Amazon Book Reviews Management System

Github

Satya Anudeep Kotaru
UBIT NO: 50538630

Bhavya Teja Inturi
UBIT NO: 50544957

Vineet Sreeram
UBIT NO: 50545716

***Abstract*—Millions of readers turn to Amazon book reviews for guidance and insights when choosing their next read. However, navigating this vast ocean of opinions can be overwhelming. Authors and publishers also lack clear data-driven insights into reader preferences and market trends. This project aims to bridge this gap by designing and implementing a database system that leverages the rich information within the Amazon book reviews dataset.**

***Index Terms*—Books, Reviews, Users, Publishers, Sales, Authors**

I. Introduction

In today's digital age, book reviews play a crucial role in guiding readers and authors alike. However, managing the abundance of online reviews has become a challenge. Book Reviews Management Systems aim to address this by streamlining the collection and analysis of reviews. This report explores the design and implementation of such systems, highlighting their potential to revolutionize how we interact with and derive insights from book reviews.

II. Objective

A. Objective and proposed solution

This database project delves into the rich world of Amazon book reviews, aiming to extract valuable insights and recommendations for readers, authors, and publishers. Leveraging the vast dataset, we will design and implement a system that tackles data quality challenges, optimises storage and retrieval, and provides intuitive interfaces for diverse user needs. From personalised book suggestions to author analytics and advanced search functionalities, the objective of this project is to harness the potential of reviews, developing a thriving literary community and providing stakeholders with data-driven knowledge.

B. SQL over Excel

As we have a large dataset of over 5GB data, opening the excel file and performing any analytical tasks takes more time and takes a toll on the PC's performance. So, SQL is preferred over Excel for our books database project due to its scalability and efficiency. Unlike Excel's limitations with file size and performance, SQL can handle vast amounts of data and complex queries seamlessly. Its relational structure allows for organizing diverse book-related data effectively, while supporting concurrent access by multiple users ensures data integrity and security. SQL offers greater flexibility and reliability for managing the extensive dataset and diverse needs of the project compared to Excel.

C. Target Users

The Amazon book reviews database project targets researchers, data analysts, and book industry stakeholders. Data analysts can explore patterns in customer behaviour and popular book genres, while researchers can delve into sentiment analysis and recommendation systems. Businesses like publishers and retailers can benefit from market research and competition analysis, optimising strategies based on user preferences. Developers can utilise the dataset to create tools that enhance user experience and engagement in the book ecosystem.

D. Administrator and Real life Scenario

Having Amazon's database team administer our Amazon book reviews database project would be really beneficial. They could guarantee the effective storage, retrieval, and analysis of our book reviews data thanks to their experience managing enormous datasets and maximizing database performance. Consider the scenario when our project encounters an unexpected spike in data volume during a marketing campaign, leading to problems with database performance. To ensure ongoing access for users and stakeholders that depend on the data for decision-making, Amazon's database team may quickly adopt scalable solutions, such as partitioning methods or database optimizations, to maintain smooth operation and prevent downtime. Their profound understanding of database administration would be of immeasurable assistance in preserving the reliability as well as effectiveness of the database infrastructure for our project.

III. Data acquisition and transforming

A. Link to original dataset

Amazon Books Reviews

We have downloaded the 2 csv files. Merged them together and added a few columns and removed a few according to our use case. We wrote a code to manipulate and generate our data using the Faker library.

B. Final Database Schema

- Books
  - book_id (primary key)
  - title

  - author
  - category
- Users
  - user_id (primary key)
  - profile_name
  - user_email
- Reviews
  - review_id (primary key)
  - book_id (foreign key)
  - user_id (foreign key)
  - ratings_count
  - review_helpfulness
  - review_score
  - review_time
  - review_summary
- Authors
  - author_id (primary key)
  - book_id (foreign key)
  - author
  - author_email
- Publishers
  - publisher_id (primary key)
  - book_id (foreign key)
  - published_year
  - publisher
- Sales
  - sale_id (primary key)
  - publisher_id (foreign key)
  - category
  - top_country
  - books_sold
  - price
  - language

C. *Changes made from Milestone 1*

Summary of the changes made to each table:

- Books Table
  - Instead of using the ”title” as the primary key, a new attribute ”book_id” was added and designated as the primary key. While the title of books may not always be unique (especially if multiple editions or versions exist), a book_id can guarantee uniqueness within the database. This ensures that each book is uniquely identified, regardless of its title or other attributes.
- Users Table
  - The ”user_id” attribute is now the sole primary key, replacing the previous combination of ”user_id” and ”profile_name” because “user_id” can uniquely identify each tuple of the relation.
- Reviews Table
  - A new attribute ”review_id” was introduced as the primary key, using a serial ID as the primary key simplifies database design by ensuring ease of reference and modification, enhancing performance with efficient indexing, and improving data integrity with unique, auto-incremented keys
  - The ”profile_name” attribute was removed from the table, as the user information can be referenced from the ”user_id” attribute, which now acts as a foreign key.
- Authors Table
  - The ”author_id” attribute was designated as the primary key.
  - Instead of using the ”title” attribute, ”book_id” is now used to extract details from the Books table.
- Publishers Table
  - A new attribute ”publisher_id” was introduced as the primary key to uniquely define all other attributes.
  - As the database grows and new publishers are added, a numeric publisher_id can scale more effectively than relying on attributes like publisher name. Publisher names may vary in length and complexity, potentially leading to performance issues as the dataset expands. Using a publisher_id simplifies the process of adding new records and ensures consistent performance over time.
  - The ”published_date” attribute was removed due to null or poorly defined values.
- Sales Table
  - A new attribute ”sale_id” was added as the primary key to uniquely identify records.
  - Primary keys enforce entity integrity by ensuring that each record in the table is uniquely identified. By using sale_id as the primary key, we can maintain data integrity and avoid duplicate records for sales transactions.
  - A relation between the Sales and Publishers tables was established by adding ”publisher_id” to the Sales table.

These modifications aim for better data organisation, normalisation, and relationship establishment within the database schema.

