Final Report — Pharmaceutical Drug Deliveries

Kelsey Chong, Amanda Collins, Brooke Law, Michael Le, George Papadopoulos

Winter 2023

23 March 2023

Table of Contents

1. Summary (1 page executive summary)

Summary of Whole Project:

1. What We Did

   Our team collaborated on a project regarding a Pharmaceutical Database Management System, also referred to as DBMS. We set objectives for ourselves as described in the Problem Statement to create an efficient database. With that, we wanted to ensure we could perform queries and analyze data to establish trends in order to make appropriate business decisions. In order to accomplish these objectives, we have developed our a final relational schema through multiple revisions, as well as an ERD—to indicate entities and relationships between them. Then, we utilized LiveSQL in order to host our database which contained all of our entities as well as inserted data that we have come up with using SQL which allowed us to test our database system to ensure that all objectives are met.

2. Why We Chose This Topic

   We chose this topic because it is very relevant. The pharmaceutical industry is one that has affected us all in some capacity. So, having the chance to focus our project on creating a database that addresses an issue like this seemed like a great opportunity. Furthermore, the pharmaceutical industry tends to deal with hundreds and even thousands of drugs, customers, and prescriptions in the market that need to be stored in a database. Hence, this topic gave us a realistic problem to analyze and think of considerations to be made in order to establish a successful DBMS. Lastly, this is something we can apply to our future careers especially if we pursue a career in database administration.

3. How It Went

   Overall, this project went rather well. We were able to work efficiently as a team communicating through texts to make sure that everyone was on the same page. The greatest challenge was during milestone 1 and figuring out how to combine our ERD's to come up with the most accurate version. However, once we were able to overcome that and receive feedback for our diagram and outline during the milestone 1 review we were able to make great progress. During these last 2 weeks especially we have stayed in contact to update one another and make sure that we are each completing the parts of the assignment we need to complete like the Data manipulation, DML, and individual summaries.

4. What We produced

   Abiding by our objectives, our team was able to establish and develop an ERD for our pharmaceutical system as well as a relational schema which helped us create SQL code for our implementation of our database management system. In order to test our system, we have inserted data onto the database with the INSERT functionality of SQL and have performed various database updates which includes using UPDATE and DELETE to manipulate the data as well as displaying the data using SELECT. All in all, we were able to produce a well-designed database that can be implemented into the industry.

5. What We Learned

   We learned how to effectively work as a team to create and alter a database. We all learned a lot more about data manipulation through our query commands. In addition, we learned how to properly implement changes through the UPDATE and ROLLBACK commands which are really important to know how to use since those are commands used in transactions. Transactions that would occur at a pharmacy. We also learned how to implement the DELETE command with care as it is important to specify *exactly* what needs to be deleted. Otherwise, one can wind up deleting an entire table and all its information which in most cases cannot be undone. Lastly, we learned how to implement some of the best practices when creating a database like not inserting repetitive information, using good naming standards, and having adequate documentation (in our conceptual design section). In the end, all of these factors resulted in us producing a database that is very easily navigable.

2. The Problem Statement

a. Overall goals of the system:

Our main purpose of the system is to be able to create a Database Management System (DBMS) for our pharmaceutical deliveries system. Within that goal, there are several objectives that we hope to accomplish:

- Establish an efficient database where records can be retrieved quickly
- Determine and document needs of customers
- Ensure that records are highly accurate without redundancy
- Analyze, interpret, and use data visualization in order to establish large data set trends to make appropriate business decisions

b. Context and importance of the system:

It is of the utmost importance that a pharmaceutical business has an efficient and factually accurate system. The system must be able to document any personnel or consumer who gives or receives a drug within the system. This includes entities such as:

- Customer information (Name, Address, Phone Number, etc)
- Drug information (Description, Manufacturer, Price)
- Doctor Information (Name, Address, Phone Number, License Number)
- Prescription
- Inventory

This information can be used to analyze and generate data visualizations to see trends. This will be beneficial in improving business operations, ensuring customer safety and keeping the information factually accurate.

c. Scope of the project:

IN-Scope:

In our database system, information that can be added includes customer, doctor, inventory, drug, and prescription.

OUT-Scope:

In our database system, information that includes medical insurance, returns, and payment methods like credit card or debit card will not be included in the operations of our database.

3. Requirements

a. Data Requirements

For each customer, DBMS will keep track of customer ID, name, address, phone number, and email.

For each doctor, DBMS will keep track of doctor ID, name, address, phone number, and license number.

For each drug, DBMS will keep track of drug ID, drug, description of the drug, manufacturer of the drug, and price of the drug.

For each inventory, DBMS will keep track of inventory ID (which is the inventory# corresponding to a drug), drug ID, quantity, and reorder level.

For each prescription, DBMS will keep track of prescription ID, customer ID, Doctor ID, and Drug ID. The customer ID will come from the Customer entity. The Drug ID will come from the Drug entity.

For each billing, DBMS will keep track of the billing ID, prescription ID, customer ID, total cost calculated by DBMS, and payment status. The prescription ID will come from the Prescription entity. The customer ID will come from the Customer entity.

For each time that profit is calculated, DBMS will calculate the revenue based on the billing and costs based on inventory and drug tables in a given time period.

Important reports that need to be generated include reorder level, payment status (incomplete), total revenue reports, etc.

b. Business Rules and Logic

(1) The prescription ID will be generated by the pharmacy once a customer provides a valid prescription from a doctor with the doctor ID referenced on each prescription. If it was the first purchase of the customer within our system, the customer information will be recorded.

(2) The details of the prescription (**prescription_id**, customer_id, doctor_id, drug_id) will be entered into the system of this pharmacy store and will be referenced by using prescription ID in billing.

(3) When a purchase is recorded, the billing will be added into the system.

(4) When a new shipment of a new drug arrives, a new drug ID will be generated and the information of the newly arrived drug will be updated in the drug table and will be referenced to in Inventory to reflect the quantity and reorder level.

(5) When shipment of an existing drug arrives, the inventory will be updated to reflect new quantity and reorder level.

(6) When a drug is sold, the inventory quantity is updated to reflect the new quantity.

(7) Profit in a time period is computed from billing and inventory. It will be computed each time profit reports are manually generated, or done automatically after a purchase is made.

(8) DBMS will be used many times in a day, so when a record is updated or changed, the update time must be recorded in relation to the record.

(9) There will be 2 user types: customer and pharmacy employees. They will have different access/execution privileges.

(10) Billing invoices can only be created by the pharmacy. Prescription can only be changed by the Doctor.

c. Other Assumptions

(1) We will assume DBMS will be used by a small pharmacy store in a real-world setting.

(2) DBMS runs on a client/server environment, running Windows Server as OS.

(3) The underlying DB system is Enterprise Oracle.

(4) The data of all transactions are entered into the system only after the transaction occurs.

(5) The shipping-related data will not be included within the DBMS, assuming the store does not have to pay for shipping and is not responsible for shipping.

d. Problem Solution

The problem is solved by reviewing the current practice of the system, researching the successful practices of distributed systems, and implementing the presented system. The outlined system will help to manage all sources of incoming and outgoing data, and will keep everything organized. It will also query the necessary data using automated processes, which is much easier than performing these tasks manually.

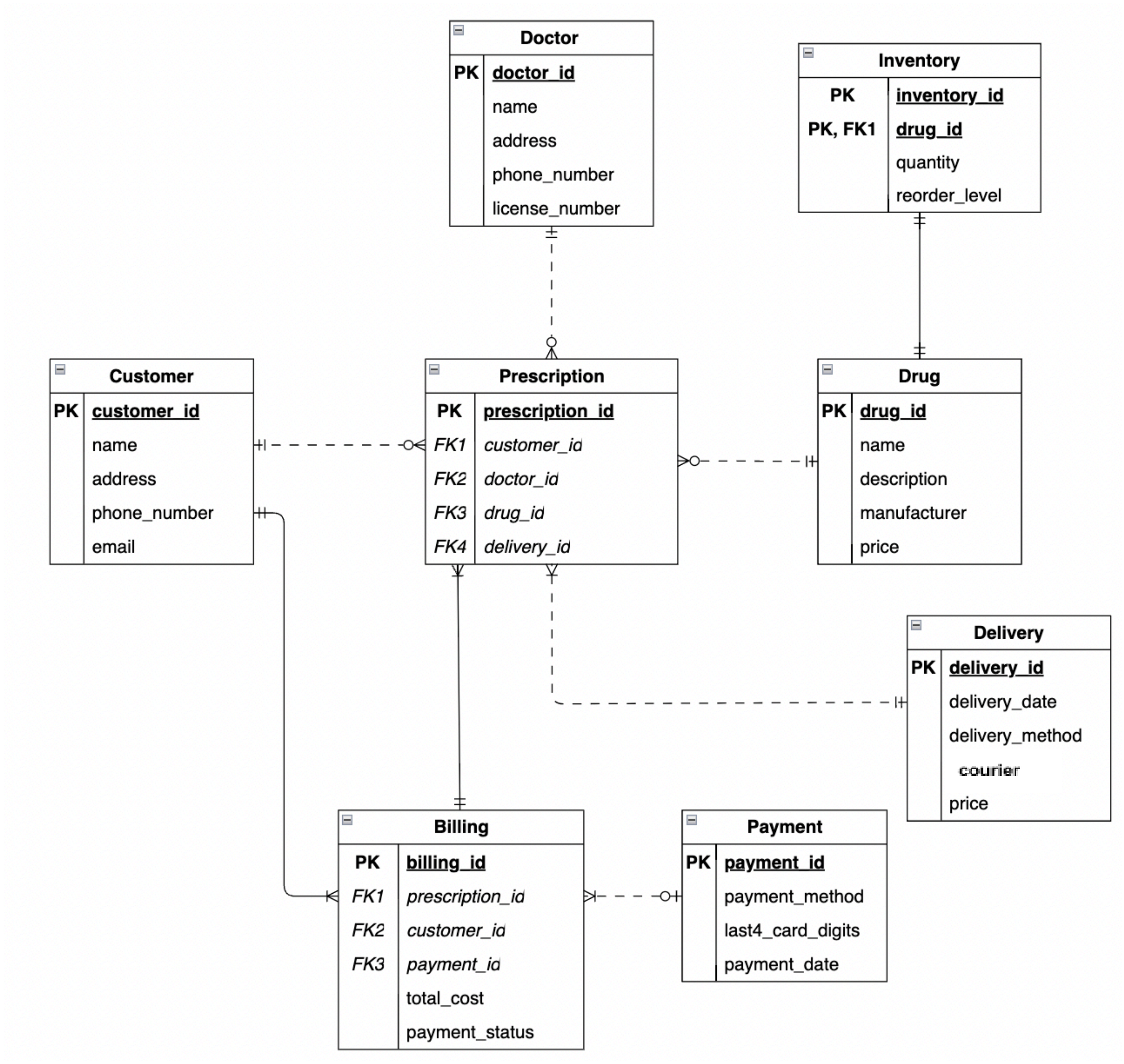
e. Data Acquisition

Data will be input manually by either the employee (for face-to-face transactions) or by the customer (for online transactions).

f. Hardware and Software

The system will be built and implemented on a local server that runs the Oracle 19c SQL system. No other software is required for the system to run at this time.

4. Conceptual Design



Our unanimously voted ERD described a database system for a pharmaceutical delivery server. This diagram consists of eight entities: customer, doctor, prescription, drug, inventory, billing, payment, and delivery.

The customer entity has a PK of customer_id and contains the information of the customer which includes name, phone_number, address, and email. Does not contain foreign keys.

The doctor entity references doctor_id as the PK, but does not contain a foreign key. It contains the name, phone_number, address, license_number of each doctor.

The prescription entity has a PK of prescription_id and contains four FK. First FK is customer_id which references Customer/ Second FK is drug_id which references Drug. The third FK is doctor_id and references Doctor. The last FK is delivery_id which references Delivery.

The drug entity has a PK named drug_id and contains information about the drug. It includes drug manufacturer, description of drug, name, and price. Drug entity does not contain foreign keys.

The inventory entity contains a PK of inventory_id and also contains a FK named drug_id which is referencing Drug. This entity contains the inventory stock of drugs and the new shipment which are named inventory and reorder_level respectively.

The billing entity contains a PK called billing_id which is a unique number identifier specific to each billing statement. This entity also contains three FKs. The first FK, called prescription_id, links each billing statement to the prescription that is being paid for. The second FK, customer_id, connects each billing statement to the customer it is addressed to. The last FK is called payment_id, and it links each bill to the method of payment being applied to it. The payment to be made by the customer is called total_cost, and the payment status of each bill (‘PAID’, ‘PENDING’, or ‘UNPAID’) is stored under payment_status.

The payment entity has a PK called payment_id, which is a unique number identifier to each payment made towards a bill. The credit card company the money is coming from, the last 4 digits of said card, and the date each payment is made, are stored in payment_method, lst4_card_digits, and payment_date respectively. There are no FKs.

The delivery entity has one PK called delivery_id, which stores a unique identifier for each delivery. There are no FKs. The date the delivery is completed, the shipping rate, the courier of the delivery, and the price for shipping, are stored in delivery_date, delivery_method, courier, and price, respectively.

Customer and Billing: A customer can have one or many billings. Billings can have only one customer.

Customer and Prescription: A customer can have one or many prescriptions. Prescriptions can have only one customer.

Customer and Billing: A customer can have one or many bills. Each bill can only have (be issued to) one customer.

Doctor and Prescription: A doctor can have (write) zero or many prescriptions. A prescription can have (be written by) only one doctor.

Drugs and Prescription: A drug can have zero or many prescriptions. Prescriptions can have only one drug.

Drugs and Inventory: A drug can only have one inventory. Inventory can contain exactly one drug.

Billing and Prescription: A bill can have (include) one or many prescriptions. Each prescription can have (be on) only one bill.

Billing and Payment: A bill can have one or many payments made. Each payment can have (go towards) one or many bills.

Delivery and Prescription: Each delivery can have (contain) one or many prescriptions. Each prescription can have (be contained in) only one delivery.

5. Relational Schema

- Relational schema (textual format; A list of tables with all the attributes, PK underlined and FKs specified using REFERENCES clause)

Customer (**customer_id**, name, address, phone_number, email)

Doctor (**doctor_id**, name, address, phone_number, license_number)

Prescription (**prescription_id**, customer_id, doctor_id, drug_id)
FOREIGN KEY customer_id REFERENCES Customer (customer_id)
FOREIGN KEY drug_id REFERENCES Drug (drug_id)
FOREIGN KEY doctor_id REFERENCES Doctor (doctor_id)
FOREIGN KEY delivery_id REFERENCES Delivery (delivery_id)

Drug (**drug_id**, name, description, manufacturer, price)

Inventory (**inventory_id**, drug_id, quantity, reorder_level)
FOREIGN KEY drug_id REFERENCES Drug (drug_id)

Billing (billing_id, prescription_id, customer_id, total_cost, payment_status)
FOREIGN KEY prescription_id REFERENCES Prescription (prescription_id)
FOREIGN KEY customer_id REFERENCES Customer (customer_id)
FOREIGN KEY payment_id REFERENCES Payment (payment_id)

Payment (payment_id, payment_method, last4_card_digits, payment_date)

Delivery (delivery_id, delivery_date, delivery_method, courier, price)

6. Data Dictionary

(For each table, document each attribute in terms of meaning, data type, domain, NOT NULL, PK, FK, and any other specific issue on attributes such as value range, enumerated values)

Customer

Column Name	Data Type	Description
customer_id	INT	Primary key identifier for a customer
name	VARCHAR(255)	Name of the customer
address	VARCHAR(255)	Address of the customer
phone_number	VARCHAR(20)	Phone number of the customer
email	VARCHAR(255)	Email address of the customer

Doctor

Column Name	Data Type	Description
doctor_id	INT	Primary key identifier for a doctor
name	VARCHAR(255)	Name of the doctor
address	VARCHAR(255)	Address of the doctor
phone_number	VARCHAR(20)	Phone number of the doctor
license_number	VARCHAR(255)	License number of the doctor

Drug

Column Name	Data Type	Description

drug_id	INT	Primary key identifier for a drug
name	VARCHAR(255)	Name of the drug
description	VARCHAR(255)	Description of the drug
manufacturer	VARCHAR(255)	Manufacturer of the drug
price	FLOAT	Price of the drug

Inventory

Column Name	Data Type	Description
inventory_id	INT	Primary key identifier for an inventory item

drug_id	INT	Foreign key referencing the drug_id column in the Drug table
quantity	INT	Quantity of the drug in the inventory
reorder_level	INT	Reorder level for the drug

Delivery

Column Name	Data Type	Description
delivery_id	INT	Primary key identifier for a delivery record
delivery_date	DATE	Date the delivery was made
delivery_method	VARCHAR(50)	Method of delivery
courier	VARCHAR(255)	Courier of delivery
price	FLOAT	Price of the delivery

Prescription

Column Name	Data Type	Description
prescription_id	INT	Primary key identifier for a prescription
customer_id	INT	Foreign key referencing the customer_id column in the Customer table
doctor_id	INT	Foreign key referencing the doctor_id column in the Doctor table
drug_id	INT	Foreign key referencing the drug_id column in the Drug table
delivery_id	INT	Foreign key referencing the delivery_id column in the Delivery table

Payment

Column Name	Data Type	Description
payment_id	INT	Primary key identifier for a payment record
payment_meth od	VARCHAR(50)	Description of what financial company the customer's payment is linked to
last4_card_digi ts	VARCHAR(4)	Last 4 digits of customer's payment method
payment_date	DATE	Date the payment was made by the customer

Billing

Column Name	Data Type	Description
billing_id	INT	Primary key identifier for a billing record

prescription_id	INT	Foreign key referencing the prescription_id column in the Prescription table
customer_id	INT	Foreign key referencing the customer_id column in the Customer table
payment_id	INT	Foreign key referencing the payment_id column in the Payment table
total_cost	FLOAT	Total cost of the prescription
payment_status	VARCHAR(50)	Status of payment for the prescription

SAMPLE DATA

Customer

customer_id	**name**	**address**	**phone_number**	**email**

1	John Doe	123 Main St., Anytown, USA	555-1234	johndoe@email.com
2	Jane Smith	456 Oak St., Anytown, USA	555-5678	janesmith@email.com
3	Bob Johnson	789 Pine St., Anytown, USA	555-9012	bobjohnson@email.com

Doctor

doctor_id	**name**	**address**	**phone_number**	**license_number**
1	Dr. John Smith	111 Main St., Anytown, USA	555-1111	12345
2	Dr. Jane Doe	222 Oak St., Anytown, USA	555-2222	67890

3	Dr. Bob Williams	333 Pine St., Anytown, USA	555-3333	24680

Prescription

prescription_id	**customer_id**	**doctor_id**	**drug_id**	**delivery_id**
1	1	1	1	1
2	2	2	2	2
3	3	3	3	3

Drug

drug_id	**name**	**description**	**manufacturer**	**price**

1	Aspirin	Pain reliever	Bayer	5.99
2	Lipitor	Cholesterol-lowering medication	Pfizer	12.99
3	Advil	Pain reliever	Johnson & Johnson	7.99

Inventory

inventory_id	**drug_id**	**quantity**	**reorder_level**
1	**1**	100	20
2	**2**	50	10
3	**3**	75	15

Billing

billing_id	prescription_id	customer_id	payment_id	total_cost	payment_status
1	1	1	1	10.99	PAID
2	2	2	2	22.99	PENDING
3	3	3	3	15.99	UNPAID

Payment

payment_id	payment_method	last4_card_digits	payment_date
1	VISA	1234	01-JAN-2023
2	AMEX	9012	02-FEB-2023

3	DISCOVER	6358	03-MAR-2023

Delivery

delivery_id	**delivery_date**	**delivery_method**	**courier**	**price**
1	2023-03-20	Express	DHL	2.99
2	2023-03-22	Standard	UPS	5.99
3	2023-03-24	Next Day	FedEx	3.99

7. SCHEMA and DATA

(Show the CREATE TABLE and INSERT commands. Each table must have at least 3 rows. Some tables may have more than 3 rows to make your database realistic. Each member should share some workloads for insertion. For example, if there are 8 tables for 3 members, each member can insert into two or three tables. That is, each member does not have to insert one row per table. Also, display all the data by performing SELECT * FROM *your_table_name*; for all tables in a readable format.)

