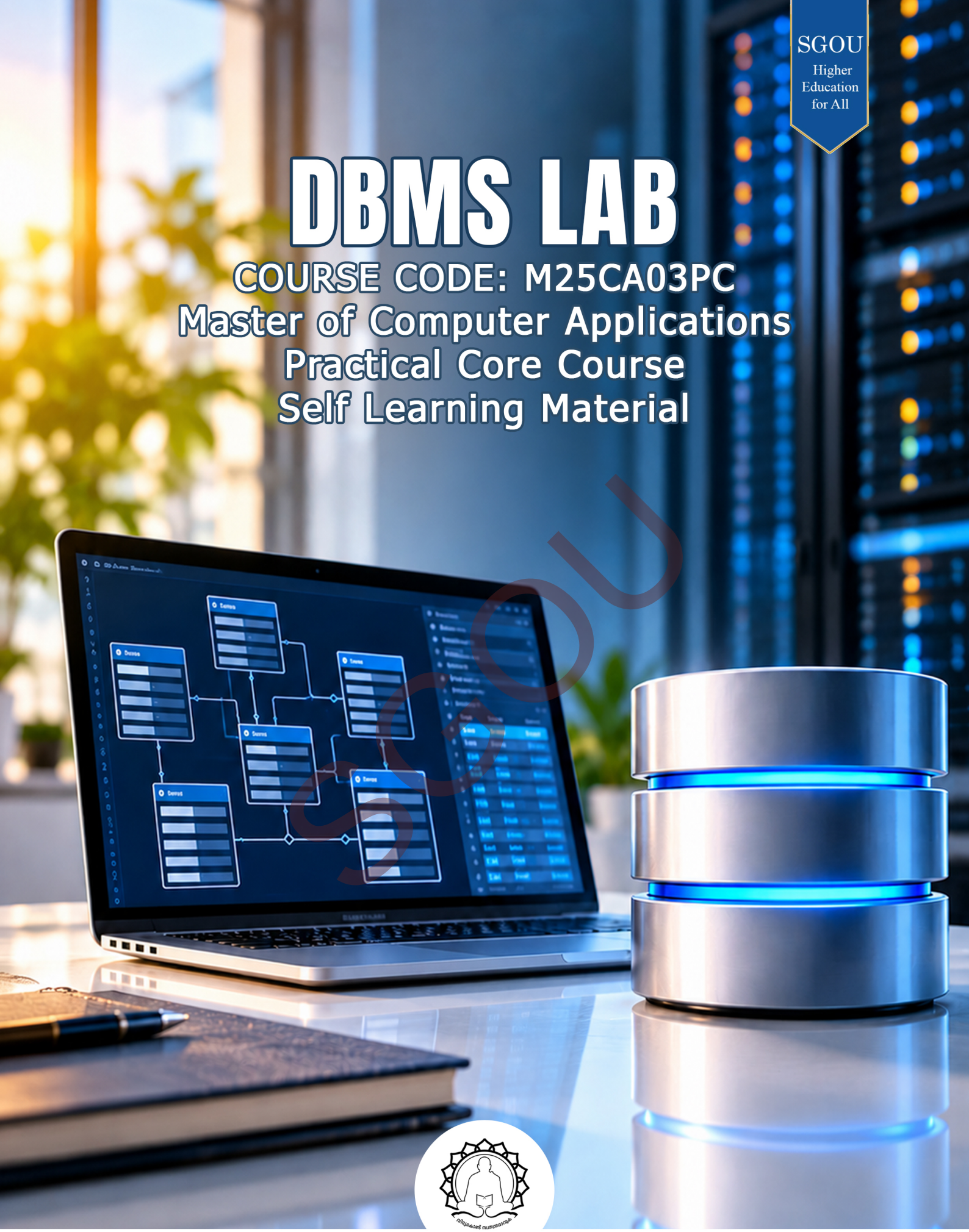
SGOU
Higher Education for All

DBMS LAB

COURSE CODE: M25CA03PC

Master of Computer Applications
Practical Core Course
Self Learning Material



SREENARAYANAGURU OPEN UNIVERSITY

SREENARAYANAGURU OPEN UNIVERSITY

The State University for Education, Training and Research in Blended Format, Kerala

SREENARAYANAGURU OPEN UNIVERSITY

SGOU

Vision

To increase access of potential learners of all categories to higher education, research and training, and ensure equity through delivery of high quality processes and outcomes fostering inclusive educational empowerment for social advancement.

Mission

To be benchmarked as a model for conservation and dissemination of knowledge and skill on blended and virtual mode in education, training and research for normal, continuing, and adult learners.

Pathway

Access and Quality define Equity.

DBMS Lab

Course Code: M25CA03PC

Semester - II

SGOU

Practical Core Course
Postgraduate Programme
Master of Computer Applications
Self Learning Material



SREENARAYANAGURU
OPEN UNIVERSITY

SREENARAYANAGURU OPEN UNIVERSITY

The State University for Education, Training and Research in Blended Format, Kerala

SREENARAYANAGURU
OPEN UNIVERSITY

DBMS LAB

Course Code: M25CA03PC
Semester- II
Practical Core Course
Master of Computer Applications

Academic Committee

Prof. (Dr.) Sabu M.K.
Dr. G. Santhoshkumar T
Prof. (Dr.) Vinod Chandra S.S.
Dr. Vinod P.
Dr.Lajish V.L.
Sreekanth M.S.
Dr. Vivek P.
Dr. Arun K.S.
Dr. Abdul Jebbar P.

Development of the Content

Shamin S.
Greeshma P.P.
Sreerekha V.K.
Dr. Kanitha D.K.
Subi Priya Laxmi S.B.N.
Aswathy V.S.

Review and Edit

Dr. Sony P.

Scrutiny

Shamin S.
Greeshma P.P.
Sreerekha V.K.
Dr. Kanitha D.K.
Subi Priya Laxmi S.B.N.
Aswathy V.S.

Cover Design

Jobin J.

Co-ordination

Dr. Gopakumar C.
Head, School of ICS

SGOU

Scan this QR Code for reading the SLM on a digital device.

Edition
September 2026

Copyright
© Sreenarayanaguru Open University

ISBN 978-81-181574-3-4

9 788118 157434

All rights reserved. No part of this work may be reproduced in any form, by mimeograph or any other means, without permission in writing from Sreenarayanaguru Open University. Printed and published on behalf of Sreenarayanaguru Open University by Registrar, SGOU, Kollam.

www.sgou.ac.m

Visit and Subscribe our Social Media Platforms

MESSAGE FROM VICE CHANCELLOR

Dear Learner,

It gives me immense pleasure and a deep sense of pride to warmly welcome you to Sreenarayanaguru Open University—a vibrant and progressive institution committed to transforming lives through inclusive, flexible, and high-quality education.

Established in September 2020 as a forward-looking initiative of the Government of Kerala, the University stands as a beacon of opportunity for learners seeking to advance their academic and professional aspirations through the open and distance learning mode. Guided by our foundational principle that "access and quality define equity," we are steadfast in our mission to democratize education while upholding uncompromising academic standards.

Our university is inspired by the timeless vision and philosophy of Sree Narayana Guru, whose ideals of knowledge, equality, and social transformation continue to guide our academic journey. His enduring legacy instils in us the responsibility to create an educational environment that empowers individuals, nurtures critical thinking, and contributes meaningfully to society.

Understanding the dynamic needs of contemporary learners, we have adopted a robust and learner-centric blended learning model, seamlessly integrating Self-Learning Materials, Academic Counselling, and Advanced Digital Learning Platforms. This holistic approach ensures flexibility without sacrificing academic depth, enabling you to learn at your own pace while remaining meaningfully connected to a vibrant academic ecosystem.

The Master of Computer Applications (MCA) programme you are embarking upon is carefully designed to position you at the forefront of the digital revolution. It uniquely blends strong theoretical foundations with practical, industry-oriented competencies. The curriculum emphasizes algorithmic thinking, system design, programming, database management, networking, and emerging technologies such as artificial intelligence, data science, and cloud computing.
What sets this programme apart is its forward-thinking design, which offers:

- Opportunities for skill enhancement aligned with industry needs
- Multidisciplinary learning pathways for broader intellectual development
- Exposure to innovative and emerging technology domains
- Multiple specialization options tailored to evolving career landscapes
- A strong focus on employability, entrepreneurship, and global career readiness
-

Our Self-Learning Materials are meticulously developed by experts, enriched with contemporary case studies, real-world applications, and practical insights to ensure clarity, engagement, and relevance. We are committed to equipping you not just with knowledge, but with the confidence and competence to excel in a competitive global environment.

