



**Islington college**  
(इस्लिङ्टन कलेज)

**CC5051NA Databases Systems**

**50% Individual Coursework**

**2020-21 Autumn**

**Student Name: Bijay Bharati**

**Group: L2C4**

**London Met ID: 19030824**

**College ID: NP01CP4A190041**

**Assignment Due Date: 20<sup>th</sup> December 2020**

**Assignment Submission Date: 20<sup>th</sup> December 2020**

I confirm that I understand my coursework needs to be submitted online via Google Classroom under the relevant module page before the deadline for my assignment to be accepted and marked. I am fully aware that late submissions will be treated as non-submission and a mark of zero will be awarded.

## Table of Contents

Chapter 1: Introduction .....	1
1.1 Introduction of the college .....	1
1.2 Current business activities .....	1
1.3 Business Rules .....	1
1.4 Identification of entities and attributes.....	3
1.4.1 List of created objects- Entities and Attributes .....	3
1.4.2 Initial ERD .....	4
Chapter 2: Normalization.....	5
2.1 Assumptions.....	5
2.2 Normalization .....	5
2.2.1 UNF (Un-Normalized Form) .....	6
2.2.2 1NF (First Normal Form).....	7
2.2.3 2NF (Second Normal Form) .....	8
2.2.4 3NF (Third Normal Form) .....	11
2.3 ERD after normalization .....	14
Chapter 3: Implementation .....	15
3.1 Creation of tables .....	15
3.1.1 Course table .....	15
3.1.2 Class Table.....	16
3.1.3 Module Table .....	17
3.1.4 Contact table .....	18
3.1.5 Address Table .....	19
3.1.6 Student table.....	20
3.1.7 Instructor table .....	22
3.2 Populating data into the tables .....	24
3.2.1 Course Table .....	25
3.2.2 Class Table.....	27
3.2.3 Module Table .....	29
3.2.4 Contact Table .....	31
3.2.5 Address Table .....	33
3.2.6 Student Table .....	35
3.2.7 Instructor Table.....	37
Chapter 4: Information and Transaction Queries.....	39

4.1 List all the students with all their addresses with their phone numbers. ....	39
4.1.1 Query.....	39
4.1.2 Screenshot.....	39
4.2 List all the modules which are taught by more than one instructor. ....	40
4.2.1 Query.....	40
4.2.2 Screenshot.....	40
4.3 List the name of all the instructors whose name contains 's' and salary is above 50,000. ....	41
4.3.1 Query.....	41
4.3.2 Screenshot.....	41
4.4 List the modules comes under the 'Multimedia' specification. ....	42
4.4.1 Query.....	42
4.4.2 Screenshot.....	42
4.5 List the name of the head of modules with the list of his phone number.....	43
4.5.1 Query.....	43
4.5.2 Screenshot.....	43
4.6 List all Students who have enrolled in 'networking' specifications. ....	44
4.6.1 Query.....	44
4.6.2 Screenshot.....	44
4.7 List the fax number of the instructor who teaches the 'database' module.....	45
4.7.1 Query.....	45
4.7.2 Screenshot.....	45
4.8 List the specification falls under the BIT course. ....	46
4.8.1 Query.....	46
4.8.2 Screenshot.....	46
4.9 List all the modules taught in any one particular class. ....	47
4.9.1 Query.....	47
4.9.2 Screenshot.....	47
4.10 List all the teachers with all their addresses who have 'a' at the end of their first name.....	48
4.10.1 Query.....	48
4.10.2 Screenshot.....	48
4.11 Show the students, course they enroll in and their fees. Reduce 10% of the fees if they are enrolled in a computing course. ....	49
4.11.1 Query.....	49
4.11.2 Screenshot.....	49

4.12 Place the default Number 1234567890 if the list of phone numbers to the location of the address is empty and give the column name as ‘Contact details. ....	50
4.12.1 Query.....	50
4.12.2 Screenshot .....	50
4.13 Show the name of all the students with the number of weeks since they have enrolled in the course .....	51
4.13.1 Query.....	51
4.13.1 Screenshot .....	51
4.14 Show the name of the instructors who got equal salary and work in the same.....	52
specification. ....	52
4.14.1 Query.....	52
4.14.2 Screenshot .....	53
4.15 List all the courses with the total number of students enrolled course name and the highest marks obtained.....	54
4.15.1.....	54
4.15.2 Screenshot .....	54
4.16 List all the instructors who are also a course leader.....	55
4.16.1 Query.....	55
4.16.2 Screenshot .....	55
Dump file creation, dropping tables.....	56
Screenshot for dump file creation .....	56
Queries to drop tables .....	56
Chapter 5: Conclusion.....	57
Chapter 6: References .....	58
References.....	58

## Table of Figures

Figure 1: Initial ERD .....	4
Figure 2: Normal Forms.....	5
Figure 3: ERD after normalization.....	14
Figure 4: Creating course table .....	15
Figure 5: DESC course .....	15
Figure 6: Creating class table.....	16
Figure 7: DESC class .....	16
Figure 8: Creating module table.....	17
Figure 9: DESC module.....	17
Figure 10: Creating contact table .....	18
Figure 11: DESC contact .....	18
Figure 12: Creating address table.....	19
Figure 13: DESC address.....	19
Figure 14: Creating student table .....	20
Figure 15: DESC student .....	21
Figure 16: Creating instructor table .....	22
Figure 17: DESC instructor .....	23
Figure 18: Inserting values in course table .....	25
Figure 19: Values in course table.....	26
Figure 20: Inserting values in class table .....	27
Figure 21: Values in course table.....	28
Figure 22: Inserting values in module table.....	30
Figure 23: Values in module table .....	30
Figure 24: Inserting values in contact table .....	32
Figure 25: Values in contact table.....	32
Figure 26: Inserting values in address table.....	34
Figure 27: Values in address table .....	34
Figure 28: Inserting values in student table .....	36
Figure 29: Values in student table.....	36
Figure 30: Inserting values in instructor table .....	38
Figure 31: Values in instructor table.....	38
Figure 32: Information query 1 .....	39
Figure 33: Information query 2.....	40
Figure 34: Seeing how many times a value appears .....	40
Figure 35: Information query 3.....	41
Figure 36: Information query 4.....	42
Figure 37: Information query 5.....	43
Figure 38: Information query 6.....	44
Figure 39: Information query 7.....	45
Figure 40: Information query 8.....	46
Figure 41: Information query 9.....	47
Figure 42: Information query 10.....	48
Figure 43: Transaction query 1 .....	49
Figure 44: Transaction query 2 .....	50
Figure 45: Transaction query 3 .....	51

Figure 46: Data in instructor table .....	53
Figure 47: Transaction query 4 .....	53
Figure 48: Transaction query 5 .....	54
Figure 49: Transaction query 6 .....	55
Figure 50: Creating dmp file .....	56
Figure 51: Dropping tables .....	56

## **Chapter 1: Introduction**

### **1.1 Introduction of the college**

Islington College is among the most reputable colleges in Nepal providing world-class IT and business academic qualifications. Established in 1996, Islington College has a long and proud history of producing industry ready graduates. Islington College provides high-quality overseas degree programs through partnership with established universities from Singapore and the UK. (Islington College, 2020)

### **1.2 Current business activities**

The college currently focuses on the following points.

1. College provides courses on multiple subjects.
2. Qualified instructors are appointed to teach the courses.
3. Courses can have many modules within them.
4. Students can enroll for any course that they want.
5. The college currently provides 7 courses Computing, Networking, Multimedia for BIT and Finance, Management, Marketing and Business for MBA courses.
6. Each module is taught within the college's own buildings in classes.

### **1.3 Business Rules**

A database is used by the college to keep its records, it follows the following rules:

1. Students and instructors need to provide information about their name, contact and address.
2. Phone number is a requirement for contact information, is permanent but email and fax can be optional and changeable.
3. All phone numbers are unique.
4. The college also keeps record of when a student enrolled in a course and their marks out of 100.

5. The college requires country, province, city, street and house number for the address, each student/instructor may provide 1 set of data, many people can have same address.
6. The address provided is also the mailing address.
7. The college also keeps track of the courses it provides.
8. The college provides 7 courses, each course either contains BIT or MBA in its name denoting specification.
9. Each course has a course leader.
10. A person cannot be the leader of more than 1 course.
11. Different courses do not have same modules.
12. Each module has a leader.
13. Course leaders cannot be module leaders.
14. Multiple people can teach the same module.
15. Course leaders and module leaders are also instructors i.e. they can teach. Each instructor id contains a letter at its start. CL for course leader and ML for module leader and I for every other instructor.
16. Students can only apply for one course at a time.
17. Students get enrolled in all the modules within a course.
18. Instructors cannot be students and students cannot be instructors within the same college.
19. Modules are taught in classes; many modules can be taught in the same class, but the same module is not taught in multiple classes.
20. Each class has a unique name.



## 1.4 Identification of entities and attributes

### 1.4.1 List of created objects- Entities and Attributes

In simple language, entities are objects in real life like school, student etc. they represent something of interest to the end user and attributes are the properties of the entities like name, age etc. The following entities were developed with following attributes:

Entities	Attributes
Course	course_id (PK), course_name, course_leader, yearly_fee
Module	module_id (PK), course (FK), module_title, module_leader, class, building
Contact	Phone (PK), email, fax
Student	student_id (PK), contact (FK), course (FK), student_name, enrollment_date, marks, country, province, city, street, house
Instructor	instructor_id (PK), contact (FK), instructor_name, salary, country, province, city, street, house, module (FK)

### 1.4.2 Initial ERD

The ERD (Entity-Relationship Diagram) represents the conceptual database as viewed by the end user. ERDs depict the database's main concepts: entities, attributes and relationship. (Carlos Coronel, 2016)

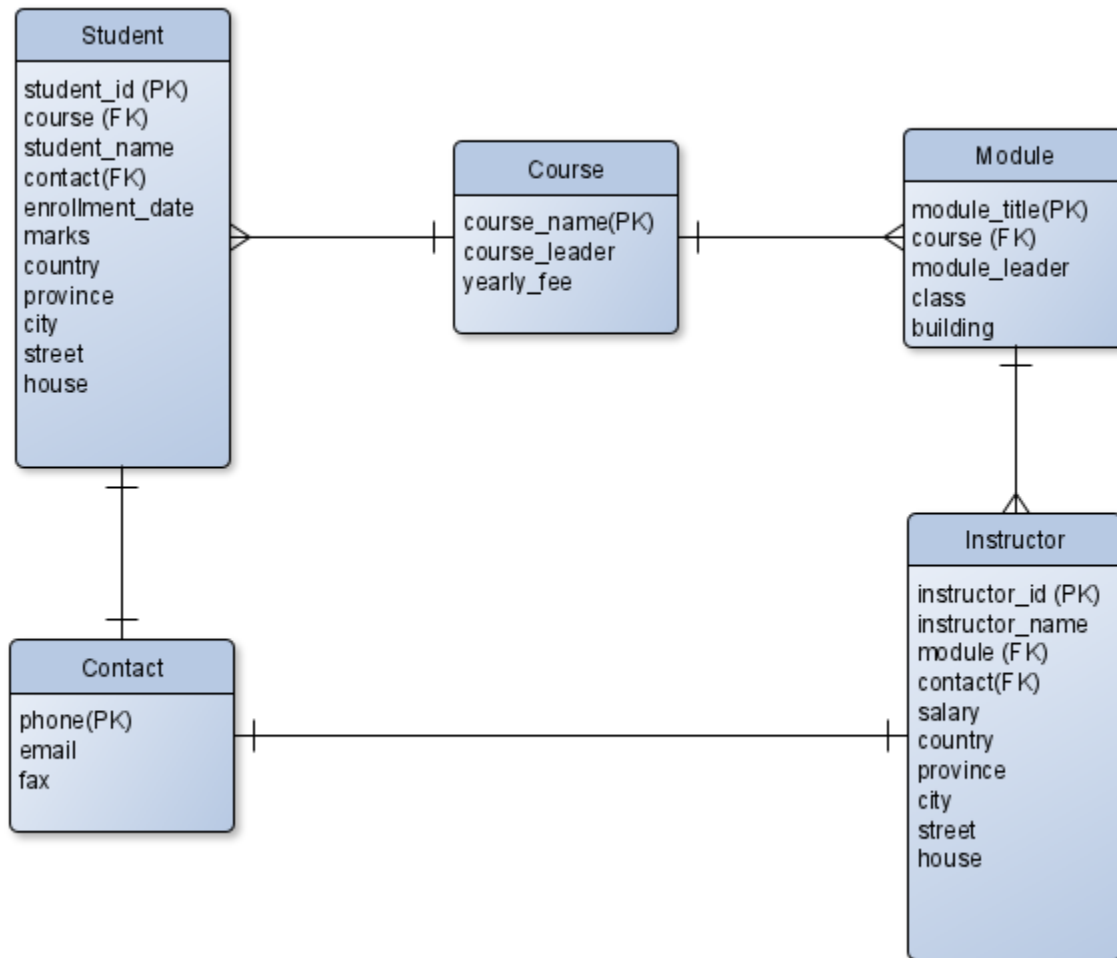


Figure 1: Initial ERD

The basic concept for the ERD shown in figure 1 is multiple students can enroll to a course, single course can have multiple modules, many instructors can teach a module and instructors and students provide details about themselves. The ER diagram may look okay at first glance, but it contains anomalies (anomalies are undesirable outcomes in a database when data is not managed properly) like insertion, update and deletion anomalies. The data in the student and instructor tables are especially redundant, to fix these problems, we need normalization.

## Chapter 2: Normalization

### 2.1 Assumptions

The assumptions that validate the ERD and normalization process are the business rules (Chapter 1 section 1.3 of the report)

### 2.2 Normalization

Normalization is a process for evaluation and correcting table structures to minimize data redundancies, thereby reducing the likelihood of data anomalies. The normalization process involves assigning attributes to tables on the concept of determination. (Carlos Coronel, 2016)

Our goal is to ensure that all the tables that we created in this database are at least in the 3<sup>rd</sup> normal form (3NF). It is to be noted that higher normalization forms exist, but in a business scenario, are discouraged.

NORMAL FORMS	
NORMAL FORM	CHARACTERISTIC
First normal form (1NF)	Table format, no repeating groups, and PK identified
Second normal form (2NF)	1NF and no partial dependencies
Third normal form (3NF)	2NF and no transitive dependencies
Boyce-Codd normal form (BCNF)	Every determinant is a candidate key (special case of 3NF)
Fourth normal form (4NF)	3NF and no independent multivalued dependencies

Figure 2: Normal Forms

(Carlos Coronel, 2016)

Before we begin normalization, let's define functional, partial and transitive dependencies.

- A functional dependency is when an attribute is dependent on another attribute. An example would be student roll number and student name for a class.

Student roll no  $\rightarrow$  student name

Roll no determines name roll no. is determinant attribute and name is dependent attribute

- A partial dependency exists when there is a functional dependence in which the determinant is only part of the primary key. For example, if  $(A, B) \rightarrow (C, D)$ ,  $B \rightarrow C$ , and  $(A, B)$  is the primary key, then the functional dependence  $B \rightarrow C$  is a partial dependency

because only part of the primary key (B) is needed to determine the value of C. (Carlos Coronel, 2016)

- A transitive dependency exists when there are functional dependencies such that  $X \rightarrow Y$ ,  $Y \rightarrow Z$ , and X is the primary key. In that case, the dependency  $X \rightarrow Z$  is a transitive dependency because X determines the value of Z via Y. (Carlos Coronel, 2016)

### 2.2.1 UNF (Un-Normalized Form)

This is the data in our initial design, as it is without any modification or changes.

We will have the data as follows in UNF. The primary keys will be underlined and denoted by PK, foreign key with FK and repeating groups will be represented inside curly brackets, data will be shown as in initial tables as per the design represented in **Chapter 1 section 1.4.2**.

- Course (course\_name (PK), course\_leader, yearly\_fee)
- Module (module\_title (PK), course (FK), module\_leader, {class, building})
- Contact (phone (PK), email, fax)
- Student (student\_id (PK), contact (FK), course (FK), student\_name, enrollment\_date, marks, {country, province, city, street, house})
- Instructor (instructor\_id (PK), contact (FK), instructor\_name, salary, {country, province, city, street, house}, module (FK))

### 2.2.2 1NF (First Normal Form)

The first step in the normalization process, it describes a relation depicted in tabular format, with no repeating groups and a primary key identified. All non-key attributed in the relation are dependent on the primary key. (Carlos Coronel, 2016)

We get the following when we separate the data into 1NF:

- **Course -1** (course\_name (PK), course\_leader, yearly\_fee)
- **Class -1** (class\_name (PK), building)
- **Module -1**(module\_title (PK), course (FK), class (FK) module\_leader)
- **Contact-1** (phone (PK), email, fax)
- **Address -1** (address\_id (PK), country, province, city, street, house)
- **Student-1** (student\_id (PK), course (FK), address (FK), contact (FK), student\_name, enrollment\_date, marks,)
- **Instructor-1** (instructor\_id (PK), module (FK), address (FK), contact (FK) instructor\_name, salary)

In 1NF, our objective was to define primary key, reduce data redundancy, make sure there are no repeating groups in our tables and all the values are dependent on the primary key.

