Лекция 10

Работа с курсором. Применение курсора. Управление курсором

Цель

Изучить концепцию курсоров в SQL, научиться создавать и управлять курсорами, понять их применение для построчной обработки данных и освоить альтернативные подходы.

Основные вопросы

1. Понятие курсора и его назначение;
2. Типы курсоров: STATIC, DYNAMIC, KEYSET, FAST_FORWARD;
3. Жизненный цикл курсора;
4. Управление курсорами: объявление, открытие, выборка, закрытие;
5. Применение курсоров в хранимых процедурах и функциях;
6. Альтернативы курсорам и лучшие практики;

Лекция

Введение в курсоры

Курсор - объект базы данных, который позволяет поочередно обрабатывать строки результирующего набора.

Основные характеристики курсоров:

- Построчный доступ к данным
- Возможность сложных преобразований
- Контроль над порядком обработки
- Интеграция с процедурной логикой

Сценарии использования курсоров:

- Сложные бизнес-расчеты для каждой строки
- Построчная обработка иерархических данных

- Миграция и преобразование данных
- Генерация сложных отчетов
- Интеграция с внешними системами

Типы курсоров

По области видимости

Локальные курсоры: Видны только в текущем соединении, пакете или процедуре.

Глобальные курсоры: Видны во всех соединениях.

По возможности изменения данных

STATIC: Снимок данных на момент открытия, изменения не видны.

DYNAMIC: Отображает все изменения данных в реальном времени.

KEYSET: Сохраняет ключи строк, позволяет видеть изменения, но не вставки.

FAST_FORWARD: Оптимизированный курсор только для чтения вперед.

По направлению движения

FORWARD_ONLY: Движение только вперед.

SCROLL: Возможность движения в любом направлении.

Жизненный цикл курсора

1. Объявление курсора

Синтаксис объявления:

```
DECLARE cursor_name CURSOR
```

```
[ LOCAL | GLOBAL ]
[ FORWARD_ONLY | SCROLL ]
[ STATIC | KEYSET | DYNAMIC | FAST_FORWARD ]
[ READ_ONLY | SCROLL_LOCKS | OPTIMISTIC ]
FOR select_statement

[ FOR UPDATE [ OF column_name [ ,...n ] ] ];
```

Примеры объявления:

```
-- Простой курсор

DECLARE EmployeeCursor CURSOR
FOR
   SELECT EmployeeID, FirstName, LastName, Salary
   FROM Employees
   WHERE DepartmentID = 5;

-- Курсор с возможностью прокрутки
DECLARE ScrollCursor CURSOR
SCROLL STATIC
FOR
   SELECT EmployeeID, FirstName, LastName
   FROM Employees
   ORDER BY LastName, FirstName;

-- Курсор с возможностью обновления
DECLARE UpdateCursor CURSOR
FOR
   SELECT EmployeeID, Salary
   FROM Employees
   WHERE DepartmentID = 3

FOR UPDATE OF Salary;
```

2. Открытие курсора

```
OPEN EmployeeCursor;
```

3. Выборка данных

Операции FETCH:

- FETCH NEXT - следующая строка
- FETCH PRIOR - предыдущая строка
- FETCH FIRST - первая строка
- FETCH LAST - последняя строка
- FETCH ABSOLUTE n - строка номер n
- FETCH RELATIVE n - строка со смещением n

```
-- Выборка в переменные
DECLARE @EmployeeID INT, @FirstName NVARCHAR(50), @LastName NVARCHAR(50), @Salary DECIMAL(10,2);

FETCH NEXT FROM EmployeeCursor
INTO @EmployeeID, @FirstName, @LastName, @Salary;
```