Oracle Formatting Commands:

```
SET LINESIZE 500

COLUMN license_number FORMAT A10

COLUMN name FORMAT A10

COLUMN Description FORMAT A20

COLUMN Manufacturer FORMAT A10

COLUMN price FORMAT $9,999.99

COLUMN address FORMAT A20

COLUMN phone_number FORMAT A10

COLUMN email FORMAT A15

COLUMN total_cost FORMAT $9,999.99

COLUMN payment_status FORMAT A10

COLUMN delivery_date FORMAT A10

COLUMN delivery_method FORMAT A10
```

a. CREATE TABLE commands

```
CREATE TABLE Customer (
   customer_id INT PRIMARY KEY,
   name VARCHAR(255),
   address VARCHAR(255),
   phone_number VARCHAR(20),
   email VARCHAR(255)
);

CREATE TABLE Doctor (
   doctor_id INT PRIMARY KEY,
   name VARCHAR(255),
   address VARCHAR(255),
   phone_number VARCHAR(20),
   license_number VARCHAR(255)
);

CREATE TABLE Drug (
   drug_id INT PRIMARY KEY,
   name VARCHAR(255),
   description VARCHAR(255),
   manufacturer VARCHAR(255),
    price FLOAT
);

CREATE TABLE Inventory (
    inventory_id INT PRIMARY KEY,
    drug_id INT,
    quantity INT,
    reorder_level INT,
    FOREIGN KEY (drug_id) REFERENCES Drug(drug_id)
);

CREATE TABLE Delivery (
    delivery_id INT PRIMARY KEY,
    delivery_date DATE,
    delivery_method VARCHAR(50),
    courier VARCHAR(255),
    price FLOAT
);

CREATE TABLE Prescription (
    prescription_id INT PRIMARY KEY,
    customer_id INT,
    doctor_id INT,

    drug_id INT,

    delivery_id INT,

    FOREIGN KEY (customer_id) REFERENCES Customer(customer_id),

    FOREIGN KEY (doctor_id) REFERENCES Doctor(doctor_id),

    FOREIGN KEY (drug_id) REFERENCES Drug(drug_id),

    FOREIGN KEY (delivery_id) REFERENCES Delivery(delivery_id)

);


CREATE TABLE Payment (

    payment_id INT PRIMARY KEY,

    payment_method VARCHAR(50),

    last4_card_digits VARCHAR(4),

    payment_date DATE

);


CREATE TABLE Billing (

    billing_id INT PRIMARY KEY,

    prescription_id INT,

    customer_id INT,

    payment_id INT,

    total_cost FLOAT,
    payment_status VARCHAR(50),

    FOREIGN KEY (prescription_id) REFERENCES Prescription(prescription_id),

    FOREIGN KEY (customer_id) REFERENCES Customer(customer_id),

    FOREIGN KEY (payment_id) REFERENCES Payment(payment_id)

);
```

b. INSERT INTO Commands

```
INSERT INTO Customer (customer_id, name, address, phone_number, email) VALUES
(1, 'John Doe', '123 Main St., Anytown, USA', '555-1234', 'johndoe@email.com');

INSERT INTO Customer (customer_id, name, address, phone_number, email) VALUES
(2, 'Jane Smith', '456 Oak St., Anytown, USA', '555-5678', 'janesmith@email.com');

INSERT INTO Customer (customer_id, name, address, phone_number, email) VALUES
(3, 'Bob Johnson', '789 Pine St., Anytown, USA', '555-9012', 'bobjohnson@email.com');


INSERT INTO Doctor (doctor_id, name, address, phone_number, license_number)
VALUES (1, 'Dr. John Smith', '111 Main St., Anytown, USA', '555-1111', '12345');

INSERT INTO Doctor (doctor_id, name, address, phone_number, license_number)
VALUES (2, 'Dr. Jane Doe', '222 Oak St., Anytown, USA', '555-2222', '67890');

INSERT INTO Doctor (doctor_id, name, address, phone_number, license_number)
VALUES (3, 'Dr. Bob Williams', '333 Pine St., Anytown, USA', '555-3333', '24680');


INSERT INTO Drug (drug_id, name, description, manufacturer, price) VALUES (1,
'Aspirin', 'Pain reliever', 'Bayer', 5.99);

INSERT INTO Drug (drug_id, name, description, manufacturer, price) VALUES (2,
'Lipitor', 'Cholesterol-lowering medication', 'Pfizer', 12.99);
```

```
INSERT INTO Drug (drug_id, name, description, manufacturer, price) VALUES (3,
'Advil', 'Pain reliever', 'Johnson & Johnson', 7.99);


INSERT INTO Inventory (inventory_id, drug_id, quantity, reorder_level) VALUES (1, 1,
100, 20);

INSERT INTO Inventory (inventory_id, drug_id, quantity, reorder_level) VALUES (2, 2,
50, 10);

INSERT INTO Inventory (inventory_id, drug_id, quantity, reorder_level) VALUES(3, 3,
75, 15);


INSERT INTO Delivery (delivery_id, delivery_date, delivery_method, courier, price)
VALUES (1, TO_DATE('20-MAR-2023', 'YYYY-MM-DD'), 'Express', 'DHL', 2.99);

INSERT INTO Delivery (delivery_id, delivery_date, delivery_method, courier, price)
VALUES (2, TO_DATE('22-MAR-2023', 'YYYY-MM-DD'), 'Standard', 'UPS', 5.99);

INSERT INTO Delivery (delivery_id, delivery_date, delivery_method, courier, price)
VALUES (3, TO_DATE('24-MAR-2023', 'YYYY-MM-DD'), 'Next Day', 'FedEx', 3.99);


INSERT INTO Prescription (prescription_id, customer_id, doctor_id, drug_id,
delivery_id) VALUES (1, 1, 1, 1, 1);

INSERT INTO Prescription (prescription_id, customer_id, doctor_id, drug_id,
delivery_id) VALUES (2, 2, 2, 2, 2);

INSERT INTO Prescription (prescription_id, customer_id, doctor_id, drug_id,
delivery_id) VALUES (3, 3, 3, 3, 3);


INSERT INTO Payment (payment_id, payment_method, last4_card_digits,
payment_date) VALUES (1, ‘VISA’, ‘1234’, ‘01-JAN-2023’);
```

```
INSERT INTO Payment (payment_id, payment_method, last4_card_digits,
payment_date) VALUES (2, ‘AMEX’, ‘9012’, ‘02-FEB-2023’);
```

```
INSERT INTO Payment (payment_id, payment_method, last4_card_digits,
payment_date) VALUES (3, ‘DISCOVER’, ‘6358’, ‘03-MAR-2023’);
```

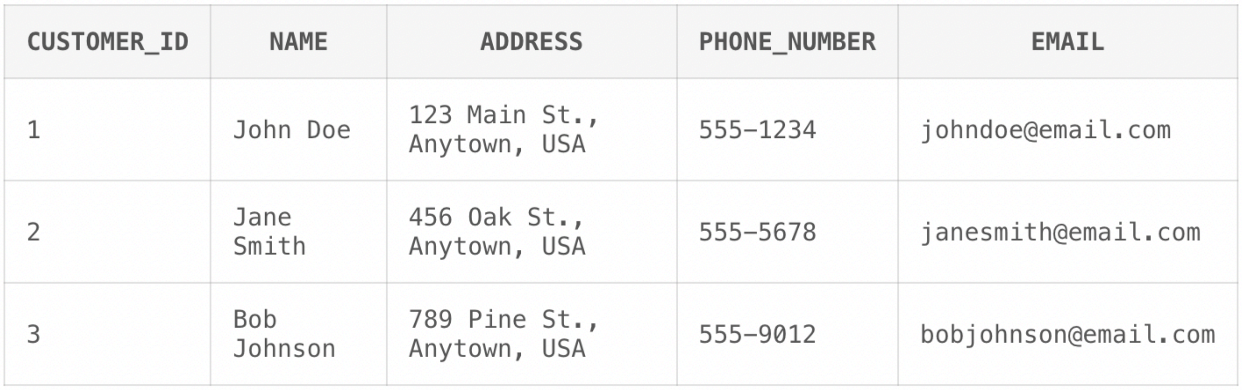
```
INSERT INTO Billing (billing_id, prescription_id, customer_id, payment_id, total_cost,
payment_status) VALUES (1, 1, 1, 1, 10.99, 'PAID');
```

```
INSERT INTO Billing (billing_id, prescription_id, customer_id, payment_id, total_cost,
payment_status) VALUES (2, 2, 2, 2, 22.99, 'PENDING');
```

```
INSERT INTO Billing (billing_id, prescription_id, customer_id, payment_id, total_cost,
payment_status) VALUES (3, 3, 3, 3, 15.99, 'UNPAID');
```

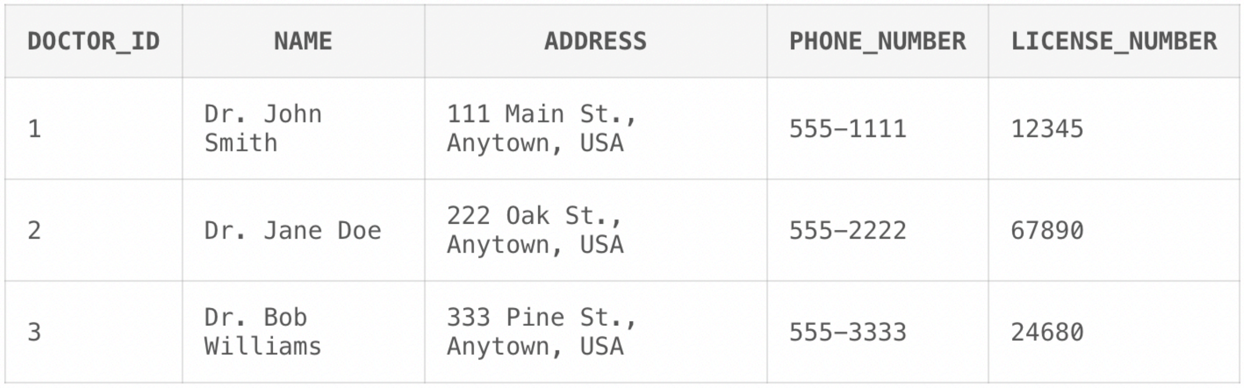
c. Snipped output from SELECT * FROM *your_table_name*; for all tables

SELECT * FROM Customer



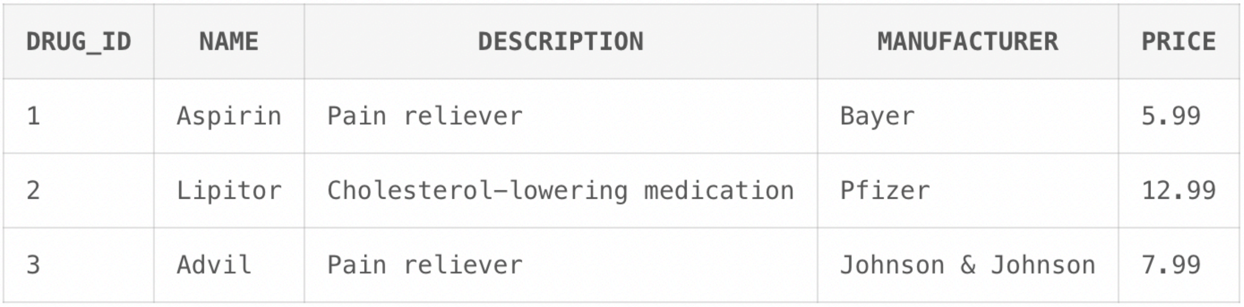
CUSTOMER_ID	NAME	ADDRESS	PHONE_NUMBER	EMAIL
1	John Doe	123 Main St., Anytown, USA	555-1234	johndoe@email.com
2	Jane Smith	456 Oak St., Anytown, USA	555-5678	janesmith@email.com
3	Bob Johnson	789 Pine St., Anytown, USA	555-9012	bobjohnson@email.com

SELECT * FROM Doctor



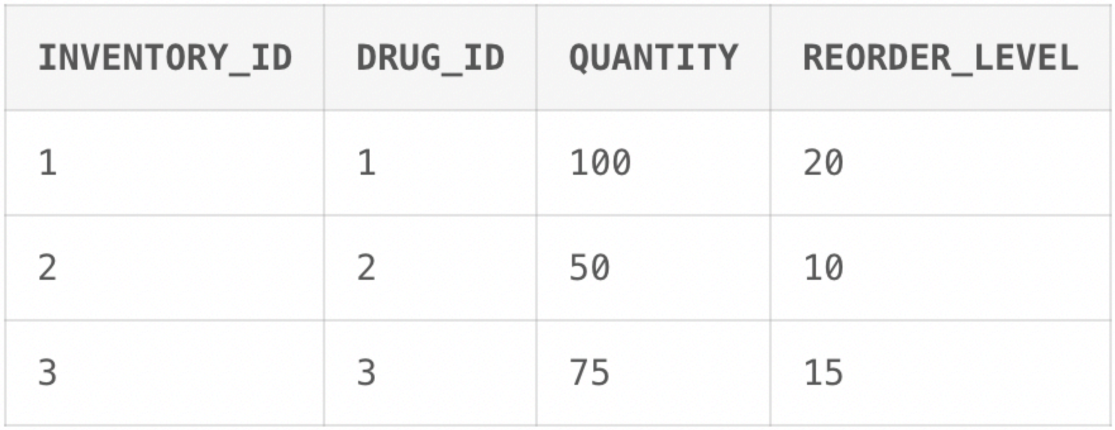
DOCTOR_ID	NAME	ADDRESS	PHONE_NUMBER	LICENSE_NUMBER
1	Dr. John Smith	111 Main St., Anytown, USA	555-1111	12345
2	Dr. Jane Doe	222 Oak St., Anytown, USA	555-2222	67890
3	Dr. Bob Williams	333 Pine St., Anytown, USA	555-3333	24680

SELECT * FROM Drug



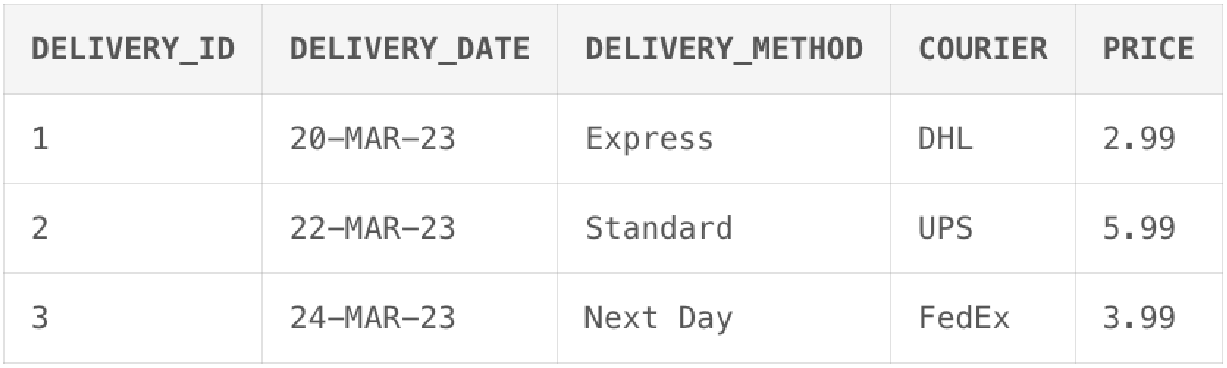
DRUG_ID	NAME	DESCRIPTION	MANUFACTURER	PRICE
1	Aspirin	Pain reliever	Bayer	5.99
2	Lipitor	Cholesterol-lowering medication	Pfizer	12.99
3	Advil	Pain reliever	Johnson & Johnson	7.99

SELECT * FROM Inventory



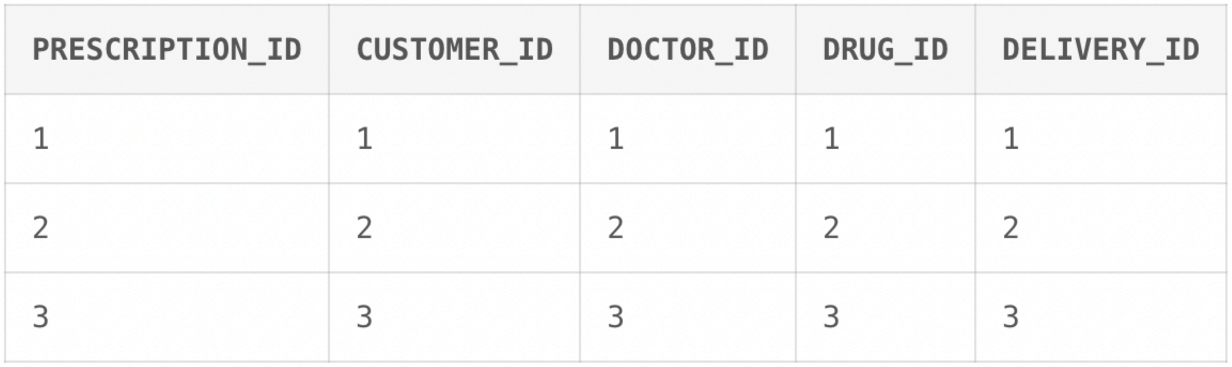
INVENTORY_ID	DRUG_ID	QUANTITY	REORDER_LEVEL
1	1	100	20
2	2	50	10
3	3	75	15

SELECT * FROM Delivery



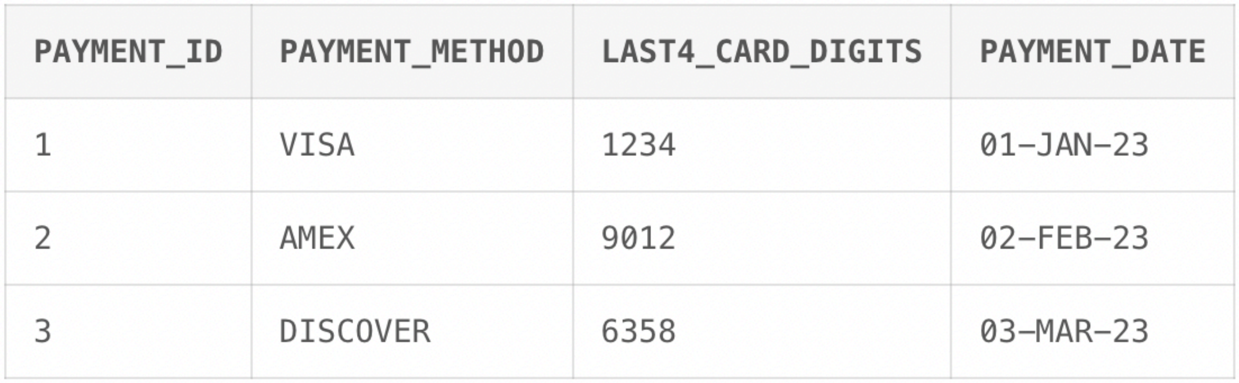
DELIVERY_ID	DELIVERY_DATE	DELIVERY_METHOD	COURIER	PRICE
1	20-MAR-23	Express	DHL	2.99
2	22-MAR-23	Standard	UPS	5.99
3	24-MAR-23	Next Day	FedEx	3.99

SELECT * FROM Prescription



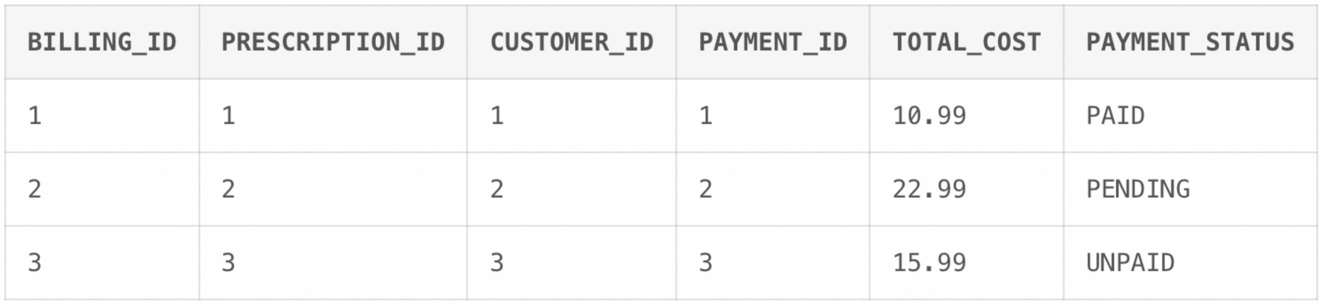
PRESCRIPTION_ID	CUSTOMER_ID	DOCTOR_ID	DRUG_ID	DELIVERY_ID
1	1	1	1	1
2	2	2	2	2
3	3	3	3	3

SELECT * FROM Payment



PAYMENT_ID	PAYMENT_METHOD	LAST4_CARD_DIGITS	PAYMENT_DATE
1	VISA	1234	01-JAN-23
2	AMEX	9012	02-FEB-23
3	DISCOVER	6358	03-MAR-23

SELECT * FROM Billing



BILLING_ID	PRESCRIPTION_ID	CUSTOMER_ID	PAYMENT_ID	TOTAL_COST	PAYMENT_STATUS
1	1	1	1	10.99	PAID
2	2	2	2	22.99	PENDING
3	3	3	3	15.99	UNPAID

8. Data Queries

(Show at least three examples of SQL queries per member. All the queries must include at least one join, an aggregate function, or a join and aggregate functions. Write down the member's name at the beginning of the queries. Show the query in English, SQL command, the output from the Oracle (**Attach snipped screenshots of the Oracle output**). Some of you may use views and create indexes for the queries. Any query with no joins will take some points off.)

