CPSC 315 – Programming Studio
Team Project 1: Database Implementation and Use

Project Goal:
Your goal in this project is in two parts. First, you are to implement a general database system with a limited amount of database functionality. You will create a standalone "library" that can be used by other programs to define, populate, and query a database. Second, you will implement a basic database application program that makes use of the database system defined.

Note that in a "real" application, it is unlikely that you would implement one of these database systems "from scratch." More likely, you would use an existing database system, and access the system via query functions, a DDL, and/or a DML. This project is meant to help you:

- Understand how a database system might work at a low-level
- Become familiar with the basic database operations and queries
- Understand good data structure choices and algorithmic implementation of basic data structures
- Become familiar with use of a version control system
- Gain practice in basic modular organization and library specification

Development Specifics:
Your project will consist of two major pieces of code: a database library and an application module. These two pieces of code should be developed independently. That is, your database library should have an API defined for it (and the API should not expose underlying functionality unnecessarily). The library code should implement that API. Your application program should access the database only via that API.

The details of the code are described below.

Specification of Database Library:
You should create an independent library that gets compiled into a .jar. This library should allow a user to specify a relational database, add data to the database, and perform queries on it. Specific features to be supported are given below (**note: items marked with an * do not need to be completed by two-person teams**):

Data Definition:

- Creation of tables. Users should be able to specify tables analogous to the SQL CREATE function. Attribute types to be supported are:
    - Integers
    - Floating-point numbers
    - Strings of variable length (you may set a reasonable maximum, e.g. 100, but the space allocated should be variable).

  In addition, you should enable users to specify a primary key for this data.
  You do not need to support weak entity sets for this program!
- * Deletion of tables.

Ideally, you should account for basic errors, such as specifying two tables with the same name, inserting a tuple with a duplicated key, etc. You do not need to check for more fundamental database organization problems.

Data Manipulation:

- Insertion of Tuples into an existing relation
- * Modification of Tuples (analogous to the SQL UPDATE command)
- * Deletion of Tuples (analogous to the SQL DELETE command)

Note that for modification and deletion of Tuples, you should support limited conditions, as described in the querying function below.

Querying:
Basic SELECT-FROM-WHERE statements should be supported. These should include support for the following (all are analogous to SQL statements, but this is *not* saying that these functions need to be provided via an SQL-

formatted query – you may choose any method you want in your API to specify the query statement, including supporting standard SQL syntax):

- In the SELECT statement:
  - The * in the SELECT statement (i.e. select all)
  - * The AS command
  - You do *not* need to support operations on the items in the select statement
- In the FROM statement
  - Up to two sources (possibly from the same relation – analogous to renamed sources in SQL)
- In the WHERE clause, support for
  - An IN statement (applied to a table with only one attribute)
  - An EXISTS statement (applied to a table)
  - Comparisons of =, !=, >, <, >=, <=
  - AND, OR, and NOT
  - * ALL
  - * ANY

  You only need to handle cases where "full parentheses" would be used – i.e. all parentheses would be used around any statements, and will never nest more than 3 deep. For example (((A>B) OR (B<C)) AND (D = E))
- Include support for SUM, COUNT, MIN, MAX on a single column of a table

Note that for this program, the results of **every query** should be stored as a view, automatically. That is, any query that is performed should generate a new table. It should be possible to query this table by a future query, just as any of the "defined" tables. A FROM statement should be able to use one of these tables as a source, but it does have to take the query result directly (i.e. there are no nested queries, here).

General Database:
Commands should be included to return information about the database as a whole. At a minimum this should include

- * Returning counts of total numbers of records in each table (could be done with the COUNT command mentioned earlier), or in the database as a whole.
- * listing all the relations/attributes in the database

Specification of Application:
You should create a separate application program that uses the database API. This program should be written to handle an airline reservation system (you can assume just for one airline if you want). You will need to write routines to create the necessary tables, populate the tables (from a file and/or from user input), and query the tables in useful ways.

You may make the application program as fancy as you want. But, at minimum, among the data you should keep in the database are the following:

- Flight information (e.g. flight number, planes, number of passengers held, starting/ending cities, etc.)
- Passenger information (e.g. Name, address, frequent flyer number if any, etc.)
- Ticket/Reservation information (e.g. flight, date, passenger, cost, etc.)

