DATABASE TECHNOLOGY - PCA25C03J

1. Implement DDL, DML commands
2. PL/SQL: Case, Loop
3. Implement 4NF normalization techniques
4. Implement integrity constraints

1.Implement DDL, DML commands

AIM:

To implement the Data Definition Language (DDL) and Data Manipulation Language (DML) commands for a university database using PL/SQL. This program will create tables with key constraints, insert sample records, and demonstrate a query to retrieve related data.

ALGORITHM:

1. Create the essential tables: Departments, Instructors, Courses, Students, and Enrolments with appropriate primary and foreign key constraints.

2. Insert sample data records into each table.

3. Commit the transactions to save the changes.

4. Write a sample join query to retrieve student names along with their enrolled course names.

PROGRAM:

```
-- DDL Commands: Create tables for University Database
-- (Converted to SQL Server T-SQL syntax)

CREATE TABLE Departments (
   dept_id INT PRIMARY KEY,
   dept_name VARCHAR(50) NOT NULL,
   location VARCHAR(100)
);

CREATE TABLE Instructors (
   instructor_id INT PRIMARY KEY,
   name VARCHAR(100) NOT NULL,
   dept_id INT,
   CONSTRAINT fk_dept_inst FOREIGN KEY (dept_id) REFERENCES Departments(dept_id)
);
```

```sql
CREATE TABLE Courses (
    course_id INT PRIMARY KEY,
    course_name VARCHAR(100) NOT NULL,
    credits INT,
    dept_id INT,
    CONSTRAINT fk_dept_course FOREIGN KEY (dept_id) REFERENCES Departments(dept_id)
);

-- FIXED: Added missing columns (name, dob, dept_id) and proper foreign key
CREATE TABLE Students (
    student_id INT PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    dob DATE,
    dept_id INT,
    CONSTRAINT fk_dept_student FOREIGN KEY (dept_id) REFERENCES Departments(dept_id)
);

-- FIXED: Added missing comma between constraints
CREATE TABLE Enrollments (
    enrollment_id INT PRIMARY KEY,
    student_id INT,
    course_id INT,
    semester VARCHAR(50),
    CONSTRAINT fk_student FOREIGN KEY (student_id) REFERENCES Students(student_id),
    CONSTRAINT fk_course FOREIGN KEY (course_id) REFERENCES Courses(course_id)
);

-- DML Commands: Insert sample records
-- Departments
INSERT INTO Departments VALUES (1, 'Computer Science', 'Building A');
INSERT INTO Departments VALUES (2, 'Physics', 'Building B');
INSERT INTO Departments VALUES (3, 'Mathematics', 'Building C');

-- Instructors
INSERT INTO Instructors VALUES (101, 'Dr. Alan Turing', 1);
INSERT INTO Instructors VALUES (102, 'Dr. Marie Curie', 2);
INSERT INTO Instructors VALUES (103, 'Dr. Isaac Newton', 3);

-- Courses
INSERT INTO Courses VALUES (201, 'Database Systems', 4, 1);
INSERT INTO Courses VALUES (202, 'Quantum Physics', 3, 2);
```

```
INSERT INTO Courses VALUES (203, 'Calculus', 4, 3);

-- Students
-- FIXED: Replaced Oracle's TO_DATE() with a standard string literal
INSERT INTO Students VALUES (301, 'Alice Smith', '2002-04-15', 1);
INSERT INTO Students VALUES (302, 'Bob Johnson', '2001-09-23', 2);
INSERT INTO Students VALUES (303, 'Clara Oswald', '2003-01-10', 3);

-- Enrollments
INSERT INTO Enrollments VALUES (401, 301, 201, 'Fall 2025');
INSERT INTO Enrollments VALUES (402, 302, 202, 'Spring 2025');
INSERT INTO Enrollments VALUES (403, 303, 203, 'Fall 2025');

-- Sample Query: Select students and their courses
-- FIXED: Completed the JOIN clause
SELECT s.name AS Student_Name, c.course_name AS Course_Name
FROM Students s
JOIN Enrollments e ON s.student_id = e.student_id
JOIN Courses c ON e.course_id = c.course_id;
```

OUTPUT:

```
Output:

Student_Name                                                                                         Course_Name
---------------------------------------------------------------------------------------------------- ------------------
Alice Smith                                                                                          Database Systems
Bob Johnson                                                                                          Quantum Physics
Clara Oswald                                                                                         Calculus
```

RESULT:

This output indicates successful execution of DDL (CREATE, ALTER, DROP) and DML (INSERT, UPDATE, DELETE) commands in PL/SQL.

2.PL/SQL: Case, Loop

AIM:

To demonstrate the use of PL/SQL CASE statements and loops by processing student grades and displaying messages based on their performance. The program will iterate over student records, assign grade categories using CASE, and display the results.

