PART A. Database Implementation

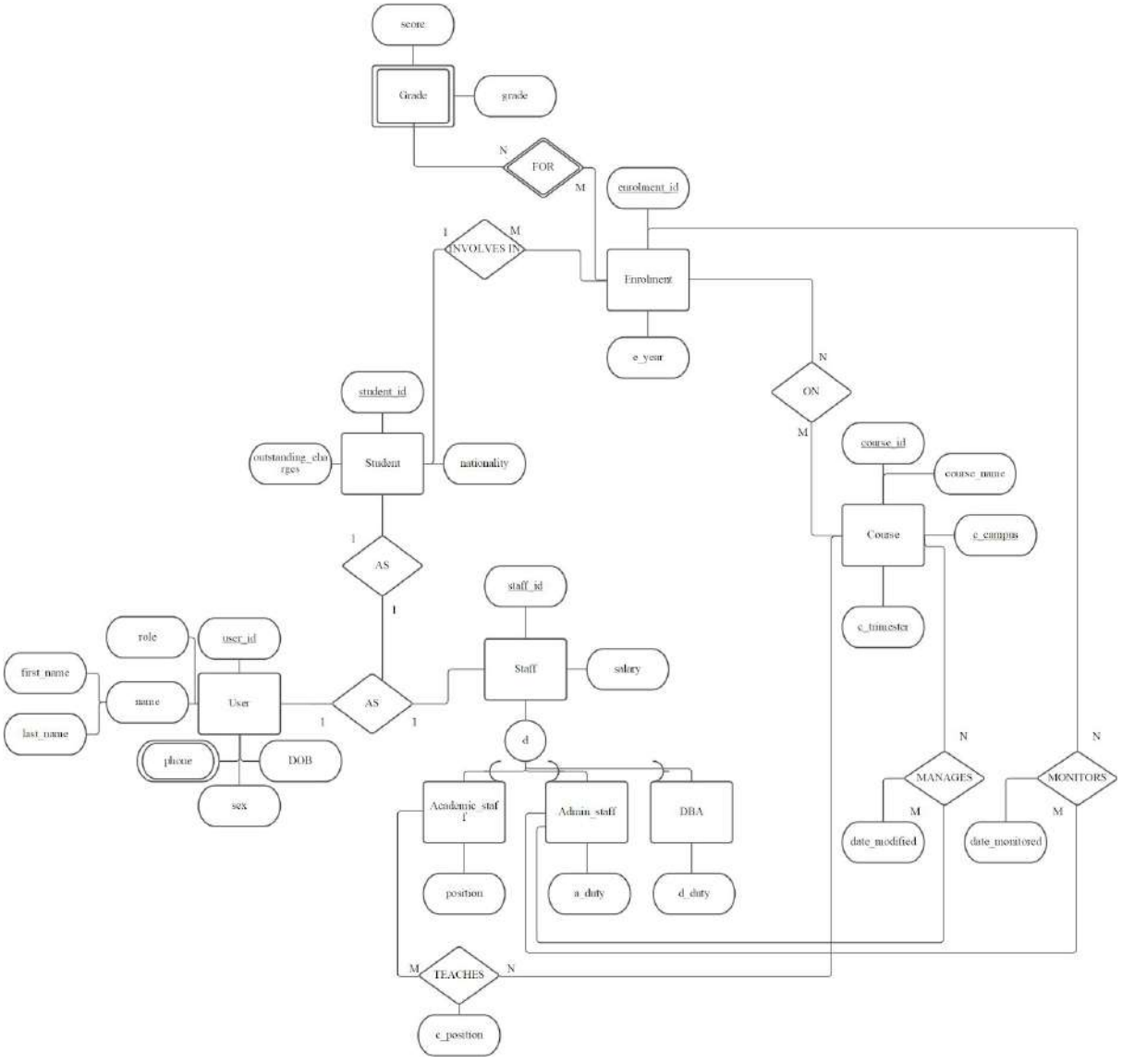
1. Database Design

From the initiated IT Security Plan, it is comprehended that Remarkable University's daily operations rely heavily on the efficiency of their database. Without an efficient database, Remarkable University's organisational performance will be strongly impacted (Bogdan & Mills, 2019). To retain data efficiency, Remarkable University will need to:

- Reduce **data redundancy** and **inconsistency** compounded by to improve the functionality of their database and facilitate quick data processing (Kovalenko, 2019).
- Address **data ambiguity** to increase the level of data readability (Cai & Zhu, 2015).
- Improve **data structure** to broaden the usability of data (Barton & Court, 2013).
- Abiding by **database table constraints** (Pollack, 2016).

Conceptual Design

The sample data provided in the Appendix is used only as a starting point. Note that the sample data does not reflect the table structure in the database and the final database design. The actual structure of the tables will be revised based on the following ERD aim to reduce data redundancy:



Academic, admin and DBA are derived from the staff entity. This is because the staff_id and name are always relevant to a staff regardless of their specification.

It is presumed that every student or staff or individual of both occupations will be a user to the system, and whichever classifications their occupation falls in should only have one user account linked to the individual. Though intrinsically, Student and Staff should have their own unique identifiers (i.e. A student has a student id and Staff a staff id) as primary keys to identify a record in its own table but it is of concern there are individuals who are both a Student as well a Staff (and they would have two unique identifiers), which would result in 2 records having the same personal information that induces data redundancy (i.e. Name, DOB, gender etc.), as can be see below:

Student Table

student_id	first_name	last_name	DOB	sex	phone	nationality	outstanding_ charges
s01	Angela	Merkal	01/01/1991	F	543210	AUS	34876

Above is the new Student table built on top of the given structure, with new attribute ‘nationality’ and ‘outstanding_charges’ to help develop more clarity and transparency of data, most importantly to improve data usability. ‘Nationality’ is helpful when the searcher is only interested in students with a specific nationality. ‘Outstanding_charges’ is useful to track the amount to repay the university, also worth mentioning the searcher can have return only the records on outstanding charges that’s above or below a number they are interested in, which facilitates the search. Like in the Student table, the Staff table below has also been added a new value ‘salary’ for the same reason as with outstanding_charges for students.

Staff Table

staff_id	first_name	last_name	DOB	sex	phone	salary
t01	Angela	Merkal	01/01/1991	F	543210	50000

However, referring to the new table structures for both Staff and Student, data ambiguity is evident for DBA reading the table data. On this account, there is a necessity of a linkage to indicate the 2 records points to the one user. To address, Students and Staff have an ‘AS’ relationship with the User entity. Both of the two tables take the primary key from the User table, which is the user_id from User table, to establish the link needed and tackle data redundancy that would have been induced without the relationship linkage. For better demonstration, see below:

Student Table

student_id	user_id	nationality	outstanding_ charges
s01	u07	AUS	34876

Staff Table

staff_id	user_id	salary
t01	u07	50000

User Table

user_id	first_name	last_name	sex	DOB	role
u07	Angela	Merkal	F	01/01/1991	Student, Academic Staff

With the new changes to the Student, Staff and User tables, it is made easy to recognise when an individual user is under two classifications, and further details on classification of the user can be referred to using their associated user_id. Note an overlapping inheritance was considered for Student and Staff table to inherit User's primary key, but conforming to the purpose of both tables serving for quick search on records corresponding to individual's occupation, it is best to retain student_id for Student and staff_id for Staff as their primary keys and not user_id. Another reason to use a general relationship over inheritance between User and Staff or Student is to ensure staff_id is being used as primary key instead of user_id due to the Staff subclass depending on its superclass's primary key to determine relevant staff records.

Additionally, the User table does not have foreign keys that reference the Student and Staff table as to prevent NULL values that would deem to be present due to the majority of individuals being either a staff or a student, only a minority is on both. Also, looking at the above table, there is one attribute that is to be noticed, that is the ‘role’ column. It was considered that, rather than having all occupations displayed in one column of the record, ‘DBA’ and ‘Student’ and other possible occupations should really be designated its own row. However, the separation was not needed as there are already tables containing records responsible for each occupation. The tables responsible for the sub categories of what classifies as a staff can be found below:

Academic_staff Table

staff_id	position
t02	Professor
t03	Asso. Professor

Admin_staff Table

staff_id	a_duty
t06	Enrolment
t07	Courses

DBA Table

staff_id	d_duty
t01	Data Maintenance

Subsequently, there are two more tables created from the Admin Staff's relationship with managing courses, and monitoring enrolment. The date_modified is added to both the tables below to assist in facilitating search of records based on dates.

Admin_staff_manages Table

staff_id	course_id	c_trimester	c_campus	date_modified
t06	**101ICT**	**2**	**GC**	01/01/2020
t06	**102ICT**	**2**	**NA**	01/02/2020
t07	**103ICT**	**3**	**LOGAN**	03/09/2020

Admin_staff_monitors Table

staff_id	enrolment_id	date_monitored
t06	**e01**	01/01/2020
t06	**e03**	01/01/2020
t07	**e09**	02/05/2020

It is assumed that an Admin Staff can either 'MANAGES' courses OR 'MONITORS' enrolments OR do both. When Admin Staff are monitoring enrolments, they are presumed to monitor at least one enrolment, however not every enrolment would be monitored by one staff. Furthermore, if Admin Staff are managing courses, they are presumed to manage at least one course, however not every course would be managed by one staff. Same rules apply to Admin Staff who does both, full participation is required when managing or monitoring enrolments and courses. For this reason, staff_id and course_id both are needed for the Admin_staff_manages table to search on a record as one staff can manage multiple courses, note by simply using the course_id attribute alone would trigger Violation of PRIMARY KEY constraint. Same applies to staff_id and enrolment_id in the Admin_staff_monitors table.

Subsequently, the table, Enrolment, accepts primary keys in both Student and Course tables as its foreign key in order to help trace records in Student and Course tables that's relevant to the Enrolment. An enrolment must have at least one course in it, and one specific course might not be in one's enrolment. One enrolment may be on more than one course, and one course can be admitted to more than one enrolment.

There are a few changes that have been done to the Enrolment table, with the aim to remove data redundancy and improve usability. The table structure that had been provided was the follows:

Enrolment Table

enrolment_id	course_id	student_id	student_name	year	trimester	campus
e01	101ICT	s01	Angela Merkal	2017	1	GC

And has been updated to adapt to the following table structure.

enrolment_id	course_id	student_id	year	trimester	campus
e01	101ICT	s01	2017	1	GC

Nothing has been changed except that student_name was removed. Reason being simply it is good practice to split the full name into first name and last name, as it will assist in facilitating the search on students, for example searching for students who have the same last name. Furthermore, the student id in the Enrolment table will give user_id and user_id in the User table, which is able to provide the student name, in fact all of the associated personal information relating to the student can be supplied, as demonstrated below.

Student Table

student_id	user_id	nationality	Outstanding_ charges
s01	u07	AUS	34876

User Table

user_id	first_name	last_name	sex	DOB	role
u07	Angela	Merkal	F	01/01/1991	Student, Academic Staff

To summarise the above student table and its 'INVOLVES IN' relationship with the associated enrolment table. An specific enrolment definitely belongs to a student, but a student does not necessarily have to have any enrolments. Many enrolments may belong to one student, but one particular enrolment can belong to one student only.

