

E-Commerce Database Management

1st Jahnavi Kollu

*Dept. of Computer Science and Engg
University at Buffalo, NY
Buffalo, New York, USA
UBID: 50591633
UBIDname: jkollu
jkollu@buffalo.edu*

2nd Likhitha Kodali

*Dept. of Computer Science and Engg
University at Buffalo, NY
Buffalo, New York, USA
UBID: 50600456
UBIDname: lkodali
lkodali@buffalo.edu*

3rd Sanjay Kumar Mandru

*Dept. of Computer Science and Engg
University at Buffalo, NY
Buffalo, New York, USA
UBID: 50598178
UBIDname: smandru
smandru@buffalo.edu*

Abstract—This project focuses on designing a relational SQL database for optimizing and organizing the Brazilian E-Commerce Public Dataset by Olist, which contains over 100,000 orders from multiple Brazilian marketplaces. The dataset includes key entities such as customers, orders, products, sellers, reviews, and payments, all linked together to provide valuable insights into e-commerce business operations. By employing entity-relationship (ER) modeling and schema normalization, the database ensures efficient data retrieval and storage. Additionally, indexing strategies and optimized queries are utilized to improve query performance, while stored procedures and triggers automate data validation and enforce business rules. This system enables real-time order monitoring, customer behavior analysis, and seller performance management, thereby facilitating more effective decision-making for e-commerce platforms.

Index Terms—E- Commerce, Databases , Tables

I. NEED OF DATABASE

The E Commerce dataset with its millions of records about orders and other e-commerce elements requires a specific framework and efficient retrieval system because standard spreadsheet tools such as Excel are insufficient for this scale of data storage requirements. A relational SQL database proves to be the best system for managing this type of data based on following reasons:

1. Scalability and Performance A lack of handling capacity for large datasets makes Excel operate poorly and results in system breakdowns while using datasets with millions of rows. The indexing together with query optimization techniques in SQL databases deliver high performance to handle the efficient storage and retrieval of millions of records.

2. Data Integrity and Consistency Excel has limitations because the system depends on human error for data input and validation yet this approach endangers the system's data integrity through accidental entries of duplicates or missing or inconsistent records. A relational database makes use of primary keys together with foreign keys and constraints to enforce data integrity so users achieve both accuracy and consistency across multiple tables.

3. Handling Relationships Between Data (Normalization) A flat database structure in Excel prevents smooth maintenance of relationships between distinct data entities such

as customers, orders and products or sellers. SQL databases implement relational models to handle efficient relationships between data entities between one and many entities as well as numerous entities thereby organizing data effectively.

4. Efficient Querying and Data Retrieval Excel has a major limitation in its manual operation of data search through filtering functions because this process becomes exceptionally slow when handling large databases. The SELECT query with JOIN constructs and filtering operations in SQL databases lets users get complex data retrieval in only seconds.

5. Multi-User Access and Security The lack of multi-user design in Excel causes file systems to become vulnerable to data conflicts as well as accidental overwrites and security threats. SQL databases deliver secure data access regulation through role-based access control (RBAC) while simultaneously managing several users who work in tandem.

6. Automation with Stored Procedures and Triggers The Excel system features limited automated processing through basic macros that must be operated by users manually. Database systems through SQL databases organize procedures and triggers which perform automatic functions for order status updates and sales computations in addition to data quality checking and verification processes free of human input.

7. Real-World Business Applications The efficiency of online platform transactions, order management and payment details tracking together with customer data evaluation requires database systems that utilize relational structures. An SQL database stands above Excel for large-scale data management because it delivers scalable performance alongside security and it handles extensive datasets.

Moving data from Excel to an SQL-based database represents an essential requirement for creating an efficient and reliable system to handle e-commerce data storage.

II. BACKGROUND OF THE PROBLEM AND OBJECTIVES

Complexity of Managing E-Commerce Data: The online platform Olist connects thousands of users daily through transactions that happen among multiple sellers and customers with various products. This database presents the real-world details

essential for managing e-commerce through its Brazilian E-Commerce Public Dataset by Olist that features dynamic information about orders and payments along with product descriptions and customer data and seller performance and logistics details. Efficient operation of such vast volumes of data depends on a purpose-built system which should both organize and retrieve data smoothly with high analysis capability.

The main obstacle for e-commerce operations involves managing various integrated data structures. The entire order process includes multiple specific entities that participate in the transaction.

Customers (who place orders)
Sellers (who fulfill orders)
Products (sold across multiple categories)
Payments (with different methods and installment options)
Consumer reviews of orders serve to influence seller standings on review platforms.
Geolocation information serves both for delivery logistics and delivery tracking purposes.
Businesses experience severe difficulties with data consistency and retrieval efficiency as well as integrity challenges because they lack a relational database structure.

Challenges in Data Organization and Processing The unprocessed dataset exists in several separate CSV files which demands human assistance for extracting valuable information. The key challenges include:

Handling Many-to-Many Relationships: A customer order brings together multiple products that sellers from various sources provide. A single order can link to various installment payment methods which collectively charge a single customer. The correct linking process must establish relationships between reviews with their associated orders and customers. The complex data relationships need proper management through a structured SQL database system.

Data Redundancy and Consistency Issues: Normalizing seller details storage prevents data inconsistencies that result from maintaining seller details. Multiple duplicate records in different CSV files create issues with data integrity because they become unintegrated with other sources. Relational databases protect referential integrity through foreign key constraints as well as prevent duplicate entries from occurring.

Efficient Data Retrieval for Business Insights:

Businesses must evaluate order patterns together with customer activities and product effectiveness levels. The absence of optimized query mechanisms causes the process of retrieving dataset insights to run slowly and poorly. The ability of SQL databases to index and query data enables rapid performance during information retrieval.

Scalability and Performance Optimization:

The process of handling growing datasets becomes less efficient along with becoming harder to control through traditional storage methods. SQL handles scaling requirement by delivering three performance elements that maintain scalability along with indexing features as well as partitioning functions with query optimization capabilities.

E Commerce Business Requirements:

Business success depends heavily on providing real-time visibility into order statuses together with payment transactions and customer reviews to customers. Data accuracy stays stable thanks to automated validation solutions implemented through SQL constraints with triggers in place. A database requires optimal design to perform queries on customer purchasing data as well as refunds and seller analytics tracking functions.

Significance The efficient management of e-commerce data remains essential for three key functions within an organization: store reports and delivery timelines as well as shipment details need to be precisely monitored for order fulfillment operations.

Customer Satisfaction Retention: Ensuring quick resolution of complaints and analyzing customer feedback trends. The analysis of seller performance reveals high-performing sellers to the company to enhance both seller-buyer exchanges. Organizational decision-making and business intelligence functions enhance with quicker access to analytics about payments along with trends and sales reports alongside customer behavioral information.

III. POTENTIAL OF PROJECT

1. Enhancing Data Organization Integrity

Ensuring proper data structure and keeping information intact stands as the main obstacle when managing e-commerce data. Various dependent components including orders, customers, products, payments, sellers and reviews make up the Brazilian E-Commerce Public Dataset by Olist. A clear management system for these interconnected entities prevents duplicate data together with inconsistency and decreases efficiency.

The project addresses this problem through: Structural design of relational database systems includes enabling data integrity by implementing primary keys and foreign keys as well as normalization rules. The database design ensures elimination of data duplication by separating customer, seller and product information into different tables that stay related to each other. The system upholds referential integrity standards to stop data errors which produce unconnected orders and payment records.

The method ensures data reliability while providing high-quality information needed by any efficient e-commerce platform.

2. Enabling Faster More Efficient Data Retrieval

The system enables efficient retrieval of data through its faster operation. Online companies depend on instantaneous data access for their strategic decision-making procedures. It takes excessive time to conduct queries on large Excel and CSV datasets due to their slow data retrieval speeds.

This project achieves data retrieval efficiency through these measures: The system implements indexing methods together with query optimization tools which allow businesses to get customer orders data as well as seller performance records and payment trend analytics immediately. Having established

SQL joins together with aggregation tools enables efficient extraction of complicated data connections such as purchase records from customers or product performance standings. What database administrators accomplish is the development of stored procedures and views because these items enable quick data access for common business inquiries which shortens the time needed to generate business insights.

The improved speed of business operations results from this advancement which enables stakeholders to receive their needed information instantly.

3. Improving E-Commerce Business Insights Decision Making The integration of a structured SQL database enhances decision-making and insight development for e-commerce businesses. A properly designed SQL database foundation enables e-commerce organizations to retrieve important business intelligence leading to the following outcomes: Market order data analysis enables companies to forecast sales patterns which helps them set inventory targets appropriately. Best-selling products together with categories should be identified for enhanced marketing strategy optimization. A complete overview of performance metrics should include seller delivery times along with ratings from customers and return rate information. Customer purchasing patterns enable the generation of customized product suggestions for individual buyers and shoppers.

Organizations obtain these business insights to drive data-based choices which optimize customer satisfaction alongside better seller alliances and stronger sales productivity.

4. Higher Order and Payment Processing Speed Requires Improvement Online platforms which handle massive order and payment operations need to maintain secure precise transaction management systems. This project contributes by: The necessary arrangement of order data should establish transparent connections between customers and sellers through proper mapping of order-party-payment data elements. A tracking system for installment payments which exists commonly in Brazilian markets enables the proper association of orders. Transactions must pass triggers to guarantee that payments avoid mistakes and duplications.

This addition avoids mistakes in order completion and stops financial inconsistencies that allow customers to experience a smooth transaction pathway.

5. Optimizing Logistics Delivery Tracking

Geolocation information within the dataset contains zip codes together with latitude-longitude mappings to enable these functions: The tracking system identifies customer delivery points to help businesses optimize their shipping network design for faster delivery times. Strategic business expansion occurs through the analysis of high-demand areas which lets companies choose their growth locations based on order frequency. The delivery success rates from specific locations help business teams resolve customer service problems promptly through location-based analysis.

Companies that utilize location information enable lower transportation expenses and better productivity and superior customer satisfaction.

Why This Contribution is Crucial

The database development represents more than collection and organization of data since it redefines e-commerce data management systems and their utilization practices. The structured database will: The system design should achieve scalability features that will accommodate more orders and customers. The storage infrastructure offers secure and dependable platform needed to safeguard customer information together with financial records. Businesses need real-time data for remaining competitive because the e-commerce industry operates at high speeds. The system should automate core operations while it minimizes two key issues in data management: manual work and human errors.

IV. TARGET USER

Who Will Use the Database?

The SQL database created to handle e-commerce data acts as a vital information system for multiple participants in the online retail structure. The key users include:

1. E-Commerce Business Managers Business managers depend on this database to access information regarding sales patterns together with order processing performance level and customer behavior along with seller outcomes.

How They Use It: Users need access to reports which show the highest-selling products and most successful sellers. Foresight of order movement patterns helps business managers design stock management systems alongside marketing enterprise strategies. By using automated processes the system evaluates customer satisfaction ratings through both review data and order feedback systems. Business analysts generate reports using database queries which identify sellers with superior ratings and speediest delivery times to help the company advance its service quality.

2. Operations and Logistics Teams The company uses this platform to deliver efficient order processing and distribution services.

How They Use It: The management of pending, shipped and delivered orders allows operations teams to achieve timely order fulfillment. The delivery routes can be optimized and logistics costs can decrease through analysis of geolocation data. The company needs to locate both products with excessive returns and delivery locations that consistently generate delivery difficulties. Operational data shows the operations manager which specific sellers regularly deliver orders after deadlines so logistics teams work together to address this problem.

3. Customer Support Representatives Customer Support Representatives fulfill their duty by responding to inquiries from clients while monitoring delivery problems and dispute resolution tasks.

How They Use It: The information about payments and previous orders aids representatives during refund and exchange request resolution. Check seller response times and review history for complaint handling. To handle order-related questions support representatives need to check shipment locations and delivery forecasts. During the call the support

agent helps the customer who wants to know the status of their refund request. Through the database query the agent obtains current information about the processed refund status.

4. Sellers and Vendors Purpose: Tracking product sales, customer feedback, and payment settlements.

How They Use It: The system provides access to viewing product sales numbers together with revenue distribution information. The evaluation of customer reviews enables the enhancement of product quality alongside service operations. The system helps users track both unpaid transactions as well as successfully finalized payments. A seller consults the database to identify products generating the most returns thus enabling better product quality assessments.

5. Finance and Payment Processing Teams Purpose: Ensuring secure payment transactions and financial accuracy.

How They Use It: The system must record every payment transaction alongside payment claims and disputes. Commercial teams should evaluate the popularity of different payment options while studying patterns of installment purchases made by customers. The platform should maintain payment verification between the sellers and the network system. The financial analyst uses the system's database to retrieve information about total credit card revenue from the recent quarter.

Who Will Administer the Database?

The Database Administrators (DBAs) along with IT teams will serve as the managers who keep the database operational. The key administrators include:

1. Database Administrator (DBA) Role: Ensuring database performance, security, and integrity.

Responsibilities: A backup system together with recovery protocols safeguards the database from data destruction. User access control allows authorized personnel to modify database data. The method used to optimize queries for achieving better results with big data. Within the real-life scenario the DBA prepares user roles coupled with permissions to enable financial analysts to view payment data without enabling them to adjust order records.

2. IT Infrastructure Team The IT Infrastructure Team handles server management responsibilities for the systems that support database hosting.

Responsibilities: The IT team must accomplish maximum database accessibility and scalability measures. IT staff should measure server consumption levels to improve operational efficiency.

3. Data Analysts and Developers Staff members in this team develop application-specific reports as well as business intelligence dashboards.

Responsibilities: Applicants must develop advanced SQL syntax to perform business analytics. The front-end applications will interface with the database through API-driven connectivity.

V. REAL LIFE SCENARIO

Users operating on e-commerce platforms that utilize SQL relational databases execute their orders for purchase. Three different sellers will send the three products included in the

order which consists of a smartwatch and wireless earbuds and a laptop stand. The platform shows delivery estimates of five days as the user selects credit card payment through three installments.

The system creates and saves an order with a special order ID which connects the order to the user account after their purchase. The system connects each product to a particular seller for delivery support while storing payment details which detail the installment arrangements. The system updates the shipping table with the delivery time table information. The smartwatch seller experiences a stock issue which results in delayed shipment while other sellers handle their orders instantly.

When the scheduled delivery date arrives the user obtains only two items but they need to reach out to Customer Support. An employee of the support team accesses the SQL database system to verify order progress. The order status database indicates the smartwatch order remains in the processing phase although the other products successfully shipped to the recipient. After contacting the seller the agent learns there is goods availability but understands the seller will not fulfill the order. The agent explains to the user two possible choices namely refund or replacement of the items.

