Advanced Data Management (CMM524)

Solution to Laboratory #3: More SQL

1. Aims

- To use SQL commands to retrieve, update, and delete data in a database.

2. Outcomes

In completing this exercise, you should be able to:

- Retrieve data from a database using various SELECT statements.
- Update data using the UPDATE statement.
- Delete data using the DELETE statement.

3. Creating the Database & Tables

Our online shopping domain has the following tables:

I started designed an ER model for the domain. Then I mapped the entities and relationships in the ER model to tables in the Relational Data Model. I do not show you the ER model here but it is expected to be the initial product of my analysis.

Table `product`:

Column	Type	NULL	Key	Extra
`productID`	`varchar(16)`	NO	Primary key	
`description`	`varchar(255)`	NO		
`currentPrice`	`float`	NO		default value 0.0

I assume the product ID has a maximum length of 16 characters, and it may have non-numeric characters. Most items have a bar code which can be used for this purpose. If you assume it to be a pure number than you can use an `int` type.

Table `customer`:

Column	Type	NULL	Key
`email`	`varchar(64)`	NO	Primary key
`name`	`varchar(64)`	NO	
`address`	`varchar(255)`	NO	
`country`	`varchar(32)`	NO	

For customers I assume the email is unique and is used as the primary key.

Table `purchase`:

Column	Type	NULL	Key	Extra
`orderID`	`int`	NO	Primary key	AUTO INCREMENT
`customerEmail`	`varchar(64)`	NO		
`productID`	`varchar(16)`	NO		

purchasePrice	float	NO		

I do not want to enter the order ID manually. So I made it an AUTO INCREMENT field. You will notice the foreign keys of product and customer I this table. Note that purchase price belongs to an instance of a purchase relationship as the purchase price may not be the current price.

Create a new database just for this lab:

- Login as `training`, create a database `lab03`.
  - Note: If you found a database `lab03` in MySQL before you create any, may be someone has used this VM before. You can `DROP` the database and re-create it from scratch.
  - To create the database:
    ```
    CREATE DATABASE lab03;
    ```
  - The *training* user has the privilege to create a database. Otherwise we will need DBA permission to create the database.
  - You can use any sensible name for the database, as long as you know what data are stored in it.
- Switch to the database `lab03`.
  - To switch the current database to `lab03`:
    ```
    USE lab03;
    ```
  - You must set the current database do this before any operation can be done on it. If you haven't, MySQL will complain as it doesn't know which database to work on. Read the error message!
- Use SQL to create tables `product`, `customer`, and `purchase`. according to the schema described above.

The following SQL statements create the 3 tables:

```
CREATE TABLE product
     (productID varchar(16),
     description varchar(255) NOT NULL,
     currentPrice float DEFAULT 0.0,
     PRIMARY KEY (productID)
     );

CREATE TABLE customer
     (email varchar(64),
     name varchar(255) NOT NULL,
     address varchar(255) NOT NULL,
     country varchar(255) NOT NULL,
     PRIMARY KEY (email)
     );

CREATE TABLE purchase
   (orderID int AUTO_INCREMENT,
   customerEmail varchar(64) NOT NULL,
   productID varchar(16) NOT NULL,
   purchasePrice float,
   PRIMARY KEY(orderID)
   );
```

Note if a column is used as the primary key, it is also unique and NOT NULL. So there is no need to specify these in the schema.

