LONDON METROPOLITAN UNIVERSITY

islington college
(इस्लिङ्टन कलेज)

CC5051NI Databases

100% Individual Coursework

Autumn 2024

Credit: 15 Semester Long Module

Student Name: Bhumika Karki

London Met ID: 23047584

Assignment Submission Date: 23rd January 2025

Word Count: 7801

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.

Bhumika Karki.docx

Islington College,Nepal

Document Details

Submission ID

trn:oid:::3618:79905937

Submission Date

Jan 23, 2025, 8:04 AM GMT+5:45

Download Date

Jan 23, 2025, 8:05 AM GMT+5:45

File Name

Bhumika Karki.docx

File Size

35.9 KB

64 Pages

4,713 Words

29,174 Characters

19% Overall Similarity

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

Match Groups

- **74** Not Cited or Quoted 19%
  Matches with neither in-text citation nor quotation marks
- **2** Missing Quotations 0%
  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

0% Internet sources
0% Publications
19% 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 Figures

List of Tables

1. Introduction

1.1. Introduction to the University

The founder of the Mid-west University, Ms. Mary proposes the E-Classroom platform to make it easier and more efficient for the university to manage their students, instructors, programs, and modules. It offers a centralized digital environment for academic operations, ensuring efficient resource distribution and stakeholder communication. Students can use the platform to access their academic information, including announcements, assessments, and modules, while teachers can manage their respective modules and distribute resources to students. Administrators can also oversee programs, assign teachers to modules, and use results to track the progress of students.

This system aims to establish a smooth and organized learning environment by reducing manual procedures and ensuring efficient data management. The teaching and learning process is improvised by features like resource sharing, assessment tracking, and automatic result generation. Additionally, announcements for modules also provide students with important academic information.

Figure 1: Mid-west University

1.2. Aims and Objectives

The ultimate aim of the organization is to develop and implement a digital platform, the E-Classroom Platform, which enables effective management of students, programs and modules, providing a structured and dynamic study environment. The following objectives will be performed in accordance with the question's requirements as follows:

- Create and implement a functional database capable of managing the details of Students, their associated Programs, and the Modules included in each program.
- Identify and track the relationships between Programs, Modules, and Students, ensuring seamless data integration for academic and administrative purposes.
- Maintain a comprehensive record of all modules, including attributes such as module credits and the programs they belong to.
- Provide a structure that enables the management of module assessments and resources in alignment with students' academic progress.
- Design a system that supports querying and analysing data for informational and transactional purposes, such as retrieving student enrolment details and module associations.
- Develop a framework that ensures scalability and supports many-to-many relationships, such as modules shared across programs.

1.3. Current Business Activities

The E-Classroom platform follows a set of core activities to control its functionality and ensure seamless operations. These activities include:

- Students are enrolled in a certain program, and each program consists of several modules.
- Students are assigned to a single program, and their data is maintained in the database.
- Each program offers multiple modules that are specific to the program's curriculum.

- Module performance is evaluated, and the results are calculated and stored in the database.
- To guarantee accuracy and accessibility; program, student, and module information is updated on a regular basis.

1.4. Business Rules

To ensure the efficient management and operation of the E-Classroom platform for Mid-west University, it is critical to establish and adhere to specific business rules. These rules outline how the platform will function and are listed below:

- Each Program is uniquely identified by a ProgramID and includes attributes such as program name and description. A Program consists of multiple modules, and each module can belong to many programs.
- Each Student is uniquely identified by a StudentID and is associated with exactly one Program, whereas a program contains multiple students. Students have details such as name, email, phone, and address.
- Each Module is uniquely identified by a ModuleID and includes attributes like module name, credits, and TeacherID (primary teacher). Modules can have associated Resources, Assessments, and Announcements.
- Each Teacher is uniquely identified by a TeacherID and is assigned to one or more Modules, each module is assigned to one teacher. Teachers have attributes such as name and department.
- Each Resource is uniquely identified by a ResourceID and is tied to a specific Module. ResourceCompletion tracks if a student has completed a resource before being granted to the next one. Resources include attributes like title, and type to facilitate structured learning.
- Each Assessment is uniquely identified by an AssessmentID and can be shared across multiple modules. Assessments include details such as title, deadline, weightage, TotalMarks. This relationship is managed through the ModuleAssessment bridge table.

- Each Announcement is uniquely identified by an AnnouncementID and is associated with a specific Module and posted by a specific teacher. Announcements contain details like content and posting date.
- Each Result is uniquely identified by a ResultID and is associated with a Student and an Assessment. Results track attributes like ObtainedMarks, and grade where grade is derived based on the percentage of ObtainedMarks out of TotalMarks.
- A Module can have multiple Resources, and Announcements, but each of these entities belongs to only one Module. A module can have multiple Assessments, and assessments can also belong to modules across different programs.

Assumptions:

- A student is enrolled in one program only. Students have details such as name, email, phone, DOB, and address.
- Each program consists of multiple modules, and a module can belong to multiple programs (many-to-many relationship). Modules are mandatory for the students who are enrolled in the programs that they are linked to.
- Resources are linked to a module, and it must be completed in a specific order. Completion status and date of the resources is tracked by using a ResourceCompletion table. Resources include attributes like title, type to facilitate structured learning.
- Each assessment is linked to a single module. However, modules can share assessments across different programs (many-to-many relationship via ModuleAssessment). Assessments include details such as title, deadline, and weightage.
- The result of assessments is tracked per student and is uniquely identified by ResultID.
- Each Teacher is assigned to one or more Modules, and each module is assigned to one primary teacher. Teacher is uniquely identified by a TeacherID and have attributes such as name and department.
- Announcements are linked to a specific module and posted by the assigned teacher. Announcements are uniquely identified by AnnouncementID.

2. Initial ERD

An entity relationship diagram (ERD) is a visual representation of a relational database. ERDs can be used to visualize and understand an existing database or to develop a new one.

Understanding what an entity is and what an entity set is, is the first step in identifying ERDs. In this context, an entity is an object or a piece of data. A group of related entities is called an entity set. The relationships between entity sets kept in a database are shown in an ERD. The characteristics of these entities may be defined by their attributes. An ERD demonstrates the logical structure of databases by defining the entities, their properties, and the relationships between them. In relational databases, entities are equivalent to tables (rows). The characteristics of that entity (columns) that you wish to record are called attributes. Lastly, relationships describe the interactions between the entities (Secoda, 2024).

2.1. List of Created Objects

As a result of the preceding steps, significant measures were taken to define the framework for the database of the E-Classroom platform. The created objects include all the major entities and their associated attributes, which are listed below:

Three distinct objects are chosen as major entities based on their intrinsic value. Each of them is interconnected to define the database design and fulfill the requirements of the E-Classroom platform. They are briefly explained as follows:

- **Program:** The program serves as the foundation for grouping related modules and students. Each program is uniquely identified by its ProgramID and includes additional details like the program name and description. Programs establish the academic structure for students.
- **Student:** Students are the primary users of the platform. Each student is uniquely identified by their StudentID and is associated with a single program. Additional attributes include the student's name, email, phone, and address.

- **Module:** Modules are essential academic units that are part of one or more programs. Each module has features like its name and credits and is uniquely identified by its ModuleID. Students, programs, assessments, and resources are all connected by modules, which provide an integrated academic framework.

2.2. Identification of Entities and Attributes

An entity can be a real-world object, either animate or inanimate, that can be easily identified. Such as students, instructors, classes, and courses offered can all be regarded as entities in a school database. Each of these entities has a set of characteristics that make them unique.

Entities are represented by means of their properties, called attributes. Every attribute has a value. A student entity, for instance, might have attributes like name, class, and age (tutorialspoint, 2024).

Student

Table 1: Defining the Data Types of the Student Table

S. No.	Attributes	Data Type	Size	Constraints
1.	StudentID	CHARACTER	25	PRIMARY KEY, NOT NULL, UNIQUE
2.	StudentName	CHARACTER	50	NOT NULL
3.	StudentEmail	CHARACTER	50	NOT NULL, UNIQUE
4.	StudentPhone	CHARACTER	15	NOT NULL, UNIQUE
5.	StudentDOB	DATE		NOT NULL
6.	StudentAddress	CHARACTER	100	NOT NULL

Program

Table 2: Defining the Data Types of the Program Table

S. No.	Attributes	Data Type	Size	Constraints
1.	ProgramID	CHARACTER	25	PRIMARY KEY, NOT NULL, UNIQUE
2.	ProgramName	CHARACTER	50	NOT NULL, UNIQUE
3.	ProgramDescription	CHARACTER	100	NOT NULL

Module

Table 3: Defining the Data Types of the Module Table

S. No.	Attributes	Data Type	Size	Constraints
1.	ModuleID	CHARACTER	25	PRIMARY KEY, NOT NULL, UNIQUE
2.	ModuleName	CHARACTER	50	NOT NULL, UNIQUE
3.	Credits	NUMBER	3	NOT NULL
4.	TeacherID	CHARACTER	25	NOT NULL, UNIQUE
5.	AssessmentID	CHARACTER	25	NOT NULL, UNIQUE
6.	ResourceID	CHARACTER	25	NOT NULL, UNIQUE
7.	AnnouncementID	CHARACTER	25	NOT NULL, UNIQUE
8.	ResultID	CHARACTER	25	NOT NULL, UNIQUE

2.3. Representation of Primary and Foreign Keys

Based on the definition and meaning associated with each of the keys, the following constraints were used to determine their base primary and foreign keys. Each of the primary keys assigned to each of the entities is thus presented below:

- **ProgramID:** As numerous program details may be similar, a unique key in the form of an ID enables them to be properly identified.
- **ModuleID:** To keep track of each module offered under various programs, each module must be assigned a unique key for subsequent inquiry and inspection.
- **StudentID:** As numerous student details (e.g., name) may be similar, a unique key in the form of an ID enables them to be properly identified.
- **ResultID:** To keep track of each student's performance in different modules, a unique identifier is assigned to ensure proper tracking and analysis.
- **AssessmentID:** As assessments within a module may overlap in terms of deadlines and titles, a unique key ensures accurate tracking for grading and scheduling.
- **TeacherID:** A unique identifier used to distinguish each teacher, ensuring accurate module assignments and avoiding confusion even when teachers have similar names or departments.
- **ResourceID:** To keep track of all resources linked to modules, each resource must be assigned a unique identifier for easy access and management.
- **AnnouncementID:** As modules can have multiple announcements, a unique identifier is assigned to track announcements related to updates and information sharing.

The relationship between the given entities can be thus described along with the cardinality that they present as follows:

- A program includes multiple modules, and a module can be part of multiple programs. This relationship establishes a many-to-many relationship between programs and modules.

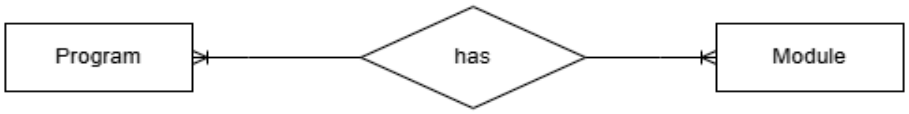

Figure 2: Initialized Relationship: Program – Module

- A program has multiple students enrolled, but a student is enrolled in only one program. This relationship establishes a one-to-many relationship between programs and students.

Figure 3: Initialized Relationship: Program – Student

2.4. Entity Relationship Diagram

This ERD represents the structure of an E-Classroom system for Mid-west University, which focuses on the relationships between students, programs, and modules. Each student can be enrolled in one program only. Programs are made up of several modules, and modules might be a part of more than one program. The system enables effective management of academic data by tracking important attributes for students, programs, and modules.

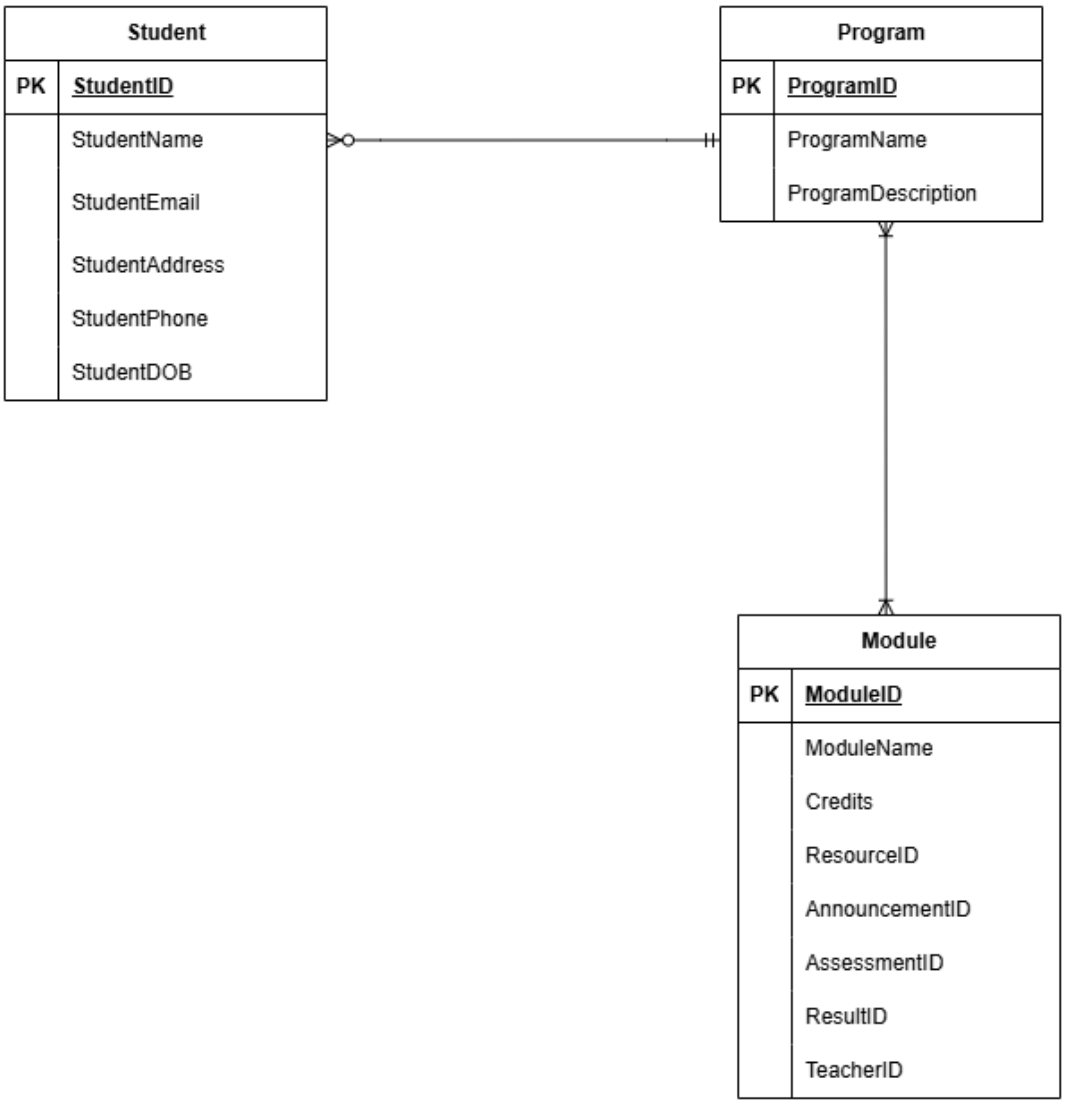

Figure 4: Initial ERD

3. Normalization

The process of organizing the provided data in a database by reducing the redundancy in a relation is known as normalization. Here, we remove the anomalies—update, insertion, and deletion. The single table is divided into smaller tables by normalization, and relationships are used to connect them. We can reduce the amount of redundancy in the database table by using the various normal forms (Rajput, 2024).

3.1. Un-Normalized Form (UNF)

The rules to be followed while the initial process of UNF is listed below:

- All the relevant entities and relationships are identified in a single table.
- Curly braces {} are used to denote the repetition of a group of letters.
- The main key for each of the entities is identified and presented.

