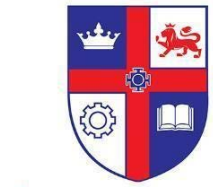
LONDON METROPOLITAN UNIVERSITY



Islington college
(इस्लिङ्टन कलेज)

CC5051NI Databases

100% Individual Coursework

Autumn 2024

Credit: 15 Semester Long Module

Student Name: Chewan Regmi

London Met ID: 23056535

Assignment Submission Date: 23 January 2025

Word Count:

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

ACKNOWLEDGEMENT

I would like to express my sincere gratitude to Mr. Adarsh Shrestha, the module leader of the Databases module, and Mr. Prashant Lal Shrestha, the tutor, for their excellent teaching and remarkable guidance throughout this coursework. Their support and mentorship were crucial in understanding and implementing real-world database concepts. I also want to thank Islington College for providing the opportunities to learn core concepts and gain practical skills through this project coursework. This coursework has been instrumental in learning about database systems.

Table of Contents

TABLE OF TABLES

TABLE OF FIGURES

1. INTRODUCTION

1.1 INTRODUCTION OF BUSINESS AND IT'S FORTE

Islington College is one of the renowned institutions for its quality education since 1996. The college is offering academic excellence along with personal development skills to prepare students for real world business. Islington college's business program focuses on developing deep understanding of fundamental business concepts and modern practices. The IT programs are designed to meet the demand of modern world by providing technical expertise with problem solving skill for the future career of the students in software engineering, networking, cybersecurity and computing. Islington college offers a strong educational foundation in both business and IT by providing academic excellence with practical industry preparation to prepare students for their career. For this, Entrepreneur Ms. Mary has proposed the development of an innovative E-classroom which is an online platform aimed to create an effective digital teaching and learning platform for students and teachers of Islington College.

1.2 OVERVIEW OF CURRENT BUSINESS ACTIVITY AND OPERATION

Islington college is primarily engaged in providing high quality academic programs in IT, Business and Computing for both undergraduate and postgraduate programs in affiliation with London metropolitan university with international standard curriculum. The college provides various learning and growing activities through workshops and seminars and other training sessions like internship opportunities. The institution organizes cultural programs, sport events and other various leadership development workshop to promote the growth or learners.

As a new method of innovative education, Islington college plans to launch the "E-Classroom Platform" which is a digital teaching and learning environment to enhance the academic growth of students and easy way of teaching for teachers. Ms. Mary's requirement for designing a database for the "E-Classroom" includes creating an efficient system to store, manage and retrieve data related to online learning environment. Ms. Mary aims for the platform that is user-friendly and scalable to meet the need of dynamic e-learning environment which manages students, teachers and programs.

The main requirement for the database is user management which includes maintaining the roles of the students, teachers and administrators and store the user details safely, record the course details, assessment details, resource management such as e-books and performance analysis of student by tracking the attendance record, progress and performance in courses.

1.3 BUSINESS RULES

Business rule is a statement that creates some form of restriction on a specific aspect of database in form of policy, procedure or principle and is applied to any organization that stores and uses data to generate information. Business rule is based on the way the organization stores and uses its data to generate the information which determines the manner in which the organization functions or conducts its business.

Student-Program

Each student must be enrolled in only one of the programs

Each program may contain multiple students.

Program-Module

Each program must have multiple modules

Each module may be included in multiple programs.

Module-Assessment

Each module may have multiple assessments

Each assessment must be linked to only one module.

Assessment-Result

Each assessment must have one result and

Each result must be linked to only one assessment.

Module-resource

Each module may contain multiple resources.

Teacher-notice

Each teacher may post multiple notices.

Notice-Module

Each notice must be linked to a single module.

Module-Teacher

Each module may be taught by multiple teachers

Each teacher must teach a single module.

Resource-Module

Each resource must be linked to single module.

Notice-Teacher

Each notice must be posted by a single teacher.

Module-Notice

Each module may contain multiple notices

1.4 ASSUMPTIONS

Module-Teacher

Each module may be taught by multiple teachers

Each teacher must teach a single module.

Resource-Module

Each resource must be linked to single module.

Notice-Teacher

Each notice must be posted by a single teacher.

Module-Notice

Each module may contain multiple notices

2. INITIAL ERD

2.1 IDENTIFICATION OF ENTITIES AND ATTRIBUTES

In an entity relationship diagram/database entity represents real world tangible (exist physically) or intangible (can exist only logically and does not have any physical existence) objects and events in which data is stored in a database. Event often involved interaction between two entity or action that changes status of entity. Each entity is named in singular form its either noun, adjective + noun, noun (String), adjective + noun + noun. In ERD entity is represented by rectangle shape.

Attribute represents the characteristics/properties or fields of an entity in a database. Entity must have unique identifier which can be single or combination of attributes. Candidate key is the attribute or combination of attributes that uniquely identifies each instance of entity. Attributes have values which can be number, character string, date, image, sound and many more. The list of the entities and attributes used in this system are listed below in the tabular form.

1. Student

Table 1 Student (Entity)

S.NO	Attribute name	Data type	Size	Constraint
1	Student_ID	NUMBER	10	Primary Key, Not Null
2	Student_Name	CHARACTER	20	Not Null
3	Student_Contact	CHARACTER	15	Unique, Not Null
4	Student_Address	CHARACTER	50	Not Null
5	Enrolled	DATE		Not Null

2. Program

Table 2 Program (Entity)

S. No	Attribute name	Data type	Size	Constraint
1	Program_Id	NUMBER	10	Primary Key, Not Null
2	Program_Name	CHARACTER	15	Not Null
3	Faculty	CHARACTER	30	Not Null

3. Teacher

Table 3 Teacher (Entity)

S. No	Attribute name	Data type	Size	Constraint
1	Teacher_Id	NUMBER	10	Primary Key, Not Null
2	Teacher_Name	CHARACTER	20	Not Null
3	Teacher_Contact	NUMBER	15	Not Null, Unique
4	Teacher_Address	CHARACTER	10	Not Null

4. Module

Table 4 Module (Entity)

S. No	Attribute name	Data type	Size	Constraint
1	Module_Id	NUMBER	10	Primary Key, Not Null
2	Module_Name	CHARACTER	25	Not Null
3	Credit_Hour	NUMBER	12	Not Null

5. Resource

Table 5 Resource (Entity)

S. No	Attribute name	Data type	Size	Constraint
1	Resource_Id	NUMBER	10	Primary Key, Not Null
2	Resource_Type	CHARACTER	10	Not Null
3	Duration	CHARACTER	10	Not Null

6. Assessment

Table 6 Assessment (Entity)

S. No	Attribute name	Data type	Size	Constraint
1	Assessment_Id	NUMBER	15	Primary Key, Not Null
2	Assessment_Title	CHARACTER	10	Not Null
3	Weightage	CHARACTER	10	Not Null
4	Status	CHARACTER	18	Not Null
5	Release	DATE		Not Null
6	Deadline	DATE		Not Null

7. Notice

Table 7 Notice (Entity)

S. No	Attribute name	Data type	Size	Constraint
1	Notice_id	NUMBER	10	Primary Key, Not Null
2	Notice_Title	CHARACTER	20	Not Null
3	Publish_on	DATE		Not Null

8. Result

Table 8 Result (Entity)

S. No	Attribute name	Data type	Size	Constraint
1	Result_Id	NUMBER	10	Primary Key, Not Null
2	result_status	CHARACTER	15	Not Null
3	Total_Marks_Obtained	NUMBER	3	Not Null

2.2 ERD

Entity relationship diagram shows the relationship, cardinality and modality between the entities of the system. For this system crow foot's notation method is used to draw the ERD.

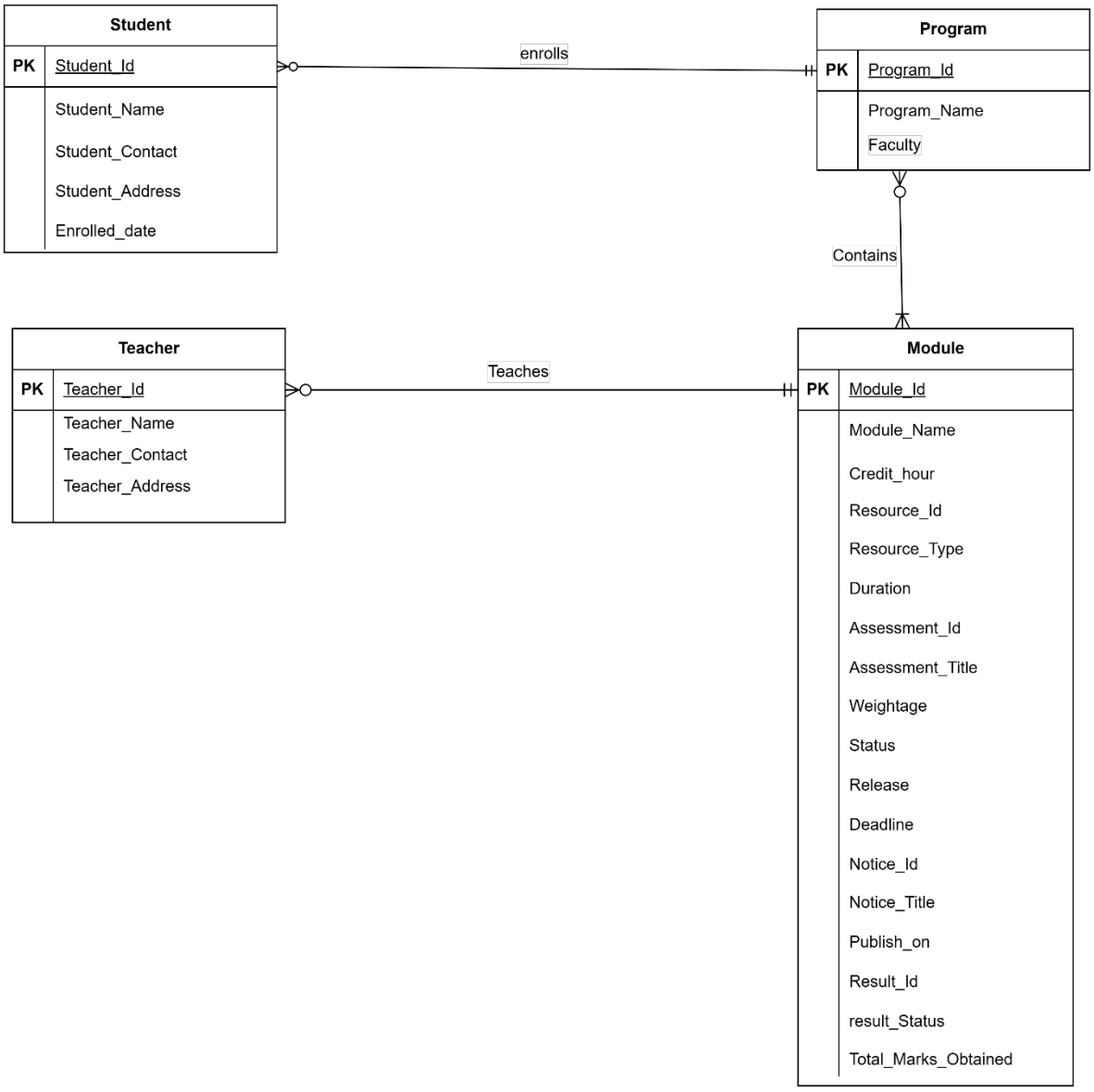


Figure 1 Initial ERD

3. NORMALIZATION

Normalization is the process of structuring data within a database to ensure it is well-organized, consistent, and free of redundancy. It involves defining entities or relations and establishing connections between tables based on specific rules aimed to protect data integrity and improving database efficiency. By simplifying complex data into a clear format that represents distinct entities, their attributes, and their relationships, normalization minimizes unnecessary data duplication. It begins with a predefined set of attributes and systematically groups or regroups them to avoid inconsistencies and anomalies (Kidd, 2024).

Steps in Normalization:

- **Step 1 Un-Normalized form (UNF)**

All attributes with repeating groups included.

- **Step 2 First normal form (1NF)**

Repeating groups should be separated to new table/relation so that there is a single value at the intersection of each row and column of the table/relation.

- **Step 3 Second normal form (2NF)**

Any **partial functional dependencies** (Attributes that are completely dependent on only one part of composite key) have been separated to a new relation.

- **Step 4 Third normal form(3NF)**

Any transitive dependencies have been separated to a new relation.

If relation meets the criteria for 3NF its also meets the criteria for 1NF and 2NF so most design problems can be avoided if the relation is in 3NF.

Functional dependencies occur when a key attribute gives non key attribute.

Full functional dependency occurs when composite key together gives non key attribute.

Transitive dependency occurs when a unique identifier in a relation gives another unique identifier which also gives other attributes independently.

UNF TO 3NF WITH VALID PROCESS DESCRIPTION

Un-Normalized Normal form (UNF)

A relation is considered unnormalized when it has repeating groups and various anomalies. Repeating group refers to an attribute or a group of attributes in an unnormalized table that can have multiple values associated with a single instance/occurrence of the nominated key attribute (Primary Key) of the table.

Step 1: List all the attributes from the E-classroom platform to create unnormalized form/structure in a single relation within a () and **name** the relation.

Step 2: Choose a unique identifier for the entity.

Step 3: Show repeating group within {} in this relation.

Student (Student_ID, Student_Name, Student_Contact, Student_Address, Enrolled, Program_Id, Program_Name, Faculty, {Module_Id, Module_Name, Credit_Hour, {Resource_Id, Resource_Type, Duration}, {Teacher_Id, Teacher_Name, Teacher_Contact, Teacher_Address, {Notice_id, Notice_Title, Release_date}}, {Assessment_Id, Assessment_Title, Weightage, Status, Release, Deadline, Result_Id, result_status, Total_Marks_Obtained}})

In this relation **Student_ID** is the **unique identifier** of the relation. This relation consists of all the attributes of the system and the **relation student** consists of **multiple repeating groups** and **repeating group under the repeating group.**

First Normal Form (1NF)

First normal form (1NF) restriction is built into relational model. Advantage of 1NF are simplicity and uniform access.

