Task 1:

Relational Database Design and Implementation

Keniyah Chestnut

Data Management — D597

SID: 012601305

A1: BUSINESS PROBLEM

I have chosen Scenario 2 for my project. In this scenario, the company needs a more efficient and scalable way to manage its global sales data. Currently, all their information is stored in a single flat CSV file, which includes 100,000 rows of raw sales data. This structure makes it hard to run queries, slows down performance, and increases the risk of errors or data duplication.

The company also wants to ensure their system is easy to maintain and secure enough to protect sensitive data. A well-structured relational database can solve these problems by organizing the data into separate, related tables and using keys to connect them. This design will help reduce redundancy, speed up performance, and make the system easier to update and scale in the future.

A2 & A3: DATA STRUCTURE AND DATABASE JUSTIFICATION

A structured relational database is the best solution for this scenario because the dataset is already organized in consistent rows and columns but is currently stored as a single, flat CSV file. Although it follows First Normal Form (1NF) with atomic values, the current design has major limitations. With 100,000 records and 14+ columns of repeated data like region, country, and item type, querying this file is inefficient and slow.

To improve performance and flexibility, I structured the data into Third Normal Form (3NF). This process included organizing the dataset into multiple related tables—Regions, Countries,

Item_Types, and Sales_Records. Each table contains only the information relevant to its specific category, which reduces redundancy and improves clarity.

This structure ensures that every non-key column is fully dependent on the primary key, and it eliminates transitive dependencies by breaking up related attributes into separate tables. For example, instead of listing “Sub-Saharan Africa” thousands of times in the raw data, it now appears once in the Regions table and is referenced using a region_id.

Using a relational database for this solution makes the system much more scalable. It allows for faster queries, easier updates, and better data integrity. As new records or products are added, they can be managed within the existing structure without affecting performance. The data will also be easier to maintain and analyze, supporting business needs like reporting, forecasting, and trend analysis.

A4: DATA USAGE WITHIN THE DATABASE SOLUTION

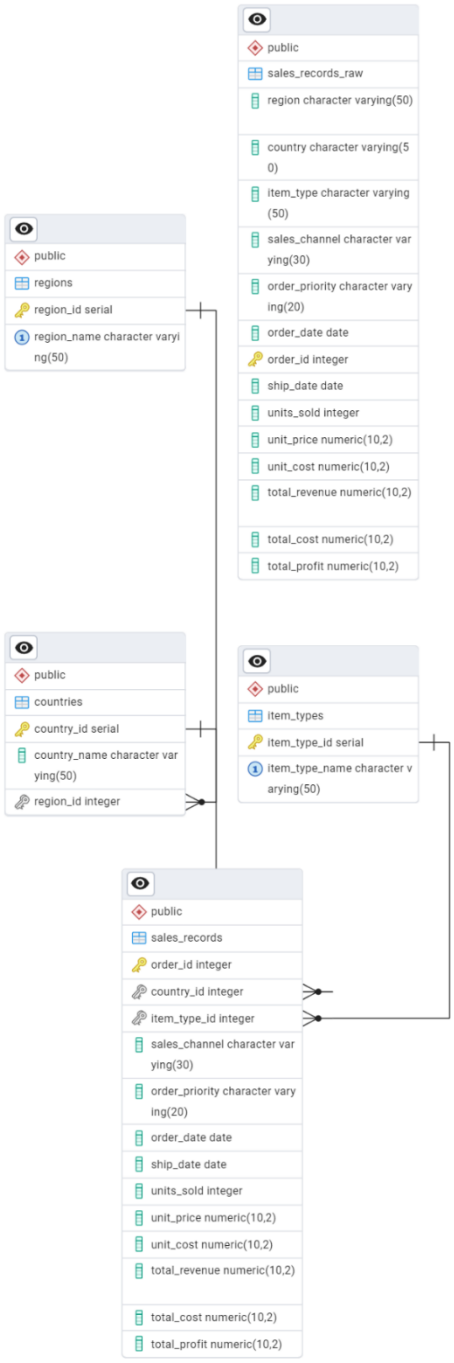
The data in this database will be used to support reporting, performance tracking, and business decision-making. By organizing the information into separate but connected tables, the company can now run efficient queries to explore trends in sales, profit, product popularity, and regional performance.

For example, managers can run queries to see which item types are selling the most, which regions are bringing in the most profit, or whether online or offline sales channels are generating

more revenue. This allows different departments such as marketing, operations, and finance to make informed decisions using accurate and up-to-date information.

