DDL_Script_MAIN.sql (Data Architecture & Modeling)

Language: PostgreSQL

```
/*
================================================================================
  OLIST E-COMMERCE — FULL DATA MODEL DDL
  CRM-Ready Segmentation Engine for PostgreSQL
================================================================================

  ENTITY-RELATIONSHIP MAP
  ═══════════════════════

  product_category_name_translation   (standalone lookup — no FKs)
       │
       └──► olist_products_dataset.product_category_name (logical join)

  olist_geolocation_dataset           (standalone reference — no FKs)
       │
       └──► zip_code_prefix joins to customers & sellers (logical join)

  olist_customers_dataset             ◄── ROOT ENTITY (customer_id PK)
       │
       └──► olist_orders_dataset      (customer_id FK)
                 │
                 ├──► olist_order_payments_dataset   (order_id FK)
                 ├──► olist_order_reviews_dataset    (order_id FK)
                 └──► olist_order_items_dataset      (order_id FK)
                           │
                           ├──► olist_products_dataset   (product_id FK)
                           └──► olist_sellers_dataset     (seller_id FK)

  EXECUTION ORDER (respects FK dependency chain):
  ────────────────
    1. product_category_name_translation  (no dependencies)
    2. olist_geolocation_dataset          (no dependencies)
    3. olist_customers_dataset            (no dependencies)
    4. olist_products_dataset             (no dependencies)
    5. olist_sellers_dataset              (no dependencies)
    6. olist_orders_dataset               (depends on: customers)
    7. olist_order_payments_dataset       (depends on: orders)
    8. olist_order_reviews_dataset        (depends on: orders)
    9. olist_order_items_dataset          (depends on: orders, products, sellers)

================================================================================
  DDL SCRIPTS — EXECUTE IN THIS EXACT ORDER
================================================================================
*/


-- ╔══════════════════════════════════════════════════════════════════════════╗
-- ║  TABLE 1: PRODUCT CATEGORY NAME TRANSLATION (Lookup Table)               ║
```

```
-- ╠  Dimension: Product Category Affinity ("What They Like")                ╣
-- ╠  71 rows — maps Portuguese category names to English                    ╣
-- ╚═══════════════════════════════════════════════════════════════════════╝

DROP TABLE IF EXISTS product_category_name_translation CASCADE;

CREATE TABLE product_category_name_translation (
    product_category_name           VARCHAR(50)     NOT NULL,
    product_category_name_english   VARCHAR(50)     NOT NULL,

    CONSTRAINT pk_category_translation
        PRIMARY KEY (product_category_name)
);

COMMENT ON TABLE product_category_name_translation
    IS 'Lookup: Portuguese-to-English product category mapping. Joins to olist_products_dataset on product_category_name.';



-- ╔═══════════════════════════════════════════════════════════════════════╗
-- ╠  TABLE 2: GEOLOCATION (Reference/Dimension Table)                       ╣
-- ╠  Dimension: Geographic Demographics ("Where")                           ╣
-- ╠  1,000,163 rows — multiple lat/lng entries per zip code prefix          ╣
-- ╠  NOTE: No natural PK. zip_code_prefix has ~19K unique values across 1M  ╣
-- ╠  rows (multiple geocoded points per zip). We add a surrogate PK.        ╣
-- ╚═══════════════════════════════════════════════════════════════════════╝

DROP TABLE IF EXISTS olist_geolocation_dataset CASCADE;

CREATE TABLE olist_geolocation_dataset (
    geolocation_id                  SERIAL          NOT NULL,
    geolocation_zip_code_prefix     INTEGER         NOT NULL,
    geolocation_lat                 NUMERIC(10,7)   NOT NULL,
    geolocation_lng                 NUMERIC(10,7)   NOT NULL,
    geolocation_city                VARCHAR(50)     NOT NULL,
    geolocation_state               CHAR(2)         NOT NULL,

    CONSTRAINT pk_geolocation
        PRIMARY KEY (geolocation_id)
);

CREATE INDEX idx_geolocation_zip
    ON olist_geolocation_dataset (geolocation_zip_code_prefix);

CREATE INDEX idx_geolocation_state
    ON olist_geolocation_dataset (geolocation_state);

