

A Gentler Introduction to MySQL Database Programming

Zhizhang Shen *

Dept. of Computer Science and Technology
Plymouth State University

Abstract

This is Part II of the lab notes prepared for the students of *CS3600 Introduction to the Database Systems* for Fall 2009. This part introduces some basic MySQL structures, using the *Student Registration Database* as contained in [1, §3.2].

We show how to define and populate tables in such a database. We then discuss most of the queries as suggested in [1, §5.2], test them out with MySQL (ver 4.1.14), and show the results.

We briefly discuss the use of view in database programming.

We also present a general PHP script that can be used to test out any query related to the above database.

Contents

1	Basic MySQL commands	2
1.1	a GUI interface	7
2	Table definition and population	7
2.1	The <code>Student</code> table	7
2.2	The <code>Professor</code> table	9
2.3	The <code>Course</code> table	10
2.4	The <code>Transcript</code> table	13
2.5	The <code>Teaching</code> table	16

*Address correspondence to Dr. Zhizhang Shen, Dept. of Computer Science and Technology, Plymouth State University, Plymouth, NH 03264, USA. *E-mail address:* `zshen@plymouth.edu`.

3	SQL Queries	19
3.1	Simple queries	20
3.2	Set operations	26
3.3	Nested queries	30
3.4	Aggregation	34
4	On the views	40
5	MySQL and PHP	43

1 Basic MySQL commands

Assume your *MySQL* account name¹ is `j_doe`, you can log into *MySQL* by entering your account information in the turing prompt via the following interaction:

```
[j_doe@turing ~]$ mysql j_doe -p
Enter password:*****
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 123324 to server version: 4.1.12-log
Type help; or \h for help. Type \c to clear the buffer.
mysql>
```

When everything goes smoothly, you should get the last *MySQL* prompt as shown and you are ready to use *MySQL*.

Below are some of the basic *MySQL* commands you need to deal with it. For more of a large collection of *MySQL* commands, please refer to [2].

- You should have already a database whose name is the same as your login name. In general, the following lets you find out all the existing databases:

```
mysql> show databases;
+-----+
| Database      |
+-----+
| another       |
| fear          |
| geography     |
| mysql         |
| registration  |
| shentest      |
```

¹Your personal account for *MySQL* has been set on `turing`, and the relevant information has also been sent to your email account. If you can't find it, or you recently added into this course, please come to talk to me.

```
| test          |
+-----+
7 rows in set (0.00 sec)
```

- Before using a database, you have to create it first. For example, the following creates the database `testDB`:

```
mysql> create database testDB;
Query OK, 1 row affected (0.00 sec)
```

```
mysql> show databases;
+-----+
| Database          |
+-----+
| another           |
| fear              |
| geography         |
| mysql             |
| registration      |
| shentest          |
| test              |
| testdb            |
+-----+
8 rows in set (0.00 sec)
```

- You also have to specify which database to use, before using it. For example, if you want to use the `registration` database, you have to enter the following:

```
mysql> use registration;
Database changed
```

- Show all the tables in the current database that you have chosen to use:

```
mysql> show tables;
+-----+
| Tables_in_registration |
+-----+
| course                  |
| hardclass               |
| professor               |
| student                 |
| teaching                |
```

```
| transcript          |
+-----+
6 rows in set (0.00 sec)
```

- Before using a database table, you have to create it first. The following creates a table aTable:

```
mysql> create table aTable (
  -> old char(10),
  -> another integer);
Query OK, 0 rows affected (0.08 sec)
```

aTable looks like the following:

```
mysql> desc aTable;
+-----+-----+-----+-----+-----+-----+
| Field  | Type   | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| old    | char(10) | YES  |     | NULL    |       |
| another | int(11)  | YES  |     | NULL    |       |
+-----+-----+-----+-----+-----+-----+
2 rows in set (0.00 sec)
```

- It is pretty easy to make mistakes when creating a table. When this happens, we can use the quite flexible **Alter Table** command to correct them. For example, the following changes the definition of column old of a table aTable to new Integer, and make it into the primary key.

```
mysql> alter table aTable change old new integer not null primary key;
Query OK, 0 rows affected (0.16 sec)
Records: 0 Duplicates: 0 Warnings: 0
```

It is now indeed changed.

```
mysql> desc aTable;
+-----+-----+-----+-----+-----+-----+
| Field  | Type   | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| new    | int(11) |      | PRI | 0        |       |
| another | int(11) | YES  |     | NULL    |       |
+-----+-----+-----+-----+-----+-----+
2 rows in set (0.00 sec)
```

Note: Alter table has a very rich syntax structure, which allows us to do many different things. For example, the following changes the name of a table `Student` to `student`.

```
mysql> show tables;
+-----+
| Tables_in_registration |
+-----+
| ClassAce                |
| ClassEnrollment        |
| ClassFailures           |
| HardClass               |
| Student                 |
| course                  |
| easyClass               |
| hardclass               |
| professor               |
| teaching                |
| tempT                   |
| transcript               |
| v                       |
+-----+
13 rows in set (0.00 sec)

mysql> alter table Student rename to student;
Query OK, 0 rows affected (0.67 sec)
```

```
mysql> show tables;
+-----+
| Tables_in_registration |
+-----+
| ClassAce                |
| ClassEnrollment        |
| ClassFailures           |
| HardClass               |
| course                  |
| easyClass               |
| hardclass               |
| professor               |
| student                 |
| teaching                |
| tempT                   |
+-----+
```

```
| transcript          |
| v                  |
+-----+
13 rows in set (0.00 sec)
```

For more details, please do check [2].

- You sometimes want to delete a table structure. The following shows how to *drop* a table `t`.

```
mysql> drop table t;
Query OK, 0 rows affected (0.12 sec)
```

Notice that this operation will delete everything, both the structure and the content, of the table to be dropped. Thus, it is the opposite of both `create table` and `insert into`.

If you just want to delete some rows from a table, the syntax is the following:

```
delete from
where
```

For example, if you just want to delete a row from the *student* table such that its `Id` is '111111111', you just need to say the following:

```
delete from Student
where Id='111111111';
```

- We often do something in one system, then switch it to a production system, thus the need for saving all the stuff we do and reproduce it somewhere else. This is called a *dumping*.

The following example shows how to dump all the tables of a database, `shentest`, its structure and content, into `shentestdump.sql`, an sql script, under `c:/temp`, which can be later executed in turing, for example, to restore the whole thing.

```
C:\Program Files\MySQL\MySQL Server 4.1\bin>
mysqldump -u root -p shentest > c:/temp/shentestdump.sql
Enter password: *****
```

Note: *This has to run at the system prompt, such as DOS or turing.*

1.1 a GUI interface

It is far easier to use a GUI interface to complete some of the database operations. One of the better and more popular GUI interface for the *MySQL/PHP* combo is *PhPMySQLAdmin*, which is available on turing.

To activate this interface in turing, just enter `http://turing.plymouth.edu/mysql/` in the browser.

If stored locally, it can be launched by opening the following page, if all the files related to the *PhPMySQLAdmin* software is collected in a folder `PhPMySQLAdmin` and placed under `Apache:Apach2:htdocs`:

`http://localhost/phpMyAdmin/`

2 Table definition and population

2.1 The Student table

1. *Structure*: Figure 3.6,[1, pp. 43]

```
Student (Id: INT, Name: STRING, Address: STRING, Status: STRING)
      Key: {Id}
```

2. *SQL code*: pp. 49

```
CREATE TABLE Student (
  Id      Integer,
  Name    Char(20) Not Null,
  Address Char(50),
  Status  Char(10) Default 'freshman'

  PRIMARY KEY (Id));
```

3. *MySQL code*:

```
create table Student (
  Id INT Not Null Primary key,
  Name Char(20) Not Null,
  Address Char(50),
  Status Char(20) default 'freshman');
```

```
mysql> desc student;
+-----+-----+-----+-----+-----+
| Field | Type   | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
| Id     | int(11)|      | PRI | 0       |       |
| Name   | char(20)|      |     |         |       |
| Address| char(50)| YES  |     | NULL    |       |
| Status | char(20)| YES  |     | freshman|       |
+-----+-----+-----+-----+-----+
4 rows in set (0.01 sec)
```

