Networks and Database Systems

Professor Tom Ellman
Lecture 14
Persistence

- Where do all those Java objects go when the program exits?
- We could write them to files and read them back next time we run the program.
- But this is combersome.
- We need a general solution.
Relational Databases

- A database is collection of tables.
- A table includes:
  - A list of column names and data types.
  - A collection of rows that form the data of the table.
- The tables are manipulated using a query language:
  - SQL: Simple Query Language.
  - SQL: Structured Query Language.
- MySQL: Open source relational database and query language from Sun. (Oops! Oracle)
Access to MySQL from Linux Systems

• Command line:
  > ssh yoursql
  ...
  > mysql –u ellman –p
  Enter Password: ******

• Netbeans:
  – Services tab.
  – RMB Databases—NewConnection.
  – Supply host (yoursql) port (3306) and database name.
PhpMyAdmin: Web Access to MySQL

- https://yoursql.cs.vassar.edu/phpmyadmin
- Creation of databases and user accounts.
- Definition of tables in databases.
- Browsing databases and tables.
- Issuing arbitrary SQL commands.
CREATE DATABASE `Vassar` ;

Creating a new database named “Vassar”.
USE Vassar;

Subsequent queries will be evaluated against the Vassar database.
CREATE TABLE `students`(
`student_id` CHAR(9) NOT NULL ,
`last_name` VARCHAR(30) NOT NULL ,
`first_name` VARCHAR(15) NOT NULL
);

SHOW TABLES

Create a table called “students” with three columns: student_id, last_name and first_name.

Showing a description of the tables (and other tables).
ALTER TABLE `students`
ADD `class_year` YEAR NULL ;

Add a new column class_year.
INSERT INTO `students`  
(`student_id`, `last_name`, `first_name`, `class_year`)  
VALUES  
('999999001', 'Doe', 'Jane', '2010'),  
('999999002', 'Doe', 'John', '2012');

SELECT * FROM `students`
Deleting one particular row from the students table, where the row is specified by the value of the student_id column.

Inserting two more rows.
 UPDATE `students` 
 SET `last_name` = 'Schmoe' 
 WHERE `student_id` = '999999002';

Changing the last_name entry in one row of the students table.
Retrieving all students whose `class_year` is 2010.

```
SELECT *
FROM `students`
WHERE `class_year` = '2010'

ALTER TABLE `students`
DROP `class_year`;
```
Creating a table called "course_registration" to hold records of course students are taking.

Changing the table to omit the title column. Why?
INSERT INTO `course_registration` 
(stUDENT_id, course_id, section) 
VALUES 
(999999001,"cmpu375",1), 
(999999003,"cmpu375",1), 
(999999001,"cmpu331",1), 
(999999002,"cmpu331",1), 
(999999001,"cmpu240",1), 
(999999002,"cmpu240",1), 
(999999003,"cmpu240",1);

Inserting several course registrations.
SELECT * FROM `course_registration`

SELECT *
FROM `students` JOIN `course_registration` USING(`student_id`);

Selecting all rows from course_registration table.

Forming a new table by joining students and course registration on equal values of the student_id column.
SELECT *  
FROM `students` JOIN `course_registration`  
USING(`student_id`)  
WHERE course_id = "cmpu375";

SELECT `first_name`, `last_name`  
FROM `students` JOIN `course_registration`  
USING(`student_id`)  
WHERE course_id = "cmpu375";

Selecting all rows in the join of students and course_registration where the course_id column is cmpu375.

