


1. System setup

For this project you will need MySQL Server and a GUI-based MySQL Workbench to access the server.

- MySQL server is available for your use in our labs. You will be able to connect to it and use it from anywhere. (Do keep in mind that before accessing any ISU computer from off campus you need a secure connection via a VPN client.). For each of you your own database, user name, and password has been created by our System Support Group. This information will be made available to you.
- For the GUI-based MySQL Workbench, you have two options. (i) Install it on your computer from <http://dev.mysql.com/downloads/workbench/>. Many of you would perhaps prefer this option. (ii) The one available in our labs. This one is also available remotely from your computer via remote desktop connection to csts.cs.iastate.edu.

To set a connection to YOUR database, start MySQLWorkbench GUI, click on  (+ in a circle) and proceed. Set the Hostname to mysql.cs.iastate.edu, Port to 3306, your database name, username, and password provided to you. A connection icon will be created for you so you do not have to reenter the needed settings everytime. You may find the short YouTube video: <https://www.youtube.com/watch?v=x9nkjPZHENA> helpful.

2. Project goal

The database given to you is empty. In this project you create and populate tables in the database and then query and modify the database. If something goes wrong, you can drop all tables - one table at a time, empty the database, and resume your development.

3. Create and populate a working folder

Create a folder on your computer. In this folder store 6 XML files from [UniversityXML.zip](#).

4. Project description

This project consists of three parts A, B, and C described below. There are 30 items. **Make sure that you provide a title for every item in form of a comment.**

It is advised that you create a batch of commands ending in semicolon (;). **>>> Make sure that you read the tips listed below carefully. <<<**

A. Create tables

GUI-based MySQL Workbench. In this part you will develop 6 create table statements. Some details of the table will follow that will give you all the clues you need to write the create table statements. In addition, two out of six create table statements have been provided to you as samples. **You should exercise extra caution in defining the primary and foreign keys correctly;** otherwise, you may face unexpected problems, especially when you populate the tables in items 7-12.

- `Person (Name, ID, Address, DOB)`
- `Instructor (InstructorID, Rank, Salary)`
- `Student (StudentID, Classification, GPA, MentorID, CreditHours)`
- `Course (CourseCode, CourseName, PreReq)`
- `Offering (CourseCode, SectionNo, InstructorID)`
- `Enrollment (CourseCode, SectionNo, StudentID, Grade)`

Item 1: The Person table. The Person table consists of the attributes Name, ID, Address, and DOB (date of birth). The types of these attributes are char(20), char(9), char(30), and date, respectively. ID will be the primary key of the table, and it cannot have a null value. These requirements could be expressed in terms of the following create statement:

```
create table Person (
  Name char (20),
  ID char (9) not null,
  Address char (30),
  DOB date,
  Primary key (ID))
```

>>> **A tip for quick start on this project:** Initially you can exclude "primary key", "foreign key" and "references" in your create table statements in Items 1-6. You can add these features later on. As an example, initially the above create table statement would be as follows:

```
create table Person (
  Name char (20),
  ID char (9) not null,
  Address char (30),
  DOB date)
```

Item 2: The Instructor table. The Instructor table consists of InstructorID, a 9 character non-null attribute that serves as the primary key and also references ID in the Person table. Other attributes are Rank and Salary. Rank can have up to 12 characters and Salary is an integer.

Item 3: The Student table. The Student table has StudentID that has same requirements as InstructorID. The Student table has four additional attributes: Classification that is a string of up to 10 characters, GPA that is a double, MentorID, consisting of 9 characters, references InstructorID in Instructor table, and CreditHours that is an integer.

Item 4: The Course table. The Course table has a 6 character non-null attribute called CourseCode, a 50 character CourseName, and 6 character PreReq, representing prerequisite of a course. Note that a course can have several prerequisites. This is why CourseCode alone cannot be a key. If a course has no prerequisites, the string ♦None♦ is entered. For a given course a tuple, will exist for every prerequisite for the course.

