```python
"""
Generate synthetic customers and loyalty data for the Airline BI database.

Populates:
  - airline.passengers
  - airline.loyalty_accounts
  - airline.miles_transactions

Assumes:
  - DATABASE_URL or AIRLINE_DB_DSN is set in the environment
  - Enums:
        airline.loyalty_tier
        airline.miles_txn_type
  - Flights already exist in airline.flights
"""

import os
import random
from datetime import datetime, timedelta

from faker import Faker
from sqlalchemy import create_engine, text


# ---------------------------------------------------------------------------
# DB connection helper
# ---------------------------------------------------------------------------

def get_db_url() -> str:
    """
    Return a SQLAlchemy connection URL from env vars.

    Priority:
      1. DATABASE_URL
      2. AIRLINE_DB_DSN

    Raises if neither is set.
    """
    url = os.environ.get("DATABASE_URL") or os.environ.get("AIRLINE_DB_DSN")
    if not url:
        raise RuntimeError(
            "Set 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)

faker = Faker("en_US")
Faker.seed(42)
random.seed(42)


# ---------------------------------------------------------------------------
# Small helpers
# ---------------------------------------------------------------------------

def random_datetime(start_year: int = 2022, end_year: int = 2026) -> datetime:
    start = datetime(start_year, 1, 1)
    end = datetime(end_year, 12, 31, 23, 59, 59)
    delta = end - start
    offset_seconds = random.randint(0, int(delta.total_seconds()))
    return start + timedelta(seconds=offset_seconds)
```

```python
def age_to_group(age: int) -> str:
    if age < 26:
        return "18-25"
    elif age < 36:
        return "26-35"
    elif age < 46:
        return "36-45"
    elif age < 61:
        return "46-60"
    else:
        return "60+"


# ---------------------------------------------------------------------------
# Fetch reference data (enums, flights, etc.)
# ---------------------------------------------------------------------------

def fetch_enum_values(conn, enum_name: str):
    """
    Fetch enum labels from Postgres, e.g.:

        SELECT unnest(enum_range(NULL::airline.loyalty_tier));

    Returns a list of strings.
    """
    sql = f"SELECT unnest(enum_range(NULL::{enum_name}));"
    rows = conn.execute(text(sql)).all()
    return [r[0] for r in rows]


def fetch_flight_ids(conn):
    rows = conn.execute(text("SELECT flight_id FROM airline.flights;")).all()
    return [r[0] for r in rows]


def fetch_passenger_ids(conn):
    rows = conn.execute(text("SELECT passenger_id FROM airline.passengers;")).all()
    return [r[0] for r in rows]


def fetch_loyalty_ids(conn):
    rows = conn.execute(text("SELECT loyalty_id FROM airline.loyalty_accounts;")).all()
    return [r[0] for r in rows]


# ---------------------------------------------------------------------------
# Generators
# ---------------------------------------------------------------------------

def generate_passenger_rows(n: int):
    genders = ["F", "M", "X"]
    rows = []

    for _ in range(n):
        first_name = faker.first_name()
        last_name = faker.last_name()
        email = faker.unique.email()
        gender = random.choice(genders)
        age = random.randint(18, 80)
        age_group = age_to_group(age)
        # US state or country name; it's just text so we can mix
        if random.random() < 0.7:
            state_or_country = faker.state_abbr()
        else:
            state_or_country = faker.country()
        created_at = random_datetime(2022, 2024)

        rows.append(
            {
                "first_name": first_name,
                "last_name": last_name,
                "email": email,
                "gender": gender,
                "age_group": age_group,
                "state_or_country": state_or_country,
                "created_at": created_at,
            }
        )

    return rows


def generate_loyalty_rows(passenger_ids, tiers, loyalty_ratio=0.6):
    """
    Give a loyalty account to ~loyalty_ratio of passengers.
    """
    rows = []
    take = int(len(passenger_ids) * loyalty_ratio)
    chosen = set(random.sample(passenger_ids, take)) if passenger_ids else set()

    for pid in chosen:
        tier = random.choice(tiers) if tiers else None
        miles_balance = random.randint(0, 100_000)
        enrolled_at = random_datetime(2022, 2025)
        rows.append(
            {
                "passenger_id": pid,
                "tier": tier,
                "miles_balance": miles_balance,
                "enrolled_at": enrolled_at,
            }
        )

    return rows


def generate_miles_txn_rows(loyalty_ids, flight_ids, txn_types):
    """
    Generate a handful of miles transactions per loyalty account.
