BIG DATA ASSIGNMENT 2 REPORT

KELVIN ASU EKUR
k.ekuri@innopolis.university

ABSTRACT

Effective and scalable database management systems are now essential due to the expansion of big data applications. Utilising a real-world e-commerce dataset, this report compares **relational** and **NoSQL** databases. We used **Neo4j**, **PostgreSQL**, and **MongoDB** to implement three different database models, each tailored to a particular data storage paradigm. These databases' **analytical capabilities**, **query performance**, and **data modelling** are all assessed.
We used **Hyperfine** to measure execution times for three analytical queries across **PostgreSQL** and **MongoDB** in order to benchmark query execution performance. The findings show that while PostgreSQL offers better relational integrity and data consistency, MongoDB performs better in the majority of queries because of its schema-less architecture and optimised document storage. Neo4j, which is intended for graph-based queries, is assessed independently for its capacity to use social network analysis and relationships.
The implications of database selection in e-commerce applications are also covered in this report, with a focus on trade-offs between **data consistency, scalability**, and **performance**. Businesses can use the results as a guide to select the best database model for their needs based on particular application requirements.

Keywords: relational databases, non-relational databases, data modelling, query performance, Hyperfine, PostgreSQL, MongoDB, Neo4j

1 DATA MODELLING

1.1 Relational model

The PostgreSQL (relational database) data model for the e-commerce database, which uses foreign key relationships to guarantee data consistency and integrity.
The `campaigns` table is the source of promotional messages and contains information about marketing campaigns that are uniquely identified by id. By using `campaign_id` and `client_id` to link campaigns to customers, the messages table tracks engagement metrics like opens, clicks, and purchases. Information about customers, uniquely identified by `client_id`, as well as the date of their first purchase and related user data, are kept in the `client_first_purchase` table. With a foreign key reference to `user_id` in `client_first_purchase`, the events table logs user activities such as views, clicks, and purchases, as well as customer interactions on the platform. Users' social network connections are represented in the friends table, which uses a composite primary key (`friend1, friend2`) to enforce bidirectional relationships.

Referential integrity is guaranteed by the model, which enables structured queries to analyze user engagement, campaign efficacy, and social influence on consumer behaviour.