COMMENT ON TABLE olist_geolocation_dataset
    IS 'Geocoded reference data. Multiple lat/lng points per zip code prefix. Join to customers/sellers on zip_code_prefix.';



-- ╔═══════════════════════════════════════════════════════════════════════╗
-- ╠  TABLE 3: CUSTOMERS (Root Entity)                                       ╣
-- ╠  Dimension: Geographic Demographics + Core Identity                     ╣
```

```
-- ╠  99,441 rows — one row per order-instance; customer_unique_id is the    ╣
-- ╠  real human (96,096 distinct). customer_id is PK (1:1 with orders).     ╣
-- ╚═══════════════════════════════════════════════════════════════════════╝

DROP TABLE IF EXISTS olist_customers_dataset CASCADE;

CREATE TABLE olist_customers_dataset (
    customer_id                 CHAR(32)        NOT NULL,
    customer_unique_id          CHAR(32)        NOT NULL,
    customer_zip_code_prefix    INTEGER         NOT NULL,
    customer_city               VARCHAR(40)     NOT NULL,
    customer_state              CHAR(2)         NOT NULL,

    CONSTRAINT pk_customers
        PRIMARY KEY (customer_id)
);

CREATE INDEX idx_customers_unique_id
    ON olist_customers_dataset (customer_unique_id);

CREATE INDEX idx_customers_state
    ON olist_customers_dataset (customer_state);

CREATE INDEX idx_customers_zip
    ON olist_customers_dataset (customer_zip_code_prefix);

COMMENT ON TABLE olist_customers_dataset
    IS 'Customer dimension. customer_id is per-order; customer_unique_id is the real person. Aggregate on customer_unique_id for RFM.';



-- ╔═══════════════════════════════════════════════════════════════════════╗
-- ╠  TABLE 4: PRODUCTS                                                    ╣
-- ╠  Dimension: Product Category Affinity + Price/Freight Sensitivity    ╣
-- ╠  32,951 rows — one row per unique product SKU                        ╣
-- ╚═══════════════════════════════════════════════════════════════════════╝

DROP TABLE IF EXISTS olist_products_dataset CASCADE;

CREATE TABLE olist_products_dataset (
    product_id                  CHAR(32)        NOT NULL,
    product_category_name       VARCHAR(50),                -- 610 NULLs in source data
    product_name_lenght         SMALLINT,                   -- sic: original column name typo
    product_description_lenght  SMALLINT,                   -- sic: original column name typo
    product_photos_qty          SMALLINT,
    product_weight_g            INTEGER,                    -- max 40,425g; 2 NULLs
    product_length_cm           SMALLINT,
    product_height_cm           SMALLINT,
    product_width_cm            SMALLINT,

    CONSTRAINT pk_products
        PRIMARY KEY (product_id)
);

```

```sql
CREATE INDEX idx_products_category
    ON olist_products_dataset (product_category_name);

COMMENT ON TABLE olist_products_dataset
    IS 'Product catalog. Join to category_translation on product_category_name for English names. 610 rows have NULL category.';



-- ╔══════════════════════════════════════════════════════════════════════╗
-- ║  TABLE 5: SELLERS                                                    ║
-- ║  Dimension: Geographic + Operational Friction (seller-side logistics)  ║
-- ║  3,095 rows — one row per unique seller                              ║
-- ╚══════════════════════════════════════════════════════════════════════╝

DROP TABLE IF EXISTS olist_sellers_dataset CASCADE;

CREATE TABLE olist_sellers_dataset (
    seller_id                   CHAR(32)        NOT NULL,
    seller_zip_code_prefix      INTEGER         NOT NULL,
    seller_city                 VARCHAR(50)     NOT NULL,
    seller_state                CHAR(2)         NOT NULL,

    CONSTRAINT pk_sellers
        PRIMARY KEY (seller_id)
);

CREATE INDEX idx_sellers_state
    ON olist_sellers_dataset (seller_state);

COMMENT ON TABLE olist_sellers_dataset
    IS 'Seller dimension. Use seller_zip_code_prefix to join to geolocation for seller-to-customer distance analysis.';



-- ╔══════════════════════════════════════════════════════════════════════╗
-- ║  TABLE 6: ORDERS (Central Fact Table)                                ║
-- ║  Dimension: Operational Friction ("Late Delivery" Factor)            ║
-- ║  99,441 rows — one row per order. FK to customers.                   ║