4. *Data*: Figure 2.1. [1, pp. 14]

```
Insert into Student (Id, Name, Address, Status)
  Values (111111111, 'Jane Doe', '123 Main St.', 'freshman');

Insert into Student (Id, Name, Address, Status)
  Values (666666666, 'Jesoph Public', '666 Hollow Rd.', 'sophomore');

Insert into Student (Id, Name, Address, Status)
  Values (111223344, 'Mary Smith', '1 Lake St.', 'freshman');

Insert into Student (Id, Name, Address, Status)
  Values (987654321, 'Bart Simpson', 'Fox 5 Tv', 'senior');

Insert into Student (Id, Name, Address, Status)
  Values (023456789, 'Homer Simpson', 'Fox 5 Tv', 'senior');

Insert into Student (Id, Name, Address, Status)
  Values (123454321, 'Joe Blow', '6 Yard Ct.', 'junior');
```

Notice that the first 0 does not show;

```
mysql> select * from student;
+-----+-----+-----+-----+
| Id     | Name           | Address           | Status   |
+-----+-----+-----+-----+
| 23456789 | Homer Simpson | Fox 5 Tv         | senior   |
| 111111111 | Jane Doe      | 123 Main St.    | freshman |
| 111223344 | Mary Smith    | 1 Lake St.      | freshman |
| 123454321 | Joe Blow      | 6 Yard Ct.      | junior   |
| 666666666 | Jesoph Public | 666 Hollow Rd.  | sophomore|
```

```
| 987654321 | Bart Simpson | Fox 5 Tv | senior |
+-----+-----+-----+-----+
6 rows in set (0.00 sec)
```

2.2 The Professor table

1. *Structure:* Figure 3.6, [1, pp. 43]

```
Professor (Id: INT, Name: STRING, DeptId: DEPTS)
Key: {Id}
```

2. *SQL code:* It is not given in the book, but can be found, e.g., in the example given in pp. 39.

```
CREATE TABLE Professor (
  Id      Integer,
  Name    Char(20) Not Null,
  DeptId  Char(2) Not Null,

  PRIMARY KEY (Id));
```

3. *MySQL code:*

```
create table Professor (
  Id INT Not Null Primary key,
  Name Char(20) Not Null,
  DeptId Char(2) Not Null);
```

Note: The following changes the type of DeptId.

```
alter table professor change DeptId DeptId Char(4);
```

```
mysql> desc professor;
+-----+-----+-----+-----+-----+-----+
| Field | Type      | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| Id    | int(11)  |      | PRI | 0        |       |
| Name  | char(20) |      |     |          |       |
| DeptId | char(4)  | YES  |     | NULL     |       |
+-----+-----+-----+-----+-----+-----+
3 rows in set (0.00 sec)
```

4. *Data*: Figure 3.5, [1, pp. 39]

```
Insert into Professor (Id, Name, DeptId)
  Values (101202303, 'John Smyth', 'CS');

Insert into Professor (Id, Name, DeptId)
  Values (783432188, 'Adrian Jones', 'MGT');

Insert into Professor (Id, Name, DeptId)
  Values (121232343, 'David Jones', 'EE');

Insert into Professor (Id, Name, DeptId)
  Values (864297351, 'Qi Chen', 'MAt');

Insert into Professor (Id, Name, DeptId)
  Values (555666777, 'Mary Doe', 'CS');

Insert into Professor (Id, Name, DeptId)
  Values (009406321, 'Jacob Taylor', 'MGT');

Insert into Professor (Id, Name, DeptId)
  Values (900120450, 'Ann White', 'MAT');
```

```
mysql> select * from professor;
+-----+-----+-----+
| Id      | Name      | DeptId |
+-----+-----+-----+
| 9406321 | Jacob Taylor | MGT    |
| 101202303 | John Smyth  | CS     |
| 121232343 | David Jones | EE     |
| 555666777 | Mary Doe   | CS     |
| 783432188 | Adrian Jones | MGT    |
| 864297351 | Qi Chen    | MAt    |
| 900120450 | Ann White  | MAT    |
+-----+-----+-----+
```

2.3 The Course table

1. *Structure*: Figure 3.6,[1, pp. 43]

```
Course (DeptId: DEPTS, CrsName: STRING, CrsCode: COURSES)
  Key: {CrsCode}, {DeptId,CrsName}
```

Notice this table comes with two key constraints.

2. *SQL code*: It is not given in the book, but is suggested with the attached data.

```
Create table Course (  
  CrsCode  Char(6),  
  DeptId   Char(4)  
  CrsName  Char(20),  
  Descr    Char(100),  
  Primary key (CrsCode),  
  Unique   (DeptId,CrsName))
```

3. *MySQL code*:

```
Create table Course (  
  CosCode Char(6) Not Null Primary key,  
  DeptId  Char(4) Not Null,  
  CrsName Char(20) Not Null,  
  Descr   Char(100),  
  CONSTRAINT course_index UNIQUE (DeptId, CrsName));
```

```
mysql> desc course;  
+-----+-----+-----+-----+-----+-----+  
| Field | Type      | Null | Key | Default | Extra |  
+-----+-----+-----+-----+-----+-----+  
| CosCode | char(6) | | PRI | | |  
| DeptId  | char(4) | | MUL | | |  
| CrsName | char(20) | | | | |  
| Descr   | char(100) | YES | | NULL | |  
+-----+-----+-----+-----+-----+-----+  
4 rows in set (0.00 sec)
```

Note: Below shows how to change the name of a column. The essence is that the primary key part should not be used, as pointed out by one MySQL user.

```
mysql> alter table course change CosCode CrsCode Char(6) Not Null Primary key;  
ERROR 1068 (42000): Multiple primary key defined  
mysql> alter table course change CosCode CrsCode Char(6) Not Null;  
Query OK, 0 rows affected (0.16 sec)  
Records: 0 Duplicates: 0 Warnings: 0
```

```
mysql> desc course;
+-----+-----+-----+-----+-----+-----+
| Field | Type   | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| CrsCode | char(6) |      | PRI |          |       |
| DeptId  | char(4) |      | MUL |          |       |
| CrsName | char(20)|      |     |          |       |
| Descr   | char(100)| YES  |     | NULL     |       |
+-----+-----+-----+-----+-----+-----+
4 rows in set (0.00 sec)
```

4. *Data*: Figure 3.5, [1, pp. 39]

```
Insert into Course (CrsCode, DeptId, CrsName, Descr)
  Values ('CS305', 'CS', 'Database Systems.',
          'On the road to high-paying job');

Insert into Course (CrsCode, DeptId, CrsName, Descr)
  Values ('CS315', 'CS', 'Transaction Processing',
          'Recover from your worst crashes');

Insert into Course (CrsCode, DeptId, CrsName, Descr)
  Values ('MGT123', 'MGT', 'Market Analysis', 'Get rich quick');

Insert into Course (CrsCode, DeptId, CrsName, Descr)
  Values ('EE101', 'EE', 'Electronic Circuits',
          'Build your own computer');

Insert into Course (CrsCode, DeptId, CrsName, Descr)
  Values ('MAT123', 'MAT', 'Algebra',
          'The world where 2+2=5');
```

```
mysql> select * from course;
+-----+-----+-----+-----+-----+-----+
| CrsCode | DeptId | CrsName                | Descr                |
+-----+-----+-----+-----+-----+-----+
| CS305   | CS     | Database System        | On the road to high-paying job |
| CS315   | CS     | Tranaction Processin  | Recover from your worst crashes |
| EE101   | EE     | Electronic Circuits    | Build your own computer |
| MAT123  | MAT    | Algebra                | There world where 2+2=5 |
| MGT123  | MGT    | Market Analysis       | Get rich quick |
+-----+-----+-----+-----+-----+-----+
5 rows in set (0.05 sec)
```

2.4 The Transcript table

1. *Structure*: Figure 3.6, [1, pp. 43]

```
Transcript (CrsCode: COURSES, StudId: INT, Grade: GRADES, Semester: SEMESTERS)
  Key: {StudId,CrsCode,Semester}
```

2. *SQL code*: Query 3.2 [1, pp. 52]

```
Create table Transcript (
  StudId Integer,
  CrsCode Char(6),
  Semester Char(6),
  Grade Char(1),
  Check (Grade in ('A','B','C','D','F')),
  Check (StudId>0 AND StudId<100000))
```

