COSC3380: Database ER Design, Normalization and Web App: Designing, Updating, and Querying a Practical Database

Instructor: Carlos Ordonez

1 Introduction

You will design a database and develop a web application program that combines SQL statements and a host language, which will be automatically executed by a GUI. Your application will feature both **transaction** and **query** processing, building upon previous homeworks and applying knowledge from the entire course. Your application must provide a front desk GUI (what the customer uses, with defaults to run easily) and back office processing (company monthly processing, automatic adding 100s of records).

2 Target Enterprise Applications

1. University: course, semester, classroom, instruictor, schedule, mode, grade, capacity, payment

   This application is for odd numbered teams: 1,3,..

   frontend: grades and payment

   backend: open/close, tuition bill, post grades

2. Ride: location, user, driver, user, category, price, time, payment

   frontend: booking, ride history, tax

   backend: pay driver, company comission

   This application is for even numbered teams: 2,4,..

3 Major Requirements

1. database model: you should design your database model at the 3 levels, top down.

   ER model: draft, preliminary and final in modern UML notation. Your team is expected to brainstorm and discuss ER design. Therefore, you should have some preliminary ER design, then a better one and finally one in a diagram tool that is polished. It is acceptable you take photos of your initial ER design in a piece of paper and include the jpg in the PDF.

   Logical (normalized database model): up to BCNF, reflected in a final ER model in UML notation. It is expected a few tables will have composite keys.

   Physical (SQL tables): list only the transaction tables DDL and their ER to relational mapping. Include indexes.

2. Application programming:

   Required language: JavaScript.

   The input will be provided by the user in a web form (combining html and JavaScript), validated before execution and then you will build SQL statements sending them to the DBMS (combining user-define input and your internal SQL tables).

3. Constraints: all your tables must fulfill entity and referential integrity. It is expected there will 2-3 tables with a composite key.

4. Database simulation: You should populate reference (lookup, not constantly changing) tables with meaningful information with at least 10 rows. There should be at least 2 tables which require transaction processing (updated frequently, recording usage, payment, interaction). Depending on the target business application some tables will be written based on input from the web form and some data will be atuomatically generated and inserted into tables.

   GUI BUTTON: Your app must automatically populate tables when clicking on "simulation" on your GUI.

5. Database server: Save and run all your SQL from JS in the course server. That is, create all tables and add data to them. The TAs and instructor will browse the tables.

   Removed: web access to the course database server, but this involves a more complicated firewall and Linux setup due to UH IT policies.

6. Transactions:

   Transactions should read and write several tables concurrently, as seen in class. You program should be prepared to handle any OLTP errors.

7. Languages and OS.

   Since your program will work on the browser in a local machine (e.g. laptop) it is expected it works well on any OS (you are encouraged to test on a Windows a Linux desktop and Android). The most common OS for the web client is Windows, but it would be a good idea to test it on other devices like a smartphone.

   Programming language: JavaScript, combined with html. Other web app languages are discouraged, but you can request permission giving technical reason why JS is a bad fit for your team. Explain in your report if you decide not to do it in JS.

   DBMS: PostgreSQL on a shared server or local machine **(default)**.

8. SQL: you must do 90% of processing in SQL. You cannot export or read entire tables row by row, doing processing outside the DBMS. You cannot process data outside the DBMS with the host language low-level processing mechanisms (i.e. locking, text files). The bulk of processing must be done with SQL statements, with a high-level control in the web app language (if/while/for).

9. Testing:

   Your program will be tested on a browser, like Chrome, Firefox or Safari on the TA laptop with the DBMS running on the same machine. Your program will generate and send SQL statements to the DBMS. The app will open in your local computer (e.g. your laptop) and will connect to the local DBMS by default. There will be several users updating the database at the same time (opening several windows). Your program can be stopped by the user at any time closing its window, even when processing a transaction.

