CC5051NI Databases

100% Individual Coursework

Autumn 2024

Credit: 15 Semester Long Module

Student Name: Paras Bikram Adhikari Chhetri

London Met ID: 23048587

Assignment Submission Date: 23rd January 2025

Word Count: 8176 words

I confirm that I understand my coursework needs to be submitted online via My Second Teacher Classroom under the relevant module page before the deadline in order 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.

23048587 parasbikramadhikarichhetri.docx

Islington College,Nepal

Document Details

Submission ID
trn:oid:::3618:79849851

Submission Date
Jan 22, 2025, 5:56 PM GMT+5:45

Download Date
Jan 22, 2025, 7:09 PM GMT+5:45

File Name
23048587 parasbikramadhikarichhetri.docx

File Size
53.7 KB

87 Pages

6,995 Words

43,712 Characters

33% Overall Similarity

The combined total of all matches, including overlapping sources, for each database.

Match Groups

- **191** Not Cited or Quoted 30%
  Matches with neither in-text citation nor quotation marks
- **21** Missing Quotations 4%
  Matches that are still very similar to source material
- **0** Missing Citation 0%
  Matches that have quotation marks, but no in-text citation
- **0** Cited and Quoted 0%
  Matches with in-text citation present, but no quotation marks

Top Sources

15% Internet sources
1% Publications
30% Submitted works (Student Papers)

Integrity Flags

0 Integrity Flags for Review

Our system's algorithms look deeply at a document for any inconsistencies that would set it apart from a normal submission. If we notice something strange, we flag it for you to review.

A Flag is not necessarily an indicator of a problem. However, we'd recommend you focus your attention there for further review.

Table of Contents

Table of tables

Table of figures

1. Introduction

1.1 Introduction of Business and its Forte:

We are implementing an Oracle 11g SQL-based database system in support of one of their most innovative projects titled "E-Classroom Platform". This database system is designed basically for the effective handling/management of important features such as student, teacher, program, and module during academic life. With prime attention being paid to flexibility and dependability, the database provides ground to a platform that ensures smooth use not only for the teacher but for students as well, easing them into present informational dynamism in education.

Ms. Mary developed the “E Classroom Platform”, a sophisticated online learning system tailored for Premier International School to enhance the learning process, for students and empower educators with teaching tools needed for their roles in education delivery and administration tasks efficiency through upgrades, for academic operations and management functions.

The E Classroom Platform stands out for its capacity to combine all the aspects of education like students, teachers, academic programs and learning materials into a system. It provides an approach that serves, to the demand of education, by enabling flexible program development close monitoring of student advancements and encouraging organized learning through methodical distribution of educational resources.

Figure 1: premier school

1.2 Business activities and Operations:

The E Classroom Platform facilitates a variety of activities to support the college's academic and administrative processes. Listed below:

1) **Student Management:**
   Grow students in the academic program and follow the progress through modules and evaluation. You can access student's information, their assessment results, and detailed performance reports.
2) **Organization of Programs and Modules:**
   Offers a variety of academic programs like BSc in Computing, BSc in Networking, BSc in Multimedia, etc. each consisting of several modules. It allows sharing of some modules between programs for greater flexibility.
3) **Teacher Management:**
   Assigns teachers to specific modules to create assessments, post announcements and mark the student works.
4) **Assessment and Grading:**
   Creates assessments for specific modules with attributes like deadlines and weighting. Tracks assessment submissions and generate detailed results.
5) **Resource Management:**
   Provides learning resources for modules with sequential access rules to ensure structured learning. Resources include videos, documentation and tests.
6) **Communication:**
   Facilitates module specific announcements from faculty to keep students informed.
7) **Performance Tracking:**
   Creates detailed reports on student performance to help teachers and students track academic progress.

1.3 Business Rule:

The operational procedures of the platform give rise to specific business rules that influence the design and functionality of the database. Such as:

1) **Programs and Modules:**
   Each program includes multiple modules. Modules can belong to multiple programs.
2) **Student Enrollment:**
   A student can enroll in one program at a time. All mandatory modules and assessments must be completed in order to graduate.
3) **Sequential Resource Access:**
   Resources in a module must be completed in order. Students cannot access the next resource until the current one is marked “completed”.
4) **Assessments:**
   There are different assessments per module along with properties like ID, Title, Deadlines and Weightage. The results should include grades and feedback for students.
5) **Teacher assignments:**
   Teachers are responsible for all aspects related to the modules they are teaching from assessment to resources and communications.
6) **Performance Tracking:**
   Student marks are tracked and detailed performance reports are generated that can be accessed by the concerned teachers and the students themselves.
7) **Announcement:**
   An announcement must be tied to a specific module and can only be posted by assigned teachers.

2. Initial ERD

2.1 Entities and Attributes:

1) Student

Attributes	Description
Student_ID	Unique identifier of the student.
First_Name	First name of the student
Last_Name	Last name of the student
DOB	Date of birth of the student
Phone_No	Contact number of the student
Email	Email address of the student
Address	Where the student lives
Student_Age	How old is the student
Program_ID	It is the foreign key also the id of program where student is enrolled in

Table 1: student attribute

2) Program

Attributes	Description
Program_ID	Unique identifier of the program
Program_Name	Name of the program
Duration	Duration of the program
Start_Date	Date when the program starts
End_Date	Date when the program ends
Total_Credits	Total credits of the program

Program_Code	Code of the program
Program_Fee	The total cost of the program
Module_ID	It is a foreign key also the id of module that is in the program

Table 2: program attribute

3) Module

Attributes	Description
Module_ID	Unique identifier of the module
Module_Name	Name of the module
Credits	Credits of the module
Module_Duration	Duration of the module
Module_Level	Level of the module
Program_ID	It is a foreign key also the id of program where module is included

Table 3: module attribute

4) Teacher

Attributes	Description
Teacher_ID	Unique identifier of the teacher
Teacher_Name	Name of the teacher
Teacher_no	Contact number of the teacher
Teacher_Email	Email address of the teacher

Table 4: teacher attribute

5) Assessment

Attributes	Description
Assessment_ID	Unique identifier of the assessment
AS_Title	Title of the assessment
AS_Weightage	Weightage of the assessment how much marks the assessment carries
AS_Deadline	Deadline of the assessment submission
Module_ID	It is a foreign key also the id of module the assessment belongs to

Table 5: assessment attribute

6) Resources

Attributes	Description
Resource_ID	Unique identifier of the resource
R_Title	Title of the resource
R_Duration	The duration of the resources
R_Type	The type of the resources like videos, texts,etc.
Module_ID	It is a foreign key also the id of module the resources belong to

Table 6: resource attribute

7) Announcement

Attributes	Description
Announcement_ID	Unique identifier of the announcement
AN_Title	Title of the announcement
AN_Posted	The date when the announcement was posted

Module_ID	It is a foreign key also the id of module the announcement belongs to

Table 7: announcement attribute

8) Result

Attributes	Description
Result_ID	Unique identifier of the result
Marks_Obtained	The total marks obtained by students
Grade	The grade students got
Assessment_ID	It is a foreign key also the id of assessment the result belongs to
Student_ID	It is a foreign key also the id of the student the result belongs to

Table 8: result attribute

2.2 Entities and Relationships:

The entities and their relationships are given below:

- A program consists of multiple modules. (one to many)
- A module has multiple students enrolled. (many to many)
- One teacher can involve in one module only. (one to one)
- A module contains several assessments and resources. (one to many)
- A student has one results linked to assessments. (one to one)
- An announcement is linked to specific module. (one to many)

2.3 Assumptions:

1) Student management assumptions:
    - A student can only enroll in single program at a time
    - Students need to complete all the assessments of the modules they are enrolled in to graduate
    - Students can only enroll in modules that is on their program

2) Program and Module assumptions:
    - A program can include many modules.
    - A module can belong to multiple programs.
    - Each module has their own assessments and results.

3) Teacher management assumptions:
    - A module has only one teacher.
    - Teacher is responsible for everything of their module. Like, assessments, announcements, and resources.

4) Assessments assumptions:
    - Module can have multiple assessments.
    - One assessment can have only one result.
    - One assessment can’t be linked to multiple modules.

5) Resource management assumptions:
    - One module can have multiple resources.
    - A student can get access to next resource only if he/she completes the current resource.
    - One resource can’t be linked to multiple assessments.

6) Announcement assumptions;
    - Announcement are linked to their specific module and only can be posted by the module's respective teacher.

7) Performance tracking assumptions:
    - The marks for each assessment of the students are tracked detailly.
    - The detailed generated reports can be accessed by both the teacher and students who are enrolled in the module.

2.4 Entity Relationship Diagram:

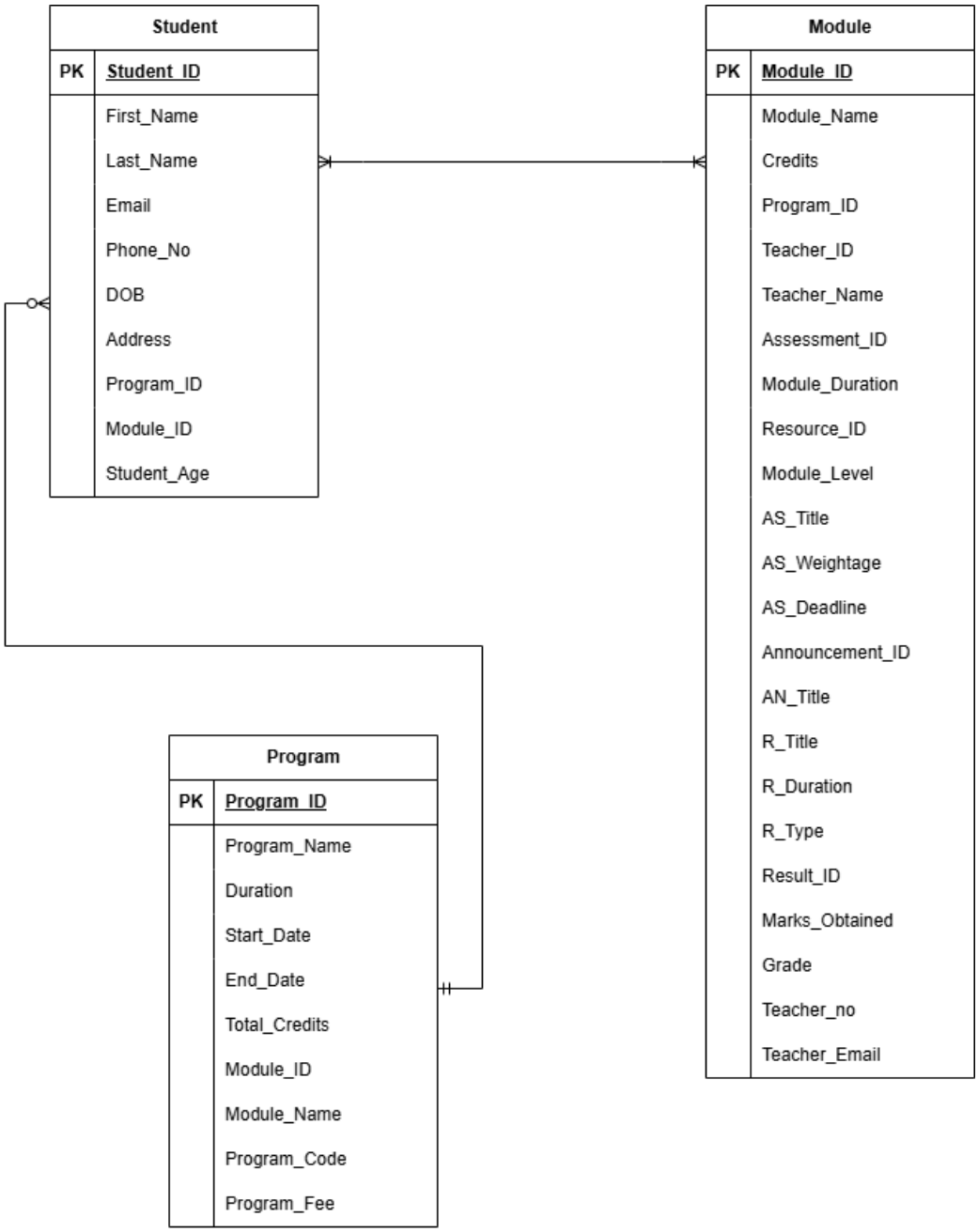

Figure 2: ERD

3. Normalization:

Normalization can be defined as the process of organizing data in a database to reduce redundancy. Redundancy in relation to the data may cause anomalies such as insertion, deletion, and update anomalies. Anomalies are undesirable side effects that can occur if relations are not in proper norma form. So, Normalization basically helps to minimize the redundancy in relations. Some Advantages of Normalization are:

- Reduce data redundancy
- Improved data consistency
- Simplified database design
- Improved query performance
- Easier database maintenance

Overall, using normalization in DBMS helps to improve data quality, increase database efficiency, and simplify database design and maintenance (geeksforgeeks, 2024).

There are 4 stages of Normalization. They are UNF, 1NF, 2NF, and 3NF. We need to undergo these 4 stages for the proper Normalization of the initial ERD. So, let's start with UNF:

3.1 UNF:

Un-normalized Form (UNF) can be defined as the simplest database model. It is also known as non-first normal form (NF2). It is the first stage of normalization. In this stage, the data may have dependencies and duplicate information that can cause issues in the future. So, in UNF we separate repeating groups and put them inside separately inside curly braces (geeksforgeeks, 2020). The UNF is given below:

Student (<u>Student_ID,</u> First_Name, Last_Name, Email, Phone_no, DOB, Address, Student_Age, Program_ID, Program_Name, Duration, Start_Date, End_Date, Program_Code, Program_Fee, **{** Module_ID, Module_Name, Credits, Teacher_ID, Teacher_Name, Teacher_no, Teacher_Email, Module_Duration, Module_Level, **{** Assessment_ID, AS_Weightage, AS_Deadline, AS_Title, Result_ID, Marks_Obtained, Grade**},** **{**Resource_ID, R_Title, R_Duration, R_Type**},** **{** Announcement_ID, AN_Title, AN_Date**}}).**

3.2 1NF

First Normal Form (1NF) is the second stage of the normalization process. A table is said to be in 1NF if all the non-key columns show functional dependency on the primary key components. The primary key must be defined.

