**Module 4 – Introduction to DBMS**

**Theory Questions**

***1. What is SQL, and why is it essential in database management?***

**SQL (Structured Query Language)** is a standard programming language used to manage and manipulate relational databases. It is essential in database management because it allows users to:

* Create and modify database structures (CREATE, ALTER, DROP).
* Insert, update, delete, and retrieve data efficiently (INSERT, UPDATE, DELETE, SELECT).
* Manage user access and control data security.
* Ensure data integrity and consistency using constraints.

SQL is the foundation for interacting with most modern relational database systems such as MySQL, PostgreSQL, Oracle, and Microsoft SQL Server.

***2. Explain the difference between DBMS and RDBMS.***

|  |  |
| --- | --- |
| **DBMS (Database Management System)** | **RDBMS (Relational Database Management System)** |
| Stores data in files or as collections of data. | Stores data in tabular form (rows and columns). |
| No support for relationships between data. | Supports relationships using foreign keys. |
| Examples: File System, XML, etc. | Examples: MySQL, PostgreSQL, Oracle, SQL Server. |
| Does not follow the rules of normalization. | Follows normalization rules to reduce redundancy. |
| No support for ACID properties. | Ensures ACID (Atomicity, Consistency, Isolation, Durability) compliance. |

***3. Describe the role of SQL in managing relational databases.***

SQL plays a crucial role in managing relational databases by providing:

* **Data Definition**: Creating and modifying table structures.
* **Data Manipulation**: Inserting, updating, and deleting records.
* **Data Querying**: Retrieving data using SELECT queries.
* **Data Control**: Managing access permissions (GRANT, REVOKE).
* **Transaction Control**: Handling transactions (COMMIT, ROLLBACK) to ensure data consistency.

***4. What are the key features of SQL?***

Key features of SQL include:

* **Data Manipulation Language (DML)**: Commands like SELECT, INSERT, UPDATE, DELETE.
* **Data Definition Language (DDL)**: Commands like CREATE, ALTER, DROP.
* **Data Control Language (DCL)**: GRANT and REVOKE to manage permissions.
* **Transaction Control Language (TCL)**: COMMIT, ROLLBACK, SAVEPOINT for handling transactions.
* **High Performance**: Optimized for fast data access and manipulation.
* **Portability**: Works across different database systems with minor adjustments.
* **Scalability**: Suitable for small to enterprise-level applications.

**LAB EXERCISES**

***Lab 1: Create a new database and table***

-- Create a new database named school\_db   
CREATE DATABASE school\_db;   
   
-- Use the database   
USE school\_db;   
   
-- Create a table called students   
CREATE TABLE students (   
   student\_id INT PRIMARY KEY,   
   student\_name VARCHAR(100),   
   age INT,   
   class VARCHAR(20),   
   address VARCHAR(255)   
); 

***Lab 2: Insert five records and retrieve them***

-- Insert five records into the students table   
INSERT INTO students (student\_id, student\_name, age, class, address) VALUES   
(1, 'Aarav Mehta', 14, '8A', 'Ahmedabad'),   
(2, 'Isha Patel', 13, '7B', 'Surat'),   
(3, 'Rahul Sharma', 15, '9C', 'Vadodara'),   
(4, 'Sneha Desai', 12, '6A', 'Rajkot'),   
(5, 'Karan Joshi', 14, '8B', 'Bhavnagar');   
   
-- Retrieve all records using SELECT statement   
SELECT \* FROM students; 

Here is a properly structured answer for your **Module 4 – SQL Syntax** assignment, covering both **Theory Questions** and **Lab Exercises**.