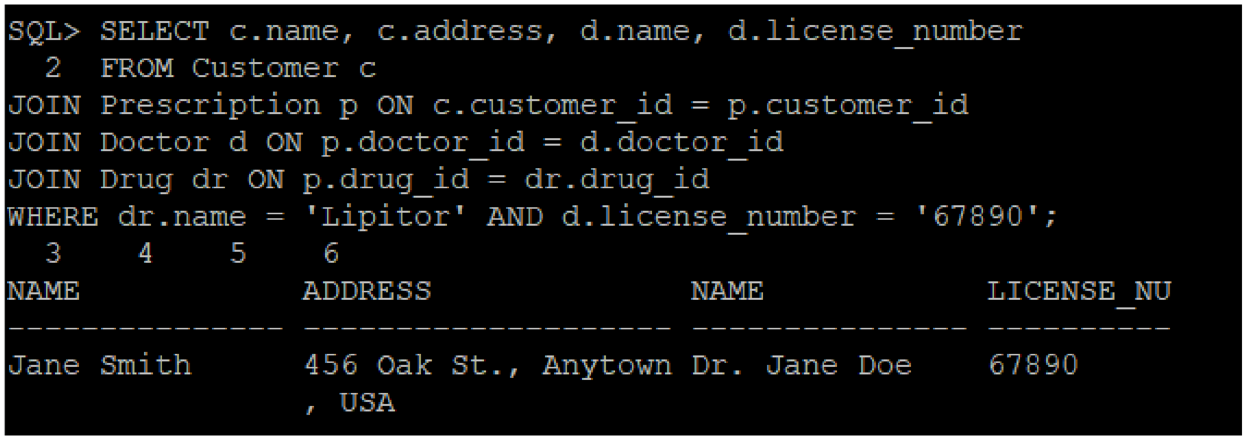
a. Queries by George Papadopoulos

Query 1

English Meaning:
SELECT query with a multi-table join: Retrieve the names and addresses of all customers who have purchased the drug Lipitor from the Doctor with license number 67890.

SQL:
SELECT c.name, c.address, d.name, d.license_number
FROM Customer c
JOIN Prescription p ON c.customer_id = p.customer_id
JOIN Doctor d ON p.doctor_id = d.doctor_id
JOIN Drug dr ON p.drug_id = dr.drug_id
WHERE dr.name = 'Lipitor' AND d.license_number = '67890';

snipped output, #rows


```
SQL> SELECT c.name, c.address, d.name, d.license_number
  2  FROM Customer c
JOIN Prescription p ON c.customer_id = p.customer_id
JOIN Doctor d ON p.doctor_id = d.doctor_id
JOIN Drug dr ON p.drug_id = dr.drug_id
WHERE dr.name = 'Lipitor' AND d.license_number = '67890';
  3    4    5    6
NAME            ADDRESS              NAME            LICENSE_NU
--------------- -------------------- --------------- ----------
Jane Smith      456 Oak St., Anytown Dr. Jane Doe    67890
                , USA
```

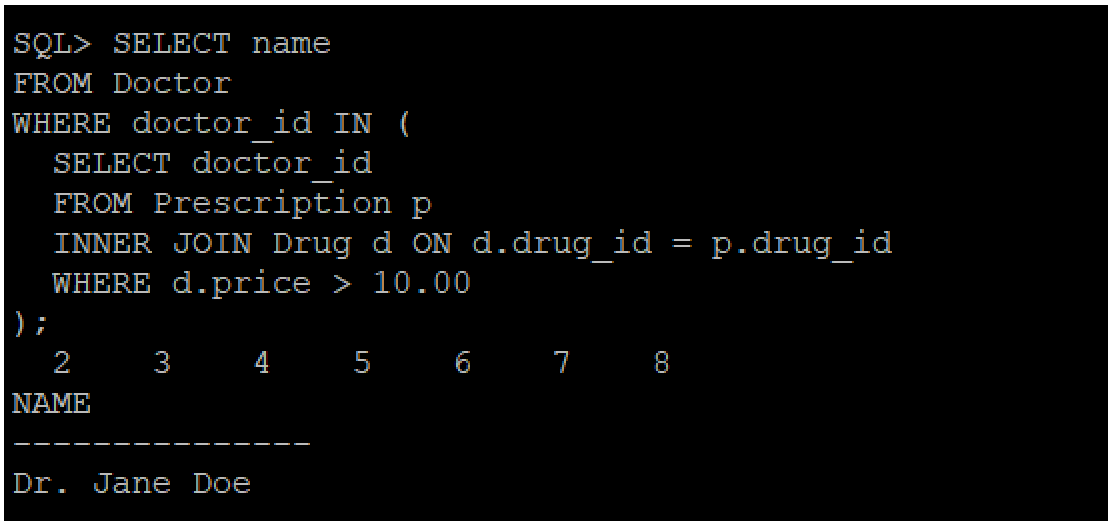
Query 2

English Meaning:

SELECT statement with subquery: List the names of all doctors who have prescribed at least one drug with a price greater than $10.00.

SQL:
```
SELECT name
FROM Doctor
WHERE doctor_id IN (
  SELECT doctor_id
  FROM Prescription p
  INNER JOIN Drug d ON d.drug_id = p.drug_id
  WHERE d.price > 10.00
);
```

snipped output, #rows



```
SQL> SELECT name
FROM Doctor
WHERE doctor_id IN (
  SELECT doctor_id
  FROM Prescription p
  INNER JOIN Drug d ON d.drug_id = p.drug_id
  WHERE d.price > 10.00
);
  2    3    4    5    6    7    8
NAME
---------------
Dr. Jane Doe
```

Query 3

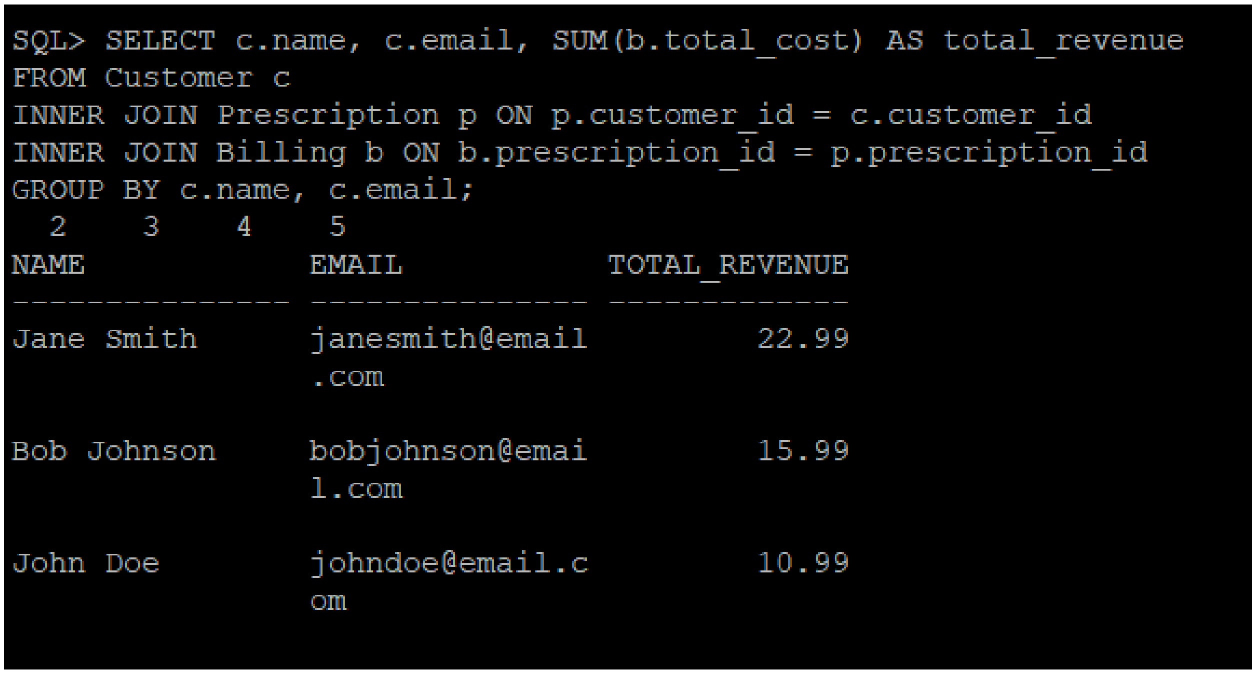
English Meaning:

SELECT statement with aggregate function: Find the total revenue generated by each customer, along with their name and email address.

SQL:
```
SELECT c.name, c.email, SUM(b.total_cost) AS total_revenue
FROM Customer c
INNER JOIN Prescription p ON p.customer_id = c.customer_id
```

INNER JOIN Billing b ON b.prescription_id = p.prescription_id
GROUP BY c.name, c.email;

snipped output, #rows



```
SQL> SELECT c.name, c.email, SUM(b.total_cost) AS total_revenue
FROM Customer c
INNER JOIN Prescription p ON p.customer_id = c.customer_id
INNER JOIN Billing b ON b.prescription_id = p.prescription_id
GROUP BY c.name, c.email;
  2    3    4    5
NAME            EMAIL           TOTAL_REVENUE
--------------- --------------- -------------
Jane Smith      janesmith@email         22.99
                .com

Bob Johnson     bobjohnson@emai         15.99
                l.com

John Doe        johndoe@email.c         10.99
                om
```

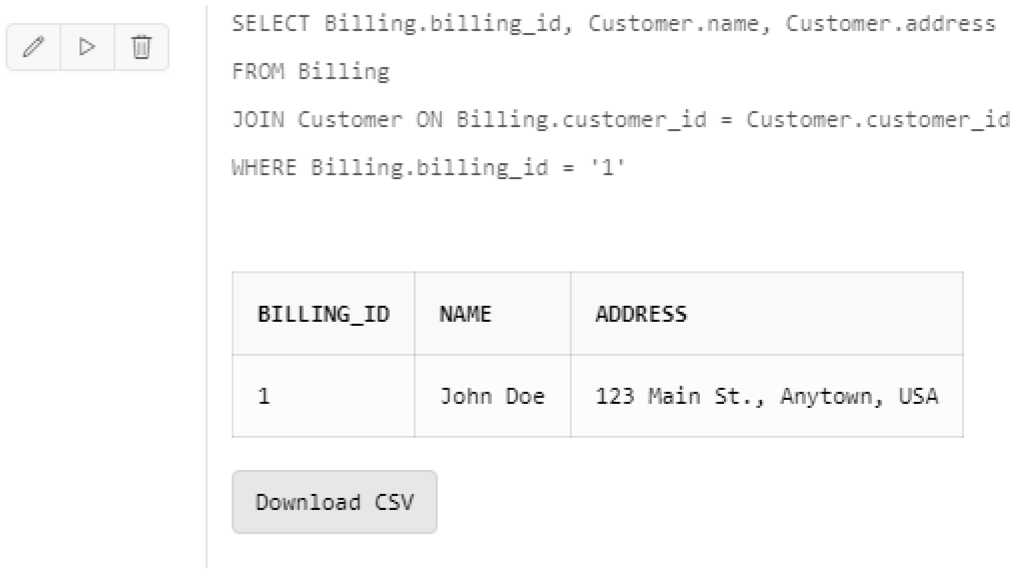
b. Queries by Michael Le

Query 1

English Meaning:
Use a SELECT query in conjunction with a JOIN clause that retrieves the information of a billing ID with the customer's name and address.

SQL:
SELECT Billing.billing_id, Customer.name, Customer.address
FROM Billing
JOIN Customer ON Billing.customer_id = Customer.customer_id
WHERE Billing.billing_id = '1';

snipped output, #rows



```
SELECT Billing.billing_id, Customer.name, Customer.address
FROM Billing
JOIN Customer ON Billing.customer_id = Customer.customer_id
WHERE Billing.billing_id = '1'
```

BILLING_ID	NAME	ADDRESS
1	John Doe	123 Main St., Anytown, USA

Query 2

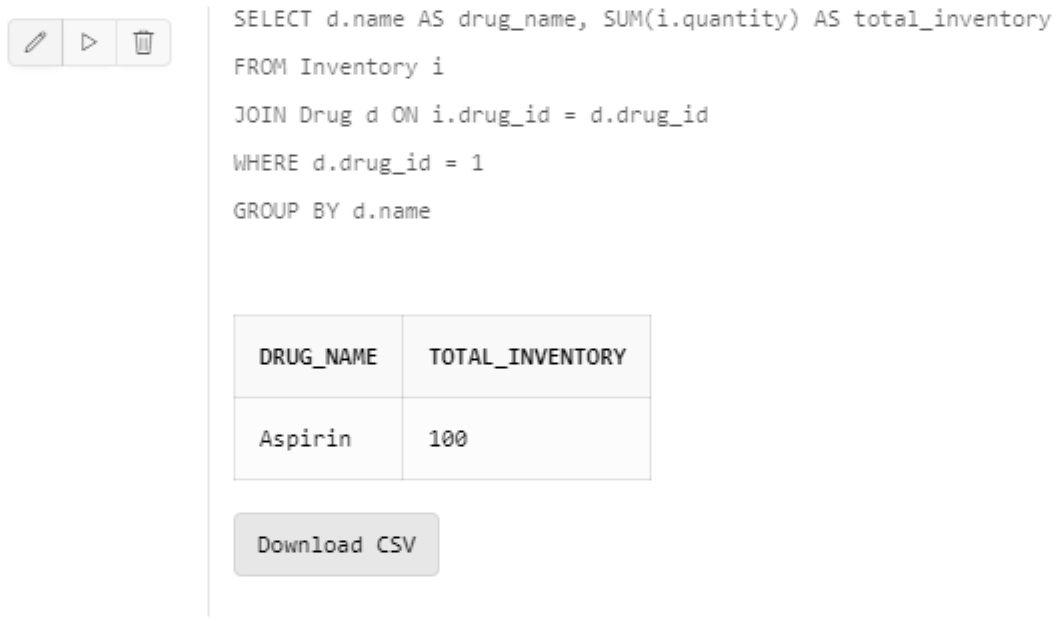
English Meaning:

Use a SELECT query in conjunction with an aggregate that retrieves the information of how much inventory for that drug given drug ID.

SQL:

```
SELECT d.name AS drug_name, SUM(i.quantity) AS total_inventory
FROM Inventory i
JOIN Drug d ON i.drug_id = d.drug_id
WHERE d.drug_id = 1
GROUP BY d.name;
```

snipped output, #rows


```
SELECT d.name AS drug_name, SUM(i.quantity) AS total_inventory
FROM Inventory i
JOIN Drug d ON i.drug_id = d.drug_id
WHERE d.drug_id = 1
GROUP BY d.name
```

DRUG_NAME	TOTAL_INVENTORY
Aspirin	100

<u>Query 3</u>

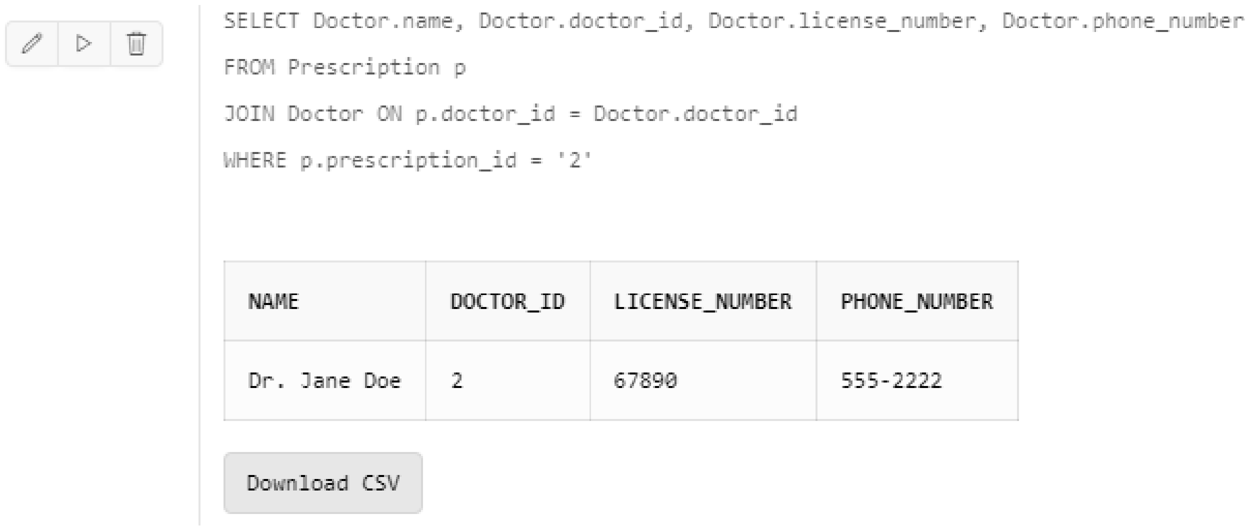
English Meaning:

Use a SELECT query in conjunction with an aggregate that retrieves the doctor name, doctor id, doctor license number and doctor phone number given prescription id.

SQL:

```
SELECT Doctor.name, Doctor.doctor_id, Doctor.license_number, Doctor.phone_number
FROM Prescription p
JOIN Doctor ON p.doctor_id = Doctor.doctor_id
WHERE p.prescription_id = '2';
```

snipped output, #rows



```
SELECT Doctor.name, Doctor.doctor_id, Doctor.license_number, Doctor.phone_number
FROM Prescription p
JOIN Doctor ON p.doctor_id = Doctor.doctor_id
WHERE p.prescription_id = '2'
```

NAME	DOCTOR_ID	LICENSE_NUMBER	PHONE_NUMBER
Dr. Jane Doe	2	67890	555-2222

c. Queries by Kelsey Chong

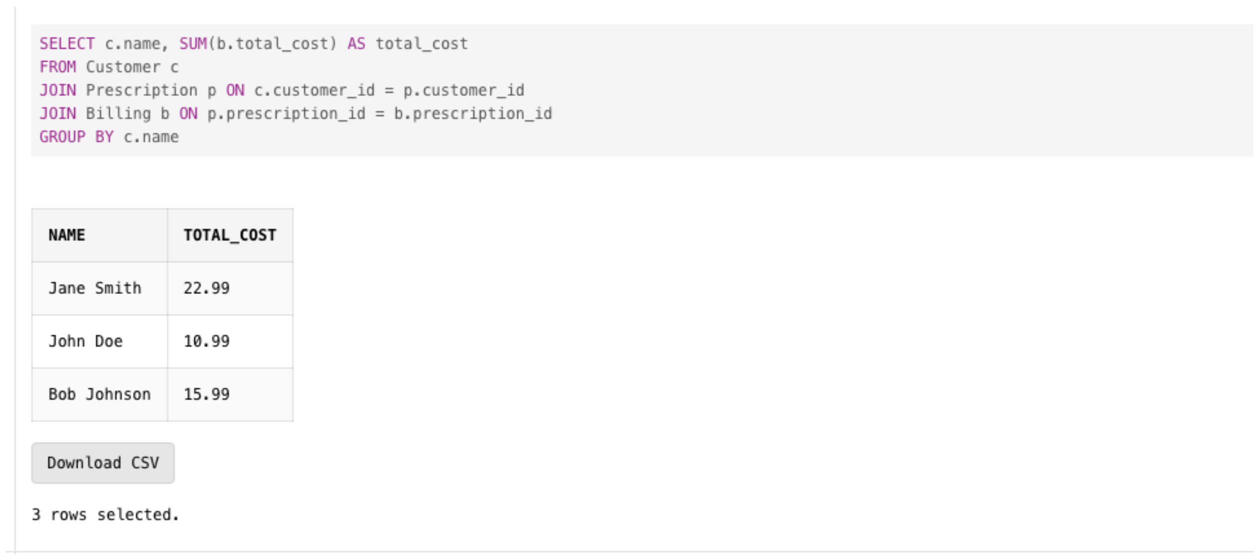
<u>Query 1</u>

English Meaning:
SELECT query with a JOIN and SUM aggregate function and GROUP BY clause: Select the name of each customer and the total cost of all their prescriptions, from the Customer table joined with the Prescription and Billing tables based on customer_id and prescription_id, and grouped by customer name.