IV. BCNF Decomposition

To ensure all relations are in Boyce-Codd Normal Form (BCNF), we need to analyse the functional dependencies (FDs) in each relation i.e., ensures that every FD’s left hand side (determinant) is a candidate key and decomposes them if necessary.

- Books Table:
  book_id $\rightarrow$ title
  book_id, title $\rightarrow$ author, category
  book_id, category $\rightarrow$ author
  Since the left side of all the functional dependencies being a superkey, and no non-trivial functional dependencies exist, hence, the Books table is in BCNF.
- Users Table:
  user_id $\rightarrow$ profile_name, user_email
  The Users table is already in BCNF. The primary key

(user_id) determines all other attributes, and no non-trivial functional dependencies exist.

- Reviews Table:
  review_id → user_id, book_id
  review_id, user_id → review_score, review_time
  review_id, book_id → review_helpfulness, review_summary
  review_id, review_time → ratings_count
  The Reviews table has each determinant as a super key, the table is already in BCNF.
- Authors Table:
  author_id → book_id
  author_id, author → book_id, author_email
  author_id, book_id → author
  The Authors table is already in BCNF. The primary key (author_id) determines all other attributes, and no non-trivial functional dependencies exist.
- Publishers Table:
  publisher_id → book_id
  publisher_id, book_id → publisher, published_year
  The Publishers table is also already in BCNF. Each determinant is a super key, and there are no non-trivial functional dependencies.
- Sales Table:
  sale_id → publisher_id
  sale_id, category → books_sold, language
  sale_id , publisher_id → category, price
  sale_id, language → top_country
  The Sales table appears to be in BCNF as well. Each determinant is a super key, and there are no non-trivial functional dependencies.

V. ER Diagram

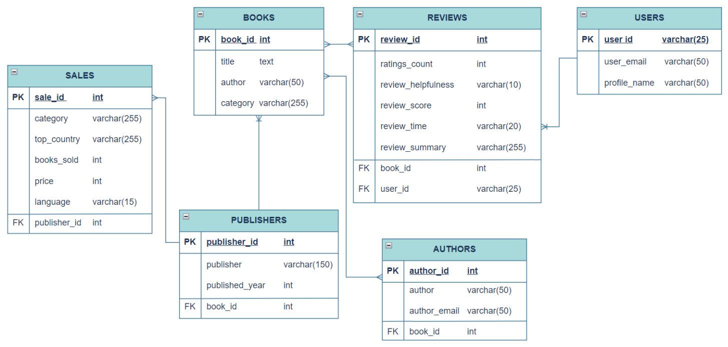


Fig. 1. ER Diagram of Amazon Books Review Management System

A. Actions taken on foreign keys when the primary key is deleted ?

When primary key is deleted

- We are using "on delete cascade" after referencing tables if foreign key is also used as the primary key. It deletes the relations of the deleted attributes.
- And "on delete set null" to foreign keys for all the keys not used as primary. It sets the deleted attribute values to null.

B. Constraints

- Primary Key Constraints:
  Each table has a primary key constraint, denoted by (PK), which uniquely identifies each record in the table.
- Foreign Key Constraints:
  Foreign key constraints, denoted by (FK), establish relationships between tables.
  For example: In the Reviews table, book_id and user_id are foreign keys referencing the Books and Users tables, respectively.
  publisher_id in the Sales table is a foreign key referencing the Publishers table.
- Unique Constraints:
  Unique constraints ensure that certain attributes have unique values, such as email addresses in the Users table.
- Check Constraints:
  Check constraints can be applied to ensure that certain attributes satisfy specific conditions.
  For example, in the Sales table, we can ensure that the price is always greater than or equal to zero.
- Not Null Constraints:
  Not null constraints ensure that certain attributes cannot have null values, such as book titles or user profiles.
- Referential Integrity Constraints:
  Referential integrity constraints ensure that foreign key values in a table match primary key values in another table, maintaining the integrity of the relationships between tables.

Implementing these constraints in the database schema ensures data integrity, consistency, and reliability. While planning to work with real datasets, it's essential to validate the design by analyzing sample data to ensure that it accurately represents the real-world scenario and that the database constraints effectively enforce the desired rules and relationships.

VI. Indexing

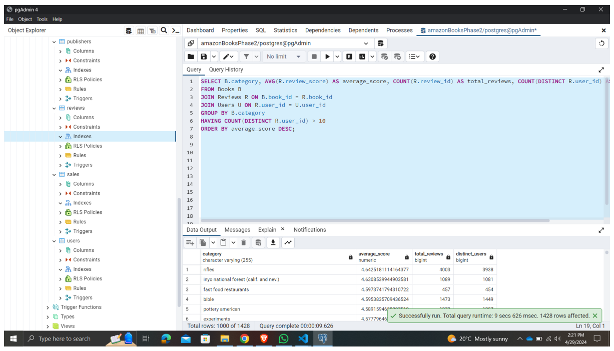
While working with larger datasets, such as the Amazon books review dataset, several challenges may arise, including:

- Performance Issues: Processing and querying large datasets can lead to performance issues, such as slow response times and high resource utilization.
- Memory Constraints: Loading large datasets into memory may exceed system memory limits, causing out-of-memory errors or slowdowns due to swapping.
- Indexing Overhead: Without proper indexing, queries on large datasets may require scanning the entire dataset, leading to slower query execution times.