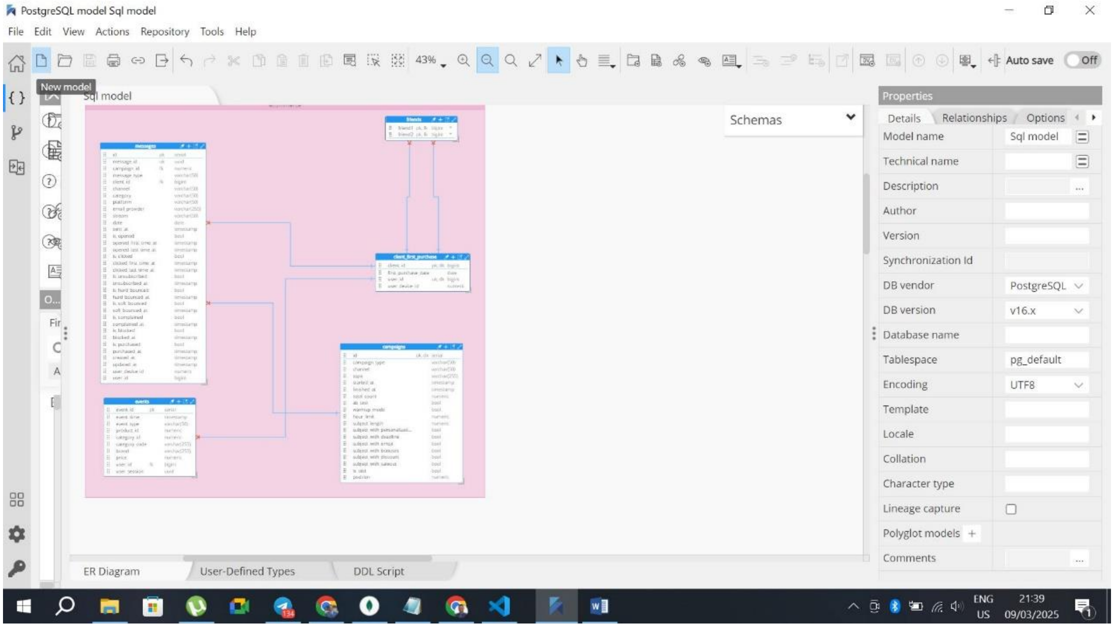


Figure 1: Model for relational Database

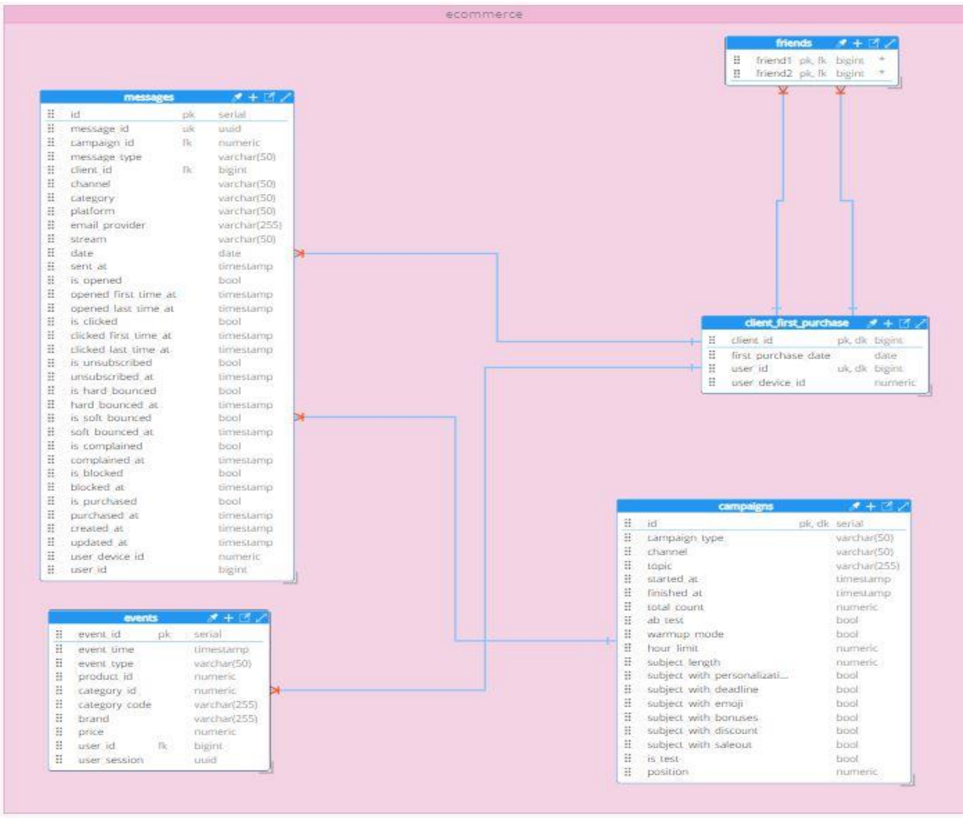


Figure 2: Data Model for relational database

1.2 NoSQL (Mongo DB)

The MongoDB model follows a document-oriented schema where related data is either embedded within documents or referenced across collections using unique identifiers. The core entity, messages, serves as the central node connecting different datasets. campaigns are linked to messages, representing marketing campaigns that send promotional content to users. The client_first_purchase collection stores customer data, which is referenced in messages to track user engagement and purchases. The events collection logs user interactions such as views, clicks, and purchases, linking back to client_first_purchase for user behavior analysis. Lastly, the friends collection models social relationships between users, supporting analyses on how peer influence affects engagement and conversions. This schema optimizes fast reads and writes, leveraging MongoDB's flexibility for efficient querying and aggregation.

Figure 3: MongoDB Model

1.3 Graph Database (Neo4J)

Campaigns, clients, products, messages, and events are all represented as nodes in the Neo4j data model, which is organised as a graph database. User interactions are defined by the relationships between these nodes. The SENT relationship links campaigns to messages, indicating that campaigns send marketing messages. With relationships to campaigns (ENGAGED_IN), products (VIEWED, ADDED_TO_CART, PURCHASED), and friends (FRIENDS_WITH), clients play a key role in the model and enable social influence analysis.

Through the RECEIVED_BY (which links messages to clients) and RESULTED_IN_PURCHASE (which links successful campaign messages to purchases) relationships, messages monitor user engagement. Events provide information about user behaviour by recording user interactions (PERFORMED) and their relationships to products (RELATED_TO_PRODUCT).
To maintain data integrity, specific constraints are applied to the following important properties: id, client_id, product_id, message_id, and event_id. Campaign effectiveness tracking, social influence analysis, and recommendation query efficiency are all improved by this graph structure.