The finance staff reviews user payment records since the transaction proceeded through three installments. The purchasing system registered the initial payment for the order after which the pending second and third payments will be handled. The processing system cancels any outstanding installments through the system which prevents the user from being charged for the non-deliverable product. The advance payment sent by users gets returned to their credit card account first and a confirmation email is immediately sent.

The system automatically flags the seller for failure to fulfill the order. The seller's cancellation rate increases, affecting their marketplace ranking. The product is temporarily removed from listings due to stock issues, and the seller receives a penalty notification, impacting their reputation score. Meanwhile, the operations team analyzes seller performance trends, customer refund patterns, and top-selling products to improve overall marketplace efficiency.

This SQL-based database is essential in streamlining order tracking, automating refunds, maintaining seller accountability, and providing actionable business insights. It ensures faster dispute resolution, improved seller transparency, and enhanced user satisfaction, all of which are crucial for a growing e-commerce business.

VI. ER DIAGRAM:

[Entity Relationship Diagram:]

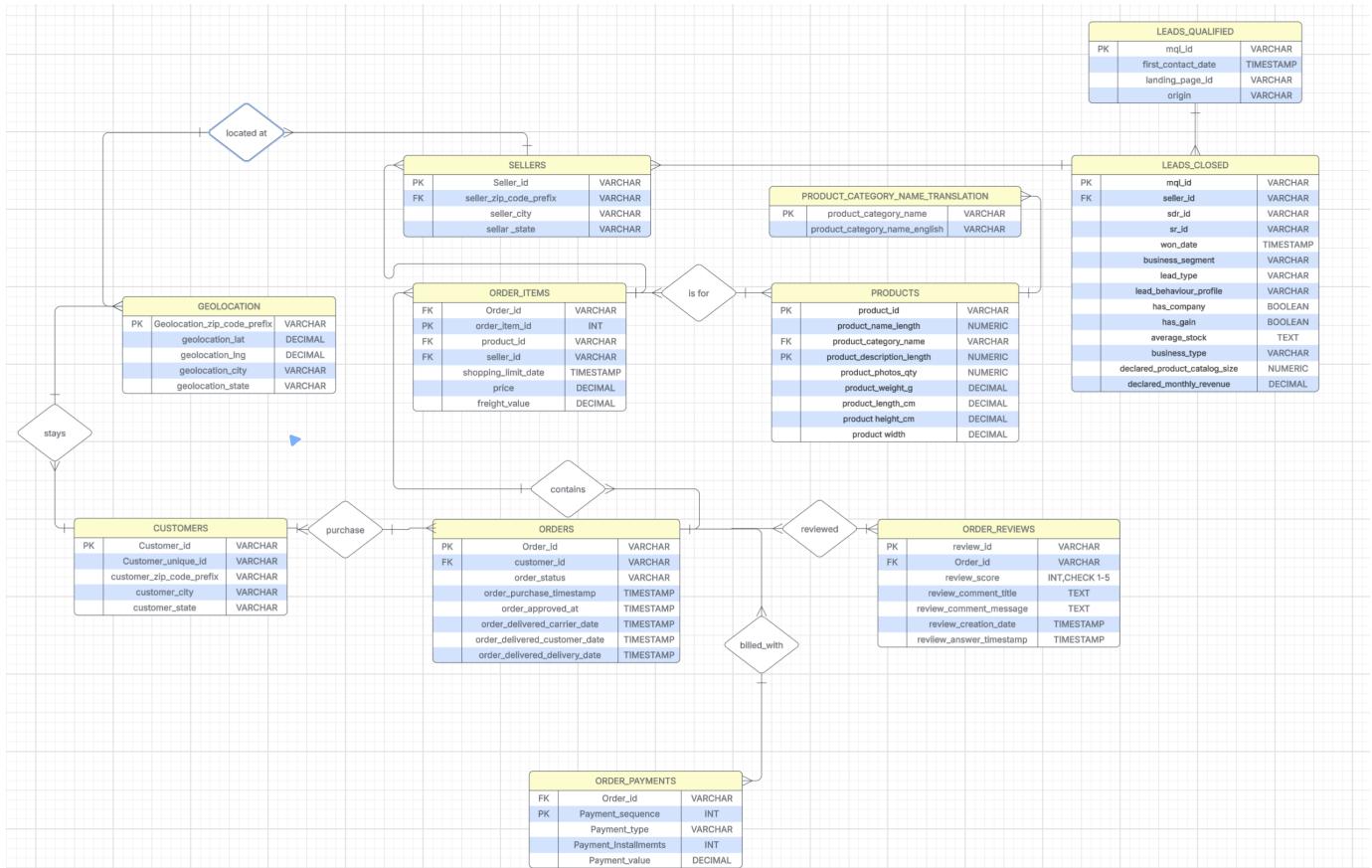


Fig. 1. ER Diagram : ER Diagram Link for more clarity visible

A. CUSTOMERS Table

1) Primary Key:

- The primary key for **CUSTOMERS** table is **customer_id** (VARCHAR).
- It will ensure that each customer is uniquely identified. The **customer_id** serves as the point of convergence between orders and other customer-related data.

2) Attributes and Descriptions:

- The **CUSTOMERS** table has the following properties:
- customer_id (VARCHAR)**: The primary identifier for this table. This column cannot be NULL. It relates customers to their payment records and orders.
- customer_unique_id (VARCHAR)**: A globally unique customer identifier made available to customers. This facilitates tracking returning customers and for analytical purposes. This field cannot be NULL.
- customer_zip_code_prefix (VARCHAR)**: The initial segment of the customer address ZIP code. Used for geolocation analysis. It cannot be NULL.

- customer_city (VARCHAR)**: City of the customer lives. This assists in territorial research and shipping facilitation. This column may be NULL.
- customer_state (VARCHAR)**: The state where the customer is available. Handy for taxation, shipping rules, and demographic data. This column may be NULL.

3) Foreign Key Relationships:

- The **CUSTOMERS** table is missing any foreign key relationships. The **ORDERS** table, however, references the **customer_id** column.
- When removing a customer, all corresponding orders in the **ORDERS** table are removed since due to the CASCADE action, preventing orphaned orders in the database.
- All of the customer's essential information is stored in the **CUSTOMERS** table.
- The database relates customers through **customer_id**, thus making it a useful component in customer relationships. The company utilizes customer orders to perform

behavioural analysis and market research based on its database technology.

B. SELLERS Table

1) Primary Key:

- The primary key of the **SELLERS** table is **seller_id** (VARCHAR).
- It uniquely identifies every seller within the system, ensuring that all seller information occur just once. The **seller_id** occurs in more than one database table, primarily linking sellers to the **ORDER ITEMS** and **LEADS CLOSED** tables to record transactions.

2) Attributes and Descriptions:

- The **SELLERS** table possesses the following characteristics:
 - seller_id** (VARCHAR): An identifier assigned for every seller. It is an area for observing seller activity, processing orders, and performance measurement. This field cannot be NULL.
 - seller_zip_code_prefix** (VARCHAR): Shows the zip seller's location code prefix. This field assists in identifying each seller's location geographically for purposes of logistics and delivery estimates. NULL cannot be set.
 - seller_city** (VARCHAR): Contains the name of the city where the vendor is located. It assists to examine the dissemination trend of sellers across different geographical areas. This field can be NULL.
 - seller_state** (VARCHAR): Holds the state where the seller runs business. It is beneficial to tax law, business policies, and regional analysis. This column may be NULL.

3) Foreign Key Relationships:

- The **SELLERS** table has three foreign keys:
 - seller_zip_code_prefix** field is linked to the **GEOLOCATION** table to which every seller can be linked to a geographical region.
 - ORDER ITEMS** table utilizes the **seller_id** field to establish connections between suppliers and their respective electronic transactions.
 - The **LEADS CLOSED** table cross-references the **seller_id** column.
- Cascade Deletion Rules:** When a seller is deleted from the database:
 - All corresponding records in the **ORDER ITEMS** table will be deleted due to the CASCADE action.
 - All corresponding records in the **LEADS CLOSED** table will be eliminated as well due to the CASCADE action.

This framework ensures that no orphaned records remain in the system when a seller is de-listed. The **SELLERS** table is required to track marketplace vendors, handle seller logistics, and offer smooth order processing.

C. PRODUCTS Table

1) Primary Key:

- The primary key for the **PRODUCTS** table is **product_id** (VARCHAR). This ensures that each product has a unique system identifier. The **product_id** field is referenced by the **ORDER ITEMS** table to link products with orders.

2) Attributes and Descriptions:

- The **PRODUCTS** table contains the following attributes:
 - product_id** (VARCHAR): The primary key for the table, uniquely identifying each product. This field is used to track product specifications, customer orders, and associated categories. This field cannot be NULL.
 - product_name_length** (NUMERIC): Represents the character count of the product name. It is used for text analysis and product structuring in the catalog. This field cannot be NULL.
 - product_category_name** (VARCHAR): Links the product to a specific category. This field references the **PRODUCT CATEGORY NAME TRANSLATION** table. This field cannot be NULL.
 - product_description_length** (NUMERIC): Stores the length of the product description in characters. It is used to assess product information completeness. This field cannot be NULL.
 - product_photo_count** (NUMERIC): Represents the number of photos associated with a product. It helps determine product display in the marketplace. This field cannot be NULL.
 - product_weight_g** (DECIMAL): Represents the weight of the product in grams. It is useful for shipping calculations and logistics. This field cannot be NULL.
 - product_length_cm** (DECIMAL): Stores the length of the product in centimeters. It helps with packaging and warehouse storage management. This field cannot be NULL.
 - product_height_cm** (DECIMAL): Represents the height of the product in centimeters. It is essential for volume-based shipping cost estimation. This field cannot be NULL.
 - product_width_cm** (DECIMAL): Stores the width of the product in centimeters. This attribute aids in packaging optimization and storage. This field cannot be NULL.

3) Foreign Key Relations:

- The **PRODUCTS** table contains three foreign keys:
 - The **product_category_name** field links to the **PRODUCT CATEGORY NAME TRANSLATION** table, ensuring that each product is assigned a valid category.
 - The **ORDER ITEMS** table references the **product_id** field to associate products with orders.
- Cascade Deletion Rules:** If a product is deleted from the database:
 - All corresponding records in the **ORDER ITEMS** table will be removed due to the CASCADE action.

This mechanism ensures data integrity by preventing orphaned records. The **PRODUCTS** table is crucial for e-commerce operations, as it maintains inventory management,

links inventory items to product categories, and supports order tracking functionalities.

D. ORDERS Table

1) Primary Key:

- The primary key of the **ORDERS** table is **order_id** (VARCHAR). This identifies each customer order uniquely. The **order_id** is a multi-table reference key. **ORDER ITEMS**, **ORDER PAYMENTS**, and **ORDER REVIEWS** tables are dependent on the **order_id** primary key contained within the **ORDERS** table. Each order maintains its relationship through referencing links to corresponding records in these tables.

2) Attributes and Descriptions:

- The **ORDERS** table contains the following attributes:
- order_id (VARCHAR)**: The primary key of the table, which uniquely identifies each order. The database refers to every action an order has an impact on using this field. This field is not NULL.
- customer_id (VARCHAR)**: Refers to the primary key in the **CUSTOMERS** table. Each order points to a recurring customer. This field is not NULL.
- order_status (VARCHAR)**: The current status of an order. Possible values are "pending," "shipped," "delivered," or "canceled." Cannot be NULL and has a default value of "pending."
- order_purchase_timestamp (TIMESTAMP)**: Stores the order purchase timestamp. This field is utilized for order tracking and analysis. An automatic default value of the current timestamp is given.
- order_approved_at (TIMESTAMP)**: Stores timestamp representing order payment approval. If NULL, the order remains in a pending approval status.
- order_delivered_carrier_date (TIMESTAMP)**: Specifies the exact date when the order is handed over to the shipping carrier. If NULL, the order is not yet shipped.
- order_delivered_customer_date (TIMESTAMP)**: Specifies the date the order is actually delivered to the customer. If NULL, the order has not yet been delivered.
- order_estimated_delivery_date (TIMESTAMP)**: Records the order's estimated delivery date. If NULL, no estimated delivery date exists.

3) Foreign Key Constraints:

- ORDERS** has a single foreign key:
- customer_id**: References the **CUSTOMERS** table to point to enrolled customers.
- ORDER ITEMS**, **ORDER PAYMENTS**, and **ORDER REVIEWS** **order_id** field refers to this table. Deletion of an order also removes all corresponding payment records, items, and review documents due to the CASCADE action. It ensures consistency across transactions and prevents orphan data records.

The **ORDERS** table maintains all customer purchase information, which is essential to business operations. It records customer orders with payment and delivery information. This table stores e-commerce transaction history in the database.

E. ORDER ITEMS Table

1) Primary Key:

- The primary key for the **ORDER ITEMS** table is **order_item_id** (INT). This allows each item within an order to be uniquely distinguishable when multiple items exist in one order. The **order_item_id** is auto-incremented for uniqueness.

2) Attributes and Descriptions:

- The **ORDER ITEMS** table contains the following attributes:
- order_id (VARCHAR)**: This attribute is a reference to the **ORDERS** table so that every order item has a valid order. This field cannot be NULL.
- order_item_id (INT)**: This is a unique identifier for every item of an order. It distinguishes multiple items in the same order. This column should not be NULL.
- product_id (VARCHAR)**: This column references the **PRODUCTS** table, linking each order item to a valid product. This column should not be NULL.
- seller_id (VARCHAR)**: This column references the **SELLERS** table, linking each order item to a valid seller. This column should not be NULL.
- shopping_limit_date (TIMESTAMP)**: This column stores the shipping time limit for the product. It is used for order item tracking. In the event NULL, no shipping due date is entered.
- price (DECIMAL)**: The selling price of the product in this particular order item. This helps in computing revenue and customers' payments. It has a default value of 0.00.
- freight_value (DECIMAL)**: This is the cost of shipping of the order item. This contributes to the computation of total cost for the customer. Its default value is 0.00.

3) Foreign Key Actions:

- The **ORDER ITEMS** table contains foreign keys:
- The **order_id** field references the **ORDERS** table, so that each item belongs to a valid order.
- The **product_id** field references the **PRODUCTS** table, linking each item to a valid product.
- The **seller_id** field references the **SELLERS** table, associating items with the seller who will ship them.
- Since **order_id**, **product_id**, and **seller_id** are referenced:
 - If an order is deleted, the related order items will be deleted by the **CASCADE** action as well.
 - If a product is deleted, related order items will also be deleted in order not to have orphaned records.
 - If a seller is deleted, all referenced order items will also be deleted.

The **ORDER ITEMS** table plays a very significant role in tracking particular products per order, assigning them to sellers, and allocating correct pricing and freight costs.

