Indian Institute of Information Technology, Nagpur
Department of Computer Science & Engineering
5th Semester CSE

DBMS Lab & DBMS Project

Date: 12th Sept, 2018

The DDL and sample data for the tables in the university, with one change: instead of using the **time** data type, which is not supported on some databases such as Oracle, we split start_time into start_hr and start_min, and end_time into end_hr and end_min, which are integer types. If the database you use supports the **time** type, and you wish to use it, you can modify the DDL and the sample data files accordingly.

1. SQL code for creating small relations

The file **smallRelationsInsertFile.sql** contains data. The file contains SQL insert statements to load data into all the tables, after first deleting any data that the tables currently contain.

The data include students taking courses outside their department, and instructors teaching courses outside their department; this helps detect errors in natural join specifications that accidentally equate department names of students or instructors with department names of courses.

2. SQL code for creating large relations

The file **largeRelationsInsertFile.sql** contains SQL insert statements for larger, randomly created relations for a truly strange

The sizes of the relations are as follows:

department	20
instructor	50
student	2000
course	200
prereq	100
section	100
time_slot	20
teaches	100
takes	30000
advisor	2000

3. Table Generation

The file **tableGen.java** contains a Java program for generating the largeRelations data. Be warned that some versions of the eclipse Java compiler lack the java.util.Scanner class that this program uses; if you run into a problem compiling this program, we suggest you upgrade to the latest version of the eclipse Java compiler, or use the Sun Java compiler instead.

4. Tasks To Do

4.1 Accessing the Database: Connect to a database, populate it with data, and run very simple SQL queries.

- Create tables and load sample data. Scripts for these tasks is **DDLDrop.sql**
- Try the queries:
  - select * from instructor
  - select name from instructor where dept_name = 'Comp. Sci.' and salary > 70000
  - select * from instructor, department where instructor.dept_name = department.dept_name

4.2 Basic SQL

1. Find the names of all the instructors from Biology department
2. Find the names of courses in Computer science department which have 3 credits
3. For the student with ID 12345 (or any other value), show all course_id and title of all courses registered for by the student.
4. As above, but show the total number of credits for such courses (taken by that student). Don't display the tot_creds value from the student table, you should use SQL aggregation on courses taken by the student.
5. As above, but display the total credits for each of the students, along with the ID of the student; don't bother about the name of the student. (Don't bother about students who have not registered for any course, they can be omitted)
6. Find the names of all students who have taken any Comp. Sci. course ever (there should be no duplicate names)
7. Display the IDs of all instructors who have never taught a couse (Notesad1) Oracle uses the keyword minus in place of except; (2) interpret "taught" as "taught or is scheduled to teach")
8. As above, but display the names of the instructors also, not just the IDs.
9. You need to create a movie database. Create three tables, one for actors(AID, name), one for movies(MID, title) and one for actor_role(MID, AID, rolename). Use appropriate data types for each of the attributes, and add appropriate primary/foreign key constraints.
10. Insert data to the above tables (approx 3 to 6 rows in each table), including data for actor "Charlie Chaplin", and for yourself (using your roll number as ID).
11. Write a query to list all movies in which actor "Charlie Chaplin" has acted, along with the number of roles he had in that movie.
12. Write a query to list all actors who have not acted in any movie
13. List names of actors, along with titles of movies they have acted in. If they have not acted in any movie, show the movie title as null. (Do not use SQL outerjoin syntax here, write it from scratch.)

4.3 Intermediate SQL

Using the university schema that you have write the following queries. In some cases you need to insert extra data to show the effect of a particular feature -- this is indicated with the question. You should then show not only the query, but also the insert statements to add the required extra data.

1. Find the maximum and minimum enrollment across all sections, considering only sections that had some enrollment, don't worry about those that had no students taking that section
2. Find all sections that had the maximum enrollment (along with the enrollment), using a subquery.
3. As in in Q1, but now also include sections with no students taking them; the enrollment for such sections should be treated as 0. Do this in two different ways (and create require data for testing)
    - Using a scalar subquery
    - Using aggregation on a left outer join (use the SQL natural left outer join syntax)
4. Find all courses whose identifier starts with the string "CS-1"
5. Find instructors who have taught all the above courses
    - Using the "not exists ... except ..." structure
    - Using matching of counts which we covered in class (don't forget the distinct clause!).
6. Insert each instructor as a student, with tot_creds = 0, in the same department
7. Now delete all the newly added "students" above (note: already existing students who happened to have tot_creds = 0 should not get deleted)
8. Some of you may have noticed that the tot_creds value for students did not match the credits from courses they have taken. Write and execute query to update tot_creds based on the credits passed, to bring the database back to consistency. (This query is provided in the book/slides.)
9. Update the salary of each instructor to 10000 times the number of course sections they have taught.

