```
import os
import random
from datetime import datetime, date, timedelta

from sqlalchemy import create_engine, text


# ---------- DB CONNECTION HELPERS ----------

def get_db_url() -> str:
    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)


# ---------- FETCH LOOKUP DATA ----------

def fetch_airports_and_airlines(conn):
    airlines = conn.execute(
        text(
            """
            SELECT airline_id, iata_code
            FROM airline.airlines
            WHERE iata_code IS NOT NULL
            """
        )
    ).mappings().all()

    if not airlines:
        raise RuntimeError(
            "No rows found in airline.airlines. "
            "Run etl/load_openflights.py first."
        )

    airports = conn.execute(
        text(
            """
            SELECT airport_id, iata_code
            FROM airline.airports
            WHERE iata_code IS NOT NULL
            """
        )
    ).mappings().all()

    if not airports:
        raise RuntimeError(
            "No rows found in airline.airports. "
            "Run etl/load_openflights.py first."
        )

    return airlines, airports


def fetch_flight_status_values(conn):
    """
    Figure out the ENUM type used by flights.status and get its labels.
```

```
    This avoids hard-coding enum names like 'Completed' vs 'completed'.
    """
    type_name = conn.execute(
        text(
            """
            SELECT udt_name
            FROM information_schema.columns
            WHERE table_schema = 'airline'
              AND table_name   = 'flights'
              AND column_name  = 'status'
            """
        )
    ).scalar_one()

    rows = conn.execute(
        text(
            """
            SELECT e.enumlabel
            FROM pg_enum e
            JOIN pg_type t ON e.enumtypid = t.oid
            WHERE t.typname = :tname
            ORDER BY e.enumsortorder
            """
        ),
        {"tname": type_name},
    ).all()

    statuses = [r[0] for r in rows]
    if not statuses:
        raise RuntimeError("Could not read enum labels for flights.status.")
    return statuses


# ---------- GENERATE SYNTHETIC FLIGHTS ----------

def generate_flights(airlines, airports, statuses, n=5000, seed=42):
    random.seed(seed)

    flights = []

    now = datetime.utcnow()
    start_date = date(now.year - 1, 1, 1)
    end_date = date(now.year + 1, 12, 31)
    total_days = (end_date - start_date).days

    for _ in range(n):
        al = random.choice(airlines)
        origin, dest = random.sample(airports, 2)

        # Date + basic schedule
        day_offset = random.randrange(total_days)
        flight_date = start_date + timedelta(days=day_offset)

        dep_hour = random.randint(5, 22)
        dep_min = random.choice([0, 15, 30, 45])
        dep_dt = datetime.combine(flight_date, datetime.min.time()) + timedelta(
            hours=dep_hour, minutes=dep_min
        )

        block_minutes = random.randint(60, 6 * 60)  # 1–6 hours
        arr_dt = dep_dt + timedelta(minutes=block_minutes)

        status = random.choice(statuses)
        status_lower = status.lower()
```

```
        # Delay / actual times logic
        if status_lower.startswith("cancel"):
            delay = random.randint(60, 300)
            delay_cause = "Cancellation"
            actual_dep = None
            actual_arr = None
        elif status_lower.startswith("sched"):
            delay = 0
            delay_cause = None
            actual_dep = None
            actual_arr = None
        else:
            delay = random.randint(0, 180)
            delay_cause = random.choice(
                ["Weather", "Crew", "Maintenance", "ATC", "Late inbound", None]
            )
            actual_dep = dep_dt + timedelta(minutes=delay)
            actual_arr = arr_dt + timedelta(minutes=delay)

        flight_number = f"{al['iata_code']}{random.randint(1, 9999):04d}"

        flights.append(
            {
                "airline_id": al["airline_id"],
                "origin_airport_id": origin["airport_id"],
                "destination_airport_id": dest["airport_id"],
                "flight_number": flight_number,
                "flight_date": flight_date,
                "scheduled_departure_utc": dep_dt,
                "scheduled_arrival_utc": arr_dt,
                "actual_departure_utc": actual_dep,
                "actual_arrival_utc": actual_arr,
                "delay_minutes": delay,
                "delay_cause": delay_cause,
                "status": status,
            }
        )

    return flights


def insert_flights(conn, flights):
    if not flights:
        print("⚠️ No flights to insert.")
        return

    conn.execute(
        text(
            """
            INSERT INTO airline.flights (
                airline_id,
                origin_airport_id,
                destination_airport_id,
                flight_number,
                flight_date,
                scheduled_departure_utc,
                scheduled_arrival_utc,
                actual_departure_utc,
                actual_arrival_utc,
                delay_minutes,
                delay_cause,
                status
            )
            VALUES (
                :airline_id,
```

```
                :origin_airport_id,
                :destination_airport_id,
                :flight_number,
                :flight_date,
                :scheduled_departure_utc,
                :scheduled_arrival_utc,
                :actual_departure_utc,
                :actual_arrival_utc,
                :delay_minutes,
                :delay_cause,
                :status
            );
            """
        ),
        flights,
    )


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

def main():
    with ENGINE.begin() as conn:
        cols = conn.execute(
            text(
                """
                SELECT column_name
                FROM information_schema.columns
                WHERE table_schema = 'airline'
                  AND table_name   = 'flights'
                ORDER BY ordinal_position
                """
            )
        ).fetchall()
        print("🔍 flights columns:", [c[0] for c in cols])

        airlines, airports = fetch_airports_and_airlines(conn)
        print(f"🌍 Using {len(airports)} airports and {len(airlines)} airlines.")

        statuses = fetch_flight_status_values(conn)
        print("📊 flight_status enum values:", statuses)

        flights = generate_flights(airlines, airports, statuses, n=5000)
        print(f"✈️ Generated {len(flights)} synthetic flights.")

        insert_flights(conn, flights)
        print("✅ Synthetic flights inserted.")


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