**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.

A computer screen with a white screen

AI-generated content may be incorrect.

3.1 DDL statements for creating Tables

A screenshot of a computer

AI-generated content may be incorrect.

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:

1. Customers: 1000 rows
2. CustomerPurchaseStats: 1000 rows
3. Categories: 15 rows
4. Products: 1000 rows
5. Locations: 999 rows
6. Orders: 1000 rows
7. OrderDetails: 1000 rows

A screenshot of a computer

AI-generated content may be incorrect.

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.

A screenshot of a computer

AI-generated content may be incorrect.

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

-- 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:**

**A screenshot of a computer

AI-generated content may be incorrect.**

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.

A screenshot of a computer

AI-generated content may be incorrect.

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.

A screenshot of a computer

AI-generated content may be incorrect.

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.

A screenshot of a computer

AI-generated content may be incorrect.

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.

A screenshot of a computer

AI-generated content may be incorrect.

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.