```
"""
Backfill dimension / audit tables derived from existing flight data.

This script:
  1) Populates airline.routes from distinct (airline_id, origin_airport_id,
destination_airport_id)
  2) Optionally computes distance_nm using airport latitude / longitude
  3) Seeds airline.aircraft with a small synthetic global fleet
  4) Assigns aircraft_id to flights
  5) Creates synthetic airline.flight_changes records for a subset of flights

Schema assumptions (from the DB):

  airline.flights
    flight_id               bigint PK
    airline_id              bigint
    route_id                bigint (nullable)
    aircraft_id             bigint (nullable)
    origin_airport_id       bigint
    destination_airport_id  bigint
    flight_number           text
    flight_date             date
    scheduled_departure_utc timestamp
    scheduled_arrival_utc   timestamp
    actual_departure_utc    timestamp
    actual_arrival_utc      timestamp
    delay_minutes           integer
    delay_cause             text
    status                  enum (flight_status)

  airline.routes
    route_id                bigint PK
    airline_id              bigint
    origin_airport_id       bigint
    destination_airport_id  bigint
    distance_nm             integer (nullable)

  airline.airports
    airport_id              bigint
    iata_code               varchar
    icao_code               varchar
    name                    text
    city                    text
    country                 text
    latitude                numeric   -- degrees
    longitude               numeric   -- degrees
    timezone                text

  airline.aircraft
    aircraft_id             bigint PK
    manufacturer            text
    model                   text
    seat_capacity           integer
    tail_number             text (nullable)

  airline.flight_changes
    change_id               bigint PK
    flight_id               bigint FK -> flights
    old_aircraft_id         bigint
    new_aircraft_id         bigint
    reason                  text
    changed_at              timestamp
"""
```

```python
import os
from sqlalchemy import create_engine, text


def get_db_url() -> str:
    url = os.getenv("DATABASE_URL") or os.getenv("AIRLINE_DB_DSN")
    if not url:
        raise RuntimeError(
            "Set DATABASE_URL or AIRLINE_DB_DSN in your environment.\n"
            "Example: postgresql+psycopg2://postgres:gpcool@localhost:5432/airline_bi"
        )
    return url


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


# ---------------------------------------------------------------------
# 1. ROUTES
# ---------------------------------------------------------------------

def backfill_routes_from_flights(compute_distance: bool = False) -> None:
    """Insert missing routes based on distinct flights."""

    print("🔹 Backfilling airline.routes from airline.flights ...")

    insert_sql = text(
        """
        INSERT INTO airline.routes (
            airline_id,
            origin_airport_id,
            destination_airport_id,
            distance_nm
        )
        SELECT DISTINCT
            f.airline_id,
            f.origin_airport_id,
            f.destination_airport_id,
            NULL::integer AS distance_nm
        FROM airline.flights f
        LEFT JOIN airline.routes r
          ON r.airline_id          = f.airline_id
         AND r.origin_airport_id   = f.origin_airport_id
         AND r.destination_airport_id = f.destination_airport_id
        WHERE r.route_id IS NULL
          AND f.airline_id IS NOT NULL
          AND f.origin_airport_id IS NOT NULL
          AND f.destination_airport_id IS NOT NULL;
        """
    )

    with ENGINE.begin() as con:
        result = con.execute(insert_sql)
        print(f"  → Inserted {result.rowcount or 0} route rows")

    if compute_distance:
        compute_route_distances()


def compute_route_distances() -> None:
    """
    Compute approximate great-circle distance in nautical miles for each route.

    Uses airline.airports.latitude / longitude (degrees).
    """
    print("🔹 Computing distance_nm for airline.routes ...")

    update_sql = text(
        """
        UPDATE airline.routes r
        SET distance_nm = sub.distance_nm::integer
        FROM (
            SELECT
                r2.route_id,
                (
                    2 * 6371 * asin(
                        sqrt(
                            sin(radians(ad.latitude - ao.latitude) / 2)^2 +
                            cos(radians(ao.latitude)) *
                            cos(radians(ad.latitude)) *
                            sin(radians(ad.longitude - ao.longitude) / 2)^2
                        )
                    ) / 1.852
                ) AS distance_nm
            FROM airline.routes r2
            JOIN airline.airports ao
              ON ao.airport_id = r2.origin_airport_id
            JOIN airline.airports ad
              ON ad.airport_id = r2.destination_airport_id
            WHERE r2.distance_nm IS NULL
              AND ao.latitude IS NOT NULL
              AND ao.longitude IS NOT NULL
              AND ad.latitude IS NOT NULL
              AND ad.longitude IS NOT NULL
        ) AS sub
        WHERE r.route_id = sub.route_id;
        """
    )

    with ENGINE.begin() as con:
        result = con.execute(update_sql)
        print(f"  → Updated distance_nm for {result.rowcount or 0} routes")


def backfill_route_ids_on_flights() -> None:
    """Update airline.flights.route_id to match airline.routes."""

    print("🔹 Backfilling flights.route_id from routes ...")