For converting UNF to 1NF we must remove the outer most repeating group to form a new relation and name it and choose unique identifier for the newly formed relation. Here the repeating group of Student entity is separated and separate entities are formed. There is repeating group inside of the repeating group so we need to separate them too (geeksforgeeks, 2025). The 1NF is given below:

Student-1 (**Student_ID,** First_Name, Last_Name, Email, Phone_no, DOB, Address, Student_Age, Program_ID, Program_Name, Duration, Start_Date, End_Date, Program_Code, Program_Fee)

Module-1 (**Module_ID, Student_ID*,** Module_Name, Credits, Module_Duration, Module_Level, Teacher_ID, Teacher_Name, Teacher_no, Teacher_Email)

Assessment-1 (Assessment_ID, Module_ID*, Student_ID* AS_Title, AS_Weightage, AS_Deadline, Result_ID, Marks_Obtained, Grade)

Resource-1 (Resource_ID, Module_ID*, Student_ID*, R_Title, R_Duration, R_Type)

Announcement-1 (Announcement_ID, Module_ID*, Student_ID* AN_Title, AN_Date)

3.3 2NF

After 1NF comes Second Normal Form (2NF). It is the third stage of the normalization process. In this stage, our task is to remove the partial functional dependencies from the relation. For converting 1NF to 2NF we must check through every entity to see if there are any partial dependencies or not.

First let's check for the Student entity:

Student-1 (**Student_ID,** First_Name, Last_Name, Email, Phone_no, DOB, Address, Student_Age, Program_ID, Program_Name, Duration, Start_Date, End_Date, Program_Code, Program_Fee)

Student entity doesn't consist of any composite key. It only consists of one primary key which is Student_ID. So, there is no partial dependencies in this entity. No changes needed to be done in this entity.

Now, for the Module entity:

Module-1 (**Module_ID, Student_ID*,** Module_Name, Credits, Module_Duration, Module_Level, Teacher_ID, Teacher_Name, Teacher_no, Teacher_Email)

As there is a composite key in this entity, there may be partial dependency. So, some changes needed to be done to this entity to remove the partial dependencies. From this Module entity we can create a new bridging entity named Student_Module which contains both Student_ID and Module_ID.

Checking partial dependencies

Module_ID, Student_ID → no attributes

Student_ID → no attributes

Module_ID → (**Module_ID,** Module_Name, Credits, Module_Duration, Module_Level, Teacher_ID, Teacher_Name, Teacher_no, Teacher_Email)

Now, for the Assessment entity:

Assessment-1 (Assessment_ID, Module_ID*, Student_ID* AS_Title, AS_Weightage, AS_Deadline, Result_ID, Marks_Obtained, Grade)

As there is a composite key in Assessment entity too, there may be partial dependency. So, some changes needed to be done to this entity to remove the partial dependencies. From this Assessment entity we can create a new bridging entity named Module_Assessment which contains Module_ID, Student_ID and Assessment_ID.

Checking partial dependencies

Assessment_ID → **(<u>Assessment_ID,</u>** AS_Title, AS_Weightage, AS_Deadline, Result_ID, Marks_Obtained, Grade)

Module_ID → no attributes

Student_ID → no attributes

Assessment_ID, Module_ID → no attributes

Student_ID, Module_ID → no attributes

Assessment_ID, Student_ID → no attributes

Now, for the resource entity:

Resource-1 (<u>Resource_ID, Module_ID*, Student_ID*,</u> R_Title, R_Duration, R_Type)

As there is a composite key in Resource entity too, there may be partial dependency. So, some changes needed to be done to this entity to remove the partial dependencies. From this Resource entity we can create a new bridging entity named Module_Resource which contains Module_ID, Student_ID and Resource_ID.

Checking partial dependencies

Student_ID → no attributes

Module_ID → no attributes

Resource_ID, Module_ID → no attributes

Resource_ID, Student_ID → no attributes

Student_ID, Module_ID → no attributes

Resource_ID → **(Resource_ID,** R_Title, R_Duration, R_Type)

Now, for the announcement entity

Announcement-1 (Announcement_ID, Module_ID*, Student_ID* AN_Title, AN_Date)

As there is a composite key in Announcement entity too, there may be partial dependency. So, some changes needed to be done to this entity to remove the partial dependencies. From this Announcement entity we can create a new bridging entity named Module_Announcement which contains Module_ID, Student_ID and Announcement_ID.

Let's check partial dependencies

Announcement_ID → **(Announcement_ID,** Module_ID*, AN_Title, AN_Date)

Student_ID → no attributes

Module_ID → no attributes

Announcement_ID, Module_ID → no attributes

Student_ID, Module_ID → no attributes

Announcement_ID, Student_ID → no attributes

Now, the final entities after 2NF are given below:

Student-2 (**Student_ID,** First_Name, Last_Name, Email, Phone_no, DOB, Address, Student_Age, Program_ID, Program_Name, Duration, Start_Date, End_Date, Program_Code, Program_Fee)

Student_Module-2 (Student_ID*, Module_ID*)

Module-2 (Module_ID, Module_Name, Credits, Module_Duration, Module_Level, Teacher_ID, Teacher_Name, Teacher_no, Teacher_Email)

Module_Assessment-2 (Assessment_ID*, Module_ID*, Student_ID*)

Assessment-2 (Assessment_ID, AS_Title, AS_Weightage, AS_Deadline, Result_ID, Marks_Obtained, Grade)

Module_Resource-2 (Resource_ID*, Module_ID*, Student_ID*)

Resource-2 (Resource_ID, R_Title, R_Duration, R_Type)

Module_Announcement-2 (Announcement_ID*, Module_ID*, Student_ID*)

Announcement-2 (Announcement_ID, AN_Title**,** AN_Date, Module_ID*)

3.4 3NF

After completing 2NF comes 3NF. Third Normal Form (3NF) is the third stage of the normalization process. In this stage the main target is to remove the transitive dependency. No transitive dependency should remain after 3NF.

First and foremost, in general transitive dependency in database is defined as a condition where an attribute depends upon another attribute, while that depends upon the primary key itself. A transitive dependency is a big problem as it may lead to data issues such as update, insertion, and deletion anomalies (Ouko, 2024).

Now, we need to check the transitive dependency and separate the attributes into new entities respectively. First let's check in the Student entity:

Student-2 (**Student_ID,** First_Name, Last_Name, Email, Phone_no, DOB, Address, Student_Age, Program_ID, Program_Name, Duration, Start_Date, End_Date, Program_Code, Program_Fee)

Here, in the student entity we can see that there is information of program like program name, duration, fees, start and end date, etc. those attributes depend on program ID. So, we need to separate those attributes and create a new entity named program.

Now, let's check in the Module entity:

Module-2 (Module_ID, Module_Name, Credits, Module_Duration, Module_Level, Teacher_ID, Teacher_Name, Teacher_no, Teacher_Email)

Here, in the module entity we can see that there is information of teacher like teacher name, teacher number, teacher email, etc. those attributes depend on teacher ID. So, we need to separate those attributes and create a new entity named teacher.

Now, let's check in the assessment entity:

Assessment-2 (Assessment_ID, AS_Title, AS_Weightage, AS_Deadline, Result_ID, Marks_Obtained, Grade)

Here, in the assessment entity we can see that there is information of result like marks obtained and grade that depends on result ID. So, we need to separate those attributes and create a new entity named result.

Now, talking about student_module, module_resource, and module_assessment:

Student_Module-2 (Student_ID*, Module_ID*)

Module_Assessment-2 (Assessment_ID*, Module_ID*, Student_ID*)

Module_Resource-2 (Resource_ID*, Module_ID*, Student_ID*)

Module_Announcement-2 (Announcement_ID*, Module_ID*, Student_ID*)

There are no non-key attributes in the above entities. So, we don't need to make any changes in the above entities as there is no transitive dependencies.

Same goes for resource and announcement entities.

Resource-2 (Resource_ID, R_Title, R_Duration, R_Type)

Announcement-2 (Announcement_ID, AN_Title**,** AN_Date , Module_ID***)**

As the attributes of the both entities depends upon their respective primary key we don't need to make changes to those entities. So, the tables should not be checked for transitive dependencies.

Now after making changes final entities after 3NF are given below:

Student-3 (**Student_ID,** First_Name, Last_Name, Email, Phone_no, DOB, Address, Student_Age, Program_ID*)

Student_Module-3 (Student_ID*, Module_ID*)

Teacher-3 (Teacher_ID, Teacher_Name, Teacher_no, Teacher_Email)

Module-3 (**Module_ID,** Module_Name, Credits, Module_Duration, Module_Level, Teacher_ID*)

Module_Assessment-3 (Assessment_ID*, Module_ID*, Student_ID*)

Assessment-3 (Assessment_ID, AS_Title, AS_Weightage, AS_Deadline)

Result-3 (Result_ID, Marks_Obtained, Grade, Assessment_ID*)

Program-3 (Program_ID, Program_Name, Duration, Start_Date, End_Date, Program_Code, Program_Fee)

Module_Resource-3 (Resource_ID*, Module_ID*, Student_ID*)

Resource-3 (Resource_ID, R_Title, R_Duration, R_Type)

Module_Announcement-3 (**Announcement_ID*, Module_ID*, Student_ID*)**

Announcement-3 (Announcement_ID, AN_Title**,** AN_Date, Module_ID***)**

4. Data Dictionary

A Data Dictionary is a collection of a set of names, definitions, and attributes about data elements that could be in use or be captured into a database, information system, or part of some research undertaking. It describes the meaning and function of data elements within a project context. It specifies how these are to be represented and how they should be used. More generally speaking, a Data Dictionary provides metadata about data elements. A Data Dictionary can contain metadata that may be used to define data element's scope and characteristics and the rules for their use and implementation (LIBRARY, 2024).

Talking about the meta data, it can be defined as the data that describes the database's structure, contents, and context which will help you to understand the connectivity of a particular data set (Atlan, 2024).

The data dictionary for the entities are given below:

4.1 Data dictionary for Student

S.NO	Attribute Name	Data Type	Size	Constraint
1.	Student ID	Number	10	Primary key
2.	First Name	Character	20	Not Null
3.	Last Name	Character	20	Not Null
4.	Email	Character	30	Unique
5.	DOB	Date	N/A	Not Null
6.	Address	Character	30	Not Null

7.	Student Age	Number	05	Not Null
8.	Phone no	Number	10	Not Null
9.	Program ID	Number	15	Foreign Key

Table 9: data dictionary student

4.2 Data Dictionary for Student_Module

S.NO	Attribute Name	Data Type	Size	Constraint	Composite Constraint
1.	Student ID	Number	10	Foreign Key	Primary Key
2.	Module ID	Number	15	Foreign Key	

Table 10: data dictionary student_module

4.3 Data Dictionary for Teacher

S.NO	Attribute Name	Data Type	Size	Constraint
1.	Teacher ID	Number	10	Primary Key
2.	Teacher Name	Character	40	Not Null
3.	Teacher no	Number	10	Not Null
4.	Teacher email	Character	30	Unique

Table 11: data dictionary teacher

4.4 Data Dictionary for Module

S.NO	Attribute Name	Data Type	Size	Constraint
1.	Module ID	Number	10	Primary Key
2.	Module Name	Character	40	Not Null
3.	Credits	Number	05	Not Null
4.	Module Duration	Character or Number	10	Not Null
5.	Module Level	Character	05	Not Null
6.	Teacher ID	Number	10	Foreign Key

Table 12: data dictionary module

4.5 Data Dictionary for Module_Assessment

S.NO	Attribute Name	Data Type	Size	Constraint	Composite Constraint
1.	Module ID	Number	10	Foreign Key	Primary Key
2.	Assessment ID	Number	10	Foreign Key	
3.	Student ID	Number	10	Foreign Key	

Table 13: data dictionary module_assessment

4.6 Data Dictionary for Assessment

S.NO	Attribute Name	Data Type	Size	Constraint
1.	Assessment ID	Number	10	Primary Key
2.	AS Title	Character	50	Not Null
3.	AS Weightage	Number	5	Not Null
4.	AS Deadline	Date	N/A	Not Null

Table 14: data dictionary assessment

4.7 Data Dictionary for Result

S.NO	Attribute Name	Data Type	Size	Constraint
1.	Result ID	Number	10	Primary Key
2.	Mark Obtained	Number	05	Not Null
3.	Grade	Character	05	
4.	Assessment ID	Number	10	Foreign Key

Table 15: data dictionary result

4.8 Data Dictionary for Module_Resource

S.NO	Attribute Name	Data Type	Size	Constraint	Composite Constraint
1.	Resource ID	Number	10	Foreign Key	Primary Key
2.	Module ID	Number	10	Foreign Key	
3.	Student ID	Number	10	Foreign Key	

Table 16: data dictionary module_resource

4.9 Data Dictionary for Program

S.NO	Attribute Name	Data Type	Size	Constraint
1.	Program ID	Number	10	Primary Key
2.	Program Name	Character	40	Not Null
3.	Duration	Character or Number	10	Not Null
4.	Start Date	Date	N/A	Not Null
5.	End Date	Date	N/A	Not Null
6.	Program Code	Number	10	Not Null
7.	Program Fee	Character	30	Not Null

Table 17: data dictionary program

4.10 Data Dictionary for Resource

S.NO	Attribute Name	Data Type	Size	Constraint
1.	Resource ID	Number	10	Primary Key
2.	R Title	Character	40	Not Null
3.	R Duration	Character or Number	10	Not Null
4.	R Type	Character	30	Not Null

Table 18: data dictionary resources

4.11 Data Dictionary for Announcement

S.NO	Attribute Name	Data Type	Size	Constraint
1.	Announcement ID	Number	10	Primary Key
2.	AN Title	Character	40	Not Null
3.	Module ID	Number	10	Foreign Key
4.	AN Date	Date	N/A	NOY NULL

Table 19: data dictionary announcement

4.12 Data Dictionary for Module_Announcement

S.NO	Attribute Name	Data Type	Size	Constraint	Composite Constraint
1.	Announcement ID	Number	10	Foreign Key	Primary Key
2.	Module ID	Number	10	Foreign Key	
3.	Student ID	Number	10	Foreign Key	