At Sreenarayanaguru Open University, you are never alone in your learning journey. Our dedicated learner support system is designed to provide continuous academic guidance, timely assistance, and effective grievance redressal. We encourage you to actively engage with us, share your concerns, and make the most of the resources and support available to you.

As you begin this important phase of your academic journey, I urge you to embrace learning with curiosity, discipline, and determination. The world of technology is ever-evolving, and your willingness to adapt, innovate, and grow will define your success.

Remember, this is not just a programme—it is a pathway to transforming your future.
I wish you a fulfilling learning experience and a successful, inspiring career ahead.

Warm regards,



Prof. (Dr.) Jagathy Raj V. P.
Vice Chancellor

SGOU

Contents

Introduction **1**

CYCLE 1 SQL Fundamentals **6**

Experiment 1 Creating and Managing Tables 7

Experiment 2 Data Manipulation 16

Experiment 3 Basic SQL Queries 21

Experiment 4 Views 28

CYCLE 2 Advanced SQL queries, PL/SQL and NoSQL **31**

Experiment 1 EXISTS, NOT EXISTS Functions and Aggregate Functions with Grouping 32

Experiment 2 String Functions 37

Experiment 3 Cursors 40

Experiment 4 NoSQL Database 48

SGOU

Introduction

The Database Management Systems (DBMS) Lab is designed to provide students with practical knowledge of database concepts through hands-on experience. It introduces the fundamental operations involved in designing and managing relational databases, including creating and managing tables, defining constraints, and organizing data efficiently. Students learn how to perform data manipulation operations such as inserting, updating, deleting, and retrieving data using Structured Query Language (SQL). The lab also focuses on writing basic SQL queries to filter, sort, and display data, helping students build a strong foundation in database interaction.

In addition, the lab covers advanced database topics such as views, which provide logical data abstraction and security, and advanced SQL queries involving joins, subqueries, and aggregation functions for complex data analysis. Students are also introduced to PL/SQL for writing procedural programs within the database, enabling automation and enhanced control over data operations. Furthermore, the lab provides an overview of NoSQL databases, highlighting their importance in handling unstructured data and modern applications. Overall, the DBMS Lab equips students with essential skills required for real-world database development and management.

Objectives of the Lab

The primary objectives of the DBMS Lab are:

- To understand the fundamentals of database systems and relational models
- To learn how to create and manage database tables
- To develop skills in writing SQL queries for data manipulation and retrieval
- To gain practical knowledge of constraints, joins, and views
- To introduce procedural database programming using PL/SQL
- To explore advanced database concepts and NoSQL technologies
- To enhance problem-solving skills using database tools

Scope of the Lab

The DBMS Lab covers a wide range of practical topics essential for database application development:

- Designing relational database schemas
- Implementing tables with constraints

SGOU

- Performing CRUD operations (Create, Read, Update, Delete)
- Writing simple to complex SQL queries
- Creating and managing views
- Understanding and applying advanced SQL concepts such as joins, nested queries, and aggregation
- Writing PL/SQL programs for procedural database operations
- Introduction to NoSQL databases and their applications

This lab prepares students for real-world applications in software development, data analysis, and database administration.

Expectations From Students

Students are expected to:

1. Practice creating and managing tables by correctly defining schemas, data types, and constraints, ensuring a strong foundation in database design.
2. Perform data manipulation operations (INSERT, UPDATE, DELETE) accurately and understand their impact on database consistency.
3. Develop the ability to write and execute basic as well as advanced SQL queries, including joins, subqueries, and aggregate functions, with proper logic and syntax.
4. Understand and implement views and PL/SQL programs effectively, demonstrating the ability to handle procedural database operations and improve query efficiency.
5. Explore NoSQL concepts and tools, understand their differences from relational databases, and apply them in suitable scenarios while maintaining proper lab records and discipline.

Lab guidelines

Students must bring their lab manual/record for every session.

- Each experiment should be completed and verified by the instructor.
- Output screenshots or results must be recorded properly.
- Do not copy queries blindly; understand and implement them.
- Use meaningful table and column names.

SGOU

- Ensure correct syntax and formatting of SQL queries.
- Maintain discipline and silence in the lab environment.
- Back up your work regularly.

Tools and Software

The following tools and software may be used in the DBMS Lab:

- Database Systems:
  - MySQL
  - Oracle Database
  - PostgreSQL
- Development Tools:
  - SQL Developer
  - MySQL Workbench
  - pgAdmin
- Programming Support:
  - C Compiler (for integration or basics if required)
  - Command Prompt / Terminal
- NoSQL Tools:
  - MongoDB

Steps to Configure the Environment

1. Install Database Software

Download and install a suitable database system such as MySQL, Oracle, or PostgreSQL. Follow the installation wizard and choose default settings unless specific configurations are required.

2. Set Up the Database Server

Configure the database server by setting a username and password (e.g., root user). Ensure the server is running properly and note the port number (commonly 3306 for MySQL).

SGOU

3. Install a Database Client Tool

Install tools like MySQL Workbench, SQL Developer, or pgAdmin to interact with the database through a graphical interface.

4. Establish Database Connection

   Open the client tool and create a new connection using:

   - Host: localhost
   - Username: root (or configured user)
   - Password: the password you set

Test and confirm that the connection is successful.

5. Create a New Database (Schema)

   Create a database using the SQL command:

   CREATE DATABASE lab_db;

   Select the database for use:

   USE lab_db;

6. Verify Installation

   Run a simple query to ensure everything works:

   SELECT 'DBMS Lab Ready' AS Message;

7. Configure Environment Settings

   - ♦ Set SQL modes if required
   - ♦ Enable auto-commit or manual commit as needed
   - ♦ Adjust display/output settings in the client tool

8. Install NoSQL Tool (Optional)

   Install MongoDB for NoSQL experiments and verify using:

   mongosh

9. Organize Lab Workspace

   Create folders to store:

   - ♦ SQL scripts
   - ♦ Output screenshots

SGOU

- Lab records

10. Test Basic Operations

    Create a sample table and insert data to confirm proper operation:

    CREATE TABLE test (

    id INT,

    name VARCHAR(50)

    );

    INSERT INTO test VALUES (1, 'Sample');

    SELECT * FROM test;

This setup ensures that the system is ready for performing experiments on table creation, data manipulation, SQL queries, PL/SQL, and NoSQL operations in the DBMS Lab.

