Understanding Views in SQL

Concept of View:

A view in SQL is a virtual table that is based on the result-set of an SQL query. It contains rows and columns, just like a real table, and the fields in a view are fields from one or more real tables in the database. Views can be used to simplify complex queries, enhance security by restricting access to certain data, and provide a layer of abstraction over the actual tables.

Benefits of Using Views:

1. **Simplicity:** Simplifies complex SQL queries by encapsulating them in a view.
2. **Security:** Restricts user access to specific rows or columns of data.
3. **Consistency:** Provides a consistent, unchanging interface to the underlying table structures even if the underlying schema changes.

Creating a Dummy Database and Tables

Let's create a sample database and tables to demonstrate views.

Step 1: Create the Database

```
CREATE DATABASE SchoolDB;
USE SchoolDB;
```

Step 2: Create Tables

```
CREATE TABLE Students (

    StudentID INT AUTO_INCREMENT PRIMARY KEY,
    FirstName VARCHAR(50),
    LastName VARCHAR(50),
    Age INT,
    Major VARCHAR(50)
);

CREATE TABLE Courses (
    CourseID INT AUTO_INCREMENT PRIMARY KEY,
    CourseName VARCHAR(100),
    Credits INT
);

CREATE TABLE Enrollments (
    EnrollmentID INT AUTO_INCREMENT PRIMARY KEY,
    StudentID INT,
    CourseID INT,
    EnrollmentDate DATE,
    FOREIGN KEY (StudentID) REFERENCES Students(StudentID),
    FOREIGN KEY (CourseID) REFERENCES Courses(CourseID)
);

INSERT INTO Students (FirstName, LastName, Age, Major) VALUES
('John', 'Doe', 20, 'Computer Science'),
('Jane', 'Smith', 22, 'Mathematics'),
('Emily', 'Davis', 21, 'Physics');

INSERT INTO Courses (CourseName, Credits) VALUES
('Database Systems', 3),
('Calculus', 4),
('Physics', 4);

INSERT INTO Enrollments (StudentID, CourseID, EnrollmentDate) VALUES
(1, 1, '2023-01-15'),
(2, 2, '2023-01-16'),
(3, 3, '2023-01-17');
```

Working with Views

Step 3: Create a View

A view can be created to provide a simplified representation of data. For example, a view to show students and their enrolled courses.

```
CREATE VIEW StudentCourses AS
SELECT
    Students.StudentID,
    Students.FirstName,
    Students.LastName,
    Courses.CourseName,
    Enrollments.EnrollmentDate
FROM
    Students
JOIN
    Enrollments ON Students.StudentID = Enrollments.StudentID
JOIN
    Courses ON Enrollments.CourseID = Courses.CourseID;
```

Step 4: Query the View

```
SELECT * FROM StudentCourses;
```

This query will show a list of students along with the courses they are enrolled in.

Step 5: Updating Data via Views

Views that are based on a single table and meet certain criteria can be updatable. For example, if we had a simple view on the `Students` table:

```
CREATE VIEW StudentNames AS
SELECT StudentID, FirstName, LastName
FROM Students;
```

We can update data through this view:

```
UPDATE StudentNames
SET FirstName = 'Jonathan'
WHERE StudentID = 1;
```

Step 6: Dropping a View

If you no longer need a view, you can drop it.

```
DROP VIEW IF EXISTS StudentCourses;
```

Understanding Stored Procedures in SQL

Concept of Stored Procedure:

A stored procedure is a set of SQL statements that can be stored in the database and executed as a single unit. They can be used to encapsulate logic, automate repetitive tasks, and improve performance by reducing the amount of data transmitted between client and server. Stored procedures can accept parameters, allow for conditional logic, and can return result sets or values.

Benefits of Using Stored Procedures:

1. **Performance:** Reduced network traffic and improved performance by executing complex operations on the server.
2. **Security:** Abstracts the database schema and allows controlled access to data.
3. **Maintainability:** Centralized code management and reusability of SQL code.
4. **Consistency:** Ensures consistent implementation of business logic across applications.

Creating a Dummy Database and Tables

Let's create a sample database and tables to demonstrate stored procedures.

