Normalized Database Documentation

1. Introduction to Database Normalization:

In this part of the project, we set up a normalized database structure so that the FlipKart e-commerce data could be stored easily. Database normalization is a methodical way to arrange data to get rid of duplicates and make sure the data is correct. Third Normal Form (3NF), which says that our application must include:

- Everything is based on the key (1NF).
- All attributes that aren't keys rely on the primary key as a whole (2NF).
- Another non-key trait does not depend on this one (3NF).

By using 3NF, we got rid of duplicate data, lowered the chance of strange updates, and built a strong base for our data warehouse.

Datawarehouse Platform Selected:- SQL Server

As my data warehouse tool, I used **SQL SERVER** to set up this normalized database. When I chose SQL Server, it was because it worked well with relational data structures, supported complex constraints and foreign key relationships, and worked quickly with structured data in a normalized context.

SQL Server was also a good choice for showing how a real-world data warehouse would be set up because it is widely used in business settings. The management tools on the platform made it easy to see how the information was organized, which helped make sure that the 3NF principles were being followed correctly.

In section 3.1 of the documentation, you can find the full DDL scripts for making all seven normalized tables with their correct constraints. Sections 5.1 through 5.4 show more verification scripts.

2. Creating the Normalized Database:

First, we made a new database called DB_FlipKart to store our tables that had been standardized. The database was made to hold information from three main places:

- Customer data (like demographics and buying habits)
- Products data (details and groupings of products)

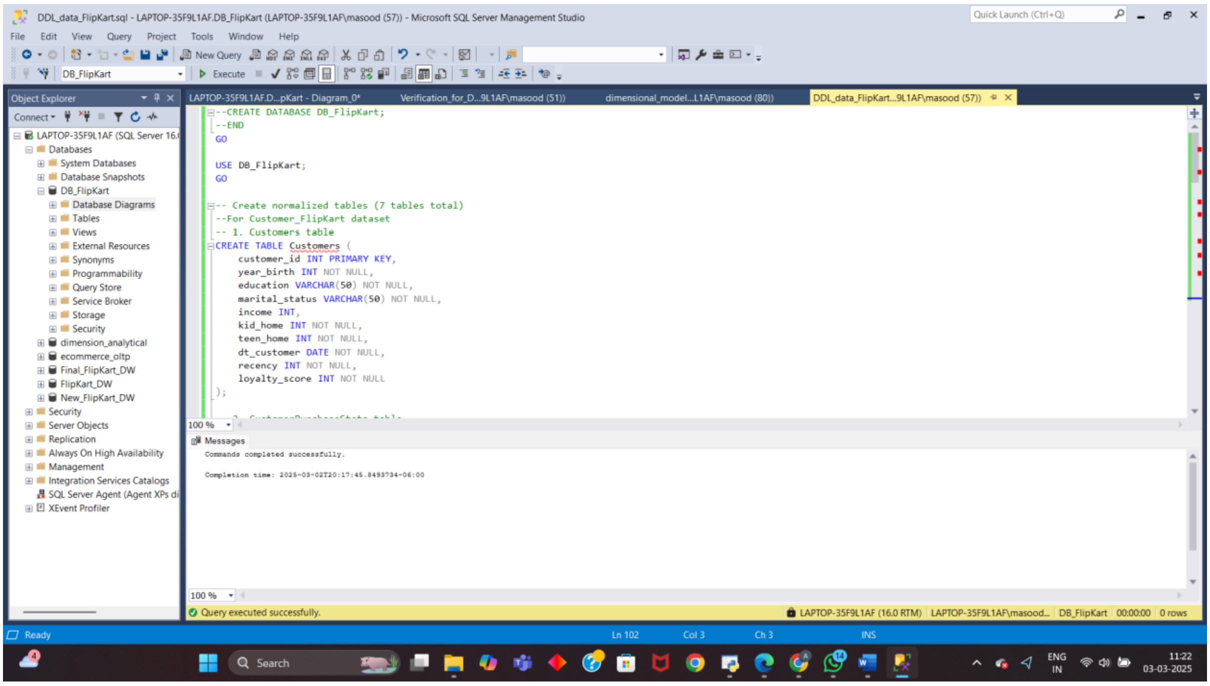
- Sales data (like orders, purchases, and shipping)