-- ║  Late delivery = order_delivered_customer_date > order_estimated_    ║
-- ║  delivery_date                                                       ║
-- ╚══════════════════════════════════════════════════════════════════════╝

DROP TABLE IF EXISTS olist_orders_dataset CASCADE;

CREATE TABLE olist_orders_dataset (
    order_id                    CHAR(32)        NOT NULL,
    customer_id                 CHAR(32)        NOT NULL,
    order_status                VARCHAR(15)     NOT NULL,
    order_purchase_timestamp    TIMESTAMP       NOT NULL,
    order_approved_at           TIMESTAMP,                  -- 160 NULLs
    order_delivered_carrier_date    TIMESTAMP,              -- 1,783 NULLs
    order_delivered_customer_date   TIMESTAMP,              -- 2,965 NULLs
    order_estimated_delivery_date   TIMESTAMP       NOT NULL,

    CONSTRAINT pk_orders
```

```sql
        PRIMARY KEY (order_id),

    CONSTRAINT fk_orders_customer
        FOREIGN KEY (customer_id)
        REFERENCES olist_customers_dataset (customer_id)
);

CREATE INDEX idx_orders_customer
    ON olist_orders_dataset (customer_id);

CREATE INDEX idx_orders_status
    ON olist_orders_dataset (order_status);

CREATE INDEX idx_orders_purchase_ts
    ON olist_orders_dataset (order_purchase_timestamp);

COMMENT ON TABLE olist_orders_dataset
    IS 'Central fact table. Compare order_delivered_customer_date vs order_estimated_delivery_date for late-delivery analysis.';



-- ╔══════════════════════════════════════════════════════════════════════╗
-- ║  TABLE 7: ORDER PAYMENTS                                             ║
-- ║  Dimension: Financial & Payment Behavior ("How They Buy")            ║
-- ║  103,886 rows — multiple payment rows per order (split payments).    ║
-- ║  Composite PK: (order_id, payment_sequential).                       ║
-- ╚══════════════════════════════════════════════════════════════════════╝

DROP TABLE IF EXISTS olist_order_payments_dataset CASCADE;

CREATE TABLE olist_order_payments_dataset (
    order_id                    CHAR(32)        NOT NULL,
    payment_sequential          SMALLINT        NOT NULL,
    payment_type                VARCHAR(20)     NOT NULL,   -- credit_card, boleto, voucher, debit_card, not_defined
    payment_installments        SMALLINT        NOT NULL,
    payment_value               NUMERIC(10,2)   NOT NULL,

    CONSTRAINT pk_order_payments
        PRIMARY KEY (order_id, payment_sequential),

    CONSTRAINT fk_payments_order
        FOREIGN KEY (order_id)
        REFERENCES olist_orders_dataset (order_id)
);

CREATE INDEX idx_payments_type
    ON olist_order_payments_dataset (payment_type);

COMMENT ON TABLE olist_order_payments_dataset
    IS 'Payment facts. Multiple rows per order for split payments. Use payment_type + payment_installments for credit behavior analysis.';



-- ╔══════════════════════════════════════════════════════════════════════╗
-- ║  TABLE 8: ORDER REVIEWS                                              ║
```

```sql
-- ╔═══════════════════════════════════════════════════════════════════════╗
-- ║  Dimension: Customer Experience & Sentiment ("Why They Left")         ║
-- ║  99,224 rows — review_id is NOT unique (814 dupes across orders).     ║
-- ║  Composite PK: (review_id, order_id) is unique across all 99,224 rows. ║
-- ╚═══════════════════════════════════════════════════════════════════════╝

DROP TABLE IF EXISTS olist_order_reviews_dataset CASCADE;

CREATE TABLE olist_order_reviews_dataset (
    review_id                       CHAR(32)        NOT NULL,
    order_id                        CHAR(32)        NOT NULL,
    review_score                    SMALLINT        NOT NULL,   -- 1 to 5
    review_comment_title            VARCHAR(50),                -- 87,656 NULLs
    review_comment_message          TEXT,                       -- 58,247 NULLs; free-text, variable length
    review_creation_date            TIMESTAMP       NOT NULL,
    review_answer_timestamp         TIMESTAMP       NOT NULL,

    CONSTRAINT pk_order_reviews
        PRIMARY KEY (review_id, order_id),

    CONSTRAINT fk_reviews_order
        FOREIGN KEY (order_id)
        REFERENCES olist_orders_dataset (order_id),

    CONSTRAINT chk_review_score