   Phase 1: Your GUI should have buttons to test your program for the front desk and the back office. Make sure your program is easy to run with a simulation (the TAs will not spend 10 mins entering information).

   There should be buttons to create tables, to initialize the database lookup tables (i.e. those that do not change frequently), to delete all rows from a table and to browse a few rows (say 10). If an operation is "dangerous" you should ask for confirmation.

   Phase 2: simulate concurrent processing by opening several connections. Provide the ability to view data inserted/updated by some transactions (a time interval with a few minutes or hours). Measure transaction time in millisecs and display it.

Concurrent access examples. Restaurant: food items stock, paying. Phone: number busy, paying.

There should be a button to simulate a real business operation. Example: 100 customers making calls during 1 hour, 100 customers placing food orders at 3 locations.

3.1 Database ER Diagram and Programming details

1. ER in UML notation. Include it in your report PDF. In addition, include the diagram files indicating which tool you used (e.g. draw.io, Visio).

2. Normalization:

   for each table you need to normalize it up to 3NF/BCNF. Notice a few derived attributes or quick lookup, violating 3NF, values are acceptable (like price+tax, fare class). That is, you can have 3NF violations, but justify them.

3. Transactions:

   all changes (INSERT, UPDATE) to the database must be done by transactions on a few records, never isolated SQL statements. It is acceptable to have queries at the lowest SQL isolation level. However, you do not need to create transaction code to initialize lookup tables.

   Tables to update: You are expected to update and insert at least two tables in the same transaction, depending on your target application and design. It is expected you will update a few records (never all records in a table).

   You can assume a maximum number of items to guarantee fast OLTP (i.e. maximum number of purchased items=100, max number of calls=1000).

   Make your app realistic, comprehensive. Part of this homeowork requires to lookup on the Internet how enterprise databases are designed (SAP, Peoplesoft).

4. Queries:

   Your app is expected to produce useful reports for real users on the target application. Imagine these users need useful information, but do not care and cannot figure out how to write SELECT statements. It is expected all your report queries will involve joins and a few will also involve GROUP BY aggregations (to get money totals).

   Your app should also provide standard queries to explore existing tables (to view table content of a few rows).

5. DML: It is recommended you use INSERT and UPDATE. Avoid DELETE.

6. Initial database state:

   Populate tables with information as much as possible and leave day-to-day web processing for transactions.

   There should be GUI options to "initialize" look up tables and your main transaction tables to repeat a long execution of transactions. Ask for a confirmation.

7. GUI: Provide many options to avoid executing code from the command line.

   The app should provide a basic interactive menu (OK to mimic some real web site). The app should have a button to cancel or stop anything. The app should have links to the README file and video. The app should have options to browse SQL sent to the server, especially for transaction and main queries (e.g. in one subwindow or in a frame below). Your program must be prepared for failures and errors, and informative diagnosis messages in the GUI.

8. SQL:

   Create "trace" files "transaction.sql" with the main transaction SQL code and "query.sql" with your main queries. These two files will be checked by the TAs and the instructor.

9. Testing concurrent processing: you must test your program opening several web connections at the same time, accesing the same tables. This aspect can be illustrated in the demo. Your program can be stopped any time.

4 Grading

Deliverables:

1. App: working web app running locally, opened with index.html or home.html (which calls the JS code inside as appropriate). Include a link to your README file and http link to the demo video (below).

2. PDF: call it "teamXY.pdf" where XY is your team id (01,02,..,21,..30). Report in PDF format including ER diagrams, written summary of web app including SQL for transaction and SQL for a few important queries (i.e. the most importan tables), and http link to demo video. Your report should include a list of web references at the end, including websites you chose as reference to build your app.

3. Source files: all source code, diagram files and any useful document you got from the Internet.

   Source code and web app originality: It is expected all source code, databases and GUI will be different even if some source code, some tables, and website ideas are obtained from the Internet.