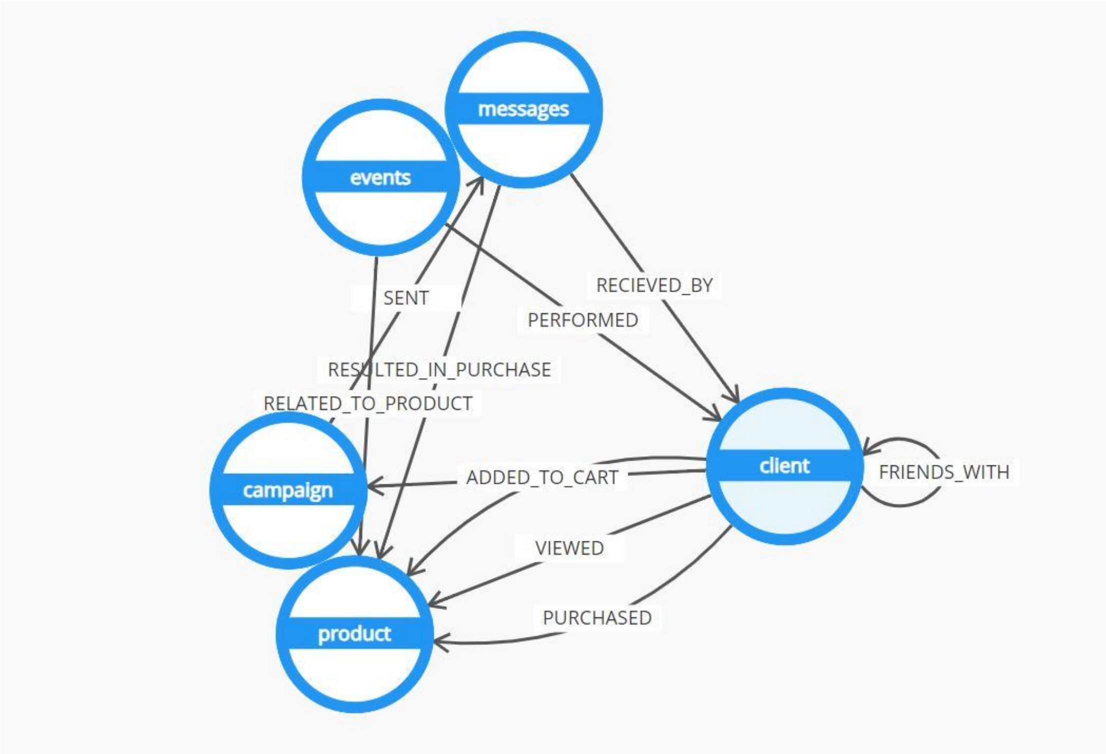


Figure 4: Neo4j Model

2 Database Implementation

2.1 PostgreSQL

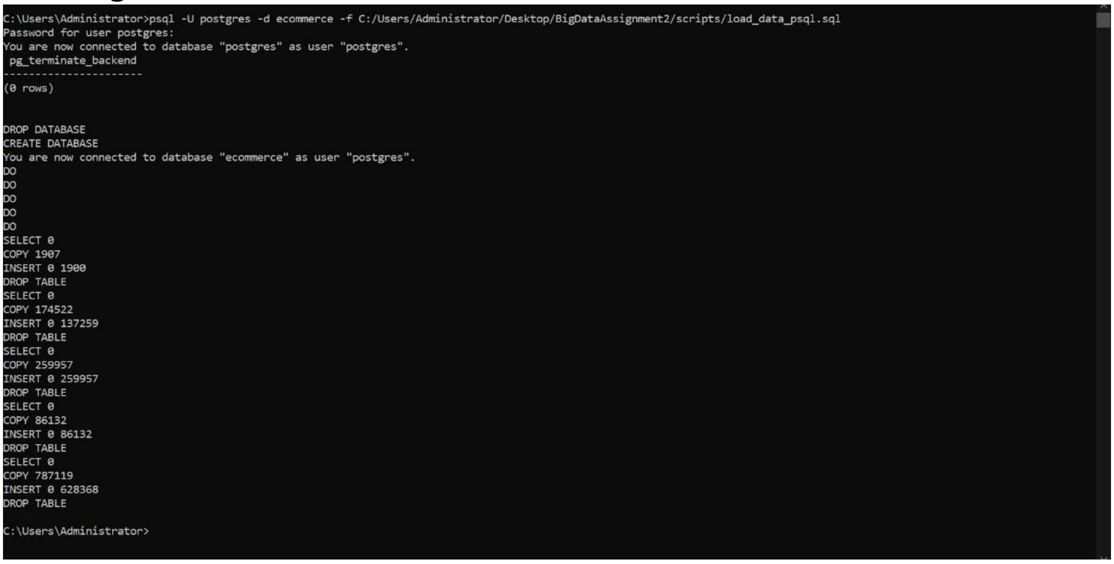


Figure 5: Load data into postgre database

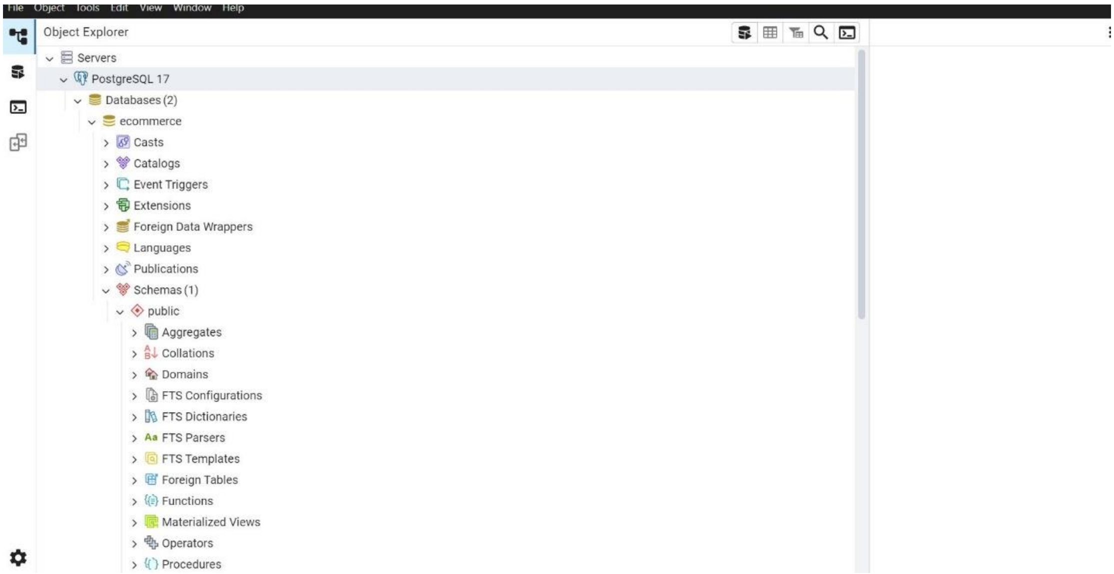
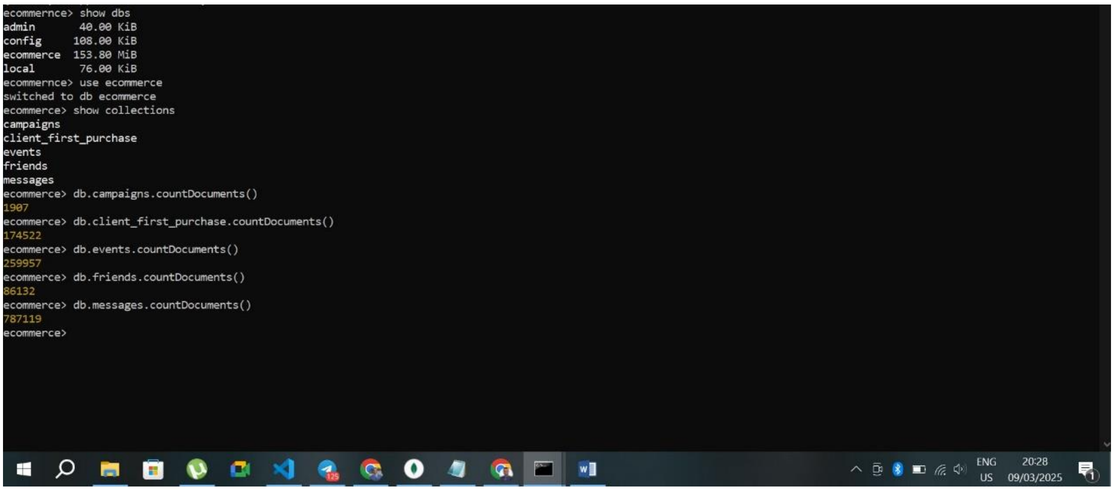


Figure 6: Relational database created with script

2.2 MongoDB



3. Alternative model

A hybrid data model to enhance the earlier models by addressing the drawbacks of relational databases (PostgreSQL), document-based databases (MongoDB), and graph databases (Neo4j) while combining their advantages is suggested. The hybrid model could optimize query efficiency, scalability, and analytical depth by combining structured, semi-structured, and highly connected data.