Different Types of Indexes:
Dense Index: In a dense index, there is an index entry for every single record in the database. Each entry points directly to its corresponding record. This makes dense indexes very fast for retrieval, but they can be space-consuming and slower to

update, as every insert, delete, or update of the records might require changes in the index.
Sparse Index: Sparse indexes only contain entries for some of the records in the database. Typically, an index entry points to the first record in a block of the database file, and not every record will have an index entry. This type of index uses less space than dense indexes and can be faster to maintain, but might be slower for certain lookups as it may require additional searches within the block.
B-tree Index: A balanced tree structure that keeps data sorted and allows searches, sequential access, insertions, and deletions in logarithmic time. Good for a wide range of query operations.
Hash Index: Hash indexes use a hash function to compute the index location for a key. They are extremely efficient for point queries where the exact match is known. They do not support range queries effectively because the hash function disperses key values randomly across the index.
We chose default indexes which are B-Tree indexes for their versatility in handling both equality and range queries and their efficiency in maintaining order, which is beneficial for sorting and complex joins typical in relational database operations.
Some of the questions faced while handling the larger dataset and their solutions include:

- Query Performance: Queries on large datasets may take a long time to execute due to the lack of indexing. Solution: Identify frequently queried columns and create indexes on those columns to improve query performance.
- Memory Usage: Loading the entire dataset into memory may not be feasible due to memory constraints. Solution: Implement pagination techniques or load data in chunks to manage memory usage effectively.
- Data Integrity: Ensuring data integrity and consistency, especially during data insertion and updates, becomes challenging with large datasets. Solution: Implement proper transaction management and error handling mechanisms to maintain data integrity.
- Index Maintenance Overhead: Creating and updating indexes can introduce overhead during data insertion and updates. Solution: Schedule regular maintenance tasks, such as index rebuilding or reorganization, to optimize index performance.



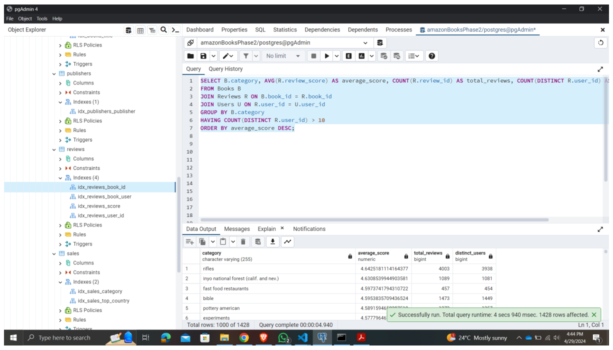
```
SELECT B.category, AVG(R.review_score) AS average_score, COUNT(R.review_id) AS total_reviews, COUNT(DISTINCT R.user_id) A
FROM Books B
JOIN Reviews R ON B.book_id = R.book_id
JOIN Users U ON R.user_id = U.user_id
GROUP BY B.category
HAVING COUNT(DISTINCT R.user_id) > 10
ORDER BY average_score DESC;
```

	category character varying (255)	average_score numeric	total_reviews bigint	distinct_users bigint
1	rifles	4.6425181114164377	4003	3938
2	inyo national forest (calif. and nev.)	4.6308539944903581	1089	1081
3	fast food restaurants	4.5973741794310722	457	454
4	bible	4.5953835709436524	1473	1449
5	pottery american	4.5891594[illegible]	[illegible]	[illegible]
6	experiments	4.57779646[illegible]		

```
Successfully run. Total query runtime: 9 secs 626 msec. 1428 rows affected.
Total rows: 1000 of 1428    Query complete 00:00:09.626    Ln 19, Col 1
```

Fig. 2. Query executed without introducing indexes

Adding an index to the data has likely improved query performance by enabling faster data retrieval and reducing the need for expensive disk I/O operations. This optimization can lead to significant improvements in query execution time, as evidenced by the decrease from 9.62 seconds to 4.94 seconds in extracting title and average review score using books and reviews table case.



```
SELECT B.category, AVG(R.review_score) AS average_score, COUNT(R.review_id) AS total_reviews, COUNT(DISTINCT R.user_id) A
FROM Books B
JOIN Reviews R ON B.book_id = R.book_id
JOIN Users U ON R.user_id = U.user_id
GROUP BY B.category
HAVING COUNT(DISTINCT R.user_id) > 10
ORDER BY average_score DESC;
```

	category character varying (255)	average_score numeric	total_reviews bigint	distinct_users bigint
1	rifles	4.6425181114164377	4003	3938
2	inyo national forest (calif. and nev.)	4.6308539944903581	1089	1081
3	fast food restaurants	4.5973741794310722	457	454
4	bible	4.5953835709436524	1473	1449
5	pottery american	4.5891594[illegible]	[illegible]	[illegible]
6	experiments	4.57779646[illegible]		

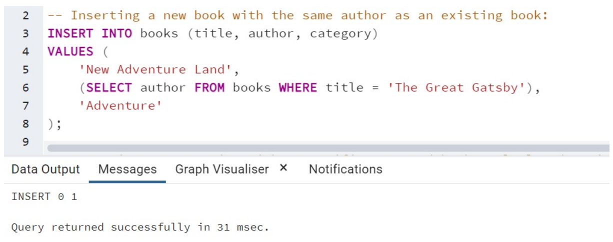
```
Successfully run. Total query runtime: 4 secs 940 msec. 1428 rows affected.
Total rows: 1000 of 1428    Query complete 00:00:04.940    Ln 1, Col 1
```

Fig. 3. Query executed with introducing indexes

By addressing these challenges and adopting indexing concepts effectively, we can improve the overall performance and scalability of handling large datasets like the Amazon books review dataset.

VII. Queries

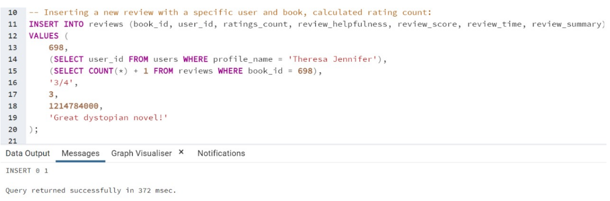
A. Insert Queries



```
-- Inserting a new book with the same author as an existing book:
INSERT INTO books (title, author, category)
VALUES (
    'New Adventure Land',
    (SELECT author FROM books WHERE title = 'The Great Gatsby'),
    'Adventure'
);
```

```
INSERT 0 1

Query returned successfully in 31 msec.
```

