Introduction to MySQL

Reading / References

- Concise Guide to Databases: A Practical Introduction[1] chapter 4
- Introduction to SQL: Mastering the Relational Database Language[2]
- MySQL[3]
- Stanford free online course on databases[4]

Notes

There are numerous relational database systems out there, all with similar functionalities, and all supporting the SQL standard. So talking to one of them is not all that dissimilar to talking to any of the others.

- MySQL[5] is one of the most popular openly available versions. MariaDB[6] is a completely open-source fork of it.
- SQLite[7] is a much lighter SQL version which is often used by Android apps for local storage.
- Microsoft SQL Server[8] is a Microsoft-owned proprietary SQL database that also sees extensive use.
- PostgreSQL[9] is another free and open-source database that sees extensive use, and comes by default in many Linux distributions and also for macOS Server.
- Many cloud services include some kind of database system.

We will be using mySQL WorkBench to inteface with mySQL. You should find it under the Developer tab in your system's start menu. If you want to work on your own computers instead, you would need to install it manually, and possibly even install mySQL if it is not already installed.

The first thing we're going to need is a connection to a database. You all have such a connection set up in vault, which is the same place that you can store webapp-related information. We will tell mySQL WorkBench about it.

[1] https://acm.skillport.com/skillportfe/main.action#summary/BOOKS/RW$31069:_ss_book:76983
[2] https://www.safaribooksonline.com/library/view/introduction-to-sql/0321305965/
[3] https://www.safaribooksonline.com/library/view/mysql-fifth-edition/9780133038552/
[4] https://lagunita.stanford.edu/courses/Home/Databases/Engineering/about
[5] https://www.mysql.com/
[6] https://mariadb.org/
[7] https://www.sqlite.org/index.html
[8] https://www.microsoft.com/en-us/sql-server
[9] https://www.postgresql.org/

Setting up mySQLWorkbench

When you first open up mySQLWorkbench, there is a "connections" section on the top left, with a plus sign to create a new connection. Here's what you need to specify in it:

- Use any name you like for the connection.
- The connection method should be standard TCP/IP.
- The host name should be vault.hanover.edu
- Your username is your Hanover/email login.
- You do not need to specify a password right away, though you can if you want to. You will be asked for a password when you try to connect.
- When it asks you for a password, that password is HC_XXXXXX where the XXXXXX is replaced by your 6-digit id number. We will learn later how to change the password.
- The "default schema", another name for "database", should be your login name. Out of the many databases/schemas that are available in vault, this one has been created for you. You will be placing all your tables etc there.
- Double-Click on the connection to connect and start working with queries.

For now we will be using the workbench only as a place to write scripts. Later on we may do more with it.

If you don't have an open query page yet, open one via the File menu.

Basic MySQL commands

We will explore now various SQL commands. SQL is essentially just another programming language, though it differs from most programming languages in that it is very **declarative** in nature. We tell it *what* we want it to return, but not *how* to do it.

Here is a list of the main SQL commands. For reference, here is also a quick cheatsheet[10].

SHOW TABLES Lists the tables present in the database/schema.
DESCRIBE TABLE Returns information about a table.
CREATE TABLE Used to create a new table.
DROP TABLE Used to remove/delete a whole table.
ALTER TABLE Used to make changes to a table's definition (e.g. add a new column, or set an index or add a constraint).
INSERT Used to insert new values/rows into a table
SELECT Probably the most used of all the commands. Returns some results according to a query. Can also be used as part of other commands.
UPDATE Used to change particular parts of particular rows.
DELETE Used to delete whole rows based on some query.

We will now consider most of these in greater detail.

[10] http://cse.unl.edu/~sscott/ShowFiles/SQL/CheatSheet/SQLCheatSheet.html

CREATE TABLE We will create a new table as an example of using the CREATE TABLE command. We will use it to showcase some of the different variable types in SQL. Before we proceed, it is customary to use all capitals for the various SQL commands, and to use lowercase for anything else. It is purely a convenience for readability.

```
CREATE TABLE students (
    id     INT  UNIQUE   NOT NULL AUTO_INCREMENT,
    login VARCHAR(20) UNIQUE NOT NULL,
    first VARCHAR(20),
    last  VARCHAR(20),
    credits INT DEFAULT 0,
    gpa     DOUBLE DEFAULT 0,
    PRIMARY KEY (id)
);
```

When you run this, it will create a new table, called students, with 6 different attributes/columns:

- An id, which is also the primary key, and must be an integer and "unique" (so that no two rows can have the same id). We also set that row to auto-increment, a feature specific to MySQL. This way we will not have to find the newest id to insert.
- A login, which must be not null and must also be unique. it is a character string of varied length, at most 20.
- Attributes first and last for the students' first and last name. They are allowed to be null.
- Number of credits the student has so far. It is an integer, and defaults to 0.
- The student's gpa, a double-precision number defaulting to 0.

