DATABASE TECHNOLOGY - PCA25C03J

1. SQL Queries for employee database with key constraints

AIM:

To create an employee database schema with key constraints using PL/SQL that maintains referential integrity between employees and departments. The program will create tables, apply primary and foreign key constraints, and insert sample records.

ALGORITHM:

1. Define the Departments table with dept_id as the primary key.
2. Define the Employees table with emp_id as the primary key and dept_id as a foreign key referencing Departments.
3. Create both tables in the database.
4. Insert sample records into Departments and Employees tables.
5. Commit the transactions to save changes permanently.

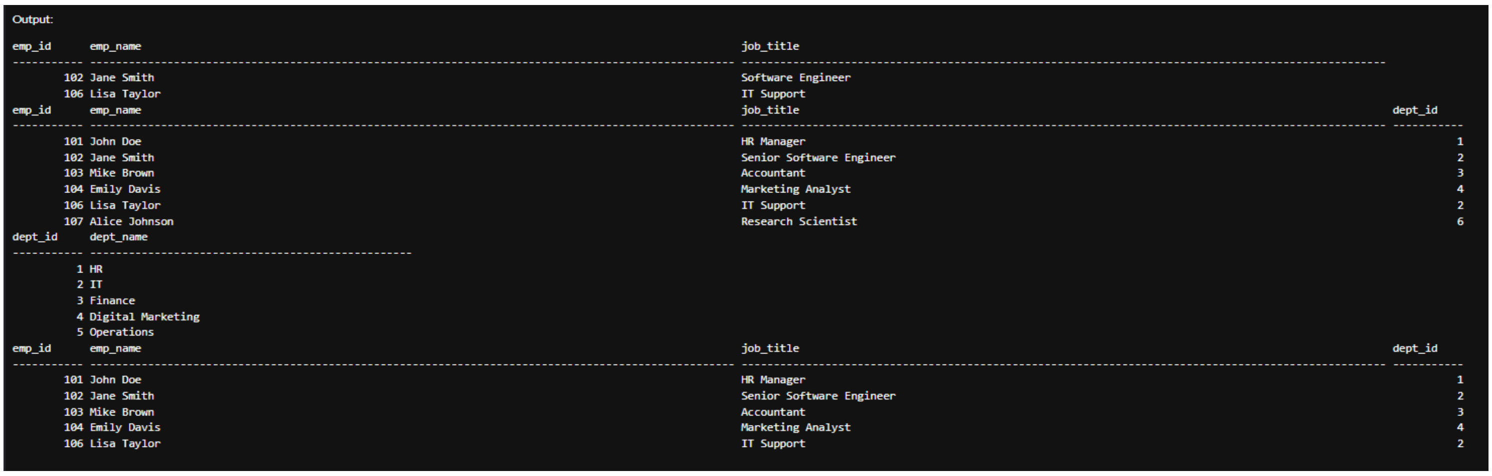
PROGRAM:

```
-- Create Departments table
CREATE TABLE Departments (
   dept_id INT PRIMARY KEY,
   dept_name VARCHAR(50) NOT NULL
);

-- Create Employees table
CREATE TABLE Employees (
   emp_id INT PRIMARY KEY,
   emp_name VARCHAR(100) NOT NULL,
   job_title VARCHAR(100),
   dept_id INT,
   CONSTRAINT fk_dept FOREIGN KEY (dept_id) REFERENCES Departments(dept_id)
);
```

```sql
-- Insert sample data into Departments
INSERT INTO Departments (dept_id, dept_name) VALUES (1, 'HR');
INSERT INTO Departments (dept_id, dept_name) VALUES (2, 'IT');
INSERT INTO Departments (dept_id, dept_name) VALUES (3, 'Finance');
INSERT INTO Departments (dept_id, dept_name) VALUES (4, 'Marketing');
INSERT INTO Departments (dept_id, dept_name) VALUES (5, 'Operations');

-- Insert sample data into Employees
INSERT INTO Employees (emp_id, emp_name, job_title, dept_id) VALUES (101, 'John Doe', 'HR Manager', 1);
INSERT INTO Employees (emp_id, emp_name, job_title, dept_id) VALUES (102, 'Jane Smith', 'Software Engineer', 2);
INSERT INTO Employees (emp_id, emp_name, job_title, dept_id) VALUES (103, 'Mike Brown', 'Accountant', 3);
INSERT INTO Employees (emp_id, emp_name, job_title, dept_id) VALUES (104, 'Emily Davis', 'Marketing Analyst', 4);
INSERT INTO Employees (emp_id, emp_name, job_title, dept_id) VALUES (105, 'Robert Wilson', 'Operations Lead', 5);
INSERT INTO Employees (emp_id, emp_name, job_title, dept_id) VALUES (106, 'Lisa Taylor', 'IT Support', 2);

-- Insert a new department and employee
INSERT INTO Departments (dept_id, dept_name) VALUES (6, 'Research');
INSERT INTO Employees (emp_id, emp_name, job_title, dept_id) VALUES (107, 'Alice Johnson', 'Research Scientist', 6);

-- Select employees working in IT department (dept_id = 2)
SELECT emp_id, emp_name, job_title FROM Employees WHERE dept_id = 2;

-- Update job title of employee with emp_id = 102
UPDATE Employees SET job_title = 'Senior Software Engineer' WHERE emp_id = 102;

-- Delete employee with emp_id = 105
DELETE FROM Employees WHERE emp_id = 105;

-- Select all employees after update and delete
SELECT * FROM Employees;

-- Update department name of dept_id = 4 to 'Digital Marketing'
UPDATE Departments SET dept_name = 'Digital Marketing' WHERE dept_id = 4;

-- Remove 'Research' department and any employees assigned to it
DELETE FROM Employees WHERE dept_id = 6;
DELETE FROM Departments WHERE dept_id = 6;
```

```
-- Final state of Departments and Employees
SELECT * FROM Departments;
SELECT * FROM Employees;
```