The new database structure also makes it easier to connect with data visualization tools such as Tableau or Power BI. These tools can be used to create dashboards and reports that display insights clearly and quickly. Overall, the business will benefit from better access to reliable data, faster reporting, and a system that can grow with the company.

B: LOGICAL DATA MODEL



C: OBJECTS AND STORAGE

The tables in my database are Sales_Records, Countries, Item_Types, and Regions. The primary key for the Sales_Records table is Order_ID. This table also includes three foreign keys: Region_ID, Country_ID, and Item_Type_ID. Each of these foreign keys connects to its corresponding table, where the referenced value is the primary key.

To summarize, here is the structure of each table:

The Regions table contains:

- Region_ID INT (Primary Key)
- Region_Name VARCHAR(50)

The Countries table contains:

- Country_ID INT (Primary Key)
- Country_Name VARCHAR(50)
- Region_ID INT (Foreign Key that references Regions.Region_ID)

The Item_Types table contains:

- Item_Type_ID INT (Primary Key)
- Item_Type_Name VARCHAR(50)

The Sales_Records table contains:

- Order_ID INT (Primary Key)
- Country_ID INT (Foreign Key that references Countries.Country_ID)
- Item_Type_ID INT (Foreign Key that references Item_Types.Item_Type_ID)
- Sales_Channel VARCHAR(30)
- Order_Priority VARCHAR(20)
- Order_Date DATE
- Ship_Date DATE
- Units_Sold INT
- Unit_Price DECIMAL(10, 2)
- Unit_Cost DECIMAL(10, 2)

- Total_Revenue DECIMAL(10, 2)
- Total_Cost DECIMAL(10, 2)
- Total_Profit DECIMAL(10, 2)

This structure keeps the data clean, organized, and efficient for querying. It also makes it easier to maintain the database as new regions, countries, or item types are added in the future.

D: SCALABILITY

As discussed above, now that the data is structured in Third Normal Form (3NF), it is much easier to scale and manage in a sustainable way that aligns with EcoMart's growing data needs. By separating repeating values such as countries, regions, and item types into their own tables, we reduce redundancy and ensure data consistency.

This modular structure supports scalability. For example, if new countries, regions, or item categories are added as EcoMart expands globally, they can simply be added to their respective dimension tables without altering existing sales records. This avoids the need to update thousands of rows and helps maintain cleaner, more manageable data.

Additionally, I implemented indexing strategies on key fields (such as country_id, item_type_id, and sales_channel) to speed up query performance. These indexes allow EcoMart to efficiently retrieve insights even as the dataset grows to include millions of sales records.

By reducing data duplication, organizing the schema with foreign keys, and applying indexing, this solution is flexible, performant, and built to support EcoMart's long-term business growth and reporting needs.

E: PRIVACY AND SECURITY

Because customer and sales data is sensitive, it must be properly protected. This means the data should be encrypted both at rest and during transmission to prevent unauthorized access. The company should implement role-based access control (RBAC) to make sure only authorized users can view or modify certain parts of the database.

Audit logging should also be used to keep track of who accesses the system and when, which can help identify unusual or unauthorized behavior. Enforcing foreign key constraints will help maintain data integrity by preventing orphan records from being created.

To strengthen security even further, the company could also consider adding two-factor authentication for users who have access to the database and performing regular security reviews to stay ahead of potential vulnerabilities.

F1: DATABASE INSTANCE CREATION

Here is my script to create the database based on the logical data model I described earlier. I created the main table, Sales_Records, which includes all of the fields needed for importing the full dataset from the original CSV file.

```sql
CREATE TABLE sales_records_raw (
 region VARCHAR(50),
 country VARCHAR(50),
 item_type VARCHAR(50),
 sales_channel VARCHAR(30),
 order_priority VARCHAR(20),
 order_date DATE,
 order_id INT PRIMARY KEY,
 ship_date DATE,
 units_sold INT,
 unit_price DECIMAL(10,2),
 unit_cost DECIMAL(10,2),
 total_revenue DECIMAL(10,2),
 total_cost DECIMAL(10,2),
 total_profit DECIMAL(10,2)
);
```

I also created separate tables for the redundant information: Regions, Countries, and Item_Types. Since these values appear repeatedly in the sales data, moving them into separate tables helps save space and makes queries more efficient. This process brings the structure into Third Normal Form (3NF) by reducing duplication and increasing flexibility.

```sql
CREATE TABLE regions (
    region_id SERIAL PRIMARY KEY,
    region_name VARCHAR(50) UNIQUE
);

CREATE TABLE countries (
    country_id SERIAL PRIMARY KEY,
    country_name VARCHAR(50),
    region_id INT REFERENCES regions(region_id)
);

CREATE TABLE item_types (
    item_type_id SERIAL PRIMARY KEY,
    item_type_name VARCHAR(50) UNIQUE
);
```

Once these tables were created, I began inserting distinct values from the original Sales_Records_raw table into the new dimension tables and added foreign key relationships to connect them. This helped prepare the data structure for analysis while ensuring better performance and maintainability.

