Student Name: Shanikwa Haynes

Student ID:

Date: 03/31/2025

D597 MKN1 Task 1: Relational Database Design and Implementation

A.

1.

EcoMart is a growing online marketplace committed to promoting sustainable living by offering ethically sourced, eco-friendly products across categories such as groceries, apparel, and home goods. However, the organization currently experiences inefficiencies in managing product inventory, processing customer orders, and evaluating revenue performance. Without a relational database system, EcoMart struggles with data retrieval speed, security of sensitive order data, and scalability to support its expanding operations. By implementing a structured relational database, EcoMart can solve these challenges through improved data organization, secured storage, and scalable solutions tailored to its online retail operations.

2.

A relational database schema will be implemented with separate tables for Orders, Products, and Sales, ensuring data normalization, integrity, and improved performance. To address the identified business problem, a normalized relational database structure was designed. This structure includes three main tables: Orders, Products, and Sales. The Orders table stores customer and transaction data such as OrderDate, Country, and OrderPriority. The Products table contains product-specific data such as ItemType, UnitCost, and UnitPrice. The Sales table holds transactional outcomes, including UnitsSold, TotalRevenue, and TotalProfit, and uses foreign keys to relate back to Orders and Products. The schema adheres to third normal form (3NF), ensuring data consistency and eliminating redundancy.

3.

Implementing a relational database provides several benefits that address the business needs of EcoMart. First, structured tables and relationships between them ensure consistent, accurate data and reduce redundancy. Second, the database supports efficient data retrieval through indexing and optimization techniques. Third, security is enhanced via controlled access and backup strategies. Finally, the database model allows for future scalability, enabling EcoMart to expand its product lines and geographic coverage without degrading performance.

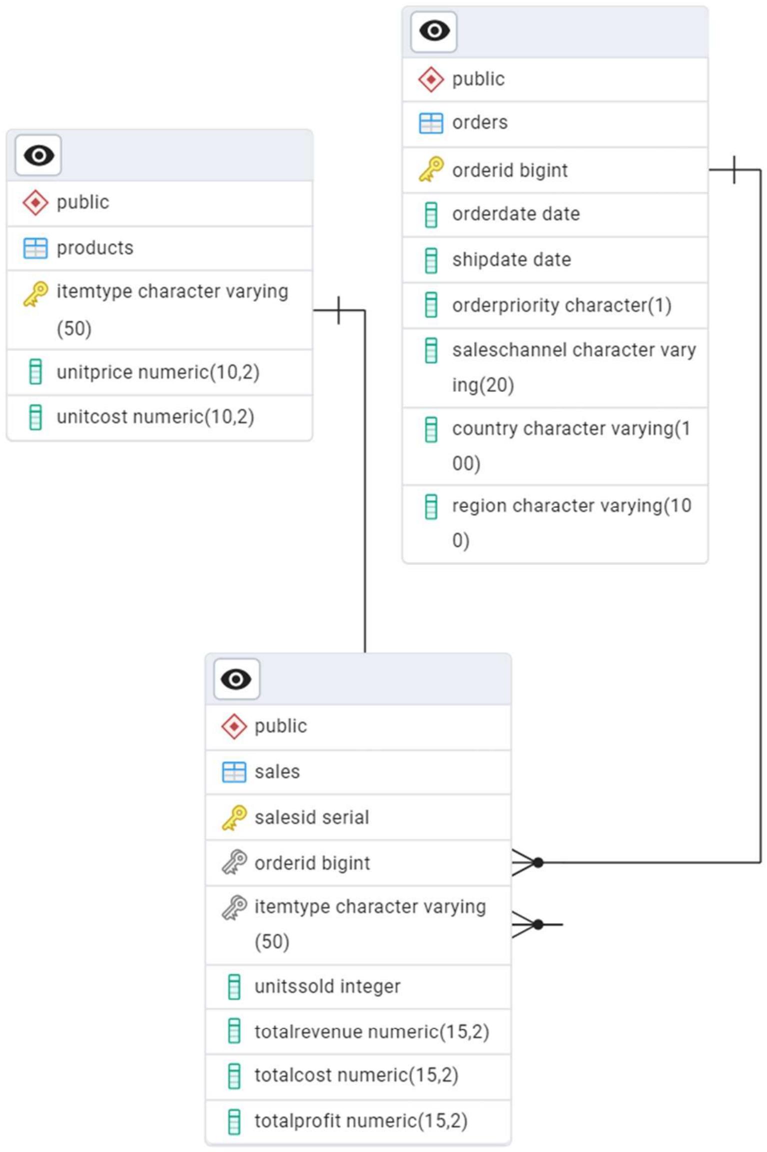
4.

The database will be used to store, retrieve, and manage key business data efficiently. The Orders table captures order details including customer location and priority, supporting operations and reporting. The Products table stores product attributes such as category, price, and cost, supporting product management and profitability analysis. The Sales table records financial results of transactions and serves as the core table for generating analytics. Each table supports real-time querying and aggregation, enabling EcoMart to answer business questions such as which products are top-selling or which regions are most profitable.

B.

A logical ERD is designed to represent relationships between key entities, ensuring 3rd Normal Form (3NF) for optimal data integrity. The logical data model follows third normal form (3NF) and defines key entities: Orders, Products, and Sales. Each table has a primary key: OrderID for Orders, ItemType for Products, and SalesID for Sales. The Sales table includes foreign keys referencing Orders.OrderID and Products.ItemType to maintain referential integrity. The structure allows for efficient joins and eliminates duplication. The model is complete and aligns with the database schema implemented in

PostgreSQL.


public
products
itemtype character varying (50)
unitprice numeric(10,2)
unitcost numeric(10,2)
public
orders
orderid bigint
orderdate date
shipdate date
orderpriority character(1)
saleschannel character varying(20)
country character varying(100)
region character varying(100)
public
sales
salesid serial
orderid bigint
itemtype character varying (50)
unitssold integer
totalrevenue numeric(15,2)
totalcost numeric(15,2)
totalprofit numeric(15,2)

C.

The database contains three primary tables (Orders, Products, and Sales) and associated indexes for efficient query execution. Each table was created using CREATE TABLE statements in PostgreSQL. Primary keys were assigned to uniquely identify records. Foreign keys in the Sales table enforce relational integrity between transactions, orders, and products. Indexes were created on frequently queried columns such as OrderDate, Region, ItemType, and TotalRevenue. The data from CSV files was mapped column-by-column into corresponding tables using pgAdmin4's import tool. File attributes such as headers, delimiters, and encoding were configured to ensure clean imports.

The proposed relational database consists of three main tables: Orders, Products, and Sales. These tables are designed to optimize query performance while maintaining data integrity.

Tables & Indexes

- Primary Keys:
  - Orders: OrderID
  - Products: ItemType
  - Sales: SalesID
- Foreign Keys:
  - Sales → Orders (OrderID)
  - Sales → Products (ItemType)
- Indexes:
  - Improve query performance for frequently accessed columns.

```
CREATE INDEX idx_order_date ON Orders(OrderDate);

CREATE INDEX idx_country ON Orders(Country);

CREATE INDEX idx_itemtype ON Products(ItemType);

CREATE INDEX idx_region ON Orders(Region);
```

D.

Scalability was addressed through several strategies. Vertical scalability ensures the database can support larger workloads by upgrading hardware resources. Horizontal scalability provides flexibility through data replication or sharding. Table partitioning is considered for future implementation to enhance performance on large datasets. Indexes were added to optimize read performance on columns used in WHERE, GROUP BY, and JOIN operations. These strategies allow the system to handle increasing volumes of sales and order data as EcoMart expands its customer base and inventory catalog.

E.