EXPERIMENTAL SETUP

Operating system:	Windows 10 Pro 22H2 (OS Build 19045.3803)
RAM	16.0 GB (15.8 GB usable)
CPU	Inter ® Core ™ i5-8265U@1.60GHz 180GHz
All software components installed separately on windows (no virtualization or Docker)	
PostgreSQL	psql (PostgreSQL): 17.4
Mongo DB version	db version v8. 0. 4
Neo4J version downloaded:	1.6.1 Java JRE: 1.8.0_202 **(Neo4j not running)**
Hackolade studio	8.0.3
Hyperfine	1.19.0

EXPERIMENTAL RESULTS

Data Analysis:

PostgreSQL

Query 1: Find Campaign Effectiveness by tracking purchases

```
C: > Users > Administrator > Desktop > BigDataAssignment2 > scripts > q1.sql
-- Find campaign effectiveness by tracking purchases
SELECT
    c.id AS campaign_id,
    c.topic AS campaign_topic,
    COUNT(DISTINCT m.client_id) AS total_recipients,
    COUNT(DISTINCT e.user_id) AS total_purchasers,
    ROUND((COUNT(DISTINCT e.user_id)::DECIMAL / COUNT(DISTINCT m.client_id)) * 100, 2) AS conversion_rat
FROM messages m
JOIN campaigns c ON m.campaign_id = c.id
LEFT JOIN events e ON m.client_id = e.user_id AND e.event_type = 'purchase'
GROUP BY c.id, c.topic
ORDER BY conversion_rate DESC;
```

Figure 7: q1.sql

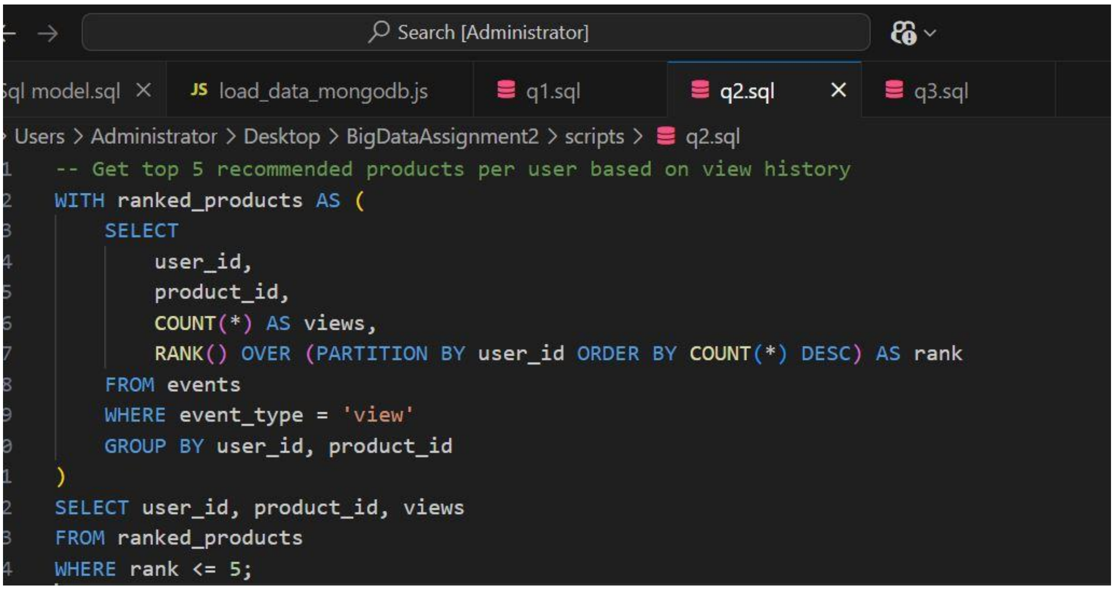
Results of the queries are stored in csv files in the parent directory of the Analysis results folder