Besides the changes made for the Enrolment table structure, the structure of the Course table has also been altered, to improve usability and readability of the data in the table. The Course table now accepts three attributes, specifically course_id, c_trimester as well as c_campus to form a primary key which can uniquely identify a course record in its table. The intention for the Course table to be designed this way is so instead of having combined values for all trimesters available for the course in one row like shown below:

Course Table

course_id	c_trimester	course_name	c_campus	convenor
101ICT	1,2	Information Management	GC	Seb Binary

The trimesters can be separated into different rows to help distinguish the same course offered in different times, indicated below:

Course Table

course_id	c_trimester	course_name	c_campus	convenor
101ICT	**1**	Information Management	**GC**	Seb Binary
101ICT	**2**	Information Management	**GC**	Seb Binary

Now more refined a search can be conducted on the table by specifying a specific trimester a course offers in, as well the campus the course offers in. Note this will not be achieved by simply using the course_id or course_id and c_trimester attribute alone, as it would trigger Violation of PRIMARY KEY constraint, hence why need c_trimester and c_campus to support course_id to form a primary key.

Moreover, the convenor column was removed from the Course table, instead there is a c_position attribute that adds on top of what has been derived into a new table from the relationship for the teaching of a course by an academic staff. The c_position is added to the table Academic_staff_teaches to assist in not only allowing accessibility to the convenors' names but also other information that belong to convenor of a specific course. New tables can be seen below:

Course Table

course_id	c_trimester	course_name	c_campus
101ICT	**1**	Information Management	**GC**
101ICT	**2**	Information Management	**GC**

Academic_staff_teaches Table

staff_id	course_id	c_trimester	c_campus	c_position
t01	**101ICT**	**1**	**GC**	Convenor
t09	**101ICT**	**2**	**GC**	Tutor

The above table illustrates the relationship for Academic Staff 'TEACHES' a course. Every Academic Staff is presumed to teach at least one course, and every course is taught by at least a staff. One staff member can teach more than one course, and one course can be taught by more than one staff member. A many-to-many relationship of 'TEACHES' is established, and both the primary key of Academic Staff table's as well as of Course is needed in a new table to associate the course with its teaching staff.

Finally, Grade is a weak entity as can be seen from the given table structure:

Grade

enrolment_id	score	grade
e01	75	6

The Grade table does not have a primary key itself that can help identify a record and needs to rely on other table's, namely the Enrolment table and its primary key, which is the enrolment_id, to recognise a grade record. The Enrolment table has its foreign keys that points directly to the student and course it is governing, thereby uniquely identifying a grade record that belongs to that student on one specific course. Additionally, an enrolment doesn't necessarily have a grade associated(i.e. When courses are all dropped). However, when a grade is present it means an enrolment is valid. One grade may belong to more than one enrolment, and one enrolment can consist of more than one grade.

Logical Design

User (user_id, first_name, last_name, sex, DOB, role)

PK: primary key (user_id)

User_PhoneNumbers(user_id, phone)

PK: primary key (user_id)

FK: user_id references User(user_id)

Student(student_id, user_id, nationality, outstanding_charges)

PK: primary key (student_id)

FK: user_id references User(user_id)

Staff(staff_id, user_id, salary)

PK: primary key (staff_id)

FK: user_id references User(user_id)

Academic_staff(staff_id, position)

PK: primary key (staff_id)

FK: staff_id references Staff(staff_id)

Admin_staff(staff_id, a_duty)

PK: primary key (staff_id)

FK: staff_id references Staff(staff_id)

DBA(staff_id, d_duty)

PK: primary key (staff_id)

FK: staff_id references Staff(staff_id)

Course(course_id, c_trimester, course_name, c_campus)

PK: primary key (course_id, c_trimester, c_campus)

Academic_staff_teaches(staff_id, course_id, c_trimester, c_campus, c_position)

PK: primary key (staff_id, course_id, c_trimester, c_campus)

FK: course_id, c_trimester, c_campus references Course(course_id, c_trimester, c_campus)

staff_id references Staff(staff_id)

Admin_staff_manages(staff_id, course_id, c_trimester, c_campus, date_modified)

PK: primary key (staff_id, course_id, c_trimester, c_campus)

FK: course_id, c_trimester, c_campus references Course(course_id, c_trimester, c_campus)

staff_id references Staff(staff_id)

Admin_staff_monitors(staff_id, enrolment_id, date_monitored)

PK: primary key (staff_id, enrolment_id)

FK: enrolment_id references Enrolment(enrolment_id)

staff_id references Staff(staff_id)

Enrolment(enrolment_id, course_id, student_id, e_year, c_trimester, c_campus)

PK: primary key (enrolment_id)

FK: course_id, c_trimester, c_campus references Course(course_id, c_trimester, c_campus)

student_id references Student(student_id)

Grade(enrolment_id, score, grade)

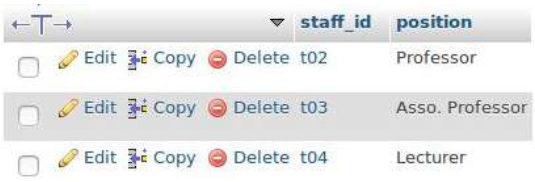
PK: primary key (enrolment_id)

FK: enrolment_id references Enrolment(enrolment_id)

2. SQL Implementation

Considering the amount of SQL statements and comments in the assignment2.sql file, it will not be included in this section. The SQL statements and comments can be found in the attached assignment2.sql file instead.

Academic_staff



	staff_id	position
Edit Copy Delete	t02	Professor
Edit Copy Delete	t03	Asso. Professor
Edit Copy Delete	t04	Lecturer

#	Name	Type	Collation	Attributes	Null	Default	Extra	Action
1	staff_id	varchar(10)	latin1_swedish_ci		No	None		Change Drop Primary Unique Index Spatial Fulltext Distinct values More
2	position	varchar(20)	latin1_swedish_ci		No	None		Change Drop Primary Unique Index Spatial Fulltext Distinct values More

Academic_staff_teaches

	staff_id	course_id	c_trimester	c_campus	c_position
Edit Copy Delete	t02	101CS	1	GC	Convenor
Edit Copy Delete	t02	101ICT	1	GC	Convenor
Edit Copy Delete	t02	101ICT	2	GC	Convenor
Edit Copy Delete	t03	101STA	1	GC	Tutor
Edit Copy Delete	t03	102ICT	2	NA	Convenor
Edit Copy Delete	t04	102CS	3	GC	Convenor
Edit Copy Delete	t04	103ICT	3	LOGAN	Tutor

#	Name	Type	Collation	Attributes	Null	Default	Extra	Action
1	staff_id	varchar(10)	latin1_swedish_ci		No	None		Change Drop Primary Unique Index Spatial Fulltext Distinct values More
2	course_id	varchar(10)	latin1_swedish_ci		No	None		Change Drop Primary Unique Index Spatial Fulltext Distinct values More
3	c_trimester	char(1)	latin1_swedish_ci		No	None		Change Drop Primary Unique Index Spatial Fulltext Distinct values More
4	c_campus	varchar(10)	latin1_swedish_ci		No	None		Change Drop Primary Unique Index Spatial Fulltext Distinct values More
5	c_position	varchar(10)	latin1_swedish_ci		No	None		Change Drop Primary Unique Index Spatial Fulltext Distinct values More

Admin_staff

	staff_id	a_duty
Edit Copy Delete	t05	Enrolment
Edit Copy Delete	t06	Enrolment
Edit Copy Delete	t07	Courses
Edit Copy Delete	t09	Courses

#	Name	Type	Collation	Attributes	Null	Default	Extra	Action
1	staff_id	varchar(10)	latin1_swedish_ci		No	None		Change Drop Primary Unique Index Spatial Fulltext Distinct values More
2	a_duty	varchar(10)	latin1_swedish_ci		No	None		Change Drop Primary Unique Index Spatial Fulltext Distinct values More

Admin_staff_manages

+ Options

	staff_id	course_id	c_trimester	c_campus	date_modified
Edit Copy Delete	t07	101ICT	2	GC	2020-01-01
Edit Copy Delete	t07	102ICT	2	NA	2020-01-02
Edit Copy Delete	t07	103ICT	3	LOGAN	2020-03-09
Edit Copy Delete	t09	101CS	1	GC	2020-03-07
Edit Copy Delete	t09	101STA	1	GC	2020-09-02
Edit Copy Delete	t09	102CS	3	GC	2020-04-01

#	Name	Type	Collation	Attributes	Null	Default	Extra	Action
1	staff_id	varchar(10)	latin1_swedish_ci		No	None		Change Drop Primary Unique Index Spatial Fulltext Distinct values More
2	course_id	varchar(10)	latin1_swedish_ci		No	None		Change Drop Primary Unique Index Spatial Fulltext Distinct values More
3	c_trimester	char(1)	latin1_swedish_ci		No	None		Change Drop Primary Unique Index Spatial Fulltext Distinct values More
4	c_campus	varchar(10)	latin1_swedish_ci		No	None		Change Drop Primary Unique Index Spatial Fulltext Distinct values More
5	date_modified	date			No	None		Change Drop Primary Unique Index Spatial Fulltext Distinct values More

Admin_staff_monitors

	staff_id	enrolment_id	date_monitored
Edit Copy Delete	t05	e02	2020-07-09
Edit Copy Delete	t05	e04	2020-08-08
Edit Copy Delete	t05	e11	2020-06-06
Edit Copy Delete	t06	e01	2020-01-01
Edit Copy Delete	t06	e03	2020-01-01
Edit Copy Delete	t06	e09	2020-02-05

#	Name	Type	Collation	Attributes	Null	Default	Extra	Action
1	staff_id	varchar(10)	latin1_swedish_ci		No	None		Change Drop Primary Unique Index Spatial Fulltext Distinct values More
2	enrolment_id	varchar(10)	latin1_swedish_ci		No	None		Change Drop Primary Unique Index Spatial Fulltext Distinct values More
3	date_monitored	date			Yes	NULL		Change Drop Primary Unique Index Spatial Fulltext Distinct values More

Course

	course_id	c_trimester	course_name	c_campus
Edit Copy Delete	101CS	1	Data Analytics	GC
Edit Copy Delete	101ICT	1	Information Management	GC
Edit Copy Delete	101ICT	2	Information Management	GC
Edit Copy Delete	101STA	1	Statistics	GC
Edit Copy Delete	101STA	1	Statistics	NA
Edit Copy Delete	102CS	1	Information Retrieval	GC
Edit Copy Delete	102CS	3	Information Retrieval	GC
Edit Copy Delete	102ICT	2	Object Oriented Programming	NA
Edit Copy Delete	103ICT	3	Programming Principles	LOGAN

#	Name	Type	Collation	Attributes	Null	Default	Extra	Action
1	course_id	varchar(10)	latin1_swedish_ci		No	None		Change Drop Primary Unique Index Spatial Fulltext Distinct values More
2	c_trimester	char(1)	latin1_swedish_ci		No	None		Change Drop Primary Unique Index Spatial Fulltext Distinct values More
3	course_name	varchar(30)	latin1_swedish_ci		No	None		Change Drop Primary Unique Index Spatial Fulltext Distinct values More
4	c_campus	varchar(10)	latin1_swedish_ci		No	None		Change Drop Primary Unique Index Spatial Fulltext Distinct values More

