Pharma Engine - Transforming Medicine Information

Naga Manasa Palaparthi
UBIT Name: nagamana
Email: nagamana@buffalo.edu
Section: CSE 560

Shubham Chandra
UBIT NAME: sc366
Email: sc366@buffalo.edu
Section: CSE 560

Christopher Stuhler
UBIT Name: cqstuhle
Email: cqstuhle@buffalo.edu
Section: CSE 460

I. Introduction

The pharmaceutical industry is an essential field that plays a significant role in both healthcare and business, affecting many people. As the industry continues to grow, it faces the considerable challenge of managing an increasingly diverse range of medicines. Each of these medicines has its own unique characteristics, such as chemical composition, side effects, who makes it, and how it interacts with other drugs. Given the complexity and sheer amount of this information, a more sophisticated management system is needed, far surpassing the abilities of basic tools like Excel spreadsheets.

II. Problem Statement

How can we effectively handle a large pharmaceutical inventory that includes complex details like how drugs interact, their chemical compositions, and who produces them, while also making sure the data is accurate and can be quickly accessed?

III. Why Database Over Excel?

A. Scalability

Designed to efficiently handle large volumes of data, making it more scalable for growing inventories.

B. Data Scalability

Features like foreign key constraints and unique identifiers ensure data integrity, reducing the risk of anomalies.

C. Concurrent Access

Allows multiple users to access and manipulate data concurrently without overwriting each other's changes, essential in a multi-user environment like a hospital or pharmacy.

D. Complex Queries

SQL support enables complex queries to fetch or manipulate data, allowing for more nuanced insights and operational capabilities.

E. Security

Offers more advanced security features, such as role based access control, encryption, and audit logs, ensuring that sensitive medical data is better protected.

IV. Background

The rapid expansion of the pharmaceutical industry has outpaced traditional methods of inventory management, such as paper records and basic electronic spreadsheets, creating an urgent need for a more robust solution. As the volume and complexity of medications have grown, so too have the operational and regulatory challenges associated with managing them. Previous solutions, like Excel spreadsheets, have proven inadequate for handling large-scale data and ensuring compliance with stringent healthcare laws, underscoring the need for a more integrated, scalable, and reliable system for pharmaceutical data management.

V. Significance of the Problem

The healthcare and pharmaceutical sectors frequently face the challenging responsibility of overseeing a diverse range of medicines. Every medicine has unique attributes like active components, possible adverse effects, and the companies that produce them. Ensuring precise and quick data management is crucial for multiple reasons, such as safeguarding patient well-being, streamlining operations, and adhering to stringent medical guidelines.

VI. Potential Contribution

Our project aims to create a strong, easily expandable, and efficient database specifically designed for the intricate needs of managing medications. This system is crucial for reducing the dangers of adverse drug interactions, ensuring medicines are stocked and distributed promptly, and meeting healthcare legal standards. By addressing these pressing issues, the initiative is set to significantly improve both patient care and operational efficiency.

VII. Target Users

A. Pharmacists

To look up drug information, check stock levels, and ensure no dangerous drug interactions.

B. Inventory Managers

To manage stock, update price information, and generate reports.

C. Medical Practitioners

To check drug interactions, side effects, and compositions.

VIII. Database Administrators

A. IT Department

Responsible for the upkeep, troubleshooting, backups, and data integrity of the database.

IX. Real Life Scenario

Consider John, a pharmacist at a hospital that also has an in-house pharmacy. John frequently gets prescriptions that require quick verification for potential drug interactions, especially for critical care patients. Simultaneously, Emily, the inventory manager, has the task of ensuring that essential medicines are always in stock. She needs real-time data to make quick reordering decisions.

Using our database, John can instantly pull up all relevant information about drug interactions and side effects, making sure he dispenses medication that is both effective and safe. On the other side, Emily gets real-time inventory levels, enabling her to place timely orders and avoid stockouts. In an emergency, such as an unexpected influx of patients from a mass casualty event, both John and Emily can operate efficiently, ensuring that the right medicines are available and safe to use, ultimately improving patient outcomes.

This personal scenario illustrates how our database system is not just a tool but a vital asset in healthcare, serving diverse yet interconnected roles seamlessly.