Scenario of UNF:

Student (**StudentID**, StudentName, StudentEmail, StudentDOB, StudentAddress, StudentPhone, ProgramID, ProgramName, ProgramDescription, {ModuleID, ModuleName, Credits, TeacherID, TeacherName, Department, {AssessmentID, Title, Deadline, Weightage, TotalMarks, ResultID, ObtainedMarks, Grade}, {ResourceID, ResourceName, ResourceType, CompletionOrder, CompletionStatus, CompletionDate}, {AnnouncementID, AnnouncementDetails, PostedDate}})

3.2. First Normalized Form (1NF)

The following requirements must be followed in 1NF:

- It is crucial that all of the values in a column originate from the same domain; single-valued attributes and columns should only be used when absolutely necessary.
- The primary key of the base table is sent to the inner tables as composite keys; the repeating group should be separated from the main base table.

Before 1NF:

Student (**StudentID**, StudentName, StudentEmail, StudentDOB, StudentAddress, StudentPhone, ProgramID, ProgramName, ProgramDescription, {ModuleID, ModuleName, Credits, TeacherID, TeacherName, Department, {AssessmentID, Title, Deadline, Weightage, TotalMarks, ResultID, ObtainedMarks, Grade}, {ResourceID, ResourceName, ResourceType, CompletionOrder, CompletionStatus, CompletionDate}, {AnnouncementID, AnnouncementDetails, PostedDate}})

Scenario of 1NF:

Student (**StudentID**, StudentName, StudentEmail, StudentDOB, StudentAddress, StudentPhone, ProgramID, ProgramName, ProgramDescription)

Module (**ModuleID**, ModuleName, Credits, TeacherID, TeacherName, Department)

Assessment (**AssessmentID**, Title, Deadline, Weightage, TotalMarks, **ModuleID***)

Result (**ResultID**, ObtainedMarks, Grade, **StudentID***, **AssessmentID***)

Resource (**ResourceID**, ResourceName, ResourceType, CompletionOrder, **ModuleID***)

ResourceCompletion (**StudentID***, **ResourceID***, CompletionStatus, CompletionDate)

Announcement (**AnnouncementID**, AnnouncementDetails, PostedDate, **ModuleID***, **TeacherID***)

Functional Dependencies in 1NF:

StudentID ⟶ StudentName, StudentEmail, StudentDOB, StudentAddress, StudentPhone, ProgramID, ProgramName, ProgramDescription

ModuleID ⟶ ModuleName, Credits, TeacherID, TeacherName, Department

AssessmentID ⟶ Title, Deadline, Weightage, TotalMarks, ModuleID

ResultID ⟶ StudentID, AssessmentID, ObtainedMarks, Grade

ResourceID ⟶ ResourceName, ResourceType, CompletionOrder, ModuleID

StudentID, ResourceID ⟶ CompletionStatus, CompletionDate

AnnouncementID ⟶ AnnouncementDetails, PostedDate, ModuleID, TeacherID

3.3. Second Normalized Form (2NF)

These instructions must be followed in order to convert to 2NF:

- If an entity has a partial dependency, it can be eliminated by looking through all of its composite keys or unique identifiers.
- Partial dependency analysis across the tables is not necessary when representing a single unique identifier.

Before 2NF:

Student (**StudentID**, StudentName, StudentEmail, StudentDOB, StudentAddress, StudentPhone, ProgramID, ProgramName, ProgramDescription)

Module (**ModuleID**, ModuleName, Credits, TeacherID, TeacherName, Department)

Assessment (**AssessmentID**, Title, Deadline, Weightage, TotalMarks, **ModuleID***)

Result (**ResultID**, ObtainedMarks, Grade, **StudentID***, **AssessmentID***)

Resource (**ResourceID**, ResourceName, ResourceType, CompletionOrder**, ModuleID***)

ResourceCompletion (**StudentID***, **ResourceID***, CompletionStatus, CompletionDate)

Announcement (**AnnouncementID**, AnnouncementDetails, PostedDate, **ModuleID***, **TeacherID***)

Scenario of 2NF:

Checking for Partial Dependency

Student (**StudentID**, StudentName, StudentEmail, StudentDOB, StudentAddress, StudentPhone, ProgramID, ProgramName, ProgramDescription)

The attributes ProgramName, and ProgramDescription are depended on ProgramID and not on StudentID which shows that they are partially dependent. So, Program is separated into its own table.

StudentID ⟶ StudentName, StudentEmail, StudentDOB, StudentAddress, StudentPhone, ProgramID

ProgramID ⟶ ProgramName, ProgramDescription

Module (**ModuleID**, ModuleName, Credits, TeacherID, TeacherName, Department)

ProgramID and **ModuleID** form a composite key in the **ProgramModule** table, which resolves the many-to-many relationship between programs and modules. Also, the attributes ModuleName, Credits, and TeacherID are fully dependent on ModuleID and not on ProgramID; so, the partial dependency is removed accordingly.

ProgramID → ProgramName, ProgramDescription

ModuleID → ModuleName, Credits, TeacherID, TeacherName, Department

This becomes necessary to evaluate characteristics to see how many are functionally dependent on unique identifiers and how many are not because of the presence of composite keys or unique identifiers in this scenario.

ProgramID, ModuleID $\rightarrow$

ProgramModule (**ProgramID*, ModuleID***)

Similarly, a StudentModule table is introduced to resolve the many-to-many relationship between students and modules.

StudentID, ModuleID $\rightarrow$

StudentModule (**StudentID*, ModuleID***)

Modules can share assessments between different programs which leads to a many-to-many relationship between Module and Assessment. A single module can have multiple assessments and, the same assessment can be shared across modules in different programs.

To resolve this, a bridge table called ModuleAssessment is introduced.

ModuleID, AssessmentID $\rightarrow$

ModuleAssessment (**ModuleID*, AssessmentID***)

Final Tables after 2NF:

Student (**StudentID**, StudentName, StudentEmail, StudentDOB, StudentAddress, StudentPhone, **ProgramID***)

Program (**ProgramID**, ProgramName, ProgramDescription)

Module (**ModuleID**, ModuleName, Credits, TeacherID, TeacherName, Department)

ProgramModule (**ProgramID*, ModuleID***)

StudentModule (**StudentID*, ModuleID***)

Assessment (**AssessmentID**, Title, Deadline, Weightage, TotalMarks)

ModuleAssessment (**ModuleID*, AssessmentID***)

Result (**ResultID**, ObtainedMarks, Grade, **StudentID***, **AssessmentID***)

Resource (**ResourceID**, ResourceName, ResourceType, Duration, CompletionOrder, **ModuleID***)

ResourceCompletion (**StudentID***, **ResourceID***, CompletionStatus, CompletionDate)

Announcement (**AnnouncementID**, AnnouncementDetails, PostedDate, **ModuleID***, **TeacherID***)

3.4. Third Normalized Form (3NF)

The following guidelines must be followed in order to complete 3NF:

- Every table's transitive dependency must be eliminated.
- Every non-key attribute in a relation that has more than one must be checked for.
- Later separation and transitive dependency are necessary.

Before 3NF:

Module (**ModuleID**, ModuleName, Credits, TeacherID, TeacherName, Department)

Since the module, referred through its ID provides the non-key attributes such as TeacherName and Department based on TeacherID, the scenario must be examined for transitive dependency.

Scenario of 3NF:

ModuleID → TeacherID → TeacherName, Department

This indicates a transitive dependency because TeacherName and Department depend on TeacherID, which in turn depends on ModuleID. To eliminate this transitive dependency, teacher details must be separated into a distinct Teacher table, leaving only TeacherID in the Module table.

Functional Dependencies in 3NF:

StudentID → StudentName, StudentEmail, StudentDOB, StudentAddress, StudentPhone, ProgramID

ProgramID → ProgramName, ProgramDescription

ModuleID → ModuleName, Credits, TeacherID

TeacherID → TeacherName, Department

AssessmentID → Title, Deadline, Weightage, TotalMarks, ModuleID

ResultID → StudentID, AssessmentID, ObtainedMarks, Grade

ResultID, StudentID, AssessmentID → ObtainedMarks

ResourceID → ResourceName, ResourceType, CompletionOrder, ModuleID

StudentID, ResourceID → CompletionStatus, CompletionDate

AnnouncementID → AnnouncementDetails, PostedDate, ModuleID, TeacherID

Therefore, the final tables after normalization are as follows:

Student (**StudentID**, StudentName, StudentEmail, StudentDOB, StudentAddress, StudentPhone, **ProgramID***)

Program (**ProgramID**, ProgramName, ProgramDescription)

Module (**ModuleID**, ModuleName, Credits, **TeacherID***)

Teacher (**TeacherID**, TeacherName, Department)

ProgramModule (**ProgramID*, ModuleID***)

StudentModule (**StudentID*, ModuleID***)

Assessment (**AssessmentID**, Title, Deadline, Weightage, TotalMarks)

ModuleAssessment (**ModuleID*, AssessmentID***)

Result (**ResultID**, ObtainedMarks, Grade, **StudentID***, **AssessmentID***)

Resource (**ResourceID**, ResourceName, ResourceType, Duration, CompletionOrder, **ModuleID***)

ResourceCompletion (**StudentID***, **ResourceID***, CompletionStatus, CompletionDate)

Announcement (**AnnouncementID**, AnnouncementDetails, PostedDate, **ModuleID***, **TeacherID***)

4. Final ERD

Following the normalization process, all the tabulated forms are now grouped into a single chart, each of which is assigned a primary key and a foreign key in order to link or join two distinct tables that share common properties. The most critical visualization of all the procedures is the development of a bridge entity, which is responsible for reducing data redundancy and anomalies that might contaminate the database and make it appear messy. Thus, the Program-Module and Student-Module illustrating the Many to Many relationships are further broken down by the relationship shown by the specified bridge entity: ProgramModule and StudentModule.

Similarly, the relationship between modules and teachers follows a one-to-many structure, where each module is assigned to a specific teacher through a foreign key (TeacherID). Transitive dependencies are removed by separating teacher details, such as TeacherName and Department, into a distinct Teacher table, ensuring that module records only store the TeacherID. Additionally, the ModuleAssessment table is introduced to resolve the many-to-many relationship between module and assessment, which allows a single assessment to be shared across multiple modules in different programs. Each of the entities is connected to the others, providing the mandatory relationships on each side to demonstrate how the database values for each of the entities were optimized.

Furthermore, a ResourceCompletion table is established to track the progress of students on an individual resources within a module. This table makes sure that a resource must be completed before the access is given to the next one which also supports structured learning. The StudentModule table ensures that students are properly associated with the modules they are enrolled in, which resolves the many-to-many relationships and makes it possible to connect students to resources, assessments, and results for specific modules.

The relationship between Module and Assessment is further improvised through the ModuleAssessment bridge table, ensuring the flexibility in linking assessments to multiple modules. The relationship between Assessment and Result is made in such a way that Result depends on both AssessmentID and StudentID to accurately track student’s performance without redundancy.

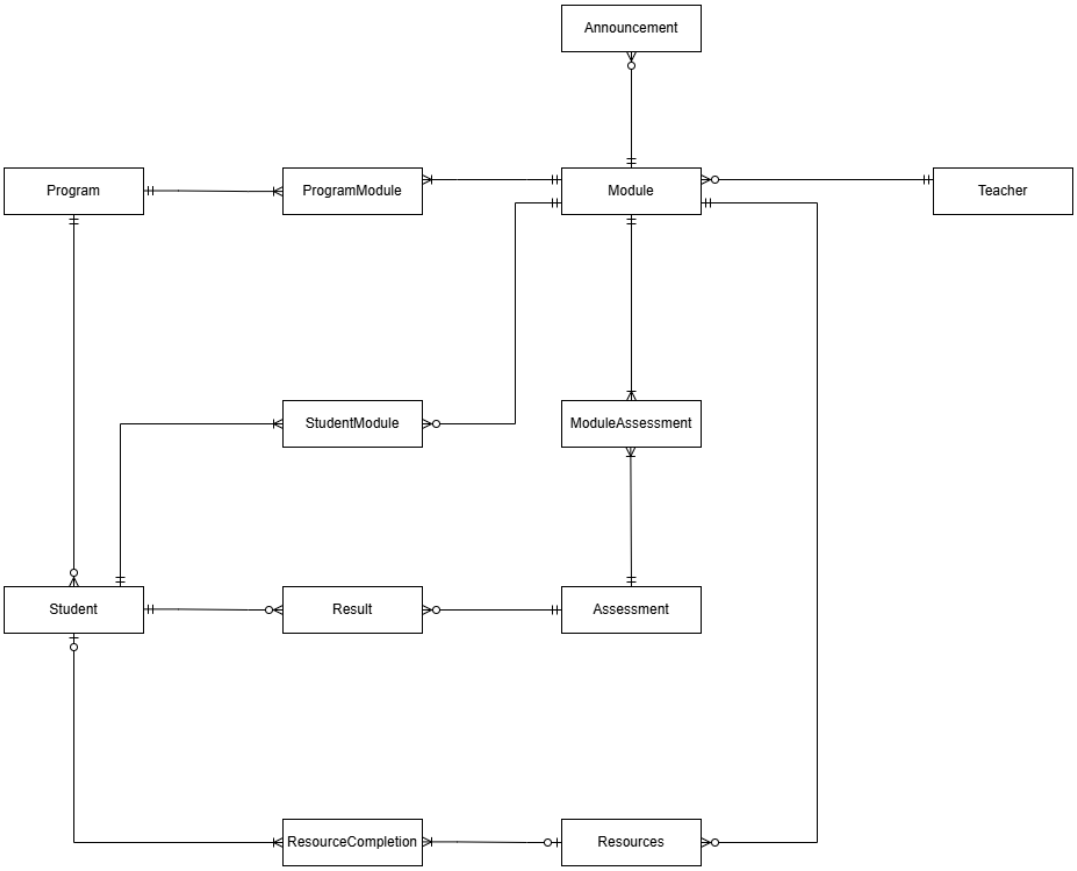

Figure 5: Final ERD

A Program consists of multiple modules, and each module can belong to many programs. This establishes a many-to-many relationship between programs and modules through the ProgramModule bridge table.

Modality: A program must have at least one module, a module cannot exist without being linked to a program.

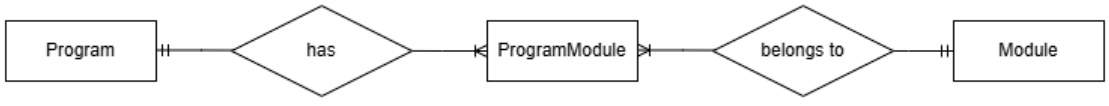

Figure 6: Finalized Relationship: Program-Module

A student can enroll in only one program, but a program has multiple students. This establishes a one-to-many relationship between Program and Student.

Modality: A student must be enrolled in one program; a program can exist without students.

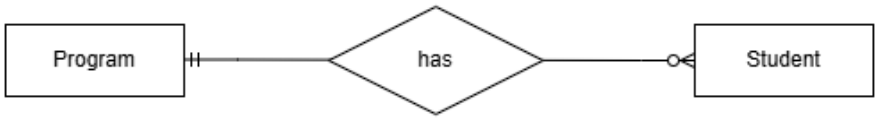

Figure 7: Finalized Relationship: Program-Student

A student can enroll in multiple modules as a part of their respective program, and a module can be associated with multiple students. This establishes a many-to-many relationship between Student and module.

Modality: A student must be enrolled in modules which is a part of their program; a module can exist without students.

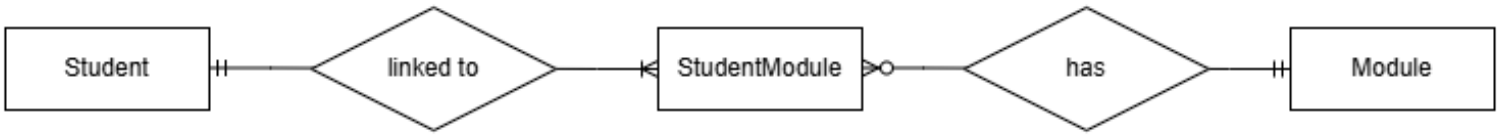

Figure 8: Finalized Relationship: Student-Module

A teacher can be assigned to multiple modules, but each module is taught by only one teacher. This establishes a one-to-many relationship between teachers and modules.

Modality: Each module must have a teacher; a teacher can exist without being assigned to a module.

Figure 9: Finalized Relationship: Teacher-Module

A module can have multiple assessments, but each assessment belongs to only one module and the assessment is shared across modules in various programs. This establishes a many-to-many relationship between modules and assessments.

Modality: Each assessment must belong to a module; a module must have at least one assessment.