3. Normalized Table Structure:

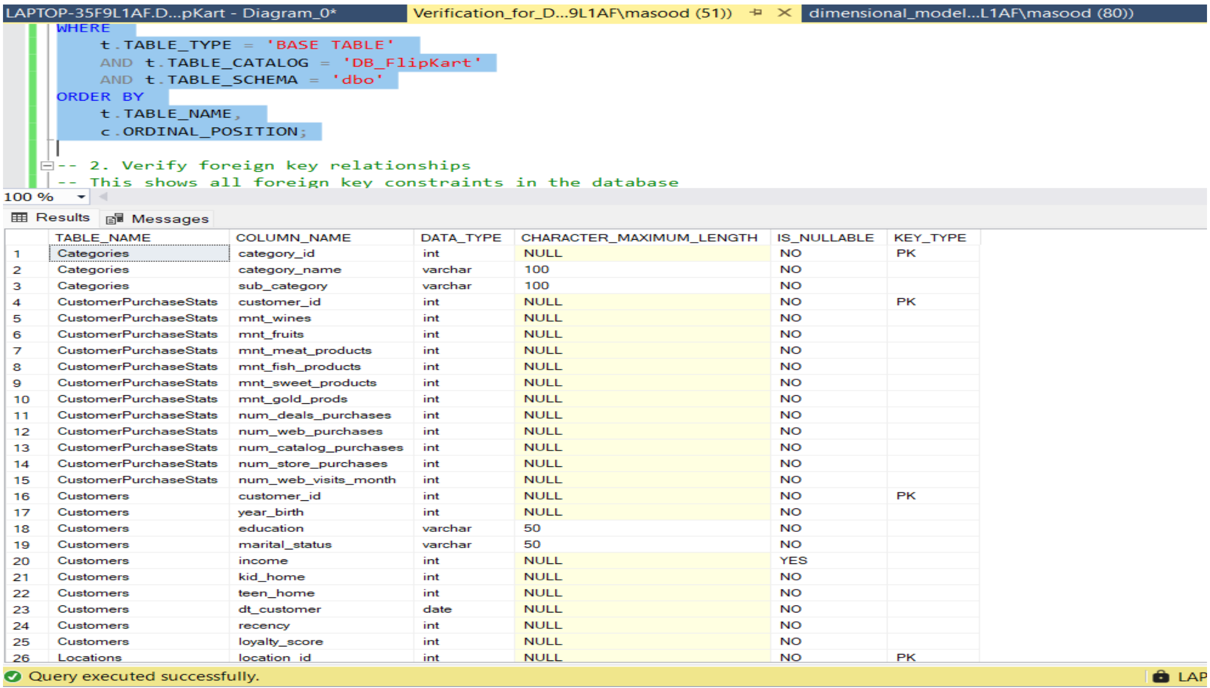
To get 3NF, we set up different tables for each of the key entities in our data. Utilizing this method got rid of unnecessary information while keeping data connections by using primary and foreign keys.

a. Customers and Purchase Behavior Tables:

We separated demographic information about customers from information about what they bought, making two linked tables called Customers and CustomerPurchaseStats. We kept demographic information in the Customers table, like the customer's year of birth, level of education, and income. The main key was the customer's ID. The foreign key customer_id connects the CustomerPurchaseStats table to the Customers table and shows how much customers spend on different types of products and how often they buy them.