3. *SQL code*: with foreign keys, in pp. 94. Notice that it also contains a foreign key on semester table, but that table does not exist, thus deleted in the mysql definition.

```
Create table Transcript (
  StudId Integer,
  CrsCode Char(6),
  Semester Char(6),
  Grade Char(1),
  Primary key (StudId, CrsCode, Semester),
  Foreign key (StudId) references Student(Id),
  Foreign key (CrsCode) references Course(CrsCode))
  Foreign key (Semester) references Semester (SemCode)
```

4. *MySQL code*:

```
Create table Transcript (
  StudId INT Not Null,
  CrsCode Char(6) Not Null,
  Semester Char(6) Not Null,
  Grade Char(1),
  Primary key (StudId, CrsCode, Semester),
  Foreign key (StudId) references Student(Id),
  Foreign key (CrsCode) references Course(CrsCode),
  Constraint grade_condition Check (Grade in ('A','B','C','D','F')),
  Constraint id_range Check (StudId>0 AND StudId<100000));
```

```
mysql> desc transcript;
+-----+-----+-----+-----+-----+-----+
| Field      | Type      | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| StudId     | int(11)   |      | PRI | 0        |       |
| CrsCode    | char(6)   |      | PRI |          |       |
| Semester   | char(6)   |      | PRI |          |       |
| Grade      | char(1)   | YES  |     | NULL     |       |
+-----+-----+-----+-----+-----+-----+
4 rows in set (0.00 sec)
```

5. *Data*: Figure 3.5, [1, pp. 39]

```
Insert into Transcript (StudId, CrsCode, Semester, Grade)
  Values (666666666, 'MGT123', 'F1994', 'A');
```

```
Insert into Transcript (StudId, CrsCode, Semester, Grade)
  Values (666666666, 'EE101', 'S1991', 'B');
```

```
Insert into Transcript (StudId, CrsCode, Semester, Grade)
  Values (666666666, 'MAT123', 'F1997', 'B');
```

```
Insert into Transcript (StudId, CrsCode, Semester, Grade)
  Values (987654321, 'CS305', 'F1995', 'C');
```

```
Insert into Transcript (StudId, CrsCode, Semester, Grade)
  Values (987654321, 'MGT123', 'F1994', 'B');
```

```
Insert into Transcript (StudId, CrsCode, Semester, Grade)
  Values (123454321, 'CS315', 'F1997', 'A');
```

```
Insert into Transcript (StudId, CrsCode, Semester, Grade)
  Values (123454321, 'CS305', 'S1996', 'A');
```

```
Insert into Transcript (StudId, CrsCode, Semester, Grade)
  Values (123454321, 'MAT123', 'S1996', 'C');
```

```
Insert into Transcript (StudId, CrsCode, Semester, Grade)
  Values (023456789, 'EE101', 'F1995', 'B');
```

```
Insert into Transcript (StudId, CrsCode, Semester, Grade)
```

```

Values (023456789, 'CS305', 'S1996', 'A');

Insert into Transcript (StudId, CrsCode, Semester, Grade)
  Values (111111111, 'EE101', 'F1997', 'A');

Insert into Transcript (StudId, CrsCode, Semester, Grade)
  Values (111111111, 'MAt123', 'F1997', 'B');

Insert into Transcript (StudId, CrsCode, Semester, Grade)
  Values (111111111, 'MGT123', 'F1997', 'B');

```

```

mysql> select * from transcript;
+-----+-----+-----+-----+
| StudId   | CrsCode | Semester | Grade |
+-----+-----+-----+-----+
| 23456789 | CS305   | S1996   | A     |
| 23456789 | EE101   | F1995   | B     |
| 111111111 | EE101   | F1997   | A     |
| 111111111 | MAt123  | F1997   | B     |
| 111111111 | MGT123  | F1997   | B     |
| 123454321 | CS305   | S1996   | A     |
| 123454321 | CS315   | F1997   | A     |
| 123454321 | MAT123  | S1996   | C     |
| 666666666 | EE101   | S1991   | B     |
| 666666666 | MAT123  | F1997   | B     |
| 666666666 | MGT123  | F1994   | A     |
| 987654321 | CS305   | F1995   | C     |
| 987654321 | MGT123  | F1994   | B     |
+-----+-----+-----+-----+
13 rows in set (0.00 sec)

```

Notice that the following won't work, as it violates a foreign key constraint:

```

Insert into Transcript (StudId, CrsCode, Semester, Grade)
  Values (111111111, 'MGT456', 'F1997', 'B');

mysql> Insert into Transcript (StudId, CrsCode, Semester, Grade)
->      Values (111111111, 'MGT456', 'F1997', 'B');
ERROR 1216 (23000): Cannot add or update a child row:
a foreign key constraint fails

```

2.5 The Teaching table

1. *Structure*: Figure 3.6,[1, pp. 43]

```
Teaching (ProfId:Integer, CrsCode:String, Semester:String)
  Key: {CrsCode,Semester}
```

2. *SQL code*: [1, pp. 54]

```
Create table Teaching (
  ProfId      Integer,
  CrsCode Char(6),
  Semester    Char(6),
  Primary key (CrsCode,Semester),
  Foreign key (CrsCode) references Course,
  Foreign key (ProfId) references (Professor(Id)))
```

3. *SQL code with triggers*, [1, pp. 57]. Notice that a single course can be taught by multiple professors in different semester. The primary constraint seems to enforce that only one section will be offered in one semester, thus

- both `CrsCode` and `Semester` attributes must be declared `Not Null`; and
- you can't enter two rows with the same values for those two attributes. For example, the first and the fourth rows in pp. 40 can't be entered. When you tried, the following message is returned.

```
mysql> Insert into teaching (ProfId, CrsCode, Semester)
->      Values (864297531, 'MGT123', 'F1994');
ERROR 1062 (23000): Duplicate entry 'MGT123-F1994' for key 1
```

This is an issue, since it is not consistent with the data to be entered in pp. 40, although it does not seemingly violate a user constraint.

- Another point is that the no action option for delete, which is called `restrict` in the MySQL code since it does not allow a row to be deleted if it matches a condition.

```
Create table Teaching (
  ProfId      Integer,
  CrsCode    Char(6),
  Semester    Char(6),
  Primary Key (CrsCode,Semester),
  Foreign key (ProfId) reference Professor (Id)
  On delete No action
  On Update Cascade,
```

```

Foreign key (CrsCode) references Course(CrsCode)
  On Delete Set null
  On Update Cascade)

```

4. *MySQL code:*

```

Create table Teaching (
  ProfId    Integer,
  CrsCode   Char(6) Not Null,
  Semester  Char(6) Not Null,
  Primary Key (CrsCode,Semester),
  Foreign key (ProfId) references Professor (Id)
    On Delete restrict
    On Update Cascade,
  Foreign key (CrsCode) references Course(CrsCode)
    On Delete Set null
    On Update Cascade)

```

```
mysql> desc Teaching;
```

Field	Type	Null	Key	Default	Extra
ProfId	int(11)	YES	MUL	NULL	
CrsCode	char(6)	NO	PRI	NULL	
Semester	char(6)	NO	PRI	NULL	

```
3 rows in set (0.01 sec)
```

5. *Data:* Figure 3.5, [1, pp. 39]

```

Insert into teaching (ProfId, CrsCode, Semester)
  Values (009406321, 'MGT123', 'F1994');

```

```

Insert into teaching (ProfId, CrsCode, Semester)
  Values (121232343, 'EE101', 'S1991');

```

```

Insert into teaching (ProfId, CrsCode, Semester)
  Values (555666777, 'Cs305', 'F1995');

```

```

Insert into teaching (ProfId, CrsCode, Semester)

```

```

    Values (101202303, 'CS315', 'S1997');

Insert into teaching (ProfId, CrsCode, Semester)
    Values (900120450, 'MAT123', 'S1996');

Insert into teaching (ProfId, CrsCode, Semester)
    Values (121232343, 'EE101', 'F1995');

Insert into teaching (ProfId, CrsCode, Semester)
    Values (101202303, 'CS305', 'S1996');

Insert into teaching (ProfId, CrsCode, Semester)
    Values (900120450, 'MAT123', 'F1997');

Insert into teaching (ProfId, CrsCode, Semester)
    Values (783432188, 'MGT123', 'F1997');

```

```

mysql> select * from teaching;
+-----+-----+-----+
| ProfId    | CrsCode | Semester |
+-----+-----+-----+
| 9406321   | MGT123  | F1994    |
| 101202303 | CS305   | S1996    |
| 101202303 | CS315   | S1997    |
| 121232343 | EE101   | F1995    |
| 121232343 | EE101   | S1991    |
| 555666777 | Cs305   | F1995    |
| 783432188 | MGT123  | F1997    |
| 900120450 | MAT123  | F1997    |
| 900120450 | MAT123  | S1996    |
+-----+-----+-----+
9 rows in set (0.00 sec)

