01-05"),

(5, "Sandy", "Cheeks", 17.25, "2023-01-06");

**Rows with missing data**

VALUES (6, "Sheldon", "Plankton"); #This will give you an error, instead do:

INSERT INTO employees (employee\_id, first\_name, last\_name)

VALUES (6, "Sheldon", "Plankton");

**Retrieving Full name of every employee**

SELECT first\_name, last\_name

FROM employees;

**To change the order of the column to retrieve full name**

SELECT last\_name,first\_name

FROM employees;

**If querying something specific, such as (criteria => employee\_id = 5)**

SELECT \*

FROM employees

WHERE employee\_id = 5;

**If querying employees whose hourly rate is greater than $15**

SELECT \*

FROM employees

WHERE hourly\_pay >= 15;

**If querying employees whose hire date is before 2023-01-03**

SELECT \*

FROM employees

WHERE hire\_date <= "2023-01-03";

**To query if something is not equal to (Not comparison operator (!=))**

SELECT \*

FROM employees

WHERE employee\_id != 1;

**To update the fields (single column) in the table (eg: hourly pay for employee\_id = 6)**

UPDATE employees

SET hourly\_pay = 10.25

WHERE employee\_id = 6;

**To update the fields (multiple column) in the table (eg: hourly pay and hire\_date for employee\_id = 6)**

UPDATE employees

SET hourly\_pay = 10.50,

hire\_date = "2023-01-07"

WHERE employee\_id = 6;

**To delete a row**

DELETE from employees

WHERE employee\_id = 6;

**Unique Constraints -> ensures that the values in the column are different. The UNIQUE keywords below allow to have a unique product name in the column. We will now create the second table “products” within same database.**

CREATE TABLE products(

product\_id INT,

product\_name VARCHAR(25) UNIQUE,

price DECIMAL (4, 2)

);

**If in case, we forgot to write UNIQUE keywords the alternate code below could be used.**

ALTER TABLE products

ADD CONSTRAINT

UNIQUE(product\_name);

**Add values in the products table**

INSERT INTO products

VALUES (100, "hamburger", 3.99),

(101, "fries", 1.89),

(102, "soda", 1.00),

(103, "ice cream", 1.49);

**Use of NOT NULL constraints**

CREATE TABLE products(

product\_id INT,

product\_name VARCHAR(25),

price DECIMAL (4, 2) NOT NULL

);

**For an already built table, that did not have any NOT NULL constraints added before.**

ALTER TABLE products

MODIFY price DECIMAL(4,2) NOT NULL;

**Evaluate the CHECK CONSTRAINTS => Used to limit what values can be placed in a column. Less than $10 hourly\_pay violates the table rule.**

ALTER TABLE employees

ADD CONSTRAINT chk\_hourly\_pay CHECK(hourly\_pay >= 10.00);

**To delete the check constraints**

ALTER TABLE employees

DROP CHECK chk\_hourly\_pay;

**Evaluate the DEFAULT CONSTRAINTS => When inserting a row, we don’t specify the value for the column, by default we can add the value that we set.**

ALTER TABLE products

ALTER price SET DEFAULT 0;

**Primary Key Constraints in MySQL => Applied to a column where each value in that column must be both UNIQUE and NOT NULL. It’s typically used as a Unique Identifier. A table can have only one primary key constraint.**

CREATE TABLE transactions(

transaction\_id INT PRIMARY KEY,

amount DECIMAL(5, 2)

);

**Auto increment attribute in MYSQL => Applied to a column which is a key, especially when inserting a data in a row, primary key can be populated automatically by increasing each subsequent row by 1.**

CREATE TABLE transactions(

transaction\_id INT PRIMARY KEY AUTO\_INCREMENT,

amount DECIMAL(5,2)

);

**FOREIGN KEY CONSTRAINTS => A foreign key is a primary key from one table that can be found within a different table. Using a FOREIGN KEY, we can establish a relationship between two tables. Two primary benefits to this: a. In my transactions table if I were to take a look at the customer\_id who initiated this transaction, I could refer to the customer table and find the first and last name of that customer. b. Creates link between two tables which prevents any action that would destroy that link between them.**

For this situation, we will first create customers table.

CREATE TABLE customers(

customer\_id INT PRIMARY KEY AUTO\_INCREMENT,

first\_name VARCHAR(50),

last\_name VARCHAR(50)

);

And populate the customers table

INSERT INTO customers (first\_name, last\_name)

VALUES (“Fred”, “Ross”),

(“Cersie”, “Stark”)

(“Rachel”,”Brown”);

Now we’ll create a link between our transactions table and the customer table, via our customer\_id. First we will drop all the transactions table earlier and will create it again with the third column customer\_id and we will add a FOREIGN KEY constraint to this column.

