```python
import os
import uuid
from datetime import datetime

import pandas as pd
from sqlalchemy import create_engine, text

# ----------------------------------------------------------------------
# DB connection helper — same pattern as load_openflights & synth_flights
# ----------------------------------------------------------------------
def get_db_url() -> str:
    """
    Resolve the database URL from environment variables.

    Prefer DATABASE_URL (what you're using now), but fall back
    to AIRLINE_DB_DSN if it's present.
    """
    url = os.getenv("DATABASE_URL") or os.getenv("AIRLINE_DB_DSN")
    if not url:
        raise RuntimeError(
            "Set either DATABASE_URL or AIRLINE_DB_DSN in your environment / .env.\n"
            "Example: postgresql+psycopg2://postgres:password@localhost:5432/airline_bi"
        )
    return url


ENGINE = create_engine(get_db_url(), future=True, pool_pre_ping=True)


RAW_PATH = "data/bts_cleaned.csv"
CHUNK = 200_000

def normalize_chunk(df: pd.DataFrame) -> pd.DataFrame:
    """
    Normalize a BTS aggregate chunk into the schema expected by airline.flight_performance.

    Supports two column naming schemes:

    1) Original aggregate (what the script used to expect):
       year, month, carrier, airport, arr_flights, arr_del15, arr_cancelled,
       arr_diverted, arr_delay, carrier_delay, weather_delay, nas_delay,
       security_delay, late_aircraft_delay

    2) Already-final names (often from bts_cleaned.csv):
       year, month, airline_iata, airport_iata, arrivals, arrivals_delayed_15min,
       arr_cancelled, arr_diverted, total_arrival_delay_min, carrier_delay,
       weather_delay, nas_delay, security_delay, late_aircraft_delay
    """

    cols = set(df.columns)

    # Case 1: old names — map them to final names
    if {"carrier", "airport", "arr_flights", "arr_del15", "arr_delay"}.issubset(cols):
        df = df.rename(
            columns={
                "carrier": "airline_iata",
                "airport": "airport_iata",
                "arr_flights": "arrivals",
                "arr_del15": "arrivals_delayed_15min",
                "arr_delay": "total_arrival_delay_min",
            }
        )

    # Case 2: already-final names — nothing to rename
    elif {
        "year",
        "month",
        "airline_iata",
        "airport_iata",
        "arrivals",
        "arrivals_delayed_15min",
        "arr_cancelled",
        "arr_diverted",
        "total_arrival_delay_min",
        "carrier_delay",
        "weather_delay",
        "nas_delay",
        "security_delay",
        "late_aircraft_delay",
    }.issubset(cols):
        pass

    else:
        raise KeyError(
            f"Unexpected BTS columns: {sorted(df.columns.tolist())}. "
            "Expected either ['carrier','airport','arr_flights','arr_del15','arr_delay',
...] "
            "or ['airline_iata','airport_iata','arrivals','arrivals_delayed_15min', ...]."
        )

    # Now select the canonical columns
    keep = [
        "year",
        "month",
        "airline_iata",
        "airport_iata",
        "arrivals",
        "arrivals_delayed_15min",
        "arr_cancelled",
        "arr_diverted",
        "total_arrival_delay_min",
        "carrier_delay",
        "weather_delay",
        "nas_delay",
        "security_delay",
        "late_aircraft_delay",
    ]

    df = df[keep].copy()

    # Clean IATA codes
    df["airline_iata"] = df["airline_iata"].astype(str).str.strip().str.upper()
    df["airport_iata"] = df["airport_iata"].astype(str).str.strip().str.upper()

    # Ensure numeric columns are numeric (fill NaN with 0)
    numeric_cols = [
        "arrivals",
        "arrivals_delayed_15min",
        "arr_cancelled",
        "arr_diverted",
        "total_arrival_delay_min",
        "carrier_delay",
        "weather_delay",
        "nas_delay",
        "security_delay",
        "late_aircraft_delay",
    ]
    for col in numeric_cols:
        df[col] = pd.to_numeric(df[col], errors="coerce").fillna(0)
    # Build a snapshot_id that matches your flight_performance PK
    df["snapshot_id"] = (
        df["year"].astype(int).astype(str).str.zfill(4)
        + "-"