DBA

	staff_id	d_duty
Edit Copy Delete	t08	Data Maintenance

#	Name	Type	Collation	Attributes	Null	Default	Extra	Action
1	staff_id	varchar(10)	latin1_swedish_ci		No	None		Change Drop Primary Unique Index Spatial Fulltext Distinct values More
2	d_duty	varchar(20)	latin1_swedish_ci		No	None		Change Drop Primary Unique Index Spatial Fulltext Distinct values More

Enrolment

	enrolment_id	course_id	c_trimester	student_id	e_year	c_campus
Edit Copy Delete	e01	101ICT	1	s01	2017	GC
Edit Copy Delete	e02	101STA	1	s01	2017	GC
Edit Copy Delete	e03	101CS	1	s01	2017	GC
Edit Copy Delete	e04	102ICT	2	s02	2018	NA
Edit Copy Delete	e05	101CS	1	s02	2017	GC
Edit Copy Delete	e06	102CS	3	s03	2018	GC
Edit Copy Delete	e07	102ICT	2	s04	2018	NA
Edit Copy Delete	e08	101STA	1	s04	2017	NA
Edit Copy Delete	e09	101CS	1	s04	2017	GC
Edit Copy Delete	e10	102CS	1	s04	2018	GC
Edit Copy Delete	e11	103ICT	3	s01	2019	LOGAN

#	Name	Type	Collation	Attributes	Null	Default	Extra	Action
1	**enrolment_id**	varchar(10)	latin1_swedish_ci		No	*None*		Change Drop Primary Unique Index Spatial Fulltext Distinct values More
2	**course_id**	varchar(10)	latin1_swedish_ci		Yes	*NULL*		Change Drop Primary Unique Index Spatial Fulltext Distinct values More
3	**c_trimester**	char(1)	latin1_swedish_ci		Yes	*NULL*		Change Drop Primary Unique Index Spatial Fulltext Distinct values More
4	**student_id**	varchar(10)	latin1_swedish_ci		Yes	*NULL*		Change Drop Primary Unique Index Spatial Fulltext Distinct values More
5	**e_year**	char(4)	latin1_swedish_ci		No	*None*		Change Drop Primary Unique Index Spatial Fulltext Distinct values More
6	**c_campus**	varchar(10)	latin1_swedish_ci		No	*None*		Change Drop Primary Unique Index Spatial Fulltext Distinct values More

Grade

←T→	enrolment_id	score	grade
Edit Copy Delete	e01	75.00	6
Edit Copy Delete	e02	80.00	6
Edit Copy Delete	e03	92.00	7
Edit Copy Delete	e04	86.00	7
Edit Copy Delete	e05	71.00	5
Edit Copy Delete	e06	65.00	5
Edit Copy Delete	e07	55.00	4
Edit Copy Delete	e08	66.00	5
Edit Copy Delete	e09	80.00	6
Edit Copy Delete	e10	86.00	7

#	Name	Type	Collation	Attributes	Null	Default	Extra	Action
1	**enrolment_id**	varchar(10)	latin1_swedish_ci		No	*None*		Change Drop Primary Unique Index Spatial Fulltext Distinct values More
2	**score**	double(5,2)			Yes	*NULL*		Change Drop Primary Unique Index Spatial Fulltext Distinct values More
3	**grade**	tinyint(1)			Yes	*NULL*		Change Drop Primary Unique Index Spatial Fulltext Distinct values More

Staff

	staff_id	user_id	salary
Edit Copy Delete	t01	u07	50000.00
Edit Copy Delete	t02	u04	98000.00
Edit Copy Delete	t03	u05	73000.00
Edit Copy Delete	t04	u06	60000.00
Edit Copy Delete	t05	u11	83000.00
Edit Copy Delete	t06	u02	55000.00
Edit Copy Delete	t07	u03	62000.00
Edit Copy Delete	t08	u01	102000.00
Edit Copy Delete	t09	u12	94000.00

#	Name	Type	Collation	Attributes	Null	Default	Extra	Action
1	staff_id	varchar(10)	latin1_swedish_ci		No	None		Change Drop Primary Unique Index Spatial Fulltext Distinct values More
2	user_id	varchar(10)	latin1_swedish_ci		Yes	NULL		Change Drop Primary Unique Index Spatial Fulltext Distinct values More
3	salary	double(8,2)			No	None		Change Drop Primary Unique Index Spatial Fulltext Distinct values More

Student

	student_id	user_id	nationality	outstanding_charges
Edit Copy Delete	s01	u07	AUS	34876.00
Edit Copy Delete	s02	u08	US	189.00
Edit Copy Delete	s03	u09	UK	2345.00
Edit Copy Delete	s04	u10	ES	378.00

Check all *With selected:* Edit Copy Delete Export

Show all | Number of rows: 25 Filter rows: Search this table

#	Name	Type	Collation	Attributes	Null	Default	Extra	Action
1	student_id	varchar(10)	latin1_swedish_ci		No	None		Change Drop Primary Unique Index Spatial Fulltext Distinct values More
2	user_id	varchar(10)	latin1_swedish_ci		Yes	NULL		Change Drop Primary Unique Index Spatial Fulltext Distinct values More
3	nationality	varchar(10)	latin1_swedish_ci		Yes	NULL		Change Drop Primary Unique Index Spatial Fulltext Distinct values More
4	outstanding_charges	double(8,2)			No	None		Change Drop Primary Unique Index Spatial Fulltext Distinct values More

User

	user_id	first_name	last_name	sex	DOB	role
Edit Copy Delete	u01	Bruce	Whyne	M	1980-01-01	DBA
Edit Copy Delete	u02	Cristiano	Penaldo	M	1983-06-01	Admin Staff
Edit Copy Delete	u03	Lionel	Missy	M	1990-05-07	Admin Staff
Edit Copy Delete	u04	Seb	Binary	M	1980-02-07	Academic Staff
Edit Copy Delete	u05	Jazz	Wood	M	1984-09-08	Academic Staff
Edit Copy Delete	u06	Miguel	Franco	M	1985-03-01	Academic Staff
Edit Copy Delete	u07	Angela	Merkal	F	1991-01-01	Student, Academic Staff
Edit Copy Delete	u08	Donaldo	True	M	1992-02-02	Student
Edit Copy Delete	u09	Hillarious	Blinton	F	1993-03-03	Student
Edit Copy Delete	u10	Tarra	Obana	M	1994-04-04	Student
Edit Copy Delete	u11	Liz	Cappilo	F	1973-06-01	Admin Staff
Edit Copy Delete	u12	Carlo	Lok	M	1992-09-07	Admin Staff

#	Name	Type	Collation	Attributes	Null	Default	Extra	Action
1	user_id	varchar(10)	latin1_swedish_ci		No	None		Change Drop Primary Unique Index Spatial Fulltext Distinct values More
2	first_name	varchar(20)	latin1_swedish_ci		No	None		Change Drop Primary Unique Index Spatial Fulltext Distinct values More
3	last_name	varchar(20)	latin1_swedish_ci		No	None		Change Drop Primary Unique Index Spatial Fulltext Distinct values More
4	sex	char(1)	latin1_swedish_ci		Yes	NULL		Change Drop Primary Unique Index Spatial Fulltext Distinct values More
5	DOB	date			Yes	NULL		Change Drop Primary Unique Index Spatial Fulltext Distinct values More
6	role	varchar(100)	latin1_swedish_ci		No	None		Change Drop Primary Unique Index Spatial Fulltext Distinct values More

User_PhoneNumbers

	user_id	phone
Edit Copy Delete	u01	433820
Edit Copy Delete	u02	543210
Edit Copy Delete	u03	398478
Edit Copy Delete	u04	219283
Edit Copy Delete	u05	283772
Edit Copy Delete	u06	102938
Edit Copy Delete	u07	543210
Edit Copy Delete	u08	123456
Edit Copy Delete	u09	112233
Edit Copy Delete	u10	221134
Edit Copy Delete	u11	182923
Edit Copy Delete	u12	788888

#	Name	Type	Collation	Attributes	Null	Default	Extra	Action
1	user_id	varchar(10)	latin1_swedish_ci		No	None		Change Drop Primary Unique Index Spatial Fulltext Distinct values More
2	phone	char(6)	latin1_swedish_ci		No	None		Change Drop Primary Unique Index Spatial Fulltext Distinct values More

3. Database Backup

Due to the high cost of frequent backups, Remarkable University can only afford occasional backup of the whole database. However, the Student Table, Enrolment Table, Course Table, Grade Table, Staff Table, and User Table of the database will need to be frequently backuped despite the expensive cost, as these tables are closely linked to Remarkable University's daily operations and organisational profit.

Any organisation (except for non-profit organisations) including Remarkable University will have the ultimate goal of generating profit. As Remarkable University is an educational organisation, it is assumed that their key financial contributors are students, and their main value-creating activity is students paying tuition for their enrolled courses. With the ultimate goal of generating profit, no organisation will leave potential risks in their value-creating activity. Hence, it is essential to frequently backup the **Student Table**, **Enrolment Table** and **Course Table** related to this main value-creating activity to ensure that students can enrol in courses and can pay tuition for their enrolled courses. With value-creating activity linked to these database tables, frequent backups become a necessity.

Considering that some students might want to modify their failed grades to a pass to dodge the educational penalty and the financial obligation in studying the course again. **Grade Table** needs to be backed up frequently to prevent this from happening, as if otherwise, Remarkable University will not only lose profit but will also breach their educational policy. With the Student Table frequently backed up can ensure that if one of the backups is corrupted, then the other backups will allow restorability.

All financial-related tables such as **Staff Table** needs to be backed up frequently to ensure that the financial-related attributes such as the salary attribute of the Staff Table is kept to date and the correct data is maintained. With the Staff Table frequently backed up ensures that any skeptical activity such as staff unauthorisedly modify the salary attributes can be captured and reverted through the backup retrieval of an older version.

The **User Table** needs to have extra attention for two reasons. Firstly, it records personal information about students and staff, which attracts the interests of hackers. Having frequent backups in hand meaning that if the current User Table is corrupted, Remarkable University will have the ability to restore the User Table to its latest state. Secondly, new users, mostly being students are frequently added to the User Table of Remarkable University's database. Since the User Table changes and updates frequently, it will be essential to backup the User Table frequently to ensure that no data is lost or misrecorded.

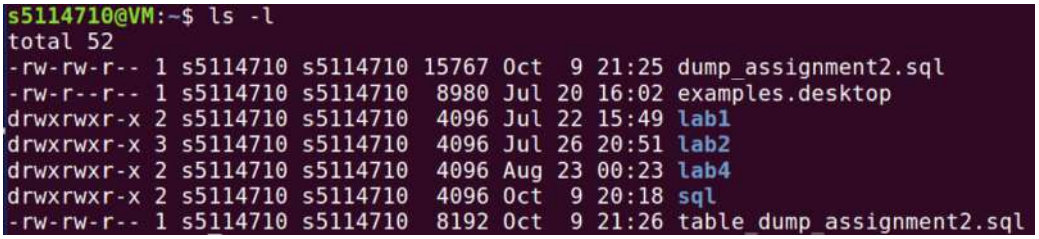
Mysqldump commands are used to implement the backup and recovery for the whole assignment2 database and those six tables requiring more frequent backup.