F. ORDER PAYMENTS Table

1) Primary Key:

- The primary key for the **ORDER PAYMENTS** table is **payment_sequence** (INT). This uniquely identifies each payment transaction associated with an order. Since orders may involve multiple payments (installments), each payment within the same order is assigned a unique sequence number.

2) *Attributes and Descriptions:*

- The **ORDER PAYMENTS** table contains the following attributes:
- order_id** (VARCHAR): This attribute references the **ORDERS** table, ensuring that every payment belongs to a valid order. This field cannot be NULL.
- payment_sequence** (INT): This is a unique identifier for each payment installment related to an order. It ensures proper tracking of multiple payments per order. This field cannot be NULL.
- payment_type** (VARCHAR): This field specifies the method of payment, such as “credit card,” “debit card,” “boleto,” or “voucher.” This field cannot be NULL and defaults to “unknown.”
- payment_installments** (INT): This attribute indicates the number of installments used for the payment. If NULL, it means the payment was made in full. The default value is 1.
- payment_value** (DECIMAL): This field stores the total amount of the payment installment. It helps track financial transactions and revenue. The default value is 0.00.

3) *Foreign Key Actions:*

- The **ORDER PAYMENTS** table contains one foreign key:
- The **order_id** attribute references the **ORDERS** table, ensuring that each payment is linked to a valid order.

Since **order_id** is referenced in **ORDER PAYMENTS**:

- If an order is deleted, all corresponding payment records will also be deleted due to the CASCADE action.
- This ensures data integrity by preventing orphaned payment records.

The **ORDER PAYMENTS** table is crucial for tracking payment transactions, handling installment-based payments, and ensuring proper financial records for each order.

G. ORDER REVIEWS Table

1) *Primary Key:*

- Primary key for the **ORDER REVIEWS** table is **review_id** (VARCHAR). This specifically sets all the reviews that a customer has written. It guarantees that there are no repeated reviews and enables tracking of individual reviews for examination and customer comment.

2) *Attributes and Descriptions:*

- The **ORDER REVIEWS** table attributes are as follows:
- review_id** (VARCHAR): It is the primary key for each customer review. It gives accurate monitoring of feedback for orders. This column cannot be NULL.

- order_id** (VARCHAR): This column is the **ORDERS** table reference so that each review is linked with a valid order. This field must not be NULL.
- review_score** (INT, CHECK 1–5): This column represents the rating provided by the customer out of 1 to 5. It helps in assessing customer satisfaction. This field cannot be NULL.
- review_comment_title** (TEXT): This property contains a short title that summarizes the review. It provides a brief insight into the customer’s feedback. This field may be NULL.
- review_comment_message** (TEXT): This field has the entire review message provided by the customer. It assists in assessing in-depth customer opinions. This column may be NULL.
- review_creation_date** (TIMESTAMP): This property contains the date and time of posting of the review. It helps to understand trends in feedback over time. The default is the present timestamp.
- review_answer_timestamp** (TIMESTAMP): This attribute documents the date and time when a response was allocated to the review. If NULL, then the review has yet to be answered.

3) *Foreign Key Actions:*

- The **ORDER REVIEWS** table has a foreign key:
- The **order_id** attribute references the **ORDERS** table to ensure reviews are only created for real orders.

Since **order_id** is referenced in **ORDER REVIEWS**:

- When a user order is cancelled, all its associated reviews are also deleted by the CASCADE action.
- This avoids reviews from being orphaned.

The **ORDER REVIEWS** table is required for noting customer feedback, service quality, and improved user experience based on sentiment analysis and satisfaction ratings.

H. PRODUCT CATEGORY NAME TRANSLATION Table

1) *Primary Key:*

- The primary key for the **PRODUCT CATEGORY NAME TRANSLATION** table is **product_category_name** (VARCHAR). This ensures that each product category name is uniquely identified and mapped to its English equivalent. Since product categories may be stored in different languages, this table provides a reference for standardized names.

2) *Attributes and Descriptions:*

- The **PRODUCT CATEGORY NAME TRANSLATION** table contains the following attributes:
- product_category_name** (VARCHAR): This is the unique identifier for each product category. It represents the category name used in the original dataset, which may be in a different language. This field cannot be NULL.
- product_category_name_english** (VARCHAR): This attribute provides the English translation of the product category name. It helps in standardizing category names for analysis and reporting. This field cannot be NULL.

3) *Foreign Key Actions:*

- The **PRODUCT CATEGORY NAME TRANSLATION** table does not contain any foreign keys, as it serves as a reference table for category names.
- The **PRODUCT CATEGORY NAME TRANSLATION** table is essential for maintaining a consistent naming convention across different languages, enabling better data analysis, reporting, and category-based filtering.

I. *GEOLOCATION Table*

1) *Primary Key:*

- The primary key for the **GEOLOCATION** table is **geolocation_zip_code_prefix** (VARCHAR). This ensures that each geographical location entry is uniquely identified based on the zip code prefix. The zip code prefix serves as a regional identifier for mapping locations.

2) *Attributes and Descriptions:*

- The **GEOLOCATION** table contains the following attributes:
 - **geolocation_zip_code_prefix** (VARCHAR): This is the unique identifier for a geographic location. It represents the zip code prefix associated with a particular region. This field cannot be NULL.
 - **geolocation_lat** (DECIMAL(9,6)): This attribute stores the latitude coordinate of the location. It allows for accurate geospatial mapping and distance calculations. This field cannot be NULL.
 - **geolocation_lng** (DECIMAL(9,6)): This attribute stores the longitude coordinate of the location. It is used in conjunction with latitude for geographical positioning. This field cannot be NULL.
 - **geolocation_city** (VARCHAR): This field stores the name of the city corresponding to the zip code prefix. It helps in regional analysis and logistics. This field can be NULL.
 - **geolocation_state** (VARCHAR): This attribute represents the state where the location is situated. It is useful for shipping logistics, taxation, and regional analytics. This field can be NULL.

3) *Foreign Key Actions:*

- The **GEOLOCATION** table does not contain any foreign keys. However, it serves as a reference for other tables such as **CUSTOMERS** and **SELLERS** to link zip codes with location details.
- The **GEOLOCATION** table is crucial for mapping customer and seller locations, enabling logistics management, delivery tracking, and distance-based optimizations for e-commerce operations.

J. *LEADS QUALIFIED Table*

- 1) *Primary Key:* The primary key for the **LEADS QUALIFIED** table is **mql_id** (VARCHAR). This uniquely identifies each marketing qualified lead (MQL). The primary key ensures that no duplicate leads exist and allows proper tracking of lead conversions.

2) *Attributes and Descriptions:*

- The **LEADS QUALIFIED** table contains the following attributes:
 - **mql_id** (VARCHAR): This is the unique identifier for a marketing qualified lead. It allows tracking of potential customers who have shown interest in products or services. This field cannot be NULL.
 - **first_contact_date** (TIMESTAMP): This attribute stores the date and time when the lead first interacted with the platform. It is useful for analyzing the lead conversion timeline. The default value is the current timestamp.
 - **landing_page_id** (VARCHAR): This field identifies the landing page that the lead visited before conversion. It helps track the effectiveness of marketing campaigns. This field can be NULL.
 - **origin** (VARCHAR): This attribute represents the source through which the lead was acquired, such as social media, email campaigns, or advertisements. It is useful for marketing analysis. This field can be NULL.

3) *Foreign Key Actions:*

- The **LEADS QUALIFIED** table does not contain any foreign keys. However, it serves as a reference for analyzing customer acquisition strategies and marketing funnel efficiency.
- The **LEADS QUALIFIED** table is crucial for monitoring lead conversions, optimizing marketing campaigns, and understanding customer acquisition trends.

K. *LEADS CLOSED Table*

1) *Primary Key:*

- The primary key for the **LEADS CLOSED** table is **mql_id** (VARCHAR). This clearly qualifies each closed marketing qualified lead (MQL). It prevents the duplication of leads and enables tracking of leads converted successfully.

2) *Attributes and Descriptions:*

- The **LEADS CLOSED** table has the following characteristics:
 - **mql_id** (VARCHAR): It is the primary key for a marketing qualified lead that has closed. It makes sure precise monitoring of lead conversion. This is not NULL.
 - **seller_id** (VARCHAR): This is the **SELLERS** table reference, linking every closed lead to a specific seller. It must not be NULL.
 - **sdr_id** (VARCHAR): This field defines the sales development representative (SDR) responsible for engaging with the lead. This column may be NULL.
 - **sr_id** (VARCHAR): The column contains the senior representative (SR) who oversaw the process of lead conversion. This column may be NULL.
 - **won_date** (TIMESTAMP): The date is stored in this field when the lead was successfully converted into a customer. It helps to measure lead conversion efficiency. This field can be NULL.

- **business_segment (VARCHAR)**: This field specifies the industry of business related to the lead, i.e., retail or technology. This column may be NULL.
- **lead_type (VARCHAR)**: This column defines the type of lead, i.e., inbound or outbound. This field can be NULL.
- **lead_behaviour_profile (VARCHAR)**: This field provides an indication of the lead's profile of behavior based on interactions. This column is nullable.
- **has_company (BOOLEAN, DEFAULT FALSE)**: This attribute indicates whether the lead is associated with a company. The default is FALSE.
- **has_gain (BOOLEAN, DEFAULT FALSE)**: This field suggests whether the lead is prospective to support revenue growth. The default is FALSE.
- **average_stock (VARCHAR)**: The column holds information regarding the lead's average product inventory. This field can be NULL.
- **business_type (VARCHAR)**: This column is the nature of business the lead is engaged in, i.e., B2B or B2C. This column may be NULL.
- **declared_product_catalog_size (DECIMAL(10,2))**: This property captures the approximate size of the product catalog kept by the lead. It gives details on business potential. This column may be NULL.
- **Monthly_declared_revenue (DECIMAL(10,2))**: This field is the rough monthly income reported by the lead. It aids in business forecasting. This can be NULL.

3) *Foreign Key Actions:*

- The **LEADS CLOSED** table has a foreign key:
- **seller_id** attribute is a reference to the **SELLERS** table, which guarantees that all closed leads are linked to a valid seller.

Since **seller_id** is highlighted in **LEADS CLOSED**:

- When a seller is removed, all associated closed leads will also be deleted due to the ON DELETE CASCADE action.
- This ensures data integrity by preventing orphaned lead records.

LEADS CLOSED table is employed to note successfully converted leads, monitoring seller performance, and researching business expansion patterns.

L. *Cardinality of the Relations in given ER diagram*

CUSTOMERS → ORDERS (One-to-Many) [1:N] Customers have the ability to create numerous orders through the system. One customer can possess a single order. Each customer creates numerous orders which always relate back to an individual customer.

ORDERS → ORDER_ITEMS (One-to-Many) [1:N] The same order contains an unlimited number of order items. All order items exclusively link to a single order in the system. The cardinality shows that one order contains multiple items even though every order item connects with a single order.

PRODUCTS → ORDER_ITEMS (One-to-Many) [1:N] A single product can exist in several order items when placed

for sale. There is just one product related to each order item. A product appears in multiple order items while each order item contains one and only one product.

SELLERS → ORDER_ITEMS (One-to-Many) [1:N]

Multiple products can be sold through each seller and thus appear in different order items. All order items directly link with one individual seller. Each seller maintains at least one order item though every order item relates to a single seller.

ORDERS → ORDER_PAYMENTS (One-to-Many) [1:N]

An order contains several payment installments depending on the number of payments involved. The payment installment extends only to a single order. Each order contains many payments although each payment belongs to a single order.

ORDERS → ORDER_REVIEWS (One-to-One) [1:1]

The review section of an order contains maximum one entry. One order contains only a single active review. Each order only receives one review since a single review links to exactly one order.

GEOLOCATION → CUSTOMERS (One-to-Many) [1:N]

One geolocation zip code prefix may link up with an unlimited number of customers. Customers have association with a single geolocation prefix throughout the database system. The relationship operates under cardinality rules which state that one single zip code matches several customers while every individual client belongs to only one such zone.

GEOLOCATION → SELLERS (One-to-Many) [1:N]

One geolocation zip code prefix establishes associations with various sellers in the system. A seller connects to only one single geographic area prefix. Each zip code serves as a home for various sellers yet all sellers match exactly one unique geographical code.

PRODUCT_CATEGORY_NAME_TRANSLATION → PRODUCTS (One-to-Many) [1:N] A translated product category name maintains relationships with several products. Every product receives its classification from just one category name. A single category name connects to different products though every product participates in just one category.

LEADS_QUALIFIED → LEADS_CLOSED (One-to-One) [1:1] Marketers consider marketing qualified leads as closeable once. The precise relationship exists between each single closed lead with a single unique MQL. An MQL qualifies as one complete lead that transforms into a closed deal.

SELLERS → LEADS_CLOSED (One-to-Many) [1:N]

A single seller can work on different closed leads despite their association with multiple sellers. Each final lead can only be connected to one specific seller. Each lead requires single closure through a particular seller even though one seller can handle various leads during the process.