### 2.2.3 2NF (Second Normal Form)

Second normal form is achieved when the tables are already in 1NF and partial dependencies are eliminated.

We get the following tables in 2NF:

- **Course -2** (course\_name (PK), course\_leader, yearly\_fee)
- **Class-2** (class\_name (PK), building)
- **Module -2** (module\_title (PK), course (FK), class (FK) module\_leader)
- **Contact-2** (phone (PK), email, fax)
- **Address -2** (address\_id (PK), country, province, city, street, house)
- **Student-2** (student\_id (PK), course (FK), address (FK), contact (FK), student\_name, enrollment\_date, marks,)
- **Instructor-2** (instructor\_id (PK), module (FK), address (FK), contact (FK) instructor\_name, salary)

We eliminated partial dependencies (check **Chapter 2 section 2.2**) for 2NF. If there is only one key value, then we don't have to check for partial dependencies, else we need to check. Below is the description for how we achieved 2NF for each table.

- Course: There are no partial dependencies, as there is only 1 key in the table
- Class: There is only 1 key, so there is no partial dependency
- Module: This table has foreign keys, so there may be partial dependencies. We check this by seeing if any key other than primary key can give the value for non-key attribute. The process is as follows:

Module -1(module\_title (PK), course (FK), class (FK) module\_leader)

module\_title → module\_leader

course →

class →

module\_title, course →

module\_title, course, class →

- Contact: There is only 1 key so, there is no partial dependency.
- Address: There is only 1 key so, partial dependencies do not exist.
- Student: The process is same as described for Module. The process is:

Student-1 (student\_id (PK), course (FK), address (FK), contact (FK), student\_name, enrollment\_date, marks,)

Student\_id → student\_name, enrollment\_date, marks

course →

address →

contact →

student\_id, course →

student\_id, address →

student\_id, contact →

course, contact →

address, contact →

course, address →

student\_id, course, contact, address →

all attributes are given by student\_id, partial dependencies do not exist, so the table is in 2NF.

- Instructor: Process is given below:

Instructor-1 (instructor\_id (PK), module (FK), address (FK), contact (FK)  
instructor\_name, salary)

$\text{instructor\_id} \rightarrow \text{instructor\_name, salary}$

$\text{module} \rightarrow$

$\text{address} \rightarrow$

$\text{contact} \rightarrow$

$\text{instructor\_id, contact} \rightarrow$

$\text{instructor\_id, module} \rightarrow$

$\text{instructor\_id, address} \rightarrow$

$\text{module, contact} \rightarrow$

$\text{module, address} \rightarrow$

$\text{address, contact} \rightarrow$

$\text{instructor\_id, module, address, contact} \rightarrow$

all attributes are given by instructor\_id so partial dependencies do not exist, so the table is in 2NF



### 2.2.4 3NF (Third Normal Form)

Third normal form is achieved when the tables are already in 2NF and transitive dependencies are eliminated.

We get the following tables in 3NF:

- **Course -3** (course\_name (PK), course\_leader, yearly\_fee)
- **Class-3** (class\_name (PK), building)
- **Module -3** (module\_title (PK), course (FK), class (FK) module\_leader)
- **Contact-3** (phone (PK), email, fax)
- **Address -3** (address\_id (PK), country, province, city, street, house)
- **Student-3** (student\_id (PK), course (FK), address (FK), contact (FK), student\_name, enrollment\_date, marks)
- **Instructor-3** (instructor\_id (PK), module (FK), address (FK), contact (FK) instructor\_name, salary)

We eliminated transitive dependencies (check **Chapter 2 section 2.2**) i.e. no non key value should be able to give any other non key value. Below is the description for how we achieved 3NF for each table.

- Course:

Course -2 (course\_name (PK), course\_leader, yearly\_fee)

The table is already in 3NF there are no partial dependency. Course\_name gives course\_leader and yearly\_fee and neither of those can give value for the other.

- Class:

Class-2 (class\_name (PK), building)

There is only 1 non key so, there is no transitive dependency.

- Module:

Module -2(module\_title (PK), course (FK), class (FK) module\_leader)

There is only 1 non key so, there is no transitive dependency.

- Contact:

Contact-2 (phone (PK), email, fax)

Email and fax are non-key attributes, they cannot give the value for one another, so there is no transitive dependency in the table.

- Address:

Address -2 (address\_id (PK), country, province, city, street, house)

Province numbers in different countries can be same, house numbers can be same in different streets, two cities can have same names and so on. The table is already in 3NF as no non-key attribute can clearly give the value for any other non-key attribute.

- Student:

Student-2 (student\_id (PK), course (FK), address (FK), contact (FK), student\_name, enrollment\_date, marks,)

Student\_id, course, address, contact → student\_name, enrollment\_date

Student\_name cannot give enrollment\_date, as many students with same name can enroll in different dates and enrollment\_date cannot give student\_name, as many students can enroll in same date. The table is free of transitive dependencies.

- Instructor:

Instructor-2 (instructor\_id (PK), module (FK), address (FK), contact (FK)  
instructor\_name, salary)

instructor\_id, module, address, contact  $\rightarrow$  instructor\_name, salary

instructor\_name cannot give salary as 2 people with same name can have different salary and salary cannot give instructor\_name as, many instructors can have the same salary, so there are no transitive dependencies.

## 2.3 ERD after normalization

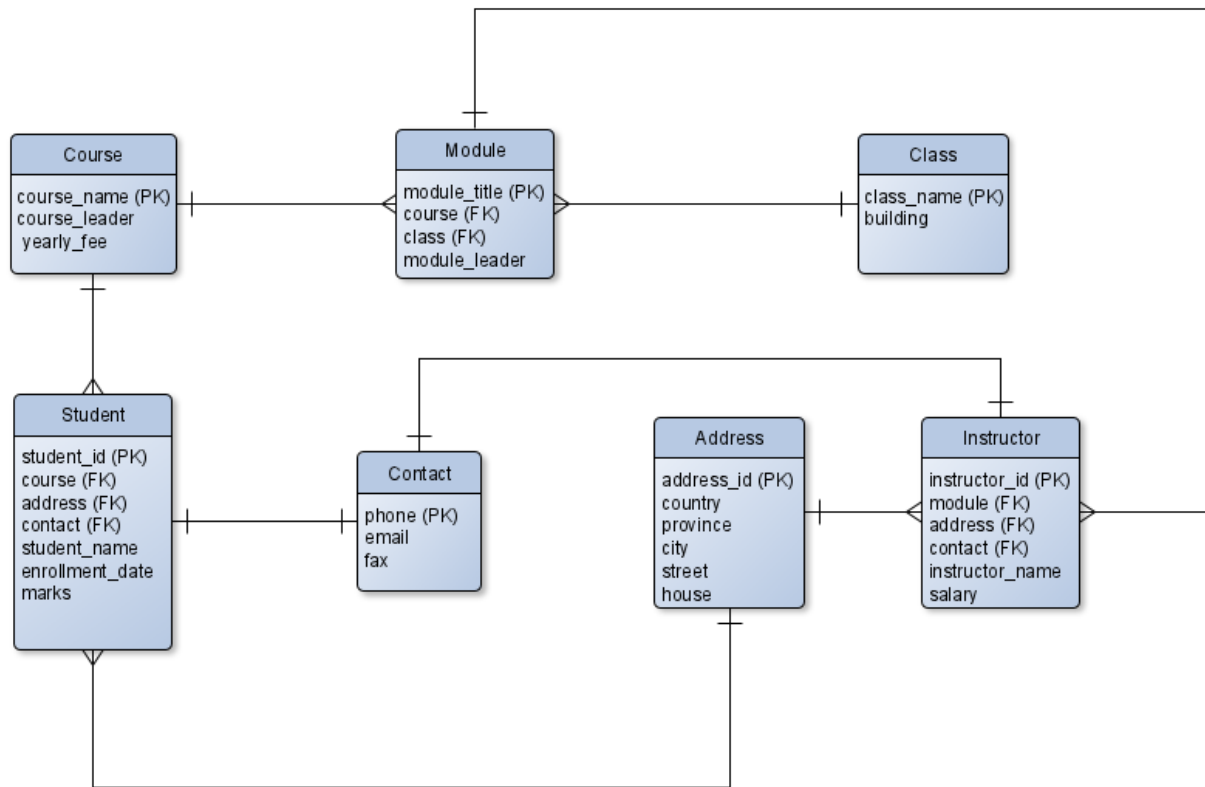


Figure 3: ERD after normalization

This is the final ERD, it has less data redundancy and fewer data anomalies compared to the design in initial ERD. Space is saved for our database and efficiency is increased as a result of normalization. It also makes it easier to manipulate the data in our database and makes maintenance easier.

## Chapter 3: Implementation

### 3.1 Creation of tables

CREATE TABLE statement is used to create tables in MYSQL. It is part of Data Definition Language (DDL), statements for creating and modifying tables, users etc. in a database. The basic structure of the statement is given below:

- CREATE TABLE [table name] ([column name, datatype, constraints]);

#### 3.1.1 Course table

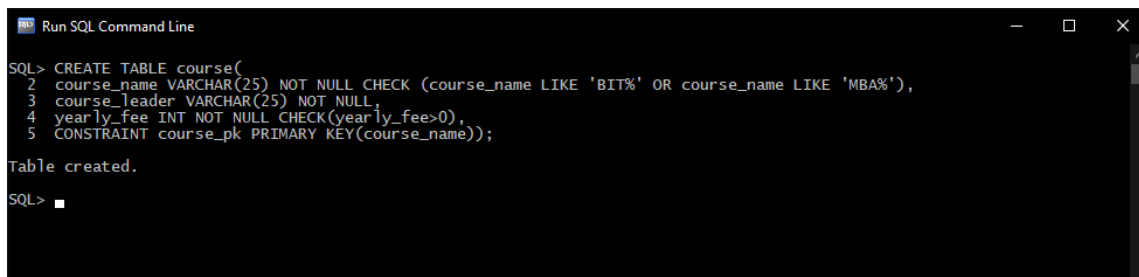
CREATE TABLE course (

course\_name VARCHAR (25) NOT NULL CHECK (course\_name LIKE 'BIT%' OR course\_name LIKE 'MBA%'),

course\_leader VARCHAR (25) NOT NULL,

yearly\_fee INT NOT NULL CHECK (yearly\_fee>0),

CONSTRAINT course\_pk PRIMARY KEY (course\_name));



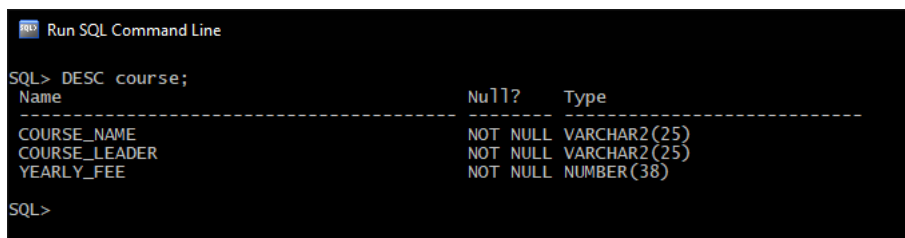
```

Run SQL Command Line
SQL> CREATE TABLE course(
2  course_name VARCHAR(25) NOT NULL CHECK (course_name LIKE 'BIT%' OR course_name LIKE 'MBA%'),
3  course_leader VARCHAR(25) NOT NULL,
4  yearly_fee INT NOT NULL CHECK(yearly_fee>0),
5  CONSTRAINT course_pk PRIMARY KEY(course_name));
Table created.
SQL>

```

Figure 4: Creating course table

VARCHAR and INT specify the datatypes for the columns. CHECK is used to make sure only certain types of values can be entered PRIMARY KEY is to denote primary key for the table.



```

Run SQL Command Line
SQL> DESC course;

```

Name	Null?	Type
COURSE_NAME	NOT NULL	VARCHAR2(25)
COURSE_LEADER	NOT NULL	VARCHAR2(25)
YEARLY_FEE	NOT NULL	NUMBER(38)

```

SQL>

```

Figure 5: DESC course

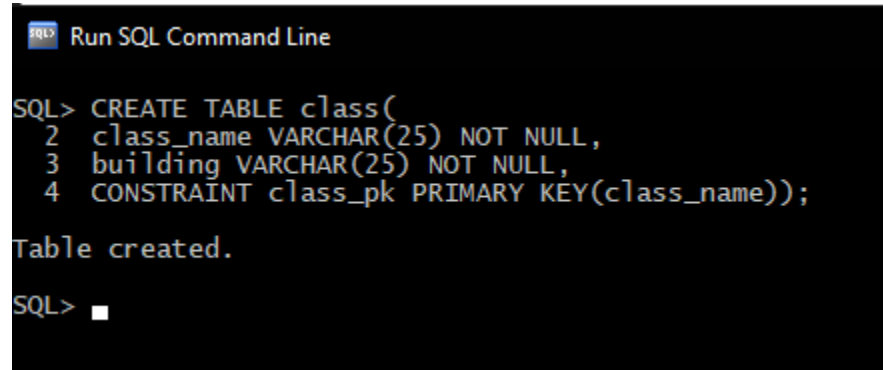
### 3.1.2 Class Table

```
CREATE TABLE class(
```

```
class_name VARCHAR(25) NOT NULL,
```

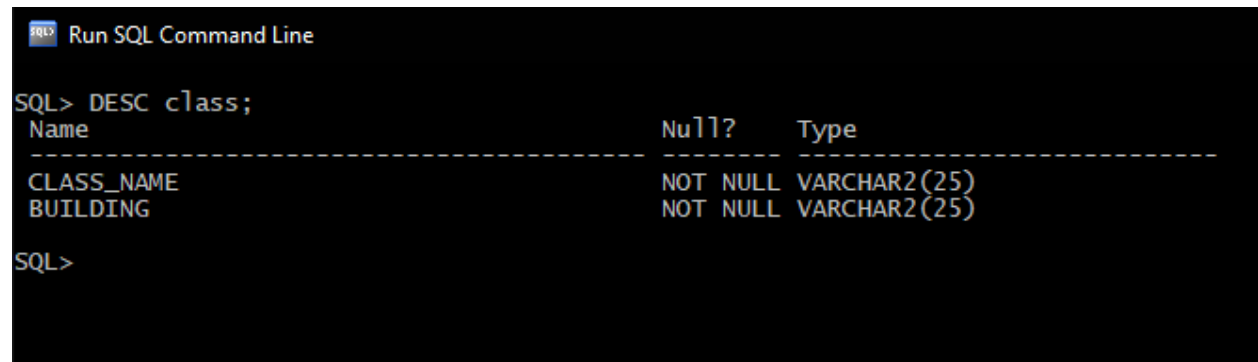
```
building VARCHAR(25) NOT NULL,
```

```
CONSTRAINT class_pk PRIMARY KEY(class_name));
```



```
SQL> CREATE TABLE class(  
2 class_name VARCHAR(25) NOT NULL,  
3 building VARCHAR(25) NOT NULL,  
4 CONSTRAINT class_pk PRIMARY KEY(class_name));  
  
Table created.  
  
SQL> █
```

Figure 6: Creating class table



```
SQL> DESC class;  
Name Null? Type  
-----  
CLASS_NAME NOT NULL VARCHAR2(25)  
BUILDING NOT NULL VARCHAR2(25)  
  
SQL>
```

Figure 7: DESC class

**3.1.3 Module Table**

```
CREATE TABLE module(
```

```
  module_title VARCHAR(25) NOT NULL,
```

```
  course VARCHAR(25) NOT NULL CHECK(course LIKE 'BIT%' OR course LIKE 'MBA%'),
```