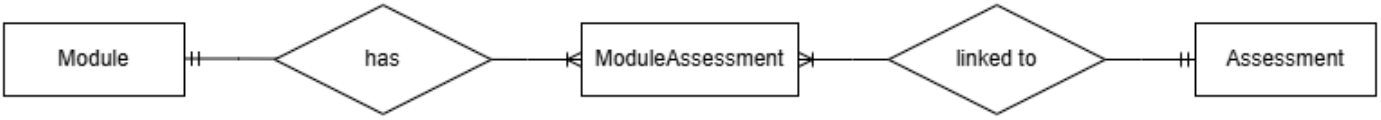

Figure 10: Finalized Relationship: Module-Assessment

A student can have multiple results, but each result belongs to one student and is associated with one assessment. This establishes a many-to-many relationship between Student and Assessment through the Result table.

Modality: Each result must belong to an assessment and a student; assessment or student can exist without a result.

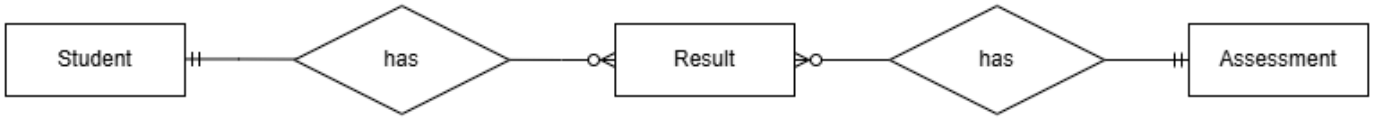

Figure 11: Finalized Relationship: Student-Assessment

A module can generate multiple announcements, but each announcement is tied to only one module. This establishes a one-to-many relationship between modules and announcements.

Modality: Announcements cannot exist without being linked to a module; a module may have no announcements.

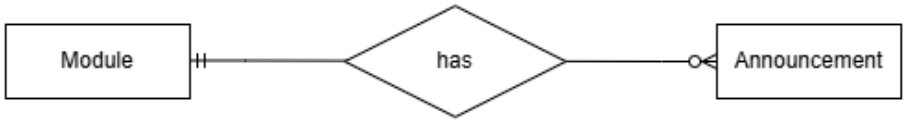

Figure 12: Finalized Relationship: Module-Announcement

A module can have multiple resources, but each resource is linked to only one module. This establishes a one-to-many relationship between modules and resources.

Modality: A resource cannot exist without being linked to a module; a module may have no resources.

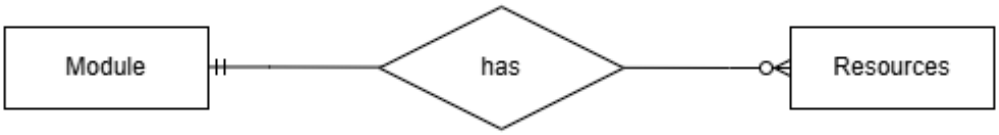

Figure 13: Finalized Relationship: Module-Resource

A student can complete multiple resources and a resource can be completed by multiple students. This establishes a many-to-many relationship between Student and Resources through the ResourceCompletion table. ResourceCompletion tracks if a student has completed a resource before being granted to the next one.

Modality: A student may not have interacted with a resource yet, and a resource may noy have been accessed by any students.

Figure 14: Finalized Relationship: Student-Resources

5. Data Dictionary

A data dictionary is a part of a database management system (DBMS) that contains a list of names, definitions, and attributes for the data elements that are used in the database. Metadata, or information about the database, is stored in the database. After that, these data elements are included into an information system, database, or research project.

It contains all of the data associated with tables or relationships, including the schema and constraints that were applied. Every piece of metadata is preserved. Metadata generally refers to data-related information. Therefore, a data dictionary or system directory is a single structure that contains the connection scheme and other metadata. A data dictionary, which contains all of the details about each relationship in the database, functions similarly to an A-Z dictionary in a relational database system.

The data dictionary consists of two words, data, which represents data collected from several sources, and dictionary, which represents where this data is available. Because it provides additional information about the connections between various database tables, the data dictionary is a crucial component of relational databases. A DBMS's data dictionary assists users in managing data in a systematic way, avoiding data redundancy (geeksforgeeks, 2024).

5.1. Data Dictionary for Student Table

Table 4: Data Dictionary for Student Table

S. No.	Attributes	Data Type	Size	Constraints
1.	StudentID	CHARACTER	25	PRIMARY KEY, NOT NULL, UNIQUE
2.	StudentName	CHARACTER	50	NOT NULL
3.	StudentEmail	CHARACTER	50	NOT NULL, UNIQUE
4.	StudentPhone	CHARACTER	15	NOT NULL, UNIQUE
5.	StudentDOB	DATE		NOT NULL
6.	StudentAddress	CHARACTER	100	NOT NULL
7.	ProgramID	CHARACTER	25	NOT NULL

5.2. Data Dictionary for Program Table

Table 5: Data Dictionary for Program Table

S. No.	Attributes	Data Type	Size	Constraints
1.	ProgramID	CHARACTER	25	PRIMARY KEY, NOT NULL, UNIQUE
2.	ProgramName	CHARACTER	50	NOT NULL
3.	ProgramDescription	CHARACTER	100	NOT NULL

5.3. Data Dictionary for Module Table

Table 6: Data Dictionary for Module Table

S. No.	Attributes	Data Type	Size	Constraints
1.	ModuleID	CHARACTER	25	PRIMARY KEY, NOT NULL, UNIQUE
2.	ModuleName	CHARACTER	50	NOT NULL
3.	Credits	NUMBER	3	NOT NULL
4.	TeacherID	CHARACTER	25	FOREIGN KEY, NOT NULL, UNIQUE

5.4. Data Dictionary for ProgramModule Table

Table 7: Data Dictionary for ProgramModule Table

S. No.	Attributes	Data Type	Size	Constraints	Composite Constraint
1.	ProgramID	CHARACTER	25	FOREIGN KEY, NOT NULL, UNIQUE	PRIMARY KEY
2.	ModuleID	CHARACTER	25	FOREIGN KEY, NOT NULL, UNIQUE	

5.5. Data Dictionary for StudentModule Table

Table 8: Data Dictionary for StudentModule Table

S. No.	Attributes	Data Type	Size	Constraints	Composite Constraint
1.	StudentID	CHARACTER	25	FOREIGN KEY, NOT NULL, UNIQUE	PRIMARY KEY
2.	ModuleID	CHARACTER	25	FOREIGN KEY, NOT NULL, UNIQUE	

5.6. Data Dictionary for Teacher Table

Table 9: Data Dictionary for Teacher Table

S. No.	Attributes	Data Type	Size	Constraints
1.	TeacherID	CHARACTER	25	PRIMARY KEY, NOT NULL, UNIQUE
2.	TeacherName	CHARACTER	50	NOT NULL
3.	Department	CHARACTER	50	NOT NULL

5.7. Data Dictionary for Assessment Table

Table 10: Data Dictionary for Assessment Table

S. No.	Attributes	Data Type	Size	Constraints
1.	AssessmentID	CHARACTER	25	PRIMARY KEY, NOT NULL, UNIQUE
2.	AssessmentTitle	CHARACTER	50	NOT NULL
3.	Deadline	DATE		NOT NULL
4.	Weightage	NUMBER	(5, 2)	NOT NULL
5.	TotalMarks	NUMBER	3	
6.	ModuleID	CHARACTER	25	FOREIGN KEY, NOT NULL, UNIQUE

5.8. Data Dictionary for ModuleAssessment Table

Table 11: Data Dictionary for ModuleAssessment Table

S. No.	Attributes	Data Type	Size	Constraints	Composite Constraint
1.	StudentID	CHARACTER	25	FOREIGN KEY, NOT NULL, UNIQUE	PRIMARY KEY
2.	ModuleID	CHARACTER	25	FOREIGN KEY, NOT NULL, UNIQUE	

5.9. Data Dictionary for Result Table

Table 12: Data Dictionary for Result Table

S. No.	Attributes	Data Type	Size	Constraints	Composite Constraint
1.	ResultID	CHARACTER	25	PRIMARY KEY, NOT NULL, UNIQUE	
2.	ObtainedMarks	NUMBER	(5, 2)	NOT NULL	
3.	Grade	NUMBER	5	NOT NULL	
4.	AssessmentID	CHARACTER	25	FOREIGN KEY, NOT NULL, UNIQUE	PRIMARY KEY
5.	StudentID	CHARACTER	25	FOREIGN KEY, NOT NULL, UNIQUE	

5.10. Data Dictionary for Announcement Table

Table 13: Data Dictionary for Announcement Table

S. No.	Attributes	Data Type	Size	Constraints
1.	AnnouncementID	CHARACTER	25	PRIMARY KEY, NOT NULL, UNIQUE
2.	AnnouncementDetails	CHARACTER	100	NOT NULL
3.	PostedDate	DATE		NOT NULL
4.	ModuleID	CHARACTER	25	FOREIGN KEY, NOT NULL, UNIQUE
5.	TeacherID	CHARACTER	25	FOREIGN KEY, NOT NULL, UNIQUE

5.11. Data Dictionary of Resources Table

Table 14: Data Dictionary for Resources Table

S. No.	Attributes	Data Type	Size	Constraints
1.	ResourceID	CHARACTER	25	PRIMARY KEY, NOT NULL, UNIQUE
2.	ResourceName	CHARACTER	50	NOT NULL
3.	ResourceType	CHARACTER	50	NOT NULL
4.	CompletionOrder	NUMBER	3	
5.	ModuleID	CHARACTER	25	FOREIGN KEY, NOT NULL, UNIQUE

5.12. Data Dictionary for ResourceCompletion Table

Table 15: Data Dictionary for ResourceCompletion Table

S. No.	Attributes	Data Type	Size	Constraints	Composite Constraint
1.	StudentID	CHARACTER	25	FOREIGN KEY, NOT NULL, UNIQUE	PRIMARY KEY
2.	ResourceID	CHARACTER	50	FOREIGN KEY, NOT NULL, UNIQUE	
3.	CompletionStatus	CHARACTER	15	NOT NULL	
4.	CompletionDate	DATE		NOT NULL	

6. Data Implementation

Based on the obtained normalized database and its tables, it is now necessary to start working on the implementation of the data and populating each of the finalized tables. The system is connected with the SQL Command Line followed by a user creation and its implementation to grant privileges so that it can access every resource to the user. Thus, the different steps of creation, description, insertion and selection is carried out to the database for its implementation, each of which are defined below.

Connect system

CREATE USER Bhumika_Karki **Identified by** 23047584;

GRANT CONNECT, RESOURCE to Bhumika_Karki;

CONNECT Bhumika_Karki/23047584;

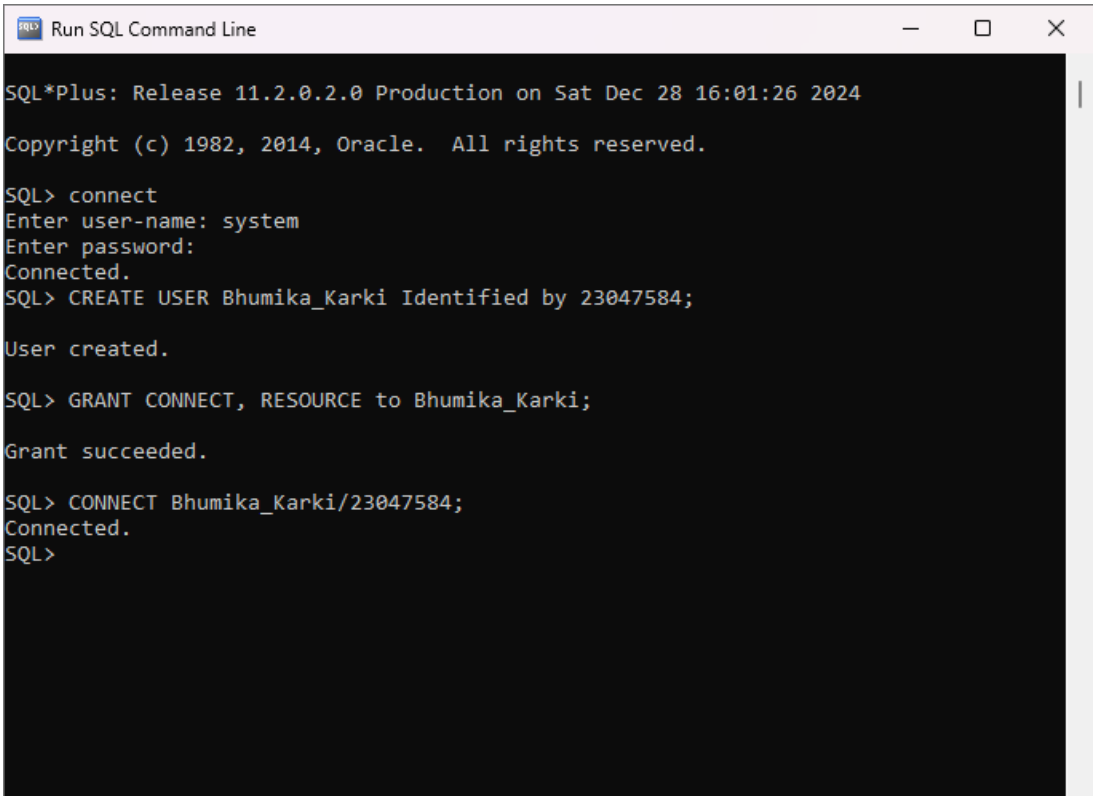

Figure 15: Creation and Connection of a new user

6.1. Implementation of Program Table

CREATE TABLE Program

(ProgramID **VARCHAR2 (25) PRIMARY KEY**,

ProgramName **VARCHAR2 (50) NOT NULL UNIQUE**,

ProgramDescription **VARCHAR2 (100) NOT NULL**);

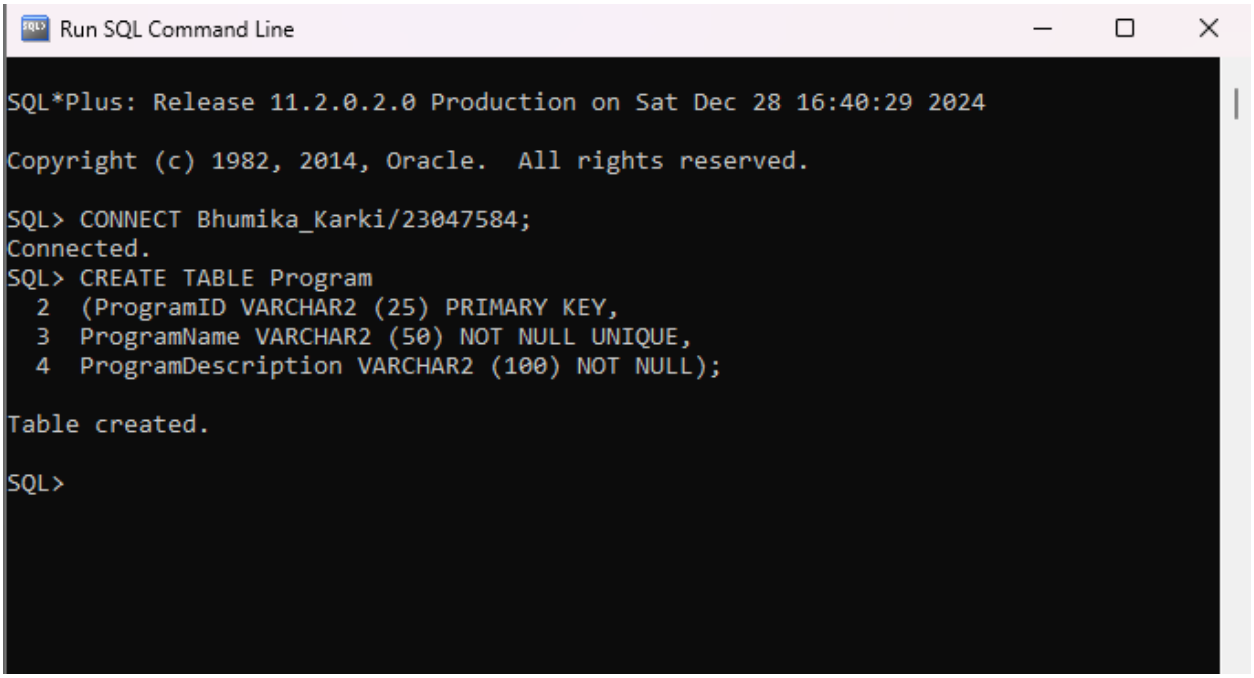

Figure 16: Creation of the Program Table

DESC Program;