X. Entity Relationship Diagram

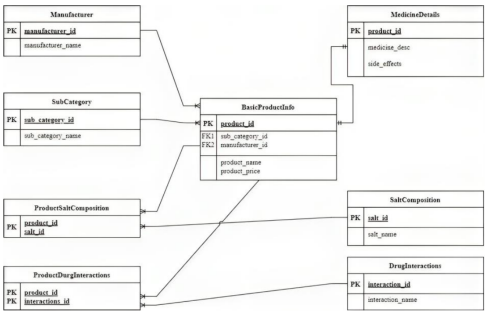

Fig. 1. ER Diagram of Indian Medicine Database.

XI. Database Schema

XII. Acquiring Dataset

We have obtained a real dataset from Kaggle, specifically from this source: Medicine Data on Kaggle. Our objective is to utilize this dataset to establish a database, implement updates, and conduct intriguing searches as part of our project. The dataset contained information such as the type of medicine, product name, salt composition, price, manufacturer details, medicine description, side effects, and drug interactions.

TABLE I
Database Schema Summary

Table Name	Primary Key	Foreign Keys
Basic Product Info	product_id	sub_category_id, manufacturer_
Medicine Details	product_id	product_id
Manufacturer Table	manufacturer_id	None
Product Drug Interactions	product_id, interaction_id	product_id, interaction_id
Product Salt Composition	product_id, salt_id	product_id, salt_id
Salt Composition	salt_id	None
Sub Category	sub_category_id	None
Drug Interactions Table	interaction_id	product_id

XIII. Data Transformation

To ensure that this dataset fits our predefined schema, Python scripts were written to perform various data transformation tasks. These tasks were crucial as the dataset contained several anomalies and discrepancies that needed to be reconciled with our schema.
Steps Taken for Data Transformation

A. Normalization

The dataset was first normalized to remove redundancy.

B. Data Type Conversion

Fields such as product_price were converted to the appropriate data types.

C. Handling Duplicates

Duplicate entries, especially in the product_name field, were either updated or skipped based on the requirement.

D. Breaking Down Complex Fields

Fields like drug_interactions, which contained JSON like data, were broken down into simpler formats to populate separate tables.

E. Data Integrity

Ensured that all Foreign Key and Unique constraints were satisfied during the transformation.

The transformed dataset has been designed to support complex SQL queries involving joins, aggregations, and sub-queries.

XIV. Tools and Libraries Used

- Language: Python
- Libraries: pandas, sqlite
- Environment: Jupyter Notebook

XV. Normalization to Boyce-Codd Form (BCNF)

Normalization is a key process in database design to reduce redundancy and dependency, ensuring efficient data organization. In the given schema, all tables are in Boyce-Codd Normal Form (BCNF), a stricter form of normalization than Third Normal Form (3NF).

The Basic Product Info table, with product_id as its primary key, includes attributes like product name, price, sub-category ID, and manufacturer ID, all functionally dependent on product_id. Similarly, the Medicine Details table, which links medicine descriptions and side effects to product_id, adheres to BCNF standards.

The Product Salt Composition and Product Drug Interactions tables serve as join tables with composite primary keys (product_id, salt_id and product_id, interaction_id, respectively), and they are in BCNF as well.

Tables Manufacturer, Sub Category, Salt Composition, and Drug Interactions Table each contain only two columns: a primary key and a dependent attribute, aligning them with BCNF criteria.

XVI. Dealing with Large Datasets

we encountered several challenges when handling larger datasets. Our primary issue was the slow performance of queries, especially those involving joins, searches, and sorting operations. This was particularly evident when working with our ”Basic Product Info” and ”Medicine Details” tables, which had grown significantly in size due to the extensive data we were processing.

- Identified Problems:

A. low Joins

Queries joining the ”Basic Product Info” and ”Medicine Details” tables were taking too long, impacting our ability to quickly retrieve and analyze data.

B. Inefficient Searches

Searches based on product_price and medicine_desc were sluggish, leading to delays in fetching results.

- Adopted Solutions: To address these issues, we decided to implement indexing, a database optimization technique. Indexing is akin to a book’s index, allowing the database to find data without scanning the entire table.

C. Indexing product_id

Post-indexing, the join operations between these two tables became significantly faster. The database was now able to quickly locate the related rows in both tables using the index, reducing the query time.

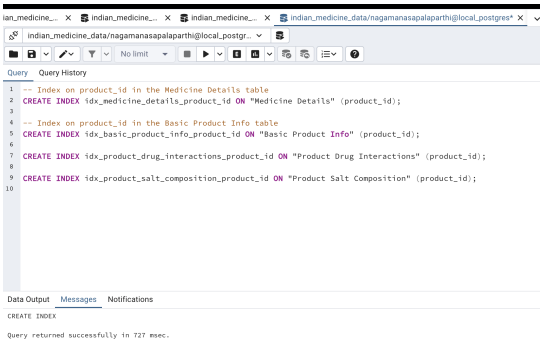

Fig. 2. product_id indexing

D. Indexing product_price

Searches based on product_price were much quicker. Queries that were previously doing full table scans now utilized the index, leading to a substantial decrease in response time.