**2.QL Syntax**

**Theory Questions:**

***1. What are the basic components of SQL syntax?***

The basic components of SQL syntax include:

* **Keywords**: Reserved words used to perform SQL operations (e.g., SELECT, FROM, WHERE, INSERT, UPDATE, DELETE).
* **Identifiers**: Names of database objects such as tables, columns, and databases.
* **Operators**: Symbols used for operations (e.g., =, >, <, !=, AND, OR).
* **Literals**: Fixed values such as numbers, strings, or dates used in queries.
* **Clauses**: Components that define specific parts of a query (e.g., SELECT, FROM, WHERE, ORDER BY).
* **Expressions**: Combinations of identifiers, literals, and operators that produce a value.
* **Semicolon (;)**: Used to end an SQL statement in many database systems.

***2. Write the general structure of an SQL SELECT statement.***

SELECT column1, column2, ...   
FROM table\_name   
WHERE condition   
GROUP BY column   
HAVING condition   
ORDER BY column ASC|DESC; 

* SELECT: Specifies the columns to retrieve.
* FROM: Specifies the table from which to retrieve data.
* WHERE: Filters records based on a condition.
* GROUP BY: Groups rows sharing a property so aggregate functions can be applied.
* HAVING: Filters groups based on aggregate conditions.
* ORDER BY: Sorts the result set by one or more columns.

***3. Explain the role of clauses in SQL statements.***

Clauses define specific parts or conditions of an SQL statement. Each clause plays a unique role in shaping the query result:

* **SELECT**: Determines which columns of data to show.
* **FROM**: Identifies the table(s) from which data is to be retrieved.
* **WHERE**: Filters rows based on specified criteria.
* **GROUP BY**: Groups rows that have the same values in specified columns.
* **HAVING**: Filters the grouped rows based on aggregate conditions.
* **ORDER BY**: Orders the output based on one or more columns.

Clauses are essential for building accurate, meaningful, and efficient queries.

**LAB EXERCISES**

***Lab 1: Retrieve specific columns (student\_name and age)***

SELECT student\_name, age   
FROM students; 

***Lab 2: Retrieve students whose age is greater than 10***

SELECT \*   
FROM students   
WHERE age > 10; 

**3. SQL Constraints**

**Theory Questions:**

***1. What are constraints in SQL? List and explain the different types of constraints.***

**Constraints** in SQL are rules applied to columns in a table to enforce data integrity, accuracy, and reliability. They help ensure that the data stored in the database meets certain rules.

**Types of SQL Constraints:**

1. **PRIMARY KEY**: Uniquely identifies each record in a table. It must contain unique and non-null values.
2. **FOREIGN KEY**: Establishes a relationship between two tables by referencing the PRIMARY KEY of another table.
3. **NOT NULL**: Ensures that a column cannot have a NULL value.
4. **UNIQUE**: Ensures that all values in a column are different.
5. **CHECK**: Ensures that all values in a column satisfy a specific condition.
6. **DEFAULT**: Sets a default value for a column when no value is specified.

***2. How do PRIMARY KEY and FOREIGN KEY constraints differ?***

|  |  |
| --- | --- |
| **PRIMARY KEY** | **FOREIGN KEY** |
| Uniquely identifies each record in a table. | Creates a link between two tables. |
| Cannot contain NULL values. | Can contain NULL values. |
| Only one PRIMARY KEY per table. | A table can have multiple FOREIGN KEYS. |
| Enforces **entity integrity**. | Enforces **referential integrity**. |
| Example: student\_id in students table. | Example: teacher\_id in students referencing teachers. |

***3. What is the role of NOT NULL and UNIQUE constraints?***

* **NOT NULL**: Prevents a column from storing NULL values. It ensures that a field must always have a value when inserting or updating records.
* **UNIQUE**: Ensures that all values in a column are different. It prevents duplicate entries in a column, helping maintain data uniqueness.

Example:

email VARCHAR(100) UNIQUE NOT NULL 

This means every teacher must have a unique, non-empty email address.

