Project

We strongly recommend that students pair up in teams of 2.

You are welcome to work individually.

We recommend teaming up, but teams larger than 2 are not allowed.

Project Timeline

The following is a tentative schedule of the project's milestones.

Part 1 Schema Design	20%	(by beginning of class)
Part 2 Entry Forms	20%	(demo day -- sign up for a 15-min slot)
Part 3 Reports and Decision Support Queries	30%	(demo day -- sign up for a 20-min slot)
Part 4 Constraints	15%	(demo day -- sign up for a 20-min slot)
Part 5 Materialized Views and Maintenance	15%	Parts 4 and 5 are demo-ed at the same time, during the same demo slot

The goal of the class project will be the development of the TritonLink132B, a TritonLink-like application. The project will teach you how to build web applications on top of databases. You will be surprised to see how applications with extensive data maintenance and querying requirements can be rapidly, cleanly, and efficiently be built with a database system.

Computing Environment

- Your application will be built the open-source **Postgres DBMS** **(https://www.postgresql.org/download/)**
- The programming language will be Java.
- You will use **Postgres's JDBC Driver** **(https://jdbc.postgresql.org/)** to connect to the database.
- You will use HTML and **Java Server Pages 2.1 (JSPs)** **(http://www.oracle.com/technetwork/java/javaee/jsp/index.html)** to develop the web interface of the project.
- You will use **Apache Tomcat** **(http://tomcat.apache.org/)** as the application server that will contain your web application.
- The TAs will guide you with JDBC and computing environment issues. However, they will not help you on Java programming issues...

Project Description

Data Set

The database application will have to store information about:

1. *Students*: Students have a student-id, first, last name, optional middle name, social security number.

- An undergraduate student belongs to one of the six colleges of UCSD.
- Undergraduate students have major and minor.
- Graduate students belong to a department.
- A student may be enrolled or not enrolled.
- A student has periods of attendance. For example, a student may have attended from Spring 1999-Winter 2000 and from Fall 2000-Spring 2001.
- He may already hold some degrees (e.g., Bachelors) from UCSD or other universities.
- A student may be a California resident, a foreign student or a non-CA US student.
- Students have taken classes in past quarters and are also enrolled in classes of the current quarter. For each class taken in the past there is a grade or an "Incomplete".
- A student may be on probation for a period of time (e.g., Spring 2015-Spring 2016.) The probation has been assigned for a reason.
- Graduate students are classified into MS and PhD students. There is also a special class of

students, who are undergraduates who continued into the MS program, i.e., they are in the 5-year Bachelor's/MS program.
- PhD students are further classified into PhD candidates and pre-candidacy students. All candidates are required to have an advisor and a thesis committee, which is described below.

2. Courses: Courses have numbers (e.g., CSE132A). Sometimes their number may have changed. For example, CSE132A is what was called CSE132 until 1998.

- Each course is given by a department and has prerequisites. Some courses (e.g., CSE291) do not have a specific list of prerequisites but they require consent of the instructor.
- Courses can be taken for a number of units or, sometimes, for a range of units. For example, you can take CSE291A for 1-8 units.
- Some courses can be taken for letter grade only. Other ones can be taken for letter grade or S/U and some courses can be taken only for S/U.
- Courses may or may not require laboratory work.

3. Classes: Each class corresponds to a course and has a title. A class is an instance of a course for a specific quarter and year. For example, course "CSE132B" is offered in Spring 2018 as a class under the following title "DB System Applications".

- It may be given in one or more sections, which have a separate section id and section id's are unique across years. When a class has more than one sections it may be the same or different instructors who teach the sections.
- Assume that section id's are unique across years.
- Each section has a bunch of weekly meetings characterized by types lecture (LE), discussion (DI), etc.
- Some times the discussion is mandatory and sometimes it is not.
- Lectures, discussions and lab sessions happen weekly, at specified dates and times, at rooms of buildings. In addition there are review sessions. The database must indicate the date and time of each review session. Note that review sessions are not regular (they are not weekly.)
- There is an enrollment limit and a set of students are enrolled in the section. There is also a wait list. Each enrolled student receives a letter grade or an S/U (depending on whether the class allows for letter grade and/or S/U grade.) Each section has an instructor, who comes from the faculty set.

4. Faculty: Faculty members have names and titles ("Lecturer", "Assistant Professor", "Associate Professor", "Professor"). They have taught classes, they may be teaching classes in the current quarter and they are scheduled to teach specific classes for next year.

- Assume that faculty name is key.