This query finds the total cost of prescriptions for each customer.

SQL:
```
SELECT c.name, SUM(b.total_cost) AS total_cost
FROM Customer c
JOIN Prescription p ON c.customer_id = p.customer_id
JOIN Billing b ON p.prescription_id = b.prescription_id
GROUP BY c.name;
```

snipped output, #rows



```
SELECT c.name, SUM(b.total_cost) AS total_cost
FROM Customer c
JOIN Prescription p ON c.customer_id = p.customer_id
JOIN Billing b ON p.prescription_id = b.prescription_id
GROUP BY c.name
```

NAME	TOTAL_COST
Jane Smith	22.99
John Doe	10.99
Bob Johnson	15.99

Download CSV

3 rows selected.

Query 2

English Meaning:

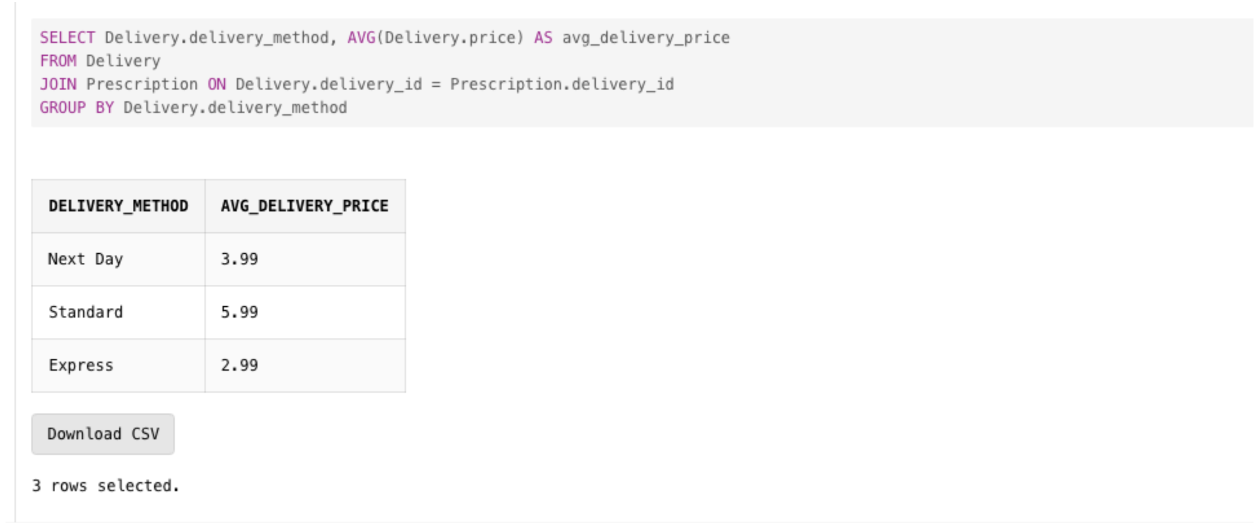
SELECT query with a JOIN, and AVG aggregate function, and GROUP BY clause: Select all delivery methods and the average price of each delivery from the Delivery table, joined with the Prescription table on the delivery_id column. Group the results by delivery method.

This query finds the average delivery price for each delivery method.

SQL:

SELECT Delivery.delivery_method, AVG(Delivery.price) AS avg_delivery_price
FROM Delivery
JOIN Prescription ON Delivery.delivery_id = Prescription.delivery_id
GROUP BY Delivery.delivery_method;

snipped output, #rows



```
SELECT Delivery.delivery_method, AVG(Delivery.price) AS avg_delivery_price
FROM Delivery
JOIN Prescription ON Delivery.delivery_id = Prescription.delivery_id
GROUP BY Delivery.delivery_method
```

DELIVERY_METHOD	AVG_DELIVERY_PRICE
Next Day	3.99
Standard	5.99
Express	2.99

```
3 rows selected.
```

Query 3

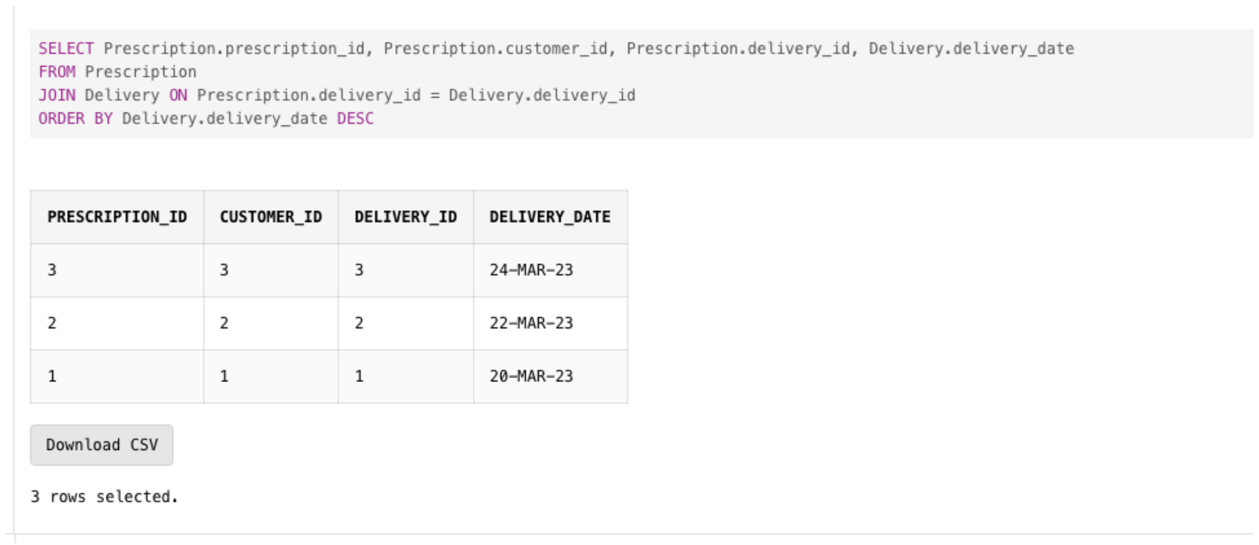
English Meaning:

SELECT query that uses JOIN function and ORDER BY clause: Select all prescriptions sorted with the latest delivery date at the top with the associated customer_id, delivery_id, and prescription_id.

SQL:

SELECT Prescription.prescription_id, Prescription.customer_id, Prescription.delivery_id, Delivery.delivery_date
FROM Prescription
JOIN Delivery ON Prescription.delivery_id = Delivery.delivery_id
ORDER BY Delivery.delivery_date DESC;

snipped output, #rows


```
SELECT Prescription.prescription_id, Prescription.customer_id, Prescription.delivery_id, Delivery.delivery_date
FROM Prescription
JOIN Delivery ON Prescription.delivery_id = Delivery.delivery_id
ORDER BY Delivery.delivery_date DESC
```

PRESCRIPTION_ID	CUSTOMER_ID	DELIVERY_ID	DELIVERY_DATE
3	3	3	24-MAR-23
2	2	2	22-MAR-23
1	1	1	20-MAR-23

```
3 rows selected.
```

d. Queries by Brooke Law

Query 1

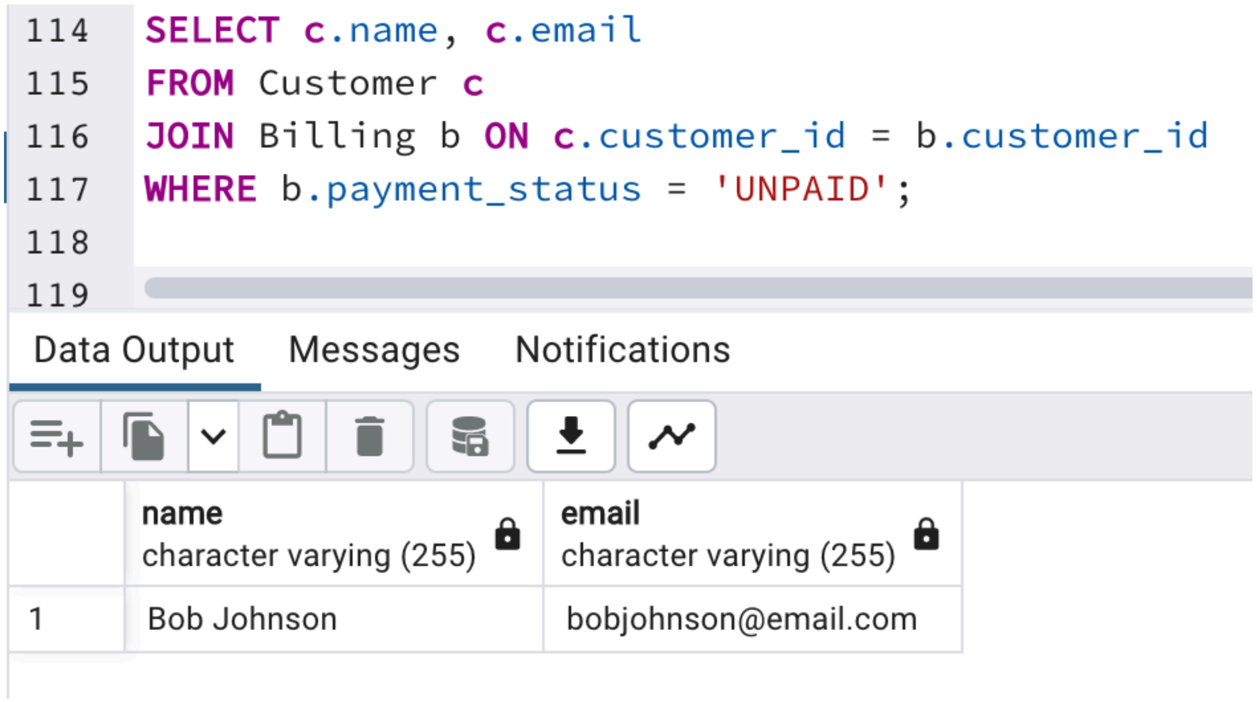
English Meaning:

SELECT query using a JOIN clause: Select all customer name & email for all customers who have an unpaid billing status.

SQL:

```
SELECT c.name, c.email
FROM Customer c
JOIN Billing b ON c.customer_id = b.customer_id
WHERE b.payment_status = 'UNPAID';
```

snipped output, #rows



```
SELECT c.name, c.email
FROM Customer c
JOIN Billing b ON c.customer_id = b.customer_id
WHERE b.payment_status = 'UNPAID';
```

	name character varying (255)	email character varying (255)
1	Bob Johnson	bobjohnson@email.com

<u>Query 2</u>

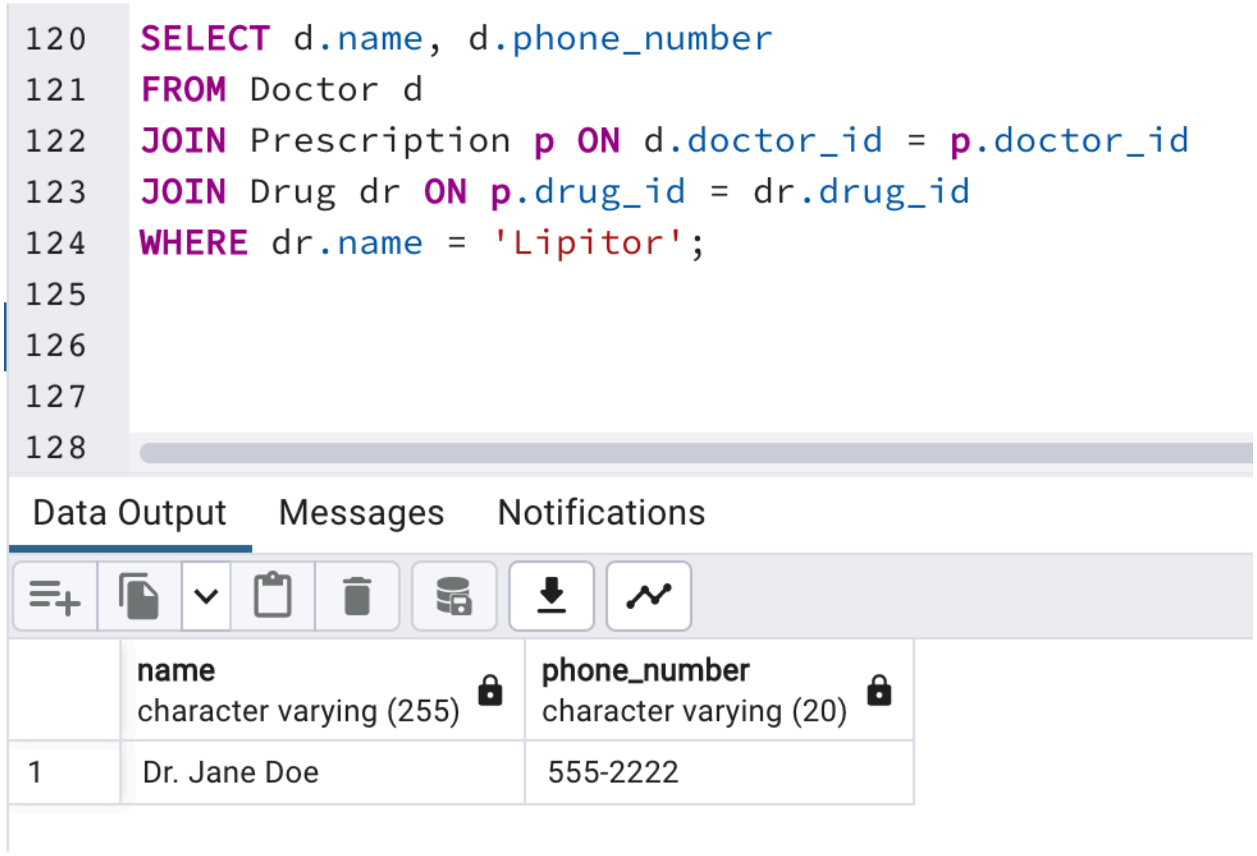
English Meaning:

SELECT query that uses two JOIN clauses: Select doctor name & phone number for all doctors who have prescribed the drug Lipitor.

SQL:

```
SELECT d.name, d.phone_number
FROM Doctor d
JOIN Prescription p ON d.doctor_id = p.doctor_id
JOIN Drug dr ON p.drug_id = dr.drug_id
WHERE dr.name = 'Lipitor';
```

snipped output, #rows



```
SELECT d.name, d.phone_number
FROM Doctor d
JOIN Prescription p ON d.doctor_id = p.doctor_id
JOIN Drug dr ON p.drug_id = dr.drug_id
WHERE dr.name = 'Lipitor';
```

	name character varying (255)	phone_number character varying (20)
1	Dr. Jane Doe	555-2222

Query 3

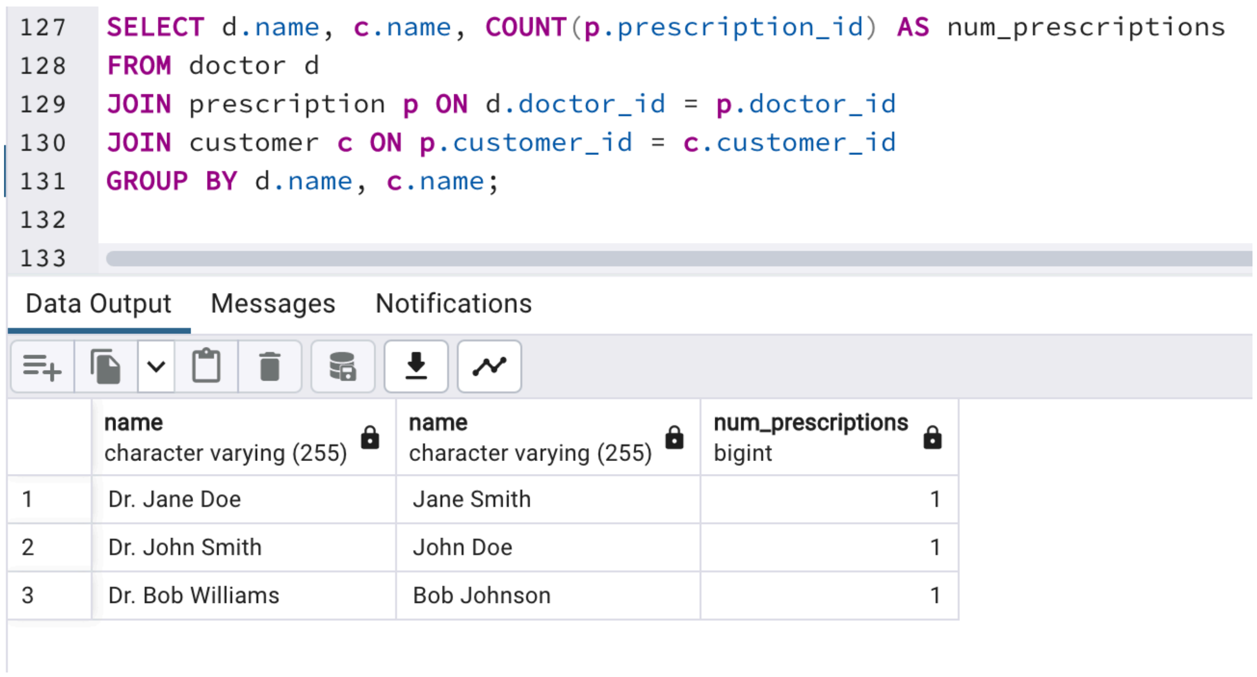
English Meaning:

SELECT query that uses aggregate function COUNT and JOIN command: Select doctor name & customer name to find how many prescriptions each doctor has prescribed to each customer.

SQL:

SELECT d.name, c.name, COUNT(p.prescription_id) AS num_prescriptions
FROM doctor d
JOIN prescription p ON d.doctor_id = p.doctor_id
JOIN customer c ON p.customer_id = c.customer_id
GROUP BY d.name, c.name;

snipped output, #rows



```
SELECT d.name, c.name, COUNT(p.prescription_id) AS num_prescriptions
FROM doctor d
JOIN prescription p ON d.doctor_id = p.doctor_id
JOIN customer c ON p.customer_id = c.customer_id
GROUP BY d.name, c.name;
```

	name character varying (255)	name character varying (255)	num_prescriptions bigint
1	Dr. Jane Doe	Jane Smith	1
2	Dr. John Smith	John Doe	1
3	Dr. Bob Williams	Bob Johnson	1

e. Queries by Amanda Collins

Query 1

English Meaning:
SELECT query using two JOIN clauses: Select the customer ID, drug ID, and drug name belonging to customers that have successfully paid their bill

SQL:
SELECT p.customer_id, p.drug_id, d.name
FROM Prescription p
JOIN Billing b ON p.customer_id = b.customer_id
JOIN Drug d ON p.drug_id = d.drug_id
WHERE b.payment_status = 'PAID'

snipped output, #rows

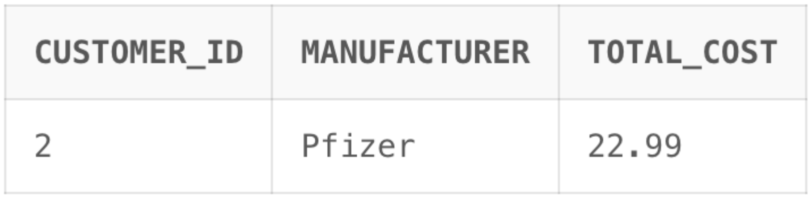
CUSTOMER_ID	DRUG_ID	NAME
1	1	Aspirin

Query 2

English Meaning:
SELECT query using the AVG aggregate function: Select the customer_ID, drug manufacturer, and cost of the customer's bill where the total cost of the bill is greater than the average of all customers' total costs.

SQL:
```
SELECT c.customer_id, d.manufacturer, b.total_cost
FROM Drug d
JOIN Prescription p ON d.drug_id = p.drug_id
JOIN Customer c ON p.customer_id = c.customer_id
JOIN Billing b ON c.customer_id = b.customer_id
WHERE b.total_cost > (SELECT AVG(total_cost)
                      FROM Billing)
```

snipped output, #rows



CUSTOMER_ID	MANUFACTURER	TOTAL_COST
2	Pfizer	22.99

<u>Query 3</u>

English Meaning:
SELECT query using aggregation of data via '+' operator: Find the cost including delivery paid by each customer, and display their customer ID.