Table 20: data dictionary module_announcement

5. Final ERD

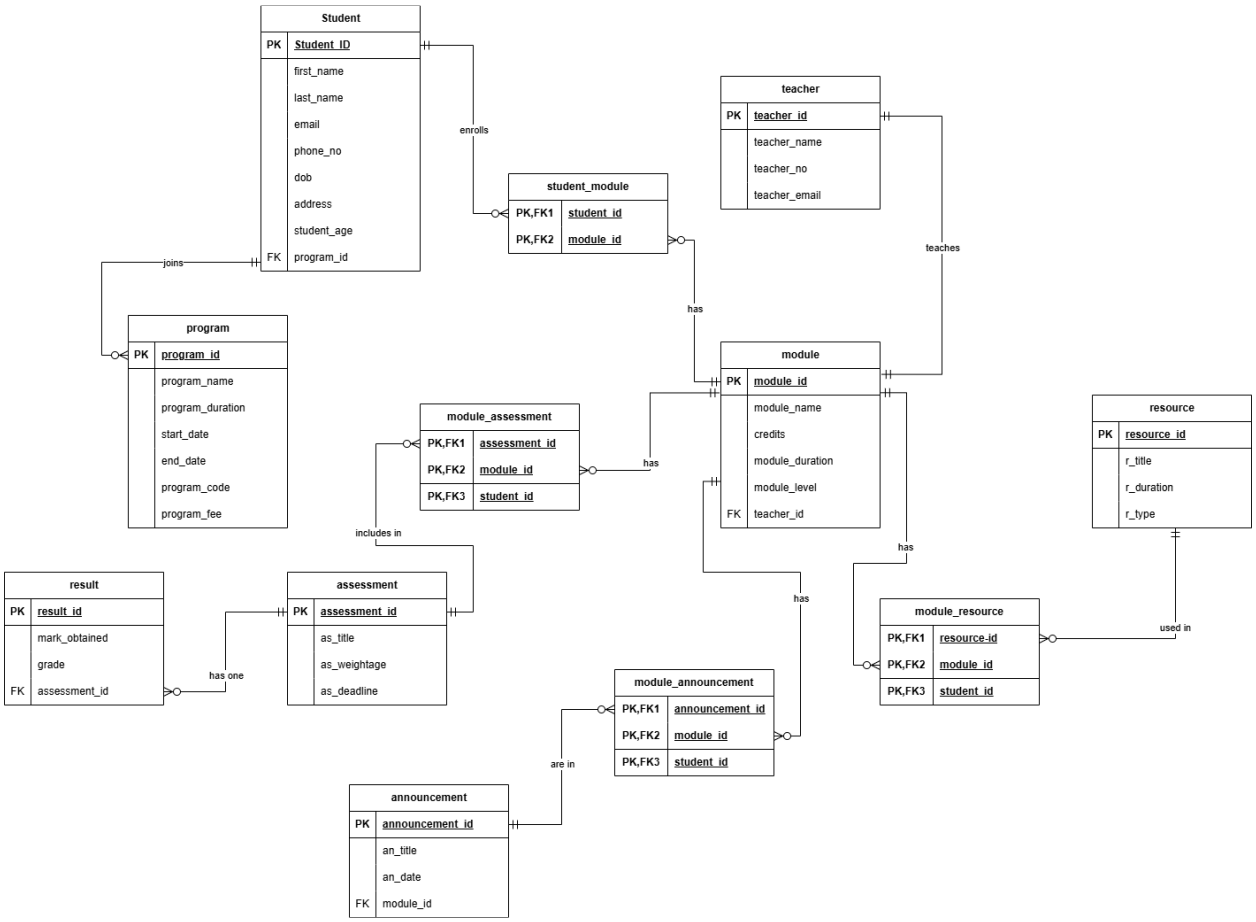

Figure 3: final ERD

6. Implementation

6.1 Creating tables

As we just completed our normalization process. Now, we need to start the implementation part by creating tables for the entities we obtained after completing normalization. First, we need to create a user and grant privilege. Then, in oracle, table are created by using CREATE TABLE command. By using this command, we need to create table for each entity. So, let's start by creating a user then we shall continue with creating tables.

6.1.1 Creating User

CONNECT system

Password: test123

CREATE USER ParasBikramAdhikari **IDENTIFIED BY** 23048587;

GRANT CONNECT, **RESOURCE** to ParasBikramAdhikari;

CONNECT ParasBikramAdhikari/23048587

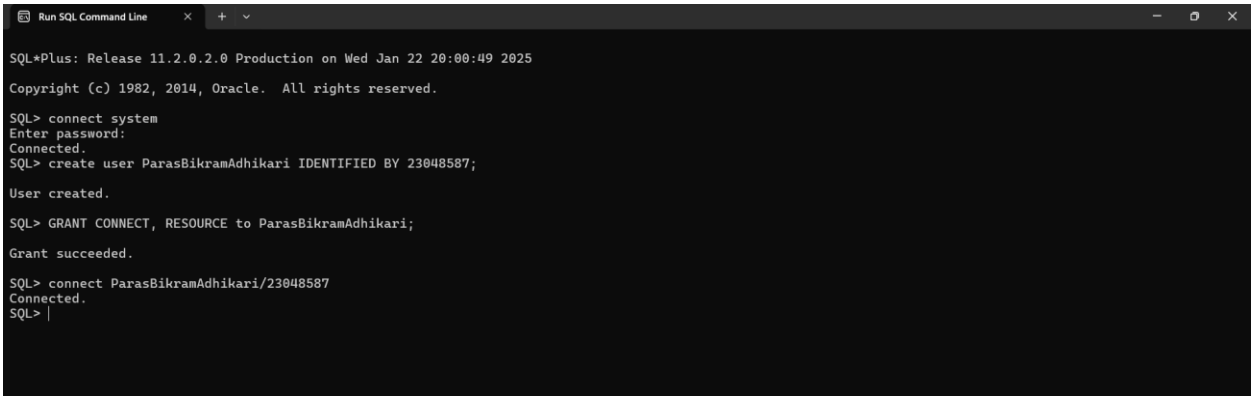

Figure 4: creating user

6.1.2 Creating Program table

CREATE TABLE Programm

(Program_ID Number(10) **PRIMARY KEY**,

Program_Name Character(40) **NOT NULL**,

Duration Character(10) **NOT NULL**,

Start_Date Date **NOT NULL**,

End_Date Date **NOT NULL**,

Program_Code Number(10) **NOT NULL**,

Program_Fee Character(30) **NOT NULL**);

```
SQL> connect ParasBikramAdhikari/23048587
Connected.
SQL> CREATE TABLE Programm
  2    (Program_ID Number(10) PRIMARY KEY,
  3    Program_Name Character(40) NOT NULL,
  4    Duration Character(10) NOT NULL,
  5    Start_Date Date NOT NULL,
  6    End_Date Date NOT NULL,
  7    Program_Code Number(10) NOT NULL,
  8    Program_Fee Character(30) NOT NULL);

Table created.

SQL>
```

Figure 5: creating program table

Desc program;

```
SQL> desc programm;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 PROGRAM_ID                                NOT NULL NUMBER(10)
 PROGRAM_NAME                              NOT NULL CHAR(40)
 DURATION                                  NOT NULL CHAR(10)
 START_DATE                                NOT NULL DATE
 END_DATE                                  NOT NULL DATE
 PROGRAM_CODE                              NOT NULL NUMBER(10)
 PROGRAM_FEE                               NOT NULL CHAR(30)

SQL>
```

Figure 6: description program table

6.1.3 Creating Student Table

```
CREATE TABLE Student (
   student_id NUMBER(10) PRIMARY KEY,
   first_name CHARACTER(40) NOT NULL,
   last_name CHARACTER(40) NOT NULL,
   email CHARACTER(30) UNIQUE,
   phone_no NUMBER(10) NOT NULL,
   DOB DATE NOT NULL,
   address CHARACTER(30) NOT NULL,
   student_age NUMBER(5) NOT NULL,
   program_id NUMBER(10) NOT NULL,
   CONSTRAINT fk_Programm
      FOREIGN KEY (program_id)
      REFERENCES Programm(PROGRAM_ID));
```

Figure 7: creating student table

Desc Student;

```
SQL> desc student;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 STUDENT_ID                                NOT NULL NUMBER(10)
 FIRST_NAME                                NOT NULL CHAR(40)
 LAST_NAME                                 NOT NULL CHAR(40)
 EMAIL                                              CHAR(30)
 PHONE_NO                                  NOT NULL NUMBER(10)
 DOB                                       NOT NULL DATE
 ADDRESS                                   NOT NULL CHAR(30)
 STUDENT_AGE                               NOT NULL NUMBER(5)
 PROGRAM_ID                                NOT NULL NUMBER(10)

SQL>
```

Figure 8: description student table

6.1.4 Creating Teacher Table

CREATE TABLE Teacher (

teacher_id NUMBER(10) **PRIMARY KEY**,

teacher_name CHARACTER(50) **NOT NULL**,

teacher_no NUMBER(10) **NOT NULL**,

teacher_email CHARACTER(30) **UNIQUE**);

```
SQL> CREATE TABLE Teacher (
  2      teacher_id NUMBER(10) PRIMARY KEY,
  3      teacher_name CHARACTER(50) NOT NULL,
  4      teacher_no NUMBER(10) NOT NULL,
  5      teacher_email CHARACTER(30) UNIQUE );

Table created.

SQL>
```

Figure 9: creating teacher table

Desc teacher;

```
SQL> desc teacher;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 TEACHER_ID                                NOT NULL NUMBER(10)
 TEACHER_NAME                              NOT NULL CHAR(50)
 TEACHER_NO                                NOT NULL NUMBER(10)
 TEACHER_EMAIL                                      CHAR(30)

SQL>
```

Figure 10: description teacher table

6.1.5 Creating Module Table

CREATE TABLE Module (

module_id NUMBER(10) **PRIMARY KEY**,

module_name CHARACTER(40) **NOT NULL**,

credits NUMBER(5) **NOT NULL**,

module_duration CHARACTER(10) **NOT NULL**,

module_level NUMBER(05) **NOT NULL**,

teacher_id NUMBER(10) **NOT NULL**,

CONSTRAINT fk_Teacher

FOREIGN KEY (teacher_id)

REFERENCES Teacher(teacher_id));

```
SQL> CREATE TABLE Module (
  2      module_id NUMBER(10) PRIMARY KEY,
  3      module_name CHARACTER(40) NOT NULL,
  4      credits NUMBER(5) NOT NULL,
  5      module_duration CHARACTER(10) NOT NULL,
  6      module_level NUMBER(05) NOT NULL,
  7      teacher_id NUMBER(10) NOT NULL,
  8      CONSTRAINT fk_Teacher
  9          FOREIGN KEY (teacher_id)
 10          REFERENCES Teacher(teacher_id));

Table created.

SQL>
```

Figure 11: creating module table

Desc module;

```
SQL> desc module;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 MODULE_ID                                 NOT NULL NUMBER(10)
 MODULE_NAME                               NOT NULL CHAR(40)
 CREDITS                                   NOT NULL NUMBER(5)
 MODULE_DURATION                           NOT NULL CHAR(10)
 MODULE_LEVEL                              NOT NULL NUMBER(5)
 TEACHER_ID                                NOT NULL NUMBER(10)

SQL>
```

Figure 12: description module table

6.1.6 Creating Assessment Table

CREATE TABLE Assessment (

assessment_id NUMBER(10) **PRIMARY KEY,**

as_title CHARACTER(50) **NOT NULL**,

as_weightage NUMBER(5) **NOT NULL**,

as_deadline DATE **NOT NULL**);

```
SQL> CREATE TABLE Assessment (
  2      assessment_id NUMBER(10) PRIMARY KEY,
  3      as_title CHARACTER(50) NOT NULL,
  4      as_weightage NUMBER(5) NOT NULL,
  5      as_deadline DATE NOT NULL);

Table created.

SQL>
```

Figure 13: creating assessment table

Desc assessment;

```
SQL> desc assessment;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 ASSESSMENT_ID                             NOT NULL NUMBER(10)
 AS_TITLE                                  NOT NULL CHAR(50)
 AS_WEIGHTAGE                              NOT NULL NUMBER(5)
 AS_DEADLINE                               NOT NULL DATE

SQL>
```

Figure 14: description assessment table

6.1.7 Creating Result Table

CREATE TABLE Result (

result_id NUMBER(10) **PRIMARY KEY,**

mark_obtained NUMBER(5) **NOT NULL**,

grade CHARACTER(5) **NOT NULL**,

assessment_id NUMBER(10) **NOT NULL**,

CONSTRAINT fk_Assessment

FOREIGN KEY (assessment_id)

REFERENCES Assessment(assessment_id));

```
SQL> CREATE TABLE Result (
  2      result_id NUMBER(10) PRIMARY KEY,
  3      mark_obtained NUMBER(5) NOT NULL,
  4      grade CHARACTER(5) NOT NULL,
  5      assessment_id NUMBER(10) NOT NULL,
  6      CONSTRAINT fk_Assessment
  7          FOREIGN KEY (assessment_id)
  8          REFERENCES Assessment(assessment_id));

Table created.
```

Figure 15: creating result table

Desc result;

```
SQL> desc result;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 RESULT_ID                                 NOT NULL NUMBER(10)
 MARK_OBTAINED                             NOT NULL NUMBER(5)
 GRADE                                     NOT NULL CHAR(5)
 ASSESSMENT_ID                             NOT NULL NUMBER(10)

SQL>
```

Figure 16: description result table

6.1.8 Create Announcement Table

CREATE TABLE Announcement (
announcement_id NUMBER(10) **PRIMARY KEY**,
an_title CHARACTER(40) **NOT NULL**,
module_id NUMBER(10) **NOT NULL**,
an_date DATE **NOT NULL**,
CONSTRAINT fk_module_M **FOREIGN KEY** (module_id) **REFERENCES** Module(module_id)
);

```
SQL> CREATE TABLE Announcement (
  2      announcement_id NUMBER(10) PRIMARY KEY,
  3      an_title CHARACTER(40) NOT NULL,
  4      module_id NUMBER(10) NOT NULL,
  5      an_date DATE NOT NULL,
  6      CONSTRAINT fk_module_M FOREIGN KEY (module_id) REFERENCES Module(module_id)
  7  );

Table created.

SQL>
```

Figure 17: creating announcement table

Desc announcement;

```
SQL> set linesize 150;
SQL> desc announcement;
 Name                                                                                  Null?    Type
 ------------------------------------------------------------------------------------- -------- ----------------------------------------------------
 ANNOUNCEMENT_ID                                                                       NOT NULL NUMBER(10)
 AN_TITLE                                                                              NOT NULL CHAR(40)
 MODULE_ID                                                                             NOT NULL NUMBER(10)

SQL>
```