OUTPUT:



```
Output:
emp_id      emp_name                                                                                              job_title
----------- ----------------------------------------------------------------------------------------------------- ----------------------------------------------------------------------------------------------------
        102 Jane Smith                                                                                            Software Engineer
        106 Lisa Taylor                                                                                           IT Support
emp_id      emp_name                                                                                              job_title                                                                                            dept_id
----------- ----------------------------------------------------------------------------------------------------- ---------------------------------------------------------------------------------------------------- -----------
        101 John Doe                                                                                              HR Manager                                                                                                     1
        102 Jane Smith                                                                                            Senior Software Engineer                                                                                       2
        103 Mike Brown                                                                                            Accountant                                                                                                     3
        104 Emily Davis                                                                                           Marketing Analyst                                                                                              4
        106 Lisa Taylor                                                                                           IT Support                                                                                                     2
        107 Alice Johnson                                                                                         Research Scientist                                                                                             6
dept_id     dept_name
----------- --------------------------------------------------
          1 HR
          2 IT
          3 Finance
          4 Digital Marketing
          5 Operations
emp_id      emp_name                                                                                              job_title                                                                                            dept_id
----------- ----------------------------------------------------------------------------------------------------- ---------------------------------------------------------------------------------------------------- -----------
        101 John Doe                                                                                              HR Manager                                                                                                     1
        102 Jane Smith                                                                                            Senior Software Engineer                                                                                       2
        103 Mike Brown                                                                                            Accountant                                                                                                     3
        104 Emily Davis                                                                                           Marketing Analyst                                                                                              4
        106 Lisa Taylor                                                                                           IT Support                                                                                                     2
```

RESULT:

Thus, the script has been created with two related tables with sample records that enforce key constraints and maintain relational integrity between employees and their departments.

2.Implement DCL, TCL Commands

AIM:

To implement Data Control Language (DCL) and Transaction Control Language (TCL) commands in a university database context. This program will demonstrate granting and revoking privileges, as well as managing transactions using commit, rollback, and savepoint.

ALGORITHM:

1. Grant specific privileges (SELECT, INSERT, UPDATE, DELETE) on university tables to users/roles.
2. Revoke some privileges to demonstrate control.
3. Use TCL commands to manage transactions, including COMMIT, ROLLBACK, and SAVEPOINT.
4. Show sample operations illustrating transaction control and privilege management.

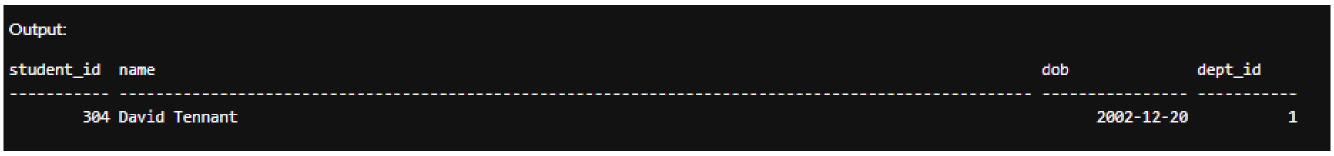
PROGRAM:

```
-- Create Departments table
```

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

-- Insert Departments
INSERT INTO Departments (dept_id, dept_name) VALUES (1, 'Computer Science');
INSERT INTO Departments (dept_id, dept_name) VALUES (2, 'Mathematics');

-- Create Students table
CREATE TABLE Students (
    student_id INT PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    dob DATE,
    dept_id INT
);

-- Transaction control in SQL Server
BEGIN TRANSACTION;