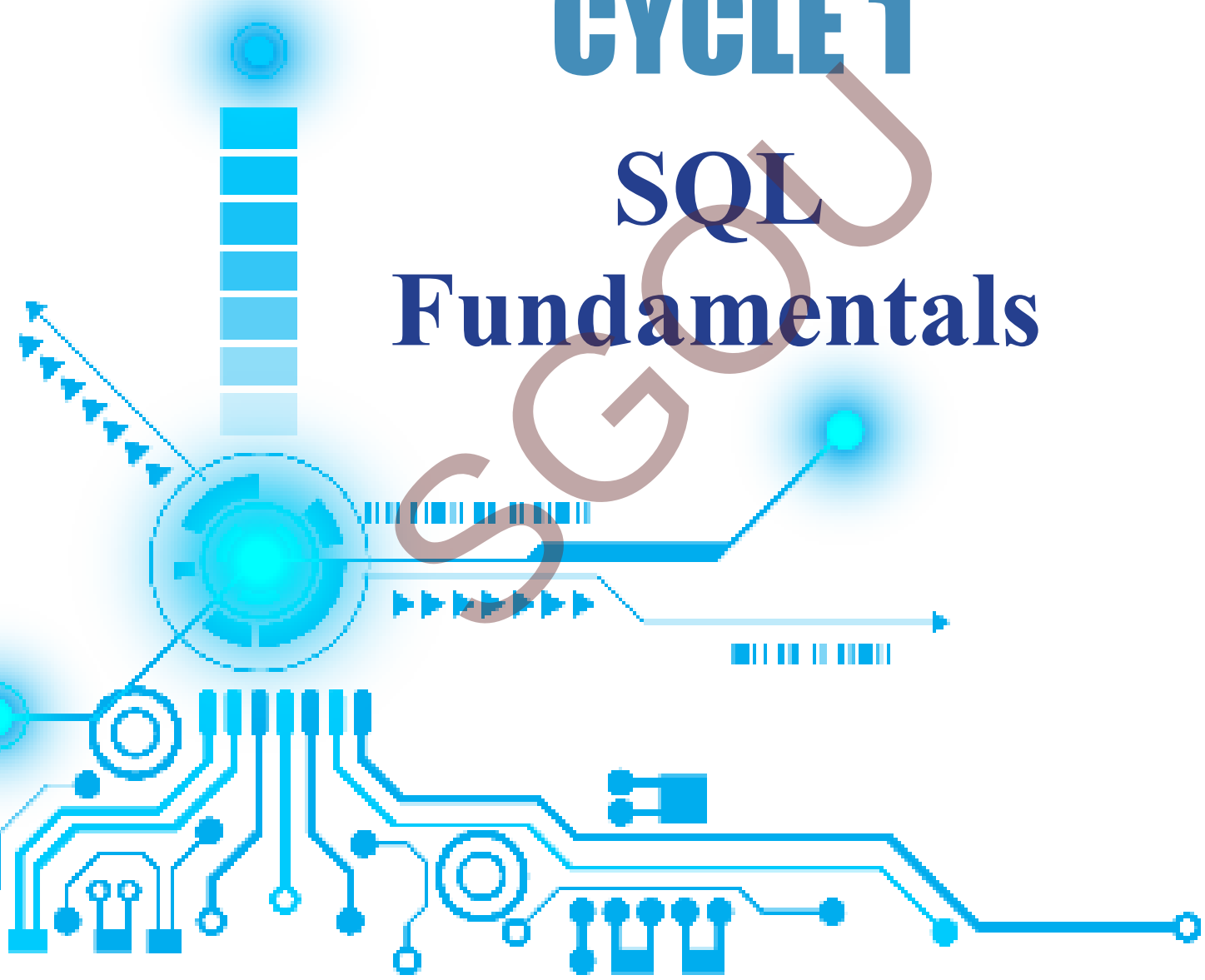
Code of Conduct

- Maintain discipline and respect lab rules.
- Do not damage lab equipment or software.
- Avoid unauthorized access to systems.
- Do not use mobile phones unnecessarily.
- Follow ethical practices while using data.
- Respect peers and instructors.
- Report any technical issues immediately.

SGOU

CYCLE 1

SQL Fundamentals



SGOU

Experiment 1

Creating and Managing Tables

Objectives

- To understand how to create tables using different data types.
- To apply constraints such as PRIMARY KEY, FOREIGN KEY, NOT NULL, and UNIQUE.
- To learn how to modify tables using ALTER TABLE.
- To perform operations like adding and dropping columns in a table.

1.1.1 Theory

A table in a relational database is used to store data in rows and columns. Each column has a specific data type that defines the kind of data it can store, such as:

- **INT** : Stores integer values
- **VARCHAR** : Stores variable-length text
- **DATE** : Stores date values

Constraints

Constraints are rules applied to columns to ensure data integrity:

- **PRIMARY KEY**: Uniquely identifies each record in a table.
- **FOREIGN KEY**: Establishes a relationship between two tables.
- **NOT NULL**: Ensures that a column cannot have NULL values.
- **UNIQUE**: Ensures all values in a column are different.

ALTER TABLE

The ALTER TABLE command is used to modify the structure of an existing table, such as:

- Adding new columns
- Deleting existing columns
- Modifying column data types

SGOU

1.1.2 Implementation

This experiment demonstrates the creation and modification of relational database tables using SQL. The examples cover commonly used data types such as INT, VARCHAR, and DATE, along with important constraints such as PRIMARY KEY, FOREIGN KEY, NOT NULL, and UNIQUE. The ALTER TABLE statement is also demonstrated for modifying the structure of an existing table.

I. Create a Table with INT, VARCHAR, and DATE as data types.

The CREATE TABLE statement is used to create a new table.

Syntax:

```
CREATE TABLE table_name (
    column1 INT,
    column2 VARCHAR(size),
    column3 DATE
);
```

Example 1:

In this example, the Student table contains an integer student ID, a variable-length character field for the student's name, and a date of birth.

Query:

```
CREATE TABLE Student (
    StudentID INT,
    Name VARCHAR(50),
    DateOfBirth DATE
);
```

Description:

The Student table is created with three columns. StudentID stores integer values, Name stores text values up to 50 characters, and DateOfBirth stores date values.

Output / Result:

Table Student is successfully created.

SGOU

II. Create a Table with PRIMARY KEY, FOREIGN KEY, NOT NULL and UNIQUE Constraints

Create a Table with PRIMARY KEY

A PRIMARY KEY uniquely identifies each record in a table. A primary key cannot contain duplicate values.

Example 2:

Query:

```
CREATE TABLE Student (
    StudentID INT PRIMARY KEY,
    Name VARCHAR(50),
    DateOfBirth DATE
);
```

Description:

Here, StudentID is defined as the primary key. Therefore, every student must have a unique StudentID.

Output / Result:

Table Student is created with StudentID as the primary key.

Create a Table with NOT NULL

The NOT NULL constraint ensures that a column must contain a value. It prevents NULL values from being inserted into the specified column.

Example 3:

Query:

```
CREATE TABLE Student (
    StudentID INT PRIMARY KEY,
    Name VARCHAR(50) NOT NULL,
    DateOfBirth DATE
);
```

Description:

The Name column is defined with the NOT NULL constraint. Therefore, a student record cannot be inserted without providing a name.

SGOU

Output / Result:

Table Student is created successfully with a mandatory Name column.

Create a Table with UNIQUE Constraint

The UNIQUE constraint ensures that duplicate values are not stored in the specified column.

Example 4:

Query:

```
CREATE TABLE Student (
    StudentID INT PRIMARY KEY,
    Name VARCHAR(50) NOT NULL,
    Email VARCHAR(100) UNIQUE,
    DateOfBirth DATE
);
```

Description:

The Email column is defined as UNIQUE. Therefore, two students cannot have the same email address.

Output / Result:

The Student table is created with a UNIQUE constraint on Email.

Create a Table with Multiple Constraints

A table can contain several constraints at the same time.

Example 5:

Query:

```
CREATE TABLE Student (
    StudentID INT PRIMARY KEY,
    Name VARCHAR(50) NOT NULL,
    Email VARCHAR(100) UNIQUE,
    DateOfBirth DATE
);
```

SGOU

Description:

In this table, StudentID uniquely identifies each student, Name cannot be left empty, Email cannot contain duplicate values, and DateOfBirth stores the student's date of birth.

Output / Result:

Table Student is successfully created with the specified columns and constraints.

Create a Table with FOREIGN KEY

A FOREIGN KEY establishes a relationship between two tables. It refers to a primary key or other suitable key in another table.

Example 6:

Query:

```
CREATE TABLE Course (
    CourseID INT PRIMARY KEY,
    CourseName VARCHAR(50) NOT NULL,
    StudentID INT,
    FOREIGN KEY (StudentID) REFERENCES Student(StudentID)
);
```

Description:

The StudentID column in the Course table is a foreign key referencing StudentID in the Student table. This establishes a relationship between the two tables.

Output / Result:

Table Course is created and linked with the Student table using a foreign key.

III. Alter Table and Add a New Column

The ALTER TABLE statement is used to modify the structure of an existing table. The ADD clause can be used to add a new column.

Syntax:

```
ALTER TABLE table_name
ADD column_name data_type;
```

SGOU

Example 7:

Query:

```
ALTER TABLE Student

ADD Phone VARCHAR(15);
```

Description:

A new column named Phone is added to the Student table. The column can store telephone numbers as character values.

Output / Result:

Column Phone is added successfully.

Add a New Column

A new column can also be added with an appropriate constraint.

Example 8:

Query:

```
ALTER TABLE Student

ADD City VARCHAR(30);
```

Description:

The City column is added to the Student table and can be used to store the student's city.

Output / Result:

Column City is added successfully.

Alter Table and Drop a Column

The DROP COLUMN clause is used to remove an existing column from a table.

Syntax:

```
ALTER TABLE table_name

DROP COLUMN column_name;
```

Example 9:

Query:

```
ALTER TABLE Student

DROP COLUMN Phone;
```

SGOU

Description:

The Phone column is permanently removed from the structure of the Student table.

Output / Result:

Column Phone is removed successfully.

View the Structure of a Table

The DESCRIBE statement is used in MySQL to display the structure of a table.

Syntax:

DESCRIBE table_name;

Example 10:

Query:

DESCRIBE Student;

Description:

The command displays information about the columns of the table, including their names, data types, whether NULL values are permitted, key information, default values, and other properties.

Output / Result:

The structure of the Student table is displayed.

Describe the Course Table

The structure of the Course table can also be examined using DESCRIBE.

Example 11:

Query:

DESCRIBE Course;

Description:

The command displays the columns and constraints associated with the Course table. It also helps verify that StudentID has been defined as a foreign key.

Output / Result:

The structure of the Course table is displayed.

Create a Table with All Major Constraints

The following example combines PRIMARY KEY, NOT NULL, UNIQUE, and FOREIGN KEY constraints.

