# CSC 3300 Homework 3 – Security & Languages

## Description

Homework 3 has two parts. Part 1 is an exercise in database security. In particular, Part 1 has practice problems in which your will add constraints to ensure a databases integrity. Part 2 is an exercise in database connectivity and programming using general purpose programming languages. For both part1 and part2, the host, username, and password for the MySQL server are localhost, root, and coursework, and the host, username, and password for the Firebird server are localhost, SYSDBA, and coursework.

## Part 1 – Do you have integrity

In a DBMS, protecting integrity of data against unauthorized changes is crucial. We also want to make sure that the data in our tables accurately reflect the information in the real world. Therefore, for Part 1 of this assignment, you will implement some integrity controls on a database.

There are three main objectives to integrity control in database management systems: control who can edit data, control what data is sent, and control the data at rest. We can control modification to data by creating user accounts and assigning each role or users account with certain privileges that can restrict what can be modified by whom and also in what way. Additionally, we can monitor all modifications made by users and restrict their privileges if they are not acting in the interest of the system.

Integrity can also be enforced while creating the tables for the database. Options such as data types, primary and foreign keys, not null, auto increment, etc. allow the DBMS to reject data that does not adhere to its intended structure. The difficulty here is that the administrator will need to understand the security policy and the structure of the DBMS, and map the security policies as database integrity constraints to the database schema carefully. For instance, every resident of the United States has a Social Security Number (SSN) and that is why it can be used to uniquely identify an individual. Therefore, a SSN column should not allow null values to be written.

Lastly, Regular backups offer extra protection. Databases can be corrupted from attacks or system errors. If the system admin has been making regularly scheduled backups, the admin should be a be able to restore the database to a previously consistent state.

In this exercise, we want to create some tables based on our data definition in security policy in an intelligent way so that the database will never allow invalid data to reach an inconsistent state.

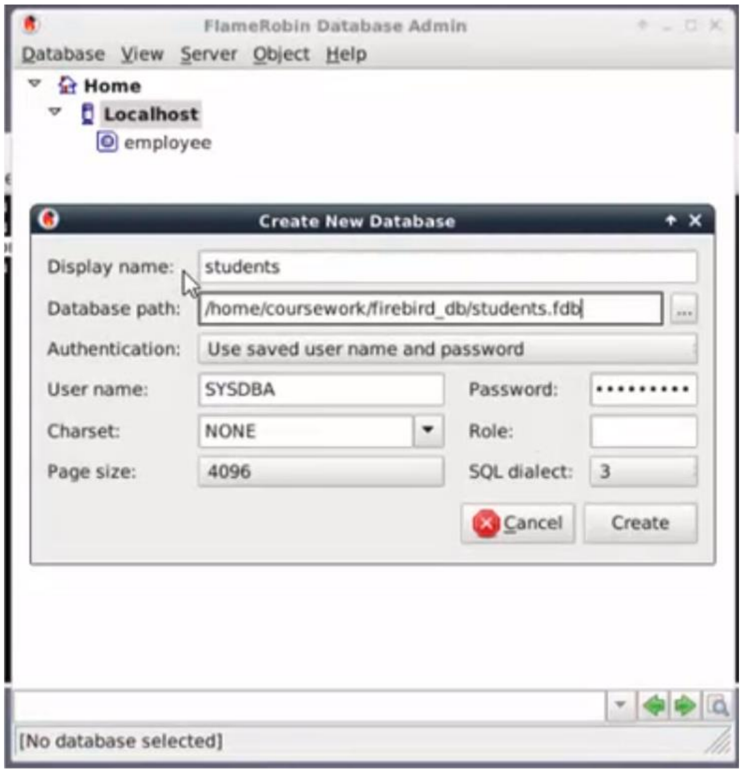
### Creating the database

Before you can begin creating tables, you must create a new database. Firebird requires a new file to hold the new database. However, we can not just point Firebird to any file, because Debian installs Firebird server so that it runs as the user "firebird". So, you will create a directory in your home directory (/homes/coursework) that the firebird server can read from and write to. Therefore, execute the following commands:

```
$ cd ~
$ mkdir firebird_db
$ sudo chgrp firebird firebird_db/
$ chmod g+rwx firebird_db/
```

The first command makes the current directory your home directory. The second command creates a new directory. The third command changes the directories group to be the firebird user, and the fourth command allows the firebird group to read, write, and change into that directory.

