# Pharmaceutical Company Database

Rahul Rajan – AM.SC.U4CSE23069

Swayam Agrahari – AM.SC.U4CSE23073

Sakar Shrestha – AM.SC.U4CSE23076

Sudhir Kumar Sah – AM.SC.U4CSE23080

## Introduction

The "Pharmaceutical Company Database" is a system designed to efficiently manage the information related to medicines, medical stores, and their transactions. The database is intended to track medicine production, sales, inventory, and distribution across various medical stores. This system also allows the pharmaceutical company to manage its relationships with stores, the distribution of medicines, and sales data.

### Problem Statement

The "Pharmaceutical Company Database" aims to maintain comprehensive records for a pharmaceutical company by developing a structured and effective database system. The system should handle information on medicine production, medicine distribution to various stores, pricing, and inventory levels. Additionally, it will maintain the details of medical stores, track the supply of medicines to stores, and monitor sales and stock quantities. The goal is to provide the company with a robust, scalable database to ensure smooth operations, proper tracking of inventory, and accurate order management.

### Specification

We began by creating the **Medicines** entity set, which includes essential details about each medicine, such as the **medicine name**, **company name**, **date of manufacture**, **expiry date**, and **price**. These fields help the pharmaceutical company maintain accurate records about the medicines it manufactures.

Next, we established the **Stores** entity set to store details of various medical stores, such as **store_id**, **store_name**, and **location**. Stores are basically the customers for the Pharmaceutical company.

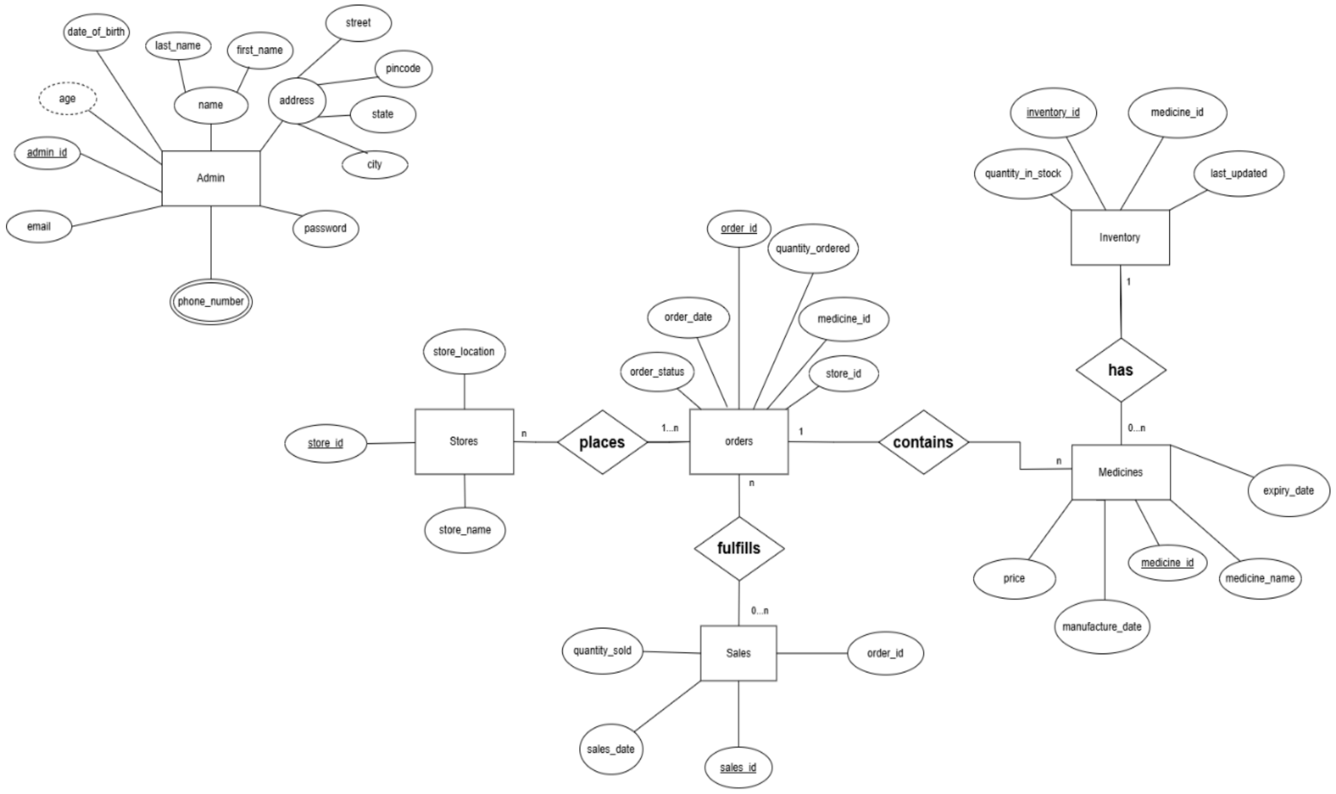
To manage the quantity of medicines, we introduced an **Inventory** entity that tracks the **quantity in stock** for every medicine, along with the **last updated** timestamp to ensure accurate inventory records. The inventory system is directly tied to the medicines, enabling real-time tracking of stock levels.

We also created an **Orders** entity to record the supply of medicines to stores. This entity includes details such as the **order_id**, **store_id**, **medicine_id**, **order_date**, **quantity_ordered**, and **order_status**. The **Orders** table ensures that the supply chain from the pharmaceutical company to the stores is properly managed, including the tracking of order statuses like "pending," "fulfilled," and "cancelled."