3.1 DDL statements for creating Tables



LAPTOP-35F9L1AF.D...pKart - Diagram_0* | Verification_for_D...9L1AF\masood (51)) | dimensional_model...L1AF\masood (80))

```sql
WHERE
    t.TABLE_TYPE = 'BASE TABLE'
    AND t.TABLE_CATALOG = 'DB_FlipKart'
    AND t.TABLE_SCHEMA = 'dbo'
ORDER BY
    t.TABLE_NAME,
    c.ORDINAL_POSITION;

-- 2. Verify foreign key relationships
-- This shows all foreign key constraints in the database
```

100 %

Results | Messages

	TABLE_NAME	COLUMN_NAME	DATA_TYPE	CHARACTER_MAXIMUM_LENGTH	IS_NULLABLE	KEY_TYPE
1	Categories	category_id	int	NULL	NO	PK
2	Categories	category_name	varchar	100	NO	
3	Categories	sub_category	varchar	100	NO	
4	CustomerPurchaseStats	customer_id	int	NULL	NO	PK
5	CustomerPurchaseStats	mnt_wines	int	NULL	NO	
6	CustomerPurchaseStats	mnt_fruits	int	NULL	NO	
7	CustomerPurchaseStats	mnt_meat_products	int	NULL	NO	
8	CustomerPurchaseStats	mnt_fish_products	int	NULL	NO	
9	CustomerPurchaseStats	mnt_sweet_products	int	NULL	NO	
10	CustomerPurchaseStats	mnt_gold_prods	int	NULL	NO	
11	CustomerPurchaseStats	num_deals_purchases	int	NULL	NO	
12	CustomerPurchaseStats	num_web_purchases	int	NULL	NO	
13	CustomerPurchaseStats	num_catalog_purchases	int	NULL	NO	
14	CustomerPurchaseStats	num_store_purchases	int	NULL	NO	
15	CustomerPurchaseStats	num_web_visits_month	int	NULL	NO	
16	Customers	customer_id	int	NULL	NO	PK
17	Customers	year_birth	int	NULL	NO	
18	Customers	education	varchar	50	NO	
19	Customers	marital_status	varchar	50	NO	
20	Customers	income	int	NULL	YES	
21	Customers	kid_home	int	NULL	NO	
22	Customers	teen_home	int	NULL	NO	
23	Customers	dt_customer	date	NULL	NO	
24	Customers	recency	int	NULL	NO	
25	Customers	loyalty_score	int	NULL	NO	
26	Locations	location_id	int	NULL	NO	PK

Query executed successfully.

3.2 Customers and CustomerPurchaseStats tables

By saving demographic information about a customer only once and keeping track of purchases in a linked table, this separation gets rid of unnecessary duplication.

b. Product and Category Tables:

We put category information into a different table so that product data wouldn't be duplicated. With an auto-generated category_id as its main key, the Categories table stores unique combinations of categories and subcategories. The Categories table is linked to the Products table through a foreign key, which holds information about the products, such as their name, price, and brand.

In line with 3NF principles, this layout keeps category information from being repeated across product records.

c. Orders and Location Tables:

We put normalized locations into a different table and set up a relationship structure for order data. Number of unique combinations of country, state, and city are kept in the Locations table. Foreign keys keep track of the connections between the Orders table and both Customers and Locations. As a transaction record, the OrderDetails table links orders to goods and stores information that is unique to the transaction.

In my normalized tables, I can see that the data was loaded correctly. It shows the following 7 tables that are normalised:

i. Customers: 1000 rows
ii. CustomerPurchaseStats: 1000 rows
iii. Categories: 15 rows
iv. Products: 1000 rows
v. Locations: 999 rows
vi. Orders: 1000 rows
vii. OrderDetails: 1000 rows

```
-- 3. Verify record counts in each table
-- This shows how many rows are in each normalized table
SELECT 'Customers' AS Table_Name, COUNT(*) AS Row_Count FROM Customers
UNION ALL
SELECT 'CustomerPurchaseStats', COUNT(*) FROM CustomerPurchaseStats
UNION ALL
SELECT 'Categories', COUNT(*) FROM Categories
UNION ALL
SELECT 'Products', COUNT(*) FROM Products
UNION ALL
SELECT 'Locations', COUNT(*) FROM Locations
UNION ALL
SELECT 'Orders', COUNT(*) FROM Orders
UNION ALL
SELECT 'OrderDetails', COUNT(*) FROM OrderDetails
ORDER BY Table_Name;


-- Verification for Loading Raw Data into Normalised Database
```

100 %

Results | Messages

	Table_Name	Row_Count
1	Categories	15
2	CustomerPurchaseStats	1000
3	Customers	1000
4	Locations	999
5	OrderDetails	1000
6	Orders	1000
7	Products	1000