Figure 18: description announcement table

6.1.9 Creating Resources table

CREATE TABLE Resources (

resource_id NUMBER(10) **PRIMARY KEY,**

r_title CHARACTER(40) **NOT NULL**,

r_duration CHARACTER(20) **NOT NULL**,

r_type CHARACTER(20) **NOT NULL**);

```
SQL> CREATE TABLE Resources (
  2      resource_id NUMBER(10) PRIMARY KEY,
  3      r_title CHARACTER(40) NOT NULL,
  4      r_duration CHARACTER(20) NOT NULL,
  5      r_type CHARACTER(20) NOT NULL);

Table created.

SQL>
```

Figure 19: creating resource table

Desc resources;

```
SQL> desc resources;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 RESOURCE_ID                               NOT NULL NUMBER(10)
 R_TITLE                                   NOT NULL CHAR(40)
 R_DURATION                                NOT NULL CHAR(20)
 R_TYPE                                    NOT NULL CHAR(20)

SQL>
```

Figure 20: description resources table

6.1.10 Creating Student_Module Table

CREATE TABLE Student_Module (

student_id NUMBER(10) **NOT NULL**,

module_id NUMBER(10) **NOT NULL**,

PRIMARY KEY (student_id, module_id),

CONSTRAINT fk_Student

FOREIGN KEY (student_id)

REFERENCES Student(student_id),

CONSTRAINT fk_Module

FOREIGN KEY (module_id)

REFERENCES Module(module_id));

```
SQL> CREATE TABLE Student_Module (
  2      student_id NUMBER(10) NOT NULL,
  3      module_id NUMBER(10) NOT NULL,
  4      PRIMARY KEY (student_id, module_id),
  5      CONSTRAINT fk_Student
  6          FOREIGN KEY (student_id)
  7          REFERENCES Student(student_id),
  8      CONSTRAINT fk_Module
  9          FOREIGN KEY (module_id)
 10          REFERENCES Module(module_id));

Table created.

SQL>
```

Figure 21: creating student_module table

Desc student_module;

```
SQL> desc student_module;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 STUDENT_ID                                NOT NULL NUMBER(10)
 MODULE_ID                                 NOT NULL NUMBER(10)

SQL>
```

Figure 22: description student_module table

6.1.11 Creating Module_Assessment table

```
CREATE TABLE Module_Assessment (
    assessment_id NUMBER(10),
    module_id NUMBER(10),
    student_id NUMBER(10),
    PRIMARY KEY (assessment_id, module_id, student_id),
    CONSTRAINT fk_Assessment_Module
        FOREIGN KEY (assessment_id)
        REFERENCES Assessment(assessment_id),
    CONSTRAINT fk_Module_Module
        FOREIGN KEY (module_id)
        REFERENCES Module(module_id),
    CONSTRAINT fk_Student_Module
        FOREIGN KEY (student_id)
        REFERENCES Student(student_id));
```

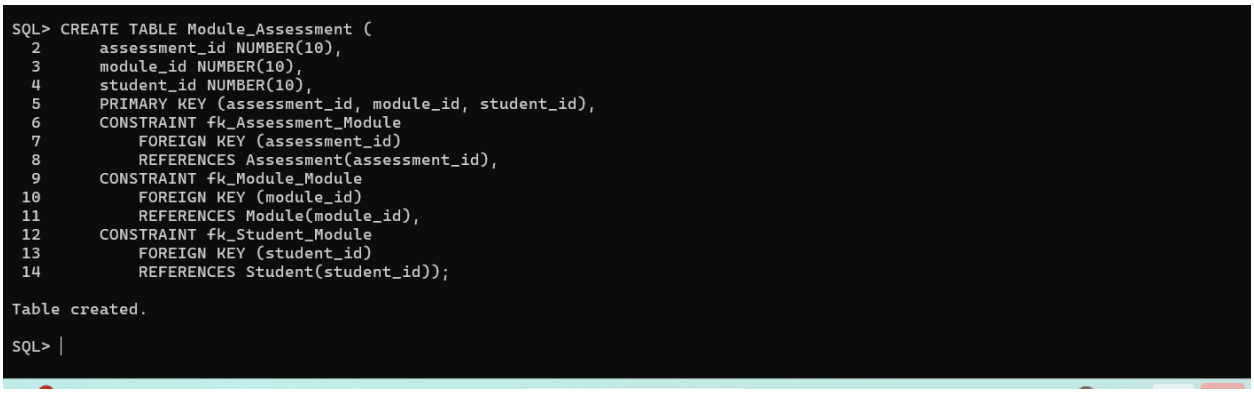

Figure 23: creating module_assessment table

Desc module_assessment;

```
SQL> desc module_assessment;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 ASSESSMENT_ID                             NOT NULL NUMBER(10)
 MODULE_ID                                 NOT NULL NUMBER(10)
 STUDENT_ID                                NOT NULL NUMBER(10)

SQL>
```

Figure 24: description module_assessment table

6.1.12 Creating module_resource table

```
CREATE TABLE Module_Resources (
   resource_id NUMBER(10),
   module_id NUMBER(10),
   student_id NUMBER(10),
   PRIMARY KEY (resource_id, module_id, student_id),
   CONSTRAINT fk_Resource_Resource
      FOREIGN KEY (resource_id)
      REFERENCES Resources(resource_id),
   CONSTRAINT fk_Module_Resource
      FOREIGN KEY (module_id)
      REFERENCES Module(module_id),
   CONSTRAINT fk_Student_Resource
      FOREIGN KEY (student_id)
      REFERENCES Student(student_id)
);
```

```
SQL> CREATE TABLE Module_Resources (
  2      resource_id NUMBER(10),
  3      module_id NUMBER(10),
  4      student_id NUMBER(10),
  5      PRIMARY KEY (resource_id, module_id, student_id),
  6      CONSTRAINT fk_Resource_Resource
  7          FOREIGN KEY (resource_id)
  8          REFERENCES Resources(resource_id),
  9      CONSTRAINT fk_Module_Resource
 10          FOREIGN KEY (module_id)
 11          REFERENCES Module(module_id),
 12      CONSTRAINT fk_Student_Resource
 13          FOREIGN KEY (student_id)
 14          REFERENCES Student(student_id)
 15  );

Table created.

SQL>
```

Figure 25: creating module_resources table

Desc module_resources;

```
SQL> desc module_resources;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 RESOURCE_ID                               NOT NULL NUMBER(10)
 MODULE_ID                                 NOT NULL NUMBER(10)
 STUDENT_ID                                NOT NULL NUMBER(10)

SQL>
```

Figure 26: description module_resources table

6.1.13 Creating module_announcement table

CREATE TABLE Module_Announcement (

announcement_id NUMBER(10),

module_id NUMBER(10),

student_id NUMBER(10),

CONSTRAINT fk_announcement_an **FOREIGN KEY** (announcement_id) **REFERENCES** Announcement(announcement_id),

CONSTRAINT fk_module_mn **FOREIGN KEY** (module_id) **REFERENCES** Module(module_id),

CONSTRAINT fk_student_sn **FOREIGN KEY** (student_id) **REFERENCES** Student(student_id));

```
SQL> CREATE TABLE Module_Announcement (
  2      announcement_id NUMBER(10),
  3      module_id NUMBER(10),
  4      student_id NUMBER(10),
  5      CONSTRAINT fk_announcement_an FOREIGN KEY (announcement_id) REFERENCES Announcement(announcement_id),
  6      CONSTRAINT fk_module_mn FOREIGN KEY (module_id) REFERENCES Module(module_id),
  7      CONSTRAINT fk_student_sn FOREIGN KEY (student_id) REFERENCES Student(student_id)
  8  );

Table created.

SQL>
```

Figure 27: creating module_announcement table

Desc module_announcement;

```
SQL> desc module_announcement;
 Name                                                                                  Null?    Type
 ------------------------------------------------------------------------------------- -------- ------------------------------------------------------------
 ANNOUNCEMENT_ID                                                                               NUMBER(10)
 MODULE_ID                                                                                     NUMBER(10)
 STUDENT_ID                                                                                    NUMBER(10)

SQL>
```

Figure 28: description module_announcement table

6.2 Inserting data in tables

After we finally completed creating the tables of every entity, we need to insert values in them too. We have only created the table so every table are empty. In oracle, to insert values into tables we use INSERT INTO (table name) VALUES command. Every insertion is done by using this command. So, using this command, we have to insert values in every table. Let's start with inserting the values in program table.

6.2.1 Inserting data in Program Table

INSERT INTO Programm **VALUES** (1001, 'Computer Science', '3', '15-JAN-24', '14-JAN-27', 101, 25000);

INSERT INTO Programm **VALUES** (1002, 'Information Technology', '3', '01-FEB-24', '31-JAN-27', 102, 24000);

INSERT INTO Programm **VALUES** (1003, 'Business Administration', '3', '01-MAR-24', '28-FEB-27', 103, 23000);

INSERT INTO Programm **VALUES** (1004, 'Data Science', '2', '01-APR-24', '31-MAR-26', 104, 26000);

INSERT INTO Programm **VALUES** (1005, 'Cybersecurity', '2', '01-MAY-24', '30-APR-26', 105, 27000);

INSERT INTO Programm **VALUES** (1006, 'Civil Engineering', '4', '01-JUL-24', '30-JUN-28', 107, 29000);

INSERT INTO Programm **VALUES** (1007, 'Artificial Intelligence', '2', '01-AUG-24', '31-JUL-26', 108, 26500);

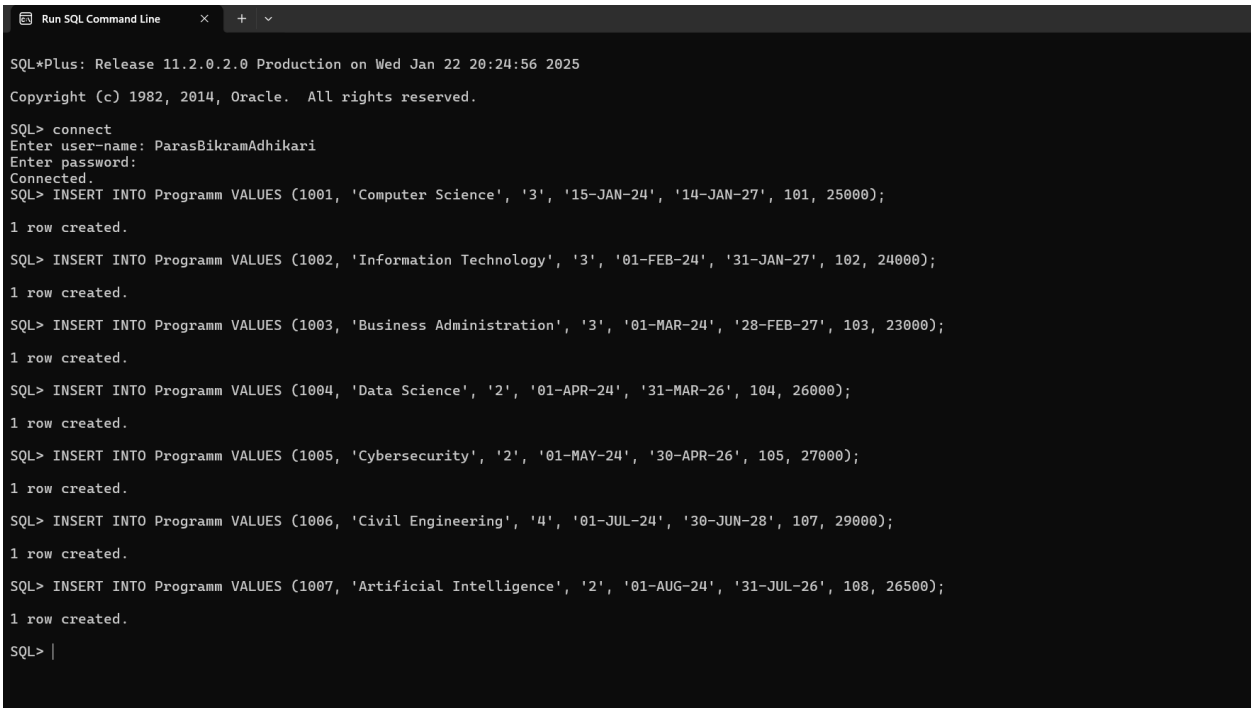

Figure 29: inserting values in program table

Select * from program;

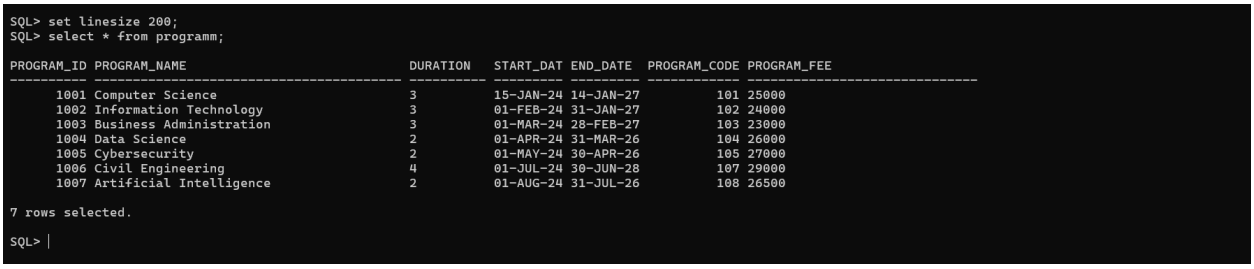

Figure 30: selection of program table

6.2.2 Inserting into teacher tabl

INSERT INTO Teacher **VALUES** (20001, 'Alice Johnson', 9812345678, 'alice.johnson@example.com');

INSERT INTO Teacher **VALUES** (20002, 'Robert Smith', 9823456789, 'robert.smith@example.com');

INSERT INTO Teacher **VALUES** (20003, 'Emily Brown', 9834567890, 'emily.brown@example.com');

INSERT INTO Teacher **VALUES** (20004, 'Michael Davis', 9845678901, 'michael.davis@example.com');

INSERT INTO Teacher **VALUES** (20005, 'Sarah Wilson', 9856789012, 'sarah.wilson@example.com');

INSERT INTO Teacher **VALUES** (20006, 'Chris Taylor', 9867890123, 'chris.taylor@example.com');