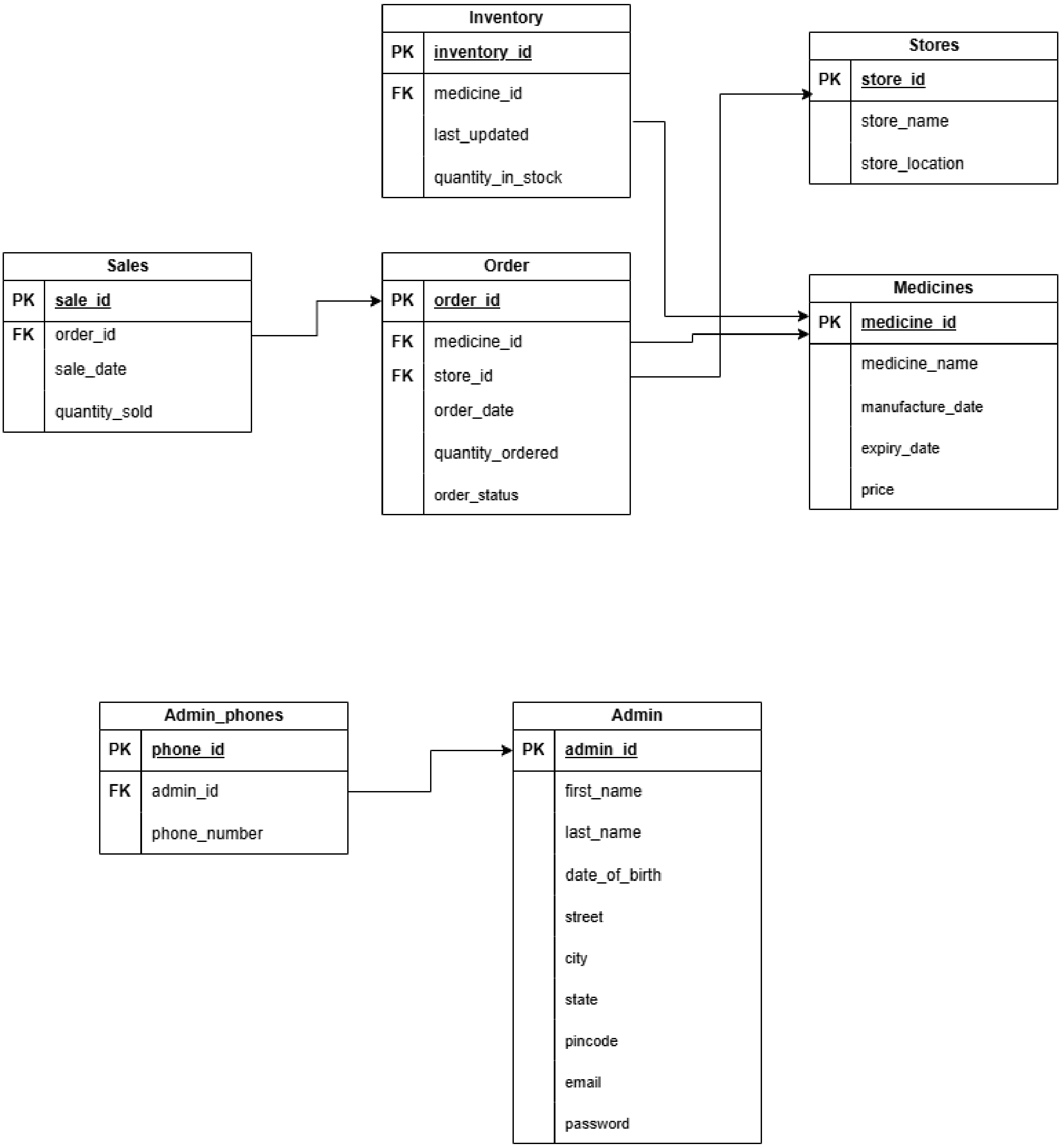
Finally, the **Sales** entity was implemented to track the sale of medicines at the stores. This entity records information such as the **sale_id**, **order_id**, **sale_date**, and **quantity_sold**. This allows the pharmaceutical company to monitor how much of each medicine has been sold and provides important insights into sales performance.

The entire database structure is designed to ensure that the company can efficiently manage its inventory, track the distribution of medicines, and monitor the sales across different stores.

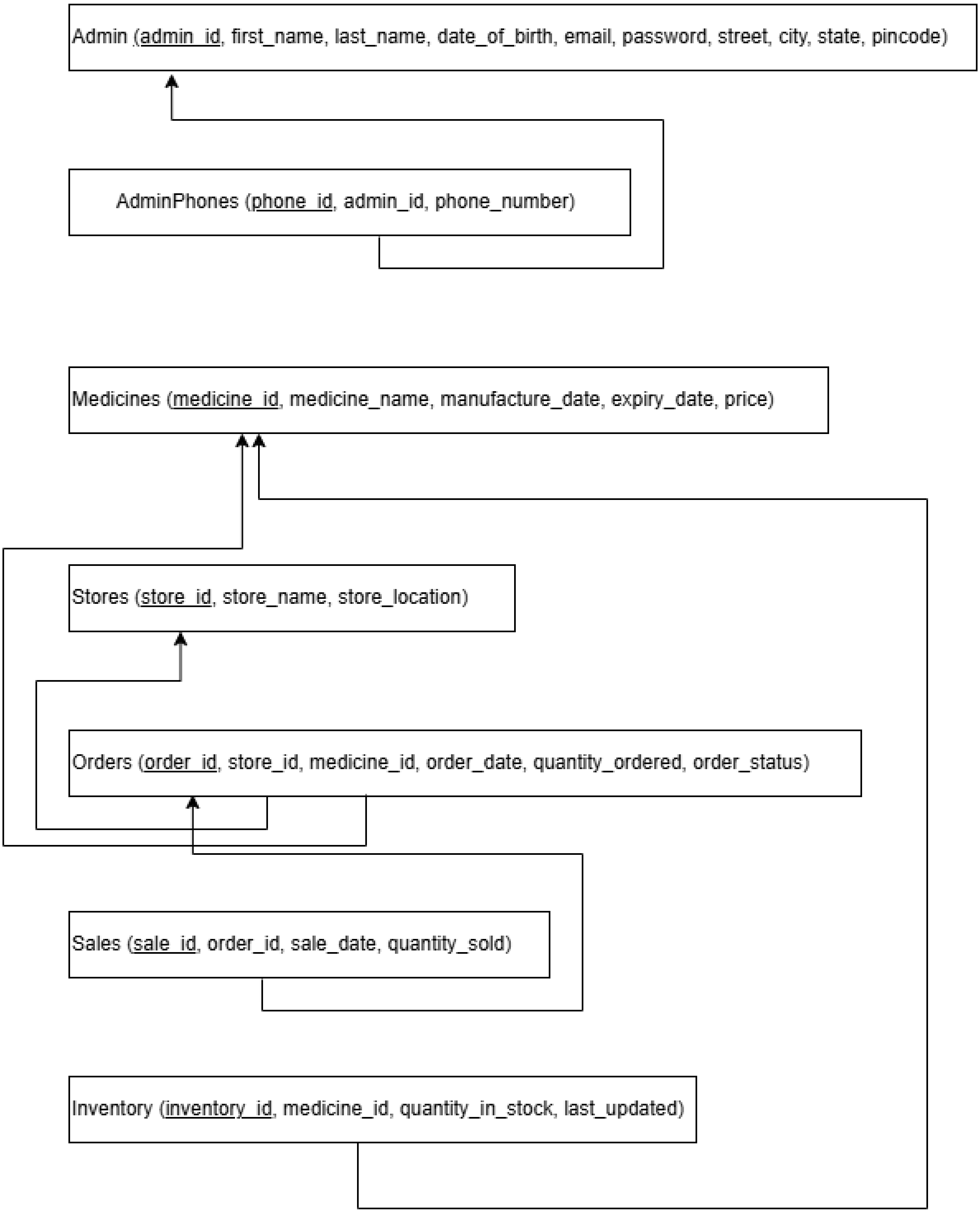
## ER Diagram


date_of_birth
last_name
first_name
street
pincode
age
name
address
state
admin_id
Admin
city
email
password
phone_number
inventory_id
medicine_id
quantity_in_stock
last_updated
Inventory
1
has
0...n
order_id
quantity_ordered
order_date
medicine_id
order_status
store_id
store_location
store_id
Stores
n
places
1...n
orders
1
contains
n
Medicines
expiry_date
store_name
n
fulfills
0...n
price
medicine_id
medicine_name
manufacture_date
quantity_sold
Sales
order_id
sales_date
sales_id

# Schema Diagram



## Relational Schema Diagram



Admin (admin_id, first_name, last_name, date_of_birth, email, password, street, city, state, pincode)

AdminPhones (phone_id, admin_id, phone_number)

Medicines (medicine_id, medicine_name, manufacture_date, expiry_date, price)

Stores (store_id, store_name, store_location)

Orders (order_id, store_id, medicine_id, order_date, quantity_ordered, order_status)

Sales (sale_id, order_id, sale_date, quantity_sold)

Inventory (inventory_id, medicine_id, quantity_in_stock, last_updated)

## Normalization

**0) Universal Table**

PharmaceuticalSystem (admin_id, first_name, last_name, date_of_birth, age, phone_number, email, password, street, city, state, pincode, medicine_id, medicine_name, manufacture_date, expiry_date, price, store_id, store_name, store_location, order_id, order_date, quantity_ordered, order_status, sale_id, sale_date, quantity_sold, inventory_id, quantity_in_stock, last_updated)

**1) NF**

**Rule:** Each attribute must contain atomic values only

Breaking down composite and multi-valued attributes.