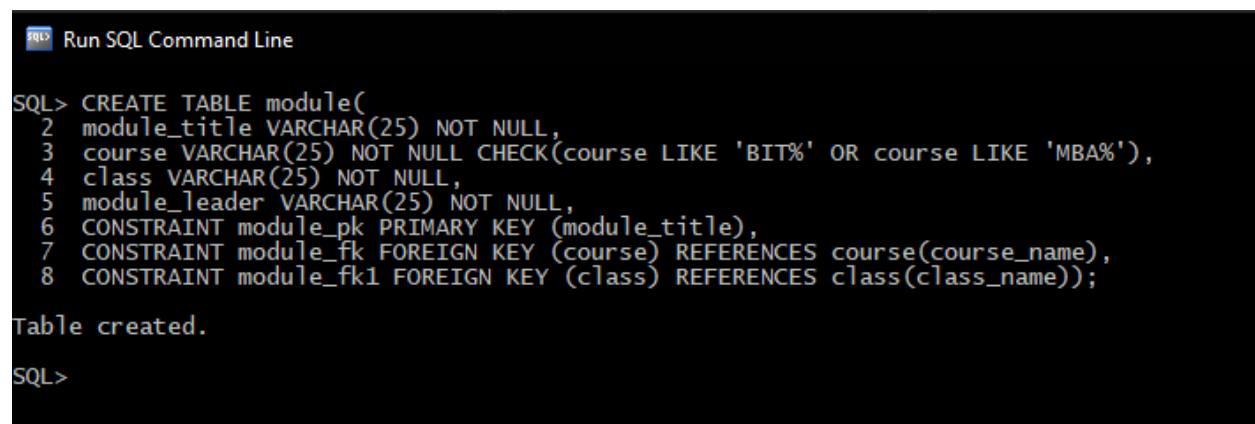
```
  class VARCHAR(25) NOT NULL,
```

```
  module_leader VARCHAR(25) NOT NULL,
```

```
  CONSTRAINT module_pk PRIMARY KEY (module_title),
```

```
  CONSTRAINT module_fk FOREIGN KEY (course) REFERENCES course(course_name),
```

```
  CONSTRAINT module_fk1 FOREIGN KEY (class) REFERENCES class(class_name));
```



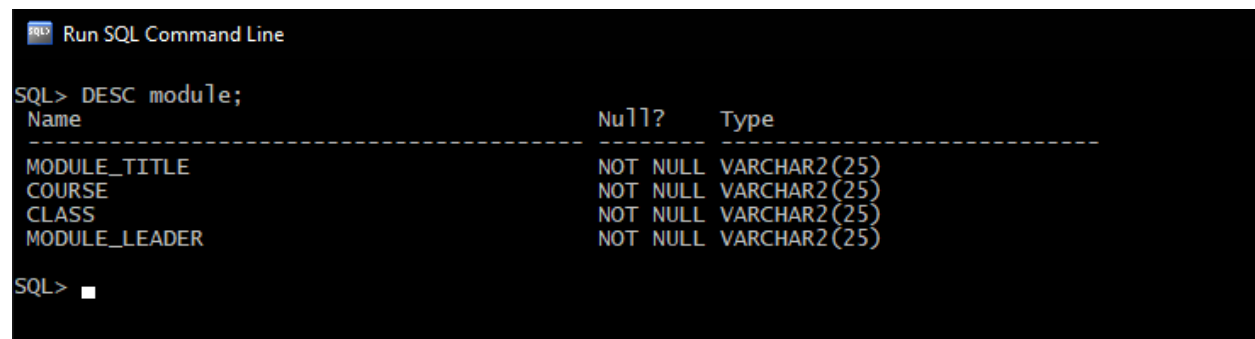
```
Run SQL Command Line

SQL> CREATE TABLE module(
  2  module_title VARCHAR(25) NOT NULL,
  3  course VARCHAR(25) NOT NULL CHECK(course LIKE 'BIT%' OR course LIKE 'MBA%'),
  4  class VARCHAR(25) NOT NULL,
  5  module_leader VARCHAR(25) NOT NULL,
  6  CONSTRAINT module_pk PRIMARY KEY (module_title),
  7  CONSTRAINT module_fk FOREIGN KEY (course) REFERENCES course(course_name),
  8  CONSTRAINT module_fk1 FOREIGN KEY (class) REFERENCES class(class_name));

Table created.

SQL>
```

Figure 8: Creating module table



```
Run SQL Command Line

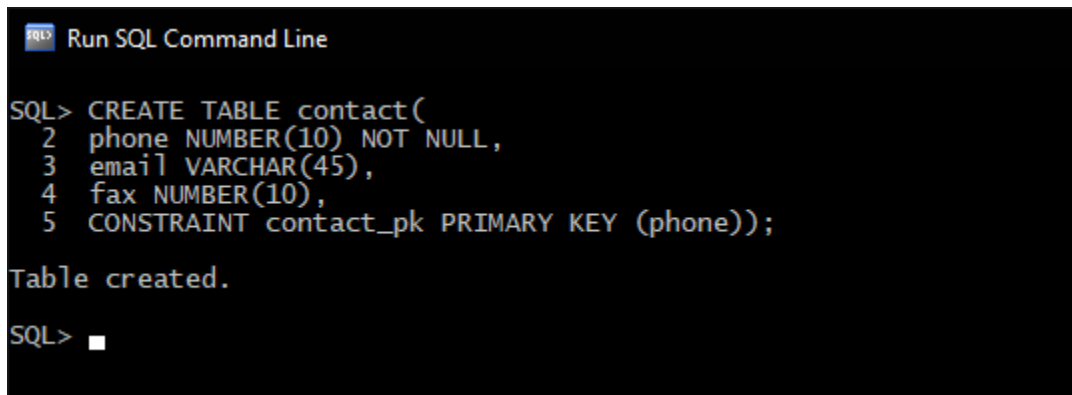
SQL> DESC module;
Name                                Null?    Type
-----
MODULE_TITLE                        NOT NULL VARCHAR2(25)
COURSE                              NOT NULL VARCHAR2(25)
CLASS                               NOT NULL VARCHAR2(25)
MODULE_LEADER                       NOT NULL VARCHAR2(25)

SQL> ■
```

Figure 9: DESC module

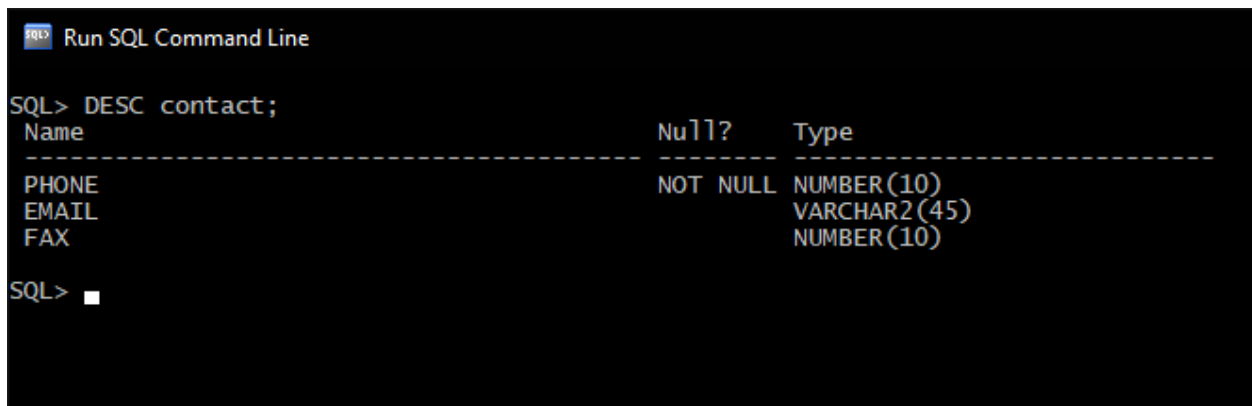
### 3.1.4 Contact table

```
CREATE TABLE contact(  
  
phone NUMBER(10) NOT NULL,  
  
email VARCHAR(45),  
  
fax NUMBER(10),  
  
CONSTRAINT contact_pk PRIMARY KEY(phone));
```



```
SQL> CREATE TABLE contact(  
2  phone NUMBER(10) NOT NULL,  
3  email VARCHAR(45),  
4  fax NUMBER(10),  
5  CONSTRAINT contact_pk PRIMARY KEY (phone));  
  
Table created.  
  
SQL> █
```

Figure 10: Creating contact table



```
SQL> DESC contact;  
Name                               Null?    Type  
-----  
PHONE                             NOT NULL NUMBER(10)  
EMAIL                             VARCHA2(45)  
FAX                               NUMBER(10)  
  
SQL> █
```

Figure 11: DESC contact



**3.1.5 Address Table**

```
CREATE TABLE address(

address_id INT NOT NULL,

country VARCHAR(25) NOT NULL,

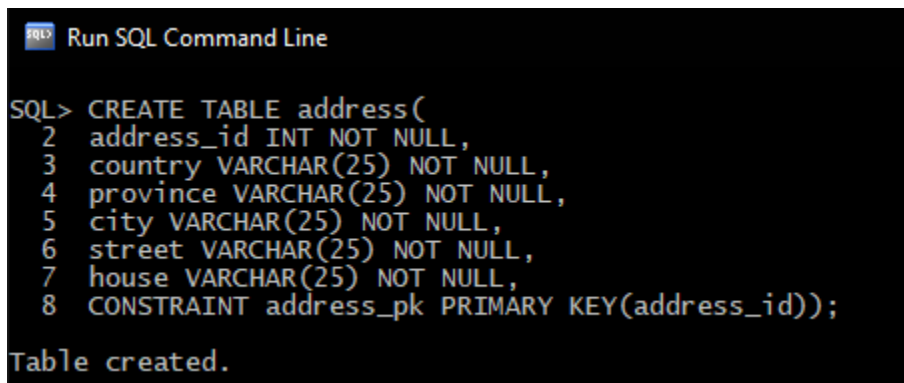
province VARCHAR(25) NOT NULL,

city VARCHAR(25) NOT NULL,

street VARCHAR(25) NOT NULL,

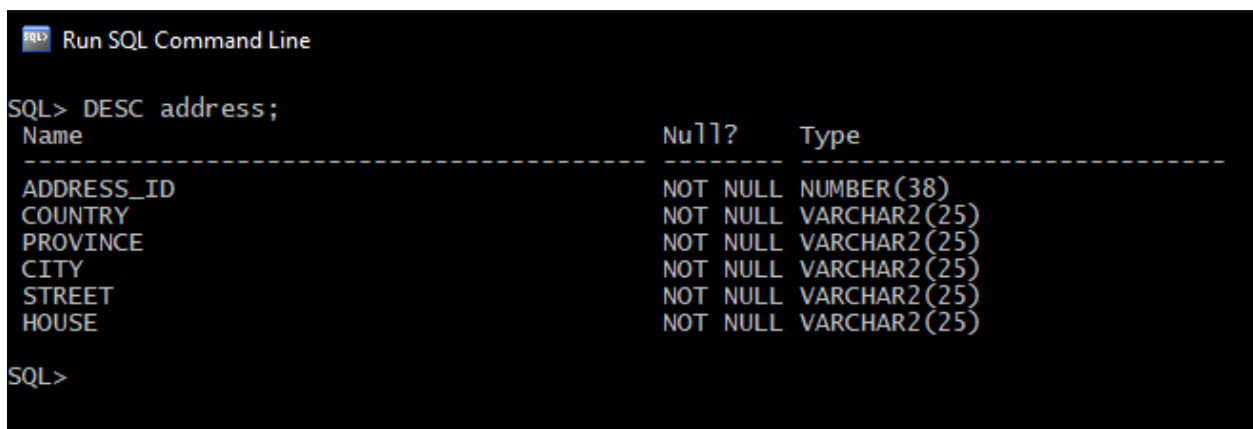
house VARCHAR(25) NOT NULL,

CONSTRAINT address_pk PRIMARY KEY(address_id));
```



```
SQL> CREATE TABLE address(
  2 address_id INT NOT NULL,
  3 country VARCHAR(25) NOT NULL,
  4 province VARCHAR(25) NOT NULL,
  5 city VARCHAR(25) NOT NULL,
  6 street VARCHAR(25) NOT NULL,
  7 house VARCHAR(25) NOT NULL,
  8 CONSTRAINT address_pk PRIMARY KEY(address_id));

Table created.
```

*Figure 12: Creating address table*


```
SQL> DESC address;

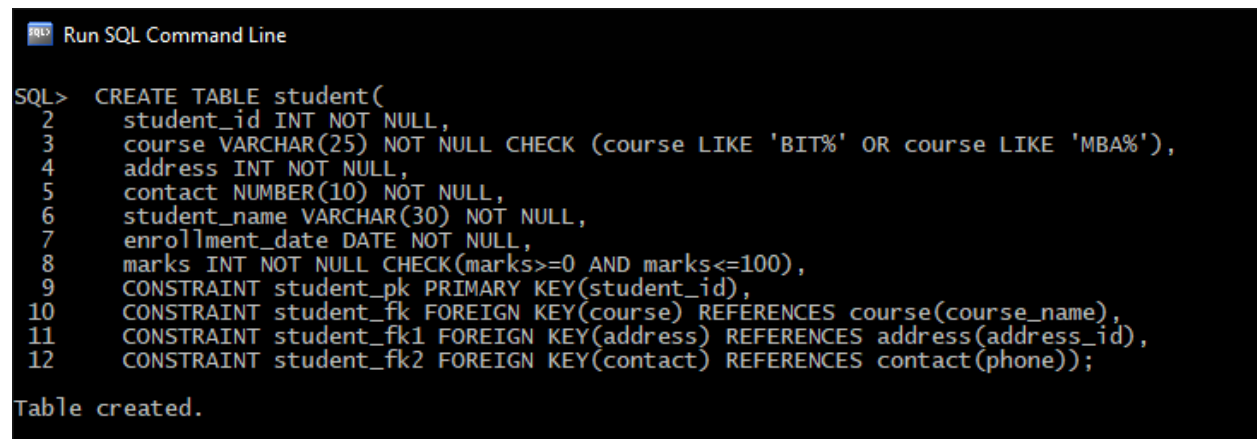
Name                               Null?    Type
-----
ADDRESS_ID                         NOT NULL NUMBER(38)
COUNTRY                           NOT NULL VARCHAR2(25)
PROVINCE                          NOT NULL VARCHAR2(25)
CITY                              NOT NULL VARCHAR2(25)
STREET                            NOT NULL VARCHAR2(25)
HOUSE                             NOT NULL VARCHAR2(25)

SQL>
```

*Figure 13: DESC address*

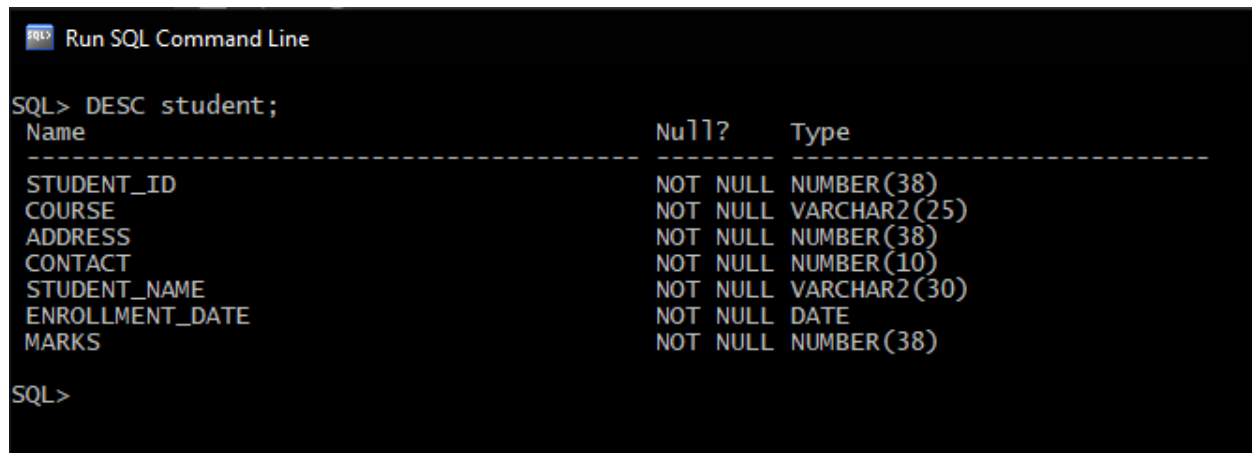
### 3.1.6 Student table

```
CREATE TABLE student(  
  
    student_id INT NOT NULL,  
  
    course VARCHAR(25) NOT NULL CHECK (course LIKE 'BIT%' OR course LIKE 'MBA%'),  
  
    address INT NOT NULL,  
  
    contact NUMBER(10) NOT NULL,  
  
    student_name VARCHAR(30) NOT NULL,  
  
    enrollment_date DATE NOT NULL,  
  
    marks INT NOT NULL CHECK(marks>=0 AND marks<=100),  
  
    CONSTRAINT student_pk PRIMARY KEY(student_id),  
  
    CONSTRAINT student_fk FOREIGN KEY(course) REFERENCES course(course_name),  
  
    CONSTRAINT student_fk1 FOREIGN KEY(address) REFERENCES address(address_id),  
  
    CONSTRAINT student_fk2 FOREIGN KEY(contact) REFERENCES contact(phone));
```



```
SQL> CREATE TABLE student(  
2     student_id INT NOT NULL,  
3     course VARCHAR(25) NOT NULL CHECK (course LIKE 'BIT%' OR course LIKE 'MBA%'),  
4     address INT NOT NULL,  
5     contact NUMBER(10) NOT NULL,  
6     student_name VARCHAR(30) NOT NULL,  
7     enrollment_date DATE NOT NULL,  
8     marks INT NOT NULL CHECK(marks>=0 AND marks<=100),  
9     CONSTRAINT student_pk PRIMARY KEY(student_id),  
10    CONSTRAINT student_fk FOREIGN KEY(course) REFERENCES course(course_name),  
11    CONSTRAINT student_fk1 FOREIGN KEY(address) REFERENCES address(address_id),  
12    CONSTRAINT student_fk2 FOREIGN KEY(contact) REFERENCES contact(phone));  
Table created.
```

