Лекция 9

Обеспечение целостности данных. Триггеры. Типы триггеров. Прямая и косвенная рекурсия

Цель

Изучить механизмы обеспечения целостности данных с помощью триггеров, понять различные типы триггеров и их применение, освоить концепции рекурсии и методы управления сложной бизнес-логикой.

Основные вопросы

1. Понятие триггеров и их роль в обеспечении целостности;
2. Типы триггеров: DML, DDL, LOGON;
3. Триггеры INSTEAD OF и AFTER;
4. Прямая и косвенная рекурсия в триггерах;
5. Управление вложенностью и обработка ошибок;
6. Практические примеры использования триггеров;

Лекция

Введение в триггеры

Триггер - специальный тип хранимой процедуры, который автоматически выполняется при наступлении определенного события в базе данных.

Основные характеристики триггеров:

- Автоматическое выполнение
- Не могут быть вызваны явно
- Выполняются в контексте транзакции
- Могут влиять на выполнение инициирующей операции

Назначение триггеров:

- Обеспечение сложной бизнес-логики

- Аудит изменений данных
- Обеспечение ссылочной целостности
- Реализация каскадных операций
- Валидация сложных ограничений

Типы триггеров

DML триггеры (Data Manipulation Language)

Выполняются при операциях INSERT, UPDATE, DELETE.

Синтаксис создания DML триггера:

```
CREATE TRIGGER [schema_name.]trigger_name
ON { table | view }
[ WITH <dml_trigger_option> [ ,...n ] ]
{ FOR | AFTER | INSTEAD OF }
{ [ INSERT ] [ , ] [ UPDATE ] [ , ] [ DELETE ] }
AS

{ sql_statement [ ; ] [ ,...n ] }
```

DDL триггеры (Data Definition Language)

Выполняются при операциях CREATE, ALTER, DROP.

Синтаксис создания DDL триггера:

```
CREATE TRIGGER trigger_name

ON { ALL SERVER | DATABASE }
[ WITH <ddl_trigger_option> [ ,...n ] ]
{ FOR | AFTER } { event_type | event_group }
AS

{ sql_statement [ ; ] [ ,...n ] }
```

LOGON триггеры

Выполняются при событиях аутентификации.

```
CREATE TRIGGER trigger_name
```

```
ON ALL SERVER
[ WITH <logon_trigger_option> [ ,...n ] ]
{ FOR | AFTER } LOGON
AS

{ sql_statement [ ; ] [ ,...n ] }
```

Триггеры AFTER и INSTEAD OF

Триггеры AFTER

- Выполняются после завершения операции;
- Могут быть использованы только на таблицах;
- Видят результаты операции;

Пример триггера AFTER для аудита

```
CREATE TRIGGER trg_Employees_Audit
ON Employees
AFTER INSERT, UPDATE, DELETE
AS
BEGIN
   SET NOCOUNT ON;

   DECLARE @OperationType CHAR(1);

   -- Определение типа операции
   IF EXISTS (SELECT * FROM inserted) AND EXISTS (SELECT * FROM
deleted)
      SET @OperationType = 'U'; -- Update
   ELSE IF EXISTS (SELECT * FROM inserted)
      SET @OperationType = 'I'; -- Insert
   ELSE
      SET @OperationType = 'D'; -- Delete

   -- Запись в таблицу аудита
   INSERT INTO AuditLog (TableName, OperationType, OperationDate,
UserName)
   VALUES ('Employees', @OperationType, GETDATE(), SYSTEM_USER);

   -- Для операций UPDATE записываем детали изменений
   IF @OperationType = 'U'
```

```
    BEGIN
        INSERT INTO EmployeeChanges (EmployeeID, ChangeDate, ChangedBy,
OldSalary, NewSalary)
        SELECT
            i.EmployeeID,
            GETDATE(),
            SYSTEM_USER,
            d.Salary,
            i.Salary
        FROM inserted i
        JOIN deleted d ON i.EmployeeID = d.EmployeeID
        WHERE i.Salary != d.Salary;
    END

END;
```

Триггеры INSTEAD OF

- Выполняются вместо операции
- Могут быть использованы на таблицах и представлениях
- Полезны для обновления необновляемых представлений