Next you will create the new database. Open the Flamerobin graphical interface to Firebird. Right click on "localhost" and when the menu appears, click on "create new database". Fill in the prompt according to the picture below:



Display name: students
Database path: /home/coursework/firebird_db/students.fdb
Authentication: Use saved user name and password
User name: SYSDBA
Password: ••••••••••
Charset: NONE
Role:
Page size: 4096
SQL dialect: 3
[No database selected]

The password should be "coursework". Click "Create" and, if you did everything correctly, a new database will be create and it will show up in the Flamerobin GUI. Double click on the new database, named "students", and Flamerobin will log you into the database.

Next, Create two tables called student and takes based on the following definition. Remember, from assignment 2, that in the Flamerobin GUI, you can right-click on the database and select "Execute SQL Statements" and Flamerobin will give you a screen that will let you enter database commands.

| Table Name | Column Name | Type Info | Constraints | Notes |
|---|---|---|---|---|
| **student** | tnum | fixed length string | Required, uniquely identifies a student, must be “Txxxxxxxx” where x is a digit | |
| | ssn | string | Unique, not null, exactly 9 characters each of which must be a digit | |
| | first_name | string | Required, up to 20 characters | |
| | last_name | string | Required, up to 20 characters | |
| | credits | real number | Required, default is 0, Up to 3 digits with no digits after the decimal place | |
| **takes** | tnum | string | Required, uniquely identifies a row when combined with course, only exists if there is a corresponding student in the student table, deleting a student should also delete the takes rows that have the same tnum | A row in this table is uniquely identified by the tnum, course, semester and year together |
| | course | string | Required, up to 12 characters | |
| | semester | string | Required, must be one of FALL, SPRING or SUMMER | |
| | year | string | Required, must be a 4 digit number | |
| | grade | single character | Not required, if given, must be ‘A’, ‘B’, ‘C’, ‘D’, or ‘F’ | |

Required fields should use the NOT NULL constraint, and you should define default values using the DEFAULT keyword. Additionally, use a CHECK constraint and the Firebird SIMILAR TO operator to validate the tnum and the ssn. Documentation on SIMILAR TO can be found here. Use a CHECK constraint and the IN operator to validate the grade. Also, use a CHECK constraint to constrain the semester.  Give the ssn the UNIQUE flag to ensure two students can never have the same social security number. Finally, add the ON DELETE CASCADE syntax to your foreign key in takes for the cascading delete. For questions about syntax, see the Firebird documentation page for CREATE TABLE at http://www.ibphoenix.com/files/60sqlref.html#RSf13665. You should know how to do the rest as you practiced creating tables with some constraints on the previous assignment.

Take a screenshot of the statements that you use to create the tables. Make sure you show that statements being executed. Name the screenshots part1_student.png and part1_takes.png for each of the respective tables.

This easy and straightforward way of defining database schema with integrity controls in place can aid in protecting data from unauthorized changes. Restrictions placed in database schema ensure that it will never allow data to be added that violates its integrity property. The tables you just created will always ensure:

- That both tables have a primary key
- That data types follow necessary formats
- Students with duplicate SSNs cannot be added in student table.
- No student's course data can be added in takes table, who does not exist in students table.
- Critical data like SSN, names cannot be left blank.
- If credits is left blank, it will start off with 0
- Credits can ca be from 0 to 999
- takes cannot retain student data for students removed from takes table.
- A student cannot take a class in that same semester more than once.

Backing up the database

Periodical backups are very useful in case of unintentional errors or intentional attacks occur. Restoring backups, brings data back to correct/valid state.

Exit Flamerobin. Open the command terminal and type the following, and when prompted for a password, enter the password for the root account..

```
gbak -backup -v -user SYSDBA -password coursework
"localhost:/home/coursework/firebird_db/students.fdb"
studentsbackup.fbk
```

This command dumps the given database (students) into the file name that we supplied (studentsbackup.fbk). Take a screenshot of this command successfully completing and save it to the file part1backup.png. You should be able to fit all the output in a single screenshot.