3.3 Counts in each table

4. Mapping Original Datasets to Normalized Tables:

This is how the standardized tables connect to the three original datasets:

a. From **customers_FlipKart.csv**:

- Customers table (customer demographic data)
- CustomerPurchaseStats table (customer purchase behavior data)

b. From **products_FlipKart.csv**:

- Products table (product information)
- Categories table (extracted category and subcategory information)

c. From **sales_Flipkart.csv**:

- Orders table (order header information)
- OrderDetails table (transaction/line item information)
- Locations table (extracted location information)

To get 3NF, each of my original datasets were standardized into multiple tables by getting rid of transitive dependencies, getting rid of duplicate data, making sure there were correct key dependencies, and setting up the right relationships between entities that were related.

```
-- 2. Verify foreign key relationships
-- This shows all foreign key constraints in the database
SELECT
    fk.name AS FK_Name,
    OBJECT_NAME(fk.parent_object_id) AS Table_Name,
    COL_NAME(fkc.parent_object_id, fkc.parent_column_id) AS Column_Name,
    OBJECT_NAME(fk.referenced_object_id) AS Referenced_Table,
    COL_NAME(fkc.referenced_object_id, fkc.referenced_column_id) AS Referenced_Column
FROM
    sys.foreign_keys fk
INNER JOIN
    sys.foreign_key_columns fkc ON fk.object_id = fkc.constraint_object_id
ORDER BY
    Table_Name,
    Column_Name;
```

100 %

Results Messages

	FK_Name	Table_Name	Column_Name	Referenced_Table	Referenced_Column
1	FK__CustomerP__custo__398D8EEE	CustomerPurchaseStats	customer_id	Customers	customer_id
2	FK__OrderDeta__order__48CFD27E	OrderDetails	order_id	Orders	order_id
3	FK__OrderDeta__produ__49C3F6B7	OrderDetails	product_id	Products	product_id
4	FK__Orders__customer__44FF419A	Orders	customer_id	Customers	customer_id
5	FK__Orders__location__45F365D3	Orders	location_id	Locations	location_id
6	FK__Products__catego__3F466844	Products	category_id	Categories	category_id

4.1 Foreign key relationships

5. Loading Raw Data into Normalized Tables:

We had to load the data from our raw CSV files into our normalized database tables after constructing the normalized structure. Several crucial steps were involved in this process:

- We started by creating temporary staging tables that matched the CSV files' format.
- Then, we loaded the raw data into these temporary databases using BULK INSERT operations.
- The data was then converted and added to our normalized tables.
- Lastly, we verified the information to make sure referential integrity was preserved.

This was the SQL Query Script we followed:

```
-- Create temporary tables for the raw data

CREATE TABLE #temp_customers (...);

CREATE TABLE #temp_products (...);

CREATE TABLE #temp_sales (...);


-- Load data from CSV files into temporary tables

BULK INSERT #temp_customers

FROM 'C:\Users\masood\Downloads\Data Warehouse Project\Raw Data Folder\customers_FlipKart.csv'

WITH (

   FIELDTERMINATOR = ',',

   ROWTERMINATOR = '\n',

   FIRSTROW = 2

);


-- Similar BULK INSERT statements for other CSV files


-- Insert data into normalized tables in the correct order
```

```sql
-- First, insert categories and locations (lookup tables)
INSERT INTO Categories (category_name, sub_category)
SELECT DISTINCT category, sub_category FROM #temp_products;

INSERT INTO Locations (country, state, city)
SELECT DISTINCT country, state, city FROM #temp_sales;

-- Then insert customer data
INSERT INTO Customers (...) SELECT ... FROM #temp_customers;
INSERT INTO CustomerPurchaseStats (...) SELECT ... FROM #temp_customers;

-- Insert product data with category references
INSERT INTO Products (...)
SELECT p.*, c.category_id
FROM #temp_products p
JOIN Categories c ON p.category = c.category_name AND p.sub_category =
c.sub_category;

-- Finally, insert order data with appropriate relationships
INSERT INTO Orders (...) SELECT ... FROM #temp_sales;
INSERT INTO OrderDetails (...) SELECT ... FROM #temp_sales;
```