Step 1: Repeating group should be removed to **separate relation** and **name that relation**.

Step 2: Carry forward the **key attribute** from the UNF relation to the new relation.

Step 3: Assign a **key attribute** from attributes to the new relation.

In the above UNF, module is the repeating group which contains repeating group resource, assessment and teacher which contain repeating group notice.

Student (**Student_ID**, Student_Name, Student_Contact, Student_Address, Enrolled, Program_Id, Program_Name, Faculty)

Relation without repeating group is already in 1NF as in Student relation

Student-Module (**Student_ID**, **Module_Id**, Module_Name, Credit_Hour)

As **module** is repeating group it is separated from the student and **student id** is carry forwarded to module relation and **module id** along with student id is assigned as the **key attribute** of the relation.

Student-Module-Resource (**Student_ID**, **Module_Id**, **Resource_Id**, Resource_Type, Duration)

As **resource** is repeating group in module it is separated from the student-module relation and **student id** and **module id** are carry forwarded to resource and **resource id** along with student id and module id is assigned as the **key attribute** of the relation.

Student-Module-Teacher (**Student_ID**, **Module_Id**, **Teacher_Id**, Teacher_Name, Teacher_Contact, Teacher_Address)

As **Teacher** is repeating group in module it is separated from the student-module relation and **student id** and **module id** are carry forwarded to teacher and **teacher id** along with student id and module id is assigned as the **key attribute** of the relation.

Student-Module-Teacher-Notice (**Student_ID**, **Module_Id**, **Teacher_Id**, **Notice_id**, Notice_Title, Publish_on)

As notice is the repeating group in student-module-teacher relation so **student id, module id** and **teacher id** are carry forwarded to notice and **notice id** along with student id, module id and teacher is id assigned as key attribute of the relation.

Student-Module-Assessment (**Student_ID**, **Module_Id**, **Assessment_Id**, Assessment_Title, Weightage, Status, Release, Deadline, Result_Id, result_status, Total_Marks_Obtained)

Repeating group in module, **assessment** is separated from the student-module relation and **student id** and **module id** is carry forward to resource and **assessment id** along with student id and module id is assigned as the **key attribute** of the relation.

Second Normal Form (2NF)

Step 1: Check for functional dependencies (Full Functional + Partial) in any relation from 1NF with **composite unique identifiers** / primary key. Identifying partial functional dependencies.

Step 2: Relation without composite key is already in second normal form 2NF.

Step 3: Identify and Separate Partial Functional Dependency (Attributes that are completely dependent on only one part of composite key) by listing all combination of Composite Determinant (Primary Key) and part of composite Determinant (Primary key) and how are non-key attributes dependent on determinants to a new relation and name that relation.

Step 4: make each determinant Primary Key of new relation.

Any partial functional dependencies have been removed from the relations of the first normal form keep the old relation of 1NF as it is in 2NF with only composite keys even if the composite key does not determine any non-key attribute while checking the functional dependency. Relation in 1NF is already in 2NF if no non key attributes exist in relation and every non key attribute is functionally dependent on full set of primary key attributes.

Applying 2NF

Relation in 1NF is already in 2NF if the primary key consists of only one attribute as in Student.

Student (**Student_ID**, Student_Name, Student_Contact, Student_Address, Enrolled, Program_Id, Program_Name, Faculty)

Checking dependencies

Student-Module (**Student_ID, Module_Id,** Module_Name, Credit_Hour)

Student_ID, Module_Id → ×

As the composite identifiers student id and module id does not show full functional dependency with any of the attributes so we keep this relation as it is.

Student-Module (**Student_ID, Module_Id**)

Student_ID → ×

Student id does not give any attribute so it is discarded as there is no partial dependency with this determinant.

Module_Id → Module_Name, Credit_Hour

This relation shows partial dependency as the non-key attributes depends on just a part of composite primary key which is module id. So, this is separated to a new relation and named as module.

Module (**Module_Id,** Module_Name, Credit_Hour)

Student-Module-Resource (**Student_ID, Module_Id, Resource_Id,** Resource_Type, Duration)

Student_ID, Module_Id, Resource_Id → ×

As the composite identifiers' student id, module id and resource id does not show full functional dependency with any of the attributes so we keep this relation as it is.

Student-Module-Resource (**Student_ID, Module_Id, Resource_Id**)

Module_Id, Resource_Id → ×

Module id and resource id does not give any attribute so it is discarded as there is no partial dependency with these determinants.

Student_ID, Resource_Id → ×

Student id and resource id does not give any attribute so it is discarded as there is no partial dependency with these determinants.

Student_ID, Module_Id → ×

Student id and module id does not give any attribute so it is discarded as there is no partial dependency with these determinants.

Resource_Id → Resource_Type, Duration

This relation shows partial dependency as the non-key attributes depends on just a part of composite primary key which is resource id. So, this is separated to a new relation named as resource.

Resource (**Resource_Id**, Resource_Type, Duration)

Module_Id → ×

Module id does not give any attribute so it is discarded as there is no partial dependency with this determinant.

Student_ID → ×

Student id does not give any attribute so it is discarded as there is no partial dependency with this determinant.

Student-Module-Teacher (**Student_ID, Module_Id, Teacher_Id**, Teacher_Name, Teacher_Contact, Teacher_Address)

Student_ID, Module_Id, Teacher_Id → ×

As the composite identifiers' student id, module id and teacher id does not show full functional dependency with any attributes so we keep this relation as it is.

Student-Module-Teacher (Student_ID, Module_Id, Teacher_Id)

Student_ID, Module_Id → ×

Student id and module id does not give any attribute so it is discarded as there is no partial dependency with these determinants.

Student_ID, Teacher_Id → ×

Student id and teacher id does not give any attribute so it is discarded as there is no partial dependency with these determinants.

Module_Id, Teacher_Id → ×

Module id and teacher id does not give any attribute so it is discarded as there is no partial dependency with these determinants.

Student_ID → ×

Student id does not give any attribute so it is discarded as there is no partial dependency with this determinant.

Module_Id → ×

Module id does not give any attribute so it is discarded as there is no partial dependency with this determinant.

Teacher_Id → Teacher_Name, Teacher_Contact, Teacher_Address

This is partial dependency as the non-key attributes depends on just a part of composite primary key which is teacher id. So, this is separated to a new relation named as teacher.

Teacher (**Teacher_Id**, Teacher_Name, Teacher_Contact, Teacher_Address)

Student-Module-Teacher-Notice (**Student_ID, Module_Id, Teacher_Id, Notice_id**, Notice_Title, Publish_on)

Student_ID, Module_Id, Teacher_Id, Notice_id → ×

As the composite identifiers' student id, module id teacher id and notice id does not show full functional dependency with any attributes so we keep this relation as it is.

Student-Module-Teacher-Notice (**Student_ID, Module_Id, Teacher_Id, Notice_id**)

Student_ID, Module_Id, Teacher_Id→ ×

Student id, module id and teacher id does not give any attribute so it is discarded as there is no partial dependency with these determinants.

Student_ID, Module_Id, Notice_id→ ×

Student id, module id and notice id does not give any attribute so it is discarded as there is no partial dependency with these determinants.

Student_ID, Teacher_Id, Notice_id→ ×

Student id, notice id and teacher id does not give any attribute so it is discarded as there is no partial dependency with these determinants.

Module_Id, Teacher_Id, Notice_id→ ×

Module id, teacher id and notice id does not give any attribute so it is discarded as there is no partial dependency with these determinants.

Student_ID, Module_Id→ ×

Student id and module id does not give any attribute so it is discarded as there is no partial dependency with these determinants.

Student_ID, Teacher_Id→ ×

Student id and teacher id does not give any attribute so it is discarded as there is no partial dependency with these determinants.

Student_ID, Notice_id→ ×

Student id and notice id does not give any attribute so it is discarded as there is no partial dependency with these determinants

Teacher_Id, Notice_id_→ ×

Teacher id and notice id does not give any attribute so it is discarded as there is no partial dependency with these determinants.

Module_Id, Teacher_Id → ×

Module id and teacher id does not give any attribute so it is discarded as there is no partial dependency with these determinants.

Module_Id, Notice_id→ ×

Module id and notice id does not give any attribute so it is discarded as there is no partial dependency with these determinants.

Student_ID→ ×

Student id does not give any attribute so it is discarded as there is no partial dependency with this determinant.

Module_Id→ ×

Module id does not give any attribute so it is discarded as there is no partial dependency with this determinant.

Teacher_Id → ×

Teacher id does not give any attribute so it is discarded as there is no partial dependency with this determinant.

Notice_id → Notice_Title, Publish_on

This is partial dependency as the non-key attributes depends on just a part of composite primary key which is notice id. So, this is separated to a new relation named as notice.

Notice (**Notice_id**, Notice_Title, Publish_on)

Student-Module-Assessment (**Student_ID, Module_Id, Assessment_Id**, Assessment_Title, Weightage, Status, Release, Deadline, Result_Id, result_status, Total_Marks_Obtained)

Student_ID, Module_Id, Assessment_Id→ Result_Id, result_status, Total_Marks_Obtained

This relation shows fully functional dependencies among the non-key attributes on composite primary keys. So, it is separated to a new relation named as student-module-assessment.

Student-Module-Assessment (**Student_ID, Module_Id, Assessment_Id,** Result_Id, result_status, Total_Marks_Obtained)

Student_ID, Module_Id →×

Student id and module id does not give any attribute so it is discarded as there is no partial dependency with this determinant.

Student_ID, Assessment_Id→ ×

Student it and assessment id does not give any attribute so it is discarded as there is no partial dependency with this determinant.

Module_Id, Assessment_Id → ×

Module id and assessment id does not give any attribute so it is discarded as there is no partial dependency with these determinants.

Student_ID → ×

Student id does not give any attribute so it is discarded as there is no partial dependency with this determinant.

Module_Id → ×

Module id does not give any attribute so it is discarded as there is no partial dependency with this determinant.

Assessment_Id→ Assessment_Title, Weightage, Status, Release, Deadline

This is partial dependency as the non-key attributes depends on just a part of composite primary key which is notice id. So, this is separated to a new relation named as assessment.

Assessment (**Assessment_Id**, Assessment_Title, Weightage, Status, Deadline, Release)

2NF relations

Student (**Student_ID**, Student_Name, Student_Contact, Student_Address, Enrolled, Program_Id, Program_Name, Faculty)

Student-Module (**Student_ID, Module_Id**)

Module (**Module_Id**, Module_Name, Credit_Hour)

Student-Module-Resource (**Student_ID, Module_Id, Resource_Id**)

Resources (**Resource_Id**, Resource_Type, Duration)

Student-Module-Teacher (**Student_ID, Module_Id, Teacher_Id**)

Teacher (**Teacher_Id**, Teacher_Name, Teacher_Contact, Teacher_Address)

Student-Module-Teacher-Notice (**Student_ID, Module_Id, Teacher_Id, Notice_id**)

Notice (**Notice_id**, Notice_Title, Publish_on)

Student-Module-Assessment (**Student_ID, Module_Id, Assessment_Id**, Result_Id, result_status, Total_Marks_Obtained)

Assessment (**Assessment_Id**, Assessment_Title, Weightage, Status, Release, Deadline)

Third normalized form (3NF)

Relation with only one non key attribute is already in third normal form.

Step 1 Check for transitive dependencies in 2NF check for the relation that have more than one non key attribute.

Step 2: identify and separate transitive functional dependency as per 3NF rule to a new relation.

Any transitive functional dependencies have been removed from the relations of second normal form.

Student (**Student_ID**, Student_Name, Student_Contact, Student_Address, Enrolled, Program_Id, Program_Name, Faculty)

Student_ID→ Student_Name, Student_Contact, Student_Address, Enrolled, Program_Id

Program_Id → Program_Name, Faculty

In relation student id gives student name, contact, address, enrolled date, program id and program id give program name, type duration and faculty. It shows transitive dependency so the program table is separated from student table.

Student (**Student_ID**, **Program_Id,** Student_Name, Student_Contact, Student_Address, Enrolled)

Program (**Program_Id**, Program_Name, Faculty)

Student-Module-Assessment (**Student_ID, Module_Id, Assessment_Id**, Result_Id, result_status, Total_Marks_Obtained)

Student_ID, Module_Id, Assessment_Id → Result_Id

Result_Id → result_status, Total_Marks_Obtained

In this relation student id, module id, assessment id gives result id and result id gives result status total marks obtained. It shows transitive dependency so the result table is separated.

