Task 1: Relational Database Design and Implementation
By Eric Williams

A1: BUSINESS PROBLEM

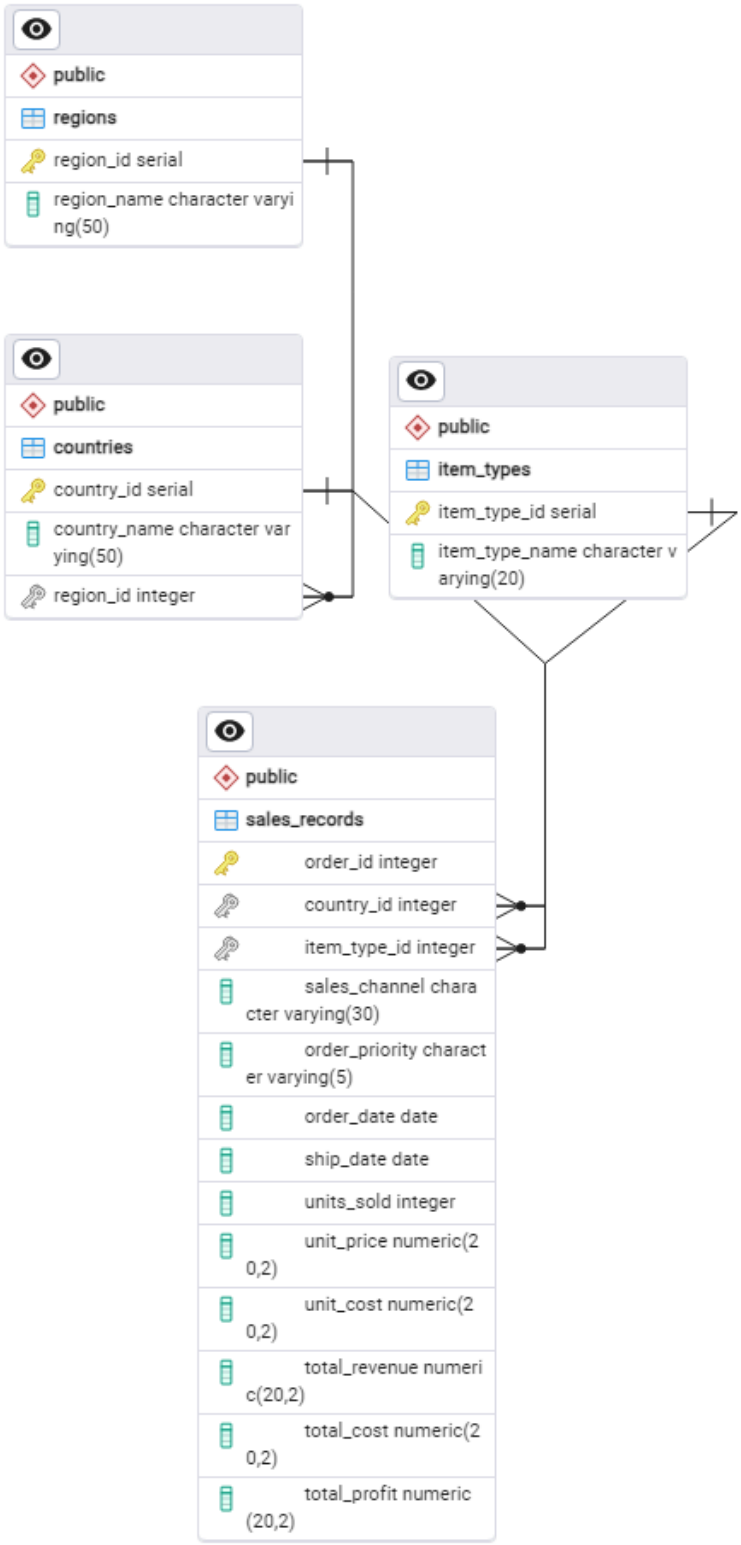
I have chosen Scenario 2 for my project. In this scenario, EcoMart wants to update their database to be more flexible and scalable so it can accommodate a diverse range of products and increased data volumes. They also wanted the security measures to be safe, including encryption, to protect consumer data. Lastly, they want it to be easy to maintain. This can be addressed with a database design that is conducive to their needs, which I will explain below.