PharmaSystem (admin_id, first_name, last_name, date_of_birth, age, phone_number, email, password, street, city, state, pincode, medicine_id, medicine_name, manufacture_date, expiry_date, price, store_id, store_name, store_location, order_id, order_date, quantity_ordered, order_status, sale_id, sale_date, quantity_sold, inventory_id, quantity_in_stock, last_updated)

PRIMARY KEY (admin_id, medicine_id, store_id, order_id, sale_id, inventory_id)

AdminPhones (admin_id, phone_number)

PRIMARY KEY (admin_id, phone_number)

## Functional Dependency

**1. Admin Dependencies:**

admin_id → first_name, last_name, date_of_birth, street, city, state, pincode

date_of_birth → age

**2. Medicine Dependencies:**

medicine_id → medicine_name, manufacture_date, expiry_date, price

**3. Store Dependencies:**

store_id → store_name, store_location

**4. Order Dependencies:**

order_id → store_id, medicine_id, order_date, quantity_ordered, order_status

**5. Sale Dependencies:**

sale_id → order_id, sale_date, quantity_sold

**6. Inventory Dependencies:**

inventory_id → medicine_id, quantity_in_stock, last_updated

**2) 2NF**

**Rule:** No partial dependencies on the primary key

i. Admin (admin_id, first_name, last_name, date_of_birth, email, password, age, street, city, state, pincode)
ii. AdminPhones (admin_id, phone_number)
iii. Medicines (medicine_id, medicine_name, manufacture_date, expiry_date, price)
iv. Stores (store_id, store_name, store_location)
v. Orders (order_id, store_id, medicine_id, order_date, quantity_ordered, order_status)
vi. Sales (sale_id, order_id, sale_date, quantity_sold)
vii. Inventory (inventory_id, medicine_id, quantity_in_stock, last_updated)

**3) 3NF**

**Rule:** No transitive dependencies

Since the decomposition from 2NF does not introduce transitive dependencies, all tables are already in 3NF.

i. Admin (admin_id, first_name, last_name, date_of_birth, email, password, street, city, state, pincode)
ii. AdminPhones (phone_id, admin_id, phone_number)
iii. Medicines (medicine_id, medicine_name, manufacture_date, expiry_date, price)
iv. Stores (store_id, store_name, store_location)
v. Orders (order_id, store_id, medicine_id, order_date, quantity_ordered, order_status)
vi. Sales (sale_id, order_id, sale_date, quantity_sold)
vii. Inventory (inventory_id, medicine_id, quantity_in_stock, last_updated)

## Queries

```
CREATE TABLE Admin (

   admin_id INT PRIMARY KEY,

   first_name VARCHAR(50) NOT NULL,

   last_name VARCHAR(50) NOT NULL,

   date_of_birth DATE NOT NULL,

   street VARCHAR(100) NOT NULL,

   city VARCHAR(50) NOT NULL,

   state VARCHAR(50) NOT NULL,

   pincode VARCHAR(10) NOT NULL

);

ALTER TABLE Admin
```

```
ADD COLUMN email VARCHAR(50) UNIQUE NOT NULL;
ALTER TABLE Admin
ADD COLUMN password VARCHAR(255) NOT NULL;
CREATE TABLE AdminPhones (
    phone_id SERIAL PRIMARY KEY,
    admin_id INT,
    phone_number VARCHAR(15) NOT NULL,
    FOREIGN KEY (admin_id) REFERENCES Admin(admin_id)
        ON DELETE CASCADE
        ON UPDATE CASCADE
);

CREATE TABLE Medicines (
    medicine_id SERIAL PRIMARY KEY,
    medicine_name VARCHAR(255) NOT NULL,
    manufacture_date DATE NOT NULL,
    expiry_date DATE NOT NULL,
    price DECIMAL(10,2) NOT NULL,
    CONSTRAINT valid_dates CHECK (expiry_date > manufacture_date),
    CONSTRAINT valid_price CHECK (price > 0)
);

CREATE TABLE Stores (
    store_id SERIAL PRIMARY KEY,
    store_name VARCHAR(255) NOT NULL,
    store_location VARCHAR(255) NOT NULL
);

CREATE TABLE Orders (
    order_id SERIAL PRIMARY KEY,
    store_id INT NOT NULL,
    medicine_id INT NOT NULL,
    order_date DATE NOT NULL,
    quantity_ordered INT NOT NULL,
    order_status VARCHAR(50) NOT NULL,
    FOREIGN KEY (store_id) REFERENCES Stores(store_id)
        ON DELETE CASCADE
        ON UPDATE CASCADE,
    FOREIGN KEY (medicine_id) REFERENCES Medicines(medicine_id)
        ON DELETE CASCADE
        ON UPDATE CASCADE,
    CONSTRAINT valid_status CHECK (order_status IN ('Pending', 'Fulfilled', 'Cancelled')),
    CONSTRAINT valid_quantity CHECK (quantity_ordered > 0)
);

CREATE TABLE Sales (
    sale_id SERIAL PRIMARY KEY,
    order_id INT NOT NULL,
    sale_date DATE NOT NULL,
    quantity_sold INT NOT NULL,
    FOREIGN KEY (order_id) REFERENCES Orders(order_id)
        ON DELETE CASCADE
        ON UPDATE CASCADE,
    CONSTRAINT valid_sale_quantity CHECK (quantity_sold > 0),
    CONSTRAINT fulfilled_orders_only CHECK (
        order_id IN (SELECT order_id FROM Orders WHERE order_status = 'Fulfilled')
    )
);
```

```
CREATE TABLE Inventory (
    inventory_id SERIAL PRIMARY KEY,
    medicine_id INT NOT NULL,
    quantity_in_stock INT NOT NULL,
    last_updated TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (medicine_id) REFERENCES Medicines(medicine_id)
        ON DELETE CASCADE
        ON UPDATE CASCADE,
    CONSTRAINT valid_quantity CHECK (quantity_in_stock >= 0)
);
```

