CSC 471 Project

PROJECT OVERVIEW

Learning by doing is an important strategy to retain knowledge. This course helps the students to develop the ability to develop a real-world database application. In the development process, the students apply the theories learnt in class. This is *not* a team project. Instead, each student should complete it *independently*.

After this project, students should be able to

- identify, collect and analyze database requirements
- refine the requirements by using ER modeling
- design a relational database
- load and query database
- improve performance using indexes and views
- develop a small web application

The project is incremental. Five deliverables are as follows:

Part #1. Database design using ER/EER data model	7%
Part #2. Relational database design (tables, constraints)	7%
Part #3. Database implementation using commercial DBMS (SQL)	8%
Part #4. Performance analysis (bulk load, indexes, views)	6%
Part #5. Web application development	12%

PROJECT PART #1, Database Design Using ER/EER Data Model

Option A:
Suppose that you find a job in the Human Resources Department of “Vision Finance”. You are responsible for building a database to store various information about the employees, departments, and so on. In this part of the project, you will complete a conceptual design using ER/EER data model for a much simplified HR database based on the following set of requirements:

- The employees work in departments for some projects.
- An employee can work on multiple projects that may be participated by several departments.
- Any employee can only be either a salariedEmp or an hourlyEmp but not both. An employee is uniquely identified by a ssn. We should record dob(date of birth), name, sex, and address of the employee. Name is composed of Fname, Minit, and Lname.
- A salariedEmp has a monthly salary and a hourlyEmp has a rate of hourPay.
- An employee may have dependents (described by name, sex, and relationship). Dependents of different employees may have the same names.
- Each department, identified by deptNum, has deptName and one or more locations. The numEmp (number of employees) can be regarded as a calculated attribute of the department.

- A department has a manager who has an officeNum and a startDate.
- A project is identified by projName and projNum together. It has a description (projDesc).

Please draw an ER/EER diagram. Make sure that your diagram includes composite and multi-valued attributes, ternary relationship and weak entity, and EER part of generalization.

Option B: With permission of the instructor, you can choose your own database and your new “Company”. Please discuss with the instructor. You also need to rewrite the project part #1 description (similar to the description shown above in Option A).

PROJECT PART #2, Relational Database Design (Tables, Constraints)

- Specify the domains of each attribute listed by relations. If you have special considerations such as Nullness and uniqueness, please point out.
- Add semantic checks. SSN must be exactly nine digits, and hourPay must be at least $7.50.
- Convert your ER/EER diagram into relational tables.
- Specify the entity constraints (by underlining) and referential integrity constraints (by FK-PK relationship arrows).

PROJECT PART #3, Database Implementation Using Commercial DBMS (SQL)

In this part, you will implement your relational schema designed in part #2, insert some tuples into the tables and query the database. Be sure to know what you should submit specifically.

1. [25] **Create** the tables. Submit your SQL scripts of CREATE TABLE commands. Enforce key and referential integrity constraints and any other constraints you have (e.g. unique, not null, check etc.).

2. [15] **Insert** into each table 4 records. Submit your INSERT commands and the final result of running "select * from YOUR_TABLE" for each of your tables (hint: you can cut and paste to a WORD document named as YOUR_LOGIN_prj3.doc as a submission for example). You can insert 10 records for one or two tables to look at more interesting query results.

3. [20] **Update** a record in one table. Update potentially several records at once using subqueries.

4. [40] **Query** your database. Submit the SQL scripts of five queries into the tables together with their running results on SQL server or Oracle. At least one query includes join operation; one has subquery in it; one has group by and having in it; one has set operation (union , intersect or except) in it.

5. [extra marks] Any extra useful SQL scripts developed by you.

PROJECT PART #4, Performance Analysis (Bulk Load, Indexes, Views)

Indexes and views can improve the performance of frequently asked queries. Instead of only inserting several tuples into tables by hand as you did in last part, this time you will choose one or two tables from your schema to load large number of tuples in order to make meaningful comparison of running times.

1. [20] **Generate** data set which contains at least 2,000 tuples (no more than 5,000) in a table you choose. Use your favorite language to generate a text file (example format: fields delimited by comma, tuples delimited by new line). Submit the source code file of data generation. Show the first 10 lines, 10 lines in some middle position, and last 10 lines of the text file.

2. [30] **Load** into the table with the newly generated text file using the facility of underlying database system. You may want to delete all records before loading. Before and after loading, show the result of query "select count(*) from TABLE".

3. [20] **Create an index** for a field. Run a query containing a condition of that field. Compare the running times of the query with and without using the index. Submit the scripts and running times. Explain why. Lastly drop the index.

4. [30] **Create a view.** Run another query with and without using the view. Try to update the view and see what happens. Explain why. Lastly, drop the view.

PROJECT PART #5, Database Implementation Using Commercial DBMS (SQL)

You will build your web site with both front-end and back-end implementations. Thanks to your efforts of previous parts, your backend database should be already up and running. Therefore, you will focus more on the front-end e.g. user interface design and the interactions between the application and DBMS. Your user interface does not need to be very fancy but must be clear and have the required functionalities.

1. [50] **Basic functionality:** your application must support basic database operations such as search, insert, delete, and update. All these must be implemented in your application program and interact with the database.

2. [20] **Web based:** users can access your database on line from a web browser. You can implement it using any server side scripting language, or any other suitable technology.

3. [20] **Error checking:** your application should be robust enough against bad inputs. For example, a ssn must be exactly 9 digits etc. Is the checking done by application or by database? Explain the trade-offs and give examples of both.

4. [10] **Referential integrity constraint:** let the user see the available input values at runtime (e.g. an existing ssn) during insertion and cascade deletes.

Submission instructions: instructions will be given for the submission of each “part”.