Лекция 6

Основы программирования в СУБД. Создание хранимых процедур. Виды хранимых процедур

Цель

Освоить основы программирования в СУБД, изучить создание и использование хранимых процедур, понять их типы и практическое применение для инкапсуляции бизнес-логики на стороне сервера.

Основные вопросы

1. Понятие и преимущества хранимых процедур
2. Синтаксис создания и выполнения хранимых процедур
3. Параметры хранимых процедур: входные, выходные, значения по умолчанию
4. Виды хранимых процедур: системные, пользовательские, расширенные
5. Управление транзакциями в хранимых процедурах
6. Обработка ошибок в хранимых процедурах

Лекция

Введение в программирование СУБД

Программирование в СУБД - создание программных объектов на стороне сервера базы данных для реализации бизнес-логики, валидации данных и оптимизации производительности.

Преимущества серверного программирования:

- Высокая производительность (выполнение рядом с данными);
- Снижение сетевого трафика;
- Централизация бизнес-логики;
- Повышенная безопасность;
- Упрощение клиентских приложений;

Хранимые процедуры: основные понятия

Хранимая процедура - предварительно скомпилированный набор SQL-операторов, хранящийся в базе данных и выполняемый как единое целое.

Ключевые характеристики:

- Постоянное хранение в базе данных;
- Предварительная компиляция (в большинстве СУБД);
- Выполнение на стороне сервера;
- Возможность принимать параметры;
- Возврат результатов и выходных параметров;

Синтаксис создания хранимых процедур

В Microsoft SQL Server

```
CREATE PROCEDURE [schema_name.]procedure_name
    [ @parameter1 data_type [ = default_value ] [ OUTPUT ],
    [ @parameter2 data_type [ = default_value ] [ OUTPUT ],
    ...
AS
BEGIN
    -- Тело процедуры
    SQL-операторы

END
```

В PostgreSQL

```
CREATE OR REPLACE FUNCTION function_name
    (parameter1 data_type, parameter2 data_type DEFAULT default_value)
RETURNS return_type AS
$$
BEGIN
    -- Тело функции
    SQL-операторы;
    RETURN value;
END;
```

```
$$ LANGUAGE plpgsql;
```

В MySQL

```
DELIMITER //
CREATE PROCEDURE procedure_name
   (IN parameter1 data_type, OUT parameter2 data_type)
BEGIN
   -- Тело процедуры
   SQL-операторы;
END //

DELIMITER ;
```

Создание простых хранимых процедур

Пример 1: Процедура без параметров

```
-- SQL Server
CREATE PROCEDURE GetAllEmployees
AS
BEGIN
   SELECT EmployeeID, FirstName, LastName, Department
   FROM Employees
   ORDER BY LastName, FirstName;
END;

-- Вызов процедуры

EXEC GetAllEmployees;
```

Пример 2: Процедура с входными параметрами

```
CREATE PROCEDURE GetEmployeesByDepartment

   @DepartmentName NVARCHAR(50),
   @MinSalary DECIMAL(10,2) = 0
AS
BEGIN
   SELECT EmployeeID, FirstName, LastName, Salary
   FROM Employees
```

```
    WHERE Department = @DepartmentName
      AND Salary >= @MinSalary
    ORDER BY Salary DESC;
END;

-- Вызов с параметрами
EXEC GetEmployeesByDepartment @DepartmentName = 'IT', @MinSalary =
50000;

EXEC GetEmployeesByDepartment 'Sales'; -- Используется значение по
умолчанию
```

Пример 3: Процедура с выходными параметрами

```
CREATE PROCEDURE CalculateDepartmentStats
    @DepartmentName NVARCHAR(50),
    @EmployeeCount INT OUTPUT,
    @AvgSalary DECIMAL(10,2) OUTPUT,
    @TotalSalary DECIMAL(15,2) OUTPUT
AS
BEGIN
    SELECT
        @EmployeeCount = COUNT(*),
        @AvgSalary = AVG(Salary),
        @TotalSalary = SUM(Salary)
    FROM Employees
    WHERE Department = @DepartmentName;

    -- Проверка на случай отсутствия данных
    IF @EmployeeCount = 0
    BEGIN
        SET @AvgSalary = 0;
        SET @TotalSalary = 0;
    END
END;

-- Вызов с выходными параметрами
DECLARE @Count INT, @Avg DECIMAL(10,2), @Total DECIMAL(15,2);
EXEC CalculateDepartmentStats
    @DepartmentName = 'IT',
    @EmployeeCount = @Count OUTPUT,
```

```
    @AvgSalary = @Avg OUTPUT,
    @TotalSalary = @Total OUTPUT;

SELECT @Count AS EmployeeCount, @Avg AS AverageSalary, @Total AS TotalSalary;
```

Виды хранимых процедур

1. Пользовательские хранимые процедуры

Создаются разработчиками для конкретных бизнес-задач.

