VIETNAM NATIONAL UNIVERSITY, HO CHI MINH CITY
UNIVERSITY OF TECHNOLOGY
FACULTY OF COMPUTER SCIENCE AND ENGINEERING

DATABASE SYSTEM COURSE - CO2013

Database System Assignment 2

Advisors: Mr. Pham Nguyen Hoang Nam
Mr. Do Thanh Thai

Students:
Bui Quang Tien - 1953018
Tran Quoc Duy - 1952214
Le Minh Dang - 1952041
Hoang Viet Thong - 1750060

Ho Chi Minh City, October 2021

Contents

1 Workload

No.	Work	Name	Contribution
1	Improve Database	Le Minh Dang, Tran Quoc Duy	Edit ER, Schema, SQL code (all 100%)
2	Database Security	Bui Quang Tien	100‘%
3	Normalization	Le Minh Dang, Hoang Viet Thong	(100‘%)
4	Query (AAO)	Le Minh Dang	100%
5	Query (Faculty)	Tran Quoc Duy	100%
6	Query (Lecturer)	Hoang Viet Thong	100%
7	Query (Student)	Bui Quang Tien	100%
8	Indexing	Le Minh Dang	100%
9	Stored Procedure, Trigger	Tran Quoc Duy	100%

2 Overview

In this assignment, we are asked to implement a case study to a database management system based on the given requirements. Specifically, we will be designing a teaching database system. For assignment 1, we will be developing an database with SQL codes using the database management system MySQL. In the Assignment 2, we will go deeper in our database to do some Trigger, Procedures, Indexing, Database Security...

3 Requirement Description for Teaching System

3.1 Requirement Description

- In the course registration system, there are two kind of **STUDENTS**, students in "active" status can enroll to any **COURSE**, on the other hand, they will be suspended for a semester. These two entities **COURSE** and **STUDENT** has a relationship named **REGISTER** whose attribute is **Semester**. One **STUDENT** can join in two or more or no **COURSE** (if he or she is suspended) , and one **COURSE** can have zero (if the course is not open in the semester) or more **STUDENT**. This is **Many to Many** relationship.

- In a semester, a **COURSES** arr organized into many classes, each **CLASS** can be only belong to one **COURSE**. The relationship between these two entities is named **HAS**, this is **one to Many** relationship.

- Each **STUDENT** can only join in a **CLASS** up to 60 people and only one **CLASS** for a course.

- Each class is taught by some lecturers in different weeks of a semester, we have another entity **LECTURER** and a relationship **TEACHES** between **LECTURER** and **CLASS**. Vice versa, a **LECTURER** can take the responsibility to teach theory for more than one **CLASS**.

- Each class can be only graded by a specific **LECTURER** whose take responsibility to the students' score board, so we have one more relationship between **LECTURER** and **CLASS** which is named **GRADE**.

- Some textbooks are available for each course. Some courses may use same textbooks. Therefore, **COURSE** entity has a many-to-many relationship **USES** with **TEXTBOOK**.

- Theory **LECTURERS** can assign 1 to 3 **TEXTBOOKS** for each specific course, beside that, each **TEXTBOOK** can be assigned by one or more **LECTURERS**.

- A **TEXTBOOK** is written by some **AUTHORS** and published by one **PUBLISHER** An **AUTHORS** may writes more than one **TEXTBOOK**. So we have an entity AUTHOR which has a Many-to-Many relationship WRITES with **TEXTBOOK**, and **PUBLISHER** has a one-to-many relationship **PUBLISHES** with **TEXTBOOK**.
- Beside that, a **TEXTBOOK** is **bought** by some **PUBLISHERS** which are domestic or oversea, vice versa. So this is many to many relationship
- **FACULTY** is a big Unit in a University which manage **STUDENT**, **LECTURER** and **COURSES**, These Relationship between **FACULTY** AND **LECTURER, STUDENT, COURSES** is 1 to N relationship [3]

3.2 Constraints

- Students can join many classes and courses
- Student is only allowed to join one class for a course.
- Each class has at most 60 students
- One class is organized for a specific course
- Each course has from 1 to 3 credits
- Student can enroll at most 18 credits in a semester
- Lecturer can assign 1 to 3 textbooks to managed courses in a semester
- Main textbooks must be published at most 10 years ago

4 Conceptual Design

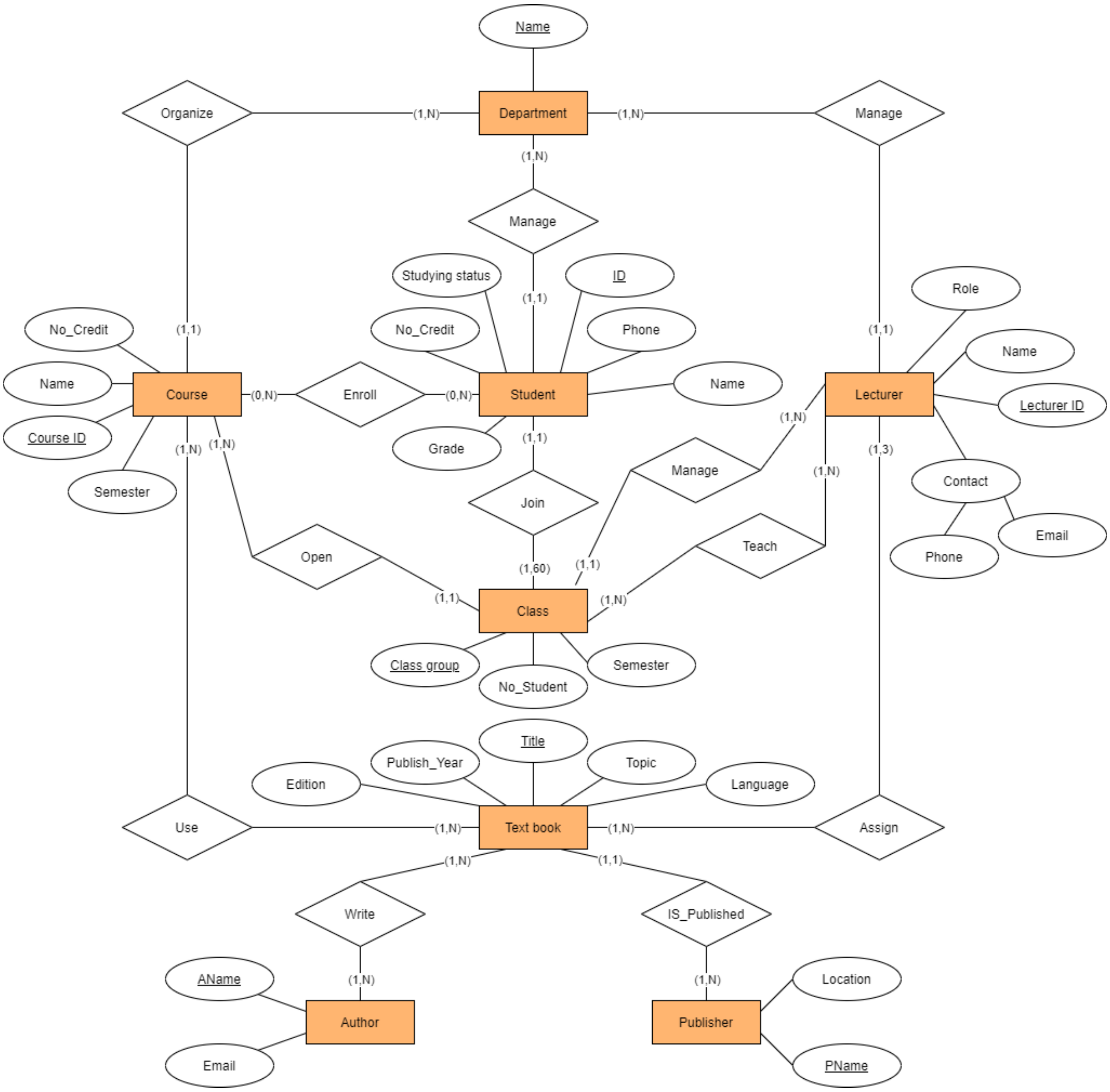
4.1 Entities Description

Entity	Description
Student	The list of the students in the education system. Unless students have normal studying status, they can not enroll in the courses. Each semester, a student can register maximum 18 credits.
Course	The list of courses opened in a semester, normally have 1 to 3 credits. Each course may has one or more classes. A course may use one or more text books.
Class	The list of classes opened by a course, each class has a maximum of 60 students. Each class has one main lecturer recording the scores for students at the end of the semester.
Lecturer	The list of the lecturers who in charge of the classes in a semester. Each lecturer teach one or more classes.
Text book	The list of the text books used in the courses. A text book may be used in one or more courses. The main text book must has the publish year less than 10 years.
Author	The list of the authors of the textbooks used in the university.
Publisher	The list of the publishers from whom the text books are published and bought, consist of local and global publishers.
Faculty	The list of units in a university, each faculty manage and organize every activity of a Major, including Students, Lecturers and Courses that belong to a specific major