Fig. 4. Query to insert a new book with the same author

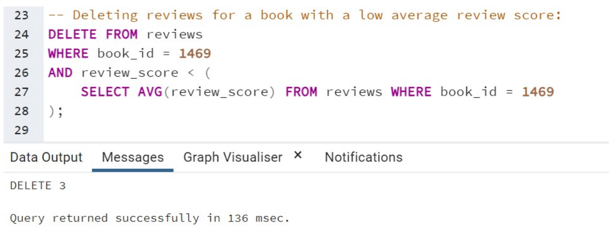


```
-- Inserting a new review with a specific user and book, calculated rating count:
INSERT INTO reviews (book_id, user_id, ratings_count, review_helpfulness, review_score, review_time, review_summary)
VALUES (
    698,
    (SELECT user_id FROM users WHERE profile_name = 'Theresa Jennifer'),
    (SELECT COUNT(*) + 1 FROM reviews WHERE book_id = 698),
    '3/4',
    3,
    1214784000,
    'Great dystopian novel!'
);
```

```
INSERT 0 1

Query returned successfully in 372 msec.
```

Fig. 5. Query to insert new review with a specific user, book and calculated rating count

B. Delete Queries

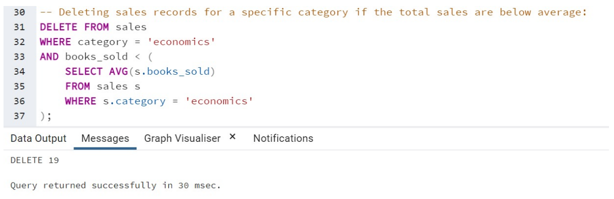


```sql
-- Deleting reviews for a book with a low average review score:
DELETE FROM reviews
WHERE book_id = 1469
AND review_score < (
    SELECT AVG(review_score) FROM reviews WHERE book_id = 1469
);
```

Data Output   Messages   Graph Visualiser ×   Notifications

```
DELETE 3

Query returned successfully in 136 msec.
```

Fig. 6. Query to Delete reviews for a book with low average review score



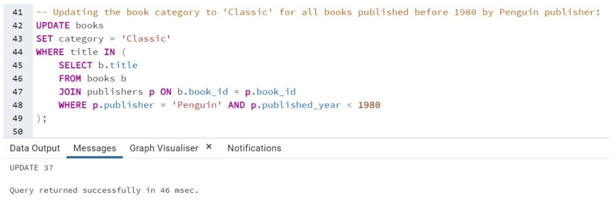
```sql
-- Deleting sales records for a specific category if the total sales are below average:
DELETE FROM sales
WHERE category = 'economics'
AND books_sold < (
    SELECT AVG(s.books_sold)
    FROM sales s
    WHERE s.category = 'economics'
);
```

Data Output   Messages   Graph Visualiser ×   Notifications

```
DELETE 19

Query returned successfully in 30 msec.
```

Fig. 7. Query to delete sale records for a specific category with below-average total sales

C. Update Queries

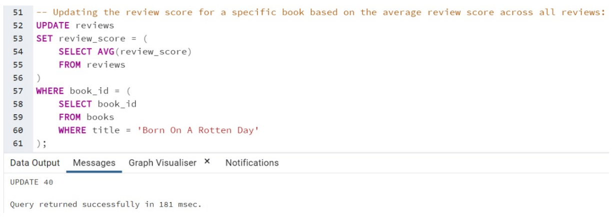


```sql
-- Updating the book category to 'Classic' for all books published before 1980 by Penguin publisher:
UPDATE books
SET category = 'Classic'
WHERE title IN (
    SELECT b.title
    FROM books b
    JOIN publishers p ON b.book_id = p.book_id
    WHERE p.publisher = 'Penguin' AND p.published_year < 1980
);
```

Data Output   Messages   Graph Visualiser ×   Notifications

```
UPDATE 37

Query returned successfully in 46 msec.
```

Fig. 8. Query to update book category for all Penguin- published books before 1980r



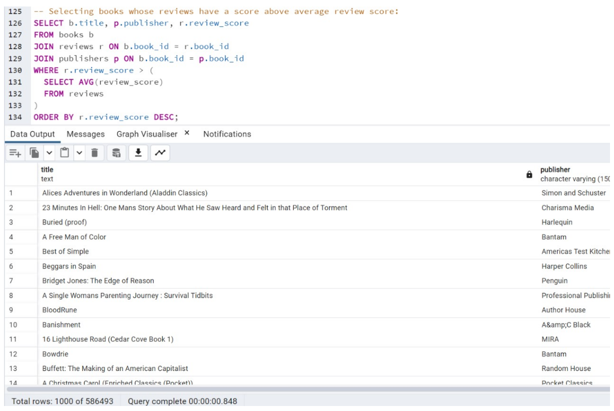
```sql
-- Updating the review score for a specific book based on the average review score across all reviews:
UPDATE reviews
SET review_score = (
    SELECT AVG(review_score)
    FROM reviews
)
WHERE book_id = (
    SELECT book_id
    FROM books
    WHERE title = 'Born On A Rotten Day'
);
```

Data Output   Messages   Graph Visualiser ×   Notifications

```
UPDATE 40

Query returned successfully in 181 msec.
```

Fig. 9. Query to update review score of books based on average review score

D. Select Queries



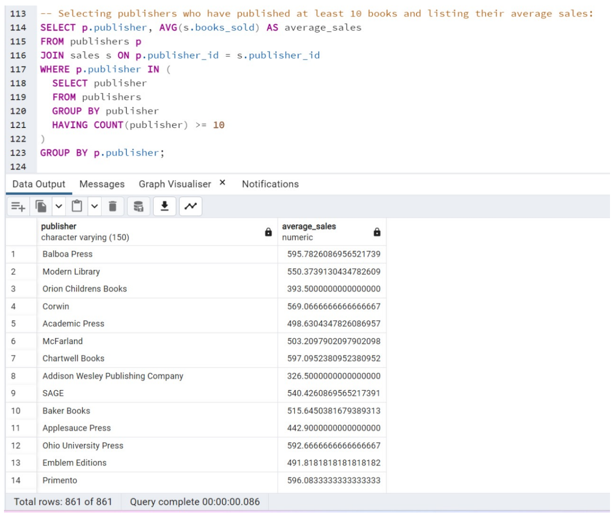
```sql
-- Selecting books whose reviews have a score above average review score:
SELECT b.title, p.publisher, r.review_score
FROM books b
JOIN reviews r ON b.book_id = r.book_id
JOIN publishers p ON b.book_id = p.book_id
WHERE r.review_score > (
  SELECT AVG(review_score)
  FROM reviews
)
ORDER BY r.review_score DESC;
```