Классификация пользовательских процедур:

- Процедуры выборки данных - возвращают результирующие наборы;
- Процедуры модификации данных - выполняют INSERT, UPDATE, DELETE;
- Процедуры бизнес-логики - реализуют сложные алгоритмы;
- Процедуры обслуживания - для административных задач;

2. Системные хранимые процедуры

Предопределены в СУБД для административных задач.

Примеры в SQL Server:

- `sp_help` - информация об объектах базы данных;
- `sp_who` - информация о текущих подключениях;
- `sp_rename` - переименование объектов;
- `sp_configure` - настройка сервера;

```
-- Пример использования системной процедуры

EXEC sp_help 'Employees';
```

3. Временные хранимые процедуры

Существуют временно и автоматически удаляются.

Локальные временные процедуры:

- Имена начинаются с `#`

- Видны только в текущем соединении

```
CREATE PROCEDURE #TempProc
AS
BEGIN
   SELECT GETDATE() AS CurrentTime;
END;


EXEC #TempProc;
```

Глобальные временные процедуры:

- Имена начинаются с ##
- Видны во всех соединениях

4. Расширенные хранимые процедуры

Выполняют код вне SQL Server (обычно на C/C++).

Пример: xp_cmdshell - выполнение команд операционной системы

5. CLR-хранимые процедуры

Реализованы на языках .NET Framework.

Управление транзакциями в хранимых процедурах

Явное управление транзакциями

```
CREATE PROCEDURE TransferMoney
   @FromAccount INT,
   @ToAccount INT,
   @Amount DECIMAL(10,2)
AS
BEGIN
   BEGIN TRY
      BEGIN TRANSACTION;

      -- Списание с одного счета
      UPDATE Accounts
      SET Balance = Balance - @Amount
```

```
        WHERE AccountID = @FromAccount;

        -- Проверка отрицательного баланса
        IF (SELECT Balance FROM Accounts WHERE AccountID =
@FromAccount) < 0
        BEGIN
            RAISERROR('Недостаточно средств', 16, 1);
            ROLLBACK TRANSACTION;
            RETURN;
        END

        -- Зачисление на другой счет
        UPDATE Accounts
        SET Balance = Balance + @Amount
        WHERE AccountID = @ToAccount;

        -- Логирование операции
        INSERT INTO TransactionLog (FromAccount, ToAccount, Amount,
TransactionDate)
        VALUES (@FromAccount, @ToAccount, @Amount, GETDATE());

        COMMIT TRANSACTION;

        PRINT 'Транзакция выполнена успешно';
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;

        DECLARE @ErrorMessage NVARCHAR(4000) = ERROR_MESSAGE();
        DECLARE @ErrorSeverity INT = ERROR_SEVERITY();
        DECLARE @ErrorState INT = ERROR_STATE();

        RAISERROR(@ErrorMessage, @ErrorSeverity, @ErrorState);
    END CATCH

END;
```

Обработка ошибок в хранимых процедурах

Механизм TRY...CATCH

```
CREATE PROCEDURE SafeDataUpdate
    @EmployeeID INT,
    @NewSalary DECIMAL(10,2)
AS
BEGIN
    BEGIN TRY
        -- Проверка существования сотрудника
        IF NOT EXISTS (SELECT 1 FROM Employees WHERE EmployeeID =
@EmployeeID)
        BEGIN
            RAISERROR('Сотрудник с ID %d не найден', 16, 1, @EmployeeID);
            RETURN;
        END

        -- Проверка корректности зарплаты
        IF @NewSalary <= 0
        BEGIN
            RAISERROR('Зарплата должна быть положительной', 16, 1);
            RETURN;
        END

        -- Обновление данных
        UPDATE Employees
        SET Salary = @NewSalary,
            LastUpdated = GETDATE()
        WHERE EmployeeID = @EmployeeID;

        PRINT 'Данные успешно обновлены';
    END TRY
    BEGIN CATCH
        DECLARE @ErrorMsg NVARCHAR(4000) = ERROR_MESSAGE();
        DECLARE @ErrorLine INT = ERROR_LINE();
        DECLARE @ErrorProc NVARCHAR(126) = ERROR_PROCEDURE();

        -- Логирование ошибки
        INSERT INTO ErrorLog (ErrorMessage, ErrorLine, ErrorProcedure,
ErrorTime)
        VALUES (@ErrorMsg, @ErrorLine, @ErrorProc, GETDATE());

        -- Повторный выброс ошибки
        THROW;
```

```
    END CATCH
END;
```