You should support interaction with this system, including such things as:

- Find all passengers on a particular flight
- Find information (including flights) for a passenger

You are strongly encouraged to extend this beyond such minimal extent – e.g. supporting questions like finding out how full a particular flight tends to be, etc. Doing simply the absolute minimum would not earn a particularly good grade. However, do not overshoot, so that you are unable to get your program working – it is better to have less functionality that works than more functionality that does not.

Note that any two-person team will have much lower expectations in this application program than a three-person team would.

Team Organization:
Each project team has two or three members, one of which is the "project leader" and has additional responsibilities. Each month-long team project will consist of three phases:

Week 1: Project leader develops high-level specification of Java objects to be instantiated and identifies which team member will be responsible for each object. This specification takes the form of the names of the objects, their methods (and parameters), and any externally available constants or variables and comments describing each of the above.

Weeks 2-3: Team members implement their assigned software components, communicating as needed to ensure necessary changes to the original specification are known to all team members.

Week 4: All team members work to pull together their components into the final, working system and generate all necessary documentation for the project.

After the project: Each team member will be asked to assess the work of the other team members.

For this assignment, the teams will be as follows (project leaders are listed in *italics*):

Team 1:
Jeffrey Deuel
Kourtney Kebodeaux
Zachary Edens

Team 2:
Brandon Jarratt
Gabriel Copley
Andrew Reagan

Team 3:
Jacob Lillard
Travis Kosarek
Christopher Aikens

Team 4:
Julio Montero
Christopher Bennett
John Laky

Team 5:
Patrick Robinson
Mark Hill
Matthew Moss

Team 6:
Andrew Johnson
Drew Havard
Thomas Robbins

Team 7:
Benjamin Unsworth
Christopher Weldon
Ryan Mcauley

Team 8:
Jorge Cereijo-Perez
Brett Hlavinka

Team 9:
Robert Kern
Jillian Graczek

Code Organization:
This project is to be completed in Java.

The project leader is required to maintain its development using a version control system. You should choose and set up an SVN repository/project site in which your team will keep current versions of source code, documentation, etc. This should be one of the first decisions your team makes. CodePlex, Google Code, and Sourceforge are suggested locations for these repositories, but you may choose another option if you would prefer. Note that your team is to make use of this repository for all development. You should **not** pass around files by email, saving in shared directories, etc. Significant points will be taken off of the project if code is shared via a method other than the SVN system. (You may use a version control system other than SVN, if you would prefer).

Your project leader should give access to both the Instructor and the TA to the SVN repository for download. Your project leader is required to **send email** to both giving the location of your team's project, and instructions for access to the SVN repository (this should be done by the time of Project Update 1, below). The instructor/TA may download SVN software at any point.

Intermediate Updates:
Organization Statement: Your project leader is to turn in a written update regarding the overall project design. Specifically, your report should have 3 main components:

- A brief summary of what system you are using for version control, and a short justification (1/2 page) for why your team chose that option. By this point, you should have emailed the professor and TA with details on your SVN repository, and any details they need regarding access.
- A statement describing how your team will be organized and any responsibilities assigned to team members. You may use any organization, but you should give a justification/reasoning behind whatever you have chosen. This summary should be ½ to 1 page in length.
- A brief (1 to at most 2 pages) summary of the approach your team will be taking for implementation of the project. It is recommended that you try to identify the major implementation issues, set intermediate deadlines for the team (building in some slack time), including for production of documentation, and assign responsibilities to members (at minimum, for the initial work)

API Specification: Your team is to create a complete description of the API for the database library. You should be sure to document exactly what functionality your library will provide, and how to access this functionality. Note that you do *not* have to have the actual library implemented by this point, but you should have a clearly defined interface designed. Your API should be given in a form that is easily accessible by others. One option is to use a documentation area of whatever source code repository you are using. Another option is to set up an HTML document that you post on the web. This API description should serve as the primary documentation for someone wanting to make use of your library. On the due date for this specification, you are to email both the instructor and TA to give the location of your API specification.

