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.
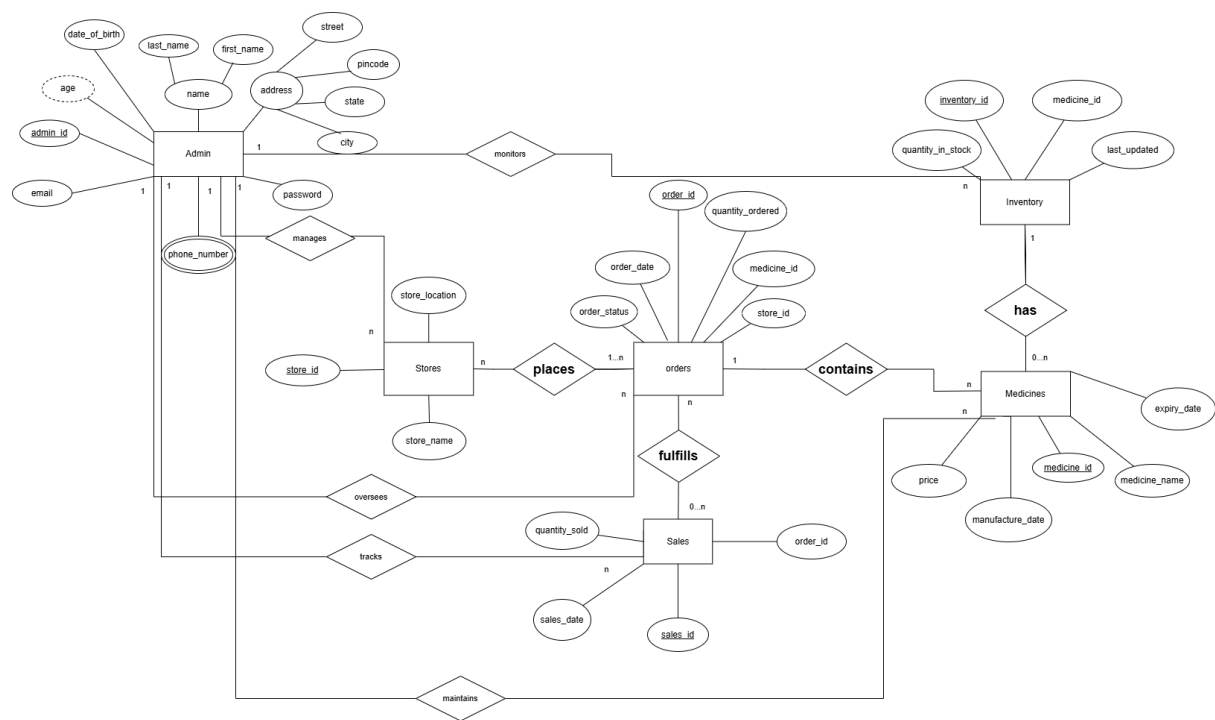
To manage the quantity of medicines, we introduced an **Inventory** entity that tracks the **quantity in stock** for every medicine, along with the **last updated** timestamp to ensure accurate inventory records. The inventory system is directly tied to the medicines, enabling real-time tracking of stock 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."