SQL:
```
SELECT c.customer_id, b.total_cost + d.price AS Full_Cost
FROM Customer c
JOIN Billing b ON c.customer_id = b.customer_id
JOIN Prescription p ON c.customer_id = p.customer_id
JOIN Delivery d ON p.delivery_id = d.delivery_id
```

snipped output, #rows

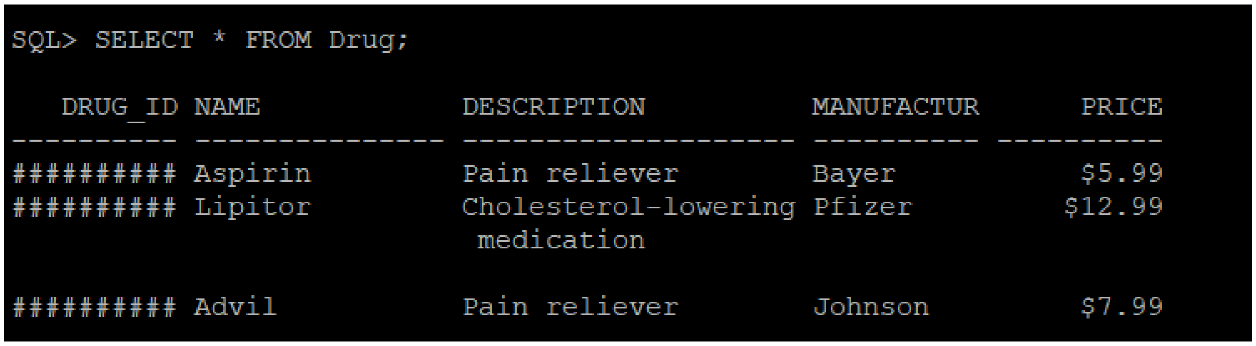
CUSTOMER_ID	FULL_COST
1	13.98
2	28.98
3	19.98

9. DATA MANIPULATION

(Each member must include at least one example of UPDATE and DELETE to any table in your database in the context of your requirements. For DML commands, use SET AUTOCOMMIT OFF so that you can rollback. For each deletion and update command, you must display the data before and after the command to confirm the correctness of the command. That is, I want you to practice insertion/deletion/update in your projects.

a. DML by George Papadopoulos

i. Data before the UPDATE command



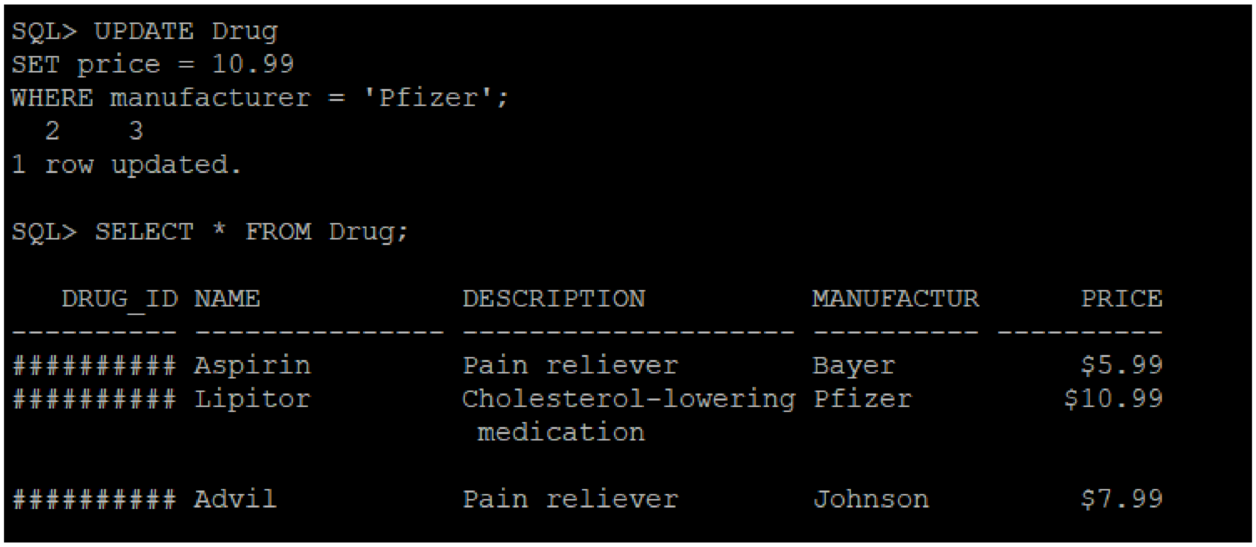
```
SQL> SELECT * FROM Drug;

   DRUG_ID NAME            DESCRIPTION          MANUFACTUR      PRICE
---------- --------------- -------------------- ---------- ----------
########## Aspirin         Pain reliever        Bayer           $5.99
########## Lipitor         Cholesterol-lowering Pfizer         $12.99
                            medication

########## Advil           Pain reliever        Johnson         $7.99
```

ii. UPDATE command:

UPDATE command with a WHERE condition on a numeric field: Update the price of all drugs made by Pfizer to $10.99.

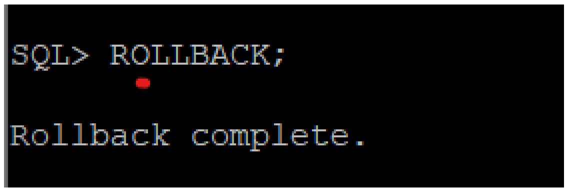
```
UPDATE Drug

SET price = 10.99

WHERE manufacturer = 'Pfizer';
```

iii. Data after the UPDATE command:

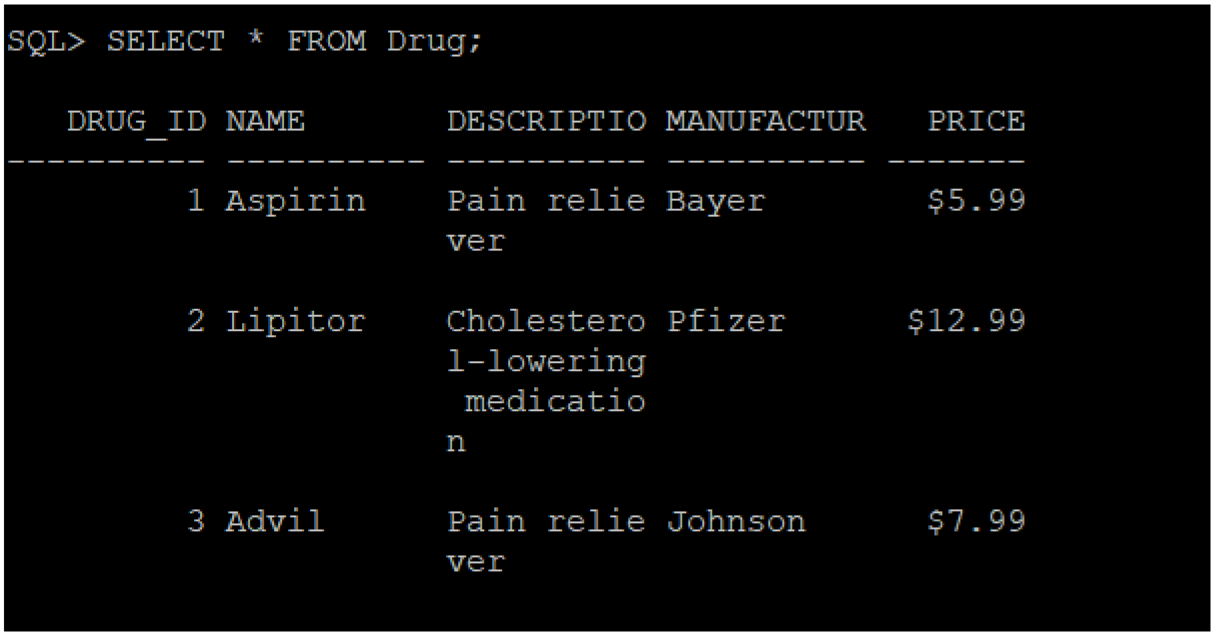


```
SQL> UPDATE Drug
SET price = 10.99
WHERE manufacturer = 'Pfizer';
  2    3
1 row updated.

SQL> SELECT * FROM Drug;

   DRUG_ID NAME            DESCRIPTION          MANUFACTUR      PRICE
---------- --------------- -------------------- ---------- ----------
########## Aspirin         Pain reliever        Bayer           $5.99
########## Lipitor         Cholesterol-lowering Pfizer         $10.99
                            medication

########## Advil           Pain reliever        Johnson         $7.99
```

iv. ROLLBACK

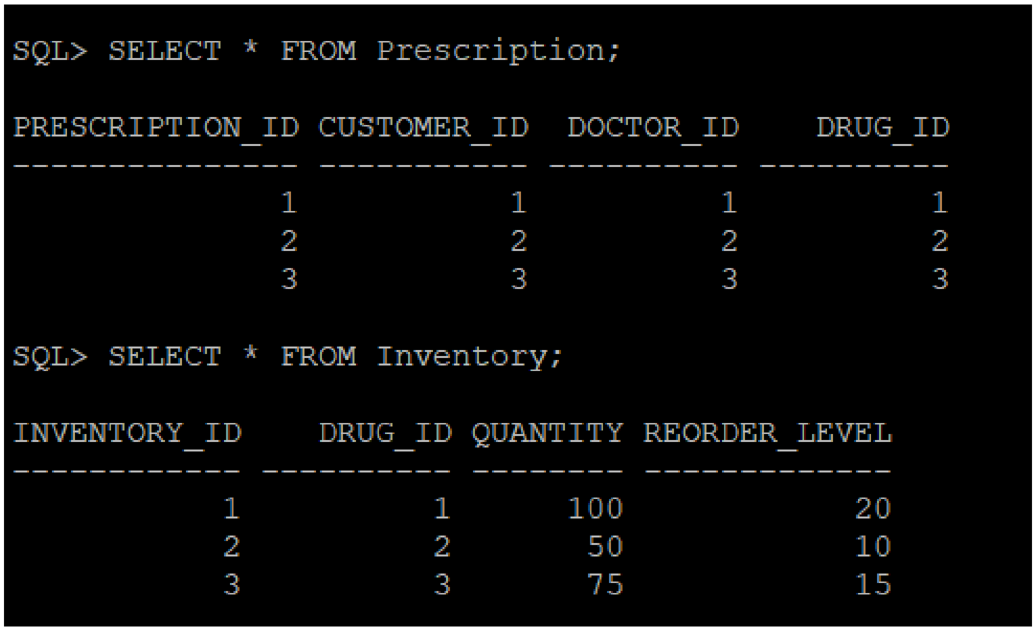


```
SQL> ROLLBACK;

Rollback complete.
```



```
SQL> SELECT * FROM Drug;

   DRUG_ID NAME       DESCRIPTIO MANUFACTUR   PRICE
---------- ---------- ---------- ---------- -------
         1 Aspirin    Pain relie Bayer        $5.99
                      ver

         2 Lipitor    Cholestero Pfizer      $12.99
                      l-lowering
                       medicatio
                      n

         3 Advil      Pain relie Johnson      $7.99
                      ver
```

v. Data before the DELETE command



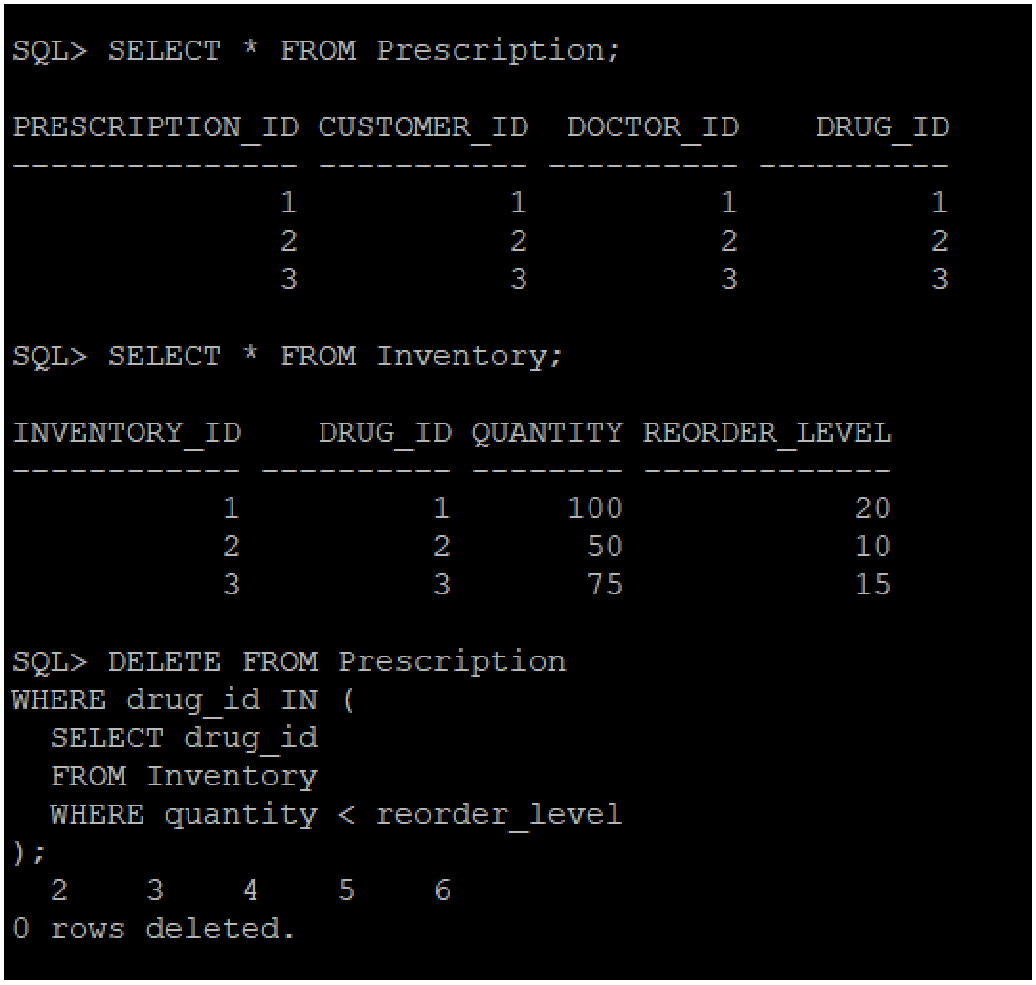
```
SQL> SELECT * FROM Prescription;

PRESCRIPTION_ID CUSTOMER_ID  DOCTOR_ID    DRUG_ID
--------------- ----------- ---------- ----------
              1           1          1          1
              2           2          2          2
              3           3          3          3

SQL> SELECT * FROM Inventory;

INVENTORY_ID    DRUG_ID QUANTITY REORDER_LEVEL
------------ ---------- -------- -------------
           1          1      100            20
           2          2       50            10
           3          3       75            15
```

vi. DELETE command

DELETE command with subquery: Delete all prescriptions for drugs that are below the reorder level in inventory.

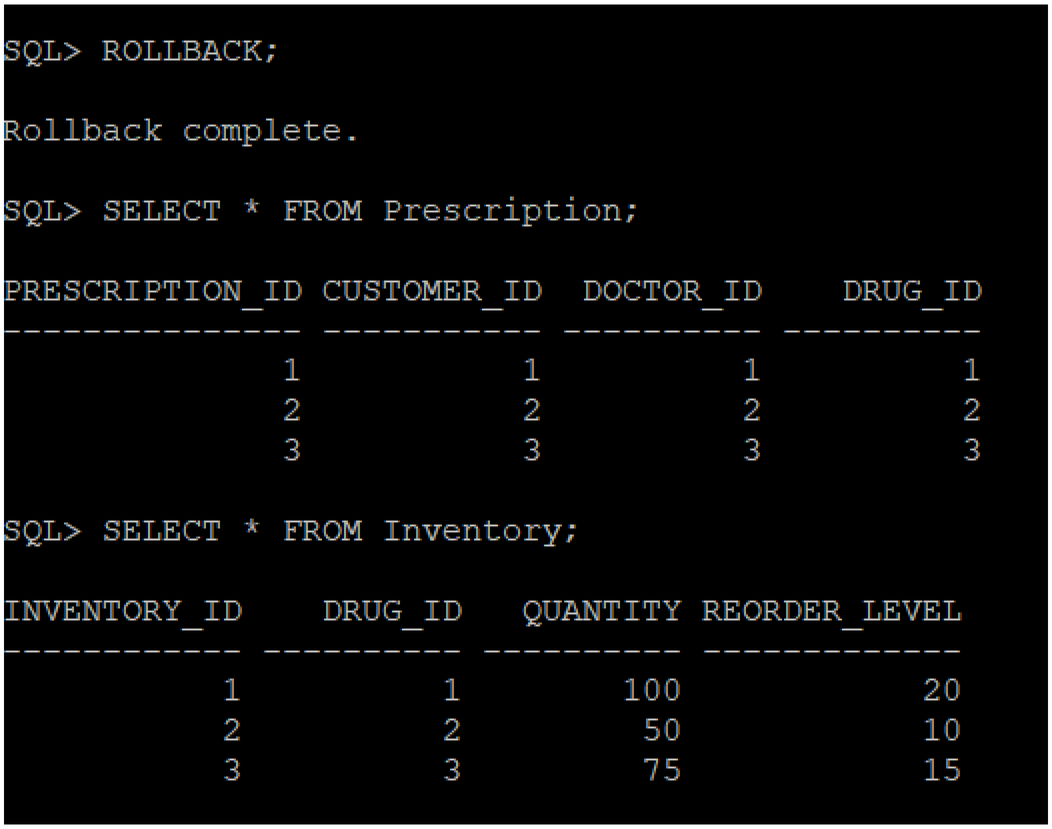
```
DELETE FROM Prescription

WHERE drug_id IN (

  SELECT drug_id

  FROM Inventory

  WHERE quantity < reorder_level

);
```

vii. Data after the DELETE command



```
SQL> SELECT * FROM Prescription;

PRESCRIPTION_ID CUSTOMER_ID  DOCTOR_ID    DRUG_ID
--------------- ----------- ---------- ----------
              1           1          1          1
              2           2          2          2
              3           3          3          3

SQL> SELECT * FROM Inventory;

INVENTORY_ID    DRUG_ID QUANTITY REORDER_LEVEL
------------ ---------- -------- -------------
           1          1      100            20
           2          2       50            10
           3          3       75            15

SQL> DELETE FROM Prescription
WHERE drug_id IN (
  SELECT drug_id
  FROM Inventory
  WHERE quantity < reorder_level
);
  2    3    4    5    6
0 rows deleted.
```

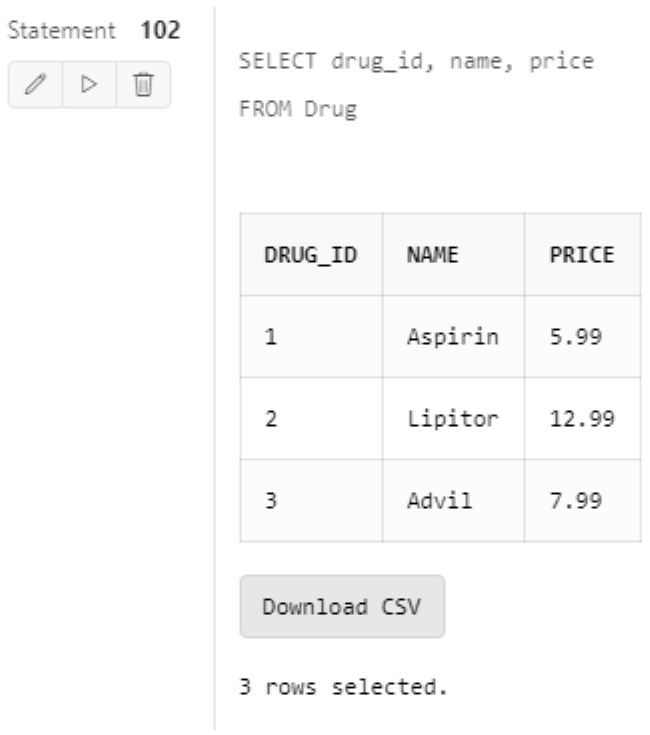
Rollback



```
SQL> ROLLBACK;

Rollback complete.

SQL> SELECT * FROM Prescription;

PRESCRIPTION_ID CUSTOMER_ID  DOCTOR_ID    DRUG_ID
--------------- ----------- ---------- ----------
              1           1          1          1
              2           2          2          2
              3           3          3          3

SQL> SELECT * FROM Inventory;

INVENTORY_ID    DRUG_ID   QUANTITY REORDER_LEVEL
------------ ---------- ---------- -------------
           1          1        100            20
           2          2         50            10
           3          3         75            15
```

b. DML by Michael Le

i. Data before the UPDATE command:


Statement 102

```
SELECT drug_id, name, price
FROM Drug
```

DRUG_ID	NAME	PRICE
1	Aspirin	5.99
2	Lipitor	12.99
3	Advil	7.99

```
3 rows selected.
```

