01_data_modeling_spreadsheets_to_schema.ipynb

```
#%%
from pathlib import Path
import pandas as pd
from pandas.api.types import is_numeric_dtype, is_datetime64_any_dtype

#%%
REPO_ROOT = Path.cwd()

DATA_DIR = (REPO_ROOT / "data") if (REPO_ROOT / "data").exists() else REPO_ROOT

OUTPUT_DIR = REPO_ROOT / "reports"
OUTPUT_DIR.mkdir(parents=True, exist_ok=True)

print("Data dir   :", DATA_DIR.resolve())
print("Output dir :", OUTPUT_DIR.resolve())

expected = {
    "Inventory_Snapshots_2024.csv",
    "Products_2024.csv",
    "Promotions_2024.csv",
    "Sales_2024_Q1.csv",
    "Sales_2024_Q2.csv",
    "Sales_2024_Q3.csv",
    "Sales_2024_Q4.csv",
}

present = {p.name for p in DATA_DIR.glob("*.csv")}
missing = expected - present
extra   = present - expected

print("\nAll files in data dir:")
for p in DATA_DIR.iterdir():
    print(" -", p.name)

print("\nCSV files found:", sorted(present))
print("Missing files   :", sorted(missing) if missing else "None ✅")
print("Unexpected extras:", sorted(extra) if extra else "None")

#%%
products = pd.read_csv(DATA_DIR / "Products_2024.csv", low_memory=False)

for c in ["product_id", "category_id", "vendor_id"]:
    if c in products.columns:
        products[c] = pd.to_numeric(products[c], errors="coerce").astype("Int64")

print("Products shape:", products.shape)
display(products.head(5))
print("\nDtypes:\n", products.dtypes)

if "product_id" in products.columns:
    dup_count = products.duplicated(subset=["product_id"]).sum()
    print(f"\nDuplicate product_id rows: {dup_count}")
else:
    print("\nNote: 'product_id' column not found — we'll confirm the schema in the next step.")

#%%
promotions = pd.read_csv(DATA_DIR / "Promotions_2024.csv", low_memory=False)

for c in ["start_date", "end_date"]:
    if c in promotions.columns:
        promotions[c] = pd.to_datetime(promotions[c], errors="coerce")

for c in ["promo_id", "category_id", "store_id"]:
    if c in promotions.columns:
        promotions[c] = pd.to_numeric(promotions[c], errors="coerce").astype("Int64")

print("Promotions shape:", promotions.shape)
display(promotions.head(5))
print("\nDtypes:\n", promotions.dtypes)
```

```
if {"start_date", "end_date"}.issubset(promotions.columns):
    bad_order = (promotions["start_date"] > promotions["end_date"]).sum()
    print(f"\nRows where start_date > end_date: {bad_order}")

if "discount_pct" in promotions.columns:
    out_of_range = ~((promotions["discount_pct"].between(0, 1)) | (promotions["discount_pct"].between(0, 100)))
    print(f"Rows with discount_pct outside [0-1] or [0-100]: {out_of_range.sum()}")

#%%
def load_sales_csv(path):
    df = pd.read_csv(path, low_memory=False)
    if "txn_date" not in df.columns and "date" in df.columns:
        df = df.rename(columns={"date": "txn_date"})
    if "txn_date" in df.columns:
        df["txn_date"] = pd.to_datetime(df["txn_date"], errors="coerce")

    for c, kind in {
        "txn_id": "Int64",
        "store_id": "Int64",
        "product_upc": "Int64",
        "product_id": "Int64",
        "qty": "Int64",
    }.items():
        if c in df.columns:
            df[c] = pd.to_numeric(df[c], errors="coerce").astype(kind)
    for c in ["unit_price", "line_amount"]:
        if c in df.columns:
            df[c] = pd.to_numeric(df[c], errors="coerce")
    return df

sales_q1 = load_sales_csv(DATA_DIR / "Sales_2024_Q1.csv")