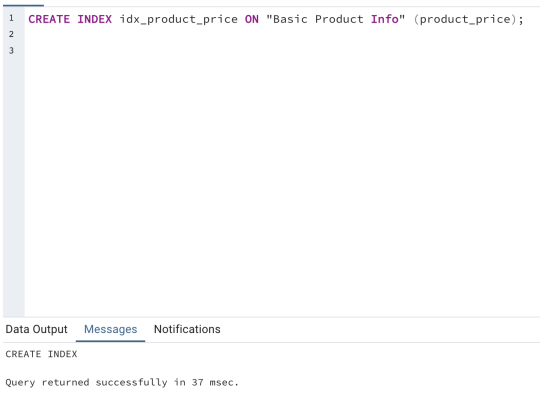

Fig. 3. product_price indexing

XVII. Queries Execution

A. Insertion

- INSERT INTO ”Manufacturer Table” (”product_manufactured”) VALUES (’New Pharma Inc.’);

  This query adds a new row to the ”Manufacturer” table with a manufacturer_id of 790 and a manufacturer_name of ’New Pharma Inc.’.

- INSERT INTO ”Basic Product Info” (product_id, sub_category_id, manufacturer_id, product_name, product_price) VALUES (6044, 1, 1, ’New Medicine’, ’150’);

  This query adds a new product to the ”Basic Product

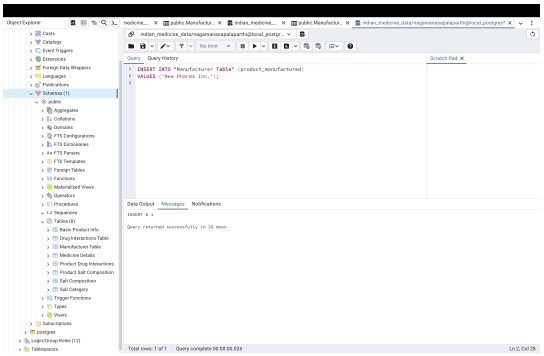

Fig. 4. insertion query 1

Info" table. It includes a new product_id (6044, along with the sub_category_id, manufacturer_id, the name of the product ('New Medicine'), and its price ('150'). The values for sub_category_id and manufacturer_id should exist in their respective reference tables.

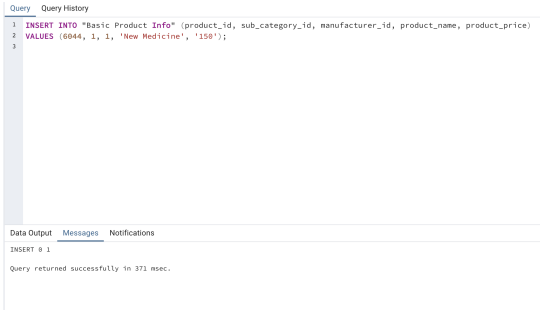

Fig. 5. insertion query 2

B. Updation

- UPDATE "Medicine Details" SET side_effects = 'None' WHERE product_id IN (SELECT product_id FROM "Basic Product Info" WHERE product_price ¡ '50');

  This query updates the side effects to 'None' in the "Medicine Details" table for all products that are priced below $50.

- UPDATE "Manufacturer" SET manufacturer_name = 'Updated Pharma Ltd' WHERE manufacturer_id = 101;

  This query updates the name of the manufacturer with ID 101 to 'Updated Pharma Ltd' in the "Manufacturer Table" table.

C. Deletion

- DELETE FROM "Manufacturer" WHERE manufacturer_id = 790;

  This query deletes the manufacturer with manufacturer_id 790 from the "Manufacturer" table.