SGOU

Example 12:

Query:

```
CREATE TABLE Enrollment (
    EnrollmentID INT PRIMARY KEY,
    StudentID INT NOT NULL,
    CourseCode VARCHAR(20) NOT NULL,
    Email VARCHAR(100) UNIQUE,
    EnrollmentDate DATE,
    FOREIGN KEY (StudentID) REFERENCES Student(StudentID)
);
```

Description:

EnrollmentID uniquely identifies each enrollment. StudentID and CourseCode are mandatory fields. Email must contain a unique value, while EnrollmentDate stores the date of enrollment. The foreign key connects the enrollment record with a student in the Student table.

Output / Result:

The Enrollment table is successfully created with multiple constraints.

Important SQL Concepts

- **PRIMARY KEY:** Uniquely identifies each record in a table.
- **FOREIGN KEY:** Establishes a relationship between two tables by referencing a key in another table.
- **NOT NULL:** Ensures that a column cannot contain NULL values.
- **UNIQUE:** Ensures that duplicate values are not stored in the specified column.
- **ALTER TABLE:** Modifies the structure of an existing table, such as adding or removing columns.
- **CREATE TABLE:** Creates a new table with specified columns, data types, and constraints.
- **DESCRIBE:** Displays the structure and properties of a table in MySQL.

SGOU

1.1.3 Lab Practice Questions

1. Create a table Employee with columns: EmpID, Name, Salary, and JoinDate.
2. Apply PRIMARY KEY and NOT NULL constraints to appropriate columns.
3. Create another table Department and link it with the Employee using a FOREIGN KEY.
4. Add a new column Email to the Employee table.
5. Remove the column Salary from the Employee table.
6. Modify the data type of a column using ALTER TABLE.
7. Add a UNIQUE constraint to a column.
8. Create a Books table with BookID, Title, and PublicationDate, then use ALTER TABLE to add a Price column.
9. Add an Author column of type VARCHAR(100) to the Books table using ALTER TABLE and verify the change using DESCRIBE.
10. Add a Publisher column of type VARCHAR(100) to the Books table and display the modified table structure using DESCRIBE.
11. Remove the Price column from the Books table using ALTER TABLE ... DROP COLUMN and verify the result using DESCRIBE.
12. Add an ISBN column to the Books table, display the updated table structure, and then remove the column using ALTER TABLE.

SGOU

Experiment 2

Data Manipulation

Objectives

- To understand and perform CRUD operations (Create, Read, Update, Delete).
- To learn how to insert records into a table.
- To update existing data using SQL queries.
- To delete records from a table.
- To retrieve and display data using SELECT queries.

1.2.1 Theory

Data manipulation is an important part of database management that involves adding, retrieving, modifying, and removing data stored in database tables. SQL provides a set of commands known as Data Manipulation Language (DML) to perform these operations. The four basic data manipulation operations are Create, Read, Update, and Delete, commonly referred to as CRUD operations. These operations enable users to efficiently manage records in a relational database. The INSERT, SELECT, UPDATE, and DELETE statements are used, respectively, to perform these operations.

1. Create – INSERT

The INSERT statement is used to add new records (rows) to a table.

Syntax

INSERT INTO table_name (column1, column2, column3, ...)

VALUES (value1, value2, value3, ...);

Example

Consider a table named Student:

Student_ID	Name	Age	Course
101	Anu	22	MCA
102	Rahul	23	MCA

SGOU

To insert a new student:

INSERT INTO Student (Student_ID, Name, Age, Course)

VALUES (103, 'Meera', 22, 'MCA');

The new record is added to the Student table.

2. Read – SELECT

The SELECT statement is used to retrieve or view data from a table.

Syntax

To retrieve all records:

SELECT * FROM table_name;

To retrieve specific columns:

SELECT column1, column2

FROM table_name;

Example

To display all records from the Student table:

SELECT * FROM Student;

To display only the student name and course:

SELECT Name, Course

FROM Student;

The SELECT statement does not modify the data; it only retrieves the required information.

3. Update – UPDATE

The UPDATE statement is used to modify existing records in a table.

Syntax

UPDATE table_name

SET column1 = value1, column2 = value2

WHERE condition;

The WHERE clause specifies which record or records should be updated.

SGOU

Example

To change the course of the student whose ID is 103:

```
UPDATE Student
SET Course = 'MCA Data Science'
WHERE Student_ID = 103;
```

The course of student 103 is changed from MCA to MCA Data Science.

Note: The WHERE clause is important. Without it, the specified column will be updated for all records in the table.

4. Delete – DELETE

The DELETE statement is used to remove existing records from a table.

Syntax

```
DELETE FROM table_name
WHERE condition;
```

Example

To delete the record of the student whose ID is 103:

```
DELETE FROM Student
WHERE Student_ID = 103;
```

The record with Student_ID = 103 is removed from the table.

Note: If WHERE is omitted, all records in the table will be deleted:

```
DELETE FROM Student;
```

The table structure remains, but all its records are removed.

Summary

CRUD Operation	SQL Command	Purpose
Create	INSERT	Add new records
Read	SELECT	Retrieve records
Update	UPDATE	Modify existing records
Delete	DELETE	Remove records

These four operations form the basic mechanism for manipulating data in a relational database.

SGOU

1.2.2 Implementation

Create a table named Student with the following attributes:

- Student_ID – Integer
- Name – Variable-length character
- Age – Integer
- Course – Variable-length character

Perform the following CRUD operations on the Student table:

1. Insert at least three student records.

```
CREATE TABLE Student (
    Student_ID INT PRIMARY KEY,
    Name VARCHAR(50),
    Age INT,
    Course VARCHAR(50)
);
```

Insert records

```
INSERT INTO Student (Student_ID, Name, Age, Course)
VALUES (101, 'Anu', 22, 'MCA');
INSERT INTO Student (Student_ID, Name, Age, Course)
VALUES (102, 'Rahul', 23, 'MCA');
INSERT INTO Student (Student_ID, Name, Age, Course)
VALUES (103, 'Meera', 22, 'MCA');
```

2. Display all the records in the table.

```
SELECT * FROM Student;
```

3. Display only the Name and Course of all students.

```
SELECT Name, Course
FROM Student;
```

SGOU

4. Update the course of the student whose Student_ID is 102.

   UPDATE Student

   SET Course = 'MCA Data Science'

   WHERE Student_ID = 102;

5. Delete the record of the student whose Student_ID is 103.

   DELETE FROM Student

   WHERE Student_ID = 103;

6. Display the records after performing the update and delete operations.

   SELECT * FROM Student;

Expected Final Output

Student_ID	Name	Age	Course
101	Anu	22	MCA
102	Rahul	23	MCA Data Science

1.2.3 Lab Practice Questions

Insert 5 records into the Employee table with different departments.

1. Display only employees working in the IT department.
2. Update the salary of all employees in the HR department by 10%.
3. Delete employees whose salary is less than 30000.
4. Retrieve only the Name and Salary columns.
5. Update the department of an employee using their EmpID.
6. Delete all records from the table without removing the table structure.
7. Insert a record with missing optional fields and observe the result.
8. Write a query to display employees with salary greater than 35000.

SGOU

Experiment 3

Basic SQL Queries

Objectives

- To understand and use aliases in SQL queries.
- To retrieve data using SELECT * and DISTINCT.
- To apply the WHERE clause for filtering records.
- To use the LIKE operator for pattern matching.
- To perform set operations (UNION, INTERSECT, etc.).
- To implement different types of JOINS.
- To construct and execute nested subqueries.

1.3.1 Theory

SQL (Structured Query Language) is used to interact with relational databases. Basic SQL queries allow users to retrieve, filter, and manipulate data efficiently.

1. Aliases

Aliases provide temporary names for columns or tables using the AS keyword, improving readability.

2. SELECT and DISTINCT

SELECT * retrieves all columns from a table.

DISTINCT eliminates duplicate values.

3. WHERE Clause

Used to filter records based on conditions.

4. LIKE Operator

Used for pattern matching:

- % → Represents multiple characters
- _ → Represents a single character

SGOU

5. Set Operations

- UNION → Combines results and removes duplicates
- INTERSECT → Returns common records
- EXCEPT → Returns records from one table that are not in another

6. Joins

Used to combine data from multiple tables:

- INNER JOIN
- LEFT JOIN
- RIGHT JOIN

7. Subqueries

A subquery is a query inside another query, used for complex data retrieval.