A2: DATA STRUCTURE and A3: DATABASE JUSTIFICATION

A structured relational database would be a great tool for this job because the data EcoMart currently has is well structured and is organized in neat rows. The first step to creating a structured database will be to organize the data in tables. Currently, the dataset is in a csv file, so it will have to be imported. Also, it is currently in First Normal Form (1NF), as the columns have atomic values and the Order_ID could serve as the primary key. But with 19 different columns and 100,000 records, the data could be organized in a more convenient way. In its current state, it would be cumbersome and slow to query.

The data could instead be structured in Third Normal Form (3NF) using multiple tables. I could do this by a) ensuring every column is fully dependent on the primary key and b) eliminating transitive dependencies by splitting the tables into related tables. By splitting the tables into Region, Country, Item_Type, and Sales_Records, it would be 3NF and could be more functional and scalable for EcoMart. By organizing the data and splitting it to be 3NF, the queries will have much, much less data to sift through to produce results. There will also be less redundancies, so there will be just as much information but not as many entries. For example, there are thousands of duplicate rows regarding the region and country for the orders. My design will eliminate these redundancies.

B: Logical Data Model



C: OBJECTS AND STORAGE
The tables in my database are Sales_Records, Countries, Item_Types, and Regions. The primary key is Order_ID for the Sales_Records table. The foreign keys are Region_ID, Country_ID, and Item_ID, in the Sales_Records table, as they each link to other tables where those are the primary keys.

To summarize what is pictured above, here is a description of the storage in each table:

- The Countries table includes Country_ID as the primary key. It also contains:
    - Country_name VARCHAR(50)
    - Region_ID INT which references Regions(Region_ID)

- The Item_Types table contains the Item_Type_ID as a primary key. It also contains:
    - Item_Type_Name VARCHAR(20)

- The Regions table contains Region_ID INT as the primary key. It also contains:
    - Region_Name VARCHAR(50)

- The Sales_Records table contains Order_ID INT as a primary key. It also contains:
    - Country_ID INT which references Countries(Country_ID)
    - Item_Type_ID INT which references Item_Types(Item_Type_ID)
    - Sales_Channel VARCHAR(30)
    - Order_Priority VARCHAR(5)
    - Order_Date DATE
    - Ship_Date DATE
    - Units_Sold INTEGER
    - Unit_Price DECIMAL(20, 2)
    - Unit_Cost DECIMAL(20, 2)
    - Total_Revenue DECIMAL(20, 2)
    - Total_Cost DECIMAL(20, 2)
    - Total_Profit DECIMAL(20, 2)

D: SCALABILITY
As discussed above, now that the data is structured in Third Normal Form (3NF), it will be much easier to scale. Instead of just adding new orders to the bottom of the massive dataset, the data will be added in an organized way. Assuming there are no new regions or countries to ship to, those tables will never have to be edited or added to another row. This will save a lot of space. In the event that it becomes necessary to change or add a country or region, we can do so without the need to comb through and edit thousands of rows of data. This makes the data much easier to scale than it did before. Reducing redundancies in the country, region, item ID overall reduced the overall space of the dataset, which will ensure the server can handle larger datasets and thus more orders.

E: PRIVACY AND SECURITY
Because consumer data is considered sensitive information, it must be protected. This means the data would need to be encrypted. The company could also create access controls and require audit logging, as well as using 2-factor authentication for anyone accessing the database. They could also conduct regular security tests to make sure the security is up to date and that the data is not able to be breached from the outside.

F1: DATABASE INSTANCE

Here is my script to create a database based on the logical data model I described above. I created the table Sales_Records and included all the column names that will be needed for importing:

```
CREATE TABLE Sales_Records (
    Region VARCHAR(50),
    Country VARCHAR(50),
    Item_Type VARCHAR(20),
    Sales_Channel VARCHAR(30),
    Order_Priority CHAR(5),
    Order_Date DATE,
    Order_ID INT PRIMARY KEY,
    Ship_Date DATE,
    Units_Sold INT,
    Unit_Price DECIMAL(20, 2),
    Unit_Cost DECIMAL(20, 2),
    Total_Revenue DECIMAL(20, 2),
    Total_Cost DECIMAL(20, 2),
    Total_Profit DECIMAL(20, 2)
);
```

I also created a table for the redundant information (Regions, Countries, and Item_Types). Because this data was repeated so often, I put them in their own table to save on space and to improve query speed. After this is complete, the data will be in 3NF and much more efficient.