        CHECK (review_score BETWEEN 1 AND 5)
);

CREATE INDEX idx_reviews_order
    ON olist_order_reviews_dataset (order_id);

CREATE INDEX idx_reviews_score
    ON olist_order_reviews_dataset (review_score);

COMMENT ON TABLE olist_order_reviews_dataset
    IS 'Review sentiment. Join At-Risk customers to their last review_score for service-recovery targeting. review_score 1-2 = negative.';



-- ╔═══════════════════════════════════════════════════════════════════════╗
-- ║  TABLE 9: ORDER ITEMS (Granular Line-Item Fact Table)                 ║
-- ║  Dimension: Product Affinity + Price/Freight Sensitivity              ║
-- ║  112,650 rows — multiple items per order. Composite PK.               ║
-- ║  freight_value / price = freight ratio for bargain-hunter analysis.   ║
-- ╚═══════════════════════════════════════════════════════════════════════╝

DROP TABLE IF EXISTS olist_order_items_dataset CASCADE;

CREATE TABLE olist_order_items_dataset (
    order_id                        CHAR(32)        NOT NULL,
    order_item_id                   SMALLINT        NOT NULL,   -- sequential item number within order (1..n)
    product_id                      CHAR(32)        NOT NULL,
    seller_id                       CHAR(32)        NOT NULL,
    shipping_limit_date             TIMESTAMP       NOT NULL,
    price                           NUMERIC(8,2)    NOT NULL,   -- range: 0.85 – 6,735.00
```

```
    freight_value                 NUMERIC(8,2)    NOT NULL,   -- range: 0.00 – 409.68

    CONSTRAINT pk_order_items
        PRIMARY KEY (order_id, order_item_id),

    CONSTRAINT fk_items_order
        FOREIGN KEY (order_id)
        REFERENCES olist_orders_dataset (order_id),

    CONSTRAINT fk_items_product
        FOREIGN KEY (product_id)
        REFERENCES olist_products_dataset (product_id),

    CONSTRAINT fk_items_seller
        FOREIGN KEY (seller_id)
        REFERENCES olist_sellers_dataset (seller_id)
);

CREATE INDEX idx_items_product
    ON olist_order_items_dataset (product_id);

CREATE INDEX idx_items_seller
    ON olist_order_items_dataset (seller_id);

COMMENT ON TABLE olist_order_items_dataset
    IS 'Line-item facts. Use price + freight_value for bargain-hunter ratio. Join to products for category affinity.';


/*
================================================================================
  CSV IMPORT ORDER FOR pgAdmin (COPY command or Import/Export wizard)
================================================================================

  Execute imports in this EXACT sequence to satisfy all foreign key constraints:

  1. product_category_name_translation.csv   → product_category_name_translation
  2. olist_geolocation_dataset.csv           → olist_geolocation_dataset
     (omit the geolocation_id column – SERIAL auto-generates it)
  3. olist_customers_dataset.csv             → olist_customers_dataset
  4. olist_products_dataset.csv              → olist_products_dataset
  5. olist_sellers_dataset.csv               → olist_sellers_dataset
  6. olist_orders_dataset.csv                → olist_orders_dataset
  7. olist_order_payments_dataset.csv        → olist_order_payments_dataset
  8. olist_order_reviews_dataset.csv         → olist_order_reviews_dataset
  9. olist_order_items_dataset.csv           → olist_order_items_dataset

  IMPORTANT: Tables 1-5 have no FK dependencies and can technically be loaded
  in any order. Tables 6-9 MUST follow the sequence above because:
    • Orders depends on Customers (FK)
    • Payments, Reviews depend on Orders (FK)
    • Order Items depends on Orders, Products, AND Sellers (3 FKs)

  pgAdmin COPY example:
```

```
    ------------------------
    COPY product_category_name_translation
    FROM '/path/to/product_category_name_translation.csv'
    WITH (FORMAT csv, HEADER true, DELIMITER ',', QUOTE '"');

    For olist_geolocation_dataset, specify target columns to skip SERIAL PK:
    COPY olist_geolocation_dataset
         (geolocation_zip_code_prefix, geolocation_lat, geolocation_lng,
          geolocation_city, geolocation_state)
    FROM '/path/to/olist_geolocation_dataset.csv'
    WITH (FORMAT csv, HEADER true, DELIMITER ',', QUOTE '"');
================================================================================
*/
```