Практические примеры сложных процедур

Пример: Процедура массового обновления с валидацией

```
CREATE PROCEDURE BulkUpdateSalaries
    @DepartmentName NVARCHAR(50),
    @IncreasePercent DECIMAL(5,2)
AS
BEGIN
    SET NOCOUNT ON;
    BEGIN TRY
        BEGIN TRANSACTION;
        -- Валидация входных параметров
        IF @IncreasePercent <= 0 OR @IncreasePercent > 100
        BEGIN
            RAISERROR('Процент увеличения должен быть между 0 и 100', 16, 1);
            ROLLBACK TRANSACTION;
            RETURN;
        END
        IF NOT EXISTS (SELECT 1 FROM Departments WHERE Name =
@DepartmentName)
        BEGIN
            RAISERROR('Отдел %s не найден', 16, 1, @DepartmentName);
            ROLLBACK TRANSACTION;
            RETURN;
        END
        -- Создание временной таблицы для аудита
        CREATE TABLE #SalaryChanges (
            EmployeeID INT,
            OldSalary DECIMAL(10,2),
            NewSalary DECIMAL(10,2),
            ChangeDate DATETIME
        );
        -- Обновление зарплат и запись изменений
        UPDATE Employees
        SET Salary = Salary * (1 + @IncreasePercent / 100),
```

```
            LastUpdated = GETDATE()
        OUTPUT inserted.EmployeeID, deleted.Salary, inserted.Salary, GETDATE()
        INTO #SalaryChanges
        WHERE Department = @DepartmentName;
        -- Сохранение аудита в постоянную таблицу
        INSERT INTO SalaryAudit (EmployeeID, OldSalary, NewSalary,
ChangeDate, ChangedBy)
        SELECT EmployeeID, OldSalary, NewSalary, ChangeDate,
SYSTEM_USER
        FROM #SalaryChanges;
        -- Возврат статистики
        SELECT
            COUNT(*) AS EmployeesUpdated,
            SUM(NewSalary - OldSalary) AS TotalIncrease,
            @DepartmentName AS Department
        FROM #SalaryChanges;
        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;
        DECLARE @ErrMsg NVARCHAR(4000) = ERROR_MESSAGE();
        RAISERROR('Ошибка при обновлении зарплат: %s', 16, 1, @ErrMsg);
    END CATCH

END;
```

Управление хранимыми процедурами

Модификация процедур

```
-- Изменение существующей процедуры
ALTER PROCEDURE GetEmployeesByDepartment
    @DepartmentName NVARCHAR(50),
    @MinSalary DECIMAL(10,2) = 0,
    @IncludeInactive BIT = 0
AS
BEGIN
    SELECT EmployeeID, FirstName, LastName, Salary, IsActive
    FROM Employees
    WHERE Department = @DepartmentName
```

```
    AND Salary >= @MinSalary
    AND (IsActive = 1 OR @IncludeInactive = 1)
  ORDER BY Salary DESC;

END;
```

Удаление процедур

```
DROP PROCEDURE GetEmployeesByDepartment;
```

Просмотр информации о процедурах

```
-- Получение определения процедуры
sp_helptext 'GetEmployeesByDepartment';
-- Просмотр зависимостей
sp_depends 'GetEmployeesByDepartment';

-- Список всех процедур в базе
SELECT name, create_date, modify_date
FROM sys.procedures

ORDER BY name;
```

Контрольные вопросы

1. Что такое хранимая процедура и каковы ее основные преимущества?
2. Какие типы параметров могут быть у хранимой процедуры?
3. В чем разница между пользовательскими и системными хранимыми процедурами?
4. Как обрабатывать ошибки в хранимых процедурах?
5. Каким образом управлять транзакциями внутри процедур?
6. Приведите пример практического использования хранимой процедуры.

Литература

1. Дейт К. Дж. Введение в системы баз данных. - Глава 8.
2. Коннолли Т., Бегг К. Базы данных: проектирование, реализация и сопровождение. - Глава 8.
3. Microsoft SQL Server Documentation: Stored Procedures
4. PostgreSQL Documentation: CREATE FUNCTION