After populating the dimension tables, I created the final normalized sales_records table. This table links to the countries and item_types tables using foreign keys. It stores transactional data like sales quantity, revenue, cost, and profit. I then inserted the transformed data into this table by joining the staging table with the dimension tables to retrieve the correct foreign key IDs. This completed the normalization process and prepared the database for analysis.

```sql
-- Step 5: Create final normalized table
CREATE TABLE sales_records (
    order_id INT PRIMARY KEY,
    country_id INT REFERENCES countries(country_id),
    item_type_id INT REFERENCES item_types(item_type_id),
    sales_channel VARCHAR(30),
    order_priority VARCHAR(20),
    order_date DATE,
    ship_date DATE,
    units_sold INT,
    unit_price DECIMAL(10,2),
    unit_cost DECIMAL(10,2),
    total_revenue DECIMAL(10,2),
    total_cost DECIMAL(10,2),
    total_profit DECIMAL(10,2)
);

-- Step 6: Populate normalized table
INSERT INTO sales_records (
    order_id, country_id, item_type_id, sales_channel, order_priority,
    order_date, ship_date, units_sold, unit_price, unit_cost,
    total_revenue, total_cost, total_profit
)
```

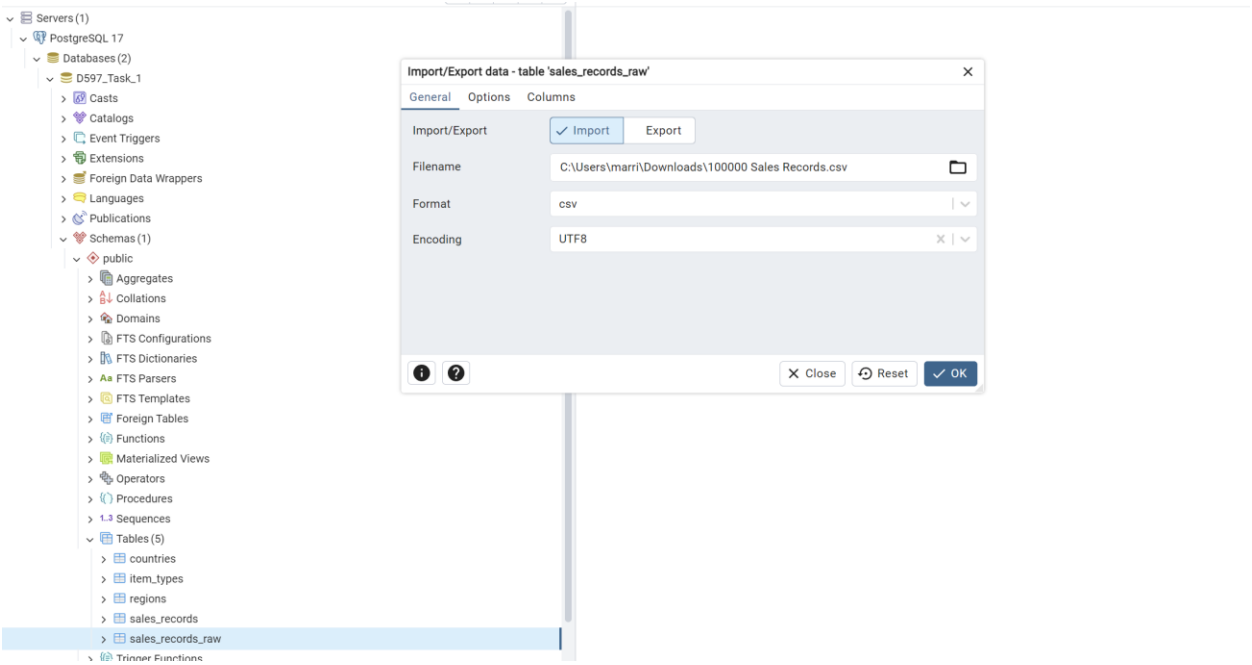
F2: DATA IMPORT

To begin populating the database, I first imported the raw sales data from the CSV file into a staging table called sales_records_raw. This table temporarily holds the original flat structure

of the data, allowing me to later transform and normalize it into a more scalable format.

```sql
-- STEP 1: Create raw import table to hold original CSV data
CREATE TABLE sales_records_raw (
    region VARCHAR(50),
    country VARCHAR(50),
    item_type VARCHAR(50),
    sales_channel VARCHAR(30),
    order_priority VARCHAR(20),
    order_date DATE,
    order_id INT PRIMARY KEY,
    ship_date DATE,
    units_sold INT,
    unit_price DECIMAL(10,2),
    unit_cost DECIMAL(10,2),
    total_revenue DECIMAL(10,2),
    total_cost DECIMAL(10,2),
    total_profit DECIMAL(10,2)
);
```

