Лекция 7

Основы программирования в СУБД. Создание пользовательских функций. Виды функций

Цель

Изучить создание и использование пользовательских функций в СУБД, понять различия между функциями и хранимыми процедурами, освоить различные виды функций и их практическое применение.

Основные вопросы

1. Понятие пользовательских функций и их отличия от хранимых процедур;
2. Скалярные функции: создание и использование;
3. Табличные функции: встроенные и многооператорные;
4. Детерминированные и недетерминированные функции;
5. Встроенные функции СУБД vs пользовательские функции;
6. Оптимизация производительности функций;

Лекция

Введение в пользовательские функции

Пользовательская функция - программный объект базы данных, который принимает параметры, выполняет действия и возвращает результат.

Основные характеристики функций:

- Всегда возвращают значение (скалярное или табличное);
- Могут использоваться в выражениях SQL;
- Не могут выполнять операции изменения данных (в большинстве СУБД);
- Вызываются как часть выражений;

Отличия функций от хранимых процедур

Характеристика	Функции	Хранимые процедуры
Возвращаемое значение	Обязательно	Не обязательно
Использование в SELECT	Да	Нет
Изменение данных	Ограничено	Полностью
Вызов	В выражениях	EXEC/EXECUTE
Транзакции	Ограниченная поддержка	Полная поддержка
OUTPUT параметры	Нет	Да

Скалярные функции

Скалярная функция возвращает единственное значение определенного типа данных.

Создание скалярных функций

В SQL Server:

```
CREATE FUNCTION dbo.CalculateAge
    (@BirthDate DATE)
RETURNS INT
AS
BEGIN
    DECLARE @Age INT;

    SET @Age = DATEDIFF(YEAR, @BirthDate, GETDATE()) -
            CASE
```

```
            WHEN DATEADD(YEAR, DATEDIFF(YEAR, @BirthDate, GETDATE()), @BirthDate) > GETDATE()
            THEN 1
            ELSE 0
        END;

    RETURN @Age;

END;
```

В PostgreSQL:

```
CREATE OR REPLACE FUNCTION calculate_age(birth_date DATE)
RETURNS INTEGER AS
$$
DECLARE
    age INTEGER;
BEGIN
    age := EXTRACT(YEAR FROM AGE(birth_date));
    RETURN age;
END;

$$ LANGUAGE plpgsql;
```

Использование скалярных функций:

```
-- В SELECT
SELECT
    FirstName,
    LastName,
    BirthDate,
    dbo.CalculateAge(BirthDate) AS Age
FROM Employees;

-- В WHERE
SELECT *
FROM Employees
WHERE dbo.CalculateAge(BirthDate) > 30;

-- В ORDER BY
```

```
SELECT *
FROM Employees

ORDER BY dbo.CalculateAge(BirthDate) DESC;
```

Табличные функции

Табличная функция возвращает таблицу в качестве результата.

Встроенные табличные функции (Inline Table-Valued Functions)

Содержат один оператор SELECT.

```
CREATE FUNCTION dbo.GetEmployeesByDepartment
   (@DepartmentName NVARCHAR(50))
RETURNS TABLE
AS
RETURN
   SELECT
      EmployeeID,
      FirstName,
      LastName,
      Salary,
      Email
   FROM Employees

   WHERE Department = @DepartmentName;
```

Использование:

```
SELECT * FROM dbo.GetEmployeesByDepartment('IT');

SELECT * FROM dbo.GetEmployeesByDepartment('Sales') WHERE Salary > 50000;
```

Многооператорные табличные функции (Multi-Statement Table-Valued Functions)

Содержат сложную логику с несколькими операторами.

```
CREATE FUNCTION dbo.GetEmployeeStatistics
    (@DepartmentID INT)
RETURNS @Stats TABLE
    (
        DepartmentName NVARCHAR(50),
        EmployeeCount INT,
        AvgSalary DECIMAL(10,2),
        MaxSalary DECIMAL(10,2),
        MinSalary DECIMAL(10,2)
    )
AS
BEGIN
    INSERT INTO @Stats
    SELECT
        d.Name,
        COUNT(e.EmployeeID),
        AVG(e.Salary),
        MAX(e.Salary),
        MIN(e.Salary)
    FROM Departments d
    LEFT JOIN Employees e ON d.DepartmentID = e.DepartmentID
    WHERE d.DepartmentID = @DepartmentID
    GROUP BY d.Name;

    -- Дополнительная логика
    IF (SELECT EmployeeCount FROM @Stats) = 0
    BEGIN
        INSERT INTO @Stats
        SELECT Name, 0, 0, 0, 0
        FROM Departments
        WHERE DepartmentID = @DepartmentID;
    END

    RETURN;

END;
```

Использование:

```
SELECT * FROM dbo.GetEmployeeStatistics(1);
```

Детерминированные и недетерминированные функции

Детерминированная функция всегда возвращает одинаковый результат для одинаковых входных параметров.