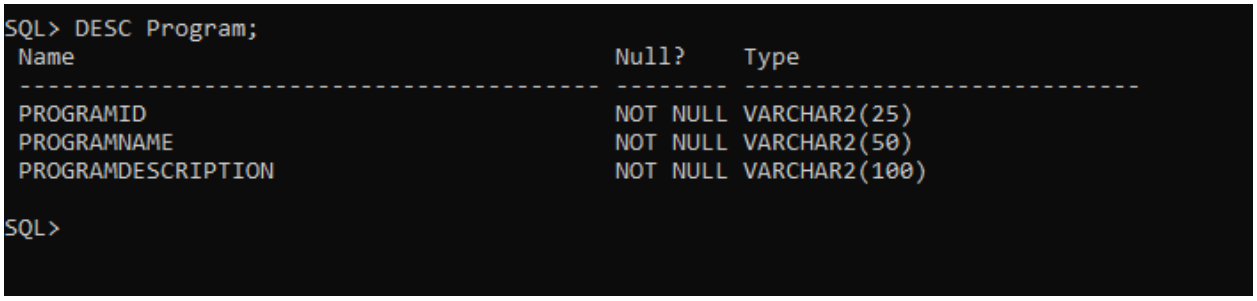

Figure 17: Description of the Program Table

INSERT INTO Program **VALUES** ('CS2001', 'Computing', 'BSc Program');

INSERT INTO Program **VALUES** ('CS2002', 'Networking', 'BSc Program');

INSERT INTO Program **VALUES** ('CS2003', 'Multimedia', 'BSc Program'');

INSERT INTO Program **VALUES** ('CS2004', 'Artificial Intelligence', 'BSc Program'');

INSERT INTO Program **VALUES** ('BA2001', 'Business Administration', 'BA Program'');

INSERT INTO Program **VALUES** ('BA2003', 'Accounting and Finance', 'BA Program');

INSERT INTO Program **VALUES** ('BA2002', 'Digital Business Management', 'BA Program'');

```
SQL> INSERT INTO Program VALUES ('CS2001', 'Computing', 'BSc Program');

1 row created.

SQL> INSERT INTO Program VALUES ('CS2002', 'Networking', 'BSc Program');

1 row created.

SQL> INSERT INTO Program VALUES ('CS2003', 'Multimedia', 'BSc Program');

1 row created.

SQL> INSERT INTO Program VALUES ('CS2004', 'Artificial Intelligence', 'BSc Program');

1 row created.

SQL> INSERT INTO Program VALUES ('BA2001', 'Business Administration', 'BA Program');

1 row created.

SQL> INSERT INTO Program VALUES ('BA2003', 'Accounting and Finance', 'BA Program');;
INSERT INTO Program VALUES ('BA2003', 'Accounting and Finance', 'BA Program');
                                                                              *
ERROR at line 1:
ORA-00911: invalid character


SQL> INSERT INTO Program VALUES ('BA2003', 'Accounting and Finance', 'BA Program');

1 row created.

SQL> INSERT INTO Program VALUES ('BA2002', 'Digital Business Management', 'BA Program');

1 row created.

SQL>
```

Figure 18: Insertion of values to the Program Table

SELECT * FROM Program;

```
SQL> SET LINESIZE 1000;
SQL> SET PAGESIZE 1000;
SQL> SELECT * FROM Program;

PROGRAMID            PROGRAMNAME                 PROGRAMDESCRIPTION
-------------------- --------------------------- ----------------------------
CS2001               Computing                   BSc Program
CS2002               Networking                  BSc Program
CS2003               Multimedia                  BSc Program
CS2004               Artificial Intelligence     BSc Program
BA2001               Business Administration     BA Program
BA2003               Accounting and Finance      BA Program
BA2002               Digital Business Management BA Program

7 rows selected.

SQL>
```

Figure 19: Selection of all records from the Program Table

6.2. Implementation of Teacher Table

CREATE TABLE Teacher

(TeacherID **VARCHAR2 (25) PRIMARY KEY**,

TeacherName **VARCHAR2 (50) NOT NULL**,

Department **VARCHAR2 (50) NOT NULL**);

```
SQL> CREATE TABLE Teacher
  2  (TeacherID VARCHAR2 (25) PRIMARY KEY,
  3  TeacherName VARCHAR2 (50) NOT NULL,
  4  Department VARCHAR2 (50) NOT NULL);

Table created.

SQL>
```

Figure 20: Creation of the Teacher Table

DESC Teacher;

```
SQL> DESC Teacher;
 Name                                              Null?    Type
 ------------------------------------------------- -------- ------------------------------------
 TEACHERID                                         NOT NULL VARCHAR2(25)
 TEACHERNAME                                       NOT NULL VARCHAR2(50)
 DEPARTMENT                                        NOT NULL VARCHAR2(50)

SQL>
```

Figure 21: Description of the Teacher Table

INSERT INTO Teacher **VALUES** ('T001', 'Manju Karki', 'Computing');

INSERT INTO Teacher **VALUES** ('T002', 'Sambhav Gurung', 'Networking');

INSERT INTO Teacher **VALUES** ('T003', 'Sophiya Karki', 'Multimedia');

INSERT INTO Teacher **VALUES** ('T004', 'Sandhya Thapa', 'Artificial Intelligence');

INSERT INTO Teacher **VALUES** ('T005', 'Binisha Sharma'', 'Business Administration');

INSERT INTO Teacher **VALUES** ('T006', 'Saugat Gurung', 'Accounting and Finance');

INSERT INTO Teacher **VALUES** ('T007', 'Biplove KC', 'Digital Business Management');

```
SQL> INSERT INTO Teacher VALUES ('T001', 'Manju Karki', 'Computing');

1 row created.

SQL> INSERT INTO Teacher VALUES ('T002', 'Sambhav Gurung', 'Networking');

1 row created.

SQL> INSERT INTO Teacher VALUES ('T003', 'Sophiya Karki', 'Multimedia');

1 row created.

SQL> INSERT INTO Teacher VALUES ('T004', 'Sandhya Thapa', 'Artificial Intelligence');

1 row created.

SQL> INSERT INTO Teacher VALUES ('T005', 'Binisha Sharma', 'Business Administration');

1 row created.

SQL> INSERT INTO Teacher VALUES ('T006', 'Saugat Gurung', 'Accounting and Finance');

1 row created.

SQL> INSERT INTO Teacher VALUES ('T007', 'Biplove KC', 'Digital Business Management');

1 row created.

SQL>
```

Figure 22: Insertion of values to the Teacher Table

SELECT * FROM Teacher;

```
SQL> SELECT * FROM Teacher;

TEACHERID            TEACHERNAME                                        DEPARTMENT
-------------------- -------------------------------------------------- --------------------------------------------------
T001                 Manju Karki                                        Computing
T002                 Sambhav Gurung                                     Networking
T003                 Sophiya Karki                                      Multimedia
T004                 Sandhya Thapa                                      Artificial Intelligence
T005                 Binisha Sharma                                     Business Administration
T006                 Saugat Gurung                                      Accounting and Finance
T007                 Biplove KC                                         Digital Business Management

7 rows selected.

SQL>
```

Figure 23: Selection of all records from the Teacher Table

6.3. Implementation of Module Table

CREATE TABLE Module

(ModuleID **VARCHAR2 (25) PRIMARY KEY**,

ModuleName **VARCHAR2 (50) NOT NULL**,

Credits **NUMBER (4,1) NOT NULL**,

TeacherID **VARCHAR2 (25) NOT NULL**,

FOREIGN KEY (TeacherID) **REFERENCES** Teacher (TeacherID));

```
SQL> CREATE TABLE Module
  2  (ModuleID VARCHAR2 (25) PRIMARY KEY,
  3  ModuleName VARCHAR2 (50) NOT NULL,
  4  Credits NUMBER (4,1) NOT NULL,
  5  TeacherID VARCHAR2 (25) NOT NULL,
  6  FOREIGN KEY (TeacherID) REFERENCES Teacher(TeacherID));

Table created.

SQL>
```

Figure 24: Creation of the Module Table

DESC Module;

```
SQL> DESC Module;
 Name                                      Null?    Type
 ----------------------------------------- -------- ------------------------------------
 MODULEID                                  NOT NULL VARCHAR2(25)
 MODULENAME                                NOT NULL VARCHAR2(50)
 CREDITS                                   NOT NULL NUMBER(4,1)
 TEACHERID                                 NOT NULL VARCHAR2(25)

SQL>
```

Figure 25: Description of the Module Table

INSERT INTO Module **VALUES** ('M004', 'Software Engineering', '30', 'T001');

INSERT INTO Module **VALUES** ('M007', 'Network OS', '30', 'T002');

INSERT INTO Module **VALUES** ('M009', 'Network Security', '15'', ''T002');

INSERT INTO Module **VALUES** ('M002'', 'Databases', '15', 'T003');

INSERT INTO Module **VALUES** ('M001', 'Programming', '15', ''T004'');

INSERT INTO Module **VALUES** ('M903'', 'Professional Ethics', '15', 'T005');

INSERT INTO Module **VALUES** (‘M005’, 'Account', ‘30', 'T006’);

INSERT INTO Module **VALUES** ('M006’, 'Economics', '30', 'T007’);

INSERT INTO Module **VALUES** ('M010', 'Digital Marketing', '15', 'T007’);

INSERT INTO Module **VALUES** ('M008’, 'AI Basics', '15', 'T004');

```
SQL> INSERT INTO Module VALUES ('M004', 'Software Engineering', '30', 'T001');

1 row created.

SQL> INSERT INTO Module VALUES ('M007', 'Network OS', '30', 'T002');

1 row created.

SQL> INSERT INTO Module VALUES ('M009', 'Network Security', '15', 'T002');

1 row created.

SQL> INSERT INTO Module VALUES ('M002', 'Databases', '15', 'T003');

1 row created.

SQL> INSERT INTO Module VALUES ('M001', 'Programming', '15', 'T004');

1 row created.

SQL> INSERT INTO Module VALUES ('M003', 'Professional Ethics', '15', 'T005');

1 row created.

SQL> INSERT INTO Module VALUES ('M005', 'Account', '30', 'T006');

1 row created.

SQL> INSERT INTO Module VALUES ('M006', 'Economics', '30', 'T007');

1 row created.

SQL> INSERT INTO Module VALUES ('M010', 'Digital Marketing', '15', 'T007');

1 row created.

SQL> INSERT INTO Module VALUES ('M008', 'AI Basics', '15', 'T004');

1 row created.
```

Figure 27: Insertion of values to the Module Table

SELECT * FROM Module;

```
SQL> SELECT * FROM Module;

MODULEID             MODULENAME                                          CREDITS TEACHERID
-------------------- -------------------------------------------------- ---------- ----------
M004                 Software Engineering                                       30 T001
M007                 Network OS                                                 30 T002
M009                 Network Security                                           15 T002
M002                 Databases                                                  15 T003
M001                 Programming                                                15 T004
M003                 Professional Ethics                                        15 T005
M005                 Account                                                    30 T006
M006                 Economics                                                  30 T007
M010                 Digital Marketing                                          15 T007
M008                 AI Basics                                                  15 T004
```

Figure 26: Selection of all records from the Module Table

6.4. Implementation of ProgramModule Table

CREATE TABLE ProgramModule

(ProgramID **VARCHAR2 (25) NOT NULL**,

ModuleID **VARCHAR2 (25) NOT NULL**,

PRIMARY KEY (ProgramID, ModuleID),

FOREIGN KEY (ProgramID) **REFERENCES** Program (ProgramID),

FOREIGN KEY (ModuleID) **REFERENCES** Module (ModuleID));

```
SQL> CREATE TABLE ProgramModule
  2  (ProgramID VARCHAR2 (25) NOT NULL,
  3  ModuleID VARCHAR2 (25) NOT NULL,
  4  PRIMARY KEY (ProgramID, ModuleID),
  5  FOREIGN KEY (ProgramID) REFERENCES Program(ProgramID),
  6  FOREIGN KEY (ModuleID) REFERENCES Module(ModuleID));

Table created.

SQL>
```

Figure 28: Creation of the ProgramModule Table

DESC ProgramModule

```
SQL> DESC ProgramModule
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 PROGRAMID                                 NOT NULL VARCHAR2(25)
 MODULEID                                  NOT NULL VARCHAR2(25)

SQL>
```

Figure 29: Description of the ProgramModule Table

INSERT INTO ProgramModule **VALUES** ('CS2001', 'M004');

INSERT INTO ProgramModule **VALUES** ('CS2002', ‘M007');

INSERT INTO ProgramModule **VALUES** (‘CS2002', ‘M009’);

INSERT INTO ProgramModule **VALUES** ('CS2003’, ‘M002’);

INSERT INTO ProgramModule **VALUES** ('CS2004', ‘M001');

INSERT INTO ProgramModule **VALUES** (‘CS2004’, ‘M008’);

INSERT INTO ProgramModule **VALUES** ('BA2001', 'M003’):

INSERT INTO ProgramModule **VALUES** (‘BA2002', ‘M006’);

INSERT INTO ProgramModule **VALUES** ('BA2002', 'M010');

INSERT INTO ProgramModule **VALUES** ('BA2003', 'M005');

INSERT INTO ProgramModule **VALUES** ('CS2001', ‘M001');

```
Run SQL Command Line
SQL> INSERT INTO ProgramModule VALUES ('CS2001', 'M004');

1 row created.

SQL> INSERT INTO ProgramModule VALUES ('CS2002', 'M007');

1 row created.

SQL> INSERT INTO ProgramModule VALUES ('CS2002', 'M009');

1 row created.

SQL> INSERT INTO ProgramModule VALUES ('CS2003', 'M002');

1 row created.

SQL> INSERT INTO ProgramModule VALUES ('CS2004', 'M001');

1 row created.

SQL> INSERT INTO ProgramModule VALUES ('CS2004', 'M008');

1 row created.

SQL> INSERT INTO ProgramModule VALUES ('BA2001', 'M003');

1 row created.

SQL> INSERT INTO ProgramModule VALUES ('BA2002', 'M006');

1 row created.

SQL> INSERT INTO ProgramModule VALUES ('BA2002', 'M010');

1 row created.

SQL> INSERT INTO ProgramModule VALUES ('BA2003', 'M005');

1 row created.

SQL> INSERT INTO ProgramModule VALUES ('CS2001', 'M001');

1 row created.
```

Figure 30: Insertion of values to the ProgramModule Table

SELECT * FROM ProgramModule;

```
SQL> SELECT * FROM ProgramModule;

PROGRAMID                 MODULEID
------------------------- -------------------------
CS2001                    M004
CS2002                    M007
CS2002                    M009
CS2003                    M002
CS2004                    M001
CS2004                    M008
BA2001                    M003
BA2002                    M006
BA2002                    M010
BA2003                    M005
CS2001                    M001

11 rows selected.
```

Figure 31: Selection of all records from the ProgramModule Table

6.5. Implementation of Student Table

CREATE TABLE Student

(StudentID **VARCHAR2 (25) PRIMARY KEY**,

StudentName **VARCHAR2 (50) NOT NULL**,

StudentEmail **VARCHAR2 (50) NOT NULL UNIQUE**,

StudentPhone **VARCHAR2 (15) NOT NULL UNIQUE**,

StudentDOB **DATE NOT NULL**,

StudentAddress **VARCHAR2 (100) NOT NULL**,

ProgramID **VARCHAR2 (25) NOT NULL**,

FOREIGN KEY (ProgramID) **REFERENCES** Program(ProgramID));

```
SQL> CREATE TABLE Student
  2  (StudentID VARCHAR2 (25) PRIMARY KEY,
  3  StudentName VARCHAR2 (50) NOT NULL,
  4  StudentEmail VARCHAR2 (50) NOT NULL UNIQUE,
  5  StudentPhone VARCHAR2 (15) NOT NULL UNIQUE,
  6  StudentDOB DATE NOT NULL,
  7  StudentAddress VARCHAR2 (100) NOT NULL,
  8  ProgramID VARCHAR2 (25) NOT NULL,
  9  FOREIGN KEY (ProgramID) REFERENCES Program(ProgramID));

Table created.

SQL>
```