	A	B	C	D	E	F
1	campaign	campaign_	total_recip	total_purc	conversion_rate	
2	26	order ship	2683	0	0	
3	27	order crea	13520	0	0	
4	28	order crea	386	0	0	
5	29	order canc	10809	0	0	
6	31	order reac	282	0	0	
7	32	order reac	12791	0	0	
8	33	order pick	197	0	0	
9	54	order crea	782	0	0	
10	55	order reac	723	0	0	
11	58	order crea	157	0	0	
12	59	order ship	111	0	0	
13	63	sale out	3960	0	0	
14	64	sale out	21422	0	0	
15	78	sale out	5676	0	0	
16	79	sale out	21883	0	0	
17	89		4365	0	0	
18	105	order reac	98	0	0	
19	110	sale out	7261	0	0	
20	111	sale out	21645	0	0	
21	122	order ship	222	0	0	
22	123	order pick	9	0	0	

campaign_effectiveness

The query results indicate that none of the campaigns resulted in any purchases. All campaigns have a 0% conversion rate, which indicates that none of the customers who were contacted went on to make a purchase.

Query 2: **Get top 5 recommended products per user based on view history**



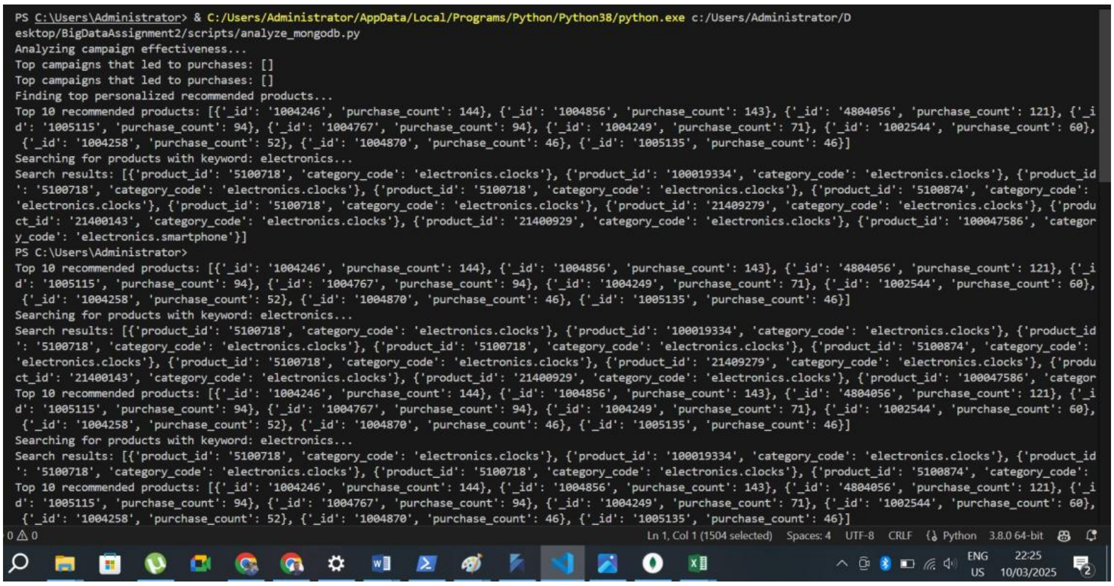

```sql
-- Get top 5 recommended products per user based on view history
WITH ranked_products AS (
    SELECT
        user_id,
        product_id,
        COUNT(*) AS views,
        RANK() OVER (PARTITION BY user_id ORDER BY COUNT(*) DESC) AS rank
    FROM events
    WHERE event_type = 'view'
    GROUP BY user_id, product_id
)
SELECT user_id, product_id, views
FROM ranked_products
WHERE rank <= 5;
```

Figure 8: products recommendation query in postgre

The result is stored
`Analysis_Results/personalized_recommendations.csv`
The goal of the query was to determine each user's top 5 recommended products based on their viewing history. The dataset includes 54,691 recommendations for 18,279 distinct products, made for 11,621 distinct users.

Before being listed among the top 5 recommendations, each suggested product was viewed by users an average of 2.37 times. This implies that the recommendation system is predicated on goods with which users regularly engage, guaranteeing tailored recommendations. To confirm their efficacy, it is crucial to conduct additional analysis on the rate at which these suggestions result in purchases.

Results of MongoDB queries:



Bench Marking

The hyperfine results were exported to csv files.