Data Output   Messages   Graph Visualiser ×   Notifications

	title text	publisher character varying (150
1	Alices Adventures in Wonderland (Aladdin Classics)	Simon and Schuster
2	23 Minutes In Hell: One Mans Story About What He Saw Heard and Felt in that Place of Torment	Charisma Media
3	Buried (proof)	Harlequin
4	A Free Man of Color	Bantam
5	Best of Simple	Americas Test Kitche
6	Beggars in Spain	Harper Collins
7	Bridget Jones: The Edge of Reason	Penguin
8	A Single Womans Parenting Journey : Survival Tidbits	Professional Publishi
9	BloodRune	Author House
10	Banishment	A&amp;C Black
11	16 Lighthouse Road (Cedar Cove Book 1)	MIRA
12	Bowdrie	Bantam
13	Buffett: The Making of an American Capitalist	Random House
14	A Christmas Carol (Enriched Classics (Pocket))	Pocket Classics

Total rows: 1000 of 586493   Query complete 00:00:00.848

Fig. 10. Query to select books with above average review score



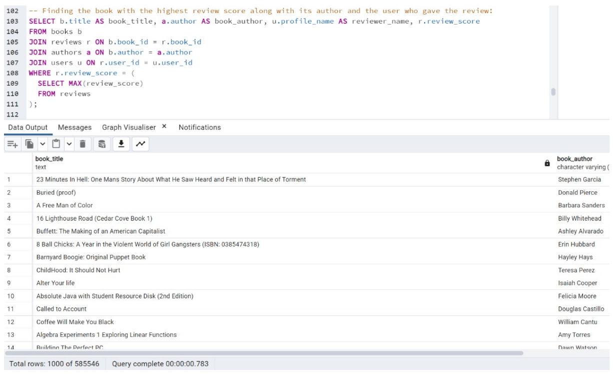
```sql
-- Selecting publishers who have published at least 10 books and listing their average sales:
SELECT p.publisher, AVG(s.books_sold) AS average_sales
FROM publishers p
JOIN sales s ON p.publisher_id = s.publisher_id
WHERE p.publisher IN (
  SELECT publisher
  FROM publishers
  GROUP BY publisher
  HAVING COUNT(publisher) >= 10
)
GROUP BY p.publisher;
```

Data Output   Messages   Graph Visualiser ×   Notifications

	publisher character varying (150)	average_sales numeric
1	Balboa Press	595.7826086956521739
2	Modern Library	550.3739130434782609
3	Orion Childrens Books	393.5000000000000000
4	Corwin	569.0666666666666667
5	Academic Press	498.6304347826086957
6	McFarland	503.2097902097902098
7	Chartwell Books	597.0952380952380952
8	Addison Wesley Publishing Company	326.5000000000000000
9	SAGE	540.4260869565217391
10	Baker Books	515.6450381679389313
11	Applesauce Press	442.9000000000000000
12	Ohio University Press	592.6666666666666667
13	Emblem Editions	491.8181818181818182
14	Primento	596.0833333333333333

Total rows: 861 of 861   Query complete 00:00:00.086

Fig. 11. Query to select publishers and their average sales for publishers with at least 10 books



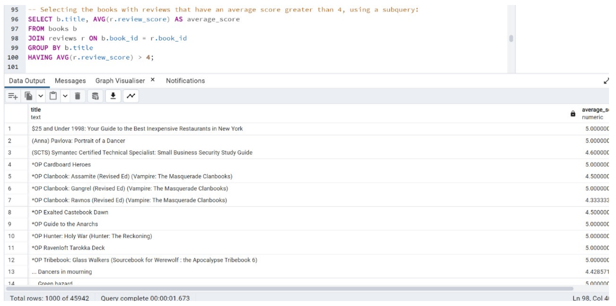
```sql
-- Finding the book with the highest review score along with its author and the user who gave the review:
SELECT b.title AS book_title, a.author AS book_author, u.profile_name AS reviewer_name, r.review_score
FROM books b
JOIN reviews r ON b.book_id = r.book_id
JOIN authors a ON b.author = a.author
JOIN users u ON r.user_id = u.user_id
WHERE r.review_score = (
  SELECT MAX(review_score)
  FROM reviews
);
```

Data Output   Messages   Graph Visualiser ×   Notifications

	book_title text	book_author character varying (
1	23 Minutes In Hell: One Mans Story About What He Saw Heard and Felt in that Place of Torment	Stephen Garcia
2	Buried (proof)	Donald Pierce
3	A Free Man of Color	Barbara Sanders
4	16 Lighthouse Road (Cedar Cove Book 1)	Billy Whitehead
5	Buffett: The Making of an American Capitalist	Ashley Alvarado
6	8 Ball Chicks: A Year in the Violent World of Girl Gangsters (ISBN: 0385474318)	Erin Hubbard
7	Barnyard Boogie: Original Puppet Book	Hayley Hays
8	ChildHood: It Should Not Hurt	Teresa Perez
9	Alter Your life	Isaiah Cooper
10	Absolute Java with Student Resource Disk (2nd Edition)	Felicia Moore
11	Called to Account	Douglas Castillo
12	Coffee Will Make You Black	William Cantu
13	Algebra Experiments 1 Exploring Linear Functions	Amy Torres
14	Building The Perfect PC	Dawn Watson

Total rows: 1000 of 585546   Query complete 00:00:00.783

Fig. 12. Query to select books, author and users who gave the review with highest score



```
-- Selecting the books with reviews that have an average score greater than 4, using a subquery:
SELECT b.title, AVG(r.review_score) AS average_score
FROM books b
JOIN reviews r ON b.book_id = r.book_id
GROUP BY b.title
HAVING AVG(r.review_score) > 4;
```