Figure 14: Creating student table



```
SQL> DESC student;
```

Name	Null?	Type
STUDENT_ID	NOT NULL	NUMBER(38)
COURSE	NOT NULL	VARCHAR2(25)
ADDRESS	NOT NULL	NUMBER(38)
CONTACT	NOT NULL	NUMBER(10)
STUDENT_NAME	NOT NULL	VARCHAR2(30)
ENROLLMENT_DATE	NOT NULL	DATE
MARKS	NOT NULL	NUMBER(38)

```
SQL>
```

Figure 15: DESC student

### 3.1.7 Instructor table

CREATE TABLE instructor(

instructor\_id VARCHAR(25) NOT NULL CHECK(instructor\_id LIKE 'CL%' OR instructor\_id LIKE 'ML%' OR instructor\_id LIKE 'I%'),

module VARCHAR(25) NOT NULL,

address INT NOT NULL,

contact NUMBER(10) NOT NULL,

instructor\_name VARCHAR(30) NOT NULL,

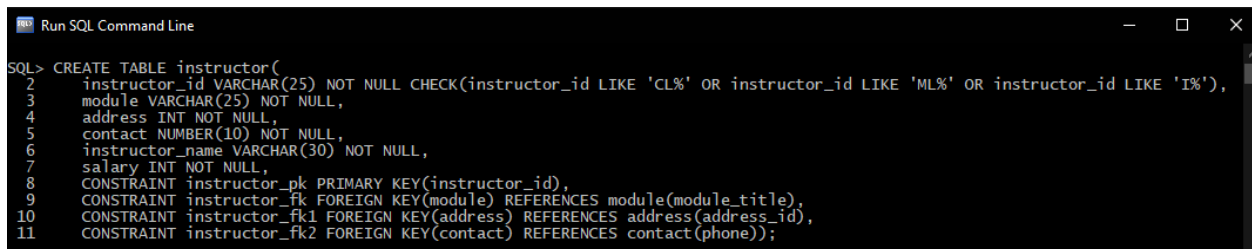
salary INT NOT NULL,

CONSTRAINT instructor\_pk PRIMARY KEY(instructor\_id),

CONSTRAINT instructor\_fk FOREIGN KEY(module) REFERENCES module(module\_title),

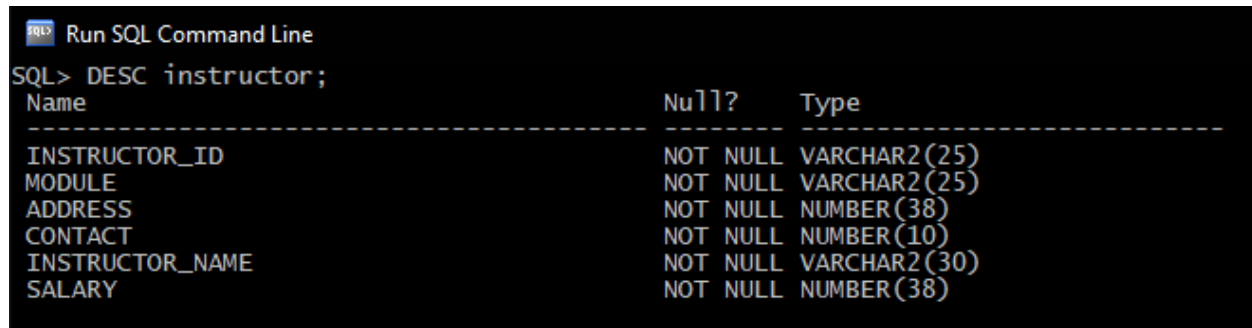
CONSTRAINT instructor\_fk1 FOREIGN KEY(address) REFERENCES address(address\_id),

CONSTRAINT instructor\_fk2 FOREIGN KEY(contact) REFERENCES contact(phone));

A screenshot of a 'Run SQL Command Line' window. The window has a title bar with a small icon, the text 'Run SQL Command Line', and standard window controls (minimize, maximize, close). The main area is a dark-themed text editor showing SQL code. The code is as follows:

```
SQL> CREATE TABLE instructor(  
2     instructor_id VARCHAR(25) NOT NULL CHECK(instructor_id LIKE 'CL%' OR instructor_id LIKE 'ML%' OR instructor_id LIKE 'I%'),  
3     module VARCHAR(25) NOT NULL,  
4     address INT NOT NULL,  
5     contact NUMBER(10) NOT NULL,  
6     instructor_name VARCHAR(30) NOT NULL,  
7     salary INT NOT NULL,  
8     CONSTRAINT instructor_pk PRIMARY KEY(instructor_id),  
9     CONSTRAINT instructor_fk FOREIGN KEY(module) REFERENCES module(module_title),  
10    CONSTRAINT instructor_fk1 FOREIGN KEY(address) REFERENCES address(address_id),  
11    CONSTRAINT instructor_fk2 FOREIGN KEY(contact) REFERENCES contact(phone));
```

Figure 16: Creating instructor table



```
Run SQL Command Line
SQL> DESC instructor;
Name                               Null?    Type
-----
INSTRUCTOR_ID                     NOT NULL VARCHAR2(25)
MODULE                           NOT NULL VARCHAR2(25)
ADDRESS                           NOT NULL NUMBER(38)
CONTACT                           NOT NULL NUMBER(10)
INSTRUCTOR_NAME                   NOT NULL VARCHAR2(30)
SALARY                            NOT NULL NUMBER(38)
```

Figure 17: DESC instructor

### **3.2 Populating data into the tables**

The INSERT, INSERT ALL and COMMIT statements are used to enter data into the tables. INSERT statement enters one row of data at a time while INSERT ALL enters multiple rows of data. They are part of DML (Data Modification Language), used to add, change remove data, here we are only using INSERT to add data.

We use COMMIT statement to finalize entering data into our tables. It is part of TCL (Transaction Control Language) used to manage data that we enter via DML. There are other TCL like ROLLBACK, but here we are only concerned with COMMIT.

Use the COMMIT statement to end your current transaction and make permanent all changes performed in the transaction. A transaction is a sequence of SQL statements that Oracle Database treats as a single unit. This statement also erases all save points in the transaction and releases transaction locks. (Oracle, 2016)

The queries for entering data and necessary screenshots are provided below.

### 3.2.1 Course Table

#### 3.2.1.1 Statement

```
INSERT INTO course VALUES ('BIT Computing', 'James',150000);
```

```
INSERT INTO course VALUES ('BIT Multimedia', 'Ron',150000);
```

```
INSERT INTO course VALUES ('BIT Networking', 'Tom',200000);
```

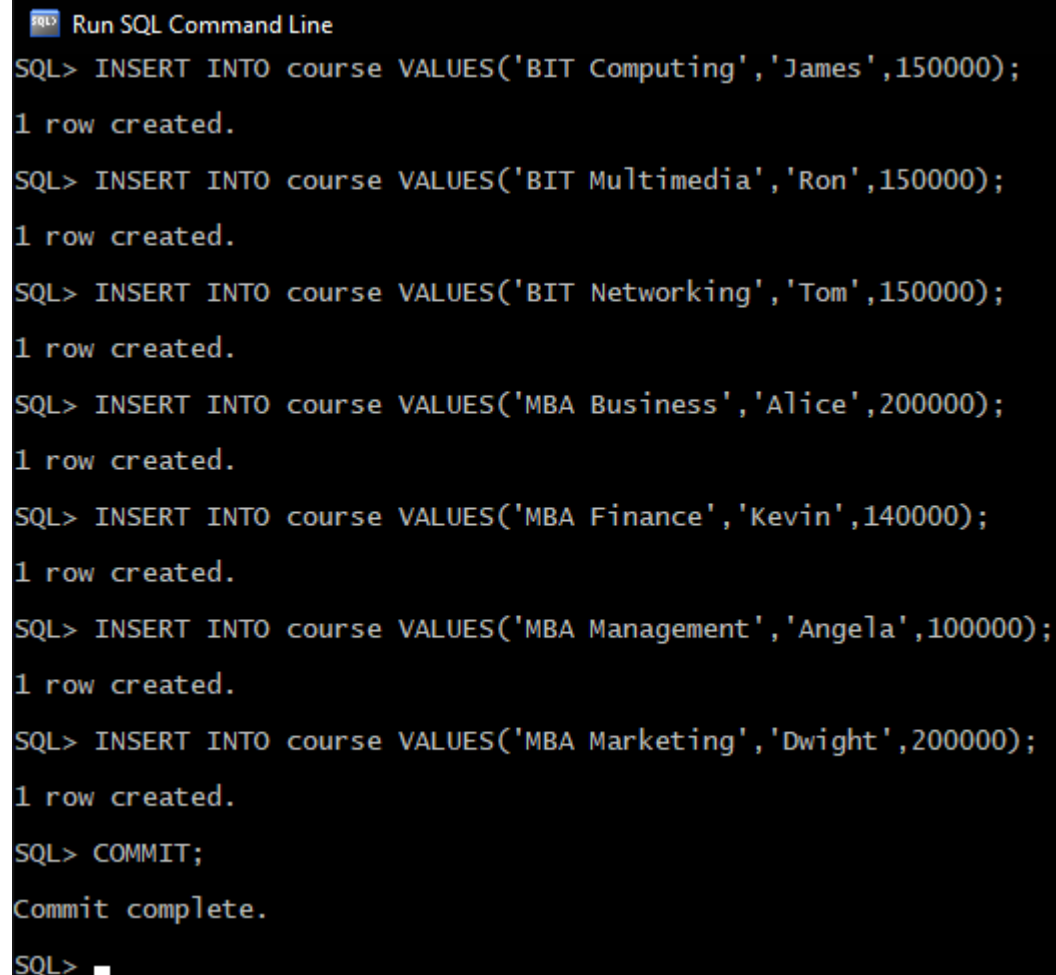
```
INSERT INTO course VALUES ('MBA Business', 'Alice',100000);
```

```
INSERT INTO course VALUES ('MBA Finance', 'Kevin',100000);
```

```
INSERT INTO course VALUES ('MBA Management', 'Angelina',140000);
```

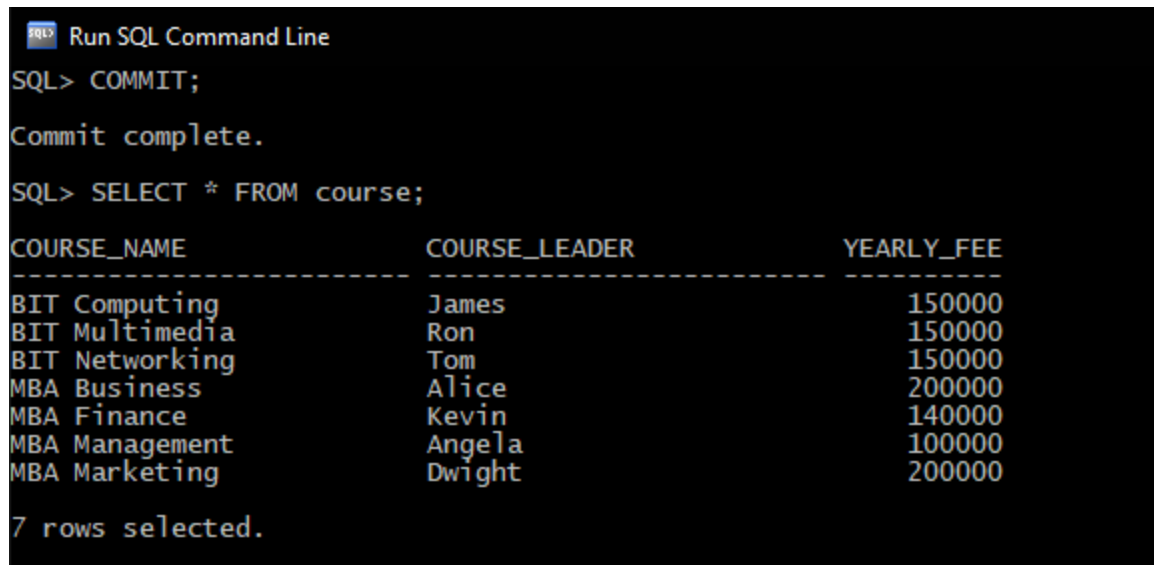
```
INSERT INTO course VALUES ('MBA Marketing', 'Dwight',120000);
```

#### 3.2.1.2 Screenshots



```
SQL> Run SQL Command Line
SQL> INSERT INTO course VALUES('BIT Computing','James',150000);
1 row created.
SQL> INSERT INTO course VALUES('BIT Multimedia','Ron',150000);
1 row created.
SQL> INSERT INTO course VALUES('BIT Networking','Tom',150000);
1 row created.
SQL> INSERT INTO course VALUES('MBA Business','Alice',200000);
1 row created.
SQL> INSERT INTO course VALUES('MBA Finance','Kevin',140000);
1 row created.
SQL> INSERT INTO course VALUES('MBA Management','Angela',100000);
1 row created.
SQL> INSERT INTO course VALUES('MBA Marketing','Dwight',200000);
1 row created.
SQL> COMMIT;
Commit complete.
SQL>
```

Figure 18: Inserting values in course table



```
Run SQL Command Line
SQL> COMMIT;
Commit complete.
SQL> SELECT * FROM course;

COURSE_NAME      COURSE_LEADER      YEARLY_FEE
-----
BIT Computing    James              150000
BIT Multimedia   Ron                150000
BIT Networking   Tom                150000
MBA Business     Alice              200000
MBA Finance      Kevin              140000
MBA Management   Angela             100000
MBA Marketing    Dwight             200000

7 rows selected.
```

Figure 19: Values in course table



### 3.2.2 Class Table

#### 3.2.2.1 Statement

```
INSERT INTO class VALUES('Pokhara','West');
```

```
INSERT INTO class VALUES('Kathmandu','Centre');
```

```
INSERT INTO class VALUES('Sagarmatha','New');
```

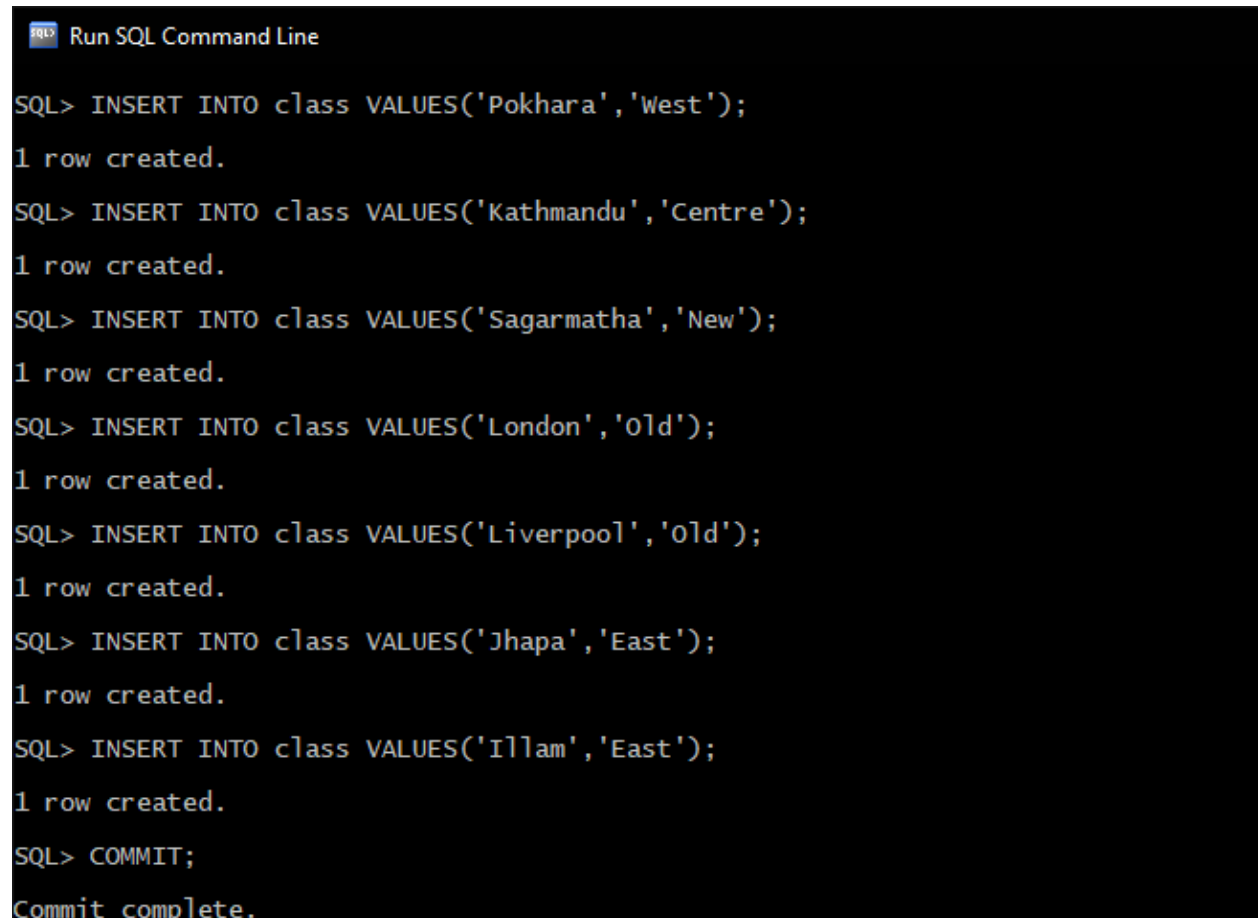
```
INSERT INTO class VALUES('London','Old');
```

```
INSERT INTO class VALUES('Liverpool','Old');
```

```
INSERT INTO class VALUES('Jhapa','East');
```

```
INSERT INTO class VALUES('Illam','East');
```

