A Gentler Introduction to *MariaDB* Database Programming

Zhizhang Shen [*]
Dept. of Computer Science and Technology
Plymouth State University

November 2, 2021

Abstract

This is Part II of the lab notes prepared for the students of *CS 3600 Database Management Systems* for Fall 2021. This part introduces some basic *MariaDB* structures, using the *Student Registration Database* as contained in [2, §3.2].

After learning some of the basic *PHP* programming features, we come to this set of the notes to show how to define table structures, and populate tables, in such a database. We then explore many queries as suggested in [2, §5.2], test them out with *MariaDB* (ver 5.5.56), and show the results. Lab assignments are done with a sample *Supplier* database, initially suggested in [1].

We will then switch back to Part I of the labnotes, *A Gentler Introduction to PhP and Its Application in Database Programming,* to further integrate *MariaDB* and *PhP* to make it real. ☺

You can certainly download this document. But, considering the fact that this document is "live", the most current information is obtained by directly accessing this document from the course site for *CS 3600 Database Management Systems.*

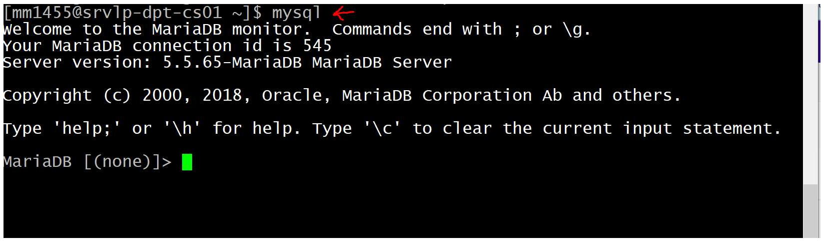
Every effort has been made to test out the queries as contained in this documentation, but please do let me know if you spot any error. ☹

[*] *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`.

Contents

1 Basic *MariaDB* commands

Once you log into `turing`, enter `mysql` at the prompt, as shown in Figure 1.



```
[mm1455@srvlp-dpt-cs01 ~]$ mysql
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 545
Server version: 5.5.65-MariaDB MariaDB Server

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]>
```

Figure 1: How to log into *mariaDB*?

Your `mariaDB` password is kept in a hidden file, `.my.cnf`, if you want to look at it, do the following:

```
/home/zshen > less .my.cnf
passWord
```

When you get the *MariaDB* prompt, i.e., “`MariaDB [(none)]>`”, you are ready to go. ☺

Below are some of the basic *MariaDB* commands that you need to use often. You might also want to go through [3, Sec. Getting Started].

1. What is the version of *MariaDB* running there?

   ```
   MariaDB [(none)]> Select version();
   +-----------------+
   | version()       |
   +-----------------+
   | 5.5.68-MariaDB |
   +-----------------+
   1 row in set (0.00 sec)
   ```

2. You should have already a database set up on *turing*, with its name being your login name [1]. In general, the following lets you find out all the existing databases:

   ```
   MariaDB [(none)]> show databases;
   +---------------+
   | Database      |
   +---------------+
   ```

[1] You should have received such a massage with your *MariaDB* credential in your `plymouth` mailbox. If you have yet to receive one, let me know.

```
| another      |
| fear         |
| geography    |
| jDoe         |
| mysql        |
| registration |
| shentest     |
| test         |
+--------------+
7 rows in set (0.00 sec)
```

3. You also have to specify which database to use, before using *it.* After logging into *MariaDB*, if Jane Doe, with her log-in name being `jDoe,` sees `MariaDB [(none)]`, she should do the following:

   ```
   MariaDB [(none)]> use jDoe
   Database changed
   MariaDB [jDoe]>
   ```

   On the other hand, if she wants to use another database, e.g., "`testDB`", she has to enter the following:

   ```
   MariaDB [(jDoe]> use  testDB;
   Reading table information for completion of table and column names
   You can turn off this feature to get a quicker startup with -A

   Database changed
   MariaDB [testDB]>
   ```

4. Before using a database, such as "`testDB`", you have to create it first:

   ```
   MariaDB [jDoe]> create database testDB;
   Query OK, 1 row affected (0.00 sec)

   MariaDB [jDoe]> show databases;
   +--------------+
   | Database     |
   +--------------+
   | another      |
   | fear         |
   | geography    |
   | jDoe         |
   ```

```
| mysql        |
| registration |
| shentest     |
| test         |
| testDB       |
+--------------+
8 rows in set (0.00 sec)
```

Note: You might not be able to create a database with *MariaDB* ☹ because you are not granted this access right. If that is the case, you will get the following message:

```
ERROR 1044 (42000): Access denied for user 'jDoe'@'%' to database 'testDB'
```

Then, you just create all the tables in your database, e.g., `jDoe`.

Throughout this set of notes, we use either “`testDB`” or “`registration`” as the database, while Jane Doe should use “`jDoe`”, if she cannot create such databases.

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

```
MariaDB [testDB]> show tables;
+------------------+
| Tables_in_testDB |
+------------------+
| author           |
| book             |
+------------------+
```

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

```
MariaDB [testDB]> Create Table aTable (
    ->   old varchar(10),
    ->   another integer);
```

If you want to check out the structure of this `aTable` table, use the `desc` (description) command:

```
MariaDB [testDB]> desc aTable;
+---------+-------------+------+-----+---------+-------+
| Field   | Type        | Null | Key | Default | Extra |
+---------+-------------+------+-----+---------+-------+
| old     | varchar(10) | YES  |     | NULL    |       |
| another | int(11)     | YES  |     | NULL    |       |
+---------+-------------+------+-----+---------+-------+
```

Here, "varchar(10)" refers to a variable-length string, which is a pretty popular data type used to represent character strings. For more details, check out [4, Sec. Varchar].

Let's make sure we do have our new table, aTable.

```
MariaDB [testDB]> show tables;
+-----------------+
| Tables_in_testDB |
+-----------------+
| aTable          |
| author          |
| book            |
+-----------------+
```

7. It is pretty easy to make mistakes when creating a table. And indeed, during its long span of life, it is necessary to revise various structural aspects of 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,* which uniquely identifies all the rows in a table.

```
MariaDB [testDB]>  alter table aTable change
    ->     old new integer
    ->     not null primary key;
Query OK, 0 rows affected (0.02 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

   It is now indeed changed.

```
MariaDB [testDB]> desc aTable;
+---------+---------+------+-----+---------+-------+
| Field   | Type    | Null | Key | Default | Extra |
+---------+---------+------+-----+---------+-------+
| new     | int(11) | NO   | PRI | NULL    |       |
| another | int(11) | YES  |     | NULL    |       |
+---------+---------+------+-----+---------+-------+
```

   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 aTable to ATable.

```
MariaDB [testDB]> show tables;
+-------------------+
| Tables_in_testDB |
+-------------------+
| aTable           |
| author           |
| book             |
+-------------------+

MariaDB [testDB]> alter table aTable rename to ATable;
Query OK, 0 rows affected (0.00 sec)

MariaDB [testDB]> show tables;'
+-------------------+
| Tables_in_testDB |
+-------------------+
| ATable           |
| author           |
| book             |
+-------------------+
```

Incidentally, the name of a table is case sensitive in *MariaDB*, but those of the columns, such as “`new`”, are not.

8. If you want to delete a table structure, you can use *drop*. For example, the following line “drops” `aTable`.

```
MariaDB [testDB]> drop table ATable;
Query OK, 0 rows affected (0.00 sec)
```

It is now gone.

```
MariaDB [testDB]> show tables;
+-------------------+
| Tables_in_testDB |
+-------------------+
| author           |
| book             |
+-------------------+
```

Notice that this pretty disruptive operation will delete everything, both the structure and the content of the table to be dropped.

9. We often do something in a research system and, when we are ready, 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, `testDB,` its structure and content, into `testDBdump.sql,` an *SQL* script, under `c:/temp`, which can be later executed in *turing*, for example, to restore the whole thing there.

   ```
   C:\Program Files\MySql\MySql Server 4.1\bin>
   mysqldump -u root -p testDB > c:/temp/testDBdump.sql
   Enter password: ********
   ```

   For more details, check out the "mysqldump" Section [4]. We will *not* discuss this aspect further in this course.

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 interfaces for the *MariaDB/PhP* combo is *PhPMariaDBAdmin,* which is available on *turing* via `https://turing.plymouth.edu/mysql/`. You need your *MariaDB* log-in information to get in.

Before you move over to this GUI, I would urge you to work out the stuff with the *MariaDB* prompt to achieve a basic understanding of the involved issues.

2 Table definition and population

We will demonstrate many of the database features with the `registration` database, which we could create with the following:

```
MariaDB [(none)]> create database registration;
Query OK, 1 row affected (0.00 sec)

MariaDB [testDB]> use registration;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
MariaDB [registration]>
```

If you could not create a database because of access right issues, bypass this part, and use, e.g., "`jDoe`" in place of "`registration`". But this is what should be done in real life.

2.1 The Student table

1. *Structure:* Figure 3.6 [2, Page 43]

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

2. *SQL code:* Page 49 [2]

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

     PRIMARY KEY (Id));
   ```

3. *MariaDB code:*

   ```
   MariaDB [registration]> Create Table Student (
       -> Id int(11) Not Null Primary key,
       -> Name varchar(20) Not Null,
       -> Address varchar(50),
       -> Status varchar(10) Default 'freshman');
   Query OK, 0 rows affected (0.01 sec)

   MariaDB [registration]> desc Student;
   +---------+-------------+------+-----+----------+-------+
   | Field   | Type        | Null | Key | Default  | Extra |
   +---------+-------------+------+-----+----------+-------+
   | Id      | int(11)     | NO   | PRI | NULL     |       |
   | Name    | varchar(20) | NO   |     | NULL     |       |
   | Address | varchar(50) | YES  |     | NULL     |       |
   | Status  | varchar(10) | YES  |     | freshman |       |
   +---------+-------------+------+-----+----------+-------+
   ```

4. *Data:* Figure 2.1. [2, 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');
```

To insert them into the `Student` table, just copy the above and past then on the *MariaDB* prompt.

```
MariaDB [registration]> 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)
```

Notice that the initiating zeros do not show.

2.2 The `Professor` table

1. *Structure:* Figure 3.6 [2, 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(4) Not Null,

  PRIMARY KEY (Id));
```

3. *MariaDB code:*

```
MariaDB [registration]> Create Table Professor (
    ->   Id INT Not Null Primary key,
    ->   Name varchar(20) Not Null,
    ->   DeptId varchar(4) Not Null);
Query OK, 0 rows affected (0.01 sec)

MariaDB [registration]> desc Professor;
+--------+-------------+------+-----+---------+-------+
| Field  | Type        | Null | Key | Default | Extra |
+--------+-------------+------+-----+---------+-------+
| Id     | int(11)     | NO   | PRI | NULL    |       |
| Name   | varchar(20) | NO   |     | NULL    |       |
| DeptId | varchar(4)  | NO   |     | NULL    |       |
+--------+-------------+------+-----+---------+-------+
```

4. *Data:* Figure 3.5, [2, pp. 39]

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

Insert into Professor (Id, Name, DeptId)
  Values (783432188, ’Adrian Jones’, ’MG’);

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

Insert into Professor (Id, Name, DeptId)
  Values (864297351, ’Qi Chen’, ’MA’);

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

Insert into Professor (Id, Name, DeptId)
```

```
  Values (009406321, ’Jacob Taylor’, ’MG’);

Insert into Professor (Id, Name, DeptId)
  Values (900120450, ’Ann White’, ’MA’);
```

Add them as well:

```
MariaDB [registration]> Select * From Professor;
+-----------+--------------+--------+
| Id        | Name         | DeptId |
+-----------+--------------+--------+
|   9406321 | Jacob Taylor | MG     |
| 101202303 | John Smyth   | CS     |
| 121232343 | David Jones  | EE     |
| 555666777 | Mary Doe     | CS     |
| 783432188 | Adrian Jones | MG     |
| 864297351 | Qi Chen      | MA     |
| 900120450 | Ann White    | MA     |
+-----------+--------------+--------+
```

2.3 The Course table

1. *Structure:* Figure 3.6,[2, 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. *MariaDB code:*

```
MariaDB [registration]> Create table Course (
    ->   CrsCode   varchar(6) Primary key,
    ->   DeptId    varchar(4),
    ->   CrsName   varchar(20),
    ->   Descr     varchar(100),
    ->   Unique key   (DeptId,CrsName));
Query OK, 0 rows affected (0.00 sec)

MariaDB [registration]> desc Course;
+---------+--------------+------+-----+---------+-------+
| Field   | Type         | Null | Key | Default | Extra |
+---------+--------------+------+-----+---------+-------+
| CrsCode | varchar(6)   | NO   | PRI | NULL    |       |
| DeptId  | varchar(4)   | YES  | MUL | NULL    |       |
| CrsName | varchar(20)  | YES  |     | NULL    |       |
| Descr   | varchar(100) | YES  |     | NULL    |       |
+---------+--------------+------+-----+---------+-------+
```

4. *Data:* Figure 3.5, [2, 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’);
```

Throw them into the table as we did before... .

```
MariaDB [registration]> Select * From Course;
+---------+--------+----------------------+--------------------------------+
| CrsCode | DeptId | CrsName              | Descr                          |
+---------+--------+----------------------+--------------------------------+
| CS305   | CS     | Database Systems.    | On the road to high-paying job |
| CS315   | CS     | Transaction Processi | Recover from your worst crashes |
| EE101   | EE     | Electronic Circuits  | Build your own computer        |
| MAT123  | MAT    | Algebra              | The world where 2+2=5          |
| MGT123  | MGT    | Market Analysis      | Get rich quick                 |
+---------+--------+----------------------+--------------------------------+
```

2.4 The Transcript table

1. *Structure:* Figure 3.6, [2, Page 43]

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

2. *SQL code:* Query 3.2 [2, Page 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))
```

We now add in foreign keys, as shown in [2, Page 94]. Notice that it also contains a foreign key on semester table, but that table does not exist, thus deleted in the *MariaDB* code.

```
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))
```

3. *MariaDB code:*

```
MariaDB [registration]> Create table Transcript (
    ->   StudId     INT Not Null,
    ->   CrsCode    varchar(6) Not Null,
    ->   Semester   varchar(6) Not Null,
    ->   Grade      varchar(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));
Query OK, 0 rows affected (0.01 sec)
```

Let’s have a look at its structure.

```
MariaDB [registration]> desc Transcript;
+----------+------------+------+-----+---------+-------+
| Field    | Type       | Null | Key | Default | Extra |
+----------+------------+------+-----+---------+-------+
| StudId   | int(11)    | NO   | PRI | NULL    |       |
| CrsCode  | varchar(6) | NO   | PRI | NULL    |       |
| Semester | varchar(6) | NO   | PRI | NULL    |       |
| Grade    | varchar(1) | YES  |     | NULL    |       |
+----------+------------+------+-----+---------+-------+
```

4. *Data:* Figure 3.5 [2, Page 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, ’MGT123’, ’F1994’, ’A’);

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’, ’F1995’, ’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’);
```

After adding them in, we have the following table instance for the moment.

```
MariaDB [registration]> 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   | F1995    | 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     |
+-----------+---------+----------+-------+
```

Notice that the following won’t work, as it violates the primary key constraint, as it would lead to a duplicated primary key, consisting of the same combination of `StudId`, `CrsCode` and `Semester`.

```
MariaDB [registration]> Insert into Transcript (StudId, CrsCode, Semester, Grade)
    ->   Values (111111111, ’MGT123’, ’F1997’, ’A’);
ERROR 1062 (23000): Duplicate entry ’111111111-MGT123-F1997’ for key ’PRIMARY’
```

Notice the following should not work, either, as it violates a foreign key constraint, as “222222222”, the value of `StudId` of the record, has yet to exist in the `Student` table. But, this record is accepted.

```
MariaDB [registration]> Insert into Transcript (StudId, CrsCode, Semester, Grade)
    -> Values (222222222, ’MGT123’, ’F1998’, ’A’);
Query OK, 1 row affected (0.01 sec)
MariaDB [registration]>
```

Thus, *MariaDB* has not correctly implemented the foreign key constraint.

By [4, Foreign Keys], “Foreign keys can only be used with storage engines that support them. The default InnoDB and the obsolete PBXT support foreign keys.”

2.4.1 Enforce a foreign key constraint in *MariaDB*

The following example is taken from the “Foreign Keys” section [4].

```
CREATE TABLE author (
id SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
name VARCHAR(100) NOT NULL
) ENGINE = InnoDB;

CREATE TABLE book (
id MEDIUMINT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
title VARCHAR(200) NOT NULL,
author_id SMALLINT UNSIGNED NOT NULL,
FOREIGN KEY (author_id) REFERENCES author (id)
) ENGINE = InnoDB;
```

Notice that this “InnoDB piece” enforces the foreign key constraint.

Both tables are created for the testDB database:

```
MariaDB [testDB]> show tables;
+------------------+
| Tables_in_testDB |
+------------------+
| author           |
| book             |
+------------------+
2 rows in set (0.00 sec)
```

The table book looks like the following:

```
MariaDB [testDB]> desc book;
+-----------+-----------------------+------+-----+---------+----------------+
| Field     | Type                  | Null | Key | Default | Extra          |
+-----------+-----------------------+------+-----+---------+----------------+
| id        | mediumint(8) unsigned | NO   | PRI | NULL    | auto_increment |
| title     | varchar(200)          | NO   |     | NULL    |                |
| author_id | smallint(5) unsigned  | NO   | MUL | NULL    |                |
+-----------+-----------------------+------+-----+---------+----------------+
3 rows in set (0.00 sec)
```

Question: Can we start to add in books? No, we have to add in the authors first, because of the foreign key constraint.

```
MariaDB [testDB]> INSERT INTO book (title, author_id)
    ->                  VALUES (’Necronomicon’, 1);
ERROR 1452 (23000): Cannot add or update a child row: a foreign key constraint
fails (‘testDB‘.‘book‘, CONSTRAINT ‘book_ibfk_1‘ FOREIGN KEY (‘author_id‘)
REFERENCES ‘author‘ (‘id‘))
```

Thus, we add in an author first.

```
MariaDB [testDB]> INSERT INTO author (name) VALUES (’Abdul Alhazred’);
Query OK, 1 row affected (0.00 sec)
```

Abdul is in...

```
MariaDB [testDB]> Select * From author;
+----+----------------+
| id | name           |
+----+----------------+
|  1 | Abdul Alhazred |
+----+----------------+
```

Add in another:

```
MariaDB [testDB]> INSERT INTO author (name) VALUES (’H.P. Lovecraft’);
Query OK, 1 row affected (0.00 sec)
```

Now, we have two authors entered into the `author` table.

```
MariaDB [testDB]> Select * From author;
+----+----------------+
| id | name           |
+----+----------------+
|  1 | Abdul Alhazred |
|  2 | H.P. Lovecraft |
+----+----------------+
```

We can now add in a book, or two.

```
MariaDB [testDB]> INSERT INTO book (title, author_id) VALUES
    -> (’The call of Cthulhu’, LAST_INSERT_ID()),
    -> (’The colour out of space’, LAST_INSERT_ID());
Query OK, 2 rows affected (0.01 sec)
Records: 2  Duplicates: 0  Warnings: 0

MariaDB [testDB]> select * from book;
+----+-------------------------+-----------+
| id | title                   | author_id |
+----+-------------------------+-----------+
|  1 | The call of Cthulhu     |         2 |
|  1 | The colour out of space |         2 |
+----+-------------------------+-----------+
```

Question: Can we delete an author? Yes or No, depending on who you want to delete. We can delete Abdul Alhazred, since no `book` row is related to this `author` row.

```
MariaDB [testDB]> delete from author where id=1;
Query OK, 1 row affected (0.02 sec)

MariaDB [testDB]> Select * From author;
+----+----------------+
| id | name           |
+----+----------------+
|  2 | H.P. Lovecraft |
+----+----------------+
```

Question: Can we delete H.P. Lovecraft? No, at least one row in the `book` table is connected to this `author` row under the foreign key constraint.

```
MariaDB [testDB]> delete from author where id=2;
ERROR 1451 (23000): Cannot delete or update a parent row:
a foreign key constraint fails (‘testDB‘.‘book‘, CONSTRAINT ‘book_ibfk_1‘
FOREIGN KEY (‘author_id‘) REFERENCES ‘author‘ (‘id‘))
```

Question: Can we drop the `author` table? No...

```
MariaDB [testDB]> drop table author;
ERROR 1217 (23000): Cannot delete or update a parent row:
a foreign key constraint fails
```

Indeed, we have to drop them in the sequence of `book` then `author` because of their dependence under the foreign-key constraint.

2.5 The `Teaching` table

1. *Structure:* Figure 3.6,[2, Page 43]

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

2. *SQL code:* [2, Page 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. *MariaDB code:*

   ```
   MariaDB [registration]> Create table Teaching (
       ->   ProfId    Int,
       ->   CrsCode   varchar(6) Not Null,
       ->   Semester  varchar(6) Not Null,
       ->   Primary Key (CrsCode,Semester),
       ->   Foreign key (ProfId) references Professor (Id),
       ->   Foreign key (CrsCode) references Course(CrsCode));
   Query OK, 0 rows affected (0.02 sec)
   ```

   What does it look alike?

```
MariaDB [registration]> desc Teaching;
+----------+------------+------+-----+---------+-------+
| Field    | Type       | Null | Key | Default | Extra |
+----------+------------+------+-----+---------+-------+
| ProfId   | int(11)    | YES  | MUL | NULL    |       |
| CrsCode  | varchar(6) | NO   | PRI | NULL    |       |
| Semester | varchar(6) | NO   | PRI | NULL    |       |
+----------+------------+------+-----+---------+-------+
```

4. *Data:* Figure 3.5, [2, Page 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’, ’F1997’);

Insert into Teaching (ProfId, CrsCode, Semester)
  Values (900120450, ’MAT123’, ’F1995’);

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’);
```

We will certainly throw in all the data:

```
MariaDB [registration]> Select * From Teaching;
```

```
+-----------+---------+----------+
| ProfId    | CrsCode | Semester |
+-----------+---------+----------+
|   9406321 | MGT123  | F1994    |
| 101202303 | CS305   | S1996    |
| 101202303 | CS315   | F1997    |
| 121232343 | EE101   | F1995    |
| 121232343 | EE101   | S1991    |
| 555666777 | CS305   | F1995    |
| 783432188 | MGT123  | F1997    |
| 900120450 | MAT123  | F1997    |
| 900120450 | MAT123  | F1995    |
+-----------+---------+----------+
```

Question: Can we ask a professor with `ProfId` being ‘9406123’ teach a course? No, this Id does not show up in the `Professor` table.

```
MariaDB [registration]> Insert into Teaching (ProfId, CrsCode, Semester)
    ->     Values (009406123, ’MGT123’, ’F1995’);
ERROR 1452 (23000): Cannot add or update a child row: a foreign key
constraint fails (‘registration‘.‘Teaching‘, CONSTRAINT ‘Teaching_ibfk_1‘
FOREIGN KEY (‘ProfId‘) REFERENCES ‘Professor‘ (‘Id‘))
```

Question: Can we ask two different professors to teach the same course in the same semester? You cannot because of the primary key constraint.

```
MariaDB [registration]> Insert into Teaching (ProfId, CrsCode, Semester)
    ->     Values (101202303, ’MGT123’, ’F1994’);
ERROR 1062 (23000): Duplicate entry ’MGT123-F1994’ for key ’PRIMARY’
```

This is not the case *here* at Plymouth State, where multiple professors could teach the same course in the same semester. Thus, when you design a database, you have to understand your users’s needs.

Labwork 2:

1. Create and populate all the aforementioned tables, five of them, in your database, accessible through your *MariaDB* account.

2. With the `Registration` instance that you have set up in Step 1, run the queries as given in pp. 12 through pp. 16 in the notes of Unit 3, namely, *RBA Basics via a Case Study.* Notice that the student IDs are no longer just 4 digit in this real instance.

3. Use Item 1 as an example, convert the following schema into a relational database, *SupplyPart*, also in your database.

   For all the ids, you may use `char(2)` as its type, use `char(20)` for the names, including that for `City`, `char(6)` for `Color`, `int(2)` for `Status`, `float` for `Weight.` Then populate all the tables:

   - `Supplier`

     ```
     Supplier(SupplierId:String, SName:String, Status:Integer, City:String)
       Key: {SupplierId)
     ```

SupplierId	SName	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, PName:String, Color:String,
          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}
       FK: SupplierId references Supplier(SupplierId)
           PartId references Part(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

We will now work on the queries, lots of them. You get to understand all the examples before jumping to the labworks.

The following examples are taken from [2, §5.2].

3.1 Simple queries

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

(a) Visual inspection: If you look at the `Professor` instance on Page 12, it is clear that David Jones is the only EE professor.

(b) Relational algebraic expression.

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

The above expression will start by selecting only those rows in the current instance of the `Professor` table, where the value of `DeptId` equals “EE”, then make a projection on the `Name` attribute to get following:

Name
David Jones

(c) *MariaDB* code: The above expression immediately leads to the following query.

```
MariaDB [registration]> 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) Visual inspection: If you look at the `Teaching` instance on Page 22, you can find that there is just one record, saying that a professor with her `ProfId` being 9406321 taught in Fall 1994. To find out whom she really is, we have to look at the `Professor` instance, and find that the professor is Jacob Taylor.

(b) Relational algebraic expression.

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

To apply it to the current instance, we start from inside out: the restriction $\sigma_{\texttt{Semester='F1994'}}(\texttt{Teaching})$ put a restriction on the `Teaching` table, and get us the following:

ProfId	CrsCode	Semester
9406321	MGT123	F1994

The next step, $(\texttt{Professor} \bowtie_{\texttt{Id=ProfId}} (\sigma_{\texttt{Semester='F1994'}}(\texttt{Teaching})))$, will join the above with the current `Professor` instance as follows:

Name	DeptId	ProfId	CrsCode	Semester
Jacob Taylor	MG	9406321	MGT123	F1994

Finally, the projection operation will project the above table on the `Name` attribute as follows:

Name
Jacob Taylor

(c) *MariaDB* code:

```
MariaDB [registration]> Select P.Name
    -> From Professor P, Teaching T
    -> Where P.Id=T.ProfId
    ->       And T.Semester='F1994';

+--------------+
| Name         |
+--------------+
| Jacob Taylor |
+--------------+
1 row in set (0.00 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) Visual inspection: If you look at the `Teaching` instance on Page 22, you can find that there are two courses taught in Fall 1995: EE101, taught by a professor with her `ProfId` being 121232343, and CS305, taught by a professor with her `ProfId` being 555666777. To find out those courses, and whom those professors are, we have to move over to the `Course` table; and the `Professor` table. We eventually find out Mary Doe taught *Database,* and David Jones taught *Electronic Circuits.*

(b) Relational algebraic expression: It is pretty clear that we have to do a restriction on the Semester in the `Teaching` table, then join the resulting table with both the `Course` and `Professor` table, and finally make a projection on the `CrsName` and the `Name` attributes.

Another "rougher" way is to construct the product of the three involved tables, then join the related rows, as well as applying a restriction of the desired restriction, as follows:

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

By going through the process, we should get the following result of applying the above to the current instance:

CrsName	Name
Database Systems	Mary Doe
Electronic Circuits	David Jones

(c) *MariaDB* code:

```
MariaDB [registration]> 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 Systems.   | Mary Doe    |
| Electronic Circuits | David Jones |
+---------------------+-------------+
```

This result can also be checked out with the current instances of both `Professor` and `Teaching.`

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

(a) Visual inspection: It will takes us a while to find out Homer, Jane, Joe, Joseph, Bart all took at least two different courses.

(b) 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.

$$\begin{aligned}&\pi_{\texttt{StudId}}(\sigma_{\texttt{CrsCode}\neq\texttt{CrsCode2}}(\texttt{Trancript}\\&\bowtie \texttt{Transcript}[\texttt{StudID}, \texttt{CrscCode2}, \texttt{Semester2}, \texttt{Grade2}]))\end{aligned}$$

We dare not go through the process here, since the intermediate step of product will result in a table with 122 rows. ☹

(c) *MariaDB* code:

```
MariaDB [register]> Select distinct S.Name
    -> From Transcript T1, Transcript T2, Student S
    -> Where T1.CrsCode<>T2.CrsCode
    ->        And T1.StudId=T2.StudId
    ->    And T1.StudId=S.Id;

+---------------+
| Name          |
+---------------+
| Homer Simpson |
| Jane Doe      |
| Joe Blow      |
| Jesoph Public |
| Bart Simpson  |
+---------------+
5 rows in set (0.00 sec)
```

Question: Why "distinct"?

Answer: Why not? Try the above out without it... . Technically, *MariaDB* gives back a multiset. If you want to have a set instead, use the restrictive `distinct`.

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

```
MariaDB [registration]> 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  |
+-----------+---------+
```

We certainly only need the following:

```
MariaDB [registration]> 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  |
+-----------+---------+
```

We can give more descriptive names for the resulting attributes.

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

```
MariaDB [registration]> Select P.Name As Professor
    -> From Professor P Where DeptID='EE';



+-------------+
| Professor   |
+-------------+
| David Jones |
+-------------+
```

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

1. Visual inspection: The `Professor` instance shows that all but Jacob Taylor, don't work in the EE department.

2. Relational algebraic expression: Should be straightforward:

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

3. *MariaDB code:* Straightforward as well.

```
MariaDB [registration]> 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     |
+---------------+
```

Labwork 3.1: For each of the following queries, you have to do the following:

1. visually inspect the current instance of the *SupplyPart* database, to write down the result for the English query itself;
2. construct the RA expression of the query, and come up with its result in terms of the current *SupplyPart* instance;
3. construct an *MariaDB* query, run it with *MariaDB*, and get a result from the system;
4. redo the above work if any of the above three results do not match.

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

1. Get the names of all the parts stored in Rome.
2. Get the names of 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 the names of suppliers who supply part P2.
5. Get the names of suppliers who supply at least one red part.

You need to send me three results for each of the five queries, that is fifteen results in total.

3.2 Set operations

Notice that, with Version10.3, all the three set operations, `Union,Intersect,` and `Except,` are supported. But, with Version 5.5.36 installed in *turing*, only `Union` is.

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

(a) Visual inspection: If you look at the `Professor` instance, it is clear that both John Smyth and Mary Doe are Computer Science professors, and David Jones are EE professor.

(b) Relational algebraic expression and its result:

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

The first part, i.e., $\pi_{\texttt{Name}}(\sigma_{\texttt{DeptId}='\texttt{CS}'}(\texttt{Professor}))$ get us the following

Name
John Smyth
Mary Doe

The second part, $\pi_{\texttt{Name}}(\sigma_{\texttt{DeptId}='\texttt{EE}'}(\texttt{Professor}))$, gets us the following:

Name
David Jones

Finally, since both parts are union compatible, the whole expression leads to the following result:

Name
John Smyth
Mary Doe
David Jones

(c) *MariaDB* code and the result:

```
MariaDB [registration]> (Select P.Name From Professor P
    ->  Where DeptID='CS')
    ->  Union (Select P.Name From Professor P
    ->         Where DeptID='EE');

+-------------+
| Name        |
+-------------+
| John Smyth  |
| Mary Doe    |
| David Jones |
+-------------+
```

(d) Another form in *MariaDB* (*5.16*).

```
MariaDB [registration]> Select P.Name From Professor P
    -> Where P.DeptID='CS' OR P.DeptId='EE';
+-------------+
| Name        |
+-------------+
| John Smyth  |
| David Jones |
| Mary Doe    |
+-------------+
```

5.17: *Get those who are either CS faculty or taught a CS course.*

(a) Visual inspection: If you check out the `Registration` instance, there are just two CS faculty: John and Mary, and they are also the only one who taught a CS course.

(b) Algebraic expression: First, we find out CS faculty:

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

Now, those who taught a CS course. Here, word "`Like`" can be used to match patterns.

$$\pi_{Name}(\sigma_{\texttt{Professor.Id=Teaching.ProfId And Teaching.CrsCode Like }'\texttt{CS}'}(\texttt{Teaching} \times \texttt{Professor}))$$

Finally, use a union to get both sets together:

Name
John Smyth
Mary Doe

(c) *MariaDB* code: Should be straightforward.

```
MariaDB [registration]> 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   |
+------------+
```

5.20: *Get those who have taken both 'CS305' and 'CS315'*

(a) Visual inspection: It is easy to find out from the `Transcript` instance, on Page 16, then the `Student` instance that Joe, with his `StudId` bring 123454321 took both courses.

(b) Algebraic expression: It is easy to find those who took `CS 305` and those who took `CS 315`, then construct an intersection of both sets.

The following gets all who took `CS 305`:

$$\pi_{\texttt{Name}}(\texttt{Student} \bowtie_{Student.Id=Transcript.StudId} (\sigma_{\texttt{Crscode='CS305'}}\texttt{Transcript}))$$

This should bring back the following result:

Name
Homer Simpson
Joe Blow
Bart Simpson

The expression for those who took `CS 315` is in parallel:

$$\pi_{\texttt{Name}}(\texttt{Student} \bowtie_{Student.Id=Transcript.StudId} (\sigma_{\texttt{Crscode='CS315'}}\texttt{Transcript}))$$

This should bring back the following result:

Name
Joe Blow

The latter result is also the intersection of those two.

(c) *MariaDB* code: If you use `MariaDB` Version 3.10.0 or later, you should be able to use something like this:

```
Select S.Name
From Student S, Transcript T
Where S.Id=T.StudId And T.CrsCode=’CS305
INTERSECT
Select S.Name
From Student S, Transcript T
Where S.Id=T.StudId And T.CrsCode=’CS315
```

Unfortunately, we use an older version, *MariaDB* 5.5.56, which does not support `INTERSECT`:

```
MariaDB [registration]> (Select StudId from Transcript where CrsCode=’CS305’)
    -> Intersect
    -> (Select StudId from Transcript where CrsCode=’CS315’);
ERROR 1064 (42000): You have an error in your SQL syntax; check the
manual that corresponds to your MariaDB server version for the right
syntax to use near ’Intersect
(Select StudId from Transcript where CrsCode=’CS315’)’ at line 2
```

Same story goes with "Except":

```
MariaDB [registration]> (Select StudId from Transcript where CrsCode='CS305')
    -> Except
    -> (Select StudId from Transcript where CrsCode='CS315');
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual
that corresponds to your MariaDB server version for the right syntax to use
near 'Except
(Select StudId from Transcript where CrsCode='CS315')' at line 2
MariaDB [registration]>
```

Notice that, starting with version 10.3.0, *MariaDB* supports both operators. Please check the relevant links in the course page.
Thus we have to do the following:

```
MariaDB [register]> 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)
```

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

(a) Visual inspection: Well, from the `Professor` instance, it is easy to see that both John and Mary work in the CS department, and David in EE.

(b) Relational algebraic expression.

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

It is easy to find out, after a restriction, then a projection, that the result should be the following:

Name
John Smyth
Mary Doe
David Jones

(c) *MariaDB* code and the result:

```
MariaDB [registration]> Select P.Name From Professor P
    -> Where P.DeptId In ('CS', 'EE');

+-------------+
| Name        |
+-------------+
| John Smyth  |
| David Jones |
| Mary Doe    |
+-------------+
```

Labwork 3.2:

1. Check out the *MariaDB* site to find out the correct syntax, starting with Version 10.3.0, for all the three operators as "`Union, Intersect`" and "`Except`", with simple examples.

2. For each of the following queries, you have to do the following, using Example 5.15 as an example.

   (a) visually inspect the current instance of the *SupplyPart* database, and write down the result for the English query itself;

   (b) construct the RA expression of the query, and demonstrate the process of coming up with its result with the current *Registration* instance;

   (c) construct an *MariaDB* query, run it with *MariaDB*, and get a result;

   (d) redo the above work if any of the above three results do not match.

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

   (a) Get the names of 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 in either Rome or London and sell at least two kinds of parts.

   (e) Get supplier names for suppliers who do not supply red parts.

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 *MariaDB* v. 5.5, does not support all the structures, e.g., it does not support the quantifiers.

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

```
MariaDB [registration]> 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 |
+--------------+
```

This information can be confirmed with the current database instance.

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

```
MariaDB [registration]> Select S.Name From Student S
    -> Where S.Id Not in
    -> # A nested subquery
    ->       (Select T.StudId From Transcript  T);

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

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.*

```
MariaDB [registration]> Select distinct R.StudId,P.Id,R.CrsCode
    -> From Transcript R,Professor P
    -> Where R.CrsCode in
    -> # courses taught by P.Id in F1994
    ->     (Select T1.CrsCode From Teaching  T1
    ->      Where T1.ProfId=P.Id And T1.Semester=’F1994’);
```

```
+-----------+---------+---------+
| StudId    | Id      | CrsCode |
+-----------+---------+---------+
| 666666666 | 9406321 | MGT123  |
| 111111111 | 9406321 | MGT123  |
+-----------+---------+---------+
```

Question: Is the above answer correct? No! Jane Doe, with her Id being 111111111, did not take MGT123, which Prof. Taylor taught in F1994. She took it in F1997 with Prof. Jones.

We should also enforce this condition that someone not only has to take a course taught by a professor back in F1994, but she has to take *it in the same semester of Fall 1994,* with the assumption of this database that *only one professor teaches a course in a semester.*

```
MariaDB [registration]> Select distinct R.StudId,P.Id,R.CrsCode
    -> From Transcript R,Professor P
    -> Where R.Semester=’F1994’ And R.CrsCode in
    -> # courses taught by P.Id in F1994
    ->     (Select T1.CrsCode From Teaching  T1
    ->      Where T1.ProfId=P.Id And T1.Semester=’F1994’);
```

```
+-----------+---------+---------+
| StudId    | Id      | CrsCode |
+-----------+---------+---------+
| 666666666 | 9406321 | MGT123  |
+-----------+---------+---------+
```

We sometimes want to use the `Exists` quantifier.

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

```
MariaDB [registration]> Select S.Name 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%’);
```

The result is simply the following:

```
+---------------+
| Name          |
+---------------+
| Jane Doe      |
| Jesoph Public |
| Mary Smith    |
+---------------+
```

Wrap up: *Get all students who were taught by all the Computer Science professors.*

This is a fairly complicated example, and it will be as far as we will go. For derivation details, please check out the lecture notes.

```
MariaDB [registration]> Select Name From Student
    -> Where Id Not In (
    ->   Select Distinct S.Id
    ->   From Student S,
    ->   # All CS Professors
    ->      (Select P.Id From Professor P
    ->       Where P.DeptId=’CS’) As C
    ->   Where C.Id Not In
    ->   # Professors who has taught S
    ->      (Select T.ProfId
    ->       From Teaching T, Transcript R
    ->       Where T.CrsCode=R.CrsCode And
    ->           T.Semester=R.Semester And
    ->           S.Id=R.StudId));
+----------+
| Name     |
+----------+
| Joe Blow |
+----------+
```

Labwork 3.3:

1. What is the current version of *MariaDB*? Check out its support to the *All* and the *Exist* quantifiers. Give their respective syntax and an example of its application to the *SupplyPart* database.

2. For each of the following queries, you have to do the following:

   (a) visually inspect the current instance of the *Registration* database, to write down the results for the English query itself;

(b) construct a nested *MariaDB* query, run it with *MariaDB*, and get a result;

(c) redo the above work if the above two results do not match.

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

(a) Get all the details of those parts supplied by someone located in London.

(b) Get supplier names for suppliers who supply at least one red part.

(c) Get supplier names for those who supply nuts.

(d) Find supplier names who supply all the parts. (Hint: This is entirely parallel to the example as we went through in the class, i.e., the above *Wrap up* example on who has been taught by all the CS professors. Check out that part from Page 95 through 103 in the lecture notes.)

3.4 Aggregation

A database will be in existence for quite sometime, thus it has to go through some structural changes over the time. As an example, let's augment the tables in the `registration` data base a little 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:

```
MariaDB [registration]> alter table Professor add Age Int Not Null;
Query OK, 7 rows affected (0.34 sec)
Records: 7  Duplicates: 0  Warnings: 0

MariaDB [registration]> 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       |       |
+--------+----------+------+-----+---------+-------+
```

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, as we discussed in Section 1.1.

```
MariaDB [registration]> Select * from Professor;

+-----------+-------------+-------+-----+
| Id        | Name        | DeptId | Age |
+-----------+-------------+-------+-----+
|   9406321 | Jacob Taylor | MG     |  45 |
| 101202303 | John Smyth   | CS     |  32 |
| 121232343 | David Jones  | EE     |  56 |
| 555666777 | Mary Doe     | CS     |  67 |
| 783432188 | Adrian Jones | MG     |  55 |
| 864297351 | Qi Chen      | MA     |  34 |
| 900120450 | White        | MA     |  43 |
+-----------+--------------+--------+-----+
```

Also revise the Student table as follows:

```
MariaDB [registration]> 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 |
+-----------+---------------+----------------+-----------+-----+------+
```

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

Kick off: *Find out the average age of student body. MariaDB* Code and the result.

```
MariaDB [register]> Select ROUND(AVG(S.Age),1) From Student S;
+---------------------+
| ROUND(AVG(S.Age),1) |
+---------------------+
|                20.7 |
+---------------------+
1 row in set (0.00 sec)
```

Kick off: *Find out the minimum age among professors in the Management Department. MariaDB* code and the result:

```
MariaDB [registration]> Select Min(P.Age) From Professor P
    -> Where P.DeptId='MG';

+------------+
| Min(P.Age) |
+------------+
|         45 |
+------------+
```

Kick off: *Find out the youngest professor(s) in the Management Department. MariaDB* Code and the result:

```
MariaDB [registration]> Select P.Name,P.Age From Professor P
    -> Where P.DeptId='MG' And
    ->       P.Age=(Select Min(P1.Age)
    ->              From Professor P1
    ->              Where P1.DeptId='MG');

+--------------+-----+
| Name         | Age |
+--------------+-----+
| Jacob Taylor |  45 |
+--------------+-----+
```

The following is a bit different from Query 5.31 in [2].

5.31. *Find out the student(s) with the highest GPA. MariaDB* code and the result:

```
MariaDB [registration]> Select S.Name, S.Id From Student S
    -> Where S.GPA >= (Select Max(S1.GPA)
    ->                 From Student S1);
+--------------+-----------+
| Name         | Id        |
+--------------+-----------+
| Bart Simpson | 987654321 |
+--------------+-----------+
```

5.32(a). *Get the number of professors in the Management Department. MariaDB* code and the result.

```
MariaDB [registration]> Select count(P.Name) From Professor P
    -> Where P.DeptId='MG';
+---------------+
| count(P.Name) |
+---------------+
|             2 |
+---------------+
```

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

```
MariaDB [registration]> Select count(distinct P.Name)
    -> From Professor P Where P.DeptId=’MG’;
+-------------------------+
| count(distinct P.Name) |
+-------------------------+
|                       2 |
+-------------------------+
```

In this case, the word “distinct” does not make a difference, because of the following data. This does not need to be the same in general.

```
MariaDB [registration]> Select P.Name
    -> From Professor P where P.DeptId=’MG’;
+--------------+
| Name         |
+--------------+
| Jacob Taylor |
| Adrian Jones |
+--------------+
```

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

```
MariaDB [registration]> Select T.StudId, Count(*) As NumCrs,
    -> ROUND(Avg(T.Grade),1) As CrsAvg
    -> From Transcript T
    -> Group By T.StudId;
+-----------+--------+--------+
| StudId    | NumCrs | CrsAvg |
+-----------+--------+--------+
|  23456789 |      1 |    0.0 |
| 111111111 |      4 |    0.0 |
| 123454321 |      3 |    0.0 |
| 666666666 |      3 |    0.0 |
| 987654321 |      2 |    0.0 |
| 999999999 |      1 |    0.0 |
+-----------+--------+--------+
6 rows in set, 14 warnings (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.*

```
MariaDB [register]> Select P.DeptId, count(P.Name) As DeptSize,
    ->    ROUND(Avg(P.Age), 1) As AvgAge
    ->    From Professor P
    ->    Group By P.DeptId;
+--------+----------+--------+
| DeptId | DeptSize | AvgAge |
+--------+----------+--------+
| CS     |        2 |   49.5 |
| EE     |        1 |   56.0 |
| MA     |        2 |   36.0 |
| MG     |        2 |   48.5 |
+--------+----------+--------+
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.*

```
MariaDB [register]> Select P.DeptId As DeptName, count(P.Name) As DeptSize,
    -> ROUND(Avg(P.Age), 1) As AvgAge
    -> From Professor P
    -> Group By P.DeptId
    -> Order By DeptName;
+----------+----------+--------+
| DeptName | DeptSize | AvgAge |
+----------+----------+--------+
| CS       |        2 |   49.5 |
| EE       |        1 |   56.0 |
| MA       |        2 |   36.0 |
| MG       |        2 |   48.5 |
+----------+----------+--------+
4 rows in set (0.00 sec)
```

Labwork 3.4: For each of the following queries, you have to do the following:

1. visually inspect the current instance of the *Registration* database, to write down the results for the English query itself;

2. construct an *MariaDB* query, run it with *MariaDB*, and get a result;

3. redo the above work if the above two results do not match.

The following queries are based on the *SupplyPart* database that you have created in Labwork 2 or the *Registration* database, as extended at the beginning of this sub-section:

1. Find out the meaning and usage of the “sum” aggregation operator, and use it to get the total number of different parts supplied by supplier S1.
2. Get the total quantity of part P1 supplied by S1.
3. Get supplier names for those with status less than the current maximum status in the Supplier table.

To construct the following two queries, you need to first revise the registration database, as instructed at the beginning of this Section.

4. Find out the name of the youngest student.
5. Find the average age of students who received an ‘A’ for some course.

4 On the *Views*

As we mentioned in the lecture [§5.2.8] [2], the view concept provides an external, customized, perspective 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.

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:

```
MariaDB [registration]> Create table easyClass (
    ->   CrsCode    varchar(6) Not Null,
    ->   Semester   varchar(6) Not Null,
    ->   AceRate    float,
    ->   Primary key (CrsCode, Semester),
    ->   Foreign key (CrsCode) references Course(CrsCode))
    ->   ENGINE = InnoDB;
Query OK, 0 rows affected (0.02 sec)

MariaDB [registration]> desc easyClass;

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

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 aced a class for each class.