Data Output | Messages | Graph Visualiser | Notifications

	title text	average_s numeric
1	$25 and Under 1998: Your Guide to the Best Inexpensive Restaurants in New York	5.00000C
2	(Anna) Pavlova: Portrait of a Dancer	5.00000C
3	(SCTS) Symantec Certified Technical Specialist: Small Business Security Study Guide	4.60000C
4	*OP Cardboard Heroes	5.00000C
5	*OP Clanbook: Assamite (Revised Ed) (Vampire: The Masquerade Clanbooks)	4.50000C
6	*OP Clanbook: Gangrel (Revised Ed) (Vampire: The Masquerade Clanbooks)	5.00000C
7	*OP Clanbook: Ravnos (Revised Ed) (Vampire: The Masquerade Clanbooks)	4.333333
8	*OP Exalted Castebook Dawn	4.50000C
9	*OP Guide to the Anarchs	5.00000C
10	*OP Hunter: Holy War (Hunter: The Reckoning)	5.00000C
11	*OP Ravenloft Tarokka Deck	5.00000C
12	*OP Tribebook: Glass Walkers (Sourcebook for Werewolf : the Apocalypse Tribebook 6)	5.00000C
13	... Dancers in mourning	4.428571
14	... Green hazard	5.00000C

Total rows: 1000 of 45942 Query complete 00:00:01.673 Ln 98, Col 4

Fig. 13. Query to select books, reviews with average score ¿ 4 using a subquery

VIII. Query Optimization for complex queries

A. Query 1

SELECT Users.profile_name, Books.title, Reviews.review_summary, Reviews.review_score
FROM Users
JOIN Reviews ON Users.user_id = Reviews.user_id
JOIN Books ON Reviews.book_id = Books.book_id
WHERE Reviews.review_score = 5
ORDER BY Reviews.review_score DESC, Reviews.review_time DESC;

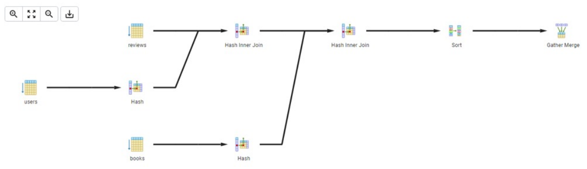


Fig. 14. Select profile name, title, review summary and review score without indexes

Proposed Solution:
Add Indexes on Reviews.review_score, and Reviews.review_time and Limiting the number of rows returned.
CREATE INDEX idx_review_score ON Reviews(review_score);
CREATE INDEX idx_review_time ON Reviews(review_time);
SELECT Users.profile_name, Books.title, Reviews.review_summary, Reviews.review_score
FROM Users
JOIN Reviews ON Users.user_id = Reviews.user_id
JOIN Books ON Reviews.book_id = Books.book_id
WHERE Reviews.review_score = 5
ORDER BY Reviews.review_score DESC, Reviews.review_time DESC
LIMIT 100;

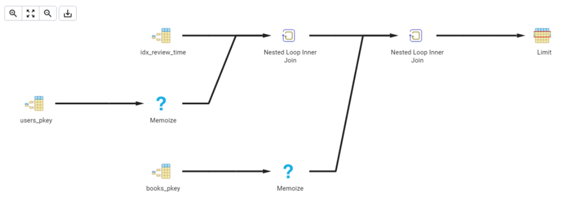
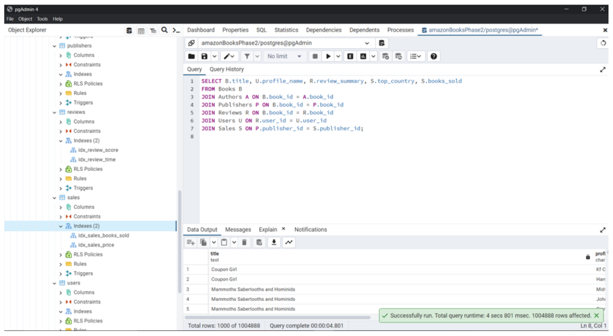


Fig. 15. Select profile name, title, review summary and review score with indexes

After optimization, there is substantial improvement in execution time, the 1st query took 2.30 seconds and after optimization, the query was executed in 0.53 seconds

B. Query 2

Joining multiple tables to extract comprehensive review and sales information for books can lead to inefficiency.



Dashboard | Properties | SQL | Statistics | Dependencies | Dependents | Processes | amazonBooksPhase2/postgres@pgAdmin*

amazonBooksPhase2/postgres@pgAdmin

```
SELECT B.title, U.profile_name, R.review_summary, S.top_country, S.books_sold
FROM Books B
JOIN Authors A ON B.book_id = A.book_id
JOIN Publishers P ON B.book_id = P.book_id
JOIN Reviews R ON B.book_id = R.book_id
JOIN Users U ON R.user_id = U.user_id
JOIN Sales S ON P.publisher_id = S.publisher_id;
```

Data Output | Messages | Explain | Notifications

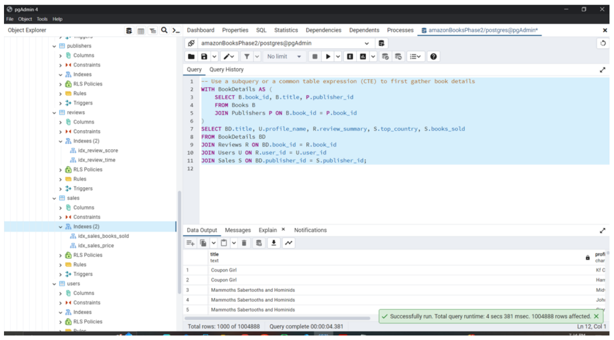
	title text
1	Coupon Girl
2	Coupon Girl
3	Mammoths Sabertooths and Hominids
4	Mammoths Sabertooths and Hominids
5	Mammoths Sabertooths and Hominids

✓ Successfully run. Total query runtime: 4 secs 801 msec. 1004888 rows affected.