Backup (done in UNIX prompt)**:**

Command	**Explanation**
`mysql> use assignment2;`	Remarkable University has a single database named assignment2 that has to be occasionally backuped.

`mysql> show tables;`
`+-----------------------+`
`| Tables_in_assignment2 |`
`+-----------------------+`
`| Academic_staff |`
`| Academic_staff_teaches |`
`| Admin_Staff |`
`| Admin_staff_manages |`
`| Admin_staff_monitors |`
`| Course |`
`| DBA |`
`| Enrolment |`
`| Grade |`
`| Staff |`
`| Student |`
`| User |`
`| User_PhoneNumbers |`
`+-----------------------+`
	The assignment2 database has six tables (indicated in red underlines): Student Table, Enrolment Table, Course Table, Grade Table, Staff Table, and User Table that needs to be frequently backuped.

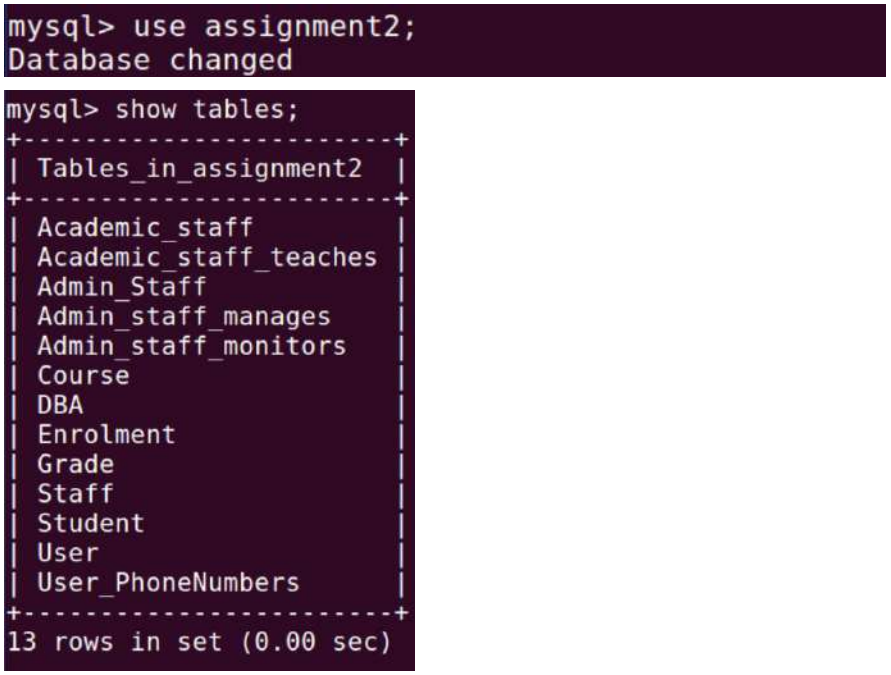
`mysqldump -u root -pseedubuntu assignment2 > dump_assignment2.sql`	This command outputs the backup of single database assignment2 to a file dump_assignment2.sql.
`mysqldump -u root -pseedubuntu assignment2 Student Enrolment Course Grade Staff User > table_dump_assignment2.sql`	This command outputs the backup of the six tables in the database assignment2 to a file dump_assignment2.sql.

	The *ls-l* command is used to check whether the two files: dump_assignment2.sql and table_dump_assignment2.sql are successfully created.

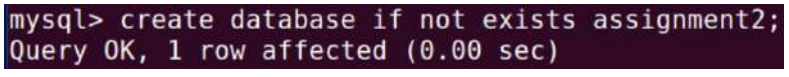
Recovery (needs to be connected to MySQL console through the Terminal)

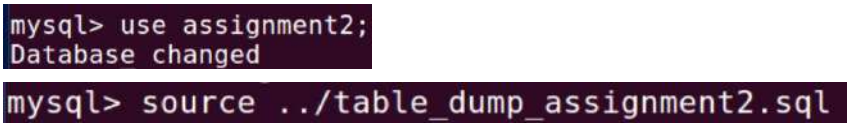
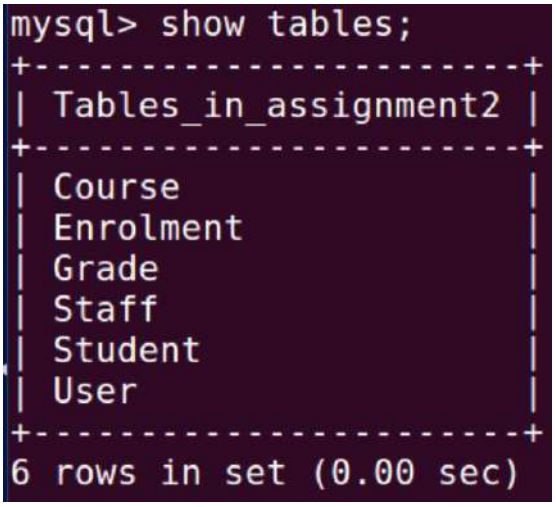
Recover the single assignment2 database using the dump_assignment2.sql file:

Command	Explanation
`mysql -u root -pseedubuntu`	This command is used to login to the MySQL console.
`mysql> drop database assignment2;` `Query OK, 13 rows affected (0.14 sec)`	Recovery cannot be done when the assignment2 database exists. Hence, it is essential to drop the assignment2 database using the *drop database* command.
`mysql> source ../dump_assignment2.sql`	Then, recover the single assignment2 database using the backup file dump_assignment2.sql through the *source* commands. To backup for the assignment2 database, the *create database if not exists* command does not need to be used , as this command is already in the dump_assignment2.sql file.

	When the backup source of dump_assignment2.sql is used, all tables within the assignment2 database will be recovered. To prove that the whole single assignment2 database is recovered, the *show tables* command is used.

Recover the six tables in the assignment2 database using table_dump_assignment2.sql file:

Command	**Explanation**
	Drop the assignment2 database before recovering using table_dump_assignment2.sql file.
	Different from dump_assignment2.sql file, table_dump_assignment2.sql file does not have the *create database if not exists* command written. Hence, it is essential to type this command in the

	MySQL console, otherwise the six tables cannot be recovered as the assignment2 database does not exist.
	After creating the assignment2 database, backup for the six tables can be sourced into the assignment2 database using the *source* command.
	Different from the dump_assignment2.sql file, when the table_dump_assignment2 file is used, only the six tables (Course, Enrolment, Grade, Staff, STudent and User) in the assignment2 database will be recovered. To prove that only the six tables are recovered, the *how tables* command is used.

PART B. Users and Privileges

1. Create Users

SQL STATEMENTS FOR CREATING USERS

```
CREATE USER 'u01'@'%' IDENTIFIED BY 'ThisIsBruce1';
CREATE USER 'u02'@'%' IDENTIFIED BY 'CristianoHere1)';
CREATE USER 'u03'@'%' IDENTIFIED BY 'LionelPassword123';
CREATE USER 'u04'@'%' IDENTIFIED BY 'HiSeb321';
CREATE USER 'u05'@'%' IDENTIFIED BY 'WJazz312241';
CREATE USER 'u06'@'%' IDENTIFIED BY 'MF13241';
CREATE USER 'u07'@'%' IDENTIFIED BY 'AngMer010';
CREATE USER 'u08'@'%' IDENTIFIED BY '0888DTru';
CREATE USER 'u09'@'%' IDENTIFIED BY 'HBlinton1';
CREATE USER 'u10'@'%' IDENTIFIED BY 'ObanaPW1!';
CREATE USER 'u11'@'%' IDENTIFIED BY 'LizMe8';
CREATE USER 'u12'@'%' IDENTIFIED BY 'CarloZ9888';
```

STAFF NAME/ STUDENT NAME	USER NAME	HOST NAME	PASSWORD
Bruce Whyne	u01	%	ThisIsBruce1
Cristiano Penaldo	u02	%	CristianoHere1)
Lionel Missy	u03	%	LionelPassword123
Seb Binary	u04	%	HiSeb321
Jazz Wood	u05	%	WJazz312241
Miguel Franco	u06	%	MF13241
Angela Merkal	u07	%	AngMer010
Donaldo True	u08	%	0888DTru
Hillarious Blinton	u09	%	HBlinton1
Tarra Obana	u10	%	ObanaPW1!
Liz Cappilo	u11	%	LizMe8
Carlo Lok	u12	%	CarloZ9888

2. Assign Table-level Privileges

The tables associate user privileges are summarised in the following access matrix (Wang, 2020):

i. Entries represent types of privilege. Note O, R, W, X denotes Own, Read, Write and Execute correspondingly.

ii. * denotes all columns in parallel table

	Course Table	**Enrolment Table**	**User Table**	**Student Table**	**Staff Table**	**Academic_staff_ teaches**
DBA	O(*), R(*), W(*), X(*)	O(*), R(*), W(*), X(*)	O(*), R(*), W(*), X(*)	O(*), R(*), W(*), X(*)	O(*), R(*), W(*), X(*)	O(*), R(*), W(*), X(*)
Admin Staff monitoring enrolment		R(*), W(*)				
Admin Staff manages courses	R (*), W(can modify course_id)					
Academic Staff	R (*), W (cannot modify course_id)		R (cannot read phone number and DOB)	R (cannot read nationality and outstanding_charges)	R (cannot read salary)	R (*)
Student /Academic staff	R (*), W (cannot modify course_id)		R (cannot read phone number and DOB)	R (cannot read nationality and outstanding_charges)	R (cannot read salary)	R (*)
Student	R (*)					R (*)

Table-level privileges	SQL statements (Comments can be found in the.sql file for clarity)	Description
Privilege 1: Admin staff managing enrollment can modify only enrollment information, and admin staff managing courses can modify only course information.	**Grants for Admin Staff responsible for monitoring enrolment** GRANT SELECT, UPDATE, INSERT, DELETE ON assignment2.Enrolment TO 'u02'@'%'; GRANT SELECT, UPDATE, INSERT, DELETE ON assignment2.Enrolment TO 'u11'@'%';	All admin staff that monitors enrollment can SELECT to display, as well as to UPDATE the values in all columns of the Enrolment table(enrolment_id, course_id, c_trimester, student_id, e_year, c_campus) to check information belonging to any student. The admins also are granted the privileges for INSERT or DELETE a row of the enrolment record.
	Grants for Admin Staff responsible for managing courses GRANT SELECT, UPDATE, INSERT, DELETE ON assignment2.Course TO 'u03'@'%'; GRANT SELECT, UPDATE, INSERT, DELETE ON assignment2.Course TO 'u12'@'%';	On the other hand, all admin staff that manage courses can SELECT to display the values in all columns of the Course table(course_id, c_trimester, course_name, c_campus) to check information about a course. Additionally, the admins who fall into this category are allowed to UPDATE the values in the Course table, along with permission on INSERT or DELETE rows of the course records.

