DRUG DATA METRICS

Preethi Abhilasha Vaddi
SEAS
University at Buffalo
Buffalo,NY,USA
preethia@buffalo.edu

Sai Teja Dondeti
SEAS
University at Buffalo
Buffalo,NY,USA
saitejad@buffalo.edu

Vinitha Vudhayagiri
SEAS
University at Buffalo
Buffalo,NY,USA
vvudhaya@buffalo.edu

Abstract**—** The drugstore system is designed to address the challenges of tracking daily sales data in pharmaceutical stores. By analyzing sales patterns and trends, the system provides valuable insights that enable management to make data-driven decisions to optimize sales and profits. The system allows tracking of drug sales by sellers, stores, locations, and pricing, as well as monitoring sales to different customer segments. With this data, inventory levels can be optimized, pricing can be managed, and supply chain management can be improved. Forecasting sales and planning promotions and marketing campaigns can be done more efficiently. The system also provides insights into sales performance, customer behavior, and market trends, giving the pharmacy a competitive edge in the industry.

I. Introduction

Drug data table contains the information about the store, customers, sellers, orders, products which will help pharmacies track and manage their daily sales data. The database system can provide insights into sales patterns and trends, enabling pharmacy managers to make data-driven decisions to optimize sales and profits.

The system can track sales of different drugs, by different sellers, across different stores and locations, and at different price points. It can also monitor sales to different customer segments, such as new customers or existing customers, and identify high-value customers.

Drug data management can be implemented using database systems as the DB systems can give access to the right user. Using database, we can give the access based on their requirement. Using excel files which this type of data is not safe as anyone can perform the actions on the data and the entire data will be visible to all level of people. By using DB system, we can restrict the access based on their role like customers, store owners.

II. Target Users

A. Users

The users of my database will be pharmacists, salespeople, inventory managers, and office employees at a medical store or pharmacy.

B. Administors of Database

The database will be administered by a specialized IT team. The IT team will be responsible for ensuring the performance, reliability, and security of the database. They will also oversee adding updates and new features as needed.

C. RealLife Scenario

- A pharmacist at a medical store uses the database to store and manage patient information, drug interactions, and dosage recommendations.

- A salesperson at the same store uses the database to track customer orders, sales data, and inventory levels.

- An inventory manager at the store uses the database to track stockouts, purchase orders, and inventory levels.

- An office employee at the store uses the database to manage payroll, financial records, and employee records.

- The IT team at the store manages the database, ensuring its performance, reliability, and security. They also oversee adding updates and new features as needed.

The database helps all these employees perform their jobs more efficiently and effectively. It also helps the store to improve its sales, inventory management, and customer service.

III. Database Design

This database design has 8 tables: store, customer, seller, orders, product, employee, and sale, fact_sales_daily.

A. Constarints

The Store table stores information about stores. This information includes the store's ID, name, address, city, country, and phone number. The Store table is a critical part of any retail or e-commerce database, as it provides a central location for storing and managing store information.

Primary Key: store_id

- **store_id -** a string column that stores the unique identifier for each store.

- **store_name -** a string column that stores the name of each store.

- **address -** a string column that stores the address of each store.

- **city -** a string column that stores the city where the store is located.

- **country -** a string column that stores the country where the store is located.

- **phone_number -** a string column that stores the phone number of each store.

This information can be used for a variety of purposes, such as:

- Tracking sales by location
- Identifying high-traffic stores
- Planning marketing campaigns
- Optimizing delivery routes

The Customer table stores information about customers, such as the customer ID, first name, last name, email address, address, and phone number.

Primary Key: customer_id

- **customer_id** - a string column that stores the unique identifier for each customer.

- **first_name** - a string column that stores the first name of each customer.

- **last_name** - a string column that stores the last name of each customer.

- **email** - a string column that stores the email address of each customer.

- **address** - a string column that stores the address of each customer.

- **phone_number** - a string column that stores the phone number of each customer.

This information can be used for a variety of purposes, such as:

- Tracking sales by customer
- Sending marketing emails
- Providing customer support
- Generating reports on customer demographics

The Seller table stores information about sellers, such as the seller ID, name, address, phone number, email, and year of joining.

Primary Key: seller_id

- **seller_id** - a string column that stores the unique identifier for each seller.

- **seller_name** - a string column that stores the name of each seller.

- **address** - a string column that stores the address of each seller.

- **phone_number** - a string column that stores the phone number of each seller.

- **email** - a string column that stores the email id of each seller.

- **joined_year** - a numeric column that stores the year when the seller joined the company.

This information can be used for a variety of purposes, such as:

- Tracking sales by seller
- Providing performance reviews
- Providing training and development opportunities
- Generating reports on seller demographics

The Orders table stores information about orders, such as the order ID, type of order, payment method, quantity, mode of delivery, and delivery partner.

Primary Key: order_id

- **order_id** - a string column that stores the unique identifier for each order.

- **type_of_order** - a string column that stores the type of the order.

- **payment_method** - a string column that stores the mode of payment used by the customer.

- **quantity** - a numeric column that stores the quantity of items ordered.

- **mode_of_delivery** - a string column that stores the mode of delivery.

- **delivery_partner** - a string column that stores the name of the delivery partner.

This information can be used for a variety of purposes, such as:

- Tracking sales
- Managing inventory
- Shipping orders
- Billing customers
- Providing customer support

The Product table stores information about products, such as the product ID, name, brand, price, quantity left in stock, manufacturer date, expiry date, manufacturer, and the disease the product is meant to treat.

Primary Key: product_id

- **product_id** - a string column that stores the unique identifier for each product.
- **name** - a string column that stores the name of each product.
- **brand** - a string column that stores the brand of the product.
- **price** - a numeric column that stores the price of the product.
- **qty_left_in_stock** - a numeric column that stores the quantity of the product left in stock.
- **manf_date** - a date column that stores the manufacturing date of the product.
- **exp_date** - a date column that stores the expiry date of the product.
- **manufacturer** - a string column that stores the manufacturer details.
- **disease** - a string column that stores the disease for which the product can be used.

This information can be used for a variety of purposes, such as:

- Tracking sales
- Managing inventory
- Promoting products
- Generating reports

The Employee table stores information about employees, such as the employee ID, first name, last name, username, address, and salary.

Primary Key: employee_id

- **employee_id** - a string column that stores the unique identifier for each employee.
- **first_name** - a string column that stores the first name of each employee.
- **last_name** - a string column that stores the last name of each employee.
- **username** - a string column that stores the username of each employee.
- **address** - a string column that stores the address of each employee.
- **salary** - a numeric column that stores the salary of each employee.

This information can be used for a variety of purposes, such as:

- Tracking employee performance
- Providing benefits
- Managing payroll
- Generating report

The Sale table stores information about sales, such as the sale date, sale ID, product ID, order ID, and price.

Primary Key: sale_id
Foreign key: order_id, product_id

- **sale_date** - a date column that stores the date of the sale.
- **sale_id** - a string column that stores the unique identifier for each sale.
- **order_id** - a string column that stores the ID of the order, it is a foreign key that references the order_id in the Order table.
- **product_id** - a string column that stores the ID of the product, it is a foreign key that references the product_id in the Product table.
- **price** - a numeric column that stores the price of the product.

This information can be used for a variety of purposes, such as:

- Tracking sales
- Managing inventory
- Generating reports

The fact_sales_daily table is the central table in the database. It links all the other tables together, providing a single source of truth for sales data. This makes it easy to track sales performance, identify trends, and make informed decisions about business strategy.
Primary Key: drug_id
Foreign key: customer_id, employee_id, order_id, product_id, sale_id, seller_id, store_id

- **drug_id -** a string column that stores the unique identifier for each drug.

- **sale_id** - a string column that stores the id of the product sold. This is a foreign key that references the sale_id column in the sale table.

- **store_id** - a string column that stores the id of the store. This is a foreign key that references the store_id column in the store table.

- **seller_id** - a string column that stores the id of the seller. This is a foreign key that references the seller_id column in the seller table.

- **product_id** - a string column that stores the id of the product. This is a foreign key that references the product_id column in the product table.

- **order_id** - a string column that stores the id of the order. This is a foreign key that references the order_id column in the order table.

- **employee_id** - a string column that stores the id of the employee. This is a foreign key that references the employee_id column in the employee table.

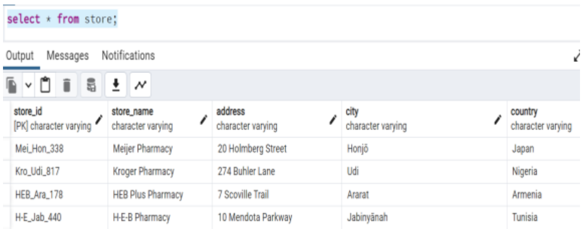
- **customer_id** - a string column that stores the id of the customer. This is a foreign key that references the customer_id column in the customer table.