4. Video (8 minutes max): stored in some public website (youtube, dropbox, Google drive, etc). No password please, long URL OK, make sure it opens by anyone in one click. Format: mp4, viewable on any browser. Do not upload video files to our servers as it takes too much space and this violates UH IT policies.

   You should record a video, covering both DB model, important SQL (transaction code block and a few queries) and showing the finished web application in Phase 2. We will have a Teams session where all teams, TAs and the professor will watch it. Format: MP4.

5. Phase 1: Should have a preliminary ER model covering the minimum attributes listed above, at least five SQL tables. Payment with card must be recorded on a bank account table, subtracting the amount from the balance. You should develop a basic web app functionality sending SQL to the DBMS: at least one transaction code block and at least 3 insightful queries, web app working locally. The transaction must update at least two tables (one of them should be the bank account). All queries should involve joins, which assumes your database ER model is normalized.

   The ER model should include the most detailed table for each target application: (1) detail cellphone calls (duration), data usage (MBs) and total minutes/data and amount to pay. (2) Each item in the receipt (price, quantity or weight) and total amount to pay. Phase 1 will be PASS/FAIL. A team receiving FAIL cannot submit Phase 2.

6. Phase 2: a fully working web app, with a polished DB model, well-designed ER model complying with ER-to-relational mapping, with all tables updated with transactions (no isolated INSERT or UPDATE). The project report PDF will be submitted 1 day after the web app is submitted (to give time for text polishing, but no changes allowed in source code).

5 Submission

5.1 Files to submit

all SQL (saved from JS into .sql files), all your DDLs (create table/index), a home.html page opening your app, JS files, a PDF with a report, a 5-min video in MP4 format. Given the complexity of this

final HW and the fact that it is team-based submission will be done in 2 phases. Submit source files under a folder named hw4 on the course server (same as previous HWs).

All JS/HTML/SQL source code, PDF will be uploaded to the course server.

Table sizes: You can develop code with small tables, but you should make them larger in your demo. Your app must automatically populate tables when clicking on "simulation" on your GUI. In general, you should have a minimum 10 rows per table (more is better!). DO NOT save tables in csv files..all data should inside the DBMS.

5.2 Grading

Scores will cover these aspects: meeting minimum set of attributes, iterative ER design (initial, intermediate, final), normalization (to 3NF at a minimum, BCNF highly recommended, justify why), denormalization violating 3NF expected for a few "essential" columns (explain), comprehensiveness of captured information (realistic app, look up web sites on the Internet), database correctly queried and updated with ACID transactions, and useful reports (via queries). Each aspect will receive a score from 1 to 10. Therefore, you need to test and retest your program before submission (i.e. not tested by the JS developer, but tested by a different person in the team or even a different team!).

Score: a well-designed normalized database, with correct and efficient SQL, with a well developed and working app will receive a high score. A beautiful GUI is secondary (do no waste too much time with frames, animation, colors, images, fonts, TAs will not give extra points). Given the freedom in the ER design, tables and GUI there is no specific database design correctness score. However, it is expected to have reasonable 3NF, updating with SQL transactions, not failing when used. We strongly discourage resubmissions. Therefore, ask at least 1 week before before the deadline. There may be an opportunity for resubmission of minor fixes, like HW1. Major database design changes or major program changes are not allowed.

Team complaints: it is expected work is divided in a reasonable manner: someone does JS development, someone does web page deployment, someone does ER design/diagrams, someone cranks SQL and tests it. Who did what must be explained in your report and the README file. Teams have freedom to discuss and decide the best task split. But conflicts may arise. Team members that do not show up for meetings (in person or online), who do not complete expected work (too many excuses), who are difficult to work with (bad temper, unwilling to listen, little participation, poor programming skills), should be reported by email to the instructor in Phase 1. Then the instructor will split a team, without any grade penalty, in order to improve team collaboration and development speed. However, when a team is split requirements remain the same, but they can be reconsidered when the final project is submitted (e.g. one person working alone, but discouraged).