Note that after your API specification is published, you should generally not make substantial changes to it. However, if it becomes necessary to do so (due to problems discovered when implementing it), you should be sure that your specification clearly shows the change history of the API. Furthermore, you should publish such updates as soon as you possibly can.

Code Checkpoint 1: The code checkpoint is an intermediate point at which you will "release" an early version of your database library. Your code at this point should be substantially complete, with only small features/details missing, and bugs remaining. The TA should be able to include your library at this point, and make all of the specified function calls, however, there may be some missing functionality and/or bugs in the version. Where possible, these missing features and bugs should be noted/documented.

It is strongly suggested that for your code "release", you release your code via your source code repository web page. You should email the the instructor and TA, giving information about where to go to download this intermediate code release.

Besides this release, you should submit a "snapshot" of your code via CSNET.

Code Checkpoint 2: The second code checkpoint is meant to be the point at which the database library functionality is "complete." The TA should, after this point, be able to use your library within an application, with full functionality matching that specified in the API. Thus, you should plan to have all functionality implemented by this point. Note that after this code checkpoint, your team may still need to respond to bugs found in the database. It is suggested that, like the first checkpoint, you should "release" your code via your team's project source code repository. Besides this release, you should submit a "snapshot" of your code via CSNET.

Documentation of Final Project:
Your final report should include three pieces of documentation. In addition, you should submit your code in its final state via CSNET. This submission will be your final graded project, and the turnin time will be based on that code turnin.

First, you are to include documentation describing the API for your library (in its final form) and for operation of the application program you provide. The API should be substantially the same as the initial specification, with changes clearly documented.

Second, you should include a document describing the use of your application program (this is the part that will be graded, below). This document should (without being too wordy) describe the features of your application program, including how to perform all of the basic commands desired. Note that your team may be asked to demonstrate your application program shortly after the time it is turned in.

Finally, your team should turn in an evaluation document that describes how well your team met its planned goals/schedule/etc. as outlined in the initial organization statement.

Turning in Work:
Except as specified otherwise (e.g. emails), you are to submit work and reports via the CSNET turnin system. Individual reports (e.g. the final report regarding teamwork) should be submitted by individuals. The team reports and code should be submitted by the project leader.

Final Report:
Individuals will be required to put together a specific report detailing how the teamwork went. The results of this report will figure into the grade allocation among team members. Details of this final report will be given later.

Project Grading:
The overall project will be graded as described below. This grade will be used to determine the team grade, and individual grades will be determined by apportioning the team grade among the team members.

10% Code Checkpoint 1 – Is a usable, mostly-implemented library provided?
10% Code Checkpoint 2 – Is a complete implementation of the specified API provided?
10% Final Application Documentation
30% Completeness/Correct Operation of Database library
30% Functionality/Extent/Correct Operation of Application program
10% Code style: naming/layout/commenting

The project leader will be graded as described below.

10% Source system selection & explanation of use
30% Database API Specification – How complete is the API? Is it specified clearly for team members.
30% Design of Application – How complete is the design? Is it specified clearly for team members.
20% Description of Team Organization (who is responsible for what)
10% How well the team worked (from individual reports)

Dates and Deadlines:
The following are the intermediate deadlines for this project. Details of the particular requirements are described above. Note that these are final deadlines; it is likely that your team would want to complete several of these (e.g. the two Project Updates) well in advance of the "due date"

Tuesday, February 3
Project Assigned

Tuesday, February 10, 5:00 p.m.
Email Instructor and TA regarding SVN repository
Project Leader Report (source system, team organization, and API/application design/code stumps)

Tuesday, February 17, 5:00 p.m.
Code checkpoint 1: Initial Release of Database functionality to TA

Tuesday, February 24, 5:00 p.m.
Code checkpoint 2: Delivery of Database Functionality to TA

Tuesday, March 3, 11:59 p.m.
All final project code turned in
Project documentation turned in along with code

Two days after final code turned in (or **Thursday, March 5**, 11:59 p.m.)
Final group report turned in

One day after group report turned in (or **Friday, March 6**, 11:59 p.m.)
Individual Reports on teamwork turned in