- **order_date** - a date column that stores the date of the product ordered.

- **price** - a numeric column that stores the price of the product. The value is referenced from the order table.

B. *Tables*

- **STORE TABLE**

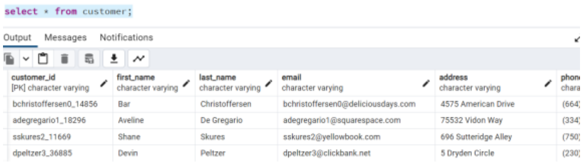


```
select * from store;
```

Output Messages Notifications

store_id [PK] character varying	store_name character varying	address character varying	city character varying	country character varying
Mei_Hon_338	Meijer Pharmacy	20 Holmberg Street	Honjō	Japan
Kro_Udi_817	Kroger Pharmacy	274 Buhler Lane	Udi	Nigeria
HEB_Ara_178	HEB Plus Pharmacy	7 Scoville Trail	Ararat	Armenia
H-E_Jab_440	H-E-B Pharmacy	10 Mendota Parkway	Jabinyānah	Tunisia

- **CUSTOMER TABLE**

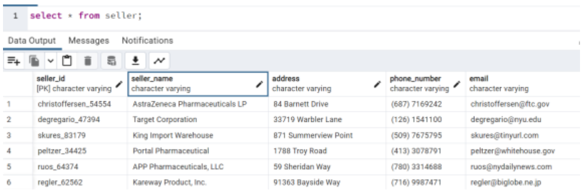


```
select * from customer;
```

Output Messages Notifications

customer_id [PK] character varying	first_name character varying	last_name character varying	email character varying	address character varying	phon chara
bchristoffersen0_14856	Bar	Christoffersen	bchristoffersen0@deliciousdays.com	4575 American Drive	(664)
adegregario1_18296	Aveline	De Gregario	adegregario1@squarespace.com	75532 Vidon Way	(334)
sskures2_11669	Shane	Skures	sskures2@yellowbook.com	696 Sutteridge Alley	(750)
dpeltzer3_36885	Devin	Peltzer	dpeltzer3@clickbank.net	5 Dryden Circle	(230)

- **SELLER TABLE**

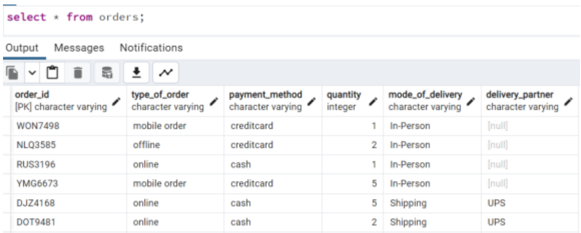


```
select * from seller;
```

Data Output Messages Notifications

	seller_id [PK] character varying	seller_name character varying	address character varying	phone_number character varying	email character varying
1	christoffersen_54554	AstraZeneca Pharmaceuticals LP	84 Barnett Drive	(687) 7169242	christoffersen@ftc.gov
2	degregario_47394	Target Corporation	33719 Warbler Lane	(126) 1541100	degregario@nyu.edu
3	skures_83179	King Import Warehouse	871 Summerview Point	(509) 7675795	skures@tinyurl.com
4	peltzer_34425	Portal Pharmaceutical	1788 Troy Road	(413) 3078791	peltzer@whitehouse.gov
5	ruos_64374	APP Pharmaceuticals, LLC	59 Sheridan Way	(780) 3314688	ruos@nydailynews.com
6	regler_62562	Kareway Product, Inc.	91363 Bayside Way	(716) 9987471	regler@biglobe.ne.jp

- **ORDERS TABLE**

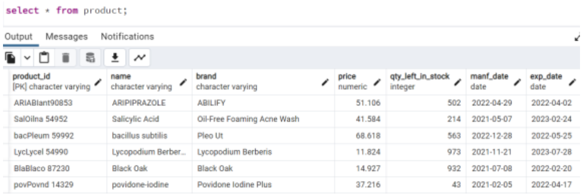


```
select * from orders;
```

Output Messages Notifications

order_id [PK] character varying	type_of_order character varying	payment_method character varying	quantity integer	mode_of_delivery character varying	delivery_partner character varying
WON7498	mobile order	creditcard	1	In-Person	[null]
NLQ3585	offline	creditcard	2	In-Person	[null]
RUS3196	online	cash	1	In-Person	[null]
YMG6673	mobile order	creditcard	5	In-Person	[null]
DJZ4168	online	cash	5	Shipping	UPS
DOT9481	online	cash	2	Shipping	UPS

- **PRODUCT TABLE**

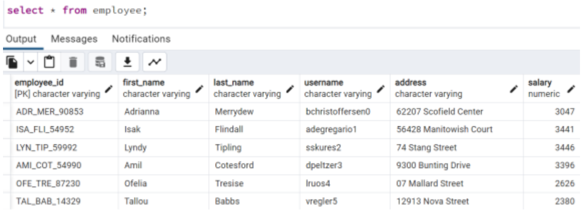


```
select * from product;
```

Output Messages Notifications

product_id [PK] character varying	name character varying	brand character varying	price numeric	qty_left_in_stock integer	manf_date date	exp_date date
ARIABlant90853	ARIPIPRAZOLE	ABILIFY	51.106	502	2022-04-29	2022-04-02
SalOilna 54952	Salicylic Acid	Oil-Free Foaming Acne Wash	41.584	214	2021-05-07	2023-02-24
bacPleum 59992	bacillus subtilis	Pleo Ut	68.618	563	2022-12-28	2022-05-25
LycLycel 54990	Lycopodium Berber...	Lycopodium Berberis	11.824	973	2021-11-21	2023-07-28
BlaBlaco 87230	Black Oak	Black Oak	14.927	932	2021-07-08	2022-02-20
povPovnd 14329	povidone-iodine	Povidone Iodine Plus	37.216	43	2021-02-05	2022-04-17

- **EMPLOYEE TABLE**

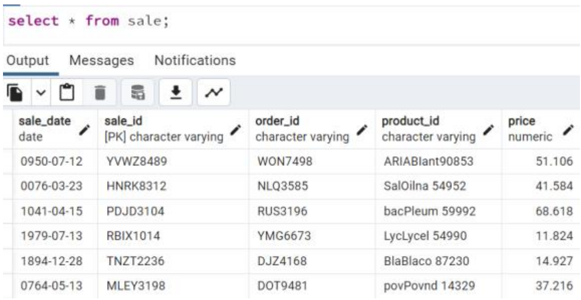


```
select * from employee;
```

Output Messages Notifications

employee_id [PK] character varying	first_name character varying	last_name character varying	username character varying	address character varying	salary numeric
ADR_MER_90853	Adrianna	Merrydew	bchristoffersen0	62207 Scofield Center	3047
ISA_FLI_54952	Isak	Flindall	adegregario1	56428 Manitowish Court	3441
LYN_TIP_59992	Lyndy	Tipling	sskures2	74 Stang Street	3446
AMI_COT_54990	Amil	Cotesford	dpeltzer3	9300 Bunting Drive	3396
OFE_TRE_87230	Ofelia	Tresise	lruos4	07 Mallard Street	2626
TAL_BAB_14329	Tallou	Babbs	vregler5	12913 Nova Street	2380

- **SALE TABLE**

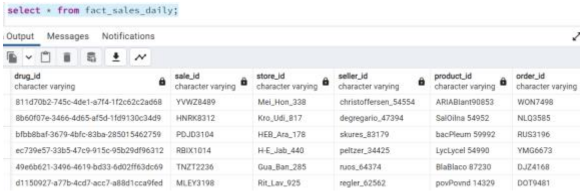


```
select * from sale;
```

Output Messages Notifications

sale_date date	sale_id [PK] character varying	order_id character varying	product_id character varying	price numeric
0950-07-12	YVWZ8489	WON7498	ARIABlant90853	51.106
0076-03-23	HNRK8312	NLQ3585	SalOilna 54952	41.584
1041-04-15	PDJD3104	RUS3196	bacPleum 59992	68.618
1979-07-13	RBIX1014	YMG6673	LycLycel 54990	11.824
1894-12-28	TNZT2236	DJZ4168	BlaBlaco 87230	14.927
0764-05-13	MLEY3198	DOT9481	povPovnd 14329	37.216

- **FACT_SALES_DAILY TABLE**



```
select * from fact_sales_daily;
```

Output Messages Notifications

drug_id character varying	sale_id character varying	store_id character varying	seller_id character varying	product_id character varying	order_id character varying
811d70b2-745c-4de1-a7f4-1f2c62c2ad68	YVWZ8489	Mei_Hon_338	christoffersen_54554	ARIABlant90853	WON7498
8b60f07e-3466-4d65-af5d-1fd9130c34d9	HNRK8312	Kro_Udi_817	degregario_47394	SalOilna 54952	NLQ3585
bfbb8baf-3679-4bfc-83ba-285015462759	PDJD3104	HEB_Ara_178	skures_83179	bacPleum 59992	RUS3196
ec739e57-33b5-47c9-915c-95b29df96312	RBIX1014	H-E_Jab_440	peltzer_34425	LycLycel 54990	YMG6673
49e6b621-3496-4619-bd33-6d02ff63dc69	TNZT2236	Gua_Ban_285	ruos_64374	BlaBlaco 87230	DJZ4168
d1150927-a77b-4cd7-acc7-a88d1cca9fed	MLEY3198	Rit_Lav_925	regler_62562	povPovnd 14329	DOT9481

C. *NOT NULL AND DEFAULT VALUES*

All the columns have not null constraint, no default values are set for any columns in the database.

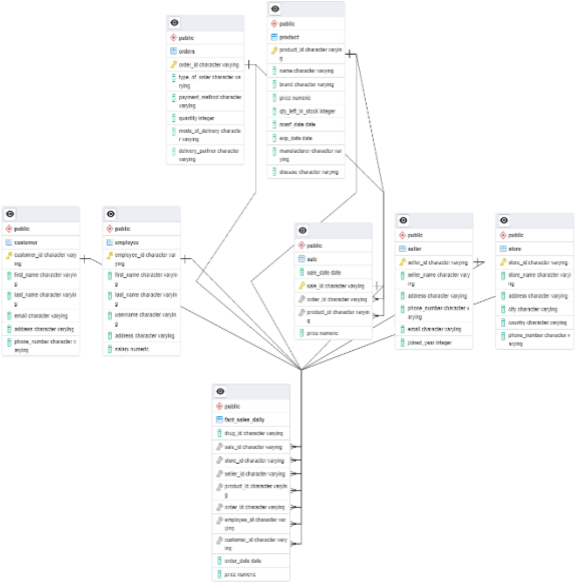
When the primary key is deleted, the following actions will be taken on any foreign key.

NO ACTION The foreign key constraint will not be enforced. The row in the reference table will remain unchanged.

D. ER DIAGRAM

ERD shows the following relationships:

- A customer can have many orders.
- An order can have many products.
- A product can be sold in many stores.
- A seller can sell many products.
- A store can sell many products.


public
orders
order_id character varying
type_of_order character varying
payment_method character varying
quantity integer
mode_of_delivery character varying
delivery_partner character varying
public
product
product_id character varying
name character varying
brand character varying
price numeric
qty_left_in_stock integer
manf_date date
exp_date date
manufacturer character varying
disease character varying
public
customer
customer_id character varying
first_name character varying
last_name character varying
email character varying
address character varying
phone_number character varying
public
employee
employee_id character varying
first_name character varying
last_name character varying
username character varying
address character varying
salary numeric
public
sale
sale_date date
sale_id character varying
order_id character varying
product_id character varying
price numeric
public
seller
seller_id character varying
seller_name character varying
address character varying
phone_number character varying
email character varying
joined_year integer
public
store
store_id character varying
store_name character varying
address character varying
city character varying
country character varying
phone_number character varying
public
fact_sales_daily
drug_id character varying
sale_id character varying
store_id character varying
seller_id character varying
product_id character varying
order_id character varying
employee_id character varying
customer_id character varying
order_date date
price numeric

IV. FUNCTIONAL DEPENDENCIES

A relation is said to be in BCNF if it satisfies1NF, 2NF, 3NF along with BCNF condition. If it does not satisfy any of the condition, we must decompose the relation further to satisfy the BCNF condition.

Customer table:
customer_id → first_name
customer_id → last_name
customer_id → email
customer_id → address
customer_id → phone_number

In the Customer table, the customer_id is the primary key. There are several functional dependencies of the form X -> Y, where customer_id is the super key and Y is a non-key attribute. Specifically, customer_id -> first_name, customer_id -> last_name, customer_id -> email, customer_id -> address, and customer_id -> phone_number. These dependencies satisfy the conditions of BCNF (Boyce-Codd Normal Form).

Employee table:
employee_id → first_name
employee_id → last_name
employee_id → username
employee_id → address
employee_id → salary

The Employee table has several functional dependencies of the form employee_id -> Y, where Y is a non-key attribute. These include employee_id -> first_name, employee_id -> last_name, employee_id -> username, employee_id -> address, and employee_id -> salary. This means that the employee_id is the super key and uniquely determines each of these attributes.

Orders table:
order_id → type_of_order
order_id → payment_method
order_id → quantity
order_id → mode_of_delivery
order_id → delivery_partner

The Orders table has several functional dependencies of the form order_id -> Y, where Y is a non-key attribute. These include order_id -> type_of_order, order_id -> payment_method, order_id -> quantity, order_id -> mode_of_delivery, and order_id -> delivery_partner. This means that the order_id is the super key and uniquely determines each of these attributes.

Product table:
product_id → name
product_id → brand
product_id → price
product_id → qty_left_in_stock
product_id → manf_date
product_id → exp_date
product_id → manufacturer
product_id → disease

The Product table has several functional dependencies of the form product_id -> Y, where Y is a non-key attribute. These include product_id -> name, product_id -> brand, product_id -> price, product_id -> qty_left_in_stock, product_id -> manf_date, product_id -> exp_date, product_id -> manufacturer, and product_id -> disease. This means that the product_id is the super key and uniquely determines each of these attributes.

Sale table:
sale_id → sale_date
sale_id → order_id
sale_id → product_id
sale_id → price

The Sale table has several functional dependencies of the form sale_id -> Y, where Y is a non-key attribute. These include sale_id -> sale_date, sale_id -> order_id, sale_id -> product_id, and sale_id -> price. This means that the sale_id is the super key and uniquely determines each of these attributes.

Seller table:
seller_id → seller_name
seller_id → address
seller_id → phone_number
seller_id → email
seller_id → joined_year

The Seller table has several functional dependencies of the form seller_id -> Y, where Y is a non-key attribute. These include seller_id -> seller_name, seller_id -> address, seller_id -> phone_number, seller_id -> email, and seller_id -> joined_year. This means that the seller_id is the super key and uniquely determines each of these attributes.

Store table:
store_id → store_name
store_id → address
store_id → city
store_id → country
store_id → phone_number

The Store table has several functional dependencies of the form store_id -> Y, where Y is a non-key attribute. These include store_id -> store_name, store_id -> address, store_id -> city, store_id -> country, and store_id -> phone_number. This means that the store_id is the super key and uniquely determines each of these attributes.

The fact_sales_daily table is in third normal form (3NF) because the columns product_id and quantity are not functionally dependent on the sale_id column. Instead, they are foreign keys referencing other tables, and therefore not prime attributes. Specifically, the product_id column refers to the Product table, and the quantity column refers to the Order table. By eliminating this dependency on non-key attributes, the fact_sales_daily table avoids certain types of data anomalies that can occur in lower normal forms.

V. CREATION AND LOADING OF DATA

We have loaded the downloaded data, which was generated using the Mockaroo site for random data generation. To do this, we used two separate files: create.sql and load.sql. The create.sql file contains the schema definition for the tables, while the load.sql file contains the random data that was generated from the site. We are now performing various operations on this data, such as insert, update, view, and delete.

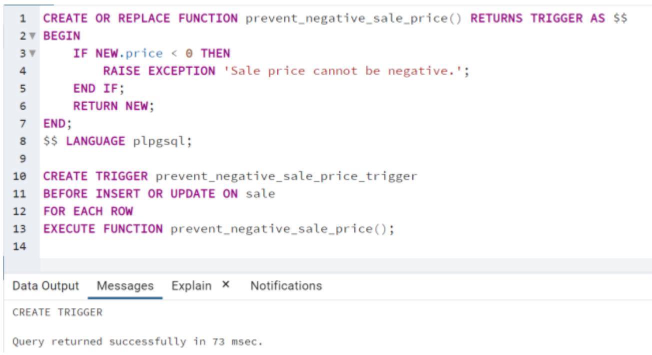
Mockaroo

table	row_count
customer	25000
employee	25000
fact_sales_daily	24990
seller	25000
sale	24990
store	25000
product	25000
orders	25000

VI. TRIGGERS

Triggers are a type of stored procedure that is automatically executed in response to certain events or changes in the database. Triggers can be defined to execute either before or after an event, such as an insert, update, or delete operation, occurs on a table or view in the database. Triggers can be used to enforce complex data validation rules or to maintain referential integrity between related tables.

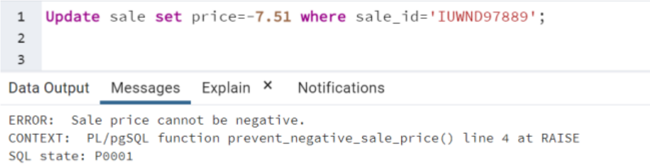
The price of any product must be a positive value. If a negative value is entered in the price field during insertion or updating of product details, a trigger will be executed. This trigger will generate an error message stating that the sale price cannot be negative.



```
CREATE OR REPLACE FUNCTION prevent_negative_sale_price() RETURNS TRIGGER AS $$
BEGIN
    IF NEW.price < 0 THEN
        RAISE EXCEPTION 'Sale price cannot be negative.';
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER prevent_negative_sale_price_trigger
BEFORE INSERT OR UPDATE ON sale
FOR EACH ROW
EXECUTE FUNCTION prevent_negative_sale_price();
```

```
CREATE TRIGGER

Query returned successfully in 73 msec.
```

Suppose we attempt to update the sale price of a product to a negative value. In this case, the trigger that we have set up will be executed and an error will be raised. The error message will state that the sale price cannot be negative.



```
Update sale set price=-7.51 where sale_id='IUWND97889';
```

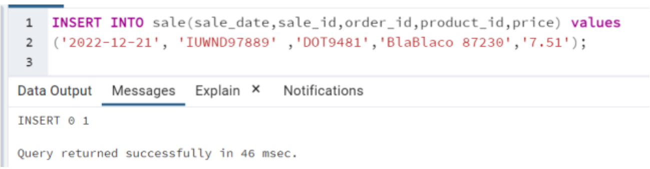
```
ERROR:  Sale price cannot be negative.
CONTEXT:  PL/pgSQL function prevent_negative_sale_price() line 4 at RAISE
SQL state: P0001
```

VII. SQL QUERIES

We will now run several queries on our database, including insert, update, delete, and select operations.

A. *Inserting*

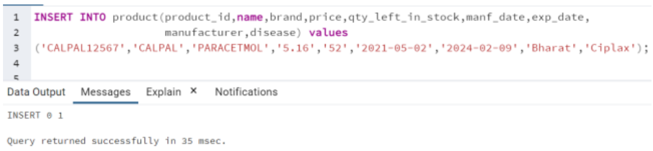
1. Inserting into Sale Table



```
INSERT INTO sale(sale_date,sale_id,order_id,product_id,price) values
('2022-12-21', 'IUWND97889' ,'DOT9481','BlaBlaco 87230','7.51');
```

```
INSERT 0 1

Query returned successfully in 46 msec.
```

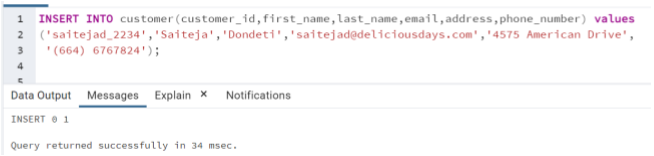
2. Inserting into Product Table



```
INSERT INTO product(product_id,name,brand,price,qty_left_in_stock,manf_date,exp_date,
                    manufacturer,disease) values
('CALPAL12567','CALPAL','PARACETMOL','5.16','52','2021-05-02','2024-02-09','Bharat','Ciplax');
```

```
INSERT 0 1

Query returned successfully in 35 msec.
```

3. Inserting into Customer Table

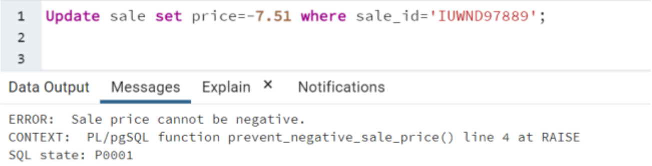


```sql
INSERT INTO customer(customer_id,first_name,last_name,email,address,phone_number) values
('saitejad_2234','Saiteja','Dondeti','saitejad@deliciousdays.com','4575 American Drive',
 '(664) 6767824');
```

```
INSERT 0 1

Query returned successfully in 34 msec.
```

B. Updating

4. Updating sale price for the id “IUWND97889”

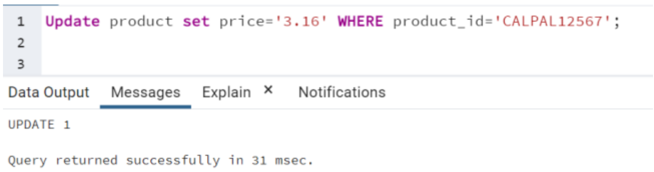


```sql
Update sale set price=-7.51 where sale_id='IUWND97889';
```

```
ERROR:  Sale price cannot be negative.
CONTEXT:  PL/pgSQL function prevent_negative_sale_price() line 4 at RAISE
SQL state: P0001
```

The trigger was activated when we tried to execute the above query, as the price value was negative.

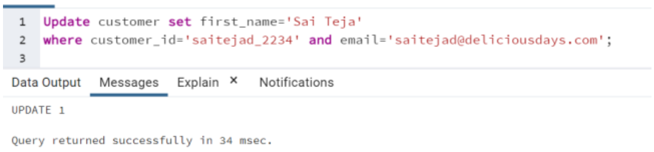
5. Updating product price for the id “CALPAL12567”



```sql
Update product set price='3.16' WHERE product_id='CALPAL12567';
```

```
UPDATE 1

Query returned successfully in 31 msec.
```

6. Updating the Customer First Name

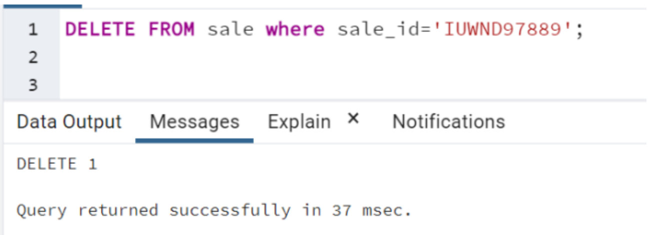


```sql
Update customer set first_name='Sai Teja'
where customer_id='saitejad_2234' and email='saitejad@deliciousdays.com';
```

```
UPDATE 1

Query returned successfully in 34 msec.
```

C. Deleting

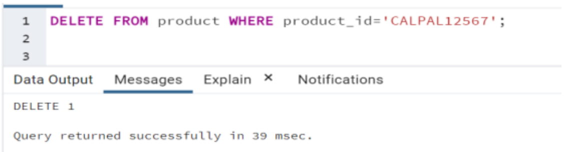
7. Here we are going to Delete from sale table with sale id “IUWND97889.”



```sql
DELETE FROM sale where sale_id='IUWND97889';
```

```
DELETE 1

Query returned successfully in 37 msec.
```

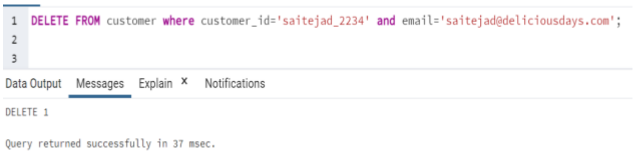
8. Here we are going to Delete from product table with product _id “CALPAL12567”.



```sql
DELETE FROM product WHERE product_id='CALPAL12567';
```

```
DELETE 1

Query returned successfully in 39 msec.
```

9. Deleting the details from customer table

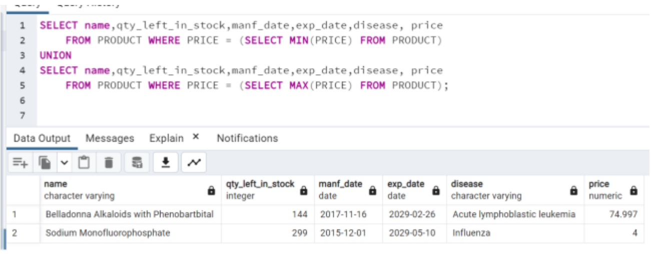


```sql
DELETE FROM customer where customer_id='saitejad_2234' and email='saitejad@deliciousdays.com';
```

```
DELETE 1

Query returned successfully in 37 msec.
```

D. General Queries

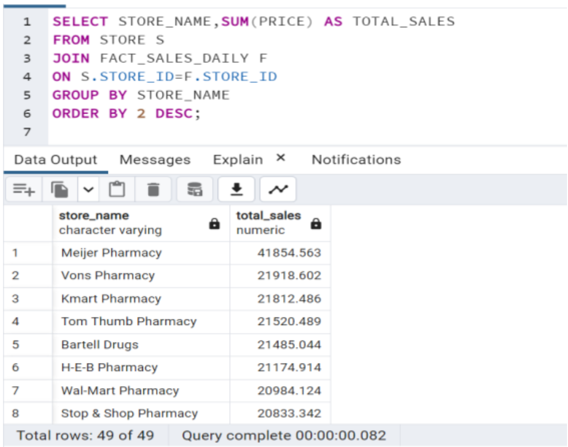
10. To get the information about products with the minimum and maximum prices from the PRODUCT table.



```sql
SELECT name,qty_left_in_stock,manf_date,exp_date,disease, price
    FROM PRODUCT WHERE PRICE = (SELECT MIN(PRICE) FROM PRODUCT)
UNION
SELECT name,qty_left_in_stock,manf_date,exp_date,disease, price
    FROM PRODUCT WHERE PRICE = (SELECT MAX(PRICE) FROM PRODUCT);
```

	name character varying	qty_left_in_stock integer	manf_date date	exp_date date	disease character varying	price numeric
1	Belladonna Alkaloids with Phenobartbital	144	2017-11-16	2029-02-26	Acute lymphoblastic leukemia	74.997
2	Sodium Monofluorophosphate	299	2015-12-01	2029-05-10	Influenza	4

11. To retrieve the total sales for each store and printing the results in descending order

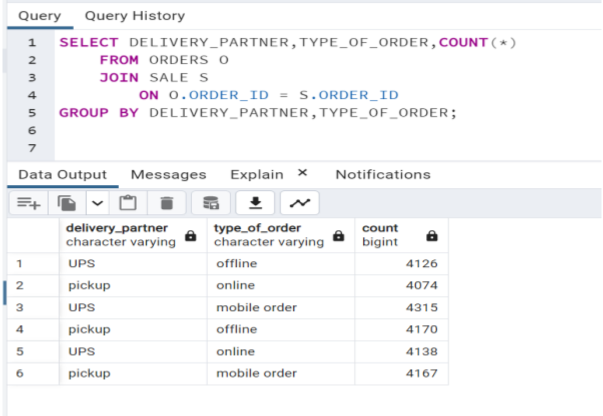


```sql
SELECT STORE_NAME,SUM(PRICE) AS TOTAL_SALES
FROM STORE S
JOIN FACT_SALES_DAILY F
ON S.STORE_ID=F.STORE_ID
GROUP BY STORE_NAME
ORDER BY 2 DESC;
```

	store_name character varying	total_sales numeric
1	Meijer Pharmacy	41854.563
2	Vons Pharmacy	21918.602
3	Kmart Pharmacy	21812.486
4	Tom Thumb Pharmacy	21520.489
5	Bartell Drugs	21485.044
6	H-E-B Pharmacy	21174.914
7	Wal-Mart Pharmacy	20984.124
8	Stop & Shop Pharmacy	20833.342

Total rows: 49 of 49   Query complete 00:00:00.082

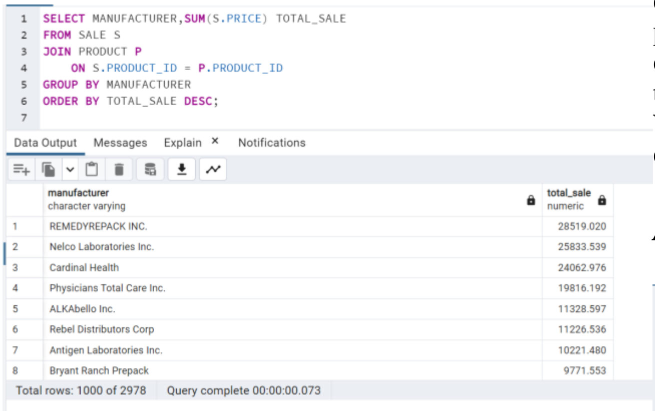
12. We are getting the count of orders grouped by delivery partner and type of order.



```sql
SELECT DELIVERY_PARTNER,TYPE_OF_ORDER,COUNT(*)
     FROM ORDERS O
     JOIN SALE S
          ON O.ORDER_ID = S.ORDER_ID
GROUP BY DELIVERY_PARTNER,TYPE_OF_ORDER;
```

	delivery_partner character varying	type_of_order character varying	count bigint
1	UPS	offline	4126
2	pickup	online	4074
3	UPS	mobile order	4315
4	pickup	offline	4170
5	UPS	online	4138
6	pickup	mobile order	4167

13. To get the total sales for each manufacturer in the SALE and PRODUCT tables, sorted in descending order by the total sale amount.

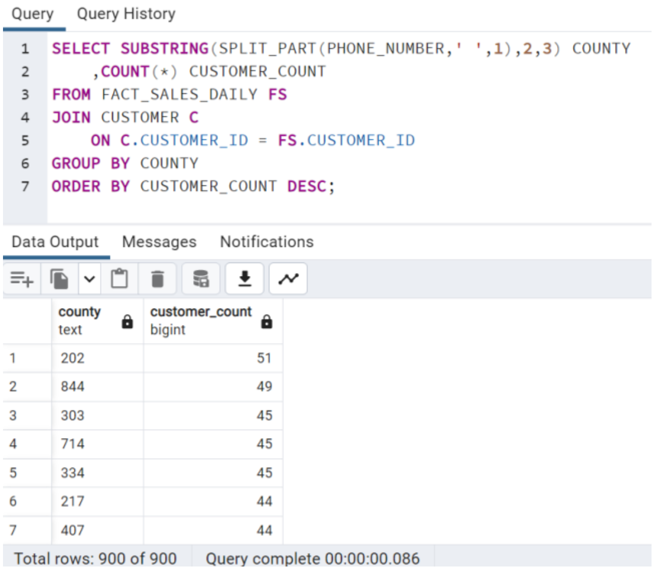


```sql
SELECT MANUFACTURER,SUM(S.PRICE) TOTAL_SALE
FROM SALE S
JOIN PRODUCT P
    ON S.PRODUCT_ID = P.PRODUCT_ID
GROUP BY MANUFACTURER
ORDER BY TOTAL_SALE DESC;

```

	manufacturer character varying	total_sale numeric
1	REMEDYREPACK INC.	28519.020
2	Nelco Laboratories Inc.	25833.539
3	Cardinal Health	24062.976
4	Physicians Total Care Inc.	19816.192
5	ALKAbello Inc.	11328.597
6	Rebel Distributors Corp	11226.536
7	Antigen Laboratories Inc.	10221.480
8	Bryant Ranch Prepack	9771.553

Total rows: 1000 of 2978 Query complete 00:00:00.073

14. Counting the number of customers in each county based on their phone numbers.

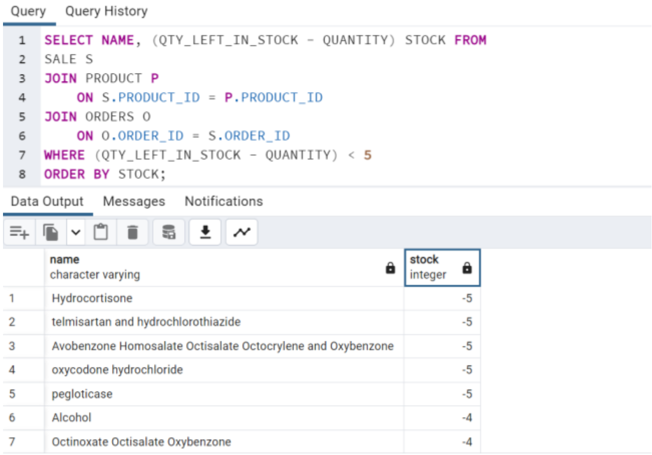


```sql
SELECT SUBSTRING(SPLIT_PART(PHONE_NUMBER,' ',1),2,3) COUNTY
     ,COUNT(*) CUSTOMER_COUNT
FROM FACT_SALES_DAILY FS
JOIN CUSTOMER C
     ON C.CUSTOMER_ID = FS.CUSTOMER_ID
GROUP BY COUNTY
ORDER BY CUSTOMER_COUNT DESC;
```

	county text	customer_count bigint
1	202	51
2	844	49
3	303	45
4	714	45
5	334	45
6	217	44
7	407	44

Total rows: 900 of 900 Query complete 00:00:00.086

15. Getting the name of the product and the quantity left in the stock.



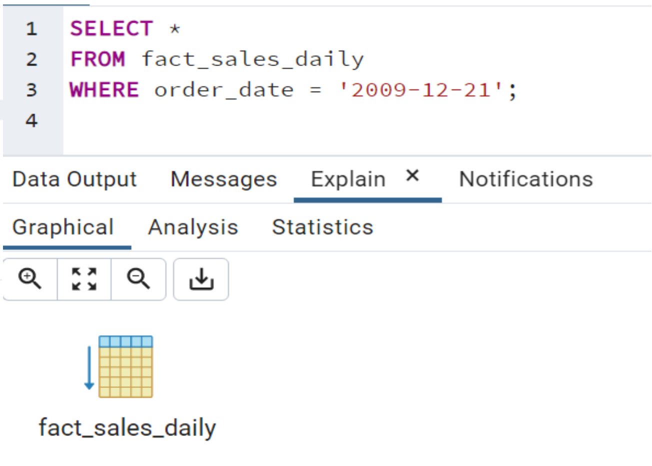
```sql
SELECT NAME, (QTY_LEFT_IN_STOCK - QUANTITY) STOCK FROM
SALE S
JOIN PRODUCT P
    ON S.PRODUCT_ID = P.PRODUCT_ID
JOIN ORDERS O
    ON O.ORDER_ID = S.ORDER_ID
WHERE (QTY_LEFT_IN_STOCK - QUANTITY) < 5
ORDER BY STOCK;
```

	name character varying	stock integer
1	Hydrocortisone	-5
2	telmisartan and hydrochlorothiazide	-5
3	Avobenzone Homosalate Octisalate Octocrylene and Oxybenzone	-5
4	oxycodone hydrochloride	-5
5	pegloticase	-5
6	Alcohol	-4
7	Octinoxate Octisalate Oxybenzone	-4

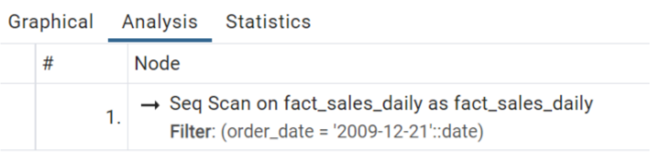
VIII. QUERY OPTIMIZATION

Query optimization is the process of selecting the most efficient query execution plan from among many possible plans that will achieve the desired result. The goal of query optimization is to minimize the time and resources required to process a query, while still producing the correct results. We have done the indexing and subqueries for the optimization.

A. Optmization:1

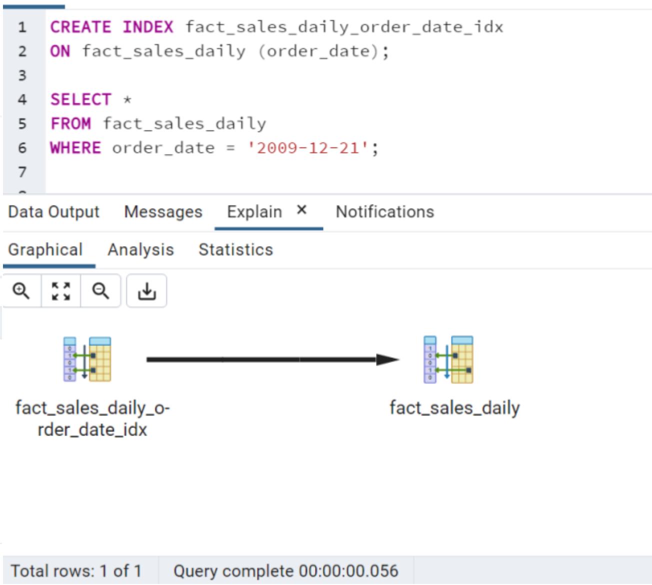


```sql
SELECT *
FROM fact_sales_daily
WHERE order_date = '2009-12-21';

```



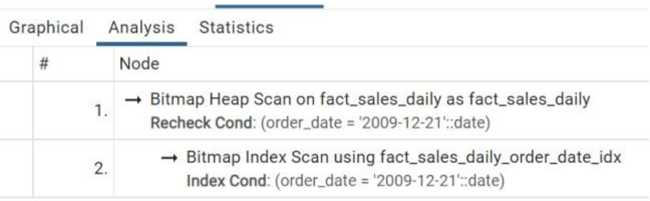
#	Node
1.	→ Seq Scan on fact_sales_daily as fact_sales_daily Filter: (order_date = '2009-12-21'::date)

The query took a total of 92 milliseconds to run. To optimize the query, an index can be created on the order_date column. By doing so, the database will be able to efficiently locate the rows that match the specified date, without having to scan the entire table.



```sql
CREATE INDEX fact_sales_daily_order_date_idx
ON fact_sales_daily (order_date);

SELECT *
FROM fact_sales_daily
WHERE order_date = '2009-12-21';

```

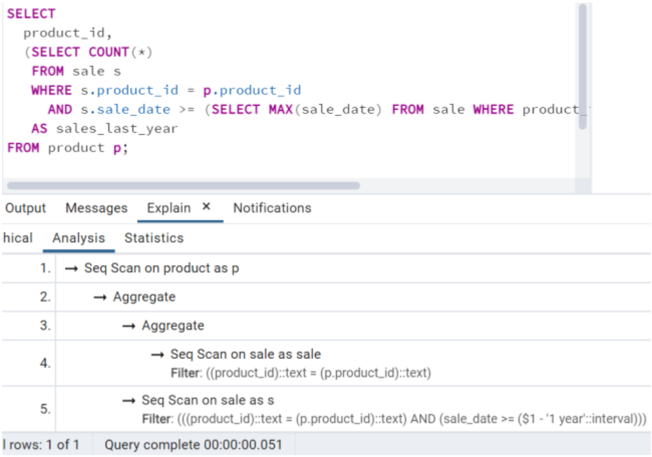
Total rows: 1 of 1 Query complete 00:00:00.056



#	Node
1.	→ Bitmap Heap Scan on fact_sales_daily as fact_sales_daily **Recheck Cond**: (order_date = '2009-12-21'::date)
2.	→ Bitmap Index Scan using fact_sales_daily_order_date_idx **Index Cond**: (order_date = '2009-12-21'::date)

Now that we have optimized the query by creating an index on the order_date column, the execution time has significantly improved, taking only 52 milliseconds.

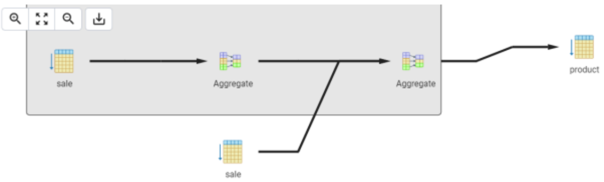
B. Optmization:2



```sql
SELECT
  product_id,
  (SELECT COUNT(*)
   FROM sale s
   WHERE s.product_id = p.product_id
     AND s.sale_date >= (SELECT MAX(sale_date) FROM sale WHERE product_
   AS sales_last_year
FROM product p;
```

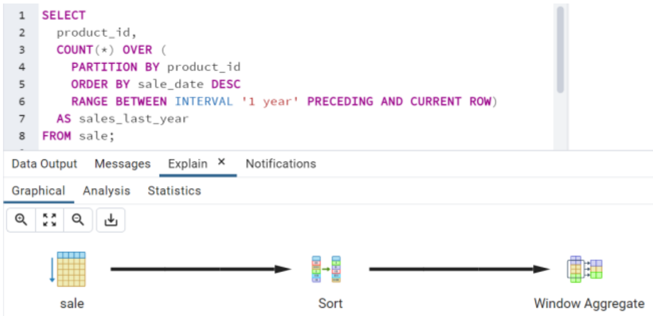
1.	→ Seq Scan on product as p
2.	→ Aggregate
3.	→ Aggregate
4.	→ Seq Scan on sale as sale **Filter**: ((product_id)::text = (p.product_id)::text)
5.	→ Seq Scan on sale as s **Filter**: (((product_id)::text = (p.product_id)::text) AND (sale_date >= ($1 - '1 year'::interval)))

l rows: 1 of 1 Query complete 00:00:00.051

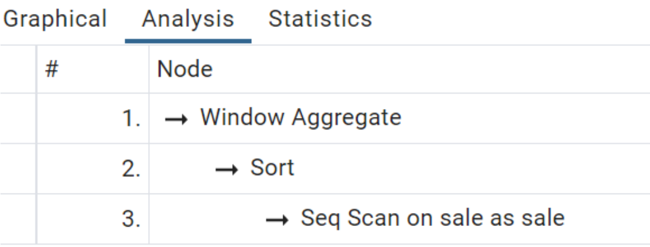


The given query utilizes a correlated subquery to compute the number of sales for each product in the past year. The subquery is correlated to the outer query using the product_id column and incorporates another subquery to determine the maximum sale date for each product.