4. Закрытие курсора

```
CLOSE EmployeeCursor;
```

5. Освобождение ресурсов

```
DEALLOCATE EmployeeCursor;
```

Полный пример работы с курсором

```
-- Объявление переменных
DECLARE @EmployeeID INT;
DECLARE @FirstName NVARCHAR(50);
DECLARE @LastName NVARCHAR(50);
DECLARE @Salary DECIMAL(10,2);
DECLARE @TotalSalary DECIMAL(15,2) = 0;
DECLARE @EmployeeCount INT = 0;

-- Объявление курсора
DECLARE EmployeeCursor CURSOR
FOR
    SELECT EmployeeID, FirstName, LastName, Salary
    FROM Employees
    WHERE DepartmentID = 5
    ORDER BY Salary DESC;
```

```
-- Открытие курсора
OPEN EmployeeCursor;

-- Первая выборка
FETCH NEXT FROM EmployeeCursor
INTO @EmployeeID, @FirstName, @LastName, @Salary;

-- Обработка данных
WHILE @@FETCH_STATUS = 0
BEGIN
    -- Логика обработки
    SET @EmployeeCount = @EmployeeCount + 1;
    SET @TotalSalary = @TotalSalary + @Salary;

    -- Вывод информации
    PRINT CONCAT('Сотрудник: ', @FirstName, ' ', @LastName, ', Зарплата: ',
@Salary);

    -- Следующая выборка
    FETCH NEXT FROM EmployeeCursor
    INTO @EmployeeID, @FirstName, @LastName, @Salary;
END;

-- Закрытие и освобождение курсора
CLOSE EmployeeCursor;
DEALLOCATE EmployeeCursor;

-- Вывод результата
PRINT CONCAT('Обработано сотрудников: ', @EmployeeCount);
PRINT CONCAT('Общая зарплата отдела: ', @TotalSalary);

PRINT CONCAT('Средняя зарплата: ', @TotalSalary /
NULLIF(@EmployeeCount, 0));
```

Системные функции для работы с курсорами

@@FETCH_STATUS: Возвращает статус последней операции FETCH

- 0 = успешно

- -1 = строка не найдена
- -2 = строка удалена

@@CURSOR_ROWS: Возвращает количество строк в последнем открытом курсоре

CURSOR_STATUS: Возвращает статус курсора

Пример использования:

```
DECLARE SampleCursor CURSOR FOR
SELECT TOP 5 EmployeeID, FirstName FROM Employees;

OPEN SampleCursor;

-- Проверка количества строк
SELECT @@CURSOR_ROWS AS CursorRowCount;

FETCH NEXT FROM SampleCursor;
WHILE @@FETCH_STATUS = 0
BEGIN
   FETCH NEXT FROM SampleCursor;
END;

CLOSE SampleCursor;

DEALLOCATE SampleCursor;
```

Применение курсоров в хранимых процедурах

Пример: Процедура для расчета бонусов

```
CREATE PROCEDURE CalculateEmployeeBonuses
   @DepartmentID INT,
   @BonusRate DECIMAL(5,2)
AS
BEGIN
   SET NOCOUNT ON;

   DECLARE @EmployeeID INT;
   DECLARE @Salary DECIMAL(10,2);
```

```
    DECLARE @Bonus DECIMAL(10,2);
    DECLARE @ProcessedCount INT = 0;

    DECLARE BonusCursor CURSOR
    FOR
        SELECT EmployeeID, Salary
        FROM Employees
        WHERE DepartmentID = @DepartmentID
          AND IsActive = 1;

    OPEN BonusCursor;
    FETCH NEXT FROM BonusCursor INTO @EmployeeID, @Salary;

    WHILE @@FETCH_STATUS = 0
    BEGIN
        -- Расчет бонуса
        SET @Bonus = @Salary * @BonusRate / 100;

        -- Обновление записи
        UPDATE Employees
        SET Bonus = @Bonus,
            LastBonusDate = GETDATE()
        WHERE EmployeeID = @EmployeeID;

        -- Логирование
        INSERT INTO BonusLog (EmployeeID, BonusAmount, CalculationDate)
        VALUES (@EmployeeID, @Bonus, GETDATE());

        SET @ProcessedCount = @ProcessedCount + 1;
        FETCH NEXT FROM BonusCursor INTO @EmployeeID, @Salary;
    END;

    CLOSE BonusCursor;
    DEALLOCATE BonusCursor;

    PRINT CONCAT('Обработано сотрудников: ', @ProcessedCount);

END;
```

Курсоры в функциях

Ограничение: В функциях можно использовать только курсоры STATIC и READ_ONLY.

Пример функции с курсором:

```
CREATE FUNCTION dbo.GetDepartmentSalarySummary
   (@DepartmentID INT)
RETURNS @Summary TABLE
   (
      EmployeeCount INT,
      TotalSalary DECIMAL(15,2),
      AverageSalary DECIMAL(10,2),
      MaxSalary DECIMAL(10,2),
      MinSalary DECIMAL(10,2)
   )
AS
BEGIN
   DECLARE @EmployeeCount INT = 0;
   DECLARE @TotalSalary DECIMAL(15,2) = 0;
   DECLARE @MaxSalary DECIMAL(10,2) = 0;
   DECLARE @MinSalary DECIMAL(10,2) = 0;
   DECLARE @Salary DECIMAL(10,2);
   DECLARE @FirstRow BIT = 1;

   DECLARE SalaryCursor CURSOR STATIC READ_ONLY
   FOR
      SELECT Salary
      FROM Employees
      WHERE DepartmentID = @DepartmentID
        AND IsActive = 1;

   OPEN SalaryCursor;
   FETCH NEXT FROM SalaryCursor INTO @Salary;

   WHILE @@FETCH_STATUS = 0
   BEGIN
      SET @EmployeeCount = @EmployeeCount + 1;
      SET @TotalSalary = @TotalSalary + @Salary;

      -- Инициализация min/max для первой строки
      IF @FirstRow = 1
```

```
        BEGIN
            SET @MaxSalary = @Salary;
            SET @MinSalary = @Salary;
            SET @FirstRow = 0;
        END
        ELSE
        BEGIN
            IF @Salary > @MaxSalary
                SET @MaxSalary = @Salary;
            IF @Salary < @MinSalary
                SET @MinSalary = @Salary;
        END

        FETCH NEXT FROM SalaryCursor INTO @Salary;
    END;

    CLOSE SalaryCursor;
    DEALLOCATE SalaryCursor;

    INSERT INTO @Summary
    SELECT
        @EmployeeCount,
        @TotalSalary,
        CASE WHEN @EmployeeCount > 0 THEN @TotalSalary /
@EmployeeCount ELSE 0 END,
        @MaxSalary,
        @MinSalary;

    RETURN;

END;
```

