**README FILE:** ByAndrew Brown.

1. **Database Design:**

* First, a database named “projectMYSQL” would be created to for a store transaction. This database would contain sales, customers, and products information.
* **SQL Code**

CREATE DATABASE projectMYSQL;

USE projectMYSQL;

* **Explanation.**

The database was created/activated using the codes above. The code creates the database using the ‘CREATE DATABASE’ function, and the ‘USE’ function allows using the created database to create, and add values to tables. The semi-colon ends each code statement which allows the use of multiple codes at an instance.

**Design of Tables:** here, tables of products, categories, customers, and sales are created to ensure ensures the database has data essential to the business. This also ensures that the database obeys every rule of database/SQL design. Atomicity is one of the requirements. The tables’ design are in detail below.

**Table: Category.**

* The table is created to have the product category or business sector attributed to the product in the store. It has a primary key as the “category\_id”. Also, has the name of the business sectors as unique (i.e no similar names).
* **SQL Code**

CREATE TABLE category (

category\_id INT PRIMARY KEY,

category\_name VARCHAR (255) NOT NULL,

UNIQUE (category\_name)

);

* **Explanation:**

category\_id: the unique identifier of the business category.

category\_name: name of the business sector.

**Table: Products.**

* The table has essential information about the products in the store. It also has foreign keys which links the table to the category table. The foreign allows the interaction of this table with the category table. All variables in the table has contents in every row, thus, ascribed as “not null”. Also, the price of each product contain cost with 2 decimal places.
* **SQL Code.**

CREATE TABLE products (

product\_id INT PRIMARY KEY,

name VARCHAR (255) NOT NULL,

category\_id INT NOT NULL,

stock\_quantity INT NOT NULL,

price\_per\_quantity DECIMAL(10, 2) NOT NULL,

FOREIGN KEY (category\_id) REFERENCES category(category\_id),

UNIQUE (name)

);

* **Explanation:**

product\_id: unique identifier of the product in store.

name: name of the product.

category\_id: category identification number which the product is classed.

stock\_quantity: quantity of the product available in store.

price\_per\_quantity: cost per unit of the product.

**Table: Customers.**

* The table has information of the customers of the store. Has a unique primary key as “customer\_id”, and unreatable email address of the customers.
* **SQL Code.**

CREATE TABLE customers (

customer\_id INT PRIMARY KEY,

customer\_name VARCHAR (255) NOT NULL,

email VARCHAR (255),

gender VARCHAR (255),

city VARCHAR (255),

country VARCHAR (255),

UNIQUE (email)

);

* **Explanation:**

customer\_id: a unique identifier of the customers.

customer\_name: has name of the customers.

email: the email address of the customers.

gender: the gender of each customer.

city: living location of each customer.

country: the country the customer inhabits.

**Table: Sales.**

* The sales table has information on sales performed in the store. Has a unique sale identification number (primary key) which record the amount of a product at one purchase by a customer. However, every non-similar product purchased at a point by a customer has a unique sale identification number. The table also has product\_id, and customer\_id numbers as foreign keys reference from the products and customers’ table respectively. In the table, every variable require an input for a sale.
* **SQL Code.**

CREATE TABLE sales (

sales\_id INT PRIMARY KEY,

product\_id INT NOT NULL,

customer\_id INT NOT NULL,

date DATE NOT NULL,

stock\_amount INT NOT NULL,

total\_price DECIMAL(10, 2) NOT NULL,

FOREIGN KEY (product\_id) REFERENCES products(product\_id),

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

);

* **Explanation:**

sales\_id: unique sale identifier of a product purchased at a point.

product\_id: product identifier value.

customer\_id: customer identifier.

date: day of transaction in format “YYYY-MM-DD”.

stock\_amount: amount of the product purchased at a point.

total\_price: sum of amount spent on a product cost. A product of the cost and quantity purchased at a point by a customer.

1. **Data Population:** data type of each variable specified as while the tables were created was used. Here information are added into the tables.