```

Labwork 2:

1. Create and populate all the aforementioned tables in your MySQL account.
2. Create and populate the following tables:

- Supplier

```

Supplliers(SupplierId:String, SName:String, Status:Integer, City:String)
    Key: {SupplierId}

```

SupplierID	Name	Status	City
S1	Smith	20	London
S2	Jones	10	Paris
S3	Blake	30	Paris
S4	Clark	20	London
S5	Adams	30	Athens

- Part

Part(PartID:String, Name:String, Color:Integer, Weight:Float, City:String)
 Key: {PartID}

PartID	PNAME	COLOR	WEIGHT	CITY
P1	Nut	Red	12.0	London
P2	Bolt	Green	17.0	Paris
P3	Screw	Blue	17.0	Rome
P4	Screw	Red	14.0	London
P5	Cam	Blue	12.0	Paris
P6	Cog	Red	19.0	London

- SupplyPart

SupplyPart(SupplierId:String, PartId:String, Quantity:Integer)
 Key: {SupplierId, PartId}

SupplierID	PartID	Quantity
S1	P1	300
S1	P2	200
S1	P3	400
S1	P4	200
S1	P5	100
S1	P6	100
S2	P1	300
S2	P2	400
S3	P2	200
S4	P2	200
S4	P4	300
S4	P5	400

3 SQL Queries

The following examples are taken from [1, §5.2].

3.1 Simple queries

5.5. Get the names of all the professors in the EE department.

(a) Relational algebraic expression.

$$\pi_{Name}(\sigma_{DeptId='EE'}(Professor))$$

(b) MySQL code;

```
Select P.Name From Professor P Where DeptID='EE'
```

```
mysql> Select P.Name From Professor P Where DeptID='EE';
```

```
+-----+
| Name      |
+-----+
| David Jones |
+-----+
```

```
1 row in set (0.00 sec)
```

5.6. Get the names of all the professors who taught in Fall 1994.

(a) Relational algebraic expression.

$$\pi_{Name}(Professor \bowtie_{Id=ProfId} (\sigma_{Semester='F1994'}(Teaching))) \quad (5.7)$$

(b) MySQL code;

```
Select P.Name From Professor P, Teaching T Where P.Id=T.ProfId
And T.Semester='F1994'
```

```
mysql> Select P.Name From Professor P, Teaching T Where P.Id=T.ProfId And
T.Semester='F1994';
```

```
+-----+
| Name      |
+-----+
| Jacob Taylor |
+-----+
```

```
1 row in set (0.08 sec)
```

We can verify the above result as follows:

```
mysql> select * from Professor;
```

```
+-----+-----+-----+
| Id      | Name      | DeptId |
+-----+-----+-----+
```

```

| 9406321 | Jacob Taylor | MGT |
| 101202303 | John Smyth | CS |
| 121232343 | David Jones | EE |
| 555666777 | Mary Doe | CS |
| 783432188 | Adrian Jones | MGT |
| 864297351 | Qi Chen | MAt |
| 900120450 | Ann Whit | MAT |
+-----+-----+-----+
7 rows in set (0.00 sec)

```

```

mysql> select * from Teaching;
+-----+-----+-----+
| ProfId | CrsCode | Semester |
+-----+-----+-----+
| 9406321 | MGT123 | F1994 |
| 101202303 | CS305 | S1996 |
| 101202303 | CS315 | S1997 |
| 121232343 | EE101 | F1995 |
| 121232343 | EE101 | S1991 |
| 555666777 | Cs305 | F1995 |
| 783432188 | MGT123 | F1997 |
| 900120450 | MAT123 | F1997 |
| 900120450 | MAT123 | S1996 |
+-----+-----+-----+
9 rows in set (0.00 sec)

```

5.10. Get the names of all the course taught in fall 1995 together with the names of these professors who taught them.

(a) Relational algebraic expression.

$$\pi_{CrsName, Name}(\sigma_{Id=ProfId \text{ And } Teaching.CrsCode=Course.CrsCode \text{ And } Semester='F1995'}(Professor \times Teaching \times Course))$$

(b) MySQL code;

```

Select c.CrsName, P.Name From Professor P, Teaching T, Course C
Where T.Semester='F1995' And P.Id=T.ProfId And T.CrsCode=C.CrsCode

mysql> Select c.CrsName, P.Name From Professor P, Teaching T, Course C
-> Where T.Semester='F1995' And P.Id=T.ProfId And T.CrsCode=C.CrsCode;
+-----+-----+
| CrsName | Name |

```

```

+-----+-----+
| Database System      | Mary Doe      |
| Electronic Circuits | David Jones   |
+-----+-----+
2 rows in set (0.03 sec)

```

This result can be checked out with the following data.

```

mysql> select * from Teaching;
+-----+-----+-----+
| ProfId   | CrsCode | Semester |
+-----+-----+-----+
| 9406321  | MGT123  | F1994    |
| 101202303 | CS305   | S1996    |
| 101202303 | CS315   | S1997    |
| 121232343 | EE101   | F1995    |
| 121232343 | EE101   | S1991    |
| 555666777 | Cs305   | F1995    |
| 783432188 | MGT123  | F1997    |
| 900120450 | MAT123  | F1997    |
| 900120450 | MAT123  | S1996    |
+-----+-----+-----+
9 rows in set (0.01 sec)

```

5.11. *Get the ids of all the students who took at least two courses.*

- (a) Relational algebraic expression. As we stated in the lecture, we rename all the attributes, except the `StudId`, of the `Transcript` table then join it with the original to agree on the `StudId` item, thus get all the transcript records of all the students. Notice with the condition, all the students who took only one will be left out.

$$\pi_{\text{StudId}}(\sigma_{\text{CrsCode} \neq \text{CrsCode2}}(\text{Transcript} \bowtie \text{Transcript}[\text{StudID}, \text{CrscCode2}, \text{Semester2}, \text{Grade2}]))$$

- (b) MySQL code;

```

Select distinct T1.StudId
From Transcript T1, Transcript T2
Where T1.CrsCode<>T2.CrsCode
      And T1.StudId=T2.StudId

mysql> Select T1.StudId

```

```

-> From Transcript T1, Transcript T2
-> Where T1.CrsCode<>T2.CrsCode
->      And T1.StudId=T2.StudId;

```

```

+-----+
| StudID |
+-----+
| 23456789 |
| 123454321 |
| 123454321 |
| 987654321 |
| 123454321 |
| 123454321 |
| 23456789 |
| 111111111 |
| 111111111 |
| 666666666 |
| 666666666 |
| 111111111 |
| 111111111 |
| 123454321 |
| 123454321 |
| 666666666 |
| 666666666 |
| 111111111 |
| 111111111 |
| 666666666 |
| 666666666 |
| 987654321 |

```

```

+-----+
22 rows in set (0.05 sec)

```

This just gets us too much duplicated information. Since we only want to know the id once per student, we add in the `distinct` word.

```

mysql> Select distinct T1.StudID
-> From Transcript T1, Transcript T2
-> Where T1.CrsCode<>T2.CrsCode
->      And T1.StudId=T2.StudId;

```

```

+-----+
| StudID |
+-----+
| 23456789 |

```

```

| 123454321 |
| 987654321 |
| 111111111 |
| 666666666 |
+-----+
5 rows in set (0.03 sec)

```

Indeed, except the one with 111223344, every student took at least two courses.

The restrictive `distinct` is indeed very useful.

Distinction: *Get the ids of professors who has taught together with the courses they taught.*

```
Select T.ProfId T.CrsCode From Teaching T;
```

```
mysql> Select T.ProfId, T.CrsCode From Teaching T;
+-----+-----+
| ProfId   | CrsCode |
+-----+-----+
|  9406321 | MGT123  |
| 101202303 | CS305   |
| 101202303 | CS315   |
| 121232343 | EE101   |
| 121232343 | EE101   |
| 555666777 | Cs305   |
| 783432188 | MGT123  |
| 900120450 | MAT123  |
| 900120450 | MAT123  |
+-----+-----+
9 rows in set (0.00 sec)
```

We certainly only need the following:

```
mysql> Select distinct T.ProfId, T.CrsCode From Teaching T;
+-----+-----+
| ProfId   | CrsCode |
+-----+-----+
|  9406321 | MGT123  |
| 101202303 | CS305   |
| 101202303 | CS315   |
| 121232343 | EE101   |
| 555666777 | Cs305   |
| 783432188 | MGT123  |

```

```

| 900120450 | MAT123 |
+-----+-----+
7 rows in set (0.00 sec)

```

We can give more descriptive names for the resulting attributes.