**LAB EXERCISES**

***Lab 1: Create the teachers table with constraints***

CREATE TABLE teachers (   
   teacher\_id INT PRIMARY KEY,   
   teacher\_name VARCHAR(100) NOT NULL,   
   subject VARCHAR(50) NOT NULL,   
   email VARCHAR(100) UNIQUE   
); 

***Lab 2: Add a FOREIGN KEY constraint from students to teachers***

Assuming the students table has a teacher\_id column (you may need to add it if not present):

-- First, add the teacher\_id column to the students table (if not already present)   
ALTER TABLE students   
ADD teacher\_id INT;   
   
-- Then, add the FOREIGN KEY constraint   
ALTER TABLE students   
ADD CONSTRAINT fk\_teacher   
FOREIGN KEY (teacher\_id)   
REFERENCES teachers(teacher\_id); 

**4.Main SQL Commands and Sub-commands (DDL)**

**Theory Questions:**

***1. Define the SQL Data Definition Language (DDL).***

**SQL Data Definition Language (DDL)** consists of SQL commands used to define and manage the structure of database objects such as tables, schemas, indexes, and views.

The primary DDL commands are:

* **CREATE**: To create new tables, databases, views, etc.
* **ALTER**: To modify existing database objects.
* **DROP**: To delete database objects.
* **TRUNCATE**: To remove all records from a table quickly without logging individual row deletions.

DDL commands define the schema and structure of the database and are auto-committed, meaning changes are saved permanently once executed.

***2. Explain the CREATE command and its syntax.***

The **CREATE** command in SQL is used to create new database objects such as databases, tables, indexes, and views.

**Syntax for creating a table:**

CREATE TABLE table\_name (   
   column1 datatype constraint,   
   column2 datatype constraint,   
   ...   
); 

**Example:**

CREATE TABLE employees (   
   emp\_id INT PRIMARY KEY,   
   emp\_name VARCHAR(100) NOT NULL   
); 

The CREATE command defines the structure, column names, data types, and constraints for the table.

***3. What is the purpose of specifying data types and constraints during table creation?***

**Purpose of specifying data types:**

* To ensure each column stores only valid and expected data (e.g., numbers, text, dates).
* To optimize storage and improve query performance.

**Purpose of specifying constraints:**

* To maintain data accuracy and integrity.
* To prevent invalid data entry (e.g., using NOT NULL, UNIQUE, PRIMARY KEY, FOREIGN KEY).
* To enforce rules on relationships between tables.

Defining both data types and constraints at table creation helps prevent errors and maintains consistent data throughout the database.

**LAB EXERCISES**

***Lab 1: Create the courses table***

CREATE TABLE courses (   
   course\_id INT PRIMARY KEY,   
   course\_name VARCHAR(100),   
   course\_credits INT   
); 

***Lab 2: Create a new database university\_db***

CREATE DATABASE university\_db; 

Here is a structured and assignment-ready answer for **Module 4 – ALTER Command**, covering both **Theory Questions** and **Lab Exercises**.

**5. ALTER Command**

**Theory Questions:**

***1. What is the use of the ALTER command in SQL?***

The **ALTER** command in SQL is used to modify the structure of an existing table. It allows you to:

* Add new columns to a table.
* Modify the data type or constraints of existing columns.
* Drop (remove) columns from a table.
* Rename columns or the table itself.
* Add or drop constraints (e.g., primary keys, foreign keys).

This command is essential when the database schema needs to evolve or adapt to new requirements without deleting and recreating the entire table.

***2. How can you add, modify, and drop columns from a table using ALTER?***

**To add a new column:**

ALTER TABLE table\_name   
ADD column\_name datatype; 

**To modify an existing column:**

ALTER TABLE table\_name   
MODIFY column\_name new\_datatype; 

*Note: In some databases like SQL Server, use ALTER COLUMN instead of MODIFY.*

**To drop a column:**

ALTER TABLE table\_name   
DROP COLUMN column\_name; 

These operations allow flexible schema changes to match changing application or data needs.

