SQL Practice - Command Line

Practice SQL DDL and DML using the CL Client

Fundamentals:

Connect to MySQL:

- *$ mysql -u root -p*

Select database (or create one):

- *> show databases; // or > create database dbname;*
- *> use dbname;*

Display tables (or create them):

- *> show tables;*

Execute Queries:

- *> [select, insert, update, etc.]*

Helpful Commands:

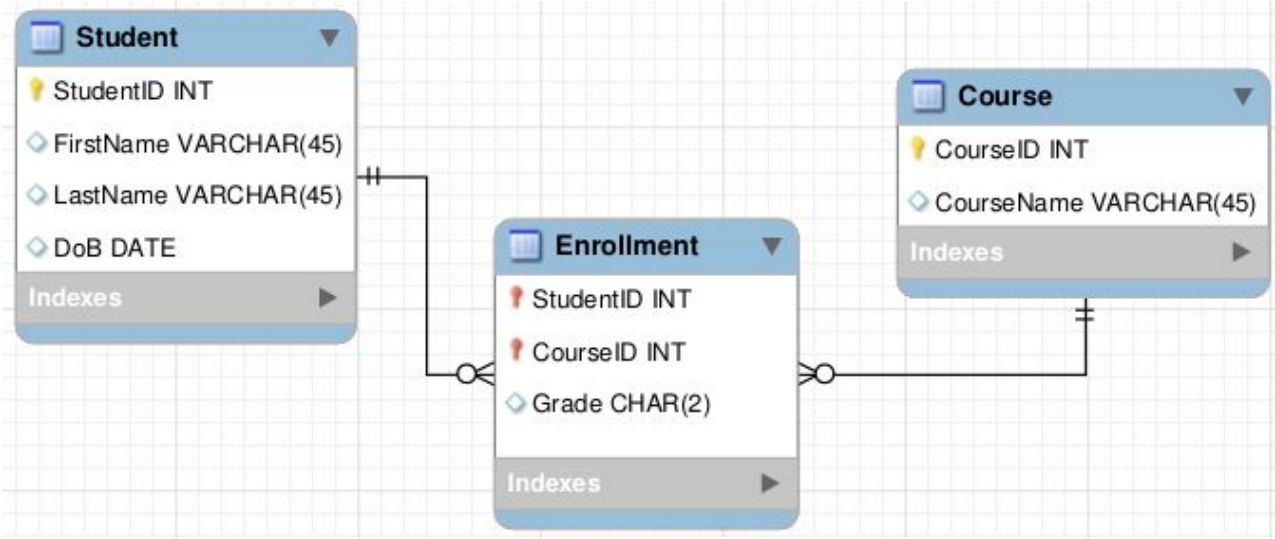
- *> \# (enables autocomplete)*
- *> describe table; (shows structure of the table)*
- *> help (gives a list of all MySQL commands)*

Exit MySQL Shell:

- *> exit (or quit)*

Practice Exercise:

1. Create a database called *school* and use that database.
2. Create tables in your database according to this EER Diagram. Auto-increment the StudentID in the Student table.



- Your output for describing each table should be:

```
mysql> describe Student;
+-----------+-------------+------+-----+---------+----------------+
| Field     | Type        | Null | Key | Default | Extra          |
+-----------+-------------+------+-----+---------+----------------+
| StudentID | int(11)     | NO   | PRI | NULL    | auto_increment |
| FirstName | varchar(45) | YES  |     | NULL    |                |
| LastName  | varchar(45) | YES  |     | NULL    |                |
| DoB       | date        | YES  |     | NULL    |                |
+-----------+-------------+------+-----+---------+----------------+
4 rows in set (0.00 sec)
```

```
mysql> describe Course;
+------------+-------------+------+-----+---------+-------+
| Field      | Type        | Null | Key | Default | Extra |
+------------+-------------+------+-----+---------+-------+
| CourseID   | int(11)     | NO   | PRI | NULL    |       |
| CourseName | varchar(45) | YES  |     | NULL    |       |
+------------+-------------+------+-----+---------+-------+
2 rows in set (0.00 sec)
```

```
mysql> describe Enrollment;
+-----------+---------+------+-----+---------+-------+
| Field     | Type    | Null | Key | Default | Extra |
+-----------+---------+------+-----+---------+-------+
| StudentID | int(11) | NO   | PRI | NULL    |       |
| CourseID  | int(11) | NO   | PRI | NULL    |       |
| Grade     | char(2) | YES  |     | NULL    |       |
+-----------+---------+------+-----+---------+-------+
3 rows in set (0.00 sec)
```