**Categories Table.**

* Here, the building/construction, food, manufacturing, and the academic industry have been identified.
* **SQL Code**

INSERT INTO category (category\_id, category\_name)

VALUES (1, "building/construction");

INSERT INTO category (category\_id, category\_name)

VALUES (2, "food");

INSERT INTO category (category\_id, category\_name)

VALUES (3, "manufacturing");

INSERT INTO category (category\_id, category\_name)

VALUES (4, "academics");

**Products Table.**

* Products available at store are added here. The products include spaghetti, shovel, welding gun, glue, glues (for a separate industry, from previously mentioned glue), short notes (notepads), head pans, chocolates, and grinding machines.
* **SQL Code**

INSERT INTO products (product\_id, name, category\_id, stock\_quantity, price\_per\_quantity)

VALUES (1, "spaghetti", 2, 100, 0.99);

INSERT INTO products (product\_id, name, category\_id, stock\_quantity, price\_per\_quantity)

VALUES (2, "shovel", 1, 50, 3.45);

INSERT INTO products (product\_id, name, category\_id, stock\_quantity, price\_per\_quantity)

VALUES (3, "welding gun", 3, 25, 2.99);

INSERT INTO products (product\_id, name, category\_id, stock\_quantity, price\_per\_quantity)VALUES (4, "short notes", 4, 35, 1.99);

INSERT INTO products (product\_id, name, category\_id, stock\_quantity, price\_per\_quantity)

VALUES (5, "head pan", 1, 13, 9.99);

INSERT INTO products (product\_id, name, category\_id, stock\_quantity, price\_per\_quantity)

VALUES (6, "chocolate", 2, 200, 0.99);

INSERT INTO products (product\_id, name, category\_id, stock\_quantity, price\_per\_quantity)

VALUES (7, "glue", 4, 40, 3.55);

INSERT INTO products (product\_id, name, category\_id, stock\_quantity, price\_per\_quantity)

VALUES (8, "glues", 1, 10, 1.55);

INSERT INTO products (product\_id, name, category\_id, stock\_quantity, price\_per\_quantity)

VALUES (9, "grinding machine", 4, 6, 50.99);

INSERT INTO products (product\_id, name, category\_id, stock\_quantity, price\_per\_quantity)

VALUES (10, "floor brush", 2, 15, 0.88);

**Customers Table:**

* This table has information of customers. This table is designed to help in the understanding of customers’ behaviour. A total of seven customers are recorded in the table.
* **SQL Code**

INSERT INTO customers (customer\_id, customer\_name, email, gender, city, country)

VALUES (1, "Mfon brown", "mfonb@yahoo.com", "male", "Wolverhampton", "United Kingdom");

INSERT INTO customers (customer\_id, customer\_name, email, gender, city, country)

VALUES (2, "Mfon AKpan", "mfonA@yahoo.com", "female", "Uyo", "Nigeria");

INSERT INTO customers (customer\_id, customer\_name, email, gender, city, country)

VALUES (3, "Emelda Eze", "ezee@yahoo.com", "female", "Enugu", "Nigeria");

INSERT INTO customers (customer\_id, customer\_name, email, gender, city, country)

VALUES (4, "Ellie Adelle", "ellieA@yahoo.com", "female", "Birmingham", "United Kingdom");

INSERT INTO customers (customer\_id, customer\_name, email, gender, city, country)

VALUES (5, "John Doe", "johndoe@yahoo.com", "male", "Washington", "United States");

INSERT INTO customers (customer\_id, customer\_name, email, gender, city, country)

VALUES (6, "andrew Drinkpepsi", "andyp@yahoo.com", "male", "Newcastle", "United Kingdom");

INSERT INTO customers (customer\_id, customer\_name, email, gender, city, country)

VALUES (7, "Fiona Alexandra", "fbiby@yahoo.com", "female", "Birmingham", "United Kingdom");

**Sales Table.**

* This table has two special data types. These data types are the 2 decimal numbers for the total price, the DATE format for the date of purchase. The table currently has eleven recorded unique product purchase at an instant.
* **SQL Code**

INSERT INTO sales (sales\_id, product\_id, customer\_id, date, stock\_amount, total\_price)

VALUES (1, 3, 4, "2024-12-21", 3, 5.50);

INSERT INTO sales (sales\_id, product\_id, customer\_id, date, stock\_amount, total\_price)

VALUES (2, 3, 1, "2023-09-21", 2, 5.50);

