```
"""
Synthetic revenue generator for the Airline BI database.

- Creates synthetic bookings for existing passengers & flights
- Creates one payment per booking
- Respects UNIQUE (passenger_id, flight_id) on airline.bookings

Schema assumptions:

  airline.bookings
    - booking_id        BIGSERIAL PRIMARY KEY
    - passenger_id      BIGINT (FK -> passengers.passenger_id)
    - flight_id         BIGINT (FK -> flights.flight_id)
    - booking_date      TIMESTAMP WITHOUT TIME ZONE
    - fare_class        TEXT
    - base_price_usd    NUMERIC(10,2)
    - booking_channel   TEXT
    - UNIQUE (passenger_id, flight_id)

  airline.payments
    - payment_id        BIGSERIAL PRIMARY KEY
    - booking_id        BIGINT (FK -> bookings.booking_id)
    - amount_usd        NUMERIC(10,2)
    - method            airline.payment_method
    - status            airline.payment_status
    - paid_at           TIMESTAMP WITHOUT TIME ZONE NOT NULL
"""

import os
import random
from decimal import Decimal
from datetime import timedelta

from dotenv import load_dotenv
from sqlalchemy import create_engine, text
from faker import Faker

# ---------------------------------------------------------------------------
# Configuration
# ---------------------------------------------------------------------------

TARGET_NEW_BOOKINGS = 20_000  # try to generate this many new bookings

FARE_CLASSES = ["Basic", "Standard", "Flexible", "Business", "First"]
BOOKING_CHANNELS = ["Web", "Mobile", "Call Center", "Travel Agent"]

PAYMENT_METHODS = ["Card", "Points", "Cash", "Voucher"]
PAYMENT_STATUSES = ["Authorized", "Captured", "Refunded", "Failed"]

fake = Faker()


def get_db_url() -> str:
    """
    Get the SQLAlchemy DB URL from environment.

    Prefer DATABASE_URL (your current .env), fall back to AIRLINE_DB_DSN.
    """
    load_dotenv()
    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)


# ---------------------------------------------------------------------------
# Helpers
# ---------------------------------------------------------------------------

def money(x: float) -> Decimal:
    """Round to 2 decimal places and return Decimal."""
    return Decimal(f"{x:.2f}")


def fetch_passengers_and_flights(con):
    """Return lists of passenger_ids and flight_ids."""
    passenger_ids = [
        row[0]
        for row in con.execute(
            text("SELECT passenger_id FROM airline.passengers ORDER BY passenger_id")
        )
    ]
    flight_ids = [
        row[0]
        for row in con.execute(
            text("SELECT flight_id FROM airline.flights ORDER BY flight_id")
        )
    ]

    if not passenger_ids:
        raise RuntimeError("No rows found in airline.passengers.")
    if not flight_ids:
        raise RuntimeError("No rows found in airline.flights.")

    print(f"👥 Found {len(passenger_ids)} passengers.")
    print(f"✈️  Found {len(flight_ids)} flights.")
    return passenger_ids, flight_ids


def fetch_existing_booking_pairs(con):
    """
    Load existing (passenger_id, flight_id) pairs from airline.bookings
    so we don't violate UNIQUE (passenger_id, flight_id).
    """
    rows = con.execute(
        text("SELECT passenger_id, flight_id FROM airline.bookings")
    ).fetchall()
    existing = {(int(r[0]), int(r[1])) for r in rows}
    print(f"🔗 Existing booking pairs (passenger, flight): {len(existing)}")
    return existing


def generate_booking_payloads(passenger_ids, flight_ids, n_bookings: int, used_pairs: set):
    """
    Generate payload dictionaries for airline.bookings WITHOUT booking_id.

    Args:
        passenger_ids: list[int]
        flight_ids: list[int]
        n_bookings: desired number of new bookings
        used_pairs: set of (passenger_id, flight_id) already present
```

```
    Returns:
        bookings_payload: list[dict] ready for INSERT (no booking_id)
    """
    bookings = []

    print(f"🧾 Generating up to {n_bookings} synthetic bookings (unique per
passenger/flight)...")
    attempts = 0
    max_attempts = n_bookings * 10  # generous upper bound

    while len(bookings) < n_bookings and attempts < max_attempts:
        attempts += 1

        passenger_id = random.choice(passenger_ids)
        flight_id = random.choice(flight_ids)
        key = (passenger_id, flight_id)

        # Avoid violating UNIQUE (passenger_id, flight_id)
        if key in used_pairs:
            continue
        used_pairs.add(key)

        booking_date = fake.date_time_between(start_date="-9M", end_date="+3M")

        fare_class = random.choices(
            FARE_CLASSES, weights=[0.35, 0.30, 0.20, 0.10, 0.05]
        )[0]
        channel = random.choices(
            BOOKING_CHANNELS, weights=[0.55, 0.25, 0.10, 0.10]
        )[0]

        # Base price ~ 80–900 with some long tail
        base_price = money(random.lognormvariate(4.5, 0.5))
        base_price = max(money(80), min(base_price, money(900)))

        bookings.append(
            {
                "passenger_id": passenger_id,
                "flight_id": flight_id,
                "booking_date": booking_date,
                "fare_class": fare_class,
                "base_price_usd": base_price,
                "booking_channel": channel,
            }
        )

    if len(bookings) < n_bookings:
        print(
            f"⚠️ Only generated {len(bookings)} unique booking pairs "
            f"out of requested {n_bookings} (attempts={attempts})."
