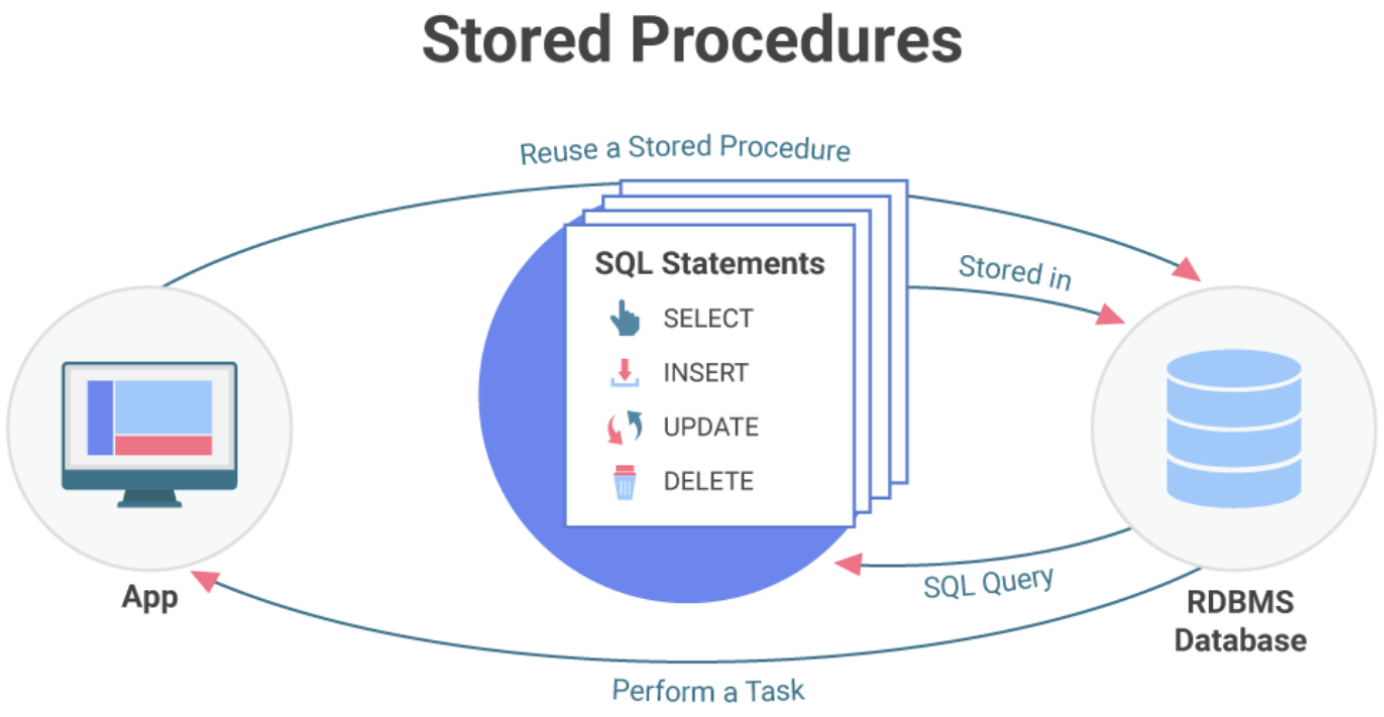
15-Stored Procedure

Author: Vincent Lau

Learning Objectives

- Understand the purpose of creating database procedures
- Understand the ways of implementing procedures
- Understand the advantages and disadvantages of using procedures

Introduction



In a database management system (DBMS), a procedure is a set of precompiled SQL statements that perform a specific task or sequence of tasks. Procedures are often used to encapsulate reusable business logic, making it easier to manage and execute complex operations within the database. Procedures can accept parameters, perform data manipulations, and return results.

Creating a Procedure

```sql
CREATE PROCEDURE procedure_name ([IN | OUT | INOUT] parameter_name data_type
[, ...])
BEGIN
    -- SQL statements
END;
/
```

Example of Creating a Procedure

Suppose you have a database with a table named "STAFF".

You can create a procedure to update a staff's salary with a specified ID.

```sql
-- USE DATABASE SYS;
USE SYS;

-- CREATE TABLE STAFF
CREATE TABLE STAFF (
    ID INTEGER,
    SALARY VARCHAR(1)
);

-- DROP PROCEDURE
DROP PROCEDURE IF EXISTS UPDATE_SALARY;

-- CREATE PROCEDURE AND COMPILE THE PROCEDURE
DELIMITER //

CREATE PROCEDURE UPDATE_SALARY (IN STAFF_ID INTEGER) -- one input parameter
BEGIN
    UPDATE STAFF SET SALARY = 20000 WHERE ID = STAFF_ID;
END;
//

DELIMITER ;

-- CALL PROCEDURE
CALL SYS.UPDATE_SALARY(11);
```