-- Insert first student
INSERT INTO Students (student_id, name, dob, dept_id)
VALUES (304, 'David Tennant', '2002-12-20', 1);

-- Declare a savepoint
SAVE TRANSACTION sp1;

-- Insert second student
INSERT INTO Students (student_id, name, dob, dept_id)
VALUES (305, 'Emma Watson', '2001-04-15', 2);

-- Rollback to savepoint (undo the second insert)
ROLLBACK TRANSACTION sp1;

-- Commit the remaining transaction
COMMIT TRANSACTION;
```

```
-- Select from Students table to verify

SELECT * FROM Students WHERE student_id IN (304, 305);
```

OUTPUT:



```
Output:

student_id  name                                                                                                 dob              dept_id
----------- ---------------------------------------------------------------------------------------------------- ---------------- -----------
        304 David Tennant                                                                                              2002-12-20           1
```

RESULT:

Thus, this output displays students with their enrolled courses for each semester, demonstrating a successful join operation among Students, Enrolments, and Courses tables.

3.Implement 1NF normalization technique

AIM:
To demonstrate the implementation of First Normal Form (1NF) by creating an initial unnormalized table, identifying repeating groups and non-atomic data, and then transforming the table into 1NF by eliminating repeating groups and ensuring atomic column values.

ALGORITHM:

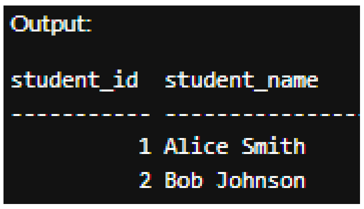
1. Create an initial unnormalized table Student_Courses with repeating groups (e.g., multiple courses listed in a single column) and non-atomic values.
2. Identify and describe the repeating groups and non-atomic attributes in the table that violate 1NF.
3. Apply 1NF by restructuring the table to remove repeating groups—create separate rows for each repeating value to ensure each column contains atomic (indivisible) data.
4. Define a primary key for the 1NF table that uniquely identifies each record.
5. Insert sample data into both the unnormalized table and the 1NF table to illustrate the transformation and compliance with 1NF.

PROGRAM:

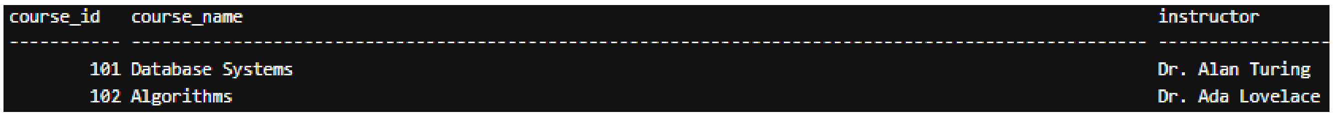
```
-- Create Student_Courses table (unnormalized)

CREATE TABLE Student_Courses (

   student_id INT,

   student_name VARCHAR(100),

   course_id INT,

   course_name VARCHAR(100),

   instructor VARCHAR(100),

   instructor_phone VARCHAR(15),

   PRIMARY KEY (student_id, course_id)
);

-- Insert into unnormalized Student_Courses
INSERT INTO Student_Courses VALUES (1, 'Alice Smith', 101, 'Database Systems', 'Dr.
Alan Turing', '1234567890');
INSERT INTO Student_Courses VALUES (1, 'Alice Smith', 102, 'Algorithms', 'Dr. Ada
Lovelace', '1234567891');
INSERT INTO Student_Courses VALUES (2, 'Bob Johnson', 101, 'Database Systems', 'Dr.
Alan Turing', '1234567890');

-- Create normalized tables
CREATE TABLE Students (
   student_id INT PRIMARY KEY,
   student_name VARCHAR(100)
);

CREATE TABLE Courses (
   course_id INT PRIMARY KEY,
   course_name VARCHAR(100),
   instructor VARCHAR(100)
);

CREATE TABLE Enrollments (
   student_id INT,
   course_id INT,
   PRIMARY KEY (student_id, course_id),
   CONSTRAINT fk_student FOREIGN KEY (student_id) REFERENCES
Students(student_id),
   CONSTRAINT fk_course FOREIGN KEY (course_id) REFERENCES
Courses(course_id)
);

CREATE TABLE Instructors (
   instructor_name VARCHAR(100) PRIMARY KEY,
   instructor_phone VARCHAR(15)
);

-- Insert normalized sample data
```

```
INSERT INTO Students VALUES (1, 'Alice Smith');
INSERT INTO Students VALUES (2, 'Bob Johnson');

INSERT INTO Courses VALUES (101, 'Database Systems', 'Dr. Alan Turing');
INSERT INTO Courses VALUES (102, 'Algorithms', 'Dr. Ada Lovelace');