After creating the table, I imported the data using the built-in import tool in pgAdmin. Once imported, I verified that all records appeared correctly in the sales_records_raw table.



```
Select *
FROM sales_records_raw;
```

Output Messages Notifications

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

region character varying (50)	country character varying (50)	item_type character varying (50)	sales_channel character varying (30)	order_priority character varying (20)	order_date date	order_id [PK] integer	ship_date date	units_sold integer	unit_price numeric (10,2)	unit_cost numeric (10,2)	total_revenue numeric (10,2)	total_co numeric
Middle East and North Africa	Azerbaijan	Snacks	Online	C	2014-10-08	535113847	2014-10-23	934	152.58	97.44	142509.72	
Central America and the Caribbean	Panama	Cosmetics	Offline	L	2015-02-22	874708545	2015-02-27	4551	437.20	263.33	1989697.20	11
Sub-Saharan Africa	Sao Tome and Principe	Fruits	Offline	M	2015-12-09	854349935	2016-01-18	9986	9.33	6.92	93169.38	
Sub-Saharan Africa	Sao Tome and Principe	Personal Care	Online	M	2014-09-17	892836844	2014-10-12	9118	81.73	56.67	745214.14	5
Central America and the Caribbean	Belize	Household	Offline	H	2010-02-04	1292806602	2010-03-05	5858	668.27	502.54	3914725.66	25
Europe	Denmark	Clothes	Online	C	2013-02-20	473105037	2013-02-28	1149	109.28	35.84	125562.72	
Europe	Germany	Cosmetics	Offline	M	2013-03-31	754046475	2013-05-03	7964	437.20	263.33	3481860.80	20
Middle East and North Africa	Turkey	Fruits	Online	C	2012-03-26	772153747	2012-04-07	6307	9.33	6.92	58844.31	
Europe	United Kingdom	Snacks	Online	H	2012-12-29	847788178	2013-01-15	8217	152.58	97.44	1253749.86	8
Asia	Kazakhstan	Cosmetics	Online	H	2015-09-11	471623599	2015-09-18	2758	437.20	263.33	1205797.60	7
Central America and the Caribbean	Haiti	Cosmetics	Online	C	2013-12-27	554646337	2014-01-01	1031	437.20	263.33	450753.20	2
Europe	Italy	Clothes	Online	M	2013-12-17	278155219	2014-01-10	1165	109.28	35.84	127311.20	
Europe	Malta	Household	Offline	L	2015-03-10	243761575	2015-04-17	3322	668.27	502.54	2219992.94	16
Middle East and North Africa	Jordan	Household	Offline	L	2014-07-09	223389232	2014-07-18	4693	668.27	502.54	3136191.11	23
Asia	Cambodia	Vegetables	Offline	H	2017-09-14	509274518	2017-09-29	4502	154.06	90.93	693578.12	4
Central America and the Caribbean	Saint Kitts and Nevis	Office Supplies	Online	H	2011-08-10	563966762	2011-08-29	9004	651.21	524.96	5863494.84	47
Sub-Saharan Africa	Cameroon	Fruits	Online	H	2015-12-18	370325791	2016-02-06	6486	9.33	6.92	60514.38	
Middle East and North Africa	Bahrain	Vegetables	Offline	L	2016-06-26	881526935	2016-07-19	2284	154.06	90.93	348791.84	2
Australia and Oceania	Solomon Islands	Beverages	Offline	C	2015-05-11	871056020	2015-05-30	3688	47.45	31.79	174995.60	1

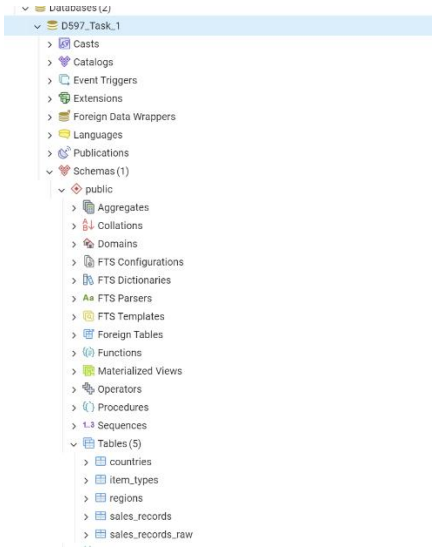
This confirmed that the data had been imported correctly.