INSERT INTO sales (sales\_id, product\_id, customer\_id, date, stock\_amount, total\_price)

VALUES (3, 5, 4, "2024-04-21", 1, 9.99);

INSERT INTO sales (sales\_id, product\_id, customer\_id, date, stock\_amount, total\_price)

VALUES (4, 2, 7, "2020-12-21", 4, 7.00);

INSERT INTO sales (sales\_id, product\_id, customer\_id, date, stock\_amount, total\_price)

VALUES (5, 6, 2, "2022-04-16", 5, 4.50);

INSERT INTO sales (sales\_id, product\_id, customer\_id, date, stock\_amount, total\_price)

VALUES (6, 10, 5, "2020-02-24", 3, 2.50);

INSERT INTO sales (sales\_id, product\_id, customer\_id, date, stock\_amount, total\_price)

VALUES (7, 9, 3, "2014-10-14", 1, 50.99);

INSERT INTO sales (sales\_id, product\_id, customer\_id, date, stock\_amount, total\_price)

VALUES (8, 7, 6, "2010-12-28", 3, 10.60);

INSERT INTO sales (sales\_id, product\_id, customer\_id, date, stock\_amount, total\_price)

VALUES (9, 8, 7, "2024-05-01", 2, 3.10);

INSERT INTO sales (sales\_id, product\_id, customer\_id, date, stock\_amount, total\_price)

VALUES (10, 6, 4, "2024-02-26", 5, 4.50);

INSERT INTO sales (sales\_id, product\_id, customer\_id, date, stock\_amount, total\_price)

VALUES (11, 1, 5, "2023-02-28", 2, 1.80);

1. **Queries for Analysis:** the following queries were to perform following tasks below

* The code below displays all product with prices and stock quantity. It obtains this information from the products table were the necessary data is stored.

**SQL Code**

SELECT name, price\_per\_quantity, stock\_quantity

FROM products;

* The code below identifies products low in stock. Precisely, products which has quantity in store lower than 21. The product are then arranged in order of fewer stock beginning with one with lowest.

**SQL Code**

SELECT name, stock\_quantity

FROM products

WHERE stock\_quantity <= 20

ORDER BY stock\_quantity ASC;

* The code below calculates the total revenue generated from sales. This is done by adding up all total price sales (amount purchased multiplied by the unit price, and subtracting discount) made.

**SQL Code**

SELECT SUM(total\_price)

FROM sales;

* The code below lists the top three bestselling products. It groups the quantity of the products purchased by the products’ identification number. That the name of the product purchased be assigned to the entire quantity of the products purchased, the sales table is joined to the products table. Finally, the total number of the product purchased is arranged by product name from top to bottom, then, the list is limited to 3 rows.

**SQL Code**

SELECT p.name, SUM(s.stock\_amount) AS quantity\_sold

FROM sales AS s

LEFT JOIN products AS p

ON p.product\_id = s.product\_id

GROUP BY p.product\_id

ORDER BY SUM(stock\_amount) DESC

LIMIT 3;

* The code below shows all purchases made by a specific customer (using a customer id and name). the code counts the unique sales\_id of a purchase (of a type of product at an instance) by a customer, and returns the count of every product purchase made by every customer. Optionally, the list is sorted to show customers with most purchases as top-on-list. Also, to see the name of the customers alongside their identification number, the sales table is joined with the customers table.

**SQL Code**

SELECT s.customer\_id, c.customer\_name, COUNT(s.sales\_id) AS number\_of\_purchases

FROM sales AS s

LEFT JOIN customers AS c

ON s.customer\_id = c.customer\_id

GROUP BY s.customer\_id

ORDER BY number\_of\_purchases DESC;

* The code below calculates the total number of products sold in each category. The code adds up the number of products sold from the sales table for the various category of the products. To return the result based on category names, the sales table is first joined with the product table (which has the category\_id as a foreign key. Afterwards, another join is made with the category table to class the result as expected.

**SQL Code**

SELECT cat.category\_name, SUM(s.stock\_amount) AS numbers\_sold

FROM sales AS s

LEFT JOIN products as P

ON p.product\_id = s.product\_id

LEFT JOIN category AS cat

ON cat.category\_id = p.category\_id

GROUP BY cat.category\_name;