#### 3.2.2.2 Screenshots



```
Run SQL Command Line

SQL> INSERT INTO class VALUES('Pokhara','West');
1 row created.

SQL> INSERT INTO class VALUES('Kathmandu','Centre');
1 row created.

SQL> INSERT INTO class VALUES('Sagarmatha','New');
1 row created.

SQL> INSERT INTO class VALUES('London','Old');
1 row created.

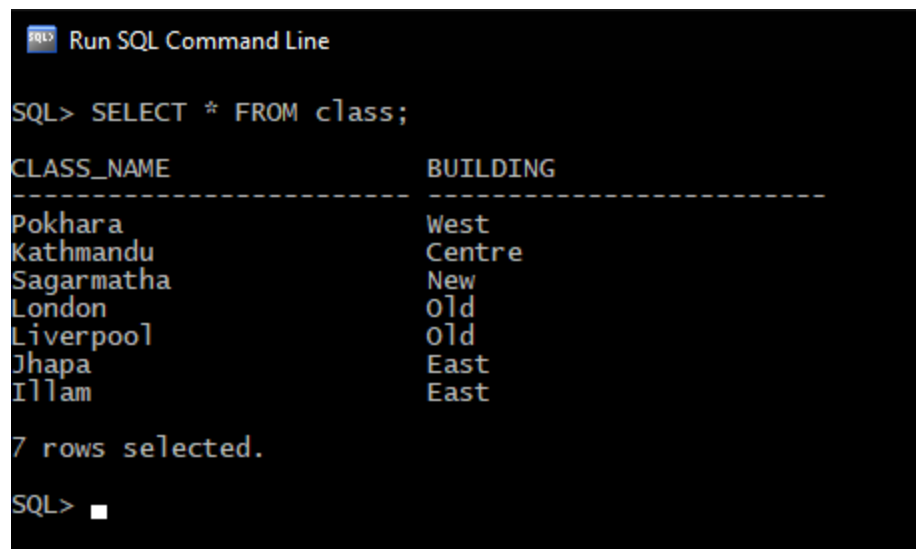
SQL> INSERT INTO class VALUES('Liverpool','Old');
1 row created.

SQL> INSERT INTO class VALUES('Jhapa','East');
1 row created.

SQL> INSERT INTO class VALUES('Illam','East');
1 row created.

SQL> COMMIT;
Commit complete.
```

Figure 20: Inserting values in class table



The screenshot shows a terminal window titled "Run SQL Command Line". The prompt is "SQL>". The command entered is "SELECT \* FROM class;". The output is a table with two columns: "CLASS\_NAME" and "BUILDING". The table has 7 rows of data. Below the table, it says "7 rows selected." and the prompt "SQL>" is followed by a cursor.

```
SQL> SELECT * FROM class;
```

CLASS_NAME	BUILDING
Pokhara	West
Kathmandu	Centre
Sagarmatha	New
London	Old
Liverpool	Old
Jhapa	East
Illam	East

7 rows selected.

```
SQL> █
```

Figure 21: Values in course table

### 3.2.3 Module Table

#### 3.2.3.1 Statement

INSERT ALL

INTO module (module\_title,course,class,module\_leader)

VALUES ('Database','BIT Computing','Pokhara','Jim')

INTO module (module\_title,course,class,module\_leader)

VALUES ('Operating Systems','BIT Computing','Pokhara','Joe')

INTO module (module\_title,course,class,module\_leader)

VALUES ('Networks','BIT Networking','Kathmandu','Angelina')

INTO module (module\_title,course,class,module\_leader)

VALUES ('Hardware and Software','BIT Networking','London','Hanna')

INTO module (module\_title,course,class,module\_leader)

VALUES ('Modelling','BIT Multimedia','London','Phil')

INTO module (module\_title,course,class,module\_leader)

VALUES ('Game Design','BIT Multimedia','Illam','Angela')

INTO module (module\_title,course,class,module\_leader)

VALUES ('Accounting','MBA Finance','Jhapa','Sam')

INTO module (module\_title,course,class,module\_leader)

VALUES ('Resource Management','MBA Management','Liverpool','Bob')

SELECT \* FROM DUAL;

## 3.2.3.2 Screenshots

```

Run SQL Command Line

SQL> INSERT ALL
2 INTO module (module_title,course,class,module_leader) VALUES ('Database','BIT Computing','Pokhara','Jim')
3 INTO module (module_title,course,class,module_leader) VALUES ('Operating Systems','BIT Computing','Pokhara','Joe')
4 INTO module (module_title,course,class,module_leader) VALUES ('Programming','BIT Computing','Kathmandu','Micheal')
5 INTO module (module_title,course,class,module_leader) VALUES ('Networks','BIT Networking','Kathmandu','Angelina')
6 INTO module (module_title,course,class,module_leader) VALUES ('Hardware and Software','BIT Networking','London','Hanna')
7 INTO module (module_title,course,class,module_leader) VALUES ('Modelling','BIT Multimedia','London','Phil')
8 INTO module (module_title,course,class,module_leader) VALUES ('Game Design','BIT Multimedia','Illam','Angela')
9 INTO module (module_title,course,class,module_leader) VALUES ('Accounting','MBA Finance','Jhapa','Sam')
10 INTO module (module_title,course,class,module_leader) VALUES ('Resource Management','MBA Management','Liverpool','Bob')
11 SELECT * FROM DUAL;

9 rows created.

SQL> COMMIT;

Commit complete.

SQL>

```

Figure 22: Inserting values in module table

```

Run SQL Command Line

SQL> SELECT * FROM module;

MODULE_TITLE      COURSE      CLASS      MODULE_LEADER
-----
Database          BIT Computing  Pokhara    Jim
Operating Systems BIT Computing  Pokhara    Joe
Programming       BIT Computing  Kathmandu  Micheal
Networks          BIT Networking Kathmandu  Angelina
Hardware and Software BIT Networking London      Hanna
Modelling         BIT Multimedia London      Phil
Game Design       BIT Multimedia Illam      Angela
Accounting        MBA Finance   Jhapa      Sam
Resource Management MBA Management Liverpool   Bob

9 rows selected.

SQL>

```

Figure 23: Values in module table

### 3.2.4 Contact Table

#### 3.2.4.1 Statement

INSERT ALL

INTO contact (phone,email,fax) VALUES (9876548756,',5656788898)

INTO contact (phone,email,fax) VALUES (9856874533,',')

INTO contact (phone,email,fax) VALUES (9867675454,'dom@mail.com',')

INTO contact (phone,email,fax) VALUES (8976543255,',')

INTO contact (phone,email,fax) VALUES (3622486537,'freddy@mail.com',')

INTO contact (phone,email,fax) VALUES (9877665544,'barryfreddy@mail.com',')

INTO contact (phone,email,fax) VALUES (2222386746,'lisa@mail.com',2222666689)

INTO contact (phone,email,fax) VALUES (8997657876,'john@mail.com',12121211)

INTO contact (phone,email,fax)

VALUES (9812333678,'angelinawhite@mail.com',9999998888)

INTO contact (phone,email,fax) VALUES (8897657878,'jamesjon2mail.com',')

INTO contact (phone,email,fax) VALUES (1234098765,'dondoe@mail.com',1212121222)

INTO contact (phone,email,fax) VALUES (8998667535,'loremipsum@mail.com',9090909999)

INTO contact (phone,email,fax) VALUES (1092387465,'angelinablu@mail.com',')

INTO contact (phone,email,fax) VALUES (9018765443,'sambl@mail.com',')

INTO contact (phone,email,fax) VALUES (9998787878,'anabn@mail.com',2121214444)

INTO contact (phone,email,fax) VALUES (9000076543,'drewd@mail.com',3232321111)

SELECT \* FROM DUAL;

## 3.2.4.2 Screenshots

```

Run SQL Command Line

SQL> INSERT ALL
  2 INTO contact (phone,email,fax) VALUES (9876548756,'',5656788898)
  3 INTO contact (phone,email,fax) VALUES (9856874533,'','')
  4 INTO contact (phone,email,fax) VALUES (9867675454,'dom@mail.com','')
  5 INTO contact (phone,email,fax) VALUES (8976543255,'','')
  6 INTO contact (phone,email,fax) VALUES (3622486537,'freddy@mail.com','')
  7 INTO contact (phone,email,fax) VALUES (9877665544,'barryfreddy@mail.com','')
  8 INTO contact (phone,email,fax) VALUES (2222386746,'lisa@mail.com',2222666689)
  9 INTO contact (phone,email,fax) VALUES (8997657876,'john@mail.com',1212121211)
 10 INTO contact (phone,email,fax) VALUES (9812333678,'angelinawhite@mail.com',9999998888)
 11 INTO contact (phone,email,fax) VALUES (8897657878,'jamesjon2mail.com','')
 12 INTO contact (phone,email,fax) VALUES (1234098765,'dondoe@mail.com',1212121222)
 13 INTO contact (phone,email,fax) VALUES (8998667535,'loremipsum@mail.com',9090909999)
 14 INTO contact (phone,email,fax) VALUES (1092387465,'angelinablu@mail.com','')
 15 INTO contact (phone,email,fax) VALUES (9018765443,'sambl@mail.com','')
 16 INTO contact (phone,email,fax) VALUES (9998787878,'anabn@mail.com',2121214444)
 17 INTO contact (phone,email,fax) VALUES (9000076543,'drewd@mail.com',3232321111)
 18 SELECT * FROM DUAL;

16 rows created.

SQL> COMMIT;

Commit complete.

SQL> █

```

Figure 24: Inserting values in contact table

```

Run SQL Command Line

SQL> SELECT * FROM contact;

-----
PHONE EMAIL                                FAX
-----
9876548756                                5656788898
9856874533
9867675454 dom@mail.com
8976543255
3622486537 freddy@mail.com
9877665544 barryfreddy@mail.com
2222386746 lisa@mail.com                2222666689
8997657876 john@mail.com                1212121211
9812333678 angelinawhite@mail.com        9999998888
8897657878 jamesjon2mail.com
1234098765 dondoe@mail.com                1212121222
-----
PHONE EMAIL                                FAX
-----
8998667535 loremipsum@mail.com           9090909999
1092387465 angelinablu@mail.com
9018765443 sambl@mail.com
9998787878 anabn@mail.com                2121214444
9000076543 drewd@mail.com                3232321111

16 rows selected.

SQL> █

```

Figure 25: Values in contact table

### 3.2.5 Address Table

#### 3.2.5.1 Statement

INSERT ALL

INTO address (address\_id,country,province,city,street,house)

VALUES (1,'Nepal','3','Koteshwor','Kotdevi','11')

INTO address (address\_id,country,province,city,street,house)

VALUES (2,'Nepal','3','Koteshwor','Dronacharya','11')

INTO address (address\_id,country,province,city,street,house)

VALUES (3,'Nepal','3','Patan','111','90/2')

INTO address (address\_id,country,province,city,street,house)

VALUES (4,'Nepal','3','Changunrayan','Changu','100')

INTO address (address\_id,country,province,city,street,house)

VALUES (5,'Nepal','1','Patan','44','1')

INTO address (address\_id,country,province,city,street,house)

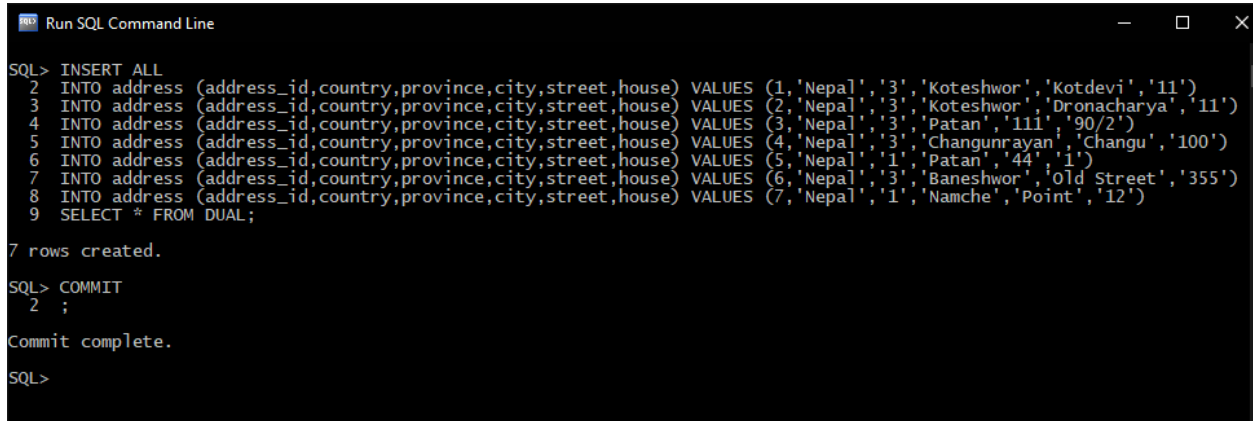
VALUES (6,'Nepal','3','Baneshwor','Old Street','355')

INTO address (address\_id,country,province,city,street,house)

VALUES (7,'Nepal','1','Namche','Point','12')

SELECT \* FROM DUAL

### 3.2.5.1 Screenshots



```

SQL> INSERT ALL
2 INTO address (address_id,country,province,city,street,house) VALUES (1,'Nepal','3','Koteshwor','Kotdevi','11')
3 INTO address (address_id,country,province,city,street,house) VALUES (2,'Nepal','3','Koteshwor','Dronacharya','11')
4 INTO address (address_id,country,province,city,street,house) VALUES (3,'Nepal','3','Patan','111','90/2')
5 INTO address (address_id,country,province,city,street,house) VALUES (4,'Nepal','3','Changunrayan','Changu','100')
6 INTO address (address_id,country,province,city,street,house) VALUES (5,'Nepal','1','Patan','44','1')
7 INTO address (address_id,country,province,city,street,house) VALUES (6,'Nepal','3','Baneshwor','Old Street','355')
8 INTO address (address_id,country,province,city,street,house) VALUES (7,'Nepal','1','Namche','Point','12')
9 SELECT * FROM DUAL;

7 rows created.

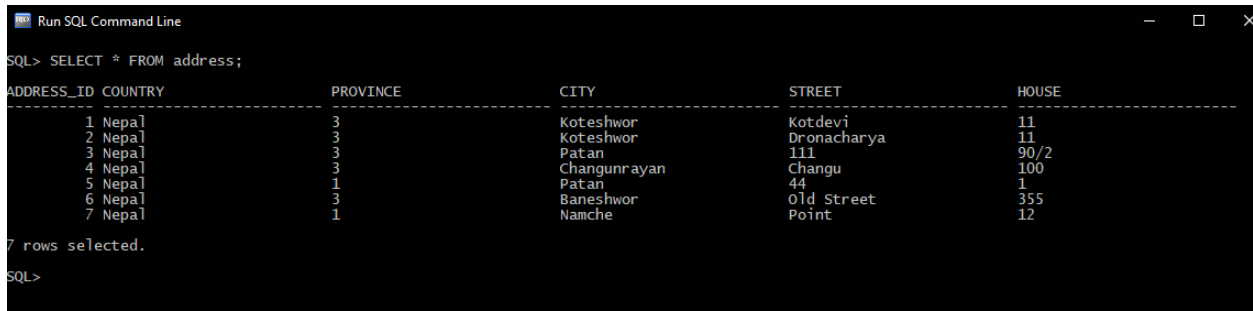
SQL> COMMIT
2 ;

Commit complete.

SQL>