CREATE TABLE transactions(

transaction\_id INT PRIMARY KEY AUTO\_INCREMENT,

amount DECIMAL(5,2),

customer\_id INT,

FOREIGN KEY (customer\_id) REFERENCES customers (customer\_id)

);

**To drop a FOREIGN KEY**

ALTER TABLE tranasctions

DROP FOREIGN KEY transaction\_ibfk\_1

**To give FOREIGN KEY a unique name.**

ALTER TABLE transactions

ADD CONSTRAINT fk\_customer\_id

FOREIGN KEY(customer\_id) REFERENCES customers(customer\_id);

**JOINS in SQL => Join is a clause that is used to combine rows from two or more tables based on a related columns between them such as a FOREIGN KEYS.**

INNER JOINS (Select all rows from two tables that have matching customer\_id)

SELECT \*

FROM transactions INNER JOIN customers

ON transactions.customer\_id = customers.customer\_id;

In INNER JOIN, to only display certain columns

SELECT transaction\_id, amount, first\_name, last\_name

FROM transactions INNER JOIN customers

ON transactions.customer\_id = customers.customer\_id;

LEFT JOIN – We are going to display everything from the table on the left. Pulls any matching rows from the right even if there is no data.

SELECT \*

FROM transactions LEFT JOIN customers

ON transactions.customer\_id = customers.customer\_id;

RIGHT JOIN – We are going to display everything from the table on the right. Pulls any matching rows from the left even if there is no data.

SELECT \*

FROM transactions RIGHT JOIN customers

ON transactions.customer\_id = customers.customer\_id;

**FUNCTION IN SQL => A fuction is a stored program that you can pass parameters into to return a value.**

COUNT of transactions that in certain column.

SELECT COUNT(amount)

FROM transactions;

Find maximum amount in the column amount. (“AS maximum” is the alias we passed in the code to display our results.

SELECT MAX(amount) AS maximum

FROM transactions;

Find minimum amount in the column amount.

SELECT MIN(amount) AS minimum

FROM transactions;

Find average amount in the column amount.

SELECT AVG(amount) AS average

FROM transactions;

Find average sum in the column amount.

SELECT SUM(amount) AS sum

FROM transactions;

Combine two columns (first\_name and last\_name) in an employee’s table to use CONCAT function and create a new column with this value.

SELECT CONCAT(first\_name, " ", last\_name) AS full\_name

FROM employees;

**LOGICAL OPERATORS in SQL => keywords used to combine more than one condition.**

First add new column “job” in employees table

ALTER TABLE employees

ADD COLUMN job VARCHAR(25) AFTER hourly\_pay;

Update the values in the column “jobs” (below is the example for one, follow the same for others)

UPDATE employees

SET job = "janitor"

WHERE employee\_id = 6;

AND Operators (Condition = Find cooks hired before January 5th both conditions need to be true)

SELECT \*

FROM employees

WHERE hire\_date < "2023-01-05" AND job = "cook";

OR Operator (Condition = any cooks or cashiers (any one condition needs to be true))

SELECT \*

FROM employees

WHERE job = "cook" OR job = "cashier";

NOT Operator (reverses anything you say. Find any employees that are not a manager)

SELECT \*

FROM employees

WHERE NOT job = "manager";

Combine the Logical Operators. Let’s find anybody that’s not a manager and not an assistant manager)

SELECT \*

FROM employees

WHERE NOT job = "manager" AND NOT job = "assistant manager";

BETWEEN Operator (Used within a single column. Lets say employees hired between certain dates)

SELECT \*

FROM employees

WHERE hire\_date BETWEEN "2023-01-04" AND "2023-01-07";

IN Operator (Used to find values that are within a set. Let’s say the job is in “cook”, “cashier”, and “janitor”)

SELECT \*

FROM employees

WHERE job IN("cook", "cashier", "janitor");

**WILD CARD CHARACTERS (% \_) => Used to substitute one or more characters in a string. (The following codes are for different character and will give different results. First two is using separately and last is using together)**

SELECT \* FROM employees

WHERE first\_name LIKE "s%";

SELECT \* FROM employees

WHERE job LIKE "\_ook";

SELECT \* FROM employees

WHERE job LIKE "\_a%";

**ORDER BY clause => Sorts the result of the query either in ascending or the descending order based on which column we list. (EG: list employees in an alphabetical order). Default is ascending order. To list in descending order type DESC.**

SELECT \* FROM employees

ORDER BY last\_name DESC;

**LIMIT clause => used to limit the number of records that are queried. Very useful if you’re working with lots of data. Also can be used to display a large data on different pages aka pagination.**

SELECT \* FROM customers

LIMIT 4;

We can also combine LIMIT clause with the ORDER BY clause

SELECT \* FROM customers

ORDER BY last\_name LIMIT 2;

**UNION Operator => combine the results of two or more SELECT statements. Union will not work if the number of columns in 2 tables are different. But we can apply union to distinct columns.**

SELECT first\_name, last\_name FROM employees

UNION

SELECT first\_name, last\_name FROM customers;