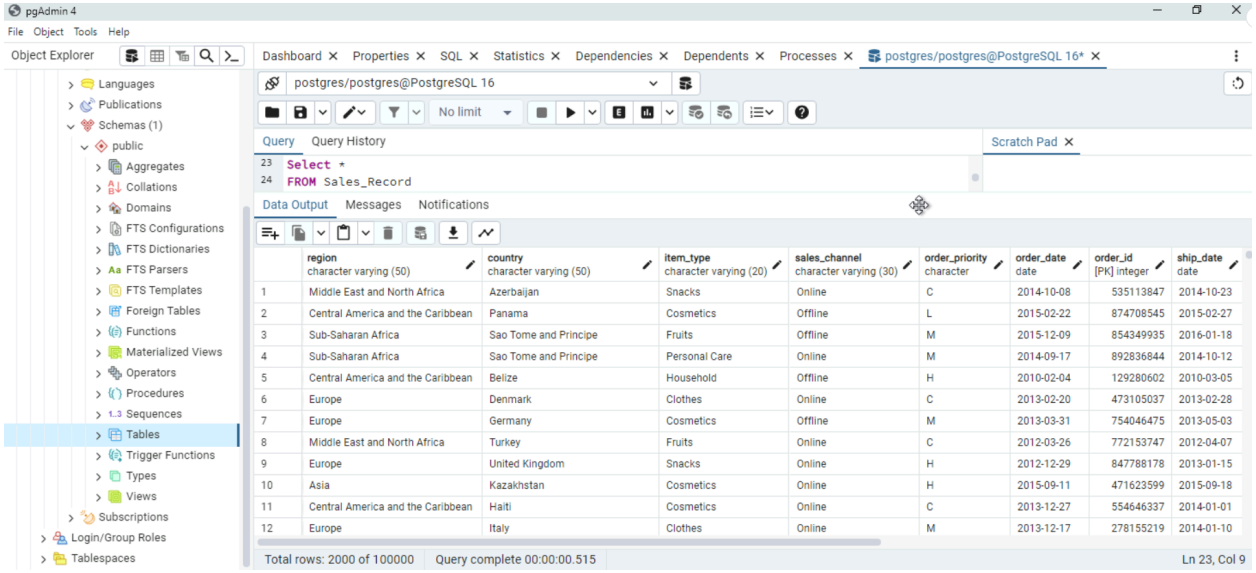
```
CREATE TABLE Regions (
    Region_ID SERIAL PRIMARY KEY,
    Region_Name VARCHAR(50) UNIQUE
);

CREATE TABLE Countries (
    Country_ID SERIAL PRIMARY KEY,
    Country_Name VARCHAR(50) UNIQUE
);

CREATE TABLE Item_Types (
    Item_Type_ID SERIAL PRIMARY KEY,
    Item_Type_Name VARCHAR(20) UNIQUE
);
```

F2: INSERT RECORDS

I then inserted the records from the CSV file into the Sales_Record table:

```
--Importing the data --
COPY Sales_Records
From 'C:\WGU\D597\Task 1\Scenario 2\Sales_Records.csv'
    DELIMITER ','
    CSV HEADER;
```

I then checked to ensure everything was imported properly. Below is a screenshot of the Sales_Record populating in the WM:



	region character varying (50)	country character varying (50)	item_type character varying (20)	sales_channel character varying (30)	order_priority character	order_date date	order_id [PK] integer	ship_date date
1	Middle East and North Africa	Azerbaijan	Snacks	Online	C	2014-10-08	535113847	2014-10-23
2	Central America and the Caribbean	Panama	Cosmetics	Offline	L	2015-02-22	874708545	2015-02-27
3	Sub-Saharan Africa	Sao Tome and Principe	Fruits	Offline	M	2015-12-09	854349935	2016-01-18
4	Sub-Saharan Africa	Sao Tome and Principe	Personal Care	Online	M	2014-09-17	892836844	2014-10-12
5	Central America and the Caribbean	Belize	Household	Offline	H	2010-02-04	129280602	2010-03-05
6	Europe	Denmark	Clothes	Online	C	2013-02-20	473105037	2013-02-28
7	Europe	Germany	Cosmetics	Offline	M	2013-03-31	754046475	2013-05-03
8	Middle East and North Africa	Turkey	Fruits	Online	C	2012-03-26	772153747	2012-04-07
9	Europe	United Kingdom	Snacks	Online	H	2012-12-29	847788178	2013-01-15
10	Asia	Kazakhstan	Cosmetics	Online	H	2015-09-11	471623599	2015-09-18
11	Central America and the Caribbean	Haiti	Cosmetics	Online	C	2013-12-27	554646337	2014-01-01
12	Europe	Italy	Clothes	Online	M	2013-12-17	278155219	2014-01-10

