package procedure etc

Aim: To gain a comprehensive understanding of advanced SQL concepts and their practical implementation, including indexes, cursors, triggers, and views.

Course Outcomes:

- **CO4:** Implement packages, procedures, and triggers. (This experiment covers triggers extensively and touches upon procedures through the use of cursors.)

Theory:

This experiment delves into advanced SQL features that provide greater control over data manipulation, retrieval, and integrity. These features are essential for building complex database applications and ensuring efficient data management.

1. Indexes:

Indexes are database structures that improve the speed of data retrieval operations. They are similar to an index in a book, allowing the database to quickly locate specific rows without scanning the entire table.

- **Benefits:** Significantly faster data retrieval, especially for large tables.
- **Types:** Various types of indexes exist, including unique indexes, composite indexes, and functional indexes, each serving different purposes.
- **Considerations:** While indexes improve query performance, they also require additional storage space and can slow down data modification operations (insert, update, delete).

2. Cursors:

Cursors provide a way to retrieve and process data from a result set one row at a time. They are useful for performing row-by-row operations that cannot be easily accomplished with standard SQL queries.

- **Types:** Implicit cursors are automatically created by the database for certain operations, while explicit cursors are defined by the user for more control.
- **Operations:** Cursors support operations like FETCH (retrieve the next row), %NOTFOUND (check if there are more rows), and CLOSE (release the cursor).

3. Triggers:

Triggers are special stored procedures that are automatically executed in response to specific events on a table or view. They are used to enforce data integrity, perform audits, and automate tasks.

- **Types:** Triggers can be defined to execute BEFORE, AFTER, or INSTEAD OF specific DML operations (INSERT, UPDATE, DELETE).
- **Applications:** Triggers are commonly used for tasks like validating data before insertion, logging changes to a table, and maintaining relationships between tables.

4. Views:

Views are virtual tables based on the result-set of an SQL statement. They provide a customized way to look at data from one or more tables, simplifying complex queries and enhancing data security.

- **Benefits:** Simplify data access, hide complexity, provide data security by restricting access to specific columns or rows.
- **Updatability:** Some views are updatable, allowing modifications to the underlying tables through the view.

Learning Outcomes:

Upon completion of this experiment, students will be able to:

- Understand the purpose and benefits of indexes, cursors, triggers, and views.
- Create and manage indexes to optimize query performance.
- Utilize cursors to perform row-by-row data processing.
- Implement triggers to enforce data integrity and automate tasks.
- Design and use views to simplify data access and enhance security.
- Apply these advanced SQL features to build more robust and efficient database applications.

```
-- Creating a table called 'Employees'
CREATE TABLE Employees (
    EmployeeID INT NOT NULL PRIMARY KEY,
    LastName VARCHAR(255) NOT NULL,
    FirstName VARCHAR(255),
    Age INT,
    City VARCHAR(255)
);

-- Creating an index on the LastName column
CREATE INDEX idx_lastname ON Employees (LastName);

-- Inserting data into the Employees table
INSERT INTO Employees (EmployeeID, LastName, FirstName, Age, City) VALUES
(1, 'Smith', 'John', 30, 'London'),
(2, 'Jones', 'Sarah', 25, 'Paris'),
(3, 'Williams', 'David', 40, 'New York');


-- Creating a view that shows employees older than 30
CREATE VIEW OlderEmployees AS
SELECT LastName, FirstName
FROM Employees
WHERE Age > 30;

-- Querying the view
SELECT * FROM OlderEmployees;


-- Creating a trigger that prevents updates to the Employees table on
weekends
CREATE TRIGGER tr_Employees_NoUpdates ON Employees
FOR UPDATE
AS
BEGIN
    IF (DATEPART(dw, GETDATE()) IN (1, 7))  -- 1=Sunday, 7=Saturday
    BEGIN
        RAISERROR('You cannot update the Employees table on weekends.', 16,
1)
        ROLLBACK TRANSACTION
    END
END;


-- Using a cursor to loop through employees and print their names
DECLARE @EmployeeID INT, @LastName VARCHAR(255), @FirstName VARCHAR(255);

DECLARE cur_employees CURSOR FOR
SELECT EmployeeID, LastName, FirstName FROM Employees;

OPEN cur_employees;

FETCH NEXT FROM cur_employees INTO @EmployeeID, @LastName, @FirstName;

WHILE @@FETCH_STATUS = 0
BEGIN
    PRINT 'Employee ID: ' + CAST(@EmployeeID AS VARCHAR) + ', Name: ' +
@LastName + ', ' + @FirstName;
```

```
    FETCH NEXT FROM cur_employees INTO @EmployeeID, @LastName, @FirstName;
END

CLOSE cur_employees;
DEALLOCATE cur_employees;
```