(The INT display width is defaulted to 11 digits for MySQL. The INT display width doesn't do anything unless a column is UNSIGNED ZEROFILL - where numbers are zero-padded on the left.)

3. Insert several rows of students into your Student table. (Note: Since StudentID will be auto-incremented, specify the other three columns in your insert statement where you will actually insert data). Example of a Student table:

```
mysql> Select * from Student;
+-----------+-----------+------------+------------+
| StudentID | FirstName | LastName   | DoB        |
+-----------+-----------+------------+------------+
|         1 | Allison   | Rodenbaugh | 1996-10-19 |
|         2 | John      | Doe        | 2000-01-01 |
|         3 | John      | Smith      | 1999-10-01 |
+-----------+-----------+------------+------------+
3 rows in set (0.00 sec)
```

4. Insert several rows of classes into your Course table. Example of a Course table:

```
mysql> Select * from Course;
+----------+-------------------------------------+
| CourseID | CourseName                          |
+----------+-------------------------------------+
|     1300 | Intro to Programming                |
|     2270 | Data Structures                     |
|     2400 | Computer Systems                    |
|     3104 | Algorithms                          |
|     3155 | Principles of Programming Languages |
+----------+-------------------------------------+
```

5. The Enrollment table is an *Association Entity* to the Student and Course tables (A.K.A. child table). This means that any insertions into Enrollment should only contain StudentIDs and CourseIDs that already exist in the Student/Course tables, since the child's Foreign Keys reference the parents' Primary Keys. ALTER the Enrollment table to identify our Foreign Keys that reference the parent tables so that the insertions can only contain valid StudentIDs and CourseIDs. Use an alteration that allows you to give your foreign keys a name. Naming foreign keys: "[table_name]_fk_[field_name]" is good practice. Else MySQL will name it "tableName_ibfk_[index]" (ibfk = innodb FK) which is ambiguous for future alterations.

   ***To list your existing [FK] constraints, run this query: Select constraint_name from information_schema.table_constraints where table_name='Enrollment';

6. Enroll your students into several of your courses by inserting rows into the Enrollment table. Ignore the *Grade* attribute for now by specifying the other two columns you are inserting into. Attempt to add students/courses that do not already exist in your Student and Course tables to see if you get an error due to your FK constraints (as you should). Ex:

```
mysql> INSERT INTO Enrollment (StudentID, CourseID) VALUES (1, 4000);
ERROR 1452 (23000): Cannot add or update a child row: a foreign key constraint fails
(`school`.`Enrollment`, CONSTRAINT `Enrollment_ibfk_2` FOREIGN KEY (`CourseID`) REFER
ENCES `Course` (`CourseID`))
mysql>
```

   (Invalid insertion - I do not have a course in my Course table where CourseID=4000)

   - After valid insertions, your Enrollment table may look something like this:

```
mysql> Select * from Enrollment Order by StudentID;
+-----------+----------+-------+
| StudentID | CourseID | Grade |
+-----------+----------+-------+
|         1 |     2400 | NULL  |
|         1 |     3155 | NULL  |
|         2 |     1300 | NULL  |
|         2 |     2270 | NULL  |
|         3 |     1300 | NULL  |
+-----------+----------+-------+
5 rows in set (0.00 sec)
```