**LAB EXERCISES**

***Lab 1: Add a column course\_duration to the courses table***

ALTER TABLE courses   
ADD course\_duration VARCHAR(50); 

***Lab 2: Drop the course\_credits column from the courses table***

ALTER TABLE courses   
DROP COLUMN course\_credits; 

Here is a complete and well-structured answer for **Module 4 – DROP Command**, covering both **Theory Questions** and **Lab Exercises**:

**6.DROP Command**

**Theory Questions:**

***1. What is the function of the DROP command in SQL?***

The **DROP** command in SQL is used to permanently delete database objects such as:

* **Tables**
* **Databases**
* **Views**
* **Indexes**
* **Constraints**

When you execute a DROP command on a table or database, it removes the object and all of its data and structure from the database. This action **cannot be undone**.

***2. What are the implications of dropping a table from a database?***

Dropping a table has several important implications:

* **Permanent Deletion**: The table and all its data are permanently deleted. This action cannot be rolled back (unless inside a transaction in some DBMS).
* **Loss of Relationships**: Any foreign key relationships or constraints involving the table will be lost.
* **Dependent Objects Affected**: Views, triggers, or procedures that reference the table may become invalid or throw errors.
* **Space Freed**: It frees up storage used by the table and its indexes.

**Caution:** The DROP command should be used carefully, especially in production environments, as the data cannot be recovered once deleted.

**LAB EXERCISES**

***Lab 1: Drop the teachers table from the school\_db database***

-- Use the school\_db database   
USE school\_db;   
   
-- Drop the teachers table   
DROP TABLE teachers; 

***Lab 2: Drop the students table from the school\_db database and verify***

* -- Drop the students table   
  DROP TABLE students;

**7. Data Manipulation Language (DML)**

**Theory Questions:**

***1. Define the INSERT, UPDATE, and DELETE commands in SQL.***

* **INSERT**: Adds new rows of data to a table.

INSERT INTO table\_name (column1, column2, ...)   
VALUES (value1, value2, ...); 

* **UPDATE**: Modifies existing records in a table.

UPDATE table\_name   
SET column1 = value1   
WHERE condition; 

* **DELETE**: Removes existing records from a table.

DELETE FROM table\_name   
WHERE condition; 

***2. What is the importance of the WHERE clause in UPDATE and DELETE operations?***

The WHERE clause is **essential** to:

* **Target specific rows** for updates or deletions.
* **Prevent accidental updates or deletions** of all rows in the table.

Without the WHERE clause, **all rows** may be modified or deleted.

**LAB EXERCISES:**

***Lab 1: Insert three records into the courses table***

INSERT INTO courses (course\_id, course\_name, course\_duration)   
VALUES (101, 'Mathematics', '3 Months'),   
      (102, 'Physics', '4 Months'),   
      (103, 'Chemistry', '3.5 Months'); 

***Lab 2: Update course duration of a specific course***

UPDATE courses   
SET course\_duration = '5 Months'   
WHERE course\_id = 102; 

***Lab 3: Delete a course by course\_id***

DELETE FROM courses   
WHERE course\_id = 103; 

**8. Data Query Language (DQL)**

**Theory Questions:**

***1. What is the SELECT statement, and how is it used to query data?***

The **SELECT** statement retrieves data from one or more tables in a database.

**Syntax:**

SELECT column1, column2   
FROM table\_name   
WHERE condition; 

It’s the most commonly used query for viewing records.

***2. Explain the use of ORDER BY and WHERE clauses in SQL queries.***

* **WHERE**: Filters rows based on specified conditions.
* **ORDER BY**: Sorts the result set by one or more columns, either ascending (ASC) or descending (DESC).