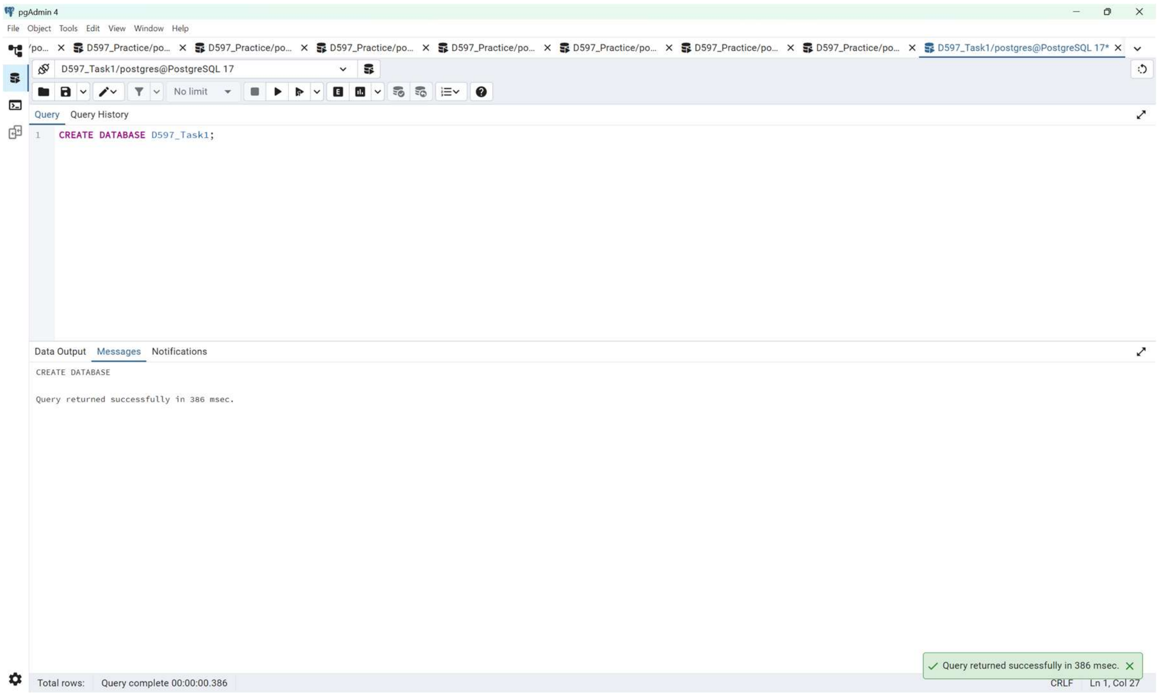
The database design incorporates strong privacy and security measures. Role-Based Access Control (RBAC) restricts access to sensitive data based on user roles, ensuring only authorized users can view or modify specific data. Data encryption secures confidential information such as order records. Audit logging tracks changes and access to the database to monitor suspicious behavior. Regular backups are scheduled to protect against data loss due to hardware failure or human error. These measures align with industry best practices and ensure data security and regulatory compliance.

Ensuring data security and compliance with regulations is a key priority. The database will implement:

- Role-Based Access Control (RBAC): Restricting user permissions to prevent unauthorized data access.
- Data Encryption: Securing sensitive data using encryption techniques.
- Audit Logging: Maintaining logs of database modifications and access.
- Regular Backups: Implementing scheduled backups for disaster recovery.

F1.

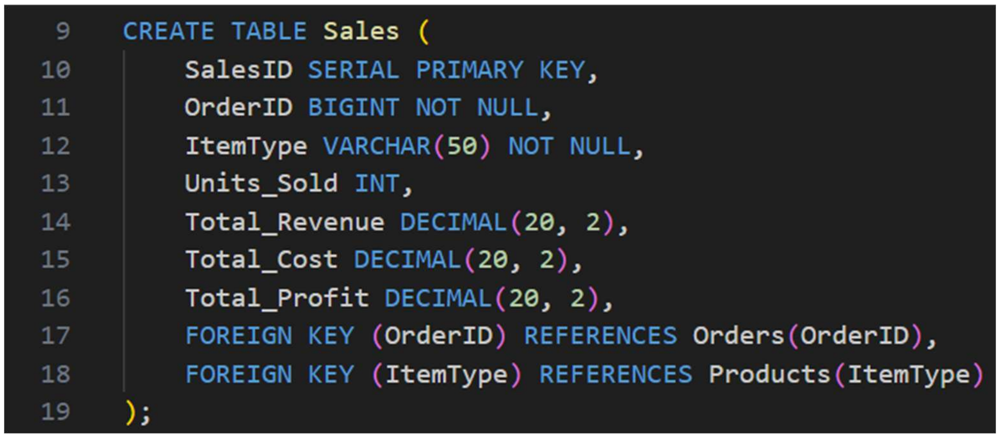
Here is the script to create a database based on the logical data model described above.


pgAdmin 4
File Object Tools Edit View Window Help
D597_Task1/postgres@PostgreSQL 17
No limit
Query Query History
Data Output Messages Notifications

```sql
CREATE DATABASE D597_Task1;
```

```
CREATE DATABASE

Query returned successfully in 386 msec.
```

Query returned successfully in 386 msec.
Total rows: Query complete 00:00:00.386
CRLF Ln 1, Col 27

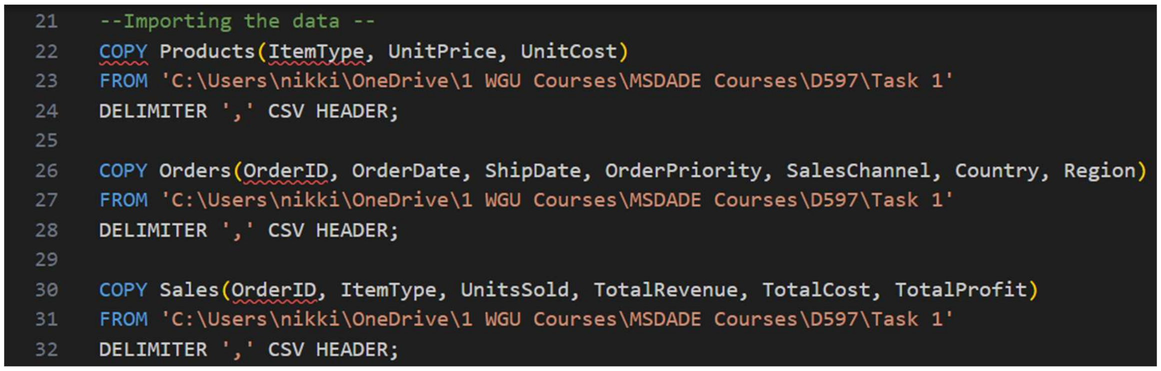


```sql
CREATE TABLE Sales (
    SalesID SERIAL PRIMARY KEY,
    OrderID BIGINT NOT NULL,
    ItemType VARCHAR(50) NOT NULL,
    Units_Sold INT,
    Total_Revenue DECIMAL(20, 2),
    Total_Cost DECIMAL(20, 2),
    Total_Profit DECIMAL(20, 2),
    FOREIGN KEY (OrderID) REFERENCES Orders(OrderID),
    FOREIGN KEY (ItemType) REFERENCES Products(ItemType)
);
```

F2.

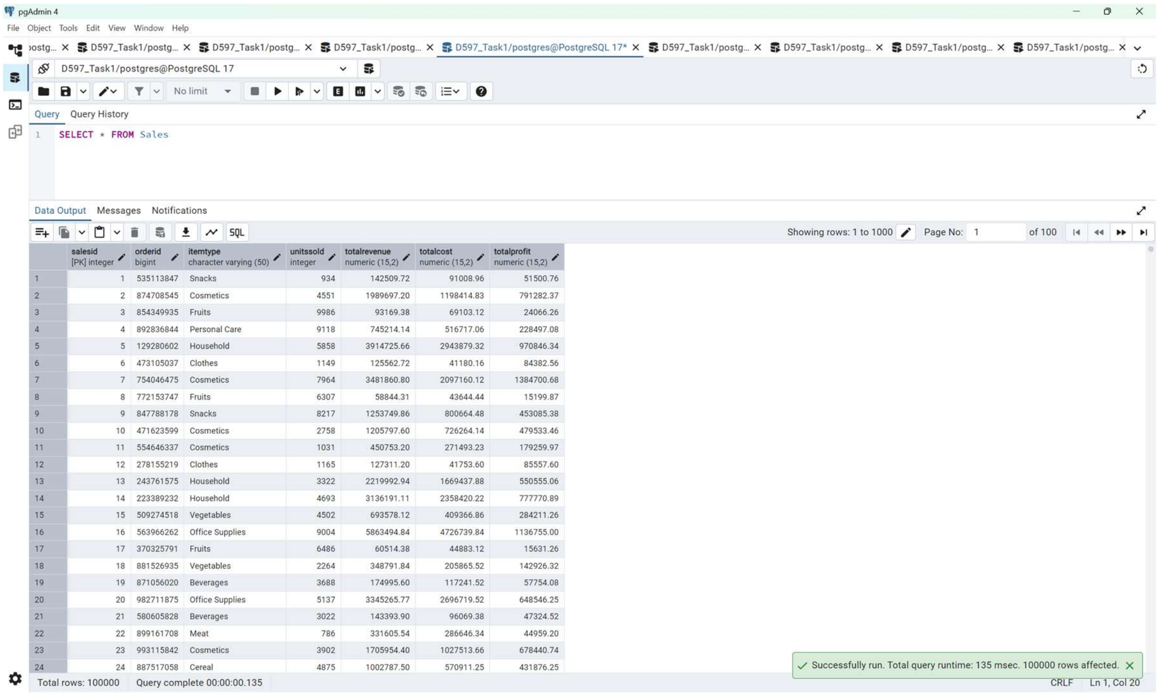
To ensure efficient data loading and maintain data integrity, CSV files containing orders, products, and sales records will be imported into the database.