INSERT INTO Teacher **VALUES** (20007, 'David Clark', 9878901234, 'david.clark@example.com');

```
SQL> INSERT INTO Teacher VALUES (20001, 'Alice Johnson', 9812345678, 'alice.johnson@example.com');

1 row created.

SQL> INSERT INTO Teacher VALUES (20002, 'Robert Smith', 9823456789, 'robert.smith@example.com');

1 row created.

SQL> INSERT INTO Teacher VALUES (20003, 'Emily Brown', 9834567890, 'emily.brown@example.com');

1 row created.

SQL> INSERT INTO Teacher VALUES (20004, 'Michael Davis', 9845678901, 'michael.davis@example.com');

1 row created.

SQL> INSERT INTO Teacher VALUES (20005, 'Sarah Wilson', 9856789012, 'sarah.wilson@example.com');

1 row created.

SQL> INSERT INTO Teacher VALUES (20006, 'Chris Taylor', 9867890123, 'chris.taylor@example.com');

1 row created.

SQL> INSERT INTO Teacher VALUES (20007, 'David Clark', 9878901234, 'david.clark@example.com');

1 row created.

SQL>
```

Figure 31: inserting into teacher table

Select * from teacher

```
SQL> select * from teacher;

TEACHER_ID TEACHER_NAME                                       TEACHER_NO TEACHER_EMAIL
---------- -------------------------------------------------- ---------- ------------------------------
     20001 Alice Johnson                                      9812345678 alice.johnson@example.com
     20002 Robert Smith                                       9823456789 robert.smith@example.com
     20003 Emily Brown                                        9834567890 emily.brown@example.com
     20004 Michael Davis                                      9845678901 michael.davis@example.com
     20005 Sarah Wilson                                       9856789012 sarah.wilson@example.com
     20006 Chris Taylor                                       9867890123 chris.taylor@example.com
     20007 David Clark                                        9878901234 david.clark@example.com

7 rows selected.

SQL>
```

Figure 32: selection from teacher table

6.2.3 Inserting into Assessment table

INSERT INTO Assessment **VALUES** (40001, 'Programming Assignment 1', 20, '15-JAN-2024');

INSERT INTO Assessment **VALUES** (40002, 'Database Project', 30, '20-FEB-2024');

INSERT INTO Assessment **VALUES** (40003, 'Web Development Lab', 25, '15-MAR-2024');

INSERT INTO Assessment **VALUES** (40004, 'AI Midterm Exam', 15, '05-APR-2024');

INSERT INTO Assessment **VALUES** (40005, 'Cybersecurity Final Report', 40, '30-MAY-2024');

INSERT INTO Assessment **VALUES** (40006, 'Networking Quiz', 10, '10-JUN-2024');

INSERT INTO Assessment **VALUES** (40007, 'Data Science Capstone', 50, '01-JUL-2024');

```
SQL> INSERT INTO Assessment VALUES (40001, 'Programming Assignment 1', 20, '15-JAN-2024');
1 row created.
SQL> INSERT INTO Assessment VALUES (40002, 'Database Project', 30, '20-FEB-2024');
1 row created.
SQL> INSERT INTO Assessment VALUES (40003, 'Web Development Lab', 25, '15-MAR-2024');
1 row created.
SQL> INSERT INTO Assessment VALUES (40004, 'AI Midterm Exam', 15, '05-APR-2024');
1 row created.
SQL> INSERT INTO Assessment VALUES (40005, 'Cybersecurity Final Report', 40, '30-MAY-2024');
1 row created.
SQL> INSERT INTO Assessment VALUES (40006, 'Networking Quiz', 10, '10-JUN-2024');
1 row created.
SQL> INSERT INTO Assessment VALUES (40007, 'Data Science Capstone', 50, '01-JUL-2024');
1 row created.
SQL>
```

Figure 33: inserting values in assessment table

Select * from assessment;

```
SQL> set linesize 150;
SQL> select * from assessment;

ASSESSMENT_ID AS_TITLE                                           AS_WEIGHTAGE AS_DEADLI
------------- -------------------------------------------------- ------------ ---------
        40001 Programming Assignment 1                                   20 15-JAN-24
        40002 Database Project                                           30 20-FEB-24
        40003 Web Development Lab                                        25 15-MAR-24
        40004 AI Midterm Exam                                            15 05-APR-24
        40005 Cybersecurity Final Report                                 40 30-MAY-24
        40006 Networking Quiz                                            10 10-JUN-24
        40007 Data Science Capstone                                      50 01-JUL-24

7 rows selected.

SQL>
```

Figure 34: selection of assessment table

6.2.4 Inserting into resources table

INSERT INTO Resources **VALUES** (50001, 'Python Programming Basics', '3 Hours', 'Video');

INSERT INTO Resources **VALUES** (50002, 'Database Design Principles', '2 Hours', 'PDF');

INSERT INTO Resources **VALUES** (50003, 'Web Development Tutorial', '5 Hours', 'Video');

INSERT INTO Resources **VALUES** (50004, 'AI Fundamentals Guide', '4 Hours', 'eBook');

INSERT INTO Resources **VALUES** (50005, 'Cybersecurity Best Practices', '2.5 Hours', 'PDF');

INSERT INTO Resources **VALUES** (50006, 'Networking Labs', '3 Hours', 'Lab Manual');

INSERT INTO Resources **VALUES** (50007, 'Data Science Introduction', '6 Hours', 'Video');

Figure 35: inserting values to resources table

Select * from resources;

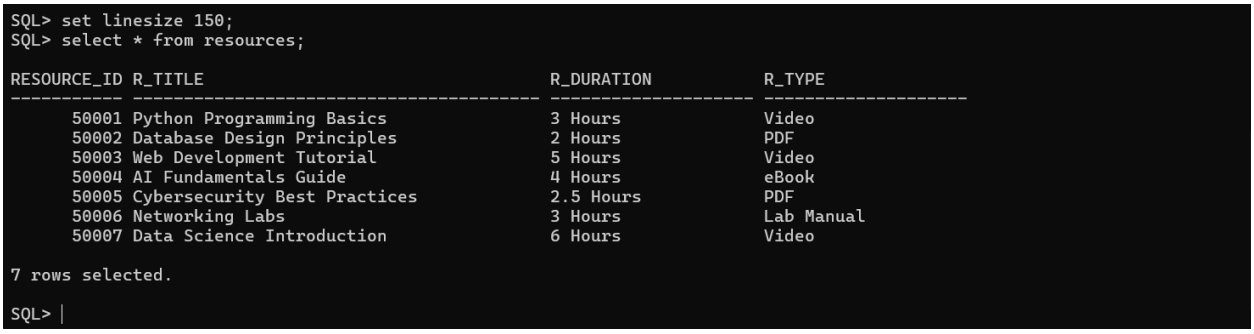

Figure 36: selection of resources table

6.2.5 Inserting into Student Table

INSERT INTO Student **VALUES** (10013, 'Mikey', 'Lee', 'mikey.lee@example.com', 9812345678,'15-JAN-2000', 'Kalanki', 24, 1001);

INSERT INTO Student **VALUES** (10025, 'Jimmee', 'Smith', 'jimmee.smith@example.com', 9823456789, '20-MAY-2001', 'Baneshwor', 23, 1002);

INSERT INTO Student **VALUES** (10037, 'Michael', 'Brown', 'michael.brown@example.com', 9834567890, '12-MAR-1999', 'Lalitpur', 25, 1003);

INSERT INTO Student **VALUES** (10045, 'Emily', 'Davis', 'emily.davis@example.com', 9845678901, '30-SEP-2000', 'Thamel', 24, 1004);

INSERT INTO Student **VALUES** (10059, 'Chris', 'Wilson', 'chris.wilson@example.com', 9856789012, '01-AUG-2001', 'Kritipur', 23, 1005);

INSERT INTO Student **VALUES** (10061, 'Sarah', 'Miller', 'sarah.miller@example.com', 9867890123, '10-NOV-1998', 'Bhaktapur', 26, 1002);

INSERT INTO Student **VALUES** (10072, 'David', 'Clark', 'david.clark@example.com', 9878901234, '05-JUL-2002', 'Naxal', 22, 1005);

```
SQL> INSERT INTO Student VALUES (10013, 'Mikey', 'Lee', 'mikey.lee@example.com', 9812345678,'15-JAN-2000', 'Kalanki', 24, 1001);
1 row created.
SQL> INSERT INTO Student VALUES (10025, 'Jimmee', 'Smith', 'jimmee.smith@example.com', 9823456789, '20-MAY-2001', 'Baneshwor', 23, 1002);
1 row created.
SQL> INSERT INTO Student VALUES (10037, 'Michael', 'Brown', 'michael.brown@example.com', 9834567890, '12-MAR-1999', 'Lalitpur', 25, 1003);
1 row created.
SQL> INSERT INTO Student VALUES (10045, 'Emily', 'Davis', 'emily.davis@example.com', 9845678901, '30-SEP-2000', 'Thamel', 24, 1004);
1 row created.
SQL> INSERT INTO Student VALUES (10059, 'Chris', 'Wilson', 'chris.wilson@example.com', 9856789012, '01-AUG-2001', 'Kritipur', 23, 1005);
1 row created.
SQL> INSERT INTO Student VALUES (10061, 'Sarah', 'Miller', 'sarah.miller@example.com', 9867890123, '10-NOV-1998', 'Bhaktapur', 26, 1002);
1 row created.
SQL> INSERT INTO Student VALUES (10072, 'David', 'Clark', 'david.clark@example.com', 9878901234, '05-JUL-2002', 'Naxal', 22, 1005);
1 row created.
SQL>
```

Figure 37: inserting into student table

Select * from student;

```
SQL> set linesize 200;
SQL> select * from student;

STUDENT_ID FIRST_NAME                                LAST_NAME                                EMAIL                          PHONE_NO DOB       ADDRESS                          STUDENT_AGE PROGRAM_ID
---------- ----------------------------------------- ---------------------------------------- ------------------------------ -------- --------- -------------------------------- ----------- ----------
     10013 Mikey                                     Lee                                      mikey.lee@example.com          9812345678 15-JAN-00 Kalanki                                  24       1001
     10025 Jimmee                                    Smith                                    jimmee.smith@example.com       9823456789 20-MAY-01 Baneshwor                                23       1002
     10037 Michael                                   Brown                                    michael.brown@example.com      9834567890 12-MAR-99 Lalitpur                                 25       1003
     10045 Emily                                     Davis                                    emily.davis@example.com        9845678901 30-SEP-00 Thamel                                   24       1004
     10059 Chris                                     Wilson                                   chris.wilson@example.com       9856789012 01-AUG-01 Kritipur                                 23       1005
     10061 Sarah                                     Miller                                   sarah.miller@example.com       9867890123 10-NOV-98 Bhaktapur                                26       1002
     10072 David                                     Clark                                    david.clark@example.com        9878901234 05-JUL-02 Naxal                                    22       1005

7 rows selected.

SQL>
```

Figure 38: selection student table

6.2.6 Inserting into module table

INSERT INTO Module **VALUES** (30001, 'Introduction to Programming', 3, '6 Months', 1, 20001);

INSERT INTO Module **VALUES** (30002, 'Database Management Systems', 4, '6 Months', 2, 20002);

INSERT INTO Module **VALUES** (30003, 'Web Development', 4, '6 Months', 2, 20003);

INSERT INTO Module **VALUES** (30004, 'Artificial Intelligence', 5, '1 Year', 3, 20004);

INSERT INTO Module **VALUES** (30005, 'Cybersecurity', 4, '1 Year', 3, 20005);

INSERT INTO Module **VALUES** (30006, 'Data Science Fundamentals', 5, '1 Year', 3, 20006);

INSERT INTO Module **VALUES** (30007, 'Advanced Networking', 4, '6 Months', 2, 20007);

```
SQL> INSERT INTO Module VALUES (30001, 'Introduction to Programming', 3, '6 Months', 1, 20001);
1 row created.
SQL> INSERT INTO Module VALUES (30002, 'Database Management Systems', 4, '6 Months', 2, 20002);
1 row created.
SQL> INSERT INTO Module VALUES (30003, 'Web Development', 4, '6 Months', 2, 20003);
1 row created.
SQL> INSERT INTO Module VALUES (30004, 'Artificial Intelligence', 5, '1 Year', 3, 20004);
1 row created.
SQL> INSERT INTO Module VALUES (30005, 'Cybersecurity', 4, '1 Year', 3, 20005);
1 row created.
SQL> INSERT INTO Module VALUES (30006, 'Data Science Fundamentals', 5, '1 Year', 3, 20006);
1 row created.
SQL> INSERT INTO Module VALUES (30007, 'Advanced Networking', 4, '6 Months', 2, 20007);
1 row created.
SQL>
```

Figure 39: inserting into module table

Select * from module;

```
SQL> select * from module;

 MODULE_ID MODULE_NAME                              CREDITS MODULE_DUR MODULE_LEVEL TEACHER_ID
---------- ---------------------------------------- ---------- ---------- ------------ ----------
     30001 Introduction to Programming                     3 6 Months             1      20001
     30002 Database Management Systems                     4 6 Months             2      20002
     30003 Web Development                                 4 6 Months             2      20003
     30004 Artificial Intelligence                         5 1 Year               3      20004
     30005 Cybersecurity                                   4 1 Year               3      20005
     30006 Data Science Fundamentals                       5 1 Year               3      20006
     30007 Advanced Networking                             4 6 Months             2      20007

7 rows selected.

SQL>
```

Figure 40: selection module table

6.2.7 Inserting into result table

INSERT INTO Result **VALUES** (20001, 85, 'A', 40001);

INSERT INTO Result **VALUES** (20002, 72, 'B+', 40002);

INSERT INTO Result **VALUES** (20003, 90, 'A+', 40003);

INSERT INTO Result **VALUES** (20004, 65, 'C', 40004);

INSERT INTO Result **VALUES** (20005, 78, 'B', 40005);

INSERT INTO Result **VALUES** (20006, 88, 'A', 40006);

INSERT INTO Result **VALUES** (20007, 95, 'A+', 40007);

```
SQL> INSERT INTO Result VALUES (20001, 85, 'A', 40001);
1 row created.
SQL> INSERT INTO Result VALUES (20002, 72, 'B+', 40002);
1 row created.
SQL> INSERT INTO Result VALUES (20003, 90, 'A+', 40003);
1 row created.
SQL> INSERT INTO Result VALUES (20004, 65, 'C', 40004);
1 row created.
SQL> INSERT INTO Result VALUES (20005, 78, 'B', 40005);
1 row created.
SQL> INSERT INTO Result VALUES (20006, 88, 'A', 40006);
1 row created.
SQL> INSERT INTO Result VALUES (20007, 95, 'A+', 40007);
1 row created.
SQL>
```