Item 5: The Offering table. The Offering table contains three non-null attributes CourseCode, SectionNo, and InstructorID that are of type char(6), int, and char(9). A CourseCode in the Offering table should appear in the Course table. This could be enforced by an integrity constraint of the form "check (CourseCode in (select ...))". But it seems that Microsoft Access or the ODBC driver that we are using will not support it. Therefore, we will ignore this requirement. The InstructorID references InstructorID in Instructor table. The primary key for this table will be formed using CourseCode and SectionNo attributes.

Item 6: The Enrollment table. The Enrollment table consists of four non-null attributes CourseCode, SectionNo, StudentID, and Grade, with types char(6), int, char(9), and char(4), respectively. CourseCode and SectionNo reference the Offering table. StudentID references the Student table. The primary key for this table will consist of CourseCode and StudentID attributes. SectionNo is not included in the primary key. (Why?) Note that we would expect that a CourseCode, SectionNo pair in the Offering table must occur in the Course table. This requirement will not be captured by our design. For your reference, the create statement for Enrollment table is given below.

```
create table Enrollment (
  CourseCode char(6) NOT NULL,
  SectionNo int NOT NULL,
  StudentID char(9) NOT NULL references Student,
  Grade char(4) NOT NULL,
  primary key (CourseCode, StudentID),
  foreign key (CourseCode, SectionNo) references Offering(CourseCode, SectionNo))
```

B. Populate the database and verify the database state

After creating these tables, you need to populate them. You already have the data in form of XML files that you have stored somewhere. say C:\MyDataFolder.

For populating tables from XML data, a MySQL command is available to you. The following command will load the data from Person.xml in the Person table. (Note that this may not be a standard SQL command.)

```
load xml local infile 'C:/MyFolder//Person.xml'
into table Person
rows identified by <Person>;
```

Items 7-12. Load Person, Instructor, Student, Course, Offering, and Enrollment tables.

C. Develop the SQL commands

Now you will develop a sequence of SQL commands. You should develop these commands sequentially, a single command at a time ending every command in a semicolon(;). .

Some ad hoc queries for the database are stated below in English. **You should express each query as a single SQL statement, possibly containing sub-queries, and execute them. This rule applies to all queries, but not to updates.**

Item 13. List the IDs of students and the IDs of their Mentors for students who are junior or senior having a GPA above 3.8.

Item 14. List the distinct course codes and sections for courses that are being taken by sophomore.

Item 15. List the name and salary for mentors of all freshmen.

Item 16. Find the total salary of all instructors who are not offering any course.

Item 17. List all the names and DOBs of students who were born in 1976. The expression "Year(x.DOB) = 1976", checks if x is born in the year 1976.

Item 18. List the names and rank of instructors who neither offer a course nor mentor a student.

Item 19. Find the IDs, names and DOB of the youngest student(s).

Item 20. List the IDs, DOB, and Names of Persons who are neither a student nor a instructor.

Item 21. For each instructor list his / her name and the number of students he / she mentors.

Item 22. List the number of students and average GPA for each classification. Your query should not use constants such as "Freshman".

Item 23. Report the course(s) with lowest enrollments. You should output the course code and the number of enrollments.

Item 24. List the IDs and Mentor IDs of students who are taking some course, offered by their mentor.

Item 25. List the student id, name, and completed credit hours of all freshman born in or after 1976.

Now we you have a sequence of insertions, deletions and updates to the database. To confirm that such statements had the desired effect on the database, you will also execute some queries. Insertions are specified by some values given to you and may lead to **insertions of tuples in several tables in a specific sequence**. For example, before inserting a student tuple in Student table, you may have to insert him/her as a person in Person

table and make sure that the mentor of the student exists as an instructor. **Similar remarks apply to deletions and updates. Therefore, approach such items carefully.**

Item 26. Insert following information in the database: Student name: Briggs Jason; ID: 480293439; address: 215 North Hyland Avenue; date of birth: 15th January 1975. He is a junior with a GPA of 3.48 and with 75 credit hours. His mentor is the instructor with InstructorID 201586985. Jason Briggs is taking two courses CS311 Section 2 and CS330 Section 1. He has an **A** on CS311 and **A-** on CS330.