Примеры детерминированных функций:

- Математические операции
- Работа со строками (LEFT, UPPER)
- Пользовательские функции без внешних зависимостей

Недетерминированная функция может возвращать разные результаты при одинаковых параметрах.

Примеры недетерминированных функций:

- GETDATE()
- NEWID()
- RAND()
- Функции, обращающиеся к изменяющимся данным

Создание детерминированной функции:

```
CREATE FUNCTION dbo.CalculateVAT
   (@Amount DECIMAL(10,2), @VATRate DECIMAL(5,2))
RETURNS DECIMAL(10,2)
WITH SCHEMABINDING
AS
BEGIN
   RETURN @Amount * @VATRate / 100;

END;
```

Практические примеры пользовательских функций

Пример 1: Функция для форматирования данных

```
CREATE FUNCTION dbo.FormatPhoneNumber

   (@PhoneNumber NVARCHAR(20))
RETURNS NVARCHAR(20)
AS
BEGIN
```

```
    -- Удаление всех нецифровых символов
    SET @PhoneNumber =
REPLACE(REPLACE(REPLACE(REPLACE(@PhoneNumber, ' ', ''), '-', ''), '(', ''),
')', '');

    -- Форматирование в международный формат
    IF LEN(@PhoneNumber) = 10
        SET @PhoneNumber = '+7 (' + SUBSTRING(@PhoneNumber, 1, 3) + ') ' +
                           SUBSTRING(@PhoneNumber, 4, 3) + '-' +
                           SUBSTRING(@PhoneNumber, 7, 2) + '-' +
                           SUBSTRING(@PhoneNumber, 9, 2);
    ELSE IF LEN(@PhoneNumber) = 11 AND LEFT(@PhoneNumber, 1) = '7'
        SET @PhoneNumber = '+7 (' + SUBSTRING(@PhoneNumber, 2, 3) + ') ' +
                           SUBSTRING(@PhoneNumber, 5, 3) + '-' +
                           SUBSTRING(@PhoneNumber, 8, 2) + '-' +
                           SUBSTRING(@PhoneNumber, 10, 2);

    RETURN @PhoneNumber;

END;
```

Пример 2: Функция для сложных расчетов

```
CREATE FUNCTION dbo.CalculateEmployeeBonus
    (@EmployeeID INT, @Year INT)
RETURNS DECIMAL(10,2)
AS
BEGIN
    DECLARE @Bonus DECIMAL(10,2);
    DECLARE @BaseSalary DECIMAL(10,2);
    DECLARE @PerformanceRating DECIMAL(3,2);
    DECLARE @YearsService INT;

    -- Получение базовых данных
    SELECT
        @BaseSalary = Salary,
        @YearsService = DATEDIFF(YEAR, HireDate,
DATEFROMPARTS(@Year, 12, 31))
    FROM Employees
    WHERE EmployeeID = @EmployeeID;
```

```
    -- Получение рейтинга производительности
    SELECT @PerformanceRating = AVG(Rating)
    FROM PerformanceReviews
    WHERE EmployeeID = @EmployeeID
      AND YEAR(ReviewDate) = @Year;

    -- Расчет бонуса по сложной формуле
    SET @Bonus = @BaseSalary * 0.1; -- Базовый бонус 10%

    -- Надбавка за стаж
    IF @YearsService > 5
        SET @Bonus = @Bonus + (@BaseSalary * 0.02 * (@YearsService - 5));

    -- Надбавка за производительность
    IF @PerformanceRating > 4.0
        SET @Bonus = @Bonus * 1.2;
    ELSE IF @PerformanceRating > 3.0
        SET @Bonus = @Bonus * 1.1;

    RETURN ROUND(@Bonus, 2);

END;
```