```

Figure 26: Inserting values in address table



```

SQL> SELECT * FROM address;

ADDRESS_ID COUNTRY  PROVINCE  CITY              STREET            HOUSE
-----
1 Nepal     3         Koteshwor  Kotdevi           11
2 Nepal     3         Koteshwor  Dronacharya       11
3 Nepal     3         Patan      111               90/2
4 Nepal     3         Changunrayan Changu            100
5 Nepal     1         Patan      44               1
6 Nepal     3         Baneshwor  Old Street        355
7 Nepal     1         Namche     Point             12

7 rows selected.

SQL>

```

Figure 27: Values in address table



### 3.2.6 Student Table

#### 3.2.6.1 Statement

INSERT ALL

INTO student VALUES (1001,'BIT Computing',1,9876548756,'John Doe','01-sep-2017',85)

INTO student VALUES (1002,'BIT Computing',1,9856874533,'Peter Parker','01-sep-2017',90)

INTO student VALUES (1003,'BIT Multimedia',3,9867675454,'Dom Mann','01-sep-2017',100)

INTO student VALUES (1004,'BIT Multimedia',4,8976543255,'George Guy','01-sep-2018',75)

INTO student VALUES (1005,'BIT Networking',5,3622486537,'Freddy Day','01-sep-2018',63)

INTO student VALUES (1006,'BIT Networking',6,9877665544,'Barry Berry','01-sep-2019',70)

INTO student VALUES (1007,'MBA Management',6,2222386746,'Lisa Mona','01-sep-2019',100)

INTO student VALUES (1008,'MBA Management',2,8997657876,'john Smith','01-sep-2019',90)

SELECT \* FROM DUAL;

## 3.2.6.2 Screenshots

```

Run SQL Command Line

SQL> INSERT ALL
  2 INTO student VALUES (1001,'BIT Computing',1,9876548756,'John Doe','01-sep-2017',85)
  3 INTO student VALUES (1002,'BIT Computing',1,9856874533,'Peter Parker','01-sep-2017',90)
  4 INTO student VALUES (1003,'BIT Multimedia',3,9867675454,'Dom Mann','01-sep-2017',100)
  5 INTO student VALUES (1004,'BIT Multimedia',4,8976543255,'George Guy','01-sep-2018',75)
  6 INTO student VALUES (1005,'BIT Networking',5,3622486537,'Freddy Day','01-sep-2018',63)
  7 INTO student VALUES (1006,'BIT Networking',6,9877665544,'Barry Berry','01-sep-2019',70)
  8 INTO student VALUES (1007,'MBA Management',6,2222386746,'Lisa Mona','01-sep-2019',100)
  9 INTO student VALUES (1008,'MBA Management',2,8997657876,'john Smith','01-sep-2019',90)
 10 SELECT * FROM DUAL;

8 rows created.

SQL> COMMIT;

Commit complete.

```

Figure 28: Inserting values in student table

```

Run SQL Command Line

SQL> SELECT * FROM student;

STUDENT_ID  COURSE              ADDRESS  CONTACT  STUDENT_NAME  ENROLLMEN  MARKS
-----
1001 BIT Computing    1 9876548756 John Doe    01-SEP-17    85
1002 BIT Computing    1 9856874533 Peter Parker 01-SEP-17    90
1003 BIT Multimedia   3 9867675454 Dom Mann    01-SEP-17   100
1004 BIT Multimedia   4 8976543255 George Guy   01-SEP-18    75
1005 BIT Networking   5 3622486537 Freddy Day   01-SEP-18    63
1006 BIT Networking   6 9877665544 Barry Berry   01-SEP-19    70
1007 MBA Management   6 2222386746 Lisa Mona    01-SEP-19   100
1008 MBA Management   2 8997657876 john Smith    01-SEP-19    90

8 rows selected.

```

Figure 29: Values in student table

### 3.2.7 Instructor Table

#### 3.2.7.1 Statement

INSERT ALL

    INTO instructor VALUES ('CL1000','Resource Management',7,9812333678,'Angelina White',60000)

    INTO instructor VALUES ('CL1001','Database',5,8897657878,'James John',55000)

    INTO instructor VALUES ('I1002','Database',7,1234098765,'Dona De',55000)

    INTO instructor VALUES ('I1003','Networks',4,8998667535,'Lorem Ipsum',44000)

    INTO instructor VALUES ('ML1004','Networks',2,1092387465,'Angelina Blue',52000)

    INTO instructor VALUES ('ML1005','Accounting',3,1092387465,'Sam Bla',50000)

    INTO instructor VALUES ('I1006','Game Design',3,9998787878,'Ana Bna',51000)

    INTO instructor VALUES ('ML1007','Modelling',1,9000076543,'Drew D',55000)

SELECT \* FROM DUAL;

### 3.2.7.2 Statement

```

Run SQL Command Line

SQL> INSERT ALL
  2 INTO instructor VALUES ('CL1000','Resource Management',7,9812333678,'Angelina White',60000)
  3 INTO instructor VALUES ('CL1001','Database',5,8897657878,'James John',55000)
  4 INTO instructor VALUES ('II002','Database',7,1234098765,'Dona De',55000)
  5 INTO instructor VALUES ('II003','Networks',4,8998667535,'Lorem Ipsum',44000)
  6 INTO instructor VALUES ('ML1004','Networks',2,1092387465,'Angelina Blue',52000)
  7 INTO instructor VALUES ('ML1005','Accounting',3,1092387465,'Sam Bla',50000)
  8 INTO instructor VALUES ('II006','Game Design',3,9998787878,'Ana Bna',51000)
  9 INTO instructor VALUES ('ML1007','Modelling',1,9000076543,'Drew D',55000)
 10 SELECT * FROM DUAL;

8 rows created.

SQL> COMMIT;

Commit complete.

SQL>

```

Figure 30: Inserting values in instructor table

```

Run SQL Command Line

SQL> SELECT * FROM instructor;

INSTRUCTOR_ID    MODULE                ADDRESS    CONTACT    INSTRUCTOR_NAME    SALARY
-----
CL1000           Resource Management   7 9812333678 Angelina White      60000
CL1001           Database              5 8897657878 James John          55000
II002            Database              7 1234098765 Dona De             55000
II003            Networks              4 8998667535 Lorem Ipsum         44000
ML1004           Networks              2 1092387465 Angelina Blue       52000
ML1005           Accounting            3 1092387465 Sam Bla            50000
II006            Game Design           3 9998787878 Ana Bna            51000
ML1007           Modelling             1 9000076543 Drew D             55000

8 rows selected.

```

Figure 31: Values in instructor table

## Chapter 4: Information and Transaction Queries

### 4.1 List all the students with all their addresses with their phone numbers.

#### 4.1.1 Query

```

SELECT

s.student_name,

s.contact,

a.country,

a.province,

a.city,

a.street,

a.house

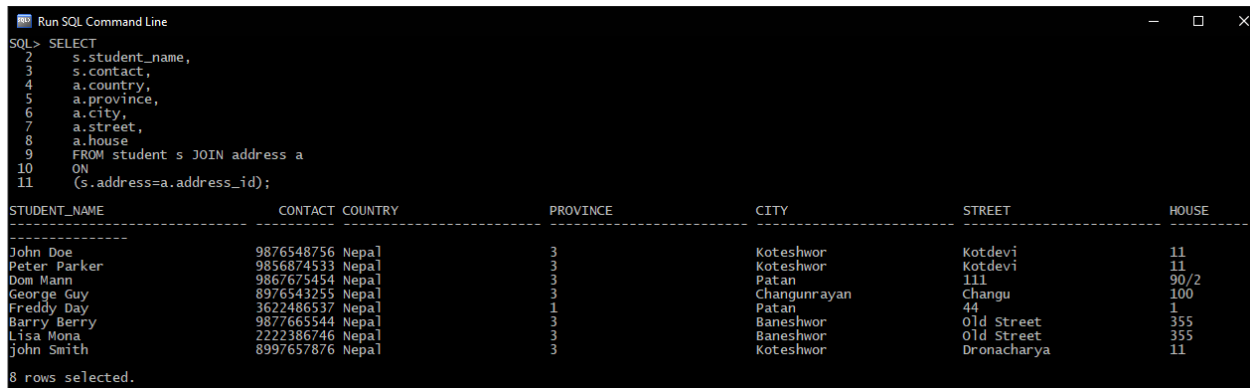
FROM student s JOIN address a

ON

(s.address=a.address_id);

```

#### 4.1.2 Screenshot



```

SQL> SELECT
2  s.student_name,
3  s.contact,
4  a.country,
5  a.province,
6  a.city,
7  a.street,
8  a.house
9  FROM student s JOIN address a
10 ON
11 (s.address=a.address_id);

```

STUDENT_NAME	CONTACT	COUNTRY	PROVINCE	CITY	STREET	HOUSE
John Doe	9876548756	Nepal	3	Koteshwor	Kotdevi	11
Peter Parker	9856874533	Nepal	3	Koteshwor	Kotdevi	11
Dom Mann	9867675454	Nepal	3	Patan	111	90/2
George Guy	8976543255	Nepal	3	Changunrayan	Changu	100
Freddy Day	3622486537	Nepal	1	Patan	44	1
Barry Berry	9877665544	Nepal	3	Baneshwor	Old Street	355
Lisa Mona	2222386746	Nepal	3	Baneshwor	Old Street	355
John Smith	8997657876	Nepal	3	Koteshwor	Dronacharya	11

8 rows selected.

Figure 32: Information query 1

student table was given alias s and a for address table. JOIN with ON was used to join student and address table based on address FK in student table and address\_id PK in address table.

## 4.2 List all the modules which are taught by more than one instructor.

### 4.2.1 Query

SELECT

module "modules\_by\_many\_instructors"

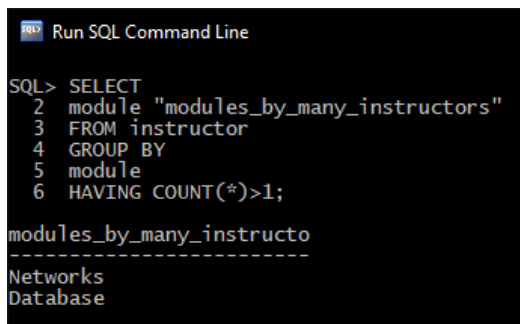
FROM instructor

GROUP BY

module

HAVING COUNT(\*)>1;

### 4.2.2 Screenshot



```

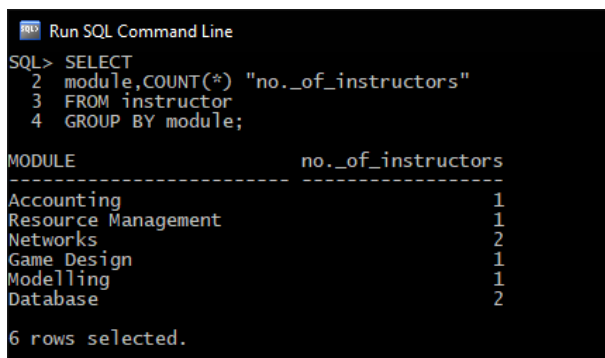
Run SQL Command Line
SQL> SELECT
  2 module "modules_by_many_instructors"
  3 FROM instructor
  4 GROUP BY
  5 module
  6 HAVING COUNT(*)>1;

modules_by_many_instructo
-----
Networks
Database

```

Figure 33: Information query 2

The instructor table holds information about which instructor teaches which module. To find module taught by more than one instructor, COUNT(\*) was used to see how many times a module appeared and if it was more than 1, it was taken and displayed using HAVING statement. GROUP BY is used to take the matching values for the columns. Note: ( If we want to see the number of instructors, following query can be used:)



```

Run SQL Command Line
SQL> SELECT
  2 module,COUNT(*) "no._of_instructors"
  3 FROM instructor
  4 GROUP BY module;

MODULE                                no._of_instructors
-----
Accounting                            1
Resource Management                   1
Networks                              2
Game Design                           1
Modelling                             1
Database                              2

6 rows selected.

```

Figure 34: Seeing how many times a value appears

### 4.3 List the name of all the instructors whose name contains 's' and salary is above 50,000.

#### 4.3.1 Query

SELECT

instructor\_name

FROM instructor

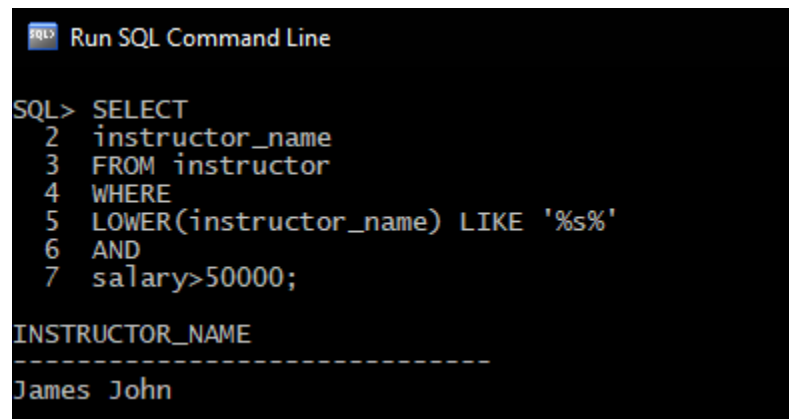
WHERE

LOWER(instructor\_name) LIKE '%s%'

AND

salary>50000;

#### 4.3.2 Screenshot



The screenshot shows a SQL Command Line window with a black background and white text. The title bar reads 'Run SQL Command Line'. The command entered is: SQL> SELECT  
2 instructor\_name  
3 FROM instructor  
4 WHERE  
5 LOWER(instructor\_name) LIKE '%s%'  
6 AND  
7 salary>50000; The result is displayed as a table with the header 'INSTRUCTOR\_NAME' and a single row containing 'James John'.

```
SQL> SELECT
2  instructor_name
3  FROM instructor
4  WHERE
5  LOWER(instructor_name) LIKE '%s%'
6  AND
7  salary>50000;

INSTRUCTOR_NAME
-----
James John
```

Figure 35: Information query 3

LIKE was used to specify what to search and LOWER was used so all data would be in lowercase and nothing will be missed. There is only 1 result because the sample size of the database was small, but the query will function even for large databases.

#### 4.4 List the modules comes under the 'Multimedia' specification.

##### 4.4.1 Query

SELECT

Module\_title "Modules under Multimedia"

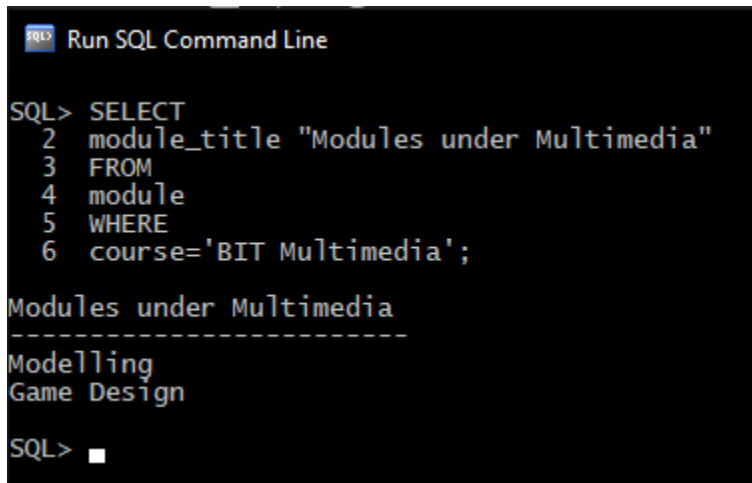
FROM

Module

WHERE

Course = 'BIT Multimedia';

##### 4.4.2 Screenshot



```
SQL> SELECT
  2  module_title "Modules under Multimedia"
  3  FROM
  4  module
  5  WHERE
  6  course='BIT Multimedia';

Modules under Multimedia
-----
Modelling
Game Design
SQL> 
```

Figure 36: Information query 4

WHERE clause was used to select modules under Multimedia.