Пример триггера INSTEAD OF:

```
CREATE TRIGGER trg_Employees_InsteadOfDelete
ON Employees
INSTEAD OF DELETE
AS
BEGIN
    SET NOCOUNT ON;

    -- Вместо удаления помечаем запись как неактивную
    UPDATE Employees
    SET IsActive = 0,
        TerminationDate = GETDATE(),
        LastUpdated = GETDATE()
    WHERE EmployeeID IN (SELECT EmployeeID FROM deleted);

    PRINT 'Записи помечены как неактивные вместо удаления';

END;
```

Специальные таблицы inserted и deleted

Таблица inserted содержит новые значения для операций INSERT и UPDATE.

Таблица deleted содержит старые значения для операций UPDATE и DELETE.

Особенности работы с inserted и deleted:

- Доступны только в контексте триггера;
- Для операций UPDATE содержат соответственно новые и старые значения;
- Для массовых операций содержат все затрагиваемые строки;

Пример использования:

```
CREATE TRIGGER trg_Employees_SalaryCheck
ON Employees
AFTER UPDATE
AS
BEGIN
    IF UPDATE(Salary)
    BEGIN
        -- Проверяем, что зарплата не уменьшается
        IF EXISTS (
            SELECT 1
            FROM inserted i
            JOIN deleted d ON i.EmployeeID = d.EmployeeID
            WHERE i.Salary < d.Salary
        )
        BEGIN
            RAISERROR('Зарплата не может быть уменьшена', 16, 1);
            ROLLBACK TRANSACTION;
        END

        -- Проверяем максимальное увеличение зарплаты (не более 50%)
        IF EXISTS (
            SELECT 1
            FROM inserted i
            JOIN deleted d ON i.EmployeeID = d.EmployeeID
            WHERE i.Salary > d.Salary * 1.5
```

```
        )
        BEGIN
            RAISERROR('Увеличение зарплаты не может превышать 50%', 16, 1);
            ROLLBACK TRANSACTION;
        END
    END

END;
```

Рекурсия в триггерах

Прямая рекурсия

Возникает, когда триггер выполняет операцию, которая вызывает тот же триггер.

Пример прямой рекурсии:

```
CREATE TRIGGER trg_Employees_UpdateTimestamp
ON Employees
AFTER UPDATE
AS
BEGIN
    -- Этот UPDATE вызовет тот же триггер снова
    UPDATE Employees
    SET LastUpdated = GETDATE()
    WHERE EmployeeID IN (SELECT EmployeeID FROM inserted);

END;
```

Косвенная рекурсия

Возникает, когда триггер таблицы A выполняет операцию на таблице B, а триггер таблицы B выполняет операцию на таблице A.

Пример косвенной рекурсии:

```
-- Триггер для таблицы Orders
CREATE TRIGGER trg_Orders_Insert
ON Orders
AFTER INSERT
AS
```

```
BEGIN
    -- Обновляем дату последнего заказа у клиента
    UPDATE Customers
    SET LastOrderDate = GETDATE()
    WHERE CustomerID IN (SELECT CustomerID FROM inserted);
END;

-- Триггер для таблицы Customers
CREATE TRIGGER trg_Customers_Update
ON Customers
AFTER UPDATE
AS
BEGIN
    IF UPDATE(LastOrderDate)
    BEGIN
        -- Логируем изменение даты заказа
        INSERT INTO CustomerHistory (CustomerID, Action, ActionDate)
        SELECT CustomerID, 'OrderDate Updated', GETDATE()
        FROM inserted;
    END

END;
```

Управление рекурсией

Отключение рекурсии на уровне базы данных:

```
ALTER DATABASE MyDatabase SET RECURSIVE_TRIGGERS OFF;
```

Отключение рекурсии на уровне соединения:

```
SET RECURSIVE_TRIGGERS OFF;
```

Предотвращение бесконечной рекурсии:

```
CREATE TRIGGER trg_Employees_SafeUpdate
ON Employees
AFTER UPDATE
AS
BEGIN
    SET NOCOUNT ON;
```

```
    -- Проверяем уровень вложенности
    IF TRIGGER_NESTLEVEL(OBJECT_ID('trg_Employees_SafeUpdate')) > 1
        RETURN;

    -- Логика триггера выполняется только для первого вызова
    INSERT INTO EmployeeHistory (EmployeeID, ChangeDate, ChangeType)
    SELECT EmployeeID, GETDATE(), 'UPDATE'
    FROM inserted;

END;
```

Вложенность триггеров

Уровень вложенности - количество триггеров, вызванных в цепочке.