Управление курсорами

Параметры курсоров

SCROLL: Позволяет перемещаться в любом направлении.

READ_ONLY: Запрещает изменения через курсор.

SCROLL_LOCKS: Блокирует строки для обеспечения возможности обновления.

OPTIMISTIC: Оптимистичная блокировка, проверяет изменения при обновлении.

Обновление данных через курсор

```
CREATE PROCEDURE UpdateEmployeeSalaries
    @DepartmentID INT,
    @IncreasePercent DECIMAL(5,2)
AS
BEGIN
    DECLARE @EmployeeID INT;
    DECLARE @CurrentSalary DECIMAL(10,2);
    DECLARE @NewSalary DECIMAL(10,2);

    DECLARE UpdateCursor CURSOR
    FOR
        SELECT EmployeeID, Salary
        FROM Employees
        WHERE DepartmentID = @DepartmentID
        FOR UPDATE OF Salary;

    OPEN UpdateCursor;
    FETCH NEXT FROM UpdateCursor INTO @EmployeeID, @CurrentSalary;

    WHILE @@FETCH_STATUS = 0
    BEGIN
        -- Расчет новой зарплаты
        SET @NewSalary = @CurrentSalary * (1 + @IncreasePercent / 100);

        -- Обновление через курсор
        UPDATE Employees
        SET Salary = @NewSalary
        WHERE CURRENT OF UpdateCursor;

        -- Логирование изменения
        INSERT INTO SalaryHistory (EmployeeID, OldSalary, NewSalary,
ChangeDate)
        VALUES (@EmployeeID, @CurrentSalary, @NewSalary, GETDATE());
```

```
        FETCH NEXT FROM UpdateCursor INTO @EmployeeID, @CurrentSalary;
    END;

    CLOSE UpdateCursor;
    DEALLOCATE UpdateCursor;

END;
```

Обработка ошибок в курсорах

```
CREATE PROCEDURE SafeCursorProcessing
AS
BEGIN
    DECLARE @EmployeeID INT;
    DECLARE @FirstName NVARCHAR(50);
    DECLARE @ProcessedCount INT = 0;
    DECLARE @ErrorCount INT = 0;

    DECLARE SafeCursor CURSOR
    FOR
        SELECT EmployeeID, FirstName
        FROM Employees;

    BEGIN TRY
        OPEN SafeCursor;
        FETCH NEXT FROM SafeCursor INTO @EmployeeID, @FirstName;

        WHILE @@FETCH_STATUS = 0
        BEGIN
            BEGIN TRY
                -- Сложная логика обработки
                IF LEN(@FirstName) < 2
                    RAISERROR('Слишком короткое имя', 16, 1);

                -- Успешная обработка
                SET @ProcessedCount = @ProcessedCount + 1;

            END TRY
            BEGIN CATCH
                -- Обработка ошибки для конкретной строки
                SET @ErrorCount = @ErrorCount + 1;
```

```
            INSERT INTO ProcessingErrors (EmployeeID, ErrorMessage,
ErrorTime)
            VALUES (@EmployeeID, ERROR_MESSAGE(), GETDATE());

            -- Продолжаем обработку следующих строк
        END CATCH

        FETCH NEXT FROM SafeCursor INTO @EmployeeID, @FirstName;
    END;

    CLOSE SafeCursor;
    DEALLOCATE SafeCursor;

    PRINT CONCAT('Успешно обработано: ', @ProcessedCount);
    PRINT CONCAT('Ошибок обработки: ', @ErrorCount);

  END TRY
  BEGIN CATCH
    -- Обработка критических ошибок
    IF CURSOR_STATUS('local', 'SafeCursor') >= 0
    BEGIN
      CLOSE SafeCursor;
      DEALLOCATE SafeCursor;
    END

    DECLARE @ErrorMessage NVARCHAR(4000) = ERROR_MESSAGE();
    RAISERROR('Критическая ошибка: %s', 16, 1, @ErrorMessage);
  END CATCH

END;
```