Privilege 2: Both academic staff and students can see information about courses, but students cannot edit it.	**Grants for Academic Staff** GRANT SELECT, UPDATE(c_trimester,course_name,c_campus) ON assignment2.Course TO 'u04'@'%'; GRANT SELECT, UPDATE(c_trimester,course_name,c_campus) ON assignment2.Course TO 'u05'@'%'; GRANT SELECT, UPDATE(c_trimester,course_name,c_campus) ON assignment2.Course TO 'u06'@'%'; GRANT SELECT, UPDATE(c_trimester,course_name,c_campus) ON assignment2.Course TO 'u07'@'%';	The academic staff can SELECT to view all columns in the Course table, and is able to UPDATE the values in columns(c_trimester,course_name,c_campus) of the Course table. The primary key, course_id, can only be edited by admin staff who is responsible for managing courses.
	Grants for Students GRANT SELECT ON assignment2.Course TO 'u08'@'%'; GRANT SELECT ON assignment2.Course TO 'u09'@'%'; GRANT SELECT ON assignment2.Course TO 'u10'@'%';	Conversely, all students can SELECT to view the values in all columns of the Course table(course_id, c_trimester, course_name, c_campus), however they are not permitted to edit any values in the table.
Privilege 3: Academic staff can see the names and genders of the students but not their private information like birthdays or phone numbers.	**Grants for Academic Staff** GRANT SELECT(user_id, first_name, last_name, sex) ON assignment2.User TO 'u04'@'%'; GRANT SELECT(user_id, first_name, last_name, sex) ON assignment2.User TO 'u05'@'%';	Academic staff can SELECT to display the values in columns of User table(user_id, first_name, last_name, sex) and Student table(student_id, user_id) to check student's information. The two tables are

	GRANT SELECT(user_id, first_name, last_name, sex) ON assignment2.User TO 'u06'@'%'; GRANT SELECT(user_id, first_name, last_name, sex) ON assignment2.User TO 'u07'@'%'; GRANT SELECT(student_id, user_id) ON assignment2.Student TO 'u04'@'%'; GRANT SELECT(student_id, user_id) ON assignment2.Student TO 'u05'@'%'; GRANT SELECT(student_id, user_id) ON assignment2.Student TO 'u06'@'%'; GRANT SELECT(student_id, user_id) ON assignment2.Student TO 'u07'@'%';	related inferenced by user_id.
	Grants for DBA (Create View) GRANT SELECT, CREATE VIEW ON assignment2.Student TO 'u01'@'%'; GRANT SELECT, CREATE VIEW ON assignment2.User TO 'u01'@'%'; CREATE VIEW assignment2.academicSeeStudentNameGen AS SELECT s.student_id, u.first_name, u.last_name, u.sex FROM assignment2.User AS u, assignment2.Student AS s WHERE u.user_id = s.user_id; GRANT SELECT ON assignment2.academicSeeStudentNameGen TO 'u04'@'%'; GRANT SELECT ON assignment2.academicSeeStudentNameGen TO 'u05'@'%'; GRANT SELECT ON assignment2.academicSeeStudentNameGen TO 'u06'@'%';	It is bothersome for academic staff to have to check two tables individually and make decisions for determining a connection of two tables relating to one student. To address this, the DBA can SELECT all values within and CREATE VIEW to join the two tables (User and Student), retaining the information (student_id, first_name, last_name, sex) needed by academic staff, and grant the view to them.

	GRANT SELECT ON assignment2.academicSeeStudentNameGen TO 'u07'@'%';	
Privilege 4: All academic staffs and students can see information about all courses (including course_id, trimester, campus) and information on the staff teaching the course (including first_name, last_name and their position in the course)	**Grants for Academic Staff** GRANT SELECT(staff_id, course_id, c_trimester, c_campus, c_position) ON assignment2.Academic_staff_teaches TO ‘u04’@’%’; GRANT SELECT(staff_id, course_id, c_trimester, c_campus, c_position) ON assignment2.Academic_staff_teaches TO ‘u05’@’%’; GRANT SELECT(staff_id, course_id, c_trimester, c_campus, c_position) ON assignment2.Academic_staff_teaches TO ‘u06’@’%’; GRANT SELECT(staff_id, course_id, c_trimester, c_campus, c_position) ON assignment2.Academic_staff_teaches TO ‘u07’@’%’; GRANT SELECT(staff_id, user_id) ON assignment2.Staff TO 'u04'@'%'; GRANT SELECT(staff_id, user_id) ON assignment2.Staff TO 'u05'@'%'; GRANT SELECT(staff_id, user_id) ON assignment2.Staff TO 'u06'@'%'; GRANT SELECT(staff_id, user_id) ON assignment2.Staff TO 'u07'@'%'; GRANT SELECT(user_id, first_name, last_name) ON assignment2.User TO 'u04'@'%'; GRANT SELECT(user_id, first_name, last_name) ON assignment2.User TO 'u05'@'%';	Academic staff can SELECT to display the values in columns of Academic_staff_teaches table(staff_id, course_id, c_trimester, c_campus, c_position) to check course information and using staff_id as a reference on Staff table(staff_id, user_id), that gives user_id. Using the user_id found in the Staff table will return staff info(user_id, first_name, last_name) in the User table. The DBA can SELECT and CREATE VIEW to join the three tables(Academic_staff_teaches, Staff, User) while still retaining the information (course_id, c_trimester, c_campus, c_position, first_name, last_name) needed by academic staff, and grant the view to them.

GRANT SELECT(user_id, first_name, last_name) ON assignment2.User TO 'u06'@'%';

GRANT SELECT(user_id, first_name, last_name) ON assignment2.User TO 'u07'@'%';

Grants for Student

GRANT SELECT(staff_id, course_id, c_trimester, c_campus, c_position) ON assignment2.Academic_staff_teaches TO ‘u08’@’%’;

GRANT SELECT(staff_id, course_id, c_trimester, c_campus, c_position) ON assignment2.Academic_staff_teaches TO ‘u09’@’%’;

GRANT SELECT(staff_id, course_id, c_trimester, c_campus, c_position) ON assignment2.Academic_staff_teaches TO ‘u10’@’%’;

GRANT SELECT(staff_id, user_id) ON assignment2.Staff TO 'u08'@'%';

GRANT SELECT(staff_id, user_id) ON assignment2.Staff TO 'u09'@'%';

GRANT SELECT(staff_id, user_id) ON assignment2.Staff TO 'u10'@'%';

GRANT SELECT(user_id, first_name, last_name) ON assignment2.User TO 'u08'@'%';

GRANT SELECT(user_id, first_name, last_name) ON assignment2.User TO 'u09'@'%';

GRANT SELECT(user_id, first_name, last_name) ON assignment2.User TO 'u10'@'%';

Grants for DBA (Create View)

	GRANT SELECT, CREATE VIEW ON assignment2.Academic_staff_teaches TO 'u01'@'%'; GRANT SELECT, CREATE VIEW ON assignment2.Staff TO 'u01'@'%'; GRANT SELECT, CREATE VIEW ON assignment2.User TO 'u01'@'%'; CREATE VIEW assignment2.courseTeachingStaffDetails AS SELECT t.course_id, t.c_trimester, t.c_campus, t.c_position, u.first_name, u.last_name FROM assignment2.Academic_staff_teaches AS t, assignment2.Staff AS s, assignment2.User AS u WHERE t.staff_id = s.staff_id AND s.user_id = u.user_id; GRANT SELECT ON assignment2.courseTeachingStaffDetails TO 'u04'@'%'; GRANT SELECT ON assignment2.courseTeachingStaffDetails TO 'u05'@'%'; GRANT SELECT ON assignment2.courseTeachingStaffDetails TO 'u06'@'%'; GRANT SELECT ON assignment2.courseTeachingStaffDetails TO 'u07'@'%'; GRANT SELECT ON assignment2.courseTeachingStaffDetails TO 'u08'@'%'; GRANT SELECT ON assignment2.courseTeachingStaffDetails TO 'u09'@'%'; GRANT SELECT ON assignment2.courseTeachingStaffDetails TO 'u10'@'%';	

Privilege 5: All academic and admin staff can see the name and role of all staff, but are not allowed to modify. DBA can see and modify all the data staff can see, on top that also user_id, sex and DOB.	**Grants for Admin Staff** GRANT SELECT (first_name, last_name, role) ON assignment2.User TO 'u03'@'%'; GRANT SELECT, UPDATE, INSERT, DELETE ON assignment2.User TO 'u11'@'%'; GRANT SELECT, UPDATE, INSERT, DELETE ON assignment2.User TO 'u12'@'%'; GRANT SELECT, UPDATE, INSERT, DELETE ON assignment2.User TO 'u02'@'%'; **Grants for Academic Staff** GRANT SELECT, UPDATE, INSERT, DELETE ON assignment2.User TO 'u04'@'%'; GRANT SELECT, UPDATE, INSERT, DELETE ON assignment2.User TO 'u05'@'%'; GRANT SELECT, UPDATE, INSERT, DELETE ON assignment2.User TO 'u06'@'%'; GRANT SELECT, UPDATE, INSERT, DELETE ON assignment2.User TO 'u07'@'%'; **Grants for DBA** GRANT SELECT, UPDATE, INSERT, DELETE ON assignment2.User TO 'u01'@'%';	All staff can SELECT to display the values in columns of the User table(first_name, last_name, role) to check their own and fellow staff member's information. Like all other staff, the DBA can SELECT, but in addition to that they are also allowed to UPDATE the values in the User table, along with permission on INSERT or DELETE rows of the user records.