1.3.2 Implementation

Example 1: Create Tables

```
CREATE TABLE Employee (
  EmpID INT PRIMARY KEY,
  Name VARCHAR(50),
  DepartmentID INT,
  Salary INT
);
CREATE TABLE Department (
  DepartmentID INT PRIMARY KEY,
  DepartmentName VARCHAR(50)
);
```

SGOU

OUTPUT

Column	Data Type	Meaning
EmpID	INT	Employee ID
Name	VARCHAR(50)	Employee name
DepartmentID	INT	Department ID
Salary	INT	Employee salary

Column	Data Type	Meaning
DepartmentID	INT	Department ID
DepartmentName	VARCHAR(50)	Name of the department

Example 2: Insert Sample Data

```
INSERT INTO Employee VALUES (101, 'Anu', 1, 30000);
INSERT INTO Employee VALUES (102, 'Rahul', 2, 40000);
INSERT INTO Employee VALUES (103, 'Meera', 1, 35000);
INSERT INTO Department VALUES (1, 'HR');
INSERT INTO Department VALUES (2, 'IT');
```

OUTPUT

Employee Table

EmpID	Name	DepartmentID	Salary
101	Anu	1	30000
102	Rahul	2	40000
103	Meera	1	35000

Department Table

DepartmentID	DepartmentName
1	HR
2	IT

SGOU

Example 3: Using SELECT * and DISTINCT

SELECT * FROM Employee;

Output:

EmpID	Name	DepartmentID	Salary
101	Anu	1	30000
102	Rahul	2	40000
103	Meera	1	35000

SELECT DISTINCT DepartmentID FROM Employee;

Output

DepartmetID
1
2

Example 4: Using Aliases

SELECT Name AS Employee_Name, Salary AS Employee_Salary

FROM Employee;

Output

Columns displayed with alias names.

Output

Employee_Name	Employee_Salary
Anu	30000
Rahul	40000
Meera	35000

Example 5: WHERE Clause

SELECT * FROM Employee WHERE Salary > 30000;

Output

EmpID	Name	DepartmentID	Salary
102	Rahul	2	40000
103	Meera	1	35000

SGOU

Example 6: LIKE Operator

```
SELECT * FROM Employee
WHERE Name LIKE 'A%';
```

Output

EmpID	Name	DepartmentID	Salary
101	Anu	1	30000

Example 7: Set Operations

```
SELECT DepartmentID FROM Employee
UNION
SELECT DepartmentID FROM Department;
```

Output

DepartmentID
1
2

Example 8: JOIN Operation

```
SELECT Employee.Name, Department.DepartmentName
FROM Employee
INNER JOIN Department
ON Employee.DepartmentID = Department.DepartmentID;
```

Output

Name	DepartmentName
Anu	HR
Rahul	IT
Meera	HR

SGOU

Example 9: Nested Subquery

```
SELECT Name
FROM Employee
WHERE DepartmentID = (
  SELECT DepartmentID
  FROM Department
  WHERE DepartmentName = 'HR'
);
```

Output

Name
Anu
Meera

1.3.3 Lab Practice Questions

I. Create a Product table with the following fields: ProductID, ProductName, Category, and Price. Insert suitable records into the table and write SQL queries to:

1. Display all product details using an alias for the table name.
2. Retrieve distinct prices from the Product table.
3. Display products whose names end with 'n' using LIKE.
4. Find products with prices between 500 and 2000.
5. Use UNION to combine two result sets from the Product table.
6. Write a subquery to find products whose price is greater than the average price.
7. Display the category names along with the number of products in each category.

II. Create a Student table with the fields StudentID, StudentName, Course, and Mark. Insert suitable records into the table and write SQL queries to:

1. Display all student details using a table alias.

SGOU

2. Retrieve distinct marks from the Student table.
3. Display students whose names start with 'A' using LIKE.
4. Find students whose marks are between 60 and 90.
5. Use UNION to combine two result sets from the Student table.
6. Write a subquery to find students who scored more than the average mark.
7. Display each course along with the number of students in that course.

SGOU

Experiment 4

Views

Objectives

- To understand the concept and purpose of views in SQL.
- To create a view with selected columns from a table.
- To modify an existing view by adding additional columns.
- To delete (drop) a view from the database.
- To understand how views improve data security and simplicity.

1.4.1 Theory

A view is a virtual table in SQL that is based on the result of a query. It does not store data physically but displays data retrieved from one or more tables.

Features of Views

- Simplifies complex queries.
- Enhances security by restricting access to specific columns.
- Provides abstraction by hiding implementation details.

Types of Views

- **Simple View**: A view based on a single table.
- **Complex View**: A view based on multiple tables with joins or functions

Common Operations on Views

- **CREATE VIEW** → To create a new view
- **ALTER VIEW / CREATE OR REPLACE VIEW** → To modify a view
- **DROP VIEW** → To delete a view

SGOU

1.4.2 Implementation

1. **Create a Base Table**

```
CREATE TABLE Employee (
  EmpID INT PRIMARY KEY,
  Name VARCHAR(50),
  Department VARCHAR(50),
  Salary INT
);
```

2. **Insert Sample Data**

```
INSERT INTO Employee VALUES (101, 'Anu', 'HR', 30000);
INSERT INTO Employee VALUES (102, 'Rahul', 'IT', 40000);
INSERT INTO Employee VALUES (103, 'Meera', 'Finance', 35000);
```

3. **Create a View (Specific Columns)**

```
CREATE VIEW Employee_View AS
SELECT Name, Department
FROM Employee;
```

Output

View Employee_View is created successfully showing only Name and Department.

4. **Display Data from View**

```
SELECT * FROM Employee_View;
```

Output

Name	Department
Anu	HR
Rahul	IT
Meera	Finance

SGOU

5. **Modify the View (Add Column)**

```
CREATE OR REPLACE VIEW Employee_View AS
SELECT Name, Department, Salary
FROM Employee;
```

(Note: In some SQL systems, use ALTER VIEW instead.)

Output

The view is updated successfully to include Salary column.

6. **Display Modified View**

```
SELECT * FROM Employee_View;
```

Output

Name	Department	Salary
Anu	HR	30000
Rahul	IT	40000
Meera	Finance	35000

7. **Drop the View**

```
DROP VIEW Employee_View;
```

Output

The view is deleted successfully.

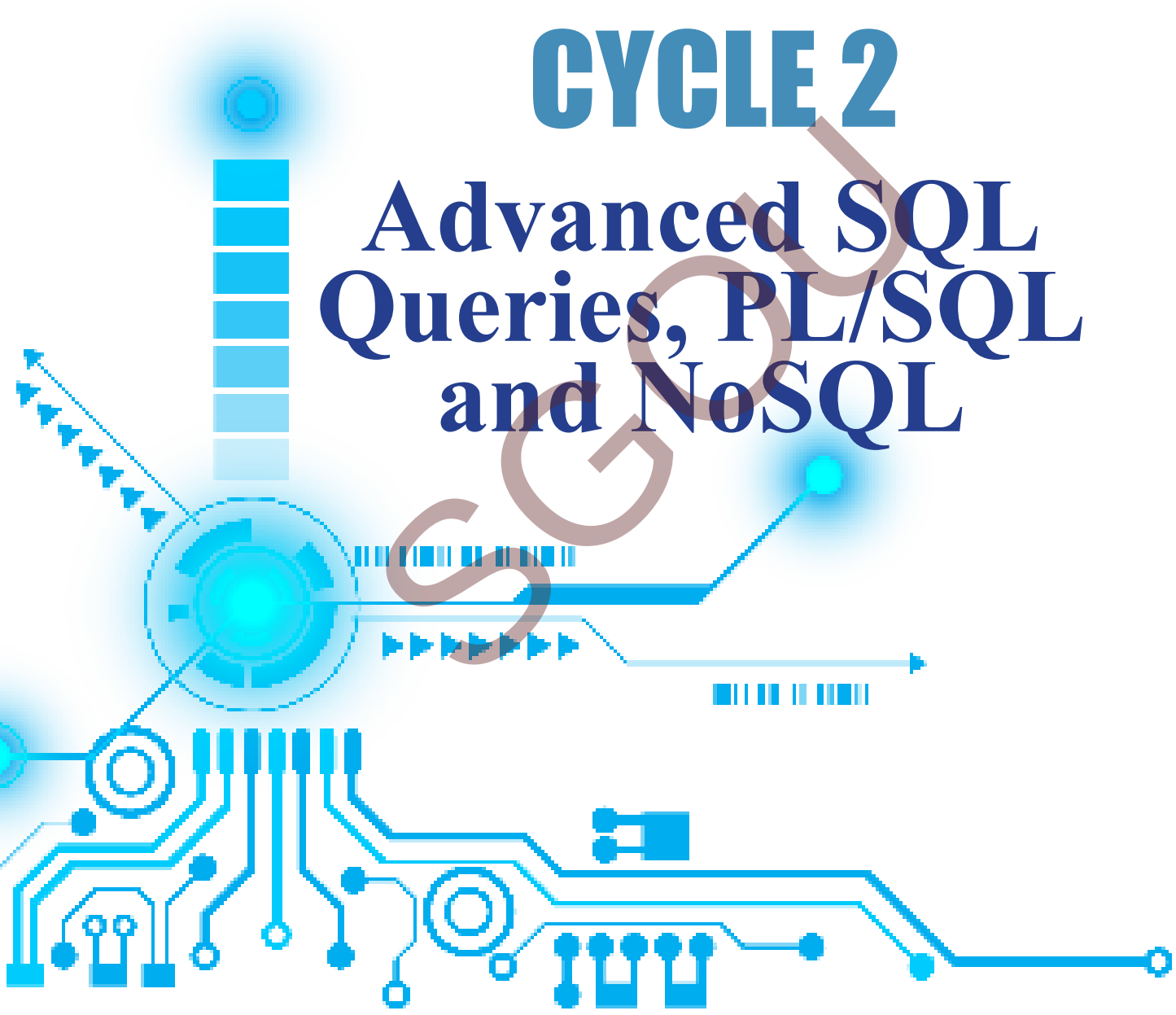
1.4.3 Lab Practice Questions

1. Create a view that displays only employee names and salaries.
2. Modify the view to include department details.
3. Create a view based on two tables using aJOIN.
4. Drop an existing view from the database.
5. Write a query to display data from a view.
6. Create multiple views for different departments.
7. Write a query to create a complex view using conditions.