Student-Module-Assessment (**Student_ID, Module_Id, Assessment_Id, Result_Id**)

Result (**Result_Id**, result_status, Total_Marks_Obtained)

Relation with only composite key identifier without any attribute are already in 3NF

Student-Module (**Student_ID, Module_Id)**

Student-Module-Resources (**Student_ID, Module_Id, Resource_Id)**

Student-Module-Teacher (**Student_ID, Module_Id, Teacher_Id**)

Student-Module-Teacher-Notice (**Student_ID, Module_Id, Teacher_Id, Notice_id**)

Relation with only one unique identifier is in 3NF

Module (**Module_Id**, Module_Name, Credit_Hour)

Resources (**Resource_Id**, Resource_Type, Duration)

Teacher (**Teacher_Id**, Teacher_Name, Teacher_Contact, Teacher_Address)

Notice (**Notice_id**, Notice_Title, Publish_on)

Assessment (**Assessment_Id**, Assessment_Title, Weightage, Status, Release, Deadline)

3.1 SHOW FULLY NORMALIZED TABLES

Table 9 Normalized table

UNF	1NF	2NF	3NF
Student (**Student_ID**, Student_Name, Student_Contact, Student_Address, Enrolled, Program_Id, Program_Name, Duration, Faculty, {Module_Id, Module_Name, Credit_Hour, {Resource_Id, Resource_Type, Duration}, {Teacher_Id, Teacher_Name, Teacher_Contact, Teacher_Address, {Notice_id, Notice_Title, Publish_on }, {Assessment_Id, Assessment_Title, Weightage, Status, Release, Deadline, Result_Id, result_status, Total_Marks_Obtained}})	**Student** (**Student_ID**, Student_Name, Student_Contact, Student_Address, Enrolled, Program_Id, Program_Name, Duration, Faculty) **Student-Module** (**Student_ID**, **Module_Id**, Module_Name, Credit_Hour) **Student-Module-Resource** (**Student_ID**, **Module_Id**, **Resource_Id**, Resource_Type, Duration) **Student-Module-Teacher** (**Student_ID**, **Module_Id**, **Teacher_Id**, Teacher_Name, Teacher_Contact, Teacher_Address)	**Student** (**Student_ID**, Student_Name, Student_Contact, Student_Address, Enrolled, Program_Id, Program_Name, Duration, Faculty) **Student-Module** (**Student_ID, Module_Id**) **Module** (**Module_Id**, Module_Name, Credit_Hour) **Student-Module-Resource** (**Student_ID, Module_Id, Resource_Id**) **Resource** (**Resource_Id**, Resource_Type, Duration) **Student-Module-Teacher** (**Student_ID, Module_Id, Teacher_Id**)	**Student** (**Student_ID**, **Program_Id,** Student_Name, Student_Contact, Student_Address, Enrolled) **Program** (**Program_Id**, Program_Name, Faculty) **Student-Module-Assessment** (**Student_ID, Module_Id, Assessment_Id, Result_Id**) **Result** (**Result_Id**, result_status, Total_Marks_Obtained) **Student-Module** (**Student_ID, Module_Id**) **Student-Module-Resources** (**Student_ID, Module_Id, Resource_Id**)

	Student-Module-Teacher-Notice (**Student_ID**, **Module_Id**, **Teacher_Id**, **Notice_id**, Notice_Title, Publish_on) **Student-Module-Assessment** (**Student_ID**, **Module_Id**, **Assessment_Id**, Assessment_Title, Weightage, Status, Release, Deadline, Result_Id, result_status, Total_Marks_Obtained)	**Teacher** (**Teacher_Id**, Teacher_Name, Teacher_Contact, Teacher_Address) **Student-Module-Teacher-Notice** (**Student_ID, Module_Id, Teacher_Id, Notice_id**) **Notice** (**Notice_id**, Notice_Title, Publish_on) **Student-Module-Assessment** (**Student_ID, Module_Id, Assessment_Id**, Result_Id, result_status, Total_Marks_Obtained) **Assessment** (**Assessment_Id**, Assessment_Title, Weightage, Status, Release, Deadline)	**Student-Module-Teacher** (**Student_ID, Module_Id, Teacher_Id**) **Student-Module-Teacher-Notice** (**Student_ID, Module_Id, Teacher_Id, Notice_id**) **Module** (**Module_Id**, Module_Name, Credit_Hour) **Resources** (**Resource_Id**, Resource_Type, Duration) **Teacher** (**Teacher_Id**, Teacher_Name, Teacher_Contact, Teacher_Address) **Notice** (**Notice_id**, Notice_Title, Publish_on) **Assessment** (**Assessment_Id**, Assessment_Title, Weightage, Status, Release, Deadline)

4. DATA DICTIONARY AND FINAL ERD

4.1 DATA DICTIONARY

A data dictionary is a document that contains the details of the tables, fields, and other components of a database. It serves as a storage for the database's metadata, which provides essential information to help users understand and work with the data efficiently. A data dictionary describes and explains the elements that form the database (Metabase, n.d.).

Student

Table 10 Data Dictionary for student table

S. No	Attribute	Datatype	Size	Constraint
1	Student_ID	NUMBER	10	Primary Key, Not Null
2	Program_Id	NUMBER	10	Foreign key
3	Student_Name	CHARACTER	20	Not Null
4	Student_Contact	CHARACTER	15	Unique, Not Null
5	Student_Address	CHARACTER	20	Not Null
6	Enrolled	DATE		Not Null

Program

Table 11 Data Dictionary for program table

S. No	Attribute	Datatype	Size	Constraint
1	Program_Id	NUMBER	10	Primary Key, Not Null
2	Program_Name	CHARACTER	15	Not Null
3	Faculty	CHARACTER	30	Not Null

Result

Table 12 Data Dictionary for Result table

S. No	Attribute	Datatype	Size	Constraint
1	Result_Id	NUMBER	7	Primary Key, Not Null
2	result_status	CHARACTER	7	Not Null
3	Total_Marks_Obtained	NUMBER	3	Not Null

Module

Table 13 Data Dictionary for Module table

S. No	Attribute	Datatype	Size	Constraint
1	Module_Id	NUMBER	10	Primary Key, Not Null
2	Module_Name	CHARACTER	25	Not Null
3	Credit_Hour	NUMBER	12	Not Null

Student-Module

Table 14 Data Dictionary for student-module table

S. No	Attribute	Datatype	Size	Constraint	Composite constraint
1	Student_ID	NUMBER	10	Foreign Key, Not Null	Composite primary key
2	Module_Id	NUMBER	10	Foreign Key, Not Null	

Student-Module-Resources

Table 15 Data Dictionary for student-module-resource table

S. No	Attribute	Datatype	Size	Constraint	Composite constraint
1	Student_ID	NUMBER	10	Foreign Key, Not Null	Composite primary key
2	Module_Id	NUMBER	10	Foreign Key, Not Null	
3	Resource_Id	NUMBER	10	Foreign Key, Not Null	

Student-Module-Assessment

Table 16 Data Dictionary for student-module-assessment table

S. No	Attribute	Datatype	Size	Constraint	Composite constraint
1	Student_ID	NUMBER	10	Foreign Key, Not Null	Composite primary key
2	Module_Id	NUMBER	10	Foreign Key, Not Null	
3	Assessment_Id	NUMBER	10	Foreign Key, Not Null	
4	Result_Id	NUMBER	10	Foreign key, Not Null	

Student-Module-Teacher

Table 17 Data Dictionary for student-module-teacher table

S. No	Attribute	Datatype	Size	Constraint	Composite constraint
1	Student_ID	NUMBER	10	Foreign Key, Not Null	Composite primary key
2	Module_Id	NUMBER	10	Foreign Key, Not Null	
3	Teacher_Id	NUMBER	10	Foreign Key, Not Null	

Student-Module-Teacher-Notice

Table 18 Data Dictionary for student-module-teacher-notice table

S. No	Attribute	Datatype	Size	Constraint	Composite constraint
1	Student_ID	NUMBER	10	Foreign Key, Not Null	Composite primary key
2	Module_Id	NUMBER	10	Foreign Key, Not Null	
3	Teacher_Id	NUMBER	10	Foreign Key, Not Null	
4	Notice_id	NUMBER	10	Foreign Key, Not Null	

Resources

Table 19 Data Dictionary for resource table

S. No	Attribute	Datatype	Size	Constraint
1	Resource_Id	NUMBER	12	Primary Key, Not Null
2	Resource_Type	CHARACTER	20	Not Null
3	Duration	CHARACTER	10	Not Null

Teacher

Table 20 Data Dictionary for teacher table

S. No	Attribute	Datatype	Size	Constraint
1	Teacher_Id	NUMBER	10	Primary Key, Not Null
2	Teacher_Name	CHARACTER	20	Not Null
3	Teacher_Contact	CHARACTER	15	Not Null
4	Teacher_Address	CHARACTER	20	Not Null

Notice

Table 21 Data Dictionary for notice table

S. No	Attribute	Datatype	Size	Constraint
1	Notice_id	NUMBER	10	Primary Key, Not Null
2	Notice_Title	CHARACTER	20	Not Null
3.	Publish_on	DATE		Not Null

Assessment

Table 22 Data Dictionary for assessment table

S. No	Attribute	Datatype	Size	Constraint
1	Assessment_Id	NUMBER	10	Primary Key, Not Null
2	Assessment_Title	CHARACTER	20	Not Null
3	Weightage	CHARACTER	10	Not Null
4	Status	CHARACTER	10	Not Null
5	Release_	DATE		Not Null
6	Deadline	DATE		Not Null

4.2 FINAL ERD

An Entity-Relationship Diagram (ERD) is a graphical representation that illustrates how different entities within a database are interconnected. ERDs are a specialized type of flowchart designed to show the relationships between various entities in a system. An ERD visually represents a system by outlining the connections between people, objects, places, concepts, or events. It clarifies dependencies and provides a structured overview of a database using a diagram. ERDs are essential for modelling business processes based on the system's current functionality or future requirements (Stryker, 2024).

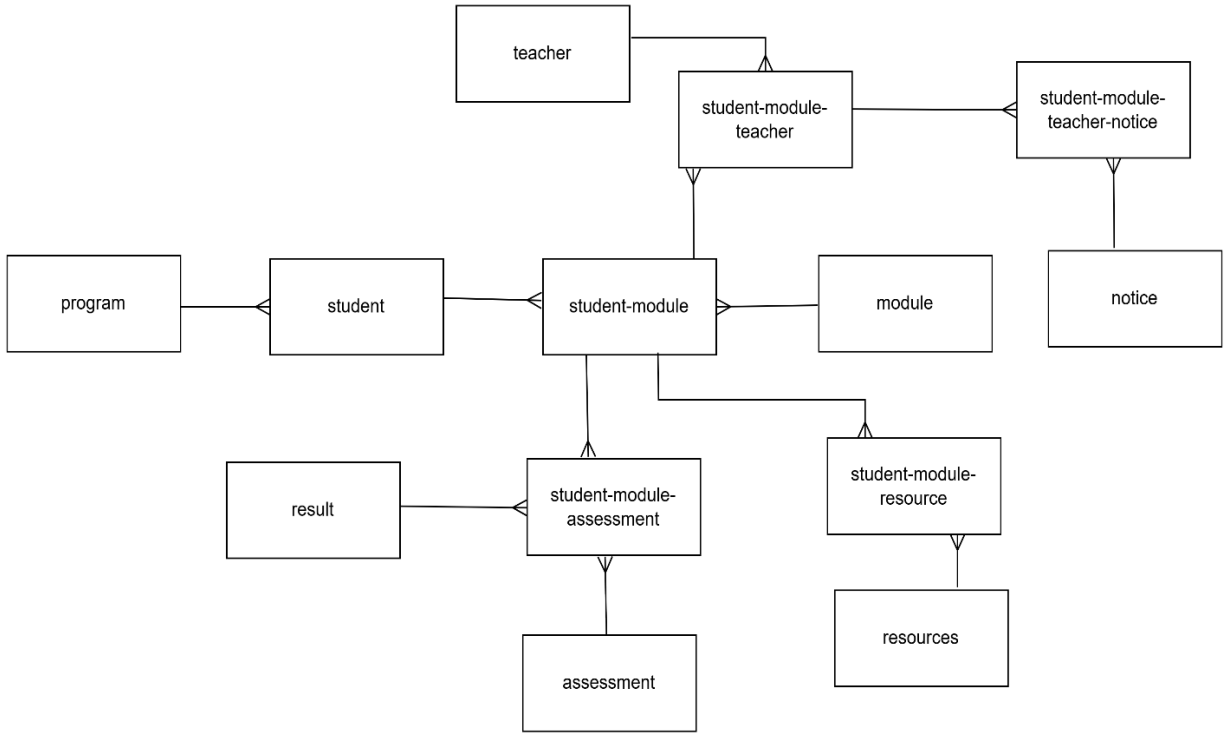


Figure 2 Final ERD

5. IMPLEMENTATION

5.1 CREATE USER

To create a new user, we use ‘CREATE USER’ command of Oracle SQL. After creating user, we need to provide connect and resources to the new user created.