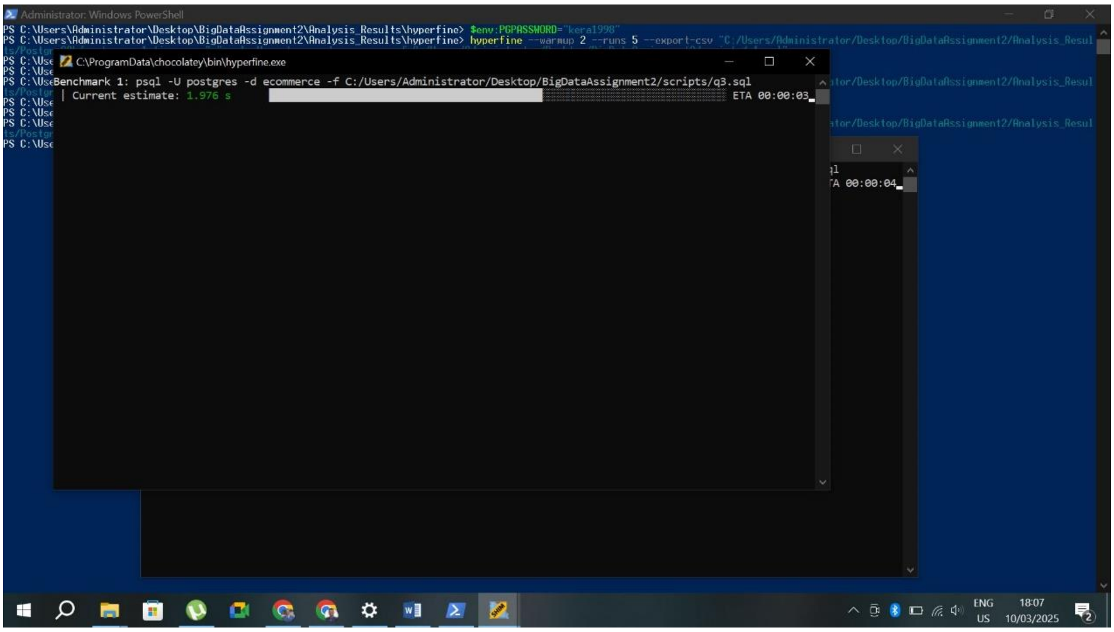


Figure 9: Hyperfine execution for psql

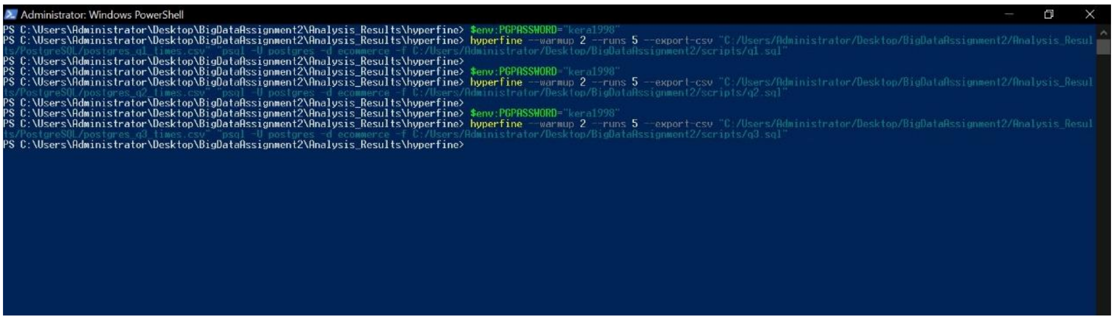


Figure 10: Postgre hyperfine execution

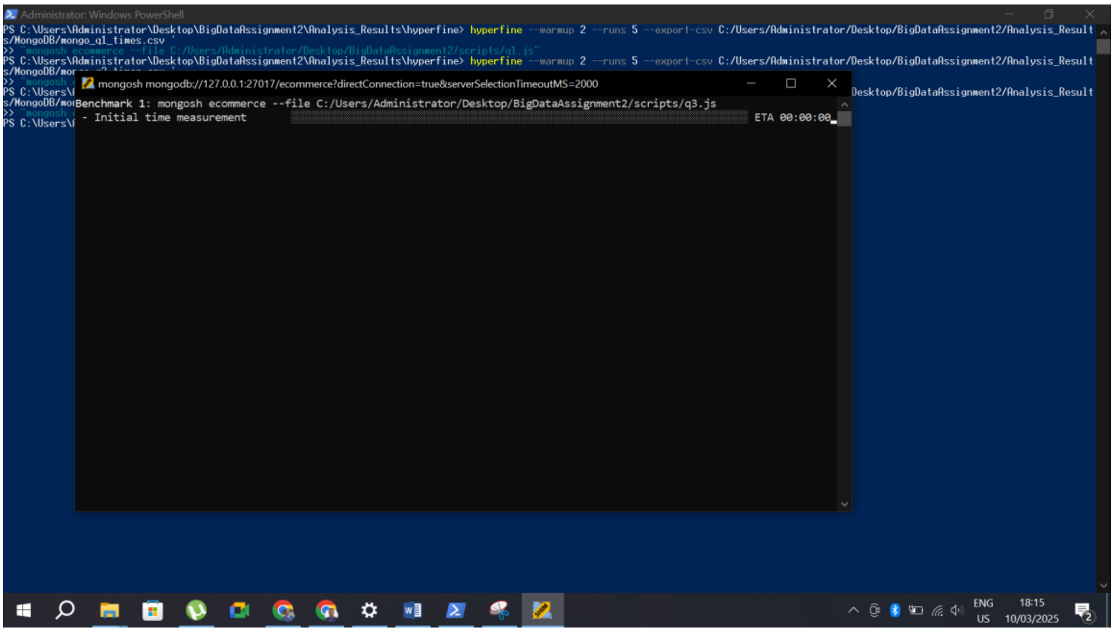
Postgre

command	mean	stddev	median	user	system	min	max
psql -U postgres -d ecommerce -f C:/Users/Administrator/Desktop/BigDataAssignment2/scripts/q1.sql	3.510874	0.690451	3.749069	0.058125	0.030625	2.776157	4.28663

	A	B	C	D	E	F	G	H
1	command	mean	stddev	median	user	system	min	max
2	psql -U postgres -d ecommerce -f C:/Users/Administrator/Desktop/BigDataAssignment2/scripts/q2.sql	1.330244	0.169643	1.312169	0.484688	0.097188	1.143497	1.590607
3								

	A	B	C	D	E	F	G	H
1	command	mean	stddev	median	user	system	min	max
2	psql -U postgres -d ecommerce -f C:/Users/Administrator/Desktop/BigDataAssignment2/scripts/q3.sql	1.80588	0.300948	1.714652	0.046875	0.04	1.505522	2.246677

Mongo DB



	A	B	C	D	E	F	G	H