Next, I created three lookup tables to support database normalization: regions, countries, and item_types. These tables contain unique values found in the raw dataset, which reduces redundancy and improves performance.

```sql
-- Step 3: Create lookup tables
CREATE TABLE regions (
    region_id SERIAL PRIMARY KEY,
    region_name VARCHAR(50) UNIQUE
);

CREATE TABLE countries (
    country_id SERIAL PRIMARY KEY,
    country_name VARCHAR(50),
    region_id INT REFERENCES regions(region_id)
);

CREATE TABLE item_types (
    item_type_id SERIAL PRIMARY KEY,
    item_type_name VARCHAR(50) UNIQUE
);
```



I populated each of the lookup tables using INSERT INTO ... SELECT DISTINCT statements from the sales_records_raw table.

```
INSERT INTO regions (region_name)
SELECT DISTINCT region FROM sales_records_raw;

INSERT INTO countries (country_name, region_id)
SELECT DISTINCT sr.country, r.region_id
FROM sales_records_raw sr
JOIN regions r ON sr.region = r.region_name;

INSERT INTO item_types (item_type_name)
SELECT DISTINCT item_type FROM sales_records_raw;
```

Next, I created the final normalized table called sales_records. This table is structured to use foreign key references from the dimension tables countries and item_types, aligning with the principles of Third Normal Form (3NF). Instead of repeating region, country, and item type text in every row, those values are now linked by ID, which reduces redundancy and improves query performance.

To complete the normalization process, I inserted the transformed data into the sales_records table using a query that joins the raw data with the dimension tables. This ensures that each sales record references its related country and item type using foreign keys:

```sql
-- Step 6: Populate normalized table
INSERT INTO sales_records (
    order_id, country_id, item_type_id, sales_channel, order_priority,
    order_date, ship_date, units_sold, unit_price, unit_cost,
    total_revenue, total_cost, total_profit
)
SELECT
    sr.order_id,
    c.country_id,
    it.item_type_id,
    sr.sales_channel,
    sr.order_priority,
    sr.order_date,
    sr.ship_date,
    sr.units_sold,
    sr.unit_price,
    sr.unit_cost,
    sr.total_revenue,
    sr.total_cost,
    sr.total_profit
FROM sales_records_raw sr
JOIN countries c ON sr.country = c.country_name
JOIN item_types it ON sr.item_type = it.item_type_name;
```

F3: BUSINESS QUERIES

I wrote queries for three business questions using the new normalized database. Since EcoMart's goal was to improve efficiency and scalability, I chose questions that focus on business performance insights.

Question 1: What are the top-selling products?

This query helps EcoMart determine which item categories sell best, supporting decisions related to stocking, product bundling, and marketing strategies.

```sql
SELECT it.item_type_name, SUM(sr.units_sold) AS total_units_sold
FROM sales_records sr
JOIN item_types it ON sr.item_type_id = it.item_type_id
GROUP BY it.item_type_name
ORDER BY total_units_sold DESC
LIMIT 5;
```

Results:

	item_type_name character varying (50)	total_units_sold bigint
1	Office Supplies	42293330
2	Cereal	42254418
3	Cosmetics	41924464
4	Baby Food	41911620
5	Clothes	41773440

Question 2: Which regions are most and least profitable?

EcoMart needs to identify where their most profitable markets are located. By summarizing total profits by region, they can better allocate marketing budgets or reconsider logistics in underperforming areas.

```
110  -- 1. Which regions are most and least profitable?
111  SELECT r.region_name, SUM(sr.total_profit) AS total_profit
112  FROM sales_records sr
113  JOIN countries c ON sr.country_id = c.country_id
114  JOIN regions r ON c.region_id = r.region_id
115  GROUP BY r.region_name
116  ORDER BY total_profit DESC;
```

Results:

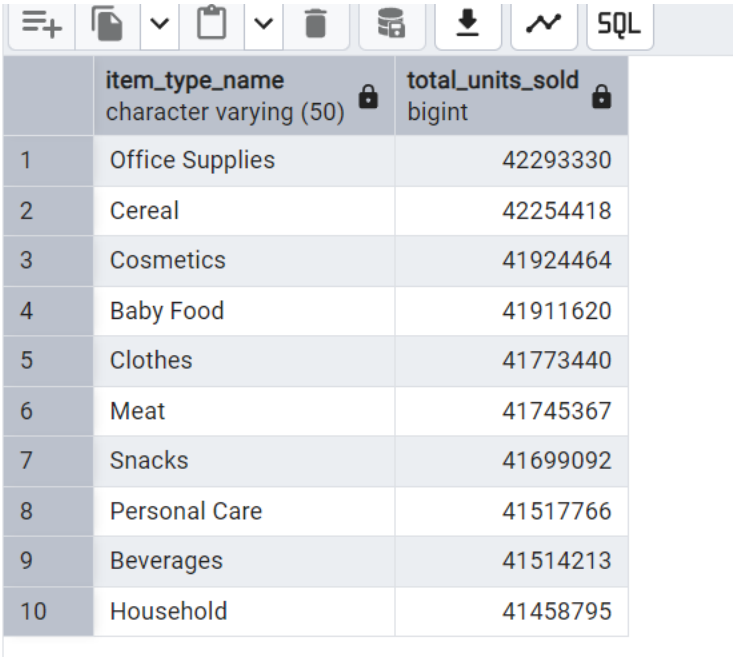
	region_name character varying (50)	total_profit numeric
1	Sub-Saharan Africa	10306312642.23
2	Europe	10080579491.05
3	Asia	5707511516.76
4	Middle East and North Africa	4979534378.88
5	Central America and the Caribbean	4287210522.47
6	Australia and Oceania	3175423561.38
7	North America	872551616.84

Question 2: What are the top-selling products?

This query helps EcoMart determine which item categories sell best, supporting decisions related to stocking, product bundling, and marketing strategies.

```sql
-- 2. What are the top-selling product types?
SELECT it.item_type_name, SUM(sr.units_sold) AS total_units_sold
FROM sales_records sr
JOIN item_types it ON sr.item_type_id = it.item_type_id
GROUP BY it.item_type_name
ORDER BY total_units_sold DESC
LIMIT 10;
```

Results:



	item_type_name character varying (50)	total_units_sold 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

Question 3: Which sales channel produces the most revenue?

EcoMart sells both online and offline. This query reveals which channel generates more revenue, helping the company decide whether to invest more in digital transformation or enhance brick-and-mortar experiences.

```sql
-- 3. Which sales channel produces the most revenue?
SELECT sales_channel, SUM(total_revenue) AS total_revenue
FROM sales_records
GROUP BY sales_channel
ORDER BY total_revenue DESC;
```

Results:

	sales_channel character varying (30)	total_revenue numeric
1	Online	66856341348.55
2	Offline	66750331717.86

F4: OPTIMIZATION TECHNIQUES

I applied several optimization techniques to improve the performance of the queries in F3. The primary goal was to convert the dataset into a structured format that supports fast and efficient analysis. Below is a summary of the optimization process:

- Converted the data into 3rd Normal Form (3NF): This reduced redundancy by separating regions, countries, and item types into their own tables with foreign key references.

- Indexed Key Columns: I created indexes on country_id, item_type_id, sales_channel, and total_profit to improve the speed of filters and joins.

- Primary Keys and Foreign Keys: Each table uses a PRIMARY KEY, and the sales_records table uses FOREIGN KEYS for all relationships. This enforces referential itegrity and improves JOIN performance.

- Analyzed the database using ANALYZE to help PostgreSQL optimize query execution plans.

Here is what the original unoptimized table looked like:

```sql
CREATE TABLE sales_records_raw (
  region VARCHAR(50),
  country VARCHAR(50),
  item_type VARCHAR(50),
  sales_channel VARCHAR(30),
  order_priority VARCHAR(20),
  order_date DATE,
  order_id INT PRIMARY KEY,
  ship_date DATE,
  units_sold INT,
  unit_price DECIMAL(10,2),
  unit_cost DECIMAL(10,2),
  total_revenue DECIMAL(10,2),
  total_cost DECIMAL(10,2),
  total_profit DECIMAL(10,2)
);
```