The CREATE USER chewan IDENTIFIED BY 23056535 command creates a new database user named chewan with the password 23056535. The GRANT CONNECT, RESOURCE TO chewan command grants the CONNECT and RESOURCE roles to the chewan user, allowing user to connect to the database and create resources such as tables and indexes. The CONNECT command then logs into the database as the user. Finally, the SET LINESIZE 100; command configures the SQL*Plus environment to set the line width to 100 characters,

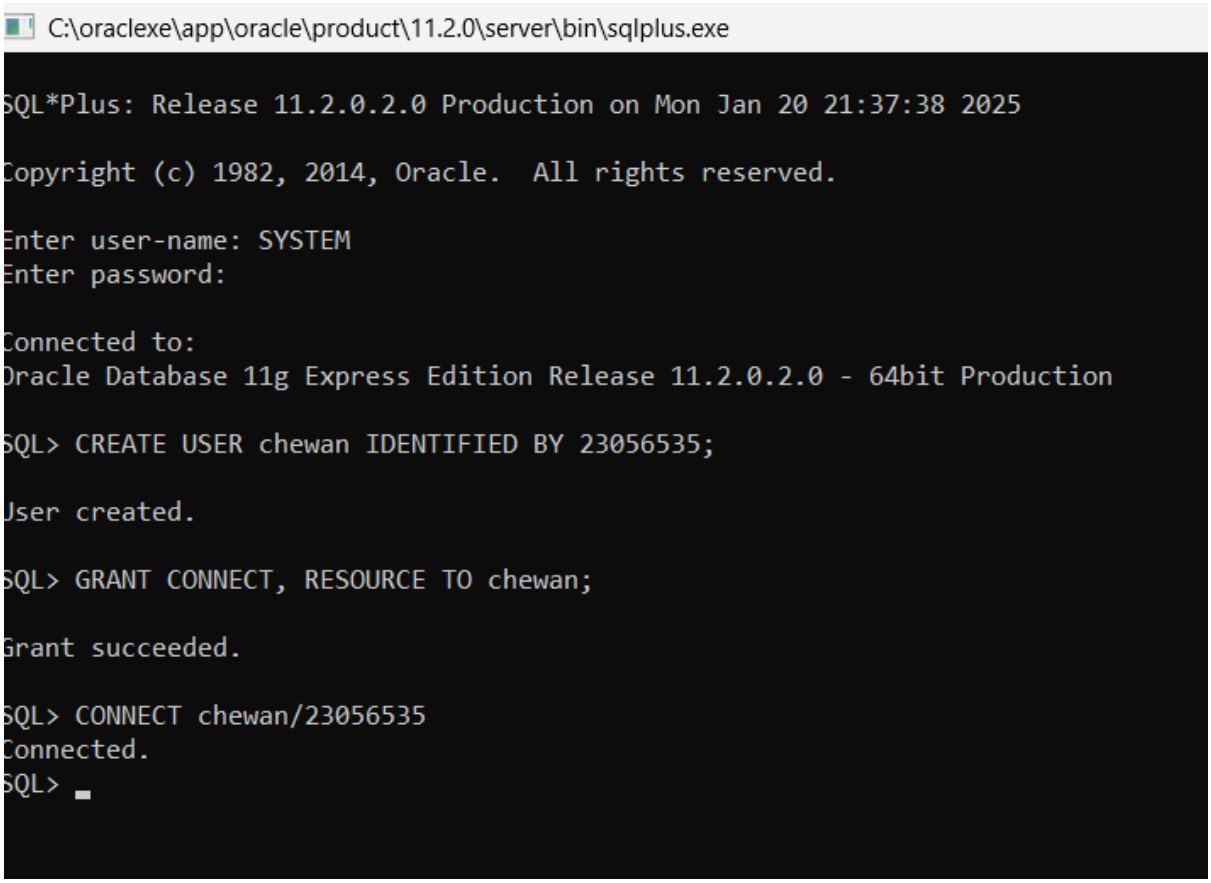


Figure 3 Creating user

5.2 CREATE TABLES

To create table, we use “CREATE TABLE” command.

The PROGRAM table stores information about academic programs. It includes columns for program identifiers, names, and associated faculties.

```
SQL> CREATE TABLE Program (
  2      Program_Id NUMBER(10) PRIMARY KEY NOT NULL,
  3      Program_Name CHARACTER(15) NOT NULL,
  4      Faculty CHARACTER(30) NOT NULL
  5  );

Table created.

SQL> DESC Program;
 Name                                                  Null?    Type
 ----------------------------------------------------- -------- ------------
 PROGRAM_ID                                            NOT NULL NUMBER(10)
 PROGRAM_NAME                                          NOT NULL CHAR(15)
 FACULTY                                               NOT NULL CHAR(30)
```

Figure 4 Create table program

Create table Module

The MODULE table contains information about different modules or courses. It includes columns for module identifiers, names, and credit hours.

```
SQL> CREATE TABLE Module (
  2      Module_Id NUMBER(10) PRIMARY KEY NOT NULL,
  3      Module_Name CHARACTER(25) NOT NULL,
  4      Credit_Hour NUMBER(12) NOT NULL
  5  );

Table created.

SQL> DESC Module;
 Name                                                  Null?    Type
 ----------------------------------------------------- -------- ------------------
 MODULE_ID                                             NOT NULL NUMBER(10)
 MODULE_NAME                                           NOT NULL CHAR(25)
 CREDIT_HOUR                                           NOT NULL NUMBER(12)
```

Figure 5 Create table module

Create table teacher

The TEACHER table holds information about the teachers. It includes columns for teacher identifiers, names, contact information, and addresses.

```
SQL> CREATE TABLE Teacher (
  2      Teacher_Id NUMBER(10) PRIMARY KEY NOT NULL,
  3      Teacher_Name CHARACTER(20) NOT NULL,
  4      Teacher_Contact CHARACTER(15) NOT NULL,
  5      Teacher_Address CHARACTER(20) NOT NULL
  6  );

Table created.

SQL> DESC Teacher;
 Name                                                  Null?    Type
 ----------------------------------------------------- -------- ------------------
 TEACHER_ID                                            NOT NULL NUMBER(10)
 TEACHER_NAME                                          NOT NULL CHAR(20)
 TEACHER_CONTACT                                       NOT NULL CHAR(15)
 TEACHER_ADDRESS                                       NOT NULL CHAR(20)
```

Figure 6 Create table teacher

Create table resources:

The RESOURCES table contains details about various resources available for modules. It includes columns for resource identifiers, types, and durations.

```
SQL> CREATE TABLE Resources (
  2      Resource_Id NUMBER(12) PRIMARY KEY NOT NULL,
  3      Resource_Type CHARACTER(20) NOT NULL,
  4      Duration CHARACTER(10) NOT NULL
  5  );

Table created.

SQL> DESC Resources;
 Name                                                  Null?    Type
 ----------------------------------------------------- -------- ------------------
 RESOURCE_ID                                           NOT NULL NUMBER(12)
 RESOURCE_TYPE                                         NOT NULL CHAR(20)
 DURATION                                              NOT NULL CHAR(10)
```

Figure 7 Create table resources

Create table assessment:

The ASSESSMENT table stores details about various assessments. It includes columns for assessment identifiers, titles, weightage, status, release dates, and deadlines

```
SQL> CREATE TABLE Assessment (
  2      Assessment_Id NUMBER(15) PRIMARY KEY NOT NULL,
  3      Assessment_Title CHARACTER(20) NOT NULL,
  4      Weightage CHARACTER(10) NOT NULL,
  5      Status CHARACTER(10) NOT NULL,
  6      Release DATE NOT NULL,
  7      Deadline DATE NOT NULL
  8  );

Table created.

SQL> DESC Assessment;
 Name                                                  Null?    Type
 ----------------------------------------------------- -------- ------------
 ASSESSMENT_ID                                         NOT NULL NUMBER(15)
 ASSESSMENT_TITLE                                      NOT NULL CHAR(20)
 WEIGHTAGE                                             NOT NULL CHAR(10)
 STATUS                                                NOT NULL CHAR(10)
 RELEASE                                               NOT NULL DATE
 DEADLINE                                              NOT NULL DATE
```

Figure 8 Create table assessment

Create table notice:

The NOTICE table holds announcements or notices. It includes columns for notice identifiers, titles, and publication dates.

```
SQL> CREATE TABLE Notice (
  2      Notice_id NUMBER(10) PRIMARY KEY NOT NULL,
  3      Notice_Title CHARACTER(20) NOT NULL,
  4      Publish_on DATE NOT NULL
  5  );

Table created.

SQL> DESC Notice;
 Name                                                  Null?    Type
 ----------------------------------------------------- -------- ------------
 NOTICE_ID                                             NOT NULL NUMBER(10)
 NOTICE_TITLE                                          NOT NULL CHAR(20)
 PUBLISH_ON                                            NOT NULL DATE
```

Figure 9 Create table notice

Create table result:

The RESULT table holds the results of assessments. It includes columns for result identifiers, statuses, and total marks obtained.

```
SQL> CREATE TABLE Result (
  2      Result_Id NUMBER(10) PRIMARY KEY NOT NULL,
  3      result_status CHARACTER(15) NOT NULL,
  4      Total_Marks_Obtained NUMBER(3) NOT NULL
  5  );

Table created.

SQL> DESC Result;
 Name                                                  Null?    Type
 ----------------------------------------------------- -------- ---------------
 RESULT_ID                                             NOT NULL NUMBER(10)
 RESULT_STATUS                                         NOT NULL CHAR(15)
 TOTAL_MARKS_OBTAINED                                  NOT NULL NUMBER(3)
```

Figure 10 Create table result

Create table student:

The STUDENT table stores information about students. It includes columns for student identifiers, program identifiers, names, contact information, addresses, and enrollment dates.

```
SQL> CREATE TABLE Student (
  2      Student_ID NUMBER(10) PRIMARY KEY NOT NULL,
  3      Program_Id NUMBER(10) REFERENCES Program(Program_Id),
  4      Student_Name CHARACTER(20) NOT NULL,
  5      Student_Contact CHARACTER(15) UNIQUE NOT NULL,
  6      Student_Address CHARACTER(20) NOT NULL,
  7      Enrolled DATE NOT NULL
  8  );

Table created.

SQL> DESC Student;
 Name                                                  Null?    Type
 ----------------------------------------------------- -------- ----------------
 STUDENT_ID                                            NOT NULL NUMBER(10)
 PROGRAM_ID                                                     NUMBER(10)
 STUDENT_NAME                                          NOT NULL CHAR(20)
 STUDENT_CONTACT                                       NOT NULL CHAR(15)
 STUDENT_ADDRESS                                       NOT NULL CHAR(20)
 ENROLLED                                              NOT NULL DATE
```

Figure 11 Create table student

Create table Student_Module:

The STUDENT_MODULE table is a junction table that links students to the modules they are enrolled in. It includes Student_ID and Module_Id, both of which are foreign keys referencing the Student and Module tables, respectively. Student_ID and Module_Id forms the composite primary key, ensuring each student-module enrollment is uniquely identified.

```
SQL> CREATE TABLE Student_Module (
  2      Student_ID NUMBER(10) NOT NULL,
  3      Module_Id NUMBER(10) NOT NULL,
  4      PRIMARY KEY (Student_ID, Module_Id),
  5      FOREIGN KEY (Student_ID) REFERENCES Student(Student_ID),
  6      FOREIGN KEY (Module_Id) REFERENCES Module(Module_Id)
  7  );

Table created.

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

Figure 12 Create table student_module

CREATE TABLE Student_Module_Resources:

The STUDENT_MODULE_RESOURCES table links students and modules to the resources it includes Student_ID, Module_Id, and Resource_Id, all of which are foreign keys and the composite primary key of table referencing the Student, Module, and Resources tables.

```
SQL> CREATE TABLE Student_Module_Resources (
  2      Student_ID NUMBER(10) NOT NULL,
  3      Module_Id NUMBER(10) NOT NULL,
  4      Resource_Id NUMBER(12) NOT NULL,
  5      PRIMARY KEY (Student_ID, Module_Id, Resource_Id),
  6      FOREIGN KEY (Student_ID) REFERENCES Student(Student_ID),
  7      FOREIGN KEY (Module_Id) REFERENCES Module(Module_Id),
  8      FOREIGN KEY (Resource_Id) REFERENCES Resources(Resource_Id)
  9  );

Table created.

SQL> DESC Student_Module_Resources;
 Name                                                  Null?    Type
 ----------------------------------------------------- -------- ----------------
 STUDENT_ID                                            NOT NULL NUMBER(10)
 MODULE_ID                                             NOT NULL NUMBER(10)
 RESOURCE_ID                                           NOT NULL NUMBER(12)
```

Figure 13 Create table Student_Module_Resources

Create table Student_Module_Teacher

The STUDENT_MODULE_TEACHER table links students and modules to the teachers who teach them. It includes Student_ID, Module_Id, and Teacher_Id, all of which are foreign keys and the composite primary key of the table referencing the Student, Module, and Teacher tables, respectively.

```
SQL> CREATE TABLE Student_Module_Teacher (
  2      Student_ID NUMBER(10) NOT NULL,
  3      Module_Id NUMBER(10) NOT NULL,
  4      Teacher_Id NUMBER(10) NOT NULL,
  5      PRIMARY KEY (Student_ID, Module_Id, Teacher_Id),
  6      FOREIGN KEY (Student_ID) REFERENCES Student(Student_ID),
  7      FOREIGN KEY (Module_Id) REFERENCES Module(Module_Id),
  8      FOREIGN KEY (Teacher_Id) REFERENCES Teacher(Teacher_Id)
  9  );

Table created.

SQL> DESC Student_Module_Teacher;
 Name                                                  Null?    Type
 ----------------------------------------------------- -------- ----------------
 STUDENT_ID                                            NOT NULL NUMBER(10)
 MODULE_ID                                             NOT NULL NUMBER(10)
 TEACHER_ID                                            NOT NULL NUMBER(10)
```

Figure 14 Create table student_module_teacher

Create table Student_Module_Teacher_Notice

The STUDENT_MODULE_TEACHER_NOTICE table links students, modules, and teachers to the notices they receive. It includes Student_ID, Module_Id, Teacher_Id, and Notice_id, all of which are foreign keys referencing the Student, Module, Teacher, and Notice tables, respectively. The combination of these four columns forms the composite primary key.

```
SQL> CREATE TABLE Student_Module_Teacher_Notice (
  2      Student_ID NUMBER(10) NOT NULL,
  3      Module_Id NUMBER(10) NOT NULL,
  4      Teacher_Id NUMBER(10) NOT NULL,
  5      Notice_id NUMBER(10) NOT NULL,
  6      PRIMARY KEY (Student_ID, Module_Id, Teacher_Id, Notice_id),
  7      FOREIGN KEY (Student_ID) REFERENCES Student(Student_ID),
  8      FOREIGN KEY (Module_Id) REFERENCES Module(Module_Id),
  9      FOREIGN KEY (Teacher_Id) REFERENCES Teacher(Teacher_Id),
 10      FOREIGN KEY (Notice_id) REFERENCES Notice(Notice_id)
 11  );

Table created.

SQL> DESC Student_Module_Teacher_Notice;
 Name                                                      Null?    Type
 --------------------------------------------------------- -------- ----------------------
 STUDENT_ID                                                NOT NULL NUMBER(10)
 MODULE_ID                                                 NOT NULL NUMBER(10)
 TEACHER_ID                                                NOT NULL NUMBER(10)
 NOTICE_ID                                                 NOT NULL NUMBER(10)
```

Figure 15 Create table student_module_teacher_notice

Create table Student_Module_Assessment:

The STUDENT_MODULE_ASSESSMENT table links students and modules to the assessments they take. It includes Student_ID, Module_Id, Assessment_Id, and Result_Id, all of which are foreign keys referencing the Student, Module, Assessment, and Result tables, respectively. The combination of these four columns forms the composite primary key.