Once you have a backup you can restore it by typing in the following command.

```
gbak -replace -v -user SYSDBA -password coursework studentsbackup.fbk
"localhost:/home/coursework/firebird_db/students.fdb"
```

This command will do the opposite of the previous one and restore the backup studentsbackup.fdb to the database students.

Take a screenshot of this command successfully completing and save it to the file part1restore.png.

## Language Smorgasbord

For Part 2 of this assignment, you will be writing small programs that interact with the MySQL database. You will write these program in the various languages that I presented on the slides in class.

I realize that you have probably have no experience with some of these languages. However, the programs are simple, and you will be able to implement them with a little research that you can do on your own. Make sure that you use good programming practices. For example, make sure that, for each of the following programs, you incorporate error handing. In other words, check return values or catch exceptions, according to the error handing capabilities of the language. Also, make sure you close connections and deallocate variables were appropriate.

The makefile (see "Turn-in" below) should give the hostname (localhost), database name (TTU), database user name (root), and database password (coursework) when it runs each program.

For the problems below that require you to create a table, put **ENGINE=INNODB** at the end of your **CREATE TABLE** statements so that your foreign keys work.

Here are the programming problems:

1. Create a database using Java and JDBC. Note that since the database does not yet exists, So do not supply a database name in your URL. Once connected, send the "CREATE DATABASE TTU" SQL command to the MySQL server. After creating the database, close your connection and open another connection. When you open the second connection, include TTU as the database name in your URL. Then create the table called students with the following columns:

| Column Name | Type |
|---|---|
| **TNumber** | char(8) |
| **FirstName** | varchar(20) |
| **LastName** | varchar(20) |
| **DateOfBirth** | date |

   - The TNumber should be the table's primary key and first and last name are required. Create an index on LastName to speed up queries.
   - Note: It is a good idea to make your program so that you can run it multiple times. To do so, add a command to the beginning of your program, after you connect, that drops the database: "DROP DATABASE IF EXISTS TTU".
   - After creating the table, insert one row into the table. The data for the one row should be:

| TNumber | FirstName | LastName | DateOfBirth |
|---|---|---|---|
| 12345678 | John | Jones | 07/17/1976 |

   After inserting the row, in your Java program show that your insert worked by including a query for the row in the table using a SELECT statement and printing the row from the result set. To print things nicely aligned, use the Java format command (see https://docs.oracle.com/javase/tutorial/java/data/numberformat.html). You will get points deducted if you output does not line up in columns, or if you do not use the Java format command. Do not use tabs ("\t") to align your columns.
   To get your java program to run, you will need to include the MySQL jar file in the classpath. The command below shows what you need to do:
```
java -cp ./:/usr/share/java/mysql.jar program4
```
2. Write a C program using the MySQL C drivers that connects to the MySQL database and inserts 4 more rows. The rows that your program inserts should be as follows:

| TNumber | FirstName | LastName | DateOfBirth |
|---|---|---|---|
| 00012233 | John | Smith | 01/26/1970 |
| 00023052 | Jane | Doe | 04/24/1986 |
| 00120330 | Ivan | Ivanoff | 02/29/1974 |
| 00001203 | Billy | Williams | 09/19/1992 |

   **Remember that dates are entered in the following format: 'YYYY-MM-DD'.**
   To show that your inserts work, in your C program, query for all the rows in the table and print them. Make sure your columns are all nicely aligned. You will get points deducted if you output does not line up in columns. Do not use tabs ("\t") to align your columns.
3. Write a TCL program using the Java JDBC MySQL driver that connects to the MySQL database and queries for all students in the database with a last name that starts with a capital letter 'D'. The program should print the results to the screen nicely formatted, i.e. every column lined up.

Use the TCL format command to align your columns (see https://www.tcl.tk/man/tcl8.6/TclCmd/format.htm).