Verifying if Raw Data was loaded successfully into Normalized Database:

```sql
-- Shows sample data from each table to verify data was loaded correctly
SELECT TOP 5 * FROM Customers;
SELECT TOP 5 * FROM CustomerPurchaseStats;
SELECT TOP 5 * FROM Categories;
SELECT TOP 5 * FROM Products;
SELECT TOP 5 * FROM Locations;
SELECT TOP 5 * FROM Orders;
```

100 %

Results | Messages

	customer_id	year_birth	education	marital_status	income	kid_home	teen_home	dt_customer	recency	loyalty_score
1	1001	2002	Associate	Married	58545	2	3	2005-12-07	21	51
2	1002	1977	Master	Single	105408	3	3	1900-01-01	11	80
3	1003	1973	Master	Divorced	116851	1	1	2019-07-02	1	60
4	1004	1984	Master	Married	78974	3	2	1900-01-01	8	87
5	1005	1988	Graduate	Divorced	110761	1	1	1900-01-01	3	59

	customer_id	mnt_wines	mnt_fruits	mnt_meat_products	mnt_fish_products	mnt_sweet_products	mnt_gold_prods	num_deals_purchases	num_web_purchases	num_catalog_purchases	num_store_purchases	num_web_visits_month
1	1001	318	144	173	131	89	92	8	8	5	8	0
2	1002	422	193	243	8	46	29	7	1	0	5	8
3	1003	141	145	128	110	4	67	4	4	8	6	5
4	1004	131	32	209	97	14	8	7	10	9	4	8
5	1005	57	56	273	135	25	100	4	6	7	4	3

	category_id	category_name	sub_category
1	1	Electronics	Chairs
2	11	Electronics	Laptops
3	7	Electronics	Printers
4	10	Electronics	Smartphones
5	5	Electronics	Tables

	product_id	product_name	category_id	brand	price	stock_quantity	discount	rating	supplier
1	P3001	Fight Desk	9	HP	2208.51	248	24	2.9	Target
2	P3002	Threat Monitor	11	Lenovo	694.67	16	13	1.6	Walmart
3	P3003	No Laptop	15	IKEA	2110.28	290	3	4.6	Amazon
4	P3004	Buy Monitor	15	Dell	763.30	127	3	4.8	Newegg
5	P3005	Describe La...	6	HP	394.72	147	12	1.8	Best Buy

5.1 Verification 1- 4 tables loaded successfully into Normalized Database

During the loading process, the links between tables had to be carefully managed. Our first load was the lookup tables, which included Categories and Locations. Then came the main entities, which included Customers and Products, and finally the transactional data, which included Orders and OrderDetails. This order made sure that all references to foreign keys were met.

```sql
SELECT TOP 5 * FROM Locations;
SELECT TOP 5 * FROM Orders;
SELECT TOP 5 * FROM OrderDetails;
```