5. Thesis Committee: Graduate students have to formulate a thesis committee, consisting of at least 3 professors of their department.

- PhD thesis committees also require at least one professor from another department.

6. Degrees: Departments require the completion of a total number of units in order to award a degree.

- It is very common that the total number of units is broken into more than one categories of courses and a minimum number of units is required from some or all of the categories. Also, a minimum average grade is often needed for some of the categories. For example, the Computer Science Department requires 134 units in order to award a B.S, and at least 70 must be from the lower division.
- Conferral of an MS degree may also require that a "concentration" requirement may have to be satisfied. In particular, there are concentrations of courses that are considered to be part of the same general topic. For example, CSE132A, CSE132B and CSE232A are components of the database concentration.

In addition to the above we would like each team to introduce at least one class of entities and relationships that does not appear above. For example, you could consider accounting information.

Data Entry

You are requested to build applications for inserting pieces of the data set in the database. In particular, the following data entry forms are required. Please enforce the primary-foreign key intergrity constraints on the database at table creation time.

1. **Course Entry Form**: Provide forms that prompt for course data, one at a time, and appropriately insert them into the database. Course data include prerequisite information.
2. **Class Entry Form**: Provide forms that prompt for class data (excluding the list of students who are taking the class) and insert them into the database. Classes will have to refer to courses you have already entered.
3. **Student Entry Form**: Forms, again. Exclude information about the classes that the student takes or has taken, the probations he may have received, and his committee info. They will be covered by forms described below.
4. **Faculty Entry Form**
5. **Course Enrollment**: Provide a form that allows us to insert in the database that student X

takes the course Y in the current quarter. If the course has multiple sections prompt for section. If the course is flexible on the number of units prompt the student for the number of units he wants to take.

6. **Classes taken in the Past**: Provide a form that allows us to insert in the database that student X took the course Y in quarter Z. If the course has multiple sections prompt for section. Also, ask for the grade G of the student.
7. **Thesis Committee Submission**: Provide a form that allows graduate students to submit their thesis committee.
8. **Probation Info Submission**
9. **Review Session Info Submission**
10. **Degree Requirements' Info Submission**

You should make sure that the entry forms that you have built make it possible to enter every possible piece of data described in the data set.

The user interface will be pretty simple. Examples are provided in class, in the slides.

You could have additional forms if you think it is necessary (for instance to enter departments).

Queries

The users of the database must be provided with the following functionality. You are supposed to build simple query forms that prompt for the input of X, Y, etc and they return reports that display what is requested.

Some important notes on the terminology used below:

- The word "quarter", whenever used, refers to (QUARTER, YEAR) attribute pairs and not simple to the attribute QUARTER.
- Whenever something about the "current quarter" is requested, in the WHERE clause the QUARTER should be set equal to 'SPRING' (please use capitals exactly as shown) and the YEAR equal to 2018.
- Information about whether a student is enrolled in a given quarter is kept in the STUDENT_ENROLLMENT table.
- Information about whether a student is enrolled in a given section is stored in the STUDENT_SECTION table.
- A student is said to be enrolled in a class if he/she is enrolled in a section of that class.
- The word "professor" refers to any faculty member and not just those with TITLE equal to 'PROFESSOR'.
- Assume that a student cannot enroll in multiple sections of the same class.

- **Reports I**: The following queries are relatively simple.

a. Display the classes currently taken by student X:

1. The form is an HTML SELECT control with all students enrolled in the current quarter. Display the SSN, FIRSTNAME, MIDDLENAME and LASTNAME attributes of STUDENTs given their SSN attribute.
2. The report should display the classes taken by X in the current quarter.
3. On the report page display all attributes of the CLASS entity and the UNITS and SECTION attributes of the relationship connecting STUDENTS with the CLASS they take.

b. Display the roster of class Y:

1. The form is a HTML SELECT control with all classes. Display the COURSE, QUARTER and YEAR attributes of CLASSes, given the TITLE attribute of the CLASS.
2. The report should display all students taking the class Y.
3. On the report page display all attributes of STUDENTs and the UNITS and GRADE_OPTION attribute.

c. Produce the grade report of student X:

1. The form is a HTML SELECT control with all students ever enrolled (i.e. students that were enrolled at some quarter). Display the SSN, FIRSTNAME, MIDDLENAME and LASTNAME attributes of STUDENT X, but pass to the request only the SSN of X.
2. Display all the classes (all attributes of CLASS entities) that have been taken by X. Group the classes by quarter (see at the start of phase 3 what we mean by quarter). Include the GRADE and UNITS attributes in the report.
3. Display the GPA of each quarter and the cumulative GPA. Exclude the incomplete grades (represented by 'IN') from the GPA computation.
4. Use the letter grade-to-number conversion table shown below.

d. Assist an undergraduate student X in figuring out remaining degree requirements for a bachelors in Y:

1. The form has two HTML SELECT controls, one with all undergraduate student names enrolled in the current quarter, and one with all BSC degrees. Display the SSN, FIRSTNAME, MIDDLENAME and LASTNAME attributes of STUDENT, but pass to the request only the SSN attribute. Similarly, display the NAME and TYPE attributes of DEGREEs, but pass to the request only the NAME attribute of DEGREE.