```
SQL> CREATE TABLE Student_Module_Assessment (
  2      Student_ID NUMBER(10) NOT NULL,
  3      Module_Id NUMBER(10) NOT NULL,
  4      Assessment_Id NUMBER(15) NOT NULL,
  5      Result_Id NUMBER(7) NOT NULL,
  6      PRIMARY KEY (Student_ID, Module_Id, Assessment_Id),
  7      FOREIGN KEY (Student_ID) REFERENCES Student(Student_ID),
  8      FOREIGN KEY (Module_Id) REFERENCES Module(Module_Id),
  9      FOREIGN KEY (Assessment_Id) REFERENCES Assessment(Assessment_Id),
 10      FOREIGN KEY (Result_Id) REFERENCES Result(Result_Id)
 11  );

Table created.

SQL> DESC Student_Module_Assessment;
 Name                                                      Null?    Type
 --------------------------------------------------------- -------- --------------------
 STUDENT_ID                                                NOT NULL NUMBER(10)
 MODULE_ID                                                 NOT NULL NUMBER(10)
 ASSESSMENT_ID                                             NOT NULL NUMBER(15)
 RESULT_ID                                                 NOT NULL NUMBER(7)
```

Figure 16 Create table student_module_assessment

List of all the tables created

```
SQL> SELECT*FROM TAB;

TNAME                          TABTYPE  CLUSTERID
------------------------------ ------- ----------
ASSESSMENT                     TABLE
MODULE                         TABLE
NOTICE                         TABLE
PROGRAM                        TABLE
RESOURCES                      TABLE
RESULT                         TABLE
STUDENT                        TABLE
STUDENT_MODULE                 TABLE
STUDENT_MODULE_ASSESSMENT      TABLE
STUDENT_MODULE_RESOURCES       TABLE
STUDENT_MODULE_TEACHER         TABLE

TNAME                          TABTYPE  CLUSTERID
------------------------------ ------- ----------
STUDENT_MODULE_TEACHER_NOTICE  TABLE
TEACHER                        TABLE

13 rows selected.
```

Figure 17 List of all the tables

5.3 INSERT

Insert into program

```
SQL> INSERT INTO Program (Program_Id, Program_Name, Faculty) VALUES (10001, 'Computing', 'IT');

1 row created.

SQL> INSERT INTO Program (Program_Id, Program_Name, Faculty) VALUES (10002, 'Networking', 'IT');

1 row created.

SQL> INSERT INTO Program (Program_Id, Program_Name, Faculty) VALUES (10003, 'Multimedia', 'IT');

1 row created.

SQL> INSERT INTO Program (Program_Id, Program_Name, Faculty) VALUES (10004, 'BBM', 'Management');

1 row created.

SQL> INSERT INTO Program (Program_Id, Program_Name, Faculty) VALUES (10005, 'BSW', 'Management');

1 row created.

SQL> INSERT INTO Program (Program_Id, Program_Name, Faculty) VALUES (10006, 'BBS', 'Management');

1 row created.

SQL> INSERT INTO Program (Program_Id, Program_Name, Faculty) VALUES (10007, 'BBA', 'Management');

1 row created.

SQL> SELECT * FROM Program;

PROGRAM_ID PROGRAM_NAME    FACULTY
---------- --------------- ------------------------------
     10001 Computing       IT
     10002 Networking      IT
     10003 Multimedia      IT
     10004 BBM             Management
     10005 BSW             Management
     10006 BBS             Management
     10007 BBA             Management

7 rows selected.
```

Figure 18 Insert into program table

INSERT INTO MODULE

```
SQL> INSERT INTO Module (Module_Id, Module_Name, Credit_Hour) VALUES (10001, 'Databases', 4);

1 row created.

SQL> INSERT INTO Module (Module_Id, Module_Name, Credit_Hour) VALUES (10002, 'Data Structures', 3);

1 row created.

SQL> INSERT INTO Module (Module_Id, Module_Name, Credit_Hour) VALUES (10003, 'Networking Fundamentals', 3);

1 row created.

SQL> INSERT INTO Module (Module_Id, Module_Name, Credit_Hour) VALUES (10004, 'Operating Systems', 4);

1 row created.

SQL> INSERT INTO Module (Module_Id, Module_Name, Credit_Hour) VALUES (10005, 'Software Engineering', 4);

1 row created.

SQL> INSERT INTO Module (Module_Id, Module_Name, Credit_Hour) VALUES (10006, 'Web Development', 3);

1 row created.

SQL> INSERT INTO Module (Module_Id, Module_Name, Credit_Hour) VALUES (10007, 'Artificial Intelligence', 4);

1 row created.

SQL> SELECT * FROM Module;

 MODULE_ID MODULE_NAME               CREDIT_HOUR
---------- ------------------------ -----------
     10001 Databases                            4
     10002 Data Structures                      3
     10003 Networking Fundamentals              3
     10004 Operating Systems                    4
     10005 Software Engineering                 4
     10006 Web Development                      3
     10007 Artificial Intelligence              4

7 rows selected.
```

Figure 19 Insert into module table

Insert into teacher

```
SQL> INSERT INTO Teacher (Teacher_Id, Teacher_Name, Teacher_Contact, Teacher_Address) VALUES (10001, 'Dr. Ramesh Adhikari', '9800001001', 'Kathmandu');

1 row created.

SQL> INSERT INTO Teacher (Teacher_Id, Teacher_Name, Teacher_Contact, Teacher_Address) VALUES (10002, 'Prof. Sushila Karki', '9800001002', 'Lalitpur');

1 row created.

SQL> INSERT INTO Teacher (Teacher_Id, Teacher_Name, Teacher_Contact, Teacher_Address) VALUES (10003, 'Dr. Binod Shrestha', '9800001003', 'Bhaktapur');

1 row created.

SQL> INSERT INTO Teacher (Teacher_Id, Teacher_Name, Teacher_Contact, Teacher_Address) VALUES (10004, 'Prof. Anju Sharma', '9800001004', 'Kathmandu');

1 row created.

SQL> INSERT INTO Teacher (Teacher_Id, Teacher_Name, Teacher_Contact, Teacher_Address) VALUES (10005, 'Dr. Prakash Thapa', '9800001005', 'Lalitpur');

1 row created.

SQL> INSERT INTO Teacher (Teacher_Id, Teacher_Name, Teacher_Contact, Teacher_Address) VALUES (10006, 'Prof. Meera Singh', '9800001006', 'Bhaktapur');

1 row created.

SQL> INSERT INTO Teacher (Teacher_Id, Teacher_Name, Teacher_Contact, Teacher_Address) VALUES (10007, 'Dr. Sunil Joshi', '9800001007', 'Kathmandu');

1 row created.

SQL> SELECT * FROM Teacher;

TEACHER_ID TEACHER_NAME         TEACHER_CONTACT TEACHER_ADDRESS
---------- -------------------- --------------- --------------------
     10001 Dr. Ramesh Adhikari  9800001001      Kathmandu
     10002 Prof. Sushila Karki  9800001002      Lalitpur
     10003 Dr. Binod Shrestha   9800001003      Bhaktapur
     10004 Prof. Anju Sharma    9800001004      Kathmandu
     10005 Dr. Prakash Thapa    9800001005      Lalitpur
     10006 Prof. Meera Singh    9800001006      Bhaktapur
     10007 Dr. Sunil Joshi      9800001007      Kathmandu

7 rows selected.
```

Figure 20 Insert into teacher table

Insert into table resources

```
SQL> INSERT INTO Resources (Resource_Id, Resource_Type, Duration) VALUES (10001, 'Type 1', '2 hours');

1 row created.

SQL> INSERT INTO Resources (Resource_Id, Resource_Type, Duration) VALUES (10002, 'Type 2', '1 hour');

1 row created.

SQL> INSERT INTO Resources (Resource_Id, Resource_Type, Duration) VALUES (10003, 'Type 3', '5 hours');

1 row created.

SQL> INSERT INTO Resources (Resource_Id, Resource_Type, Duration) VALUES (10004, 'Type 2', '30 mins');

1 row created.

SQL> INSERT INTO Resources (Resource_Id, Resource_Type, Duration) VALUES (10005, 'Type 3', '3 hours');

1 row created.

SQL> INSERT INTO Resources (Resource_Id, Resource_Type, Duration) VALUES (10006, 'Type 1', '1.5 hrs');

1 row created.

SQL> INSERT INTO Resources (Resource_Id, Resource_Type, Duration) VALUES (10007, 'Type 2', '2.5 hrs');

1 row created.

SQL> SELECT * FROM Resources;

RESOURCE_ID RESOURCE_TYPE        DURATION
----------- -------------------- ----------
      10001 Type 1               2 hours
      10002 Type 2               1 hour
      10003 Type 3               5 hours
      10004 Type 2               30 mins
      10005 Type 3               3 hours
      10006 Type 1               1.5 hrs
      10007 Type 2               2.5 hrs

7 rows selected.
```

Figure 21 Insert into resources table

Insert into assessment

```
SQL> INSERT INTO Assessment (Assessment_Id, Assessment_Title, Weightage, Status, Release, Deadline) VALUES (10001, 'Test 1', '30', 'Completed', TO_DATE('2024-04-01', 'YYYY-MM-DD'), TO_DATE('2024-04-15', 'YYYY-MM
-DD'));

1 row created.

SQL> INSERT INTO Assessment (Assessment_Id, Assessment_Title, Weightage, Status, Release, Deadline) VALUES (10002, 'Test 2', '70', 'Missed', TO_DATE('2024-05-01', 'YYYY-MM-DD'), TO_DATE('2024-05-30', 'YYYY-MM-DD
'));

1 row created.

SQL> INSERT INTO Assessment (Assessment_Id, Assessment_Title, Weightage, Status, Release, Deadline) VALUES (10003, 'Coursework 1', '10', 'Submitted', TO_DATE('2024-03-01', 'YYYY-MM-DD'), TO_DATE('2024-03-05', 'Y
YYY-MM-DD'));

1 row created.

SQL> INSERT INTO Assessment (Assessment_Id, Assessment_Title, Weightage, Status, Release, Deadline) VALUES (10004, 'Coursework 2', '10', 'Missed', TO_DATE('2024-03-15', 'YYYY-MM-DD'), TO_DATE('2024-03-20', 'YYYY
-MM-DD'));

1 row created.

SQL> INSERT INTO Assessment (Assessment_Id, Assessment_Title, Weightage, Status, Release, Deadline) VALUES (10005, 'Test 1', '20', 'Missed', TO_DATE('2024-04-10', 'YYYY-MM-DD'), TO_DATE('2024-04-20', 'YYYY-MM-DD
'));

1 row created.

SQL> INSERT INTO Assessment (Assessment_Id, Assessment_Title, Weightage, Status, Release, Deadline) VALUES (10006, 'Test 2', '50', 'Completed', TO_DATE('2024-05-10', 'YYYY-MM-DD'), TO_DATE('2024-05-25', 'YYYY-MM
-DD'));

1 row created.

SQL> INSERT INTO Assessment (Assessment_Id, Assessment_Title, Weightage, Status, Release, Deadline) VALUES (10007, 'Coursework 1', '30', 'Submitted', TO_DATE('2024-04-05', 'YYYY-MM-DD'), TO_DATE('2024-04-10', 'Y
YYY-MM-DD'));

1 row created.

SQL> SELECT * FROM Assessment;

ASSESSMENT_ID ASSESSMENT_TITLE    WEIGHTAGE  STATUS     RELEASE   DEADLINE
------------- ------------------- ---------- ---------- --------- ---------
        10001 Test 1              30         Completed  01-APR-24 15-APR-24
        10002 Test 2              70         Missed     01-MAY-24 30-MAY-24
        10003 Coursework 1        10         Submitted  01-MAR-24 05-MAR-24
        10004 Coursework 2        10         Missed     15-MAR-24 20-MAR-24
        10005 Test 1              20         Missed     10-APR-24 20-APR-24
        10006 Test 2              50         Completed  10-MAY-24 25-MAY-24
        10007 Coursework 1        30         Submitted  05-APR-24 10-APR-24

7 rows selected.
```

Figure 22 Insert into assessment table

Insert into notice

```
SQL> INSERT INTO Notice (Notice_id, Notice_Title, Publish_on) VALUES (1000001, 'Exam Schedule', TO_DATE('2024-05-10', 'YYYY-MM-DD'));

1 row created.

SQL> INSERT INTO Notice (Notice_id, Notice_Title, Publish_on) VALUES (1000002, 'Assignment Deadline', TO_DATE('2024-05-15', 'YYYY-MM-DD'));

1 row created.

SQL> INSERT INTO Notice (Notice_id, Notice_Title, Publish_on) VALUES (1000003, 'Holiday Notice', TO_DATE('2024-05-20', 'YYYY-MM-DD'));

1 row created.

SQL> INSERT INTO Notice (Notice_id, Notice_Title, Publish_on) VALUES (1000004, 'Workshop Notice', TO_DATE('2024-05-25', 'YYYY-MM-DD'));

1 row created.

SQL> INSERT INTO Notice (Notice_id, Notice_Title, Publish_on) VALUES (1000005, 'Seminar Invitation', TO_DATE('2024-05-05', 'YYYY-MM-DD'));

1 row created.

SQL> INSERT INTO Notice (Notice_id, Notice_Title, Publish_on) VALUES (1000006, 'Project Submission', TO_DATE('2024-05-18', 'YYYY-MM-DD'));

1 row created.

SQL> INSERT INTO Notice (Notice_id, Notice_Title, Publish_on) VALUES (1000007, 'Guest Lecture', TO_DATE('2024-05-28', 'YYYY-MM-DD'));

1 row created.