More Complex Procedure

```sql
DELIMITER //

CREATE PROCEDURE calculate_avg_salary(OUT avg) -- No input parameter

BEGIN
    DECLARE notfound INT DEFAULT FALSE;
    DECLARE dept_id INT;
    DECLARE total_salary DECIMAL(10, 2);
    DECLARE emp_count INT;
    DECLARE avg_salary DECIMAL(10, 2);

    -- Declare cursor for department IDs
    DECLARE dept_cursor CURSOR FOR
        SELECT DISTINCT department_id
        FROM employees;

    -- Declare continue handler for cursor
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET notfound = TRUE;

    -- Create temporary table to store results
    CREATE TEMPORARY TABLE temp_results (
        department_id INT,
        average_salary DECIMAL(10, 2)
    );

    -- Open the cursor
    OPEN dept_cursor;
    dept_loop: LOOP     -- Loop through departments
        FETCH dept_cursor INTO dept_id;
        IF notfound THEN
            LEAVE dept_loop;
        END IF;

        -- Calculate total salary and employee count for the department
        SELECT SUM(salary), COUNT(*) INTO total_salary, emp_count
        FROM employees
        WHERE department_id = dept_id;

        -- Calculate average salary
        IF emp_count > 0 THEN
            SET avg_salary = total_salary / emp_count;
        ELSE
            SET avg_salary = 0;
        END IF;

        -- Insert into temporary results table
        INSERT INTO temp_results VALUES (dept_id, avg_salary);
    END LOOP;

    -- Close the cursor
    CLOSE dept_cursor;

    -- Insert results from temp table to summary table
    INSERT INTO department_avg_salary SELECT * FROM temp_results;

    -- Drop temporary table
    DROP TEMPORARY TABLE temp_results;
END;
//

DELIMITER ;
```

Executing a Procedure

```sql
CALL procedure_name([parameter_value, ...]);
```

Example of Executing a Procedure

To update the salary of an employee with ID 101 by 10%:

```sql
CALL update_salary(101, 10);
```

Why using Procedures

Benefits of Using Procedures

1. **Code Reusability**

Procedures encapsulate business logic, making it reusable across different parts of an application.

2. **Performance**

Procedures are precompiled, potentially leading to better execution performance compared to running individual SQL statements.

3. **Security**

Procedures can limit direct access to underlying tables, controlling data manipulations through the procedure's interface.

4. **Data Consistency**

Procedures can ensure that specific operations are carried out consistently across different parts of the application.

5. **Abstraction**

Procedures abstract the details of complex operations, making the code easier to understand and maintain.

Downsides of Using Procedures

1. **Learning Curve**

Developing and maintaining procedures might require familiarity with the procedural language used by the DBMS (e.g., PL/SQL in Oracle, T-SQL in SQL Server).

2. **Database Locking**

Procedures might lead to resource contention, especially if multiple users or transactions execute them simultaneously.

3. **Limited Portability**

Procedures are specific to the DBMS you're using, making code less portable across different database systems.

4. **Complexity**

Overusing procedures for simple operations can lead to unnecessary complexity.

Procedures vs Direct SQL Statements

Procedures and direct SQL statements are two different ways to interact with a database in a database management system (DBMS). Each has its own advantages and use cases.

Let's explore the differences between them:

Procedures

1. Encapsulation of Logic

- Procedures allow you to encapsulate a sequence of SQL statements and logic into a single reusable unit. This can simplify complex operations by abstracting the details.
- The logic within a procedure can involve conditional statements (IF-THEN-ELSE), loops (FOR, WHILE), and error handling.

2. Code Reusability

- Procedures can be defined once and executed multiple times, making them useful for reusable business logic.
- The same procedure can be used by different parts of an application, promoting code reuse and consistency.

3. Security and Permissions

- Procedures can be used to control access to underlying tables by exposing only the procedure interface.
- This helps manage security and restrict unauthorized access to data.

4. Performance Optimization

- Complex calculations and operations can be precomputed and stored in procedures, potentially improving performance by reducing redundant calculations.

5. Abstraction

    - Procedures abstract the complexity of SQL queries and operations, making it easier to understand and maintain the codebase.

Direcrt SQL Statements

1. Immediate Execution

- Direct SQL statements are executed immediately when they are issued to the database.
- They are useful for quick ad hoc queries or operations that don't require complex logic encapsulation.

2. Flexibility

- Direct SQL statements provide flexibility to interact with the database on-demand without the need to define and manage stored procedures.

3. Less Overhead

- For simple operations or one-time tasks, executing direct SQL statements can be more lightweight compared to creating and maintaining procedures.

4. Real-Time Data Access

- Direct SQL statements provide immediate access to real-time data, as they are executed on-demand.

5. Readability

- For simple operations, direct SQL statements can be more readable and straightforward compared to navigating through procedure definitions.

Choosing Between Procedures and Direct SQL Statements

- Use procedures when you need to encapsulate complex business logic, promote code reuse, enforce security, and optimize performance for frequently executed operations.
- Use direct SQL statements for quick, ad hoc queries, simple operations, and real-time data access. They are suitable when you don't need to reuse the logic across multiple parts of your application.

In many cases, a combination of both procedures and direct SQL statements is used in a database-driven application. Procedures provide a structured way to manage complex operations, while direct SQL statements offer flexibility for various types of queries and operations. The choice depends on the specific requirements of your application and the nature of the tasks you need to perform.

Summary

Procedures are valuable tools for encapsulating and managing business logic within the database. They help improve code organization, performance, and security. However, it's important to strike a balance between using procedures and direct SQL statements, considering factors like application architecture and performance requirements.