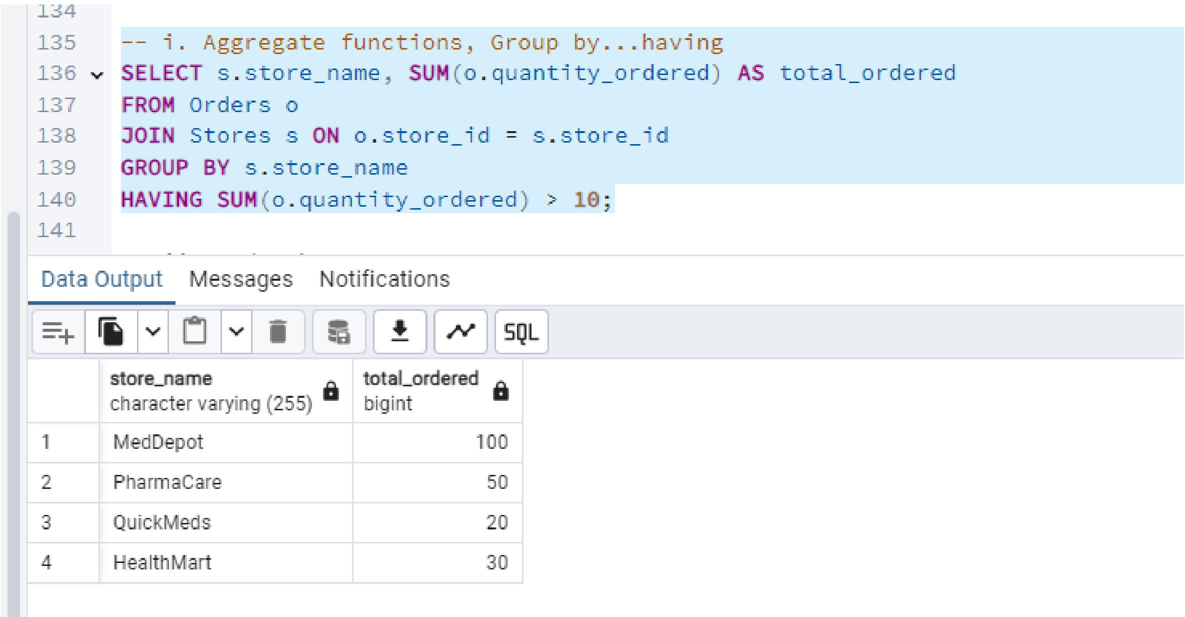
**Tuples Insertion**

```
-- Insertion of rows in each tables
INSERT INTO Medicines (medicine_id, medicine_name, manufacture_date, expiry_date, price)
VALUES
(1, 'Paracetamol', '2023-01-10', '2025-01-10', 50.00),
(2, 'Ibuprofen', '2022-06-15', '2024-06-15', 80.00),
(3, 'Amoxicillin', '2023-03-20', '2025-03-20', 120.00),
(4, 'Cough Syrup', '2022-11-01', '2024-11-01', 60.00),
(5, 'Aspirin', '2022-12-05', '2024-12-05', 40.00);

INSERT INTO Stores (store_id, store_name, store_location)
VALUES
(1, 'PharmaCare', 'Main Street, Cityville'),
(2, 'MedDepot', 'Oak Street, Townsville'),
(3, 'HealthMart', 'Pine Street, Villagetown'),
(4, 'QuickMeds', 'Birch Street, Lakeside'),
(5, 'CurePlus', 'Cedar Street, Riverside');
```

```
INSERT INTO Orders (store_id, medicine_id, order_date, quantity_ordered, order_status)
VALUES
(1, 1, '2023-12-01', 50, 'Pending'),
(2, 2, '2023-11-15', 100, 'Fulfilled'),
(3, 3, '2023-12-10', 30, 'Cancelled'),
(4, 4, '2023-12-05', 20, 'Fulfilled'),
(5, 5, '2023-12-20', 10, 'Pending');

INSERT INTO Sales (order_id, sale_date, quantity_sold)
VALUES
(1, '2023-12-15', 30),
(2, '2023-11-20', 100),
(3, '2023-12-12', 15),
(4, '2023-12-07', 18),
(5, '2023-12-25', 8);

INSERT INTO Inventory (medicine_id, quantity_in_stock)
VALUES
(1, 500),
(2, 300),
(3, 200),
(4, 150),
(5, 400);
```

## Generate Queries

### i. Aggregate functions, Group by...having

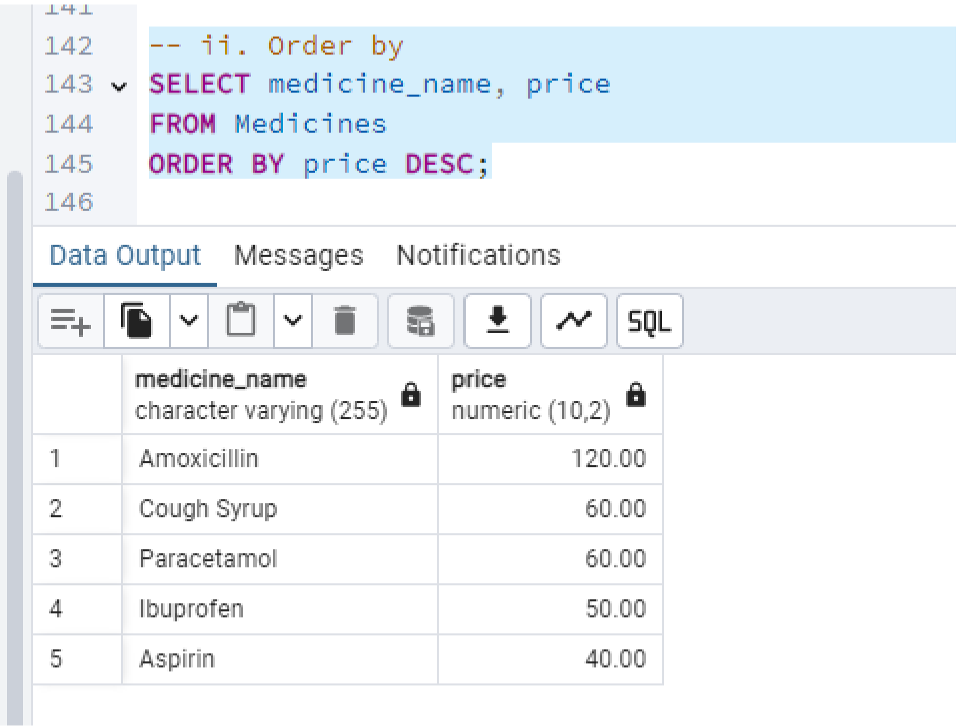


```sql
-- i. Aggregate functions, Group by...having
SELECT s.store_name, SUM(o.quantity_ordered) AS total_ordered
FROM Orders o
JOIN Stores s ON o.store_id = s.store_id
GROUP BY s.store_name
HAVING SUM(o.quantity_ordered) > 10;
```

| | store_name<br>character varying (255) | total_ordered<br>bigint |
|---|---|---|
| 1 | MedDepot | 100 |
| 2 | PharmaCare | 50 |
| 3 | QuickMeds | 20 |
| 4 | HealthMart | 30 |

This query sums the quantity_ordered for each store and filters only those stores that have ordered more than 100 units using the HAVING clause.