I then inserted the data into the regions, countries, and item_type tables:

```
INSERT INTO Regions (Region_Name)
SELECT DISTINCT Region
FROM Sales_Records;

INSERT INTO Countries (Country_Name)
SELECT DISTINCT country
FROM Sales_Records;

INSERT INTO Item_Types (Item_Type_Name)
SELECT DISTINCT Item_Type
FROM Sales_Records;

--Check the import in Regions --
SELECT * FROM Regions
```

I then had to add columns to the sales record to replace the original region, countries, and item_type table. I also had to add the new foreign keys.

```
58  --Add the ID's to Sales Records--
59  ALTER TABLE Sales_Records
60      ADD COLUMN Region_ID INT,
61      ADD COLUMN Country_ID INT,
62      ADD COLUMN Item_Type_ID INT;
63
64  --Add the new Foreign Key columns --
65  UPDATE Sales_Records sr
66  SET Region_ID = r.Region_ID
67  FROM Regions r
68  WHERE sr.Region = r.Region_Name;
69
70  UPDATE Sales_Records sr
71  SET Country_ID = c.Country_ID
72  FROM Countries c
73  WHERE sr.Country = c.Country_Name;
74
75  UPDATE Sales_Records sr
76  SET Item_Type_ID = it.Item_Type_ID
77  FROM Item_Types it
78  WHERE sr.Item_Type = it.Item_Type_Name;
79
80  ALTER TABLE Sales_Records
81      ADD CONSTRAINT fk_region FOREIGN KEY (Region_ID) REFERENCES Regions(Re
82      ADD CONSTRAINT fk_country FOREIGN KEY (Country_ID) REFERENCES Countrie
83      ADD CONSTRAINT fk_item_type FOREIGN KEY (Item_Type_ID) REFERENCES Item
84
```

The last step was to drop the redundant columns, finally completing the process of making our database more efficient:

```
89  --Drop Redundant Tables--
90  ALTER TABLE Sales_Records
91      DROP COLUMN Region,
92      DROP COLUMN Country,
93      DROP COLUMN Item_Type;
94
```

F3: QUERIES

I wrote queries for three business questions using the new database. Because the business problem was to create a more efficient, optimized, scalable dataset and because the company values eco-friendly practices, I chose my questions to focus on business efficiency.

Question 1: Which regions are most and least profitable?

As they are an online marketplace with worldwide clientele, the business might be interested in which region they should target for growth and advertising because it is especially profitable. If a region is especially unprofitable, they might decide their resources would be better used elsewhere. Below is the query,

```
--Three business queries --
--1. Which Regions are most and least profitable? --
SELECT Region_id, SUM(Total_Profit) AS Total_Profit
    FROM Sales_Records
    GROUP BY Region_id
    ORDER BY Total_Profit DESC;
```

This is the result:

	region_id integer	total_profit numeric
1	5	10306312642.23
2	7	10080579491.05
3	2	5707511516.76
4	6	4979534378.88
5	3	4287210522.47
6	1	3175423561.38
7	4	872551616.84

Here is a cross reference to the regions table:

	region_id [PK] integer	region_name character varying (50)
1	1	Australia and Oceania
2	2	Asia
3	3	Central America and the Caribbean
4	4	North America
5	5	Sub-Saharan Africa
6	6	Middle East and North Africa
7	7	Europe

Question 2: What are the top selling products?

For similar reasons, EcoMart might want to look at what products are doing well and which are not. If they wish to stay efficient and eco friendly, they should analyze if any of their products aren't selling well enough to justify from a profit and environment standpoint. Below is the query:

```
--2. What are the top selling products? --
SELECT Item_Type_id, SUM(Units_Sold) AS Total_Units_Sold
FROM Sales_Records
GROUP BY Item_Type_id
ORDER BY Total_Units_Sold DESC
LIMIT 10;
```

Here is the result:

	item_type_id integer	total_units_sold bigint
1	9	42293330
2	6	42254418
3	7	41924464
4	2	41911620
5	5	41773440
6	8	41745367
7	11	41699092
8	12	41517766
9	3	41514213
10	10	41458795

Here is a cross reference to the item types:

	item_type_id [PK] integer	item_type_name character varying (20)
1	1	Fruits
2	2	Baby Food
3	3	Beverages
4	4	Vegetables
5	5	Clothes
6	6	Cereal
7	7	Cosmetics
8	8	Meat
9	9	Office Supplies
10	10	Household
11	11	Snacks
12	12	Personal Care

Question 3: Which sales channel produces the most revenue?