Figure 32: Creation of the Student Table

DESC Student

```
SQL> DESC Student
 Name                      Null?    Type
 ------------------------- -------- ----------------
 STUDENTID                 NOT NULL VARCHAR2(25)
 STUDENTNAME               NOT NULL VARCHAR2(50)
 STUDENTEMAIL              NOT NULL VARCHAR2(50)
 STUDENTPHONE              NOT NULL VARCHAR2(15)
 STUDENTDOB                NOT NULL DATE
 STUDENTADDRESS            NOT NULL VARCHAR2(100)
 PROGRAMID                 NOT NULL VARCHAR2(25)

SQL>
```

Figure 33: Description of the Student Table

INSERT INTO Student **VALUES** ('S001', 'Subin Khatiwada', 'subin@gmail.com', '9867452367', TO_DATE('2003-01-01', 'YYYY-MM-DD'), 'Dhading', 'CS2001');

INSERT INTO Student **VALUES** ('S002', 'Aarya Singh', 'aarya@gmail.com', '9840452367', TO_DATE('2004-05-01', 'YYYY-MM-DD'), 'Jhapa', 'CS2002');

INSERT INTO Student **VALUES** ('S003', 'Roshani Rajbhandari', 'roshani@gmail.com', '9840452378', TO_DATE('2004-07-15', 'YYYY-MM-DD'), 'Kathmandu', 'CS2003');

INSERT INTO Student **VALUES** ('S004', 'Deepesh Bashyal', 'deepesh@gmail.com', '9840672399', TO_DATE('2004-10-12', 'YYYY-MM-DD'), 'Palpa', 'CS2004');

INSERT INTO Student **VALUES** ('S005', 'Nischita Acharya', 'nischita@gmail.com', '9840672380', TO_DATE('2005-12-10', 'YYYY-MM-DD'), 'Bhaktapur', 'BA2001');

INSERT INTO Student **VALUES** ('S006', 'Sita Gurung', 'sita@gmail.com', '9841652380', TO_DATE('2003-05-10', 'YYYY-MM-DD'), 'Dhading', 'BA2002');

INSERT INTO Student **VALUES** ('S007', 'Manish Thapa', 'manish@gmail.com', '9841652310', TO_DATE('2003-04-21', 'YYYY-MM-DD'), 'Kathmandu', 'BA2003');

INSERT INTO Student **VALUES** ('S008', 'Sabina Bhattarai', 'sabina@gmail.com', '9840688310', TO_DATE('2003-01-12', 'YYYY-MM-DD'), 'Kathmandu', 'BA2003');

INSERT INTO Student **VALUES** ('S009', 'Dikshya Bashyal', 'dikshya@gmail.com', '9840672366', TO_DATE('2005-10-10', 'YYYY-MM-DD'), 'Palpa', 'CS2001');

COMMIT;

```
SQL> INSERT INTO Student VALUES ('S001', 'Subin Khatiwada', 'subin@gmail.com', '9867452367', TO_DATE('2003-01-01', 'YYYY-MM-DD'), 'Dhading', 'CS2001');
1 row created.
SQL> INSERT INTO Student VALUES ('S002', 'Aarya Singh', 'aarya@gmail.com', '9840452367', TO_DATE('2004-05-01', 'YYYY-MM-DD'), 'Jhapa', 'CS2002');
1 row created.
SQL> INSERT INTO Student VALUES ('S003', 'Roshani Rajbhandari', 'roshani@gmail.com', '9840452378', TO_DATE('2004-07-15', 'YYYY-MM-DD'), 'Kathmandu', 'CS2003');
1 row created.
SQL> INSERT INTO Student VALUES ('S004', 'Deepesh Bashyal', 'deepesh@gmail.com', '9840672399', TO_DATE('2004-10-12', 'YYYY-MM-DD'), 'Palpa', 'CS2004');
1 row created.
SQL> INSERT INTO Student VALUES ('S005', 'Nischita Acharya', 'nischita@gmail.com', '9840672380', TO_DATE('2005-12-10', 'YYYY-MM-DD'), 'Bhaktapur', 'BA2001');
1 row created.
SQL> INSERT INTO Student VALUES ('S006', 'Sita Gurung', 'sita@gmail.com', '9841652380', TO_DATE('2003-05-10', 'YYYY-MM-DD'), 'Dhading', 'BA2002');
1 row created.
SQL> INSERT INTO Student VALUES ('S007', 'Manish Thapa', 'manish@gmail.com', '9841652310', TO_DATE('2003-04-21', 'YYYY-MM-DD'), 'Kathmandu', 'BA2003');
1 row created.
SQL> INSERT INTO Student VALUES ('S008', 'Sabina Bhattarai', 'sabina@gmail.com', '9840688310', TO_DATE('2003-01-12', 'YYYY-MM-DD'), 'Kathmandu', 'BA2003');
1 row created.
SQL> INSERT INTO Student VALUES ('S009', 'Dikshya Bashyal', 'dikshya@gmail.com', '9840672366', TO_DATE('2005-10-10', 'YYYY-MM-DD'), 'Palpa', 'CS2001');
1 row created.
SQL> COMMIT;
Commit complete.
SQL>
```

Figure 34: Insertion of values to the Student Table

SELECT * FROM Student;

```
SQL> SELECT * FROM Student;

STUDENTID            STUDENTNAME          STUDENTEMAIL            STUDENTPHONE    STUDENTDO STUDENTADDRESS       PROGRAMID
-------------------- -------------------- ----------------------- -------------- --------- -------------------- ----------
S001                 Subin Khatiwada      subin@gmail.com         9867452367     01-JAN-03 Dhading              CS2001
S002                 Aarya Singh          aarya@gmail.com         9840452367     01-MAY-04 Jhapa                CS2002
S003                 Roshani Rajbhandari  roshani@gmail.com       9840452378     15-JUL-04 Kathmandu            CS2003
S004                 Deepesh Bashyal      deepesh@gmail.com       9840672399     12-OCT-04 Palpa                CS2004
S005                 Nischita Acharya     nischita@gmail.com      9840672380     10-DEC-05 Bhaktapur            BA2001
S006                 Sita Gurung          sita@gmail.com          9841652380     10-MAY-03 Dhading              BA2002
S007                 Manish Thapa         manish@gmail.com        9841652310     21-APR-03 Kathmandu            BA2003
S008                 Sabina Bhattarai     sabina@gmail.com        9840688310     12-JAN-03 Kathmandu            BA2003
S009                 Dikshya Bashyal      dikshya@gmail.com       9840672366     10-OCT-05 Palpa                CS2001

9 rows selected.

SQL>
```

Figure 35: Selection of all records from the Student Table

6.6. Implementation of StudentModule Table

CREATE TABLE StudentModule

(StudentID **VARCHAR2 (25) NOT NULL**,

ModuleID **VARCHAR2 (25) NOT NULL**,

PRIMARY KEY (StudentID, ModuleID),

FOREIGN KEY (StudentID) **REFERENCES** Student(StudentID),

FOREIGN KEY (ModuleID) **REFERENCES** Module(ModuleID));

```
SQL> CREATE TABLE StudentModule
  2  (StudentID VARCHAR2 (25) NOT NULL,
  3  ModuleID VARCHAR2 (25) NOT NULL,
  4  PRIMARY KEY (StudentID, ModuleID),
  5  FOREIGN KEY (StudentID) REFERENCES Student(StudentID),
  6  FOREIGN KEY (ModuleID) REFERENCES Module(ModuleID));

Table created.

SQL>
```

Figure 36: Creation of the StudentModule Table

desc StudentModule

```
SQL> desc StudentModule
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 STUDENTID                                 NOT NULL VARCHAR2(25)
 MODULEID                                  NOT NULL VARCHAR2(25)

SQL>
```

Figure 37: Description of the StudentModule Table

INSERT INTO StudentModule (StudentID, ModuleID)

SELECT s.StudentID, pm.ModuleID

FROM Student s

JOIN ProgramModule pm **ON** s.ProgramID = pm.ProgramID;

```
SQL> INSERT INTO StudentModule (StudentID, ModuleID)
  2  SELECT s.StudentID, pm.ModuleID
  3  FROM Student s
  4  JOIN ProgramModule pm ON s.ProgramID = pm.ProgramID;

14 rows created.

SQL>
```

Figure 38: Insertion of all the values to the StudentModule Table

SELECT * FROM StudentModule;

```
SQL> SELECT * FROM StudentModule;

STUDENTID                 MODULEID
------------------------- -------------------------
S005                      M003
S006                      M006
S006                      M010
S008                      M005
S007                      M005
S009                      M001
S001                      M001
S009                      M004
S001                      M004
S002                      M007
S002                      M009
S003                      M002
S004                      M001
S004                      M008

14 rows selected.

SQL>
```

Figure 39: Selection of all records from the StudentModule Table

6.7. Implementation of Assessment Table

CREATE TABLE Assessment

(AssessmentID **VARCHAR2 (25) PRIMARY KEY**,

ModuleID **VARCHAR2 (25) NOT NULL**,

AssessmentTitle **VARCHAR2 (50) NOT NULL**,

Deadline **DATE NOT NULL**,

Weightage **NUMBER (5,2) NOT NULL**,

FOREIGN KEY (ModuleID) **REFERENCES** Module(ModuleID));

```
SQL> CREATE TABLE Assessment
  2  (AssessmentID VARCHAR2 (25) PRIMARY KEY,
  3  ModuleID VARCHAR2 (25) NOT NULL,
  4  AssessmentTitle VARCHAR2 (50) NOT NULL,
  5  Deadline DATE NOT NULL,
  6  Weightage NUMBER (5,2) NOT NULL,
  7  FOREIGN KEY (ModuleID) REFERENCES Module(ModuleID));

Table created.

SQL>
```

Figure 40: Creation of the Assessment Table

DESC Assessment

```
SQL> DESC Assessment
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 ASSESSMENTID                              NOT NULL VARCHAR2(25)
 MODULEID                                  NOT NULL VARCHAR2(25)
 ASSESSMENTTITLE                           NOT NULL VARCHAR2(50)
 DEADLINE                                  NOT NULL DATE
 WEIGHTAGE                                 NOT NULL NUMBER(5,2)

SQL>
```

Figure 41: Description of the Assessment Table

INSERT INTO Assessment **VALUES** (‘A001’, ‘M001’, ‘CW1’, TO DATE ('2025-01-15', ‘YYYY-MM-DD’), 40.00);

INSERT INTO Assessment **VALUES** ('A002’, ‘M008’, ‘First Term', TO_DATE ('2025-01-17', ‘YYYY-MM-DD’), 20.00);

INSERT INTO Assessment **VALUES** ('A003', ‘M004’, 'Project', TO_DATE ('2025-01-30', ‘YYYY-MM-DD’), 20.00);

INSERT INTO Assessment **VALUES** ('A004’, ‘M002', ‘CW1’, TO DATE ('2025-01-20’, ‘YYYY-MM-DD’), 50.00);

INSERT INTO Assessment **VALUES** (‘A005', ‘M005’, ‘CW5’, TO_DATE ('2025-01-25', ‘YYYY-MM-DD’), 60.00);

INSERT INTO Assessment **VALUES** ('A006', ‘M003’, ‘MidTerm’, TO_DATE ('2025-02-05’, ‘YYYY-MM-DD’), 30.00);

INSERT INTO Assessment **VALUES** (‘A007’, ‘M006’, ‘CW1’, TO_DATE (‘2025-01-15', ‘YYYY-MM-DD’), 50.00);

```
SQL> INSERT INTO Assessment VALUES ('A001', 'M001', 'CW1', TO_DATE('2025-01-15', 'YYYY-MM-DD'), 40.00);
1 row created.
SQL> INSERT INTO Assessment VALUES ('A002', 'M008', 'First Term', TO_DATE('2025-01-17', 'YYYY-MM-DD'), 20.00);
1 row created.
SQL> INSERT INTO Assessment VALUES ('A003', 'M004', 'Project', TO_DATE('2025-01-30', 'YYYY-MM-DD'), 20.00);
1 row created.
SQL> INSERT INTO Assessment VALUES ('A004', 'M002', 'CW1', TO_DATE('2025-01-20', 'YYYY-MM-DD'), 50.00);
1 row created.
SQL> INSERT INTO Assessment VALUES ('A005', 'M005', 'CW5', TO_DATE('2025-01-25', 'YYYY-MM-DD'), 60.00);
1 row created.
SQL> INSERT INTO Assessment VALUES ('A006', 'M003', 'MidTerm', TO_DATE('2025-02-05', 'YYYY-MM-DD'), 30.00);
1 row created.
SQL> INSERT INTO Assessment VALUES ('A007', 'M006', 'CW1', TO_DATE('2025-01-15', 'YYYY-MM-DD'), 50.00);
1 row created.
SQL> COMMIT;
Commit complete.
```

Figure 42: Insertion of values to the Assessment Table

SELECT FROM * Assessment;

```
SQL> SELECT * FROM Assessment;

ASSESSMENTID             MODULEID                ASSESSMENTTITLE                                    DEADLINE   WEIGHTAGE
------------------------ ------------------------ -------------------------------------------------- --------- ----------
A001                     M001                    CW1                                                15-JAN-25         40
A002                     M008                    First Term                                         17-JAN-25         20
A003                     M004                    Project                                            30-JAN-25         20
A004                     M002                    CW1                                                20-JAN-25         50
A005                     M005                    CW5                                                25-JAN-25         60
A006                     M003                    MidTerm                                            05-FEB-25         30
A007                     M006                    CW1                                                15-JAN-25         50

7 rows selected.

SQL>
```

Figure 43: Selection of all records from the Assessment Table (1)

ALTER TABLE Assessment

ADD PassingMarks **NUMBER (5) DEFAULT 40 NOT NULL**;

DESC Assessment;

```
SQL> ALTER TABLE Assessment
  2  ADD PassingMarks NUMBER(5) DEFAULT 40 NOT NULL;

Table altered.

SQL> DESC Assessment;
 Name                      Null?    Type
 ------------------------- -------- ----------------
 ASSESSMENTID              NOT NULL VARCHAR2(25)
 MODULEID                  NOT NULL VARCHAR2(25)
 ASSESSMENTTITLE           NOT NULL VARCHAR2(50)
 DEADLINE                  NOT NULL DATE
 WEIGHTAGE                 NOT NULL NUMBER(5,2)
 PASSINGMARKS              NOT NULL NUMBER(5)
```

Figure 44: Alteration of the Assessment Table (1)

ALTER TABLE Assessment

ADD TotalMarks **NUMBER(5) DEFAULT 100 NOT NULL**;

Desc Assessment

```
SQL> ALTER TABLE Assessment
  2  ADD TotalMarks NUMBER(5) DEFAULT 100 NOT NULL;

Table altered.

SQL> desc assessment
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 ASSESSMENTID                              NOT NULL VARCHAR2(25)
 MODULEID                                  NOT NULL VARCHAR2(25)
 ASSESSMENTTITLE                           NOT NULL VARCHAR2(50)
 DEADLINE                                  NOT NULL DATE
 WEIGHTAGE                                 NOT NULL NUMBER(5,2)
 PASSINGMARKS                              NOT NULL NUMBER(5)
 TOTALMARKS                                NOT NULL NUMBER(5)

SQL>
```

Figure 45: Alteration of the Assessment Table (2)

ALTER TABLE Assessment

DROP COLUMN ModuleID;

SELECT * FROM Assessment;

```
SQL> ALTER TABLE Assessment
  2  DROP COLUMN ModuleID;

Table altered.

SQL> SELECT * FROM Assessment;

ASSESSMENTID             ASSESSMENTTITLE                                    DEADLINE   WEIGHTAGE PASSINGMARKS TOTALMARKS
------------------------ -------------------------------------------------- --------- ---------- ------------ ----------
A001                     CW1                                                15-JAN-25         40           40        100
A002                     First Term                                         17-JAN-25         20           40        100
A003                     Project                                            30-JAN-25         20           40        100
A004                     CW1                                                20-JAN-25         50           40        100
A005                     CW5                                                25-JAN-25         60           40        100
A006                     MidTerm                                            05-FEB-25         30           40        100
A007                     CW1                                                15-JAN-25         50           40        100

7 rows selected.

SQL>
```