### ii. Order by

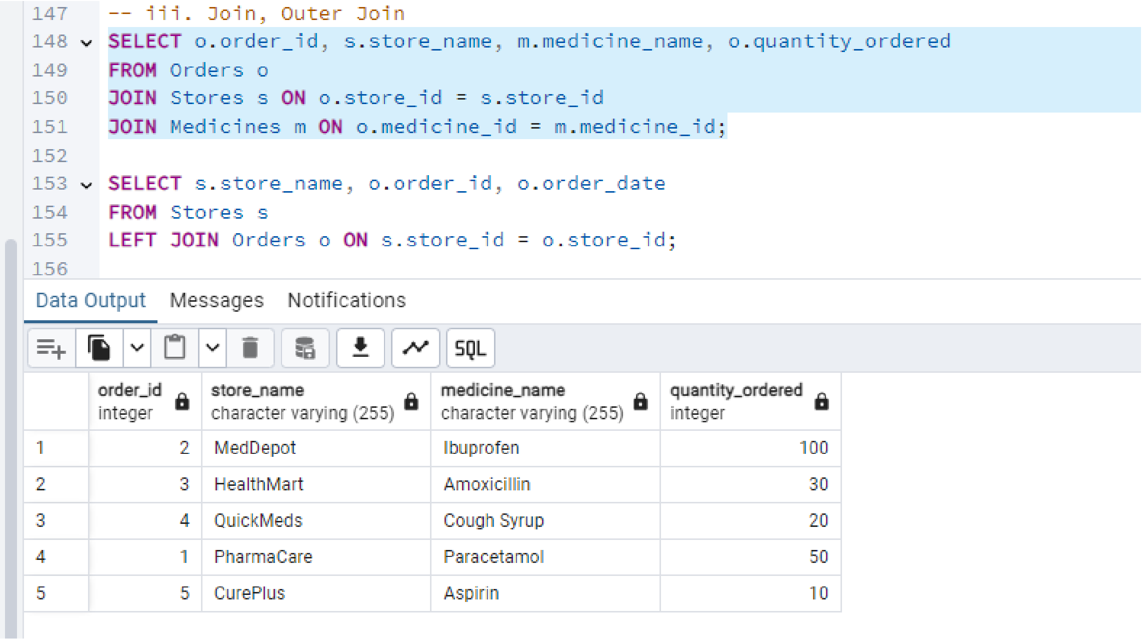


```sql
-- ii. Order by
SELECT medicine_name, price
FROM Medicines
ORDER BY price DESC;
```

| | medicine_name<br>character varying (255) | price<br>numeric (10,2) |
|---|---|---|
| 1 | Amoxicillin | 120.00 |
| 2 | Cough Syrup | 60.00 |
| 3 | Paracetamol | 60.00 |
| 4 | Ibuprofen | 50.00 |
| 5 | Aspirin | 40.00 |

This query orders the Medicines table by price in descending order.

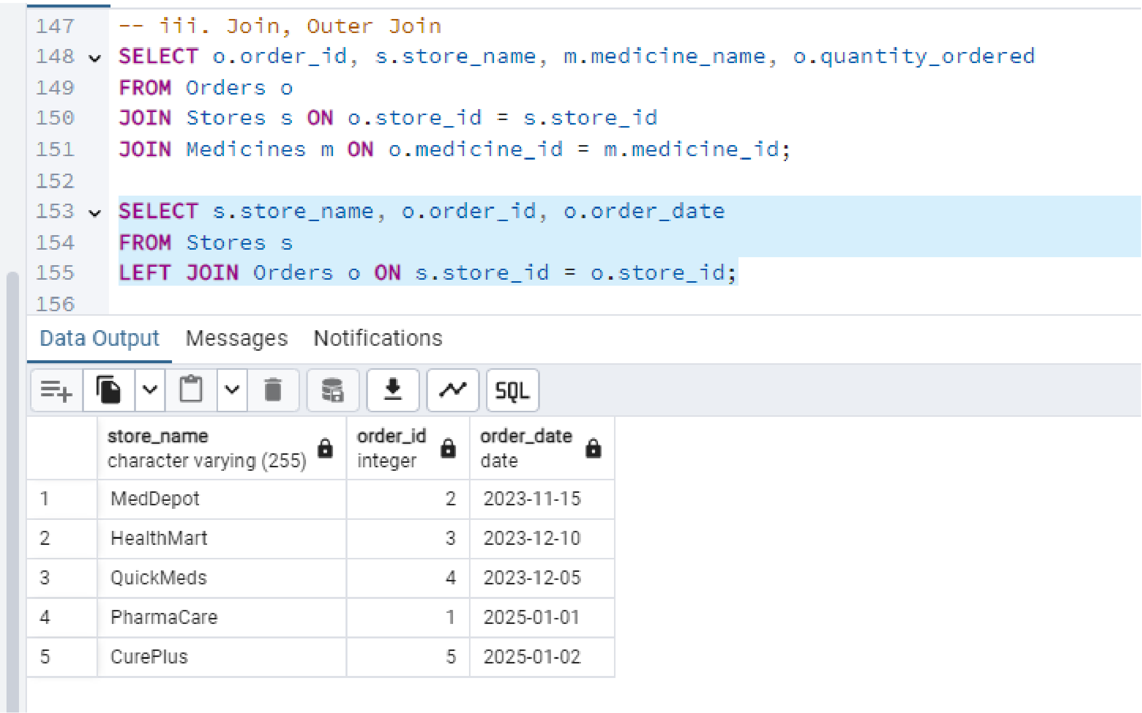
### iii. Join, Outer Join



```
-- iii. Join, Outer Join
SELECT o.order_id, s.store_name, m.medicine_name, o.quantity_ordered
FROM Orders o
JOIN Stores s ON o.store_id = s.store_id
JOIN Medicines m ON o.medicine_id = m.medicine_id;

SELECT s.store_name, o.order_id, o.order_date
FROM Stores s
LEFT JOIN Orders o ON s.store_id = o.store_id;

```

| | order_id<br>integer | store_name<br>character varying (255) | medicine_name<br>character varying (255) | quantity_ordered<br>integer |
|---|---|---|---|---|
| 1 | 2 | MedDepot | Ibuprofen | 100 |
| 2 | 3 | HealthMart | Amoxicillin | 30 |
| 3 | 4 | QuickMeds | Cough Syrup | 20 |
| 4 | 1 | PharmaCare | Paracetamol | 50 |
| 5 | 5 | CurePlus | Aspirin | 10 |

This query only returns rows where there are matching records in all Orders, Stores and Medicines tables.

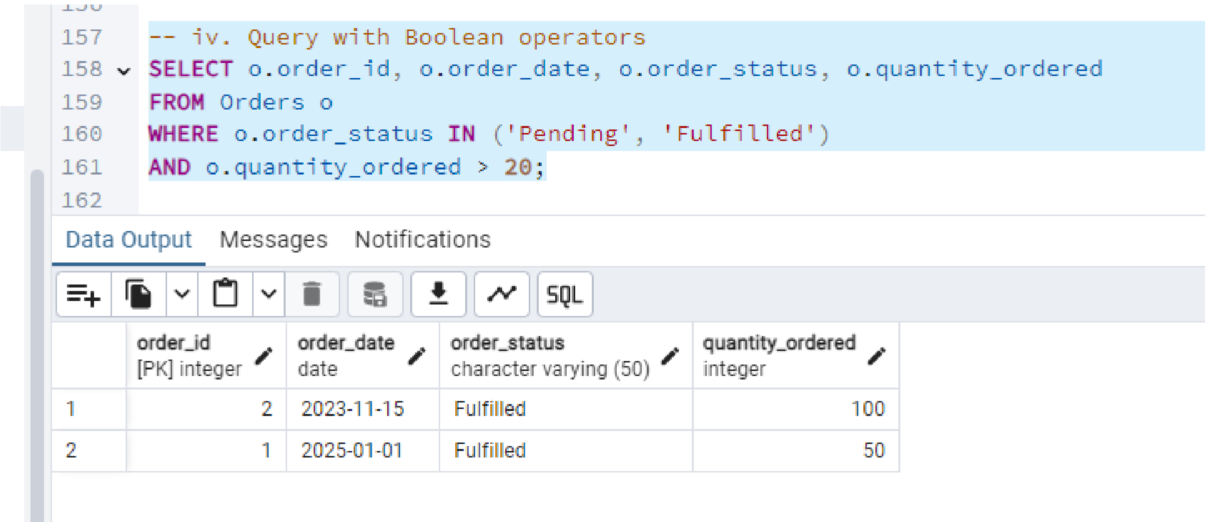


```
-- iii. Join, Outer Join
SELECT o.order_id, s.store_name, m.medicine_name, o.quantity_ordered
FROM Orders o
JOIN Stores s ON o.store_id = s.store_id
JOIN Medicines m ON o.medicine_id = m.medicine_id;

SELECT s.store_name, o.order_id, o.order_date
FROM Stores s
LEFT JOIN Orders o ON s.store_id = o.store_id;