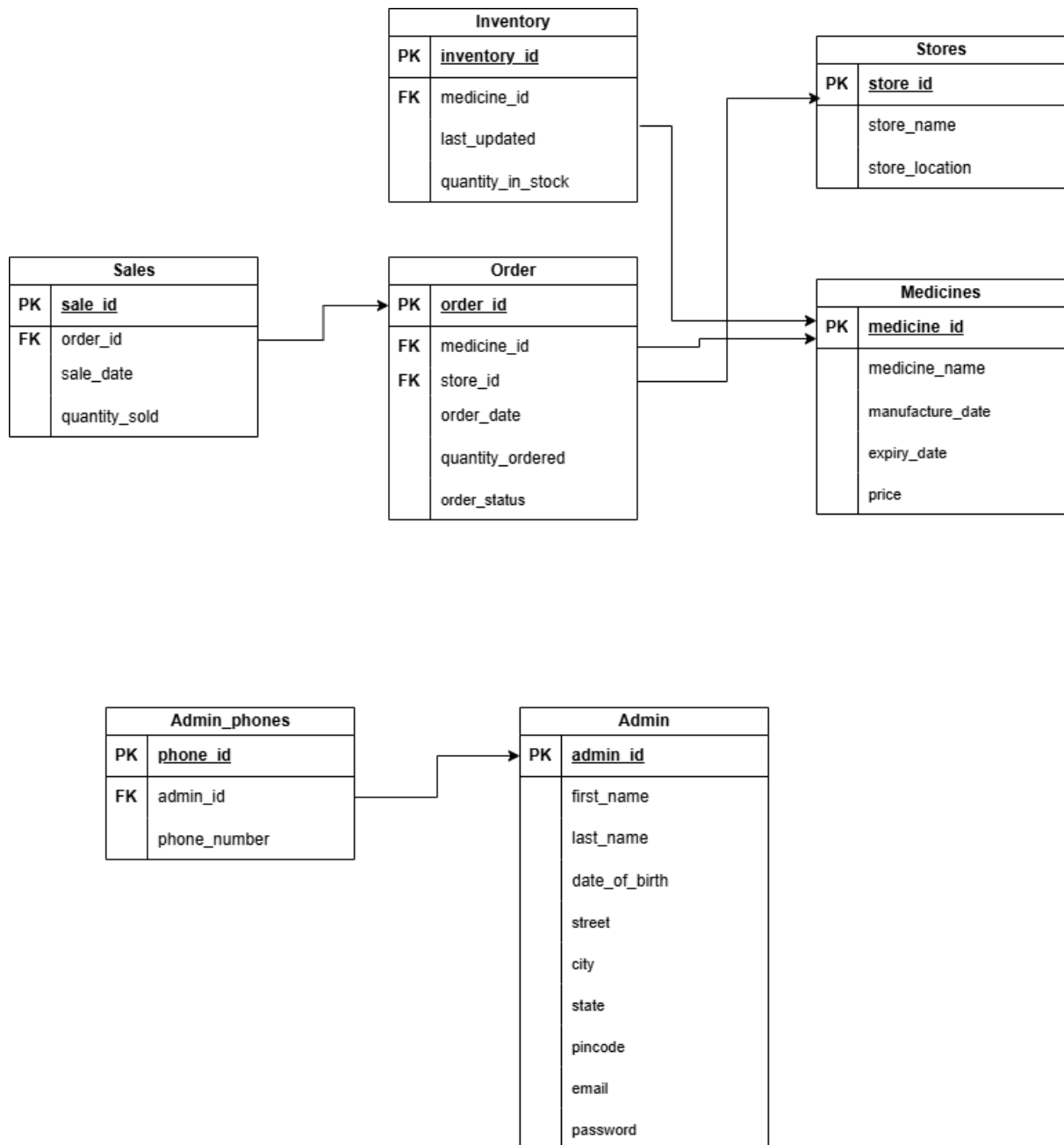
Finally, the **Sales** entity was implemented to track the sale of medicines at the stores. This entity records information such as the **sale_id**, **order_id**, **sale_date**, and **quantity_sold**. This allows the pharmaceutical company to monitor how much of each medicine has been sold and provides 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.
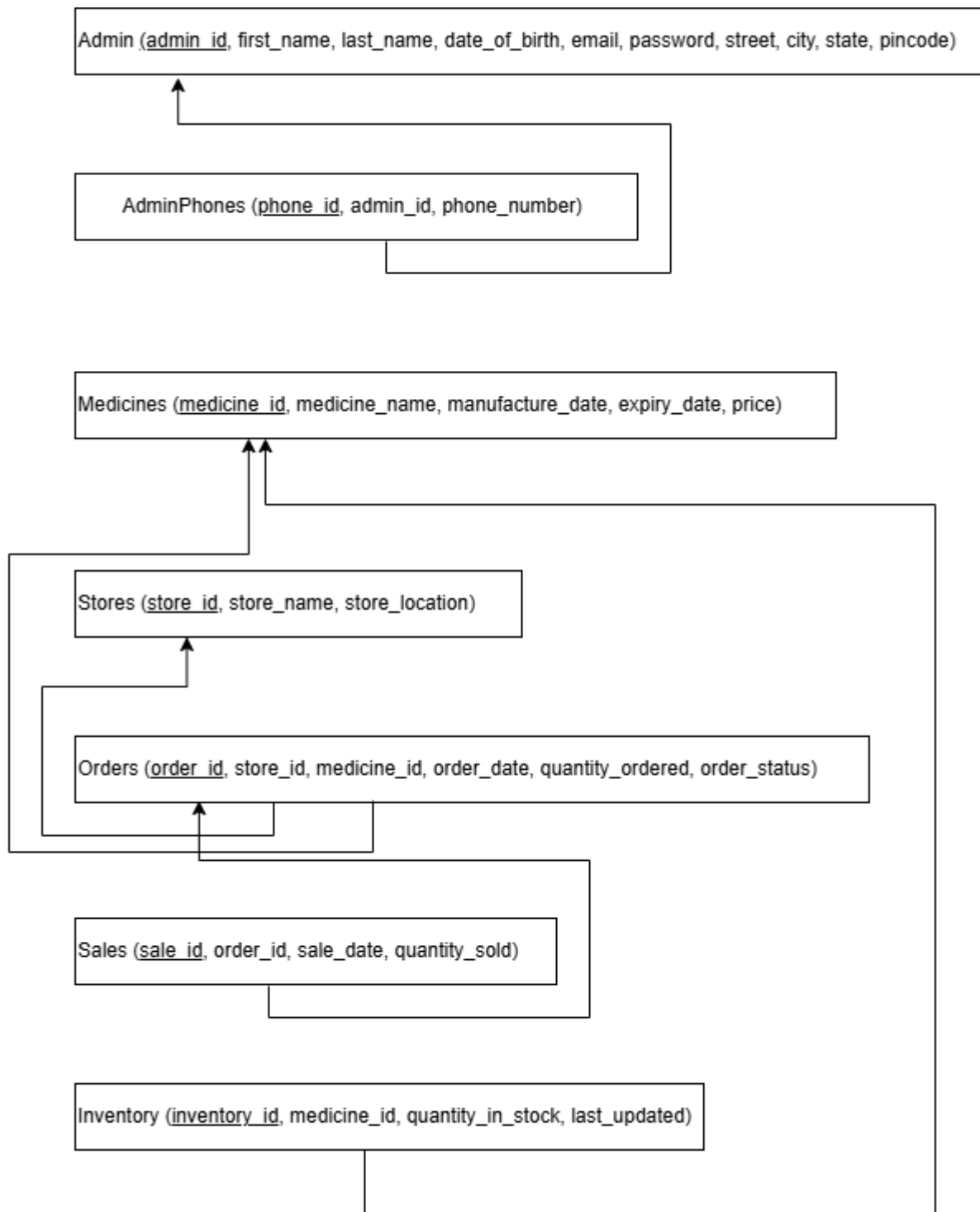
ER Diagram