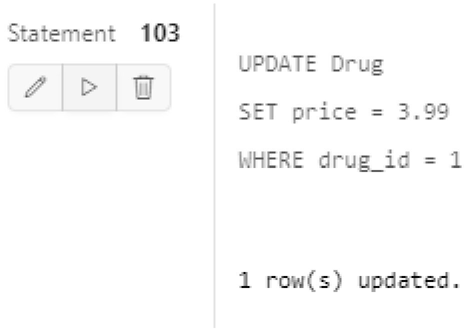
ii. UPDATE command:

Update Command to change the drug price of Aspirin with drug_id = 1 from 5.99 to 3.99. *The changes in the demand for the drug, Aspirin, have led to a decrease in the price.*

UPDATE Drug

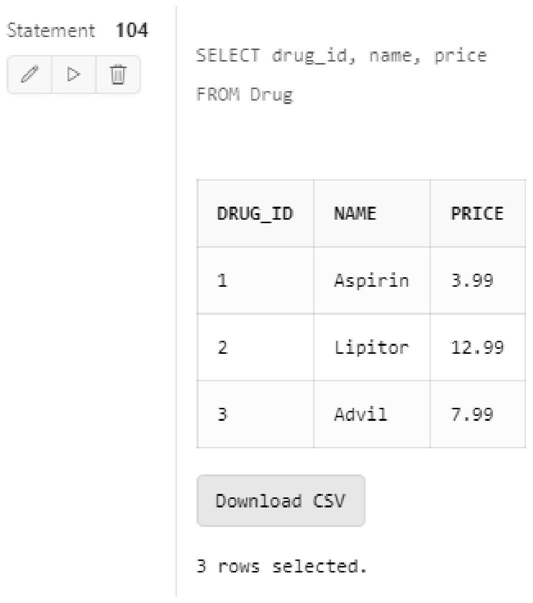
SET price = 3.99

WHERE drug_id = 1


Statement 103

```
UPDATE Drug
SET price = 3.99
WHERE drug_id = 1

1 row(s) updated.
```

iii. Data after the UPDATE command:


Statement 104

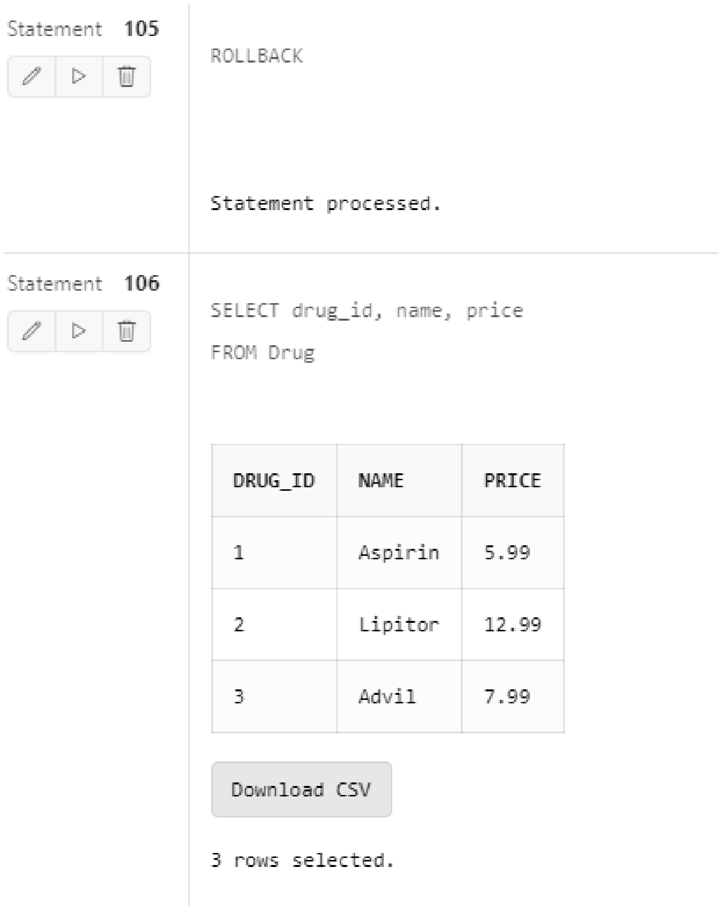
```
SELECT drug_id, name, price
FROM Drug
```

DRUG_ID	NAME	PRICE
1	Aspirin	3.99
2	Lipitor	12.99
3	Advil	7.99

Download CSV

```
3 rows selected.
```

<u>iv. ROLLBACK</u>


Statement 105

```
ROLLBACK
```

```
Statement processed.
```

Statement 106

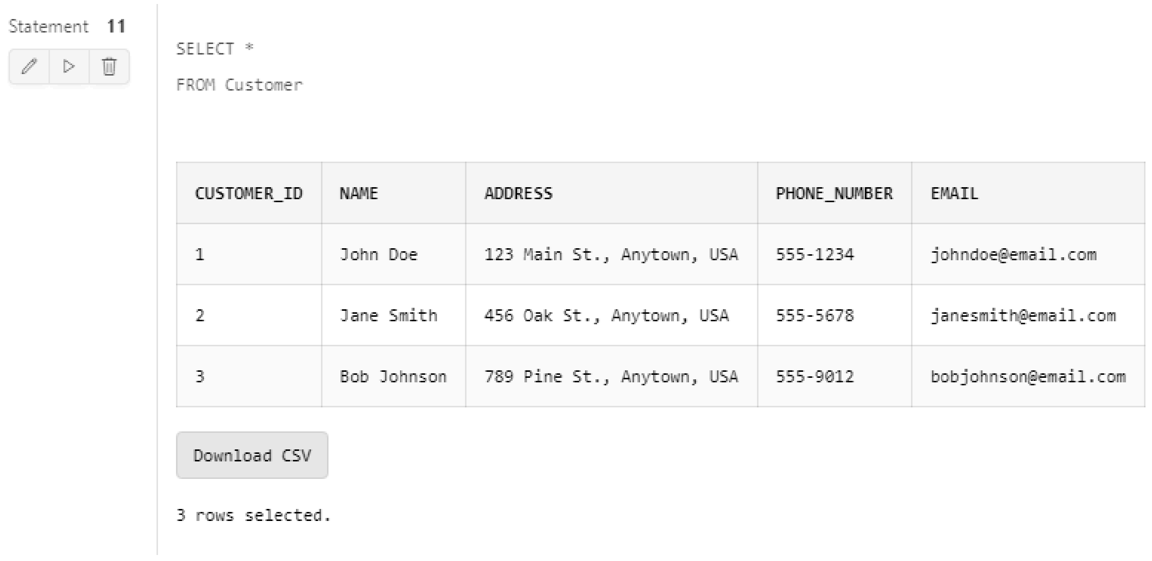
```
SELECT drug_id, name, price
FROM Drug
```

DRUG_ID	NAME	PRICE
1	Aspirin	5.99
2	Lipitor	12.99
3	Advil	7.99

Download CSV

```
3 rows selected.
```

v. Data before the DELETE command


Statement 11

```
SELECT *
FROM Customer
```

CUSTOMER_ID	NAME	ADDRESS	PHONE_NUMBER	EMAIL
1	John Doe	123 Main St., Anytown, USA	555-1234	johndoe@email.com
2	Jane Smith	456 Oak St., Anytown, USA	555-5678	janesmith@email.com
3	Bob Johnson	789 Pine St., Anytown, USA	555-9012	bobjohnson@email.com

```
3 rows selected.
```

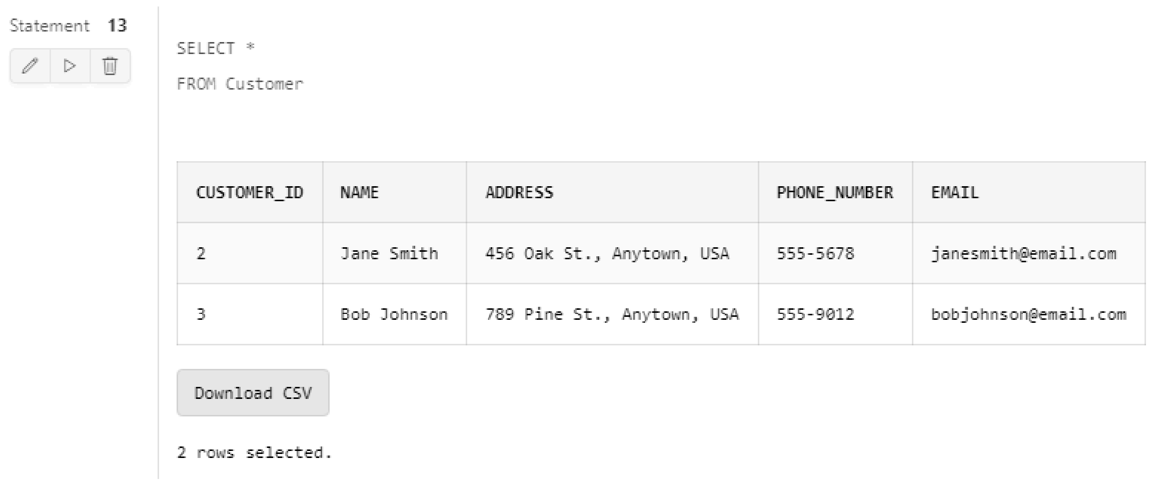
vi. DELETE command

DELETE FROM Customer

WHERE customer_id = 1 AND name = 'John Doe'

Delete where customer_id = 1 and name 'John Doe'. *The pharmacy is required by law or to delete certain types of records after a specified period of time has passed, such as expired prescription records or those related to controlled substances.*

vii. Data after the DELETE command



Statement 13

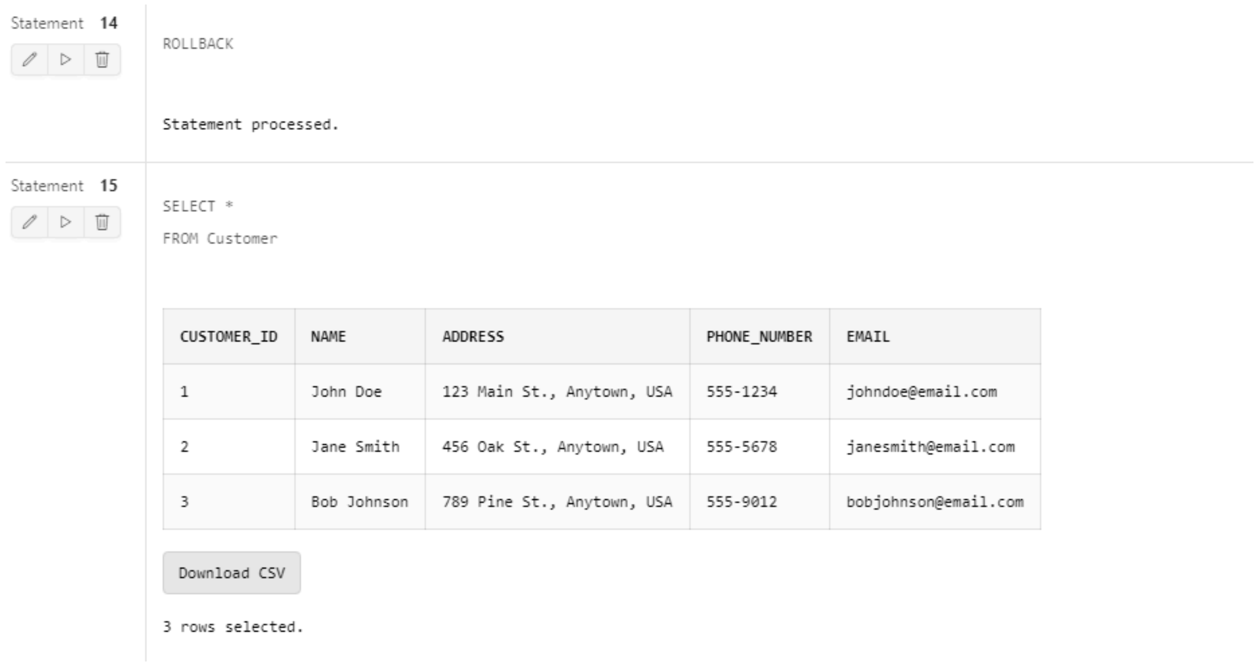
```
SELECT *
FROM Customer
```

CUSTOMER_ID	NAME	ADDRESS	PHONE_NUMBER	EMAIL
2	Jane Smith	456 Oak St., Anytown, USA	555-5678	janesmith@email.com
3	Bob Johnson	789 Pine St., Anytown, USA	555-9012	bobjohnson@email.com

Download CSV

```
2 rows selected.
```

viii. ROLLBACK



Statement 14

```
ROLLBACK
```

```
Statement processed.
```

Statement 15

```
SELECT *
FROM Customer
```

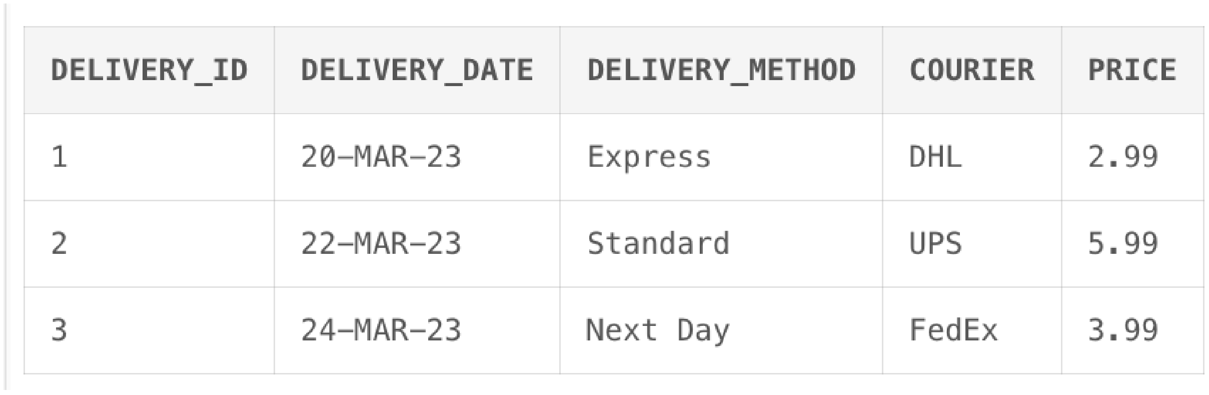
CUSTOMER_ID	NAME	ADDRESS	PHONE_NUMBER	EMAIL
1	John Doe	123 Main St., Anytown, USA	555-1234	johndoe@email.com
2	Jane Smith	456 Oak St., Anytown, USA	555-5678	janesmith@email.com
3	Bob Johnson	789 Pine St., Anytown, USA	555-9012	bobjohnson@email.com

Download CSV

```
3 rows selected.
```

c. DML by Kelsey Chong

i. Data before the UPDATE command



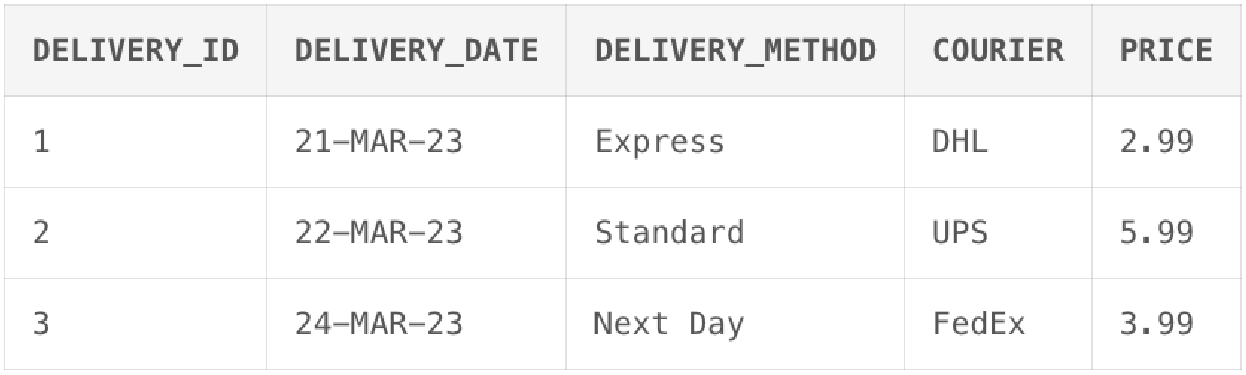
DELIVERY_ID	DELIVERY_DATE	DELIVERY_METHOD	COURIER	PRICE
1	20-MAR-23	Express	DHL	2.99
2	22-MAR-23	Standard	UPS	5.99
3	24-MAR-23	Next Day	FedEx	3.99

ii. UPDATE command

UPDATE delivery_date to 21-MAR-23 WHERE delivery_id is 1. *There was a 1-day delivery delay for delivery_id=1 therefore the delivery_date has to be updated.*

```
UPDATE Delivery
SET delivery_date = TO_DATE('2023-03-21', 'YYYY-MM-DD')
WHERE delivery_id = 1;
```

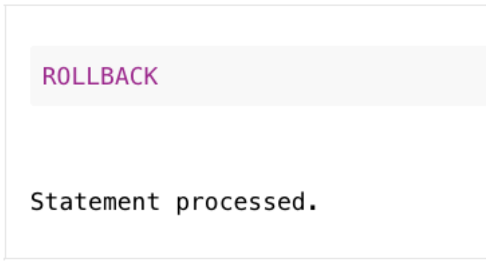
iii. Data after the UPDATE command



DELIVERY_ID	DELIVERY_DATE	DELIVERY_METHOD	COURIER	PRICE
1	21-MAR-23	Express	DHL	2.99
2	22-MAR-23	Standard	UPS	5.99
3	24-MAR-23	Next Day	FedEx	3.99

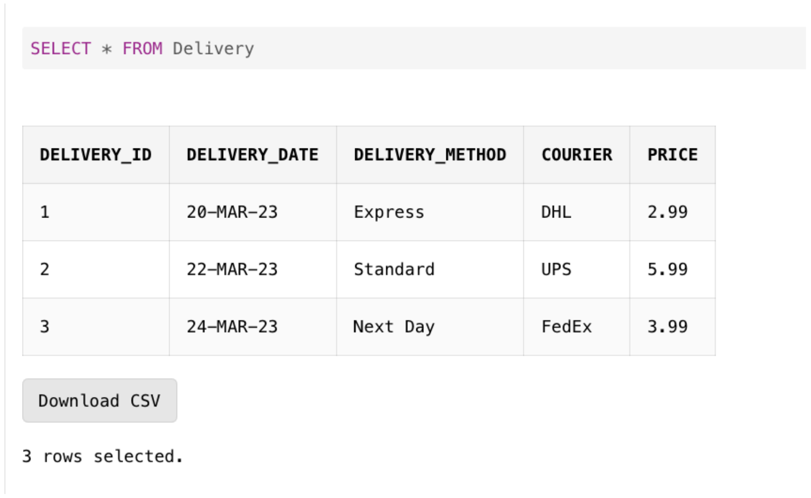
iv. ROLLBACK

```
ROLLBACK;
```



```
ROLLBACK
```

```
Statement processed.
```



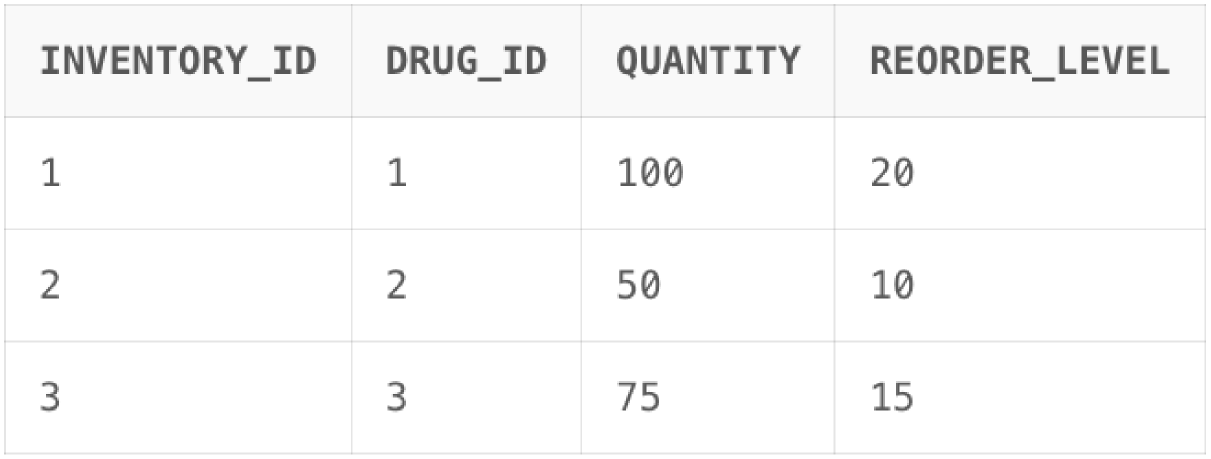
```
SELECT * FROM Delivery
```

DELIVERY_ID	DELIVERY_DATE	DELIVERY_METHOD	COURIER	PRICE
1	20-MAR-23	Express	DHL	2.99
2	22-MAR-23	Standard	UPS	5.99
3	24-MAR-23	Next Day	FedEx	3.99