VII. TASK 3: LOAD THE LARGE DATASET

In data they are some Nan values and redundancies we removed them and loaded them into SQL using python script because the data is large so we used script to insert the data into tables

below figure is python script :

```
 1 #!/usr/bin/python
 2 
 3 # Import project
 4 from __future__ import absolute_import
 5 import sys
 6 
 7 # Set environment
 8 os.environ['DJANGO_SETTINGS_MODULE'] = 'data_django.settings'
 9 os.environ['DJANGO_CONFIGURATION'] = 'Production'
10 
11 # Import Django
12 from django.core.wsgi import get_wsgi_application
13 application = get_wsgi_application()
14 
15 # Import models
16 from data_django.models import *
17 
18 # Set connection
19 db_name = "testdb"
20 db_password = "test"
21 db_username = "root"
22 db_host = "127.0.0.1"
23 db_port = "3306"
24 
25 # Set table
26 table_name = "category"
27 column_names = ["category_id", "category_name"]
28 timestamp_columns = []
29 
30 # Set file
31 file_name = "category.csv"
32 file_path = os.path.join(os.path.dirname(__file__), file_name)
33 
34 # Set connection
35 connection = MySQLdb.connect(host=db_host, user=db_username, passwd=db_password, db=db_name, port=db_port)
36 cursor = connection.cursor()
37 
38 # Set file
39 file = open(file_path, 'r')
40 file_content = file.read()
41 file.close()
42 
43 # Set headers
44 headers = file_content.split("\n")[0].split(",")
45 
46 # Set data
47 data_lines = file_content.split("\n")[1:]
48 data = []
49 for line in data_lines:
50     row = {}
51     for index, header in enumerate(headers):
52         row[header] = line.split(",")[index]
53     data.append(row)
54 
55 # Create table
56 cursor.execute("CREATE TABLE IF NOT EXISTS %s (%s)" % (table_name, ", ".join(column_names)))
57 
58 # Insert data
59 for data_line in data:
60     values = []
61     for column in column_names:
62         if column in data_line:
63             values.append(data_line[column])
64         else:
65             values.append(None)
66     cursor.execute("INSERT INTO %s (%s) VALUES (%s)" % (table_name, ", ".join(column_names), ", ".join(values)))
67 
68 # Commit transaction
69 connection.commit()
70 
71 # Close connection
72 connection.close()
73 
74 print("Successfully loaded %d records into %s(%s)" % (len(data), table_name))
75 
76 except Exception as e:
77     print("Error: %s" % str(e))
78 
79 # End of file
```

Fig. 2. Python script

Creating Tables

we have created the tables using primary key constraint

```

1 -- Drop existing tables
2 DROP TABLE IF EXISTS `Order`;
3 DROP TABLE IF EXISTS `Products`;
4 DROP TABLE IF EXISTS `Order_Placement`;
5 DROP TABLE IF EXISTS `Order_Details`;
6 DROP TABLE IF EXISTS `Orders`;
7 DROP TABLE IF EXISTS `Customers`;
8 DROP TABLE IF EXISTS `Product_Categories`;
9 DROP TABLE IF EXISTS `Translations`;
10 DROP TABLE IF EXISTS `Sellers`;
11 DROP TABLE IF EXISTS `Execution`;
12 DROP TABLE IF EXISTS `Lead_Quotations`;
13 DROP TABLE IF EXISTS `Lead_Closed CASCADE`;
14
15 -- Create Tables
16
17 ✓ CREATE TABLE Customers (
18     customer_id VARCHAR(50) PRIMARY KEY,
19     customer_name VARCHAR(100),
20     customer_email VARCHAR(50) UNIQUE NOT NULL,
21     customer_phone VARCHAR(10),
22     customer_street VARCHAR(50),
23     customer_zipcode VARCHAR(10)
24 );
25
26 ✓ Creating Orders Table
27
28 ✓ CREATE TABLE Orders (
29     order_id INT PRIMARY KEY,
30     customer_id VARCHAR(50),
31     order_purchase_timestamp TIMESTAMP NOT NULL,
32     order_approved_at TIMESTAMP,
33     order_delivered_customer_date TIMESTAMP,
34     order_delivered_driver_date TIMESTAMP,
35     order_estimated_delivery_date TIMESTAMP
36 );
37
38 ✓ Creating Products Table
39
40 ✓ CREATE TABLE Products (
41     product_id INT PRIMARY KEY,
42     product_category_name VARCHAR(100),
43     product_name VARCHAR(100) PRIMARY KEY,
44     product_description TEXT,
45     product_length DECIMAL(10,2),
46     product_width DECIMAL(10,2),
47     product_height DECIMAL(10,2),
48     product_weight DECIMAL(10,2),
49     product_creation_date DATE
50 );
51
52 ✓ Creating Order_Placement Table
53
54 ✓ CREATE TABLE Order_Placement (
55     order_id INT,
56     product_id INT,
57     quantity INT,
58     unit_price DECIMAL(10,2),
59     FOREIGN KEY (order_id) REFERENCES Orders(order_id),
60     FOREIGN KEY (product_id) REFERENCES Products(product_id)
61 );
62
63 ✓ Creating Order_Details Table
64
65 ✓ CREATE TABLE Order_Details (
66     order_id INT,
67     product_id INT,
68     quantity INT,
69     unit_price DECIMAL(10,2),
70     FOREIGN KEY (order_id) REFERENCES Orders(order_id),
71     FOREIGN KEY (product_id) REFERENCES Products(product_id)
72 );
73

```

Fig. 3. Create Tables 1

```

178 -- Creating Geolocation Table
179 CREATE TABLE Geolocation (
180     geolocation_zip_code_prefix VARCHAR(10) PRIMARY KEY,
181     geolocation_lat DECIMAL(10,6),
182     geolocation_lng DECIMAL(10,6),
183     geolocation_city VARCHAR(100),
184     geolocation_state VARCHAR(50)
185 );
186
187 -- Creating Sellers Table
188 CREATE TABLE Sellers (
189     seller_id VARCHAR(50) PRIMARY KEY,
190     seller_zip_code_prefix VARCHAR(10),
191     seller_city VARCHAR(100),
192     seller_state VARCHAR(50)
193 );
194
195 -- Creating Order Items Table
196 CREATE TABLE Order_Items (
197     order_id VARCHAR(50),
198     order_item_id INT,
199     product_id VARCHAR(50),
200     seller_id VARCHAR(50),
201     shipping_address_lat DECIMAL(10,6),
202     shipping_address_lng DECIMAL(10,6) NOT NULL,
203     freight_value DECIMAL(10,2) NOT NULL,
204     PRIMARY KEY (order_id, order_item_id)
205 );
206
207 -- Creating Order Payment Tables
208 CREATE TABLE Order_Payments (
209     order_id VARCHAR(50),
210     payment_sequential INT,
211     payment_type VARCHAR(50),
212     payment_installments INT CHECK (payment_installments >= 0),
213     payment_value DECIMAL(10,2) NOT NULL,
214     PRIMARY KEY (order_id, payment_sequential)
215 );
216

```

Fig. 4. Create Tables 2

Alter Tables

we added foreign key constraint using alter tables commands:

```

218 -- Creating Order_Reviews Table
219 CREATE TABLE `order_reviews` (
220   `id` VARCHAR(50) PRIMARY KEY,
221   `order_id` VARCHAR(50) UNIQUE,
222   `review_score` INT CHECK(`review_score` BETWEEN 1 AND 5),
223   `review_content_title` TEXT,
224   `review_comment_message` TEXT,
225   `review_creation_date` TIMESTAMP,
226   `review_answerer_timestamp` TIMESTAMP
227 );
228
229 -- Creating Leads_Qualified Table
230 CREATE TABLE `leads_qualified` (
231   `mql_id` VARCHAR(50) PRIMARY KEY,
232   `first_contact_date` TIMESTAMP,
233   `landing_page_id` VARCHAR(50),
234   `origin` VARCHAR(50)
235 );
236
237 -- Creating Leads_Closed Table
238 CREATE TABLE `leads_closed` (
239   `mql_id` VARCHAR(50),
240   `seller_id` VARCHAR(50),
241   `sdr_id` VARCHAR(50),
242   `sr_id` VARCHAR(50),
243   `won_date` TIMESTAMP,
244   `businnes_name` VARCHAR(100),
245   `lead_type` VARCHAR(50),
246   `lead_behaviour` profile VARCHAR(50),
247   `has_company` BOOLEAN,
248   `has_gtin` BOOLEAN,
249   `average_stock` TEXT,
250   `base_product_catalog_size` INTEGER(50),
251   `declared_product_catalog_size` NUMERIC(20,2),
252   `declared_monthly_revenue` DECIMAL(12,2),
253   PRIMARY KEY (mql_id, seller_id)
254 );
255
256

```

Fig. 5. Create Tables 3

```
295 <-- ALTER TABLE Orders
296 ADD CONSTRAINT fk_orders_customer FOREIGN KEY (customer_id)
297 REFERENCES Customers(customer_id) ON DELETE CASCADE;
298
299 <-- ALTER TABLE Products
300 ADD CONSTRAINT fk_products_category FOREIGN KEY (product_category_name)
301 REFERENCES Product_Category_Name_Translation(product_category_name);
302
303 <-- ALTER TABLE Sellers
304 ADD CONSTRAINT fk_sellers_geolocation FOREIGN KEY (seller_zip_code_prefix)
305 REFERENCES Geolocation(geolocation_zip_code_prefix);
306
307 <-- ALTER TABLE Order_Items
308 ADD CONSTRAINT fk_order_items_order FOREIGN KEY (order_id)
309 REFERENCES Orders(order_id) ON DELETE CASCADE;
310
311 <-- ALTER TABLE Order_Items
312 ADD CONSTRAINT fk_order_items_product FOREIGN KEY (product_id)
313 REFERENCES Products(product_id) ON DELETE CASCADE;
314
315 <-- ALTER TABLE Order_Items
316 ADD CONSTRAINT fk_order_items_seller FOREIGN KEY (seller_id)
317 REFERENCES Sellers(seller_id) ON DELETE CASCADE;
318
319 <-- ALTER TABLE Order_Payments
320 ADD CONSTRAINT fk_order_payments_order FOREIGN KEY (order_id)
321 REFERENCES Orders(order_id) ON DELETE CASCADE;
322
323 <-- ALTER TABLE Order_Reviews
324 ADD CONSTRAINT fk_order_reviews_order FOREIGN KEY (order_id)
325 REFERENCES Orders(order_id) ON DELETE CASCADE;
326
327 <-- ALTER TABLE Leads_Closed
328 ADD CONSTRAINT fk_leads_closed_mql FOREIGN KEY (mql_id)
329 REFERENCES Leads_Qualified(mql_id) ON DELETE CASCADE;
330
331 <-- ALTER TABLE Leads_Closed
332 ADD CONSTRAINT fk_leads_closed_seller FOREIGN KEY (seller_id)
333 REFERENCES Sellers(seller_id) ON DELETE CASCADE;
```

Fig. 6. Alter Table 1

Now we are verifying whether the tables created and the data is loaded properly

```

1 + select * from customers
2
3 select * from orders
4
5 select * from geolocation
6
7 select * from leads_closed
8
9 select * from leads_qualified
10
11 select * from order_items
12
13 select * from order_payments
14
15 select * from order_reviews
16
17 select * from products
18
19 select * from sellers
20
21 select * from product_category_name_translation
22
23 SELECT customer_id, COUNT(*) AS duplicate_count
24 FROM customers
25
Data Output: Messages Notifications


```

Fig. 7. Verifying Table 1

```

3 select * from orders
4 select * from geolocation
5 select * from leads_closed
6 select * from leads_qualified
7 select * from order_items
8 select * from order_payments
9 select * from order_reviews
10 select * from products
11 select * from sellers
12 select * from product_category_name_translation
13
14
15
16
17
18
19
20
21
22 SELECT customer_unique_id, COUNT(*) AS duplicate_count
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
689
690
691
692
693
694
695
696
697
698
699
699
700
701
702
703
704
705
706
707
708
709
709
710
711
712
713
714
715
716
717
718
719
719
720
721
722
723
724
725
726
727
728
729
729
730
731
732
733
734
735
736
737
738
739
739
740
741
742
743
744
745
746
747
748
749
749
750
751
752
753
754
755
756
757
758
759
759
760
761
762
763
764
765
766
767
768
769
769
770
771
772
773
774
775
776
777
778
779
779
780
781
782
783
784
785
786
787
788
789
789
790
791
792
793
794
795
796
797
798
799
799
800
801
802
803
804
805
806
807
808
809
809
810
811
812
813
814
815
816
817
818
819
819
820
821
822
823
824
825
826
827
828
829
829
830
831
832
833
834
835
836
837
838
839
839
840
841
842
843
844
845
846
847
848
849
849
850
851
852
853
854
855
856
857
858
859
859
860
861
862
863
864
865
866
867
868
869
869
870
871
872
873
874
875
876
877
878
879
879
880
881
882
883
884
885
886
887
888
889
889
890
891
892
893
894
895
896
897
898
899
899
900
901
902
903
904
905
906
907
908
909
909
910
911
912
913
914
915
916
917
918
919
919
920
921
922
923
924
925
926
927
928
929
929
930
931
932
933
934
935
936
937
938
939
939
940
941
942
943
944
945
946
947
948
949
949
950
951
952
953
954
955
956
957
958
959
959
960
961
962
963
964
965
966
967
968
969
969
970
971
972
973
974
975
976
977
978
979
979
980
981
982
983
984
985
986
987
988
989
989
990
991
992
993
994
995
996
997
998
999
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1089
1090
1091
1092
1093
1094
1095
1096
1097
1097
1098
1099
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1189
1190
1191
1192
1193
1194
1195
1195
1196
1197
1198
1198
1199
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1289
1290
1291
1292
1293
1294
1295
1296
1297
1297
1298
1299
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1389
1390
1391
1392
1393
1394
1395
1396
1396
1397
1398
1398
1399
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1409
1410
1411
1412
1413
1414
1415
1416
1416
1417
1418
1419
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1489
1490
1491
1492
1493
1494
1495
1496
1496
1497
1498
1498
1499
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1509
1510
1511
1512
1513
1514
1515
1516
1516
1517
1518
1519
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1589
1590
1591
1592
1593
1594
1595
1596
1596
1597
1598
1598
1599
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1609
1610
1611
1612
1613
1614
1615
1616
1616
1617
1618
1619
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1689
1690
1691
1692
1693
1694
1695
1696
1696
1697
1698
1698
1699
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1709
1710
1711
1712
1713
1714
1715
1716
1716
1717
1718
1719
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1789
1790
1791
1792
1793
1794
1795
1796
1796
1797
1798
1798
1799
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1809
1810
1811
1812
1813
1814
1815
1816
1816
1817
1818
1819
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1889
1890
1891
1892
1893
1894
1895
1896
1896
1897
1898
1898
1899
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1909
1910
1911
1912
1913
1914
1915
1916
1916
1917
1918
1919
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1989
1990
1991
1992
1993
1994
1995
1996
1996
1997
1998
1998
1999
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2089
2090
2091
2092
2093
2094
2095
2096
2096
2097
2098
2098
2099
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2109
2110
2111
2112
2113
2114
2115
2116
2116
2117
2118
2119
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2189
2190
2191
2192
2193
2194
2195
2196
2196
2197
2198
2198
2199
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2209
2210
2211
2212
2213
2214
2215
2216
2216
2217
2218
2219
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2289
2290
2291
2292
2293
2294
2295
2296
2296
22
```

5 Find Orders with Unusual Payment Installments

```

66 ✓ SELECT order_id, payment_installments, COUNT(*)
FROM Order_Payments
GROUP BY order_id, payment_installments
HAVING payment_installments > 12;
70

```

Data Output Messages Notifications

order_id	payment_installments	count
1	15	1
2	15	1
3	15	1
4	18	1
5	15	1
6	15	1
7	13	1
8	17	1
9	18	1
10	14	1
11	13	1
12	17	1

Fig. 14. Query 5

6 Find Orders with Customer Details

```

11 ✓ SELECT o.order_id, c.customer_unique_id, c.customer_city, c.customer_state, o.order_status, o.order_purchase_timestamp
FROM Orders o
JOIN Customers c ON o.customer_id = c.customer_id
ORDER BY o.order_purchase_timestamp DESC
LIMIT 10;
15

```

Data Output Messages Notifications

order_id	customer_unique_id	customer_city	customer_state	order_status	order_purchase_timestamp
10a45dfe6a50c1e7efed0384e16	87abffec99b8bd574917de5df0fc	sorocaba	SP	cancelled	2016-09-17 17:30:18
392e0a714a37c4767040a43e477	989795ebd567511a1a02f5e50bc46..	guininhos	SP	cancelled	2016-09-29 09:13:03
baac7ef7070d0094508461d2040	9cf3ffefeb7a0a41bc274596c56	belo horizonte	MG	cancelled	2016-09-25 11:59:16
b3d3647c02329384981d1a9e8b5a5	e71036eb0212994419c2664219e01	belo horizonte	MG	cancelled	2016-09-17 21:21:16
e8d44c52c19797e2332197e5837161	c1ee153060cb73b94914345fb3f54e..	mata	SC	cancelled	2016-09-13 09:56:12
7f853529520505050505050505050	09895194530511702930194	taubate	SP	cancelled	2016-09-17 14:45:44
46875294529452945294529452945	0206-09-17 14:45:44	santos de parnaiba	SP	cancelled	2016-09-16 14:45:47
a7a1120a9e0701a17e4991d80a83a..	6a5daad711105294107030101377aef	prata grande	SP	cancelled	2016-09-01 10:48:12
463319484529452945294529452945	96f4fa1e1c445b6c1e1a3c3bf6102	sao paulo	SP	cancelled	2016-09-03 14:14:25
3a3cddaa5a7a27951b96c3313412840	e9609185d2427a5e02e2741a5d4aa	santos	SP	cancelled	2016-08-31 16:13:44

Fig. 15. Query 6

7 Find Top 5 Best-Selling Products with Category

```

100 ✓ SELECT p.product_id, pc.product_category_name_english, COUNT(oi.product_id) AS total_sold
FROM Order_Items oi
JOIN Products p ON oi.product_id = p.product_id
JOIN Product_Categories_Translation pc ON p.product_category_name = pc.product_category_name
GROUP BY p.product_id, pc.product_category_name_english
ORDER BY total_sold DESC
106 LIMIT 5;
107
108

```

Data Output Messages Notifications

product_id	product_category_name_english	total_sold
1	furniture_decor	527
2	bed_bath_table	488
3	garden_tools	484
4	garden_tools	392
5	garden_tools	388

Fig. 16. Query 7

8 Find Sellers with the Highest Number of Orders

```

111 ✓ SELECT s.seller_id, s.seller_city, COUNT(oi.order_id) AS total_orders
FROM Sellers s
JOIN Order_Items oi ON s.seller_id = oi.seller_id
GROUP BY s.seller_id, s.seller_city
ORDER BY total_orders DESC
116 LIMIT 5;
117

```

Data Output Messages Notifications

seller_id	seller_city	total_orders
1	sao paulo	2033
2	ibitinga	1987
3	sao jose do rio preto	1931
4	santo andre	1775
5	piracicaba	1551

Fig. 17. Query 8

9 Find Orders That Took the Longest to Deliver

```

124 ✓ SELECT o.order_id, c.customer_city, o.order_purchase_timestamp, o.order_delivered_customer_date,
(o.order_delivered_customer_date - o.order_purchase_timestamp) AS delivery_time
FROM Orders o
JOIN Customers c ON o.customer_id = c.customer_id
WHERE o.order_delivered_customer_date IS NOT NULL
ORDER BY delivery_time DESC
130
131
132

```

Data Output Messages Notifications

order_id	customer_city	order_purchase_timestamp	order_delivered_customer_date	delivery_time
1	sao paulo	2016-10-04 19:41:32	2016-08-28 14:58:51	692 days 19:17:19
2	florianopolis	2016-10-05 13:22:20	2016-08-28 17:18:28	692 days 03:55:08
3	suzano	2016-10-08 12:13:38	2016-08-27 21:22:58	688 days 09:09:20
4	rio de janeiro	2016-10-05 11:23:13	2016-08-23 23:48:32	687 days 12:25:35
5	sinap	2016-10-04 11:44:01	2016-08-07 19:28:50	672 days 07:44:49

Fig. 18. Query 9

10 Find Most Popular Payment Methods

```

141 ✓ SELECT op.payment_type, COUNT(op.order_id) AS total_payments
FROM Order_Payments op
143 GROUP BY op.payment_type
144 ORDER BY total_payments DESC;
145

```

Data Output Messages Notifications

payment_type	total_payments
credit_card	76795
boleto	19784
voucher	5775
debit_card	1529
not_defined	3

Fig. 19. Query 10

11 Find the Most Common Review Score by Product Category

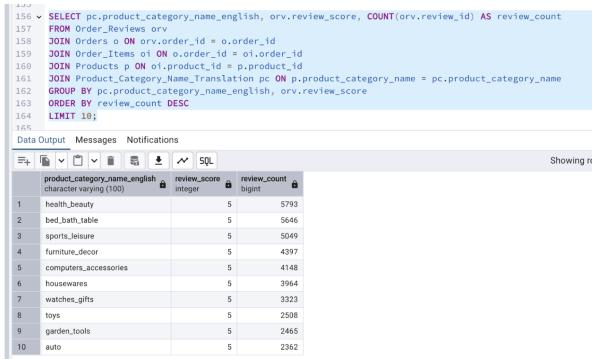


Fig. 20. Query 11

12 Find Total Revenue by Seller

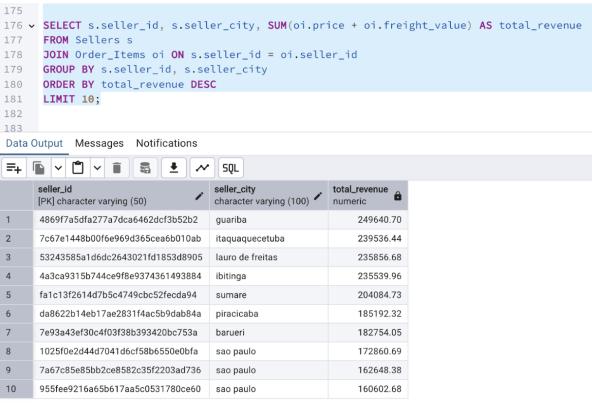


Fig. 21. Query 12

VIII. TASK 4: BCNF RELATIONS

Checking the BCNF of all realtions:

A. Customers Table (BCNF)

Attributes:

- customer_id (PK)
- customer_unique_id
- customer_zip_code_prefix
- customer_city
- customer_state

Functional Dependencies (FDs):

- customer_id → customer_unique_id, customer_zip_code_prefix, customer_city, customer_state
- customer_zip_code_prefix → (customer_city, customer_state)

BCNF Check: This table is already in BCNF.

B. Orders Table (BCNF)

Attributes:

- order_id (PK)
- customer_id (FK → Customers)

- order_status
- order_purchase_timestamp
- order_approved_at
- order_delivered_carrier_date
- order_delivered_customer_date
- order_estimated_delivery_date

Functional Dependencies (FDs):

- order_id → customer_id, order_status, order_purchase_timestamp, order_approved_at, order_delivered_carrier_date, order_delivered_customer_date, order_estimated_delivery_date

BCNF Check: This table is already in BCNF.

C. Product_Category_Name_Translation Table (BCNF)

Attributes:

- product_category_name (PK)
- product_category_name_english

Functional Dependencies (FDs):

- product_category_name → product_category_name_english

BCNF Check: This table is already in BCNF.

D. Products Table (BCNF)

Attributes:

- product_id (PK)
- product_category_name (FK → Product_Cat_Name_Translation)
- product_name_length
- product_description_length
- product_photos_qty
- product_weight_g
- product_length_cm
- product_height_cm
- product_width_cm

Functional Dependencies (FDs):

- product_id → product_category_name, product_name_length, product_description_length, product_photos_qty, product_weight_g, product_length_cm, product_height_cm, product_width_cm

BCNF Check: This table is already in BCNF.

E. Geolocation Table (BCNF)

Attributes:

- geolocation_zip_code_prefix (PK)
- geolocation_lat
- geolocation_lng
- geolocation_city
- geolocation_state

Functional Dependencies (FDs):

- geolocation_zip_code_prefix → geolocation_lat, geolocation_lng, geolocation_city, geolocation_state

BCNF Check: This table is already in BCNF.

F. Sellers Table (BCNF)

Attributes:

- seller_id (PK)
- seller_zip_code_prefix (FK → Geolocation)
- seller_city
- seller_state

Functional Dependencies (FDs):

- seller_id → seller_zip_code_prefix, seller_city, seller_state
- seller_zip_code_prefix → geolocation_city, geolocation_state from Geolocation.

BCNF Check: The primary key is seller_id, and all attributes are functionally dependent on it. This table is already in BCNF.

G. Order_Items Table (BCNF)

Attributes:

- order_id (PK, FK → Orders)
- order_item_id (PK)
- product_id (FK → Products)
- seller_id (FK → Sellers)
- shipping_limit_date
- price
- freight_value

Functional Dependencies (FDs):

- order_id, order_item_id → product_id, _id, shipping_limit_date, price, freight_value
- product_id → product_category_name
seller_id → seller_zip_code_prefix

BCNF Check: This table is already in BCNF.

H. Order_Payments Table (BCNF)

Attributes:

- order_id (PK, FK → Orders)
- payment_sequential (PK)
- payment_type
- payment_installments
- payment_value

Functional Dependencies (FDs):

- order_id, payment_sequential → payment_type, payment_installments payment_value
- order_id → customer_id (from Orders)

BCNF Check: This table is already in BCNF.

I. Order_Reviews Table (BCNF)

Attributes:

- review_id (PK)
- order_id (FK → Orders)
- review_score
- review_comment_title
- review_comment_message

- review_creation_date

- review_answer_timestamp

Functional Dependencies (FDs):

- order_id, order_item_id → product_id, seller_id, shipping_limit_date, price, freight_value
- product_id → product_category_name
seller_id → seller_zip_code_prefix
- review_id → order_id, review_score, review_comment_title, review_comment_message, review_creation_date, review_answer_timestamp

- order_id → customer_id (from Orders)

BCNF Check: This table is already in BCNF.

10. Leads_Qualified Table (BCNF)

Attributes:

- mql_id (PK)
- first_contact_date
- landing_page_id
- origin

Functional Dependencies (FDs):

- mql_id → first_contact_date, landing_page_id, origin

BCNF Check: The primary key is mql_id, and all attributes are functionally dependent on it. This table is already in BCNF.

11. Leads_Closed Table (BCNF)

Attributes:

- mql_id (PK, FK → Leads_Qualified)
- seller_id (PK, FK → Sellers)
- sdr_id
- sr_id
- won_date
- business_segment
- lead_type
- lead_behaviour_profile
- has_company
- has_gtin
- average_stock
- business_type
- declared_product_catalog_size
- declared_monthly_revenue

Functional Dependencies (FDs):

- mql_id, seller_id → sdr_id, sr_id, won_date, business_segment, lead_type, lead_behaviour_profile, has_company, has_gtin, average_stock, business_type, declared_product_catalog_size, declared_monthly_revenue

BCNF Check: The composite primary key is (mql_id, seller_id), and all attributes are functionally dependent on it. This table is already in BCNF.

TASK 5 QUERY EXECUTION AND PROCEDURE DESIGN

This task focuses on executing practical operations on the designed SQL database using DML statements like INSERT, UPDATE, DELETE, diverse SELECT queries, and creating stored procedures/functions for reusable operations. All operations were validated with SELECT queries post execution to ensure correctness.

Insert Query 1 – New Customer

Insert Query to insert record into Customer table



The screenshot shows the DBeaver interface with the following details:

- Toolbar:** Includes icons for file operations, database connections, and various tools.
- Query History:** A dropdown menu showing a single entry: "PhaseIDB/postgres@DMQL_phase1".
- Query Editor:** The main area contains the following SQL code:

```
1 ✓ INSERT INTO Customers (
2     customer_id, customer_unique_id, customer_zip_code_prefix, customer_city, customer_state
3 ) VALUES (
4     'new_customer_001', 'unique_abc123', 12345, 'Buffalo', 'NY'
5 )
6 );
7 |
```
- Data Output:** A tab labeled "Messages" is selected, showing the message "success" and the value "1".
- Notifications:** An empty tab.

Fig. 1. Inserting into customer table

Validating the insertion of record

```
Phase10B/postgres@DMQL_phase1: ~
Phase10B/postgres@DMQL_phase1: ~
Query History
1 ✓ INSERT INTO customers (
2     customer_id, customer_unique_id, customer_zip_code_prefix, customer_city, customer_state
3 ) VALUES (
4     'new_customer_801!', 'unique_abc123', 12345, 'Buffalo', 'NY'
5 );
6
7
8 select * from customers where customer_id='new_customer_801';
9
10 UPDATE Customers
11 SET customer_city = 'Albany', customer_state = 'NY'
12 WHERE customer_id = 'new_customer_801';
13
14 select * from customers

Data Output Messages Notifications
customer_id | customer_unique_id | customer_zip_code_prefix | customer_city | customer_state
Showing rows: 1
```

Fig. 2. Validation

Update Query 1 – Customer

Updating record in customer table

```
PhaseDB@postgres@DMQL_phase1* 
PhaseDB@postgres@DMQL_phase1 
Query History 
1 ✓ INSERT INTO Customers (
2     customer_id, customer_unique_id, customer_zip_code_prefix, customer_city, customer_state
3 ) VALUES (
4     'new_customer_001', 'unique_abc123', 12345, 'Buffalo', 'NY'
5 );
6
7
8 select * from customers where customer_id='new_customer_001';
9
10 ✓ UPDATE Customers
11 SET customer_city = 'Albany', customer_state = 'NY'
12 WHERE customer_id = 'new_customer_001';
13
14 select * from customers

Data Output Messages Notifications
UPDATE 1

Query returned successfully in 92 msec.
```

Fig. 3. Updating Customer records

Validating the update query

```
Phase1DB/postgres@DML_phase1*  
Phase1DB/postgres@DML_phase1  
Query History  
1 ✓ INSERT INTO Customers (2   customer_id, customer_unique_id, customer_zip_code_prefix, customer_city, customer_state3 ) VALUES (4   'new_customer_001', 'unique_abcd123', 12345, 'Buffalo', 'NY'  
5 );  
6  
7  
8 select * from customers where customer_id='new_customer_001';  
9 ✓ UPDATE Customers  
10 SET customer_city = 'Albany', customer_state = 'NY'  
11 WHERE customer_id = 'new_customer_001';  
12  
13 select * from customers where customer_id='new_customer_001';  
14  
Data Output Messages Notifications  
customer_id character varying(50) primary key  
customer_unique_id character varying(50)  
customer_zip_code_prefix character varying(10)  
customer_city character varying(100)  
customer_state character varying(50)  
1 new_customer_001 unique_abcd123 12345 Albany NY
```