Schema Diagram

Inventory

PK	inventory_id
FK	medicine_id
	last_updated
	quantity_in_stock

Stores

PK	store_id
	store_name
	store_location

Sales

PK	sale_id
FK	order_id
	sale_date
	quantity_sold

Order

PK	order_id
FK	medicine_id
FK	store_id
	order_date
	quantity_ordered
	order_status

Medicines

PK	medicine_id
	medicine_name
	manufacture_date
	expiry_date
	price

Admin_phones

PK	phone_id
FK	admin_id
	phone_number

Admin

PK	admin_id
	first_name
	last_name
	date_of_birth
	street
	city
	state
	pincode
	email
	password

Relational Schema Diagram

Admin (<u>admin_id</u>, first_name, last_name, date_of_birth, email, password, street, city, state, pincode)

AdminPhones (<u>phone_id</u>, admin_id, phone_number)

Medicines (<u>medicine_id</u>, medicine_name, manufacture_date, expiry_date, price)

Stores (<u>store_id</u>, store_name, store_location)

Orders (<u>order_id</u>, store_id, medicine_id, order_date, quantity_ordered, order_status)

Sales (<u>sale_id</u>, order_id, sale_date, quantity_sold)

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

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

AdminPhones (<u>admin_id</u>, <u>phone_number</u>)

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 (<u>admin_id</u>, first_name, last_name, date_of_birth, email, password, age, street, city, state, pincode)

    ii.   AdminPhones (<u>admin_id</u>, <u>phone_number</u>)

    iii.   Medicines (<u>medicine_id</u>, medicine_name, manufacture_date, expiry_date, price)

    iv.   Stores (<u>store_id</u>, store_name, store_location)

    v.   Orders (<u>order_id</u>, store_id, medicine_id, order_date, quantity_ordered, order_status)

    vi.   Sales (<u>sale_id</u>, order_id, sale_date, quantity_sold)

    vii.   Inventory (<u>inventory_id</u>, 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 (<u>admin_id</u>, first_name, last_name, date_of_birth, email, password, street, city, state, pincode)

    ii.   AdminPhones (<u>phone_id</u>, admin_id, phone_number)

    iii.   Medicines (<u>medicine_id</u>, medicine_name, manufacture_date, expiry_date, price)

    iv.   Stores (<u>store_id</u>, store_name, store_location)

    v.   Orders (<u>order_id</u>, store_id, medicine_id, order_date, quantity_ordered, order_status)

    vi.   Sales (<u>sale_id</u>, order_id, sale_date, quantity_sold)

    vii.   Inventory (<u>inventory_id</u>, 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
```

```sql
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 (
```

```sql
    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')

    )

);
```

```sql
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

```sql
-- 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');
```

```sql
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

```
134
135    -- i. Aggregate functions, Group by...having
136  ✓ SELECT s.store_name, SUM(o.quantity_ordered) AS total_ordered
137    FROM Orders o
138    JOIN Stores s ON o.store_id = s.store_id
139    GROUP BY s.store_name
140    HAVING SUM(o.quantity_ordered) > 10;
141
```

Data Output Messages Notifications

	store_name character varying (255)	total_ordered 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