SQL> SELECT*FROM Notice;

 NOTICE_ID NOTICE_TITLE         PUBLISH_O
---------- -------------------- ---------
   1000001 Exam Schedule        10-MAY-24
   1000002 Assignment Deadline  15-MAY-24
   1000003 Holiday Notice       20-MAY-24
   1000004 Workshop Notice      25-MAY-24
   1000005 Seminar Invitation   05-MAY-24
   1000006 Project Submission   18-MAY-24
   1000007 Guest Lecture        28-MAY-24

7 rows selected.
```

Figure 23 Insert into notice table

Insert into result

```
SQL> INSERT INTO Result (Result_Id, result_status, Total_Marks_Obtained) VALUES (10001, 'Passed', 85);

1 row created.

SQL> INSERT INTO Result (Result_Id, result_status, Total_Marks_Obtained) VALUES (10002, 'Failed', 35);

1 row created.

SQL> INSERT INTO Result (Result_Id, result_status, Total_Marks_Obtained) VALUES (10003, 'Passed', 75);

1 row created.

SQL> INSERT INTO Result (Result_Id, result_status, Total_Marks_Obtained) VALUES (10004, 'Failed', 20);

1 row created.

SQL> INSERT INTO Result (Result_Id, result_status, Total_Marks_Obtained) VALUES (10005, 'Passed', 90);

1 row created.

SQL> INSERT INTO Result (Result_Id, result_status, Total_Marks_Obtained) VALUES (10006, 'Failed', 20);

1 row created.

SQL> INSERT INTO Result (Result_Id, result_status, Total_Marks_Obtained) VALUES (10007, 'Passed', 65);

1 row created.

SQL> SELECT * FROM Result;

 RESULT_ID RESULT_STATUS   TOTAL_MARKS_OBTAINED
---------- --------------- --------------------
     10001 Passed                            85
     10002 Failed                            35
     10003 Passed                            75
     10004 Failed                            20
     10005 Passed                            90
     10006 Failed                            20
     10007 Passed                            65

7 rows selected.
```

Figure 24 Insert into result table

Insert into student

```
SQL> INSERT INTO Student (Student_ID, Program_Id, Student_Name, Student_Contact, Student_Address, Enrolled) VALUES (2394740, 10001, 'Ram Shrestha', '9800000001', 'Kathmandu', TO_DATE('2024-01-15', 'YYYY-MM-DD'))
;

1 row created.

SQL> INSERT INTO Student (Student_ID, Program_Id, Student_Name, Student_Contact, Student_Address, Enrolled) VALUES (2394741, 10002, 'Sita Tamang', '9800000002', 'Lalitpur', TO_DATE('2024-01-20', 'YYYY-MM-DD'));

1 row created.

SQL> INSERT INTO Student (Student_ID, Program_Id, Student_Name, Student_Contact, Student_Address, Enrolled) VALUES (2394742, 10003, 'Hari Bahadur', '9800000003', 'Bhaktapur', TO_DATE('2024-02-10', 'YYYY-MM-DD'))
;

1 row created.

SQL> INSERT INTO Student (Student_ID, Program_Id, Student_Name, Student_Contact, Student_Address, Enrolled) VALUES (2394743, 10004, 'Gita Rai', '9800000004', 'Kathmandu', TO_DATE('2024-03-05', 'YYYY-MM-DD'));

1 row created.

SQL> INSERT INTO Student (Student_ID, Program_Id, Student_Name, Student_Contact, Student_Address, Enrolled) VALUES (2394744, 10005, 'Krishna Gurung', '9800000005', 'Lalitpur', TO_DATE('2024-04-12', 'YYYY-MM-DD')
);

1 row created.

SQL> INSERT INTO Student (Student_ID, Program_Id, Student_Name, Student_Contact, Student_Address, Enrolled) VALUES (2394745, 10006, 'Radha Magar', '9800000006', 'Bhaktapur', TO_DATE('2024-05-18', 'YYYY-MM-DD'));


1 row created.

SQL> INSERT INTO Student (Student_ID, Program_Id, Student_Name, Student_Contact, Student_Address, Enrolled) VALUES (2394746, 10007, 'Bishnu Thapa', '9800000007', 'Kathmandu', TO_DATE('2024-06-25', 'YYYY-MM-DD'))
;

1 row created.

SQL> SELECT * FROM Student;

STUDENT_ID PROGRAM_ID STUDENT_NAME         STUDENT_CONTACT STUDENT_ADDRESS      ENROLLED
---------- ---------- -------------------- --------------- -------------------- ---------
   2394740      10001 Ram Shrestha         9800000001      Kathmandu            15-JAN-24
   2394741      10002 Sita Tamang          9800000002      Lalitpur             20-JAN-24
   2394742      10003 Hari Bahadur         9800000003      Bhaktapur            10-FEB-24
   2394743      10004 Gita Rai             9800000004      Kathmandu            05-MAR-24
   2394744      10005 Krishna Gurung       9800000005      Lalitpur             12-APR-24
   2394745      10006 Radha Magar          9800000006      Bhaktapur            18-MAY-24
   2394746      10007 Bishnu Thapa         9800000007      Kathmandu            25-JUN-24

7 rows selected.
```

Figure 25 Insert into student table

Insert into student_module

```
SQL> INSERT INTO Student_Module (Student_ID, Module_Id) VALUES (2394740, 10001);

1 row created.

SQL> INSERT INTO Student_Module (Student_ID, Module_Id) VALUES (2394741, 10002);

1 row created.

SQL> INSERT INTO Student_Module (Student_ID, Module_Id) VALUES (2394742, 10003);

1 row created.

SQL> INSERT INTO Student_Module (Student_ID, Module_Id) VALUES (2394743, 10004);

1 row created.

SQL> INSERT INTO Student_Module (Student_ID, Module_Id) VALUES (2394744, 10005);

1 row created.

SQL> INSERT INTO Student_Module (Student_ID, Module_Id) VALUES (2394745, 10006);

1 row created.

SQL> INSERT INTO Student_Module (Student_ID, Module_Id) VALUES (2394746, 10007);

1 row created.

SQL> SELECT * FROM Student_Module;

STUDENT_ID  MODULE_ID
---------- ----------
   2394740      10001
   2394741      10002
   2394742      10003
   2394743      10004
   2394744      10005
   2394745      10006
   2394746      10007

7 rows selected.
```

Figure 26 Insert into student_module table

Insert into student_module_resource

```
SQL> INSERT INTO Student_Module_Resources (Student_ID, Module_Id, Resource_Id) VALUES (2394740, 10001, 10001);

1 row created.

SQL> INSERT INTO Student_Module_Resources (Student_ID, Module_Id, Resource_Id) VALUES (2394741, 10002, 10002);

1 row created.

SQL> INSERT INTO Student_Module_Resources (Student_ID, Module_Id, Resource_Id) VALUES (2394742, 10003, 10003);

1 row created.

SQL> INSERT INTO Student_Module_Resources (Student_ID, Module_Id, Resource_Id) VALUES (2394743, 10004, 10004);

1 row created.

SQL> INSERT INTO Student_Module_Resources (Student_ID, Module_Id, Resource_Id) VALUES (2394744, 10005, 10005);

1 row created.

SQL> INSERT INTO Student_Module_Resources (Student_ID, Module_Id, Resource_Id) VALUES (2394745, 10006, 10006);

1 row created.

SQL> INSERT INTO Student_Module_Resources (Student_ID, Module_Id, Resource_Id) VALUES (2394746, 10007, 10007);

1 row created.

SQL> SELECT * FROM Student_Module_Resources;

STUDENT_ID  MODULE_ID RESOURCE_ID
---------- ---------- -----------
   2394740      10001       10001
   2394741      10002       10002
   2394742      10003       10003
   2394743      10004       10004
   2394744      10005       10005
   2394745      10006       10006
   2394746      10007       10007

7 rows selected.
```

Figure 27 Insert into student_module_resources table

Insert into student_mdoule_teacher

```
SQL> INSERT INTO Student_Module_Teacher (Student_ID, Module_Id, Teacher_Id) VALUES (2394740, 10001, 10001);

1 row created.

SQL> INSERT INTO Student_Module_Teacher (Student_ID, Module_Id, Teacher_Id) VALUES (2394741, 10002, 10002);

1 row created.

SQL> INSERT INTO Student_Module_Teacher (Student_ID, Module_Id, Teacher_Id) VALUES (2394742, 10003, 10003);

1 row created.

SQL> INSERT INTO Student_Module_Teacher (Student_ID, Module_Id, Teacher_Id) VALUES (2394743, 10004, 10004);

1 row created.

SQL> INSERT INTO Student_Module_Teacher (Student_ID, Module_Id, Teacher_Id) VALUES (2394744, 10005, 10005);

1 row created.

SQL> INSERT INTO Student_Module_Teacher (Student_ID, Module_Id, Teacher_Id) VALUES (2394745, 10006, 10006);

1 row created.

SQL> INSERT INTO Student_Module_Teacher (Student_ID, Module_Id, Teacher_Id) VALUES (2394746, 10007, 10007);

1 row created.

SQL> SELECT * FROM Student_Module_Teacher;

STUDENT_ID  MODULE_ID TEACHER_ID
---------- ---------- ----------
   2394740      10001      10001
   2394741      10002      10002
   2394742      10003      10003
   2394743      10004      10004
   2394744      10005      10005
   2394745      10006      10006
   2394746      10007      10007

7 rows selected.
```

Figure 28 Insert into student_module_teacher table

Insert into student_module_assessment

```
SQL> INSERT INTO Student_Module_Assessment (Student_ID, Module_Id, Assessment_Id, Result_Id) VALUES (2394740, 10001, 10001, 10001);

1 row created.

SQL> INSERT INTO Student_Module_Assessment (Student_ID, Module_Id, Assessment_Id, Result_Id) VALUES (2394741, 10002, 10002, 10002);

1 row created.

SQL> INSERT INTO Student_Module_Assessment (Student_ID, Module_Id, Assessment_Id, Result_Id) VALUES (2394742, 10003, 10003, 10003);

1 row created.

SQL> INSERT INTO Student_Module_Assessment (Student_ID, Module_Id, Assessment_Id, Result_Id) VALUES (2394743, 10004, 10004, 10004);

1 row created.

SQL> INSERT INTO Student_Module_Assessment (Student_ID, Module_Id, Assessment_Id, Result_Id) VALUES (2394744, 10005, 10005, 10005);

1 row created.

SQL> INSERT INTO Student_Module_Assessment (Student_ID, Module_Id, Assessment_Id, Result_Id) VALUES (2394745, 10006, 10006, 10006);

1 row created.

SQL> INSERT INTO Student_Module_Assessment (Student_ID, Module_Id, Assessment_Id, Result_Id) VALUES (2394746, 10007, 10007, 10007);

1 row created.

SQL> SELECT * FROM Student_Module_Assessment;

STUDENT_ID  MODULE_ID ASSESSMENT_ID  RESULT_ID
---------- ---------- ------------- ----------
   2394740      10001         10001      10001
   2394741      10002         10002      10002
   2394742      10003         10003      10003
   2394743      10004         10004      10004
   2394744      10005         10005      10005
   2394745      10006         10006      10006
   2394746      10007         10007      10007

7 rows selected.
```

Figure 29 Insert into student_module_assessment table

Insert into student_module_teacher_notice

```
SQL> INSERT INTO Student_Module_Teacher_Notice (Student_ID, Module_Id, Teacher_Id, Notice_id) VALUES (2394740, 10001, 10001, 1000001);

1 row created.

SQL> INSERT INTO Student_Module_Teacher_Notice (Student_ID, Module_Id, Teacher_Id, Notice_id) VALUES (2394741, 10002, 10002, 1000002);

1 row created.

SQL> INSERT INTO Student_Module_Teacher_Notice (Student_ID, Module_Id, Teacher_Id, Notice_id) VALUES (2394742, 10003, 10003, 1000003);

1 row created.

SQL> INSERT INTO Student_Module_Teacher_Notice (Student_ID, Module_Id, Teacher_Id, Notice_id) VALUES (2394743, 10004, 10004, 1000004);

1 row created.

SQL> INSERT INTO Student_Module_Teacher_Notice (Student_ID, Module_Id, Teacher_Id, Notice_id) VALUES (2394744, 10005, 10005, 1000005);

1 row created.

SQL> INSERT INTO Student_Module_Teacher_Notice (Student_ID, Module_Id, Teacher_Id, Notice_id) VALUES (2394745, 10006, 10006, 1000006);

1 row created.

SQL> INSERT INTO Student_Module_Teacher_Notice (Student_ID, Module_Id, Teacher_Id, Notice_id) VALUES (2394746, 10007, 10007, 1000007);

1 row created.

SQL> SELECT * FROM Student_Module_Teacher_Notice;

STUDENT_ID  MODULE_ID TEACHER_ID  NOTICE_ID
---------- ---------- ---------- ----------
   2394740      10001      10001    1000001
   2394741      10002      10002    1000002
   2394742      10003      10003    1000003
   2394743      10004      10004    1000004
   2394744      10005      10005    1000005
   2394745      10006      10006    1000006
   2394746      10007      10007    1000007

7 rows selected.