sales_q2 = load_sales_csv(DATA_DIR / "Sales_2024_Q2.csv")
sales_q3 = load_sales_csv(DATA_DIR / "Sales_2024_Q3.csv")
sales_q4 = load_sales_csv(DATA_DIR / "Sales_2024_Q4.csv")

sales = pd.concat([sales_q1, sales_q2, sales_q3, sales_q4], ignore_index=True)

print("Sales shapes:", { "Q1": sales_q1.shape, "Q2": sales_q2.shape, "Q3": sales_q3.shape, "Q4": sales_q4.shape })
print("Sales (all):", sales.shape)
display(sales.head(5))
print("\nDtypes:\n", sales.dtypes)

neg_cols = [c for c in ["qty", "unit_price", "line_amount"] if c in sales.columns]
if neg_cols:
    neg_counts = {c: int((sales[c] < 0).sum()) for c in neg_cols}
    print("\nNegative values:", neg_counts)

if "txn_id" in sales.columns:
    dup_txn = int(sales.duplicated(subset=["txn_id"]).sum())
    print("Duplicate txn_id rows:", dup_txn)
else:
    subset = [c for c in ["store_id", "product_upc", "product_id", "txn_date"] if c in sales.columns]
    if subset:
        dup_combo = int(sales.duplicated(subset=subset).sum())
        print(f"Duplicate rows by {subset}:", dup_combo)

if all(c in sales.columns for c in ["qty", "unit_price", "line_amount"]):
    calc = sales["qty"] * sales["unit_price"]
    rel_err = (sales["line_amount"] - calc).abs() / calc.replace(0, pd.NA)
    bad = int((rel_err.fillna(0) > 0.01).sum())
    print("Rows where line_amount differs from qty*unit_price by >1%:", bad)

#%%
inv = pd.read_csv(DATA_DIR / "Inventory_Snapshots_2024.csv", low_memory=False)

if "snapshot_date" not in inv.columns:
    for alt in ["as_of_date", "date"]:
        if alt in inv.columns:
            inv = inv.rename(columns={alt: "snapshot_date"})
            break
inv["snapshot_date"] = pd.to_datetime(inv["snapshot_date"], errors="coerce")
```

```
for c, kind in {"store_id": "Int64", "product_upc": "Int64", "qty_on_hand": "Int64"}.items():
    if c in inv.columns:
        inv[c] = pd.to_numeric(inv[c], errors="coerce").astype(kind)

print("Inventory shape:", inv.shape)
display(inv.head(5))
print("\nDtypes:\n", inv.dtypes)

alerts = []

if "qty_on_hand" in inv.columns:
    neg = int((inv["qty_on_hand"] < 0).sum())
    alerts.append(("Negative qty_on_hand", neg))

subset = [c for c in ["store_id", "product_upc", "snapshot_date"] if c in inv.columns]
if len(subset) == 3:
    dup = int(inv.duplicated(subset=subset).sum())
    alerts.append((f"Duplicate rows by {subset}", dup))

if "snapshot_date" in inv.columns:
    not_2024 = int(inv["snapshot_date"].dropna().map(lambda d: d.year != 2024).sum())
    alerts.append(("Rows with snapshot_date not in 2024", not_2024))

if "product_upc" in inv.columns and "product_upc" in globals()["products"].columns:
    orphan = int(~inv["product_upc"].isin(products["product_upc"]).sum())
    orphan = int((~inv["product_upc"].isin(products["product_upc"])).sum())
    alerts.append(("Inventory product_upc not found in Products", orphan))

print("\nAlerts:")
for name, count in alerts:
    print(f" - {name}: {count}")

#%%
tables = {
    "Products": products,
    "Promotions": promotions,
    "Sales_Q1": sales_q1,
    "Sales_Q2": sales_q2,
    "Sales_Q3": sales_q3,
    "Sales_Q4": sales_q4,
    "Sales_All": sales,
    "Inventory": inv,
}

print("=== ROW COUNTS SUMMARY ===")
for name, df in tables.items():
    print(f"{name:12s} : {len(df):,} rows")

#%%