**LAB EXERCISES:**

***Lab 1: Retrieve all courses***

SELECT \* FROM courses; 

***Lab 2: Sort courses by course\_duration in descending order***

SELECT \* FROM courses   
ORDER BY course\_duration DESC; 

***Lab 3: Limit results to top two courses***

SELECT \* FROM courses   
LIMIT 2; 

**9. Data Control Language (DCL)**

**Theory Questions:**

***1. What is the purpose of GRANT and REVOKE in SQL?***

* **GRANT**: Gives users permission to perform actions on database objects (e.g., SELECT, INSERT).
* **REVOKE**: Removes previously granted permissions.

***2. How do you manage privileges using these commands?***

Privileges are managed using:

* **GRANT**:

GRANT SELECT ON courses TO user1; 

* **REVOKE**:

REVOKE INSERT ON courses FROM user1; 

This ensures **controlled access** to data, improving **security and accountability**.

**LAB EXERCISES:**

***Lab 1: Create users and grant SELECT to user1***

CREATE USER 'user1'@'localhost' IDENTIFIED BY 'password1';   
CREATE USER 'user2'@'localhost' IDENTIFIED BY 'password2';   
   
GRANT SELECT ON school\_db.courses TO 'user1'@'localhost'; 

***Lab 2: Revoke INSERT from user1 and grant it to user2***

REVOKE INSERT ON school\_db.courses FROM 'user1'@'localhost';   
GRANT INSERT ON school\_db.courses TO 'user2'@'localhost'; 

**10. Transaction Control Language (TCL)**

**Theory Questions:**

***1. What is the purpose of the COMMIT and ROLLBACK commands in SQL?***

* **COMMIT**: Saves all changes made during the current transaction permanently.
* **ROLLBACK**: Undoes changes made during the current transaction.

They ensure **data integrity and control over data changes**.

***2. Explain how transactions are managed in SQL databases.***

* A **transaction** is a set of SQL operations performed as a single unit.
* Transactions start implicitly or explicitly using BEGIN.
* Can be **committed** to save changes or **rolled back** to undo them.
* **SAVEPOINTs** allow partial rollbacks within a transaction.

**LAB EXERCISES:**

***Lab 1: Insert and COMMIT***

START TRANSACTION;   
   
INSERT INTO courses (course\_id, course\_name, course\_duration)   
VALUES (104, 'Biology', '4 Months');   
   
COMMIT; 

***Lab 2: Insert and ROLLBACK***

START TRANSACTION;   
   
INSERT INTO courses (course\_id, course\_name, course\_duration)   
VALUES (105, 'History', '2 Months');   
   
ROLLBACK; 

***Lab 3: Use SAVEPOINT and ROLLBACK to a specific point***

START TRANSACTION;   
   
UPDATE courses SET course\_duration = '6 Months' WHERE course\_id = 101;   
   
SAVEPOINT before\_second\_update;   
   
UPDATE courses SET course\_duration = '7 Months' WHERE course\_id = 102;   
   
-- Rollback to the savepoint   
ROLLBACK TO before\_second\_update;   
   
COMMIT; 

**11. SQL Joins**

**Theory Questions**

1. **JOIN in SQL**: Joins combine rows from two or more tables based on a related column.
2. **INNER JOIN**: Returns only matching rows.
3. **LEFT JOIN**: Returns all rows from the left table and matching rows from the right.
4. **RIGHT JOIN**: Returns all rows from the right table and matching rows from the left.
5. **FULL OUTER JOIN**: Returns all rows when there's a match in either table.
6. **Joins for Combining Data**: Joins allow you to query data from multiple tables as if they are one, helping relate information like employees and their departments.

**Lab Exercises**

* **Lab 1: INNER JOIN**

SELECT e.employee\_name, d.department\_name   
FROM employees e   
INNER JOIN departments d ON e.department\_id = d.department\_id; 

* **Lab 2: LEFT JOIN**

SELECT d.department\_name, e.employee\_name   
FROM departments d   
LEFT JOIN employees e ON d.department\_id = e.department\_id; 

**12. SQL GROUP BY**

**Theory Questions**

1. **GROUP BY Clause**: Groups rows with the same values into summary rows (used with functions like COUNT, AVG, etc.).
2. **GROUP BY vs ORDER BY**:
3. **GROUP BY**: Groups data.
4. **ORDER BY**: Sorts data.