Note that every SQL command ends with a semicolon. The whitespace is all optional.

For more practice let us also create courses:

```
CREATE TABLE courses (
    id      INT  UNIQUE   NOT NULL AUTO_INCREMENT,
    prefix CHAR(4) NOT NULL,
    no      INT NOT NULL,
    title   VARCHAR(55) NOT NULL,
    credits INT NOT NULL DEFAULT 4,
    UNIQUE KEY fullCode (prefix, no),
    PRIMARY KEY (id)
);
```

Some points to note:

- We made the prefix into a CHAR(4) type, which means that we cannot store more than 4 characters in it but that it should also use 4 bytes for it, even for shorter prefixes.
- We introduced a unique key for the pair of (prefix, no). We called that key fullCode, in case we want to later delete it. This will impose a constraint: The system will not allow two different courses to have the same prefix and number, and it will give us an error if we try to create one.

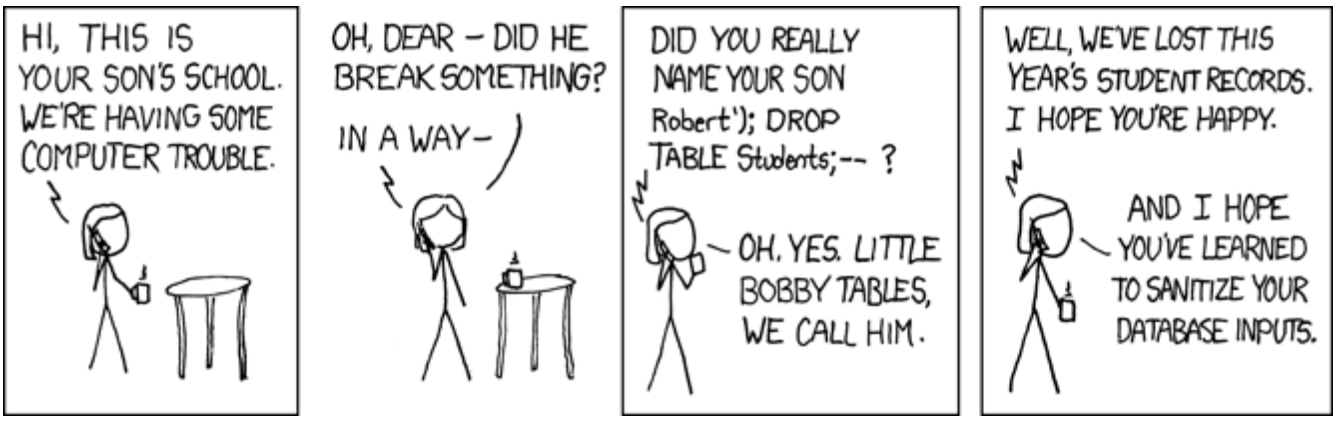

Figure 1: Little Bobby Tables

Now we come to the complicated part. We want to create the concept of students enrolled in courses. Since every student can enroll in many courses, and many courses can have many students in them, this is a many-to-many relationship. To express such a relationship we need a third **association table**. At its simplest this table will contain pairs of a student's id and a course's id. It may optionally contain more information, for instance whether the student is taking the course for credit, and what their grade in the course is (if the course is completed). We will create this new table, and link it to the other two via **foreign keys**.

```
CREATE TABLE enrollments (
    student_id INT NOT NULL,
    course_id INT NOT NULL,
    letter_grade  CHAR(2),
    point_grade   DOUBLE,
    FOREIGN KEY (student_id) REFERENCES students(id) ON DELETE CASCADE,
    FOREIGN KEY (course_id)  REFERENCES courses(id) ON DELETE CASCADE,
    PRIMARY KEY (student_id, course_id)
);
```

The new item here is the FOREIGN KEY line, which forces the corresponding column in enrollment to reference the id column in the students table. You would not be allowed to add an enrollment for a student id that doesn't exist. Also we have added ON DELETE CASCADE, which means that if a student is deleted from the students table, that deletion will cascade to the enrollments table, and all the enrollments of that student in courses will also be deleted.