Inserted the records from the CSV file into the Sales table:



```
--Importing the data --
COPY Products(ItemType, UnitPrice, UnitCost)
FROM 'C:\Users\nikki\OneDrive\1 WGU Courses\MSDADE Courses\D597\Task 1'
DELIMITER ',' CSV HEADER;

COPY Orders(OrderID, OrderDate, ShipDate, OrderPriority, SalesChannel, Country, Region)
FROM 'C:\Users\nikki\OneDrive\1 WGU Courses\MSDADE Courses\D597\Task 1'
DELIMITER ',' CSV HEADER;

COPY Sales(OrderID, ItemType, UnitsSold, TotalRevenue, TotalCost, TotalProfit)
FROM 'C:\Users\nikki\OneDrive\1 WGU Courses\MSDADE Courses\D597\Task 1'
DELIMITER ',' CSV HEADER;
```

Checked to ensure everything was imported properly. Below is a screenshot of the Sales populating:


pgAdmin 4
File Object Tools Edit View Window Help
D597_Task1/postgres@PostgreSQL 17*
D597_Task1/postgres@PostgreSQL 17
No limit
Query Query History
Data Output Messages Notifications
SQL

```
SELECT * FROM Sales
```

Showing rows: 1 to 1000 Page No: 1 of 100

	salesid [PK] integer	orderid bigint	itemtype character varying (50)	unitssold integer	totalrevenue numeric (15,2)	totalcost numeric (15,2)	totalprofit numeric (15,2)
1	1	535113847	Snacks	934	142509.72	91008.96	51500.76
2	2	874708545	Cosmetics	4551	1989697.20	1198414.83	791282.37
3	3	854349935	Fruits	9986	93169.38	69103.12	24066.26
4	4	892836844	Personal Care	9118	745214.14	516717.06	228497.08
5	5	129280602	Household	5858	3914725.66	2943879.32	970846.34
6	6	473105037	Clothes	1149	125562.72	41180.16	84382.56
7	7	754046475	Cosmetics	7964	3481860.80	2097160.12	1384700.68
8	8	772153747	Fruits	6307	58844.31	43644.44	15199.87
9	9	847788178	Snacks	8217	1253749.86	800664.48	453085.38
10	10	471623599	Cosmetics	2758	1205797.60	726264.14	479533.46
11	11	554646337	Cosmetics	1031	450753.20	271493.23	179259.97
12	12	278155219	Clothes	1165	127311.20	41753.60	85557.60
13	13	243761575	Household	3322	2219992.94	1669437.88	550555.06
14	14	223389232	Household	4693	3136191.11	2358420.22	777770.89
15	15	509274518	Vegetables	4502	693578.12	409366.86	284211.26
16	16	563966262	Office Supplies	9004	5863494.84	4726739.84	1136755.00
17	17	370325791	Fruits	6486	60514.38	44883.12	15631.26
18	18	881526935	Vegetables	2264	348791.84	205865.52	142926.32
19	19	871056020	Beverages	3688	174995.60	117241.52	57754.08
20	20	982711875	Office Supplies	5137	3345265.77	2696719.52	648546.25
21	21	580605828	Beverages	3022	143393.90	96069.38	47324.52
22	22	899161708	Meat	786	331605.54	286646.34	44959.20
23	23	993115842	Cosmetics	3902	1705954.40	1027513.66	678440.74
24	24	887517058	Cereal	4875	1002787.50	570911.25	431876.25

```
✓ Successfully run. Total query runtime: 135 msec. 100000 rows affected.
Total rows: 100000    Query complete 00:00:00.135    CRLF    Ln 1, Col 20
```

The business data for Orders, Products, and Sales was imported into the PostgreSQL database using pgAdmin4's built-in Import/Export tool rather than through manual INSERT INTO statements. This approach allowed for efficient bulk loading of the CSV files provided (Orders.csv, Products.csv, and Cleaned_Sales.csv) while preserving data structure and integrity.

Each CSV was mapped to its corresponding table by aligning columns during the import process. After import, SELECT * FROM queries were executed to verify successful data insertion.

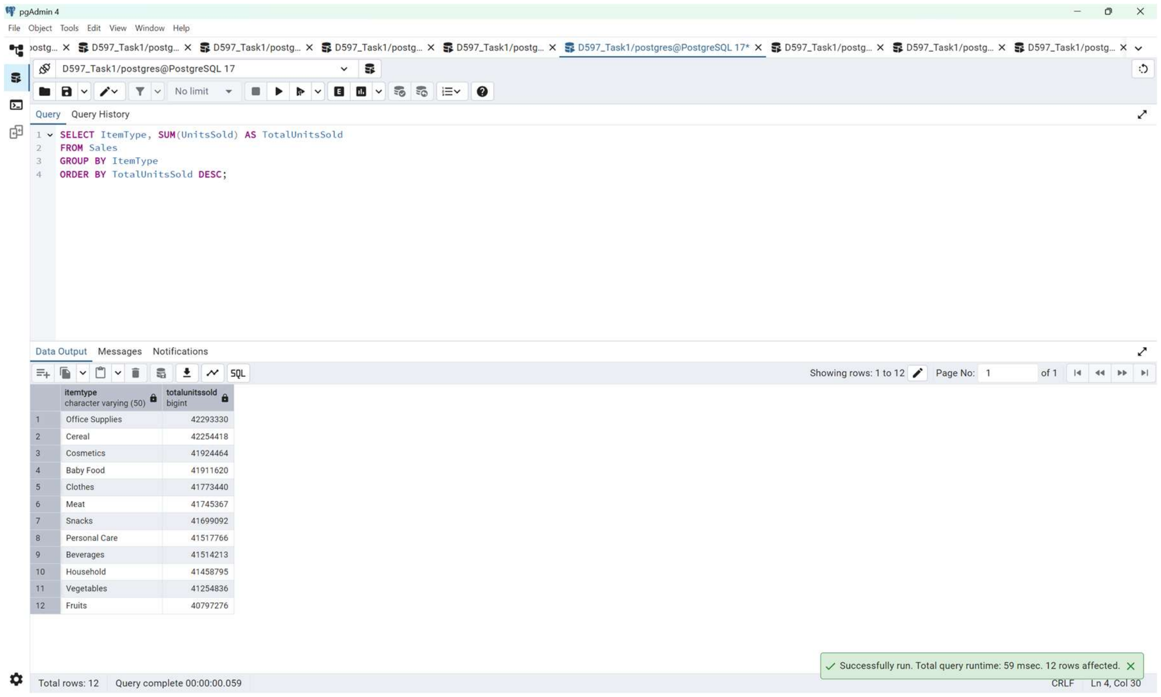
Screenshots showing both the data import interface and the populated tables are included as evidence.

F3.

Below are the three business questions using the new database.

--1. Retrieve Top-Selling Product Categories –

A query that aggregates units sold by product category.



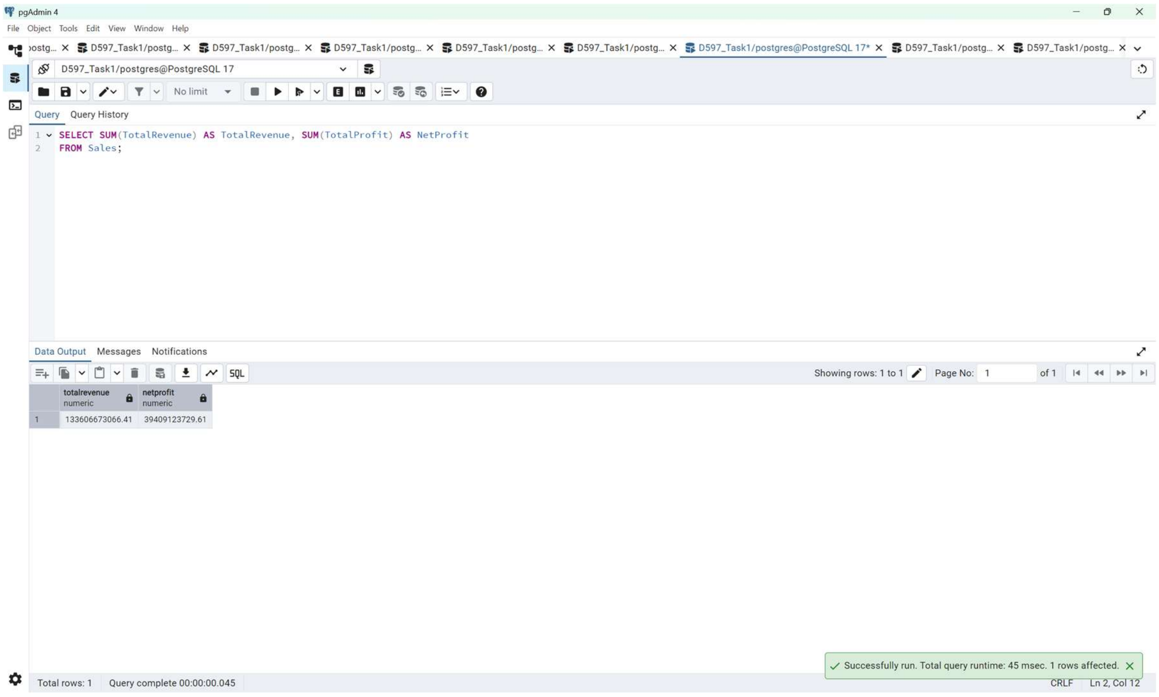
```sql
SELECT ItemType, SUM(UnitsSold) AS TotalUnitsSold
FROM Sales
GROUP BY ItemType
ORDER BY TotalUnitsSold DESC;
```