As mentioned in the video of my report, EcoMart might be interested in whether online or offline sales produce the most revenue. If they wish to be more eco-friendly, they could push to do more online sales and to go paperless for orders, thus meeting their business goals. Below is the query:

```
--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;
```

Here is the result:

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

F4: OPTIMIZATION

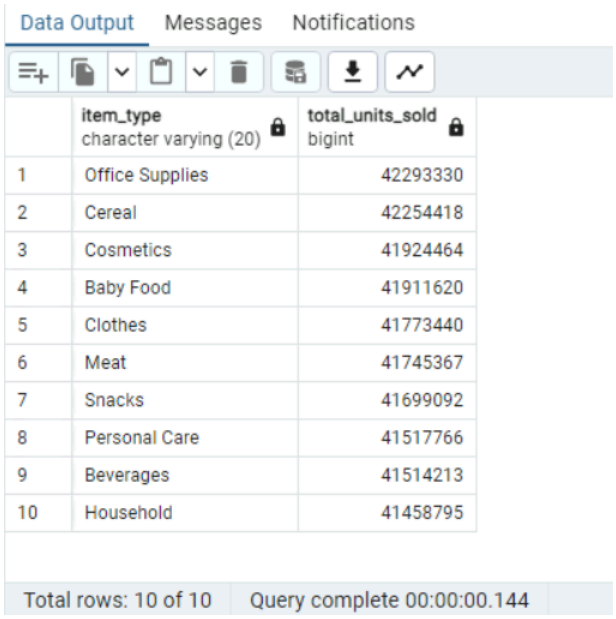
I provided the optimized techniques above to produce the queries in F3; I converted the data structure to be 3NF by ensuring every column is dependent on primary key and eliminating transitive dependencies; I defined a primary index with the primary key Order_ID; and I performed single column indexing for Country, Item Type, and Region, thus reducing the overall datasize and improving efficiency. This essentially maximized the optimization in the queries above.

Below, I have the queries and results BEFORE the optimization:

```
CREATE TABLE Sales_Record (
    Region VARCHAR(50),
    Country VARCHAR(50),
    Item_Type VARCHAR(20),
    Sales_Channel VARCHAR(30),
    Order_Priority CHAR(5),
    Order_Date DATE,
    Order_ID INT PRIMARY KEY,
    Ship_Date DATE,
    Units_Sold INT,
    Unit_Price DECIMAL(20, 2),
    Unit_Cost DECIMAL(20, 2),
    Total_Revenue DECIMAL(20, 2),
    Total_Cost DECIMAL(20, 2),
    Total_Profit DECIMAL(20, 2)
);
```

```
--Importing the data --
COPY Sales_Record
From 'C:\WGU\D597\Task 1\Scenario 2\Sales_Records.csv'
    DELIMITER ','
    CSV HEADER;
```

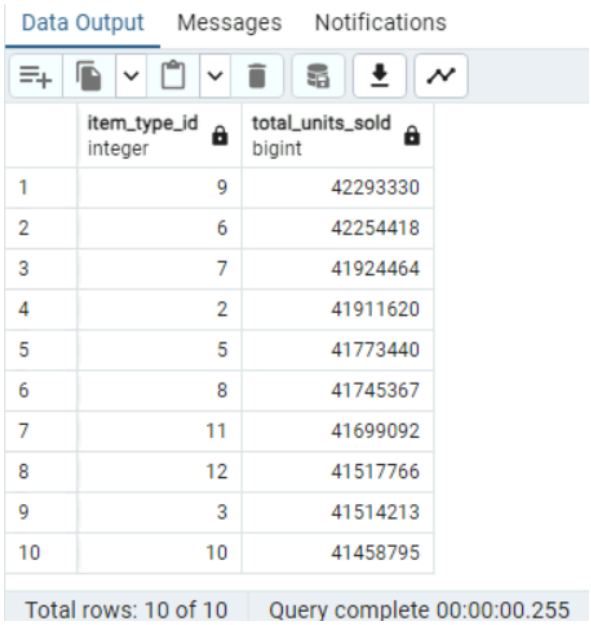
I then performed the 3 queries above and found a decrease in data load time from an average of 0.19 seconds to about .3 seconds. The optimized, updated model listed above provides results roughly 2-4 times faster than before. Here are the two screenshots of two query results on question 2. Note that the optimized runtime is 0.144 seconds and the less optimized run time of 0.255 seconds.