    update_sql = text(
        """
        UPDATE airline.flights f
        SET route_id = r.route_id
        FROM airline.routes r
        WHERE f.route_id IS NULL
          AND f.airline_id        = r.airline_id
          AND f.origin_airport_id = r.origin_airport_id
          AND f.destination_airport_id = r.destination_airport_id;
        """
    )

    with ENGINE.begin() as con:
        result = con.execute(update_sql)
        print(f"  → Updated route_id on {result.rowcount or 0} flights")


# -------------------------------------------------------------------
# 2. AIRCRAFT + FLIGHT.AIRCRAFT_ID
# -------------------------------------------------------------------

def backfill_aircraft() -> None:
    """
    Seed airline.aircraft with a small synthetic global fleet.

    Schema:
      aircraft_id   bigserial PK
      manufacturer  text
      model         text
      seat_capacity integer
      tail_number   text (nullable)
    """

    print("🔷 Backfilling airline.aircraft with synthetic fleet ...")

    insert_sql = text(
        """
        -- Only seed if table is currently empty
        INSERT INTO airline.aircraft (manufacturer, model, seat_capacity, tail_number)
        SELECT manufacturer, model, seat_capacity, tail_number
        FROM (
            VALUES
                ('Airbus', 'A320',      150, NULL),
                ('Boeing', '737-800',   165, NULL),
                ('Airbus', 'A321',      185, NULL),
                ('Boeing', '787-8',     242, NULL),
                ('Airbus', 'A350-900',  300, NULL)
        ) AS v(manufacturer, model, seat_capacity, tail_number)
        WHERE NOT EXISTS (SELECT 1 FROM airline.aircraft);
        """
    )

    with ENGINE.begin() as con:
        result = con.execute(insert_sql)
        print(f"  → Inserted {result.rowcount or 0} aircraft rows")


def assign_aircraft_to_flights() -> None:
    """
    Assign an aircraft_id to each flight, picking randomly from all aircraft.
    (aircraft is not tied to a specific airline in your schema.)
    """

    print("🔷 Assigning aircraft_id to flights ...")

    update_sql = text(
        """
        WITH choices AS (
            SELECT
                f.flight_id,
                (
                  SELECT ac2.aircraft_id
                  FROM airline.aircraft ac2
                  ORDER BY random()
                  LIMIT 1
                ) AS aircraft_id
            FROM airline.flights f
            WHERE f.aircraft_id IS NULL
        )
        UPDATE airline.flights f
        SET aircraft_id = c.aircraft_id
        FROM choices c
        WHERE f.flight_id = c.flight_id;
        """
    )

    with ENGINE.begin() as con:
        result = con.execute(update_sql)
        print(f"  → Updated aircraft_id on {result.rowcount or 0} flights")


# ------------------------------------------------------------------
# 3. FLIGHT_CHANGES
# ------------------------------------------------------------------

def generate_flight_changes(change_fraction: float = 0.05) -> None:
    """
    Create synthetic aircraft change events for a subset of flights.

    change_fraction: approximate fraction of flights to get a change row.
    """

    print(f"🔷 Generating flight_changes for ~{change_fraction*100:.1f}% of flights ...")

    insert_sql = text(
        f"""
        WITH candidate_flights AS (
            SELECT f.flight_id, f.aircraft_id, f.flight_date
            FROM airline.flights f
            WHERE f.aircraft_id IS NOT NULL
        ),
        sampled AS (
            SELECT *
            FROM candidate_flights
            WHERE random() < {change_fraction}
        ),
        new_aircraft AS (
            SELECT
                s.flight_id,
                s.aircraft_id AS original_aircraft_id,
                s.flight_date,
                (
                  SELECT ac2.aircraft_id
                  FROM airline.aircraft ac2
                  WHERE ac2.aircraft_id <> s.aircraft_id
                  ORDER BY random()
                  LIMIT 1
                ) AS new_aircraft_id
            FROM sampled s
        )
        INSERT INTO airline.flight_changes (
            flight_id,
            old_aircraft_id,
            new_aircraft_id,
            reason,
            changed_at
        )
        SELECT
            n.flight_id,
            n.original_aircraft_id,
            n.new_aircraft_id,
            (ARRAY['Maintenance','Crew Reassignment','Operational','Equipment Downgrade'])[1
+ floor(random()*4)]::text,
            n.flight_date::timestamp - INTERVAL '2 hours' + (random() * INTERVAL '90
minutes')
        FROM new_aircraft n
        WHERE n.new_aircraft_id IS NOT NULL;
        """
    )

    with ENGINE.begin() as con:
        result = con.execute(insert_sql)
        print(f"  → Inserted {result.rowcount or 0} flight_changes rows")

    # Optionally: update flights.aircraft_id to reflect the "new" aircraft
    update_sql = text(
        """
        UPDATE airline.flights f
        SET aircraft_id = fc.new_aircraft_id
        FROM airline.flight_changes fc
        WHERE fc.flight_id = f.flight_id;
        """
    )

    with ENGINE.begin() as con:
        result = con.execute(update_sql)
        print(f"  → Updated aircraft_id on {result.rowcount or 0} flights to match
flight_changes")


# -------------------------------------------------------------------
# MAIN
# -------------------------------------------------------------------

def run():
    print("=== Backfill derived tables from flights ===")
    backfill_routes_from_flights(compute_distance=False)  # set True if you want distances
computed
    backfill_route_ids_on_flights()
    backfill_aircraft()
    assign_aircraft_to_flights()
    generate_flight_changes(change_fraction=0.05)
    print("=== Done. ✅ ===")


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