After executing the insert statement for the above item, execute the following query statements.

```
Select *
From Person P
Where P.Name= 'Briggs Jason';

Select *
From Student S
Where S.StudentID= '480293439';

Select *
From Enrollment E
Where E.StudentID = '480293439';
```

Item 27. Next, delete the records of students from the database who have a GPA less than 0.5. Note that it is not sufficient to delete these records from Student table; you have to delete them from the Enrollment table first. (Why?) On the other hand, do not delete these students from the Person table.

After finishing the above, execute the query statement given below.

```
Select *
From Student S
Where S.GPA < 0.5
```

Item 28. In this part you update the Instructor table. To confirm the update, you execute two queries before and after making the update. First, execute the SQL query statements that are given below.

>>> This query is optional, its up to you to do it or not do it, there is neither any deduction for not doing it or extra credit for doing it.

```
Select P.Name, I.Salary
From Instructor I, Person P
Where I.InstructorID = P.ID
and P.Name = 'Ricky Ponting';

Select P.Name, I.Salary
From Instructor I, Person P
Where I.InstructorID = P.ID
and P.Name = 'Darren Lehmann';
```

Second, update Instructor table as follows: For instructors, Ricky Ponting and Darren Lehmann, if they five or more students whose GPA is greater than 3.0, then raise their Salary by 10%.

Third, execute the above queries again. This is to help verify that the intended raises did take place.

Item 29. Insert the following information into the Person table. Name: Trevor Horns; ID: 000957303; Address: 23 Canberra Street; date of birth: 23rd November 1964. Then execute the following query:

```
Select *
From Person P
```

```
Where P.Name = 'Trevor Horns'
```

Item 30. Delete the record for Jan Austin from the Person table. If she is a student or an instructor, you should do the deletion with usual care. Then execute the following query:

```
Select *  
From Person P  
Where P.Name = 'Jan Austin'
```

5. Some tips

The quality of GUI features of MySQL workbench is mixed - it ranges from being very good to not so good. It is recommended that you should try to **enter SQL statements directly in text form rather than relying the GUI elements** to compose them in bits and pieces. For example in my opinion, the GUI elements for composing a create table statement is simply horrible and NOT helpful. Just enter the create table statements directly from the keyboard. (The number of problems with the GUI elements in this context are quite a few and not worthy of discussion here.)

The best tip in general that I can offer you for this project is to think that you are developing **a batch of SQL commands separated by semicolons**. Think that at anytime, you have already developed some commands and now you are adding a new command to your batch. At anytime you can highlight one or more commands and execute them. Although this can lead to some undesired problems, but you can turn it around completely in your favor. **Here is a "nasty trick"**: start your batch with six "drop table" statements. Drop Enrollment, Offering, Course, Student, Instructor, and Person tables in that order. Then follow it with the SQL statements you are asked to develop. Keep on saving your batch frequently. When you in doubt about a command you are developing. Highlight all the preceding commands and execute them. This is like starting with a clean slate every time you need to.

A caution about the "nasty trick". In practice you do not drop tables on the fly. Some tables may stay around for years, perhaps decades. The objective here is to learn like constructing buildings out of Lego pieces that you can dismantle at will - but you cannot dismantle real buildings in order to make some additions.

6. Submission

Step 1. The electronic submission. Submit **the text file containing your batch of commands on Blackboard** before the deadline. As stated above, make sure that you have provided a title for every item in form of a comment. The items numbers must be consistent with the project statement. It is a good idea to execute the entire batch in one go before the submission.

Step 2. The hardcopy submission. Print your commands, on the top right hand corner enter your name and the first letter of your last name, and submit it in the class. This should be repeated by each member of the team per line.