2. Display how many units the student has to take in order to graduate with degree Y.
3. Display the minimum number of units the student has to take from each category (e.g. lower division units required, technical elective units required, etc.) in degree Y. Do this only for categories in Y and not concentrations.
4. If a class is contained in more than one categories, it will be counted as taken for evey category it belongs to. For example assume that class 'DB APPS' is both an upper division course and a technical elective for degree 'BSC IN CS'. Also assume that in order to get the degree 'BSC IN CS' somebody should take at least 30 units of upper division courses and 12 units of technical electives. Then if a student has taken 'DB APPS' for 4 units, they still have to take at least 26 units of upper division courses and 8 units of technical electives to be awarded the degree.
5. For this query ignore the MIN_GPA, GRADEOPTION and GRADE.

e. Assist a MS student in figuring out remaining degree requirements for a MS in Y:

1. The form has two HTML SELECT controls, one with all MS student names enrolled in the current quarter, and one with all MS degrees. Display the same information you did on the form for query d.
2. Display the NAME of all the concentrations in Y that a student X has completed.
3. Remember that a student has completed a concentration if first, he/she has taken some minimum number of units of courses in that concentration and second, the GPA of courses he has taken in that concentration is above some minimum number.
4. Given the student name X and degree name Y, list the set of courses that the student has not yet taken from every concentration C in Y (even for the concentrations C that X has completed). Next to each course display the next time that this course is given (i.e. the earliest time at which a class of this course is given after SPRING 2020).
5. For the computation of both the GPA and the units taken per concentration, ignore courses in which the student has an incomplete grade. Furthermore for the computation of the GPA count only courses taken for letter grade (ignore the ones taken for S/U). However for the computation of the units include the courses taken for S/U.
6. For this query you need to create the letter grade to number conversion table shown in the end of the description for this phase.

- **Reports II**: The following class of queries are relatively hard. The complication comes to a large extent from having to deal with the time element of the queries.

a. Assist a student X in producing his class schedule:

1. The form is a HTML SELECT control with all student names enrolled in the current quarter. Display the SSN, FIRSTNAME, MIDDLENAME and LASTNAME attributes of STUDENT table, but pass to the request only the SSN attribute of STUDENT table (i.e., your query must take only SSN as argument).
2. Display the list of classes (TITLE, COURSE attribute of CLASS table) offered in the current quarter that a student cannot take because for every section of that class at least one of the regular meetings of this section (lectures, discussions, labs) overlaps with some regular meeting of the classes (sections) he/she already takes in the current quarter. For every class that he/she cannot take display which is the class (or classes) that conflict with it. For each one of them, display the TITLE and COURSE attributes of CLASS table.
3. For example, assume that Joe is taking only section S997-2 of CSE 997 that meets every Monday and Wednesday 3:00-5:00. Also assume that there are also two classes CSE 998 and CSE 999, each with two sections. Section S998-1 of CSE 998 meets on Monday 2:00-3:30 and Thursday 1:00-2:30 and section S998-2 of CSE 998 meets on Monday 5:00-5:50, Wednesday 3:00-3:50 and Friday 1:00-1:50. Similarly section S999-1 of CSE 999 meets on Monday 2:00-3:30 and Wednesday 1:00-2:00 and section S999-2 of CSE 999 on Monday 1:00-2:00 and Wednesday 1:00-2:00. Joe cannot take CSE 998 because each of its sections overlaps with some of the meetings of section S997-2. However Joe can take class S999-2 because all meetings of one of its sections do not overlap with the meetings of S997-2.
4. For this query use the START_DATE and END_DATE attributes to infer the days of the week and the times of the regular meetings. Do not use the month and year information for any further computations.