## 4.5 List the name of the head of modules with the list of his phone number.

### 4.5.1 Query

SELECT

instructor\_name "Module Head",

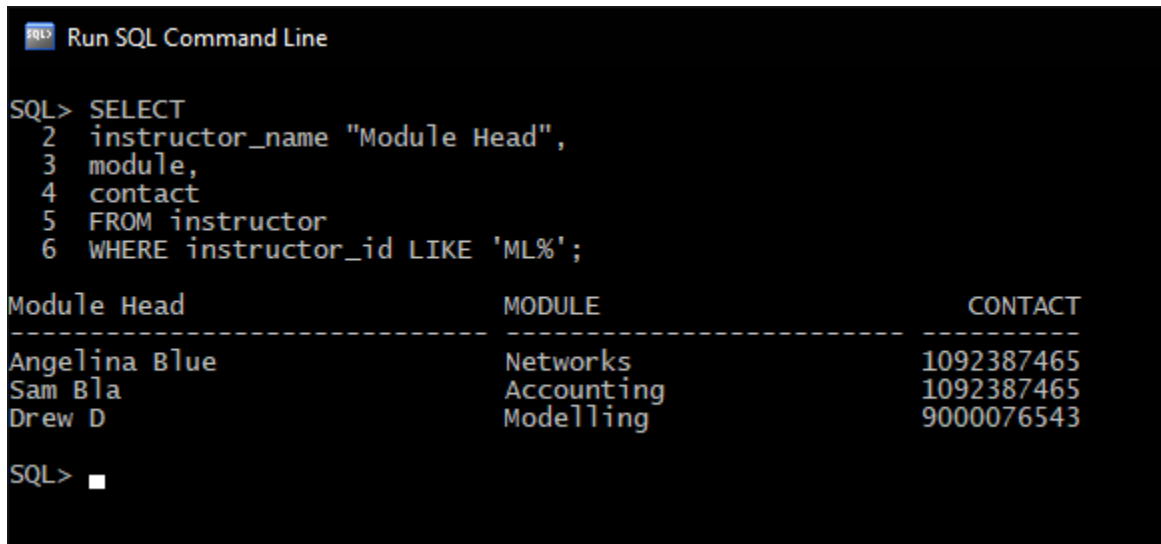
module,

contact

FROM instructor

WHERE instructor\_id LIKE 'ML%';

### 4.5.2 Screenshot



The screenshot shows an SQL Command Line window with the following content:

```

SQL> SELECT
  2  instructor_name "Module Head",
  3  module,
  4  contact
  5  FROM instructor
  6  WHERE instructor_id LIKE 'ML%';

```

The results are displayed in a table format:

Module Head	MODULE	CONTACT
Angelina Blue	Networks	1092387465
Sam Bla	Accounting	1092387465
Drew D	Modelling	9000076543

The prompt 'SQL>' is followed by a small square cursor.

Figure 37: Information query 5

Since, module leaders have ML in their id, we checked for occurrence of ML.

## 4.6 List all Students who have enrolled in 'networking' specifications.

### 4.6.1 Query

SELECT

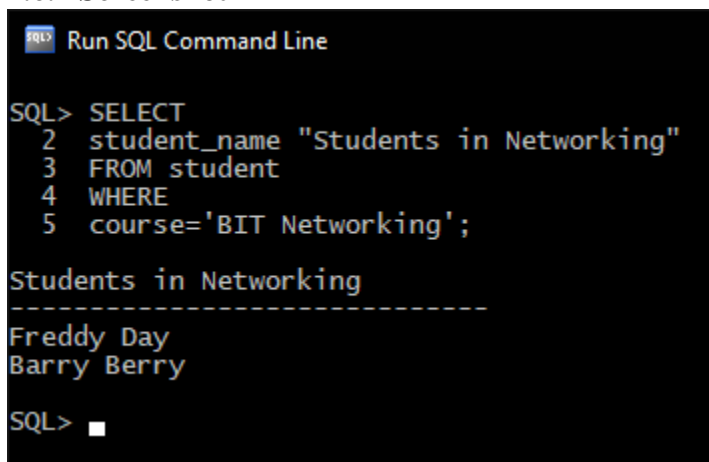
student\_name "Students in Networking"

FROM student

WHERE

course='BIT Networking';

### 4.6.2 Screenshot



```
SQL> Run SQL Command Line

SQL> SELECT
  2  student_name "Students in Networking"
  3  FROM student
  4  WHERE
  5  course='BIT Networking';

Students in Networking
-----
Freddy Day
Barry Berry

SQL> ■
```

Figure 38: Information query 6

To check for students in networking, we specified course as networking.

## 4.7 List the fax number of the instructor who teaches the 'database' module

### 4.7.1 Query

SELECT

i.instructor\_name,

c.fax

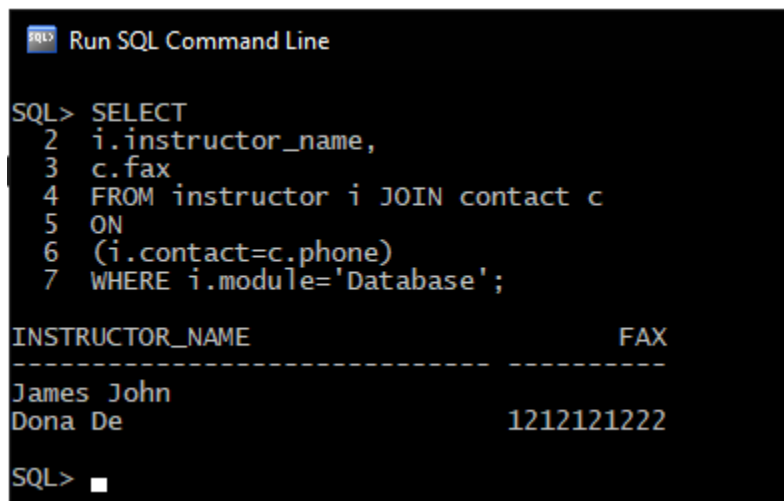
FROM instructor i JOIN contact c

ON

(i.contact=c.phone)

WHERE i.module='Database';

### 4.7.2 Screenshot



The screenshot shows a terminal window titled "Run SQL Command Line". It displays the following SQL query:

```
SQL> SELECT
2  i.instructor_name,
3  c.fax
4  FROM instructor i JOIN contact c
5  ON
6  (i.contact=c.phone)
7  WHERE i.module='Database';
```

The results of the query are displayed in a table format:

INSTRUCTOR_NAME	FAX
James John	
Dona De	1212121222

The prompt "SQL> ■" is visible at the bottom of the terminal.

Figure 39: Information query 7

We join instructor and contact table then only select fax if module is Database.

## 4.8 List the specification falls under the BIT course.

### 4.8.1 Query

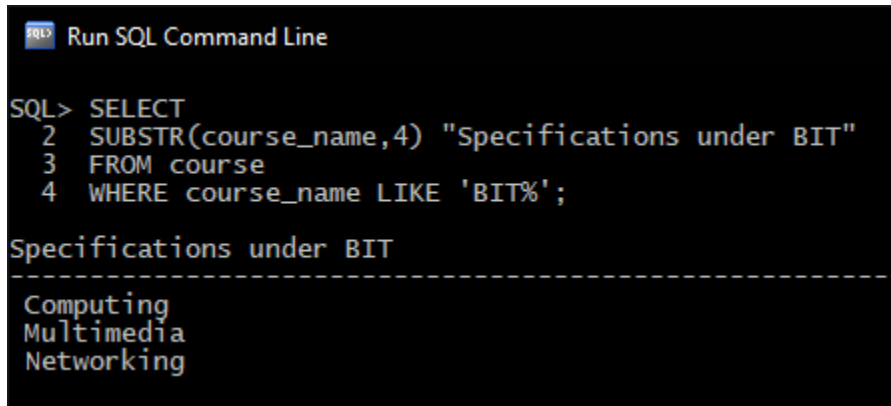
SELECT

SUBSTR(course\_name,4) "Specifications under BIT"

FROM course

WHERE course\_name LIKE 'BIT%';

### 4.8.2 Screenshot

A screenshot of a SQL command line interface. At the top, there is a button labeled "Run SQL Command Line". Below it, the SQL query is entered: SQL> SELECT, 2 SUBSTR(course\_name,4) "Specifications under BIT", 3 FROM course, 4 WHERE course\_name LIKE 'BIT%';. The results of the query are displayed below the query: Specifications under BIT, followed by a dashed line, and then the list of course names: Computing, Multimedia, and Networking.

```
Run SQL Command Line

SQL> SELECT
  2 SUBSTR(course_name,4) "Specifications under BIT"
  3 FROM course
  4 WHERE course_name LIKE 'BIT%';

Specifications under BIT
-----
Computing
Multimedia
Networking
```

Figure 40: Information query 8

As per the business rules, courses for BIT have BIT in their ids, so we use LIKE to check and display such courses.

## 4.9 List all the modules taught in any one particular class.

### 4.9.1 Query

SELECT

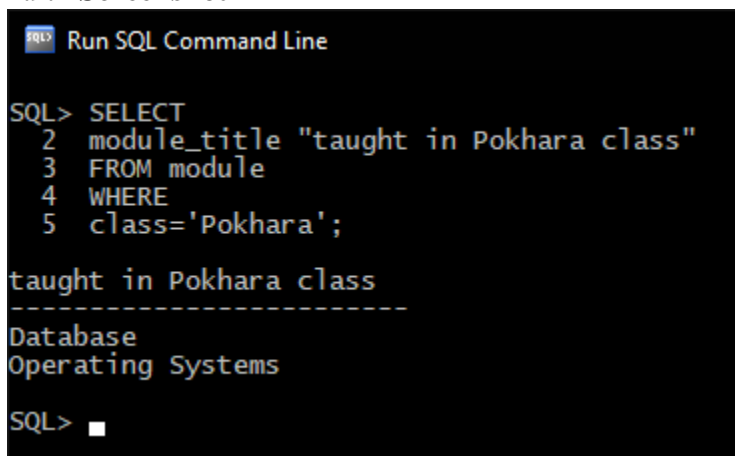
module\_title "taught in Pokhara class"

FROM module

WHERE

class='Pokhara';

### 4.9.2 Screenshot



```
Run SQL Command Line

SQL> SELECT
  2  module_title "taught in Pokhara class"
  3  FROM module
  4  WHERE
  5  class='Pokhara';

taught in Pokhara class
-----
Database
Operating Systems

SQL> █
```

Figure 41: Information query 9

#### 4.10 List all the teachers with all their addresses who have 'a' at the end of their first name.

##### 4.10.1 Query

SELECT

i.instructor\_name,

a.country,

a.province,

a.city,

a.street,

a.house

FROM instructor i JOIN address a

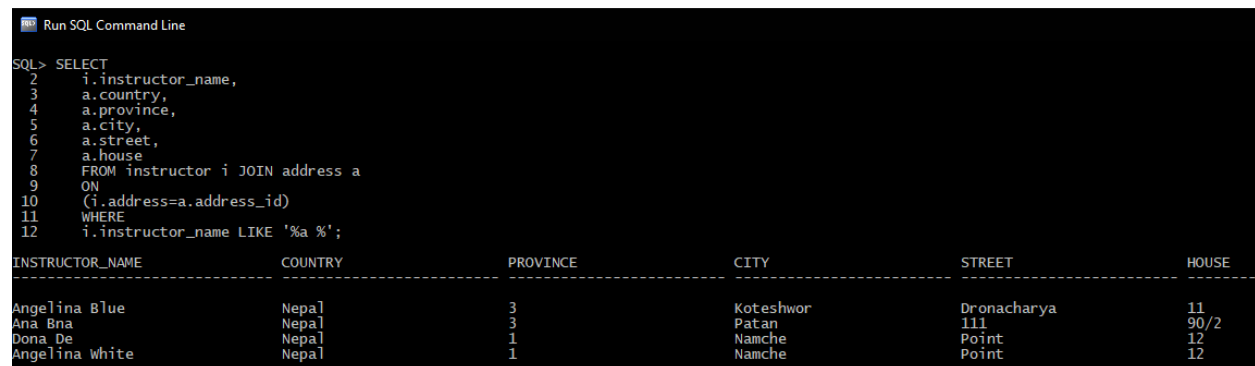
ON

(i.address=a.address\_id)

WHERE

i.instructor\_name LIKE '%a %';

##### 4.10.2 Screenshot



```

SQL> SELECT
2   i.instructor_name,
3   a.country,
4   a.province,
5   a.city,
6   a.street,
7   a.house
8 FROM instructor i JOIN address a
9 ON
10 (i.address=a.address_id)
11 WHERE
12 i.instructor_name LIKE '%a %';
  
```

INSTRUCTOR_NAME	COUNTRY	PROVINCE	CITY	STREET	HOUSE
Angelina Blue	Nepal	3	Koteshwor	Dronacharya	11
Ana Bna	Nepal	3	Patan	111	90/2
Dona De	Nepal	1	Namche	Point	12
Angelina White	Nepal	1	Namche	Point	12

Figure 42: Information query 10

Since name was stored in a single column, LIKE was used to match the given case.

## 4.11 Show the students, course they enroll in and their fees. Reduce 10% of the fees if they are enrolled in a computing course.

### 4.11.1 Query

```
SELECT

    s.student_name,

    s.course,

    c.yearly_fee,DECODE(course,'BIT Computing',yearly_fee-yearly_fee*0.1,

                        'BIT Networking',yearly_fee-yearly_fee*0.1,

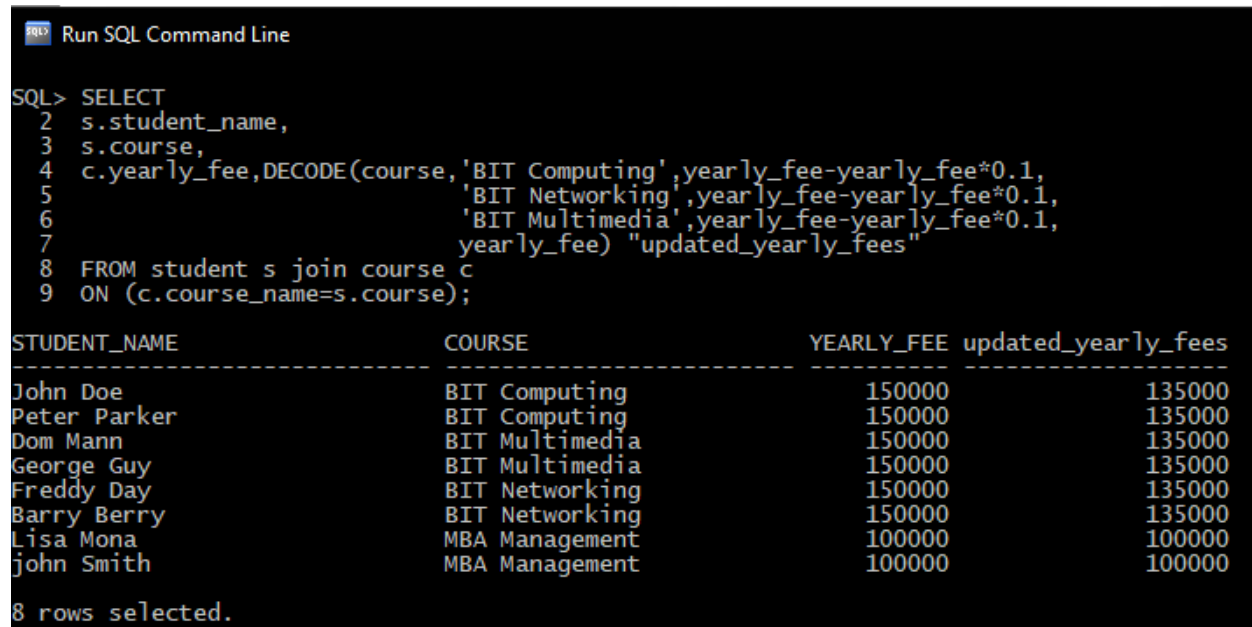
                        'BIT Multimedia',yearly_fee-yearly_fee*0.1,

                        yearly_fee) "updated_yearly_fees"

FROM student s join course c

ON (c.course_name=s.course);
```

### 4.11.2 Screenshot



```
SQL> SELECT
  2  s.student_name,
  3  s.course,
  4  c.yearly_fee,DECODE(course,'BIT Computing',yearly_fee-yearly_fee*0.1,
  5                        'BIT Networking',yearly_fee-yearly_fee*0.1,
  6                        'BIT Multimedia',yearly_fee-yearly_fee*0.1,
  7                        yearly_fee) "updated_yearly_fees"
  8  FROM student s join course c
  9  ON (c.course_name=s.course);
```

STUDENT_NAME	COURSE	YEARLY_FEE	updated_yearly_fees
John Doe	BIT Computing	150000	135000
Peter Parker	BIT Computing	150000	135000
Dom Mann	BIT Multimedia	150000	135000
George Guy	BIT Multimedia	150000	135000
Freddy Day	BIT Networking	150000	135000
Barry Berry	BIT Networking	150000	135000
Lisa Mona	MBA Management	100000	100000
john Smith	MBA Management	100000	100000

8 rows selected.