SQL> COMMIT;

Commit complete.
```

Figure 30 Insert into student_module_teacher_notice table

6. DATABASE QUERYING

A database query is a request for information from the database, written in a specific syntax. Queries allows Database Administrators (DBAs) or other users to access data from a database management system (DBMS), where information is organized in tables or collections of related data. A query retrieves and displays this data in tables so that are easy to understand. A database query acts as a communication code between a user and a machine. The query helps the machine understand the command and execute the specified action. There are various query languages used to write these queries (Motadata, n.d.).

6.1 INFORMATION QUERY

Informational queries are those that retrieve data to present data or information in a readable format for analysis or reporting without necessarily performing any data manipulation.

1. List the programs that are available in the college and the total number of students

enrolled in each.

Functionality: List Programs and Total Number of Enrolled Students.

Script 1:

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

SELECT Clause:

p.Program_Name: Retrieves the name of the program from the Program table.

COUNT(s.Student_ID) AS Total_Students: Counts the number of students enrolled in each program. The COUNT function counts the non-null Student_ID values in the Student table for each program and names the result Total_Students.

FROM Clause:

Program p: Uses the Program table and gives it the alias p.

LEFT JOIN Clause:

LEFT JOIN Student s ON p.Program_Id = s.Program_Id: Joins the Program table (p) with the Student table (s) based on the Program_Id column. This ensures all programs are included

in the result, even if no students are enrolled in them. If a program has no students, the count will be zero.

GROUP BY Clause:

GROUP BY p.Program_Name: Groups the results by the Program_Name column, ensuring the student count is calculated separately for each program.

Purpose:

This query retrieves the names of all programs available in the college and the total number of students enrolled in each program. The output includes the program name and the count of enrolled students, providing a clear overview of student distribution across different programs.

```
SQL> SELECT
  2      p.Program_Name,
  3      COUNT(s.Student_ID) AS Total_Students
  4  FROM
  5      Program p
  6  LEFT JOIN
  7      Student s ON p.Program_Id = s.Program_Id
  8  GROUP BY
  9      p.Program_Name;

PROGRAM_NAME    TOTAL_STUDENTS
--------------- --------------
BBS                          1
BBA                          1
Computing                    1
Multimedia                   1
BSW                          1
BBM                          1
Networking                   1

7 rows selected.
```

Figure 31 Query 1

2. List all announcements made for a particular module from 1st May 2024 to 28th May 2024.

Functionality: List All Announcements for a Particular Module from 1st May 2024 to 28th May 2024.

Script 2:

```
SELECT n.Notice_Title, n.Publish_on

FROM Notice n

JOIN Student_Module_Teacher_Notice smtn ON n.Notice_id = smtn.Notice_id

WHERE smtn.Module_Id = 10001

 AND n.Publish_on BETWEEN TO_DATE('2024-05-01', 'YYYY-MM-DD') AND
TO_DATE('2024-05-28', 'YYYY-MM-DD');
```

SELECT Clause:

n.Notice_Title, n.Publish_on: Selects the title and publish date of the notices.

FROM Clause:

Notice n: Specifies the Notice table as the main table.

JOIN Clause:

JOIN Student_Module_Teacher_Notice smtn ON n.Notice_id = smtn.Notice_id: Joins the Student_Module_Teacher_Notice table to the Notice table based on the Notice_id column.

WHERE Clause:

smtn.Module_Id = 10001: Filters the results to only include announcements for the specified module (Module_Id = 10001).

AND n.Publish_on BETWEEN TO_DATE('2024-05-01', 'YYYY-MM-DD') AND TO_DATE('2024-05-28', 'YYYY-MM-DD'): Further filters the results to only include announcements made between 1st May 2024 and 28th May 2024.

Purpose:

This query retrieves all announcements made for a specific module (Module_Id = 10001) within the date range from 1st May 2024 to 28th May 2024. The output includes the title and publish date of each notice, helping users to identify relevant announcements for that module during the specified period.

```
SQL> SELECT n.Notice_Title, n.Publish_on
  2  FROM Notice n
  3  JOIN Student_Module_Teacher_Notice smtn ON n.Notice_id = smtn.Notice_id
  4  WHERE smtn.Module_Id = 10001
  5    AND n.Publish_on BETWEEN TO_DATE('2024-05-01', 'YYYY-MM-DD') AND TO_DATE('2024-05-28', 'YYYY-MM-DD');

NOTICE_TITLE         PUBLISH_O
-------------------- ---------
Exam Schedule        10-MAY-24
```

Figure 32 Query 2

3. List modules that begin with 'D' and total resources uploaded for those modules.

Functionality: List Modules Beginning with 'D' and Their Total Resources**.**

Script 3:

```
SELECT m.Module_Name, COUNT(r.Resource_Id) AS Total_Resources
FROM Module m
LEFT JOIN Student_Module_Resources smr ON m.Module_Id = smr.Module_Id
LEFT JOIN Resources r ON smr.Resource_Id = r.Resource_Id
WHERE m.Module_Name LIKE 'D%'
GROUP BY m.Module_Name;
```

SELECT Clause:

m.Module_Name, COUNT(r.Resource_Id) AS Total_Resources: Selects the module name and counts the number of resources associated with each module. The count is aliased as Total_Resources.

FROM Clause:

Module m: Specifies the Module table as the main table.

LEFT JOIN Clause:

LEFT JOIN Student_Module_Resources smr ON m.Module_Id = smr.Module_Id: Joins the Student_Module_Resources table to the Module table based on the Module_Id column.

LEFT JOIN Resources r ON smr.Resource_Id = r.Resource_Id: Joins the Resources table to the Student_Module_Resources table based on the Resource_Id column.

WHERE Clause:

m.Module_Name LIKE 'D%': Filters the results to only include modules whose names begin with the letter 'D'.

GROUP BY Clause:

GROUP BY m.Module_Name: Groups the results by module name to get the total number of resources for each module.

Purpose:

This query retrieves the names of all modules that begin with the letter 'D' and the total number of resources uploaded for each of those modules. The output includes the module name and the count of associated resources, helping users to understand resource allocation for specific modules.

```
SQL> SELECT m.Module_Name, COUNT(r.Resource_Id) AS Total_Resources
  2  FROM Module m
  3  LEFT JOIN Student_Module_Resources smr ON m.Module_Id = smr.Module_Id
  4  LEFT JOIN Resources r ON smr.Resource_Id = r.Resource_Id
  5  WHERE m.Module_Name LIKE 'D%'
  6  GROUP BY m.Module_Name;

MODULE_NAME               TOTAL_RESOURCES
------------------------- ---------------
Databases                               1
Data Structures                         1
```

Figure 33 Query 3

4. List students with their enrolled programs who haven't submitted assessments for a specific module.

Functionality: List Students and Their Enrolled Program Who Have Not Submitted Any Assessments for a Particular Module.

Script 4:

```
SELECT s.Student_Name, p.Program_Name
FROM Student s
JOIN Program p ON s.Program_Id = p.Program_Id
LEFT JOIN Student_Module_Assessment sma ON s.Student_ID = sma.Student_ID AND sma.Module_Id = 10001
WHERE sma.Assessment_Id IS NULL;
```

SELECT Clause:

s.Student_Name, p.Program_Name: Selects the student name and the program name.

FROM Clause:

Student s: Specifies the Student table as the main table.

JOIN Clause:

JOIN Program p ON s.Program_Id = p.Program_Id: Joins the Program table to the Student table based on the Program_Id column.

LEFT JOIN Clause:

LEFT JOIN Student_Module_Assessment sma ON s.Student_ID = sma.Student_ID AND sma.Module_Id = 10001: Left joins the Student_Module_Assessment table to the Student table based on the Student_ID column and filters for the specific module (Module_Id = 10001). This ensures all students are included in the result, even if they have not submitted any assessments for the specified module.

WHERE Clause:

WHERE sma.Assessment_Id IS NULL: Filters the results to only include students who have not submitted any assessments for the specified module.

Purpose:

This query retrieves the names of all students along with their enrolled program who have not submitted any assessments for a particular module (Module_Id = 10001). The output includes the student name and the program name, helping users to identify students who have not completed their assessments for the specified module

```
SQL> SELECT s.Student_Name, p.Program_Name
  2  FROM Student s
  3  JOIN Program p ON s.Program_Id = p.Program_Id
  4  LEFT JOIN Student_Module_Assessment sma ON s.Student_ID = sma.Student_ID AND sma.Module_Id = 10001
  5  WHERE sma.Assessment_Id IS NULL;

STUDENT_NAME         PROGRAM_NAME
-------------------- ---------------
Sita Tamang          Networking
Hari Bahadur         Multimedia
Gita Rai             BBM
Krishna Gurung       BSW
Radha Magar          BBS
Bishnu Thapa         BBA

6 rows selected.
```

Figure 34 Query 4

5. List teachers who teach more than one module.

Functionality: List All Teachers Who Teach More Than One Module.

Script 5:

```
SELECT t.Teacher_Name

FROM Teacher t

JOIN Student_Module_Teacher smt ON t.Teacher_Id = smt.Teacher_Id

GROUP BY t.Teacher_Name

HAVING COUNT(DISTINCT smt.Module_Id) > 1;
```

SELECT Clause:

t.Teacher_Name: Selects the teacher name.

FROM Clause:

Teacher t: Specifies the Teacher table as the main table.

JOIN Clause:

JOIN Student_Module_Teacher smt ON t.Teacher_Id = smt.Teacher_Id: Joins the Student_Module_Teacher table to the Teacher table based on the Teacher_Id column.

GROUP BY Clause:

GROUP BY t.Teacher_Name: Groups the results by teacher name.

HAVING Clause:

HAVING COUNT(DISTINCT smt.Module_Id) > 1: Filters the results to only include teachers who teach more than one module. The COUNT(DISTINCT smt.Module_Id) ensures that each module is counted only once per teacher

Purpose:

This query retrieves the names of all teachers who teach more than one module. The output includes the teacher name, helping users to identify teachers who are involved in multiple modules.

```
SQL> SELECT t.Teacher_Name
  2  FROM Teacher t
  3  JOIN Student_Module_Teacher smt ON t.Teacher_Id = smt.Teacher_Id
  4  GROUP BY t.Teacher_Name
  5  HAVING COUNT(DISTINCT smt.Module_Id) > 1;

no rows selected
```

Figure 35 Query 5

6.2 TRANSACTION QUERY

Transactional queries involve specific actions to manipulate or retrieve data from the database. These queries are designed to perform tasks such as inserting, updating, or deleting records, as well as retrieving specific sets of data based on certain conditions.

1. Identify the module with the latest assessment deadline.

Functionality: Identify the Module with the Latest Assessment Deadline.

Script 1:

```
SELECT m.Module_Name, a.Deadline
FROM Module m
JOIN Student_Module_Assessment sma ON m.Module_Id = sma.Module_Id
JOIN Assessment a ON sma.Assessment_Id = a.Assessment_Id
WHERE a.Deadline = (
   SELECT MAX(Deadline)
   FROM Assessment
);
```

SELECT Clause:

m.Module_Name, a.Deadline: Selects the module name and the deadline of the assessment.

FROM Clause:

Module m: Specifies the Module table as the main table.

JOIN Clause:

JOIN Student_Module_Assessment sma ON m.Module_Id = sma.Module_Id: Joins the Student_Module_Assessment table to the Module table based on the Module_Id column.

JOIN Assessment a ON sma.Assessment_Id = a.Assessment_Id: Joins the Assessment table to the Student_Module_Assessment table based on the Assessment_Id column.

WHERE Clause:

WHERE a.Deadline = (SELECT MAX(Deadline) FROM Assessment): Filters the results to only include the assessment with the latest deadline. The subquery (SELECT MAX(Deadline) FROM Assessment) finds the latest deadline date from the Assessment table.

Purpose:

This query identifies the module that has the latest assessment deadline. The output includes the module name and the deadline date, helping users to determine which module has the most recent assessment deadline.

```
SQL> SELECT m.Module_Name, a.Deadline
  2  FROM Module m
  3  JOIN Student_Module_Assessment sma ON m.Module_Id = sma.Module_Id
  4  JOIN Assessment a ON sma.Assessment_Id = a.Assessment_Id
  5  WHERE a.Deadline = (
  6      SELECT MAX(Deadline)
  7      FROM Assessment
  8  );

MODULE_NAME               DEADLINE
------------------------- ---------
Data Structures           30-MAY-24
```

Figure 36 Transaction query 1

2. Find the top three students with the highest total scores across all modules.

Functionality: Find the Top Three Students with the Highest Total Score Across All Modules.

Script 2:

```
SELECT Student_Name, Total_Score
FROM (
  SELECT s.Student_Name, SUM(r.Total_Marks_Obtained) AS Total_Score
  FROM Student s
  JOIN Student_Module_Assessment sma ON s.Student_ID = sma.Student_ID
  JOIN Result r ON sma.Result_Id = r.Result_Id
  GROUP BY s.Student_Name
```

```
    ORDER BY Total_Score DESC
)
WHERE ROWNUM <= 3;
```

SELECT Clause:

Student_Name, Total_Score: Selects the student name and their total score.

FROM Clause:

FROM ( ): This subquery calculates the total score for each student.

Subquery:

SELECT Clause:

s.Student_Name, SUM(r.Total_Marks_Obtained) AS Total_Score: Selects the student name and calculates the total score by summing the marks obtained.

FROM Clause:

Student s: Specifies the Student table as the main table.

JOIN Clause:

JOIN Student_Module_Assessment sma ON s.Student_ID = sma.Student_ID: Joins the Student_Module_Assessment table to the Student table based on the Student_ID column.

JOIN Result r ON sma.Result_Id = r.Result_Id: Joins the Result table to the Student_Module_Assessment table based on the Result_Id column.

GROUP BY Clause:

GROUP BY s.Student_Name: Groups the results by student name to calculate the total score for each student.

ORDER BY Clause:

ORDER BY Total_Score DESC: Orders the results by total score in descending order.

WHERE Clause:

WHERE ROWNUM <= 3: Filters the results to include only the top three students.

Purpose:

This query retrieves the top three students who have the highest total score across all modules. The output includes the student's name and their total score, helping users to identify the highest-performing students.