Ограничение вложенности в SQL Server: 32 уровня.

Мониторинг вложенности:

```
-- Текущий уровень вложенности
SELECT TRIGGER_NESTLEVEL() AS NestLevel;

-- Уровень вложенности для конкретного триггера
SELECT TRIGGER_NESTLEVEL(OBJECT_ID('trg_MyTrigger')) AS
SpecificNestLevel;
```

Обработка ошибок в триггерах

```
CREATE TRIGGER trg_Products_PriceValidation
ON Products
AFTER INSERT, UPDATE
AS
BEGIN
    BEGIN TRY
        -- Проверка отрицательной цены
        IF EXISTS (SELECT 1 FROM inserted WHERE Price < 0)
        BEGIN
            RAISERROR('Цена не может быть отрицательной', 16, 1);
        END
```

```
        -- Проверка слишком высокой цены
        IF EXISTS (SELECT 1 FROM inserted WHERE Price > 1000000)
        BEGIN
            RAISERROR('Цена превышает максимально допустимую', 16, 1);
        END

        -- Проверка согласованности цен
        IF EXISTS (
            SELECT 1
            FROM inserted i
            WHERE i.Price < i.Cost * 1.1  -- Минимальная наценка 10%
        )
        BEGIN
            RAISERROR('Цена должна быть как минимум на 10% выше себестоимости', 16, 1);
        END
    END TRY
    BEGIN CATCH
        -- Откат транзакции и передача ошибки
        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;
```

Практические примеры использования триггеров

Пример 1: Триггер для обеспечения сложного ограничения

```
CREATE TRIGGER trg_Employees_ManagerCheck
ON Employees
AFTER INSERT, UPDATE
AS
BEGIN
```

```
    -- Проверяем, что менеджер принадлежит тому же отделу
    IF EXISTS (
        SELECT 1
        FROM inserted i
        JOIN Employees m ON i.ManagerID = m.EmployeeID
        WHERE i.DepartmentID != m.DepartmentID
          AND m.DepartmentID IS NOT NULL
    )
    BEGIN
        RAISERROR('Менеджер должен принадлежать тому же отделу', 16, 1);
        ROLLBACK TRANSACTION;
    END

    -- Проверяем, что сотрудник не является своим же менеджером
    IF EXISTS (
        SELECT 1
        FROM inserted
        WHERE EmployeeID = ManagerID
    )
    BEGIN
        RAISERROR('Сотрудник не может быть своим же менеджером', 16, 1);
        ROLLBACK TRANSACTION;
    END

END;
```

Пример 2: Триггер для ведения истории изменений

```
CREATE TRIGGER trg_Products_History
ON Products
AFTER UPDATE
AS
BEGIN
    SET NOCOUNT ON;

    -- Регистрируем только существенные изменения
    INSERT INTO ProductHistory (ProductID, FieldName, OldValue, NewValue,
ChangeDate, ChangedBy)
    SELECT
        i.ProductID,
        'Price' AS FieldName,
```

```
    CAST(d.Price AS NVARCHAR(50)) AS OldValue,
    CAST(i.Price AS NVARCHAR(50)) AS NewValue,
    GETDATE() AS ChangeDate,
    SYSTEM_USER AS ChangedBy
FROM inserted i
JOIN deleted d ON i.ProductID = d.ProductID
WHERE i.Price != d.Price

UNION ALL

SELECT
    i.ProductID,
    'StockQuantity' AS FieldName,
    CAST(d.StockQuantity AS NVARCHAR(50)) AS OldValue,
    CAST(i.StockQuantity AS NVARCHAR(50)) AS NewValue,
    GETDATE() AS ChangeDate,
    SYSTEM_USER AS ChangedBy
FROM inserted i
JOIN deleted d ON i.ProductID = d.ProductID
WHERE i.StockQuantity != d.StockQuantity;

END;
```