SGOU

CYCLE 2

Advanced SQL Queries, PL/SQL and NoSQL



SGOU

Experiment 1

EXISTS, NOT EXISTS Functions and Aggregate Functions with Grouping

Objectives

- To understand and use EXISTS and NOT EXISTS functions.
- To retrieve data based on the presence or absence of related records.
- To apply aggregate functions such as SUM, AVG, and COUNT.
- To group data using the GROUP BY clause.
- To analyze summarized data for decision-making.

2.1.1 Theory

1. EXISTS Operator

The EXISTS operator is used to check whether a subquery returns any records.

- Returns TRUE if the subquery returns at least one row.

2. NOT EXISTS Operator

- Returns TRUE if the subquery returns no records.
- Used to find unmatched data.

3. Aggregate Functions

Aggregate functions perform calculations on multiple rows:

- **SUM()** → Total value
- **AVG()** → Average value
- **COUNT()** → Number of records

4. GROUP BY Clause

- Used to group rows that have the same values.
- Often used with aggregate functions.

SGOU

2.1.2 Implementation

// Creating Table

```
CREATE TABLE Employee (
  EmpID INT PRIMARY KEY,
  Name VARCHAR(50),
  DepartmentID INT
);
CREATE TABLE Department (
  DepartmentID INT PRIMARY KEY,
  DepartmentName VARCHAR(50)
);
CREATE TABLE Product (
  ProductID INT PRIMARY KEY,
  ProductName VARCHAR(50),
  Category VARCHAR(50)
);
CREATE TABLE Orders (
  OrderID INT PRIMARY KEY,
  ProductID INT,
  Amount INT
);
```

// Insert Sample Data

```
INSERT INTO Employee VALUES (1, 'A', 10);
INSERT INTO Employee VALUES (2, 'B', 10);
INSERT INTO Employee VALUES (3, 'C', 10);
INSERT INTO Employee VALUES (4, 'D', 10);
```

SGOU

```
INSERT INTO Employee VALUES (5, 'E', 10);
INSERT INTO Employee VALUES (6, 'F', 10);
INSERT INTO Department VALUES (10, 'HR');
INSERT INTO Product VALUES (101, 'Pen', 'Stationery');
INSERT INTO Product VALUES (102, 'Notebook', 'Stationery');
INSERT INTO Product VALUES (103, 'Laptop', 'Electronics');
INSERT INTO Orders VALUES (1, 101, 50);
INSERT INTO Orders VALUES (2, 101, 70);
INSERT INTO Orders VALUES (3, 102, 30);
```

1. Write a SQL query that uses the EXISTS function to find all employees who work in departments that have more than 5 employees

// EXISTS Query

```
SELECT *
FROM Employee E
WHERE EXISTS (
  SELECT 1
  FROM Employee
  WHERE DepartmentID = E.DepartmentID
  GROUP BY DepartmentID
  HAVING COUNT(*) > 5
);
```

SGOU

Output

Displays employees working in departments with more than 5 employees.

EmpID	Name	DepartmentID
1	A	10
2	B	10
3	C	10
4	D	10
5	E	10
6	F	10

2. Create a SQL query using the NOT EXISTS function to list all products that have not been sold in any order.

```
SELECT *
FROM Product P
WHERE NOT EXISTS (
  SELECT 1
  FROM Orders O
  WHERE O.ProductID = P.ProductID
);
```

Output

ProductID	ProductName	Category
103	Laptop	Electronics

3. Write a SQL query that calculates the total sales, average sales, and number of sales transactions for each product category using aggregate functions.

```
SELECT P.Category,
    SUM(O.Amount) AS Total_Sales,
    AVG(O.Amount) AS Average_Sales,
    COUNT(O.OrderID) AS Number_of_Transactions
FROM Product P
JOIN Orders O ON P.ProductID = O.ProductID
GROUP BY P.Category;
```

SGOU

Output

Category	Total_Sales	Average_Sales	Number_of_Transactions
Stationery	150	50	3

2.1.3 Lab Practice Questions

1. Write a query using EXISTS to find departments with more than 3 employees.
2. Use NOT EXISTS to find employees who are not assigned to any department.
3. Calculate total salary per department using GROUP BY.
4. Find average order value for each product.
5. Count the number of employees in each department.
6. Compare EXISTS and IN with examples.
7. Find the category with the highest sales.

SGOU

Experiment 2

String Functions

Objectives

- To understand the use of string functions in SQL.
- To extract specific portions of text using the SUBSTRING() function.
- To manipulate and display formatted string data.
- To apply string functions to real-world data such as names and phone numbers.

2.2.1 Theory

String functions in SQL are used to manipulate textual data stored in database tables.

Common String Functions

- **SUBSTRING() / SUBSTR()** → Extracts a part of a string
- **LENGTH()** → Returns the length of a string
- **UPPER() / LOWER()** → Converts text to upper/lowercase
- **CONCAT()** → Combines two or more strings

SUBSTRING Syntax:

```
SUBSTRING(column_name, start_position, length)
```

- start_position → Position at which extraction begins
- length → Number of characters to extract

2.2.2 Implementation

1. Write a SQL query that extracts and displays the first three characters of each employee's last name using the SUBSTRING function.

SGOU

Create Employee Table

```
CREATE TABLE Employee (
  EmpID INT PRIMARY KEY,
  FirstName VARCHAR(50),
  LastName VARCHAR(50)
);
```

Insert Sample Data

```
INSERT INTO Employee VALUES (1, 'Anu', 'Nair');
INSERT INTO Employee VALUES (2, 'Rahul', 'Kumar');
INSERT INTO Employee VALUES (3, 'Meera', 'Menon');
```

Extract First Three Characters of Last Name

```
SELECT LastName,
    SUBSTRING(LastName, 1, 3) AS First_Three_Chars
FROM Employee;
```

Output

LastName	First_Three_Chars
Nair	Nai
Kumar	Kum
Menon	Men

2. Formulate a SQL query that utilizes string functions to extract and display the last four characters of each customer's phone number from the customer table.

Create Customer Table

```
CREATE TABLE Customer (
  CustomerID INT PRIMARY KEY,
  CustomerName VARCHAR(50),
  Phone VARCHAR(15)
);
```

SGOU

Insert Sample Data

```
INSERT INTO Customer VALUES (101, 'Arjun', '9876543210');
INSERT INTO Customer VALUES (102, 'Neha', '9123456780');
INSERT INTO Customer  VALUES (103, 'Ravi', '9988776655');
```

Extract the Last Four Digits of the Phone Number

```
SELECT CustomerName,
    SUBSTRING(Phone, LENGTH(Phone) - 3, 4) AS Last_Four_Digits
FROM Customer;
```

(Note: In some databases, use LEN() instead of LENGTH().)

Output

CustomerName	Last_Four_Digits
Arjun	3210
Neha	6780
Ravi	6655

2.2.3 Lab Practice Questions

1. Extract the first two characters of employee first names.
2. Display the last three characters of customers' names.
3. Convert all employee names to uppercase.
4. Find the length of each customer's phone number.
5. Concatenate first name and last name into a full name.
6. Extract middle characters from a string.
7. Replace part of a string using string functions.
8. Write a query to display phone numbers in a masked format (e.g., ******3210).
9. Compare SUBSTRING() and LEFT()/RIGHT() functions.
10. Write a query to extract domain names from email addresses.

SGOU

Experiment 3

Cursors

Objectives

- To understand the concept of cursors in PL/SQL.
- To distinguish between implicit and explicit cursors.
- To implement an explicit cursor to retrieve employee names and salaries.
- To fetch and process records from a cursor.
- To understand the use of cursors for processing multiple records.