        + df["month"].astype(int).astype(str).str.zfill(2)
        + "-"
        + df["airline_iata"]
        + "-"
        + df["airport_iata"]
    )

    # Order columns to match the INSERT INTO tmp_fp
    ordered = [
        "snapshot_id",
        "year",
        "month",
        "airline_iata",
        "airport_iata",
        "arrivals",
        "arrivals_delayed_15min",
        "arr_cancelled",
        "arr_diverted",
        "total_arrival_delay_min",
        "carrier_delay",
        "weather_delay",
        "nas_delay",
        "security_delay",
        "late_aircraft_delay",
    ]
    return df[ordered]


def ensure_table():
    with ENGINE.begin() as con:
        con.execute(text("""SET search_path TO airline, public;"""))
        con.execute(text("""
        CREATE TABLE IF NOT EXISTS airline.flight_performance (
            snapshot_id TEXT PRIMARY KEY,
            year INT NOT NULL,
            month INT NOT NULL,
            airline_iata VARCHAR(5) NOT NULL,
            airport_iata VARCHAR(5) NOT NULL,
            arrivals INT,
            arrivals_delayed_15min INT,
            arr_cancelled INT,
            arr_diverted INT,
            total_arrival_delay_min DOUBLE PRECISION,
            carrier_delay DOUBLE PRECISION,
            weather_delay DOUBLE PRECISION,
            nas_delay DOUBLE PRECISION,
            security_delay DOUBLE PRECISION,
            late_aircraft_delay DOUBLE PRECISION,
            CONSTRAINT uq_fp UNIQUE (year, month, airline_iata, airport_iata),
            CONSTRAINT fk_fp_airline  FOREIGN KEY (airline_iata)  REFERENCES
airline.airlines(iata_code),
            CONSTRAINT fk_fp_airport  FOREIGN KEY (airport_iata)  REFERENCES
airline.airports(iata_code)
        );
        CREATE INDEX IF NOT EXISTS idx_fp_month ON airline.flight_performance (year, month);
        CREATE INDEX IF NOT EXISTS idx_fp_airline ON airline.flight_performance
(airline_iata);
        CREATE INDEX IF NOT EXISTS idx_fp_airport ON airline.flight_performance
(airport_iata);
        """))
```

```
def load():
    ensure_table()

    reader = pd.read_csv(RAW_PATH, chunksize=CHUNK)
    for i, chunk in enumerate(reader, start=1):
        df = normalize_chunk(chunk)

        with ENGINE.begin() as con:
            # temp table for conflict-safe upsert
            con.execute(text("""
                CREATE TEMP TABLE tmp_fp(
                    snapshot_id TEXT,
                    year INT,
                    month INT,
                    airline_iata VARCHAR(5),
                    airport_iata VARCHAR(5),
                    arrivals INT,
                    arrivals_delayed_15min INT,
                    arr_cancelled INT,
                    arr_diverted INT,
                    total_arrival_delay_min DOUBLE PRECISION,
                    carrier_delay DOUBLE PRECISION,
                    weather_delay DOUBLE PRECISION,
                    nas_delay DOUBLE PRECISION,
                    security_delay DOUBLE PRECISION,
                    late_aircraft_delay DOUBLE PRECISION
                ) ON COMMIT DROP;
            """))
            df.to_sql("tmp_fp", con, if_exists="append", index=False)

            con.execute(text("""
                INSERT INTO airline.flight_performance AS fp(
                    snapshot_id, year, month, airline_iata, airport_iata,
                    arrivals, arrivals_delayed_15min, arr_cancelled, arr_diverted,
                    total_arrival_delay_min, carrier_delay, weather_delay, nas_delay,
security_delay, late_aircraft_delay
                )
                SELECT
                    t.snapshot_id, t.year, t.month, t.airline_iata, t.airport_iata,
                    t.arrivals, t.arrivals_delayed_15min, t.arr_cancelled, t.arr_diverted,
                    t.total_arrival_delay_min, t.carrier_delay, t.weather_delay,
                    t.nas_delay, t.security_delay, t.late_aircraft_delay
                FROM tmp_fp t
                JOIN airline.airports a
                  ON t.airport_iata = a.iata_code
                ON CONFLICT (snapshot_id) DO NOTHING;
            """))

    print(f"Chunk {i} inserted.")

print("✅ BTS performance load complete.")

if __name__ == "__main__":
    load()
```