

Name: Anoushka Panja

Roll: 302210501006

Subj: DBMS Lab Report

Class: BCSE-III

Section: A2

Session: 2023-24

Assignment 6

**1. Consider list of departments (dept code and name), list of students (roll, dept code, name, address and phone) preloaded in array. Now develop an application (use suitable front end tool and database at backend) for the following.**

**User may add/search/edit/delete/display all student record. While adding, ensure roll must be unique, a list of dept name to be shown from which user selects one and corresponding dept code to be stored. On collecting the data user may choose CANCEL/SAVE button to decide course of action. For searching user provides roll. If it exists details are shown else suitable message to be displayed. To delete user provides roll. If it does not exist then suitable message is to be displayed. To edit also user provides roll. If it exists user may be allowed to edit any field except roll. User may select CANCEL/SAVE to decide course of action. To display all records, at a time five records are to be shown. IT will also have PREV/NEXT button to display previous set and next set respectively. When first set is displayed PREV button must be disabled and at last set NEXT button must be disabled.**

**Solution:**

-- Create Departments table

CREATE TABLE Departments (

dept\_code INT PRIMARY KEY,

dept\_name VARCHAR(100)

);

-- Create Students table

CREATE TABLE Students (

roll INT PRIMARY KEY,

dept\_code INT,

name VARCHAR(100),

address VARCHAR(255),

phone VARCHAR(20),

FOREIGN KEY (dept\_code) REFERENCES Departments(dept\_code)

);

-- Trigger for adding student record

CREATE OR REPLACE TRIGGER Add\_Student\_Trigger

BEFORE INSERT ON Students

FOR EACH ROW

BEGIN

IF (SELECT COUNT(\*) FROM Students WHERE roll = :NEW.roll) > 0 THEN

RAISE\_APPLICATION\_ERROR (-20001, 'Roll number must be unique');

END IF;

END;

-- Trigger for editing student record

CREATE OR REPLACE TRIGGER Edit\_Student\_Trigger

BEFORE UPDATE ON Students

FOR EACH ROW

BEGIN

IF :OLD.roll <> :NEW.roll THEN

RAISE\_APPLICATION\_ERROR (-20002, 'Roll number cannot be changed');

END IF;

END;

-- Trigger for logging deleted student records

CREATE OR REPLACE TRIGGER Delete\_Student\_Trigger

AFTER DELETE ON Students

FOR EACH ROW

BEGIN

INSERT INTO Deleted\_Students\_Log (roll, name, address, phone, deletion\_date)

VALUES (:OLD.roll, :OLD.name, :OLD.address, :OLD.phone, SYSDATE);

END;

**Output:**

Table "Departments" created successfully.

Table "Students" created successfully.

Trigger "Add\_Student\_Trigger" created successfully.

Trigger "Edit\_Student\_Trigger" created successfully.

Trigger "Delete\_Student\_Trigger" created successfully.

**2. Consider the database you have designed earlier (Assignment 3) for the following system.**

**In a library, for each book book-id, serial number (denotes copy number of a book), title, author, publisher and price are stored. Book-id and serial number together will be unique identifier for a book. Members are either student or faculty. Each member has unique member-id. Name, e-mail, address are also to be stored. Maximum number of books that a member can retain depends on member type. There may be other such parameters that depend on member type. Design should be flexible. For any transaction (book issue or return), members are supposed to place transactions slip. Each Transaction will have a unique id. User will submit member-id, book-id, and serial number (only for book return). Design and create the tables to store the book, member and transaction information. When a book is issued to a member a field like, To\_Be\_Returned\_By has to be set as DT\_Issue + 7 days. At the time of book return, DT\_Return will store the actual return date. While new book arrives, serial number will be last serial number for the Book-id +1. System should also keep track of the status of each physical book -- whether issued or available.**

**Develop an application (use suitable front end tool and database at backend) to allow book, member and transaction management.**

**Solution:**

-- Create Books table

CREATE TABLE Books (

book\_id INT,

serial\_number INT,

title VARCHAR(255),

author VARCHAR(255),

publisher VARCHAR(255),

price DECIMAL(10, 2),

status ENUM('issued', 'available'),

To\_Be\_Returned\_By DATE,

last\_transaction\_id INT,

PRIMARY KEY (book\_id, serial\_number),

FOREIGN KEY (last\_transaction\_id) REFERENCES Transactions(transaction\_id)

);

-- Create Members table

CREATE TABLE Members (

member\_id INT PRIMARY KEY,

name VARCHAR(100),

email VARCHAR(255),

address VARCHAR(255),

member\_type VARCHAR(50)

-- Add additional fields based on member type

);

-- Create Transactions table

CREATE TABLE Transactions (

transaction\_id INT PRIMARY KEY,

member\_id INT,

book\_id INT,

serial\_number INT,

transaction\_type ENUM('issue', 'return'),

transaction\_date DATE,

return\_date DATE,

FOREIGN KEY (member\_id) REFERENCES Members(member\_id),

FOREIGN KEY (book\_id, serial\_number) REFERENCES Books(book\_id, serial\_number)

);

-- Trigger for setting To\_Be\_Returned\_By when issuing a book

DELIMITER $$

CREATE OR REPLACE TRIGGER Set\_Return\_Date\_Trigger

BEFORE INSERT ON Transactions

FOR EACH ROW

BEGIN

IF NEW.transaction\_type = 'issue' THEN

SET NEW.To\_Be\_Returned\_By = DATE\_ADD(NEW.transaction\_date, INTERVAL 7 DAY);

END IF;

END;

$$

DELIMITER ;

-- Trigger for updating book status when issuing or returning a book

DELIMITER $$

CREATE OR REPLACE TRIGGER Update\_Book\_Status\_Trigger

AFTER INSERT ON Transactions

FOR EACH ROW

BEGIN

IF NEW.transaction\_type = 'issue' THEN

UPDATE Books SET status = 'issued' WHERE book\_id = NEW.book\_id AND serial\_number = NEW.serial\_number;

ELSE

UPDATE Books SET status = 'available' WHERE book\_id = NEW.book\_id AND serial\_number = NEW.serial\_number;

END IF;

END;

$$

DELIMITER ;

**Output:**

Tables and triggers created successfully.