2.3.1 Theory

A cursor is a temporary memory area used by PL/SQL to process the rows returned by a SQL statement. When a query returns multiple rows, a cursor allows the program to retrieve and process each row individually.

Cursors are useful when the result of a query contains multiple records that need to be processed one at a time. They provide control over the retrieval and processing of records within a PL/SQL program.

2.3.1.1 Types of Cursors

PL/SQL provides two main types of cursors: implicit cursors and explicit cursors.

Implicit Cursor:

An implicit cursor is automatically created and managed by Oracle when a SQL statement is executed. It is commonly used with statements such as INSERT, UPDATE, DELETE, and SELECT INTO, statements that retrieve a single row.

Explicit Cursor:

An explicit cursor is declared by the programmer when a query returns multiple rows that need to be processed individually. The programmer controls the cursor by declaring, opening, fetching from, and closing it.

2.3.1.2 Explicit Cursor

An explicit cursor is associated with a SELECT statement that may return multiple rows. It allows the programmer to retrieve and process the records one at a time.

SGOU

The basic steps involved in using an explicit cursor are:

1. Declare the cursor with a SELECT statement.
2. Open the cursor to execute the query.
3. Fetch records from the cursor one at a time.
4. Process the retrieved records.
5. Close the cursor after all records have been processed.

The general structure of an explicit cursor is:

```
DECLARE
      CURSOR cursor_name IS
                SELECT column1, column2
               FROM table_name;
        variable1 datatype;
     variable2 datatype;
BEGIN
    OPEN cursor_name;
    LOOP
        FETCH cursor_name INTO variable1, variable2;
        EXIT WHEN cursor_name%NOTFOUND;
        -- Process the retrieved data
    END LOOP;
    CLOSE cursor_name;
END;
/
```

SGOU

2.3.1.3 Cursor Lifecycle

An explicit cursor follows a sequence of operations during its execution:

Declare → Open → Fetch → Process → Close

The cursor is first declared with the required query. The OPEN statement executes the query and makes the result available for processing. The FETCH statement retrieves one row at a time into PL/SQL variables. The retrieved data can then be processed. After all the records are processed, the cursor is closed.

2.3.1.4 Fetching Data from a Cursor

The FETCH statement is used to retrieve the next available row from an open cursor and store the values in PL/SQL variables.

For example:

FETCH emp_cursor INTO v_name, v_salary;

Here, the employee name and salary from the retrieved row are stored in the variables v_name and v_salary.

The %NOTFOUND cursor attribute indicates whether the most recent FETCH statement failed to retrieve a row. It becomes TRUE when a fetch operation does not retrieve a row.

For example:

EXIT WHEN emp_cursor%NOTFOUND;

This statement terminates the loop when no more rows are available. It should be placed immediately after the FETCH statement to prevent processing invalid or unchanged variable values.

2.3.1.5 Advantages of Cursors

Cursors are useful when multiple records returned by a query need to be processed individually.

- They allow records to be retrieved and processed one at a time.
- They provide better control over the processing of multiple records.
- They allow different operations to be performed on individual records.
- They are useful when each record requires separate processing within a PL/SQL program.
- They can be used in procedures and other PL/SQL program units for processing database records.

SGOU

2.3.2 Implementation

Implement an explicit cursor to retrieve and display employee names and salaries.

1. Create Employee Table

Create an employee table to store employee details such as employee ID, employee name, and salary.

```
CREATE TABLE employee (
    emp_id NUMBER PRIMARY KEY,
    emp_name VARCHAR2(50),
    salary NUMBER(10,2)
);
```

2. Insert Sample Employee Data

Insert sample employee records into the employee table.

```
INSERT INTO employee VALUES (101, 'Anu', 35000);
INSERT INTO employee VALUES (102, 'Rahul', 42000);
INSERT INTO employee VALUES (103, 'Meera', 50000);
INSERT INTO employee VALUES (104, 'Arjun', 38000);
COMMIT;
```

The COMMIT statement saves the inserted records in the database.

3. Declare an Explicit Cursor

The following program declares an explicit cursor to retrieve employee names and salaries from the employee table.

```
SET SERVEROUTPUT ON;
DECLARE
        CURSOR emp_cursor IS
                SELECT emp_name, salary
```

SGOU

```
            FROM employee
            ORDER By emp_id;
    v_name employee.emp_name%TYPE;
    v_salary employee.salary%TYPE;
BEGIN
    OPEN emp_cursor;
```

The cursor emp_cursor is declared with a SELECT statement. The variables v_name and v_salary are used to store the values retrieved from the cursor.

4. Fetch Data from the Cursor

The FETCH statement retrieves the employee name and salary one record at a time.

```
LOOP
  FETCH emp_cursor INTO v_name, v_salary;
  EXIT WHEN emp_cursor%NOTFOUND;
  DBMS_OUTPUT.PUT_LINE(
    'Employee Name: ' || v_name ||
     ', Salary: ' || v_salary
  );
END LOOP;
```

The loop continues to fetch and display records until there are no more records in the cursor. The %NOTFOUND attribute identifies the end of the result set.

5. Close the Cursor

After all the records have been processed, the cursor is closed.

```
 CLOSE emp_cursor;
END;
/
```

Closing the cursor releases the resources associated with it.

SGOU

Complete Program

The complete PL/SQL program for retrieving and displaying employee names and salaries using an explicit cursor is given below.

```
SET SERVEROUTPUT ON;
DECLARE
   CURSOR emp_cursor IS
      SELECT emp_name, salary
      FROM employee
      ORDER By emp_id;
   v_name employee.emp_name%TYPE;
   v_salary employee.salary%TYPE;
BEGIN
   OPEN emp_cursor;
   LOOP
      FETCH emp_cursor INTO v_name, v_salary;
      EXIT WHEN emp_cursor%NOTFOUND;
      DBMS_OUTPUT.PUT_LINE(
         'Employee Name: ' || v_name ||
         ', Salary: ' || v_salary
      );
   END LOOP;
   CLOSE emp_cursor;
END;
/
```

SGOU

Implement a Method to Fetch Data from a Cursor

A stored procedure can be used as a reusable method to fetch and display employee data using a cursor.

```
CREATE OR REPLACE PROCEDURE display_employee_data IS
    CURSOR emp_cursor IS
        SELECT emp_name, salary
        FROM employee
        ORDER By emp_id;
    v_name employee.emp_name%TYPE;
    v_salary employee.salary%TYPE;
BEGIN
    OPEN emp_cursor;
    LOOP
        FETCH emp_cursor INTO v_name, v_salary;
        EXIT WHEN emp_cursor%NOTFOUND;
        DBMS_OUTPUT.PUT_LINE(
            'Employee Name: ' || v_name ||
            ', Salary: ' || v_salary
        );
    END LOOP;
    CLOSE emp_cursor;
END;
/
```

SGOU

Execute the procedure:

EXEC display_employee_data;

The procedure provides a reusable way to fetch and display employee data without rewriting the cursor logic.

Sample Output

Employee Name: Anu, Salary: 35000
Employee Name: Rahul, Salary: 42000
Employee Name: Meera, Salary: 50000
Employee Name: Arjun, Salary: 38000

The employee names and salaries are successfully retrieved and displayed using an explicit cursor.

2.3.3 Lab Practice Questions

1. Modify the cursor program to display employees whose salary are greater than 40000.

2. Write a PL/SQL program using an explicit cursor to calculate and display the total salary of all employees.

3. Use the %ROWCOUNT cursor attribute to display the number of employee records processed.

4. Create a stored procedure using an explicit cursor to display employee details.

5. Modify the cursor program to display the employee with the highest salary.

6. Write a PL/SQL program to retrieve and display employee names in alphabetical order using an explicit cursor.

SGOU

Experiment 4

NoSQL Database

Objectives

- To understand the concept of NoSQL databases.
- To perform CRUD operations using MongoDB.
- To create and manage a MongoDB collection.
- To execute queries using filters and projections.
- To understand and implement indexing in MongoDB.

2.4.1 Theory

A NoSQL database is a non-relational database designed to store and manage unstructured or semi-structured data. MongoDB is a popular NoSQL database that stores data in JSON-like documents (BSON format).

Key Concepts in MongoDB

MongoDB is a NoSQL database management system that stores data in a flexible, document oriented format. Unlike traditional relational databases, MongoDB does not store data in rows and columns. Instead, it organizes data using databases, collections, and documents.

1. Database

A database is a container that stores one or more collections. It provides a logical space for organizing related data. For example, a StudentDB database can contain collections such as students, courses, and departments.

2. Collection

A collection is a group of related documents in MongoDB. It is similar to a table in a relational database. A collection does not require all documents to have exactly the same structure. For example, the students collection can contain documents representing different students.