3. Create Views and Related Privileges

Set of views and related privileges	**SQL statements** **(Comments can be found in the.sql file for clarity)**	**Description**
View 1: Students should only see the grades that belong to themselves (and read only).	**Grants for Students** CREATE VIEW assignment2.studentOwnGrade AS SELECT e.course_id, g.score, g.grade FROM assignment2.Grade AS g, assignment2.Enrolment AS e, assignment2.Student AS s, assignment2.User AS u WHERE g.enrolment_id = e.enrolment_id AND e.student_id = s.student_id AND s.user_id = u.user_id AND u.user_id = (SELECT User from mysql.user WHERE user() like concat(user,'%')); GRANT SELECT ON assignment2.studentOwnGrade TO 'u07'@'%'; GRANT SELECT ON assignment2.studentOwnGrade TO 'u08'@'%'; GRANT SELECT ON assignment2.studentOwnGrade TO 'u09'@'%'; GRANT SELECT ON assignment2.studentOwnGrade TO 'u10'@'%';	Creates a view by joining Grade, Enrolment, Student and User tables. The view Showing only course_id, and score and grade that belongs to the logged-in student. The enrolment_id in the Grade table is used on the Enrolment table to get more information on to which course the grades belong. Student_id is used as the key to get user_id in the Student table, which can ultimately help obtain information in the User table consisting of the names of the students who studied the course. However, since only the logged-in student's detail is of interest, the user_id was checked against the current logged-in user to ensure this.
View 2: Academic staff can only see the enrolment of the courses they teach.	**Grants for Academic Staff** CREATE VIEW assignment2.academicStaffOwnStudentEnrolment AS SELECT e.enrolment_id, e.course_id,	Creates a view by joining Enrolment, Staff, Academic_staff_teaches tables. The view

	e.c_trimester, e.student_id, e.e_year, e.c_campus FROM assignment2.Enrolment AS e, assignment2.Staff AS s, assignment2.Academic_staff_teaches AS a WHERE a.course_id = e.course_id AND a.staff_id = s.staff_id AND s.user_id = (SELECT User from mysql.user WHERE user() like concat(user,'%')); GRANT SELECT ON assignment2.academicStaffOwnStudentEnrolment TO 'u04'@'%'; GRANT SELECT ON assignment2.academicStaffOwnStudentEnrolment TO 'u05'@'%'; GRANT SELECT ON assignment2.academicStaffOwnStudentEnrolment TO 'u06'@'%'; GRANT SELECT ON assignment2.academicStaffOwnStudentEnrolment TO 'u07'@'%';	Showing only student_id that owns the enrolment, and enrolment_id of enrolment, which associates course_id, and information of each course including trimester, year, and campus. The course_id in the Academic_staff_teaches table is used to make reference to the Enrolment table to get more information on enrolment that is on a specific course taught by academic staff. But as the logged-in academic staff is only allowed to access enrolments of the courses taught by themselves, there is a need for matching records on logged-in academic staff's staff_id. One way to obtain staff_id of current users is through looking into the User table, with the current user's user_id and finding the corresponding staff_id. Using the found staff_id on Academic_staff_teaches table enables record accessibility to that of enrolments of the courses taught by themselves only.

View 3: Academic staff can only modify grades of the course they teach.	**Grants for Academic Staff** CREATE VIEW assignment2.academicStaffOwnStudentGrade AS SELECT e.course_id, e.student_id, g.score, g.grade FROM assignment2.Grade AS g, assignment2.Staff AS s, assignment2.Academic_staff_teaches AS a, assignment2.Enrolment AS e, assignment2.User AS u WHERE g.enrolment_id = e.enrolment_id AND e.course_id = a.course_id AND a.staff_id = s.staff_id AND s.user_id = u.user_id AND u.user_id = (SELECT User from mysql.user WHERE user() like concat(user,'%')); GRANT SELECT, UPDATE(score, grade) ON assignment2.academicStaffOwnStudentGrade TO 'u04'@'%'; GRANT SELECT, UPDATE(score, grade) ON assignment2.academicStaffOwnStudentGrade TO 'u05'@'%'; GRANT SELECT, UPDATE(score, grade) ON assignment2.academicStaffOwnStudentGrade TO 'u06'@'%'; GRANT SELECT, UPDATE(score, grade) ON assignment2.academicStaffOwnStudentGrade TO 'u07'@'%';	Creates a view by joining Grade, Staff, Academic_staff_teaches, Enrolment, User tables. The view showing only the course_id of the courses the logged-in academic staff teach, and student_id of students in the course and their score and grade. The enrolment_id in the Grade table is used to make reference to the Enrolment table to get more information on enrolment records associate grade. Additionally, course_id in enrolment is used to sanitize records in Academic_staff_teaches table on the grounds of how not every course taught by academic staff is being undertaken by students, consequently no grades are relevant. On top of that, as the logged-in academic staff is only allowed to modify grades of the courses taught by themselves, there is a need for matching records on logged-in academic staff's staff_id to the Academic_staff_teaches table. One way to obtain staff_id of

		current users is through looking into the User table, with the current user's user_id and finding the corresponding staff_id. Using the found staff_id on Academic_staff_teaches table enables record modifiability of grades of the courses taught by themselves only.
View 4: Admin who are responsible for managing courses should only be able to see courses that are modified by themselves.	**Grants for Admin Staff responsible for managing courses** CREATE VIEW assignment2.adminStaffOwnCourses AS SELECT a.course_id, a.c_trimester, a.c_campus, a.date_modified FROM assignment2.Admin_staff_manages AS a, assignment2.Staff AS s, assignment2.User AS u WHERE a.staff_id = s.staff_id AND s.user_id = u.user_id AND u.user_id = (SELECT User from mysql.user WHERE user() like concat(user,'%')); GRANT SELECT ON assignment2.adminStaffOwnCourses TO 'u03'@'%'; GRANT SELECT ON assignment2.adminStaffOwnCourses TO 'u12'@'%';	Creates a view called adminStaffOwnCourses by joining Admin_staff_manages, Staff, User tables. The view showing only course_id that the admin is responsible for managing, and the course's associated trimester and campus, as well as date_modified. Admin who are responsible for managing courses are being granted the SELECT privilege of the view. The staff_id in the Admin_staff_manages table is used to make reference to the Staff table to get more information on staff records that are managing courses. In this case, it is required that the logged-in admin who is responsible for

		managing courses should only be able to see courses that are modified by themselves. To tackle this, there is a need for matching records on logged-in admin staff's staff_id to the Admin_staff_manages table. One way to obtain staff_id of current users is through looking into the User table, with the current user's user_id and finding the corresponding staff_id. Using the found staff_id on Admin_staff_manages table enables record readability of records on the courses modified by themselves only.
View 5: Admin who are responsible for monitoring enrolments should only be able to see enrolments that are monitored by themselves.	**Grants for Admin Staff responsible for monitoring enrolments** CREATE VIEW assignment2.adminStaffOwnEnrolments AS SELECT a.enrolment_id, a.date_monitored FROM assignment2.Admin_staff_monitors AS a, assignment2.Staff AS s, assignment2.User AS u WHERE a.staff_id = s.staff_id AND s.user_id = u.user_id AND u.user_id = (SELECT User from mysql.user WHERE user() like concat(user,'%')); GRANT SELECT ON assignment2.adminStaffOwnEnrolments TO 'u02'@'%'; GRANT SELECT ON assignment2.adminStaffOwnEnrolments TO 'u11'@'%';	Creates a view called adminStaffOwnEnrolments by joining Admin_staff_monitors, Staff, User tables. The view showing only enrolment_id for which the admin is responsible for monitoring, and the date_modified. Admin who are responsible for monitoring enrolments are being granted the SELECT privilege of the view. The staff_id in the Admin_staff_monitors

		table is used to make reference to the Staff table to get more information on staff records that are monitoring enrolments. In this case, it is required that the logged-in admin who is responsible for monitoring enrolments should only be able to see enrolments that are monitored by themselves. To tackle this, there is a need for matching records on logged-in admin staff's staff_id to the Admin_staff_monitors table. One way to obtain staff_id of current users is through looking into the User table, with the current user's user_id and finding the corresponding staff_id. Using the found staff_id on Admin_staff_monitors table enables record readability of records on the enrolments monitored by themselves only.

PART C. SQL Injection Test

1. Setup SQL Injection Environment

Step 1. In your VM, create a new directory /var/www/sqlitest, and copy the files index.html, unsafe_main.php, and safe_main.php in that directory. Set the permissions of the files so that anyone can read and execute them.

```
mkdir var/www/sqlitest
```

Permission states of the three files(index.html, safe_main.php, unsafe_main.php) after change, now anyone can read and execute the three files.

```
root@VM:/home/s5017393# chown s5017393:s5017393 var/www/sqlitest/index.html
root@VM:/home/s5017393# chown s5017393:s5017393 var/www/sqlitest/safe_main.php
root@VM:/home/s5017393# chown s5017393:s5017393 var/www/sqlitest/unsafe_main.php
root@VM:/home/s5017393# su s5017393
s5017393@VM:~$ cd var/www/
s5017393@VM:~/var/www$ ls
sqlitest
s5017393@VM:~/var/www$ cd sqlitest
s5017393@VM:~/var/www/sqlitest$ ls -la
total 28
drwxr-xr-x 2 s5017393 s5017393 4096 Sep 16 18:11 .
drwxrwxr-x 3 s5017393 s5017393 4096 Oct 11 00:46 ..
-rw-r--r-- 1 seed     seed     6148 Sep 16 17:48 .DS_Store
-rwxr-xr-x 1 s5017393 s5017393 1005 Sep 16 16:18 index.html
-rwxr-xr-x 1 s5017393 s5017393 1742 Sep 16 18:16 safe_main.php
-rwxr-xr-x 1 s5017393 s5017393 1247 Sep 16 18:16 unsafe_main.php
```

Step 2. To access the web interface in your VM with the URL http://www.sqlitest.com, you need to modify the /etc/hosts file in the VM to map URL to the VM's local IP address (i.e., 127.0.0.1).

The file **/etc/hosts** originally have the following permission, which abandons write rights to anyone except the owner of the file.

```
-rw-r--r--  1 root root     518 Apr 28  2018 hosts
```

It is then changed using the following command to allow read, write as well as execution permits to anyone.

```
sudo chmod a+rwx hosts
```

Now the file can be overridden with new information.

```
-rwxrwxrwx 1 root root 518 Apr 28  2018 hosts
```

To do this, can edit the **/etc/hosts** file (using **gedit**) to append the following line

```
127.0.0.1          www.sqlitest.com
```

```
127.0.0.1       www.repackagingattacklab.com
127.0.0.1       www.seedlabclickjacking.com
127.0.0.1       www.sqlitest.com
```

Also, now would need to edit the file **/etc/apache2/sites-available/000-default.conf** in the VM to configure the Apache server. Append the following lines to the file

However, to permit new changes to be added and saved to the file, the following command is needed to allow read, write as well as execution.

```
sudo chmod a+rwx apache2/sites-available/000-default.conf
```

Now the file can be overridden with new information.

```
-rwxrwxrwx 1 root root 2104 Apr 28  2018 apache2/sites-available/000-default.conf
```

```
<VirtualHost *:80>

ServerName http://www.sqlitest.com

DocumentRoot /var/www/sqlitest

</VirtualHost>

<VirtualHost *:80>
        ServerName http://www.seedlabclickjacking.com
        DocumentRoot /var/www/seedlabclickjacking
</VirtualHost>
<VirtualHost *:80>
        ServerName http://www.sqlitest.com
        DocumentRoot /var/www/sqlitest
</VirtualHost>
```

Step 3. Restart the Apache server and you should be able to open the following webpage via the URL http://www.sqlitest.com in the browser.

To be able to modify the preexist index.html and transform the form based on the student table proposed in the assignment, the preexist index.html needs to allow permission for modification of the file.

```
-rwxr-xr-x 1 seed seed 1005 Sep 16 16:18 index.html
```

Below is the command needed for allowing read, write as well as execution permission.

```
sudo chmod a+w index.html
```

Now the preexist table in the file can be modified and updated to conform to the table structure presented in Part A.

```
-rwxrwxrwx 1 seed seed 1005 Sep 16 16:18 index.html
```

Below is an image with form based on the structure of student table

Step 4. To allow the unsafe_main.php script to connect to the database developed, note that it makes the following assumptions:

a. The database name is **assignment2**.
b. There is a table named **student** which has the columns: student_id, first_name, last_name, DOB, sex, phone.