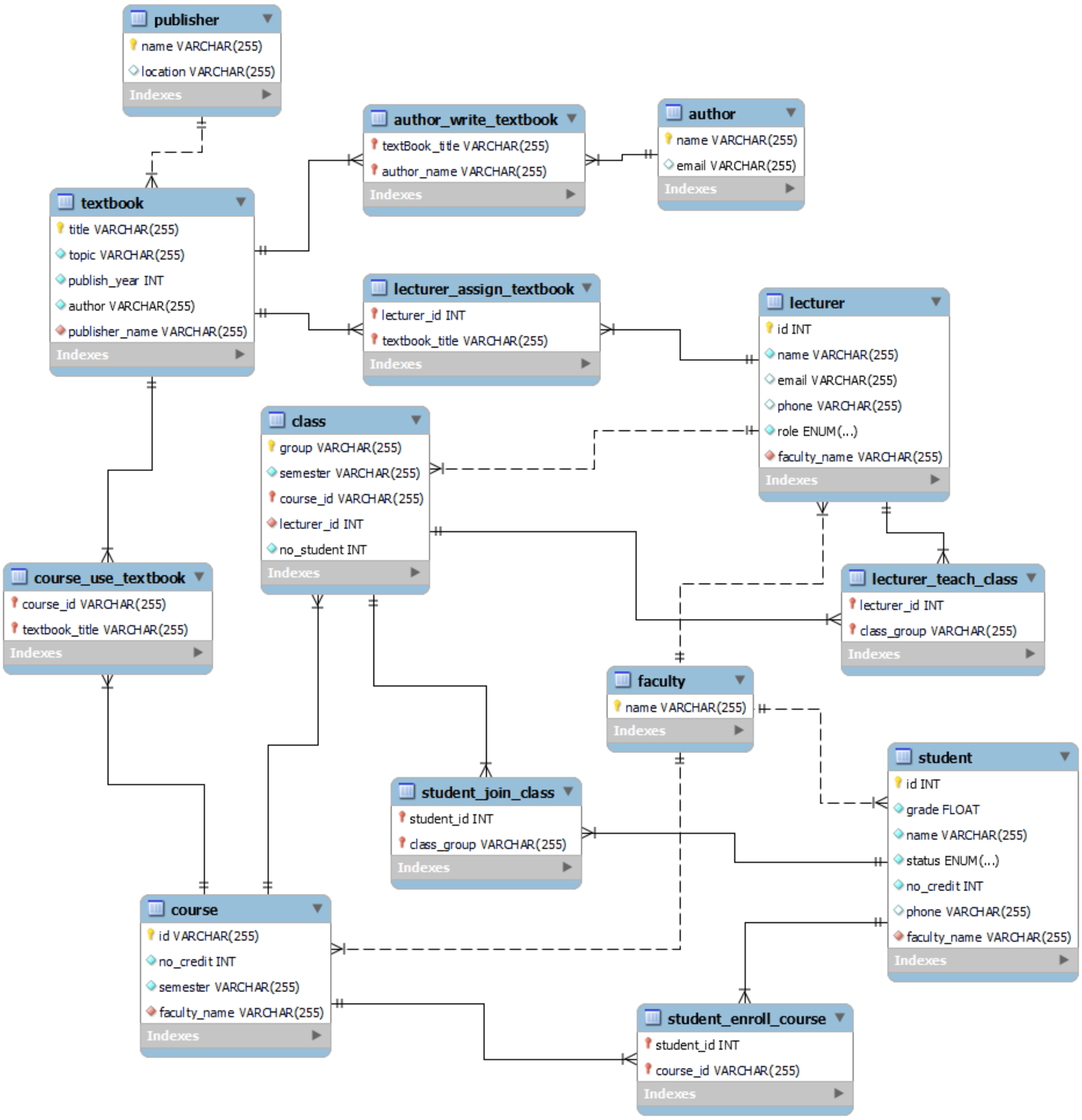
4.2 Relationship Description

Relationship	**Participants and Constrain**
Enroll	Student and Course; M:N relationship.
Join	Student and Class; 1:N relationship (1 Student to 1 Class, 1 Class to N Student).
Open	Course and Class; 1:N relationship (1 Course to N Class, 1 Class to 1 Course).
Use	Course and Text Book; M:N relationship
Assign	Lecturer and Text Book; M:N relationship
Teach	Lecturer and Class; M:N relationship.
Manage	Lecturer and Class; 1:N relationship (1 Lecturer to N Course, 1 Course to 1 Lecturer).
Write	Author and Text Book; M:N relationship.
Is_Published	Text Book and Publisher; 1:N relationship (1 Text Book to 1 Publisher, 1 Publisher to N Text Book).

4.3 Entity-Relationship Diagram



5 Logical Design



6 DBMS

6.1 The selection of DBMS: MySQL Workbench

In this project, we make considerable use of MySQL Workbench because of its simplicity and ease of use. MySQL Workbench is an unified visual tool for database architects, developer, and DBAs. MySQL Workbench offers data modeling, SQL programming, and full administrative tools for server configuration, user management, backup, and much more. MySQL Workbench can runs on Windows, Linux, and Mac OS X.

6.2 Implement the DBs

In this section, we will demonstrate the SQL code used to create the tables.

6.2.1 Faculty

```sql
create table if not exists Faculty(
`name` varchar(255) not null unique,
primary key(`Name`)
);
```

6.2.2 Student

```sql
create table if not exists Student(
`id` varchar(255) not null unique,
`grade` float not null,
`name` varchar(255)        not null,
`status` enum('Active','Suspended') not null default 'Active',
`no_credit` int not null,
`phone` int unique,
`faculty_name`         varchar(255) not null,
check(`grade` >=0 and `grade` <=10),
check(`no_credit` >=0 and `no_credit` <= 18),
primary key(`id`),
foreign key (`faculty_name`) references Faculty(`name`)
);
```

6.2.3 Class

```
create table if not exists Class(
`group` varchar(255)        not null,
`semester` varchar(255)        not null,
`course_id` varchar(255)        not null,
`lecturer_id` int not null,
`no_student` int not null default 0,
check (`no_student` >= 0 and `no_student` <= 60),
primary key (`group`,`course_id`),
foreign key (`course_id`) references Course(`id`),
foreign key (`lecturer_id`) references Lecturer(`id`)
);
```

6.2.4 Course

```
create table if not exists Course(
`id` varchar(255) not null unique,
`no_credit` int not null,
`semester`          varchar(255) not null,
`faculty_name` varchar(255)        not null,
check(`no_credit` >=1 and `no_credit` <=3),
primary key(`id`),
foreign key (`faculty_name`) references Faculty(`name`)
);
```

6.2.5 Lecturer

```
create table if not exists Lecturer(
`id` int not null unique,
`name` varchar(255) not null,
`email` varchar(255) unique,
`phone` varchar(255) unique,
`role`  enum('teaching_system','Grading') not null,
`faculty_name`        varchar(255) not null unique,
primary key(`id`),
```

```
foreign key (`faculty_name`) references Faculty(`name`)
);
```

6.2.6 Textbook

```
create table if not exists Textbook (
`title` varchar(255) not null unique,
`topic` varchar(255) not null,
`publish_year` int not null,
`author` varchar(255) not null,
`publisher_name` varchar(255)        not null,
check (`publish_year` <= 10 and `publish_year` > 0),
primary key(`title`),
foreign key (`author`) references Author(`name`),
foreign key (`publisher_name`) references Publisher(`name`)
);
```

6.2.7 Author

```
create table if not exists Author(
`name`  varchar(255) not null unique,
`email` varchar(255) not null,
primary key(`name`)
);
```

6.2.8 Publisher

```
create table if not exists Publisher(
`name` varchar(255) not null unique,
`location` varchar(255),
primary key(`name`)
);
```

6.2.9 Student_join_Class

```
create table if not exists Student_join_Class(
`student_id` varchar(255) not null,
`class_group` varchar(255) not null,
constraint `pk_join` primary key (`student_id`,`class_group`),
foreign key (`student_id`) references Student(`id`),
foreign key (`class_group`) references Class(`group`)
);
```

6.2.10 Student_enroll_course

```
create table if not exists Student_enroll_Course(
`student_id` varchar(255) not null,
`course_id` varchar(255) not null,
constraint `pk_enroll` primary key (`student_id`,`course_id`),
foreign key (`student_id`) references Student(`id`),
foreign key (`course_id`) references Course(`id`)
);
```

6.2.11 Lecturer_teach_Class

```
create table if not exists Lecturer_teach_Class(
`lecturer_id` int not null,
`class_group` varchar(255) not null,
constraint `pk_lecturer_teach_class` primary key(`class_group`,`lecturer_id`),
foreign key (`class_group`) references Class(`group`),
foreign key (`lecturer_id`) references Lecturer(`id`)
);
```

6.2.12 Course_use_Textbook

```
create table if not exists Course_use_Textbook(
`course_id` varchar(255) not null,
`textbook_title`varchar(255) not null,
```

```
constraint `pk_course_use_textbook` primary key (`course_id`,`textbook_title`),
foreign key(`course_id`) references Course(`id`),
foreign key(`textbook_title`) references Textbook(`title`)
);
```

6.2.13 Lecturer_assign_Textbook

```
create table if not exists Lecturer_assign_TextBook(
`lecturer_id` int not null,
`textbook_title` varchar(255) not null,
constraint `pk_lecturer_assign_textbook` primary key(`lecturer_id`,`textbook_title`
foreign key(`lecturer_id`) references Lecturer(`id`),
foreign key(`textbook_title`) references Textbook(`title`)
);
```

6.2.14 Author_write_Textbook

```
create table if not exists Author_write_Textbook(
`textBook_title` varchar(255) not null,
`author_name` varchar(255) not null,
constraint `pk_textbook_write_author` primary key(`textbook_title`,`author_name`),
foreign key(`textbook_title`) references Textbook(`title`),
foreign key(`author_name`) references Author(`name`)
);
```