100 %

Results | Messages

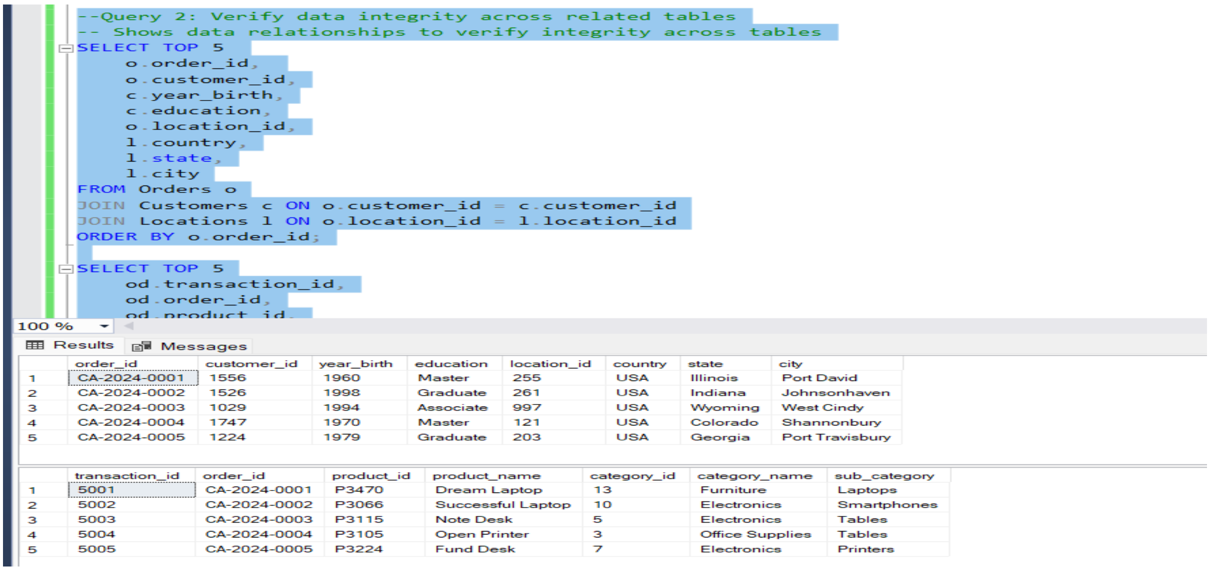
	location_id	country	state	city
1	1	USA	Alabama	Austinchester
2	2	USA	Alabama	Austinhaven
3	3	USA	Alabama	Chrischester
4	4	USA	Alabama	Dixonbury
5	5	USA	Alabama	East Christopher

	order_id	order_date	customer_id	segment	location_id
1	CA-2024-0001	2025-01-19	1556	Consumer	255
2	CA-2024-0002	2025-02-13	1526	Consumer	261
3	CA-2024-0003	2024-01-17	1029	Consumer	997
4	CA-2024-0004	2023-07-30	1747	Corporate	121
5	CA-2024-0005	2023-05-23	1224	Home Office	203

	transaction_id	order_id	product_id	ship_date	ship_mode	quantity	discount	sales	total_amount	payment_method	shipping_status
1	5001	CA-2024-0001	P3470	2024-03-10	Same Day	1	10.41	2202.06	1654.07	Bank Transfer	Shipped
2	5002	CA-2024-0002	P3066	2023-07-23	Same Day	4	0.89	2254.52	1677.33	Bank Transfer	Cancelled
3	5003	CA-2024-0003	P3115	2023-10-24	Standard	2	12.81	482.08	2041.51	Debit Card	Shipped
4	5004	CA-2024-0004	P3105	2024-03-10	Standard	3	8.83	1230.33	764.96	Credit Card	In Transit
5	5005	CA-2024-0005	P3224	2023-11-17	Express	3	10.91	722.98	2494.89	PayPal	Delivered

5.2 Verification 2- 3 tables loaded successfully into Normalized Database

The sample data confirms successful data loading since it matches the expected values from the raw CSV files. Details about customers, products, and orders are arranged correctly, and the linkages between tables are preserved. The normalized database's data consistency and integrity are guaranteed by the values' seeming consistency.



```sql
--Query 2: Verify data integrity across related tables
-- Shows data relationships to verify integrity across tables
SELECT TOP 5
    o.order_id,
    o.customer_id,
    c.year_birth,
    c.education,
    o.location_id,
    l.country,
    l.state,
    l.city
FROM Orders o
JOIN Customers c ON o.customer_id = c.customer_id
JOIN Locations l ON o.location_id = l.location_id
ORDER BY o.order_id;

SELECT TOP 5
    od.transaction_id,
    od.order_id,
    od.product_id,
```

100 %

Results | Messages

	order_id	customer_id	year_birth	education	location_id	country	state	city
1	CA-2024-0001	1556	1960	Master	255	USA	Illinois	Port David
2	CA-2024-0002	1526	1998	Graduate	261	USA	Indiana	Johnsonhaven
3	CA-2024-0003	1029	1994	Associate	997	USA	Wyoming	West Cindy
4	CA-2024-0004	1747	1970	Master	121	USA	Colorado	Shannonbury
5	CA-2024-0005	1224	1979	Graduate	203	USA	Georgia	Port Travisbury

	transaction_id	order_id	product_id	product_name	category_id	category_name	sub_category
1	5001	CA-2024-0001	P3470	Dream Laptop	13	Furniture	Laptops
2	5002	CA-2024-0002	P3066	Successful Laptop	10	Electronics	Smartphones
3	5003	CA-2024-0003	P3115	Note Desk	5	Electronics	Tables
4	5004	CA-2024-0004	P3105	Open Printer	3	Office Supplies	Tables
5	5005	CA-2024-0005	P3224	Fund Desk	7	Electronics	Printers