Changes needed to be made to the columns specified in assumption b in Step 4, namely first_name and last_name, DOB, sex and phone number all should not be there as they will foster a good number of data redundancy as was discussed in Part A during Database design phase.

Again, to be able to carry out changes on the file, it is necessary to change file permission to allow writing.

```
root@VM:/var/www/sqlitest# ls -l unsafe_main.php
-rwxr-xr-x 1 seed seed 1247 Sep 16 18:16 unsafe_main.php
root@VM:/var/www/sqlitest# sudo chmod a+w unsafe_main.php
root@VM:/var/www/sqlitest# ls -l unsafe_main.php
-rwxrwxrwx 1 seed seed 1247 Sep 16 18:16 unsafe_main.php
```

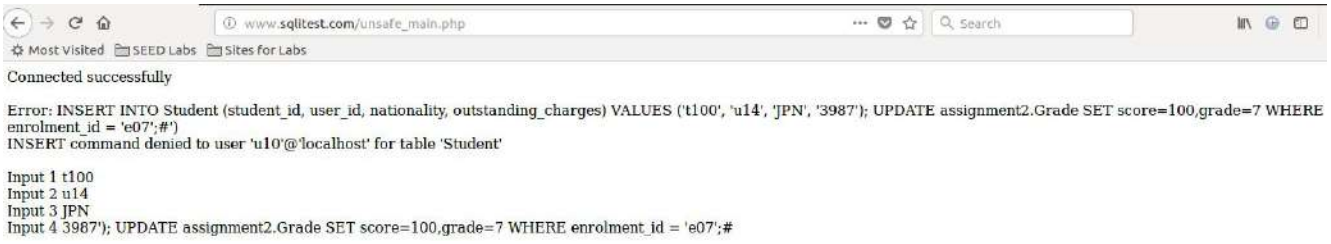
As the proposed database has a different setting than the assumed, it is essential to make changes to the unsafe_main.php file, otherwise fields will not match. The connection was tested by filling out the web form as below shown:

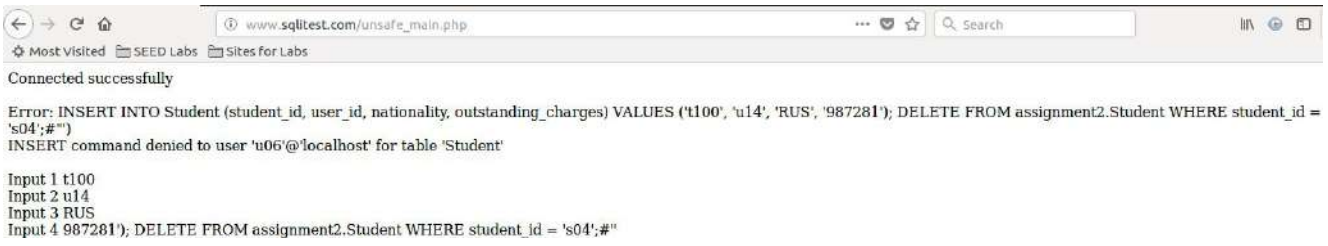
The below image showing the insertion of a new student record has been successful.

2. Make SQL Injection Attempt

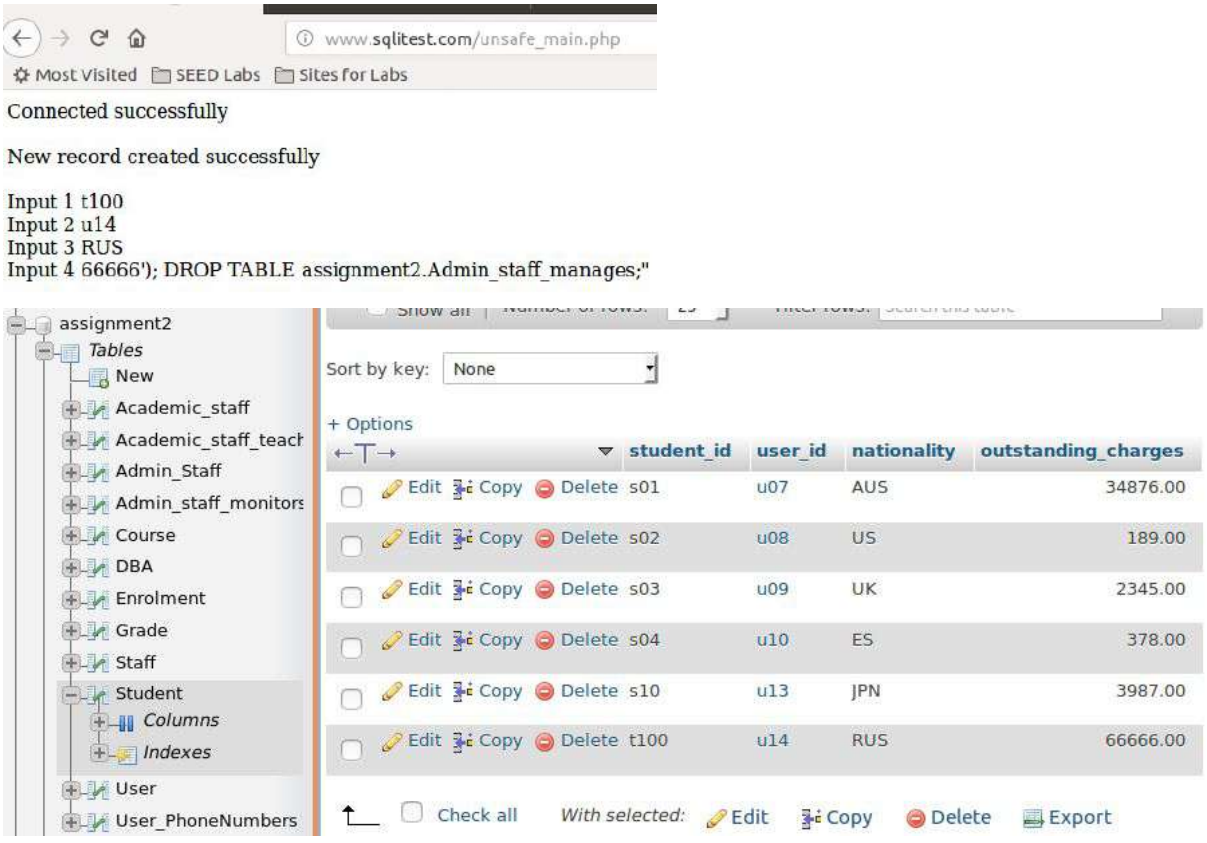
u01 is the account number for DBA, it has its own username as well as passwords to log in the backend database and accomplish privilege delegation and data maintenance. Though it was not implemented but it is assumed the DBA was granted all possible privileges to all tables that exist in the database. In other words, the DBA account is essentially the same as a root account. Consequently, due to the alike capability of root account as compared with DBA account, the DBA hence force will use the root account for any SQL injection demonstration on his/her part.

Step 5. You must complete this step using ONLY the above web form. Modify the database using SQL injection in at least 3 ways (i.e., different types of modifications on different objects), including the following 2 examples:

<table>
<tr><td colspan="2">SQL injection attempt 1</td></tr>
<tr><td>Type of modification</td><td>Type of user account</td></tr>
<tr><td>Update a grade of a student.</td><td>Student account
(u10)</td></tr>
<tr><td colspan="2">SQL injection statement:</td></tr>
<tr><td colspan="2">3987'); UPDATE assignment2.Grade SET score=100,grade=7 WHERE enrolment_id = 'e07';#"</td></tr>
<tr><td colspan="2">Result:</td></tr>
<tr><td colspan="2"> 
</td></tr>
<tr><td colspan="2">Explanation of result:</td></tr>
<tr><td colspan="2">Students do not have the INSERT privilege granted to update their grade associated enrolment e07, therefore the SQL injection to update their grade will be unsuccessful.</td></tr>
</table>

SQL injection attempt 2	
Type of modification	**Type of user account**
Delete a tuple of a student in the Student Table.	Staff account (u06)
SQL injection statement:	
987281'); DELETE FROM assignment2.Student WHERE student_id = 's04';#"	
Result:	
	
Explanation of result:	
Staff does not have DELETE privilege to delete tuples for the student with student_id s04 in the Student Table. Hence, the SQL injection to delete a tuple of a student in the Student Table will be unsuccessful.	

SQL injection attempt 3	
Type of modification	**Type of user account**
DBA drops the entire Admin_staff_manages Table.	DBA account (root)
SQL injection statement:	
66666'); DROP TABLE assignment2.Admin_staff_manages;"	
Result:	



Explanation of result:

The DROP command by DBA on Admin_staff_manages Table will be successful, because the Admin_staff_manages Table is not a parent table and does not have a parent row any other table or row depends on. Additionally, the DBA account has full privileges on the assignment2 database, and that includes the privilege to DROP the entire Admin_staff_manages Table.

Though DBA has full privilege, it will still be beneficial to test the DBA account to see whether the user input on the web programming side is sanitised correctly and whether all unauthorised access is prevented.

Step 6. Explain what you are trying to input and what are the expected results.

It was of concern that every input has to be filled to comply with the NULL constraints that the Student table has. Hence why all inputs in the form were all filled, and the code for executing SQL injection was entered through the last(4th) text input field in the website form. The input takes commands of which achieve different purposes. The primary key of the targeted table prone to SQL injection is used as the tracer, in WHERE clause to narrow down or to help direct the malicious user to the data to apply modification to.

Due to different user accounts' different privileges and their layer of protection, the expected result of SQL injection on user accounts will be different.

Albeit there are times when table constraints such as primary and foriegn keys can play a big part at failing SQL injections as found in SQL injection Attempt 3, it is known that most of the time it is guaranteed the malicious user is able to perform a specific SQL injection attack, when user accounts have the granted privileges needed. Since the DBA account is granted with all privileges including the DROP privilege to drop the entire Admin_staff_manages Table, the SQL injections on the DBA account will be successful.

As student and staff accounts have limited privileges and are unable to perform some listed specific attacks, the SQL injection on these accounts will not be successful:

- Students do not have the needed INSERT privilege to update their grades.
- Staff does not have the needed DELETE privilege to delete a tuple of a student in the Grade Table.