Renaming: *Get the names of all the professors in the EE department.*

```
Select P.Name As Professor From Professor P Where DeptID='EE'
```

```

mysql> Select P.Name As Professor From Professor P Where DeptID='EE';
+-----+
| Professor |
+-----+
| David Jones |
+-----+
1 row in set (0.03 sec)

```

Negation: *Get the names of all the professors who don't work in the EE department.*

```
Select P.Name As Professor From Professor P Where Not (DeptID ='EE')
```

```

mysql> Select P.Name As Professor From Professor P Where Not (DeptID ='EE')
-> ;
+-----+
| Professor |
+-----+
| Jacob Taylor |
| John Smyth |
| Mary Doe |
| Adrian Jones |
| Qi Chen |
| Ann White |
+-----+
6 rows in set (0.00 sec)

```

Labwork 3.1: The following queries are based on the *Supplier* database that you have created in Labwork 2:

1. Get all the parts stored in Rome.
2. Get all the suppliers who are based in London.

3. Get the color and city values of those parts that are not stored in Paris and with a weight of at least 10 tons (Oops, grams).
4. Get supplier names for suppliers who supply part P2.
5. Get supplier names for suppliers who supply at least one red part.

For each of them, go through the following steps, using the table instances as shown in Labwork 2:

1. Use the English query to get an output.
2. Write an RA expression, and apply it to the table instances to get an output.
3. Write an *MySQL* query and run it to get an output.
4. If the above three outputs are not the same, check and correct your work.

You have to send in, for each query, the query itself, its RA expression, and the *MySQL* query, together with the results.

3.2 Set operations

It seems that, at least in *MySQL* version 4.1.14, only the `Union` operation is supported, neither `Intersect` nor `Except` is.

5.15. *Get the names of all the professors in the CS department or in the EE department.*

(a) Relational algebraic expression.

$$\pi_{Name}(\sigma_{DeptID='CS'}(Professor)) \cup \pi_{Name}(\sigma_{DeptID='EE'}(Professor))$$

(b) *MySQL* code;

```
(Select P.Name From Professor P Where DeptID='CS')
Union (Select P.Name From Professor P Where DeptID='EE')
```

```
mysql> (Select P.Name From Professor P Where P.DeptID='CS') Union (Select P.Name
-> From Professor P Where P.DeptID='EE');
```

```
+-----+
| Name      |
+-----+
| John Smyth |
| Mary Doe  |
| David Jones |
+-----+
3 rows in set (0.00 sec)
```

(c) Another form in MySQL (5.16).

```
Select P.Name From Professor P Where P.DeptID='CS' or P.DeptId='EE';

mysql> Select P.Name From Professor P Where P.DeptID='CS' or P.DeptId='EE';
+-----+
| Name      |
+-----+
| John Smyth |
| David Jones |
| Mary Doe   |
+-----+
3 rows in set (0.00 sec)
```

The word `like` can be used to match patterns.

5.17: *Get those who taught a CS course.*

```
mysql> Select Distinct P.Name From Professor P, Teaching T Where (P.Id=T.ProfId
And T.CrsCode Like 'CS%') OR (P.DeptId='CS');
```

```
+-----+
| Name      |
+-----+
| John Smyth |
| Mary Doe   |
+-----+
2 rows in set (0.00 sec)
```

5.20: *Get those took both 'CS305' and 'CS315'*

(a) SQL code.

```
Select distinct S.Name From Student S,Transcript T1,Transcript T2
Where S.Id=T1.StudId And T1.CrsCode='CS305'
      And S.Id=T2.StudId And T2.CrsCode='CS315';
```

(b) MySQL code and the result:

```
mysql> Select distinct S.Name From Student S,Transcript T1,Transcript T2
-> Where S.Id=T1.StudId And T1.CrsCode='CS305'
->      And S.Id=T2.StudId And T2.CrsCode='CS315';
+-----+
| Name      |
+-----+
```

```
| Joe Blow |
+-----+
1 row in set (0.00 sec)
```

Indeed, the following result shows that Joe Blow with the id being 123454321 is the only one who has taken both courses.

```
mysql> select * from transcript;
+-----+-----+-----+-----+
| StudId   | CrsCode | Semester | Grade |
+-----+-----+-----+-----+
| 23456789 | CS305   | S1996    | A     |
| 23456789 | EE101   | F1995    | B     |
| 111111111 | EE101   | F1997    | A     |
| 111111111 | MAT123  | F1997    | B     |
| 111111111 | MGT123  | F1997    | B     |
| 123454321 | CS305   | S1996    | A     |
| 123454321 | CS315   | F1997    | A     |
| 123454321 | MAT123  | S1996    | C     |
| 666666666 | EE101   | S1991    | B     |
| 666666666 | MAT123  | F1997    | B     |
| 666666666 | MGT123  | F1994    | A     |
| 987654321 | CS305   | F1995    | C     |
| 987654321 | MGT123  | F1994    | B     |
+-----+-----+-----+-----+
13 rows in set (0.00 sec)
```

5.22: *Get those professors who work either in the Computer Science department or in the Electrical Engineering department.*

(a) Relational algebraic expression.

$$\pi_{Name}(\sigma_{DeptId \in \{ 'CS', 'EE' \}}(Professor))$$

(b) SQL code.

```
Select P.Name From Professor P
Where P.DeptId In ('CS', 'EE');
```

(c) MySQL code and the result:

```
mysql> SELECT P.Name From professor P
-> where P.DeptId in ('CS', 'EE');
+-----+
| Name          |
```

```

+-----+
| John Smyth |
| David Jones |
| Mary Doe   |
+-----+
3 rows in set (0.02 sec)

```

Notice that MySQL v.4.1.14 does not support other set operators such as **Intersect** and **Except**. For example, neither of the following runs.

```

(Select StudId from Transcript where CrsCode='CS305')
Intersect
(Select StudId from Transcript where CrsCode='CS315');

(Select StudId from Transcript where CrsCode='CS305')
Except
(Select StudId from Transcript where CrsCode='CS315');

```

Labwork 3.2:

1. Check out the *MySQL* site to see if the current version supports all the three operations. If not, which are supported, and the correct syntax, with a simple example.
2. The following queries are based on the *Supplier* database that you have created in Labwork 2:
 - (a) Get parts that are either red or green.
 - (b) Get supplier names for those who are located in either Rome or London.
 - (c) Get supplier names for suppliers who supply both nuts and bolts.
 - (d) Get supplier names for those who are located either Rome or London and sell at least two kinds of parts.
 - (e) (**Bonus:**) Get supplier names for suppliers who do not supply red parts.

For each of them, go through the following steps, using the table instances as shown in Labwork 2:

- (a) Use the English query to get an output.
- (b) Write an RA expression, and apply it to the table instances to get an output.
- (c) Write an *MySQL* query and run it to get an output.
- (d) If the above three outputs are not the same, check and correct your work.

To reiterate, you have to send in, for each query, the query itself, its RA expression, and the *MySQL* query, together with the results.

3.3 Nested queries

Now, we look at the more complicated situation, the nested queries. SQL Code is indeed achievable, since it is computationally complete. On the other hand, its implementation, such as MySQL v. 4.1.14, does not support all the structures, e.g., it does not support the quantifiers.