Download CSV

```
3 rows selected.
```

<u>v. Data before the DELETE command</u>



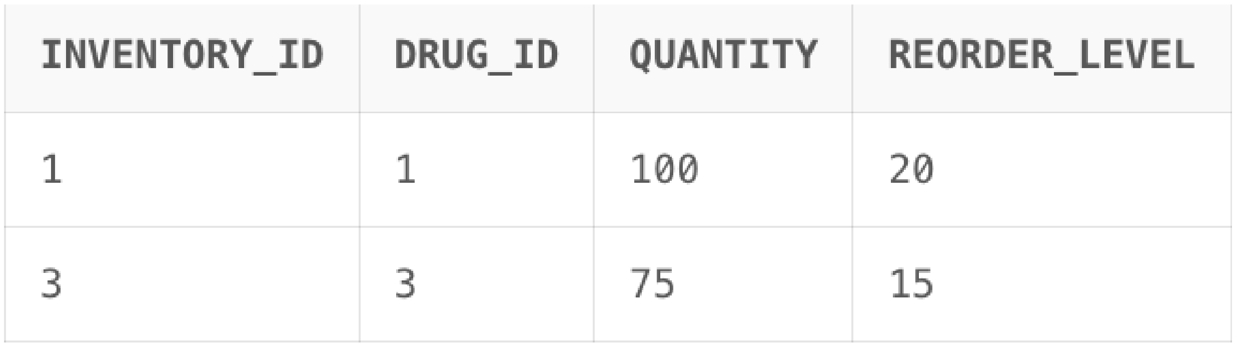
INVENTORY_ID	DRUG_ID	QUANTITY	REORDER_LEVEL
1	1	100	20
2	2	50	10
3	3	75	15

<u>vi. DELETE command</u>

DELETE from Inventory WHERE drug_id equals 2. *The pharmacy stopped supplying drug with drug_id=2 due to an FDA recall where the drug is now banned from being sold.*

```
DELETE FROM Inventory
WHERE drug_id = 2;
```

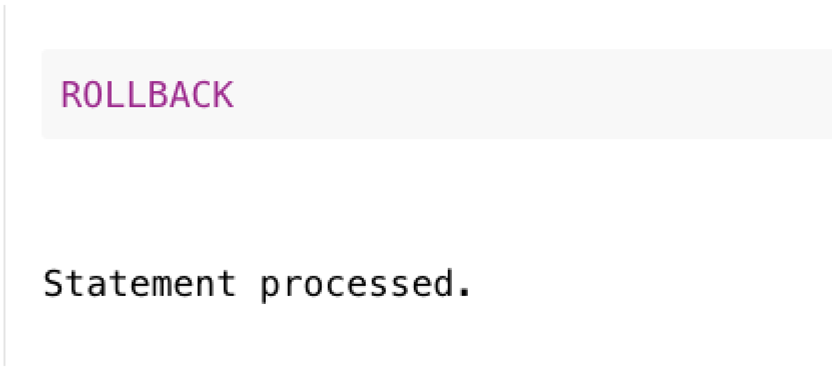
<u>vii. Data after the DELETE command</u>



INVENTORY_ID	DRUG_ID	QUANTITY	REORDER_LEVEL
1	1	100	20
3	3	75	15

<u>viii. ROLLBACK</u>

```
ROLLBACK;

SELECT * FROM Inventory
```



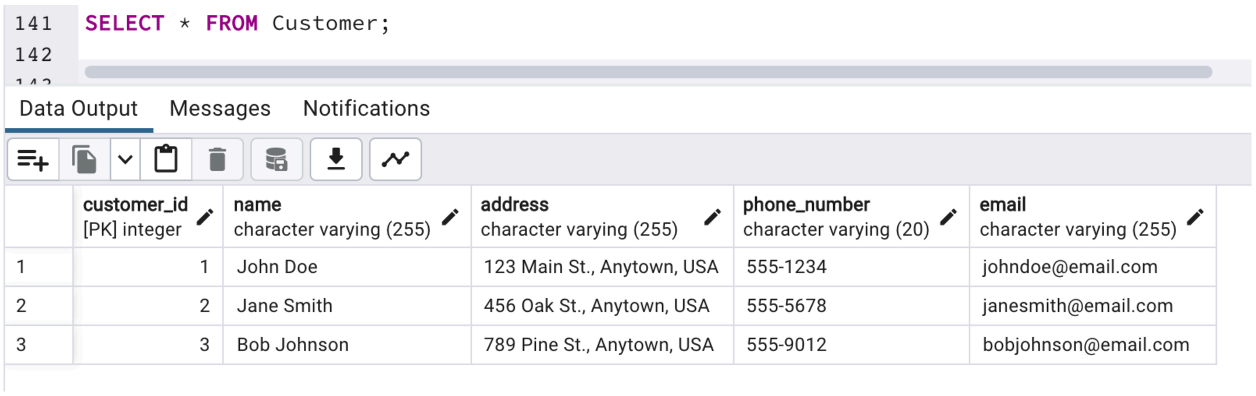
```
ROLLBACK

Statement processed.
```

```sql
SELECT * FROM Inventory
```

INVENTORY_ID	DRUG_ID	QUANTITY	REORDER_LEVEL
1	1	100	20
2	2	50	10
3	3	75	15

d. DML by Brooke Law

<u>i. Data before the UPDATE command</u>

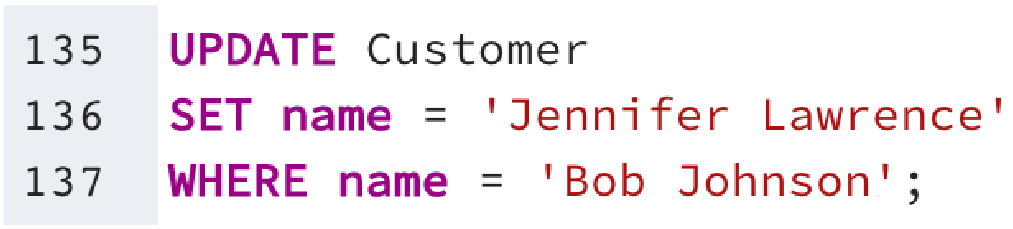


```sql
SELECT * FROM Customer;
```

	customer_id [PK] integer	name character varying (255)	address character varying (255)	phone_number character varying (20)	email character varying (255)
1	1	John Doe	123 Main St., Anytown, USA	555-1234	johndoe@email.com
2	2	Jane Smith	456 Oak St., Anytown, USA	555-5678	janesmith@email.com
3	3	Bob Johnson	789 Pine St., Anytown, USA	555-9012	bobjohnson@email.com

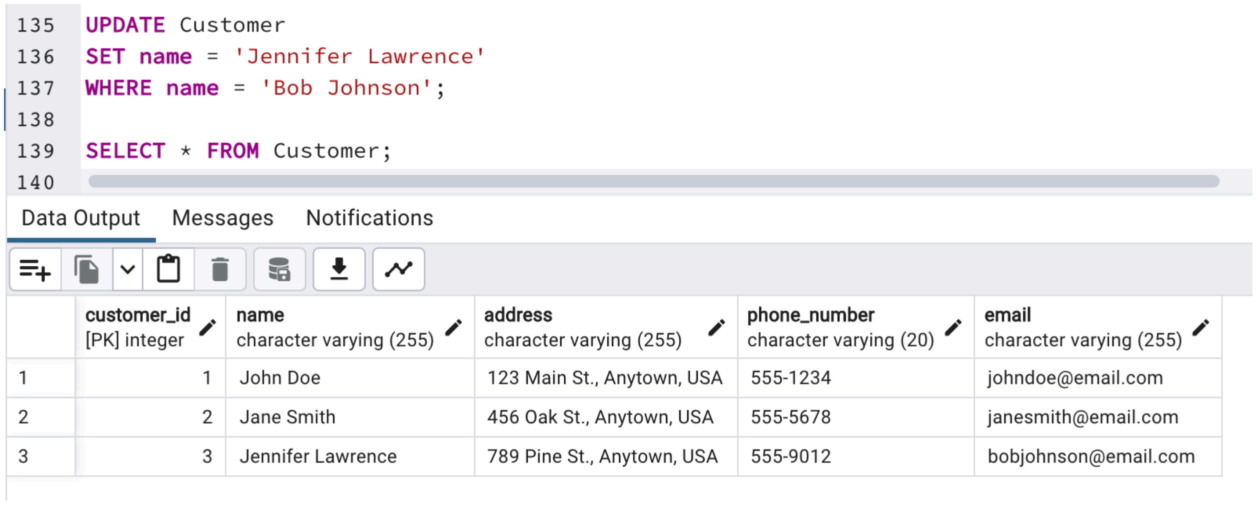
<u>ii. UPDATE command</u>

UPDATE customer name to Jennifer Lawrence WHERE name is currently Bob Johnson. *Since Bob Johnson is no longer a customer we need to update customer information with the new customer Jennifer Lawrence.*



```sql
UPDATE Customer
SET name = 'Jennifer Lawrence'
WHERE name = 'Bob Johnson';
```

iii. Data after the UPDATE command

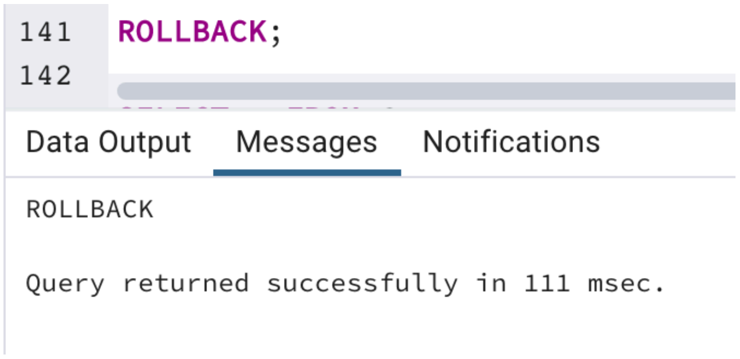


```sql
UPDATE Customer
SET name = 'Jennifer Lawrence'
WHERE name = 'Bob Johnson';

SELECT * FROM Customer;
```

Data Output Messages Notifications

	customer_id [PK] integer	name character varying (255)	address character varying (255)	phone_number character varying (20)	email character varying (255)
1	1	John Doe	123 Main St., Anytown, USA	555-1234	johndoe@email.com
2	2	Jane Smith	456 Oak St., Anytown, USA	555-5678	janesmith@email.com
3	3	Jennifer Lawrence	789 Pine St., Anytown, USA	555-9012	bobjohnson@email.com

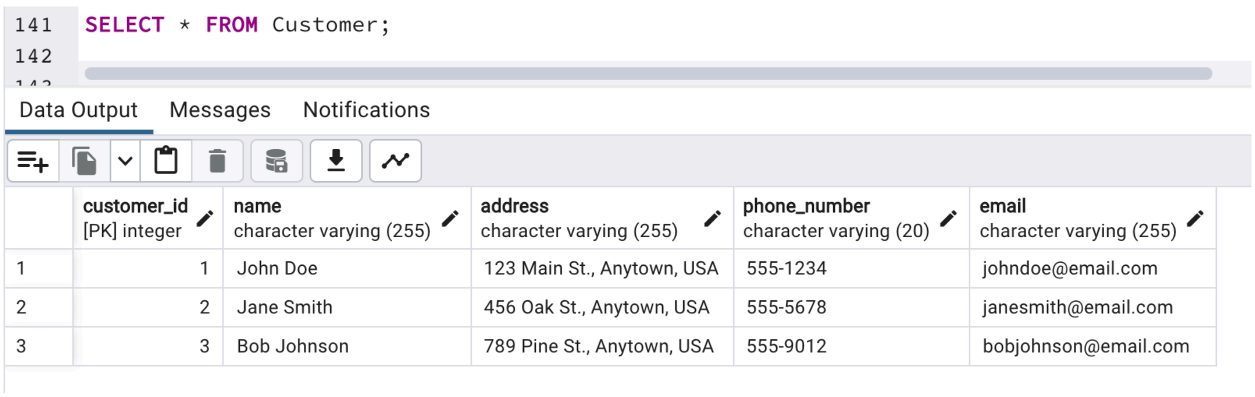
iv. ROLLBACK



```sql
ROLLBACK;
```

Data Output Messages Notifications

```
ROLLBACK

Query returned successfully in 111 msec.
```

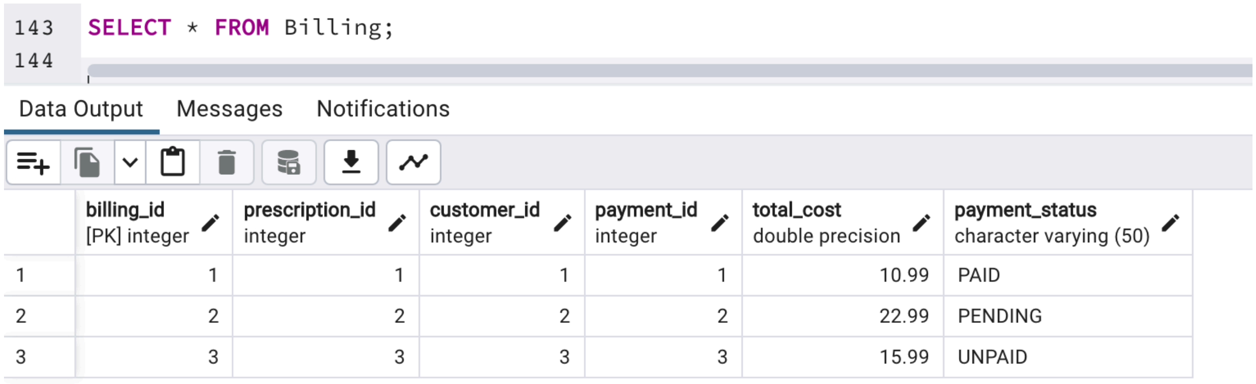


```
SELECT * FROM Customer;
```

Data Output Messages Notifications

	customer_id [PK] integer	name character varying (255)	address character varying (255)	phone_number character varying (20)	email character varying (255)
1	1	John Doe	123 Main St., Anytown, USA	555-1234	johndoe@email.com
2	2	Jane Smith	456 Oak St., Anytown, USA	555-5678	janesmith@email.com
3	3	Bob Johnson	789 Pine St., Anytown, USA	555-9012	bobjohnson@email.com

<u>v. Data before the DELETE command</u>



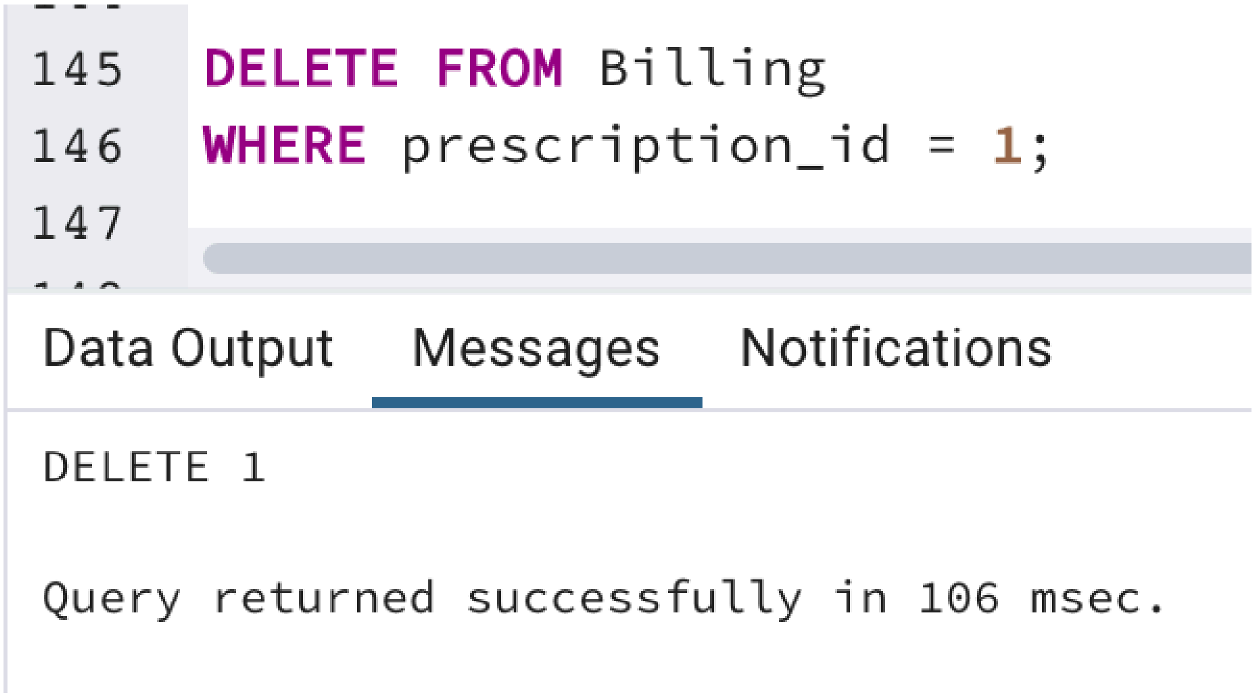
```
SELECT * FROM Billing;
```

Data Output Messages Notifications

	billing_id [PK] integer	prescription_id integer	customer_id integer	payment_id integer	total_cost double precision	payment_status character varying (50)
1	1	1	1	1	10.99	PAID
2	2	2	2	2	22.99	PENDING
3	3	3	3	3	15.99	UNPAID

<u>vi. DELETE command</u>

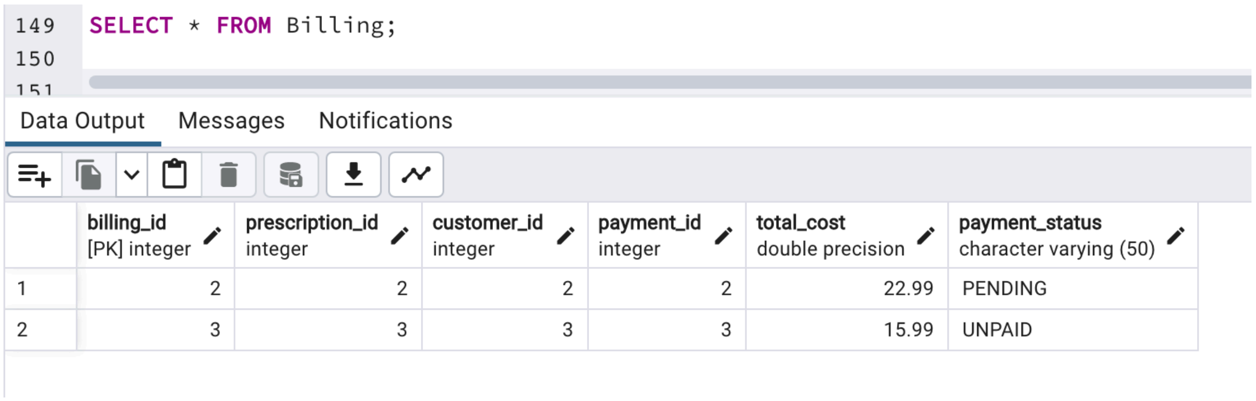
DELETE from Billing WHERE prescription_id equals 1. *The pharmacy stopped billing for prescription_id (hypothetically because manufacturer's started providing it for free).*



```sql
DELETE FROM Billing
WHERE prescription_id = 1;
```

```
DELETE 1

Query returned successfully in 106 msec.
```