```

| | store_name<br>character varying (255) | order_id<br>integer | order_date<br>date |
|---|---|---|---|
| 1 | MedDepot | 2 | 2023-11-15 |
| 2 | HealthMart | 3 | 2023-12-10 |
| 3 | QuickMeds | 4 | 2023-12-05 |
| 4 | PharmaCare | 1 | 2025-01-01 |
| 5 | CurePlus | 5 | 2025-01-02 |

This query returns all stores, even if they have no matching orders.

### iv. Query with Boolean operators

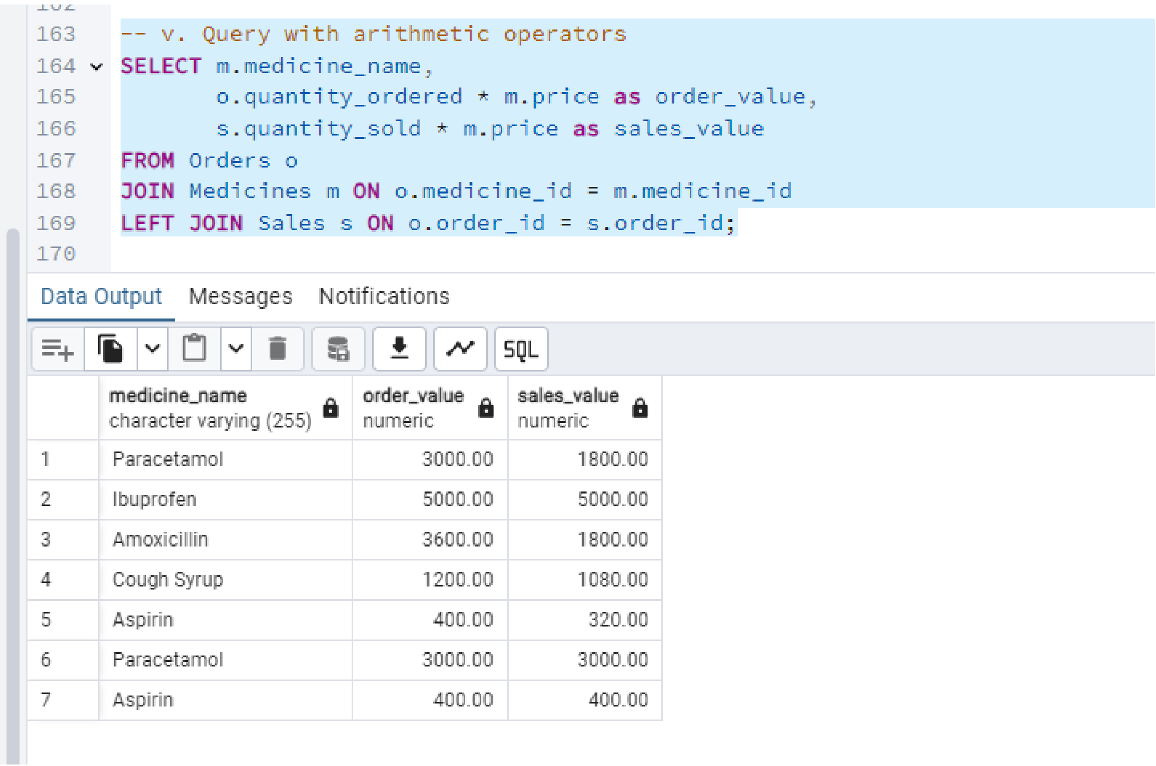


```
-- iv. Query with Boolean operators
SELECT o.order_id, o.order_date, o.order_status, o.quantity_ordered
FROM Orders o
WHERE o.order_status IN ('Pending', 'Fulfilled')
AND o.quantity_ordered > 20;
```

| | order_id<br>[PK] integer | order_date<br>date | order_status<br>character varying (50) | quantity_ordered<br>integer |
|---|---|---|---|---|
| 1 | 2 | 2023-11-15 | Fulfilled | 100 |
| 2 | 1 | 2025-01-01 | Fulfilled | 50 |

This query filters orders based on two boolean conditions: order_status is either 'Pending' or 'Fulfilled', and quantity_ordered is greater than 20.

### v. Query with arithmetic operators



```
-- v. Query with arithmetic operators
SELECT m.medicine_name,
       o.quantity_ordered * m.price as order_value,
       s.quantity_sold * m.price as sales_value
FROM Orders o
JOIN Medicines m ON o.medicine_id = m.medicine_id
LEFT JOIN Sales s ON o.order_id = s.order_id;
```

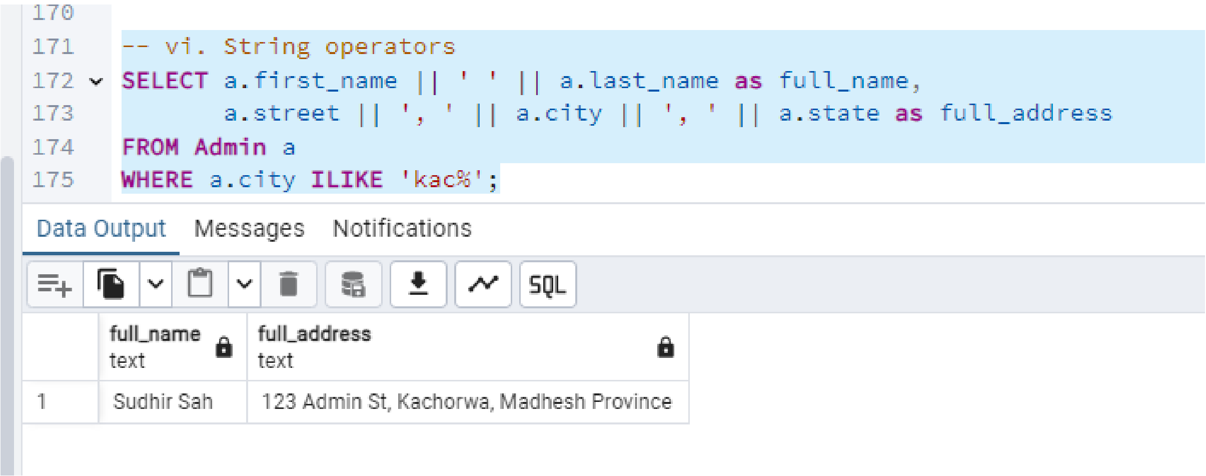
| | medicine_name<br>character varying (255) | order_value<br>numeric | sales_value<br>numeric |
|---|---|---|---|
| 1 | Paracetamol | 3000.00 | 1800.00 |
| 2 | Ibuprofen | 5000.00 | 5000.00 |
| 3 | Amoxicillin | 3600.00 | 1800.00 |
| 4 | Cough Syrup | 1200.00 | 1080.00 |
| 5 | Aspirin | 400.00 | 320.00 |
| 6 | Paracetamol | 3000.00 | 3000.00 |
| 7 | Aspirin | 400.00 | 400.00 |

This query calculates two values for each order: order_value (the total price of the ordered quantity) and sales_value (the total price of the sold quantity).