Пример 3: Триггер для каскадных обновлений

```
CREATE TRIGGER trg_Departments_Update
ON Departments
AFTER UPDATE
AS
BEGIN
    -- Если изменилось название отдела, обновляем у всех сотрудников
    IF UPDATE(DepartmentName)
    BEGIN
        UPDATE Employees
        SET DepartmentName = i.DepartmentName
        FROM Employees e
        INNER JOIN inserted i ON e.DepartmentID = i.DepartmentID
        INNER JOIN deleted d ON i.DepartmentID = d.DepartmentID
        WHERE i.DepartmentName != d.DepartmentName;
    END
```

```
    -- Если отдел деактивирован, деактивируем всех сотрудников
    IF UPDATE(IsActive)
    BEGIN
        UPDATE Employees
        SET IsActive = i.IsActive
        FROM Employees e
        INNER JOIN inserted i ON e.DepartmentID = i.DepartmentID
        INNER JOIN deleted d ON i.DepartmentID = d.DepartmentID
        WHERE i.IsActive != d.IsActive
          AND i.IsActive = 0; -- Только при деактивации
    END

END;
```

Пример 4: Триггер для сложной бизнес-логики

```
CREATE TRIGGER trg_Orders_InventoryUpdate
ON OrderDetails
AFTER INSERT, UPDATE, DELETE
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRY
        BEGIN TRANSACTION;

        -- Обработка новых заказов (INSERT)
        IF EXISTS (SELECT 1 FROM inserted) AND NOT EXISTS (SELECT 1
FROM deleted)
        BEGIN
            UPDATE p
            SET p.StockQuantity = p.StockQuantity - i.Quantity,
                p.ReservedQuantity = p.ReservedQuantity + i.Quantity
            FROM Products p
            INNER JOIN inserted i ON p.ProductID = i.ProductID
            INNER JOIN Orders o ON i.OrderID = o.OrderID
            WHERE o.Status = 'Pending';
        END

        -- Обработка изменений заказов (UPDATE)
```

```
    IF EXISTS (SELECT 1 FROM inserted) AND EXISTS (SELECT 1 FROM
deleted)
    BEGIN
        -- Возвращаем старые количества
        UPDATE p
        SET p.StockQuantity = p.StockQuantity + d.Quantity,
            p.ReservedQuantity = p.ReservedQuantity - d.Quantity
        FROM Products p
        INNER JOIN deleted d ON p.ProductID = d.ProductID
        INNER JOIN Orders o ON d.OrderID = o.OrderID
        WHERE o.Status = 'Pending';

        -- Резервируем новые количества
        UPDATE p
        SET p.StockQuantity = p.StockQuantity - i.Quantity,
            p.ReservedQuantity = p.ReservedQuantity + i.Quantity
        FROM Products p
        INNER JOIN inserted i ON p.ProductID = i.ProductID
        INNER JOIN Orders o ON i.OrderID = o.OrderID
        WHERE o.Status = 'Pending';
    END

    -- Обработка удаления заказов (DELETE)
    IF NOT EXISTS (SELECT 1 FROM inserted) AND EXISTS (SELECT 1
FROM deleted)
    BEGIN
        UPDATE p
        SET p.StockQuantity = p.StockQuantity + d.Quantity,
            p.ReservedQuantity = p.ReservedQuantity - d.Quantity
        FROM Products p
        INNER JOIN deleted d ON p.ProductID = d.ProductID
        INNER JOIN Orders o ON d.OrderID = o.OrderID
        WHERE o.Status = 'Pending';
    END

    -- Проверка достаточности остатков
    IF EXISTS (
        SELECT 1
        FROM Products p
        WHERE p.StockQuantity < 0
```

```
        )
        BEGIN
            RAISERROR('Недостаточно товара на складе', 16, 1);
            ROLLBACK TRANSACTION;
            RETURN;
        END

        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;

        DECLARE @ErrorMsg NVARCHAR(4000) = ERROR_MESSAGE();
        RAISERROR('Ошибка при обновлении инвентаря: %s', 16, 1,
@ErrorMsg);
    END CATCH

END;
```

Управление триггерами

Включение/отключение триггеров

```
-- Отключение триггера
DISABLE TRIGGER trg_Employees_Audit ON Employees;

-- