Griffith UNIVERSITY
Queensland, Australia

Master of Information Technology

'7623ICT Information and Security Management'

'Assessment 2'

Supervised by:

Dr Zhe Wang

Submitted by Group 119:

Heang SOK_s5204340;

Jingdi LIN_s5210032

Submitted on: 11/10/2020

Table of Contents

Introduction

To set up a reliable online learning platform for Remarkable University, this team designs and implements a database using SQL with proper datatypes and integrity constraints. Then create different users with specific roles and grant reasonable privileges to them, so they can carry out operations for the respective purpose. Finally, to guarantee the database security, it is necessary to use the prepared statement integrate with privileges setting as two protection layers and develop backup strategies for the database.

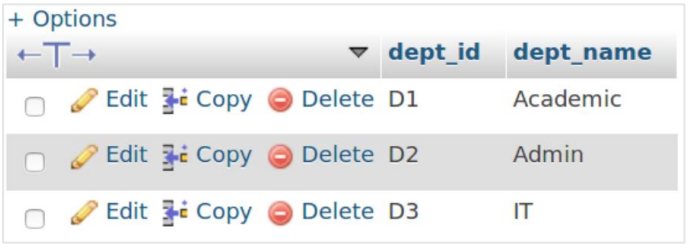
Part A: Database implementation

A.1. Database Design

The database is designed by this team according to First, Second, and Third Normalised Forms (1NF, 2NF, and 3NF). Thus, each column should contain only atomic values with the same type. Moreover, in a particular situation, the database design should describe the relationship between tables by utilising foreign keys.

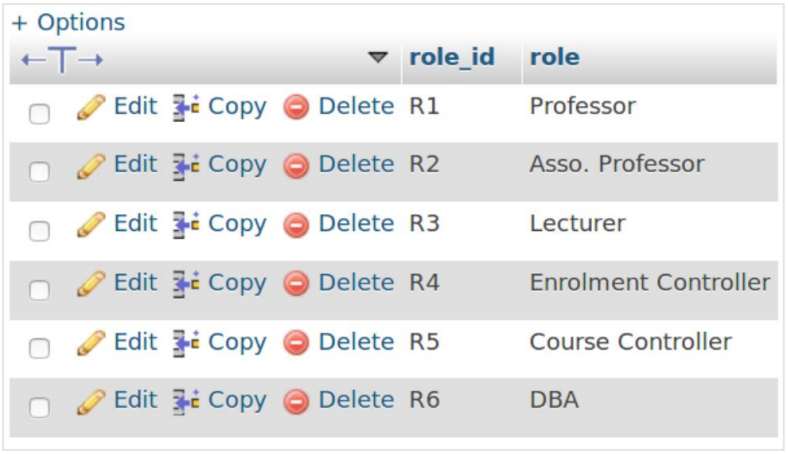
This team decided to decompose the provided sample tables called academic_staff and admin_staff because this structure is not practical and does not describe the relationship between staff and department properly; we should have a department table for each staff. For example, if the staff switches the department, it will be comfortable to update data. Moreover, one staff could have one or more roles. Therefore, a role table should be constructed as well. (Also see Appendix A: New Database Table Design)

Department:



Since academic_staff and admin_staff are in two different departments, it is important to categorise the department that the staff belongs to first. Moreover, one staff may work in many departments and one department can have many staff working in as well. For IT department, it belongs to Mr. Bruce Whyne that has a role as a database administrator (DBA).

Role:



+ Options

	role_id	role
Edit Copy Delete	R1	Professor
Edit Copy Delete	R2	Asso. Professor
Edit Copy Delete	R3	Lecturer
Edit Copy Delete	R4	Enrolment Controller
Edit Copy Delete	R5	Course Controller
Edit Copy Delete	R6	DBA

Then, roles are needed need to be classified as well. By doing so, this team can assign the roles to any old of new staff easily. For the role and staff relationship, we can infer that one staff can have one or more roles while one role can have many staffs.

Staff:

+ Options

	staff_id	dept_id	role_id	first_name	last_name	phone
Edit Copy Delete	S01	D1	R1	Seb	Binary	465796
Edit Copy Delete	S02	D1	R2	Jazz	Wood	246678
Edit Copy Delete	S03	D1	R3	Miguel	Franco	291880
Edit Copy Delete	S04	D2	R4	Cristiano	Penaldo	294472
Edit Copy Delete	S05	D2	R5	Lionel	Missy	497124
Edit Copy Delete	S06	D3	R6	Bruce	Whyne	364315

Finally, this team grouped each staff by their department and roles. This will allow the team to update staff information easily when there are changes regarding departments and roles. Note: the team has added the phone column in this new design.

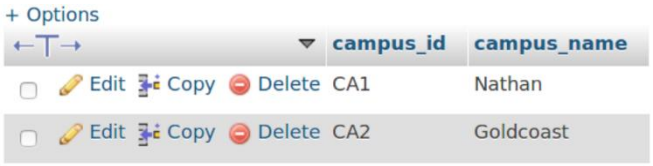
Next, this team redesigned the provided course table because it violates the 1NF. For example, the campus column, it was inserted GC, NA into a cell. Moreover, we need to show the relationship between the teachers and the courses as one teacher could have one or more courses, and vice versa.

Course:

course_id	course_name	convenor	staff_id	trimester	campus
101ICT	Information Management	Seb Binary	a01	1,2	GC
102ICT	Object Oriented Programming	Jazz Wood	a02	2	NA
101STA	Statistics	Jazz Wood	a02	1	GC, NA
101CS	Data Analytics	Seb Binary	a01	1	GC
102CS	Information Retrieval	Miguel Franco	a03	1,3	GC

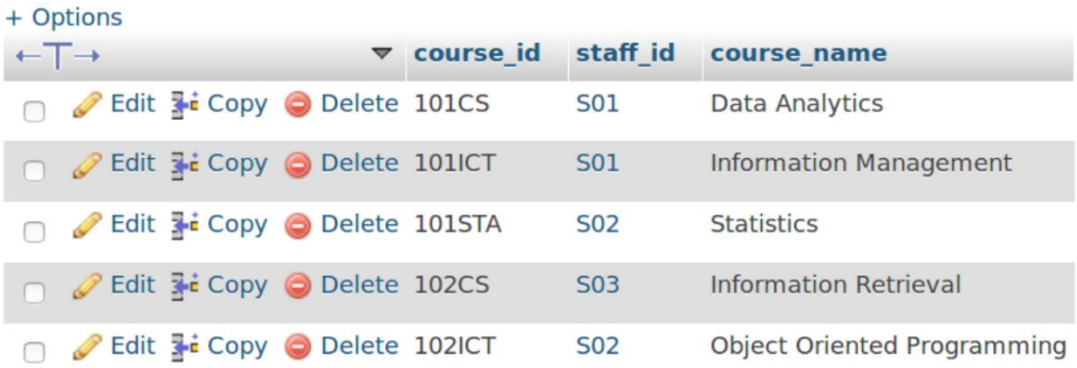
The following is the new design database table for the campus, course, and class:

Campus:



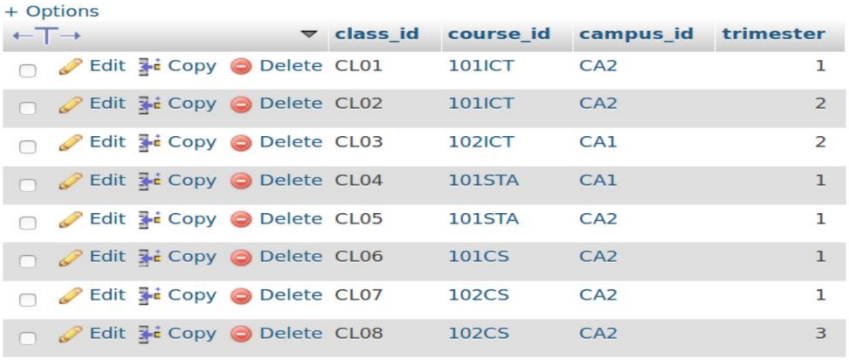
We first split the campus from the course table; it will come to be handy for showing which course at which location without the violation of 1NF.

Course:



Then this team uses the foreign key to link between the relation of staff and courses. As shown in the table above, one staff could teach more than one course and vice versa.

Class:



+ Options

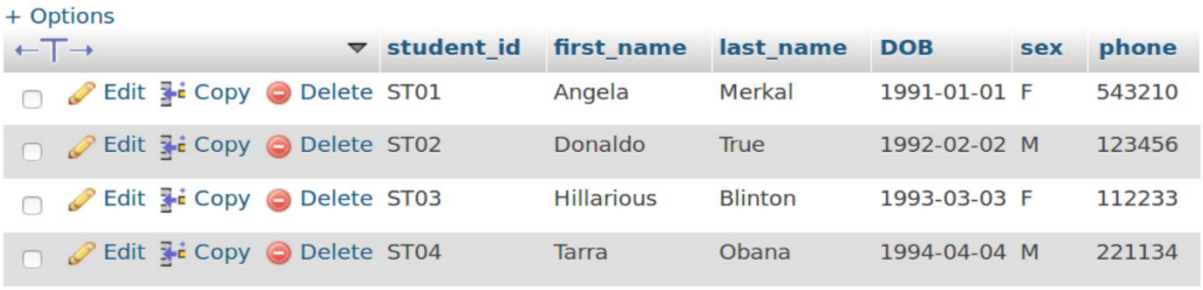
	class_id	course_id	campus_id	trimester
Edit Copy Delete	CL01	101ICT	CA2	1
Edit Copy Delete	CL02	101ICT	CA2	2
Edit Copy Delete	CL03	102ICT	CA1	2
Edit Copy Delete	CL04	101STA	CA1	1
Edit Copy Delete	CL05	101STA	CA2	1
Edit Copy Delete	CL06	101CS	CA2	1
Edit Copy Delete	CL07	102CS	CA2	1
Edit Copy Delete	CL08	102CS	CA2	3

Lastly, the team showed the available classes for each trimester in the table above. As shown in the table above, 101CS only available in trimester 1 at Goldcoast campus. The reason that this team does not include the year of the offered courses because this information is stable and apply for each year. For example, if a student wants to study the 102ICT course, he or she can enrol in trimester 2 only.

In this way it is easy for the team to update the information by using the foreign keys from the course and campus tables.

Actually, the team wanted to add the enrolment_id as the foreign key to link the student table to the enrolment table and vice versa. However, since the SQL statements in the script that this team has create are executed line by line in myphpadmin, it is impossible for this team to add enrolment_id as the foreign key now since it has not been created yet when we establish the student table. Therefore, the team decided to keep the student table the same. Moreover, the table is already in a good condition because it does not violate the 1NF, 2NF, and 3NF.

Student:



+ Options

	student_id	first_name	last_name	DOB	sex	phone
Edit Copy Delete	ST01	Angela	Merkal	1991-01-01	F	543210
Edit Copy Delete	ST02	Donaldo	True	1992-02-02	M	123456
Edit Copy Delete	ST03	Hillarious	Blinton	1993-03-03	F	112233
Edit Copy Delete	ST04	Tarra	Obana	1994-04-04	M	221134

However, the team has simplified the enrolment table. The below table is clear and concise without the repetition columns, and it also shows the relationship between student, class, and year that the students enrol. For example, student_id ST01 enrolled in 3 classes in 2017. This new design table also tells us that one student can enrol in many courses and each course can have many students, but an enrolment can have only one course and one student.

On the other hand, it is a little bit confusing to put the user_student_id and student_id together in this table. Indeed, using student_id as a foreign key is enough to link the students' information to the enrolment table. The purpose of putting these two attributes in the enrolment table is the team wanted to reduce the amount of codes when we create the view and it associated privileges as shown in the part B.3.1.

Enrolment:

enrolment_id	user_student_id	student_id	class_id	year
E01	UST01	ST01	CL01	2017
E02	UST01	ST01	CL05	2017
E03	UST01	ST01	CL06	2017
E04	UST02	ST02	CL03	2018
E05	UST02	ST02	CL06	2017
E06	UST03	ST03	CL08	2018
E07	UST04	ST04	CL03	2018
E08	UST04	ST04	CL04	2017
E09	UST04	ST04	CL06	2017
E10	UST04	ST04	CL07	2018

This team also decided to keep the grade table and make no changes because it is already in a good condition. However, unlike other tables that have primary keys, this table uses only the foreign keys from the enrolment table.

Grade:

+ Options

enrolment_id	score	grade
E01	75	6
E02	80	6
E03	92	7
E04	86	7
E05	71	5
E06	65	5
E07	55	4
E08	66	5
E09	80	6
E10	86	7

Since there are two types of users (staff and student), it would be effective to split the user accounts into two tables which are useraccount_staff and useraccount_student. As shown in the tables below, all the university staffs' user IDs start with "US" and students' user IDs start with "UST". In this way, we can tell which user IDs belong to staffs and which belong to students by looking at the initial letter of the user IDs. In addition, sometimes, a university staff might enrol as a student as well. Therefore, they need to have two different user IDs and having different types of user IDs (staff and student) in the database will give more flexibility and control of the data to the DBA. The staff_id and student_id in the tables below are the foreign keys which link to user associated information. In addition, the passwords are encrypted with a hash function as shown below. Note: in this step, this team has not created the users and set the privileges for the users yet.

Useraccount_staff:

user_staff_id	staff_id	password
US01	S01	40bd001563085fc35165329ea1ff5c5ecbdbbeef
US02	S02	0b6a63765cf0acb1022fc7c84ed8dcb104f221ed
US03	S03	f2d28ed87a287564c1ede7c7828e871bb90acf6c
US04	S04	0bab1dd8bdb38481cdd144b3acb6368847c0c662
US05	S05	fc7a734dba518f032608dfeb04f4eeb79f025aa7
US06	S06	cfa1150f1787186742a9a884b73a43d8cf219f9b

Useraccount_student:

user_student_id	student_id	password
UST01	ST01	eaa67f3a93d0acb08d8a5e8ff9866f51983b3c3b
UST02	ST02	1b75f1a8ff3bff3ef7a72c6935c3d01a94da8969
UST03	ST03	6216f8a75fd5bb3d5f22b6f9958cdede3fc086c2
UST04	ST04	4d7adc253fb88e2a45a2eb91e43b6e0ff7614587

A.2. Database Implementation

A.2.1. Create the Table and Insert the Data

This team use the following SQL statement to achieve the database implementation.

Also see the attached SQL script (.sql file) for more detail.

```
-- First, create a database for Remarkable University

DROP DATABASE IF EXISTS Remarkable_University;
CREATE DATABASE IF NOT EXISTS Remarkable_University;
USE Remarkable_University;

-----------------------------------------------------------------------------------------------------------------------------

-- Create a table for Remarkable University's department

DROP TABLE if EXISTS department;
CREATE TABLE department (
        dept_id CHAR(4) NOT NULL,
        dept_name VARCHAR(30) NOT NULL,
        CONSTRAINT dept_PK PRIMARY KEY (dept_id),
        UNIQUE(dept_name)
);
-- Insert values to the department table

INSERT INTO department VALUES ('D1','Academic');
INSERT INTO department VALUES ('D2','Admin');
INSERT INTO department VALUES ('D3','IT');

-----------------------------------------------------------------------------------------------------------------------------

-- Create a role table for staff

DROP TABLE if EXISTS role;
CREATE TABLE role (
        role_id CHAR(4) NOT NULL,
        role VARCHAR(30) NOT NULL,
        CONSTRAINT role_PK PRIMARY KEY (role_id),
```

```
	UNIQUE(role)
);
-- Insert values to the role table

INSERT INTO role VALUES ('R1','Professor');
INSERT INTO role VALUES ('R2','Asso. Professor');
INSERT INTO role VALUES ('R3','Lecturer');
INSERT INTO role VALUES ('R4','Enrolment Controller');
INSERT INTO role VALUES ('R5','Course Controller');
INSERT INTO role VALUES ('R6','DBA');
```

```
-- Create a table to store staff information, such as roles, departments, and personal information

DROP TABLE if EXISTS staff;
CREATE TABLE staff (
	staff_id CHAR(4) NOT NULL,
	dept_id CHAR(4) NOT NULL,
	role_id CHAR(4) NOT NULL,
	first_name VARCHAR(25) NOT NULL,
	last_name VARCHAR(25) NOT NULL,
	phone INT(20) NOT NULL,
	CONSTRAINT staff_PK PRIMARY KEY (staff_id),
	CONSTRAINT staff_dept_id_FK FOREIGN KEY (dept_id) REFERENCES department (dept_id) ON DELETE RESTRICT ON UPDATE CASCADE,
	CONSTRAINT staff_role_id_FK FOREIGN KEY (role_id) REFERENCES role (role_id) ON DELETE RESTRICT ON UPDATE CASCADE
);
-- Insert values to the staff table

INSERT INTO staff VALUES ('S01','D1','R1','Seb','Binary',465796);
INSERT INTO staff VALUES ('S02','D1','R2','Jazz','Wood',246678);
INSERT INTO staff VALUES ('S03','D1','R3','Miguel','Franco',291880);
INSERT INTO staff VALUES ('S04','D2','R4','Cristiano','Penaldo',294472);
```

```
INSERT INTO staff VALUES ('S05','D2','R5','Lionel','Missy',497124);
INSERT INTO staff VALUES ('S06','D3','R6','Bruce','Whyne',364315);
```

```
-- Create a table to store campus locations

DROP TABLE if EXISTS campus;
CREATE TABLE campus (
        campus_id CHAR(4) NOT NULL,
        campus_name VARCHAR(25) NOT NULL,
        CONSTRAINT campus_PK PRIMARY KEY (campus_id),
        UNIQUE(campus_name)
);
-- Insert values to the campus table

INSERT INTO campus VALUES ('CA1','Nathan');
INSERT INTO campus VALUES ('CA2','Goldcoast');
```

```
-- Create a table to store courses' information

DROP TABLE if EXISTS course;
CREATE TABLE course (
        course_id CHAR(6) NOT NULL,
        staff_id CHAR(4) NOT NULL,
        course_name VARCHAR(40) NOT NULL,
        UNIQUE(course_name),
        CONSTRAINT course_id_PK PRIMARY KEY (course_id),
        CONSTRAINT course_staff_id_FK FOREIGN KEY (staff_id) REFERENCES staff (staff_id) ON
DELETE RESTRICT ON UPDATE CASCADE
);
-- Insert values to the course table

INSERT INTO course VALUES ('101ICT','S01','Information Management');
INSERT INTO course VALUES ('102ICT','S02','Object Oriented Programming');
```

```
INSERT INTO course VALUES ('101STA','S02','Statistics');

INSERT INTO course VALUES ('101CS','S01','Data Analytics');

INSERT INTO course VALUES ('102CS','S03','Information Retrieval');
```

```
-- Create a table to store information about the available courses in each trimester for each year

DROP TABLE if EXISTS class;

CREATE TABLE class (
        class_id CHAR(4) NOT NULL,
        course_id CHAR(6) NOT NULL,
        campus_id CHAR(4) NOT NULL,
        trimester INT(4) NOT NULL,
        CONSTRAINT class_PK PRIMARY KEY (class_id),
        CONSTRAINT class_course_id_FK FOREIGN KEY (course_id) REFERENCES course (course_id)
ON DELETE RESTRICT ON UPDATE CASCADE,
        CONSTRAINT class_campus_id_FK FOREIGN KEY (campus_id) REFERENCES campus
(campus_id) ON DELETE RESTRICT ON UPDATE CASCADE
);

-- Insert values to the class table

INSERT INTO class VALUES ('CL01','101ICT','CA2',1);

INSERT INTO class VALUES ('CL02','101ICT','CA2',2);

INSERT INTO class VALUES ('CL03','102ICT','CA1',2);

INSERT INTO class VALUES ('CL04','101STA','CA1',1);

INSERT INTO class VALUES ('CL05','101STA','CA2',1);

INSERT INTO class VALUES ('CL06','101CS','CA2',1);

INSERT INTO class VALUES ('CL07','102CS','CA2',1);

INSERT INTO class VALUES ('CL08','102CS','CA2',3);
```

```
-- Create a table to store students' information

DROP TABLE if EXISTS student;
CREATE TABLE student (
        student_id CHAR(4) NOT NULL,
        first_name VARCHAR(25) NOT NULL,
        last_name VARCHAR(25) NOT NULL,
        DOB DATE NOT NULL,
        sex ENUM('M','F') NOT NULL,
        phone INT(20) NOT NULL,
        CONSTRAINT student_PK PRIMARY KEY (student_id)
);
-- Insert values to the student table

INSERT INTO student VALUES ('ST01','Angela','Merkal','1991-01-01','F',543210);
INSERT INTO student VALUES ('ST02','Donaldo','True','1992-02-02','M',123456);
INSERT INTO student VALUES ('ST03','Hillarious','Blinton','1993-03-03','F',112233);
INSERT INTO student VALUES ('ST04','Tarra','Obana','1994-04-04','M',221134);
-----------------------------------------------------------------------------------------------------------------------------------------------
-- Create a table to store staff user ID and passwords

DROP TABLE if EXISTS useraccount_staff;
CREATE TABLE useraccount_staff (
        user_staff_id CHAR(4) NOT NULL,
        staff_id CHAR(4) NOT NULL,
        password VARCHAR(50) NOT NULL,
        CONSTRAINT useraccount_staff_PK PRIMARY KEY (user_staff_id),
        CONSTRAINT useraccount_staff_staff_id_FK FOREIGN KEY (staff_id) REFERENCES staff
(staff_id) ON DELETE RESTRICT ON UPDATE CASCADE
);
-- Insert values to the useraccount_staff table

INSERT INTO useraccount_staff VALUES ('US01','S01',SHA1('123'));
```

```
INSERT INTO useraccount_staff VALUES ('US02','S02',SHA1('520'));

INSERT INTO useraccount_staff VALUES ('US03','S03',SHA1('886'));

INSERT INTO useraccount_staff VALUES ('US04','S04',SHA1('897'));

INSERT INTO useraccount_staff VALUES ('US05','S05',SHA1('777'));

INSERT INTO useraccount_staff VALUES ('US06','S06',SHA1('555'));

-- Note: SHA1 is a cryptographic hash function which converts the passwords to hexadecimal numbers

-- Note: This is just a table that store staff user ID and passwords

-----------------------------------------------------------------------------------------------------------------------------------

-- Create a table to store student's user ID and passwords

DROP TABLE if EXISTS useraccount_student;

CREATE TABLE useraccount_student (

        user_student_id CHAR(5) NOT NULL,

        student_id CHAR(4) NOT NULL,

        password VARCHAR(50) NOT NULL,

        CONSTRAINT useraccount_student_PK PRIMARY KEY (user_student_id),

        CONSTRAINT useraccount_student_student_id_FK FOREIGN KEY (student_id) REFERENCES
student (student_id) ON DELETE RESTRICT ON UPDATE CASCADE

);

-- Insert values to the useraccount_student table

INSERT INTO useraccount_student VALUES ('UST01','ST01',SHA1('888'));

INSERT INTO useraccount_student VALUES ('UST02','ST02',SHA1('867'));

INSERT INTO useraccount_student VALUES ('UST03','ST03',SHA1('111'));

INSERT INTO useraccount_student VALUES ('UST04','ST04',SHA1('889'));

-- Note: SHA1 is a cryptographic hash function which converts the passwords to hexadecimal numbers

-- Note: This is just a table that store staff user ID and passwords

-----------------------------------------------------------------------------------------------------------------------------------

-- Create a table to store students' enrolment information

DROP TABLE if EXISTS enrolment;
```

```
CREATE TABLE enrolment (
	enrolment_id CHAR(4) NOT NULL,
	user_student_id CHAR(5) NOT NULL,
	student_id CHAR(4) NOT NULL,
	class_id CHAR(4) NOT NULL,
	year YEAR NOT NULL,
	CONSTRAINT enrolment_PK PRIMARY KEY (enrolment_id),
	CONSTRAINT enrolment_user_student_id_FK FOREIGN KEY (user_student_id) REFERENCES useraccount_student (user_student_id) ON DELETE RESTRICT ON UPDATE CASCADE,
	CONSTRAINT enrolment_student_id_FK FOREIGN KEY (student_id) REFERENCES student (student_id) ON DELETE RESTRICT ON UPDATE CASCADE,
	CONSTRAINT enrolment_class_id_FK FOREIGN KEY (class_id) REFERENCES class (class_id) ON DELETE RESTRICT ON UPDATE CASCADE
);
-- Insert values to the enrolment table

INSERT INTO enrolment VALUES ('E01','UST01','ST01','CL01','2017');
INSERT INTO enrolment VALUES ('E02','UST01','ST01','CL05','2017');
INSERT INTO enrolment VALUES ('E03','UST01','ST01','CL06','2017');
INSERT INTO enrolment VALUES ('E04','UST02','ST02','CL03','2018');
INSERT INTO enrolment VALUES ('E05','UST02','ST02','CL06','2017');
INSERT INTO enrolment VALUES ('E06','UST03','ST03','CL08','2018');
INSERT INTO enrolment VALUES ('E07','UST04','ST04','CL03','2018');
INSERT INTO enrolment VALUES ('E08','UST04','ST04','CL04','2017');
INSERT INTO enrolment VALUES ('E09','UST04','ST04','CL06','2017');
INSERT INTO enrolment VALUES ('E10','UST04','ST04','CL07','2018');
-----------------------------------------------------------------------------------------------------------------------------------
-- Create a table to store grade data for students after the exam

DROP TABLE if EXISTS grade;
CREATE TABLE grade (
	enrolment_id CHAR(4) NOT NULL,
	score INT(3) NOT NULL,
```

```
	grade INT(2) NOT NULL,
	CONSTRAINT grade_enrolment_id_FK FOREIGN KEY (enrolment_id) REFERENCES enrolment (enrolment_id) ON DELETE RESTRICT ON UPDATE CASCADE
);
-- Insert values to the grade table

INSERT INTO grade VALUES ('E01',75,6);
INSERT INTO grade VALUES ('E02',80,6);
INSERT INTO grade VALUES ('E03',92,7);
INSERT INTO grade VALUES ('E04',86,7);
INSERT INTO grade VALUES ('E05',71,5);
INSERT INTO grade VALUES ('E06',65,5);
INSERT INTO grade VALUES ('E07',55,4);
INSERT INTO grade VALUES ('E08',66,5);
INSERT INTO grade VALUES ('E09',80,6);
INSERT INTO grade VALUES ('E10',86,7);
```

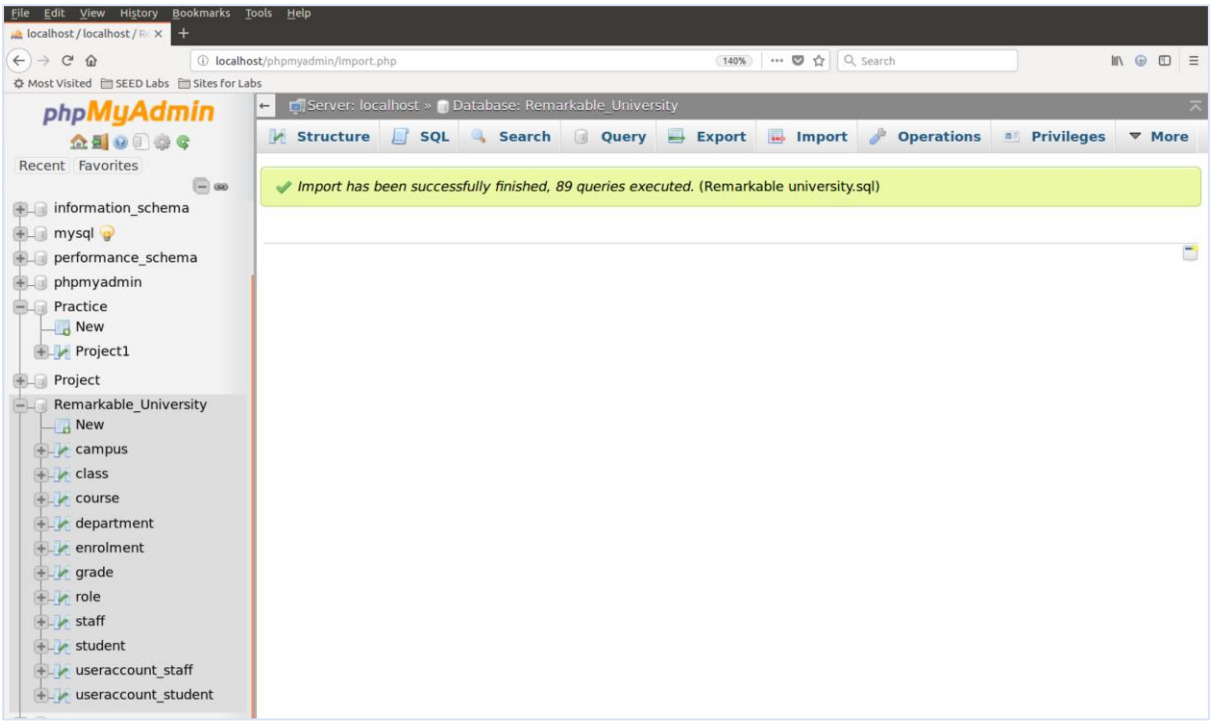
A.2.1. Comments

This team initiated with the DROP Database command as this script would be used to implement for many times to confirm whether the code could be executed successfully or not. The team also uses the DROP TABLE command every time they create a new table to prevent overlaps as this script is written by 2 students. In general situation, using the DROP DATABASE is enough.

Moreover, every command, data type, and constraint were written in capital letter to make it easy for this team to distinguish between the codes and content data. This team utilise the following command, data type, and constraint:

- Command: Drop DB/table, create DB/table, insert into
- Data type: Int, char, varchar, date
- Constraint: Not null, primary key, foreign key, unique, on delete, on update

The following screenshot shows that the script was successfully executed.



A.3. Database Backup and Recovery Implementation

A.3.1 Backup and Recovery strategy

To minimize the risk of data loss, it is important to backup databases regularly. Therefore, a well-developed backup and restore strategy is needed to help prevent data loss that caused by a variety of failures such as media failure, user errors, and hardware failure. The backup and restore strategy need to consider the speed, size, and the importance of the data. Therefore, this team decided to backup the student table, enrolment table, grade table, class table, and staff table more frequently than the other tables. This is because the university could contain many staff and students and their information is important. For example, if the university loses all the information about the student table, it will affect the enrolment table and grade table. This is similar to the staff table because it could affect the class table making the university does not know who is teaching what subject. In addition, the enrolment and grade information are also important because if this data are lost, the university would not be able to track the student payment and their marks with the associated class. Lastly, the class information is important to tell the available courses in each trimester for each year.

This team divides the following backup schedule in table 1 to ensure that the database is safe from loss:

Backup date	Man in charge	Role	Description
Weekly	Bruce Whyne	DBA	Backup student table, enrolment table, grade, class table, and staff table
Monthly	Bruce Whyne	DBA	Backup the entire database

Table 1: Backup schedule

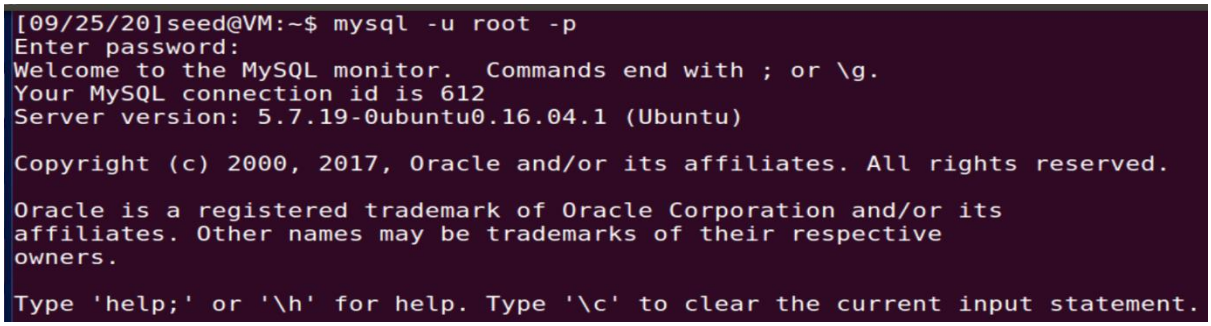
It is important to backup the entire database monthly as it will take more time and space on the hard drive to achieve, and for the student table, enrolment table, class table, and staff table are required to be backup on each week because of the important of the data. Another obvious reason that we should backup these table more frequently than the others is because most of the data in these table are keep changing. Taking enrolment table as an example, usually, the Australian universities allows students, both onshore and offshore, to pre-enrol before the class start in each trimester. Therefore, students can enrol whenever they want to, especially international students who need to enrol for a degree before they can apply for the visa. Thus, the number or the enrolment can be incremented by day or week and the universities should backup the data when there is an update.

A.3.2 Backup and Recovery mysqldump Command

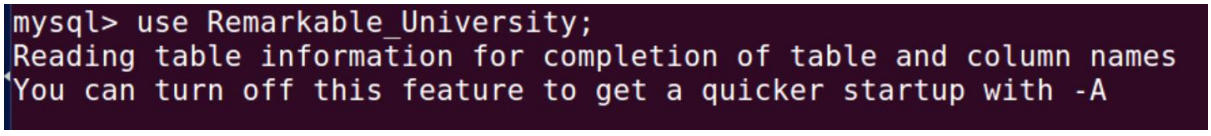
For implement the backup and recovery, this team uses mysqldump command in terminal. First login to the terminal using the following command and password:

mysql -u root -p

Password: seedubuntu



Then we select database by **use Remarkable_Univeristy;**



+ To show each table, we use this command: **show tables;**

```
mysql> show tables;
+--------------------------------+
| Tables_in_Remarkable_University |
+--------------------------------+
| campus                         |
| class                          |
| course                         |
| department                     |
| enrollment                     |
| grade                          |
| role                           |
| staff                          |
| student                        |
| useraccount_staff              |
| useraccount_student            |
+--------------------------------+
11 rows in set (0.00 sec)
```

To back up the Remarkable University's whole database, we use this statement: **mysqldump -u root -p Remarkable_University > ru.sql** . As shown in the screenshot below, a file called "ru.sql" was created and stored in /home/seed# after we executed the statement and input the password.

On the other hand, to restore the database, we are required to execute this statement and input the password: **mysql -u root -p Remarkable_University < ru.sql** . In this case, the terminal does not yield any messages and return to " /home/seed# " which means the databases is successfully restored by this team member.

```
root@VM:/home/seed# mysqldump -u root -p Remarkable_University > ru.sql
Enter password:
root@VM:/home/seed# ls
android  Customization  Documents  examples.desktop  lib    Pictures  ru.sql  Templates
bin      Desktop        Downloads  get-pip.py        Music  Public    source  Videos
root@VM:/home/seed# mysql -u root -p Remarkable_University < ru.sql
Enter password:
root@VM:/home/seed#
```

In order to backup the student table, enrolment table, grade table, class table, and staff table, we use this statement and then input the password: **mysqldump -u root -p Remarkable_University student enrolment grade class staff > ru_tables.sql** . As shown in the screenshot below, a file called "ru_tables.sql" was create and stored in /home/seed# after we executed the statement.

To restore the backup tables back, we use this statement and input the password: **mysql -u root -p Remarkable_University < ru_tables.sql** . After executed the statement, the terminal did not yield any messages and returned back to "/home/seed#" which means the tables were successfully restored by this team member.

```
[10/10/20]seed@VM:~$ sudo su
root@VM:/home/seed# mysqldump -u root -p Remarkable_University student enrolment grade class staff > ru_tables.sql
Enter password:
root@VM:/home/seed# ls
android  Customization  Documents  examples.desktop  lib    Pictures  ru.sql         source     Videos
bin      Desktop        Downloads  get-pip.py        Music  Public    ru_tables.sql  Templates
root@VM:/home/seed# mysql -u root -p Remarkable_University < ru_tables.sql
Enter password:
root@VM:/home/seed#
```

Part B. Users and Privileges

B.1. Create User

There are two alternative ways to create the user accounts for the university. It can either be created with the terminal in Linux ubuntu or myphpadmin. This team decided to do it with myphpadmin because it is much easier and faster to create users and check the results in myphpadmin GUI. In the Remarkable University, there are 2 types of users:

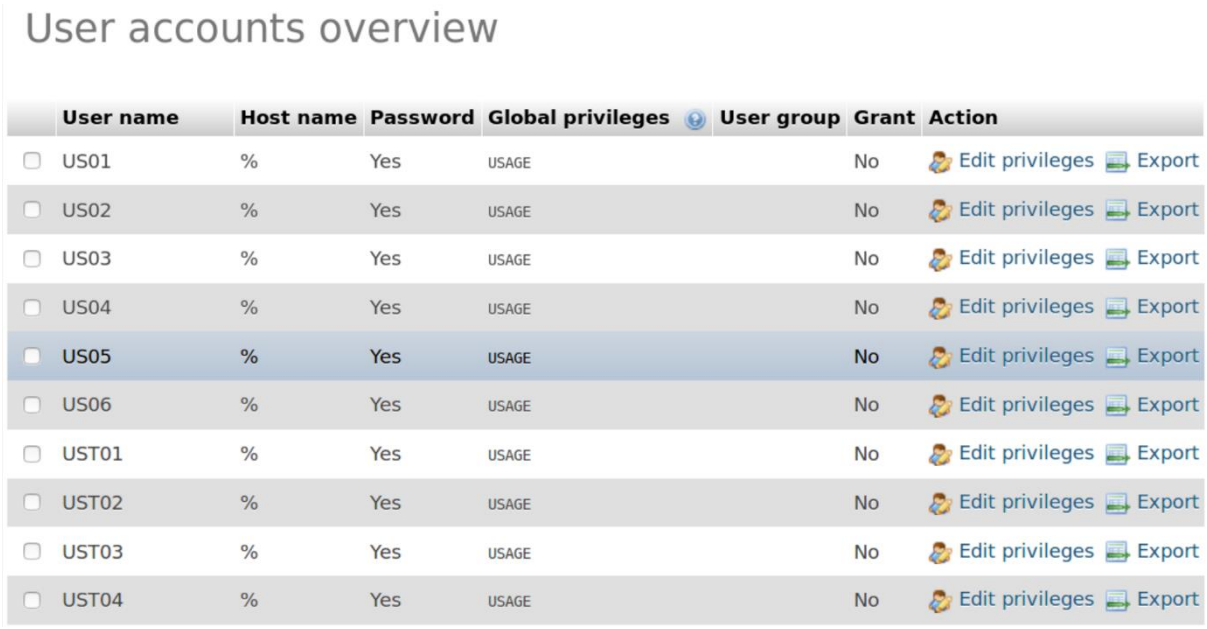
- Student
- Staff: Academic staff, Admin staff, and Database administrator

Thus, this team used the following statements to create users:

```
-- Create user account for students

DROP USER IF EXISTS 'UST01' @'%';

DROP USER IF EXISTS 'UST02' @'%';

DROP USER IF EXISTS 'UST03' @'%';

DROP USER IF EXISTS 'UST04' @'%';

CREATE USER 'UST01' @'%' IDENTIFIED BY '888';

CREATE USER 'UST02' @'%' IDENTIFIED BY '111';

CREATE USER 'UST03' @'%' IDENTIFIED BY '678';

CREATE USER 'UST04' @'%' IDENTIFIED BY '889';

-----------------------------------------------------------------------------------------------------------------------------------

-- Create user account for academic staff

DROP USER IF EXISTS 'US01' @'%';

DROP USER IF EXISTS 'US02' @'%';

DROP USER IF EXISTS 'US03' @'%';

CREATE USER 'US01' @'%' IDENTIFIED BY '123';

CREATE USER 'US02' @'%' IDENTIFIED BY '520';

CREATE USER 'US03' @'%' IDENTIFIED BY '886';

-----------------------------------------------------------------------------------------------------------------------------------

-- Create user account for admin staff_enroment controller

DROP USER IF EXISTS 'US04' @'%';

CREATE USER 'US04' @'%' IDENTIFIED BY '897';

-----------------------------------------------------------------------------------------------------------------------------------
```

```
-- Create user account for admin staff_course controller

DROP USER IF EXISTS 'US05' @'%';

CREATE USER 'US05' @'%' IDENTIFIED BY '777';

-----------------------------------------------------------------------------------------------------------------------

-- Create user account for database administrator

DROP USER IF EXISTS 'US06' @'%';

CREATE USER 'US06' @'%' IDENTIFIED BY '555';

-----------------------------------------------------------------------------------------------------------------------
```

As below screenshot shown, now the user accounts have been created. However, they have not been granted with any privileges yet. This means that they can log in but do not have the rights to perform specific actions.



User accounts overview

	User name	Host name	Password	Global privileges	User group	Grant	Action
☐	US01	%	Yes	USAGE		No	Edit privileges Export
☐	US02	%	Yes	USAGE		No	Edit privileges Export
☐	US03	%	Yes	USAGE		No	Edit privileges Export
☐	US04	%	Yes	USAGE		No	Edit privileges Export
☐	US05	%	Yes	USAGE		No	Edit privileges Export
☐	US06	%	Yes	USAGE		No	Edit privileges Export
☐	UST01	%	Yes	USAGE		No	Edit privileges Export
☐	UST02	%	Yes	USAGE		No	Edit privileges Export
☐	UST03	%	Yes	USAGE		No	Edit privileges Export
☐	UST04	%	Yes	USAGE		No	Edit privileges Export

B.2. Assign Table-Level Privileges

In order to grant privilege to each user properly, it is critical for this team to utilise the access matrix as show in the table 2 below:

Users	campus	class	course	depart-ment	Enrol-ment	grade	role	staff	student	user_ staff	user_ student
Students	---	---	R	---	---	---	---	---	---	---	---
Academic staff	---	---	RW	---	---	RW	---	---	R	---	---
Admin staff (enrolment)	---	---	---	---	RW	---	---	---	---	---	---
Admin staff (course)	---	---	RW	---	---	---	---	---	---	---	---
DBA	RWX	RWX	RWX	RWX	RWX	RWX	RWX	RWX	RWX	RWX	RWX

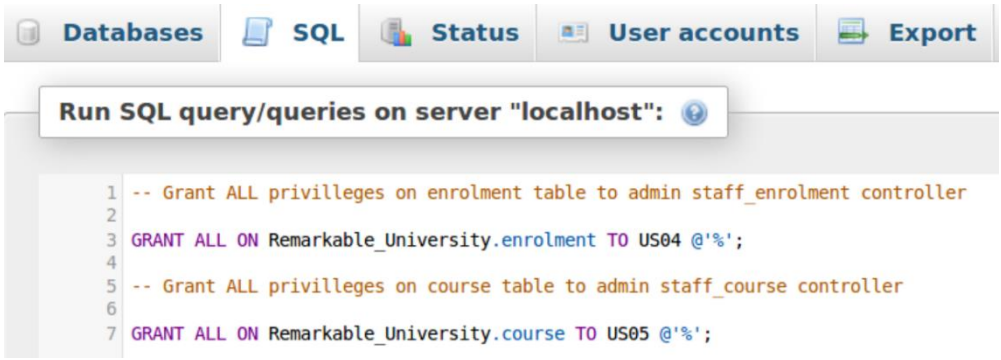
Table 2: Access matrix

This team uses the access matrix as a map to grant the privileges to the users according to their role and position. Each column in table 2 represent the table name in the Remarkable University's database. For each row, it shows the privileges that one user has over the table name in the Remarkable University's database. Moreover, one user can have either one or more privileges according to their roles, and these privileges include: ALL, CREATE VIEW, DELETE, INSERT, SELECT, UPDATE, etc.

B.2.1. Grant Privileges to admin staff

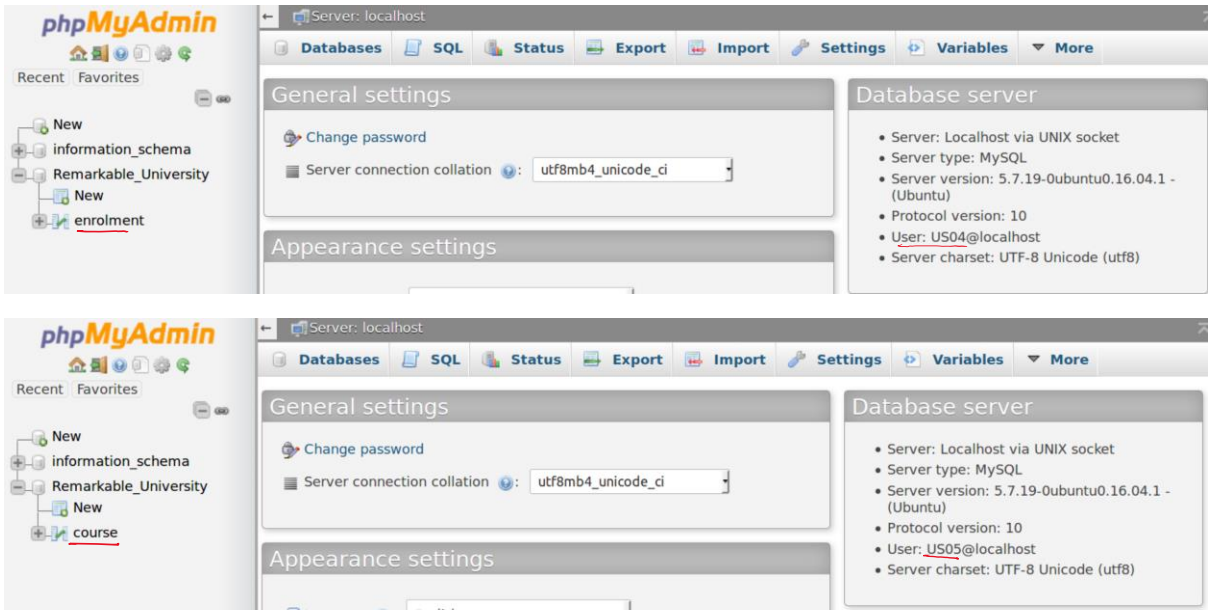
User account "US04" that has a role as an enrolment controller in admin department was granted with all privileges on the entire enrolment table. Therefore, he could perform the SELECT, INSERT, UPDATE, OR DELETE statements to modify this table only. Similarly, the user account "US05" that has a role as a course controller in admin department was also granted with all privileges; however, he could only modify the course table that he was assigned to in this case.

This team uses the following SQL commands to achieve this set of privileges:



Result:

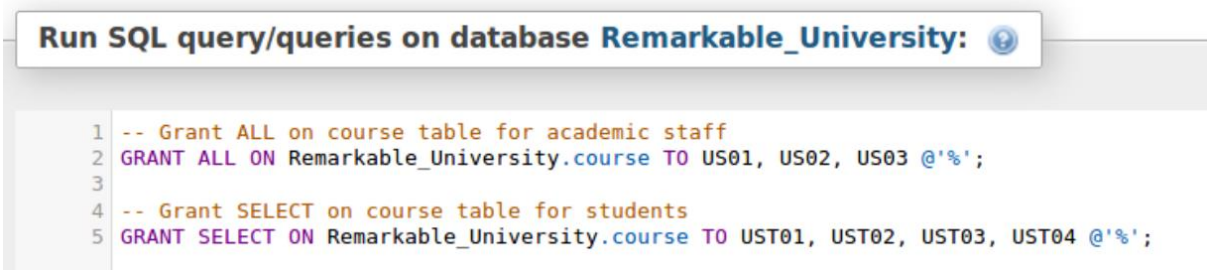
As shown in the screenshot below, user account "US04" has access to enrolment table only while user account "US05" has access to course table only.



B.2.2. Grant Privileges to Academic Staff and Students

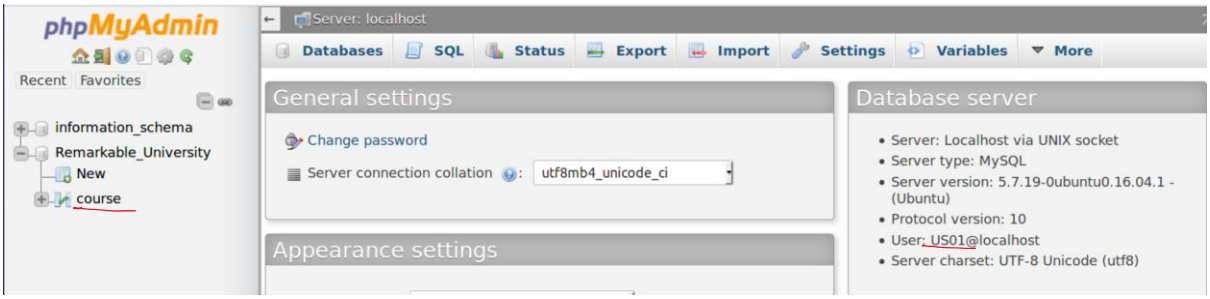
The academic staff that has user account “US01, US02, US03” can view the entire information about the course table and modify it while all the students (UST01, UST02, UST03, UST04) are granted the privilege to view the table only and cannot edit it.

This team uses the following SQL statements to achieve this set of privileges:



Result:

The academic staff has the access to the course information now, and he or she has all the privileges to control over this table.



On the other hand, the students now can access to the course, yet they can only view it.

B.2.3. Grant Privileges to Academic Staff to View Student Table

In this case, all the academic staff (US01, US02, US03) is granted to view only first_name, last_name and sex columns of the student table. For student’s private information like birthdays and phone numbers are not permitted.

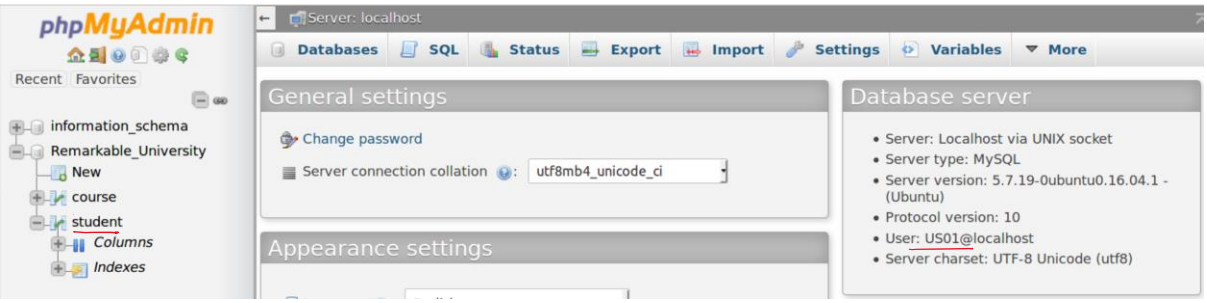
This team uses the following SQL command to achieve this set of privileges:

Run SQL query/queries on database Remarkable_University:

```
1  -- Grant SELECT on name and sex of the students only to adcademic staff
2
3  GRANT SELECT(first_name, last_name, sex) ON Remarkable_University.student TO US01, US02, US03 @'%';
4
```

Result:

Now the academic staff has access to one more table called student, yet they can view only the name and sex of the student



Show all | Number of rows: 25

Sort by key: None

+ Options

first_name	last_name	sex
Angela	Merkal	F
Donaldo	True	M
Hillarious	Blinton	F
Tarra	Obana	M

This is what the academic staff can view from their permissions.

B.2.4 Grant All the Privileges of the Entire Database to DBA

Since Mr. Bruce Whyne (user account= US06) is a database administrator, he was granted with all the privileges of the entire databases and tables, and he also had the right to grant the privileges to other employees. By doing so, it would be convenient for Mr. Bruce to perform particular tasks, such as backup, restore, and so on.

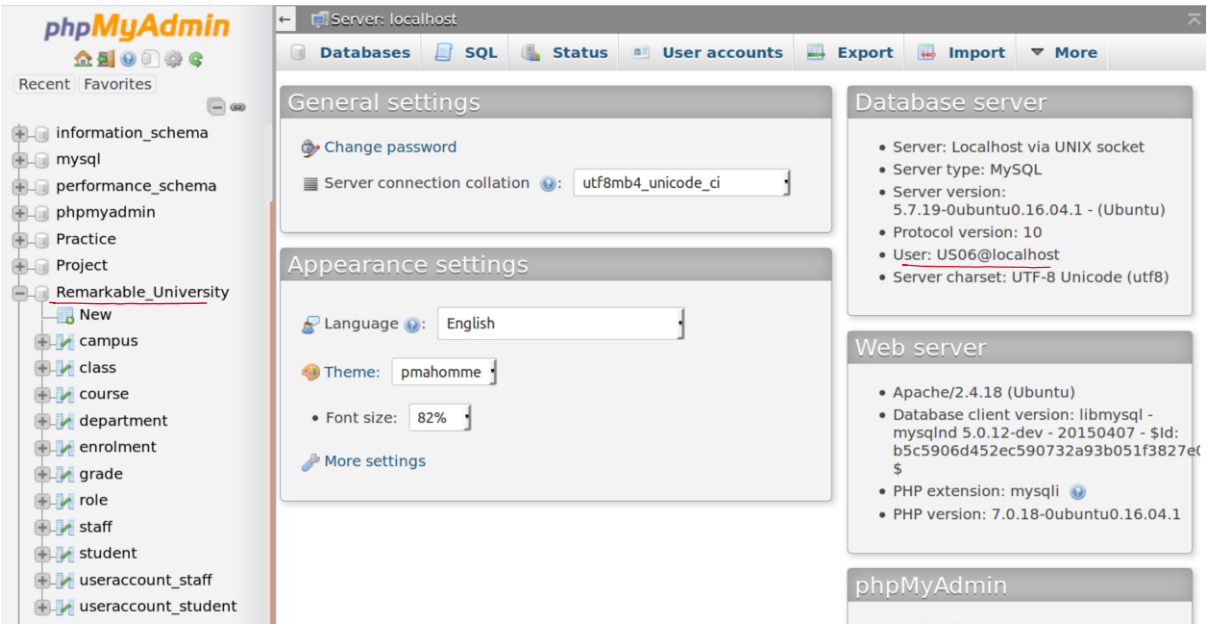
This team uses the following SQL command to achieve this set of privileges:

Run SQL query/queries on database Remarkable_University:

```
1  -- Grant ALL the privileges of the entire DB to DBA
2
3  GRANT ALL ON *.* TO US06 @'%' WITH GRANT OPTION;
```

Results:

As shown in the screenshot below, now Mr. Bruce has all the access and privileges to the databases and tables when he login with his user account.



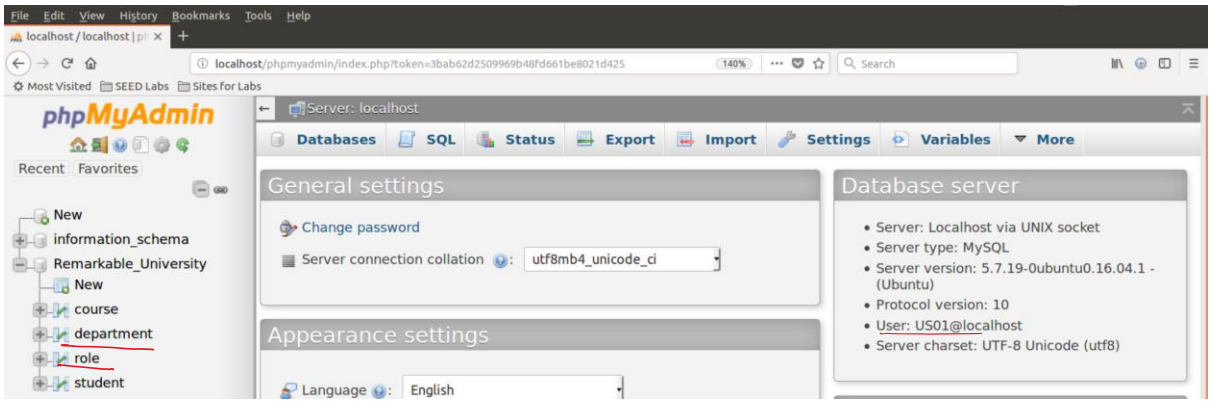
B.2.5. Grant Select Privilege on Department and Role Table to each staff

Sometimes, staffs may want to know the numbers of the departments within the university, or they may want to add more skills to their profession and then switch their role. In this case, it is important to grant select privilege on the department and role table to those staff so that they can plan a head. For example, the admin staff want to become an academic staff, but the academic staff is divided into professor, associated professor, and lecture. Therefore, the admin staff need to know all of these positions first, and then select and apply for the position that suit him/her.

According to the Oracle, we can grant only one object at a time to multiple users; therefore, this team used 2 lines of SQL statement, one is for granting the department table to each staff and another one is for the role table. Since, we just granted all the privileges of the database to DBA in B.2.4, we do not include him in these command lines.

Result:

Now all the staffs can view the department and role table as shown in the screenshot below.



B.3. Create Views and Related Privileges

In SQL, it allows the users to manipulate the view by joining two or more tables using CREATE VIEW command. Then this view can be saw or edited by the users that have been granted with the privileges.

B.3.1. Create a View and Grant Privilege to Students to See Only Their Grade

After submitted the assignments or exams, students need to check their score or grade at the end of the trimester, and some students do not want others to know about their results at all. Therefore, it is important to let the students to see only their grades. To achieve this requirement, this team created a view table called “my_grade” and showed only “user_student_id”, “calss_id”, “course_name”, and “grade” columns to them, and we had to join the “enrolment”, “class”, “course”, and “grade” tables since these columns are related to these 4 tables. Then we used “WHERE” clause to set the condition for the student to see only his or her grades. Finally, we granted the SELECT privilege of the view table for the students.

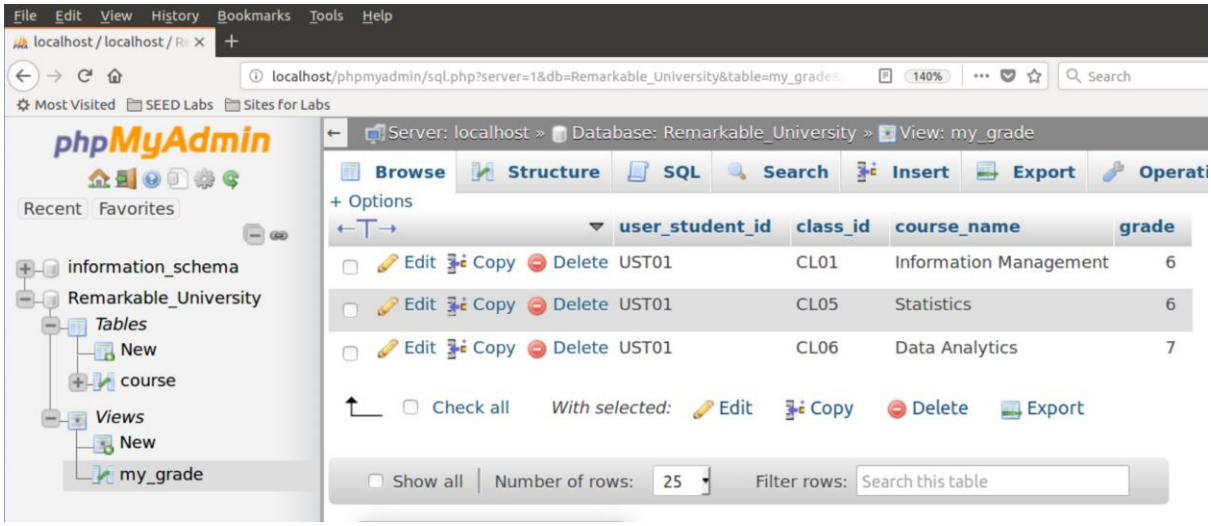
This team took a student user account called “UST01” as an example of the implementation and below is the SQL command that used for this example:

Run SQL query/queries on database Remarkable_University:

```
1 -- Create view and let students to see only their grade
2
3 CREATE VIEW Remarkable_University.my_grade as
4     SELECT enrolment.user_student_id, enrolment.class_id, course.course_name, grade.grade FROM Remarkable_University.enrolment,
Remarkable_University.course, Remarkable_University.class, Remarkable_University.grade
5     WHERE enrolment.enrolment_id = grade.enrolment_id AND enrolment.class_id = class.class_id AND class.course_id = course.course_id AND
enrolment.user_student_id = 'UST01';
6
7
8 GRANT SELECT ON my_grade TO UST01 @'%';
```

Result:

In this result, we learnt that the student user account called “UST01” had enrolled in 3 classes and received different grades; he could view only his own grades with the classes that he had enrolled.



B.3.2. Create a View and Grant Privileges to Let the Academics staff to See Only the Enrolment of the course that They Teach

The academic staff should see only the students that have enrolled in their course because it would be practical for them to manage the class, set up the teaching plans, and give marks to their students. To achieve this condition, this team members decided to create a view table named “102CS_enrolment” as an example and show only the “enrolment_id”, “user_student_id”, “first_name”, “last_name”, “course_id”, “campus_id”, “trimester”, and “year” columns to them by joining the “enrolment”, “class,” and “student” tables. Then the team used the “WHERE” clause to set the conditions that allow them to see only the students that has enrolled in their course. Finally, we granted the SELECT privilege of the view table named “102CS_enrolment” to the academic’s staff.

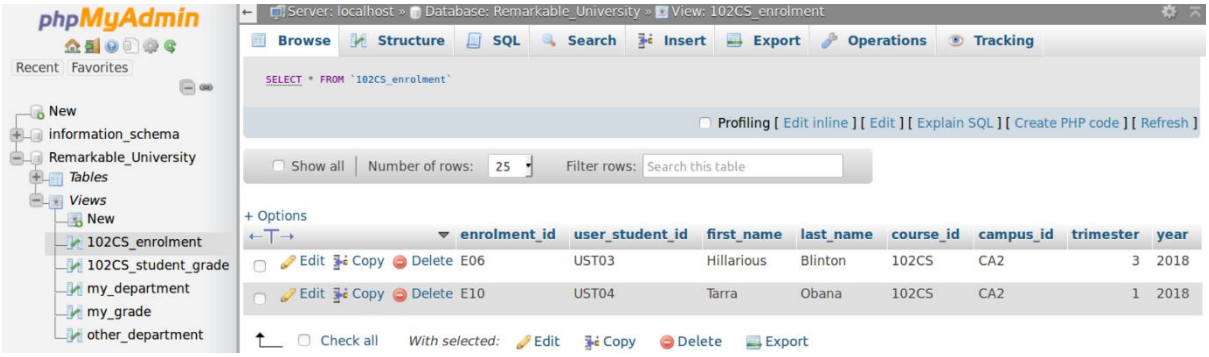
This team took an academic staff user account called “US03” as an example and executed the following SQL command:

Run SQL query/queries on database Remarkable_University:

```
-- Create view that allow the acadmeic staff to see only the students that enrolled in their course

CREATE VIEW Remarkable_University.102CS_enrolment as
    SELECT enrolment.enrolment_id, enrolment.user_student_id, student.first_name, student.last_name, class.course_id, class.campus_id,
class.trimester, enrolment.year FROM Remarkable_University.enrolment, Remarkable_University.class, Remarkable_University.student
    WHERE enrolment.class_id = class.class_id AND enrolment.student_id = student.student_id AND class.course_id = '102CS';


GRANT SELECT ON 102CS_enrolment TO US03 @'%';
```

Result:

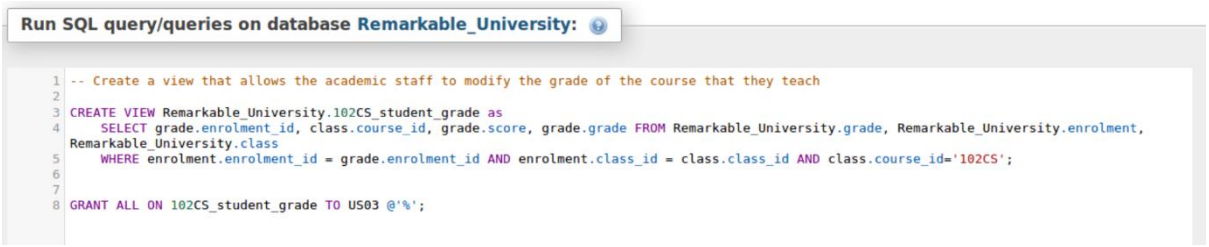
As shown in the screenshot below, the academic staff (US03) can tell who and how many students have enrolled in his class right now. So he could set the teaching plans and give marks to his students according to this view table.



B.3.3. Create a view and Grant Privileges to Academic Staff to Modify Only the Grade of the Course That They Teach

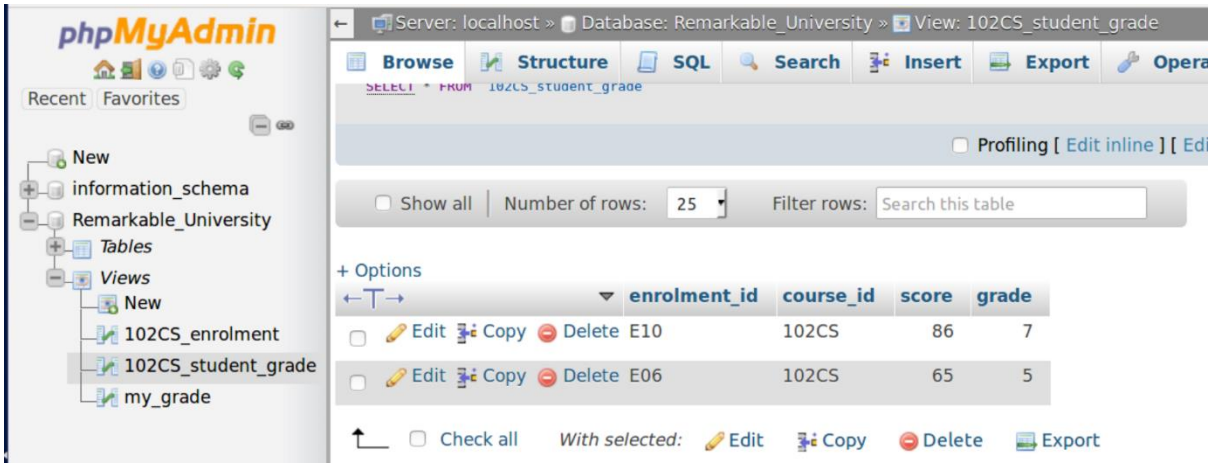
It is critical to allow an academic staff to modify only the grade of the course that he/she is teaching because he/she is responsible for his/her class only. If we allow he/she to be able to see or modify the grade of the other courses that he/she does not teach and response for, it might cause problems within the academic department. To achieve this mechanism, this team members decided to create a view table named “102CS_students_grade” and show only the “enrolment_id”, “course_id”, “score”, and “grade” column to the academic staff by joining 3 tables together which are “enrolment”, “grade”, and “class” table. This is because all the columns were retrieved from these 3 tables. Then the team use the “WHERE” clause to determine the conditions that allow the academic staff to be able to modify only the grades of the course that he/she teach. Lastly, the team granted “ALL” of the privileges of this view table to the academic staff.

This team took an academic staff user account called “US03” as an example and executed the following SQL command:



Result:

As shown in the screenshot below, the academic staff, “US03”, now can see and edit only the grade of the course that he teaches.



B.3.4. Create a view and Grant privileges to the academic staff to See Only the Staffs in Their Department

To let the academic staff to view each other names and information is very important for Remarkable University as it allows the staffs within the department to know and understand each other well. Moreover, they may need to work together in some particular circumstances; therefore, the ability to access the co-worker's information within the department could be helpful. To meet this requirement, this team decided to create a table called "my_department" and shows all the columns in the "staff" table to the academic staff within the department. Then we used the "WHERE" clause to set the condition that allows the academic staff to see each other names and information in their department. Since they can only view this information; therefore, this team granted the "SELECT" Privilege to them.

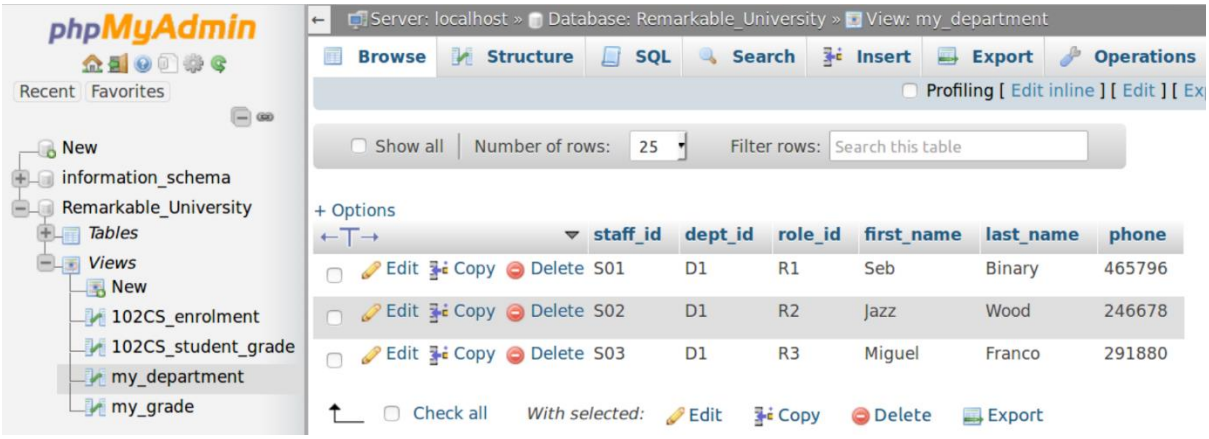
This team took an academic staff user account called "US03" as an example and executed the following SQL command:

Run SQL query/queries on database Remarkable_University:

```
-- Create a view and grant privilege to the academic staff to see only the staff in their department

CREATE VIEW Remarkable_University.my_department as
    SELECT * FROM Remarkable_University.staff
    WHERE staff.dept_id = 'D1';


GRANT SELECT ON my_department TO US03 @'%';

```

Result:

According to the result, we learn that there are 3 people in the academic department, and the user account, "US03", can view only his co-worker information within the department.



B.3.5. Crate a View and Grant Privileges to the Academic Staff to See Other Staff Name and Their Department Only

In this case, this team allows the academic staff to view other staff information at other departments. However, they can view only the “staff_id”, “dept_name”, “first_name”, and “last_name” columns, and the phone numbers are not permitted. This is because the academic staff might need to work with the admin staff or IT staff in some days as well. Therefore, they should know each other name and department. To meet this requirement, this team has created a view table named “other_department” and showed only the above columns to the academic staff by joining 2 tables which are “staff” and “department” tables. Then we used the “WHERE” clause to set the condition that allows the academic staff the see only the staff name and department name of the staff at other departments. Lastly, we granted “SELECT” privilege for the academic staff.

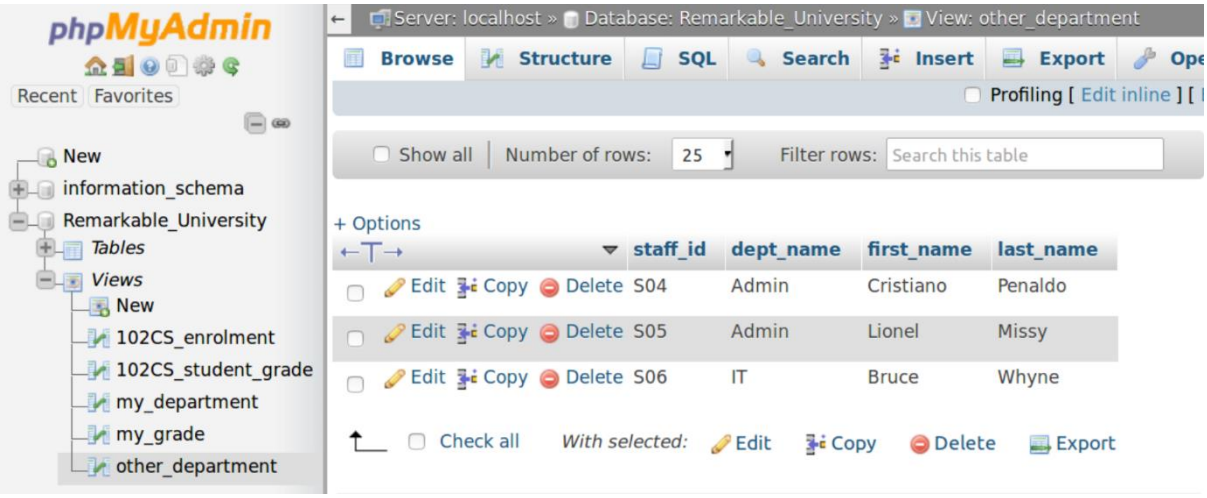
This team took an academic staff user account called “US03” as an example and executed the following SQL command:

Run SQL query/queries on database Remarkable_University:

```
1  -- Create a view and grant a privilege to let the academic staff to view other staff information (staff name & department name only) at other
   deparment
2
3  CREATE VIEW Remarkable_University.other_department as
4      SELECT staff.staff_id, department.dept_name, staff.first_name, staff.last_name FROM Remarkable_University.staff,
   Remarkable_University.department
5      WHERE staff.dept_id = department.dept_id AND staff.dept_id != 'D1';
6
7
8  GRANT ALL ON other_department TO US03 @'%';
9
```

Result:

According to the result, there are 2 different departments within the view table. The user account, “US03”, can now view the staff name and staff department of other staff at different department, yet they cannot see the phone numbers.



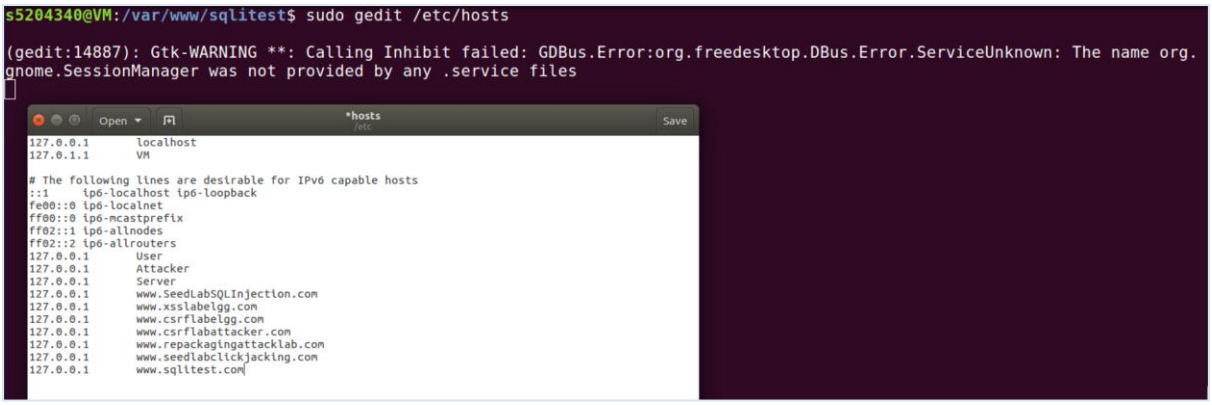
Part C: SQL Injection Test

C.1. Configuration

First of all, in order to test the SQL Injection, this team has copied the files named index.html, safe_main.php, and unsafe_main.php to the directory called /var/www/sqlitest. Then, this team change the permissions of the files to be able to read and executed.

```
[10/03/20]seed@VM:~$ su - s5204340
Password:
s5204340@VM:~$ cd /var/www
s5204340@VM:/var/www$ ls
CSRF  html  RepackagingAttack  SQLInjection  XSS
s5204340@VM:/var/www$ sudo mkdir sqlitest
s5204340@VM:/var/www$ ls
CSRF  html  RepackagingAttack  SQLInjection  sqlitest  XSS
s5204340@VM:/var/www$ cd /home/seed/Downloads/sqlitest/sqlitest
s5204340@VM:/home/seed/Downloads/sqlitest/sqlitest$ ls
index.html  safe_main.php  unsafe_main.php
s5204340@VM:/home/seed/Downloads/sqlitest/sqlitest$ sudo cp -R * /var/www/sqlitest
s5204340@VM:/home/seed/Downloads/sqlitest/sqlitest$ cd /var/www/sqlitest
s5204340@VM:/var/www/sqlitest$ ls -l
total 12
-rwxr-xr-x 1 root root 1005 Oct  3 09:16 index.html
-rwxr-xr-x 1 root root 1742 Oct  3 09:16 safe_main.php
-rwxr-xr-x 1 root root 1247 Oct  3 09:16 unsafe_main.php
s5204340@VM:/var/www/sqlitest$ sudo chmod ugo+r-w+x -R *
s5204340@VM:/var/www/sqlitest$ ls -l
total 12
-r-xr-xr-x 1 root root 1005 Oct  3 09:16 index.html
-r-xr-xr-x 1 root root 1742 Oct  3 09:16 safe_main.php
-r-xr-xr-x 1 root root 1247 Oct  3 09:16 unsafe_main.php
s5204340@VM:/var/www/sqlitest$
```

Next, we use the command “**sudo gedit /etc/hosts**” to append the “hosts” file by adding a new web address: 127.0.0.1 and www.sqlitest.com



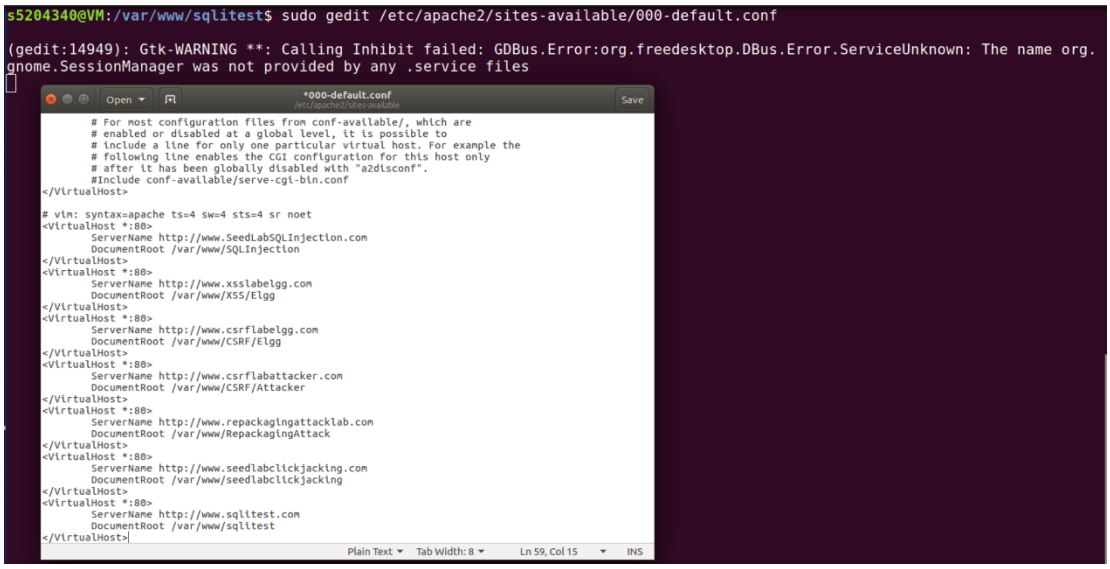
Finally, we used command "**sudo gedit /etc/apache2/sites-available/000-default.conf**" to append the "000-default.conf" file by adding:

<VirtualHost *:80>

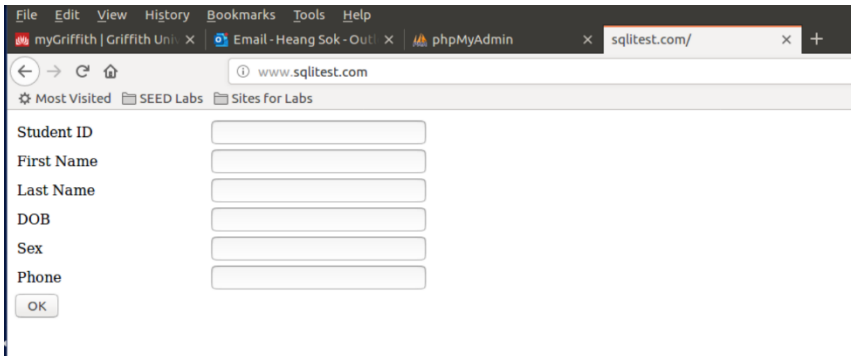
ServerName http://www.sqlitest.com

DocumentRoot /var/www/sqlitest

</VirtualHost>



When the following screen popup in the web browser, it means that we have successfully configured and hosted the interface below.



C.2. SQL Injection Testing Scenarios

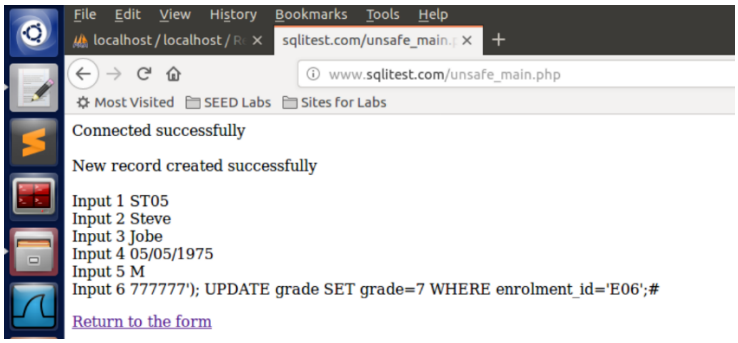
Scenario 1: Mr. Bruce Whyne, database administrator (DBA), had decided to help a student called Hillarious Blinton that has enrolment id “E06” to change his grade from grade 5 to 7.

```
mysql> select * from student;
+------------+------------+-----------+------------+-----+--------+
| student_id | first_name | last_name | DOB        | sex | phone  |
+------------+------------+-----------+------------+-----+--------+
| ST01       | Angela     | Merkal    | 1991-01-01 | F   | 543210 |
| ST02       | Donaldo    | True      | 1992-02-02 | M   | 123456 |
| ST03       | Hillarious | Blinton   | 1993-03-03 | F   | 112233 |
| ST04       | Tarra      | Obana     | 1994-04-04 | M   | 221134 |
+------------+------------+-----------+------------+-----+--------+
4 rows in set (0.00 sec)

mysql> select * from grade;
+--------------+-------+-------+
| enrolment_id | score | grade |
+--------------+-------+-------+
| E01          |    75 |     6 |
| E02          |    80 |     6 |
| E03          |    92 |     7 |
| E04          |    86 |     7 |
| E05          |    71 |     5 |
| E06          |    65 |     5 |
| E07          |    55 |     4 |
| E08          |    66 |     5 |
| E09          |    80 |     6 |
| E10          |    86 |     7 |
+--------------+-------+-------+
10 rows in set (0.00 sec)
```

First, he logs into the www.sqlitest.com using his username= “US06” and password “555”.

```
<html>
<body>

<?php
$servername = "localhost";
$username = "US06";
$password = "555";
$dbname = "Remarkable_University";

// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);

// Check connection
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}
echo "Connected successfully <br> <br>";
```

Next, he pretended to add a new student information to the Remarkable_University database, and he also inserted this command **‘); UPDATE grade SET grade=7 WHERE enrolment_id=’E06’; #**. The syntax **‘);** was used to end the first command, and then he used the **UPDATE** command to change the grade of the student to 7. Finally, he used the # to make the rest of the script become a comment statement.



Result: As shown in the screenshot below, a new student name is added, and Hillarious Blinton's grade is updated to 7 as well.

```
mysql> select * from student;
+------------+------------+-----------+------------+-----+--------+
| student_id | first_name | last_name | DOB        | sex | phone  |
+------------+------------+-----------+------------+-----+--------+
| ST01       | Angela     | Merkal    | 1991-01-01 | F   | 543210 |
| ST02       | Donaldo    | True      | 1992-02-02 | M   | 123456 |
| ST03       | Hillarious | Blinton   | 1993-03-03 | F   | 112233 |
| ST04       | Tarra      | Obana     | 1994-04-04 | M   | 221134 |
| ST05       | Steve      | Jobe      | 1975-05-05 | M   | 777777 |
+------------+------------+-----------+------------+-----+--------+
5 rows in set (0.00 sec)

mysql> select * from grade;
+--------------+-------+-------+
| enrolment_id | score | grade |
+--------------+-------+-------+
| E01          |    75 |     6 |
| E02          |    80 |     6 |
| E03          |    92 |     7 |
| E04          |    86 |     7 |
| E05          |    71 |     5 |
| E06          |    65 |     7 |
| E07          |    55 |     4 |
| E08          |    66 |     5 |
| E09          |    80 |     6 |
| E10          |    86 |     7 |
+--------------+-------+-------+
10 rows in set (0.00 sec)
```

Scenario 2: After the exam, Tarra Obana did not satisfy with his result; therefore, he decided to log in www.sqlitest.com with his student username= "UST04" and password= "889" to drop the grade table from the university database.

```
<html>
<body>

<?php
$servername = "localhost";
$username = "UST04";
$password = "889";
$dbname = "Remarkable_University";

// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);
```

Then, he tried to inject this command **'); DROP TABLE grade; #** to the input number 6. He filled the input 1, 2, and 3 with dash, yet he could not filled the dash in the input 4, 5, and 6 because these inputs were not a string. He used **');** to end the first command, and # to make the rest of the commands become comments. According to the below screenshot, the student used the correct syntax; however, he neither had the privileges to insert information to the student table nor the privileges to drop the grade table. Therefore, this student fail to execute the command and the injection attempt was unsuccessful.

Result: As a result, there is nothing change to both student and grade table after the injection attempt was committed. This result implies that if the attacker does not have the specific privileges over the database tables, ideally, it is impossible for him to successfully implement the injection command.

```
mysql> select * from student where 1=1;
+------------+------------+-----------+------------+-----+--------+
| student_id | first_name | last_name | DOB        | sex | phone  |
+------------+------------+-----------+------------+-----+--------+
| ST01       | Angela     | Merkal    | 1991-01-01 | F   | 543210 |
| ST02       | Donaldo    | True      | 1992-02-02 | M   | 123456 |
| ST03       | Hillarious | Blinton   | 1993-03-03 | F   | 112233 |
| ST04       | Tarra      | Obana     | 1994-04-04 | M   | 221134 |
| ST05       | Steve      | Jobe      | 1975-05-05 | M   | 777777 |
+------------+------------+-----------+------------+-----+--------+
5 rows in set (0.00 sec)

mysql> select * from grade;
+--------------+-------+-------+
| enrolment_id | score | grade |
+--------------+-------+-------+
| E01          |    75 |     6 |
| E02          |    80 |     6 |
| E03          |    92 |     7 |
| E04          |    86 |     7 |
| E05          |    71 |     5 |
| E06          |    65 |     5 |
| E07          |    55 |     4 |
| E08          |    66 |     5 |
| E09          |    80 |     6 |
| E10          |    86 |     7 |
+--------------+-------+-------+
10 rows in set (0.01 sec)
```

This team then tried and used the database administrator (DBA) user ID and password to drop the grade table using the same SQL statement which is **‘); DROP TABLE grade; #.** Surprisingly, the SQL injection attempt was successfully granted by the system this time for DBA as shown in the below screenshot. Then this team assumed that only those who has specify privileges over the grade table can perform SQL injection attempt successfully.

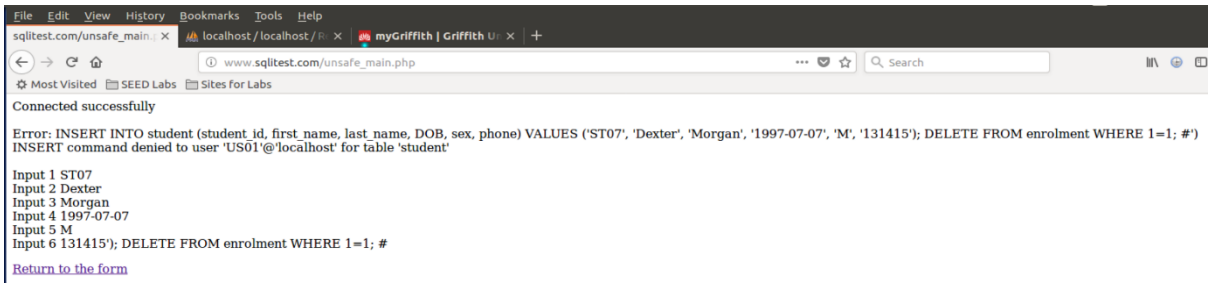
```
mysql> select * from grade;
ERROR 1146 (42S02): Table 'Remarkable_University.grade' doesn't exist
mysql>
```

Scenario 3: An academic staff named Seb Binary wanted to delete all the contents from the enrolment table; therefore, he logged in www.sqlitest.com with his username= “US01” and password= “123”.

```
<html>
<body>

<?php
$servername = "localhost";
$username = "US01";
$password = "123";
$dbname = "Remarkable_Univeristy";

// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);

// Check connection
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}
echo "Connected successfully <br> <br>";
```

Next, he pretended to add a new student called Dexter, and inserted this command **‘);DELETE FROM enrolment WHERE 1=1; #** . Since 1=1 is always true for Boolean algebra, the command was supposed to delete all the information from the enrolment table. However, he did not succeed because he did not have the privilege to insert the

new student information to student table and the operation was end here before it continued to execute the injection command. He used the syntax **');** to end the existing command and # to make the rest of the command become comments.



Result: As shown in the screenshot below, there is nothing was added to the student table, and the enrolment table content remains the same. His case is similar to the student case because they both do not have the specific privileges on the table that they wanted to modify.

```
mysql> select * from student;
+------------+------------+-----------+------------+-----+--------+
| student_id | first_name | last_name | DOB        | sex | phone  |
+------------+------------+-----------+------------+-----+--------+
| ST01       | Angela     | Merkal    | 1991-01-01 | F   | 543210 |
| ST02       | Donaldo    | True      | 1992-02-02 | M   | 123456 |
| ST03       | Hillarious | Blinton   | 1993-03-03 | F   | 112233 |
| ST04       | Tarra      | Obana     | 1994-04-04 | M   | 221134 |
| ST05       | Steve      | Jobe      | 1975-05-05 | M   | 777777 |
+------------+------------+-----------+------------+-----+--------+
5 rows in set (0.01 sec)

mysql> select * from enrolment;
+--------------+-----------------+------------+----------+------+
| enrolment_id | user_student_id | student_id | class_id | year |
+--------------+-----------------+------------+----------+------+
| E01          | UST01           | ST01       | CL01     | 2017 |
| E02          | UST01           | ST01       | CL05     | 2017 |
| E03          | UST01           | ST01       | CL06     | 2017 |
| E04          | UST02           | ST02       | CL03     | 2018 |
| E05          | UST02           | ST02       | CL06     | 2017 |
| E06          | UST03           | ST03       | CL08     | 2018 |
| E07          | UST04           | ST04       | CL03     | 2018 |
| E08          | UST04           | ST04       | CL04     | 2017 |
| E09          | UST04           | ST04       | CL06     | 2017 |
| E10          | UST04           | ST04       | CL07     | 2018 |
+--------------+-----------------+------------+----------+------+
10 rows in set (0.00 sec)
```

Once again, in this scenario, this team also tried and used DBA user ID and password to delete all the contents of the enrolment table with the same SQL injection statement in the scenario3 , and we were successfully implemented it as shown in the screenshot below. Now the data is empty for the enrolment table.

```
mysql> select * from enrolment;
Empty set (0.00 sec)

mysql>
```

Conclusion

In conclusion, it can be inferred from the above three scenario that only the database administrator (DBA) that can successfully implement the injection command since he has all the privileges over the university database. For the academic staff and the student that we have mentioned above, though they used the correct syntax for the

SQL injection attempt, they still failed to achieve their goal because they had limited privileges on the database table.

C.3. Understand the Counter Measurement

C.3.1. Testing 'safe_main.php'

In this stage, this team has edited the index.html file by replacing the 'unsafe_main.php' with 'safe_main.php'. As a result, this team could no longer be able to insert the sql injection commands:

'); UPDATE grade SET grade=7 WHERE enrolment_id='E06'; #

'); DROP TABLE grade; #

');DELETE FROM enrolment WHERE 1=1; #

This time, the system knew that something had been added to the SQL command, and it yielded at the users (DBA, student, academic staff) that they failed to insert. This means that the system is safe from the SQL Injection attack now.

C.3.2. Counter Measurement

The mechanism behind the 'safe_main.php' is it used a prepared statement instead of the normal statement. The different between the normal statement and prepared statement are:

Normal Statement

- Static
- Slow execution time
- Does not prevent SQL Injection attack

Prepared Statement

- Dynamic (using parameter)
- Faster execution time (since they are already precompile)
- Help prevent SQL Injection attack

```
// Prepared statement
$stmt = $conn->prepare("INSERT INTO student (student_id, first_name, last_name, DOB, sex, phone) VALUES (?,?,?,?,?,?)");

if ($stmt) {
    // Bind Parameters
    $stmt->bind_param('ssssssi', $student_id, $first_name, $last_name, $DOB, $sex, $phone);
    $stmt->execute();
    if ($stmt->errno) {
                echo "Failed to insert the record. <br>";
                echo "MySQL Error Code    = " . $stmt->errno . "<br>";
                echo "MySQL Error Message = " . $stmt->error . "<br> <br>";
                $result = false;
```

As shown in the above screenshot, at first, a statement template was created and sent to the Remarkable_University database, and the certain values were held in the

placeholders call parameters which labelled as question marks “(?,?,?,?,?,?)” . Then, the database compiled the statement template and stores the result without executed it. Finally, the application binds the values and the parameters of the statement template and the database then executed the statement many times with different values. By doing so it would reduce the compiling time as the preparation on the query is already done once. In conclusion, the prepared statement is very effective to prevent SQL Injection attack as the parameter are transmitted in the later time using a different protocol.

Another counter measurement besides the prepared statement is grant privilege setting. As discussed in the three scenarios above, if the attackers does not have specific privileges on the database tables that they wants to modify, ideally, it is impossible for them to successfully perform the SQL Injection attack. Therefore, the DBA should carefully check before granting the privileges to other stakeholders.

Part D: Advanced Data Management

In this part, the features of MySQL, NoSQL and graph database will be discussed. Also, a typical kind of data and several security features will be given to show the basic application of each information storage and processing methods.

D.1. Benefits and Drawbacks

Data Type:

As one of the most popular relational database management system in the world, MySQL offers an easy way to for users to communicate with database. Management, storage, and retrieval of data in an SQL database. We can get access to massive data using a query, filtering it by category and property because data is strictly structured. [1] But this also means the data structure is strictly restricted. As the structure of data is decided before it is added, errors could occur as we cannot predict what structure we may subsequently need. And it is rather difficult and expensive to optimize the structure of database. When it comes to NoSQL database, NoSQL systems are built to run across multiple servers and horizontally scale to process vast amounts of information concurrently. NoSQL support more data formats, so it is more flexible and friendly. And the straightforward forms of storage are easier to understand. Also, NoSQL often allows developers to directly change the structure of data. And the time of update can be reduced. As one kind of NoSQL database, graph database concentrates on the relationships between nodes. It shows clear and manageable representation of relationships. Real facts can be stored in a natural way. The structure used is very similar to human thinking, and therefore the links are so clear.

Scalability:

MySQL shows relatively poor performance in scaling. With a high write/read ratio, it offers low scalability. But it could be different in NoSQL: they are outward but not upward. Unlike using larger computers and more CPUs, NoSQL is created in Internet and cloud computing eras.[2] This means in this scale-out architecture, more data can be handled over a large cluster of computers by simply adding more

computers. As for the graph database, it concentrates on the relationships. So, query speed only dependent on the number of concrete relationships, and not on the amount of data. Relationships are stored in the database itself, so they don't need to be calculated in the query.[3] Some allow for very fast execution of complex pattern matching queries. But it could be difficult to scale, as designed as one-tier architecture.

Cost and Management:

MySQL is easy to use. The implementation is relatively simple considering the installment and third-party tools. By learning the language, not many problems will occur. And considering its widespread application, it is possible to implement it with price from free to $10,000. [4] It is less expensive than most other database options on the market. And it is not hard to manage it. Administers can set up users' account privileges and a user can get access to the database together with other users. But if you want to update or backup the database, it could be very expensive. Also, if you want to enhance the scalability of MySQL, it could cost much to get better CPU or better computers. And it could be less reliable when it comes to certain functions (such as references and transactions) than its peers.[1] NoSQL needs less money in administration. With data distribution and auto repair capabilities, simplified data models and fewer tuning and administration requirements, NoSQL need less than traditional RDBMSs.

Source and Supply:

As the most popular used database in the world, MySQL has many supporters and it is mature enough. When it comes to challenges, it is easier to find experts who have come up against the same problems to help you. Also, there are so many options in the market for you to choose from. But for NoSQL database, without many years of development, NoSQL is not as mature as RDBMS. It could be harder to find enough supporting tools and trained developers and consultants. With the same reason, it could be hard to find enough selections to support NoSQL and this may lead to poor maintaining and high cost. Same to graph database, most of them do not have a declarative language and lack the capability to optimize queries in a proper way. And many of them lack native implementations for different platforms.

D.2. Data and Security Features

MySQL:

MySQL could be a good tool to manage user data and e-commerce data. For example, Uber's database management system take ingestion from MySQL because they need to record a lot of real-time data in multiple disparate lines like Uber Freight and Uber Eats. MySQL can serve as a good capture system with data freshness, data quality and data efficiency, it manages vase amounts of data through standardized changelogs ensuring that services have a uniform understanding of the data available. [5] Also, MySQL ensures financial transactions and protects sensitive business information, so it is convenient for customers to make safe transactions online. [6]

Security features:

1. No one is allowed to access to the user table in the database except root account because this contains some sensitive information and information leakage could lead to economic losses and other issues.

2. Do not grant unnecessary privileges and never grant privileges to all hosts. After getting privileges, users can take specific actions to the information stored in the database like select and revoke. These actions may cause some unexpected problems which could lead to damage of the whole system.

3. Use a firewall and put MySQL behind the firewall.

NoSQL Database:

NoSQL is widely used with social network data. Like Facebook and Twitter, they find NoSQL perform better than relational database when it comes to unprecedented transaction volumes, expectations of low-latency access to massive datasets, and nearly perfect service availability while operating in an unreliable environment. [7] With a lot of schema less data in social network, NoSQL is more flexible. Also, with high demand of sharing, reading and watching, NoSQL makes full use of cloud computing and performs better.

Security features:

1.Enable access control and enforce authentication. Authentication requires that all clients and servers provide valid credentials before they can connect to the system. [8]

2. Configure role-based access control. Create a user administrator first, then create additional users. Create a unique user for each person/application that accesses the system.

3. Encrypt communication. Encrypt communication between components and applications.

4. Limit network exposure. Ensure that system runs in a trusted network environment and configure firewall or security groups to control inbound and outbound traffic.

Graph Database:

In a graph database, complex interdependent relationships between nodes can be added and removed in an easy to understand way. A typical example to use graph database is users' buying data. By creating comprehensive customer profiles based on information from search queries, click histories and other components, relationships and data among these profiles can be used to predict the target product, leading to individual product networks to be built up.

Security features:

1. Configuring the user lockout time will help minimize the chance of a successful brute force attack.[9]

2. Use encryption ensures that sensitive data and passwords are not passed over the network in clear text.

3. Enforcing a time-out after a period of inactivity prevents unauthorized access to the graph database.

4. Configure the maximum failed attempts to guard against brute force attacks.

5. A webpage should only be accessed using HTTPS instead of HTTP. It is attached to every HTTPS response.

Reference

[1] Got-it.ai, ‘NoSQL vs SQL: Advantages and Disadvantages – Querychat’, 2020. [Online]. Available: https://www.got-it.ai/solutions/sqlquerychat/sql-help/others/nosql-vs-sql-advantages-and-disadvantages/. [Accessed: 29- Sep- 2020].

[2] Mongodb.com, ‘Advantages of NoSQL Databases’, 2020. [Online]. Available: https://www.mongodb.com/nosql-explained/advantages. [Accessed: 29- Sep- 2020].

[3] Ionos.com, ‘Graph databases explained’, 2019. [Online]. Available: https://www.ionos.com/digitalguide/hosting/technical-matters/graph-database/. [Accessed: 29- Sep- 2020].

[4] Datarealm.com, ‘Five Advantages & Disadvantages Of MySQL’, 2014. [Online]. Available: https://www.datarealm.com/blog/five-advantages-disadvantages-of-mysql/. [Accessed: 29- Sep- 2020].

[5] Eng.uber.com, ‘Uber’s Data Platform in 2019: Transforming Information to Intelligence’, 2019. [Online]. Available: https://eng.uber.com/uber-data-platform-2019/. [Accessed: 29- Sep- 2020].

[6] Dev.to, ‘Tech Stack to Build Marketplaces like Airbnb’, 2020. [Online]. Available: https://dev.to/tyler032/tech-stack-to-build-marketplaces-like-airbnb-1ne1. [Accessed: 29- Sep- 2020].

[7] Papers.ssrn.com, ‘NoSQL Databases as Social Networks Storage Systems’, 2017. [Online]. Available: https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3282559. [Accessed: 29- Sep- 2020].

[8] Docs.mongodb.com, ‘Security Checklist’, 2020. [Online]. Available: https://docs.mongodb.com/manual/administration/security-checklist/. [Accessed: 29- Sep- 2020].

[9] Neo4j.com, ‘Neo4j Security Benchmark’, 2020. [Online]. Available: https://neo4j.com/developer/kb/neo4j-security-benchmark/. [Accessed: 29- Sep- 2020].

Revision History

Date of Change	Contributor	Summary of Change
14-09-2020	Heang Sok_s5204340, Jingdi Lin_s5210032	• First group discussion at Gold Coast campus, Building: G30 • Responsibilities: +Heang Sok worked on part A & part B +Jingdi Lin worked on part C & part D
21-09-2020	Heang Sok_s5204340, Jingdi Lin_s5210032	• Reviewed each other works using Marking Rubric • Individually sent the works to Dr zhe to ask for advice
28-09-2020	Heang Sok_s5204340, Jingdi Lin_s5210032	• Second group discussion at Gold Coast campus, Building: G30 • Future discussion on SQL injection • Future discussion on data backup
05-10-2020	Heang Sok_s5204340, Jingdi Lin_s5210032	• Combine the work together and test the script
10-10-2020	Heang Sok_s5204340, Jingdi Lin_s5210032	• Final check

Appendix A: New Databases Table Design

Department:

dept_id	Dept_name
D1	Academic
D2	Admin
D3	IT

Role:

role_id	role
R1	Professor
R2	Asso. Professor
R3	Lecturer
R4	Enrolment Controller
R5	Course Controller
R6	DBA

Staff:

staff_id	dept_id	role_id	first_name	last_name	phone
S01	D1	R1	Seb	Binary	465796
S02	D1	R2	Jazz	Wood	246678
S03	D1	R3	Miguel	Franco	291880
S04	D2	R4	Cristiano	Penaldo	294472
S05	D2	R5	Lionel	Missy	497124
S06	D3	R6	Bruce	Whyne	364315

Campus:

campus_id	campus_name
CA1	Nathan
CA2	Goldcoast

Course:

course_id	staff_id	course_name
101ICT	S01	Information Management
102ICT	S02	Object Oriented Programming
101STA	S02	Statistics
101CS	S01	Data Analytics
102CS	S03	Information Retrieval

Class:

class_id	course_id	campus_id	trimester
CL01	101ICT	CA2	1
CL02	101ICT	CA2	2
CL03	102ICT	CA1	2
CL04	101STA	CA1	1
CL05	101STA	CA2	1
CL06	101CS	CA2	1
CL07	102CS	CA2	1
CL08	102CS	CA2	3

Student:

student_id	first_name	last_name	DOB	Sex	phone
ST01	Angela	Merkal	1991-01-01	F	543210
ST02	Donaldo	True	1992-02-02	M	123456
ST03	Hillarious	Blinton	1993-03-03	F	112233
ST04	Tarra	Obana	1994-04-04	M	221134

Enrolment:

enrolment_id	user_student_id	student_id	class_id	year
E01	UST01	ST01	CL01	2017
E02	UST01	ST01	CL05	2017
E03	UST01	ST01	CL06	2017
E04	UST02	ST02	CL03	2018
E05	UST02	ST02	CL06	2017
E06	UST03	ST03	CL08	2018
E07	UST04	ST04	CL03	2018
E08	UST04	ST04	CL04	2017
E09	UST04	ST04	CL06	2017
E10	UST04	ST04	CL07	2018

Grade:

enrolment_id	score	grade
E01	75	6
E02	80	6
E03	92	7
E04	86	7
E05	71	5
E06	65	5
E07	55	4
E08	66	5
E09	80	6
E10	86	7

Useraccount_staff:

user_staff_id	staff_id	password
US01	S01	123
US02	S02	520
US03	S03	886
US04	S04	897
US05	S05	777
US06	S06	555

Useraccount_student:

user_student_id	student_id	password
UST01	ST01	888
UST02	ST02	111
UST03	ST03	678
UST04	ST04	889