Data Output | Messages | Notifications

	item_type character varying (20)	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

Total rows: 10 of 10 Query complete 00:00:00.144



Data Output | Messages | Notifications

	item_type_id integer	total_units_sold bigint
1	9	42293330
2	6	42254418
3	7	41924464
4	2	41911620
5	5	41773440
6	8	41745367
7	11	41699092
8	12	41517766
9	3	41514213
10	10	41458795

Total rows: 10 of 10 Query complete 00:00:00.255

Sources

No Sources were used besides the WGU course materials.

Appendix A
All the code

```
--Requirement 1. Discuss how database design and indexing strategy optimize
performance--
--Requirement 2.  Describe the technical environment used in your database
implementation --
    --Normalization 1N to 3N (ensure every column is dependent on primary key,
eliminate transitive dependencies)
    --primary indexes (primary key Order_ID)
    --single column index (Country, Item Type, and Region)
--Requirement 3. Demonstrate the functionality of the queries in the lab
environment.
--Requirement 4 will be discussed after the queries)

CREATE TABLE Sales_Records (
    Region VARCHAR(50),
    Country VARCHAR(50),
    Item_Type VARCHAR(20),
    Sales_Channel VARCHAR(30),
    Order_Priority CHAR(5),
    Order_Date DATE,
    Order_ID INT PRIMARY KEY,
    Ship_Date DATE,
    Units_Sold INT,
    Unit_Price DECIMAL(20, 2),
    Unit_Cost DECIMAL(20, 2),
    Total_Revenue DECIMAL(20, 2),
    Total_Cost DECIMAL(20, 2),
    Total_Profit DECIMAL(20, 2)
);

--Importing the data --
COPY Sales_Records
From 'C:\WGU\D597\Task 1\Scenario 2\Sales_Records.csv'
    DELIMITER ','
    CSV HEADER;

--Checking the import --
```

```sql
SELECT * FROM Sales_Records

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) UNIQUE
);

CREATE TABLE Item_Types (
    Item_Type_ID SERIAL PRIMARY KEY,
    Item_Type_Name VARCHAR(20) UNIQUE
);


INSERT INTO Regions (Region_Name)
SELECT DISTINCT Region
FROM Sales_Records;

INSERT INTO Countries (Country_Name)
SELECT DISTINCT country
FROM Sales_Records;

INSERT INTO Item_Types (Item_Type_Name)
SELECT DISTINCT Item_Type
FROM Sales_Records;

--Check the import in Regions --
SELECT * FROM Regions

--Add the ID's to Sales Records--
ALTER TABLE Sales_Records
    ADD COLUMN Region_ID INT,
    ADD COLUMN Country_ID INT,
    ADD COLUMN Item_Type_ID INT;
```

```sql
--Add the new Foreign Key columns --
UPDATE Sales_Records sr
SET Region_ID = r.Region_ID
FROM Regions r
WHERE sr.Region = r.Region_Name;

UPDATE Sales_Records sr
SET Country_ID = c.Country_ID
FROM Countries c
WHERE sr.Country = c.Country_Name;

UPDATE Sales_Records sr
SET Item_Type_ID = it.Item_Type_ID
FROM Item_Types it
WHERE sr.Item_Type = it.Item_Type_Name;

ALTER TABLE Sales_Records
    ADD CONSTRAINT fk_region FOREIGN KEY (Region_ID) REFERENCES
Regions(Region_ID),
    ADD CONSTRAINT fk_country FOREIGN KEY (Country_ID) REFERENCES
Countries(Country_ID),
    ADD CONSTRAINT fk_item_type FOREIGN KEY (Item_Type_ID) REFERENCES
Item_Types(Item_Type_ID);

--Check Sales_Records to see everything loaded properly--

SELECT * FROM Sales_Records

--Drop Redundant Tables--
ALTER TABLE Sales_Records
    DROP COLUMN Region,
    DROP COLUMN Country,
    DROP COLUMN Item_Type;

--Requirement 4. Discuss how the queries solve the identified business problem--
    --Business problem: Need a flexible, scalable database, and optimization--

--Three business queries --
--1. Which Regions are most and least profitable? --
```

```sql
SELECT Region_id, SUM(Total_Profit) AS Total_Profit
    FROM Sales_Records
    GROUP BY Region_id
    ORDER BY Total_Profit DESC;

--2. What are the top selling products? --
SELECT Item_Type_id, SUM(Units_Sold) AS Total_Units_Sold
FROM Sales_Records
GROUP BY Item_Type_id
ORDER BY Total_Units_Sold DESC
LIMIT 10;

--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;
```