ALGORITHM:

1. Create a temporary table or use an existing table with student grades data.
2. Use a cursor or loop to iterate over each student’s grade record.
3. Use a CASE statement inside the loop to classify grades into categories (e.g., Excellent, Good, Average, Poor).
4. Print the student’s name, grade, and category using DBMS_OUTPUT.
5. End the loop after processing all rows.

PROGRAM:

```
-- Create Students_Grades table (using SQL Server types)
CREATE TABLE Students_Grades (
   student_id INT PRIMARY KEY,
   name VARCHAR(100) NOT NULL,
   grade INT CHECK (grade BETWEEN 0 AND 100)
);

-- Insert sample records
INSERT INTO Students_Grades VALUES (301, 'Alice Smith', 92);
INSERT INTO Students_Grades VALUES (302, 'Bob Johnson', 78);
INSERT INTO Students_Grades VALUES (303, 'Clara Oswald', 65);
-- COMMIT is not needed here; T-SQL is in autocommit mode

-- T-SQL equivalent of your PL/SQL block
DECLARE
   @v_name VARCHAR(100),
   @v_grade INT,
   @v_category VARCHAR(20);

-- 1. Declare the cursor
DECLARE cur_student_grades CURSOR FOR
   SELECT name, grade FROM Students_Grades;

-- 2. Open the cursor
OPEN cur_student_grades;

-- 3. Fetch the first row
```

```sql
FETCH NEXT FROM cur_student_grades INTO @v_name, @v_grade;

-- 4. Loop as long as the fetch was successful
WHILE @@FETCH_STATUS = 0
BEGIN
    -- Using CASE to categorize grades
    SET @v_category = CASE
        WHEN @v_grade >= 90 THEN 'Excellent'
        WHEN @v_grade >= 75 THEN 'Good'
        WHEN @v_grade >= 60 THEN 'Average'
        ELSE 'Poor'
    END;

    -- Use PRINT and + for concatenation
    -- Numbers must be CAST to VARCHAR to be concatenated
    PRINT 'Student: ' + @v_name + ', Grade: ' + CAST(@v_grade AS VARCHAR(10)) + ', Category: ' + @v_category;

    -- 5. Fetch the next row
    FETCH NEXT FROM cur_student_grades INTO @v_name, @v_grade;
END;

-- 6. Close and deallocate the cursor
CLOSE cur_student_grades;
DEALLOCATE cur_student_grades;
```

OUTPUT:

```
Output:

Student: Alice Smith, Grade: 92, Category: Excellent
Student: Bob Johnson, Grade: 78, Category: Good
Student: Clara Oswald, Grade: 65, Category: Average
```

RESULT:

Thus, this program illustrates control structures and decision making in PL/SQL through CASE and loop usage on student grade data.

3.Implement 4NF normalization techniques

AIM:

To demonstrate Fourth Normal Form (4NF) by creating fresh tables representing multi-valued dependencies and join dependencies, then decomposing them to eliminate redundancies and anomalies according to these advanced normalization rules.

ALGORITHM:

1. Start with a table exhibiting a multi-valued dependency (MVD), e.g., Student_Activities (StudentID, Club, Sport).

2. Identify that Club and Sport are independent facts about the student. A student's clubs have no relation to their sports. This is the MVD: StudentID ->> Club and StudentID ->> Sport.

3. Illustrate the problem by inserting sample data. Show that to add one student who is in 2 clubs and 2 sports, you are forced to insert 4 redundant rows.

4. Decompose the table to achieve 4NF. Split the original table into two separate tables to isolate the independent facts:
   - Student_Clubs (StudentID, Club)
   - Student_Sports (StudentID, Sport)
5. Populate these new tables with the distinct data. Show how each fact is now stored only once, eliminating redundancy and solving the update anomalies.

PROGRAM:

```
-- Unnormalized table (fixed for SQL Server)
CREATE TABLE Student_Activities (
   student_id INT,
   student_name VARCHAR(100),
   club VARCHAR(50),
   sport VARCHAR(50),
   PRIMARY KEY (student_id, club, sport)
);


-- Insert sample data (fixed and logically completed)
-- To show the 4NF problem, we show that Alice is in 2 clubs (Chess, Drama)
-- and 2 sports (Basketball, Soccer), forcing 4 rows (2x2).
INSERT INTO Student_Activities VALUES (1, 'Alice Smith', 'Chess Club', 'Basketball');
INSERT INTO Student_Activities VALUES (1, 'Alice Smith', 'Chess Club', 'Soccer');
INSERT INTO Student_Activities VALUES (1, 'Alice Smith', 'Drama Club', 'Basketball');
INSERT INTO Student_Activities VALUES (1, 'Alice Smith', 'Drama Club', 'Soccer');
```

```sql
-- Bob is in 2 clubs (Chess, Drama) but only 1 sport (Tennis), forcing 2 rows (2x1).
INSERT INTO Student_Activities VALUES (2, 'Bob Johnson', 'Chess Club', 'Tennis');
INSERT INTO Student_Activities VALUES (2, 'Bob Johnson', 'Drama Club', 'Tennis');

-- Normalize into 4NF by splitting into two tables
-- Table for Student Clubs (fixed for SQL Server)
CREATE TABLE Student_Clubs (
   student_id INT,
   student_name VARCHAR(100),
   club VARCHAR(50),
   PRIMARY KEY(student_id, club)
);

-- Table for Student Sports (fixed for SQL Server)
CREATE TABLE Student_Sports (
   student_id INT,
   student_name VARCHAR(100),
   sport VARCHAR(50),
   PRIMARY KEY(student_id, sport)
);

-- Insert data extracted from original table to normalized tables
INSERT INTO Student_Clubs (student_id, student_name, club)
SELECT DISTINCT student_id, student_name, club FROM Student_Activities;

INSERT INTO Student_Sports (student_id, student_name, sport)
SELECT DISTINCT student_id, student_name, sport FROM Student_Activities;

-- 4NF Tables
SELECT * FROM Student_Clubs ORDER BY student_id, club;
SELECT * FROM Student_Sports ORDER BY student_id, sport;
```