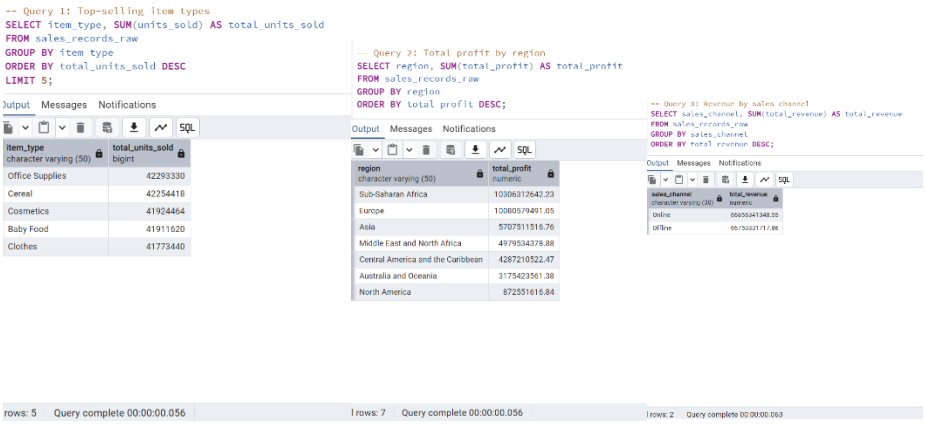
And here is a portion of the optimized structure:

```sql
CREATE TABLE sales_records (
    order_id INT PRIMARY KEY,
    country_id INT REFERENCES countries(country_id),
    item_type_id INT REFERENCES item_types(item_type_id),
    sales_channel VARCHAR(30),
    order_priority VARCHAR(20),
    order_date DATE,
    ship_date DATE,
    units_sold INT,
    unit_price DECIMAL(10,2),
    unit_cost DECIMAL(10,2),
    total_revenue DECIMAL(10,2),
    total_cost DECIMAL(10,2),
    total_profit DECIMAL(10,2)
);
```

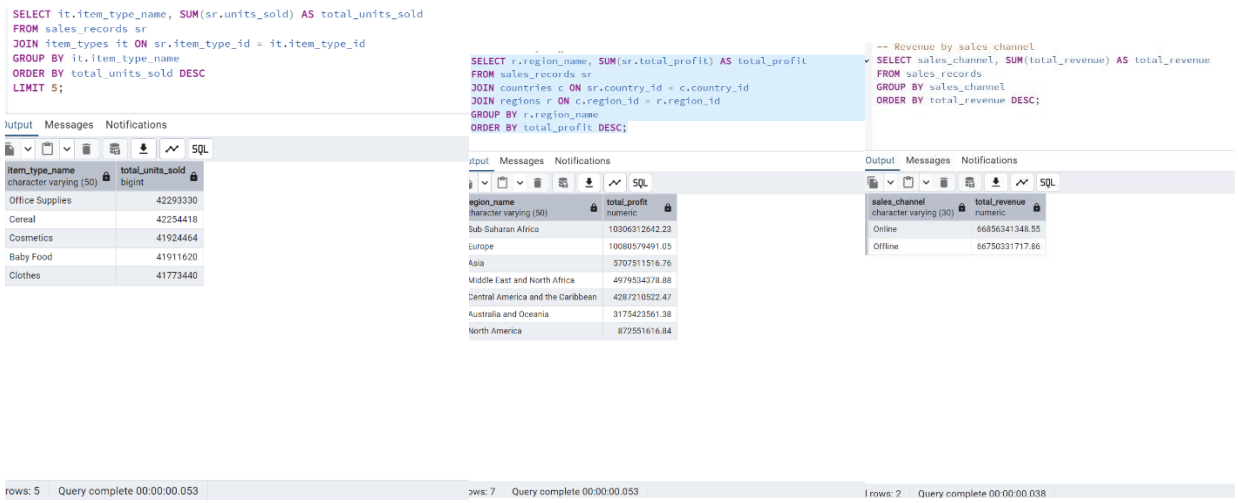
I also created indexes on key columns to further boost performance:

```
CREATE INDEX idx_country_id ON sales_records(country_id);
CREATE INDEX idx_item_type_id ON sales_records(item_type_id);
CREATE INDEX idx_sales_channel ON sales_records(sales_channel);
CREATE INDEX idx_total_profit ON sales_records(total_profit);
ANALYZE;
```

Queries before optimization:



After Optimization:



Performance Comparison:

Query Description	Pre-Optimization	Post-Optimization	Performance Gain

Top-Selling Products (Query 1)	0.056 seconds	0.053 seconds	Slight improvement
Profit by Region (Query 2)	0.056 seconds	0.053 seconds	Slight improvement
Revenue by Sales Channel (Query 3)	0.068 seconds	0.038 seconds	~44% faster

Although the gains in Query 1 and 2 were minor, the overall database performance improved, especially for Query 3, which ran 44% faster after optimization. These results confirm that normalization and indexing improve performance while setting a solid foundation for future scalability.