Figure 41: inserting into result table

Select * from result;

```
SQL> select * from result;

 RESULT_ID MARK_OBTAINED GRADE ASSESSMENT_ID
---------- ------------- ----- -------------
     20001            85 A             40001
     20002            72 B+            40002
     20003            90 A+            40003
     20004            65 C             40004
     20005            78 B             40005
     20006            88 A             40006
     20007            95 A+            40007

7 rows selected.

SQL>
```

Figure 42: selection result table

6.2.8 Inserting into Student_Module table

INSERT INTO Student_Module **VALUES** (10013, 30001);

INSERT INTO Student_Module **VALUES** (10025, 30002);

INSERT INTO Student_Module **VALUES** (10037, 30003);

INSERT INTO Student_Module **VALUES** (10045, 30004);

INSERT INTO Student_Module **VALUES** (10059, 30005);

INSERT INTO Student_Module **VALUES** (10061, 30006);

INSERT INTO Student_Module **VALUES** (10072, 30007);

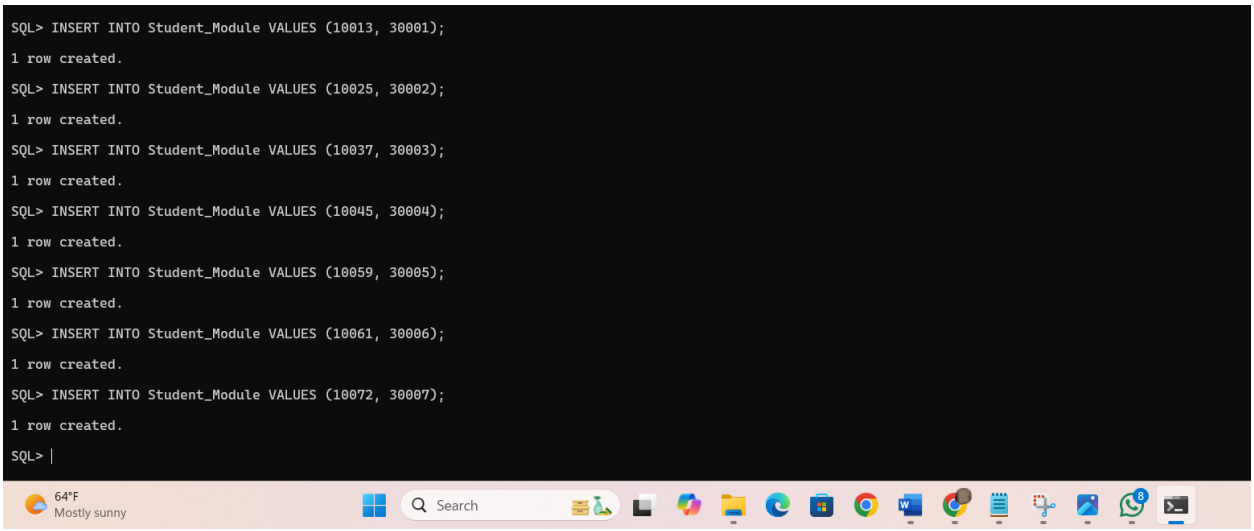

Figure 43: inserting into student_module table

Select * from student_module;

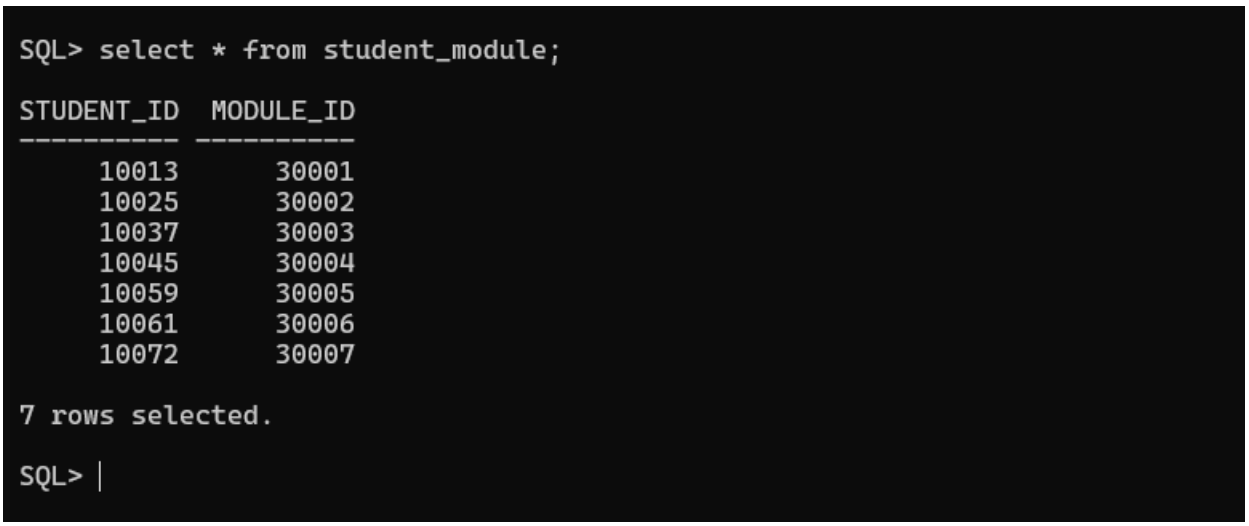

Figure 44: selection of student_module table

6.2.9 Inserting into Module_Assessment table

INSERT INTO Module_Assessment **VALUES** (40001, 30001, 10013);

INSERT INTO Module_Assessment **VALUES** (40002, 30002, 10025);

INSERT INTO Module_Assessment **VALUES** (40003, 30003, 10037);

INSERT INTO Module_Assessment **VALUES** (40004, 30004, 10045);

INSERT INTO Module_Assessment **VALUES** (40005, 30005, 10059);

INSERT INTO Module_Assessment **VALUES** (40006, 30006, 10061);

INSERT INTO Module_Assessment **VALUES** (40007, 30007, 10072);

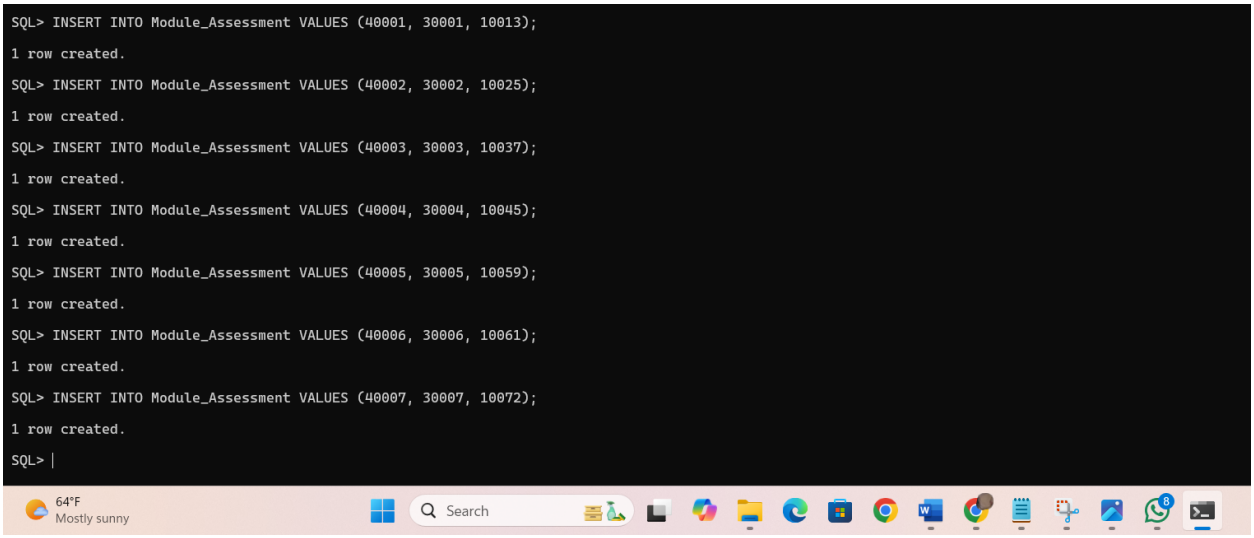

Figure 45: inserting into module_assessment table

Select * from module_assessment;

```
SQL> select * from module_assessment;

ASSESSMENT_ID  MODULE_ID STUDENT_ID
------------- ---------- ----------
        40001      30001      10013
        40002      30002      10025
        40003      30003      10037
        40004      30004      10045
        40005      30005      10059
        40006      30006      10061
        40007      30007      10072

7 rows selected.

SQL>
```

Figure 46: selection module_assessment table

6.2.10 Inserting into module_resource table

INSERT INTO Module_Resources **VALUES** (50001, 30001, 10013);

INSERT INTO Module_Resources **VALUES** (50002, 30002, 10025);

INSERT INTO Module_Resources **VALUES** (50003, 30003, 10037);

INSERT INTO Module_Resources **VALUES** (50004, 30004, 10045);

INSERT INTO Module_Resources **VALUES** (50005, 30005, 10059);

INSERT INTO Module_Resources **VALUES** (50006, 30006, 10061);

INSERT INTO Module_Resources **VALUES** (50007, 30007, 10072);

```
SQL> INSERT INTO Module_Resources  VALUES (50001, 30001, 10013);

1 row created.

SQL> INSERT INTO Module_Resources  VALUES (50002, 30002, 10025);

1 row created.

SQL> INSERT INTO Module_Resources  VALUES (50003, 30003, 10037);

1 row created.

SQL> INSERT INTO Module_Resources  VALUES (50004, 30004, 10045);

1 row created.

SQL> INSERT INTO Module_Resources  VALUES (50005, 30005, 10059);

1 row created.

SQL> INSERT INTO Module_Resources  VALUES (50006, 30006, 10061);

1 row created.

SQL> INSERT INTO Module_Resources  VALUES (50007, 30007, 10072);

1 row created.

SQL>
```

Figure 47: inserting into module_resources table

Select * from module resources;

```
SQL> select * from module_resources;

RESOURCE_ID  MODULE_ID STUDENT_ID
----------- ---------- ----------
      50001      30001      10013
      50002      30002      10025
      50003      30003      10037
      50004      30004      10045
      50005      30005      10059
      50006      30006      10061
      50007      30007      10072

7 rows selected.

SQL>
```

Figure 48: selection module_resources table

6.2.11 Inserting into Announcement table

INSERT INTO Announcement **VALUES** (001, 'Welcome Announcement', 30001, TO_DATE('2024-01-01', 'YYYY-MM-DD'));

INSERT INTO Announcement **VALUES** (002, 'Midterm Schedule', 30002, TO_DATE('2024-04-01', 'YYYY-MM-DD'));

INSERT INTO Announcement **VALUES** (003, 'Project Submission Deadline', 30003, TO_DATE('2024-04-05', 'YYYY-MM-DD'));

INSERT INTO Announcement **VALUES** (004, 'Extra Class on Saturday', 30004, TO_DATE('2024-05-20', 'YYYY-MM-DD'));

INSERT INTO Announcement **VALUES** (005, 'Final Exam Schedule', 30005, TO_DATE('2024-05-29', 'YYYY-MM-DD'));

INSERT INTO Announcement **VALUES** (006, 'New Course Material Uploaded', 30006, TO_DATE('2024-01-25', 'YYYY-MM-DD'));

INSERT INTO Announcement **VALUES** (007, 'Holiday Announcement', 30007, TO_DATE('2024-12-25', 'YYYY-MM-DD'));

```
SQL> INSERT INTO Announcement VALUES (001, 'Welcome Announcement', 30001, TO_DATE('2024-01-01', 'YYYY-MM-DD'));

1 row created.

SQL>
SQL> INSERT INTO Announcement VALUES (002, 'Midterm Schedule', 30002, TO_DATE('2024-04-01', 'YYYY-MM-DD'));

1 row created.

SQL>
SQL> INSERT INTO Announcement VALUES (003, 'Project Submission Deadline', 30003, TO_DATE('2024-04-05', 'YYYY-MM-DD'));

1 row created.

SQL>
SQL> INSERT INTO Announcement VALUES (004, 'Extra Class on Saturday', 30004, TO_DATE('2024-05-20', 'YYYY-MM-DD'));

1 row created.

SQL>
SQL> INSERT INTO Announcement VALUES (005, 'Final Exam Schedule', 30005, TO_DATE('2024-05-29', 'YYYY-MM-DD'));

1 row created.

SQL>
SQL> INSERT INTO Announcement VALUES (006, 'New Course Material Uploaded', 30006, TO_DATE('2024-01-25', 'YYYY-MM-DD'));

1 row created.

SQL>
SQL> INSERT INTO Announcement VALUES (007, 'Holiday Announcement', 30007, TO_DATE('2024-12-25', 'YYYY-MM-DD'));

1 row created.

SQL>
```

Figure 49: inserting into announcement table

Select * from announcement;

```
SQL> select * from announcement;

ANNOUNCEMENT_ID AN_TITLE                                MODULE_ID AN_DATE
--------------- ---------------------------------------- ---------- ---------
              1 Welcome Announcement                          30001 01-JAN-24
              2 Midterm Schedule                              30002 01-APR-24
              3 Project Submission Deadline                   30003 05-APR-24
              4 Extra Class on Saturday                       30004 20-MAY-24
              5 Final Exam Schedule                           30005 29-MAY-24
              6 New Course Material Uploaded                  30006 25-JAN-24
              7 Holiday Announcement                          30007 25-DEC-24

7 rows selected.

SQL>
```

Figure 50: selection announcement table

6.2.12 Inserting into module_announcement table

INSERT INTO Module_Announcement **VALUES** (001, 30001, 10013);

INSERT INTO Module_Announcement **VALUES** (002, 30002, 10025);

INSERT INTO Module_Announcement **VALUES** (003, 30003, 10037);

INSERT INTO Module_Announcement **VALUES** (004, 30004, 10045);

INSERT INTO Module_Announcement **VALUES** (005, 30005, 10059);

INSERT INTO Module_Announcement **VALUES** (006, 30006, 10061);

INSERT INTO Module_Announcement **VALUES** (007, 30007, 10072);

```
SQL> INSERT INTO Module_Announcement VALUES (001, 30001, 10013);
1 row created.
SQL> INSERT INTO Module_Announcement VALUES (002, 30002, 10025);
1 row created.
SQL> INSERT INTO Module_Announcement VALUES (003, 30003, 10037);
1 row created.
SQL> INSERT INTO Module_Announcement VALUES (004, 30004, 10045);
1 row created.
SQL> INSERT INTO Module_Announcement VALUES (005, 30005, 10059);
1 row created.
SQL> INSERT INTO Module_Announcement VALUES (006, 30006, 10061);
1 row created.
SQL> INSERT INTO Module_Announcement VALUES (007, 30007, 10072);
1 row created.
SQL>
```

Figure 51: inserting into module_announcement table

Select * from module_announcement;

```
SQL> select * from module_announcement;

ANNOUNCEMENT_ID  MODULE_ID STUDENT_ID
--------------- ---------- ----------
              1      30001      10013
              2      30002      10025
              3      30003      10037
              4      30004      10045
              5      30005      10059
              6      30006      10061
              7      30007      10072

7 rows selected.

SQL>
```

Figure 52: selection module_announcement table

7. Database Queries

7.1 Information Query

Below are some information queries given in the question. Now we are going to perform these queries in sql.

