COSC6340: Database Systems
OLTP for an Airline Database

Instructor: Carlos Ordonez

1 Introduction

You will design and create a relational database for an airline. To design the database you will create a minimal ER model. To update the database you will create a program that sends SQL statements (including OLTP SQL commands) to the DBMS. The updated database will be queried via SQL queries.

The project will be delivered in two phases: Phase 1: ER model for normalized database (3NF, BCNF). Database populated with synthetic data. Basic SQL queries. Phase 2: OLTP program. Advanced SQL queries.

2 Database model requirements

You must design a realistic database model that meets these minimum requirements:

- Trip Date
- Trip Time
- Reservation date
- Trip duration (per pair of cities) and total duration (if there are stops)
- Customer name
- Nationality, for international flights
- Economy vs business class
- Seat number
- Single or round trip (open jaw optional)
- Multiple flights on the same day between same cities
- Domestic/international
- Multi-destination trip (up to 5 cities)
- Planes with different number of seats
- Payment with card, check or bitcoin
- Taxes: local, international (if applicable)
- On-time versus delayed flight

3 ER Model Requirements

You should deliver a minimum of 1 diagrams:

- Optional: ER model with classical notation, in "unpolished/denormalized" form. To save space include only entity names and key attributes. You can consider and explain relationships cardinalities, multivalued attributes weak entities, if they apply to this database.
- Required: ER model in UML notation (do not used old ER notation). The database should be in BCNF and this diagram should be a connected graph. You should show two ER diagram versions: one with keys only and another one with all attributes.

You have freedom to use any database modeling, diagram or drawing tool (e.g. Visio). But you need to save and convert the diagram to PDF. That is, ne way or the other, your deliverable is one PDF file.

4 OLTP Program Requirements

Phase2:

- The DBMS will be Postgres, which you can download and install on your computer. Your SQL queries must work in Postgres.
- Your program should start with multiple initial empty airplanes. For simplicity, you can assume only one airplane per source-destination and the airline has no more than 5 airplanes. Gradually there will be more and more conflicts for empty seats as the program progresses.
- Your program should generate multiple threads at the same time. Number of threads will be an input parameter of the program. There should be enough threads to create conflicts (usually between 10 to 100).
- Each transaction should start with "BEGIN" or"BEGIN TRANSACTION". They should have the same SQL queries (INSERT and UPDATE). Each transaction must end with COMMIT. Rollback is optional.
- Apart from the number of threads, everything else should be random number (seat fare class, seat no., source/destination etc). So, execution should be different every time.
- Seats must have at least 2 fare classes: Economy and business. If there is no seat in fare class, the transaction should be declined.
- After all the transactions, you need a final table where you have how many tickets sold per airplane per class and how much dollars is earned. These must match to verify that your execution is correct.
- Each transaction should update at a minimum 3 tables: reservation, customer and flight.
- This program should be able to concurrently process reservations. The finest locking granularity is the seat, of course. The coarsest locking granularity is the plane, where you should forbid reservations if there are no more seats available.
- Your main program should be called "oltp_airline.py" and it will receive as parameter "nthreads=$<integer>;sec=<integer>$", specifying a number of concurrent threads trying to update the database at the same time.

- The type of ticket, source/destination and date/time should be random, but they should create "locking" conflicts (i.e. assume nthreads can be up to 20). Your program should "launch" a set of "nthreads" reservations every $secs$ seconds (2nd parameter). That is, all these reservations should try to update the tables at the same time.
- It is feasible the program is arbitrarily stopped any time (e.g. control C). The database should be maintained obeying ACID properties. The database will be continuously queried via SQL queries. That is, there will SQL queries before and after transactions are processed.

5 Sample SQL Queries

In order to evaluate if your database design and program are OK, your database should be able to answer the queries below. You can assume the TA will run any of these queries before and after transactions are processed.

Basic queries:

- Number of transactions processed: total, per minute, rolledback, committed
- Show pending/completed trips per customer
- list available flights before a certain date
- All flights between a pair of cities
- List trips that appear only once as a reservation
- List flights that still have empty seats
- Aggregate business/class number of seats per plane, per day, per week
- List trips appearing twice or more as a reservation
- List customers who purchased the same flight (based on source/destination) twice or more
- List all valid reservations with customer name and source/destination
- Find customer ids who have not traveled
- List reservations with invalid trips
- List customers who have invalid trips
- Show the city where most customers live
- Get the total number of distinct trips purchased, with and without valid trips. Both counts must be in the same row, computed by one statement.
- List customers traveling from the city where they live
- Count the number of customers with purchased trips per month
- List customers traveling to the city where they live

Advanced queries (will add more in Phase 2):

- List all reservations source and destination only for valid trips; source/destination cannot be null.
- List all reservations source and destination only for invalid trips; source/destination can be null.
- List trips not purchased by anyone.
- Customers who have traveled to every available city.
- List customers who have two or more invalid trips; List those invalid trips sorted by customer.
- Identify the top "active" month: the month with more reservations.
- A table having city as row and each day of the week as column, showing a percentage based on number of trips. That is, finding out the busiest days.

6 Deliverables

Program, diagrams and documentation should be left on your Unix home folder. Do not send emails with attachments to the TA or myself.