4. Populating the Tables

Populate your tables with the following data:

Products:

Product ID	Description	Current Price
p001	Echo Dot	49.99
p002	Echo	89.99
p003	Echo Show	199.99
p004	Echo Plus	139.99
p005	Wood filament 1.75mm 1kg	29.99
p006	iPhone X 64GB	999
p007	iPad Pro	619
p008	Google Home	99.99
p009	Google Wifi	219.99
p010	Andrex 10 rolls	14.99

Customers:

Email	Name	Address	Country
frodo@shire.net	Frodo Baggins	1 Bag End	The Shire
gandalf@gmail.com	Gandalf	Lorien Gardens	Valinor
aragorn@palace.gd	Aragorn	The Palace	Gondor
legolas@mirkwood.org	Legolas	Woodland	Mirkwood
sauron@mordor.evil	Sauron	Barad-dur	Mordor

Purchases:

Customer email	Product ID	Purchase price
frodo@shire.net	p010	14.99
frodo@shire.net	p010	34.99
gandalf@gmail.com	p010	9.99
gandalf@gmail.com	p007	599
aragorn@palace.gd	p010	18.99
aragorn@palace.gd	p009	1099.99
legolas@mirkwood.org	p010	10.99
legolas@mirkwood.org	p001	40.99
legolas@mirkwood.org	p005	29.99

The following SQL INSERT statements will do the work:

```
INSERT INTO product VALUES ('p001','Echo Dot',49.99);
INSERT INTO product VALUES ('p002','Echo',89.99);
INSERT INTO product VALUES ('p003','Echo Show',199.99);
INSERT INTO product VALUES ('p004','Echo Plus',139.99);
INSERT INTO product VALUES ('p005','Wood filament
1.75mm 1kg',29.99);
INSERT INTO product VALUES ('p006','iPhone X
64GB',999);
INSERT INTO product VALUES ('p007','iPad Pro',619);
INSERT INTO product VALUES ('p008','Google
Home',99.99);
INSERT INTO product VALUES ('p009','Google
Wifi',219.99);
INSERT INTO product VALUES ('p010','Andrex 10
rolls',14.99);
INSERT INTO customer VALUES ('frodo@shire.net','Frodo
Baggins','1 Bag End','The Shire');
INSERT INTO customer VALUES
('gandalf@gmail.com','Gandalf','Lorien Gardens','The
Shire');
INSERT INTO customer VALUES
('aragorn@palace.gd','Aragorn','The Palace','Gondor');
INSERT INTO customer VALUES
('legolas@mirkwood.org','Legolas','Woodland','Mirkwood'
);
INSERT INTO customer VALUES
('sauron@mordor.evil','Sauron','Barad-dur','Mordor');
INSERT INTO purchase
(customerEmail,productID,purchasePrice) VALUES
('frodo@shire.net','p010',14.99);
INSERT INTO purchase
(customerEmail,productID,purchasePrice) VALUES
('frodo@shire.net','p005',34.99);
INSERT INTO purchase
(customerEmail,productID,purchasePrice) VALUES
('gandalf@gmail.com','p010',9.99);
INSERT INTO purchase
(customerEmail,productID,purchasePrice) VALUES
('gandalf@gmail.com','p007',599);
INSERT INTO purchase
(customerEmail,productID,purchasePrice) VALUES
('aragorn@palace.gd','p010',18.99);
INSERT INTO purchase
(customerEmail,productID,purchasePrice) VALUES
('trump@whitehouse.gov','p009',1099.99);
INSERT INTO purchase
(customerEmail,productID,purchasePrice) VALUES
('legolas@mirkwood.org','p010',10.99);
INSERT INTO purchase
(customerEmail,productID,purchasePrice) VALUES
('legolas@mirkwood.org','p001',40.99);
```

```
INSERT INTO purchase
(customerEmail,productID,purchasePrice) VALUES
('legolas@mirkwood.org','p005',29.99);
```

Notes:

- You can use the MySQL text client interactively to enter the above data but it is liable to mistakes.
- Alternatively you can create a text file with SQL INSERT statements and "source" the file later.
    - I strongly encourage you to compose SQL files instead of typing it straight to the MySQL client/console, because:
        - You can easily modify a file and re-source it if you made any mistake.
        - You can copy-and-paste an INSERT statement and modify it quickly to insert other data.