We will only mention in passing the other commands related to tables. DROP TABLE simply removes a table from existence, and you permanently lose that table's contents. So be careful with it. ALTER TABLE can be used to make changes to a table, such as adding a new constraint or creating a new column. We will not discuss these further.

INSERT The INSERT command is used to add new values into a table. We will use it now to add numerous students and courses into the system. We will revisit it later when we combine it with SELECT queries and use the result of a SELECT query as the input to an INSERT.

The basic syntax of INSERT looks as follows:

```
INSERT INTO students (login, first, last) VALUES
    ("somebodyj1", "Joe", "Somebody"),
    ("somebodyj2", "Joel", "Somebody"),
    ("otherp1", "Peter", "Other"),
    ("otherm1", "Mary", "Other"),
    ("doem1", "Mary", "Doe"),
    ("doep1", "Peter", "Doe"),
    ("doed1", "David", "Doe");
```

So we have to indicate the table we want to insert to, and after that you typically include a list of which columns you will be specifying with the values. If you omit it, then the system would expect you to provide values for all attributes, and in the order in which they appear in the definition. It's always a good idea to specify them like we did above.

Let's add some courses:

```
INSERT INTO courses (prefix, no, title) VALUES
    ("MAT", 121, "Calculus 1"),
    ("CS", 220, "Intro to CS"),
    ("MAT", 122, "Calculus 2"),
    ("MAT", 221, "Calculus 3"),
    ("CS", 223, "Data Structures");
```

We will leave the enrollment of students to classes for later, after we discuss SELECT queries.

SELECT SELECT is the main way to read information out of the database. The simplest call is one that returns all entries from a table:

```
SELECT * FROM students;
```

This will print out the entire student table. Instead of an asterisk, we can specify which columns we want to to show:

```
SELECT first, last
FROM students;

SELECT last FROM students;
```

Notice that with that last one we saw the same values multiple times. Sometimes you want that to happen, and sometimes you don't. You can control that behavior via the keyword DISTINCT.

```
SELECT DISTINCT last
FROM students;
```

SELECT extras There are a couple of extra clauses we can add to a SELECT clause. One is a WHERE clause. For instance we can get the first names of all those whose last name is Somebody:

```
SELECT first
FROM students
WHERE last = "Somebody";
```

Let's go further, and add a second restriction for the first name:

```
SELECT login, first, last
FROM students
WHERE first = "Joe"
AND last = "Somebody";
```

We can also add an ordering:

```
SELECT first, last
FROM students
ORDER BY last, first;
```

The above line will order by last name and then break ties by first name. There is one more clause we can add to the SELECT, but we will look at that a bit later.

Practice:

1. Produce a list of the course prefixes and numbers, ordered by prefix first then by number.
2. We want the same list but now with descending prefix order, while still using ascending number order. Search online for how to do that.

INSERTS with SELECT query Let's practice some more complex inserts where the values are determined via a SELECT query. The idea is that instead of listing tuples of values, we will be placing a SELECT query, like so:

```
INSERT INTO enrollments (student_id, course_id)
SELECT id, 1
FROM students;
```

We just enrolled all students to the course with id 1.

Let's go a bit further. We will now enroll all students with last name "Somebody" to all CS courses. We will also add a safeguard in case some of those already existed, that's the keyword IGNORE. It tells the system to ignore entries that are already present or are otherwise incorrect.

```
INSERT IGNORE INTO enrollments (student_id, course_id)
SELECT s.id, c.id
FROM students AS s, courses AS c
WHERE s.last = "Somebody"
AND c.prefix = "CS";
```

So let's talk about this one, as it is considerably more complicated:

- The word IGNORE says if any problems are encountered in some rows, they are to be ignored. So if there was a pair that already existed, it will not try to add it.

- In the FROM clause we actually have two tables, and we use the AS construct to give them shorthand names. When we put multiple tables in the FROM clause, the database will look at all possible combinations of values. So in this case it will start off with all pairs of a student and a course, then use the WHERE clause to filter some out.
- In the SELECT clause we use those shorthand names to make clear which "id" we are talking about.