b. Assist a professor X in scheduling a review session for a section Y offered in the current quarter during the time period from B to E: Produce the list of date/time pairs between B and E that do not conflict with the lectures/ discussions/ labs/ review sessions in the current quarter of any of the students of section Y. *Assume that review sessions have to start after 8AM and finish before 8PM, that they last precisely one hour, and can only start on the hour.* The form is a HTML SELECT control with all the sections given in the current quarter. Display the SID attribute of SECTION table and the COURSE attribute of CLASS table, but pass to the request only the SID attribute of SECTION table.
For example, assume that students Jim and Joe take section S1 of CSE 998 and the instructor of that section wants to hold a review session between May 5 to May 9. Jim takes section S2 of CSE 997 that meets every Monday and Wednesday 3:00-5:00. Joe takes section S3 of CSE 996 that meets every Monday and Wednesday 2:00-3:30. The report of available times for a review session of S1 should be:

May 5 Monday 8:00 AM - 9:00 PM

May 5 Monday 9:00 AM - 10:00 AM

May 5 Monday 10:00 AM - 11:00 AM

May 5 Monday 11:00 AM - 12:00 PM

May 5 Monday 12:00 PM - 1:00 PM

May 5 Monday 1:00 PM - 2:00 PM

May 5 Monday 5:00 PM - 6:00 PM

May 5 Monday 6:00 PM - 7:00 PM

May 5 Monday 7:00 PM - 8:00 PM

May 6 Tuesday 8:00 AM - 9:00 AM

…

May 6 Tuesday 7:00 PM - 8:00 PM

May 7 Wednesday 8:00 AM - 9:00 AM

…

May 7 Wednesday 1:00 PM - 2:00 PM

May 7 Wednesday 5:00 PM - 6:00 PM

…

May 7 Wednesday 7:00 PM - 8:00 PM

May 8 Thursday 8:00 AM - 9:00 AM

…

May 8 Thursday 7:00 PM - 8:00 PM

May 9 Friday 8:00 AM - 9:00 AM

…

May 9 Friday 7:00 PM - 8:00 PM

For this query assume that each regular meeting is given 10 times (since the quarter has 10 weeks) starting from the first occurence given from START_DATE to END_DATE. Please read carefully the schema description on the semantics of START_DATE and END_DATE.

- **Reports III: Decision Support**

a. Assist the student who goes for high grades by providing him with information on the distribution of grades in particular courses and by particular professors. In effect, you are asked to provide the functionality of part of a datacube where the dimensions are classes, professors, quarters and the measure is the average of grades.

In particular:

1. In the following, assume that the courses are taken for letter grade only and there is no grade equal to 'IN' (for incomplete).
2. Given a course id (CID) X, a professor Y, and a quarter Z produce the count of "A", "B", "C", "D", and "other" grades that professor Y gave at quarter Z to the students taking course X. Note that course X may have had more than one corresponding sections in the quarter Z. Accumulate the counts of all sections given by professor Y.
3. Given a course id X and a professor Y produce the count of "A", "B", "C", "D", and "other" grades professor Y has given over the years.
4. Given a course id X produce the count of "A", "B", "C", "D", and "other" grades given to students in X over the years.
5. Given a course id X and a professor Y produce the grade point average that professor Y has given at course X over the years.
6. For this decision support query you need to create the following letter grade to number grade conversion table.

```
create table GRADE_CONVERSION(

        LETTER_GRADE CHAR(2) NOT NULL,
        NUMBER_GRADE DECIMAL(2,1)
);
```

```
insert into grade_conversion values('A+', 4.3);
insert into grade_conversion values('A', 4);
insert into grade_conversion values('A-', 3.7);
insert into grade_conversion values('B+', 3.4);
insert into grade_conversion values('B', 3.1);
insert into grade_conversion values('B-', 2.8);
insert into grade_conversion values('C+', 2.5);
insert into grade_conversion values('C', 2.2);
insert into grade_conversion values('C-', 1.9);
insert into grade_conversion values('D', 1.6);
```

Materialized Views and Their Maintenance

Materialized views can greatly accelerate query processing. Let us consider the queries in decision support. The materialized views will consist in tables created with the CREATE TABLE command and updated using triggers.