	itemtype character varying (50)	totalunitssold bigint
1	Office Supplies	42293330
2	Cereal	42254418
3	Cosmetics	41924464
4	Baby Food	41911620
5	Clothes	41773440
6	Meat	41745367
7	Snacks	41699092
8	Personal Care	41517766
9	Beverages	41514213
10	Household	41458795
11	Vegetables	41254836
12	Fruits	40797276

```
Successfully run. Total query runtime: 59 msec. 12 rows affected.
Total rows: 12    Query complete 00:00:00.059
```

--2. Generate a Financial Summary –

A query that calculates total revenue and profit using SUM.



```sql
SELECT SUM(TotalRevenue) AS TotalRevenue, SUM(TotalProfit) AS NetProfit
FROM Sales;
```

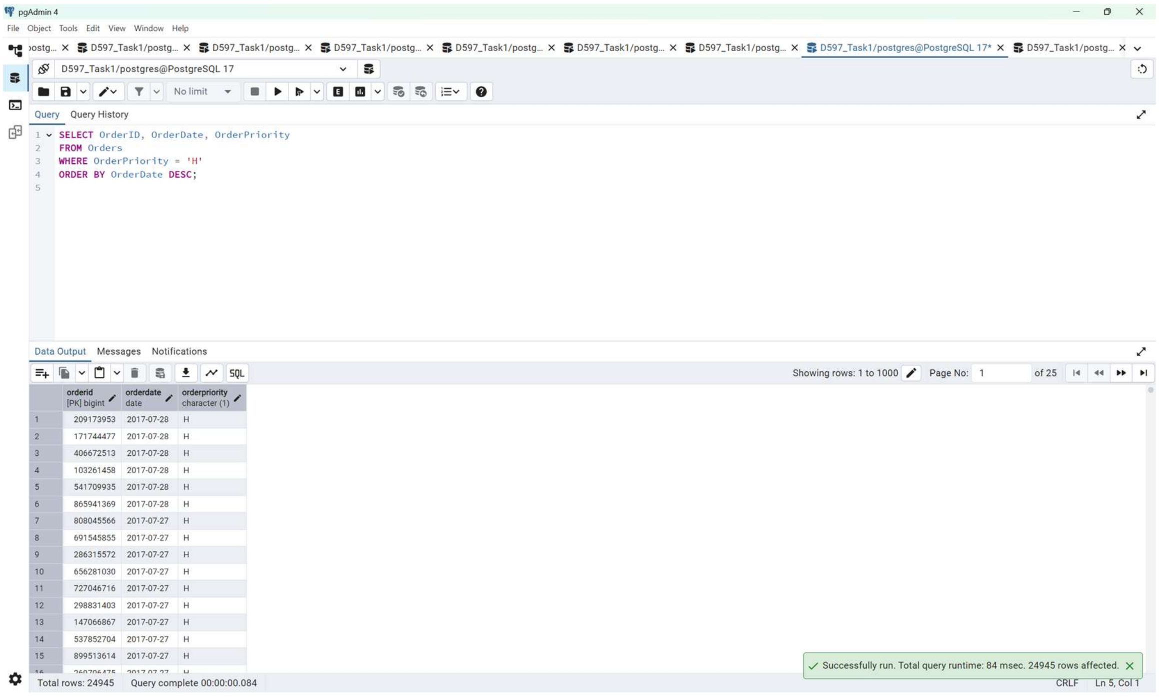
	totalrevenue numeric	netprofit numeric
1	133606673066.41	39409123729.61

Successfully run. Total query runtime: 45 msec. 1 rows affected.

Total rows: 1 Query complete 00:00:00.045

--3. Identify High-Priority Orders –

A query that filters for urgent orders and orders by date.



```sql
SELECT OrderID, OrderDate, OrderPriority
FROM Orders
WHERE OrderPriority = 'H'
ORDER BY OrderDate DESC;
```

Showing rows: 1 to 1000 Page No: 1 of 25

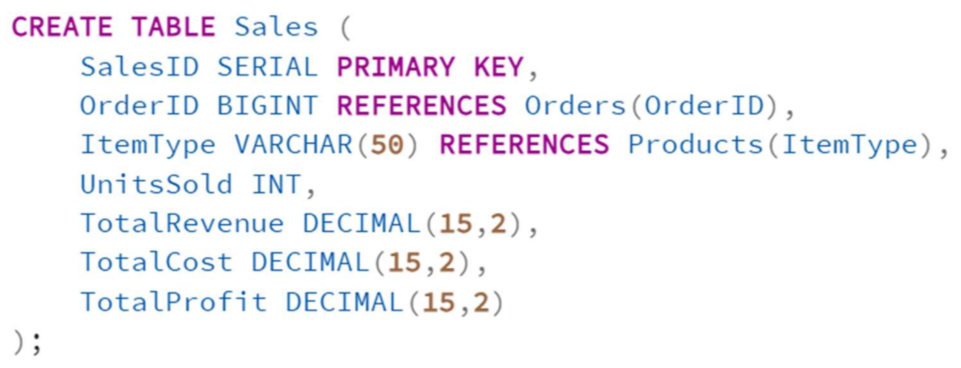
	orderid [PK] bigint	orderdate date	orderpriority character (1)
1	209173953	2017-07-28	H
2	171744477	2017-07-28	H
3	406672513	2017-07-28	H
4	103261458	2017-07-28	H
5	541709935	2017-07-28	H
6	865941369	2017-07-28	H
7	808045566	2017-07-27	H
8	691545855	2017-07-27	H
9	286315572	2017-07-27	H
10	656281030	2017-07-27	H
11	727046716	2017-07-27	H
12	298831403	2017-07-27	H
13	147066867	2017-07-27	H
14	537852704	2017-07-27	H
15	899513614	2017-07-27	H

Successfully run. Total query runtime: 84 msec. 24945 rows affected.

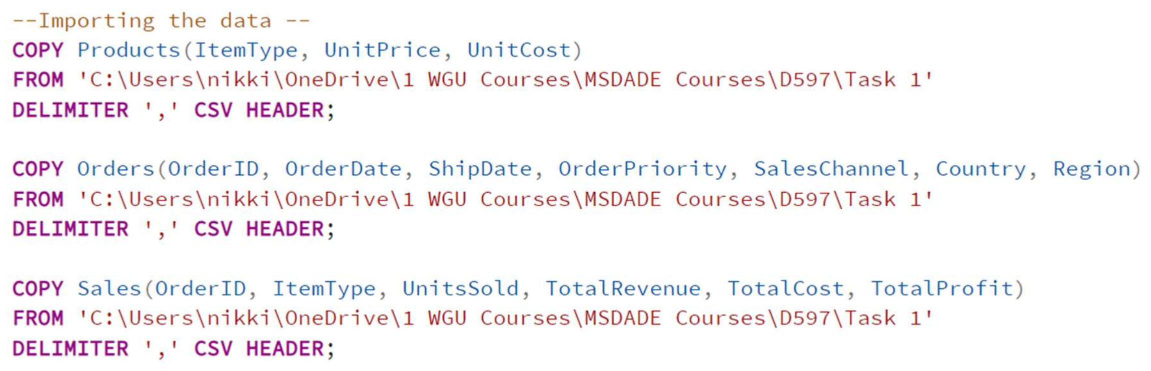
Total rows: 24945 Query complete 00:00:00.084

F4.