Same thing, but projecting only the first_name and last_name columns of the joined table.
A single course may have two course_id values (e.g., cmpu379 and meds379. We need a unique way to identify courses. Here we create a table with one row for each course. We also change the name of a column in the course_registration table. The course_key will be an automatically generated primary key. MySQL will enforce the requirement that no two rows have the same course_key.
Inserting rows into the courses table. Notice that we don’t provide a value of the course_key primary key.

```
INSERT INTO `courses`
(`title`, `units`, `registration`)
VALUES
("Language Theory", 1, "graded"),
("Networks and Databases", 1, "graded"),
("Operating Systems", 1, "graded"),
("Computer Animation", 1, "graded");
```
The course_mirrors table will record the multiple ways of referring to a course using a course tag like cmpu379 or meds379. This operation generates an error. Why?

```sql
CREATE TABLE `course_mirrors` (  
`course_key` INT NOT NULL ,  
`course_tag` CHAR( 7 ) NOT NULL ,  
PRIMARY KEY ( `course_key` )
);

INSERT INTO `course_mirrors`  
(`course_key`, `course_tag`)  
VALUES  
(1,"cmpu240"),  
(2,"cmpu375"),  
(3,"cmpu331"),  
(4,"cmpu379"),  
(4,"meds379");

Error #1062 - Duplicate entry '4' for key 1

The course_mirrors table will record the multiple ways of referring to a course using a course tag like cmpu379 or meds379. This operation generates an error. Why?
ALTER TABLE `course_mirrors`
CHANGE COLUMN `course_key` `course_id` INT NOT NULL;

INSERT INTO `course_mirrors`
(`course_id`, `course_tag`)
VALUES
(4, "meds379");

Error #1062 - Duplicate entry '4' for key 1

Altering the table to change the name of course_key to course_id and making it not a primary key. We still get the error. Why????
DROP TABLE `course_mirrors`;

CREATE TABLE `course_mirrors` (`course_id` INT NOT NULL, `course_tag` CHAR(7));

INSERT INTO `course_mirrors` (`course_id`, `course_tag`) VALUES (1,"cmpu240"), (2,"cmpu375"), (3,"cmpu331"), (4,"cmpu379"), (4,"meds379");

When we rebuild the table from scratch, it works.
SELECT `student_id` FROM `students` 
WHERE `first_name` = "Jane" AND `last_name` = "Doe";

SELECT `course_tag` 
FROM `students` JOIN `course_registration` 
USING(`student_id`) 
WHERE `student_id` = 999999001;

We want to find Jane Doe’s courses. First we find her student_id and then we find the course_tag values of the courses for which she registered.
Here we do the same thing, but we use a variable to store the result of the first query. We use the variable in the second query.

```sql
SET @the_id =
(SELECT `student_id` FROM `students`
    WHERE `first_name` = "Jane"
    AND `last_name` = "Doe");

SELECT `course_tag`
FROM `students` JOIN `course_registration`
USING(`student_id`)
WHERE `student_id` = @the_id;
```
In these queries we ask for all students’ course registrations and then we order them by students’ names. Why does the first query generate duplicates, while the second does not?

```sql
SELECT `first_name`, `last_name`, `course_tag`
FROM `students` JOIN `course_registration`
ORDER BY `last_name`, `first_name`;

SELECT `first_name`, `last_name`, `course_tag`
FROM `students` JOIN `course_registration` USING(`student_id`)
ORDER BY `last_name`, `first_name`;
```
Two students are registering for Computer Animation, in different departments/programs.

```
INSERT INTO `course_registration`
(`student_id`, `course_tag`, `section`)
VALUES
(999999001, "cmpu379", 1),
(999999002, "meds379", 1);
```
Finding all the students who registered for Computer Animation (in either department/program). First we find the unique primary key for Computer Animation in the courses table. Then we use it to look up values in a join of students and course_registration. The embedded select operation is known as a “sub-query”.

```
SET @animation_id = (SELECT `course_key` FROM `courses`
    WHERE `title` = "Computer Animation");

SELECT `first_name`, `last_name`
FROM `students` JOIN `course_registration`
USING(`student_id`)
WHERE `course_tag`
IN (SELECT `course_tag` FROM `course_mirrors`
    WHERE `course_id` = @animation_id);
```
SELECT first_name, last_name  
FROM students JOIN course_registration USING(student_id)  
WHERE course_tag IN  
(SELECT course_tag FROM course_mirrors  
WHERE course_id = (SELECT course_key FROM courses  
WHERE title = "Computer Animation"));

Here we do the same thing, using a sub-subquery.
SELECT course_tag, title
FROM courses JOIN course_mirrors
ON(course_key = course_id);

Generating a table of course titles and their associated course_tag values.
CREATE VIEW `tag_course`
AS SELECT `course_tag`, `title` 
FROM `courses` JOIN `course_mirrors` 
ON (`course_key` = `course_id`);

SELECT * FROM `tag_course`
WHERE `title` = "Computer Animation";

Creating a “view” that represents the table we generated on the last slide. Once we create the view, we can issue queries against it.
The world Database

- city: Name, Population, CountryCode …

- country: Name, Code …

- countrylanguage: CountryCode, Language …
Names of countries in which English is the official language.
SELECT `Name`, `Population` FROM `City`
WHERE `Population` > 7000000;

Names and populations of cities with over 7 million inhabitants.

Names and official languages of cities with over 7 million inhabitants.