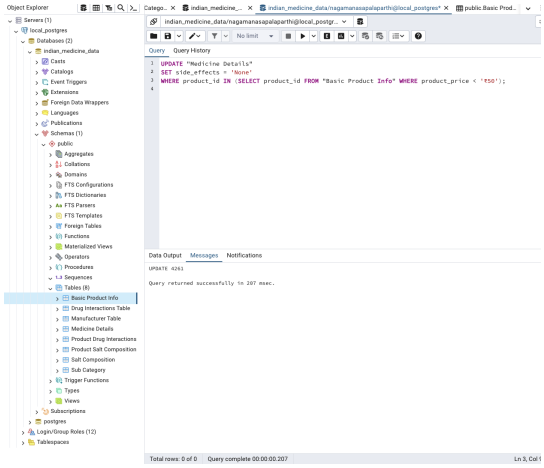

Fig. 6. updation query 1

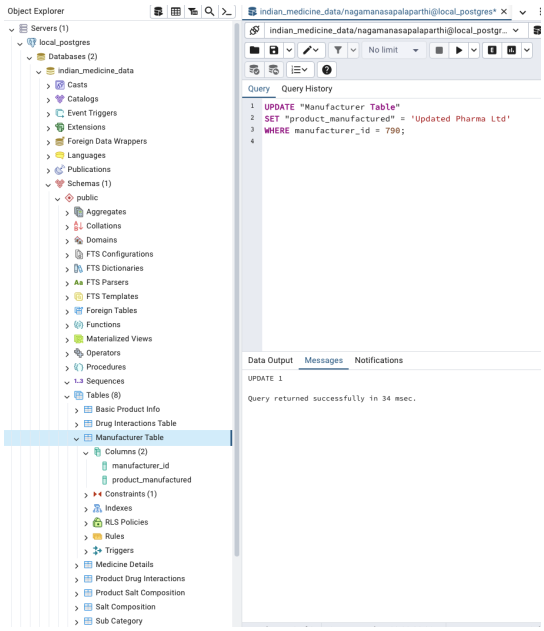

Fig. 7. updation query 2

- DELETE FROM "Basic Product Info" WHERE product= 50;

  This query deletes products from the "Basic Product Info" table that have a price equal to 50.

D. Selection

- SELECT MD.product_id, MD.medicine_desc, M.product_manufactured FROM "Medicine Details" MD JOIN "Basic Product Info" BPI ON MD.product_id = BPI.product_id JOIN "Manufacturer" M ON BPI.manufacturer_id = M.manufacturer_id ORDER BY M.product_manufactured;

  This query retrieves the product ID, medicine description, and manufacturer name, joining the "Medicine Details",

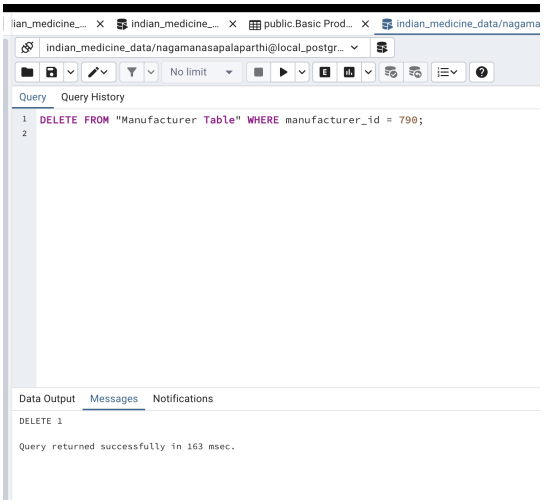

Fig. 8. deletion query 1

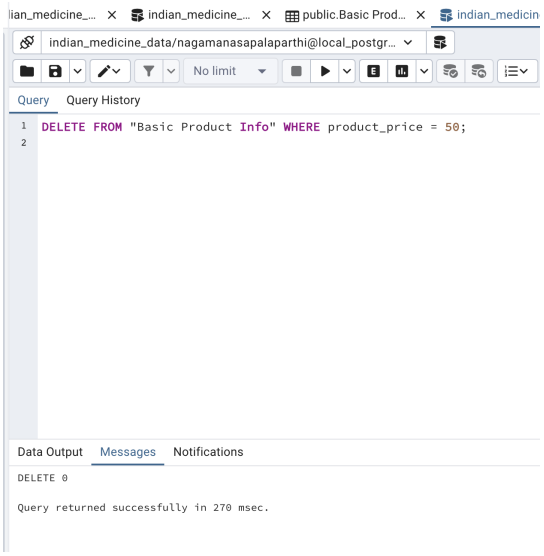

Fig. 9. deletion query 2

”Basic Product Info”, and ”Manufacturer” tables. The results are ordered by the manufacturer’s name.

- SELECT sub_category, COUNT(*) AS TotalProducts FROM ”Basic Product Info” BPI JOIN ”Sub Category” SC ON BPI.sub_category_id = SC.sub_category_id GROUP BY SC.sub_category;

  This query counts the total number of products in each sub-category by joining the ”Basic Product Info” and ”Sub Category” tables and grouping the results by sub-category.