3. Document

A document is the basic unit of data in MongoDB. It is similar to a record or row in a relational database. Documents are stored in BSON, a binary representation of JSON-like data. A document consists of fields and values.

SGOU

CRUD Operations: Used to create, read, update, and delete documents.

```
db.collection.insertOne(document)        // Create
db.collection.find()                     // Read
db.collection.updateOne(filter, update)  // Update
db.collection.deleteOne(filter)          // Delete
```

Filters: Used to select documents based on a condition.

```
db.collection.find({ field: value })
```

Projections: Used to display only selected fields.

```
db.collection.find({}, { field: 1, _id: 0 })
```

Indexing: Used to improve query performance by creating an index on a field.

```
db.collection.createIndex({ field: 1 })
```

2.4.2 Implementation

Q1. Implement basic CRUD operations using MongoDB for a student database.

```
// Select or create the database
use StudentDB
// CREATE: Insert student records
db.students.insertMany([
  {
    _id : 101,
    rollNo: 101,
    name: "Anu",
    course: "BCA",
    age: 20,
    marks: 85
  },
```

SGOU

```
    {
      _id : 102,
      rollNo: 102,
      name: "Rahul",
      course: "BCA",
      age: 21,
      marks: 78
    },
    {
      _id : 103,
      rollNo: 103,
      name: "Meera",
      course: "BSc Data Science",
      age: 20,
      marks: 91
    }
])
// READ: Display all student records
db.students.find()
// READ: Find a particular student
db.students.findOne({ rollNo: 101 })
// UPDATE: Update the marks of a student
db.students.updateOne(
    { rollNo: 101 },
    { $set: { marks: 90 } }
)
// Display the updated record
db.students.findOne({ rollNo: 101 })
// DELETE: Delete a student record
db.students.deleteOne({ rollNo: 102 })
// Display all remaining records
db.students.find()
```

SGOU

Output

```
{ acknowledged: true, insertedIds: { '0': 101, '1': 102, '2': 103 } }
[
  {
    _id: 101,
    rollNo: 101,
    name: 'Anu',
    course: 'BCA',
    age: 20,
    marks: 85
  },
  {
    _id: 102,
    rollNo: 102,
    name: 'Rahul',
    course: 'BCA',
    age: 21,
    marks: 78
  },
  {
    _id: 103,
    rollNo: 103,
    name: 'Meera',
    course: 'BSc Data Science',
    age: 20,
    marks: 91
  }
]
{
  _id: 101,
  rollNo: 101,
  name: 'Anu',
  course: 'BCA',
  age: 20,
  marks: 85
}
{
```

SGOU

```
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
{
  _id: 101,
  rollNo: 101,
  name: 'Anu',
  course: 'BCA',
  age: 20,
  marks: 90
}
{ acknowledged: true, deletedCount: 1 }
[
  {
    _id: 101,
    rollNo: 101,
    name: 'Anu',
    course: 'BCA',
    age: 20,
    marks: 90
  },
  {
    _id: 103,
    rollNo: 103,
    name: 'Meera',
    course: 'BSc Data Science',
    age: 20,
    marks: 91
  }
]
```

SGOU

Q2.Create a MongoDB collection and execute queries using filters, projections, and indexing.

```
use StudentDB
db.students.insertMany([
    _id : 101,
  { rollNo: 101, name: "Anu", course: "BCA", age: 20, marks: 85 },
    _id : 102,
  { rollNo: 102, name: "Rahul", course: "BCA", age: 21, marks: 78 },
    _id : 103,
  { rollNo: 103, name: "Meera", course: "BSc Data Science", age: 20, marks: 91
},
    _id : 104,
  { rollNo: 104, name: "Arun", course: "BCA", age: 22, marks: 88 }
])
// Filter
db.students.find({ course: "BCA" })
// Projection
db.students.find(
  {},
  { name: 1, marks: 1, _id: 0 }
)
// Filter with condition
db.students.find({ marks: { $gt: 80 } })
// Create Index
db.students.createIndex({ rollNo: 1 })
// Query using Index
db.students.find({ rollNo: 103 })
```

SGOU

Output

```
{
 acknowledged: true,
 insertedIds: { '0': 101, '1': 102, '2': 103, '3': 104 }
}
[
 {
  _id: 101,
  rollNo: 101,
  name: 'Anu',
  course: 'BCA',
  age: 20,
  marks: 85
 },
 {
  _id: 102,
  rollNo: 102,
  name: 'Rahul',
  course: 'BCA',
  age: 21,
  marks: 78
 },
 {
  _id: 104,
  rollNo: 104,
  name: 'Arun',
  course: 'BCA',
  age: 22,
  marks: 88
 }
]
[
 { name: 'Anu', marks: 85 },
 { name: 'Rahul', marks: 78 },
 { name: 'Meera', marks: 91 },
 { name: 'Arun', marks: 88 }
]
```

SGOU

```
[
  {
    _id: 101,
    rollNo: 101,
    name: 'Anu',
    course: 'BCA',
    age: 20,
    marks: 85
  },
  {
    _id: 103,
    rollNo: 103,
    name: 'Meera',
    course: 'BSc Data Science',
    age: 20,
    marks: 91
  },
  {
    _id: 104,
    rollNo: 104,
    name: 'Arun',
    course: 'BCA',
    age: 22,
    marks: 88
  }
]
rollNo_1
[
  {
    _id: 103,
    rollNo: 103,
    name: 'Meera',
    course: 'BSc Data Science',
    age: 20,
    marks: 91
  }
]
```

SGOU

2.4.3 Lab Practice Questions

1. Insert 5 new student records into the collection.
2. Retrieve students whose age is greater than 21.
3. Display only names of all students.
4. Update course of a specific student.
5. Delete students with marks less than 80.
6. Create an index on the course field.
7. Use projection to hide the marked field.
8. Find students belonging to a specific course.
9. Update multiple documents using updateMany().

SGOU

സർവ്വകലാശാലാഗീതം

വിദ്യയാൽ സ്വതന്ത്രരാകണം
വിശ്വപൗരരായി മാറണം
ഗ്രഹപ്രസാദമായ് വിളങ്ങണം
ഗുരുപ്രകാശമേ നയിക്കണേ

കൂരിരുട്ടിൽ നിന്നു ഞങ്ങളെ
സൂര്യവീഥിയിൽ തെളിക്കണം
സ്നേഹദീപ്തിയായ് വിളങ്ങണം
നീതിവൈജയന്തി പാറണം

ശാസ്ത്രവ്യാപ്തിയെന്നുമേകണം
ജാതിഭേദമാകെ മാറണം
ബോധരശ്മിയിൽ തിളങ്ങുവാൻ
ജ്ഞാനകേന്ദ്രമേ ജ്വലിക്കണേ

കുരീപ്പുഴ ശ്രീകുമാർ

SGOU

SREENARAYANAGURU OPEN UNIVERSITY

Regional Centres

Kozhikode
Govt. Arts and Science College
Meenchantha, Kozhikode,
Kerala, Pin: 673002
Ph: 04952920228
email: rckdirector@sgou.ac.in

Thalassery
Govt. Brennen College
Dharmadam, Thalassery,
Kannur, Pin: 670106
Ph: 04902990494
email: rctdirector@sgou.ac.in

Tripunithura
Govt. College
Tripunithura, Ernakulam,
Kerala, Pin: 682301
Ph: 04842927436
email: rcedirector@sgou.ac.in

Pattambi
Sree Neelakanta Govt. Sanskrit College
Pattambi, Palakkad,
Kerala, Pin: 679303
Ph: 04662912009
email: rcpdirector@sgou.ac.in

SAY NO TO RAGGING

BEFORE YOU EVEN THINK OF RAGGING



Humiliation

Blacklisting

Ruined Career

Download

ANTI RAGGING

App

Don't just stand and watch. Stop Ragging! Show Character

Visit UGC Website i.e. www.ugc.Ac.in & www antinaging.in to UGC Anti Ragging regulations.

Ministry of Education
Government of India

Are You Being Ragged ?

विश्वविद्यालय अनुदान आयोग
University Grants Commission
quality higher education for all

DON'T LET IT BE TOO LATE

SAY NO TO DRUGS

LOVE YOURSELF AND ALWAYS BE HEALTHY



SREENARAYANAGURU OPEN UNIVERSITY

The State University for Education, Training and Research in Blended Format, Kerala

SGOU

DBMS LAB

COURSE CODE: M25CA03PC

SGOU

Sreenarayanaguru Open University
Kollam, Kerala Pin- 691601, email: info@sgou.ac.in, www.sgou.ac.in Ph: +91 474 2966841

ISBN 978-81-181574-3-4