6.2.15 Adding the INDEX

```
ALTER TABLE Student
ADD INDEX student_idx(`id`);
ALTER TABLE Lecturer
ADD INDEX lecturer_idx(`id`);
ALTER TABLE Publisher
ADD INDEX publisher_idx(`name`);
ALTER TABLE Author
ADD INDEX author_idx(`name`);
```

7 Data Requirement

7.1 Academic Affairs Office

(i.1). Cập nhật đăng ký môn học của các lớp.

```sql
DELIMITER //
CREATE PROCEDURE update_courseOfClass( in `class_group` varchar(255),
in `course` varchar(255) , in `sem` varchar(255),
in `no_stu` int, in `lec_id` int)
BEGIN
UPDATE class AS c
SET c.`course_id` = `course`, c.`semester` = `sem`,
c.`no_student` = `no_stu`, c.`lecturer_id` = `lec_id`
WHERE c.`group` = `class_group`;
END //
DELIMITER ;
```

(i.2). Xem danh sách lớp đã được đăng ký bởi một sinh viên ở một học kỳ.

```sql
SELECT s.`id` AS `Student`, c.`group` AS `Class`, c.`course_id` AS `Course`
FROM Class AS c
INNER JOIN Student_Join_Class AS sc
ON  c.`group` = sc.`class_group`
INNER JOIN
Student AS s
ON sc.`student_id` = s.`id`;
```

(i.3). Xem danh sách lớp được phụ trách bởi một giảng viên ở một học kỳ.

```sql
SELECT l.`id` AS `Lecturer`, c.`course_id` AS `Course`, c.`group` AS `Class`
FROM Class AS c
INNER JOIN Lecturer_teach_class AS lc
ON  c.`group` = lc.`class_group`
INNER JOIN
Lecturer AS l
ON lc.`lecturer_id` = l.`id`;
```

(i.4). Xem danh sách môn học được đăng ký ở mỗi học kỳ ở mỗi khoa.

```
SELECT c.`id`, c.`semester`,f.`name`
FROM Course AS c
INNER JOIN Faculty AS f
ON c.`faculty_name` = f.`name`;
```

(i.5). Xem danh sách sinh viên đăng ký ở mỗi lớp ở mỗi học kỳ ở mỗi khoa.

```
SELECT s.`id` AS `Student`, c.`course_id` AS `Course`,
c.`group` AS `Class`, c.`semester`AS`Semester`, f.`name`AS `Faculty`
FROM Class AS c
INNER JOIN Student_Join_Class AS sc
ON  c.`group` = sc.`class_group`
INNER JOIN Student AS s
ON sc.`student_id` = s.`id`
INNER JOIN Faculty AS f
ON s.`faculty_name` = f.`name`;
```

(i.6). Xem danh sách giảng viên phụ trách ở mỗi lớp ở mỗi học kỳ ở mỗi khoa.

```
SELECT l.`id` AS `Lecturer`, c.`course_id` AS `Course`,
c.`group` AS `Class`, c.`semester`AS`Semester`, f.`name`AS `Faculty`
FROM Class AS c
INNER JOIN Lecturer_teach_Class AS lc
ON  c.`group` = lc.`class_group`
Inner JOIN Lecturer AS l
ON lc.`lecturer_id` = l.`id`
INNER JOIN Faculty AS f
ON l.`faculty_name` = f.`name`;
```

(i.7). Xem các giáo trình được chỉ định cho mỗi môn học ở mỗi học kỳ ở mỗi khoa.

```
SELECT t.`title` AS 'Textbook', c.`id` AS 'Course',
c.`semester` AS 'Semester', f.`name` AS 'Faculty'
FROM Textbook AS t
```

```
INNER JOIN Course_use_textbook AS ct
ON t.`title` = ct.`textbook_title`
INNER JOIN        Course AS c
ON ct.`course_id` = c.`id`
INNER JOIN faculty AS f
ON c.`faculty_name` = f.`name`;
```

(i.8). Xem tổng số môn học được đăng ký ở mỗi học kỳ ở mỗi khoa.

```
SELECT  c.`id` AS 'Course Name', COUNT(c.`id`) AS 'Total Courses',
c.`semester` AS 'Semester', f.`name` AS 'Faculty'
FROM Course AS c
LEFT OUTER JOIN Faculty AS f
ON c.`faculty_name` = f.`name`
GROUP BY c.`id`, c.`semester`, f.`name`;
```

(i.9). Xem tổng số lớp được mở ở mỗi học kỳ ở mỗi khoa.

```
SELECT DISTINCT course.`id` AS 'Course Name',
COUNT(class.`group`) AS 'Total class',
course.`semester` AS 'Semester', f.`name` AS 'Faculty'
FROM Class AS class
INNER JOIN Course AS course
ON class.`course_id` = course.`id`
RIGHT JOIN Faculty AS f
ON course.`faculty_name` = f.`name`
GROUP BY  course.`id`,class.`group`, course.`semester`, f.`name`;
```

(i.10). Xem tổng số sinh viên đăng ký ở mỗi lớp của một môn học ở một học kỳ.

```
SELECT COUNT(s.`id`) AS 'Number Students', class.`group` AS 'Class',
course.`id` AS 'Course', course.`semester` AS 'Semester'
FROM student as s
INNER JOIN student_join_class AS sc
ON s.`id` = sc.`student_id`
INNER JOIN class
ON sc.`class_group` = class.`group`
```

```
INNER JOIN course
ON class.`course_id` = course.`id`
GROUP BY s.`id`,class.`group`,course.`id`;
```

(i.11). Xem tổng số sinh viên đăng ký ở mỗi môn học ở một học kỳ.

```
SELECT COUNT(sc.`student_id`) AS 'Number of students',
c.`id` AS 'Course Name', c.`semester` AS 'Semester'
FROM student_enroll_course AS sc
INNER JOIN course AS c
ON sc.`course_id` = c.`id`
GROUP BY sc.`student_id`, c.`id`;
```

(i.12). Xem tổng số sinh viên đăng ký ở mỗi học kỳ ở mỗi khoa.

```
SELECT COUNT(sc.`student_id`) AS 'Number of students',
c.`id` AS 'Course Name',c.`semester` AS 'Semester',
c.`faculty_name` AS 'Faculty'
FROM student_enroll_course AS sc
RIGHT JOIN course AS c
ON sc.`course_id` = c.`id`
GROUP BY sc.`student_id`,c.`id`, c.`semester`, c.`faculty_name`;
```

7.2 Faculty manages Study Programs

(ii.1). Update the list of courses that are opened before the start of each semester.

```
delimiter //
create procedure update_course(in `course_id` varchar(255), in `sem` int)
begin
update Course set `semester` = `sem` where `id` = `course_id`;
end //
delimiter ;
```

(ii.2). Update the list of lecturers in charge of each class that is opened at the beginning of each semester.

```
delimiter //
create procedure update_class(in `cID` varchar(255), in `gr` varchar(255),
    in `sem` int, in `lID` int, in `no_stu` int)
begin
update Class
set `semester` = `sem`, `lecturer_id` = `lID`, `no_student` = `no_stu`
where (`course_id` = `cID` and `group` = `gr`);
end //
delimiter ;
```

(ii.3). View the list of courses in a semester.

```
use teaching_system;
select `semester`, `id`, `no_credit`, `faculty_name`
from Course
order by `semester` desc, `id` asc;
```

(ii.4). View the list of lecturers in a semester.

```
use teaching_system;
select Class.`semester`, Lecturer.`id`, Lecturer.`name`, Lecturer.`email`,
    Lecturer.`phone`, Lecturer.`faculty_name`
from Lecturer
inner join Class
on Lecturer.`id` = Class.`lecturer_id`
order by Class.`semester` desc, Lecturer.`id` asc;
```

(ii.5). View the list of classes taught by a lecturer in a semester.

```
use teaching_system;
select Class.`semester`, Class.`lecturer_id`, Class.`course_id`,
    Class.`group`, Class.`no_student`
from Lecturer
inner join Class
on Lecturer.`id` = Class.`lecturer_id`
order by Class.`semester` desc, Lecturer.`id` asc;
```

(ii.6). View the list of lecturers who will be in charge of a class in a semester.

```
use teaching_system;
select Class.`semester`, Class.`course_id`, Class.`group`, Lecturer.`id`,
    Lecturer.`name`, Lecturer.`email`, Lecturer.`phone`,
    Lecturer.`faculty_name`
from Lecturer
inner join Class
on Lecturer.`id` = Class.`lecturer_id`
order by Class.`semester` desc, Class.`course_id` asc, Class.`group` asc;
```

(ii.7). View the list of assigned textbooks for each course in a semester.

```
use teaching_system;
select Course.`semester`, Course.`id`, Textbook.`title`, Textbook.`topic`,
    Textbook.`publish_year`, Textbook.`author`, Textbook.`publisher_name`
from ((Textbook
inner join Course_use_Textbook
on Textbook.`title` = Course_use_Textbook.`textbook_title`)
inner join Course
on Course_use_Textbook.`course_id` = Course.`id`)
order by Course.`semester`desc , Course.`id` asc;
```

(ii.8). View the list of students who have enrolled for each class in a semester.

```
use teaching_system;
select Student.`id`, Student.`name`, Student.`faculty_name`, Student.`phone`,
    Student.`grade`, Student.`status`
from ((Student
inner join Student_join_Class
on Student.`id` = Student_join_Class.`student_id`)
inner join Class
on Student_join_Class.`class_group` = Class.`group`)
order by Class.`semester` desc, Class.`course_id` asc, Class.`group` asc;
```