1	command	mean	stddev	median	user	system	min	max
2	mongosh ecommerce --file C:/Users/Administrator/Desktop/BigDataAssignment2/scripts/q1.js	0.909686	0.012484	0.906203	0.364375	0.062188	0.897464	0.927526

	A	B	C	D	E	F	G	H
1	command	mean	stddev	median	user	system	min	max
2	mongosh ecommerce --file C:/Users/Administrator/Desktop/BigDataAssignment2/scripts/q2.js	0.48826	0.00721	0.487727	0.345625	0.088125	0.480786	0.500076

	A	B	C	D	E	F	G	H
1	command	mean	stddev	median	user	system	min	max
2	mongosh ecommerce --file C:/Users/Administrator/Desktop/BigDataAssignment2/scripts/q3.js	0.3663379	0.001449	0.366622	0.342813	0.0675	0.364343	0.367833

Table 1: Query execution times

	Average (s)	Std Dev (s)
PostgreSQL Q1	3.510874	0
PostgreSQL Q2	1.330244314	0
PostgreSQL Q3	1.8058802	0
MongoDB Q1	0.909685822	0
MongoDB Q2	0.488260332	0
MongoDB Q3	0.366337938	0

F

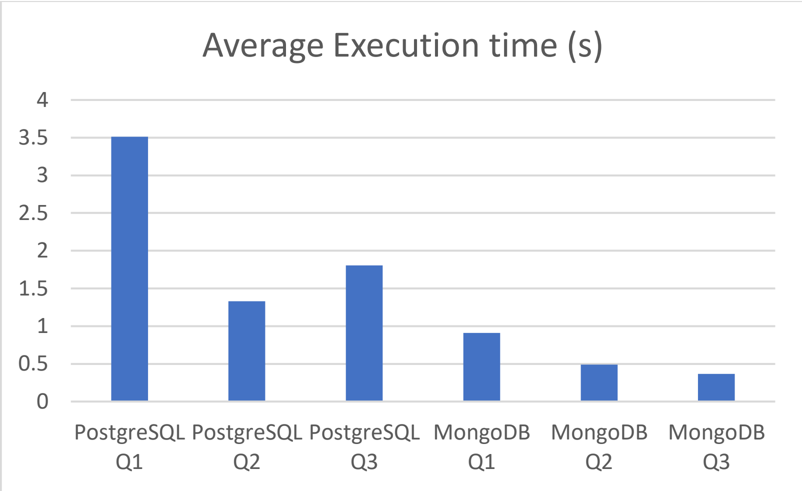


Figure 11: Query executions