JOIN between Orders (o) and Medicines (m) to fetch the details of each medicine in the order.

LEFT JOIN between Orders and Sales (s) to calculate the sales value, even for orders that might not have been sold yet (i.e., sales record might be missing for some orders).

### vi. String operators

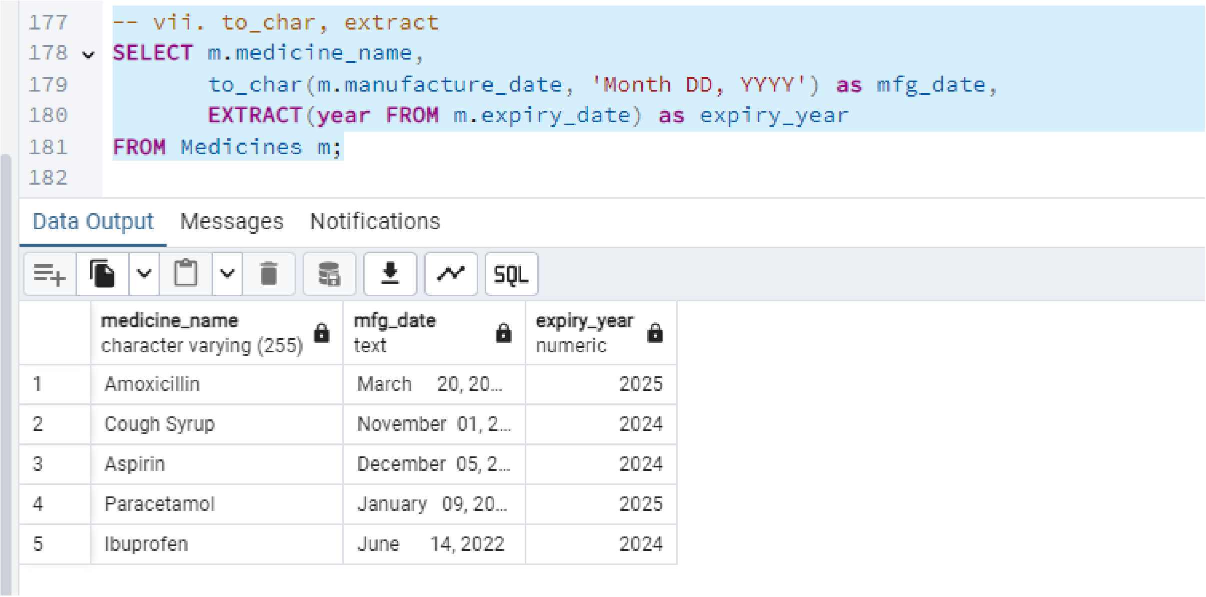


```
-- vi. String operators
SELECT a.first_name || ' ' || a.last_name as full_name,
       a.street || ', ' || a.city || ', ' || a.state as full_address
FROM Admin a
WHERE a.city ILIKE 'kac%';
```

| | full_name<br>text | full_address<br>text |
|---|---|---|
| 1 | Sudhir Sah | 123 Admin St, Kachorwa, Madhesh Province |

This query fetches a concatenation of the first_name and last_name as full_name, and street, city, and state as full_address for all admins whose city starts with "kac" (case-insensitive).

### vii. to_char, extract

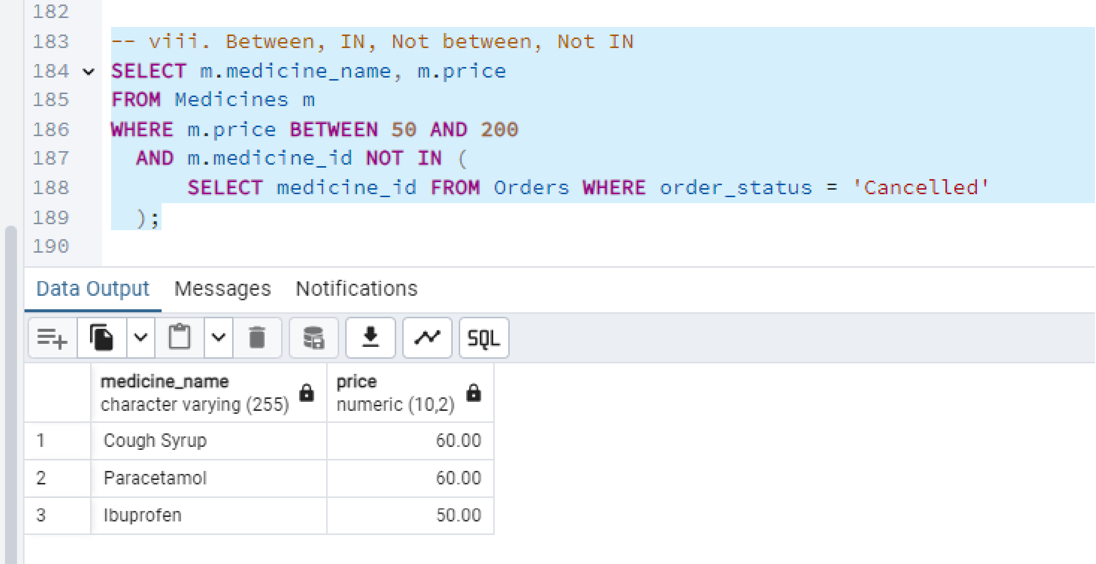


```
-- vii. to_char, extract
SELECT m.medicine_name,
       to_char(m.manufacture_date, 'Month DD, YYYY') as mfg_date,
       EXTRACT(year FROM m.expiry_date) as expiry_year
FROM Medicines m;
```

| | medicine_name<br>character varying (255) | mfg_date<br>text | expiry_year<br>numeric |
|---|---|---|---|
| 1 | Amoxicillin | March 20, 20... | 2025 |
| 2 | Cough Syrup | November 01, 2... | 2024 |
| 3 | Aspirin | December 05, 2... | 2024 |
| 4 | Paracetamol | January 09, 20... | 2025 |
| 5 | Ibuprofen | June 14, 2022 | 2024 |

This query formats the manufacture_date of each medicine as a string (e.g., "January 01, 2023") and extracts the year from the expiry_date.