(ii.9). View the total number of students registered for a semester.

```
use teaching_system;
select count(`id`)
from ((Student
inner join Student_join_Class
on Student.`id` = Student_join_Class.`student_id`)
inner join Class
on Student_join_Class.`class_group` = Class.`group`)
group by Class.`semester`, Student.`id`;
```

(ii.10). View the total number of classes opened in a semester.

```
use teaching_system;
select count(`group`)
from Class
group by `semester`, `course_id`, `group`;
```

(ii.11). View which courses have the most lecturers in charge during the course of a semester.

```
use teaching_system;
select `id`, `name`, `no_credit`, `faculty_name`, max(no_lec) as max_lecturer
from (select *, count(`lecturer_id`) as no_lec
from (select *
from Course
inner join Class on Course.`id` = Class.`course_id`
group by Class.`semester`, Course.`id`, Class.`lecturer_id`) as temp1
group by temp2.`id`) as temp2;
```

(ii.12). View the average number of students enrolled in a course over the previous three years for a semester.

```
use teaching_system;
select count_student / count_course
from (select *, count_student, count(Course.`id`) as count_course
from (select *, count(Student.`id`) as count_student
from ((Student
```

```
inner join Student_enroll_Course
on Student.`id` = Student_enroll_Course.`student_id`)
inner join Course
on Student_enroll_Course.`course_id` = Course.`id`)
where (extract(year from current_date) % 100 -
((select cast(`semester` as unsigned) from Course) -
(select cast(`semester` as unsigned) from Course) % 10)) / 10 < 3)
as temp1
group by Course.`semester`, Course.`id`) as temp2;
```

7.3 Lecturer

(iii.1). Cập nhật giáo trình chính cho môn học do mình phụ trách.

```
delimiter //
create procedure update textbook(`cID` int,`textbook_title`varchar(255))
begin
update textbook set `title`=`textbook_title` where(`course_id` = `cID`)
return 1;
end //
delimiter ;
```

(iii.2). Xem danh sách lớp học của mỗi môn học do mình phụ trách ở một học kỳ.

```
use teaching_system;
select Class.`course_id`, Class.`group`,
       Class.`no_student`
from Course
inner join Class on Course.`id` = Class.`course_id`
where `semester` = `211`
group by Course.`id`
```

(iii.3). Xem danh sách sinh viên của mỗi lớp học do mình phụ trách ở một học kỳ.

```
use teaching_system;
select Student.`id`, Student.`name`, Student.`faculty_name`, Student.`phone`,
       Student.`email`
```

```
from Student
inner join Class on Student.`class_group` = Class.`group`
where `semester` = `211`
```

(iii.4). Xem danh sách môn học và giáo trình chính cho mỗi môn học do mình phụ trách ở một học kỳ.

```
delimiter //
use teaching_system;
select Textbook.`id`, Textbook.`title`, Textbook.`topic`, Textbook.`publish_year`,
       Textbook.`edition`, Textbook.`language`, Textbook.`publisher_name`,
           Course.`id`, Course.`name`, Course.`Faculty`, Course.`no_credit`
from (((Textbook
inner join Course_use_Textbook
on Textbook.`id` = Course_use_Textbook.`textbook_id`)
inner join Course
on Course_use_Textbook.`course_id` = Course.`id`)
inner join Lecturer
where `semester` = `211`
```

(iii.5). Xem tổng số sinh viên của mỗi lớp học do mình phụ trách ở một học kỳ.

```
use teaching_system;
select count(`id`)
from Student
inner join Class on Student.`class_group` = Class.`group`
where `semester` = `211`
group by `id`;
```

(iii.6). Xem số lớp học do mình phụ trách ở mỗi học kỳ trong 3 năm liên tiếp gần đây nhất.

```
select SUM(`group`) as number_of_class
from Class
inner join Lecturer_teach_Class
          on Class.`lecturer_id` = Lecturer_teach_Class.`lecturer_id`
          group by Lecturer_teach_Class.Semester
          where date > DATEADD(year,-3,GETDATE());
```

(iii.7). Xem 5 lớp học có số sinh viên cao nhất mà giảng viên từng phụ trách.

```
select Class.`no_student` as number_of student
from Class
inner join Lecturer_teach_Class
          on Class.`lecturer_id` = Lecturer_teach_Class.`lecturer_id`
          group by Lecturer_teach_Class.`class_group`
          order by (`no_student`) DESC
limit 5;
```

(iii.8). Xem 5 học kỳ có số lớp nhiều nhất mà giảng viên từng phụ trách.

```
select SUM(`group`) as number_of_class
from Class
inner join Lecturer_teach_Class
          on Class.`lecturer_id` = Lecturer_teach_Class.`lecturer_id`
          group by Lecturer_teach_Class.Semester
          order by SUM(`class_group`) DESC
limit 5;
```

7.4 Student

(iv.1). Đăng ký môn học ở học kỳ được đăng ký.

```
select *
from `course`;
```

(iv.2). Xem danh sách môn học, lớp học, và các giảng viên phụ trách cho mỗi lớp của mỗi môn học ở học kỳ được đăng ký.

```
select c1.`Group`, c2.`id`, l1.`name`
from `class` c1
inner join `course` c2
on c1.`course_id` = c2.`id`
inner join `lecturer` l1
on c1.`lecturer_id` = l1.`id`
```

```
order by c1.`Group`;
order by c1.`Group`;
```

(iv.3). Xem danh sách môn học và giáo trình chính cho mỗi môn học mà mình đăng ký ở một học kỳ.

```
select c2.`id`, t1.`TextBook_Title`
from `class` c1
inner join `course` c2
on c1.`Course_id` = c2.`id`
inner join `course_use_textbook` t1
on c2.`id` = t1.`Course_id`
where c1.`semester` = '211';
```

(iv.4). Xem danh sách lớp học của mỗi môn học mà mình đăng ký ở một học kỳ.

```
select c1.`Group`, c2.`id`
from `class` c1
inner join `course` c2
on c1.`Course_id` = c2.`id`
where c1.`semester` = '211'
group by  c2.`id`;
```

(iv.5). Xem danh sách lớp học của mỗi môn học mà mình đăng ký có nhiều hơn 1 giảng viên phụ trách ở một học kỳ.

```
select ltc.class_group, COUNT(ltc.lecturer_id) as number_of_lectures
from lecturer_teach_class ltc
group by lecturer_id
having  COUNT(lecturer_id) > 1
order by lecturer_id;
```

(iv.6). Xem tổng số tín chỉ đã đăng ký được ở một học kỳ.

```
select sec.`student_id`, SUM(c1.`no_credit`) as number_of_credits
From `course` c1
```

```
inner join student_enroll_course sec
on c1.`id` = sec.`course_id`
group by sec.`student_id`
order by SUM(c1.`no_credit`) DESC;
```

(iv.7). Xem tổng số môn học đã đăng ký được ở một học kỳ.

```
select sec.`student_id`, COUNT(c1.`id`) as number_of_courses
From `course` c1
inner join student_enroll_course sec
on c1.`id` = sec.`course_id`
group by sec.`student_id`
order by COUNT(c1.`id`) DESC;
```

(iv.8). Xem 3 học kỳ có số tổng số tín chỉ cao nhất mà mình đã từng đăng ký.

```
select SUM(c1.`no_credit`) as number_of_credits
From `course` c1
inner join student_enroll_course sec
on c1.`id` = sec.`course_id`
group by c1.semester
order by SUM(c1.`no_credit`) DESC
limit 3;
```

7.5 Trigger and Stored Procedure

7.5.1 Trigger

We basically used a few triggers to let us automatically maintain a portion of the database.

Trigger for checking the studying status ò students.

```
delimiter //
drop trigger if exists Student_status;
create trigger Student_status before insert on Student_enroll_Course
for each row
begin
```

```
declare Sstatus varchar(100);
set Sstatus = (select Student.`status` from Student
    where Student.`id` = NEW.`student_id`);
if Sstatus <> "Active"
then
signal sqlstate '45000'
set message_text='Student must have the Active status to register Courses!';
end if;
end//
delimiter ;
```

Trigger for checking the total number of credits that students register in a semester.

```
delimiter //
drop trigger if exists Student_credit;
create trigger Student_credit after update on Student_enroll_Course
for each row
begin
declare Scredit int;
set Scredit = (select Student.`no_credit` from Student
    where Student.`id` = NEW.`student_id`);
if Scredit > 18
then
signal sqlstate '45000'
set message_text='Student can't register more than 18 credits each semester!';
end if;
end//
delimiter ;
```

Trigger for checking the publish year of the main textbookstextbooks.

```
delimiter //
drop trigger if exists Textbook_publish_year;
create trigger Textbook_publish_year before insert on Lecturer_assign_Textbook
for each row
begin
declare Pyear int;
set Pyear = (select Textbook.`publish_year` from Textbook
```

```
    where Textbook.`title` = NEW.`textbook_title`);
if (select extract(year from current_date)) - Pyear >= 10
then
signal sqlstate '45000'
set message_text='Main Textbook must have publish year less than 10 years!';
end if;
end//
delimiter ;
```

7.5.2 Stored Procedure

A stored procedure is a group of pre-compiled Structured Query Languages (SQL) that may be shared and reused by multiple programs. It has the ability to read and alter data in a database. In MySQL, stored procedures may be called using the CALL or EXEC statements. Because functions may be called from a select statement, but stored procedures cannot, stored procedures cannot be invoked from a function. Stored procedures may accept any type of parameter, both in and out.