Total rows: 1000 of 1004888 Query complete 00:00:04.801 Ln 8, Col 1

Fig. 16. Select profile name, title, review summary and review score without subquery

Proposed Solution:
To address the inefficiency, we can simplify the joining strategy by using intermediary results or views, which reduces the complexity when executed.



Dashboard | Properties | SQL | Statistics | Dependencies | Dependents | Processes | amazonBooksPhase2/postgres@pgAdmin*

amazonBooksPhase2/postgres@pgAdmin

```
-- Use a subquery or a common table expression (CTE) to first gather book details
WITH BookDetails AS (
    SELECT B.book_id, B.title, P.publisher_id
    FROM Books B
    JOIN Publishers P ON B.book_id = P.book_id
)
SELECT BD.title, U.profile_name, R.review_summary, S.top_country, S.books_sold
FROM BookDetails BD
JOIN Reviews R ON BD.book_id = R.book_id
JOIN Users U ON R.user_id = U.user_id
JOIN Sales S ON BD.publisher_id = S.publisher_id;
```

Data Output | Messages | Explain | Notifications

	title text
1	Coupon Girl
2	Coupon Girl
3	Mammoths Sabertooths and Hominids
4	Mammoths Sabertooths and Hominids
5	Mammoths Sabertooths and Hominids

✓ Successfully run. Total query runtime: 4 secs 381 msec. 1004888 rows affected.

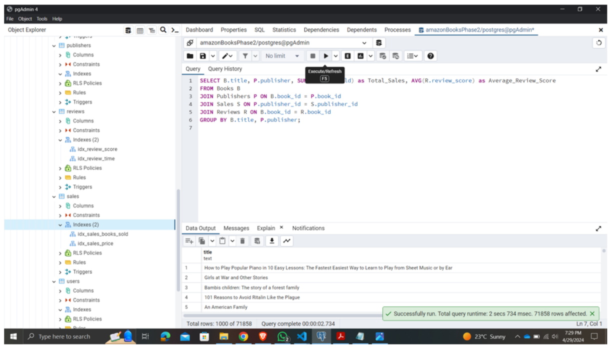
Total rows: 1000 of 1004888 Query complete 00:00:04.381 Ln 12, Col 1

Fig. 17. Select profile name, title, review summary and review score with subquery

By reducing the complexity and focusing on the necessary data, there is an improvement on execution time as shown. This approach avoids repeated joins and minimizes the amount of data being processed in later steps.

C. Query 3

This query is used to extract sales data for each book, including the total sales, average review scores, and publisher information, which can be slow due to the multiple layers of joins needed to connect sales to books through publishers.

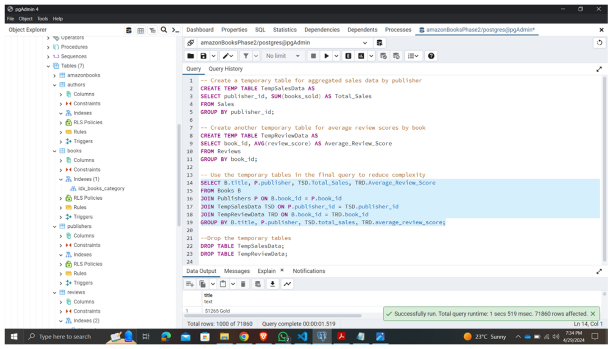


```
SELECT B.title, P.publisher, SU[illegible]ld) as Total_Sales, AVG(R.review_score) as Average_Review_Score
FROM Books B
JOIN Publishers P ON B.book_id = P.book_id
JOIN Sales S ON P.publisher_id = S.publisher_id
JOIN Reviews R ON B.book_id = R.book_id
GROUP BY B.title, P.publisher;
```

```
  title
  text
1 How to Play Popular Piano in 10 Easy Lessons: The Fastest Easiest Way to Learn to Play from Sheet Music or by Ear
2 Girls at War and Other Stories
3 Bambis children: The story of a forest family
4 101 Reasons to Avoid Ritalin Like the Plague
5 An American Family

Successfully run. Total query runtime: 2 secs 734 msec. 71858 rows affected.
Total rows: 1000 of 71858    Query complete 00:00:02.734
```

Fig. 18. Select profile name, title, review summary and review score without temporary tables

Proposed Solution:
We can use temporary tables to handle complex aggregations ahead of time, simplifying the final query and improving performance, especially useful in scenarios with large datasets and multi-layered relational data.



```
-- Create a temporary table for aggregated sales data by publisher
CREATE TEMP TABLE TempSalesData AS
SELECT publisher_id, SUM(books_sold) AS Total_Sales
FROM Sales
GROUP BY publisher_id;

-- Create another temporary table for average review scores by book
CREATE TEMP TABLE TempReviewData AS
SELECT book_id, AVG(review_score) AS Average_Review_Score
FROM Reviews
GROUP BY book_id;

-- Use the temporary tables in the final query to reduce complexity
SELECT B.title, P.publisher, TSD.Total_Sales, TRD.Average_Review_Score
FROM Books B
JOIN Publishers P ON B.book_id = P.book_id
JOIN TempSalesData TSD ON P.publisher_id = TSD.publisher_id
JOIN TempReviewData TRD ON B.book_id = TRD.book_id
GROUP BY B.title, P.publisher, TSD.total_sales, TRD.average_review_score;

--Drop the temporary tables
DROP TABLE TempSalesData;
DROP TABLE TempReviewData;
```

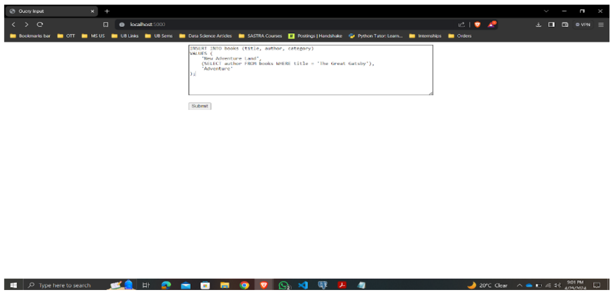
```
  title
  text
1 $1265 Gold

Successfully run. Total query runtime: 1 secs 519 msec. 71860 rows affected.
Total rows: 1000 of 71860    Query complete 00:00:01.519
```

Fig. 19. Select profile name, title, review summary and review score with temporary tables

Pre-aggregating sales and review data in temporary tables reduces the data volume for final joins, speeding up execution by linking books and publishers directly to summarized data. This approach joins on smaller datasets and enhances resource management, minimizing database load and complexity.