   Create your own query: define what you want to do in English, and then write the query in SQL. Make it as difficult as you wish, the harder the better.

4.4 Advanced SQL

Here you will write more complex SQL queries, using the usual schema by default. Again, you have to add required data to test your queries, and must include the SQL statements to create the data along with your queries.

1. The university rules allow an F grade to be overridden by any pass grade (A, B, C, D). Now, create a view that lists information about all fail grades that have not been overridden (the view should contain all attributes from the takes relation).
2. Find all students who have 2 or more non-overridden F grades as per the takes relation, and list them along with the F grades.

3. Grades are mapped to a grade point as follows: A:10, B:8, C:6, D:4 and F:0. Create a table to store these mappings, and write a query to find the CPI of each student, using this table. Make sure students who have not got a non-null grade in any course are displayed with a CPI of null.
4. Find all rooms that have been assigned to more than one section at the same time. Display the rooms along with the assigned sections; I suggest you use a with clause or a view to simplify this query.
5. Create a view faculty showing only the ID, name, and department of instructors.
6. Create a view CSinstructors, showing all information about instructors from the Comp. Sci. department.
7. Insert appropriate tuple into each of the views faculty and CSinstructors, to see what updates your database allows on views; explain what happens.
8. Grant permission to one of your friends to view all data in your student relation.
   Now grant permission to all users to see all data in your faculty view. Conversely, find a friend who has granted you permission on their faculty view, and execute a select query on that view.

4.5 Database Access From a Programming Language Here you will access database from a programming language such as Java or C#. Although phrased using Java/JDBC, the exercise can be done using other languages, OBDC or ADO.NET APIs.

4.5.1 JDBC Introduction

Here you will write a java program which takes any of the following commands as command line argument (not as input), and executes them using JDBC:

1. insert into relationname value1 value2 value3 ..
   Inserts a tuple into the specified relation with the specified values; make sure you use a prepared statement; test it with input containing single quotes (if you run from command line, enclose those in double quotes so linux doesn't interpret the quotes) you can assume that all values are strings for simplicity
2. select from relationname
   Prints all tuples from the specified relation. For the purpose of this assignment, you should assume that relationname is one of "instructor", "student" or "takes", and not use database metadata features to print the tuples.
3. select from relationname where "condition
   Executes a query with the specified condition. Note the use of double quotes so that the condition comes as a single command line parameter. Again assume relationname is one of those from the previous feature. So this is really a small addition to the code for the previous feature, do NOT make a separate copy of the code.
4. select from relationname1 relationname2
   Displays result of natural join of the two relations. This time, use the resultset metadata feature to

display all values from the query result; don't worry about displaying column names, that is optional, we only care about values being displayed.

Note: In netbeans, to pass parameters to a program, you can right click on the project, choose configuration > customize. Or run this as a standalone java program instead of using netbeans (after you develop it using netbeans). Use the parameter argv to the main function, which provides you an array of Strings, with each word above part of one string. Use argv.length to figure out how many parameters are present.

For 0 extra credit: If you finish the above early and are bored, try adding column names to the select from relationname option; assume from is not a valid column name. You get no extra marks, except that if you goof up an earlier part, you may still get full marks on the assignment if you do this right (it's very unlikely you will do this part if you goof up an earlier part, anyway)

4.6 Database Metadata Access From a Programming Language Here you will see using metadata when writing programs.

4.6.1 JDBC Metadata

Here you should use JDBC metadata features to implement keyword search on relational data, as follows.

1. Your program takes a single keyword as argument, and finds all tuples in any relation that contain that word as a substring in any column of the tuple. The output is the entire tuple. You search should not be case sensitive and can match part of a word, so a keyword Avi matches a string avi, as also Ravi. Create SQL queries dynamically, using the like clause to implement this feature.

   You will find it useful to implement a separate method that can output all tuples from a ResultSet.

For extra credit: As above, but now you should accept more than one keyword as argument, and ensure that a tuple contains all the keywords; the keywords can each be in a different attribute, as long as the tuple contains all the keywords.

4.7 Building Web Applications Here you will see the construction of Web applications. Although phrased using the Java Servlet API, the exercise can be done using other languages such as C# or PHP.

4.7.1 Servlets

There are two parts to this assignment both to be implemented using Java Servlets. You can copy the sample servlet and edit it to get your servlets working. You may find it useful to output debug statements both to the response, and to System.out (you will see it on the netbeans screen and/or in the tomcat/glassfish log files)

1. Create a web page with separate forms, and corresponding servlets, to perform each of the following actions:
   - Accept a roll number, and display the following data about the student: name, department, and all courses taken, in tabular form: list the course id, title, credits and grade (show a blank if the grade is null).