- **IN** mode:

  A variable passed as mode IN is always read-only. A variable using IN mode can be read and used by the procedure/function but can not be changed and it cannot be the receiver of an assignment operation. Internal to the scope of the procedure or function, variables pass using IN mode can be considered a constant. The IN mode is the default mode to pass a variable, however it is recommended for maintainability reasons to always define the variable passing mode when you define the variable. Variables passed IN can also be assigned a default value as discussed above.

- **OUT** mode:

  A variable passed in OUT mode is used to pass information back from the procedure to the calling program. It is a write-only variable and has no value until the block assigns it a value. Internally, an OUT variable is created and not initialized when the procedure is called. When the procedure ends, the variable value (upon ending) is copied to the variable passed in the call. As such, a variable passed in OUT mode can not be assigned a default value nor can it be read inside the procedure. Because the variable value is copied back to the passed variable when the procedure terminates, the calling code can not pass an OUT variable a literal value. If the procedure raises an exception that is not caught, it will result in the OUT variable not being copied when the procedure terminates.

- **IN OUT** mode:

A variable passed in INOUT mode has characteristics of both the IN and the OUT mode. The variable value is passed in and can be read by the procedure. The procedure can also change the value and it will be copied back to the passed variable when the procedure completes. Like a variable passed in OUT mode, an INOUT variable can not have a default value and can not be passed as a literal. If the procedure terminates abnormally (as in an exception) the INOUT variable will not be copied back to the variable passed in. [1]

Because there are few variables and variations in the query to care about, procedures are used to implement and manage the above queries.

These are our procedures that contain in this database:

- Asm2_AAO_Proc.sql
- Asm2_Faculty_Proc.sql
- Asm2_Student_Proc.sql
- Asm2_Lecturer_Proc.sql

7.6 Indexing

7.6.1 Theory

Indexing is a data structure technique for quickly retrieving entries from database files using some attributes that have been indexed. In database systems, indexing is comparable to indexing in books. The indexing properties are used to define the indexing. The types of indexing are as follows:[4]

- **Primary Index** - Primary index is defined on an ordered data file. The data file is ordered on a key field. The key field is generally the primary key of the relation.
- **Secondary Index** - Secondary index may be generated from a field which is a candidate key and has a unique value in every record, or a non-key with duplicate values
- **Clustering Index** - Clustering index is defined on an ordered data file. The data file is ordered on a non-key field.

There are also two types of ordered indexing

- Dense Index
- Sparse Index

7.6.2 Dense Index

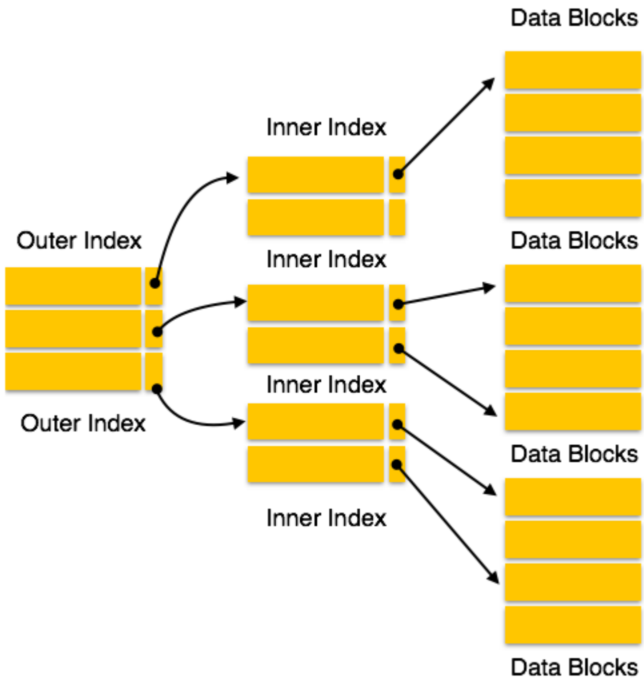
Every search key value in the database has an index record in dense index. This speeds up searches, but it takes up more room to store index information. The value of the search key is stored in index records, along with a pointer to the actual record on the disk.

7.6.3 Spare Index

For each search key, index records are not created in a sparse index. A search key and a physical pointer to the data on the disk are both contained in an index record. To find a record, we start with an index record and work our way to the data's real location. If the data we're seeking for isn't located by following the index, the system performs a sequential search until the data we're looking for is found.

7.6.4 Multilevel Index

Index records comprise search-key values and data pointers. Multilevel index is stored on the disk along with the actual database files. As the size of the database grows, so does the size of the indices. There is an immense need to keep the index records in the main memory so as to speed up the search operations. If single-level index is used, then a large size index cannot be kept in memory which leads to multiple disk accesses. [5]



7.6.5 Indexing in MySql

Btrees are used to hold most MySQL indexes (PRIMARY KEY, UNIQUE, INDEX, and FULLTEXT). Indexes for spatial data types employ R-trees MEMORY tables also provide hash indexes and FULLTEXT indexes in InnoDB use inverted lists. Indexing has a number of advantages.

- It helps in decreasing the total number of I/O operations required to retrieve that data by eliminating the necessity to access a database entry through an index structure. It also allows users to search for and retrieve data more quickly.
- We can also modify and create more indexing to maximize the performance and minimize the cost of retrieving data.

Overuse of duplicate indexing, on the other hand, has several disadvantages. The most crucial is the memory space required to keep the index files, as the amount of data grows exponentially and requires scientific management.

7.6.6 How to implement Indexing in MySql

```
CREATE UNIQUE INDEX index_name ON table_name ( column1, column2,...);
```

```sql
-- Implementation for Indexing
ALTER TABLE Student
ADD INDEX student_idx(`id`);
ALTER TABLE Lecturer
ADD INDEX lecturer_idx(`id`);
ALTER TABLE Publisher
ADD INDEX publisher_idx(`name`);
ALTER TABLE Author
ADD INDEX author_idx(`name`);
```

8 Normalization

8.1 Theory

Normalization is a database design technique that reduces data redundancy and eliminates undesirable characteristics like Insertion, Update and Deletion Anomalies. Normalization rules divides larger tables into smaller tables and links them using relationships. The purpose of Normalisation in SQL is to eliminate redundant (repetitive) data and ensure data is stored logically.

The inventor of the relational model Edgar Codd proposed the theory of normalization of data with the introduction of the First Normal Form, and he continued to extend theory with Second and Third Normal Form. Later he joined Raymond F. Boyce to develop the theory of Boyce-Codd Normal Form [2]

List of Normal Form in SQL:

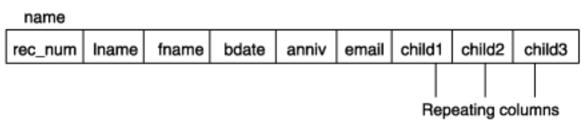
- 1NF (First Normal Form)
- 2NF (Second Normal Form)
- 3NF (Third Normal Form)
- BCNF (Boyce-Codd Normal Form)
- 4NF (Fourth Normal Form)
- 5NF (Fifth Normal Form)
- 6NF (Sixth Normal Form)

In this Assignment, will discuss three third Normal Form

8.2 First Normal Form

8.2.1 Theory discussion

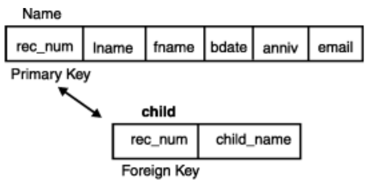
An entity is in the first normal form if it contains no repeating groups. In relational terms, a table is in the first normal form if it contains no repeating columns. Repeating columns make your data less flexible, waste disk space, and make it more difficult to search for data. In the following telephone directory example, the name table contains repeating columns, child1, child2, and child3.



Conditions for First Normal Form

- First normal form (1NF) is considered to be part of the formal definition of a relation in the basic (flat) relational model; historically, it was defined to disallow multi-valued attributes, composite attributes, and their combinations.
- It states that the domain of an attribute must include only atomic (simple, indivisible) values and that the value of any attribute in a tuple must be a single value from the domain of that attribute.
- Hence, 1NF disallows having a set of values, a tuple of values, or a combination of both as an attribute value for a single tuple. In other words, 1NF disallows relations within relations or relations as attribute values within tuples. The only attribute values permitted by 1NF are single atomic (or indivisible) values.

To eliminate the repeating columns and bring the table to the first normal form, separate the table into two tables. Put the repeating columns into one of the tables. The association between the two tables is established with a primary-key and foreign-key. combination.



8.2.2 Apply to Database

In our Database, there is a composition attribute name **Contact** in the **Lecturer** relation, by ER Mapping to database schema, we already eliminate that, so our Database satisfy the 1st NF

8.3 Second Nornal Form

8.3.1 Theory discussion

Second Normal Form (2NF) is based on the concept of full functional dependency. Second Normal Form applies to relations with composite keys, that is, relations with a primary key composed of two or more attributes. A relation with a single-attribute primary key is automatically in at least 2NF. A relation that is not in 2NF may suffer from the update anomalies.[6]

To be in second normal form, a relation must be in first normal form and relation must not contain any partial dependency. A relation is in 2NF if it has No Partial Dependency, i.e., no non-prime attribute (attributes which are not part of any candidate key) is dependent on any proper subset of any candidate key of the table.