**Lab Exercises**

* **Lab 1: Count Employees Per Department**

SELECT department\_id, COUNT(\*) AS total\_employees   
FROM employees   
GROUP BY department\_id; 

* **Lab 2: Average Salary**

SELECT department\_id, AVG(salary) AS avg\_salary   
FROM employees   
GROUP BY department\_id; 

**13. SQL Stored Procedures**

**Theory Questions**

1. **Stored Procedure**: A reusable set of SQL statements stored in the database. Unlike standard SQL, it can accept parameters.
2. **Advantages**:
3. Reusability
4. Improved performance
5. Enhanced security

**Lab Exercises**

* **Lab 1: Procedure to Get Employees by Department**

DELIMITER //   
CREATE PROCEDURE GetEmployeesByDept(IN dept\_id INT)   
BEGIN   
 SELECT \* FROM employees WHERE department\_id = dept\_id;   
END //   
DELIMITER ; 

* **Lab 2: Procedure for Course Details**

DELIMITER //   
CREATE PROCEDURE GetCourseDetails(IN c\_id INT)   
BEGIN   
 SELECT \* FROM courses WHERE course\_id = c\_id;   
END //   
DELIMITER ; 

**14. SQL View**

**Theory Questions**

1. **View**: A virtual table based on the result set of a query. It does not store data itself.
2. **Advantages**:
3. Simplifies queries
4. Enhances security
5. Maintains abstraction

**Lab Exercises**

* **Lab 1: View of Employees with Departments**

CREATE VIEW emp\_dept\_view AS   
SELECT e.employee\_name, d.department\_name   
FROM employees e   
JOIN departments d ON e.department\_id = d.department\_id; 

* **Lab 2: Modify View to Filter Salary**

CREATE OR REPLACE VIEW emp\_dept\_view AS   
SELECT e.employee\_name, d.department\_name   
FROM employees e   
JOIN departments d ON e.department\_id = d.department\_id   
WHERE e.salary >= 50000; 

**15. SQL Triggers**

**Theory Questions**

1. **Trigger**: A procedure that automatically executes in response to events like INSERT, UPDATE, DELETE.
2. **BEFORE/AFTER INSERT**
3. **BEFORE/AFTER UPDATE**
4. **BEFORE/AFTER DELETE**
5. **Differences**:
6. INSERT: Fires on new row addition.
7. UPDATE: Fires when row data changes.
8. DELETE: Fires before or after row deletion.

**Lab Exercises**

* **Lab 1: Log INSERT**

CREATE TRIGGER log\_employee\_insert   
AFTER INSERT ON employees   
FOR EACH ROW   
INSERT INTO employee\_log (action, employee\_id, log\_time)   
VALUES ('INSERT', NEW.employee\_id, NOW()); 

* **Lab 2: Update Last Modified**

CREATE TRIGGER update\_last\_modified   
BEFORE UPDATE ON employees   
FOR EACH ROW   
SET NEW.last\_modified = NOW(); 

**16. Introduction to PL/SQL**

**Theory Questions**

1. **PL/SQL**: Oracle's extension to SQL. Adds procedural capabilities (variables, loops, etc.)
2. **Benefits**:
3. Better performance
4. Error handling
5. Modular programming

**Lab Exercises**

* **Lab 1: Count Employees**

DECLARE   
 total\_emp NUMBER;   
BEGIN   
 SELECT COUNT(\*) INTO total\_emp FROM employees;   
 DBMS\_OUTPUT.PUT\_LINE('Total Employees: ' || total\_emp);   
END; 

* **Lab 2: Total Sales**

DECLARE   
 total\_sales NUMBER;   
BEGIN   
 SELECT SUM(sale\_amount) INTO total\_sales FROM orders;   
 DBMS\_OUTPUT.PUT\_LINE('Total Sales: ' || total\_sales);   
END; 

**17. PL/SQL Control Structures**

**Theory Questions**

1. **Control Structures**: Logic used to control flow.
2. **IF-THEN**
3. **LOOP, WHILE, FOR**
4. **Use in Complex Queries**: They allow conditional logic, iteration, and dynamic control over query execution.