IX. Bonus Task: Website Design

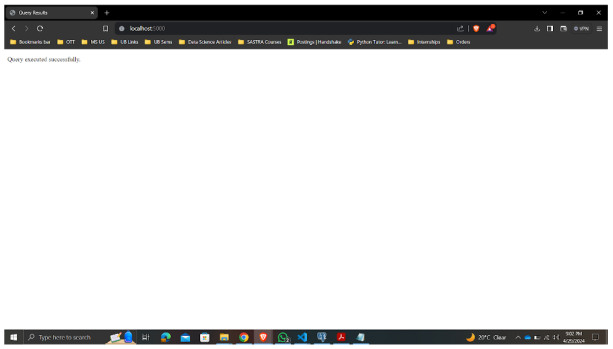
We've developed a user-friendly website that simplifies the querying process. Users simply input their query into the provided textbox and then click the 'Submit' button to execute it. This streamlined interface ensures a straightforward experience, allowing users to quickly access the information they need with minimal effort.



```
INSERT INTO books (title, author, category)
VALUES (
    'New Adventure Land',
    (SELECT author FROM books WHERE title = 'The Great Gatsby'),
    'Adventure'
);
```

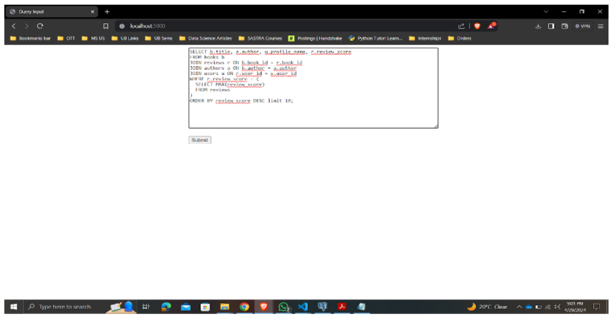
Fig. 20. Website page to insert query and submit

Upon submission of the query through the website's interface, the website will promptly execute the query as specified. For insert, delete, and update queries, upon successful execution, the website will return the message "Query executed successfully" to indicate that the operation was completed without any errors. This streamlined feedback mechanism ensures users are promptly informed of the outcome of their query execution, facilitating a smooth and efficient user experience.



```
Query executed successfully.
```

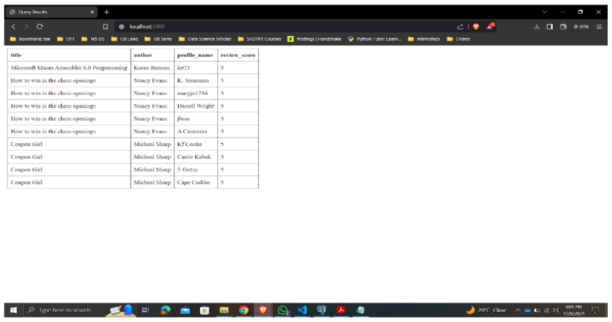
Fig. 21. Website page to insert query result



```
SELECT b.title, a.author, u.profile_name, r.review_score
FROM books b
JOIN reviews r ON b.book_id = r.book_id
JOIN authors a ON b.author = a.author
JOIN users u ON r.user_id = u.user_id
WHERE r.review_score = (
  SELECT MAX(review_score)
  FROM reviews
)
ORDER BY review_score DESC limit 10;
```

Fig. 22. Website page to select query and submit

For select queries, upon execution, the website will generate and display the query result in the form of a table, adhering to the structure and formatting specified in the query. This approach ensures that users receive a clear and comprehensive representation of the queried data, facilitating easy analysis and interpretation. The displayed table will present the retrieved data in a structured format, allowing users to efficiently review and explore the results. Additionally, features such as sorting, filtering, and pagination may be implemented to enhance the usability and functionality of the displayed table, enabling users to interact with the data effectively. Overall, the website will seamlessly execute select queries and present the query results in a user-friendly and accessible manner, catering to the diverse needs of users.



title	author	profile_name	review_score
Microsoft Macro Assembler 6.0 Programming	Karen Benson	int21	5
How to win in the chess openings	Nancy Evans	K. Stearman	5
How to win in the chess openings	Nancy Evans	maryjo1234	5
How to win in the chess openings	Nancy Evans	Darrell Wright	5
How to win in the chess openings	Nancy Evans	jbros	5
How to win in the chess openings	Nancy Evans	A Customer	5
Coupon Girl	Micheal Sharp	Kf Cooke	5
Coupon Girl	Micheal Sharp	Carrie Kabak	5
Coupon Girl	Micheal Sharp	J. Getze	5
Coupon Girl	Micheal Sharp	Cape Codder	5

Fig. 23. Website page to select query result

In the event of incorrect or invalid queries, the website will display a message indicating ”No records found” if the table is empty else if there is error in query it will show error message. This response serves to inform users that the query did not yield any results due to errors in its formulation or execution. By providing this feedback, users are alerted to the issue and can take appropriate action, such as refining their query or seeking assistance if needed. Additionally, displaying a clear error message helps maintain user confidence and ensures a positive user experience, even in the face of query errors.

The Project Github Repository: https://github.com/anudeepkotaru/DMQL_Project/tree/main

X. Contributions

- Satya Anudeep Kotaru:33.33%
- Bhavya Teja Inturi: 33.33%
- Vineet Sreeram:33.33%

Each team member has equally contributed.