For an example, We have a table with 3 attribute **STUD_NO, COURSE_NO COURSE_FEE**

STUD_NO	COURSE_NO	COURSE_FEE
1	C1	1000
2	C2	1500
1	C4	2000
4	C3	1000
4	C1	1000
2	C5	2000

- COURSE_FEE cannot alone decide the value of COURSE_NO or STUD_NO
- COURSE_FEE together with STUD_NO cannot decide the value of COURSE_NO
- COURSE_FEE together with COURSE_NO cannot decide the value of STUD_NO

Hence, COURSE_FEE would be a non-prime attribute, as it does not belong to the one only candidate key **STUD_NO, COURSE_NO** ; But, **COURSE_NO -> COURSE_FEE**, i.e., **COURSE_FEE** is dependent on COURSE_NO, which is a proper subset of the candidate key. Non-prime attribute **COURSE_FEE** is dependent on a proper subset of the candidate key, which is a partial dependency and so this relation is not in 2NF.

To convert the above relation to 2NF, we need to split the table into two tables such as :

- Table 1: STUD_NO, COURSE_NO
- Table 2: COURSE_NO, COURSE_FEE

Table 1		Table 2	
STUD_NO	COURSE_NO	COURSE_NO	COURSE_FEE
1	C1	C1	1000
2	C2	C2	1500
1	C4	C3	1000
4	C3	C4	2000
4	C1	C5	2000
2	C5		

8.3.2 Apply to Database

In our database, we have two case of primary key, these are atomic primary key and composite primary key, With our table that has atomic primary key, there is no partial dependency in our relations, for an example: **Student**:

- **'id'** -> **'grade'**
- **'id'** -> **'name'**
- **'id'** -> **'status'**
- **'id'** -> **'no_credit'**
- **'id'** -> **'phone'**
- **'id'** -> **'faculty_name'**

'id' Attribute is a Supper Key and also a Candidate Key, there is no Partial Dependency because the proper subset of **'id'** is $\emptyset$ so it doesn't determine any non-prime attribute, also in this relation we don't have any non-prime attribute.

About the Composite primary key relation, these are table which purpose is to connect between two M:N relationship. Each of it only has 2 attributes that is a foreign key and referencing to 2 relations whose relationship is M:N. There is no non-prime attribute so we can made a conclusion that it satisfy the 2NF

8.4 Third Normal Form

8.4.1 Theory discussion

Although Second Normal Form (2NF) relations have less redundancy than those in 1NF, they may still suffer from update anomalies. If we update only one tuple and not the other, the database would be in an inconsistent state. This update anomaly is caused by a transitive dependency. We need to remove such dependencies by progressing to Third Normal Form (3NF).

- A relation is in third normal form, if there is no transitive dependency for non-prime attributes as well as it is in second normal form.
- A relation is in 3NF if at least one of the following condition holds in every non-trivial function dependency X –> Y:
  1. X is a super key.
  2. Y is a prime attribute (each element of Y is part of some candidate key).

In other words, A relation that is in First and Second Normal Form and in which no non-primary-key attribute is transitively dependent on the primary key, then it is in Third Normal Form (3NF).

Note – If A->B and B->C are two FDs then A->C is called transitive dependency.

The normalization of 2NF relations to 3NF involves the removal of transitive dependencies. If a transitive dependency exists, we remove the transitively dependent attribute(s) from the relation by placing the attribute(s) in a new relation along with a copy of the determinant.

Consider the examples given below.

Example-1:In relation STUDENT given in Table 4,

STUD_NO	STUD_NAME	STUD_STATE	STUD_COUNTRY	STUD_AGE
1	RAM	HARYANA	INDIA	20
2	RAM	PUNJAB	INDIA	19
3	SURESH	PUNJAB	INDIA	21

Table 4

FD set:
{STUD_NO -> STUD_NAME, STUD_NO -> STUD_STATE, STUD_STATE -> STUD_COUNTRY, STUD_NO -> STUD_AGE}
Candidate Key:
{STUD_NO}
For this relation in table 4, **STUD_NO -> STUD_STATE** and **STUD_STATE -> STUD_COUNTRY** are true. So **STUD_COUNTRY** is transitively dependent on **STUD_NO**. It violates the third normal form. To convert it in third normal form, we will decompose the relation STUDENT **(STUD_NO, STUD_NAME, STUD_PHONE, STUD_STATE, STUD_COUNTRY_STUD_AGE)** as:
STUDENT (STUD_NO, STUD_NAME, STUD_PHONE, STUD_STATE, STUD_AGE) STATE_COUNTRY (STATE, COUNTRY) [7]

8.4.2 Apply to database

In our current project: our design is not really complicated. 'Lecturer', 'Class' have two or more foreign keys come from the same relation. In order to solve it we have to separate them into different table. For an example: **Lecturer**:

- **'id'** -> **'phone'** -> **'email'**
- **'id'** -> **'email'** -> **'phone'**

'phone' and 'email' of 'Lecturer' are relation have transitive functional dependencies on the primary key 'id'. So we have to seperate them into different table.

9 Database Security

9.1 Definition of Role-based access control(RBAC)

Role-based access control (RBAC) restricts network access based on a person's role within an organization and has become one of the main methods for advanced access control. The roles in RBAC refer to the levels of access that employees have to the network.

For example, employees are only allowed to access the information necessary to effectively perform their job duties. Access can be based on several factors, such as authority, responsibility, and job competency. In addition, access to computer resources can be limited to specific tasks such as the ability to view, create, or modify a file.

As a result, lower-level employees usually do not have access to sensitive data if they do not need it to fulfill their responsibilities. This is especially helpful if you have many employees and use third-parties and contractors that make it difficult to closely monitor network access. Using RBAC will help in securing company's sensitive data and important applications. [8]

Benefits of RBAC:

- **Reducing administrative work and IT support.** With RBAC, you can reduce the need for paperwork and password changes when an employee is hired or changes their role. Instead, you can use RBAC to add and switch roles quickly and implement them globally across operating systems, platforms and applications. It also reduces the potential for error when assigning user permissions. This reduction in time spent on administrative tasks is just one of several economic benefits of RBAC. RBAC also helps to more easily integrate third-party users into your network by giving them pre-defined roles.

- **Maximizing operational efficiency.** RBAC offers a streamlined approach that is logical in definition. Instead of trying to administer lower-level access control, all the roles can be aligned with the organizational structure of the business and users can do their jobs more efficiently and autonomously.

- **Improving compliance.** All organizations are subject to federal, state and local regulations. With an RBAC system in place, companies can more easily meet statutory and regulatory requirements for privacy and confidentiality as IT departments and executives have the ability to manage how data is being accessed and used. This is especially significant for health care and financial institutions, which manage lots of sensitive data such as PHI and PCI data.

9.2 Apply to database

According to requirements, we created 3 main roles: 'TEACHER', 'STUDENT' and 'ACADEMIC STAFF'.

```
DROP ROLE IF EXISTS 'teacher';
CREATE ROLE 'teacher';

DROP ROLE IF EXISTS 'student';
CREATE ROLE 'student';

DROP ROLE IF EXISTS 'academic staff';
CREATE ROLE 'academic staff';
```

With each role, they are assigned to their own privileges.
For student, they are allowed to see information on textbook, class, course and also can update, insert their information(student).

```
#GRANT STUDENT
GRANT SELECT on textbook to 'student';
GRANT SELECT on class to 'student';
GRANT SELECT on course to 'student';
GRANT SELECT on course_use_textbook to 'student';
GRANT SELECT on author_write_textbook to 'student';
GRANT SELECT, UPDATE, INSERT on student to 'student';
```

For teacher, they can see information of textbook, author and courses. They also can modify information on lecturer, asssigning textbook, student-enroll-course, student-join-class and falcuty.

```
#GRANT TEACHER
GRANT SELECT on author to 'teacher';
GRANT SELECT on author_write_textbook to 'teacher';
GRANT SELECT on course to 'teacher';
GRANT SELECT, UPDATE on class to 'teacher';
GRANT SELECT, UPDATE, INSERT on lecturer to 'teacher';
GRANT SELECT, UPDATE, INSERT on lecturer_assign_textbook to 'teacher';
GRANT SELECT, UPDATE, INSERT on lecturer_teach_class to 'teacher';
```

```
GRANT SELECT, UPDATE, INSERT on student_enroll_course to 'teacher';
GRANT SELECT, UPDATE, INSERT on student_join_class to 'teacher';
GRANT SELECT, UPDATE, INSERT on faculty to 'teacher';
```

For academic staff, they have all permissions on this database as they are the one who control and manage the system.

```
#GRANT STAFF
GRANT ALL on teaching_system to 'academic staff';
```

Finally, we created user accounts and grant their corresponding role to each user.

```
CREATE USER 'student'@'localhost' IDENTIFIED BY 'password';
GRANT 'student' to 'student'@'localhost';

CREATE USER 'teacher'@'localhost' IDENTIFIED BY 'password';
GRANT 'teacher' to 'teacher'@'localhost';

CREATE USER 'academic_staff'@'localhost' IDENTIFIED BY 'password';
GRANT 'academic staff' to 'academic_staff'@'localhost';
```