Figure 46: Alteration and Selection of all records from the Assessment Table (2)

6.8. Implementation of ModuleAssessment Table

CREATE TABLE ModuleAssessment

(ModuleID **VARCHAR2 (25) NOT NULL**,

AssessmentID **VARCHAR2 (25) NOT NULL**,

PRIMARY KEY (ModuleID, AssessmentID),

FOREIGN KEY (ModuleID) **REFERENCES** Module (ModuleID),

FOREIGN KEY (AssessmentID) **REFERENCES** Assessment (AssessmentID));

Desc ModuleAssessment

```
SQL> CREATE TABLE ModuleAssessment
  2  (ModuleID VARCHAR2 (25) NOT NULL,
  3  AssessmentID VARCHAR2 (25) NOT NULL,
  4  PRIMARY KEY (ModuleID, AssessmentID),
  5  FOREIGN KEY (ModuleID) REFERENCES Module(ModuleID),
  6  FOREIGN KEY (AssessmentID) REFERENCES Assessment(AssessmentID));

Table created.

SQL> desc ModuleAssessment
 Name                                                                                  Null?    Type
 ------------------------------------------------------------------------------------- -------- ------------------------
 MODULEID                                                                              NOT NULL VARCHAR2(25)
 ASSESSMENTID                                                                          NOT NULL VARCHAR2(25)

SQL>
```

Figure 47: Creation and Description of ModuleAssessment Table

INSERT INTO ModuleAssessment **VALUES** ('M001', 'A001');

INSERT INTO ModuleAssessment **VALUES** ('M008', 'A002');

INSERT INTO ModuleAssessment **VALUES** ('M004', 'A003');

INSERT INTO ModuleAssessment **VALUES** ('M002', 'A004');

INSERT INTO ModuleAssessment **VALUES** ('M005', 'A005');

INSERT INTO ModuleAssessment **VALUES** ('M003', 'A006');

INSERT INTO ModuleAssessment **VALUES** ('M006', 'A007');

```
SQL> INSERT INTO ModuleAssessment VALUES ('M001', 'A001');

1 row created.

SQL> INSERT INTO ModuleAssessment VALUES ('M008', 'A002');

1 row created.

SQL> INSERT INTO ModuleAssessment VALUES ('M004', 'A003');

1 row created.

SQL> INSERT INTO ModuleAssessment VALUES ('M002', 'A004');

1 row created.

SQL> INSERT INTO ModuleAssessment VALUES ('M005', 'A005');

1 row created.

SQL> INSERT INTO ModuleAssessment VALUES ('M003', 'A006');

1 row created.

SQL> INSERT INTO ModuleAssessment VALUES ('M006', 'A007');

1 row created.

SQL>
```

Figure 48: Insertion of values to the ModuleAssessment Table

SELECT * FROM ModuleAssessment;

```
SQL> SELECT * FROM ModuleAssessment;

MODULEID                  ASSESSMENTID
------------------------- -------------------------
M001                      A001
M008                      A002
M004                      A003
M002                      A004
M005                      A005
M003                      A006
M006                      A007

7 rows selected.

SQL>
```

Figure 49: Selection of all records from the ModuleAssessment Table

6.9. Implementation of Result Table

CREATE TABLE Result

(ResultID **VARCHAR2 (25) PRIMARY KEY**,

StudentID **VARCHAR2 (25) NOT NULL**,

AssessmentID **VARCHAR2 (25) NOT NULL**,

ObtainedMarks **NUMBER (5,2) NOT NULL**,

Grade **VARCHAR2 (2) DEFAULT 'NA'**,

FOREIGN KEY (StudentID) **REFERENCES** Student (StudentID),

FOREIGN KEY (AssessmentID) **REFERENCES** Assessment (AssessmentID));

```
SQL> CREATE TABLE Result
  2  (ResultID VARCHAR2 (25) PRIMARY KEY,
  3  StudentID VARCHAR2 (25) NOT NULL,
  4  AssessmentID VARCHAR2 (25) NOT NULL,
  5  ObtainedMarks NUMBER (5,2) NOT NULL,
  6  Grade VARCHAR2 (2) DEFAULT 'NA',
  7  FOREIGN KEY (StudentID) REFERENCES Student(StudentID),
  8  FOREIGN KEY (AssessmentID) REFERENCES Assessment(AssessmentID));

Table created.
```

Figure 50: Creation of the Result Table

Desc Result

```
SQL> desc Result
 Name                                                                                  Null?    Type
 ------------------------------------------------------------------------------------- -------- ------------------------------------
 RESULTID                                                                              NOT NULL VARCHAR2(25)
 STUDENTID                                                                             NOT NULL VARCHAR2(25)
 ASSESSMENTID                                                                          NOT NULL VARCHAR2(25)
 OBTAINEDMARKS                                                                         NOT NULL NUMBER(5,2)
 GRADE                                                                                          VARCHAR2(2)

SQL>
```

Figure 51: Description of the Result Table

INSERT INTO Result **VALUES** ('RES001', 'S001', 'A001', '85.00', 'NA');

INSERT INTO Result **VALUES** ('RES002', 'S001', 'A003', '82.00', 'NA');

INSERT INTO Result **VALUES** ('RES003', 'S003', 'A004', '73.00', 'NA');

INSERT INTO Result **VALUES** ('RES004', 'S004', 'A002', '81.00', 'NA');

INSERT INTO Result **VALUES** ('RES005', 'S004', 'A001', '87.00', 'NA');

INSERT INTO Result **VALUES** ('RES006', 'S005', 'A006', '35.00', 'NA');

INSERT INTO Result **VALUES** ('RES007', 'S006', 'A007', '37.00', 'NA');

INSERT INTO Result **VALUES** ('RES008', 'S007', 'A005', '75.00', 'NA');

```
SQL> INSERT INTO Result VALUES ('RES001', 'S001', 'A001', '85.00', 'NA');
1 row created.
SQL> INSERT INTO Result VALUES ('RES002', 'S001', 'A003', '82.00', 'NA');
1 row created.
SQL> INSERT INTO Result VALUES ('RES003', 'S003', 'A004', '73.00', 'NA');
1 row created.
SQL> INSERT INTO Result VALUES ('RES004', 'S004', 'A002', '81.00', 'NA');
1 row created.
SQL> INSERT INTO Result VALUES ('RES005', 'S004', 'A001', '87.00', 'NA');
1 row created.
SQL> INSERT INTO Result VALUES ('RES006', 'S005', 'A006', '35.00', 'NA');
1 row created.
SQL> INSERT INTO Result VALUES ('RES007', 'S006', 'A007', '37.00', 'NA');
1 row created.
SQL> INSERT INTO Result VALUES ('RES008', 'S007', 'A005', '75.00', 'NA');
1 row created.
SQL>
```

Figure 52: Insertion of values to the Result Table

```
UPDATE Result
SET Grade = CASE
    WHEN (ObtainedMarks / (SELECT TotalMarks FROM Assessment WHERE Assessment.AssessmentID = Result.AssessmentID)) >= 0.8 THEN 'A'
    WHEN (ObtainedMarks / (SELECT TotalMarks FROM Assessment WHERE Assessment.AssessmentID = Result.AssessmentID)) >= 0.7 THEN 'B'
    WHEN (ObtainedMarks / (SELECT TotalMarks FROM Assessment WHERE Assessment.AssessmentID = Result.AssessmentID)) >= 0.6 THEN 'C'
    WHEN (ObtainedMarks / (SELECT TotalMarks FROM Assessment WHERE Assessment.AssessmentID = Result.AssessmentID)) >= 0.5 THEN 'D'
    WHEN (ObtainedMarks / (SELECT TotalMarks FROM Assessment WHERE Assessment.AssessmentID = Result.AssessmentID)) >= 0.4 THEN 'E'
    ELSE 'F'
END;

SELECT * FROM Result;
```

```
SQL> UPDATE Result
  2  SET Grade = CASE
  3     WHEN (ObtainedMarks / (SELECT TotalMarks FROM Assessment WHERE Assessment.AssessmentID = Result.AssessmentID)) >= 0.8 THEN 'A'
  4     WHEN (ObtainedMarks / (SELECT TotalMarks FROM Assessment WHERE Assessment.AssessmentID = Result.AssessmentID)) >= 0.7 THEN 'B'
  5     WHEN (ObtainedMarks / (SELECT TotalMarks FROM Assessment WHERE Assessment.AssessmentID = Result.AssessmentID)) >= 0.6 THEN 'C'
  6     WHEN (ObtainedMarks / (SELECT TotalMarks FROM Assessment WHERE Assessment.AssessmentID = Result.AssessmentID)) >= 0.5 THEN 'D'
  7     WHEN (ObtainedMarks / (SELECT TotalMarks FROM Assessment WHERE Assessment.AssessmentID = Result.AssessmentID)) >= 0.4 THEN 'E'
  8     ELSE 'F'
  9  END;

8 rows updated.

SQL> SELECT * FROM Result;

RESULTID                STUDENTID               ASSESSMENTID            OBTAINEDMARKS GRADE
----------------------- ----------------------- ----------------------- ------------- -----
RES001                  S001                    A001                              85 A
RES002                  S001                    A003                              82 A
RES003                  S003                    A004                              73 B
RES004                  S004                    A002                              81 A
RES005                  S004                    A001                              87 A
RES006                  S005                    A006                              35 F
RES007                  S006                    A007                              37 F
RES008                  S007                    A005                              75 B

8 rows selected.

SQL>
```

Figure 53: Updating and Selecting all records from the Result Table

6.10. Implementation of Announcement Table

CREATE TABLE Announcement

(AnnouncementID **VARCHAR2 (25) PRIMARY KEY**,

ModuleID **VARCHAR2 (25) NOT NULL**,

AnnouncementDetails **VARCHAR2 (100) NOT NULL**,

PostedDate **DATE NOT NULL**,

FOREIGN KEY (ModuleID) **REFERENCES** Module (ModuleID));

```
SQL> CREATE TABLE Announcement
  2  (AnnouncementID VARCHAR2 (25) PRIMARY KEY,
  3  ModuleID VARCHAR2 (25) NOT NULL,
  4  AnnouncementDetails VARCHAR2 (100) NOT NULL,
  5  PostedDate DATE NOT NULL,
  6  FOREIGN KEY (ModuleID) REFERENCES Module(ModuleID));

Table created.
```

Figure 54: Creation of the Announcement Table

ALTER TABLE Announcement

ADD TeacherID **VARCHAR2 (25);**

UPDATE Announcement

SET TeacherID = (**SELECT** TeacherID

FROM Module

WHERE Module.ModuleID = Announcement.ModuleID);

ALTER TABLE Announcement

MODIFY TeacherID **VARCHAR2 (25) NOT NULL**;

ALTER TABLE Announcement

ADD CONSTRAINT FK_TeacherID

FOREIGN KEY (TeacherID) **REFERENCES** Teacher(TeacherID);

```
SQL> ALTER TABLE Announcement
  2  ADD TeacherID VARCHAR2 (25);

Table altered.

SQL> UPDATE Announcement
  2  SET TeacherID = (SELECT TeacherID
  3                   FROM Module
  4                   WHERE Module.ModuleID = Announcement.ModuleID);

7 rows updated.

SQL> ALTER TABLE Announcement
  2  MODIFY TeacherID VARCHAR2 (25) NOT NULL;

Table altered.

SQL> ALTER TABLE Announcement
  2  ADD CONSTRAINT FK_TeacherID
  3  FOREIGN KEY (TeacherID) REFERENCES Teacher(TeacherID);

Table altered.

SQL>
```

Figure 55: Alterration of the Announcement Table

DESC Announcement;

```
SQL> desc Announcement
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 ANNOUNCEMENTID                            NOT NULL VARCHAR2(25)
 MODULEID                                  NOT NULL VARCHAR2(25)
 ANNOUNCEMENTDETAILS                       NOT NULL VARCHAR2(100)
 POSTEDDATE                                NOT NULL DATE
 TEACHERID                                 NOT NULL VARCHAR2(25)

SQL>
```

Figure 56: Description of the Announcement Table

INSERT INTO Announcement **VALUES** ('AN001', 'M001', 'Registration for Hackathon 2025 is now open.', TO_DATE('2024-12-30', 'YYYY-MM-DD'));

INSERT INTO Announcement **VALUES** ('AN002', 'M007', 'It is mandatory to complete the Quiz Questions', TO_DATE('2024-12-28', 'YYYY-MM-DD'));

INSERT INTO Announcement **VALUES** ('AN003', 'M002', 'Lecture for Database Queries is Re-scheduled to Monday 5th May', TO_DATE('2024-05-03', 'YYYY-MM-DD'));

INSERT INTO Announcement **VALUES** ('AN004', 'M003', 'Guidelines for MidTerm assessment is available.', TO_DATE('2024-12-30', 'YYYY-MM-DD'));

INSERT INTO Announcement **VALUES** ('AN005', 'M008', 'Final Exam for AI Basics will be held on 7th January,2025', TO_DATE('2024-12-30', 'YYYY-MM-DD'));

INSERT INTO Announcement **VALUES** ('AN006', 'M006', 'Students are required to prepare a presentation for Economics.', TO_DATE('2024-05-15', 'YYYY-MM-DD'));

INSERT INTO Announcement **VALUES** ('AN007', 'M010', 'Extra Class for Digital Marketing scheduled for Monday, 22nd May.', TO_DATE('2024-05-17', 'YYYY-MM-DD'));

```
SQL> INSERT INTO Announcement VALUES ('AN001', 'M001', 'Registration for Hackathon 2025 is now open.', TO_DATE('2024-12-30', 'YYYY-MM-DD'));
1 row created.
SQL> INSERT INTO Announcement VALUES ('AN002', 'M007', 'It is mandatory to complete the Quiz Questions', TO_DATE('2024-12-28', 'YYYY-MM-DD'));
1 row created.
SQL> INSERT INTO Announcement VALUES ('AN003', 'M002', 'Lecture for Database Queries is Re-scheduled to Monday 5th May', TO_DATE('2024-05-03', 'YYYY-MM-DD'));
1 row created.
SQL> INSERT INTO Announcement VALUES ('AN004', 'M003', 'Guidelines for MidTerm assessment is available.', TO_DATE('2024-12-30', 'YYYY-MM-DD'));
1 row created.
SQL> INSERT INTO Announcement VALUES ('AN005', 'M008', 'Final Exam for AI Basics will be held on 7th January,2025', TO_DATE('2024-12-30', 'YYYY-MM-DD'));
1 row created.
SQL> INSERT INTO Announcement VALUES ('AN006', 'M006', 'Students are required to prepare a presentation for Economics.', TO_DATE('2024-05-15', 'YYYY-MM-DD'));
1 row created.
SQL> INSERT INTO Announcement VALUES ('AN007', 'M010', 'Extra Class for Digital Marketing scheduled for Monday, 22nd May.', TO_DATE('2024-05-17', 'YYYY-MM-DD'));
1 row created.
SQL>
```

Figure 57: Insertion of values to the Announcement Table

SELECT * FROM Announcement;

```
SQL> SELECT * FROM Announcement;

ANNOUNCEME MODULEID   ANNOUNCEMENTDETAILS                                                    POSTEDDATE   TEACHERID
---------- ---------- ------------------------------------------------------------------------------ ----------- ----------
AN001      M001       Registration for Hackathon 2025 is now open.                           30-DEC-24    T004
AN002      M007       It is mandatory to complete the Quiz Questions                         28-DEC-24    T002
AN003      M002       Lecture for Database Queries is Re-scheduled to Monday 5th May         03-MAY-24    T003
AN004      M003       Guidelines for MidTerm assessment is available.                        30-DEC-24    T005
AN005      M008       Final Exam for AI Basics will be held on 7th January,2025              30-DEC-24    T004
AN006      M006       Students are required to prepare a presentation for Economics.         15-MAY-24    T007
AN007      M010       Extra Class for Digital Marketing scheduled for Monday, 22nd May.      17-MAY-24    T007

7 rows selected.

SQL>
```

