# CS448 - Project 1 (5%)

**Overview**

In this project, you'll create a simple Oracle database & perform some common database operations. The schema of the database is almost the same as that you used in the first project.

Information about getting your Oracle account and general initial configuration is available at: http://www.cs.purdue.edu/oracle. The department has already setup Oracle accounts for you. You can get your password by logging into the CS portal (top right corner of http://www.cs.purdue.edu) and clicking on "My Accounts". As soon as you start working on the project, check that your account exists and you are able to run the *sqlplus* command in a terminal (make sure you follow the setup instructions at http://www.cs.purdue.edu/oracle to get *sqlplus* to work). Keep in mind you will need to use "x@csora" as the username when logging into your Oracle account (after entering the *sqlplus* command), where "x" is your Purdue username. If you have problems with your account, please email oracle@cs.purdue.edu or software@cs.purdue.edu as appropriate to get the problem fixed ASAP. If you would like to work on the project remotely, you can *ssh* to any of the lab machines (the lab machines are named sslab***nn***.cs.purdue.edu, e.g. sslab11.cs.purdue.edu).

**Step 1: Create the tables**

The schema for the database is as follows:

*1. STUDENT(*snum: integer, sname: string, deptid: integer, slevel: string, age: integer)*
*2. CLASS(*cname: string, meets_at: date, room: string, fid: integer)*
*3. ENROLLED(*snum:integer, *cname: string)*
*4. FACULTY(*fid: integer, fname: string, deptid: integer)*
*5. DEPARTMENT (*deptid: integer, dname: string, location:string)*

The fields marked with '*' are primary key.

The meaning of these relations is straightforward:
*STUDENT* contains one record per student identified by snum;
*CLASS* contains one record per class uniquely defined by its name; the fid field of the class gives the instructor of the class;
*ENROLLED* contains a record for each student enrolled in each course;
*FACULTY* contains one record per faculty member uniquely identified by the fid;
*DEPARTMENT* contains one record per department identified by deptid.

Create ***all the key and referential integrity constraints necessary to model the application***. Make sure that your field & table names correspond to those listed above. For this project, we're not asking you to create domain constraints or indices on any of the tables (although you will not lose any points for creating them).

**Your task:** Create a file called tables.sql, which contains five create ... statements corresponding to the tables listed above.
If you're in the directory containing tables.sql, you can create your database tables as follows:

*$ sqlplus*
*...*
*SQL> @ tables.sql*

Remember the '@' operator forces SQL to execute commands from a file.

**Step 2: Read data files**
You are given a sample data file data.sql that you can use to populate your database for your own tests.
**Your task**: Check that all data insertion runs without any problem.
If you're in the directory containing *data.sql*, you can insert the given data as follows:

*$ sqlplus*
*...*
*SQL> @ data.sql*

**Step 3: Query your database**

**Queries:**
Write SQL queries that answer the questions below (one query per question) and run them on the Oracle system. The query answers should be duplicate-free, but you should use **distinct** only when necessary. If you are making any assumptions, state them clearly, and document your queries. We will run your queries on a different dataset from the one provided with the assignment, so be careful not to hardcode values to produce correct answers ☺

1. For the department with deptid=1, print the number of faculty affiliated with the department.
2. Print the names of the oldest students in the university.
3. Print the name, department id and age of the students enrolled in a class taught by a faculty member affiliated with the Computer Sciences department (i.e. dname is Computer Sciences).
4. Print the name and department id of the student(s) who take at least one class OR are younger than 20.
5. Print the ids (snum) of the students who have at least one classmate.
6. Print the names of (distinct) faculty who teach 2 or more classes in the same room.

7. For each department, print the names of faculty affiliated with that department, ordered by department name (ascending).
8. Print the department id, name (sname) and age of students enrolled in at least 2 different classes.
9. Print the ids of faculty who teach classes E.Cho takes.
10. For the class ENG400, print the number of students enrolled in that class, who are younger than 21.

**Hints:**
**Query 1:** The "COUNT" operator will help here.
**Query 2:** The "ALL" operator will help here. Keep in mind you need to find the students with an age greater than or equal to everyone else's age.
**Query 3:** You will need a join of many tables for this query.
**Query 4:** You can use the UNION operator here.
**Query 5:** Remember, the set of classmates of a student does not include the student himself. This really means you need to find students for whom there is at least one student with a different id and who is enrolled in the same class. You don't need to use counts for this query.
**Query 6:** Remember "2 or more" really means more than 1. You don't need to use COUNT for this query. Of course if you have a way using COUNT, that's acceptable too.
**Query 7:** This is a simple join ended with the ORDER BY operator. The result will be one row for each department name-faculty name pair.
**Query 8:** This is similar to Query 6.
**Query 9:** You first need to find the classes taken by the student named E.Cho and do a join with the class.
**Query 10:** A simple count with the specified conditions on the student will suffice.

**General Hints:**
- For comparison with a string such as Computer Sciences, you can use the LIKE operator followed by the string in single quotes.
- There might be multiple ways to form a query and all of them will be acceptable as long as they result in the same record list.

**Your task:** Create a file called queries.sql, which contains the queries listed above, **in the order they are listed (jumbling the order of queries may cost you points)**. Before each query *i*, please put the following comment: *rem Query i*. If you are not able to provide a specific query, just type "rem Query i", where *i is the query number*. So, your file should look something like this:

```
rem Query 1
select ...

rem Query 2
select ....
```

...

**Don’t forget to add a semicolon at the end of each query in the file**, which is what actually runs them.

**Step 4: Views**

Here, you'll create some simple views.

Create two views (Please name them VIEWA and VIEWB) and print their contents.

A. A view that shows the faculty name followed by the classes taught by that faculty, ordered by faculty name.

B. A view that shows the names of people (students and faculty) that are expected to be present in a room each time that a relevant class is taught there. This should be one view with both faculty and student names. A student enrolled in a class is "expected" to be present for each session of the class, and a faculty member teaching a class is "expected" to be present for each session of the class. Hence, the view should contain three fields: name, room, and time.

**Your task**: Create a file called views.sql, which contains SQL commands (create view FOO as ...) to create the views listed above and the SELECT statements that list all the data of both views, i.e. your file should contain two view creation statements followed by two query statements (all of which are ended with semicolons).

**Evaluation:**

Your project will mostly be evaluated based on the correctness of your output for the queries and views. Some points will also be allocated for the proper creation of the database tables. Make sure you comply with the database schema provided above when creating the tables and writing the queries. If your table creation fails somehow, we will test your queries on a database populated with the correct tables and the table field names in your queries have to match those given in the assignment to produce correct output.

**Submission Instructions:**

Please create a README file that contains identifying information. For example:

CS348 - Project 1

Author: John Doe
Login: jdoe
Email: jdoe@cs.purdue.edu

Include here anything you might want us to know when grading your project.

To turn in your project, ssh to *lore.cs.purdue.edu*, create a folder named *project1* in your home directory and copy your .sql files and your README.txt to that folder.

After copying your files (tables.sql, queries.sql, views.sql) in the folder project1, execute the following command in your home directory:

*turnin -c cs448 -p proj1 project1*

To verify the contents of your submission, execute the following command right after submission:

*turnin -c cs448 -p proj1 –v*