def data_dictionary(df: pd.DataFrame, table_name: str, max_allowed_values=15) -> pd.DataFrame:
    rows = []
    for col in df.columns:
        s = df[col]
        dtype = str(s.dtype)
        non_null = int(s.notna().sum())
        nulls = int(s.isna().sum())
        unique = int(s.nunique(dropna=True))
        sample = s.dropna().iloc[0] if non_null else None

        allowed_values = None
        if not is_datetime64_any_dtype(s) and not is_numeric_dtype(s) and unique <= max_allowed_values:
            allowed_values = ", ".join(map(str, sorted(s.dropna().unique())[:max_allowed_values]))

        rows.append({
            "table": table_name,
            "column": col,
            "dtype": dtype,
            "non_null": non_null,
            "nulls": nulls,
            "unique": unique,
            "example_value": sample,
```

```
            "allowed_values(sampled)": allowed_values,
            "suggested_description": ""
        })
    return pd.DataFrame(rows).sort_values(["table","column"])

dd_products   = data_dictionary(products,  "products")
dd_promos     = data_dictionary(promotions,"promotions")
dd_sales      = data_dictionary(sales,     "sales_all")
dd_inventory  = data_dictionary(inv,       "inventory")

data_dict = pd.concat([dd_products, dd_promos, dd_sales, dd_inventory], ignore_index=True)

print("Data Dictionary preview (first 25 rows):")
display(data_dict.head(25))

#%%
erd_text = """
```mermaid
erDiagram
  PRODUCTS {
    INT product_upc PK
    STRING product_name
    STRING brand
    STRING department_name
    STRING category_name
    STRING size
    STRING unit
    STRING vendor_name
    STRING vendor_phone
    FLOAT regular_price
    FLOAT unit_cost
    INT pack_size
  }

  SALES {
    STRING receipt_id PK
    INT    line_number PK
    DATETIME sale_datetime
    DATE   txn_date
    INT    store_id
    STRING cashier_name
    STRING tender_type
    STRING customer_segment
    INT    product_upc FK
    INT    qty
    FLOAT  unit_price
    FLOAT  line_amount
    FLOAT  unit_price_effective
    FLOAT  line_subtotal
    FLOAT  tax_amount
    BOOLEAN weekend_flag
    INT    promo_id
    STRING promo_type
  }

  INVENTORY_SNAPSHOT {
    DATE   snapshot_date PK
    INT    store_id PK
    INT    product_upc FK PK
    INT    on_hand_qty
    FLOAT  unit_cost
    FLOAT  inventory_cost_value
  }

  PROMOTIONS {
    INT    promo_id PK
    INT    product_upc FK
    STRING promo_type
    INT    discount_percent
    DATE   start_date
    DATE   end_date
  }
```
```

```
  STORE {
    INT    store_id PK
    STRING store_name
    STRING store_address
    STRING store_city
    STRING store_state
    INT    store_zip
  }

  DIM_DATE {
    DATE   d PK
    INT    y
    INT    q
    INT    m
    INT    dow
    BOOLEAN is_holiday
  }

  PRODUCTS ||--o{ SALES : "sold as"
  PRODUCTS ||--o{ INVENTORY_SNAPSHOT : "stocked as"
  PRODUCTS ||--o{ PROMOTIONS : "promoted as"

  PROMOTIONS ||--o{ SALES : "applied to"

  STORE ||--o{ SALES : "has"
  STORE ||--o{ INVENTORY_SNAPSHOT : "tracks"

  DIM_DATE ||--o{ SALES : "occurs on"
  DIM_DATE ||--o{ INVENTORY_SNAPSHOT : "snapshot on"
"""

out_path = Path("erd_mermaid.md")
out_path.write_text(erd_text, encoding="utf-8")

print(f"ERD Markdown file created at: {out_path.resolve()}")
print("\nPreview (first 10 lines):")
print("\n".join(erd_text.splitlines()[:10]))

#%%
ddl = """

BEGIN;