```
141
142    -- ii. Order by
143  ✓ SELECT medicine_name, price
144    FROM Medicines
145    ORDER BY price DESC;
146
```

Data Output Messages Notifications

	medicine_name character varying (255)	price 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

```
147   -- iii. Join, Outer Join
148 v SELECT o.order_id, s.store_name, m.medicine_name, o.quantity_ordered
149   FROM Orders o
150   JOIN Stores s ON o.store_id = s.store_id
151   JOIN Medicines m ON o.medicine_id = m.medicine_id;
152
153 v SELECT s.store_name, o.order_id, o.order_date
154   FROM Stores s
155   LEFT JOIN Orders o ON s.store_id = o.store_id;
156
```

Data Output   Messages   Notifications

	order_id integer	store_name character varying (255)	medicine_name character varying (255)	quantity_ordered 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.

```
147   -- iii. Join, Outer Join
148 v SELECT o.order_id, s.store_name, m.medicine_name, o.quantity_ordered
149   FROM Orders o
150   JOIN Stores s ON o.store_id = s.store_id
151   JOIN Medicines m ON o.medicine_id = m.medicine_id;
152
153 v SELECT s.store_name, o.order_id, o.order_date
154   FROM Stores s
155   LEFT JOIN Orders o ON s.store_id = o.store_id;
156
```

Data Output   Messages   Notifications

	store_name character varying (255)	order_id integer	order_date 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

```
157    -- iv. Query with Boolean operators
158  ∨ SELECT o.order_id, o.order_date, o.order_status, o.quantity_ordered
159    FROM Orders o
160    WHERE o.order_status IN ('Pending', 'Fulfilled')
161    AND o.quantity_ordered > 20;
162
```

Data Output   Messages   Notifications

	order_id [PK] integer	order_date date	order_status character varying (50)	quantity_ordered 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

```
163    -- v. Query with arithmetic operators
164  ∨ SELECT m.medicine_name,
165           o.quantity_ordered * m.price as order_value,
166           s.quantity_sold * m.price as sales_value
167    FROM Orders o
168    JOIN Medicines m ON o.medicine_id = m.medicine_id
169    LEFT JOIN Sales s ON o.order_id = s.order_id;
170
```

Data Output   Messages   Notifications

	medicine_name character varying (255)	order_value numeric	sales_value 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

```
170
171    -- vi. String operators
172 ∨  SELECT a.first_name || ' ' || a.last_name as full_name,
173           a.street || ', ' || a.city || ', ' || a.state as full_address
174    FROM Admin a
175    WHERE a.city ILIKE 'kac%';
```

Data Output   Messages   Notifications

	full_name 🔒 text	full_address 🔒 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

```
177    -- vii. to_char, extract
178 ∨  SELECT m.medicine_name,
179           to_char(m.manufacture_date, 'Month DD, YYYY') as mfg_date,
180           EXTRACT(year FROM m.expiry_date) as expiry_year
181    FROM Medicines m;
182
```

Data Output   Messages   Notifications

	medicine_name 🔒 character varying (255)	mfg_date 🔒 text	expiry_year 🔒 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

```
182
183    -- viii. Between, IN, Not between, Not IN
184 v  SELECT m.medicine_name, m.price
185    FROM Medicines m
186    WHERE m.price BETWEEN 50 AND 200
187      AND m.medicine_id NOT IN (
188          SELECT medicine_id FROM Orders WHERE order_status = 'Cancelled'
189      );
190
```

Data Output   Messages   Notifications

	medicine_name character varying (255) 🔒	price 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

```
191    -- ix. Set operations
192 v  (SELECT m.medicine_name
193    FROM Medicines m
194    JOIN Orders o ON m.medicine_id = o.medicine_id
195    WHERE o.order_status = 'Fulfilled')
196    EXCEPT
197    (SELECT m.medicine_name
198    FROM Medicines m
199    JOIN Orders o ON m.medicine_id = o.medicine_id
200    WHERE o.order_status = 'Cancelled');
201
```

Data Output   Messages   Notifications

	medicine_name 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

```
202    -- x. Subquery with EXISTS/NOT EXISTS, ANY, ALL
203  ∨ SELECT m.medicine_name
204    FROM Medicines m
205    WHERE EXISTS (
206        SELECT 1
207        FROM Orders o
208        JOIN Sales s ON o.order_id = s.order_id
209        WHERE o.medicine_id = m.medicine_id
210        AND s.quantity_sold > ALL (
211            SELECT AVG(quantity_sold)
212            FROM Sales
213        )
214    );
```

Data Output   Messages   Notifications

	medicine_name character varying (255) 🔒
1	Paracetamol
2	Ibuprofen

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