**Lab Exercises**

* **Lab 1: IF-THEN Block**

DECLARE   
 dept\_id employees.department\_id%TYPE;   
BEGIN   
 SELECT department\_id INTO dept\_id FROM employees WHERE employee\_id = 101;   
 IF dept\_id = 10 THEN   
   DBMS\_OUTPUT.PUT\_LINE('IT Department');   
 END IF;   
END; 

* **Lab 2: FOR LOOP**

DECLARE   
 emp\_name employees.employee\_name%TYPE;   
BEGIN   
 FOR emp IN (SELECT employee\_name FROM employees) LOOP   
   DBMS\_OUTPUT.PUT\_LINE(emp.employee\_name);   
 END LOOP;   
END; 

**18. SQL Cursors**

**Theory Questions**

1. **Cursor**: Used to process rows returned by SQL one at a time.
2. **Implicit**: Automatically managed by PL/SQL.
3. **Explicit**: Manually defined and controlled.
4. **Use of Explicit Cursor**: Needed when multiple rows are processed row-by-row, allowing custom logic.

**Lab Exercises**

* **Lab 1: Cursor for Employees**

DECLARE   
 CURSOR emp\_cursor IS SELECT employee\_name, salary FROM employees;   
 emp\_record emp\_cursor%ROWTYPE;   
BEGIN   
 OPEN emp\_cursor;   
 LOOP   
   FETCH emp\_cursor INTO emp\_record;   
   EXIT WHEN emp\_cursor%NOTFOUND;   
   DBMS\_OUTPUT.PUT\_LINE(emp\_record.employee\_name || ' - ' || emp\_record.salary);   
 END LOOP;   
 CLOSE emp\_cursor;   
END; 

* **Lab 2: Cursor for Courses**

DECLARE   
 CURSOR course\_cursor IS SELECT \* FROM courses;   
 course\_rec course\_cursor%ROWTYPE;   
BEGIN   
 OPEN course\_cursor;   
 LOOP   
   FETCH course\_cursor INTO course\_rec;   
   EXIT WHEN course\_cursor%NOTFOUND;   
   DBMS\_OUTPUT.PUT\_LINE(course\_rec.course\_name);   
 END LOOP;   
 CLOSE course\_cursor;   
END; 

**19. ROLLBACK, COMMIT, SAVEPOINT**

**Theory Questions**

1. **SAVEPOINT**: Marks a point in a transaction to rollback to without affecting prior changes.
2. **COMMIT**: Saves changes permanently.
3. **ROLLBACK**: Undoes changes since last COMMIT or to a SAVEPOINT.
4. **Use of SAVEPOINT**: Useful in long transactions where partial rollbacks may be needed without restarting everything.

**Lab Exercises**

* **Lab 1: Insert and ROLLBACK to SAVEPOINT**

START TRANSACTION;   
   
INSERT INTO courses VALUES (201, 'English', '3 Months');   
SAVEPOINT sp1;   
   
INSERT INTO courses VALUES (202, 'French', '2 Months');   
ROLLBACK TO sp1;   
   
COMMIT; 

* **Lab 2: Partial Commit and ROLLBACK**

START TRANSACTION;   
   
INSERT INTO courses VALUES (203, 'German', '3 Months');   
SAVEPOINT sp2;   
   
INSERT INTO courses VALUES (204, 'Spanish', '4 Months');   
COMMIT;   
   
INSERT INTO courses VALUES (205, 'Chinese', '2 Months');   
ROLLBACK TO sp2; 