7. Let's assume that grades will only be entered into the Enrollment table using an UPDATE statement *after* the semester is over. Create a TRIGGER for your Enrollment table to restrict UPDATE entries into your Grade column. We should only allow valid letter grades (or W for withdrawn) to be inserted into this column. For example, if the *Grade CHAR(2*) value attempting to be inserted is **not** in the set {A, A-, B+, B, B-, C+, C, C-, D+, D, D-, F, W}, then display an error message and *do not* allow the update statement to run. Define a new delimiter to write your trigger statement in the command line.

8. UPDATE the grades of several of your students for several of the courses they are enrolled in. Grades should be in the form *CHAR(2)* (i.e. A, A-, B+, B…. or W for Withdrawn). Attempt to add invalid grades as well to make sure your UPDATE trigger works and throws your error message. For example:

```
mysql> UPDATE Enrollment SET Grade="K" WHERE StudentID=3 AND CourseID
=1300;
ERROR 1644 (45000): Error: Invalid grade. Grade must be in the form:
A, A-, B+, B, ... or W if Withdrawn.
mysql>
```

- After updating some of your rows with valid grades, your Enrollment table may look something like this:

```
mysql> Select * from Enrollment Order by CourseID;
+-----------+----------+-------+
| StudentID | CourseID | Grade |
+-----------+----------+-------+
|         2 |     1300 | C-    |
|         3 |     1300 | W     |
|         2 |     2270 | A     |
|         1 |     2400 | NULL  |
|         1 |     3155 | B     |
+-----------+----------+-------+
```

9. Display the CourseNames and the Student's First and Last names for all of the courses being taken. Order by CourseID. Your SELECT query should output something like this:

```
+-----------------------------------+-----------+------------+
| CourseName                        | FirstName | LastName   |
+-----------------------------------+-----------+------------+
| Intro to Programming              | John      | Doe        |
| Intro to Programming              | John      | Smith      |
| Data Structures                   | John      | Doe        |
| Computer Systems                  | Allison   | Rodenbaugh |
| Principles of Programming Languages | Allison | Rodenbaugh |
+-----------------------------------+-----------+------------+
```

10. Create a VIEW called *CourseEnrollment* that displays ALL of the CourseIDs and the Count for the number of students enrolled in each of those courses (give an alias). Assume some of your courses may have 0 students enrolled, and find a way to display those courses too. Selecting all rows from your VIEW should output something like this:

```
mysql> Select * from CourseEnrollment Order by StudentCount;
+----------+--------------+
| CourseID | StudentCount |
+----------+--------------+
|     3104 |            0 |
|     2270 |            1 |
|     2400 |            1 |
|     3155 |            1 |
|     1300 |            2 |
+----------+--------------+
```

11. If Students need to be removed from our database, we want all of their corresponding data to be removed as well. An ON DELETE CASCADE constraint will delete corresponding data from all child tables once a row is deleted from a parent table. Use an ALTER TABLE statement to add this constraint to your Enrollment table. *Note: when it comes to altering an existing FK constraint, you first have to drop the FK and then recreate it with the addition of the new clause.*

    (We choose not to add this “cascading deletion” constraint for the *CourseID* FK because even if a course is removed, we may still want to keep track of the student’s grade for that course so they can receive credit).

12. Student 1 just dropped out. Delete Student 1 from your Student table *only*. Then select all rows from your Enrollment table to verify that the corresponding rows were also deleted (testing the DELETE ON CASCADE constraint you added to the table). Your Enrollment table should now **not** contain rows where StudentID=1.

```
mysql> Select * from Enrollment Order by CourseID;
+-----------+----------+-------+
| StudentID | CourseID | Grade |
+-----------+----------+-------+
|         2 |     1300 | C-    |
|         3 |     1300 | W     |
|         2 |     2270 | A     |
+-----------+----------+-------+
```