5. Making Queries

Write SQL statements to perform the following tasks:

5.1. Simple Retrieval

- Show all details of all customers.

```
SELECT * FROM customer;
```

- Show all product names and their current prices.

```
SELECT description, currentPrice FROM product;
```

5.2. Retrieval with Simple Filtering

- Show details of product with ID `p010`.

```
SELECT * FROM product WHERE productID='p010';
```

- Show all purchases made by Donald Trump. (Hint: You can simply filter by Donald Trump's email.)

```
SELECT * FROM purchase WHERE
customerEmail='aragorn@palace.gd';
```

- Show product IDs and names with a current price between £20 and £100.

```
SELECT productID, description FROM product WHERE
currentPrice>=20 AND currentPrice<=100;
```

- Find all products with the word "echo" in their names.

```
SELECT * FROM product WHERE description LIKE '%echo%';
```

5.3. Retrieval with Ordering

- Show all product names in alphabetic order.

```
SELECT description FROM product ORDER BY description;
```

- Show all product names and prices in decreasing order.

```
SELECT description, currentPrice FROM product ORDER BY
currentPrice DESC;
```

- List the countries where the customers are coming from in alphabetical order, with no duplicate.

```
SELECT DISTINCT country FROM customer ORDER BY country;
```

5.4. Retrieval with Join

- Show all customers with the product IDs they have purchased. (Hint: Join required between `customer` and `purchase` tables.)

```
SELECT customer.name,  purchase.productID from purchase
INNER JOIN customer ON
purchase.customerEmail=customer.email;
```

- Show all customers with the product name and price they have purchased. (Hint: You may need an `INNER JOIN` followed by another `INNER JOIN`.)

```
SELECT customer.name, product.description,
purchase.purchasePrice FROM purchase INNER JOIN customer
ON customer.email=purchase.customerEmail INNER JOIN
product ON product.productID=purchase.productID;
```

(Note: I prefixed the columns/fields with the table name to indicate where the field comes from. Prefixes are not required if there is no ambiguity.)

5.5. Retrieval with Aggregation

- Find the average purchase price across all orders.

```
SELECT AVG(purchasePrice) FROM purchase;
```

- Find the total amount over all orders.

```
SELECT SUM(purchasePrice) FROM purchase;
```

- Find the total purchase amount made by customers in "The Shire" only. (Hint: Join required between `purchase` and `customer` tables.)

```
SELECT SUM(purchase.purchasePrice) FROM purchase INNER
JOIN customer ON purchase.customerEmail=customer.email
WHERE customer.country='The Shire';
```

- Show how many times a product has been ordered, with the product IDs only. (Hint: No join required. Only the `purchase` table is needed.)

```
SELECT COUNT(orderID) FROM purchase GROUP BY productID;
```

- Show all product names which have been purchased and the number of orders on each product. (Hint: Join required between the `purchase` and `product` tables.)

```
SELECT product.description, COUNT(purchase.orderID) from
purchase INNER JOIN product ON
purchase.productID=product.productID GROUP BY
purchase.productID;
```

6. Updating Data

Write SQL statements to perform the following tasks:

Simple retrieval:

- Change the price of "Andrex 19 rolls" to 18.99.

```
UPDATE product SET currentPrice=18.99 WHERE
productID='p010';
```

- Reduce the current price of all "`Echo`" products by 10%.

```
UPDATE product SET currentPrice=currentPrice*0.9 WHERE
description LIKE '%echo%';
```

7. Deleting Data

Write SQL statements to perform the following tasks:

- Delete all purchases where ordered an “`Echo`” product. (Hint: You can get all product IDs of Echo products first. Then use a “`WHERE … IN...`” condition.)

```
DELETE FROM purchase WHERE productID IN (SELECT productID
      FROM product WHERE description LIKE '%echo%');
```

- Delete all products with the keyword “Echo”.

```
DELETE FROM product WHERE description LIKE '%echo%’;
```