```
SQL> SELECT Student_Name, Total_Score
  2  FROM (
  3      SELECT s.Student_Name, SUM(r.Total_Marks_Obtained) AS Total_Score
  4      FROM Student s
  5      JOIN Student_Module_Assessment sma ON s.Student_ID = sma.Student_ID
  6      JOIN Result r ON sma.Result_Id = r.Result_Id
  7      GROUP BY s.Student_Name
  8      ORDER BY Total_Score DESC
  9  )
 10  WHERE ROWNUM <= 3;

STUDENT_NAME         TOTAL_SCORE
-------------------- -----------
Krishna Gurung                90
Ram Shrestha                  85
Hari Bahadur                  75
```

Figure 37 Transaction query 2

3. Find the total number of assessments and average score per program.

Functionality: Find the Total Number of Assessments and Average Score for Each Program.

Script 3: SELECT p.Program_Name,

COUNT(a.Assessment_Id) AS Total_Assessments,

AVG(r.Total_Marks_Obtained) AS Average_Score

FROM Program p

JOIN Student s ON p.Program_Id = s.Program_Id

JOIN Student_Module_Assessment sma ON s.Student_ID = sma.Student_ID

JOIN Assessment a ON sma.Assessment_Id = a.Assessment_Id

JOIN Result r ON sma.Result_Id = r.Result_Id

GROUP BY p.Program_Name;

Purpose:

This query retrieves the total number of assessments for each program and the average score across all assessments in those programs. The output includes the program name, the count of assessments, and the average score, providing a comprehensive overview of assessment activity and performance for each program.

```
SQL> SELECT p.Program_Name,
  2         COUNT(a.Assessment_Id) AS Total_Assessments,
  3         AVG(r.Total_Marks_Obtained) AS Average_Score
  4  FROM Program p
  5  JOIN Student s ON p.Program_Id = s.Program_Id
  6  JOIN Student_Module_Assessment sma ON s.Student_ID = sma.Student_ID
  7  JOIN Assessment a ON sma.Assessment_Id = a.Assessment_Id
  8  JOIN Result r ON sma.Result_Id = r.Result_Id
  9  GROUP BY p.Program_Name;

PROGRAM_NAME    TOTAL_ASSESSMENTS AVERAGE_SCORE
--------------- ----------------- -------------
BBS                             1            20
BBA                             1            65
Computing                       1            85
Multimedia                      1            75
BSW                             1            90
BBM                             1            20
Networking                      1            35

7 rows selected.
```

Figure 38 Transaction query 3

4. List students who scored above average in the 'Databases' module.

Functionality: List Students Who Have Scored Above the Average Score in the 'Databases' Module.

Script 4:

SELECT s.Student_Name, r.Total_Marks_Obtained

FROM Student s

```
JOIN Student_Module_Assessment sma ON s.Student_ID = sma.Student_ID
JOIN Result r ON sma.Result_Id = r.Result_Id
JOIN Module m ON sma.Module_Id = m.Module_Id
WHERE m.Module_Name = 'Databases'
  AND r.Total_Marks_Obtained > (
    SELECT AVG(r2.Total_Marks_Obtained)
    FROM Student_Module_Assessment sma2
    JOIN Result r2 ON sma2.Result_Id = r2.Result_Id
    JOIN Module m2 ON sma2.Module_Id = m2.Module_Id
    WHERE m2.Module_Name = 'Databases'
);
```

SELECT Clause:

s.Student_Name, r.Total_Marks_Obtained: Selects the student name and the total marks obtained.

FROM Clause:

Student s: Specifies the Student table as the main table.

JOIN Clauses:

JOIN Student_Module_Assessment sma ON s.Student_ID = sma.Student_ID: Joins the Student_Module_Assessment table to the Student table based on the Student_ID column.

JOIN Result r ON sma.Result_Id = r.Result_Id: Joins the Result table to the Student_Module_Assessment table based on the Result_Id column.

JOIN Module m ON sma.Module_Id = m.Module_Id: Joins the Module table to the Student_Module_Assessment table based on the Module_Id column.

WHERE Clause:

WHERE m.Module_Name = 'Databases': Filters the results to only include assessments for the 'Databases' module.

AND r.Total_Marks_Obtained > (SELECT AVG(r2.Total_Marks_Obtained) FROM ...): Further filters the results to only include students who have scored above the average score in the 'Databases' module. The subquery calculates the average score for the 'Databases' module.

Subquery:

SELECT Clause:

AVG(r2.Total_Marks_Obtained): Calculates the average of Total_Marks_Obtained.

FROM Clause:

Student_Module_Assessment sma2: Specifies the Student_Module_Assessment table as the main table for the subquery.

JOIN Clauses:

JOIN Result r2 ON sma2.Result_Id = r2.Result_Id: Joins the Result table to the Student_Module_Assessment table based on the Result_Id column.

JOIN Module m2 ON sma2.Module_Id = m2.Module_Id: Joins the Module table to the Student_Module_Assessment table based on the Module_Id column.

WHERE Clause:

WHERE m2.Module_Name = 'Databases': Filters the results to only include assessments for the 'Databases' module.

Purpose:

This query retrieves the names of students who have scored above the average score in the 'Databases' module. The output includes the student name and their total marks obtained, helping users to identify high-performing students in the 'Databases' module.

```
SQL> SELECT s.Student_Name, r.Total_Marks_Obtained
  2  FROM Student s
  3  JOIN Student_Module_Assessment sma ON s.Student_ID = sma.Student_ID
  4  JOIN Result r ON sma.Result_Id = r.Result_Id
  5  JOIN Module m ON sma.Module_Id = m.Module_Id
  6  WHERE m.Module_Name = 'Databases'
  7    AND r.Total_Marks_Obtained > (
  8      SELECT AVG(r2.Total_Marks_Obtained)
  9      FROM Student_Module_Assessment sma2
 10      JOIN Result r2 ON sma2.Result_Id = r2.Result_Id
 11      JOIN Module m2 ON sma2.Module_Id = m2.Module_Id
 12      WHERE m2.Module_Name = 'Databases'
 13  );

no rows selected
```

Figure 39 Transaction query 4

5. Display pass or fail status for students based on total marks in a module.

Functionality: Display Whether a Student Has Passed or Failed Based on Total Aggregate Marks in a Particular Module.

Script 5:

```
COLUMN Remarks FORMAT A7

SELECT s.Student_Name, m.Module_Name, r.Total_Marks_Obtained,

    CASE

      WHEN r.Total_Marks_Obtained >= 40 THEN 'Pass'

      ELSE 'Fail'

    END AS Remarks

FROM Student s

JOIN Student_Module_Assessment sma ON s.Student_ID = sma.Student_ID

JOIN Result r ON sma.Result_Id = r.Result_Id

JOIN Module m ON sma.Module_Id = m.Module_Id

WHERE m.Module_Name = 'Database Systems';
```

COLUMN Command:

COLUMN Remarks FORMAT A7: Sets the column width for the Remarks column to 7 characters to ensure proper display in SQL*Plus.

SELECT Clause:

s.Student_Name, m.Module_Name, r.Total_Marks_Obtained: Selects the student name, module name, and total marks obtained.

CASE WHEN r.Total_Marks_Obtained >= 40 THEN 'Pass' ELSE 'Fail' END AS Remarks: Uses a CASE statement to determine if the student has passed or failed based on their total marks. If the total marks are 40 or above, the remark is 'Pass'; otherwise, it is 'Fail'.

FROM Clause:

Student s: Specifies the Student table as the main table.

JOIN Clauses:

JOIN Student_Module_Assessment sma ON s.Student_ID = sma.Student_ID: Joins the Student_Module_Assessment table to the Student table based on the Student_ID column.

JOIN Result r ON sma.Result_Id = r.Result_Id: Joins the Result table to the Student_Module_Assessment table based on the Result_Id column.

JOIN Module m ON sma.Module_Id = m.Module_Id: Joins the Module table to the Student_Module_Assessment table based on the Module_Id column.

WHERE Clause:

WHERE m.Module_Name = 'Database Systems': Filters the results to only include assessments for the specified module ('Database Systems').

Purpose:

This query retrieves the names of students, the module name, their total marks obtained, and whether they have passed or failed based on their total aggregate marks in a particular module. The output includes the student name, module name, total marks obtained, and remarks ('Pass' or 'Fail'), helping users to identify students' performance in the specified module.

```
SQL> COLUMN Remarks FORMAT A6
SQL> SELECT s.Student_Name, m.Module_Name, r.Total_Marks_Obtained,
  2         CASE
  3             WHEN r.Total_Marks_Obtained >= 40 THEN 'Pass'
  4             ELSE 'Fail'
  5         END AS Remarks
  6  FROM Student s
  7  JOIN Student_Module_Assessment sma ON s.Student_ID = sma.Student_ID
  8  JOIN Result r ON sma.Result_Id = r.Result_Id
  9  JOIN Module m ON sma.Module_Id = m.Module_Id
 10  WHERE m.Module_Name = 'Databases';

STUDENT_NAME         MODULE_NAME               TOTAL_MARKS_OBTAINED REMARK
-------------------- ------------------------- -------------------- ------
Ram Shrestha         Databases                                   85 Pass
```

Figure 40 Transaction query 5

7. DUMP FILE CREATION AND DROP QUERIES

Dump file is the binary file which stores the data and schema of the database it is created for backup and recovery of the database.

7.1 DUMP FILE CREATION

```
Microsoft Windows [Version 10.0.26100.2894]
(c) Microsoft Corporation. All rights reserved.

C:\Users\DeLL>EXP system/dell OWNER=chewan FILE=C:\dumpfile\chewanregmi.dmp

Export: Release 11.2.0.2.0 - Production on Thu Jan 23 11:51:20 2025

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


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

About to export specified users ...
. exporting pre-schema procedural objects and actions
. exporting foreign function library names for user CHEWAN
. exporting PUBLIC type synonyms
. exporting private type synonyms
. exporting object type definitions for user CHEWAN
About to export CHEWAN's objects ...
. exporting database links
. exporting sequence numbers
. exporting cluster definitions
. about to export CHEWAN's tables via Conventional Path ...
. . exporting table                     ASSESSMENT          7 rows exported
. . exporting table                         MODULE          7 rows exported
. . exporting table                         NOTICE          7 rows exported
. . exporting table                        PROGRAM          7 rows exported
. . exporting table                      RESOURCES          7 rows exported
. . exporting table                         RESULT          7 rows exported
. . exporting table                        STUDENT          7 rows exported
. . exporting table                 STUDENT_MODULE          7 rows exported
. . exporting table      STUDENT_MODULE_ASSESSMENT          7 rows exported
. . exporting table       STUDENT_MODULE_RESOURCES          7 rows exported
. . exporting table         STUDENT_MODULE_TEACHER          7 rows exported
. . exporting table  STUDENT_MODULE_TEACHER_NOTICE          7 rows exported
. . exporting table                         TEACHER          7 rows exported
. exporting synonyms
. exporting views
. exporting stored procedures
. exporting operators
. exporting referential integrity constraints
. exporting triggers
. exporting indextypes
. exporting bitmap, functional and extensible indexes
. exporting posttables actions
. exporting materialized views
. exporting snapshot logs
. exporting job queues
. exporting refresh groups and children
. exporting dimensions
. exporting post-schema procedural objects and actions
. exporting statistics
Export terminated successfully without warnings.
```

Figure 41 Dump file

7.2 DROP QUERIES

Drop table tablename; command is used to delete tables.

```
SQL> DROP TABLE Student_Module_Teacher_Notice;

Table dropped.

SQL> DROP TABLE Student_Module_Assessment;

Table dropped.

SQL> DROP TABLE Student_Module_Teacher;

Table dropped.

SQL> DROP TABLE Student_Module_Resources;

Table dropped.

SQL> DROP TABLE Student_Module;

Table dropped.

SQL> DROP TABLE Result;

Table dropped.

SQL> DROP TABLE Notice;

Table dropped.

SQL> DROP TABLE Assessment;

Table dropped.

SQL> DROP TABLE Resources;

Table dropped.

SQL> DROP TABLE Teacher;

Table dropped.

SQL> DROP TABLE Module;

Table dropped.

SQL> DROP TABLE Student;

Table dropped.

SQL> DROP TABLE Program;

Table dropped.

SQL> select * from tab;

no rows selected
```

Figure 42 Drop queries

8. CRITICAL EVALUATION

This module helped in understanding of core database concepts, including entity-relationship diagrams (ERD), normalization, and data dictionaries. The systematic approach to normalization, from unnormalized form (UNF) to third normal form (3NF), ensured that the database design was free of redundancy and capable of maintaining data integrity. This process highlighted the importance of logical structuring, which is crucial for professional database development.

The coursework provided invaluable hands-on experience with Oracle SQL, to create users, tables, and implement relationships between entities. The practical implementation of database languages, including Data Definition Language (DDL), Data Query Language (DQL), and Data Manipulation Language (DML), strengthening ability to handle real-world database tasks. Writing and executing queries, such as retrieving module-specific notices and calculating student performance metrics, enhanced problem-solving abilities and confidence in working with complex data systems.

The design and development of a database system for the "E-Classroom" platform was an excellent real-world application of theoretical knowledge. Beginning with a clear understanding of business rules and requirements, progressing through each phase of database development defining attributes, creating relationships, and normalizing data. This structured approach ensured the database was efficient, scalable, and aligned with the functional needs of the system.

This module underscored the critical role databases play in software engineering and IT. The skills acquired during this module are directly transferable to professional settings, to design and manage databases in various industries.

In conclusion, this module has been useful in gaining knowledge of database systems and their application in the real world such as principles of analysing, designing, and implementing databases while emphasizing their importance in software engineering. The experience gained through this coursework has prepared to solve more advanced database challenges in the future and contributes significantly to my academic and professional growth.

9. REFERENCES

Kidd, C. (2024, December 02). *Data Normalization Explained: An In-Depth Guide*. Retrieved from Splunk Blogs: https://www.splunk.com/en_us/blog/learn/data-normalization.html

Metabase. (n.d.). *What is a data dictionary?* Retrieved from Metabase: https://www.metabase.com/glossary/data_dictionary

Motadata. (n.d.). *What is a Database Query?* Retrieved from Motadata: https://www.motadata.com/it-glossary/database-query/

Stryker, I. B. (2024, June 24). *What is an entity relationship diagram?* Retrieved from IBM: https://www.ibm.com/think/topics/entity-relationship-diagram