- Accept a word, and find all courses whose title contains that word
- Show all students with two or more fail grades; no need to show the courses they have failed, and I don't care if they passed the course subsequently.
- Take a roll number, name and department name, and insert a new student record with tot_creds set to 0. Make sure you catch exceptions and report them (and test it with an incorrect department name, to see what happens when a foreign key constraint is violated, and similarly with a duplicate roll number, to see what happens when a primary key constraint is violated). Make sure also to close connections, even if there is an exception.

2. This part of the assignment illustrates how it is useful to create functions that generate HTML, instead of writing it directly.
    - Write a function createDropDown(ResultSet) that takes a ResultSet, and creates a drop-down menu from the ResultSet; the first attribute is used as the result value, and all attributes are displayed in the drop-down menu, concatenated together.
    - Also write a function createDropDown(String) which does the same thing, but instead of a ResultSet it takes an SQL query as a string; the second version can be used safely as long as the query does not involve any value that the user inputs, otherwise it would be vulnerable to SQL injection attacks.

Create form interfaces to insert records for the student and instructor tables, with a drop-down menu for the department name, showing all valid department names. As before, exceptions should be caught and reported.

Important Instructions

1. Form a group of **TWO** to solve this Lab assignment.
2. E-mail the details of groups of complete class in one file **on or before 19th Sept, 2018** (CR have to take the initiative to create the file).
3. Design the E-R model for the university database by using the **DDLDrop.sql** file.
4. Innovation and creativity will encourage maximum marks.
5. Evaluation will be based on the performance of individual student in the group. But, student must be position to explain all functionalities.
6. E-mail all the SQL queries 4.2, 4.3, 4.4 output **on or before 11th Oct, 2018.**
7. Final evaluation will be on **21st Nov, 2018 from 9.00 AM to 11.00 AM**.

Some help:

A. Setting up Oracle

The Oracle database can be downloaded from
http://www.oracle.com/technology/software/products/database/index.html.
Instructions for setting up Oracle on a variety of flavours of Linux can be found at
http://www.oracle.com/technology/tech/linux/install/index.html
Instructions for installation on Windows can be found at
http://www.oracle.com/technology/obe/11gr1_db/install/dbinst/windbinst2.htm.
Oracle SQL Developer is a useful tool for browsing an Oracle database and executing queries on the database. The SQL Developer home page, including links for downloading it, can be found at http://www.oracle.com/technology/products/database/sql_developer/index.html.

B. Connecting to and Executing Queries on Oracle

Once Oracle is set up, you can connect to it using Oracle SQL Developer, or the Netbeans IDE. JDBC drivers for Oracle, as well as libraries for connection pooling of JDBC connections may be found at http://www.oracle.com/technology/software/tech/java/sqlj_jdbc/index.html.

C. SQL Tips on Oracle

Oracle supports most of the SQL standard syntax covered in the book, including most advanced features, but there are a few differences that you should watch out for. We list a few commonly encountered ones below (some of these may apply to some versions of Oracle, and not others):

- The **as** keyword is not supported in many contexts; for example, to rename a relation in the **from** clause, instead of using *relname***as***newname*, you should simply use *relname newname*.
- The **except** keyword is not supported; instead, use **minus**.

The SQL integrated programming language PL/SQL is widely used for creating stored procedures in Oracle, and SQL Developer provides facilities for PL/SQL development.

D. Setting up PostgreSQL

The PostgreSQL wiki has a nice page on how to set up PostgreSQL with sections covering each of the popular operating systems: http://wiki.postgresql.org/wiki/Detailed_installation_guides.
On unix/linux systems, a user called postgres is created as part of PostgreSQL setup. We also recommend you install the pgAdmin3 user interface. After installing postgreSQL, you need to configure it. A tutorial for configuring it on Debian/Ubuntu can be found here: http://www.debianhelp.co.uk/postgresql.htm.
By default, PostgreSQL allows access from IDEs on the same computer. If you want to access it from another machine, you have to configure it to accept remote connections. It is best to allow access only from a selected set of IP addresses, or range of IP addresses, to provide better security. You will also need to create users on the PostgreSQL database in order to use it.

E. Connecting to and Executing Queries on PostgreSQL

Once PostgreSQL is set up, you can connect to it using a tool such as pgAdmin3 to administer the system, browse databases, and execute queries. You can also browse the database and execute queries using the command line tool psql, or the Netbeans IDE.

F. SQL Tips on PostgreSQL

PostgreSQL provides good support for basic SQL syntax. Most queries in the book run as-is on current versions of PostgreSQL. PostgreSQL also has support for some advanced features such as recursive WITH queries, window queries, and some non-standard syntax such as the **limit** and **offset** clauses to fetch a specified number of rows from a specified starting point.