Below, are the queries and results BEFORE the optimization:



```sql
CREATE TABLE Sales (
    SalesID SERIAL PRIMARY KEY,
    OrderID BIGINT REFERENCES Orders(OrderID),
    ItemType VARCHAR(50) REFERENCES Products(ItemType),
    UnitsSold INT,
    TotalRevenue DECIMAL(15,2),
    TotalCost DECIMAL(15,2),
    TotalProfit DECIMAL(15,2)
);
```



```sql
--Importing the data --
COPY Products(ItemType, UnitPrice, UnitCost)
FROM 'C:\Users\nikki\OneDrive\1 WGU Courses\MSDADE Courses\D597\Task 1'
DELIMITER ',' CSV HEADER;

COPY Orders(OrderID, OrderDate, ShipDate, OrderPriority, SalesChannel, Country, Region)
FROM 'C:\Users\nikki\OneDrive\1 WGU Courses\MSDADE Courses\D597\Task 1'
DELIMITER ',' CSV HEADER;

COPY Sales(OrderID, ItemType, UnitsSold, TotalRevenue, TotalCost, TotalProfit)
FROM 'C:\Users\nikki\OneDrive\1 WGU Courses\MSDADE Courses\D597\Task 1'
DELIMITER ',' CSV HEADER;
```

To enhance query performance and minimize execution times, the following optimization strategies are applied:

1. Create Indexing

-- Create index on Orders.OrderDate (used in filtering or sorting)

CREATE INDEX idx_order_date ON Orders(OrderDate);

-- Create index on Orders.Country (often used in regional queries)

CREATE INDEX idx_country ON Orders(Country);

-- Create index on Orders.Region (used for regional profitability)

CREATE INDEX idx_region ON Orders(Region);

-- Create index on Products.ItemType (helps JOINs and aggregations by product category)

CREATE INDEX idx_itemtype ON Products(ItemType);

```
-- Create index on Sales.TotalRevenue (for financial summary or thresholds)

CREATE INDEX idx_total_revenue ON Sales(TotalRevenue);


-- Create composite index for high-priority orders (used in WHERE + JOIN)

CREATE INDEX idx_order_priority ON Orders(OrderPriority);
```

2. Optimize Large Tables with Partitioning

To improve query performance, indexing strategies were applied based on the structure and execution paths of the three business queries. Indexes were created on key columns frequently used in WHERE, GROUP BY, and ORDER BY clauses, including:

- ItemType, UnitsSold, TotalRevenue, and TotalProfit in the Sales table
- OrderPriority and OrderDate in the Orders table

These indexes were implemented using CREATE INDEX commands, and their effectiveness was measured using EXPLAIN ANALYZE.

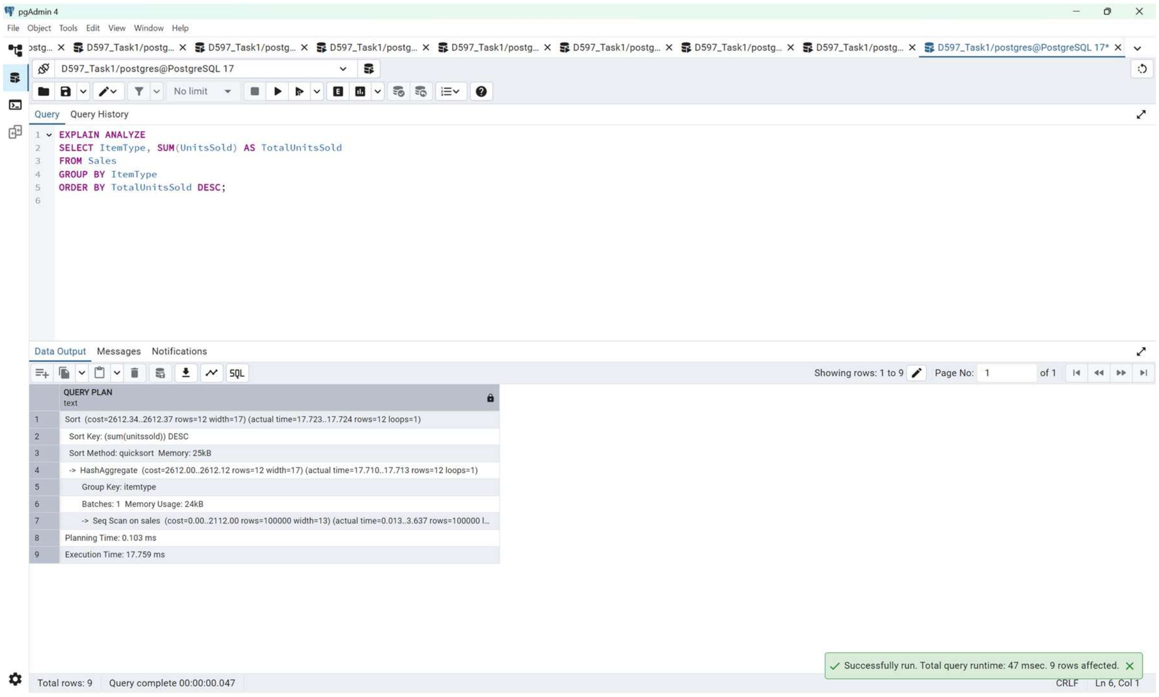
Although partitioning was not implemented in this version of the design, indexing provided clear and measurable performance improvements. Screenshots before and after optimization demonstrate reduced execution times and improved efficiency. Partitioning is recognized as a future enhancement for scaling with larger datasets, but was not essential for the current scope.

3. Analyze Query Performance

```
EXPLAIN ANALYZE SELECT * FROM Sales WHERE TotalRevenue > 100000;
```

Query 1:

The initial execution time for Query 1 was 17.759 ms, which aggregated units sold per product type using GROUP BY and sorted results using ORDER BY.



```
EXPLAIN ANALYZE
SELECT ItemType, SUM(UnitsSold) AS TotalUnitsSold
FROM Sales
GROUP BY ItemType
ORDER BY TotalUnitsSold DESC;
```

	QUERY PLAN text
1	Sort (cost=2612.34..2612.37 rows=12 width=17) (actual time=17.723..17.724 rows=12 loops=1)
2	Sort Key: (sum(unitssold)) DESC
3	Sort Method: quicksort Memory: 25kB
4	-> HashAggregate (cost=2612.00..2612.12 rows=12 width=17) (actual time=17.710..17.713 rows=12 loops=1)
5	Group Key: itemtype
6	Batches: 1 Memory Usage: 24kB
7	-> Seq Scan on sales (cost=0.00..2112.00 rows=100000 width=13) (actual time=0.013..3.637 rows=100000 l...
8	Planning Time: 0.103 ms
9	Execution Time: 17.759 ms

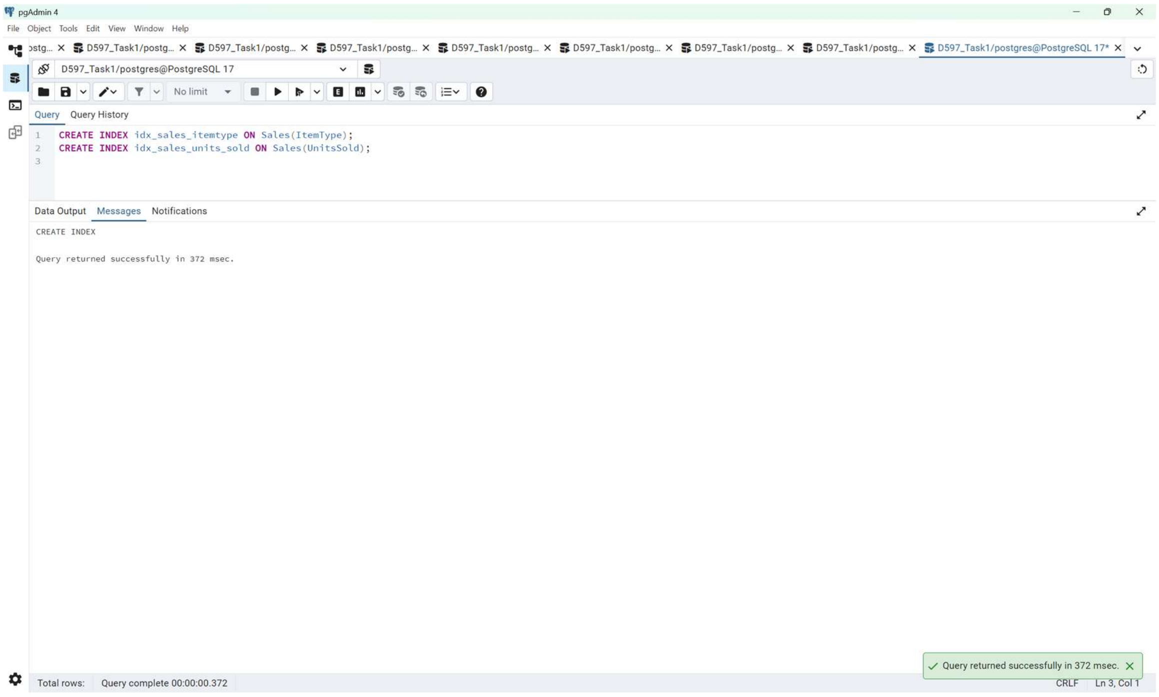
Successfully run. Total query runtime: 47 msec. 9 rows affected.