To kick off: *Get all professor who taught in F1994.*

(a) SQL code:

```
Select P.Name From Professor P
Where P.Id in
--A nested subquery
      (Select T.ProfId From Teaching T
       Where T.Semester='F1994')
```

(b) MySQL code and its result

```
mysql> Select P.Name From Professor P
-> Where P.Id in
-> --A nested subquery
->      (Select T.ProfId From Teaching T
->      Where T.Semester='F1994');
```

Name
Jacob Taylor

1 row in set (0.01 sec)

(c) This is confirmed by doing the following:

```
mysql> select * from Teaching;
```

ProfId	CrsCode	Semester
9406321	MGT123	F1994
101202303	CS305	S1996
101202303	CS315	S1997
121232343	EE101	F1995
121232343	EE101	S1991
555666777	Cs305	F1995
783432188	MGT123	F1997
900120450	MAT123	F1997
900120450	MAT123	S1996

```

+-----+-----+-----+
9 rows in set (0.00 sec)

```

5.23: *Get all students who did not take any course.*

(a) SQL code:

```

Select S.Name From Student S
Where S.Id Not in
--A nested subquery
      (Select T.StudId From Transcript T);

```

(b) MySQL code and its result

```

mysql> Select S.Name From Student S
-> Where S.Id Not in
-> --A nested subquery
->      (Select T.StudId From Transcript T);

```

```

+-----+
| Name      |
+-----+
| Mary Smith |
+-----+

```

1 row in set (0.03 sec)

Indeed, Mary Smith, with her code being 111223344, is the only one who did not take any course.

5.25: *Get all students and the courses that they took with a professor in F1994.*

(a) SQL code:

```

Select distinct R.StudId,P.Id,R.CrsCode From Transcript R,Professor P
Where R.CrsCode in
--courses taught by P.Id in F2004
      (Select T1.CrsCode From Teaching T1
      Where T1.ProfId=P.Id And T1.Semester='F1994');

```

(b) MySQL code and its result

```

mysql> Select distinct R.StudId,P.Id,R.CrsCode From Transcript R,Professor P
-> Where R.CrsCode in
-> --courses taught by P.Id in F2004
->      (Select T1.CrsCode From Teaching T1
->      Where T1.ProfId=P.Id And T1.Semester='F1994');

```

```

+-----+-----+-----+
| StudId  | Id      | CrsCode |

```

```

+-----+-----+-----+
| 111111111 | 9406321 | MGT123 |
| 666666666 | 9406321 | MGT123 |
| 987654321 | 9406321 | MGT123 |
+-----+-----+-----+
3 rows in set (0.02 sec)

```

We sometimes want to use the `Exists` quantifier.

5.26: *Get all students who never took a computer science course.*

(a) SQL code:

```

Select S.Id From Student S
Where Not Exists
--there exists no CS courses that S.Id has taken
      (Select T.CrsCode From Transcript T
        Where T.StudId=S.Id And T.CrsCode Like 'CS%');

```

(b) MySQL code and its result

```

mysql> Select S.Id From Student S
-> Where Not Exists
-> --there exists no CS courses that S.Id has taken
->      (Select T.CrsCode From Transcript T
->        Where T.StudId=S.Id And T.CrsCode Like 'CS%');
+-----+
| Id      |
+-----+
| 111111111 |
| 111223344 |
| 666666666 |
+-----+
3 rows in set (0.00 sec)

```

Wrap up: *Get all students who were not taught by at least one computer science professor.*

This is a fairly complicated example, and it will be as far as we will go.

(a) SQL code:

```

Select S.Id
From Student S,
      (Select P.Id --All CS professors
       From Professor P
       Where P.DeptId='CS') As C
Where C.Id Not In

```

```

(Select T.ProfId --All S.Id's professors
 From Teaching T, Transcript R
 Where T.CrsCode=R.CrsCode
       And T.Semester=R.Semester
       And S.Id=R.StudId)

```

(b) MySQL code and its result

```

mysql> Select distinct S.Id
-> From Student S,
->      (Select P.Id
->      From Professor P
->      Where P.DeptId='CS') As C
-> Where C.Id Not In
->      (Select T.ProfId
->      From Teaching T, Transcript R
->      Where T.CrsCode=R.CrsCode
->            And T.Semester=R.Semester
->            And S.Id=R.StudId);

```

```

+-----+
| Id      |
+-----+
| 11111111 |
| 111223344 |
| 666666666 |
| 987654321 |
| 23456789 |
| 123454321 |
+-----+

```

6 rows in set (0.00 sec)

Notice that the current version of MySQL does not support the `All` quantifier. For example, the following does not run.

```

Select S.Id From Student S
Where
  For All (Select C.CrsCode From Course C
           Where C.CrsCode Like 'CS%')
         (CrsCode in
           (Select R.CrsCode From Transcript R
            Where R.StudId=S.Id));

```

On the other hand, both `All` and `Exist` are supported with the current version MySQL v. 5.0, but there is no example for `All exists` in the MySQL site.

Labwork 3.3: The following queries are based on the *Supplier* database that you have created in Labwork 2:

- Get all the details of those parts supplied by someone located in London.
- Get supplier names for suppliers who supply at least one red part.
- Get supplier names for those who supply nuts.

For each of them, go through the following steps, using the table instances as shown in Labwork 2:

1. Use the English query to get an output.
2. Write an *MySQL* query and run it to get an output.
3. If the above two outputs are not the same, check and correct your work.

You have to send in, for each query, the query itself, and the *MySQL* query, together with the results.

3.4 Aggregation

We now turn to the aggregation stuff with a few simple examples.

Kick off: *Find out the average age of student body.*

- (a) Change the tables a bit: Notice that we need to add in an attribute `Age: INT` to both the `Student` and the `Professor` table; and a `GPA:Float` to `Student`; which can be done as follows:

```
mysql> alter table professor add Age Int Not Null;
Query OK, 7 rows affected (0.34 sec)
Records: 7 Duplicates: 0 Warnings: 0
```

```
mysql> desc professor;
```

Field	Type	Null	Key	Default	Extra
Id	int(11)		PRI	0	
Name	char(20)				
DeptId	char(4)	YES		NULL	
Age	int(11)			0	

```
4 rows in set (0.02 sec)
```

We then need to add in the missing data. We can use the Update to do it. But, the far easier way is to use the GUI interface.

```
mysql> select * from professor;
```

Id	Name	DeptId	Age
9406321	Jacob Taylor	MGT	45
101202303	John Smyth	CS	32
121232343	David Jones	EE	56
555666777	Mary Doe	CS	67
783432188	Adrian Jones	MGT	55
864297351	Qi Chen	MAt	34
900120450	White	MAT	43

7 rows in set (0.00 sec)

```
mysql> select * from student;
```

Id	Name	Address	Status	Age	GPA
23456789	Homer Simpson	Fox 5 Tv	senior	21	3.3
111111111	Jane Doe	123 Main St.	freshman	19	3.4
111223344	Mary Smith	1 Lake St.	freshman	21	3.5
123454321	Joe Blow	6 Yard Ct.	junior	20	3.2
666666666	Jesoph Public	666 Hollow Rd.	sophomore	21	3.3
987654321	Bart Simpson	Fox 5 Tv	senior	22	3.6

6 rows in set (0.00 sec)

(b) SQL code:

```
Select AVG(S.Age) From Student S;
```

(c) MySQL Code and the result.

```
mysql> select AVG(S.Age) From Student S;
```

AVG(S.Age)
20.6667

1 row in set (0.00 sec)

Kick off: Find out the minimum age among professors in the Management Department.

(a) SQL code:

```
Select Min(P.Age) From Professor P Where P.DeptId='MGT';
```

(b) MySQL code and the result:

```
mysql> Select Min(P.Age) From Professor P
-> Where P.DeptId='MGT';
+-----+
| Min(P.Age) |
+-----+
|          45 |
+-----+
1 row in set (0.00 sec)
```

Kick off: Find out the youngest professor(s) in the Management Department.

(a) SQL Code:

```
Select P.Name,P.Age From Professor P
Where P.DeptId='MGT' And
      P.Age=(Select Min(P1.Age)
             From Professor P1
             Where P1.DeptId='MGT');
```

(b) MySQL Code and the result:

```
mysql> Select P.Name,P.Age From Professor P
-> Where P.DeptId='MGT' And
->       P.Age=(Select Min(P1.Age)
->               From Professor P1
->               Where P1.DeptId='MGT');
```

```
+-----+-----+
| Name          | Age |
+-----+-----+
| Jacob Taylor  | 45  |
+-----+-----+
1 row in set (0.00 sec)
```

The following is a bit different from Query 5.31.

5.31'. Find out the student(s) with the highest GPA.