CREATE TABLE IF NOT EXISTS products (
  product_upc        BIGINT PRIMARY KEY,
  product_name       TEXT NOT NULL,
  brand              TEXT,
  department_name    TEXT,
  category_name      TEXT,
  size               TEXT,
  unit               TEXT,
  vendor_name        TEXT,
  vendor_phone       TEXT,
  regular_price      NUMERIC(10,2),
  unit_cost          NUMERIC(10,2),
  pack_size          INTEGER CHECK (pack_size IS NULL OR pack_size >= 0)
);

CREATE TABLE IF NOT EXISTS stores (
  store_id       INTEGER PRIMARY KEY,
  store_name     TEXT,
  store_address  TEXT,
  store_city     TEXT,
  store_state    VARCHAR(2),
  store_zip      INTEGER
);

CREATE TABLE IF NOT EXISTS dim_date (
  d           DATE PRIMARY KEY,
  y           INTEGER,
  q           INTEGER CHECK (q BETWEEN 1 AND 4),
```

```
    m           INTEGER CHECK (m BETWEEN 1 AND 12),
    dow         INTEGER CHECK (dow BETWEEN 0 AND 6),
    is_holiday  BOOLEAN
  );

  CREATE TABLE IF NOT EXISTS promotions (
    promo_id          INTEGER PRIMARY KEY,
    product_upc       BIGINT REFERENCES products(product_upc),
    promo_type        TEXT,
    discount_percent  NUMERIC(5,2) CHECK (discount_percent BETWEEN 0 AND 100),
    start_date        DATE NOT NULL,
    end_date          DATE NOT NULL,
    CHECK (start_date <= end_date)
  );

  CREATE TABLE IF NOT EXISTS sales_txn (
    receipt_id            TEXT NOT NULL,
    line_number           INTEGER NOT NULL,
    sale_datetime         TIMESTAMP,
    txn_date              DATE NOT NULL,
    store_id              INTEGER REFERENCES stores(store_id),
    cashier_name          TEXT,
    tender_type           TEXT,
    customer_segment      TEXT,
    product_upc           BIGINT REFERENCES products(product_upc),
    qty                   INTEGER NOT NULL CHECK (qty >= 0),
    unit_price            NUMERIC(10,2),
    line_amount           NUMERIC(12,2),
    unit_price_effective  NUMERIC(10,2),
    line_subtotal         NUMERIC(12,2),
    tax_amount            NUMERIC(12,2),
    weekend_flag          BOOLEAN,
    promo_id              INTEGER REFERENCES promotions(promo_id),
    promo_type            TEXT,
    PRIMARY KEY (receipt_id, line_number)
  );

  CREATE TABLE IF NOT EXISTS inventory_snapshot (
    snapshot_date         DATE NOT NULL,
    store_id              INTEGER NOT NULL REFERENCES stores(store_id),
    product_upc           BIGINT  NOT NULL REFERENCES products(product_upc),
    on_hand_qty           INTEGER CHECK (on_hand_qty IS NULL OR on_hand_qty >= 0),
    unit_cost             NUMERIC(10,2),
    inventory_cost_value  NUMERIC(12,2),
    PRIMARY KEY (snapshot_date, store_id, product_upc)
  );

  CREATE INDEX IF NOT EXISTS idx_sales_txn_date       ON sales_txn (txn_date);
  CREATE INDEX IF NOT EXISTS idx_sales_store          ON sales_txn (store_id);
  CREATE INDEX IF NOT EXISTS idx_sales_product        ON sales_txn (product_upc);
  CREATE INDEX IF NOT EXISTS idx_sales_store_date     ON sales_txn (store_id, txn_date);

  CREATE INDEX IF NOT EXISTS idx_promos_product_range ON promotions (product_upc, start_date, end_date);

  CREATE INDEX IF NOT EXISTS idx_inv_store_prod_date  ON inventory_snapshot (store_id, product_upc, snapshot_date);

  COMMIT;
  """

  out = Path("ddl_postgres.sql")
  out.write_text(ddl, encoding="utf-8")
  print(f"Wrote normalized schema DDL to: {out.resolve()}")

  #%%
```