10 Application

10.1 Overview

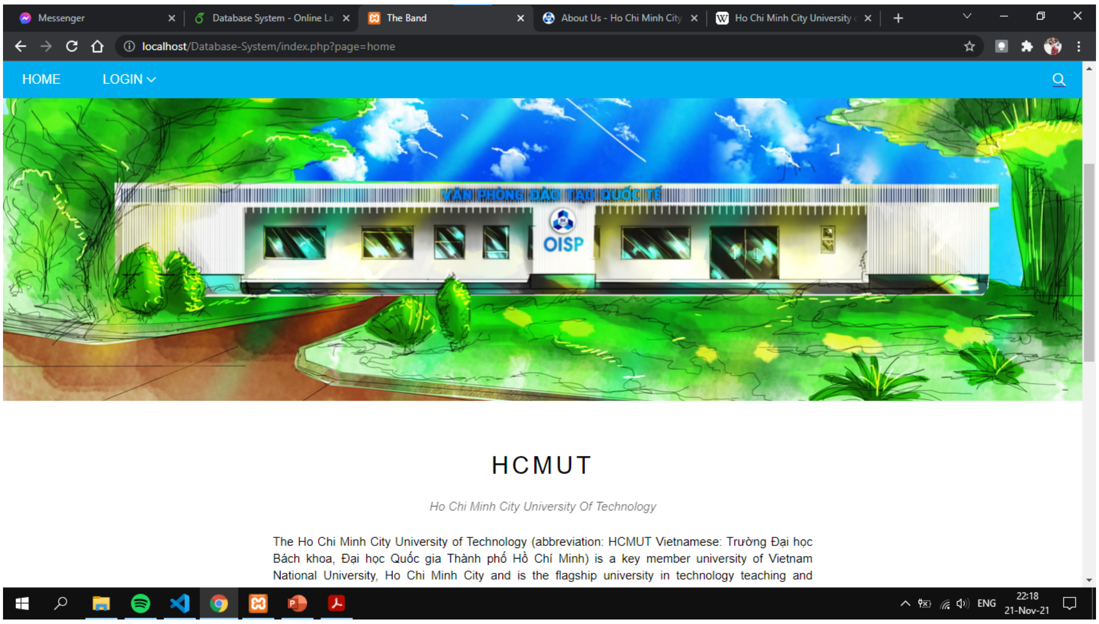
In this Assignment, we will use HTML, CSS, JS for the front-end development. And pure PHP with MySQL Database to do the Back-end

The front and Back will communicate with each other by using HTTP protocol.

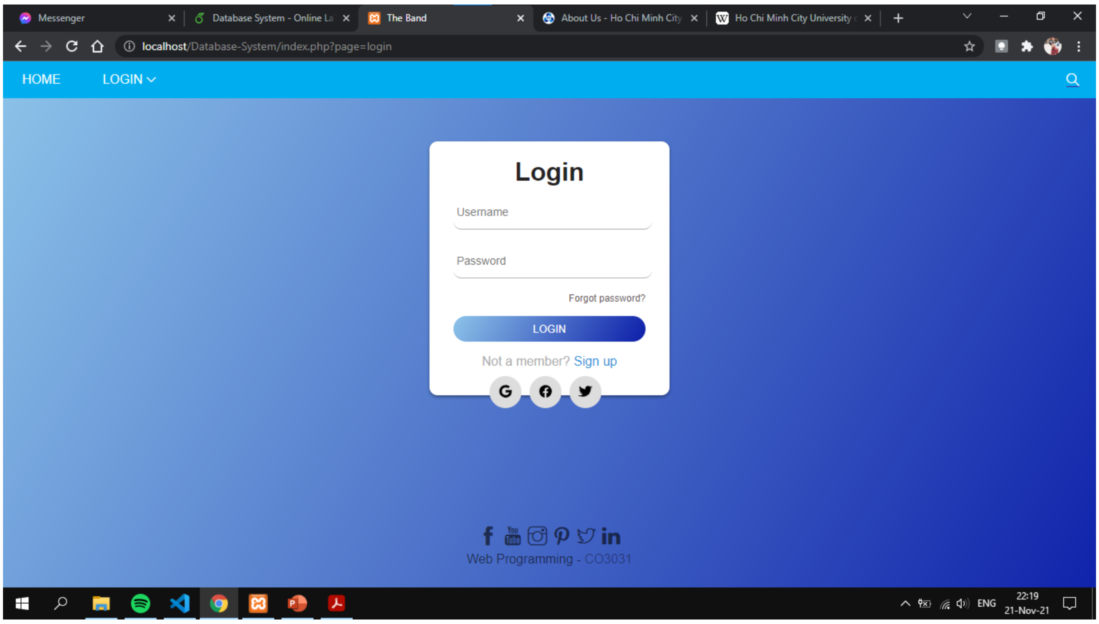
HTTP requests arrive from the browser at the backend. Those requests may contain data in the HTTP headers or request body. The intent may be to request new data or to transmit user-created data to the backend.