Step 1: Create the Database

```
CREATE DATABASE SchoolDB;
USE SchoolDB;
```

Step 2: Create Tables

```
CREATE TABLE Students (
    StudentID INT AUTO_INCREMENT PRIMARY KEY,
    FirstName VARCHAR(50),
    LastName VARCHAR(50),
    Age INT,
    Major VARCHAR(50)
);

CREATE TABLE Courses (
    CourseID INT AUTO_INCREMENT PRIMARY KEY,
    CourseName VARCHAR(100),
    Credits INT
);

CREATE TABLE Enrollments (
    EnrollmentID INT AUTO_INCREMENT PRIMARY KEY,
    StudentID INT,
    CourseID INT,
    EnrollmentDate DATE,
    FOREIGN KEY (StudentID) REFERENCES Students(StudentID),
    FOREIGN KEY (CourseID) REFERENCES Courses(CourseID)
);

INSERT INTO Students (FirstName, LastName, Age, Major) VALUES
('John', 'Doe', 20, 'Computer Science'),
('Jane', 'Smith', 22, 'Mathematics'),
('Emily', 'Davis', 21, 'Physics');

INSERT INTO Courses (CourseName, Credits) VALUES
('Database Systems', 3),
('Calculus', 4),
('Physics', 4);

INSERT INTO Enrollments (StudentID, CourseID, EnrollmentDate) VALUES
(1, 1, '2023-01-15'),
(2, 2, '2023-01-16'),
(3, 3, '2023-01-17');
```

Working with Stored Procedures

Step 3: Create a Stored Procedure

Let's create a stored procedure to add a new student and enroll them in a course.

```
DELIMITER //

CREATE PROCEDURE AddStudentAndEnroll(
    IN pFirstName VARCHAR(50),
    IN pLastName VARCHAR(50),
    IN pAge INT,
    IN pMajor VARCHAR(50),
    IN pCourseID INT,
    IN pEnrollmentDate DATE
)
BEGIN
    DECLARE newStudentID INT;

    -- Insert the new student
    INSERT INTO Students (FirstName, LastName, Age, Major)
    VALUES (pFirstName, pLastName, pAge, pMajor);

    -- Get the new student's ID
    SET newStudentID = LAST_INSERT_ID();

    -- Enroll the new student in the course
    INSERT INTO Enrollments (StudentID, CourseID, EnrollmentDate)
    VALUES (newStudentID, pCourseID, pEnrollmentDate);
END //

DELIMITER ;
```

Step 4: Execute the Stored Procedure

To execute the stored procedure and add a new student:

```
CALL AddStudentAndEnroll('Alice', 'Brown', 23, 'Chemistry', 1,
'2023-02-01');
```

Step 5: Query the Data to Verify

```
SELECT * FROM Students;
SELECT * FROM Enrollments;
```

Step 6: Modifying a Stored Procedure

Stored procedures can be altered as needed. For example, to add a logging mechanism:

```
DELIMITER //

CREATE PROCEDURE AddStudentAndEnrollWithLog(
    IN pFirstName VARCHAR(50),
    IN pLastName VARCHAR(50),
    IN pAge INT,
    IN pMajor VARCHAR(50),
    IN pCourseID INT,
    IN pEnrollmentDate DATE
)
BEGIN
    DECLARE newStudentID INT;

    -- Insert the new student
    INSERT INTO Students (FirstName, LastName, Age, Major)
    VALUES (pFirstName, pLastName, pAge, pMajor);

    -- Get the new student's ID
    SET newStudentID = LAST_INSERT_ID();

    -- Enroll the new student in the course
    INSERT INTO Enrollments (StudentID, CourseID, EnrollmentDate)
    VALUES (newStudentID, pCourseID, pEnrollmentDate);

    -- Log the action
    INSERT INTO Logs (Action, Description)
          VALUES ('AddStudentAndEnroll', CONCAT('Added student ',
newStudentID, ' and enrolled in course ', pCourseID));
END //
```

```
DELIMITER ;
```

Step 7: Dropping a Stored Procedure

If you no longer need a stored procedure, you can drop it.

```
DROP PROCEDURE IF EXISTS AddStudentAndEnroll;
```