Database Implementation Documentation

This document outlines the implementation of Transactions, Stored Procedures, Triggers, and Constraints used to enforce business rules and data integrity within the application.

1. Transaction Implementation

- **Feature:** save_new_trip (Application-Level Transaction)
- **Description:** The application utilizes a Python-managed transaction to handle the creation of a new trip plan. A "Trip" is a composite entity consisting of data across three tables: TRIP_PLANS, PREFERENCES, and TRIP_LOCATIONS.
- **Criteria & Logic:**
    1. **Atomicity:** The transaction wraps four distinct operations:
        - Calling the Stored Procedure to enforce the trip cap.
        - Inserting the trip metadata.
        - Inserting user preferences.
        - Inserting the list of locations.
    2. **Rollback Safety:** If any step fails (e.g., invalid date range caught by the Trigger, or invalid location rank), conn.rollback() is executed. This ensures that a "ghost" trip is never created without its associated locations or preferences.

2. Stored Procedure

- **Name:** sp_enforce_trip_cap
- **Purpose:** Enforces the business rule that a user may only have a maximum of 5 active trips at one time (FIFO - First In, First Out).
- Logic:
  The procedure accepts a user_id. It counts the existing trips for that user. If the count is greater than 5, it automatically identifies the oldest tripID (smallest ID) and deletes it to make room for the new entry.
- Implementation Note:
  An issue we ran into with mutating a table meant this logic is encapsulated in a Stored Procedure rather than a Trigger. The specific of the error was MySQL Error 1442 (Mutating Table), which prevents a trigger from deleting rows from the same table that is currently being inserted into. The application calls this procedure transactionally immediately before the INSERT.

3. Trigger (4%)

- **Name:** trg_validate_trip_dates
- **Event:** BEFORE INSERT on TRIP_PLANS
- **Purpose:** Enforces logical data integrity regarding time. It prevents the system from accepting a trip where the endDate chronologically precedes the startDate.
- Logic:
  If NEW.endDate < NEW.startDate, the trigger signals SQL State 45000, aborting the insert operation and causing the wrapping Python transaction to roll back.

4. Constraints

- **Name:** uk_trip_ranking
- **Type:** UNIQUE CONSTRAINT
- **Table:** TRIP_LOCATIONS
- **Purpose:** Ensures that within a specific trip, no two locations can share the same rank (e.g., two cities cannot both be the "1st stop").
- **Definition:** UNIQUE (tripID, rank_order)

File 2: db_objects.sql

SQL

```
None
-- 1. CONSTRAINTS
-- Ensure that for a specific trip, every rank_order is unique.
ALTER TABLE TRIP_LOCATIONS
ADD CONSTRAINT uk_trip_ranking UNIQUE (tripID, rank_order);

DELIMITER //

-- 2. STORED PROCEDURE
-- Handles the FIFO logic: If user has 5+ trips, delete the
oldest.
-- Called explicitly by the application before insertion.
CREATE PROCEDURE sp_enforce_trip_cap(IN p_user_id INT)
BEGIN
    DECLARE trip_count INT;
    DECLARE oldest_trip_id INT;

    -- Check current number of trips for the user
    SELECT COUNT(*) INTO trip_count
    FROM TRIP_PLANS
    WHERE userID = p_user_id;

    -- If limit is reached or exceeded, remove the oldest
    IF trip_count >= 5 THEN
        SELECT tripID INTO oldest_trip_id
        FROM TRIP_PLANS
        WHERE userID = p_user_id
        ORDER BY tripID ASC
        LIMIT 1;

        DELETE FROM TRIP_PLANS WHERE tripID = oldest_trip_id;
    END IF;
END //

-- 3. TRIGGER
```

```
-- Ensures a trip cannot end before it starts.
CREATE TRIGGER trg_validate_trip_dates
BEFORE INSERT ON TRIP_PLANS
FOR EACH ROW
BEGIN
    IF NEW.endDate < NEW.startDate THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Integrity Error: Trip End Date cannot
be before Start Date.';
    END IF;
END //

DELIMITER ;
```

Implementation Code

Python

```
None
def save_new_trip(user_id, trip_name, start_date, end_date,
precip_pref, temp_pref, locations):
    """
    Transactional save of a new trip.
    1. Calls Stored Procedure to enforce 5-trip cap (FIFO).
    2. Inserts new Trip (Trigger automatically validates dates).
    3. Inserts Preferences and Locations.
    """
    conn = db._get_connection()
    try:
        cur = conn.cursor()

        # 1. STORED PROCEDURE
        cur.execute("CALL sp_enforce_trip_cap(%s)", (user_id,))

        # 2. INSERT TRIP
        # The Trigger 'trg_validate_trip_dates' fires here
automatically.
        cur.execute(
            "INSERT INTO TRIP_PLANS (userID, planName, startDate,
endDate) VALUES (%s, %s, %s, %s)",
            (user_id, trip_name, start_date, end_date)
        )
        trip_id = cur.lastrowid

        # 3. INSERT DETAILS
        pref_sql = "INSERT INTO PREFERENCES (tripID, featureName,
preferredValue) VALUES (%s, %s, %s)"
        cur.executemany(pref_sql, [
            (trip_id, 'precipitation', precip_pref),
            (trip_id, 'temperature', temp_pref)
        ])
```

```
        loc_sql = "INSERT INTO TRIP_LOCATIONS (tripID, CITY,
STATE, rank_order) VALUES (%s, %s, %s, %s)"
        loc_data = [(trip_id, loc['city'], loc['state'], i + 1)
for i, loc in enumerate(locations)]
        cur.executemany(loc_sql, loc_data)

        conn.commit()
        return {'trip_id': trip_id}

    except Exception as e:
        conn.rollback()
        print(f"Transaction failed: {e}")
        return None
    finally:
        cur.close()
        conn.close()
```