7.1.1 1st information query

Purpose: List the program that are available in the college and total number of students enrolled in each of the programs.

Description: the above query combines data from the program and student table to retrieve the program that are available in the college and total number of students enrolled in each. The table will show the program_id, program name and the total student enrolled in each program.

```
SELECT p.Program_ID, p.Program_Name, COUNT(s.Student_ID) AS Total_Students
FROM Programm p
LEFT JOIN Student s ON p.Program_ID = s.Program_ID
GROUP BY p.Program_ID, p.Program_Name;
```

```
SQL> SELECT p.Program_ID, p.Program_Name, COUNT(s.Student_ID) AS Total_Students
  2  FROM Programm p
  3  LEFT JOIN Student s ON p.Program_ID = s.Program_ID
  4  GROUP BY p.Program_ID, p.Program_Name;

PROGRAM_ID PROGRAM_NAME                           TOTAL_STUDENTS
---------- ---------------------------------------- --------------
      1005 Cybersecurity                                         2
      1001 Computer Science                                      1
      1003 Business Administration                               1
      1004 Data Science                                          1
      1007 Artificial Intelligence                               0
      1002 Information Technology                                2
      1006 Civil Engineering                                     0

7 rows selected.

SQL>
```

Figure 53: query no 1

7.1.2 2nd information query

Purpose: List all the announcements made for a particular module starting from 1st May 2024 to 28th May 2024.

Description: the above query combines data from the module and announcement table to retrieve all the announcements made for a particular module starting from 1st May 2024 to 28th May 2024. The table will show the module_id and the announcement details.

```
SELECT a.announcement_id, a.an_title, a.an_date, a.module_id
FROM Announcement a
WHERE a.an_date BETWEEN TO_DATE('2024-05-01', 'YYYY-MM-DD')
                    AND TO_DATE('2024-05-28', 'YYYY-MM-DD')
ORDER BY a.an_date;
```

```
SQL> SELECT a.announcement_id, a.an_title, a.an_date, a.module_id
  2  FROM Announcement a
  3  WHERE a.an_date BETWEEN TO_DATE('2024-05-01', 'YYYY-MM-DD')
  4                      AND TO_DATE('2024-05-28', 'YYYY-MM-DD')
  5  ORDER BY a.an_date;

ANNOUNCEMENT_ID AN_TITLE                                AN_DATE    MODULE_ID
--------------- --------------------------------------- --------- ----------
              4 Extra Class on Saturday                 20-MAY-24      30004

SQL>
```

Figure 54: query no 2

7.1.3 3rd information query

Purpose: List the names of all modules that begin with the letter 'D', along with the total number of resources uploaded for those modules.

Description: the above query combines data from the module and resources table to retrieve the module name beginning with ‘D’ along with the total number of resources uploaded for those modules. The table will show the module name and the total resources uploaded in those modules.

```
SELECT m.module_name, COUNT(r.resource_id) AS total_resources
```

FROM Module m

JOIN Module_Resources mr **ON** m.module_id = mr.module_id

JOIN Resources r **ON** mr.resource_id = r.resource_id

WHERE m.module_name LIKE 'D%' -- Modules starting with 'D'

GROUP BY m.module_name

ORDER BY m.module_name;

```
SQL> SELECT m.module_name, COUNT(r.resource_id) AS total_resources
  2  FROM Module m
  3  JOIN Module_Resources mr ON m.module_id = mr.module_id
  4  JOIN Resources r ON mr.resource_id = r.resource_id
  5  WHERE m.module_name LIKE 'D%'  -- Modules starting with 'D'
  6  GROUP BY m.module_name
  7  ORDER BY m.module_name;

MODULE_NAME                          TOTAL_RESOURCES
------------------------------------ ---------------
Data Science Fundamentals                          1
Database Management Systems                        1

SQL>
```

Figure 55: query no 3

7.1.4 4th information query

Purpose: List the names of all students along with their enrolled program who have not submitted any assessments for a particular module

Description: the above query combines data from the student, program and assessment table to retrieve the names of all students along with their enrolled program who have not submitted any assessments for a particular module. The table will show the output accordingly if the students have not submitted their assessment, otherwise output will be no rows selected.

SELECT DISTINCT s.student_id,

s.first_name,

s.last_name,

p.program_name,

m.module_name

FROM Student s

JOIN Programm p **ON** s.program_id = p.program_id

JOIN Student_Module sm **ON** s.student_id = sm.student_id

JOIN Module m **ON** sm.module_id = m.module_id

LEFT JOIN Module_Assessment ma **ON** m.module_id = ma.module_id

LEFT JOIN Result r **ON** ma.assessment_id = r.assessment_id **AND** r.student_id = s.student_id

WHERE r.result_id **IS NULL OR** ma.assessment_id **IS NULL**

ORDER BY s.last_name, s.first_name, p.program_name, m.module_name;

```
SQL> SELECT s.first_name, s.last_name, p.program_name
  2  FROM Student s
  3  JOIN Programm p ON s.program_id = p.program_id
  4  LEFT JOIN Module_Assessment ma ON ma.student_id = s.student_id
  5  LEFT JOIN Assessment a ON a.assessment_id = ma.assessment_id
  6  LEFT JOIN Module m ON m.module_id = ma.module_id
  7  WHERE ma.student_id IS NULL
  8  AND m.module_id = 30001
  9  ORDER BY s.last_name, s.first_name;

no rows selected

SQL>
```

Figure 56: query no 4

7.1.5 5th information query

Purpose: List all the teachers who teach more than one module.

Description: the above query combines data from the module and teacher table to retrieve the data of all the teachers who teach more than one module. The table will show the data accordingly if there are any teacher who teach multiple modules at once. If there's no teacher then the output should be no rows selected.

SELECT t.teacher_name, **COUNT**(m.module_id) AS num_modules

FROM Teacher t

JOIN Module m **ON** t.teacher_id = m.teacher_id

GROUP BY t.teacher_name

HAVING COUNT(m.module_id) > 1

ORDER BY t.teacher_name;

```
SQL> SELECT t.teacher_name, COUNT(m.module_id) AS num_modules
  2  FROM Teacher t
  3  JOIN Module m ON t.teacher_id = m.teacher_id
  4  GROUP BY t.teacher_name
  5  HAVING COUNT(m.module_id) > 1
  6  ORDER BY t.teacher_name;

no rows selected

SQL>
```

Figure 57: query no 5

Since I have assumed that one teacher can teach only one module so, there are no teachers that teaches more than one module at a time. Therefore, the output says no rows selected.

7.2 Transaction Query

Below are some Transaction queries given in the question. Now we are going to perform these queries in sql.

7.2.1 1st transaction query

Purpose: Identify the module that has the latest assessment deadline

Description: the above query combines data from the module and assessment table to retrieve the data of all the module that has the latest assessment deadline. The table will show the module name and the assessment title and its deadline.

SELECT m.module_name, a.as_title, a.as_deadline

FROM Module m

JOIN Module_Assessment ma **ON** m.module_id = ma.module_id

JOIN Assessment a **ON** a.assessment_id = ma.assessment_id

WHERE a.as_deadline = (**SELECT** MAX(as_deadline) FROM Assessment)

AND ROWNUM = 1;

```
SQL> SELECT m.module_name, a.as_title, a.as_deadline
  2  FROM Module m
  3  JOIN Module_Assessment ma ON m.module_id = ma.module_id
  4  JOIN Assessment a ON a.assessment_id = ma.assessment_id
  5  WHERE a.as_deadline = (SELECT MAX(as_deadline) FROM Assessment)
  6  AND ROWNUM = 1;

MODULE_NAME                          AS_TITLE                                           AS_DEADLI
------------------------------------ -------------------------------------------------- ---------
Advanced Networking                  Data Science Capstone                              01-JUL-24

SQL>
```

Figure 58: transaction query no 1

7.2.2 2nd transaction query

Purpose: Find the top three students who have the highest total score across all modules

Description: the above query combines data from the student, module, and result table to retrieve the data of top three students who have the highest total score across all modules. The table will show the student details like their name and id, module name, and the total marks they scored.

```
SELECT *
FROM (
    SELECT s.student_id,
        s.first_name,
        s.last_name,
        m.module_name,
        SUM(r.mark_obtained) AS total_score
    FROM Result r
    JOIN Module_Assessment ma ON r.assessment_id = ma.assessment_id
    JOIN Student_Module sm ON sm.student_id = ma.student_id
    JOIN Module m ON sm.module_id = m.module_id
    JOIN Student s ON sm.student_id = s.student_id
```

GROUP BY s.student_id, s.first_name, s.last_name, m.module_name

ORDER BY total_score **DESC**

)

WHERE ROWNUM <= 3;

```
SQL> SELECT *
  2  FROM (
  3      SELECT s.student_id,
  4             s.first_name,
  5             s.last_name,
  6             m.module_name,
  7             SUM(r.mark_obtained) AS total_score
  8      FROM Result r
  9      JOIN Module_Assessment ma ON r.assessment_id = ma.assessment_id
 10      JOIN Student_Module sm ON sm.student_id = ma.student_id
 11      JOIN Module m ON sm.module_id = m.module_id
 12      JOIN Student s ON sm.student_id = s.student_id
 13      GROUP BY s.student_id, s.first_name, s.last_name, m.module_name
 14      ORDER BY total_score DESC
 15  )
 16  WHERE ROWNUM <= 3;

STUDENT_ID FIRST_NAME                               LAST_NAME                               MODULE_NAME                               TOTAL_SCORE
---------- ---------------------------------------- --------------------------------------- ---------------------------------------- -----------
     10072 David                                    Clark                                   Advanced Networking                                95
     10037 Michael                                  Brown                                   Web Development                                    90
     10061 Sarah                                    Miller                                  Data Science Fundamentals                          88

SQL>
```

Figure 59: transaction query no 2

7.2.3 3rd transaction query

Purpose: Find the total number of assessments for each program and the average score across all assessments in those programs

Description: the above query combines data from the program, assessment, and result table to retrieve the data of the total number of assessments for each program and the average score across all assessments in those programs. The table will show the program details like its name and id, total assessment, and the average score.

SELECT p.program_id, p.program_name,

COUNT(a.assessment_id) **AS** total_assessments,

AVG(r.mark_obtained) **AS** average_score

FROM Programm p

JOIN Student s **ON** p.program_id = s.program_id

JOIN Student_Module sm **ON** s.student_id = sm.student_id

JOIN Module m **ON** sm.module_id = m.module_id

JOIN Module_Assessment ma **ON** m.module_id = ma.module_id

JOIN Assessment a **ON** ma.assessment_id = a.assessment_id

JOIN Result r **ON** a.assessment_id = r.assessment_id

GROUP BY p.program_id, p.program_name;

```
SQL> SELECT p.program_id, p.program_name,
  2         COUNT(a.assessment_id) AS total_assessments,
  3         AVG(r.mark_obtained) AS average_score
  4  FROM Programm p
  5  JOIN Student s ON p.program_id = s.program_id
  6  JOIN Student_Module sm ON s.student_id = sm.student_id
  7  JOIN Module m ON sm.module_id = m.module_id
  8  JOIN Module_Assessment ma ON m.module_id = ma.module_id
  9  JOIN Assessment a ON ma.assessment_id = a.assessment_id
 10  JOIN Result r ON a.assessment_id = r.assessment_id
 11  GROUP BY p.program_id, p.program_name;

PROGRAM_ID PROGRAM_NAME                           TOTAL_ASSESSMENTS AVERAGE_SCORE
---------- ---------------------------------------- ----------------- -------------
      1005 Cybersecurity                                           2          86.5
      1001 Computer Science                                        1            85
      1003 Business Administration                                 1            90
      1004 Data Science                                            1            65
      1002 Information Technology                                  2            80

SQL>
```

Figure 60: transaction query no 3

7.2.4 4th transaction query

Purpose: List the students who have scored above the average score in the ‘Databases’ module

Description: the above query retrieves List the students who have scored above the average score in the ‘Databases’ module. The table will show the details of the students who scored above the average score in Database module.