- SELECT product_name, product_price FROM ”Basic

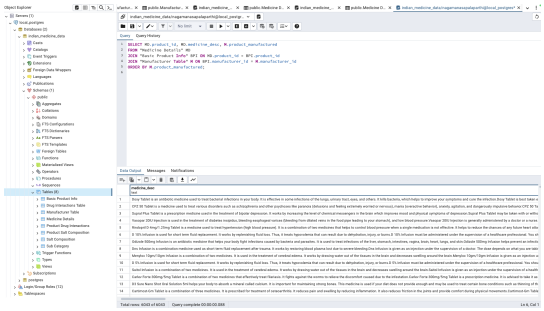

Fig. 10. selection query 1

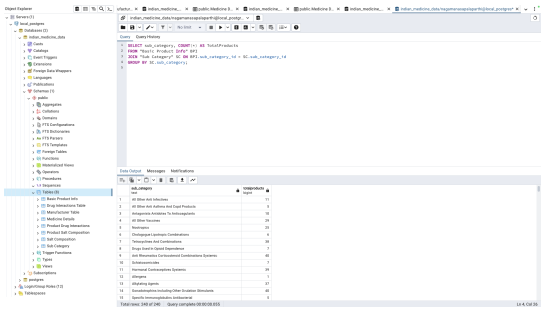

Fig. 11. selection query 2

Product Info” WHERE manufacturer_id IN (SELECT manufacturer_id FROM ”Manufacturer Table” WHERE manufacturer_name LIKE ’

This query finds the names and prices of products whose manufacturers’ names contain ’Pharma’, using a subquery in the ”Manufacturer” table.

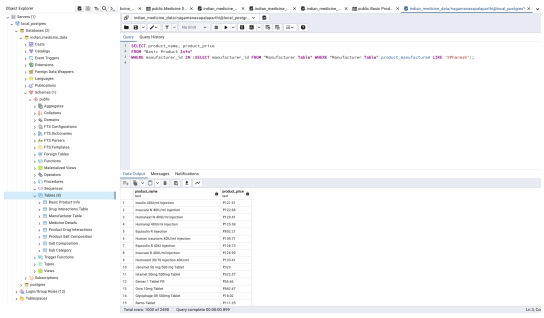

Fig. 12. selection query 3

- SELECT M.manufacturer_name, AVG(CAST(SUBSTR(BPI.product_price, 2) AS FLOAT)) AS AvgPrice FROM ”Basic Product Info” BPI JOIN ”Manufacturer Table” M ON BPI.manufacturer_id = M.manufacturer_id GROUP BY M.manufacturer_name HAVING COUNT(*) ¿ 5;

  This query calculates the average price of products

for each manufacturer, provided the manufacturer has more than 5 products. It also converts the product_price from text to a float after removing the currency symbol.

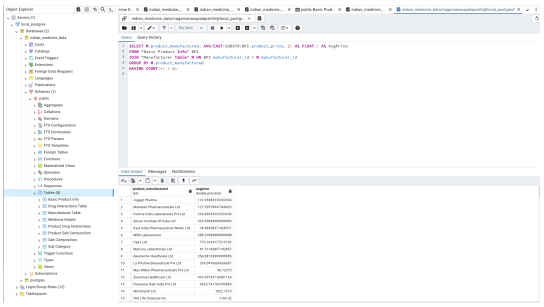

Fig. 13. selection query 4

- SELECT MD.product_id, MD.medicine_desc, SC.sub_category, M.manufacturer_name FROM ”Medicine Details” MD JOIN ”Basic Product Info” BPI ON MD.product_id = BPI.product_id JOIN ”Sub Category” SC ON BPI.sub_category_id = SC.sub_category_id JOIN ”Manufacturer Table” M ON BPI.manufacturer_id = M.manufacturer_id WHERE SC.sub_category = ’Antiepileptic’;

  This query retrieves the product ID, description, sub-category, and manufacturer name for all products classified as ’Antiepileptic’, by joining multiple tables.

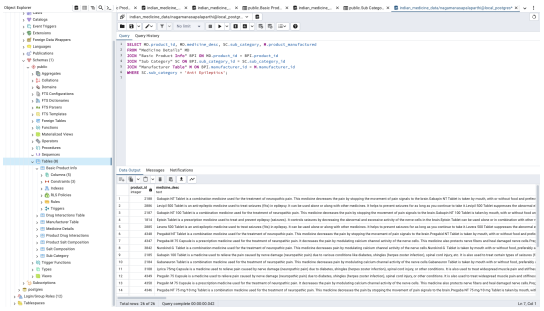

Fig. 14. selection query 5

XVIII. Query Execution Analysis: Problematic Queries

A. Inefficient Join

SELECT MD.product_id, MD.medicine_desc, BPI.product_price FROM ”Medicine Details” MD JOIN ”Basic Product Info” BPI ON MD.product_id = BPI.product_id WHERE BPI.product_price ¿ 500;

This query might be inefficient if there are no indexes on the product_id in both tables or on product_price. Also, if the join produces a large intermediate result set, it can be resource-intensive.

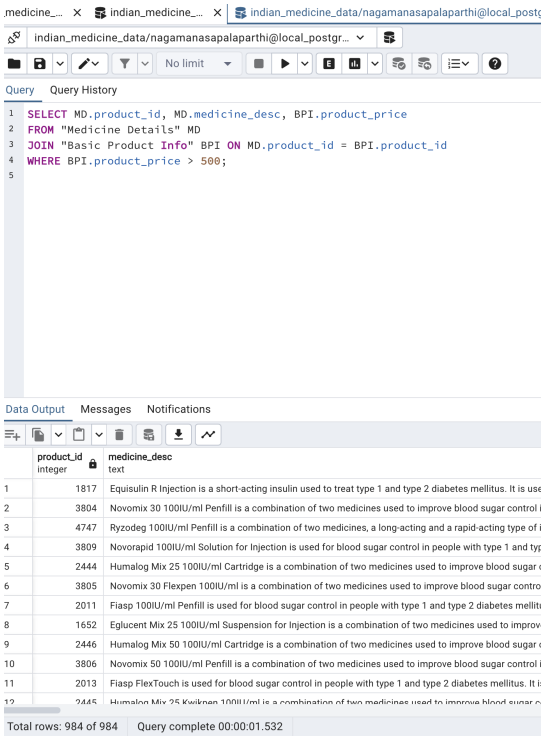

Fig. 15. inefficient join

Solution could be ensuring that product_id in both tables and product_price are indexed or consider filtering one table in a sub query before joining, to reduce the size of the join operation.

B. Full Table Scan

SELECT * FROM ”Basic Product Info” WHERE product_price ¿ 500;

If product_price is not indexed, this query will lead to a full table scan, which is inefficient, especially for large tables.

Creating an index on the product_price column to speed up the query execution. If the data allows, rewriting the query or adding additional filtering criteria can also help.

C. Join Without a Relationship

SELECT MD.product, MD.medicine_desc, BPI.product_price FROM ”Medicine Details” MD, ”Basic Product Info” BPI WHERE MD.product_id = BPI.product_id AND BPI.manufacturer_id = 1;

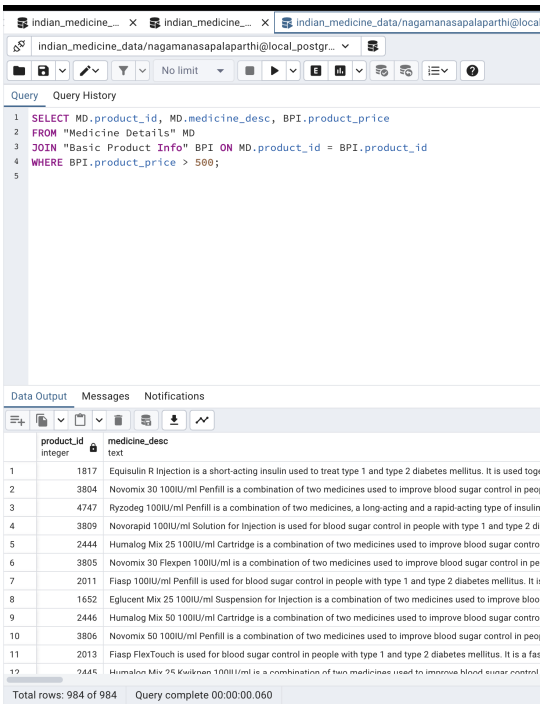

Fig. 16. inefficient join after indexing

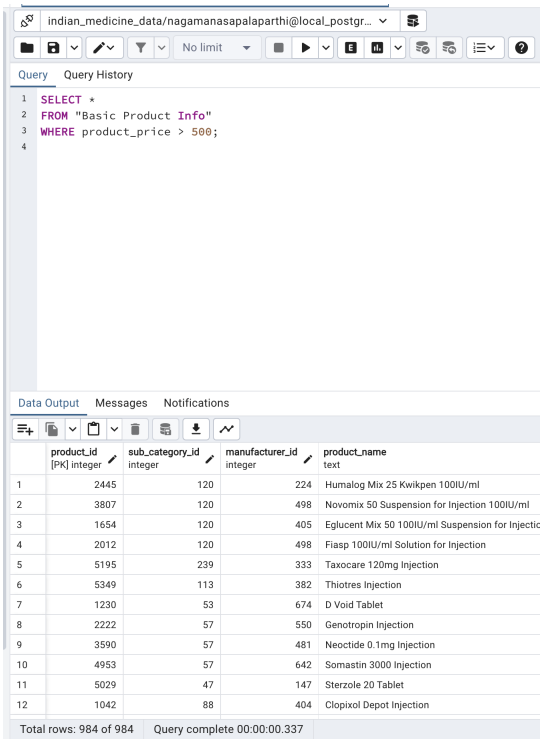

Fig. 17. before indexing (full table scan)

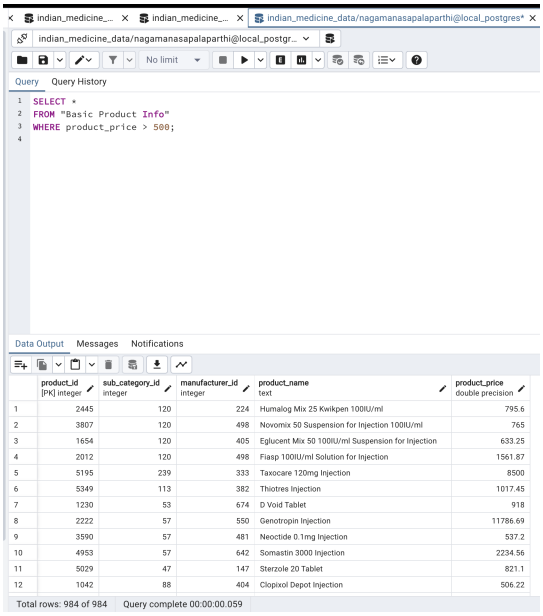

Fig. 18. after indexing

This query uses an implicit join (cross join) and then applies a filter. Without proper indexing, this can lead to an inefficient execution plan, especially if the tables are large.

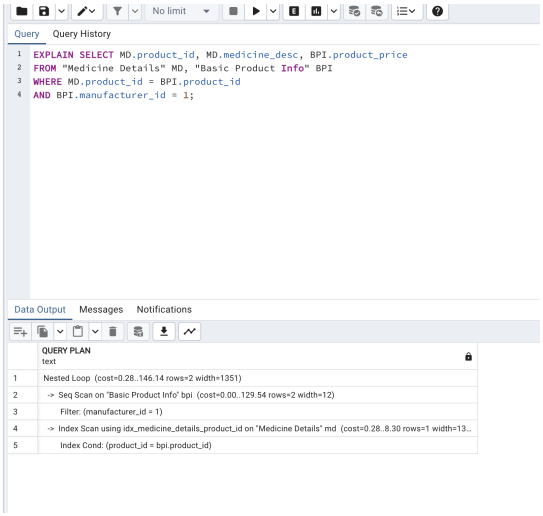

Fig. 19. Join without relationship using EXPLAIN analysis

Use an explicit JOIN clause instead of a comma-separated list of tables. This makes the query more readable and can sometimes result in a more efficient execution plan. Ensure that both product_id in ”Medicine Details” and ”Basic Product Info” tables and manufacturer_id in ”Basic Product Info” are indexed.

D. Large Offset in Pagination

SELECT * FROM ”Basic Product Info” ORDER BY product_price DESC LIMIT 10 OFFSET 1000;

Large offsets can be problematic because the database must still retrieve and then discard the offset rows. This becomes increasingly inefficient as the offset grows.