References

Western Governors University. (2025). Scenario 2 – EcoMart. [Unpublished instructional material]. Western Governors University.

Appendix

Provided code below

```sql
CREATE TABLE sales_records_raw (
    region VARCHAR(50),
    country VARCHAR(50),
    item_type VARCHAR(50),
    sales_channel VARCHAR(30),
    order_priority VARCHAR(20),
    order_date DATE,
    order_id INT PRIMARY KEY,
    ship_date DATE,
    units_sold INT,
    unit_price DECIMAL(10,2),
    unit_cost DECIMAL(10,2),
    total_revenue DECIMAL(10,2),
    total_cost DECIMAL(10,2),
    total_profit DECIMAL(10,2)
);

--  Import data from CSV using pgAdmin import tool
-- Use pgAdmin GUI: Right-click table > Import/Export > Choose CSV
--Checking the import --
SELECT * FROM Sales_Records_raw


-- Step 3: Create lookup tables
CREATE TABLE regions (
    region_id SERIAL PRIMARY KEY,
    region_name VARCHAR(50) UNIQUE
);

CREATE TABLE countries (
    country_id SERIAL PRIMARY KEY,
    country_name VARCHAR(50),
    region_id INT REFERENCES regions(region_id)
);
```

```sql
CREATE TABLE item_types (
    item_type_id SERIAL PRIMARY KEY,
    item_type_name VARCHAR(50) UNIQUE
);

-- Step 4: Populate lookup tables
INSERT INTO regions (region_name)
SELECT DISTINCT region FROM sales_records_raw;

INSERT INTO countries (country_name, region_id)
SELECT DISTINCT sr.country, r.region_id
FROM sales_records_raw sr
JOIN regions r ON sr.region = r.region_name;

INSERT INTO item_types (item_type_name)
SELECT DISTINCT item_type FROM sales_records_raw;

-- Step 5: Create final normalized table
CREATE TABLE sales_records (
    order_id INT PRIMARY KEY,
    country_id INT REFERENCES countries(country_id),
    item_type_id INT REFERENCES item_types(item_type_id),
    sales_channel VARCHAR(30),
    order_priority VARCHAR(20),
    order_date DATE,
    ship_date DATE,
    units_sold INT,
    unit_price DECIMAL(10,2),
    unit_cost DECIMAL(10,2),
    total_revenue DECIMAL(10,2),
    total_cost DECIMAL(10,2),
    total_profit DECIMAL(10,2)
);
```

```sql
-- Step 6: Populate normalized table
INSERT INTO sales_records (
    order_id, country_id, item_type_id, sales_channel, order_priority,
    order_date, ship_date, units_sold, unit_price, unit_cost,
    total_revenue, total_cost, total_profit
)
SELECT
    sr.order_id,
    c.country_id,
    it.item_type_id,
    sr.sales_channel,
    sr.order_priority,
    sr.order_date,
    sr.ship_date,
    sr.units_sold,
    sr.unit_price,
    sr.unit_cost,
    sr.total_revenue,
    sr.total_cost,
    sr.total_profit
FROM sales_records_raw sr
JOIN countries c ON sr.country = c.country_name
JOIN item_types it ON sr.item_type = it.item_type_name;

-- Step 7: Create indexes
CREATE INDEX idx_country_id ON sales_records(country_id);
CREATE INDEX idx_item_type_id ON sales_records(item_type_id);
CREATE INDEX idx_sales_channel ON sales_records(sales_channel);
CREATE INDEX idx_total_profit ON sales_records(total_profit);

-- Step 8: Analyze database
ANALYZE;
```

```sql
-- Step 8: Analyze database
ANALYZE;

--Queries

-- Query 1: What are the top-selling product types?
SELECT it.item_type_name, SUM(sr.units_sold) AS total_units_sold
FROM sales_records sr
JOIN item_types it ON sr.item_type_id = it.item_type_id
GROUP BY it.item_type_name
ORDER BY total_units_sold DESC
LIMIT 5;

-- Query 2: What is the total revenue by region?
SELECT r.region_name, SUM(sr.total_revenue) AS total_revenue
FROM sales_records sr
JOIN countries c ON sr.country_id = c.country_id
JOIN regions r ON c.region_id = r.region_id
GROUP BY r.region_name
ORDER BY total_revenue DESC;

-- Query 3: How much revenue is generated per sales channel?
SELECT sales_channel, SUM(total_revenue) AS total_revenue
FROM sales_records
GROUP BY sales_channel
ORDER BY total_revenue DESC;
```