Fig. 4. Validation

Delete Query 1 – Reviews

Delete record from review table

```
15 ✓ DELETE FROM Order_Reviews  
16 WHERE review_score = 1 AND review_comment_message IS NULL;  
17  
18 ✓ select * from Order_Reviews  
19  
20  
21  
22 select * from orders  
23  
24 INSERT INTO Orders (  
25     order_id,  
26     customer_id,  
27     order_status,  
28     order_purchase_timestamp,  
29     order_approved_at,  
30     order_delivered_carrier_date
```

Fig. 5. Deleting Reviews records

Validating the delete query

Data Output Notifications						
	File	View	Table	Text	SQL	Showing rows: 1 to 1000
19			select * from Order_Reviews			
20						
21			select * from orders			
22						
23			INSERT INTO Orders (
24			order_id,			
25			customer_id,			
26			order_status,			
27			order_created_timestamp,			
28			order_approved_at,			
29			order_delivered_customer_data			

Fig. 6. Validation

Insert Query 2 – Orders

Inserting record into Orders table

Fig. 7. Inserting Order records

Validating the insertion of record in Order table

Fig. 8. Validation

Insert Query 3 – Product category name translation

Inserting record in Product category name translation table

Fig. 9. Inserting Record

Validating the insertion

	product_category_name	product_category_name_english
52	fashion_esporte	fashion_sport
53	sinalizacao_e_seguranca	signaling_and_security
54	pcs	computers
55	artigos_de_natal	christmas_supplies
56	fashion_roupa_feminina	fashion_female_clothing
57	eletrodomesticos_2	home_appliances_2
58	livros_importados	books_imported
59	bebidas	drinks
60	cine_foto	cine_photo
61	la_cuisine	la_cuisine
62	musica	music
63	casa_comforto_2	home_comfort_2
64	portaleis_casa_forno_e_cafe	small_appliances_home_oven_and_cof..
65	cds_dvds_musicais	cds_dvds_musicals
66	dvds_blu_ray	DVDS_BLU_RAY
67	flores	flowers
68	artes_e_artesanato	arts_and_craftsmanship
69	fraldas_higiene	diapers_and_hygiene
70	fashion_roupa_infantil_juvenil	fashion_childrens_clothes
71	seguros_e_servicos	security_and_services
72	electronics	Electronics

Fig. 10. Validation

Update Query 2 – Products

Updating record in Product table

Fig. 11. Updating Product records

Validating the update query

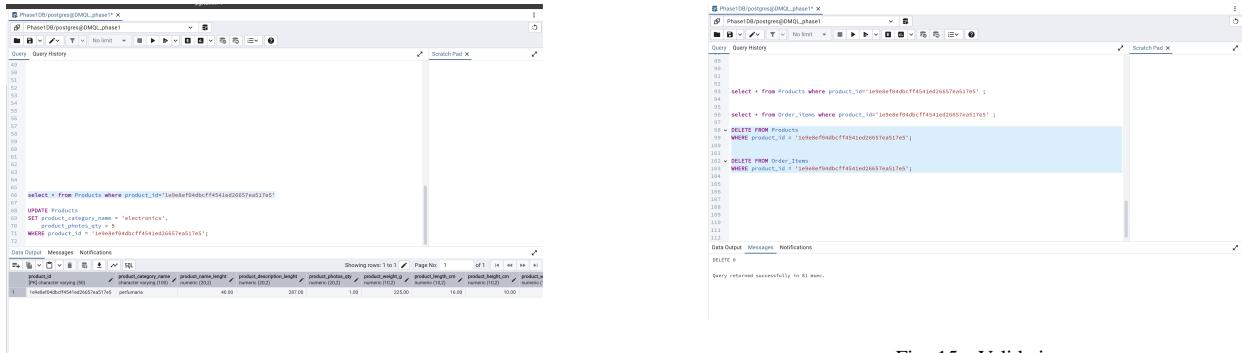


Fig. 12. Validation

Fig. 15. Validation

Validating the update query

SELECT Queries using Groupby, Orderby, Joins etc

We have demonstrated by using JOIN and ORDERBY

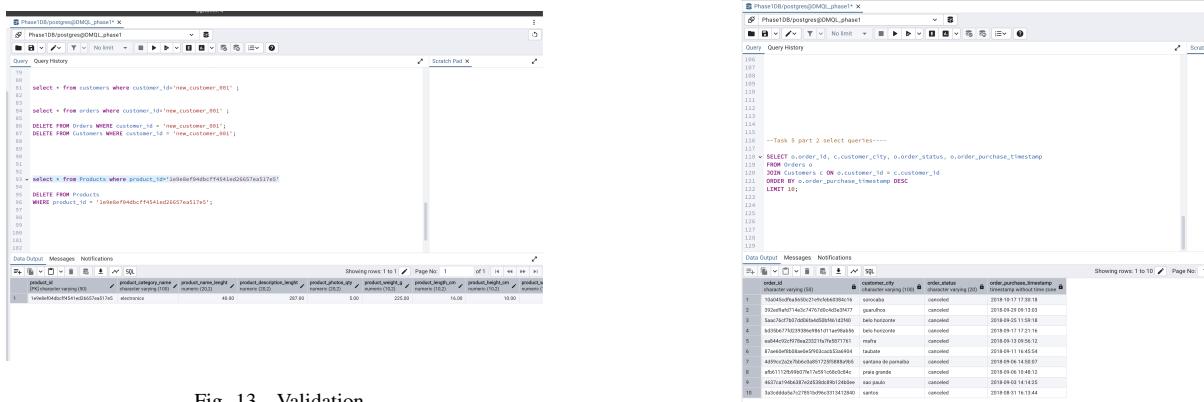


Fig. 13. Validation

Fig. 16. JOIN and ORDERBY

Delete Query 2 – Orders and Customers

JOIN clause demonstration

Delete queries on Orders and Customer table

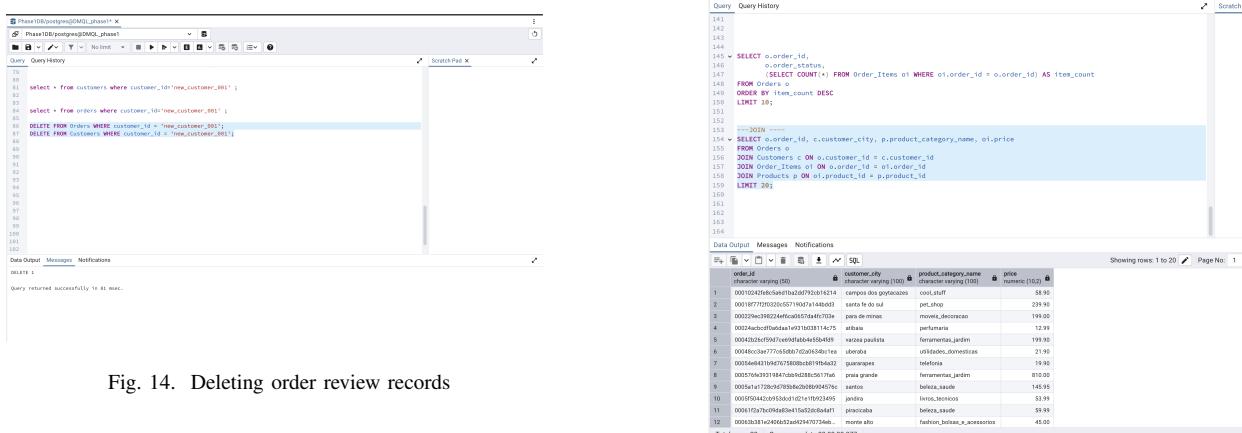


Fig. 14. Deleting order review records

Fig. 17. JOIN

Delete Query 3 – Products and order Items

JOIN GROUPBY ORDERBY Demonstration

```
Phase1DB[postgres@DMQL_phase1 ~]
Query History
Query
113
114
115
116 --Task 5 part 2 select queries-----
117
118 > SELECT o.order_id, c.customer_city, o.order_status, o.order_purchase_timestamp
119   FROM Orders o
120   JOIN Customers c ON o.customer_id = c.customer_id
121   ORDER BY o.order_purchase_timestamp DESC
122   LIMIT 10;
123
124
125
126
127
128
129 > SELECT c.customer_state, COUNT(o.order_id) AS total_orders
130   FROM Customers c
131   JOIN Orders o ON c.customer_id = o.customer_id
132   GROUP BY c.customer_state
133   ORDER BY total_orders DESC;
134
135
136
Data Output Messages Notifications


| #  | customer_state | total_orders |
|----|----------------|--------------|
| 0  | SP             | 40395        |
| 2  | RJ             | 12377        |
| 3  | MG             | 11255        |
| 4  | RS             | 5277         |
| 5  | PR             | 4882         |
| 6  | SC             | 3029         |
| 7  | BA             | 3276         |
| 8  | ES             | 2072         |
| 9  | GO             | 1663         |
| 10 | GO             | 1661         |
| 11 | PE             | 1664         |
| 12 | CE             | 1311         |


Showing rows 1 to 27 | Page No.
```

Fig. 18. JOIN, GROUPBY and ORDERBY

ORDERBY clause ,SUBQUERY,LIMIT Demonstration

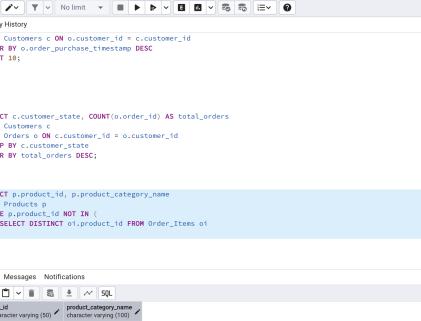
```
Phase10B/postgres@DMQL_phase1 ~
Phase10B/postgres@DMQL_phase1
Query  Query History
130
131
132 ~> SELECT p.product_id, p.product_category_name
133   FROM Products p
134 WHERE p.product_id NOT IN (
135       SELECT DISTINCT o1.product_id FROM Order_Items o1
136   )
137
138
139
140
141
142
143
144
145
146 ~>   SELECT o.order_id,
147           o.order_status,
148           (SELECT COUNT(*) FROM Order_Items oj WHERE oj.order_id = o.order_id) AS item_count
149   FROM Orders o
150 ORDER BY item_count DESC
151
152
153
154
155
156
157
158
159
160 LIMIT 10;
151
152
153
154
155
156
157
158
159
160
Data Output  Messages  Notifications
Showing rows: 1 to 10  Page first


| #  | order_id                           | order_status | item_count |
|----|------------------------------------|--------------|------------|
| 1  | 827265310371735e4f641204eae4ef     | delivered    | 21         |
| 2  | 1b1597401d41e5426052dc4cd7c171a    | delivered    | 20         |
| 3  | a013035a01d41e5426052dc4cd7c1709   | delivered    | 20         |
| 4  | 1bf170e4e3c1b310940951cd20bd4079   | delivered    | 15         |
| 5  | 42362ba503a1d41e5426052dc4cd7c16d1 | delivered    | 15         |
| 6  | 73ca5ba33071a452100179e4407179     | delivered    | 14         |
| 7  | 95d0d46c71a54a6b050257205688fc8    | delivered    | 14         |
| 8  | 37ea401157a15ab23e34049d810242b    | delivered    | 13         |
| 9  | c0596a765652720c7805e094648ab9e9   | delivered    | 12         |
| 10 | 2ca18a37103993c98510115aaecc0      | delivered    | 12         |


```

Fig. 19. ORDERBY, SUBQUERY,LIMIT

SUBQUERY Demonstration



```
SELECT c.customer_id, c.customer_name, COUNT(o.order_id) AS total_orders
FROM Customers c
JOIN Orders o ON c.customer_id = o.customer_id
WHERE o.order_purchase_timestamp BETWEEN '2016-01-01' AND '2016-03-31'
ORDER BY total_orders DESC
LIMIT 10;
```

```
SELECT p.product_id, p.product_category_name
FROM Products p
WHERE p.product_id NOT IN (
    SELECT DISTINCT oi.product_id FROM Order_Items oi
);
```

Data Output Messages Notifications

product_id productCategoryName

[PK] character varying (50) character varying (100)

Fig. 20. SUBQUERY

```
PhaseDB(phase@DMQL_phase1) 
PhaseDB/postgres@DMQL_phase1

Query History
133   SELECT DISTINCT o1.product_id FROM Order_Items o1
134 ;
141
142
143
144 ===Subquery===
145   SELECT o.order_id,
146         o.order_status,
147         (SELECT COUNT(*) FROM Order_Items o WHERE o1.order_id = o.order_id) AS item_count
148   FROM orders o
149   GROUP BY item_count DESC
150   LIMIT 10;
151
152
153 ===JOIN===
154   SELECT o.order_id, c.customer_city, p.product_category_name, o1.price
155   FROM orders o
156   JOIN customer c ON o.customer_id = c.customer_id
157   JOIN Order_Items o1 ON o.order_id = o1.order_id
158   JOIN Products p ON o1.product_id = p.product_id
159   LIMIT 20;
160
161
162 ---GROUP BY + HAVING---
Data Output Messages Notifications
#SQL
order_id character varying(50) order_status character varying(20) item_count bigint
1 422726500100776e41250eef4f delivered 21
2 1b1171610414563464526139102ce2f delivered 20
3 a014fd5c02d6336456e3830102ce2f delivered 20
4 9ef13ed949a45731a9464564bb175 delivered 15
5 428267946d4138199fc0f9a8eb0d5 delivered 15
6 73d5a010704a3439065ff4eaef297a delivered 14
7 9bd544d71aa1de460090794ee8bb8 delivered 14
8 37ee401157a2a078c8cc6db8c824b delivered 13
9 c05d9a76545a27a2789e0f304baeb delivered 12
10 2c219e6703883e008572015aaeac delivered 12
```

Fig. 21. SUBQUERY

WHERE Clause and SUBQUERY Demonstration

```
Phase1DB[postgres@DMQL_phase1 ~]
-- Phase1DB[postgres@DMQL_phase1 ~]
\q
Query History
No limit
Query
125
126
127
128 -- SELECT c.customer_state, COUNT(o.order_id) AS total_orders
129 FROM Customers c
130 JOIN Orders o ON c.customer_id = o.customer_id
131 GROUP BY c.customer_state
132 ORDER BY total_orders DESC;
133
134
135 ---WHERE clauses---
136 -- SELECT p.product_id, p.product_category_name
137 FROM Products p
138 WHERE p.product_id NOT IN (
139     SELECT DISTINCT o1.product_id FROM Order_Items o1
140 )
141
142
143 ---Subquery---
144
145 -- SELECT o.order_id,
146 --       o.order_status,
147 --       (SELECT COUNT(*) FROM Order_Items o1 WHERE o1.order_id = o.order_id) AS item_count
148 FROM Orders o
149 WHERE o.order_status < 1000
Data Output Message Notifications
```

Fig. 22. WHERE and SUBQUERY

Procedures and Functions

The procedures help encapsulate operations like inserting, updating, or deleting records safely especially when foreign key constraints are involved.

Update Procedure

This function first inserts a new product into the Products table, then updates the same product to modify certain fields, such as product photos qty or category. we created this because this is useful when a product might need updates immediately after being created like correcting initial insertion errors or auto setting default fields.

```
CREATE OR REPLACE FUNCTION insert_then_update_product()
RETURNS VOID AS $$
BEGIN
    -- Insert the product
    INSERT INTO products(
        product_id,
        product_name,
        product_name_length,
        product_description_length,
        product_weight,
        product_width_cm,
        product_height_cm,
        product_depth_cm,
        values
    )
    VALUES
        (p_product_id,
        p_product_name,
        p_product_name_length,
        p_product_description_length,
        p_product_weight,
        p_product_width_cm,
        p_product_height_cm,
        p_product_depth_cm,
        p_product_width_cm);
    -- Update the product
    UPDATE products
    SET
        product_name = p_product_name,
        product_name_length = p_product_name_length,
        product_description_length = p_product_description_length,
        product_weight = p_product_weight,
        product_width_cm = p_product_width_cm,
        product_height_cm = p_product_height_cm,
        product_depth_cm = p_product_depth_cm,
        width_cm = p_product_width_cm
    WHERE product_id = p_product_id;
END;
$$ LANGUAGE plpgsql;
```

Fig. 23. Update Procedure

```
pgAdmin 4

Phase1DB[postgreSQL@DMOL_phase1] x
Phase1DB[postgreSQL@DMOL_phase1]
Query    Query Planner    Explain    View    No limit    Help   

Query    Query Planner    Explain    View    Help   

175    CREATE OR REPLACE FUNCTION insert_new_customer(
176        p_customer_id TEXT,
177        p_unique_id TEXT,
178        p_zip_code INTEGER,
179        p_city TEXT,
180        p_state TEXT
181     )
182     RETURNS VOID AS $$
183     BEGIN
184        INSERT INTO Customers (
185            customer_id, customer_unique_id, customer_zip_code_prefix, customer_city, customer_state
186        )
187        VALUES
188        (
189            p_customer_id, p_unique_id, p_zip_code, p_city, p_state
190        );
191     END;
192     $$ LANGUAGE plpgsql;
193
194    SELECT insert_new_customer('new_customer_BB02', 'uniq_xyzz', 99999, 'Rochester', 'NY');
195
196
197
198
Data Output    Messages    Notifications
Showing rows 1 to 1    Page 1
```

Fig. 26. Insertion procedure

Fig. 24. Update Procedure

The screenshot shows the pgAdmin 4 interface with a query editor window open. The title bar indicates the connection is to Phase1DB (postgres@DMQL_phase1). The query editor contains the following SQL code:

```
CREATE OR REPLACE FUNCTION insert_new_customer(
    p_customer_id TEXT,
    p_customer_unique_id TEXT,
    p_zip_code_prefix INTEGER,
    p_city TEXT,
    p_state TEXT
)
RETURNS VOID AS $$
BEGIN
    INSERT INTO Customers (
        customer_id, customer_unique_id, customer_zip_code_prefix, customer_city, customer_state
    )
    VALUES (
        p_customer_id, p_unique_id, p_zip_code, p_city, p_state
    );
END;
$$ LANGUAGE plpgsql;
```

Below the code, there is a comment: `SELECT insert_new_customer('new_customer_002', 'uniq_xyz', 99999, 'Rochester', 'NY');`. The status bar at the bottom right shows "Showing rows: 1 to 1".

Fig. 27. Validation

Delete Procedure

Before deleting a product, this function checks whether that product is referenced in Order Items which is a foreign key. If it is referenced, it blocks the deletion and returns a warning message. If it is not referenced, it safely deletes the product.

```
pydantic 4

Phase100\pgsql\MDM_phase1.x
Query History
Scratch Pad x

Query
SELECT * FROM Products WHERE product_id = 'new_product_001';

Data Output Messages Notifications SQL
product_id product_name product_category_name product_name_length product_description product_length product_photograph product_weight product_length_cm product_weight_g product_width_cm product_height_cm product_width_mm product_height_mm
1 new_product_001 electronics 35.0 200.00 3.00 800.00 20.00 10.00 10.00
```

Fig. 25. Validation

Insert Procedure

This function inserts a new customer record into the Customers table in a reusable and safer way. We created this so that any new customer can be easily added using a function call without manually writing long insert queries every time.

Fig. 28. Delete Procedure

Fig. 29. Validation

Fig. 30. Validation

Task 6 Transaction and Triggers

The goal of this task is to implement a transaction combining multiple related database operations and handle any transaction failures gracefully using a trigger mechanism. This ensures that database consistency and atomicity are preserved even under error conditions.

We implemented a transaction to insert an order into the Orders table and its corresponding items into the Order Items table. Here we have created a Transaction Block: If any of the INSERT operations fails due to foreign key constraint violation or invalid data, the transaction is automatically rolled back.

Failure Handling: A trigger-based failure handling mechanism was created using a Transaction Log table to record failure events and a log failure function which inserts failure messages into the Transaction Log table.

```
pgAdmin 6

Phase2_Task5.edb | Phase10(postgres) [DMQL_phase1] x
Phase10@postgres[DMQL_phase1] ~

Query History
Query
CREATE TABLE Transaction_Log (
    id SERIAL PRIMARY KEY,
    message TEXT,
    log_time TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE OR REPLACE FUNCTION log_failure(p_message TEXT)
RETURNS VOID AS $$
BEGIN
    INSERT INTO Transaction_Log(message) VALUES (p_message);
END;
$$ LANGUAGE plpgsql;

CREATE OR REPLACE FUNCTION create_order_transaction()
RETURNS VOID AS $$

DECLARE
    v_order_id TEXT := 'full_order_001';
BEGIN
    Start a transaction block
    BEGIN
        Insert into Orders
        INSERT INTO Orders (
            order_id, customer_id, order_status, order_purchase_timestamp,
            order_approved_at, order_estimated_delivery_date
        )
        VALUES (
            v_order_id, 'new_customer_001', 'processing', NOW(), NOW(), NOW() + interval '5 days'
        );
    END;
END;
$$ LANGUAGE plpgsql;

Data Output Messages Notifications
```

Fig. 31. Transaction log

```

CREATE OR REPLACE FUNCTION create_order_transaction()
RETURNS VOID AS $$
DECLARE
  v_order_id TEXT := 'fail_order_001';
BEGIN
  BEGIN
    -- Insert Into Orders
    INSERT INTO orders
    (customer_id, order_status, order_purchase_timestamp,
     order_approved_at, order_estimated_delivery_date)
    VALUES (
      v_order_id, 'new_customer_001', 'processing', NOW(), NOW(), NOW() + interval '5 days'
    );
    -- Intentionally fail: Inserting with invalid product_id
    INSERT INTO order_items (
      order_id, order_item_id, product_id, seller_id, shipping_limit_date, price, freight_value
    )
    VALUES (
      v_order_id, 1, 'INVALID_PRODUCT', 'some_seller', NOW(), 100, 10
    );
  EXCEPTION
    WHEN OTHERS THEN
      PERFORM log_failure('Transaction failed: ' || SQLERRM);
      RAISE;
  END;
END;
$$ LANGUAGE plpgsql;
$$
INSERT INTO Customers (
Data Output Messages Notifications
CREATE FUNCTION

Query returned successfully in 69 msec.
```

Fig. 32. Transaction

To test the robustness an intentional failure is simulated by inserting an order item referencing a non-existent product ID. Now no partial records were inserted. The entire transaction was rolled back as expected and corresponding failure entry appeared in the Transaction Log table with the appropriate error message.

Fig. 33. Insertion

```
Query 1: Query History
  1 -- Intentionally fail - Inserting with invalid product_id
  2 INSERT INTO Order_Items (
  3     order_id, order_item_id, product_id, seller_id, shipping_limit_date, price, freight_value
  4 )
  5 VALUES (
  6     v_order_id, 1, 'INVALID_PRODUCT', 'some_seller', NOW(), 100, 10
  7 )
  8
  9
 10
 11
 12
 13
 14
 15
 16
 17
 18
 19
 20
 21
 22
 23
 24
 25
 26
 27
 28
 29
 30
 31
 32
 33
 34
 35
 36
 37
 38
 39
 40
 41
 42
 43
 44
 45
 46
 47
 48
 49
 50
 51
 52
 53
 54
 55
 56
 57
 58
 59
 60
 61
 62
 63
 64
 65
 66
 67
 68
 69

Data Output Messages Notifications

ERROR: Insert or update on table "order_items" violates foreign key constraint "fk_order_items_product"
SQL state: 23503
Detail: Key (product_id)='INVALID_PRODUCT' is not present in table "products".
Context: SQL statement "INSERT INTO Order_Items (
    order_id, order_item_id, product_id, seller_id, shipping_limit_date, price, freight_value
)
VALUES (
    v_order_id, 1, 'INVALID_PRODUCT', 'some_seller', NOW(), 100, 10
)"
```

Fig. 34. Validation

```
postgres:~ % psql -d Phase10/postgres/MQOL_phase1
```

Query History

```
1) \c Phase10/postgres/MQOL_phase1
2) \d
3) \q
```

Scratch Pad

```
1) \c Phase10/postgres/MQOL_phase1
2) \d
3) \q
```

EXCEPTION

```
1) WHEN OTHERS THEN
2)   PERFORM Log_failure('Transaction failed: ' || SQLERRM);
3)   ROLLBACK;
4) END;
```

15 LANGUAGE plpgsql;

1) \c Phase10/postgres/MQOL_phase1
2) \d
3) \q

INSERT INTO Customers (

```
1) customer_id, customer_unique_id, customer_dsp_code_prefix, customer_city, customer_state
2) VALUES ('new_customer_001', 'unique_abcd123', 12345, 'Buffalo', 'NY')
3) 
```

1) \c Phase10/postgres/MQOL_phase1
2) \d
3) \q

SELECT create_order_transaction();

```
1) \c Phase10/postgres/MQOL_phase1
2) \d
3) \q
```

SELECT * FROM TransactionLog ORDER BY log_time DESC

```
1) \c Phase10/postgres/MQOL_phase1
2) \d
3) \q
```

Data Output | Messages | Notifications

log_time

Showing rows 1 to 1 | Page No: 1 of 1 x e

1) \c Phase10/postgres/MQOL_phase1
2) \d
3) \q

Fig. 35. Error logs

Justification: When the transaction encounters an error when inserting an invalid foreign key into the Order Items table, the database automatically aborts the entire transaction and None of the partially inserted records are saved. The database maintains consistency by performing a rollback. The log failure trigger captures the error and records it in the Transaction Log table for auditing purposes. This aligns with the principle of transaction atomicity, ensuring that either all operations succeed together or none are applied. Since our design constitutes all of the above points our design is safe and reliable, preserves data integrity.

TASK 7: INDEXING AND QUERY EXECUTION ANALYSIS

The objective of this task was to identify expensive queries through query cost analysis, and to propose indexing strategies, to implement indexes, and verify performance improvements through EXPLAIN plans.

Problematic Query 1 – Frequent Lookups on product id ,Order Items Table

Before Creating Index the explain plan gives us the below result:

pgAdmin 4

Phase2_task5.sql x Phase2_Task6.sql x PhaseDB/postgres@DMQL_phase1*

PhaseDB postgres@DMQL_phase1

Query History

1 EXPLAIN ANALYZE
2 SELECT p.product_id, p.product_category_name
3 FROM products p
4 WHERE p.product_id NOT IN (
5 SELECT product_id FROM Order_Items
6)
7
8 DROP INDEX IF EXISTS idx_order_items_product_id;
9
10 CREATE INDEX idx_order_items_product_id ON Order_Items(product_id);
11
12
13 EXPLAIN ANALYZE
14 SELECT p.product_id, p.product_category_name
15 FROM products p
16 WHERE p.product_id NOT IN (
17 SELECT product_id FROM Order_Items
18)
19
20
21
22 EXPLAIN ANALYZE
23 SELECT c.customer_state, COUNT(o.order_id) AS total_orders

Data Output Messages Notifications

Errs [] Info [] Warnings [] Logs [] Bulk [] SQL []

Showing rows: 1 to 7 | Page 1

QUERY PLAN Test

1 Seq Scan on products p (cost=0.00..11.4601 width=1632) (actual time=41.239..41.240 rows=0 loops=1)
2 Filter: NOT ANY((product_id = (hashedSubPlan1).col1)) test||
3 Rows Remained by Filter 32950
4 SubPlan 1
5 Seq Scan on order_items o (cost=0.00..3425.49 width=33) (actual time=0.010..13.639 rows=112849 loops=...
6 Planning Time 0.312 ms
7 Execution Time 41.478 ms

Fig. 36. Indexing on Order Items table

```
Phase2_task5.sql | Phase2_task6.sql | Phase1DB(postgres@DMQL_phase1) x
Phase1DB(postgres@DMQL_phase1)
Query History
Query 1 of 1
Data Output Message Notifications
SQL
EXPLAIN ANALYZE
 1. SELECT p.product_id, p.product_category_name
 2. FROM Products p
 3. WHERE p.product_id NOT IN
 4.       (SELECT product_id FROM Order_Items
 5. );
 6.
 7. DROP INDEX IF EXISTS idx_order_items_product_id;
 8.
 9. CREATE INDEX idx_order_items_product_id ON Order_Items(product_id);
10.
11.
12.
13. EXPLAIN ANALYZE
14. SELECT p.product_id, p.product_category_name
15. FROM Products p
16. WHERE p.product_id NOT IN (
17.       SELECT product_id FROM Order_Items
18. );
19.
20.
21.
22. EXPLAIN ANALYZE
23. SELECT c.customer_state, COUNT(o.order_id) AS total_orders
Data Output Message Notifications
Showing rows: 1 to 8 | Page
Eg. □ □ □ □ □ □ □ □ □ SQL
QUERY PLAN
Total time: 0.000 ms
1. Seq Scan on products p (cost=2291.77..4185.81 rows=16722 width=48) (actual time=37.991..37.992 rows=0 loops=1)
   Filter: (NOT ANY (join_order_id(text) = (hashedSubPlan1.out)))::text
   Rows Removed by Filter: 32960
2. Rows Removed by Filter: 32960
3. SubPlan 1
   -> Only If Only Scanning (ix_order_items.product_id) on order_items (cost=0.42..3010.15 rows=112649 width=33) (actual time=0.079..10.438 rows=112649 loops=1)
      Heap Fetches: 195
4. Planning Time: 0.262 ms
5. Execution Time: 37.928 ms
```

Fig. 37. Result

After Indexing the EXPLAIN Plan Summary shows us that index scan used on product id Since product id is frequently used in WHERE clauses for lookups, indexing reduced the search time from a full scan to a fast indexed retrieval the cost intially is around 3700 and after indexing it reduced to around 3200.

