E-commerce & Retail Analytics

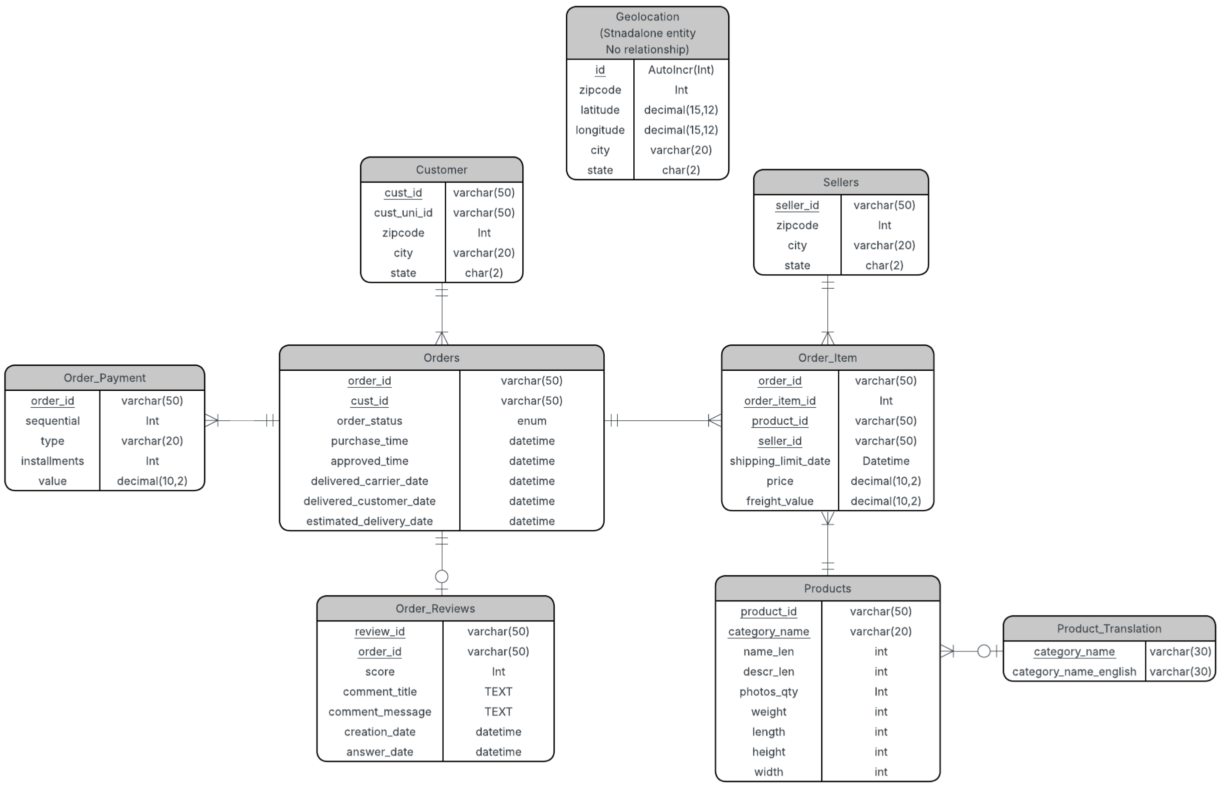
The Brazilian E-Commerce Public Dataset by Olist is a real-world dataset from the Olist marketplace in Brazil. It records over 100,000 orders placed between 2016 and 2018, including detailed information about customers, sellers, products, payments, deliveries, and reviews.

The dataset consists of multiple interconnected CSV files, such as orders, customers, sellers, order_items, payments, products, and reviews, enabling end-to-end analysis of the order journey: from purchase and payment to delivery and customer feedback.

Dataset/CSV files :

olist_customers_dataset.csv
olist_geolocation_dataset.csv
olist_order_items_dataset.csv
olist_order_payments_dataset.csv
olist_order_reviews_dataset.csv
olist_orders_dataset.csv
olist_products_dataset.csv
olist_sellers_dataset.csv
product_category_name_translation.csv

ER_Diagram :



Normalization Report (3NF / BCNF) :

The schema contains 9 fully normalized tables, each tested against 1NF , 3NF / BCNF conditions.