Figure 58: Selection of all records from the Announcement Table

6.11. Implementation of Resources Table

CREATE TABLE Resources

(ResourceID **VARCHAR2 (25) PRIMARY KEY**,

ModuleID **VARCHAR2 (25) NOT NULL**,

ResourceName **VARCHAR2 (50) NOT NULL**,

ResourceType **VARCHAR2 (50) NOT NULL**,

FOREIGN KEY (ModuleID) **REFERENCES** Module (ModuleID));

```
SQL> CREATE TABLE Resources
  2  (ResourceID VARCHAR2 (25) PRIMARY KEY,
  3  ModuleID VARCHAR2 (25) NOT NULL,
  4  ResourceName VARCHAR2 (50) NOT NULL,
  5  ResourceType VARCHAR2 (50) NOT NULL,
  6  FOREIGN KEY (ModuleID) REFERENCES Module(ModuleID));

Table created.

SQL>
```

Figure 59: Creation of the Resources Table

DESC Resources

```
SQL> DESC Resources
 Name                      Null?    Type
 ------------------------- -------- ----------------
 RESOURCEID                NOT NULL VARCHAR2(25)
 MODULEID                  NOT NULL VARCHAR2(25)
 RESOURCENAME              NOT NULL VARCHAR2(50)
 RESOURCETYPE              NOT NULL VARCHAR2(50)

SQL>
```

Figure 60: Description of the Resources Table

INSERT INTO Resources **VALUES** ('R001', 'M001', 'Java Book', 'PDF Document');

INSERT INTO Resources **VALUES** ('R002', 'M007', 'Unix/Linux', 'Quizzes');

INSERT INTO Resources **VALUES** ('R003', 'M002', 'Database Ebook', 'E-Books');

INSERT INTO Resources **VALUES** ('R004', 'M008', 'Machine Learning', 'Code Repositories');

INSERT INTO Resources **VALUES** ('R005', 'M003', 'Human Psychology', 'Case Studies');

INSERT INTO Resources **VALUES** ('R006', 'M010', 'Digital Marketing', 'Simulations');

INSERT INTO Resources **VALUES** ('R007', 'M006', 'Economics', 'Infographics');

```
SQL> INSERT INTO Resources VALUES ('R001', 'M001', 'Java Book', 'PDF Document');

1 row created.

SQL> INSERT INTO Resources VALUES ('R002', 'M007', 'Unix/Linux', 'Quizzes');

1 row created.

SQL> INSERT INTO Resources VALUES ('R003', 'M002', 'Database Ebook', 'E-Books');

1 row created.

SQL> INSERT INTO Resources VALUES ('R004', 'M008', 'Machine Learning', 'Code Repositories');

1 row created.

SQL> INSERT INTO Resources VALUES ('R005', 'M003', 'Human Psychology', 'Case Studies');

1 row created.

SQL> INSERT INTO Resources VALUES ('R006', 'M010', 'Digital Marketing', 'Simulations');

1 row created.

SQL> INSERT INTO Resources VALUES ('R007', 'M006', 'Economics', 'Infographics');

1 row created.

SQL>
```

Figure 61: Insertion of values to the Resources Table

SELECT * FROM Resources;

```
SQL> SELECT * FROM Resources;

RESOURCEID           MODULEID             RESOURCENAME                                       RESOURCETYPE
-------------------- -------------------- -------------------------------------------------- --------------------------------------------------
R001                 M001                 Java Book                                          PDF Document
R002                 M007                 Unix/Linux                                         Quizzes
R003                 M002                 Database Ebook                                     E-Books
R004                 M008                 Machine Learning                                   Code Repositories
R005                 M003                 Human Psychology                                   Case Studies
R006                 M010                 Digital Marketing                                  Simulations
R007                 M006                 Economics                                          Infographics

7 rows selected.

SQL>
```

Figure 62: Selection of all records from the Resources Table

6.12. Implementation of ResourceCompletion Table

CREATE TABLE ResourceCompletion

(StudentID **VARCHAR2 (25) NOT NULL**,

ResourceID **VARCHAR2 (25) NOT NULL**,

CompletionStatus **VARCHAR2 (15) NOT NULL**,

CompletionDate **DATE**,

CompletionOrder **NUMBER (2,0)**,

PRIMARY KEY (StudentID, ResourceID),

FOREIGN KEY (StudentID) **REFERENCES** Student (StudentID),

FOREIGN KEY (ResourceID) **REFERENCES** Resources (ResourceID));

```
SQL> CREATE TABLE ResourceCompletion
  2  (StudentID VARCHAR2 (25) NOT NULL,
  3  ResourceID VARCHAR2 (25) NOT NULL,
  4  CompletionStatus VARCHAR2 (15) NOT NULL,
  5  CompletionDate DATE,
  6  CompletionOrder NUMBER (2,0),
  7  PRIMARY KEY (StudentID, ResourceID),
  8  FOREIGN KEY (StudentID) REFERENCES Student(StudentID),
  9  FOREIGN KEY (ResourceID) REFERENCES Resources(ResourceID));

Table created.

SQL>
```

Figure 63: Creation of ResourceCompletion Table

Desc ResourceCompletion

```
SQL> desc ResourceCompletion
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 STUDENTID                                 NOT NULL VARCHAR2(25)
 RESOURCEID                                NOT NULL VARCHAR2(25)
 COMPLETIONSTATUS                          NOT NULL VARCHAR2(15)
 COMPLETIONDATE                                     DATE
 COMPLETIONORDER                                    NUMBER(2)

SQL>
```

Figure 64: Description of ResourceCompletion Table

INSERT INTO ResourceCompletion **VALUES** ('S004', 'R001', 'Completed', TO_DATE('2025-01-02', 'YYYY-MM-DD'), 1);

INSERT INTO ResourceCompletion **VALUES** ('S004', 'R004', 'Completed', TO_DATE('2025-01-10', 'YYYY-MM-DD'), 2);

INSERT INTO ResourceCompletion **VALUES** ('S002', 'R002', 'Incomplete', NULL, 1);

INSERT INTO ResourceCompletion **VALUES** ('S003', 'R003', 'Completed', TO_DATE('2025-01-12', 'YYYY-MM-DD'), 1);

INSERT INTO ResourceCompletion **VALUES** ('S005', 'R005', 'Completed', TO_DATE('2025-01-05', 'YYYY-MM-DD'), 1);

INSERT INTO ResourceCompletion **VALUES** ('S006', 'R006', 'Incomplete', NULL, 1);

INSERT INTO ResourceCompletion **VALUES** ('S006', 'R007', 'Incomplete', NULL, 2);

```
SQL> INSERT INTO ResourceCompletion VALUES ('S004', 'R001', 'Completed', TO_DATE('2025-01-02', 'YYYY-MM-DD'), 1);

1 row created.

SQL> INSERT INTO ResourceCompletion VALUES ('S004', 'R004', 'Completed', TO_DATE('2025-01-10', 'YYYY-MM-DD'), 2);

1 row created.

SQL> INSERT INTO ResourceCompletion VALUES ('S002', 'R002', 'Incomplete', NULL, 1);

1 row created.

SQL> INSERT INTO ResourceCompletion VALUES ('S003', 'R003', 'Completed', TO_DATE('2025-01-12', 'YYYY-MM-DD'), 1);

1 row created.

SQL> INSERT INTO ResourceCompletion VALUES ('S005', 'R005', 'Completed', TO_DATE('2025-01-05', 'YYYY-MM-DD'), 1);

1 row created.

SQL> INSERT INTO ResourceCompletion VALUES ('S006', 'R006', 'Incomplete', NULL, 1);

1 row created.

SQL> INSERT INTO ResourceCompletion VALUES ('S006', 'R007', 'Incomplete', NULL, 2);

1 row created.

SQL>
```

Figure 65: Insertion of values to the ResourceCompletion Table

SELECT * FROM ResourceCompletion;

```
SQL> SELECT * FROM ResourceCompletion;

STUDENTID    RESOURCEID   COMPLETIONSTATUS     COMPLETIONDATE       COMPLETIONORDER
------------ ------------ -------------------- -------------------- ---------------
S004         R001         Completed            02-JAN-25                          1
S004         R004         Completed            10-JAN-25                          2
S002         R002         Incomplete                                              1
S003         R003         Completed            12-JAN-25                          1
S005         R005         Completed            05-JAN-25                          1
S006         R006         Incomplete                                              1
S006         R007         Incomplete                                              2

7 rows selected.

SQL>
```

Figure 66: Selection of all records from the ResourceCompletion Table

7. Database Querying

Once all the implementation process is carried out and values are inserted inside the tables, it is necessary to query the system to find out the accuracy of the data entered inside the tables to provide informational as well as transactional answers to different queries.

7.1. Information Query

7.1.1. Information Query: 1

```
SELECT p.ProgramName,
COUNT (s.StudentID) AS TotalStudents FROM Program p
LEFT JOIN Student s ON p.ProgramID = s.ProgramID
GROUP BY p.ProgramName;
```

```
SQL> SELECT p.ProgramName,
  2  COUNT (s.StudentID) AS TotalStudents FROM Program p
  3  LEFT JOIN Student s ON p.ProgramID = s.ProgramID
  4  GROUP BY p.ProgramName;

PROGRAMNAME                                        TOTALSTUDENTS
-------------------------------------------------- -------------
Digital Business Management                                    1
Networking                                                     1
Artificial Intelligence                                        1
Business Administration                                        1
Computing                                                      2
Multimedia                                                     1
Accounting and Finance                                         2

7 rows selected.

SQL>
```

Figure 67: Information Query: 1

7.1.2. Information Query: 2

SELECT AnnouncementID, AnnouncementDetails, PostedDate

FROM Announcement

WHERE ModuleID = 'M002'

AND PostedDate **BETWEEN** TO_DATE('01-MAY-2024', 'DD-MON-YYYY') AND TO_DATE('28-MAY-2024', 'DD-MM-YYYY');

```
SQL> SELECT AnnouncementID, AnnouncementDetails, PostedDate
  2  FROM Announcement
  3  WHERE ModuleID = 'M002'
  4  AND PostedDate BETWEEN TO_DATE('01-MAY-2024', 'DD-MON-YYYY') AND TO_DATE('28-MAY-2024', 'DD-MM-YYYY');

ANNOUNCEMENTID           ANNOUNCEMENTDETAILS                                                  POSTEDDATE
------------------------ -------------------------------------------------------------------- ---------------
AN003                    Lecture for Database Queries is Re-scheduled to Monday 5th May       03-MAY-24

SQL>
```

Figure 68: Information Query:2

7.1.3. Information Query: 3

SELECT m.ModuleName,

COUNT (r.ResourceID) AS TotalResources

FROM Module m

LEFT JOIN Resources r **ON** m.ModuleID = r.ModuleID

WHERE m.ModuleName **LIKE** 'D%'

GROUP BY m.ModuleName;

```
SQL> SELECT m.ModuleName,
  2  COUNT (r.ResourceID) AS TotalResources
  3  FROM Module m
  4  LEFT JOIN Resources r ON m.ModuleID = r.ModuleID
  5  WHERE m.ModuleName LIKE 'D%'
  6  GROUP BY m.ModuleName;

MODULENAME                                         TOTALRESOURCES
-------------------------------------------------- --------------
Digital Marketing                                               1
Databases                                                       1

SQL>
```

Figure 69: Information Query:3

7.1.4. Information Query: 4

SELECT DISTINCT s.StudentName, p.ProgramName

FROM Student s

JOIN Program p **ON** s.ProgramID = p.ProgramID

JOIN StudentModule sm **ON** s.StudentID = sm.StudentID

WHERE sm.ModuleID = 'M005'

AND s.StudentID **NOT IN**

(**SELECT** r.StudentID

FROM Result r

JOIN Assessment a **ON** r.AssessmentID = a.AssessmentID

JOIN ModuleAssessment ma **ON** ma.AssessmentID = a.AssessmentID

WHERE ma.ModuleID = 'M005');

```
SQL> SELECT DISTINCT s.StudentName, p.ProgramName
  2  FROM Student s
  3  JOIN Program p ON s.ProgramID = p.ProgramID
  4  JOIN StudentModule sm ON s.StudentID = sm.StudentID
  5  WHERE sm.ModuleID = 'M005'
  6  AND s.StudentID NOT IN
  7     (SELECT r.StudentID
  8     FROM Result r
  9     JOIN Assessment a ON r.AssessmentID = a.AssessmentID
 10     JOIN ModuleAssessment ma ON ma.AssessmentID = a.AssessmentID
 11     WHERE ma.ModuleID = 'M005');

STUDENTNAME                                        PROGRAMNAME
-------------------------------------------------- --------------------------------------------------
Sabina Bhattarai                                   Accounting and Finance

SQL>
```

Figure 70: Information Query:4

7.1.5. Information Query: 5

SELECT t.TeacherName,

COUNT (m.ModuleID) **AS** TotalModules

FROM Teacher t

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

GROUP BY t.TeacherName

HAVING COUNT (m.ModuleID) > 1;

```
SQL> SELECT t.TeacherName,
  2  COUNT (m.ModuleID) AS TotalModules
  3  FROM Teacher t
  4  JOIN Module m ON t.TeacherID = m.TeacherID
  5  GROUP BY t.TeacherName
  6  HAVING COUNT (m.ModuleID) > 1;

TEACHERNAME                                        TOTALMODULES
-------------------------------------------------- ------------
Sandhya Thapa                                                 2
Biplove KC                                                    2
Sambhav Gurung                                                2

SQL>
```

Figure 71: Information Query:5

7.2. Transaction Query

7.2.1. Transaction Query: 1

SELECT ma.ModuleID, m.ModuleName, **MAX**(a.Deadline) **AS** Deadline

FROM ModuleAssessment ma

JOIN Module m **ON** ma.ModuleID = m.ModuleID

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

GROUP BY ma.ModuleID, m.ModuleName

HAVING MAX(a.Deadline) = (**SELECT MAX**(Deadline) **FROM** Assessment);

```
SQL> SELECT ma.ModuleID, m.ModuleName, MAX(a.Deadline) AS Deadline
  2  FROM ModuleAssessment ma
  3  JOIN Module m ON ma.ModuleID = m.ModuleID
  4  JOIN Assessment a ON ma.AssessmentID = a.AssessmentID
  5  GROUP BY ma.ModuleID, m.ModuleName
  6  HAVING MAX(a.Deadline) = (SELECT MAX(Deadline) FROM Assessment);

MODULEID                  MODULENAME                                         DEADLINE
------------------------- -------------------------------------------------- ---------
M003                      Professional Ethics                                05-FEB-25

SQL>
```

Figure 72: Transaction Query: 1

7.2.2. Transaction Query: 2

```
SELECT * FROM
(SELECT s.StudentName,
SUM (r.ObtainedMarks) AS TotalScore
FROM Student s
JOIN Result r ON s.StudentID = r.StudentID
GROUP BY s.StudentName
ORDER BY TotalScore DESC
)
WHERE ROWNUM <=3;
```

```
SQL> SELECT * FROM
  2  (SELECT s.StudentName,
  3  SUM (r.ObtainedMarks) AS TotalScore
  4  FROM Student s
  5  JOIN Result r ON s.StudentID = r.StudentID
  6  GROUP BY s.StudentName
  7  ORDER BY TotalScore DESC
  8  )
  9  WHERE ROWNUM <=3;

STUDENTNAME                                        TOTALSCORE
-------------------------------------------------- ----------
Deepesh Bashyal                                           168
Subin Khatiwada                                           167
Manish Thapa                                               75
```

Figure 73: Transaction Query: 2

7.2.3. Transaction Query: 3

SELECT p.ProgramName,

COUNT (a.AssessmentID) **AS** TotalAssessments,

AVG (r.ObtainedMarks) **AS** AverageScore

FROM Program p JOIN ProgramModule pm **ON** p.ProgramID = pm.ProgramID

JOIN ModuleAssessment ma **ON** pm.ModuleID = ma.ModuleID

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

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

GROUP BY p.ProgramName;

```
SQL> SELECT p.ProgramName,
  2  COUNT (a.AssessmentID) AS TotalAssessments,
  3  AVG (r.ObtainedMarks) AS AverageScore
  4  FROM Program p JOIN ProgramModule pm ON p.ProgramID = pm.ProgramID
  5  JOIN ModuleAssessment ma ON pm.ModuleID = ma.ModuleID
  6  JOIN Assessment a ON ma.AssessmentID = a.AssessmentID
  7  JOIN Result r ON a.AssessmentID = r.AssessmentID
  8  GROUP BY p.ProgramName;

PROGRAMNAME                                        TOTALASSESSMENTS AVERAGESCORE
-------------------------------------------------- ---------------- ------------
Digital Business Management                                       1           37
Artificial Intelligence                                           3   84.3333333
Business Administration                                           1           35
Computing                                                         3   84.6666667
Multimedia                                                        1           73
Accounting and Finance                                            1           75

6 rows selected.

SQL>
```