Пример: Курсор для обработки иерархических данных

```
CREATE PROCEDURE ProcessOrganizationHierarchy
  @RootEmployeeID INT
AS
BEGIN
  DECLARE @EmployeeID INT;
  DECLARE @ManagerID INT;
  DECLARE @FirstName NVARCHAR(50);
  DECLARE @LastName NVARCHAR(50);
```

```
    DECLARE @Level INT = 0;

    -- Создание временной таблицы для хранения уровней
    CREATE TABLE #EmployeeLevels (
        EmployeeID INT,
        ManagerID INT,
        FirstName NVARCHAR(50),
        LastName NVARCHAR(50),
        Level INT,
        HierarchyPath NVARCHAR(500)
    );

    -- Курсор для обхода иерархии в ширину
    DECLARE HierarchyCursor CURSOR FOR
    WITH EmployeeCTE AS
    (
        SELECT
            EmployeeID,
            ManagerID,
            FirstName,
            LastName,
            0 AS Level,
            CAST(FirstName + ' ' + LastName AS NVARCHAR(500)) AS
HierarchyPath
        FROM Employees
        WHERE EmployeeID = @RootEmployeeID

        UNION ALL

        SELECT
            e.EmployeeID,
            e.ManagerID,
            e.FirstName,
            e.LastName,
            ec.Level + 1,
            CAST(ec.HierarchyPath + ' -> ' + e.FirstName + ' ' + e.LastName AS
NVARCHAR(500))
        FROM Employees e
        INNER JOIN EmployeeCTE ec ON e.ManagerID = ec.EmployeeID
    )
```

```
    SELECT EmployeeID, ManagerID, FirstName, LastName, Level,
HierarchyPath
    FROM EmployeeCTE
    ORDER BY Level, LastName, FirstName;

    OPEN HierarchyCursor;
    FETCH NEXT FROM HierarchyCursor INTO @EmployeeID, @ManagerID,
@FirstName, @LastName, @Level, @HierarchyPath;

    WHILE @@FETCH_STATUS = 0
    BEGIN
        -- Вставка в временную таблицу
        INSERT INTO #EmployeeLevels (EmployeeID, ManagerID, FirstName,
LastName, Level, HierarchyPath)
        VALUES (@EmployeeID, @ManagerID, @FirstName, @LastName,
@Level, @HierarchyPath);

        -- Дополнительная обработка для каждого сотрудника
        PRINT CONCAT('Обработан: ', @HierarchyPath, ' (Уровень ', @Level, ')');

        FETCH NEXT FROM HierarchyCursor INTO @EmployeeID, @ManagerID,
@FirstName, @LastName, @Level, @HierarchyPath;
    END;

    CLOSE HierarchyCursor;
    DEALLOCATE HierarchyCursor;

    -- Использование результатов
    SELECT * FROM #EmployeeLevels ORDER BY Level, LastName, FirstName;

    DROP TABLE #EmployeeLevels;

END;
```

Рекомендации по использованию курсоров

Когда использовать курсоры:

- Сложные бизнес-процессы, требующие построчной логики;
- Миграция данных между различными системами;
- Генерация сложных отчетов;
- Интеграция с внешними системами;

- Обработка иерархических данных;

Когда избегать курсоров:

- Массовые операции с данными;
- Операции агрегации;
- Ежедневные бизнес-процессы;
- Высоконагруженные системы;

Оптимизация курсоров:

- Использование FAST_FORWARD для только чтения;
- Минимизация объема данных в курсоре;
- Быстрое закрытие курсора;
- Использование локальных курсоров;
- Правильный выбор типа курсора;

Лучшие практики:

1. Всегда закрывайте и освобождайте курсоры;
2. Обрабатывайте ошибки в цикле курсора;
3. Используйте наиболее эффективный тип курсора;
4. Избегайте курсоров в циклах;
5. Рассматривайте SET-ориентированные альтернативы;

Контрольные вопросы

1. Что такое курсор и в каких случаях его следует использовать?
2. Опишите жизненный цикл курсора и основные операции с ним.
3. Какие типы курсоров существуют и чем они отличаются?
4. Как обрабатывать ошибки при работе с курсорами?
5. Какие альтернативы курсорам вы знаете и когда их применять?
6. Приведите пример практической задачи, где использование курсора оправдано.

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: DECLARE CURSOR
4. PostgreSQL Documentation: DECLARE