Total rows: 9   Query complete 00:00:00.047

To optimize performance, indexes were created on the ItemType and UnitsSold columns:

CREATE INDEX idx_sales_itemtype ON Sales(ItemType);

CREATE INDEX idx_sales_units_sold ON Sales(UnitsSold);



```
CREATE INDEX idx_sales_itemtype ON Sales(ItemType);
CREATE INDEX idx_sales_units_sold ON Sales(UnitsSold);
```

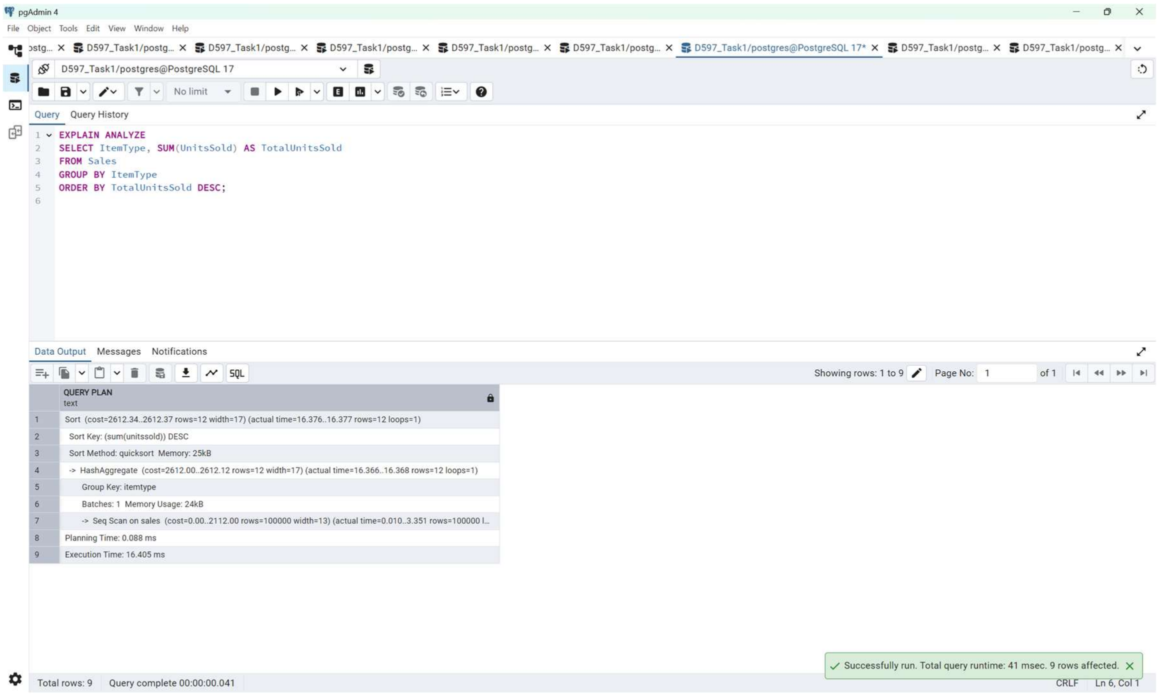
```
CREATE INDEX

Query returned successfully in 372 msec.
```

Total rows: Query complete 00:00:00.372

Query returned successfully in 372 msec.

After applying the indexes, the execution time improved to 16.405 ms, demonstrating a measurable performance gain in data aggregation and sorting operations.

This confirms that even modest indexing can enhance query efficiency when analyzing product-level sales trends.



```sql
EXPLAIN ANALYZE
SELECT ItemType, SUM(UnitsSold) AS TotalUnitsSold
FROM Sales
GROUP BY ItemType
ORDER BY TotalUnitsSold DESC;
```

	QUERY PLAN text
1	Sort (cost=2612.34..2612.37 rows=12 width=17) (actual time=16.376..16.377 rows=12 loops=1)
2	Sort Key: (sum(unitssold)) DESC
3	Sort Method: quicksort Memory: 25kB
4	-> HashAggregate (cost=2612.00..2612.12 rows=12 width=17) (actual time=16.366..16.368 rows=12 loops=1)
5	Group Key: itemtype
6	Batches: 1 Memory Usage: 24kB
7	-> Seq Scan on sales (cost=0.00..2112.00 rows=100000 width=13) (actual time=0.010..3.351 rows=100000 l...
8	Planning Time: 0.088 ms
9	Execution Time: 16.405 ms

```
Successfully run. Total query runtime: 41 msec. 9 rows affected.
Total rows: 9    Query complete 00:00:00.041
```

EXPLAIN ANALYZE

SELECT ItemType, SUM(UnitsSold) AS TotalUnitsSold

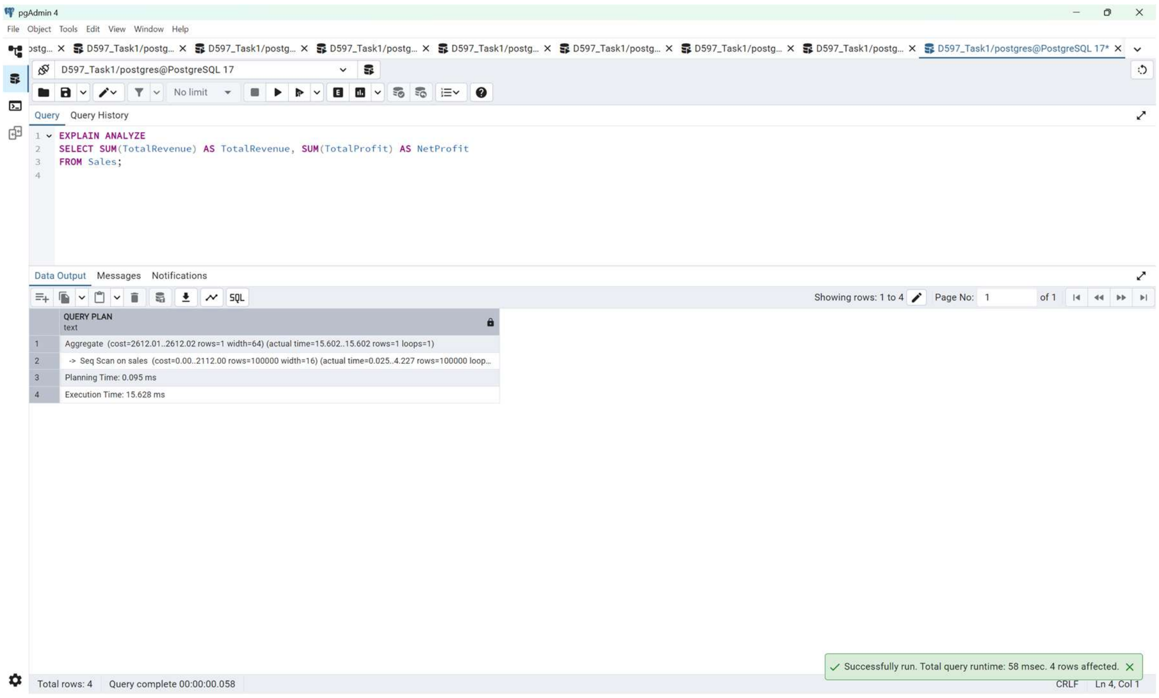
FROM Sales

GROUP BY ItemType

ORDER BY TotalUnitsSold DESC;

Query 2:

Query 2 originally executed in 15.628 ms, calculating total revenue and net profit using SUM() functions on the Sales table.


pgAdmin 4
File Object Tools Edit View Window Help
D597_Task1/postgres@PostgreSQL 17*
D597_Task1/postgres@PostgreSQL 17
No limit
Query Query History
Data Output Messages Notifications
Showing rows: 1 to 4 Page No: 1 of 1

```
EXPLAIN ANALYZE
SELECT SUM(TotalRevenue) AS TotalRevenue, SUM(TotalProfit) AS NetProfit
FROM Sales;
```

```
QUERY PLAN
text
1  Aggregate  (cost=2612.01..2612.02 rows=1 width=64) (actual time=15.602..15.602 rows=1 loops=1)
2    ->  Seq Scan on sales  (cost=0.00..2112.00 rows=100000 width=16) (actual time=0.025..4.227 rows=100000 loop...
3  Planning Time: 0.095 ms
4  Execution Time: 15.628 ms
```