Пример 3: Рекурсивная функция для иерархических данных

```
CREATE FUNCTION dbo.GetManagerHierarchy
    (@EmployeeID INT)
RETURNS @Hierarchy TABLE
    (
        Level INT,
        EmployeeID INT,
        ManagerID INT,
        EmployeeName NVARCHAR(100),
        ManagerName NVARCHAR(100)
    )
AS
BEGIN
    WITH EmployeeCTE AS
    (
        -- Якорь рекурсии
        SELECT
```

```
        1 AS Level,
        e.EmployeeID,
        e.ManagerID,
        e.FirstName + ' ' + e.LastName AS EmployeeName,
        m.FirstName + ' ' + m.LastName AS ManagerName
    FROM Employees e
    LEFT JOIN Employees m ON e.ManagerID = m.EmployeeID
    WHERE e.EmployeeID = @EmployeeID

    UNION ALL

    -- Рекурсивная часть
    SELECT
        ec.Level + 1,
        e.EmployeeID,
        e.ManagerID,
        e.FirstName + ' ' + e.LastName,
        m.FirstName + ' ' + m.LastName
    FROM Employees e
    INNER JOIN EmployeeCTE ec ON e.EmployeeID = ec.ManagerID
    LEFT JOIN Employees m ON e.ManagerID = m.EmployeeID
    WHERE ec.ManagerID IS NOT NULL
)
INSERT INTO @Hierarchy
SELECT * FROM EmployeeCTE;

RETURN;

END;
```

Использование:

```
SELECT * FROM dbo.GetManagerHierarchy(123) ORDER BY Level DESC;
```

Управление функциями

Создание и изменение функций

```
-- Создание функции
CREATE FUNCTION dbo.ExampleFunction ()
RETURNS INT
```

```
AS
BEGIN
    RETURN 1;
END;

-- Изменение функции
ALTER FUNCTION dbo.ExampleFunction ()
RETURNS INT
AS
BEGIN
    RETURN 2;
END;

-- Удаление функции

DROP FUNCTION dbo.ExampleFunction;
```

Просмотр информации о функциях

```
-- Получение определения функции
SELECT definition
FROM sys.sql_modules
WHERE object_id = OBJECT_ID('dbo.CalculateAge');

-- Просмотр зависимостей
SELECT
    OBJECT_NAME(referencing_id) AS ReferencingObject,
    referenced_entity_name AS ReferencedObject
FROM sys.sql_expression_dependencies
WHERE referenced_entity_name = 'CalculateAge';

-- Список всех функций
SELECT
    name,
    type_desc,
    create_date,
    modify_date
FROM sys.objects
WHERE type IN ('FN', 'IF', 'TF') -- Scalar, Inline Table, Table-valued

ORDER BY name;
```

Оптимизация производительности функций

Рекомендации по производительности

1. Избегайте скалярных функций в запросах к большим таблицам
    - Скалярные функции выполняются для каждой строки
    - Рассмотрите использование встроенных функций или JOIN
2. Используйте схемопривязку (SCHEMABINDING)

```
CREATE FUNCTION dbo.FastFunction ()
RETURNS INT
WITH SCHEMABINDING
AS
BEGIN
   RETURN 1;

       END;
```

3. Минимизируйте сложные вычисления в функциях;
4. Используйте встроенные табличные функции вместо многооператорных;

Пример оптимизации

Медленно (скалярная функция):

```
-- Функция выполняется для каждой строки
SELECT
   OrderID,
   dbo.CalculateOrderTotal(OrderID) AS Total
FROM Orders

WHERE OrderDate >= '2023-01-01';
```

Быстро (JOIN с агрегацией):

```
SELECT

   o.OrderID,
   SUM(od.Quantity * od.UnitPrice) AS Total
FROM Orders o
JOIN OrderDetails od ON o.OrderID = od.OrderID
```

```
WHERE o.OrderDate >= '2023-01-01'
GROUP BY o.OrderID;
```

Контрольные вопросы

1. В чем основные различия между функциями и хранимыми процедурами?
2. Какие типы пользовательских функций существуют и когда их использовать?
3. Что такое детерминированные и недетерминированные функции?
4. Как создавать и использовать табличные функции?
5. Какие ограничения накладываются на функции в различных СУБД?
6. Как оптимизировать производительность пользовательских функций?

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: User-Defined Functions;
4. PostgreSQL Documentation: CREATE FUNCTION;