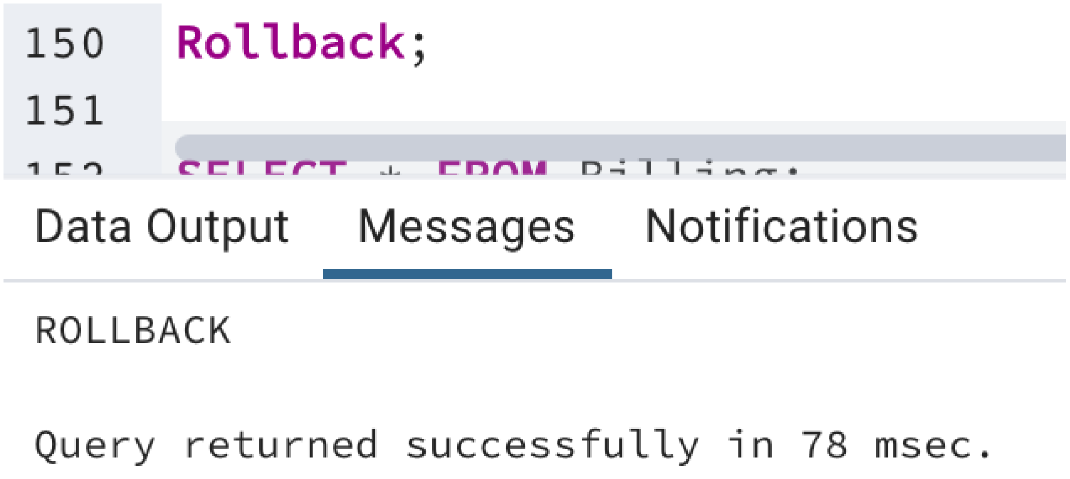
vii. Data after the DELETE command



```sql
SELECT * FROM Billing;
```

	billing_id [PK] integer	prescription_id integer	customer_id integer	payment_id integer	total_cost double precision	payment_status character varying (50)
1	2	2	2	2	22.99	PENDING
2	3	3	3	3	15.99	UNPAID

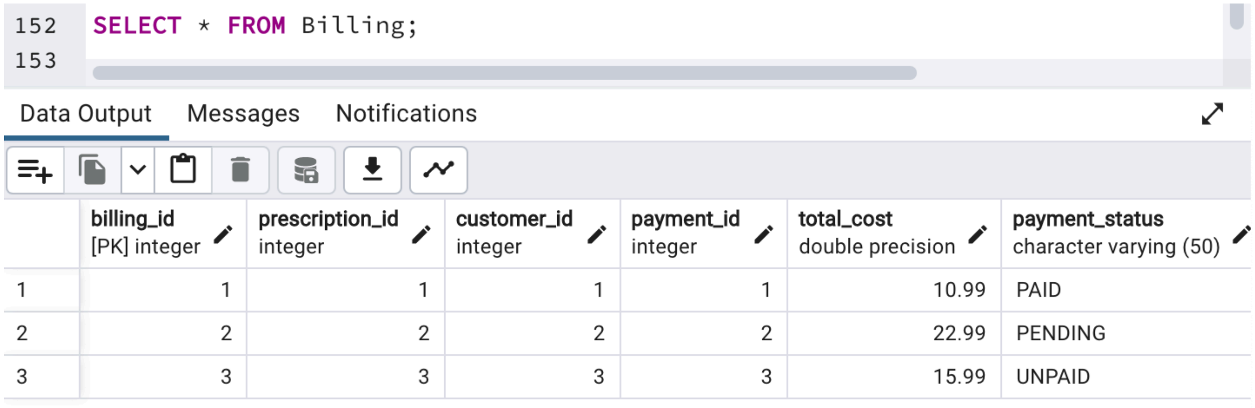
viii. ROLLBACK



```
Rollback;

SELECT * FROM Billing;
```

Data Output | Messages | Notifications

```
ROLLBACK

Query returned successfully in 78 msec.
```



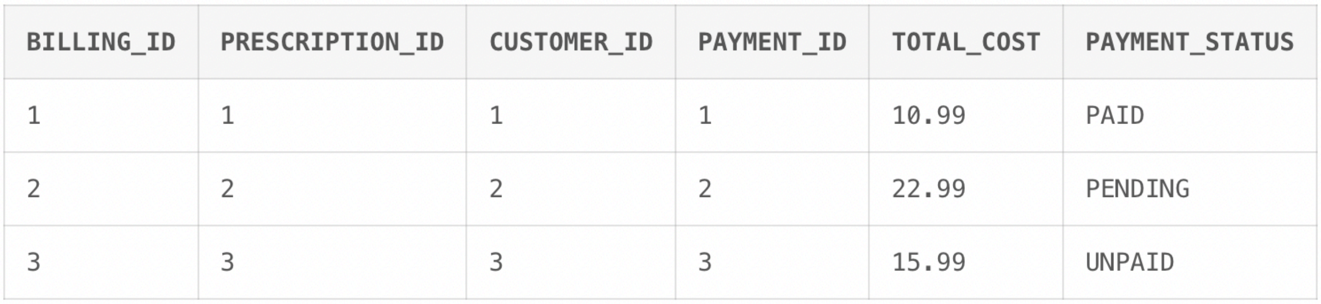
```
SELECT * FROM Billing;
```

Data Output | Messages | Notifications

	billing_id [PK] integer	prescription_id integer	customer_id integer	payment_id integer	total_cost double precision	payment_status character varying (50)
1	1	1	1	1	10.99	PAID
2	2	2	2	2	22.99	PENDING
3	3	3	3	3	15.99	UNPAID

e. DML by Amanda Collins

i. Data before the UPDATE command



BILLING_ID	PRESCRIPTION_ID	CUSTOMER_ID	PAYMENT_ID	TOTAL_COST	PAYMENT_STATUS
1	1	1	1	10.99	PAID
2	2	2	2	22.99	PENDING
3	3	3	3	15.99	UNPAID

<u>ii. UPDATE command</u>

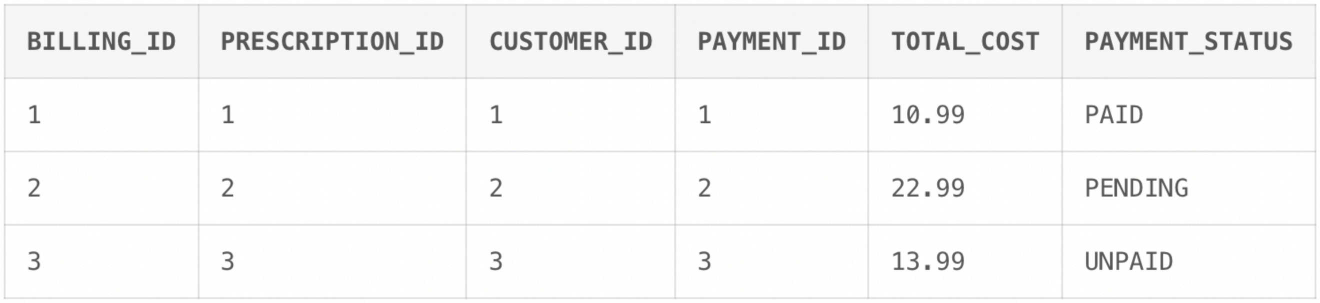
Implement a $2 discount for the customer whose ID is 3

UPDATE Billing

SET total_Cost = total_cost - 2.00

WHERE customer_id = 3;

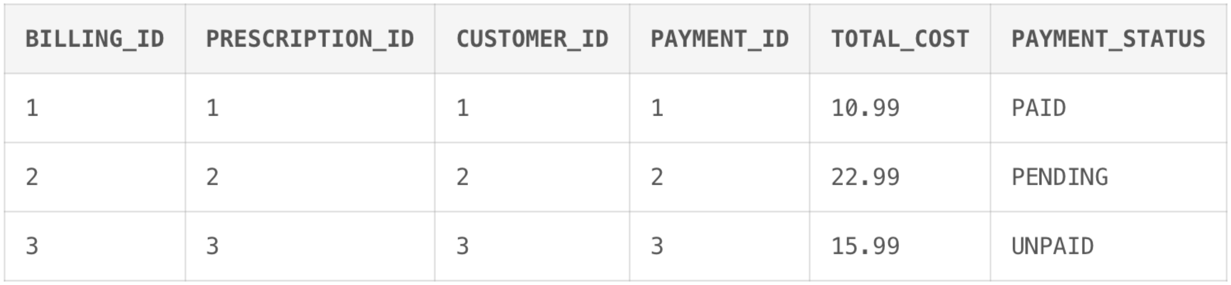
<u>iii. Data after the UPDATE command</u>



BILLING_ID	PRESCRIPTION_ID	CUSTOMER_ID	PAYMENT_ID	TOTAL_COST	PAYMENT_STATUS
1	1	1	1	10.99	PAID
2	2	2	2	22.99	PENDING
3	3	3	3	13.99	UNPAID

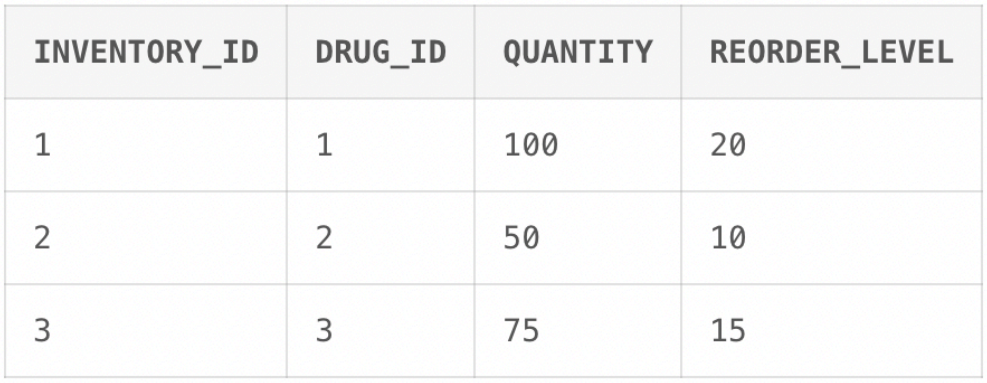
<u>iv. ROLLBACK</u>

```
UPDATE Billing
SET total_Cost = total_cost - 2.00
WHERE customer_id = 3;

ROLLBACK
```



BILLING_ID	PRESCRIPTION_ID	CUSTOMER_ID	PAYMENT_ID	TOTAL_COST	PAYMENT_STATUS
1	1	1	1	10.99	PAID
2	2	2	2	22.99	PENDING
3	3	3	3	15.99	UNPAID

<u>v. Data before the DELETE command</u>



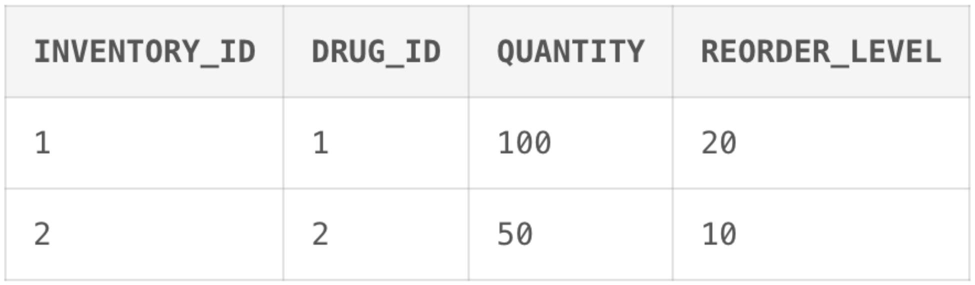
INVENTORY_ID	DRUG_ID	QUANTITY	REORDER_LEVEL
1	1	100	20
2	2	50	10
3	3	75	15

<u>vi. DELETE command</u>

The pharmaceutical company's contract with Johnson & Johnson suppliers has ended. Delete any drugs from the inventory that are manufactured by Johnson & Johnson

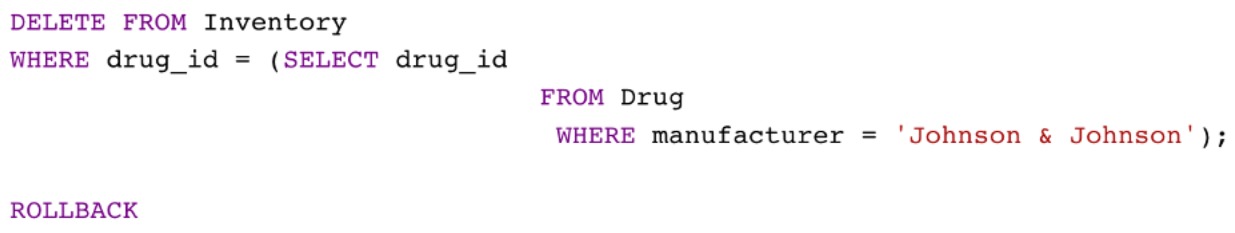
```
DELETE FROM Inventory

WHERE drug_id = (SELECT drug_id

                 FROM Drug

                  WHERE manufacturer = 'Johnson & Johnson');
```

<u>vii. Data after the DELETE command</u>

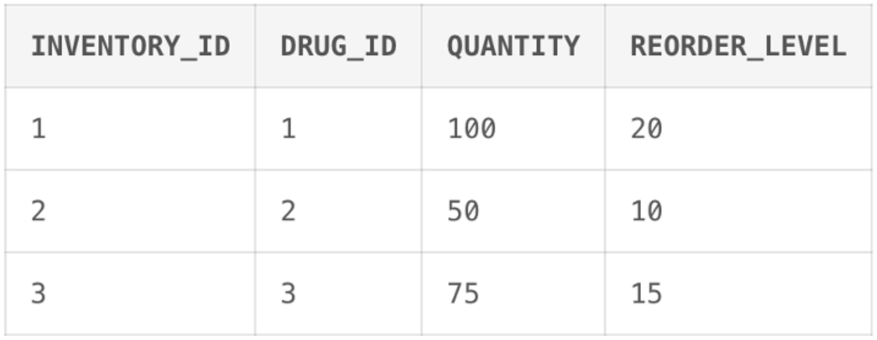


INVENTORY_ID	DRUG_ID	QUANTITY	REORDER_LEVEL
1	1	100	20
2	2	50	10

viii. ROLLBACK



```
DELETE FROM Inventory
WHERE drug_id = (SELECT drug_id
                                  FROM Drug
                                   WHERE manufacturer = 'Johnson & Johnson');

ROLLBACK
```



INVENTORY_ID	DRUG_ID	QUANTITY	REORDER_LEVEL
1	1	100	20
2	2	50	10
3	3	75	15

10. Summary

(Summarize your experience and lesson learned in terms of domain, scope, and implementation, including database creation, developing ERDs, using a database software, using SQL, populating data, special SQL tricks, and other lessons learned. You can also explain how your database can support the goals and facilitate the data management needs in the organization. You can also add what you could do more if you have more time.) The length of the summary of each member could be one paragraph (5-10 sentences).

a. Summary by George Papadopoulos

It was a lot of fun working with databases so closely, having little to no experience in databasing it was intimidating at first. Once I was able to work more closely in Oracle it started to unlock a lot of possibilities. Being that I use specialty medication it was interesting to put that into practical use while doing this project. Trying to figure out what would be the most important information for me to make the process as seamless as possible and have the correct data available to use. With so many moving pieces in the supply chain, it was difficult to decide what exactly we wanted to target and which pieces of data were most important to the overall success of the project.

b. Summary Michael Le

Before taking this class, I had zero practical experience using SQL or even managing databases. Looking back now, this course has helped me learn about the lifecycle of designing, developing, and implementing a database speaking generally. I was able to define the scope, and the domain of our project that could help our team think of different ways to implement our data and how it can be built with specific tools and framework. Along the way, we have developed relationships between entities, designing the schema, and creating tables. Not only that, we were able to collaboratively work on a project to discuss the direction of our projects and how we would want our database system to be built and to resolve any issues that come up. One issue that came up during our review of our unanimously voted ERD was that it was missing some entities and had some cardinality issues with the relationship, but we were able to modify it so that it is better suited. I am confident to say that I am proud of being able to implement a solution to our pharmaceutical database system as well as the collaboration efforts done. Not only that, I was able to take away a lot of lessons learned from this project, which includes the technical aspects of SQL, and being able to develop ERDs. If I had more time to be able to work on this project, I think we could have tried to develop a web application of sorts to be able to have a functioning pharmaceutical tool that could potentially be utilized. Nonetheless, I believe that our database can support the goals and facilitate the data management needs of our project outline.

c. Summary by Kelsey Chong

Throughout the project, I gained a deep understanding of database creation, ERD development, and SQL. My knowledge of using database software and accurately populating data improved significantly. Moreover, I discovered some useful SQL tricks that made my work more efficient, including using aliases, joining tables efficiently, and using aggregate functions. For example, I could use the SUM function to calculate the total sales for a particular product or category or the AVG function to find the average salary of employees in a department. Or, instead of using subqueries or multiple joins, we can use join methods like INNER JOIN, LEFT JOIN, RIGHT JOIN, and OUTER JOIN to combine data from different tables. This improves query performance and makes them more easily understood. One of the most crucial lessons I learned was the importance of a well-designed database schema. In the beginning, we made some mistakes which taught me how to improve a database schema. Developing a solid ERD allowed our team to create a database that was easy to use and maintain as we continued the project. Using SQL, I could retrieve data quickly and efficiently. I am proud of the work I have done on this project. I believe that the database I created could be a valuable tool for an organization such as a commercial pharmacy, which would need a reliable database that avoids redundancy and can handle large volumes of customers. Our database can support a pharmacy's business and expansion goals by allowing them to store and manage its data effectively while providing the potential for them to grow as a business as they increase the number of customers. Additionally, the database can facilitate their data management needs by providing easy access to the information they need. If I had more time, I would have focused on optimizing the queries to make them run more quickly, improving the database's performance. Overall, I found this project challenging yet rewarding, and I look forward to using the skills I have learned in future projects.

d. Summary by Brooke Law

This project has helped me to gain a greater understanding of ERD diagram implementation, database creation, and real world database applications. Because of milestone 1, I was able to learn how to create a strong ERD, how to better determine cardinality, and how to better identify strong versus weak relationships between entities. Milestone 1 also helped me to think more critically through the different parts of a problem—considering what tables & attributes would make sense to include versus which ones would make more sense to omit. Previously, I had taken a mini online course that used PostgreSQL, but the creation, manipulation, and DML phases of this project really helped me to gain a better understanding of how SQL commands work. The database manipulation phase, especially, increased my confidence with JOIN and aggregate functions such as COUNT, DISTINCT, and SUM. After working on this project, I definitely feel more confident in my, overall, SQL abilities. Our database could be implemented by an actual pharmacy to better manage their drug deliveries. Our database is pretty straightforward, making it easy to work with, enter, delete, and prevent redundant information from being added. If given more time I think our group could have added more tables to possibly keep track of insurance providers & revenue during Q1, 2, 3, and 4. I think those additions would have been useful for a pharmacy or pharmaceutical company because insurance is a big part of the healthcare sphere and so is revenue throughout the year. Overall, this experience was quite interesting and gave me a chance to work with a database in a capacity I had not prev

e. Summary by Amanda Collins

This project provided me with practical knowledge about managing databases that I have already begun implementing outside of the classroom. Before beginning this course, I had some very basic experience implementing SQL Queries, but I did not have any of the analytical understanding of how a database is built, how to determine the scope of a database, or how to create strong, non-redundant relationships within a database. Working through this project required constant communication about our database with my teammates, as well as frequent reorganization and redesign of our system. In order to be an effective teammate, I needed to ensure I was well versed in variable naming, relational integrity constraints, and ER model building. Because I was practicing these skills so frequently throughout the project, I feel more confident doing tasks like deciding which attributes would be the most effective primary key, defining cardinality in an ERD, and using aggregate functions to select data via SQL query. I believe the database my team and I have designed could be implemented within an online pharmaceutical delivery company and would perform effectively. Our tables capture data from the customer, the company itself, and related suppliers, all without redundancy, which is crucial for a business operated entirely online. Our database allows for queries between nearly every single one of our tables because of our foreign key placement, which is beneficial for company employees in accounting, the pharmacy, and customer service, as well as investors and suppliers outside of the company.

11. Appendix

Division of work among team members for any other parts that are not clear in the report such as work on Part A, data creation, Report Assembly, proofreading, etc. Any other information you want to include in the term project.

a. **Contributions by George Papadopoulos:**

- Commands for table creation and inserting data
- Data dictionary write-up
- Sample data creation

b. Contributions by Michael Le:

- Revising feedback from Milestone 1 to Final Project Document
- Document Proofreading
- Executive Summary

c. Contributions by Kelsey Chong:

- Overall information transfer from milestone 1 to final project document
- Assembly of entire Final Project Report structure
- Final ERD adjustments (creating Delivery table's "courier" column)
- Modification of commands, data dictionary and final ERD after Delivery table's column change
- Executive summary

d. Contributions by Brooke Law:

- Document proofreading
- Final ERD adjustments (primarily foreign key additions)
- Executive summary
- Final doc assembly

e. Contributions by Amanda Collins:

- Document proofreading
- Created table of contents
- Document formatting
- ERD and CREATE TABLE edits
- Assembly of word file containing queries