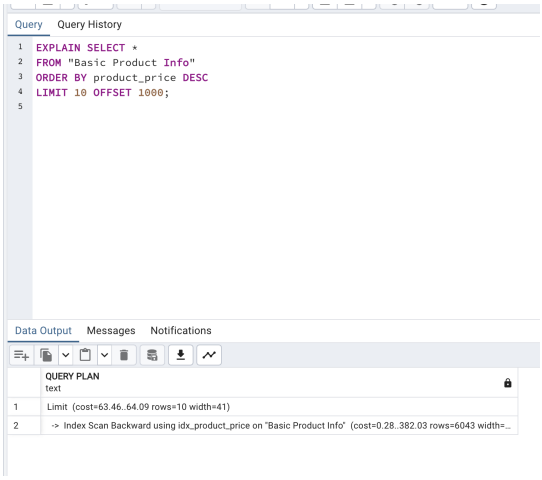

Fig. 20. Pagination using EXPLAIN analysis

Keyset pagination : Instead of using OFFSET, use keyset pagination (also known as the ”seek method”), where you query rows based on a seen-last value (e.g., the last product_price you fetched). Ensuring that there is an index on product_price to speed up the sorting operation.

XIX. User Interface

we developed Pharma Engine, a user-friendly interface designed to seamlessly execute and display results from complex database queries. This interface serves as the gateway for users to interact with our extensive pharmaceutical database, allowing them to effortlessly run queries and obtain necessary information. Pharma Engine stands as a testament to our commitment to bridging the gap between intricate data management systems and end-user accessibility, ensuring that users from various backgrounds can easily navigate and utilize our application for their diverse data needs.

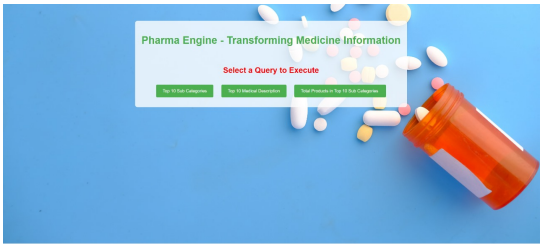

Fig. 21. User Interface main page

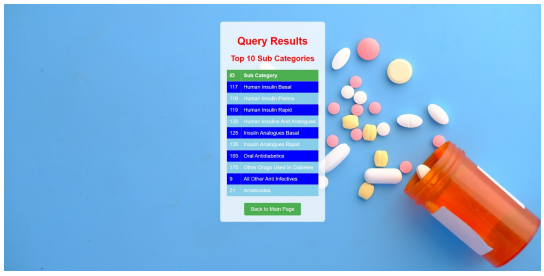

Fig. 22. Top 10 subcategories

XX. Future Improvements

A. Advanced Data Analytics

Incorporating data analytics features for predictive analysis and trends in pharmaceutical data could provide valuable insights for users.

B. Security Enhancements

Implementing advanced security measures to protect sensitive medical data, ensuring compliance with healthcare regulations like HIPAA.

C. Integration with External APIs

Integrating with external healthcare APIs to provide users with up-to-date medical information and drug data.

D. Real-time Data Processing

Implementing real-time data processing capabilities for immediate updates and notifications.

XXI. Conclusion

In concluding the Pharma Engine project, our journey through the realms of database optimization and user interface integration was both challenging and enriching. Tackling these hurdles, we’ve gained invaluable skills in problem-solving and team collaboration, greatly enhancing our technical prowess. This experience with Pharma Engine has not only broadened our understanding of technology and healthcare data management but also equipped us for future challenges and opportunities in these ever-evolving fields.

XXII. Project Contribution

A. Naga Manasa Palaparthi

- Dataset evaluation, Data Preprocessing, Initial Project Discussion, Drafting Project Report (with Shubham Chandra)
- Dataset finalization, Data Normalization, Preliminary Project Discussion, Composition of Project Report (with Shubham Chandra)
- Design of Database Tables, Identification of Primary and Foreign keys (with Christopher Stuhler) Setting up the Database in PostgreSQL, including creation of tables
- Data Entry into Database Tables (with Christopher Stuhler)
- Adjustments and Refinements to Web UI Requirements

B. Shubham Chandra

- Dataset evaluation, Data Preprocessing, Initial Project Discussion, Drafting Project Report (with Naga Manasa Palaparthi)
- Selection of Dataset; Dataset finalization, Data Normalization, Preliminary Project Discussion, Composition of
- Project Report (with Naga Manasa Palaparthi)
- Creation and Editing of Project Video (with Naga Manasa Palaparthi and Christopher Stuhler)
- Compilation of SQL Dump File

C. Christopher Stuhler

- Design of Database Tables, Identification of Primary and Foreign keys (with Naga Manasa Palaparthi)
- Data Entry into Database Tables (with Naga Manasa Palaparthi)
- Creation and Editing of Project Video (with Shubham Chandra and Naga Manasa Palaparthi)