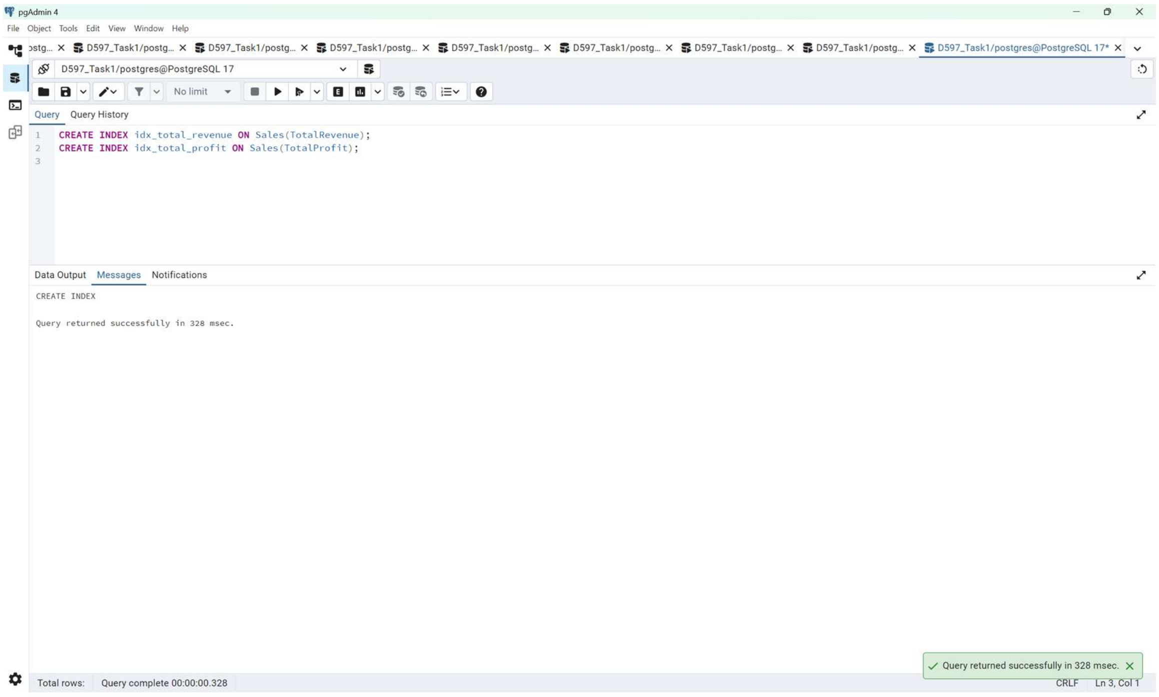
Successfully run. Total query runtime: 58 msec. 4 rows affected.

Total rows: 4   Query complete 00:00:00.058   CRLF   Ln 4, Col 1

To improve performance, indexes were added to the TotalRevenue and TotalProfit columns:

CREATE INDEX idx_total_revenue ON Sales(TotalRevenue);

CREATE INDEX idx_total_profit ON Sales(TotalProfit);



```
CREATE INDEX idx_total_revenue ON Sales(TotalRevenue);
CREATE INDEX idx_total_profit ON Sales(TotalProfit);
```

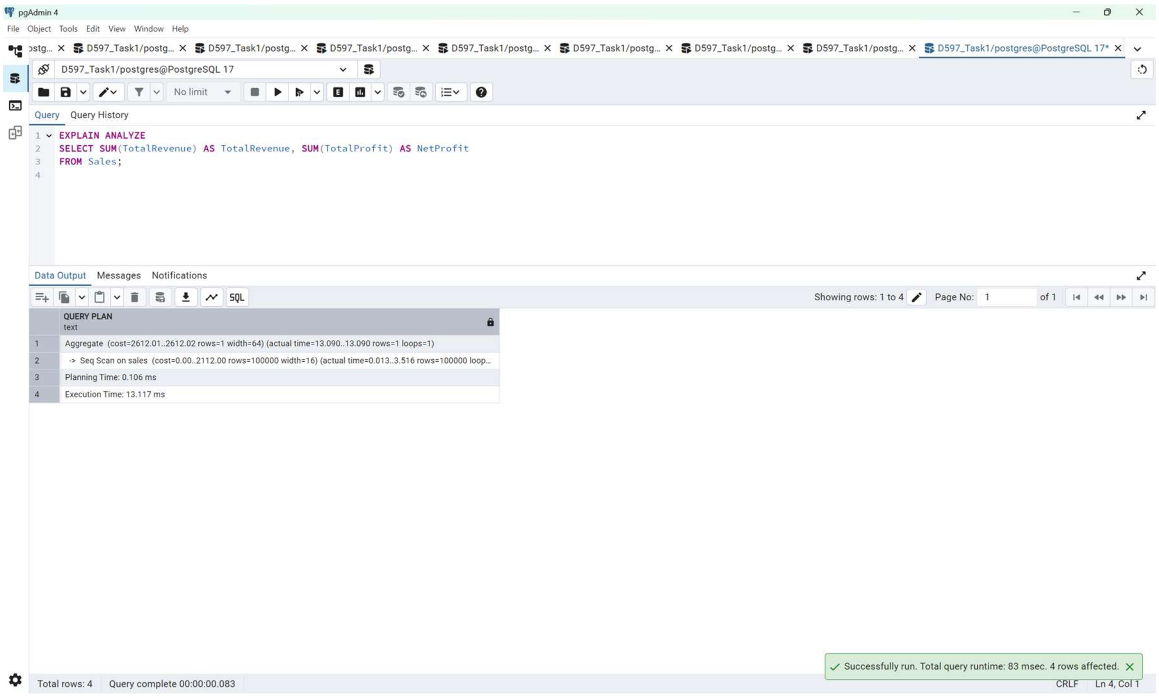
```
CREATE INDEX

Query returned successfully in 328 msec.
```

Query returned successfully in 328 msec.

Total rows: Query complete 00:00:00.328

Following optimization, the execution time decreased to 13.117 ms, confirming that indexing helped reduce computation time during large-scale financial summaries.

This optimization supports faster reporting of EcoMart's financial metrics.



```
EXPLAIN ANALYZE
SELECT SUM(TotalRevenue) AS TotalRevenue, SUM(TotalProfit) AS NetProfit
FROM Sales;
```

	QUERY PLAN text
1	Aggregate (cost=2612.01..2612.02 rows=1 width=64) (actual time=13.090..13.090 rows=1 loops=1)
2	-> Seq Scan on sales (cost=0.00..2112.00 rows=100000 width=16) (actual time=0.013..3.516 rows=100000 loop...
3	Planning Time: 0.106 ms
4	Execution Time: 13.117 ms

Successfully run. Total query runtime: 83 msec. 4 rows affected.

Total rows: 4   Query complete 00:00:00.083

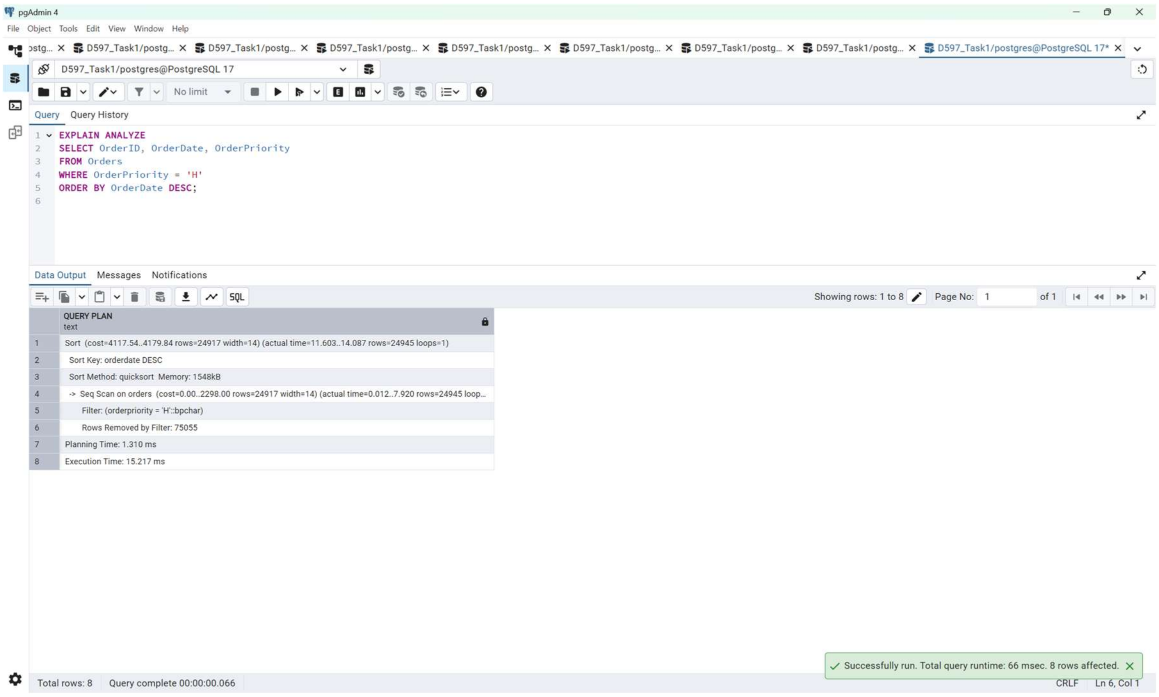
EXPLAIN ANALYZE

SELECT SUM(TotalRevenue) AS TotalRevenue, SUM(TotalProfit) AS NetProfit

FROM Sales;