    """
    if not loyalty_ids or not flight_ids or not txn_types:
        return []

    rows = []
    for lid in loyalty_ids:
        num_txns = random.randint(1, 6)
        for _ in range(num_txns):
            txn_type = random.choice(txn_types)

            # Heuristic: if the enum name hints at redemption, make it negative.
            lower = txn_type.lower()
            if "redeem" in lower or "spend" in lower or "debit" in lower:
                miles_delta = -random.randint(500, 50_000)
            else:
                miles_delta = random.randint(500, 50_000)

            flight_id = random.choice(flight_ids)
            posted_at = random_datetime(2023, 2026)
            rows.append(
                {
                    "loyalty_id": lid,
                    "flight_id": flight_id,
                    "txn_type": txn_type,
                    "miles_delta": miles_delta,
                    "posted_at": posted_at,
                }
            )

    return rows


# ---------------------------------------------------------------------------
# Inserts
# ---------------------------------------------------------------------------

def insert_passengers(conn, rows):
    if not rows:
        print("⚠️ No passenger rows generated.")
        return

    conn.execute(
        text(
            """
            INSERT INTO airline.passengers (
                first_name,
                last_name,
                email,
                gender,
                age_group,
                state_or_country,
                created_at
            )
            VALUES (
                :first_name,
                :last_name,
                :email,
                :gender,
                :age_group,
                :state_or_country,
                :created_at
            );
            """
        ),
        rows,
    )
    print(f"✅ Inserted {len(rows)} passengers.")


def insert_loyalty_accounts(conn, rows):
    if not rows:
        print("⚠️ No loyalty accounts generated.")
        return

    conn.execute(
        text(
            """
            INSERT INTO airline.loyalty_accounts (
                passenger_id,
                tier,
                miles_balance,
                enrolled_at
            )
            VALUES (
                :passenger_id,
                :tier,
                :miles_balance,
                :enrolled_at
            );
            """
        ),
        rows,
    )
    print(f"✅ Inserted {len(rows)} loyalty accounts.")


def insert_miles_transactions(conn, rows):
    if not rows:
        print("⚠️ No miles transactions generated.")
        return

    conn.execute(
        text(
            """
            INSERT INTO airline.miles_transactions (
                loyalty_id,
                flight_id,
                txn_type,
                miles_delta,
                posted_at
            )
            VALUES (
                :loyalty_id,
                :flight_id,
                :txn_type,
                :miles_delta,
                :posted_at
            );
            """
        ),
        rows,
    )
    print(f"✅ Inserted {len(rows)} miles transactions.")


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

def main():
    NUM_PASSENGERS = 5000

    print("🔗 Connecting to database...")
    with ENGINE.begin() as conn:
        # 1) Reference data
        print("📥 Fetching enum values and flights...")

        try:
            loyalty_tiers = fetch_enum_values(conn, "airline.loyalty_tier")
        except Exception as exc:  # noqa: BLE001
            print(f"⚠️ Could not fetch airline.loyalty_tier enum values: {exc}")
            loyalty_tiers = []

        try:
            miles_txn_types = fetch_enum_values(conn, "airline.miles_txn_type")
        except Exception as exc:  # noqa: BLE001
            print(f"⚠️ Could not fetch airline.miles_txn_type enum values: {exc}")
            miles_txn_types = []

        flight_ids = fetch_flight_ids(conn)
        print(f"✈️  Found {len(flight_ids)} flights in airline.flights.")
        if not flight_ids:
            raise RuntimeError("No flights found in airline.flights; run synth_flights.py
first.")

        # 2) Passengers
        print("👥 Generating synthetic passengers...")
        passenger_rows = generate_passenger_rows(NUM_PASSENGERS)
        insert_passengers(conn, passenger_rows)

        # Refresh passenger IDs from DB (includes existing + new)
        passenger_ids = fetch_passenger_ids(conn)
        print(f"👥 Total passengers now: {len(passenger_ids)}")

        # 3) Loyalty accounts
        print("💳 Generating loyalty accounts...")
        loyalty_rows = generate_loyalty_rows(passenger_ids, loyalty_tiers,
loyalty_ratio=0.6)
        insert_loyalty_accounts(conn, loyalty_rows)

        # Refresh loyalty IDs
        loyalty_ids = fetch_loyalty_ids(conn)
        print(f"💳 Total loyalty accounts now: {len(loyalty_ids)}")

        # 4) Miles transactions
        print("🧾 Generating miles transactions...")
        miles_rows = generate_miles_txn_rows(loyalty_ids, flight_ids, miles_txn_types)
        insert_miles_transactions(conn, miles_rows)

    print("🎉 Synthetic customers & loyalty data load complete.")


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