### viii. Between, IN, Not between, Not IN

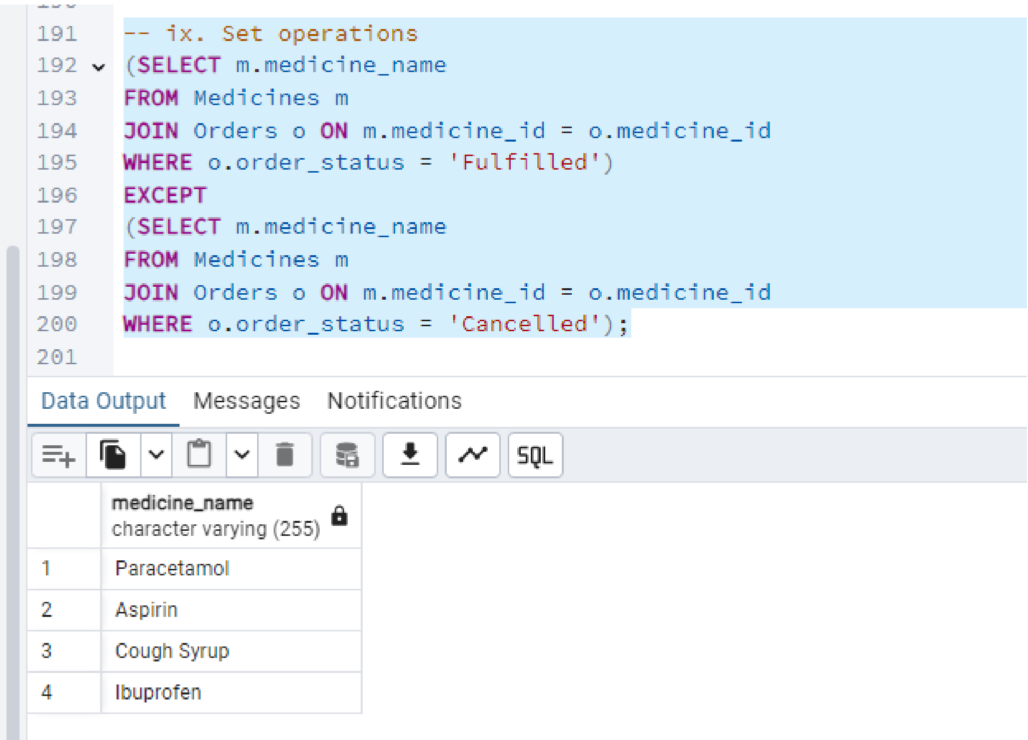


```sql
-- viii. Between, IN, Not between, Not IN
SELECT m.medicine_name, m.price
FROM Medicines m
WHERE m.price BETWEEN 50 AND 200
  AND m.medicine_id NOT IN (
      SELECT medicine_id FROM Orders WHERE order_status = 'Cancelled'
  );
```

| | medicine_name<br>character varying (255) | price<br>numeric (10,2) |
|---|---|---|
| 1 | Cough Syrup | 60.00 |
| 2 | Paracetamol | 60.00 |
| 3 | Ibuprofen | 50.00 |

This query filters medicines that have a price between 50 and 200 and were not 'Cancelled'.

### ix. Set operations

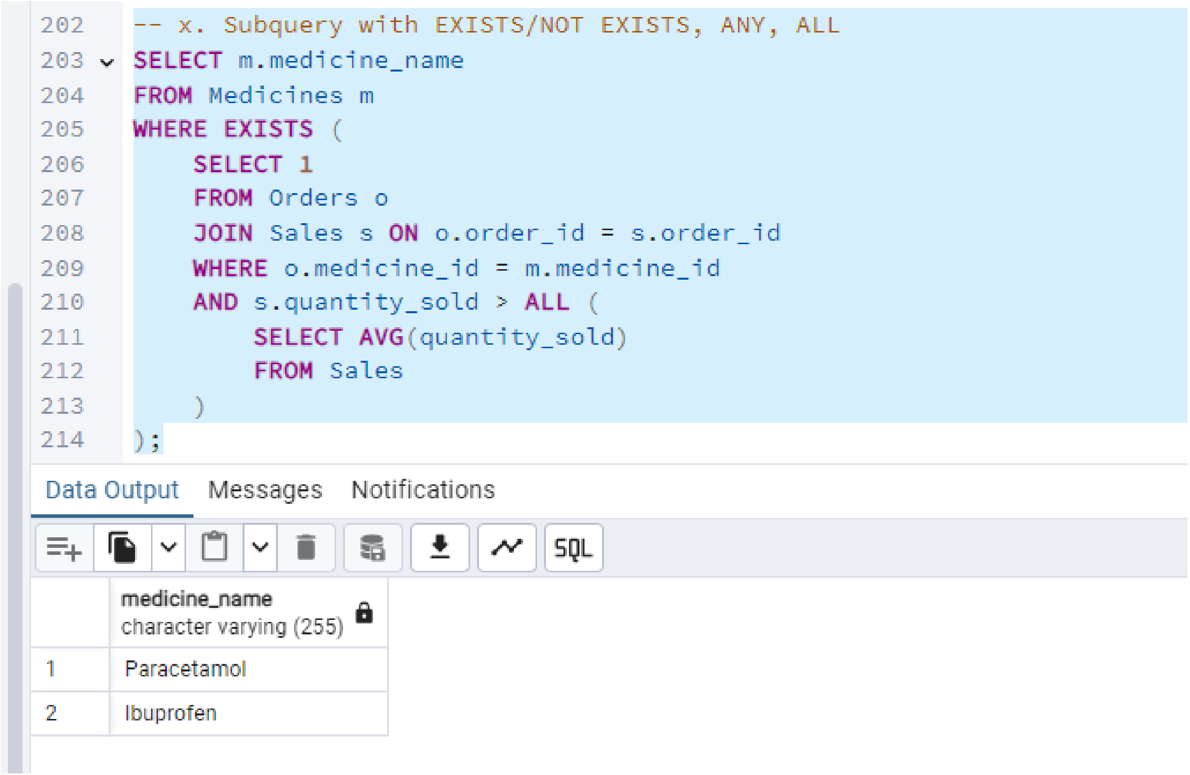


```sql
-- ix. Set operations
(SELECT m.medicine_name
FROM Medicines m
JOIN Orders o ON m.medicine_id = o.medicine_id
WHERE o.order_status = 'Fulfilled')
EXCEPT
(SELECT m.medicine_name
FROM Medicines m
JOIN Orders o ON m.medicine_id = o.medicine_id
WHERE o.order_status = 'Cancelled');
```

| | medicine_name<br>character varying (255) |
|---|---|
| 1 | Paracetamol |
| 2 | Aspirin |
| 3 | Cough Syrup |
| 4 | Ibuprofen |

This query returns the list of medicines that were successfully fulfilled in orders but were not cancelled in any orders.

## x. Subquery with EXISTS/NOT EXISTS, ANY, ALL



```sql
-- x. Subquery with EXISTS/NOT EXISTS, ANY, ALL
SELECT m.medicine_name
FROM Medicines m
WHERE EXISTS (
    SELECT 1
    FROM Orders o
    JOIN Sales s ON o.order_id = s.order_id
    WHERE o.medicine_id = m.medicine_id
    AND s.quantity_sold > ALL (
        SELECT AVG(quantity_sold)
        FROM Sales
    )
);
```

|  | medicine_name<br>character varying (255) |
|---|---|
| 1 | Paracetamol |
| 2 | Ibuprofen |

This query returns the medicines whose sold quantities exceed the average quantity sold across all sales records.