Figure 43: Transaction query 1

DECODE was used to check if the course was related to computing, then discount was applied if the result was true.

**4.12 Place the default Number 1234567890 if the list of phone numbers to the location of the address is empty and give the column name as 'Contact details.**

#### 4.12.1 Query

SELECT

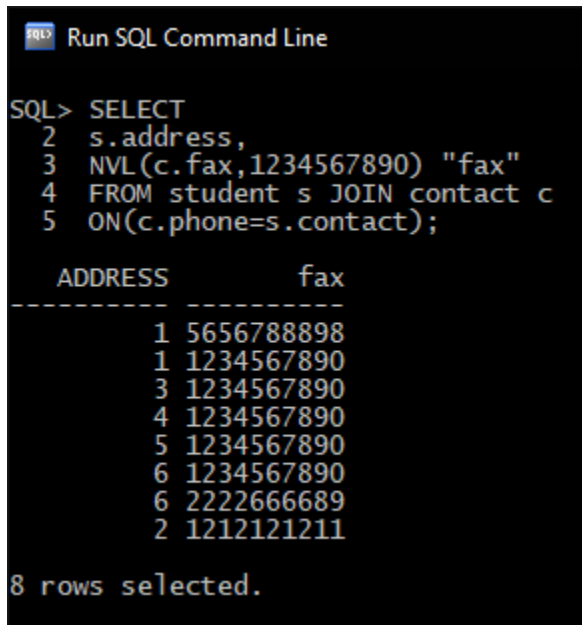
s.address,

NVL(c.fax,1234567890) "fax"

FROM student s JOIN contact c

ON(c.phone=s.contact);

#### 4.12.2 Screenshot



```

SQL> SELECT
  2  s.address,
  3  NVL(c.fax,1234567890) "fax"
  4  FROM student s JOIN contact c
  5  ON(c.phone=s.contact);

```

	ADDRESS	fax
1	5656788898	
1	1234567890	
3	1234567890	
4	1234567890	
5	1234567890	
6	1234567890	
6	2222666689	
2	1212121211	

8 rows selected.

Figure 44: Transaction query 2

As per the assumptions made, phone number cannot be empty, so we are using fax number to complete the query. NVL is used to convert the null value into the value that we have specified.



### 4.13 Show the name of all the students with the number of weeks since they have enrolled in the course

#### 4.13.1 Query

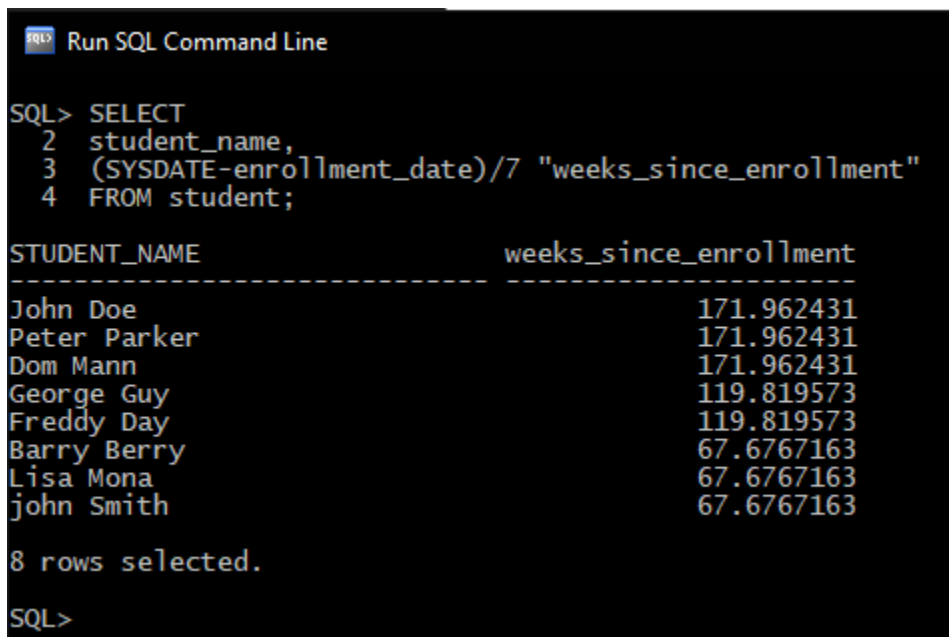
```
SELECT
```

```
    student_name,
```

```
    (SYSDATE-enrollment_date)/7 "weeks_since_enrollment"
```

```
FROM student;
```

#### 4.13.1 Screenshot



```
Run SQL Command Line

SQL> SELECT
      2 student_name,
      3 (SYSDATE-enrollment_date)/7 "weeks_since_enrollment"
      4 FROM student;

STUDENT_NAME                weeks_since_enrollment
-----
John Doe                    171.962431
Peter Parker                171.962431
Dom Mann                    171.962431
George Guy                 119.819573
Freddy Day                 119.819573
Barry Berry                 67.6767163
Lisa Mona                  67.6767163
john Smith                  67.6767163

8 rows selected.

SQL>
```

Figure 45: Transaction query 3

To find weeks since a student has enrolled in a course, we subtract enrollment date from current date to get the number of days which is divided by 7 to get number of weeks.

#### **4.14 Show the name of the instructors who got equal salary and work in the same specification.**

##### **4.14.1 Query**

SELECT

i.instructor\_name

FROM

instructor i

INNER JOIN

instructor j

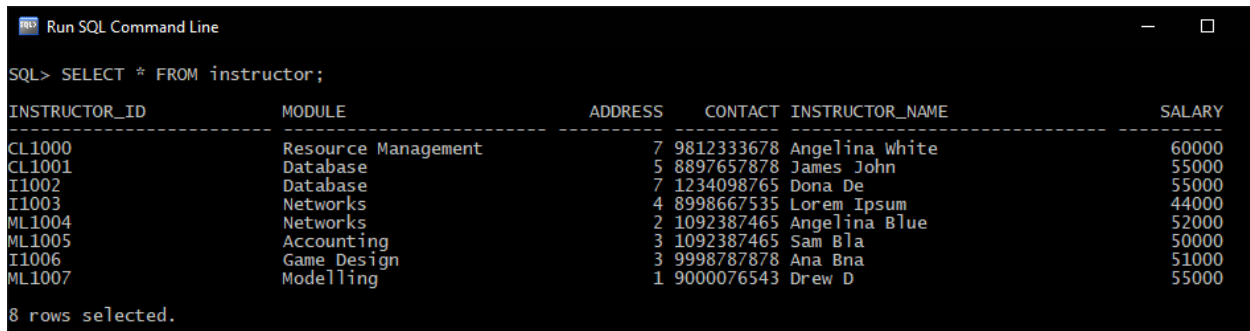
ON i.salary=j.salary

AND i.module=j.module

AND i.instructor\_id!=j.instructor\_id;

We join the instructor table with itself to compare the value in the columns. We check for same value for module and salary but check for different id. Checking for difference in id is necessary, else the query will return value for all the instructors, but in this case we only get values for which module and salary are same.

#### 4.14.2 Screenshot

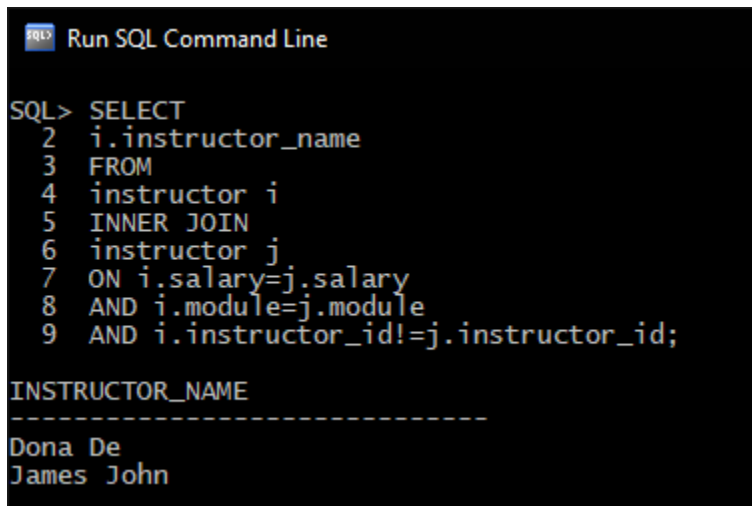


```
SQL> SELECT * FROM instructor;
```

INSTRUCTOR_ID	MODULE	ADDRESS	CONTACT	INSTRUCTOR_NAME	SALARY
CL1000	Resource Management	7	9812333678	Angelina White	60000
CL1001	Database	5	8897657878	James John	55000
II1002	Database	7	1234098765	Dona De	55000
II1003	Networks	4	8998667535	Lorem Ipsum	44000
ML1004	Networks	2	1092387465	Angelina Blue	52000
ML1005	Accounting	3	1092387465	Sam Bla	50000
II1006	Game Design	3	9998787878	Ana Bna	51000
ML1007	Modelling	1	9000076543	Drew D	55000

8 rows selected.

Figure 46: Data in instructor table



```
SQL> SELECT
2  i.instructor_name
3  FROM
4  instructor i
5  INNER JOIN
6  instructor j
7  ON i.salary=j.salary
8  AND i.module=j.module
9  AND i.instructor_id!=j.instructor_id;
```

INSTRUCTOR_NAME
Dona De
James John

Figure 47: Transaction query 4

Here only the instructors with same salary and module is displayed.

#### 4.15 List all the courses with the total number of students enrolled course name and the highest marks obtained.

##### 4.15.1

SELECT

course, COUNT(\*) "students enrolled",

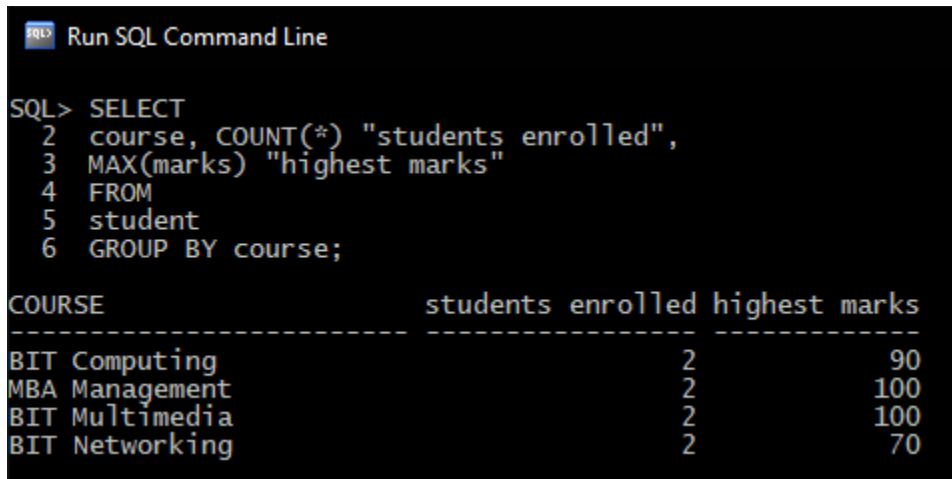
MAX(marks) "highest marks"

FROM

student

GROUP BY course;

##### 4.15.2 Screenshot



The screenshot shows a SQL Command Line window with the following query and results:

```
SQL> SELECT
2  course, COUNT(*) "students enrolled",
3  MAX(marks) "highest marks"
4  FROM
5  student
6  GROUP BY course;
```

COURSE	students enrolled	highest marks
BIT Computing	2	90
MBA Management	2	100
BIT Multimedia	2	100
BIT Networking	2	70

Figure 48: Transaction query 5

The query is performed on student table, COUNT is used to find the number of times a course appears, and MAX gives the maximum value for the given data.

## 4.16 List all the instructors who are also a course leader

### 4.16.1 Query

SELECT

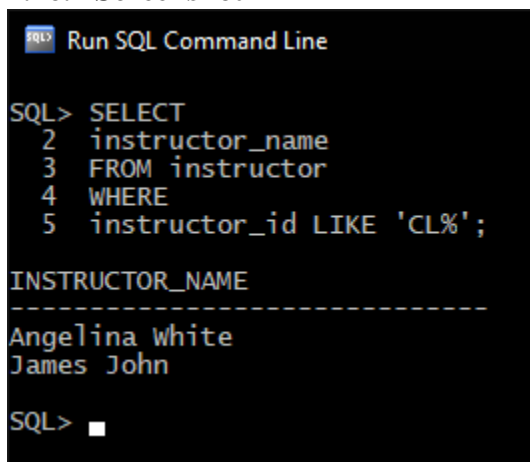
instructor\_name

FROM instructor

WHERE

instructor\_id LIKE 'CL%';

### 4.16.2 Screenshot



```
SQL> SELECT
2  instructor_name
3  FROM instructor
4  WHERE
5  instructor_id LIKE 'CL%';

INSTRUCTOR_NAME
-----
Angelina White
James John
SQL> ■
```

Figure 49: Transaction query 6

As per our business rules, instructors who are course leaders have CL in their id, so all the instructors were listed who had CL in the beginning of their id to find instructors who are also course leaders.

## Dump file creation, dropping tables

### Screenshot for dump file creation

```

Administrator: Command Prompt
D:\>exp coursework1/coursework1 file=coursework.dmp

Export: Release 11.2.0.2.0 - Production on Thu Dec 17 20:06:09 2020

Copyright (c) 1982, 2009, Oracle and/or its affiliates. All rights reserved.

Connected to: Oracle Database 11g Express Edition Release 11.2.0.2.0 - 64bit Production
Export done in WE8MSWIN1252 character set and AL16UTF16 NCHAR character set
server uses AL32UTF8 character set (possible charset conversion)

About to export specified users ...
. exporting pre-schema procedural objects and actions
. exporting foreign function library names for user COURSEWORK1
. exporting PUBLIC type synonyms
. exporting private type synonyms
. exporting object type definitions for user COURSEWORK1
About to export COURSEWORK1's objects ...
. exporting database links
. exporting sequence numbers
. exporting cluster definitions
. about to export COURSEWORK1's tables via Conventional Path...
. exporting table ADDRESS 7 rows exported
. exporting table CLASS 7 rows exported
. exporting table CONTACT 16 rows exported
. exporting table COURSE 7 rows exported
. exporting table INSTRUCTOR 8 rows exported
. exporting table MODULE 9 rows exported
. exporting table STUDENT 8 rows exported
. exporting synonyms
. exporting views
. exporting stored procedures
. exporting operators
. exporting referential integrity constraints
. exporting triggers
. exporting indextypes
. exporting bitmap, functional and extensible indexes
. exporting posttables actions
. exporting materialized views
. exporting snapshot logs
. exporting job queues
. exporting refresh groups and children
. exporting dimensions
. exporting post-schema procedural objects and actions
. exporting statistics
Export terminated successfully without warnings.

D:\>
D:\>

```

Figure 50: Creating dmp file

### Queries to drop tables

DROP TABLE instructor;

DROP TABLE student;

DROP TABLE address;

DROP TABLE contact;

DROP TABLE module;

DROP TABLE class;

DROP TABLE course;

```

Run SQL Command Line

SQL> DROP TABLE instructor;
Table dropped.

SQL> DROP TABLE student;
Table dropped.

SQL> DROP TABLE module;
Table dropped.

SQL> DROP TABLE contact;
Table dropped.

SQL> DROP TABLE class;
Table dropped.

SQL> DROP TABLE course;
Table dropped.

SQL>

```

Figure 51: Dropping tables

## **Chapter 5: Conclusion**

The learning experience was wonderful, many things were revised, and some new things were learnt during research. ERD's were formulated and normalization was carried out to sort out mistakes and improve the database.

The course work also helped in understanding design for a database, why we need normalization and what problems a poorly designed database can cause when implemented without checking for problems.

The experience in the coursework can be very useful in real life work scenario of manipulating data in a database. During the coursework one got to be familiar with different queries for extracting data and which query to use according to the scenario. The overall experience of the coursework was rewarding and a great exercise on the topics learnt so far.

## Chapter 6: References

### References

Carlos Coronel, S.M. (2016) *DATABASE SYSTEMS Design, Implementation and Management*. 12th ed. Boston: Cengage Learning.

Islington College. (2020) *Islington* [Online]. Available from: <https://islington.edu.np/about/> [Accessed 16 December 2020].

Oracle. (2016) *Database SQL Reference* [Online]. Available from: [https://docs.oracle.com/cd/B19306\\_01/server.102/b14200/statements\\_4010.htm](https://docs.oracle.com/cd/B19306_01/server.102/b14200/statements_4010.htm) [Accessed 16 December 2020].