To improve this query, we can use a window function instead of a correlated subquery. Here's an example:



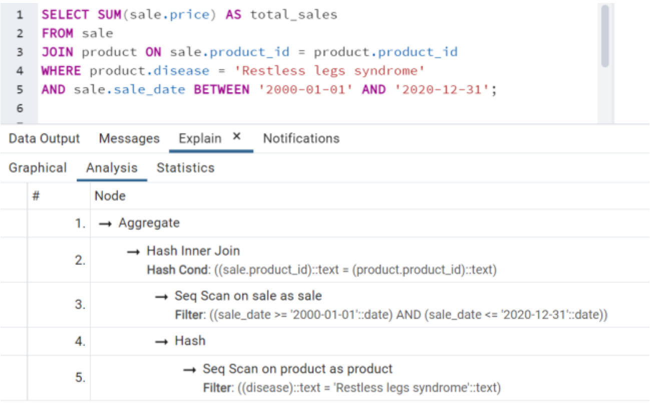
```sql
SELECT
  product_id,
  COUNT(*) OVER (
    PARTITION BY product_id
    ORDER BY sale_date DESC
    RANGE BETWEEN INTERVAL '1 year' PRECEDING AND CURRENT ROW)
  AS sales_last_year
FROM sale;
```



#	Node
1.	→ Window Aggregate
2.	→ Sort
3.	→ Seq Scan on sale as sale

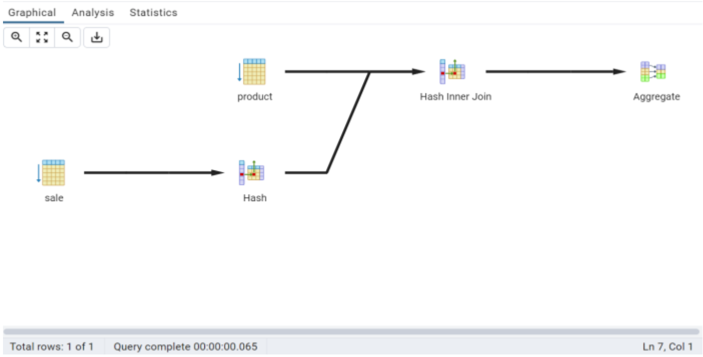
C. Optmization:3

To calculate the total sales for the products that are used to treat "Restless legs syndrome" between the dates '2000-01-01' and '2020-12-31'.



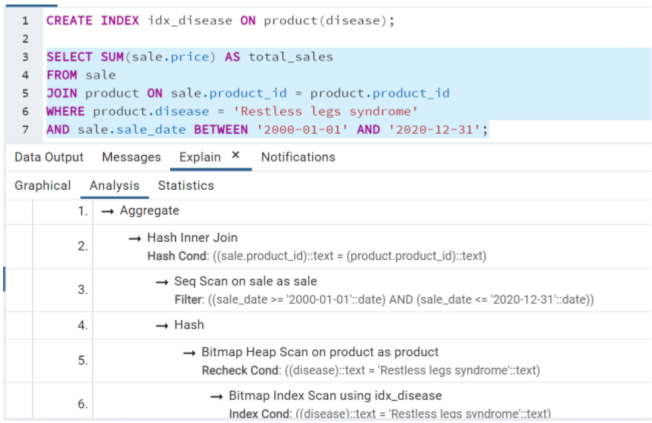
```sql
SELECT SUM(sale.price) AS total_sales
FROM sale
JOIN product ON sale.product_id = product.product_id
WHERE product.disease = 'Restless legs syndrome'
AND sale.sale_date BETWEEN '2000-01-01' AND '2020-12-31';
```

#	Node
1.	→ Aggregate
2.	→ Hash Inner Join **Hash Cond**: ((sale.product_id)::text = (product.product_id)::text)
3.	→ Seq Scan on sale as sale **Filter**: ((sale_date >= '2000-01-01'::date) AND (sale_date <= '2020-12-31'::date))
4.	→ Hash
5.	→ Seq Scan on product as product **Filter**: ((disease)::text = 'Restless legs syndrome'::text)



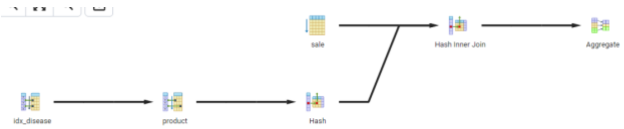
Total rows: 1 of 1 Query complete 00:00:00.065 Ln 7, Col 1

Here we are going to create the index on the disease on the product table for optimizing the query.



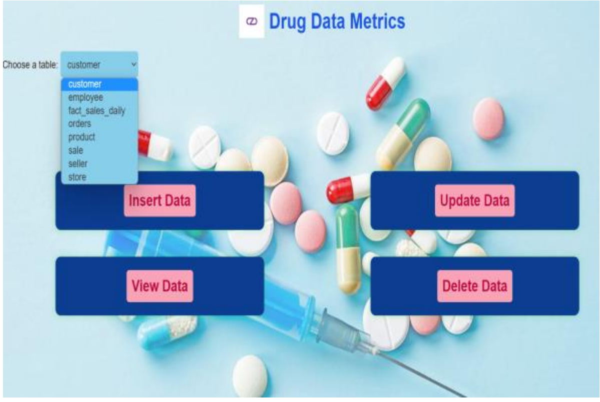
```sql
CREATE INDEX idx_disease ON product(disease);

SELECT SUM(sale.price) AS total_sales
FROM sale
JOIN product ON sale.product_id = product.product_id
WHERE product.disease = 'Restless legs syndrome'
AND sale.sale_date BETWEEN '2000-01-01' AND '2020-12-31';
```

1.	→ Aggregate
2.	→ Hash Inner Join **Hash Cond**: ((sale.product_id)::text = (product.product_id)::text)
3.	→ Seq Scan on sale as sale **Filter**: ((sale_date >= '2000-01-01'::date) AND (sale_date <= '2020-12-31'::date))
4.	→ Hash
5.	→ Bitmap Heap Scan on product as product **Recheck Cond**: ((disease)::text = 'Restless legs syndrome'::text)
6.	→ Bitmap Index Scan using idx_disease **Index Cond**: ((disease)::text = 'Restless legs syndrome'::text)

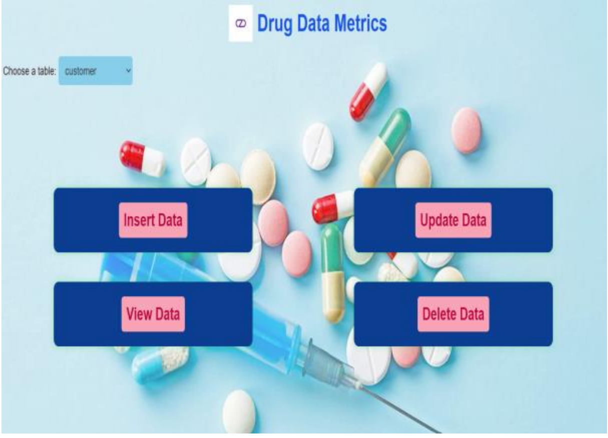


IX. Running website

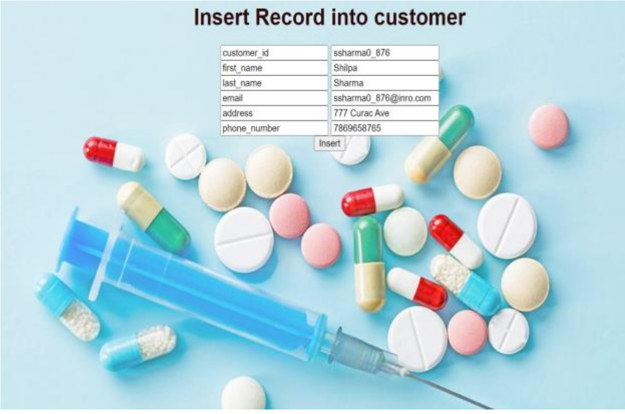
Below is the main page of our application.



On the main page of our application, there are five options. We need to select one of the eight tables from our database first before we can perform any operation. Once we've chosen a table, we can then choose from four options: insert, update, view, or delete.



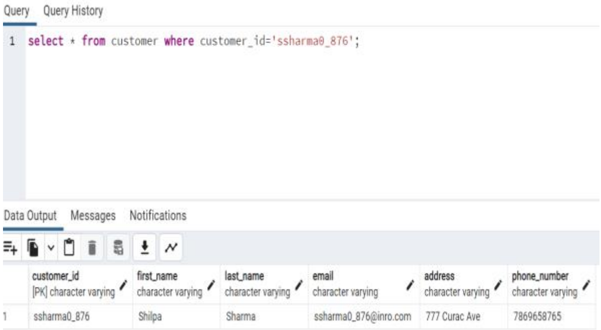
We've chosen the customer table and now we're clicking on the "insert data" tab to input information.



Insert Record into customer

customer_id	ssharma0_876
first_name	Shilpa
last_name	Sharma
email	ssharma0_876@inro.com
address	777 Curac Ave
phone_number	7869658765

To enter data, we need to input information for all columns of the selected table. By selecting a column, we can enter the corresponding data, and then we can click on the "insert" button to insert the data.

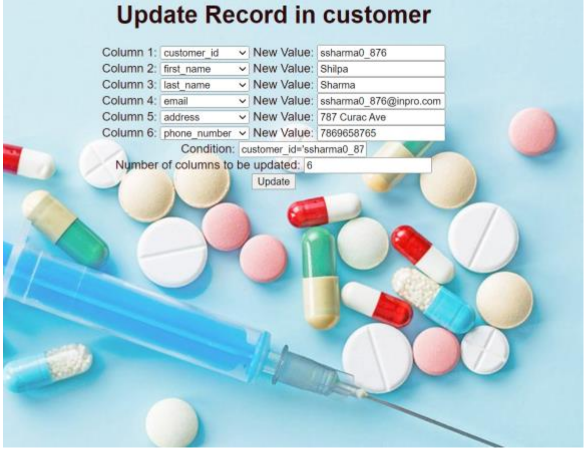


```
select * from customer where customer_id='ssharma0_876';
```

	customer_id [PK] character varying	first_name character varying	last_name character varying	email character varying	address character varying	phone_number character varying
1	ssharma0_876	Shilpa	Sharma	ssharma0_876@inro.com	777 Curac Ave	7869658765

In the above screenshot we are checking the inserted data in the pgadmin site with customer id.

Now we are going the update the details, first we have to choose the table and click on the update tab for updating the details.



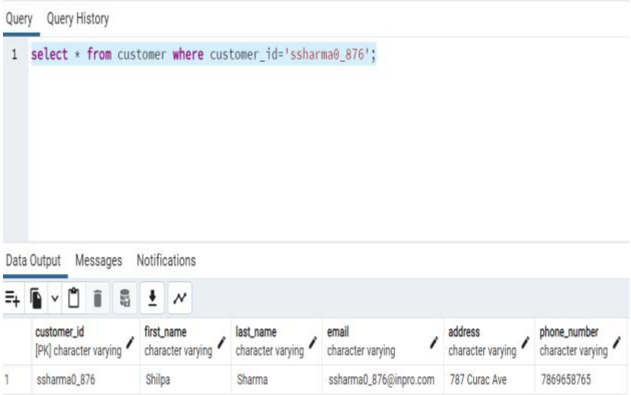
Update Record in customer

Column 1:	customer_id	New Value:	ssharma0_876
Column 2:	first_name	New Value:	Shilpa
Column 3:	last_name	New Value:	Sharma
Column 4:	email	New Value:	ssharma0_876@inpro.com
Column 5:	address	New Value:	787 Curac Ave
Column 6:	phone_number	New Value:	7869658765

Condition: customer_id='ssharma0_87

Number of columns to be updated: 6

After updating the details, we need to click on the "update" tab to finalize the changes. This will update the details accordingly.

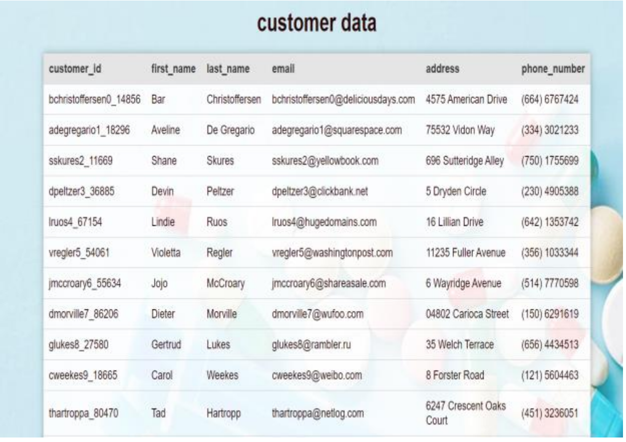


```
select * from customer where customer_id='ssharma0_876';
```

	customer_id [PK] character varying	first_name character varying	last_name character varying	email character varying	address character varying	phone_number character varying
1	ssharma0_876	Shilpa	Sharma	ssharma0_876@inpro.com	787 Curac Ave	7869658765

The customer details have been successfully updated.

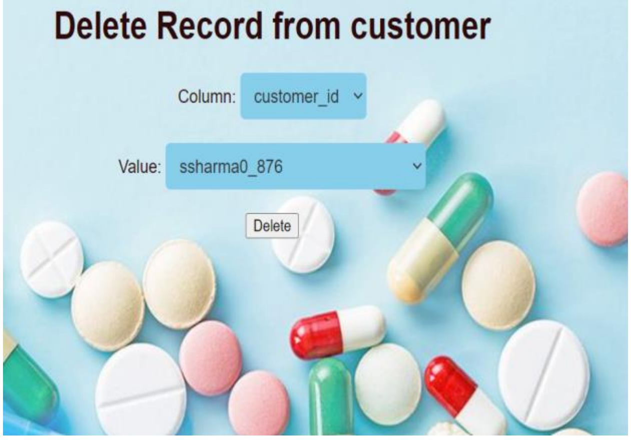
We can use the "view" tab to see the data of the table that we've selected.



customer data

customer_id	first_name	last_name	email	address	phone_number
bchristoffersen0_14856	Bar	Christoffersen	bchristoffersen0@deliciousdays.com	4575 American Drive	(664) 6767424
adegregario1_18296	Aveline	De Gregario	adegregario1@squarespace.com	75532 Vidon Way	(334) 3021233
sskures2_11669	Shane	Skures	sskures2@yellowbook.com	696 Sutteridge Alley	(750) 1755699
dpeltzer3_36885	Devin	Peltzer	dpeltzer3@clickbank.net	5 Dryden Circle	(230) 4905388
lruos4_67154	Lindie	Ruos	lruos4@hugedomains.com	16 Lillian Drive	(642) 1353742
vregler5_54061	Violetta	Regler	vregler5@washingtonpost.com	11235 Fuller Avenue	(356) 1033344
jmccroary6_55634	Jojo	McCroary	jmccroary6@shareasale.com	6 Wayridge Avenue	(514) 7770598
dmorville7_86206	Dieter	Morville	dmorville7@wufoo.com	04802 Carioca Street	(150) 6291619
glukes8_27580	Gertrud	Lukes	glukes8@rambler.ru	35 Welch Terrace	(656) 4434513
cweekes9_18665	Carol	Weekes	cweekes9@weibo.com	8 Forster Road	(121) 5604463
thartroppa_80470	Tad	Hartropp	thartroppa@netlog.com	6247 Crescent Oaks Court	(451) 3236051

We'll now demonstrate how to delete data. After selecting the table, we can click on the "delete" tab to open the page shown below.

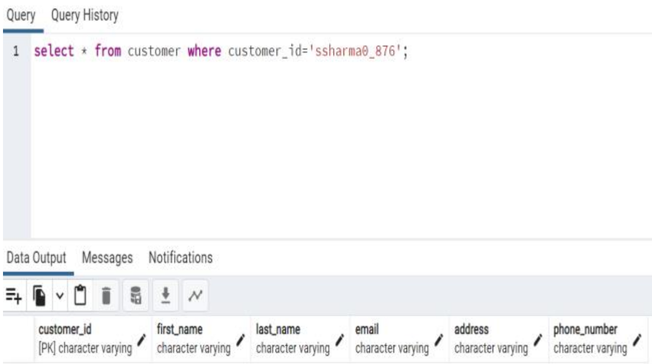


Delete Record from customer

Column: customer_id

Value: ssharma0_876

Since we're deleting details based on customer IDs, we need to select the corresponding ID in the "value" tab and then click on the "delete" button. This will remove the data from the database.



```sql
select * from customer where customer_id='ssharma0_876';
```

customer_id [PK] character varying	first_name character varying	last_name character varying	email character varying	address character varying	phone_number character varying

Contribution

- Sai Teja was responsible for normalization of the database and data generation. He also contributed to SQL queries and ER diagram creation.
- Vinitha contributed to data loading, website development, and SQL queries. She also worked on optimizing queries and was involved in data generation.
- Preethi contributed to data loading, ER diagram creation, website development, and documentation. She also worked on SQL queries and was responsible for creating the create.sql and load.sql files.
- As a team, we collaborated on creating the ER diagram, creating and loading data into the database, writing SQL queries, and optimizing queries. We also worked together to develop the website and document their work.

References

[1] "SQL-92 Aggregation Features" by C.J. Date, published in the IEEE Transactions on Knowledge and Data Engineering in 1995. This paper provides an overview of SQL's aggregation features, including GROUP BY, HAVING, and the various aggregate functions. It also discusses the semantics and implementation of these features.

[2] "Ranking Queries in Relational Databases" by Surajit Chaudhuri and Ravi Krishnamurthy, published in the IEEE Transactions on Knowledge and Data Engineering in 2004. This paper presents a comprehensive survey of ranking queries in SQL, including the syntax and semantics of the ORDER BY clause

[3] "The Theory of Normalization and Its Application to Proving the Soundness of Normalization Procedures" by C.J. Date, published in 1981 in the IEEE Transactions on Software Engineering.

[4] https://flask.palletsprojects.com/en/2.0.x/tutorial/

[5] https://www.w3schools.com/sql/

[6] https://www.youtube.com/watch?v=8aTnmsDMldY