Problematic Query 2 – Customer Based Lookups

Before Creating Index the explain plan gives us the below result:

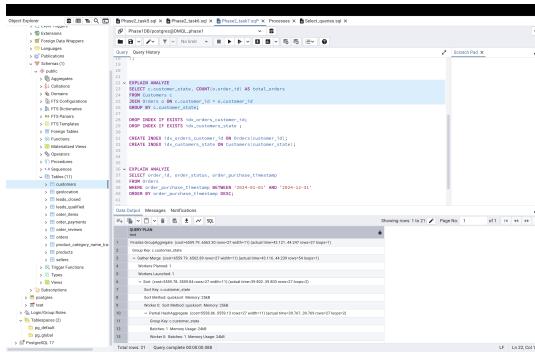


Fig. 38. Indexing on Order table

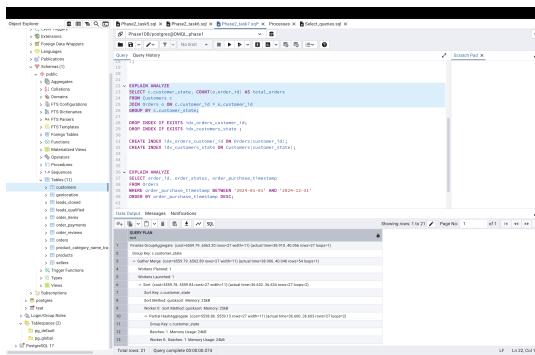


Fig. 39. Result

Customers often retrieve their orders by customer ID. Creating an index on customer id enables quick retrieval instead of scanning the entire Orders table. After Indexing the EXPLAIN Plan Summary shows us that index scan used on customer id. Since customer id is frequently used in WHERE clauses for lookups, indexing reduced the search time from a full scan to a fast indexed retrieval.

Problematic Query 3 – Filtering and Sorting by order purchase timestamp

Before Creating Index the explain plan gives us the below result:



Fig. 40. Indexing on Order table



Fig. 41. Result

Before Indexing the EXPLAIN plan summary shows that it did a full sequential scan through all orders and sorting is performed after filtering. After Indexing the EXPLAIN plan summary shows that it did index only scan on order purchase timestamp and sorting became faster. Adding an index on order purchase timestamp improved performance for time-range queries and accelerated ORDER BY operations because the data is already partially sorted in the index.

I. VISUALISATIONS USING TABLEAU

Orders by State

SQL Query:

```
SELECT customer_state,
COUNT(DISTINCT order_id)
AS total_orders
FROM customers
JOIN orders ON
customers.customer_id = orders.customer_id
GROUP BY customer_state
```

This visualization highlights the total number of orders placed from each state. Sao Paulo (SP) has the highest number of orders, followed by Rio de Janeiro (RJ) and Minas Gerais (MG). This regional concentration suggests that marketing efforts can be focused on Southeast Brazil for higher customer engagement.

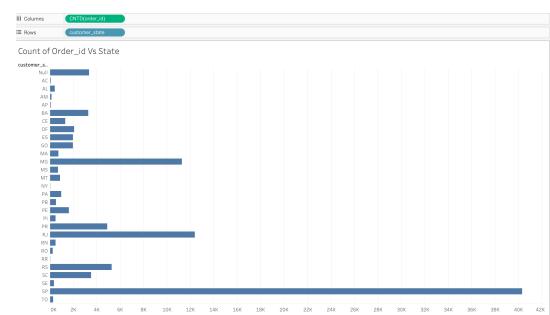


Fig. 42. Visualisation 1

Orders Over Time

```
SELECT YEAR(order_purchase_timestamp) AS year,
COUNT(DISTINCT order_id) AS total_orders
FROM orders
```

```
GROUP BY year
ORDER BY year;
```

This line chart shows the growth of orders over time, with a noticeable increase in 2017. This indicates the platform's rapid growth and adoption by consumers during this period. Understanding these trends can help predict future growth and resource planning.



Fig. 43. Visualisation 2

Top Product Categories

```
SELECT product_category_name,
COUNT(*) AS total_orders
FROM order_items
GROUP BY product_category_name
ORDER BY total_orders DESC;
```

Product categories like beleza saude and informatica aces-
sorios lead in order volume, indicating consumer preference
for health-related and electronics products. These insights can
guide inventory management and marketing strategies.

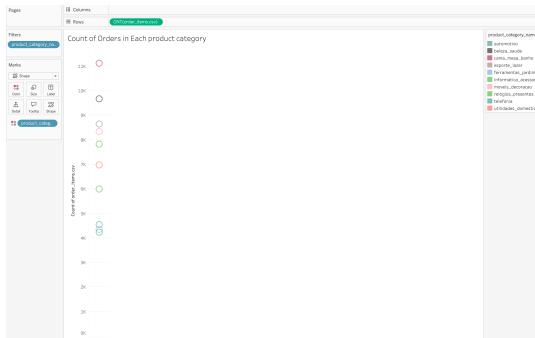


Fig. 44. Visualisation 3

Discounts by Category

```
SELECT product_category_name,
COUNT(DISTINCT price)
AS unique_price_points
FROM order_items
GROUP BY product_category_name
ORDER BY unique_price_points DESC;
```

The number of discounts for different product categories is represented in a Tableau bubble chart that derives its visual data from unique price values in order items. The visual presentation includes bubbles which represent individual categories of products and their sizes demonstrate the number of discounted items within each group. The product categories containing beleza saude, informatica acessorios and utilidades domesticas possess large bubble sizes which demonstrates their greater number of distinct discounted products. The couple of product categories brinquedos and ferramentas jardim present smaller bubble dimensions which indicates either a lack of many distinct prices or sparse discount offers. The analysis found 9179 different prices among recorded data throughout all categories.

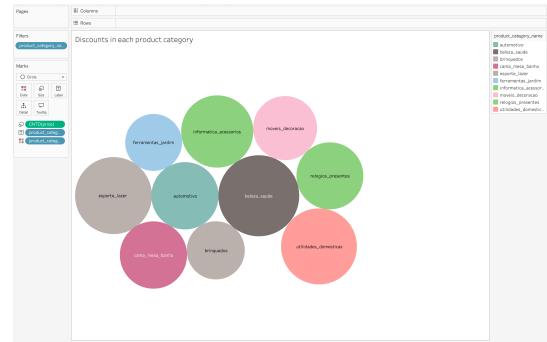


Fig. 45. Visualisation 4

Orders Sold by Sellers

The visualization depicts seller order counts through this Tableau heatmap which displays the data using seller id 1 as identifier. Order item id sums up to show the complete item quantity sold by each seller. Each bar represents a seller who has its color scheme indicating their order volume numbers through intensity changes. Between them, one seller achieves 2,898 orders while multiple others reach between 250 and 1,200 orders. The graphic illustration indicates that seller performance levels are uneven because many transactions are handled by merely a handful of sellers. A total of 60,795 order items have been reported by the top 100 sellers.

```
SELECT seller_id_1, SUM(order_item_id)
AS total_orders_sold
FROM order_items
GROUP BY seller_id_1
ORDER BY total_orders_sold DESC;
```

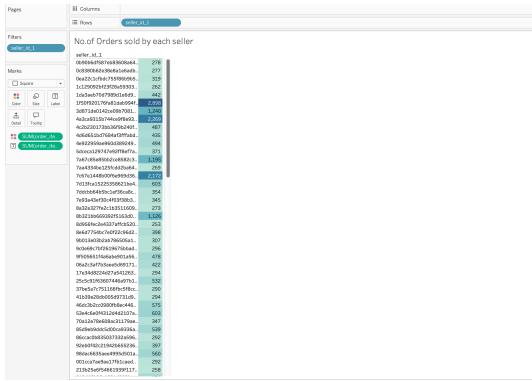


Fig. 46. Visualisation 5

Average Price of items

Using order items data the Tableau tool shows average prices of top 10 products through a stacked bar chart format. The vertical sizes of colorful bar segments stand for different product categories while showing their average pricing information. Items found in pcs and climatizacao together with construcao ferramente possess elevated average costs comparing to lower-priced products within telefonica fixa and relogios presentes. The cumulative average value amounts to 3,924.4 dollars through the combined analysis of multiple categories. The graphical representation lets companies notice which product segments exist in the premium category by reflecting their pricing structure.

```
SELECT product_category_name,
AVG(price) AS avg_price
FROM order_items
GROUP BY product_category_name
ORDER BY avg_price DESC
LIMIT 10;
```

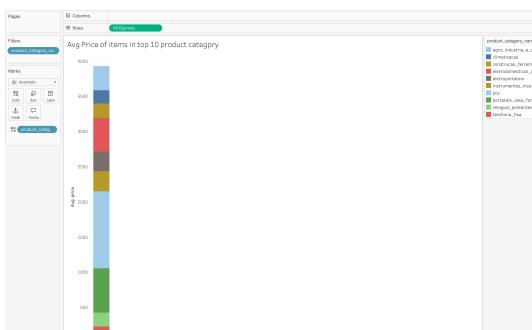


Fig. 47. Visualisation 6

Revenue generated in top categories

The revenue picture depicted by Tableau treemap demonstrates how the top 20 product categories generate revenue through item price summations in the order items database. The visualization displays categories through rectangles that present their total revenue worth through size measurements.

The three product groups beleza saude, relogios presentes and esporte lazer rule the market with the largest amount of revenue. The highest manufacturing revenue comes from three main groups: informatica acessorios, cama mesa banho, and utilidades domesticas. The pcs pet shop and eletroportateis rectangles illustrate categories with a lower level of revenue. The leading 20 retail product categories together produce approximately 11,418,780 dollars in total revenue.

```
SELECT product_category_name,
SUM(price) AS total_revenue
FROM order_items
GROUP BY product_category_name
ORDER BY total_revenue DESC
LIMIT 20;
```

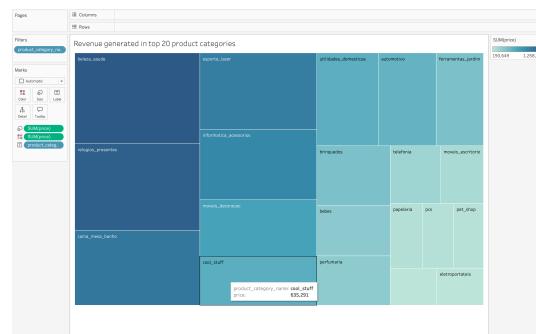


Fig. 48. Visualisation 7

The average order counts for each product category are displayed in this bubble chart using AVG(order item id) as the metric. The visual representation shows each blue circle as a product category while the circle size illustrates the average order statistics for these categories. Among the categories, the average number of placed orders stand highest in telefonica fixa and moveis sala and climatizacao while the sizes of perfumaria and beleza saude along with eletronicos fall at intermediate levels. The displayed graphic presents an even distribution of demand levels among different product categories. As displayed category data adds up to 62.67, the order intensity varies widely.

```
SELECT product_category_name,
AVG(order_item_id) AS avg_orders
FROM order_items
GROUP BY product_category_name;
```

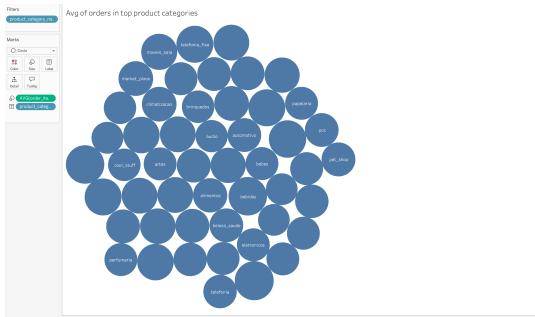


Fig. 49. Visualisation 8

The below are the link for the dashboard visualisations

- 1)View Dashboard 1
 - 2)View Dashboard 2
 - 3)View Dashboard 3

CONCLUSION

This Brazilian E-Commerce Database Design is an organized relational schema that can model much of an online market database is able to manage large-scale transactional data, seller and buyer activity, and marketing analysis with data integrity, scalability, and performance.

Through a careful analysis of relationships between entities, 11 tables were designed, which include the customers, orders, products, payments, reviews, sellers, geolocation information, and marketing leads. All tables were designed with care using primary keys, foreign keys, and referential integrity constraints, so that database consistency could be maintained. The assertion continued: entity relationship modeling to decide cardinality and create relationships among tables.

Specify attribute data like purpose, type, and constraints for every column. foreign key constraints and cascade operations for primary data integrity. Normalization rules for data redundancy reduction and query optimization.Optimal best practice SQL schema script deployment for optimal table structure. the proposed database schema is ideal for efficient retrieval, analysis, and business intelligence to decision makers like sellers, buyers, and business managers. The structured solution addresses spreadsheet data management issues with secure, stable, and query-optimized transaction processing.

Through analysis and real-world implementation, we were able to effectively illustrate key aspects of relational database management in Phase 2 of the project. Through validation procedures following each operation, we ensured referential integrity while performing data manipulation operations like insertion, updating, and deletion across several related tables.

To extract valuable information from the dataset, sophisticated SQL queries were created and run using JOIN, GROUP BY, HAVING, ORDER BY, and subqueries. Additionally, to improve modularity and code maintainability, reusable stored procedures and functions were created to encapsulate common database operations.

A composite transaction with exception handling was used for transaction management. In order to enforce atomicity and

consistency in the database even in the event of a failure, a specific trigger function was developed to log transaction failures into an audit table.

Furthermore, indexing techniques were used to optimize expensive queries, which greatly shortened execution times, as confirmed by EXPLAIN ANALYZE plans. The significance of appropriate indexing in improving database performance was highlighted by these optimizations.

Lastly, Tableau Public was used to create an interactive business intelligence dashboard. Key trends and patterns in the dataset were displayed on the dashboard, offering useful information for making important choices.

ACKNOWLEDGMENT

We would like to express our heartfelt gratitude to our professor, Dr. Shamsad Parvin, for teaching us various tools and techniques essential for this project. Your insights into the importance of conducting this type of research have been invaluable in shaping our understanding and approach. Thank you for your guidance and support throughout this journey.

CONTRIBUTION

Everyone has equal contribution in this project phase 1

Member	Percentage
jkollu	33.33%
lkodali	33.33%
smandru	33.33%

REFERENCES

- [1] <https://www.kaggle.com/datasets/olistbr/brazilian-commerce>
 - [2] <https://www.postgresql.org/docs/>
 - [3] Database System Concepts 7th Edition — Abraham Silberschatz,
Henry F. Korth, S. Sudarshan
 - [4] Lecture Slides