OUTPUT:

```
Output:
student_id  student_name                                                                                        club
----------  --------------------------------------------------------------------------------------------------  --------------
         1 Alice Smith                                                                                          Chess Club
         1 Alice Smith                                                                                          Drama Club
         2 Bob Johnson                                                                                          Chess Club
         2 Bob Johnson                                                                                          Drama Club
student_id  student_name                                                                                        sport
----------  --------------------------------------------------------------------------------------------------  --------------
         1 Alice Smith                                                                                          Basketball
         1 Alice Smith                                                                                          Soccer
         2 Bob Johnson                                                                                          Tennis
```

RESULT:

This full script shows creating tables from scratch, populating with sample data, performing normalization to 4NF, and storing decomposed data, thereby eliminating multi-valued dependencies while maintaining all information through decomposition.

4.Implement integrity constraints

AIM:

To create a fresh set of tables in a university database that implement various integrity constraints such as primary keys, foreign keys, unique constraints, not null constraints, and check constraints. The program will insert valid and invalid data to demonstrate constraint enforcement.

ALGORITHM:

1. Define tables Departments, Professors, and Courseswith integrity constraints.
2. Apply primary key constraints on ID columns for uniqueness.
3. Apply foreign key constraints to link Coursesto Departmentsand Professors.
4. Use NOT NULL and UNIQUE constraints for essential fields.
5. Use CHECK constraints to validate domain-specific rules (e.g., course credits positive).
6. Insert sample valid records to demonstrate successful constraint enforcement.
7. Attempt to insert invalid records (commented out or in explanation) to show constraint violations.

PROGRAM:

```
-- Create Departments table with NOT NULL and UNIQUE constraints

CREATE TABLE Departments (

   dept_id INT PRIMARY KEY,

   dept_name VARCHAR(50) NOT NULL UNIQUE

);


-- Create Professors table with constraints

CREATE TABLE Professors (
```

```
    professor_id INT PRIMARY KEY,
    professor_name VARCHAR(100) NOT NULL,
    dept_id INT NOT NULL,
    CONSTRAINT fk_prof_dept FOREIGN KEY (dept_id) REFERENCES Departments(dept_id)
);

-- Create Courses table with check constraints and foreign keys
CREATE TABLE Courses (
    course_id INT PRIMARY KEY,
    course_name VARCHAR(100) NOT NULL UNIQUE,
    credits INT NOT NULL CHECK (credits > 0),
    dept_id INT NOT NULL,
    professor_id INT NOT NULL,
    CONSTRAINT fk_course_dept FOREIGN KEY (dept_id) REFERENCES Departments(dept_id),
    CONSTRAINT fk_course_prof FOREIGN KEY (professor_id) REFERENCES Professors(professor_id)
); -- <-- **FIX 1: Added missing ')' and ';' here**

-- **FIX 2: Inserted the missing 'Computer Science' department for ID 1**
INSERT INTO Departments VALUES (1, 'Computer Science');
INSERT INTO Departments VALUES (2, 'Mathematics');

INSERT INTO Professors VALUES (101, 'Dr. Alan Turing', 1);
INSERT INTO Professors VALUES (102, 'Dr. Ada Lovelace', 1);
INSERT INTO Professors VALUES (201, 'Dr. Carl Gauss', 2);

INSERT INTO Courses VALUES (1001, 'Database Systems', 3, 1, 101);
INSERT INTO Courses VALUES (1002, 'Algorithms', 4, 1, 102);
INSERT INTO Courses VALUES (2001, 'Calculus', 4, 2, 201);
```

Expected Output:

- Successful creation of all tables with constraints.
- Successful insertion of valid records.
- On attempting invalid inserts (if uncommented), the database throws errors such as UNIQUE constraint violation, CHECK constraint violation, or foreign key violation.

This program enforces data integrity at multiple levels, ensuring consistent and reliable data in the university database environment.

OUTPUT:

RESULT:

This program enforces data integrity at multiple levels, ensuring consistent and reliable data in the university database environment.