(a) SQL Code:

```
Select S.Name, S.Id From Student S
Where S.GPA >= (Select Max(S1.GPA)
                From Student S1);
```

(b) MySQL code and the result:

```
mysql> Select S.Name, S.Id From Student S
-> Where S.GPA >= (Select Max(S1.GPA)
->                From Student S1);
```

```
+-----+-----+
| Name          | Id          |
+-----+-----+
| Bart Simpson  | 987654321  |
+-----+-----+
1 row in set (0.00 sec)
```

5.32(a). *Get the number of professors in the Management Department.*

(a) SQL code:

```
select count(P.Name) From Professor P Where P.DeptId='MGT';
```

(b) MySQL code and the result.

```
mysql> select count(P.Name) From Professor P Where P.DeptId='MGT';
+-----+
| count(P.Name) |
+-----+
|                2 |
+-----+
1 row in set (0.06 sec)
```

5.32(b). *Get the number of different names of professors in the Management Department.*

(a) SQL code:

```
Select count(distinct P.Name) From Professor P Where P.DeptId='MGT';
```

(b) MySQL code and the result.

```
mysql> select count(distinct P.Name) From Professor P Where P.DeptId='MGT';
+-----+
| count(distinct P.Name) |
+-----+
|                            2 |
```

```
+-----+
1 row in set (0.03 sec)
```

In this case, the results are the same, because of the following data. This does not need to be the same in general.

```
mysql> select P.Name from Professor P where P.DeptId='MGT';
```

```
+-----+
| Name          |
+-----+
| Jacob Taylor  |
| Adrian Jones |
+-----+
2 rows in set (0.00 sec)
```

Groups: *Find out the number of courses, the average grade, that every student has taken.*

(a) SQL code:

```
Select T.StudId, Count(*) As NumCrs,
      Avg(T.Grade) As CrsAvg
From Transcript T
Group By T.StudId;
```

(b) MySQL code and the result.

```
mysql> Select T.StudId, Count(*) As NumCrs,
-> Avg(T.Grade) As CrsAvg
-> From Transcript T
-> Group By T.StudId;
```

```
+-----+-----+-----+
| StudId   | NumCrs | CrsAvg |
+-----+-----+-----+
| 23456789 |      2 |      0 |
| 11111111 |      3 |      0 |
| 123454321 |      3 |      0 |
| 666666666 |      3 |      0 |
| 987654321 |      2 |      0 |
+-----+-----+-----+
5 rows in set (0.00 sec)
```

Apparently, when applied to characters, `Avg` returns 0.

Groups: *Find out the number of professors and their average age in each department.*

(a) SQL code:

```
Select P.DeptId, count(P.Name) As DeptSize,
      Avg(P.Age) As AvgSize
From Professor P
Group By P.DeptId;
```

(b) MySQL code and the result:

```
mysql> Select P.DeptId, count(P.Name) As DeptSize,
->      Avg(P.Age) As AvgSize
-> From Professor P
-> Group By P.DeptId;
```

```
+-----+-----+-----+
| DeptId | DeptSize | AvgSize |
+-----+-----+-----+
| CS     |         2 | 49.5000 |
| EE     |         1 | 56.0000 |
| MAt    |         2 | 38.5000 |
| MGT    |         2 | 50.0000 |
+-----+-----+-----+
4 rows in set (0.00 sec)
```

Order By: *Find out the number of professors and their average age in each department, ordered by their department name.*

(a) SQL code:

```
Select P.DeptId As DeptName, count(P.Name) As DeptSize,
      Avg(P.Age) As AvgSize
From Professor P
Group By P.DeptId
Order By DeptName;
```

(b) MySQL code and the result:

```
mysql> Select P.DeptId As DeptName, count(P.Name) As DeptSize,
->      Avg(P.Age) As AvgSize
-> From Professor P
-> Group By P.DeptId
-> Order By DeptName;
```

```
+-----+-----+-----+
| DeptName | DeptSize | AvgSize |
+-----+-----+-----+
| CS       |         2 | 49.5000 |
| EE       |         1 | 56.0000 |
+-----+-----+-----+
```

```

| MA+      |          2 | 38.5000 |
| MGT      |          2 | 50.0000 |
+-----+-----+-----+
4 rows in set (0.00 sec)

```

Labwork 3.4:

1. The following queries are based on the *Supplier* database that you have created in Labwork 2:
 - Get total number of parts supplied by supplier S1.
 - Get the total quantity of part P1 supplied by S1.
 - Get supplier names for those with status less than the current maximum status in the Supplier table.

For each of them, go through the following steps, using the table instances as shown in Labwork 2:

- (a) Use the English query to get an output.
 - (b) Write an *MySQL* query and run it to get an output.
 - (c) If the above three outputs are not the same, check and correct your work.
2. Complete Exercises 5.10 (a-d) and 5.17 (a-d). For each of them, write up its *MySQL* code, together with the output you get after executing the code, the same as what we did for query 5.5.

You have to send in, for each query, the query itself, and the *MySQL* query, together with the results.

4 On the views

As we mentioned in the lecture [§5.2.8]textbook, the view concept provides an external, customized, view of a database. View as a technique is particularly useful when we want to decompose a rather complicated task into a bunch of smaller and/or simpler ones. *MySQL* does support this feature since ver. 5.0.

For example, if we want to insert a bunch of tuples into a table `easyClass`, which contains classes that are so easy that more than 20% of the students got A.

The `easyClass` table can be created as follows:

```

Create table easyClass (
  CrsCode    Char(6) Not Null,
  Semester   Char(6) Not Null,
  AceRate    Float,
  Primary key (CrsCode, Semester),
  Foreign key (CrsCode) references Course(CrsCode),
  Constraint grade_condition Check (Grade in ('A','B','C','D','F')));

```

```
mysql> desc easyClass;
```

```

+-----+-----+-----+-----+-----+-----+
| Field      | Type      | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| CrsCode    | char(6)   | NO   | PRI | NULL    |       |
| Semester   | char(6)   | NO   | PRI | NULL    |       |
| AceRate    | float     | YES  |     | NULL    |       |
+-----+-----+-----+-----+-----+-----+

```

```
3 rows in set (0.01 sec)
```

It will be pretty boring to enter all the tuples to this just created table. On the other hand, we have collected all the relevant information in the database, and we can automatically populate `easyClass` with the help of the view mechanism as follows:

We create a view to collect the number of students who failed a class for each class.

```

Create view ClassAce
  (CrsCode, Semester, Aced) As
  Select T.CrsCode,T.Semester,Count(*)
  From transcript T
  Where T.Grade='A'
  Group By T.CrsCode, T.Semester;

```

Although `ClassAce` is not a table, its structure can still be checked just like a table.

```
mysql> desc ClassAce;
```

```

+-----+-----+-----+-----+-----+-----+
| Field      | Type      | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| CrsCode    | char(6)   | NO   |     | NULL    |       |
| Semester   | char(6)   | NO   |     | NULL    |       |
| Aced       | bigint(21)| NO   |     | 0       |       |
+-----+-----+-----+-----+-----+-----+

```

```
3 rows in set (0.01 sec)
```

Similarly, we can create another view that collects the enrollment for each class.

```

Create view ClassEnrollment
      (CrsCode, Semester, Enrolled) As
  Select T.CrsCode,T.Semester,Count(*)
  From transcript T
  Group By T.CrsCode, T.Semester

```

```
mysql> desc ClassEnrollment;
```

```

+-----+-----+-----+-----+-----+-----+
| Field      | Type          | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| CrsCode    | char(6)       | NO   |     | NULL    |       |
| Semester   | char(6)       | NO   |     | NULL    |       |
| Enrolled   | bigint(21)    | NO   |     | 0       |       |
+-----+-----+-----+-----+-----+-----+
3 rows in set (0.00 sec)

```

Now, the table `easyClass` can be *populated* as follows:

```

Insert into easyClass(CrsCode,Semester,AceRate)
  Select A.CrsCode,A.Semester,A.Aced/E.Enrolled
  From ClassAce A, ClassEnrollment E
  Where A.CrsCode=E.CrsCode
        And A.Semester=E.Semester
        And (A.Aced/E.Enrolled)>0.2;

```

The output is as follows:

```

mysql> select * from easyClass;
+-----+-----+-----+
| CrsCode | Semester | AceRate |
+-----+-----+-----+
| CS305   | S1996    | 1       |
| CS315   | F1997    | 1       |
| EE101   | F1997    | 1       |
| MGT123  | F1994    | 0.5     |
+-----+-----+-----+
4 rows in set (0.01 sec)

```

To verify, we include the data from the original transcript: For example, everybody who took CS305 in S1996 got 'A'; while only half of those taking MGT123 in F1994 got 'A'; and no more than 20% of those taking, e.g., MAT123, got 'A', 0% to be exact.

```

mysql> select * from transcript;
+-----+-----+-----+-----+

```

StudId	CrsCode	Semester	Grade
23456789	CS305	S1996	A
23456789	EE101	F1995	B
111111111	EE101	F1997	A
111111111	MA123	F1997	B
111111111	MGT123	F1997	B
123454321	CS305	S1996	A
123454321	CS315	F1997	A
123454321	MAT123	S1996	C
666666666	EE101	S1991	B
666666666	MAT123	F1997	B
666666666	MGT123	F1994	A
987654321	CS305	F1995	C
987654321	MGT123	F1994	B

13 rows in set (0.00 sec)

It would be tough without using views.