5.3 Data integrity and relationships between normalized tables

The results of Query 2 show that the relationships and data consistency between normalized tables were kept. Orders are correctly linked to customers and locations, and the details of an order are correctly linked to goods and categories. The query checks for foreign key constraints, which makes sure that the database is well-structured and normalized and that there are no empty records. This proves that the 3NF rules are being followed while keeping the ability to analyze.

```sql
--3.  Shows statistical measures to verify data quality
SELECT
    'Sales Statistics' AS Data_Check,
    COUNT(*) AS Total_Transactions,
    SUM(quantity) AS Total_Quantity,
    MIN(sales) AS Min_Sale,
    MAX(sales) AS Max_Sale,
    AVG(sales) AS Avg_Sale,
    SUM(total_amount) AS Total_Revenue
FROM OrderDetails;

SELECT
    'Product Statistics' AS Data_Check,
    COUNT(*) AS Total_Products,
    MIN(price) AS Min_Price,
    MAX(price) AS Max_Price,
    AVG(price) AS Avg_Price,
    COUNT(DISTINCT category_id) AS Distinct_Categories
FROM Products;
```

100 %

Results | Messages

	Data_Check	Total_Transactions	Total_Quantity	Min_Sale	Max_Sale	Avg_Sale	Total_Revenue
1	Sales Statistics	1000	3065	52.35	2499.76	1284.338100	1307953.36

	Data_Check	Total_Products	Min_Price	Max_Price	Avg_Price	Distinct_Categories
1	Product Statistics	1000	50.71	2499.76	1310.684960	15

	Data_Check	Total_Customers	Avg_Birth_Year	Min_Income	Max_Income	Avg_Income
1	Customer Statistics	1000	1977	20168	119937	70954

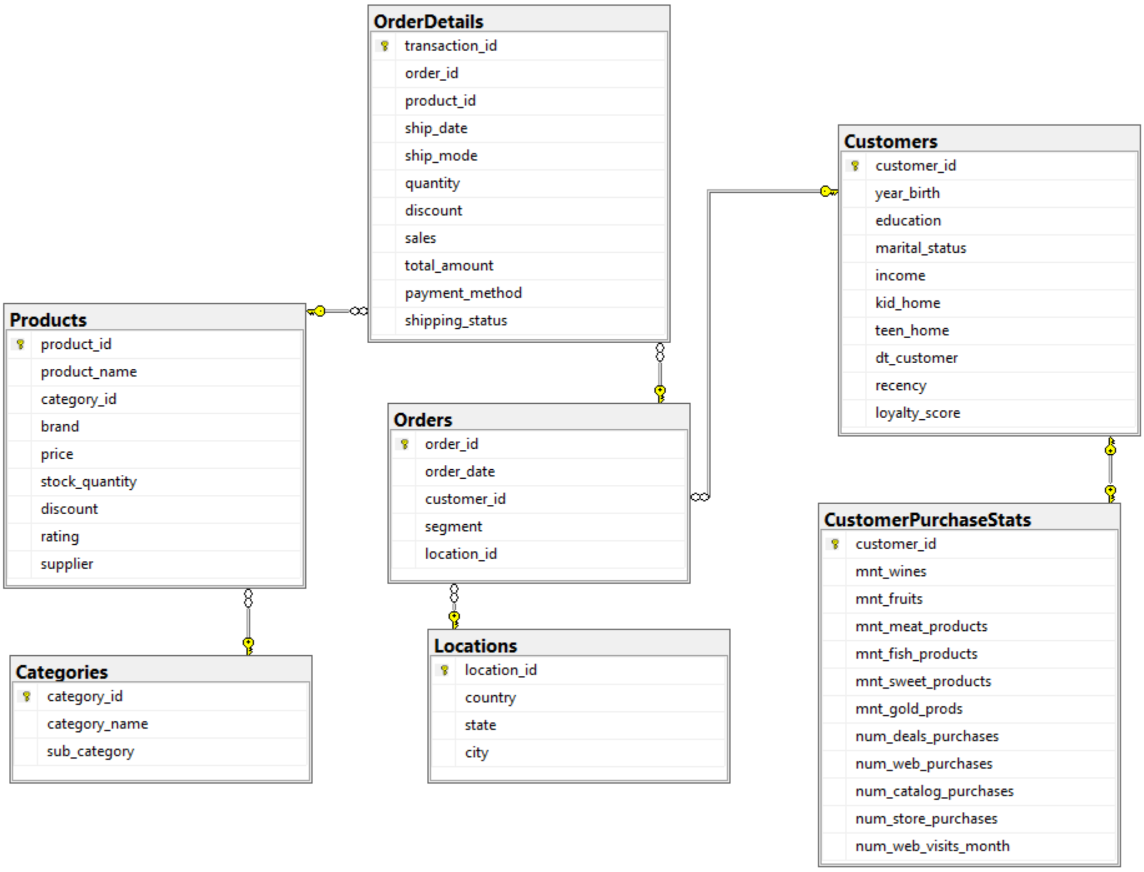
5.4 Verifying data integrity with statistics