        )

    print(f"✅ Prepared {len(bookings)} booking payloads.")
    return bookings


def insert_bookings_and_return(con, booking_payloads):
    """
    Insert into airline.bookings (letting Postgres assign booking_id)
    and then SELECT the newly inserted rows using booking_id > max_before.
    """
    if not booking_payloads:
        print("ℹ️ No bookings to insert.")
```

```
        return []

    # baseline max booking_id
    max_before = con.execute(
        text("SELECT COALESCE(MAX(booking_id), 0) FROM airline.bookings")
    ).scalar_one()

    # bulk insert WITHOUT RETURNING to avoid SQLAlchemy/psycopg2 weirdness
    con.execute(
        text(
            """
            INSERT INTO airline.bookings (
                passenger_id,
                flight_id,
                booking_date,
                fare_class,
                base_price_usd,
                booking_channel
            )
            VALUES (
                :passenger_id,
                :flight_id,
                :booking_date,
                :fare_class,
                :base_price_usd,
                :booking_channel
            );
            """
        ),
        booking_payloads,
    )

    # now pull back just the new rows
    result = con.execute(
        text(
            """
            SELECT booking_id,
                   passenger_id,
                   flight_id,
                   booking_date,
                   fare_class,
                   base_price_usd,
                   booking_channel
            FROM airline.bookings
            WHERE booking_id > :max_before
            ORDER BY booking_id;
            """
        ),
        {"max_before": max_before},
    )

    rows = result.mappings().all()
    print(f"✅ New bookings inserted: {len(rows)}")
    return rows


def build_payments_from_bookings(inserted_bookings):
    """
    Given rows returned from insert_bookings_and_return, build matching
    payments payloads.

    Each payment:
      - amount_usd ~ base_price_usd * [0.9, 1.15]
      - paid_at is always NON-NULL (some time after booking_date)
    """
```

```
    payments = []

    for row in inserted_bookings:
        booking_id = row["booking_id"]
        booking_date = row["booking_date"]
        base_price = row["base_price_usd"]

        method = random.choices(
            PAYMENT_METHODS,
            weights=[0.7, 0.1, 0.1, 0.1],
        )[0]
        status = random.choices(
            PAYMENT_STATUSES,
            weights=[0.65, 0.15, 0.10, 0.10],
        )[0]

        multiplier = random.uniform(0.9, 1.15)
        amount = money(float(base_price) * multiplier)

        # paid_at: always non-null
        offset_minutes = random.randint(0, 60 * 24)
        paid_at = booking_date + timedelta(minutes=offset_minutes)

        payments.append(
            {
                "booking_id": booking_id,
                "amount_usd": amount,
                "method": method,
                "status": status,
                "paid_at": paid_at,
            }
        )

    print(f"💳 Prepared {len(payments)} payments.")
    return payments


def insert_payments(con, payments):
    if not payments:
        print("ℹ️ No payments to insert.")
        return 0

    con.execute(
        text(
            """
            INSERT INTO airline.payments (
                booking_id,
                amount_usd,
                method,
                status,
                paid_at
            )
            VALUES (
                :booking_id,
                :amount_usd,
                :method,
                :status,
                :paid_at
            );
            """
        ),
        payments,
    )
    print(f"💳 Payments inserted: {len(payments)}")
```

```
    return len(payments)


# ---------------------------------------------------------------------------
# Main
# ---------------------------------------------------------------------------

def main():
    with ENGINE.begin() as con:
        # 🔧 1) Make sure the sequences are in sync with existing data
        con.execute(
            text(
                """
                SELECT setval(
                    pg_get_serial_sequence('airline.bookings', 'booking_id'),
                    COALESCE((SELECT MAX(booking_id) FROM airline.bookings), 0)
                );
                """
            )
        )
        con.execute(
            text(
                """
                SELECT setval(
                    pg_get_serial_sequence('airline.payments', 'payment_id'),
                    COALESCE((SELECT MAX(payment_id) FROM airline.payments), 0)
                );
                """
            )
        )

        # Debug: show columns for sanity
        booking_cols = [
            row[0]
            for row in con.execute(
                text(
                    """
                    SELECT column_name
                    FROM information_schema.columns
                    WHERE table_schema = 'airline'
                      AND table_name   = 'bookings'
                    ORDER BY ordinal_position;
                    """
                )
            )
        ]
        print(f"🔍 bookings columns: {booking_cols}")

        payment_cols = [
            row[0]
            for row in con.execute(
                text(
                    """
                    SELECT column_name
                    FROM information_schema.columns
                    WHERE table_schema = 'airline'
                      AND table_name   = 'payments'
                    ORDER BY ordinal_position;
                    """
                )
            )
        ]
        print(f"🔍 payments columns: {payment_cols}")
```

```
        passenger_ids, flight_ids = fetch_passengers_and_flights(con)
        used_pairs = fetch_existing_booking_pairs(con)

        booking_payloads = generate_booking_payloads(
            passenger_ids,
            flight_ids,
            n_bookings=TARGET_NEW_BOOKINGS,
            used_pairs=used_pairs,
        )

        inserted = insert_bookings_and_return(con, booking_payloads)
        payments = build_payments_from_bookings(inserted)
        insert_payments(con, payments)

    print("🎉 Synthetic revenue generation complete.")


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