10.2 The Application

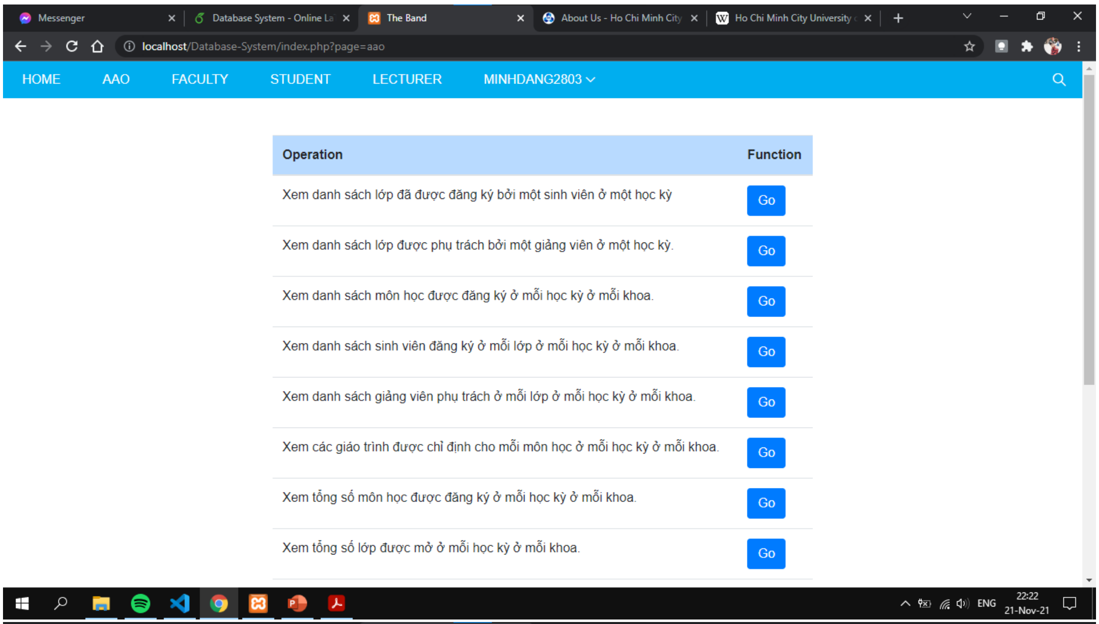
10.2.1 Home Page

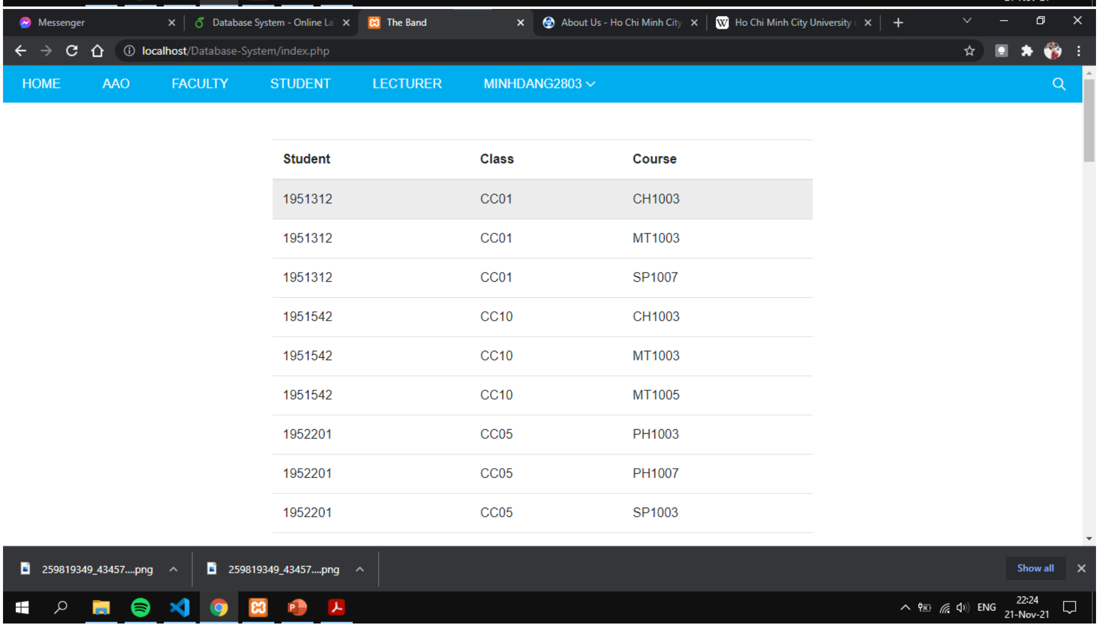


10.2.2 Login to system



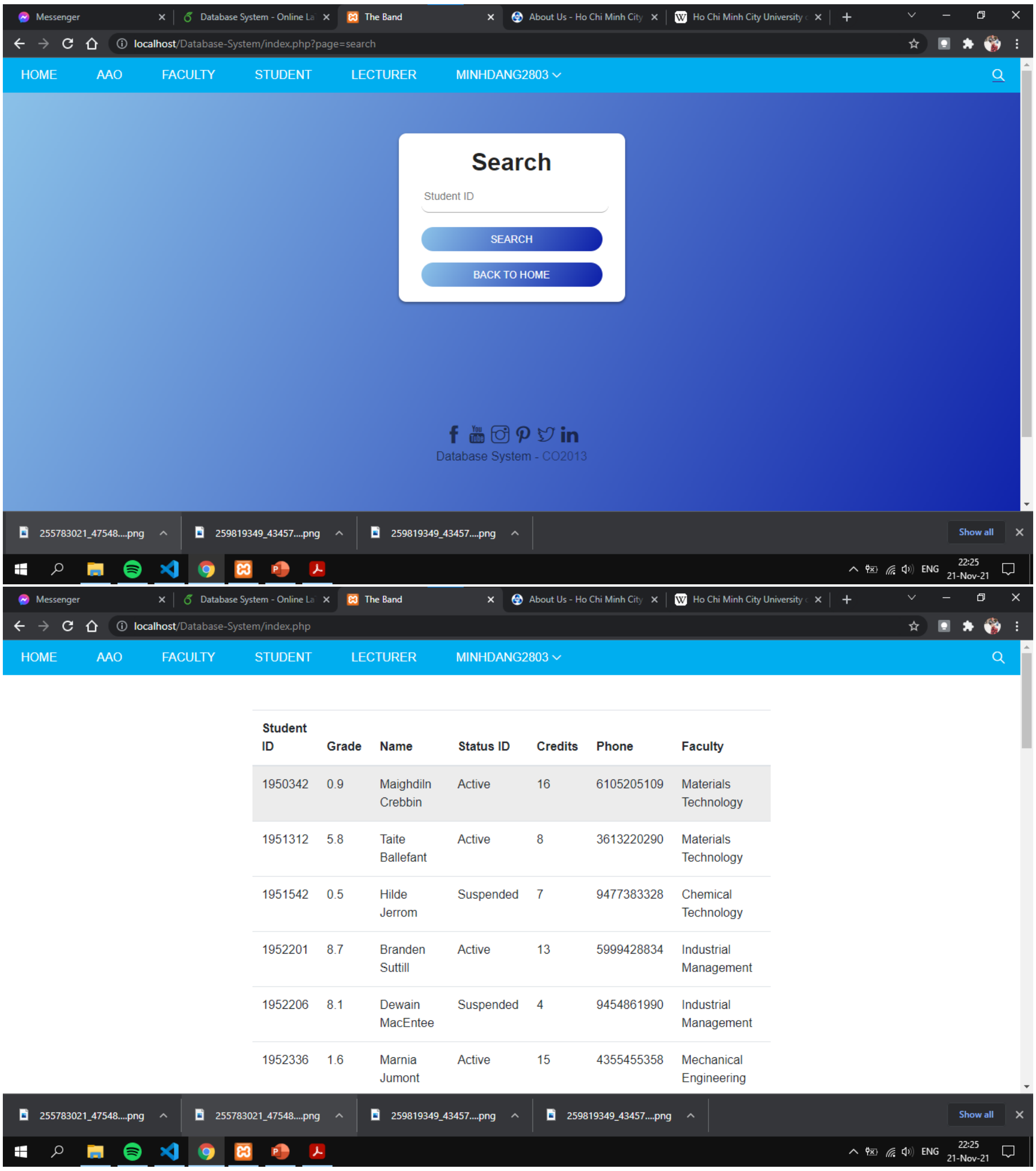
10.3 Searching by using Stored Procedure



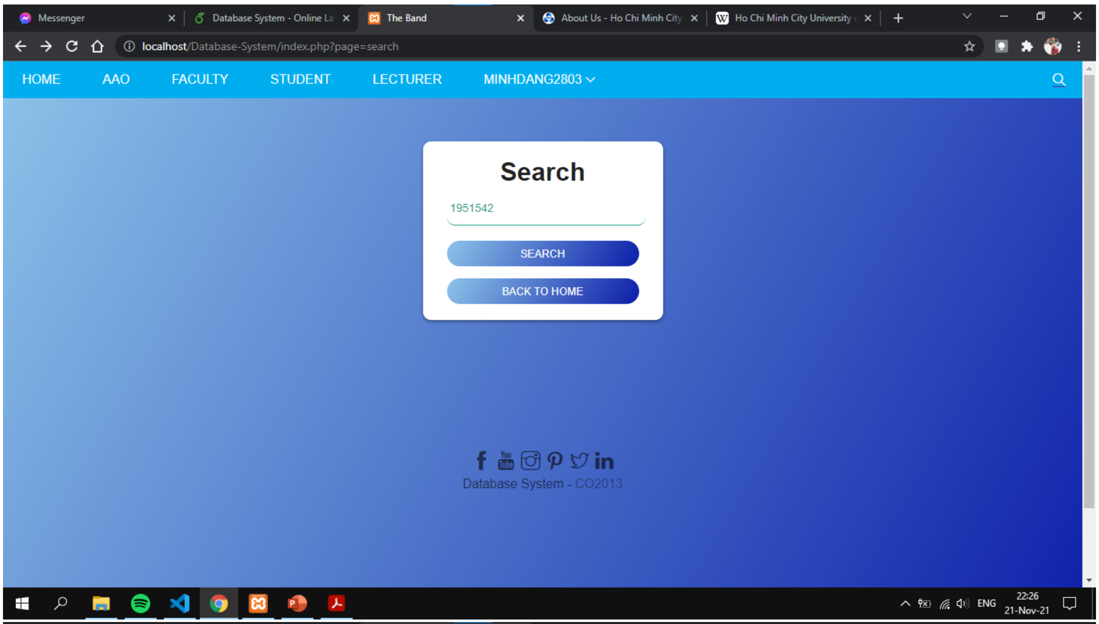


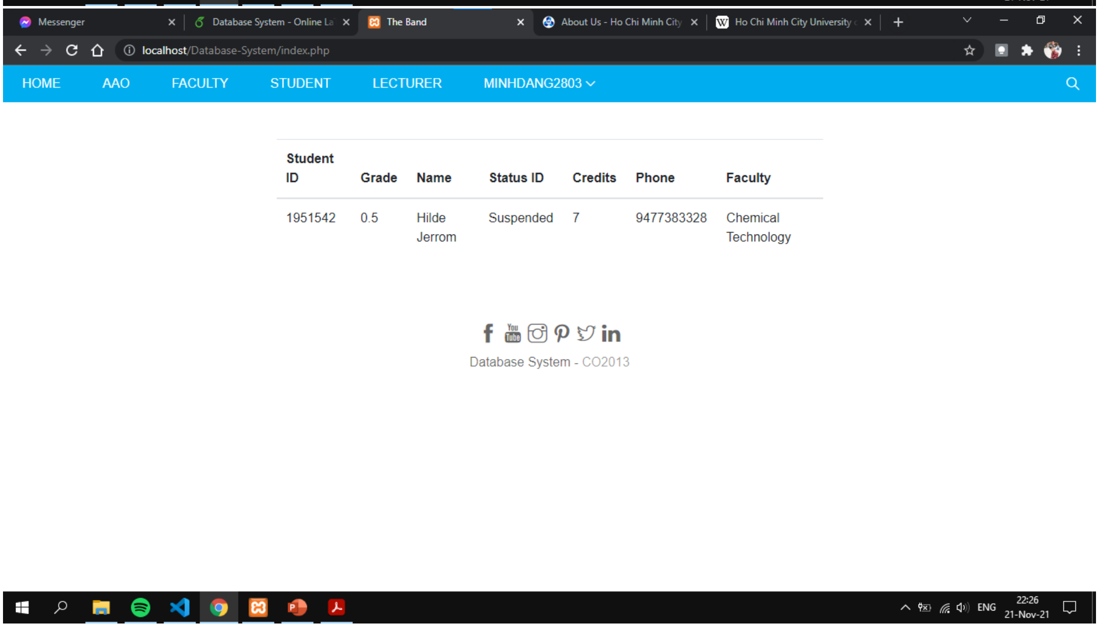
10.4 Searching by forms

With blank form:



With filled form:


localhost/Database-System/index.php?page=search
HOME
AAO
FACULTY
STUDENT
LECTURER
MINHDANG2803
Search
1951542
SEARCH
BACK TO HOME
Database System - CO2013


localhost/Database-System/index.php
HOME
AAO
FACULTY
STUDENT
LECTURER
MINHDANG2803
Student ID
Grade
Name
Status ID
Credits
Phone
Faculty
1951542
0.5
Hilde Jerrom
Suspended
7
9477383328
Chemical Technology
Database System - CO2013

References

[1] Donald Burleson. *Oracle Database Tips by Donald Burleson*. URL: http://www.dba-oracle.com/t_easyoracle_pl_sql_in_out_inout_modes.htm.

[2] IBM Documentataion. *First Normal Form (1NF)*. URL: https://www.ibm.com/docs/en/informix-servers/14.10?topic=model-first-normal-form.

[3] Ramez Elmasri and S Navathe. "Fundamentals of Database Systems Addison Wesley". In: *Reading, MA* (2003).

[4] GeeksforGeeks. *Indexing in Databases*. URL: https://www.geeksforgeeks.org/indexing-in-databases-set-1/.

[5] tutorialspoint. *DBMS - Indexing*. URL: https://www.tutorialspoint.com/dbms/dbms_indexing.htm.

[6] Mithlesh Upadhyay. *Second Normal Form (2NF)*. URL: https://www.geeksforgeeks.org/second-normal-form-2nf/.

[7] Carlo Zaniolo. *Third Normal Form (3NF)*. URL: https://www.geeksforgeeks.org/third-normal-form-3nf/.

[8] Ellen Zhang. *What is Role-Based Access Control (RBAC)? Examples, Benefits, and More*. URL: https://digitalguardian.com/blog/what-role-based-access-control-rbac-examples-benefits-and-more.