SELECT s.first_name, s.last_name, s.student_id

FROM Student s

JOIN Student_Module sm **ON** s.student_id = sm.student_id

```
JOIN Module m ON sm.module_id = m.module_id
JOIN Module_Assessment ma ON m.module_id = ma.module_id
JOIN Assessment a ON ma.assessment_id = a.assessment_id
JOIN Result r ON a.assessment_id = r.assessment_id
WHERE m.module_name = 'Database Management Systems'
AND r.mark_obtained >= (
    SELECT AVG(r2.mark_obtained)
    FROM Result r2
    JOIN Assessment a2 ON r2.assessment_id = a2.assessment_id
    JOIN Module_Assessment ma2 ON a2.assessment_id = ma2.assessment_id
    JOIN Module m2 ON ma2.module_id = m2.module_id
    WHERE m2.module_name = 'Database Management Systems'
);
```

```
SQL> SELECT s.first_name, s.last_name, s.student_id
  2  FROM Student s
  3  JOIN Student_Module sm ON s.student_id = sm.student_id
  4  JOIN Module m ON sm.module_id = m.module_id
  5  JOIN Module_Assessment ma ON m.module_id = ma.module_id
  6  JOIN Assessment a ON ma.assessment_id = a.assessment_id
  7  JOIN Result r ON a.assessment_id = r.assessment_id
  8  WHERE m.module_name = 'Database Management Systems'
  9  AND r.mark_obtained >= (
 10      SELECT AVG(r2.mark_obtained)
 11      FROM Result r2
 12      JOIN Assessment a2 ON r2.assessment_id = a2.assessment_id
 13      JOIN Module_Assessment ma2 ON a2.assessment_id = ma2.assessment_id
 14      JOIN Module m2 ON ma2.module_id = m2.module_id
 15      WHERE m2.module_name = 'Database Management Systems'
 16  );

FIRST_NAME                           LAST_NAME                            STUDENT_ID
------------------------------------ ------------------------------------ ----------
Jimmee                               Smith                                     10025

SQL>
```

Figure 61: transaction query no 4

7.2.5 5th transaction query

Purpose: Display whether a student has passed or failed as remarks as per their total aggregate marks obtained in a particular module

Description: the above query combines data from the student, module, assessment, and result table to retrieve the data of a student that has passed or failed as remarks as per their total aggregate marks obtained in a particular module. The table will show the student details like their name, the module name, total marks, total weightage, and the remarks.

```
SELECT
   s.first_name,
   s.last_name,
   m.module_name,
   SUM(r.mark_obtained) AS total_marks,
   SUM(a.as_weightage) AS total_weightage,
   CASE
      WHEN (SUM(r.mark_obtained) / SUM(a.as_weightage)) * 100 >= 50 THEN 'Pass'
      ELSE 'Fail'
   END AS remarks
FROM Student s
JOIN Student_Module sm ON s.student_id = sm.student_id
JOIN Module m ON sm.module_id = m.module_id
JOIN Module_Assessment ma ON m.module_id = ma.module_id
JOIN Assessment a ON ma.assessment_id = a.assessment_id
JOIN Result r ON a.assessment_id = r.assessment_id
GROUP BY s.first_name, s.last_name, m.module_name;
```

```
SQL> SELECT
  2      s.first_name,
  3      s.last_name,
  4      m.module_name,
  5      SUM(r.mark_obtained) AS total_marks,
  6      SUM(a.as_weightage) AS total_weightage,
  7      CASE
  8          WHEN (SUM(r.mark_obtained) / SUM(a.as_weightage)) * 100 >= 50 THEN 'Pass'
  9          ELSE 'Fail'
 10      END AS remarks
 11  FROM Student s
 12  JOIN Student_Module sm ON s.student_id = sm.student_id
 13  JOIN Module m ON sm.module_id = m.module_id
 14  JOIN Module_Assessment ma ON m.module_id = ma.module_id
 15  JOIN Assessment a ON ma.assessment_id = a.assessment_id
 16  JOIN Result r ON a.assessment_id = r.assessment_id
 17  GROUP BY s.first_name, s.last_name, m.module_name;

FIRST_NAME                               LAST_NAME                               MODULE_NAME                               TOTAL_MARKS TOTAL_WEIGHTAGE REMARKS
---------------------------------------- --------------------------------------- ---------------------------------------- ----------- ---------------- ----------
Jimmee                                   Smith                                   Database Management Systems                    72               30 Pass
Chris                                    Wilson                                  Cybersecurity                                  78               40 Pass
Sarah                                    Miller                                  Data Science Fundamentals                      88               10 Pass
Michael                                  Brown                                   Web Development                                90               25 Pass
David                                    Clark                                   Advanced Networking                            95               50 Pass
Mikey                                    Lee                                     Introduction to Programming                    85               20 Pass
Emily                                    Davis                                   Artificial Intelligence                        65               15 Pass

7 rows selected.
```

Figure 62: transaction query no 5

8. Critical evaluation

8.1 Critical Evaluation of Module

The Database Module has substantially covered all important areas related to database design and implementation by concentrating on Oracle as the principal tool for learning. From data modeling to design processes and querying techniques, the module covered all that would give a student an understanding of how data is structured and linked together. These concepts were not only developed for theoretical understanding but also in practical terms to face the challenges of a database at work.

It was complemented with lectures and tutorials that let us get our hands dirty weekly in the development, designing, querying, and optimization of any database both in work and home. The gap between theory and application was bridged by this practical approach; hence, it made complex topics such as normalization or advanced SQL very approachable.

Though the module was somewhat challenging with regard to understanding intricate queries, and abstract concepts, it did contribute to nurturing the performance of analytical thought and problem-solving.

The module linked the knowledge of databases with real-world scenarios, hence generally giving it relevance to most disciplines and instilling the zeal for robust database systems. All in all, it was a very valuable learning experience that has laid a strong foundation for academic and professional endeavors in the future.

8.2 Critical Assessment of Coursework

The project work was to design and implement a database system for the "E-Classroom Platform," with most of the part involving SQL in Oracle 11g. The main objective set was to be able to secure a system supporting good management of academic life aspects, among others: students, teachers, programs, and modules with the assurance of organizational effectiveness and high data accessibility. There were tasks like table creation and normalization, establishing relationships between entities, and running queries to unearth some meaningful insights from your database. A robust schema design that supports major operations such as querying student progress, resource management, and assessment of performance was an important part of the coursework.

Each of the steps was to be accomplished by understanding the principles on databases, data modeling, and SQL commands for a very good learning experience regarding database development.

Although, this did not prove to be a very smooth process. The major hiccup was in understanding and applying normalization to the database so that it becomes efficient and free from redundancy. This had created confusion in terms of breaking down these complex relationships into simpler forms. With so many errors in executing queries related to creating tables and inserting data, it summed up to add on the complexity of development due to reasons like mismatched data types and syntax errors causing setbacks. The process was further complicated by numerous errors in running queries and inserting data in various tables, which were mainly due to discrepancies in data types and syntax errors.

So, implementation challenges were required to be troubleshooted really carefully in order for the relationships between tables to be correct. All these challenges were, nevertheless, dealt with by thorough analysis, consultations, and iterative improvements. Successful completion of the project eventually overcame these issues and provided invaluable insight into conquering real-world database design problems.

9. Dump File creation and Drop Queries

9.1 Dump file creation

exp ParasBikramAdhikari/23048587 file = ParasAdhikari.dmp

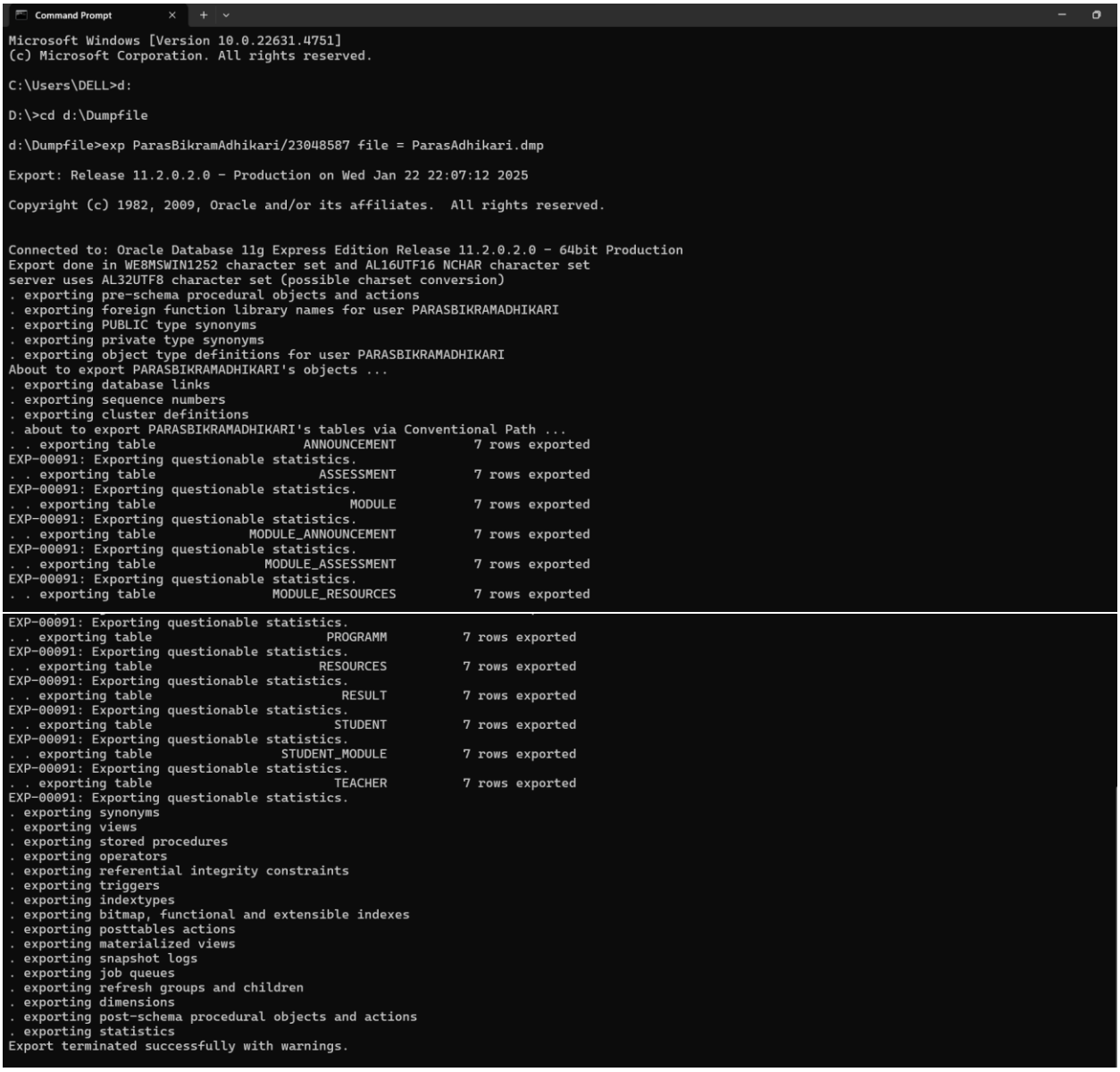

Figure 63: creating dump file

9.2 Drop Queries

DROP TABLE module_announcement;

DROP TABLE module_resources;

DROP TABLE module_assessment;

DROP TABLE module_announcement;

DROP TABLE student_module;

DROP TABLE resources;

DROP TABLE announcement;

DROP TABLE result;

DROP TABLE assessment;

DROP TABLE module;

DROP TABLE student;

DROP TABLE programm;

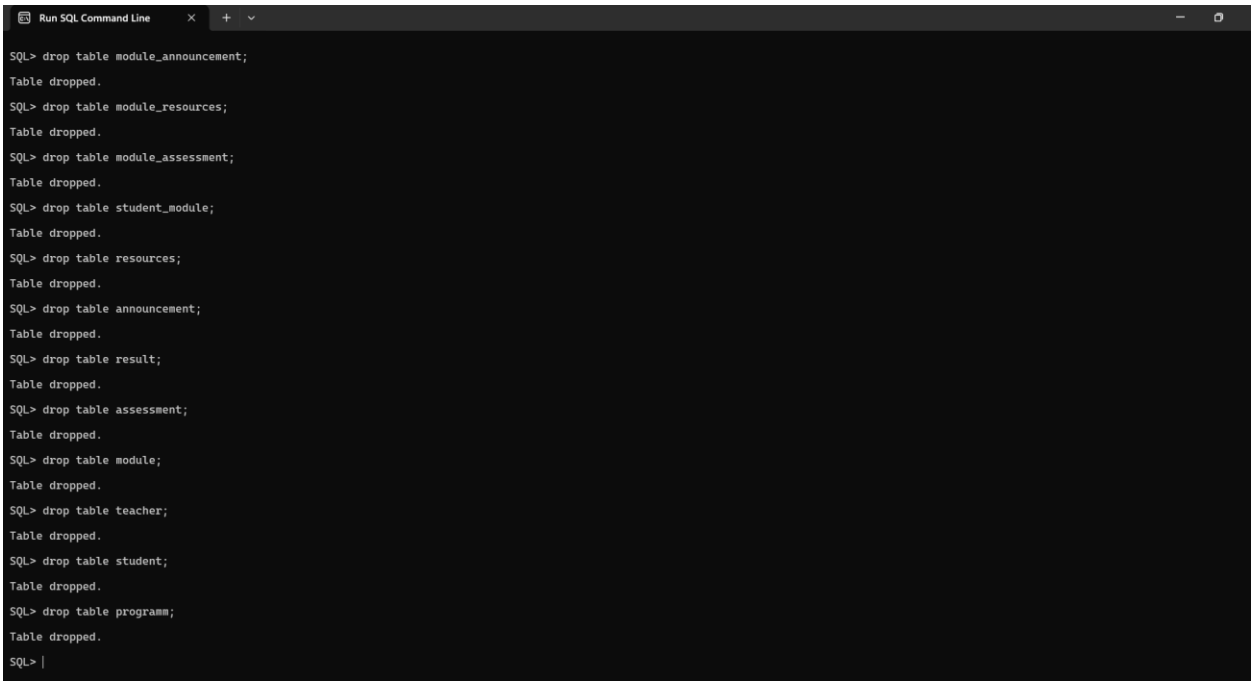

Figure 64: table drop queries

References

Atlan, T., 2024. *Atlan.* [Online]
Available at: https://atlan.com/what-is-metadata-to-test-1/
[Accessed 29 12 2024].

geeksforgeeks, 2020. *types of normal forms in dbms.* [Online]
Available at: https://www.geeksforgeeks.org/types-of-normal-forms-in-dbm
[Accessed 25 01 2025].

geeksforgeeks, 2024. *geeksforgeeks.* [Online]
Available at: https://www.geeksforgeeks.org/normal-forms-in-dbms/
[Accessed 21 december 2024].

geeksforgeeks, 2025. *first normal form.* [Online]
Available at: https://www.geeksforgeeks.org/first-normal-form-1nf/
[Accessed 09 01 2025].

LIBRARY, U. M., 2024. *UC MERCED LIBRARY.* [Online]
Available at: https://library.ucmerced.edu/data-dictionaries
[Accessed 29 12 2024].

Ouko, A., 2024. *datacamp.* [Online]
Available at: https://www.datacamp.com/tutorial/transitive-dependency
[Accessed 28 12 2024].