4. Write a Python program that connects to the database and add a new table called grades with the following columns:

| Column Name | Type |
|---|---|
| **TNumber** | char(8) |
| **CourseID** | char(7) |
| **Semester** | char(1) |
| **Year** | numeric(4) |
| **Grade** | char(1) |

The TNumber is a foreign key that references the TNumber column in the Students table. Next, insert the following rows:

| TNumber | CourseID | Semester | Year | Grade |
|---|---|---|---|---|
| 00012233 | CSC3300 | F | 2013 | A |
| 00023052 | MAT1910 | F | 2011 | C |
| 00001203 | CSC1310 | S | 2013 | A |
| 00120330 | MAT1910 | S | 2012 | D |
| 00120330 | CSC1310 | S | 2012 | F |
| 00120330 | MAT1910 | F | 2012 | A |
| 00120330 | CSC1310 | F | 2012 | D |
| 00012233 | CSC1310 | F | 2011 | C |
| 00012233 | CSC1310 | S | 2012 | C |
| 00012233 | CSC1310 | F | 2013 | B |
| 00001203 | CSC2310 | S | 2012 | D |

To show that your inserts worked, in your Python program, query for all the rows in the table and print them. Make sure your columns are all nicely aligned. Use the older Python % operator as shown in class, or use the newer string format method (docs are at https://docs.python.org/2/tutorial/inputoutput.html) to format your output.

5. Write a Lua program that LuaSQL MySQL drivers to connect to the database and query for all students. The program should print out the first and last name of the student and "YES" or "NO" depending on whether the student can take CSC2310. For the purposes of this assignment, a student can take CSC2310 if they have already passed CSC1310 with at least a grade of 'C'. To get any credit for your solution, your single SQL query must return rows containing the student name, and YES or NO. In other words, do not process the query results in Lua other than to print them, and use only a single query to get the results. The output should be nicely formatted (every column lined up). Use the Lua format command to do so. See the bottom of the web page at the following link for examples (it works like printf() in C): http://www.lua.org/pil/20.html. Do not print a student's name more than once.

6. Write a PHP program that connects to the database using the PHP MySQL drivers (not the ADO drivers - those drivers are not installed in your VM) and displays all students that have taken the same course more than once. Also list the students highest grade for the course. You must use a single query to obtain your results, or you will get no credit for this assignment. In other words, do not process the results in PHP to determine which students to display. You should display the student's tnumber, course_id, attempts, and highest grade in an HTML table. You will have to

handle the logic for printing a name only once in your PHP code (not SQL). So, your output should look like the following:

| TNumber | CourseID | Attempts | HighestGrade |
|---|---|---|---|
| 00120330 | MAT1910 | 2 | A |
|  | CSC1310 | 2 | D |
| 00012233 | CSC1310 | 3 | B |

You can run your PHP code like so:

```
$ php program6.php > out.html
```

Then, you can open Midori or Iceweasel and view the file out.html to see the resulting HTML table. If you are using iceweasel, the command will both run your program and show its output in the browser automatically:

## Turn-in

Follow the following instructions or you will get points deducted from your grade. You will create a top-level directory called yourname_hw3 (where yourname is your Moodle user name). Under this directory, you will create a directory for each program. The directories will be called program1, program2, ..., program6. You will place your program code in the appropriate subdirectory. You will also place a Makefile in the toplevel yourname_hw3 directory. I should be able to run your programs using this Makefile. For example, I will type "make program1" at the Unix prompt, and the Ruby program will run, I will type "make program2", and the Python program will run, etc. Note that your Makefile must compile as well as run the Java, C#, and C++ programs. Put your screenshots from Part1 in to the toplevel yourname_hw3 directory. Compress the entire toplevel yourname_hw3 directory and submit it via iLearn.

Following is a partial example for what your top level Makefile should look like:

```
# Top level makefile. Do NOT cut and paste from this web page.
# Note that indentation is a single tab character.
.PHONY: program1 program2 program3 program4 program5 program6
HOST=localhost
DATABASE=student
USER=root
PASSWORD=coursework
export HOST DATABASE USER PASSWORD
program1:
	make -C program1
# Note: in the line above, put a literal <tab> character to indent
# "make -C program1"
# put targets for program 2 and 3 here
# ...
```

The Makefile in your program1 directory should look something like:

```
.PHONY: run
all: run
```

```
run: program1
	./program1 $(HOST) $(DATABASE) $(USER) $(PASSWORD)
program1: program1.o
	g++ -o program1 program1.o
program1.o: program1.cpp
	g++ -c program1.cpp
```

The Makefile in your program2 directory should look like

```
all:
	tclsh program2.tcl $(HOST) $(DATABASE) $(USER) $(PASSWORD)
```

You should be able to figure out the rest. Note that for program 6, your Makefile should run the PHP script, produce the HTML output in a file, and then run the browser to display the HTML table