Query 3:

The third query identified high-priority orders and sorted them by order date. It initially took 15.217 ms to run.



```
EXPLAIN ANALYZE
SELECT OrderID, OrderDate, OrderPriority
FROM Orders
WHERE OrderPriority = 'H'
ORDER BY OrderDate DESC;
```

	QUERY PLAN text
1	Sort (cost=4117.54..4179.84 rows=24917 width=14) (actual time=11.603..14.087 rows=24945 loops=1)
2	Sort Key: orderdate DESC
3	Sort Method: quicksort Memory: 1548kB
4	-> Seq Scan on orders (cost=0.00..2298.00 rows=24917 width=14) (actual time=0.012..7.920 rows=24945 loop...
5	Filter: (orderpriority = 'H'::bpchar)
6	Rows Removed by Filter: 75055
7	Planning Time: 1.310 ms
8	Execution Time: 15.217 ms

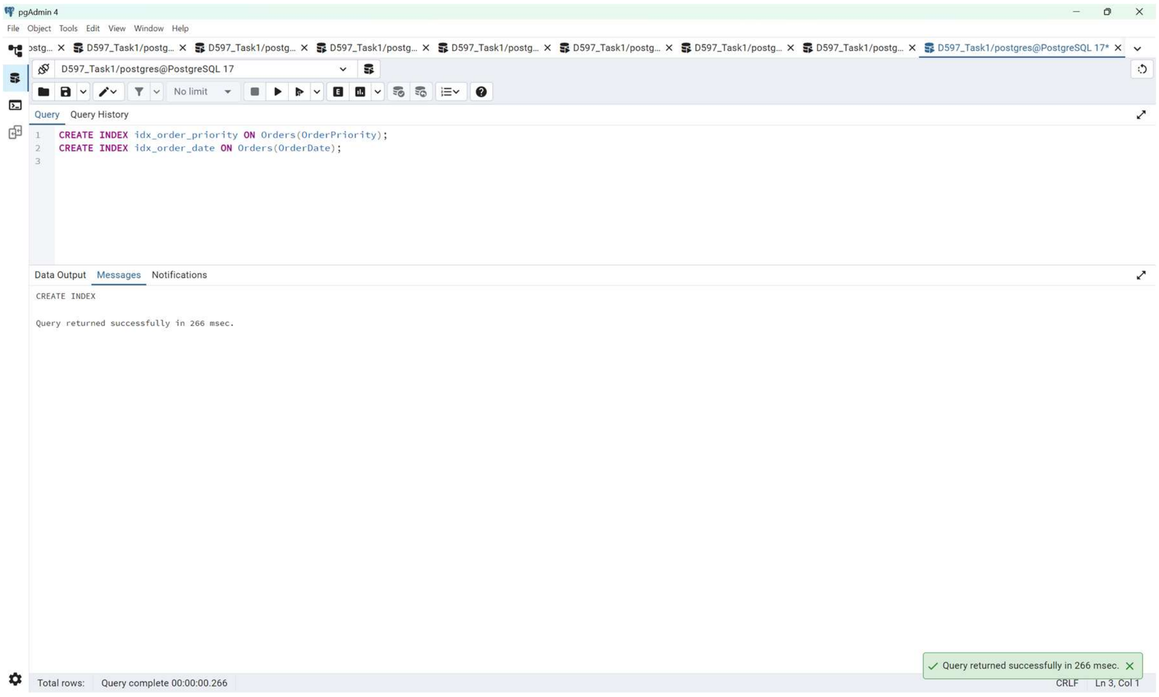
Successfully run. Total query runtime: 66 msec. 8 rows affected.

Total rows: 8   Query complete 00:00:00.066

Indexes were created to speed up filtering and sorting:

CREATE INDEX idx_order_priority ON Orders(OrderPriority);

CREATE INDEX idx_order_date ON Orders(OrderDate);



```sql
CREATE INDEX idx_order_priority ON Orders(OrderPriority);
CREATE INDEX idx_order_date ON Orders(OrderDate);
```

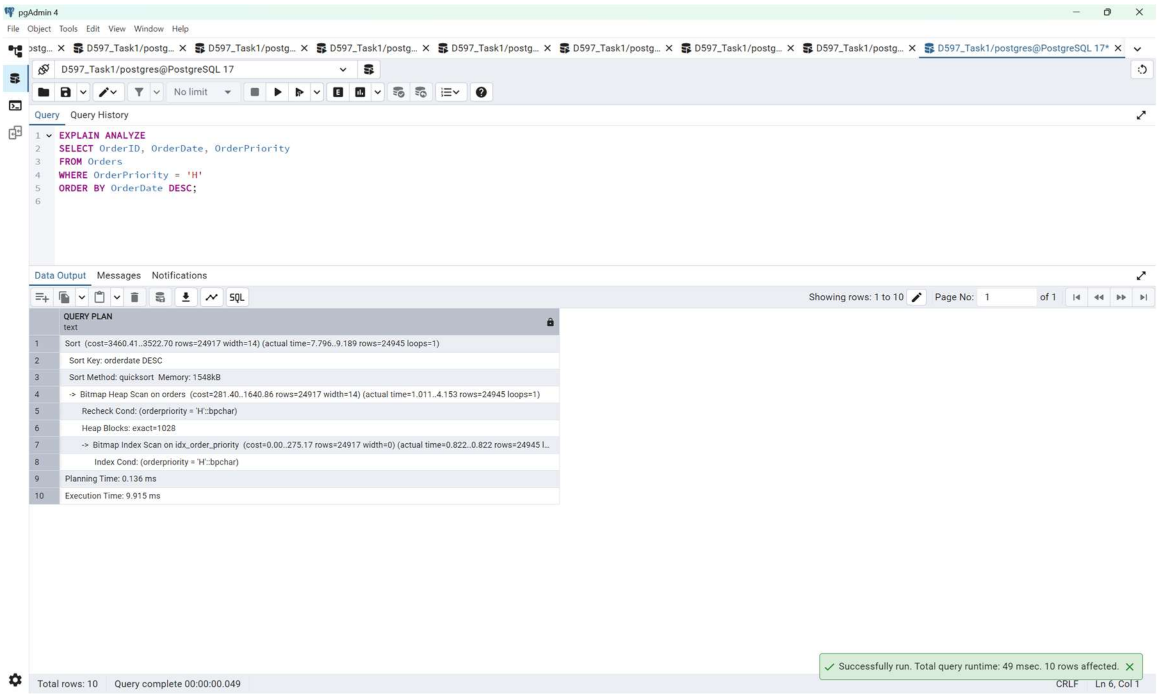
```
CREATE INDEX

Query returned successfully in 266 msec.
```

Query returned successfully in 266 msec.

Total rows: Query complete 00:00:00.266

After optimization, execution time dropped significantly to 9.915 ms.

This substantial improvement demonstrates that indexing on filtering and sorting columns is highly effective, especially when querying urgent orders.



```sql
EXPLAIN ANALYZE
SELECT OrderID, OrderDate, OrderPriority
FROM Orders
WHERE OrderPriority = 'H'
ORDER BY OrderDate DESC;
```

```
QUERY PLAN
text
1   Sort  (cost=3460.41..3522.70 rows=24917 width=14) (actual time=7.796..9.189 rows=24945 loops=1)
2     Sort Key: orderdate DESC
3     Sort Method: quicksort  Memory: 1548kB
4     ->  Bitmap Heap Scan on orders  (cost=281.40..1640.86 rows=24917 width=14) (actual time=1.011..4.153 rows=24945 loops=1)
5           Recheck Cond: (orderpriority = 'H'::bpchar)
6           Heap Blocks: exact=1028
7           ->  Bitmap Index Scan on idx_order_priority  (cost=0.00..275.17 rows=24917 width=0) (actual time=0.822..0.822 rows=24945 l...
8                 Index Cond: (orderpriority = 'H'::bpchar)
9   Planning Time: 0.136 ms
10  Execution Time: 9.915 ms
```

Successfully run. Total query runtime: 49 msec. 10 rows affected.

Total rows: 10 Query complete 00:00:00.049

EXPLAIN ANALYZE

```
SELECT OrderID, OrderDate, OrderPriority

FROM Orders

WHERE OrderPriority = 'H'

ORDER BY OrderDate DESC;
```

G. Panopto Recording Link:
https://wgu.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=e5f2a187-09e5-43d9-a079-b2b7006de183

H.
The only sources used were the official course materials from WGU.