Labwork 4:

1. Test out the above scripts with *MySQL* ver. 5.* yourself. Then, modify the given scripts to come up with a view `hardClasses`, that reports those classes in which more than 10% failed. Send in the *MySQL* code together with the original data as contained in the `Transcript` table², and the data obtained via the `hardClass` view.
2. 5.17 (e, f). For both assignments, send in the *MySQL* code together with the original data as contained in the `Transcript` table, and the data obtained via the view.
3. Complete exercise 5.27, and use the view created to generate a table `JohnSFavorite`, which collects the students that have got at least a ‘B’ from John Smyth, the courses they took with the professor, together with the grades.

5 MySQL and PHP

MySQL, as a (partial) implementation of the SQL specification, is a very good language for defining the structure of the database, and *generating ad hoc queries*. However, to build meaningful applications, the power of a full-fledged high-level programming language, such as Java, C++, or PHP, is needed.

²The current content of the `Transcript` table will obviously generate an empty report. Thus, please revise the content.

Moreover, in today's WEB age, lots of database programming are done over the Internet, using WEB, i.e., HTML, as a visual media. We will discuss, or have discussed, depending on when you take on this part, more fully in Part (I) of the lab notes, *A Gentler Introduction to PhP and Its Application in Database Programming*, about the connection between PhP and MySQL in database programming.

In this section, we merely present a PhP script, `displayQueryResult.php`, with which you can test out any query you have designed that deals with the student registration database.

```
<Html>
<!--This file, sendGeneralQuery.html, is to send a query, which will be picked up
and processed by showGeneralQueryResult.php -->
<Head>
  <Style Type="text/css">
    <!--The following specifies the style of this page. -->
    <!--
      Body, P, TD {color: black; font-family: verdana; font-size: 10 pt}
      H1 {color: black; font-family: arial; font-size:12 pt}
    -->
  </Style>
</Head>

<!--A table with only one row, consisting of two cells, the first, 1/6 width wise,
being the left cushion; and the rest of the page contains the form -->
<Table Border=0 cellPadding=10 Width=100%>
  <!--Now define the row-->
  <Tr>
    <!--The following cell specifies the left cushion edge-->
    <Td BgColor="F0F8FF" Align=Center VAlign=top Width=17%> </Td>

    <!--The following gives the right entry form part, completely white-->
    <Td BgColor="FFFFFF" Align=Left VAlign=Top Width=83%>

    <H1>Query submission form;</H1>
    <p>Enter your query and we will send you back the result.</p>

    <!--Data will be transmitted via the post method, and will be handled by the-->
    <!-- specific file. We will talk in detail about such methods, later -->
    <Form Method="post" Action="showGeneralQueryResult.php">
      <!--First input box, with a name "query", thus earmarked with the name -->
      <!-- $_POST['query'] in the handler. The value will be filled in by the user -->
      <Input Type="text" Size=25 Name="query">
      <BR><BR>
      <!--The submit button, with the name being $_POST['submit'], -->
```

```

        <!--and the message shown being 'Submit'-->
        <!--When clicked, the filled query is sent over -->
        <Input Type="submit" Name="submit" Value="Submit">
    </Form>

    </Td>
    <!--end of the row definition>
    </Tr>
</Table>
</Body>
</Html>

```

The following is the file `showGeneralQueryResult.php`, which handles the query sent over by the previous file.

```

<?php

    //Include all the functions needed to print out the content of a single table
    include("displayQueryResult.inc");
?>

<Html>
    <Head>
        <Title>The result of a Query</Title>
    </Head>
<Body>

    <!--A table with only one row, consisting of two cells, the first being the -->
    <!--left edge, 1/6; and the other contains the form, 5/6 -->
    <Table Border=0 cellPadding=10 Width=100%>
        <!--Now define the row-->
        <Tr>
            <!--The following cell gives the left cushion edge-->
            <Td BgColor="F0F8FF" Align=Center VAlign=top Width=17%> </Td>

            <!--The following gives the right entry form part, completely white-->
            <Td BgColor="FFFFFF" Align=Left VAlign=Top Width=83%>
                <?php

                    //Below gets the query passed over via the post method
                    $query_string=$_POST['query'];

                    //echo the query
                    print("The following displays the result of a query: $query_string.<BR><BR>");

```

```

        //Call the predefined function, as contained in displayQueryResult.inc,
        //to print out query result, with an appropriate border
        display_db_query($query_string, $global_dbh, TRUE, "Border=2");
        ?>
    </Td>
    <!--end of the row definition>
</Tr>
</Table>
</Body>
</Html>

```

This script includes the `displayQueryResult.inc`, which besides setting up the needed connection to the database, also defines the needed procedure to display the result of the query:

```

<?php

//This is where the private information is kept.
include("home/phpbook/phpbook-vars.inc");

//Set up all the other needed information
$global_dbh=mysql_connect($hostname, $user, $password)
    or die("Could not connect to database");

//Set the name of the database you want to work with
$db="registration";

//Select the database to work with
mysql_select_db($db, $global_dbh)
    or die("Could not select database");

//This the function to display the result of the query
function display_db_query($query_string, $connection, $header_bool, $table_params){

    //perform the database query
    $result_id=mysql_query($query_string, $connection)
        or die("display_db_query:". mysql_error());

    //find out the number of columns in result
    $column_count=mysql_num_fields($result_id)
        or die("display_db_query:". mysql_error());

    //Table form include optional HTML arguments pssed into function
    print("<Table $table_params >\n");
}

```

```

//Optionally print a bold header at top of table
if($header_bool){
    print("<Tr>");

    for ($column_num=0; $column_num< $column_count; $column_num++){
        $field_name=mysql_field_name($result_id, $column_num);
        print("<Th>$field_name</Th>");
    }
    print("</Tr>");
}

//print out the body of the table, using a cursor

while($row= mysql_fetch_row($result_id)){
    print("<Tr Align=left Valign=top>");
    for ($column_num=0; $column_num< $column_count; $column_num++){
        print("<Td>$row[$column_num]</Td>\n");
        print("</Tr>\n");
    }
    print("</Table>\n");
}
?>

```

This file itself also includes another one, `home/phpbook/phpbook-vars.inc`, storing user information, e.g.

```

<?php
//Critical data to make the connection
//You should not change the following line
$hostname='localhost';
//Change the following to your user name. The following assumes that your user name
//is "johnDoe".
$user='johnDoe';
//Change the following to your password. The following assumes that your password is
//"Colt45".
$password='Colt45';
?>

```

To test this program in a *php enabled* server such as `turing`, and assume your user name is `j_doe`, just place all the files in a folder, e.g., `PhPFiles`, under the Home folder of `turing`. Notice that the `phpbook-vars.inc` is placed in a folder `home/phpbook` under `PhPFiles`.

You can then run the program by *Opening* the following in IE:

```
http://turing.plymouth.edu/~j_doe/PhPFiles/sendGeneralQuery.html
```

Note:

1. You can use the same structure to set up the files `home/phpbook/phpbook-vars.inc`, and `displayQueryResult.inc`.
2. You can use the `display_db_query` function once it is defined, once you understand its syntax.
3. This could be a preliminary framework of your interface to your project.

Labwork 5:

1. Set up the appropriate file structure.
2. Put all the files, four of them, in the right place.
3. Test out the related scripts.
4. Use this script to test out the SQL related homework you should have done for the registration database.
5. Notice this script can only check out queries for the registration database. Make the necessary change so that you can use this script to test out the queries that you have written down for your group's project.

References

- [1] Kifer, M., Bernstein, A., and Lewis, P., *Database Systems (Introduction Version)*, (Second Ed.) Addison-Wesley, Boston, MA, 2005.
- [2] *MySQL 5.1 Reference Manual*, available from <http://dev.mysql.com/doc/refman/5.1/en/index.html>.