The results shown show that our data was loaded properly, with the right ranges of values and expected counts. Totals are the same across tables that are linked, and the statistical measures (lowest, highest, and averages) look right for an e-commerce dataset. This proves that our normalized database not only keeps its structure sound by using the right relationships, but it also keeps the original data values' correctness.

This is the last step in validating the normalized database implementation. It shows that we were able to change the raw data into a properly normalized structure while keeping the data's quality and accuracy.

6. Entity-Relationship Diagram:

Our normalized database structure is shown in the ER diagram, which includes the tables, their characteristics, and the connections between them. This diagram shows that our database follows the 3NF rules, with different tables for each type of entity, primary keys for unique identification, foreign keys for keeping track of relationships, and no dependencies that go from one type of entity to another.



6.1 ER Diagram

The ER model shows the normalized database structure correctly, showing all seven tables with their correct fields and primary keys. It shows relationships correctly, such as one-to-one relationships (Customers and CustomerPurchaseStats) and one-to-many relationships (Categories to Products, Orders to OrderDetails, etc.), making sure that the referential integrity is right. Each table is well-organized, with the main keys at the top and attributes that are set up in a way that makes sense.

The diagram shows that the database follows 3NF, which gets rid of unnecessary information while keeping clear connections between entities. This is a great visual aid for learning how the database is set up.

7. Conclusion:

Our normalized database implementation successfully turned three source files of raw CSV data into a well-structured relational database that follows Third Normal Form rules to the letter. The first step was to carefully look over the data sources to find entities, attributes, and connections between them. Then, we made seven tables that were properly normalized: Orders, Categories, Products, Locations, Customers, and OrderDetails. By putting similar attributes into different tables, like location information and category data being kept separate, this structure got rid of data duplication. Using main and foreign key constraints makes sure that all of the references in the database are correct. Statistical proof shows that all 1000 records from each source file were loaded correctly, indicating that the normalization process kept the quality of the data while making the structure more organized.

All the standards for the normalized database deliverable were met by the implementation. As shown by our schema verification, we made complete DDL scripts for the database schema with clearly stated constraints. The database fully meets the requirements of the 3NF by making sure that all non-key attributes only depend on the main key and not on any other non-key attributes.

The way we loaded the data correctly changed the raw CSV data into the normalized structure, keeping the connections between the entities. Referential integrity is kept because all foreign keys are correctly linked to their parent tables by the verification searches. The ER diagram makes the normalized structure easy to see by showing how the seven tables work together through clear connections. We will use this standardized database as a strong base for the next steps, which are ETL and dimensional modeling, to turn it into an analytical data warehouse.