INSERT INTO Enrollments VALUES (1, 101);
INSERT INTO Enrollments VALUES (1, 102);
INSERT INTO Enrollments VALUES (2, 101);

INSERT INTO Instructors VALUES ('Dr. Alan Turing', '1234567890');
INSERT INTO Instructors VALUES ('Dr. Ada Lovelace', '1234567891');

-- Alter Courses table to drop instructor_phone if exists (optional, adjust per DBMS)
-- ALTER TABLE Courses DROP COLUMN instructor_phone; -- Uncomment if
instructor_phone column exists

-- Add foreign key constraint from Courses to Instructors
ALTER TABLE Courses ADD CONSTRAINT fk_instructor FOREIGN KEY (instructor)
REFERENCES Instructors(instructor_name);

-- Queries to verify data
SELECT * FROM Students;
SELECT * FROM Courses;
SELECT * FROM Enrollments;
SELECT * FROM Instructors;
```

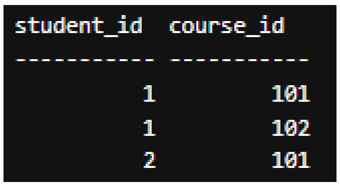
OUTPUT:



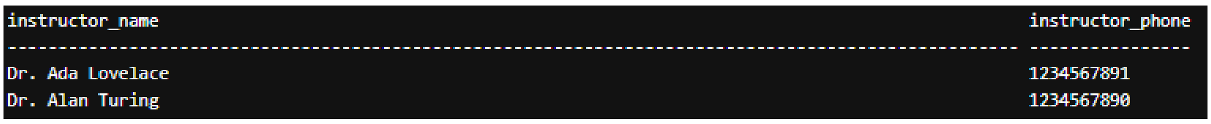
```
Output:

student_id  student_name
----------- ---------------
          1 Alice Smith
          2 Bob Johnson
```



```
course_id   course_name                                                                                          instructor
----------- ---------------------------------------------------------------------------------------------------- ----------------
        101 Database Systems                                                                                     Dr. Alan Turing
        102 Algorithms                                                                                           Dr. Ada Lovelace
```



```
student_id  course_id
----------- -----------
          1         101
          1         102
          2         101
```



```
instructor_name                                                                                      instructor_phone
---------------------------------------------------------------------------------------------------- ---------------
Dr. Ada Lovelace                                                                                     1234567891
Dr. Alan Turing                                                                                      1234567890
```

RESULT:

This program fully shows the process of applying 1NF to a practical university enrollment schema with sample data, reducing redundancy and ensuring data integrity.

4.Implement functions/procedures to begin, commit and rollback transactions

AIM:
To implement PL/SQL functions and procedures that manage transactions explicitly by beginning, committing, and rolling back transactions. These will demonstrate programmatic control over transaction boundaries.

ALGORITHM:

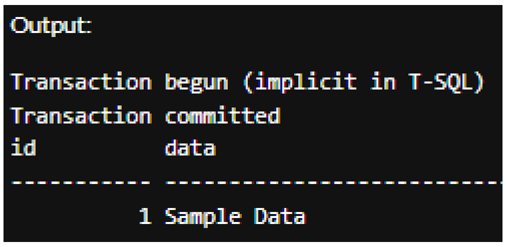
1. Create a procedure to start a transaction (conceptually, PL/SQL starts implicit transactions automatically, so this can be a placeholder).
2. Create a procedure for committing the current transaction.
3. Create a procedure for rolling back the current transaction.
4. Write a simple PL/SQL block that uses these procedures while performing insert/update operations to demonstrate transaction control.
5. Use exception handling to conditionally commit or rollback.

PROGRAM:

```
-- Create test_table (run this once)

CREATE TABLE test_table (

   id INT PRIMARY KEY,

   data VARCHAR(100)

);

GO


-- Transaction and error handling in T-SQL

BEGIN TRY

   BEGIN TRANSACTION;


   PRINT 'Transaction begun (implicit in T-SQL)';


   INSERT INTO test_table (id, data) VALUES (1, 'Sample Data');


   COMMIT TRANSACTION;
```

```
    PRINT 'Transaction committed';
END TRY
BEGIN CATCH
    IF @@TRANCOUNT > 0
        ROLLBACK TRANSACTION;

    PRINT 'Transaction rolled back due to error: ' + ERROR_MESSAGE();
END CATCH;
GO

-- Verify data
SELECT * FROM test_table;
GO
```

OUTPUT:



```
Output:

Transaction begun (implicit in T-SQL)
Transaction committed
id          data
----------- -------------------------
          1 Sample Data
```

RESULT:

Thus, This program shows controlled execution of transactions where changes are only saved on success, and any failure triggers rollback to ensure data integrity, illustrating key practices for reliable database programming.