1. **customers** table → In BCNF

   Dependencies:
   - customer_id → customer_unique_id
   - customer_id → zip_code_prefix
   - customer_id → city, state

   1NF: All fields atomic (state, city, zip, ID).
   3NF: No field depends on another non-key field.
   BCNF: customer_id (PK) determines all the columns of the table, satisfies BCNF.

2. **sellers table** → BCNF

   PK: seller_id
   BCNF: (PK) determines all the columns of the table, satisfies BCNF.

3. **products** table → BCNF

   PK: product_id

   Dependencies:
   - product_id → all product attributes

   BCNF: (PK) determines all the columns of the table, satisfies BCNF.

4. **product_category_translation** → BCNF

   PK: product_category_name
   BCNF: (PK) determines all the columns of the table, satisfies BCNF.

5. **orders Table** → BCNF

   PK: order_id
   Dependencies:
   order_id → all order_timestamp fields
   order_id → status, customer_id
   BCNF: (PK) determines all the columns of the table, satisfies BCNF.

6. **order_items** Table → BCNF

   Composite PK: (order_id, order_item_id)
   Dependencies:
   - (order_id, order_item_id) → product_id
   - (order_id, order_item_id) → seller_id
   - (order_id, order_item_id) → price, freight_value

   BCNF: (Composite Key) determines all the columns of the table, satisfies BCNF.

7. **order_payments** Table → BCNF

   Composite PK: (order_id, payment_sequential)
   Dependencies:
   - (order_id, payment_sequential) → type, installments, value

   BCNF: (Composite Key) determines all the columns of the table, satisfies BCNF.

8. **order_reviews** Table → BCNF

   PK: review_id

   Dependencies:
   - review_id → order_id
   - review_id → score, message, timestamp

   BCNF: (Primary Key) determines all the columns of the table, satisfies BCNF.

9. **geolocation** Table → 3NF
   PK: Auto-increment geolocation_id
   3NF : All attributes depend on the PK; there is no transitive dependency

Handling Data Anomalies -

1. **For order_items :** We have added the On Update and Delete constraints -

```
FOREIGN KEY (order_id)
      REFERENCES orders(order_id)
      ON UPDATE CASCADE
      ON DELETE CASCADE
```

This will allow the to handle the delete anomalies . If the record in parent table orders is deleted, respective record in order_items table will also be deleted.

2. **For order_items :**

```
FOREIGN KEY (product_id)
```

```
                    REFERENCES products(product_id)
                    ON UPDATE CASCADE
                    ON DELETE RESTRICT,
```

ON UPDATE CASCADE :

If the product_id in the products table is updated (changed):

For Example: In product table

product_id = 'ABC123' → changed to → 'XYZ999', in the order_items also
product_id 'ABC123' will be updated to 'XYZ999'

This keeps the data consistent.

ON DELETE RESTRICT

This prevents deletion. we cannot delete a product from the products table,IF it is still used in order_items.

For Example:
If a product appears in ANY order:
DELETE FROM products WHERE product_id = 'ABC123';

It will give an error : “update or delete on table "products" violates foreign key constraint”

Handling these issues will :
- prevent accidental data loss
- maintain historical order accuracy
- allow safe updates

Data inconsistency Handling -

While inserting data into the table scheme, there were some data inconsistency issues that were handled by using Pandas and SQLAlchemy in data ingestion python script.

1. Inserting Null Values :

The Olist dataset has **product** categories that do NOT appear in the **product_category_translation** table, such as:
- pc_gamer
- la_cuisine
- fashion_bags
- Home_comfort, etc.

This is a known inconsistency in the dataset.

Solution implemented :

Allow missing categories by setting ON DELETE SET NULL and cleaning category names in Python

Modifying code in ingest_data.py :

```
df_products['product_category_name'] = df_products['product_category_name'].apply(
    lambda x: x if x in df_cat['product_category_name'].values else None)
```

This will allow insertion by inserting NULL for absent records.

2. Removing Duplicate values :

   In the **order_reviews** table, multiple review entries for the same review_id were found, i.e Duplicate rows in the CSV were addressed.

Solution implemented :

Remove duplicated review_id rows before inserting into the table.

Modifying code in ingest_data.py :

```
df_reviews = df_reviews.drop_duplicates(subset=["review_id"])
```

This will allow only one row per review_id remain in the table .