1. Build the following materialized views, which facilitate building the decision support queries (3.a.ii), (3.a.iii).
    1. Build a view, named CPQG, that has one tuple for every course id X, professor Y, quarter Z, and grade W, where W is one of “A”, “B”, “C”, “D”, and “other”. The tuple contains the count of grade W’s that professor Y gave at quarter Z to the students taking course X. This view is supposed to facilitate the decision support query (3.a.2). All the explanations applicable to (3.a.2) apply to the view as well.
    2. Build a view, named CPG, that has one tuple for every course id X, professor Y and grade Z. The tuple contains the count of the specific grade Z that professor Y has given, when teaching course X. This view facilitates the decision support query (3.a.3).
2. Rebuild the decision support queries (3.a.2), (3.a.3) using the (materialized) views.
3. The next challenge is to write triggers that maintain the views as data are inserted in the database. First build a data entry form that allows you to enter the grade G that the student S got in the section C. Then build the following trigger:
    1. Build a trigger (or multiple triggers) that upon insertion of the grade G that the student S got in the section C, it updates appropriately the views CPQG, CPG. Note that the update of the views must be incremental. That is, you should not recompute the whole view. You must insert or update only the relevant tuple(s).

Project Milestones

Part 1: Schema Design

1. Design an **E/R schema** that fully captures the data set. Include key information.
   If there is information that is not captured in the schema include it in the form of notes associated with the appropriate entities or relationships.
2. Translate your E/R schema into a 3th Normal Form **relational schema**.
   Include in the relational schema all applicable information on keys and foreign keys.

You are advised to look ahead: Does the schema you produced make development of queries and constraints clean and easy? A bit more work now will save you from much more later.

Part 2: Entry Forms

You are requested to build applications for inserting pieces of the data set in the database. In particular, the following data entry forms are required. Please enforce the primary-foreign key intergrity constraints on the database at table creation time.

1. **Course Entry Form**: Provide forms that prompt for course data, one at a time, and appropriately insert them into the database. Course data include prerequisite information.
2. **Class Entry Form**: Provide forms that prompt for class data (excluding the list of students who are taking the class) and insert them into the database. Classes will have to refer to courses you have already entered.
3. **Student Entry Form**: Forms, again. Exclude information about the classes that the student takes or has taken, the probations he may have received, and his committee info. They will be covered by forms described below.
4. **Faculty Entry Form**
5. **Course Enrollment**: Provide a form that allows us to insert in the database that student X takes the course Y in the current quarter. If the course has multiple sections prompt for section. If the course is flexible on the number of units prompt the student for the number of units he wants to take.
6. **Classes taken in the Past**: Provide a form that allows us to insert in the database that student X took the course Y in quarter Z. If the course has multiple sections prompt for section. Also, ask for the grade G of the student.
7. **Thesis Committee Submission**: Provide a form that allows graduate students to submit their thesis committee.
8. **Probation Info Submission**
9. **Review Session Info Submission**
10. **Degree Requirements' Info Submission**

You should make sure that the entry forms that you have build make it possible to enter every possible piece of data described in the data set.

You could have additional forms if you think it is necessary (for instance to enter departments).

Part 3: Reports and Decision Support Queries

Build query forms and reports for the queries listed above. Your forms must prompt for the X, Y, Z, etc that appear in the query specification and they should produce reports with the results. The reports will most likely be tabular. Again, do not spend too much time on the UI.

Part 4: Constraints

As we all very well know, there are multiple constraints on students, schedules, degree requirements, etc. Such requirements must be reflected in the database by using an appropriate schema design or explicit constraints. In every case, when data are entered that violate a constraint you should reject the data and display a message that explains which constraint has been violated.

You will have to build the following constraints using triggers. Hints: You can use additional tables and variables to store intermediate results if necessary. Every time a trigger is triggered you should take necessary actions to restore the correctness of the database (if it has been compromised) and **display an appropriate message** to the user.

1. The lectures ('LE'), discussions('DI') and lab('LAB') meetings of a section should not happen at the same time.
2. If the enrollment limit of a section has been reached then additional enrollments should be rejected. It is not required to update the waitlist .
3. A professor should not have multiple sections at the same time. For example, a professor that is scheduled to teach classes X and Y should not have conflicting sections, mainly overlapping meetings. It is enough to check for the regular meetings (e.g., "LE"). Extra credit is given for checking conflicts on the irregular meetings too.

In your application, data should be entered using a Web interface. You can use the one you created in part 2, Data Entry, of the project or modify it. The error message to be displayed should include the error text coming from the DB server, possibly accompanied by your own message which clearly identifies the constraint that was violated.

Hint: you can use additional tables and variables as needed to store the intermediate results.

Part 5: Materialized Views and Maintenance

Build the decision support queries (3.a.2) and (3.a.3) again, using the views described in the Materialized Views section. Then build triggers that **incrementally**update (i.e., avoid deleting the old views and then recomputing them using the new values, but try to update the content of the materialized views in an incremental fashion: update the content of the views with the latest arrived updates) the materialized views upon insertion or change of a student’s grade.