3. Understand the Countermeasure

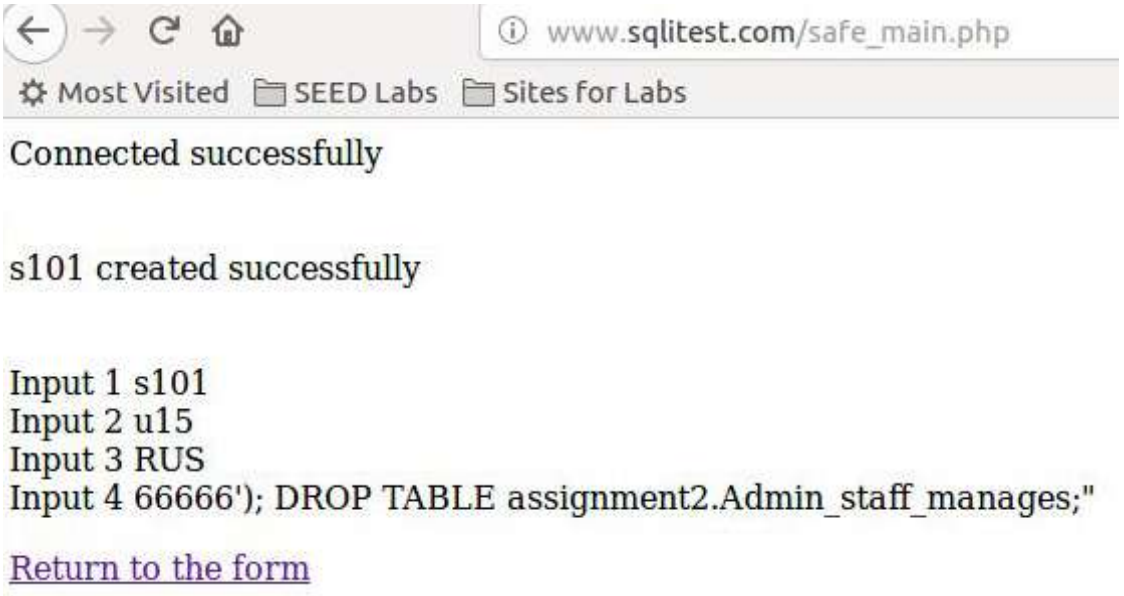
Step 7. Edit the index.html file by replacing ‘unsafe_main.php’ with ‘safe_main.php’. This is a version with SQL injection protection. Try the same inputs on which you succeeded previously.

Using the ‘unsafe_main.php’ file (refer to Step 5), it is understood that the only successful SQL injection is executed by using the DBA account, as DBA has full privileges to the assignment2 database.

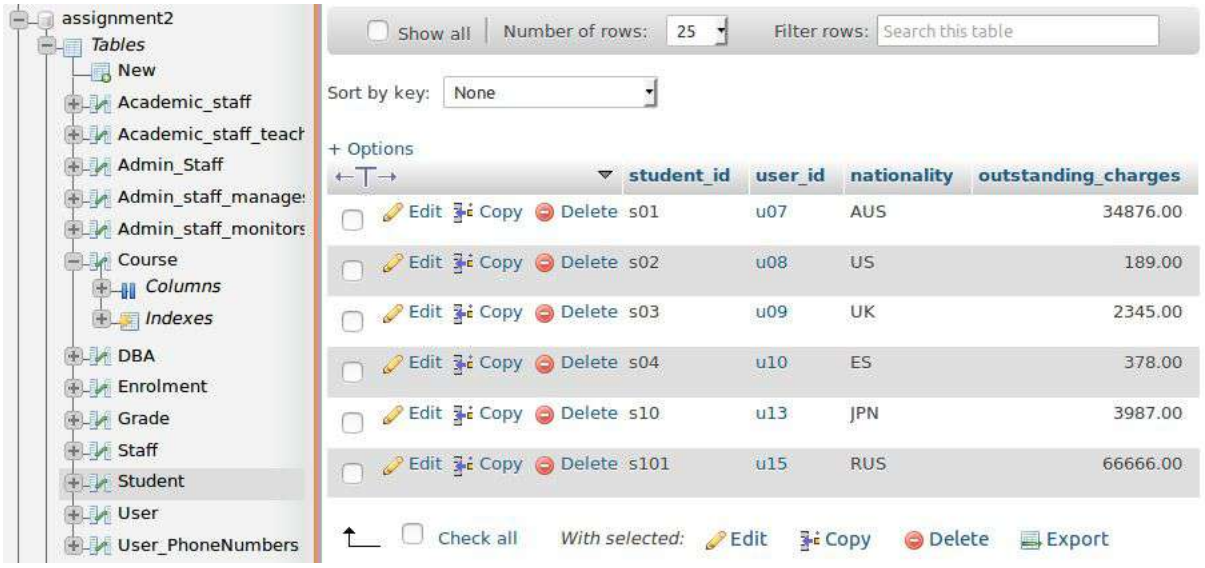
With the index.html file edited and have replaced ‘unsafe_main.php’ with ‘safe_main.php’ (a version with SQL injection protection), the SQL injection that was once successful using the DBA account will fail when the same inputs are given as in previous.

Using the same inputs succeeded previously shown below, and enter the command into the last text field in the website form:

66666'); DROP TABLE assignment2.Admin_staff_manages;"



The image above demonstrates the new student record for the student with number s101 has been inserted successfully, but the table is not being dropped. As can be seen by the attached image below indicating the database structure after the second SQL injection attempt combating data security safety measures established in safe_main.php.



The reason why the student record is able to be created was due to its fulfilment on the bind parameters of the prepared statement.

```
// Prepared statement
$stmt = $conn->prepare("INSERT INTO Student (student_id, user_id, nationality, outstanding_charges) VALUES (?,?,?,?)");

if ($stmt) {
    // Bind Parameters
    $stmt->bind_param('sssd', $student_id, $user_id, $nationality, $outstanding_charges);
    $stmt->execute();
    if ($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;
```

The bind parameters tell the database what the parameters are. The ‘sssd’ argument lists the types of data the parameters are for student_id, user_id, nationality, and outstanding_charges. In this case, the s character tells MySQL that the parameter is a string, and d tells a double data type. The input to each parameter [student_id, user_id, nationality, and outstanding_charges] correspondingly were passed in [‘s101’,’u15’,’RUS’,66666’); DROP TABLE assignment2.Admin_staff_manages;"]. The first three parameters were perfectly written, except for the last which was ambiguous. Nevertheless, MySQL treats the last parameter as a double number and discarding digits/characters after the 5th ‘6’ that’s not a valid number digit.

Step 8. Describe your observations and try to explain the difference.

The core difference between the ‘unsafe_main.php’ file and the ‘safe_main.php’ file is that the ‘unsafe_main.php’ file does not have the ‘prepare statement’ that the ‘safe_main.php’ file has.

A ‘prepared statement’ is a dynamic template-like feature that is brought in to stop the misuse of SQL commands and it aims to stop SQL injections by treating hash ‘#’, and brackets ‘()’ as text, not code. In the ‘prepare statement’, it involves developers to define the specified parameters labelled with question marks ‘?’. The marks can later be replaced by actual values passed in from the user end. Passed in values needs to comply with the data type restriction set out by prepared parameters. The ‘prepared statement’ is especially efficient in defending against SQL injection due to this characteristic (W3School, 2020).

Without the ‘prepared statement’ can mean that the user inputs are not sanitised. Unsanitised user input creates an opportunity for hackers to exploit and mix the SQL commands and user input by using hash and brackets to alter and break the SQL code and enter information that they are not supposed to do (W3School, 2020).

In other words, unsanitised user inputs make the ‘unsafe_main.php’ prone to SQL injection attacks and allow the attacks on the DBA account to be achieved.

Having ‘prepared statement’ embedded in PHP files and user input sanitised provide an additional layer of protection to the owners and users of the database (W3School, 2020).

Reference:

Barton, D., & Court, D. (2013). *Three Keys to Building A Data-driven Strategy.* McKinsey Digital. Retrieved October 11, 2020 from https://www.mckinsey.com/business-functions/mckinsey-digital/our-insights/three-keys-to-building-a-data-driven-strategy#

Bogdan, M., & Borza, A. (2019). Big Data Analytics and Organizational Performance: A Meta-Analysis Study. *Management and Economics Review, 4*(2), 147-162. https://www.researchgate.net/publication/337837289_Big_Data_Analytics_and_Organizational_Performance_A_Meta-Analysis_Study

Cai, L., & Zhu, Y. (2015). The Challenges of Data Quality and Data Quality Assessment in the Big Data Era. *Data Science Journal, 14*(1), 2. http://doi.org/10.5334/dsj-2015-002

Pollack, Ed. (2016). *The Benefits, Costs, and Documentation of Database Constraints.* SQL Shark. Retrieved October 11, 2020 from https://www.sqlshack.com/the-benefits-costs-and-documentation-of-database-constraints/

Kovalenko, K. (2019, May 14). The Costs of Data Redundancy and Data Inconsistency. *BizData.* https://www.bizdata.com.au/blogpost.php?p=costs-of-data-redundancy-and-data-inconsistency

W3School. (2020). PHP MySQL Prepared Statements. W3School. Retrieved October 11, 2020 from https://www.w3schools.com/php/php_mysql_prepared_statements.asp

Wang, Z. (2020). *Lecture H: Database Administration* [Lecture Slides]. Griffith University. https://learn-ap-southeast-2-prod-fleet01-xythos.s3.ap-southeast-2.amazonaws.com/5bb70f08ac35e/6950713?response-cache-control=private%2C%20max-age%3D21600&response-content-disposition=inline%3B%20filename%2A%3DUTF-8%27%27Lab_H_DB_Admin.pdf&response-content-type=application%2Fpdf&X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Date=20201010T120000Z&X-Amz-SignedHeaders=host&X-Amz-Expires=21600&X-Amz-Credential=AKIAYDKQORRYZBCCQFY5%2F20201010%2Fap-southeast-2%2Fs3%2Faws4_request&X-Amz-Signature=ac904d597299004f0a9983ecd6433bcbe82e4f6c64139cf95e1a927b8f19ea7d

APPENDIX

Student:

student_id	first_name	last_name	DOB	sex	phone
s01	Angela	Merkal	01/01/1991	F	543210
s02	Donaldo	True	02/02/1992	M	123456
s03	Hillarious	Blinton	03/03/1993	F	112233
s04	Tarra	Obana	04/04/1994	M	221134

Academic_staff:

staff_id	first_name	last_name	position
a01	Seb	Binary	Professor
a02	Jazz	Wood	Asso. Professor
a03	Miguel	Franco	Lecturer

Admin_staff:

staff_id	first_name	last_name	duty
f01	Cristiano	Penaldo	Enrolment
f02	Lionel	Missy	Courses

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

Enrolment:

enrolment_id	course_id	student_id	student_name	year	trimester	campus
e01	101ICT	s01	Angela Merkal	2017	1	GC
e02	101STA	s01	Angela Merkal	2017	1	GC
e03	101CS	s01	Angela Merkal	2017	1	GC
e04	102ICT	s02	Donaldo True	2018	2	NA
e05	101CS	s02	Donaldo True	2017	1	GC
e06	102CS	s03	Hillarious Blinton	2018	3	GC
e07	102ICT	s04	Tarra Obana	2018	2	NA
e08	101STA	s04	Tarra Obana	2017	1	NA
e09	101CS	s04	Tarra Obana	2017	1	GC
e10	102CS	s04	Tarra Obana	2018	1	GC

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

User:

user_id	first_name	last_name	role
u01	Bruce	Whyne	DBA
u02	Cristiano	Penaldo	admin staff
u03	Lionel	Missy	admin staff
u04	Seb	Binary	academic staff
u05	Jazz	Wood	academic staff
u06	Miguel	Franco	academic staff
u07	Angela	Merkal	student
u08	Donaldo	True	student
u09	Hillarious	Blinton	student
u10	Tarra	Obana	student