Figure 74: Transaction Query: 3

7.2.4. Transaction Query: 4

```
SELECT DISTINCT s.StudentName, sm.ModuleID, r.ObtainedMarks
FROM Student s
JOIN StudentModule sm ON s.StudentID = sm.StudentID
JOIN Module m ON sm.ModuleID = m.ModuleID
JOIN Result r ON s.StudentID = r.StudentID
JOIN Assessment a ON r.AssessmentID = a.AssessmentID
JOIN ModuleAssessment ma ON a.AssessmentID = ma.AssessmentID
WHERE m.ModuleName = 'Databases'
AND r.ObtainedMarks >=
 (SELECT AVG(r2.ObtainedMarks)
FROM Result r2
JOIN Assessment a2 ON r2.AssessmentID = a2.AssessmentID
JOIN ModuleAssessment ma2 ON a2.AssessmentID = ma2.AssessmentID
JOIN Module m2 ON ma2.ModuleID = m2.ModuleID
WHERE m2.ModuleName = 'Databases');
```

```
SQL> SELECT DISTINCT s.StudentName, sm.ModuleID, r.ObtainedMarks
  2  FROM Student s
  3  JOIN StudentModule sm ON s.StudentID = sm.StudentID
  4  JOIN Module m ON sm.ModuleID = m.ModuleID
  5  JOIN Result r ON s.StudentID = r.StudentID
  6  JOIN Assessment a ON r.AssessmentID = a.AssessmentID
  7  JOIN ModuleAssessment ma ON a.AssessmentID = ma.AssessmentID
  8  WHERE m.ModuleName = 'Databases'
  9    AND r.ObtainedMarks >=
 10      (SELECT AVG(r2.ObtainedMarks)
 11      FROM Result r2
 12      JOIN Assessment a2 ON r2.AssessmentID = a2.AssessmentID
 13      JOIN ModuleAssessment ma2 ON a2.AssessmentID = ma2.AssessmentID
 14      JOIN Module m2 ON ma2.ModuleID = m2.ModuleID
 15      WHERE m2.ModuleName = 'Databases');

STUDENTNAME                                        MODULEID                OBTAINEDMARKS
-------------------------------------------------- ----------------------- -------------
Roshani Rajbhandari                                M002                               73

SQL>
```

Figure 75: Transaction Query: 4

7.2.5. Transaction Query: 5

```
SELECT s.StudentName,
        ma.ModuleID,
        SUM(r.ObtainedMarks) AS TotalAggregateMarks,
        CASE
                WHEN SUM(r.ObtainedMarks) >= 40 THEN 'Pass'
                ELSE 'Fail'
        END AS Remarks
FROM Student s
JOIN Result r ON s.StudentID = r.StudentID
JOIN Assessment a ON r.AssessmentID = a.AssessmentID
JOIN ModuleAssessment ma ON a.AssessmentID = ma.AssessmentID
GROUP BY s.StudentName, ma.ModuleID
ORDER BY s.StudentName, ma.ModuleID;
```

```
SQL> SELECT s.StudentName,
  2         ma.ModuleID,
  3         SUM(r.ObtainedMarks) AS TotalAggregateMarks,
  4         CASE
  5             WHEN SUM(r.ObtainedMarks) >= 40 THEN 'Pass'
  6             ELSE 'Fail'
  7         END AS Remarks
  8  FROM Student s
  9  JOIN Result r ON s.StudentID = r.StudentID
 10  JOIN Assessment a ON r.AssessmentID = a.AssessmentID
 11  JOIN ModuleAssessment ma ON a.AssessmentID = ma.AssessmentID
 12  GROUP BY s.StudentName, ma.ModuleID
 13  ORDER BY s.StudentName, ma.ModuleID;

STUDENTNAME          MODULEID   TOTALAGGREGATEMARKS REMARKS
-------------------- ---------- ------------------- ----------
Deepesh Bashyal      M001                       87 Pass
Deepesh Bashyal      M008                       81 Pass
Manish Thapa         M005                       75 Pass
Nischita Acharya     M003                       35 Fail
Roshani Rajbhandari  M002                       73 Pass
Sita Gurung          M006                       37 Fail
Subin Khatiwada      M001                       85 Pass
Subin Khatiwada      M004                       82 Pass

8 rows selected.

SQL>
```

Figure 76: Transaction Query: 5

8. Critical Evaluation

8.1. Critical Evaluation of module, it’s usage and relation with other subjects

The course gave an in-depth knowledge of database design, development, and implementation with an emphasis on normalized database systems and structured relationships which ensure data integrity. It highlighted the theoretical and practical aspects of database systems, including normalization, ERD creation, and SQL query optimization, all of which are essential for any modern data-driven system.

Relational concepts like many-to-many relationships, bridge entities, and resolving transitive dependencies helped me understand how databases relate to other subjects like software engineering and system analysis. For instance, by reducing redundancy and increasing efficiency, ideas like normalization have a direct connection to software design principles. Because databases often serve as the foundation of applications, working with SQL and Oracle databases also links to programming ideas, especially backend development.

I was able to gain a deeper understanding of the relationship between data and user requirements through this module's actual use in developing entities like StudentModule and ModuleAssessment. My problem-solving abilities were enhanced by the systematic approach to resolving real-world challenges, such as introducing resource completion tracking into place and making sure assessments could be shared across several programs through modules. It also highlighted the significance of accuracy and modularity in database systems.

8.2. Critical Assessment of coursework

The coursework required analyzing and creating a comprehensive database system for the E-Classroom platform. Creating links between entities, dealing with ambiguities in data attributes, and implementing normalization up to the third normal form (3NF) were some of the issues I faced during the project. The development of bridge tables like StudentModule and ModuleAssessment was particularly educative, which resolved many-to-many relationships and minimized redundancy.

Ensuring the accuracy of the queries was one of the main hurdles, particularly when handling complex relationships like identifying students who obtained above-average grades in a module or those who missed assessments. Writing queries was not the only difficulty; debugging errors, which often came from improperly established relationships or missing keys were also equally difficult. The understanding of database theory was further strengthened when the issue of ambiguous links—such as the initial absence of direct relationships between students and modules—required redesigning the structure and adding bridge entities.

Furthermore, I was able to identify errors in my initial approach by implementing theoretical ideas like cardinality and modality into practice. For instance, creating foreign keys and composite keys in tables like Result demonstrated how minor mistakes in schema design could lead to huge complications during query execution. Finding out how normalization and relationships directly affect query performance and data integrity was important.

I also learnt the importance of collaboration and iterative development from the module. My tutor's regular feedback was beneficial in enhancing the coursework. Some of the issues, such as organizing the ResourceCompletion table to track resource completion by students or creating the Result table to manage derived attributes like grades were resolved with continuous effort.

To conclude, this course gave me an opportunity to put theoretical ideas into practice, which increased my confidence in database design and query optimization. Even though it was challenging, I was able to create a database system that involved both academic and practical aspects, so I got to have a valuable experience. Without any doubt, this project will provide a strong foundation for future database development and associated projects.

9. Dump File Creation

Exp Bhumika_Karki/23047584 file = BhumikaKarki.dmp

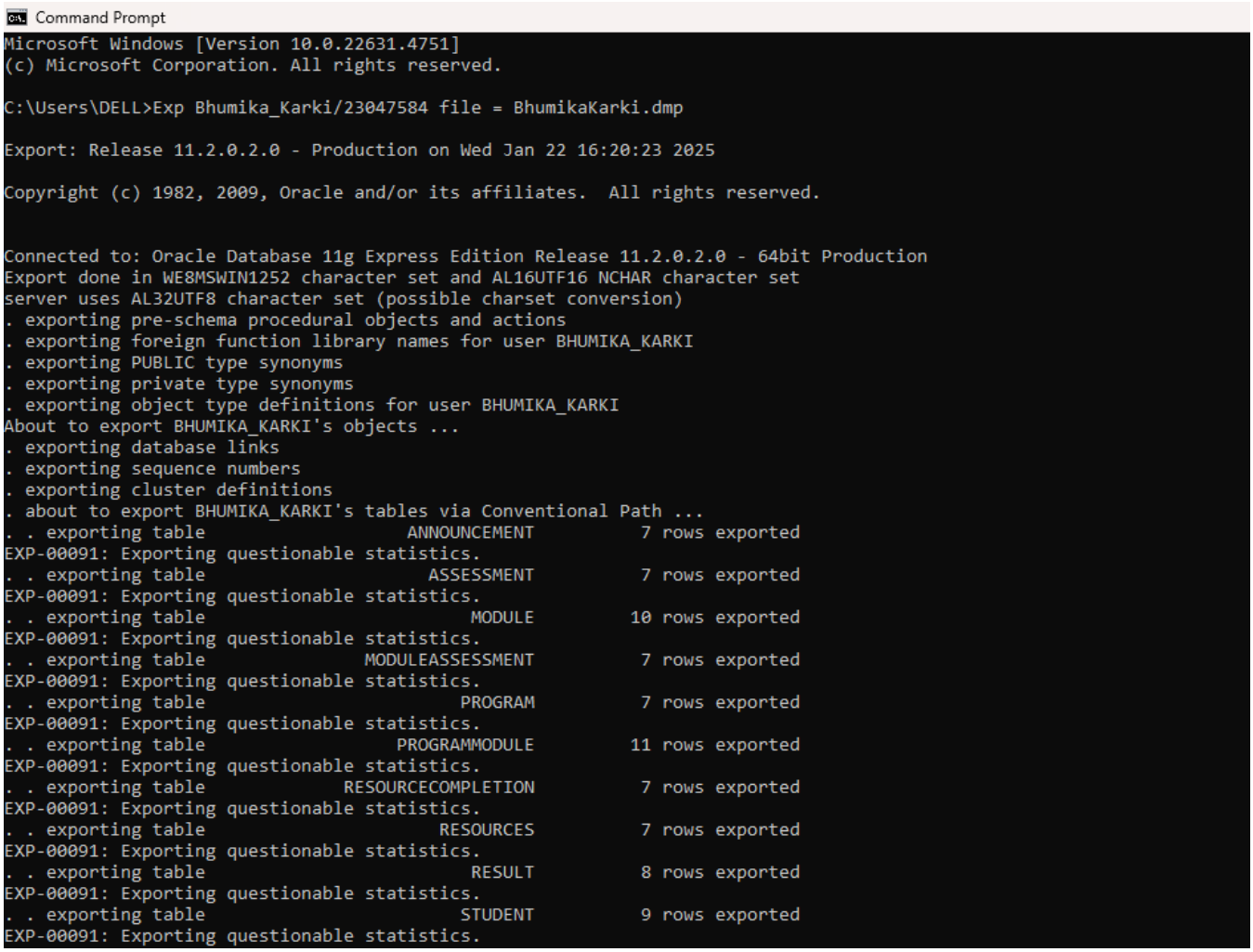

Figure 77: Creation of Dump File

10. DROP Table

10.1. Drop Bridge Tables

Drop tables with composite keys first.

DROP TABLE ResourceCompletion; (Depends on Student and Resources)

DROP TABLE ModuleAssessment; (Depends on Module and Assessment)

DROP TABLE StudentModule; (Depends on Student and Module)

DROP TABLE ProgramModule; (Depends on Program and Module)

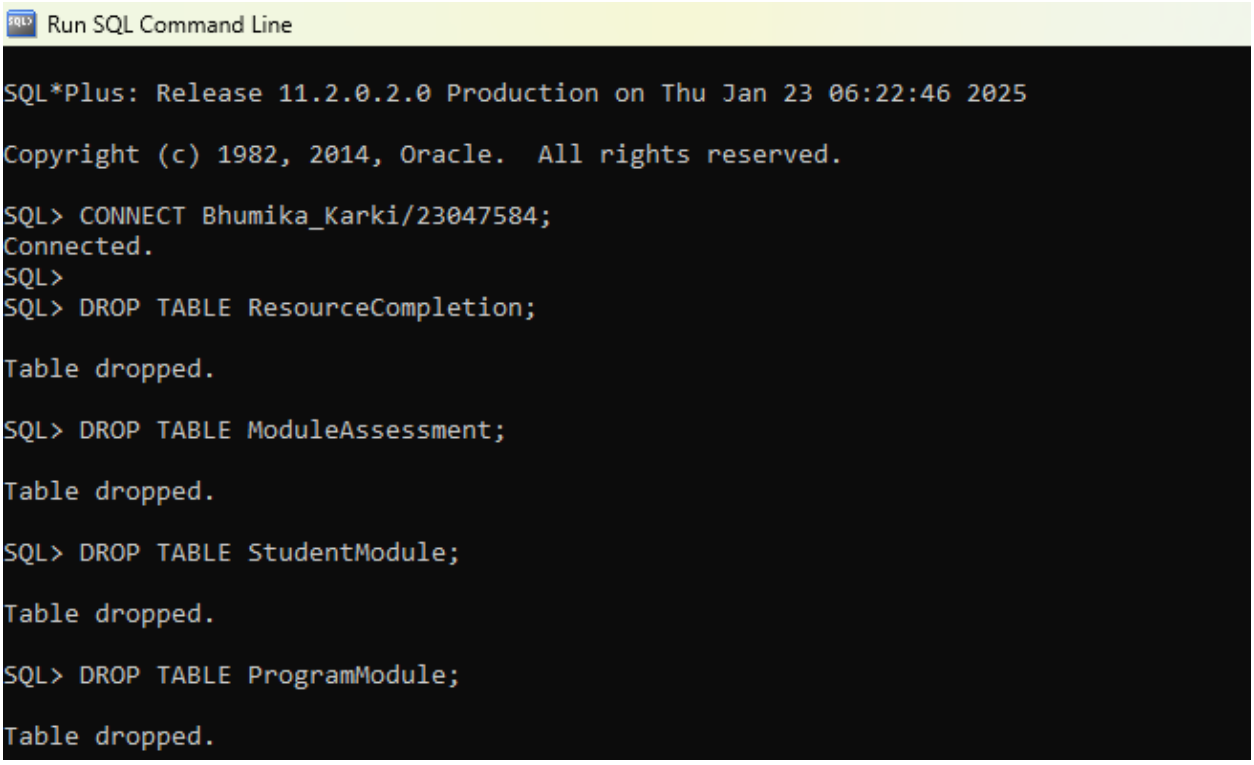

Figure 78: DROP TABLE (1/3)

10.2. DROP TABLE with foreign key dependencies

DROP TABLE Result; (Depends on Student and Assessment)

DROP TABLE Announcement; (Depends on Module and Teacher)

DROP TABLE Resources; (Depends on Module)

DROP TABLE Assessment; (can be dropped after ModuleAssessment)

```
SQL> DROP TABLE Result;

Table dropped.

SQL> DROP TABLE Announcement;

Table dropped.

SQL> DROP TABLE Resources;

Table dropped.

SQL> DROP TABLE Assessment;

Table dropped.
```

Figure 79: DROP TABLE (2/3)

10.3. DROP TABLE with main entities

DROP TABLE Module;

DROP TABLE Student;

DROP TABLE Program;

DROP TABLE Teacher;

```
SQL> DROP TABLE Module;

Table dropped.

SQL> DROP TABLE Student;

Table dropped.

SQL> DROP TABLE Program;

Table dropped.

SQL> DROP TABLE Teacher;

Table dropped.

SQL>
```

Figure 80: DROP TABLE (3/3)

11. Bibliography

geeksforgeeks, 2024. *what-is-data-dictionary.* [Online]
Available at: https://www.geeksforgeeks.org/what-is-data-dictionary/
[Accessed 30 December 2024].

Rajput, H., 2024. *normalization-in-dbms.* [Online]
Available at: https://www.naukri.com/code360/library/normalization-in-dbms
[Accessed 30 December 2024].

Secoda, 2024. *entity-relationship-diagram.* [Online]
Available at: https://www.secoda.co/glossary/entity-relationship-diagram
[Accessed 7 December 2024].

tutorialspoint, 2024. *DBMS - ER Model Basic Concepts.* [Online]
Available at: https://www.tutorialspoint.com/dbms/er_model_basic_concepts.htm
[Accessed 27 December 2024].