```
MariaDB [registration]> 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;
Query OK, 0 rows affected (0.01 sec)
```

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

```
MariaDB [registration]> 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)

MariaDB [registration]> Select * From ClassAce;

+---------+----------+------+
| CrsCode | Semester | Aced |
+---------+----------+------+
| CS305   | F1995    |    1 |
| CS305   | S1996    |    1 |
| CS315   | F1997    |    1 |
| EE101   | F1997    |    1 |
+---------+----------+------+
```

We can certainly verify these data from the `Transcript` table: one student got ‘A’ in CS305 back in F1995, etc..

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

```
MariaDB [registration]> Create view ClassEnrollment
    ->             (CrsCode, Semester, Enrolled) As
    ->   Select T.CrsCode,T.Semester,Count(*)
    ->   From Transcript T
    ->   Group By T.CrsCode, T.Semester;
```

```
MariaDB [registration]> desc ClassEnrollment;

+----------+------------+------+-----+---------+-------+
| Field    | Type       | Null | Key | Default | Extra |
+----------+------------+------+-----+---------+-------+
| CrsCode  | varchar(6) | NO   |     | NULL    |       |
| Semester | varchar(6) | NO   |     | NULL    |       |
| Enrolled | bigint(21) | NO   |     | 0       |       |
+----------+------------+------+-----+---------+-------+
```

We can certainly dig out the information as follows:

```
MariaDB [registration]> Select * From ClassEnrollment;

+---------+----------+----------+
| CrsCode | Semester | Enrolled |
+---------+----------+----------+
| CS305   | F1995    |        2 |
| CS305   | S1996    |        1 |
| CS315   | F1997    |        1 |
| EE101   | F1995    |        1 |
| EE101   | F1997    |        1 |
| EE101   | S1991    |        1 |
| MAT123  | F1997    |        2 |
| MAT123  | S1996    |        1 |
| MGT123  | F1994    |        1 |
| MGT123  | F1997    |        1 |
+---------+----------+----------+
```

We can also verify the above data from the `Transcript` table: two students, Homer and Bart, took CS305 back in F1995. Now, the table `easyClass` can be *populated* as follows:

```
MariaDB [registration]> 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;
Query OK, 5 rows affected (0.00 sec)
Records: 5  Duplicates: 0  Warnings: 0

MariaDB [registration]> select * from easyClass;
+---------+----------+---------+
| CrsCode | Semester | AceRate |
+---------+----------+---------+
| CS305   | F1995    |     0.5 |
| CS305   | S1996    |       1 |
| CS315   | F1997    |       1 |
| EE101   | F1997    |       1 |
| MGT123  | F1994    |     0.5 |
+---------+----------+---------+
5 rows in set (0.00 sec)
```

To verify, we include the data from the following `Transcript` instance [2]: Out of the two students, Joe (123454321) and Bart (987654321), who took CS305 back in F1995, only Joe got ‘A’, thus 50% of the students who took that course aced it. On the other hand, only one student took CS305 in S1996, and got ’A’, thus “AceRate” of this class in that semester is 100%. ☺

```
MariaDB [registration]> 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   | F1995    | 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     |
+-----------+---------+----------+-------+
```

Are you ready to do some work yourself?

Labwork 4:

1. (a) Study and understand all the scripts as shown in this section with *MariaDB*, with the current *registration* instance as shown in Section 2 of this set of notes.

[2] This instance is the same as that shown on Page 16 of this set of notes.

(b) Revise the current `Transcript` instance as shown on Page 16 of this set of notes into the following one [3].

```
MariaDB [registration]> select * from Transcript;
MariaDB [register]> select * from Transcript;
+-----------+---------+----------+-------+
| StudId    | CrsCode | Semester | Grade |
+-----------+---------+----------+-------+
|  23456789 | CS305   | S1996    | A     |
|  23456789 | EE101   | F1995    | A     |
| 111111111 | EE101   | F1997    | A     |
| 111111111 | MAT123  | F1997    | B     |
| 111111111 | MGT123  | F1997    | B     |
| 123454321 | CS305   | F1995    | C     |
| 123454321 | CS315   | F1997    | C     |
| 123454321 | MAT123  | F1995    | F     |
| 666666666 | EE101   | S1991    | F     |
| 666666666 | MAT123  | F1997    | B     |
| 666666666 | MGT123  | F1994    | A     |
| 987654321 | CS305   | F1995    | C     |
+-----------+---------+----------+-------+
```

(c) Modify the given scripts to come up with a view, `hardClass`, that reports those classes in which more than 10% of the students failed. Do I have to remind you about the ROUND function again?

Send in all the related *MariaDB* code as well as the output obtained by applying your code against the above revised `hardClass` view.

2. Complete 5.17 (e, f) in the textbook [2]. For both assignments, send in the *MariaDB* code, together with the output obtained by applying your code to the database instance. You need to add in a `Salary` attribute into the current `Professor` table [4]:

```
+-----------+--------------+--------+-----+--------+
| Id        | Name         | DeptId | Age | Salary |
+-----------+--------------+--------+-----+--------+
|   9406321 | Jacob Taylor | MG     |  42 |  30000 |
| 101202303 | John Smyth   | CS     |  32 |  40000 |
| 121232343 | David Jones  | EE     |  56 |  25000 |
| 555666777 | Mary Doe     | CS     |  67 |  40000 |
| 783432188 | Adrian Jones | MG     |  55 |  30000 |
```

[3] You might want to use the GUI interface as discussed in Section 1.1.

[4] You might want to use the `alter table` syntax as shown in Section 3.4 to add the “`Salary`” column, then use GUI as mentioned in Section 1.1 to add in the data.

```
| 864297351 | Qi Chen      | MA     |  34 |  35000 |
| 900120450 | Ann White    | MA     |  38 |  50000 |
+-----------+-------------+--------+-----+--------+
```

(e) Find the names of the professors whose salaries are at least 10% higher than the average salary of all professors.

(f) Find the names of all the professors whose salaries are at least 10% higher than the average salary of all professors in their departments. (Hint: Use views, as in (5.39).)

3. Complete exercise 5.27 in the textbook [2], as follows:

> "Using the relations `Teaching` and `Professor,` create a view of `Transcript` containing only rows corresponding to classes taught by John Smyth."

and utilize this just created view to generate a table `JohnSFavorite,` which collects the students that have got a 'B' or better from John Smyth, the courses they took with Professor Smyth, together with the respective grade.

Wrap up: Find out the names of the youngest straight 'A' students and their age [5].

5 *MariaDB* and *PhP*

As we have witnessed, *MariaDB*, as a (partial) implementation of the *SQL* specification, is very good at 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.

Furthermore, in today's WEB age, lots of database programming are done over the Internet, using *HTML* as a visual media.

Now that we have learned how to program in *MariaDB,* it is time for us to move back to Part (I) of the labnotes for this course, *A Gentler Introduction to PhP and Its Application in Database Programming,* to study the integration of *PhP* and *MariaDB* in database programming. Let's jump right in... . ☺

References

[1] Date, C.J., *An Introduction to Database Systems (8th Edition)*, Pearson, 2003.

[2] Kifer, M., Bernstein, A., and Lewis, P., `Database Systems (Introduction Version),(Second Ed.)` Addison-Wesley, Boston, MA, 2005.

[5] By a "straight 'A"' student, we mean someone who got 'A' in all the courses that she has taken. There might be several of them, thus the plural "*students*", all sharing the same age, thus the singular "*age*".

[3] MariaDB, *MariaDB Tutorial.* Available at `https://mariadb.org/learn/`.

[4] MariaDB, *The MariaDB Knowledge Base.* Available at `https://mariadb.com/kb/en/library/`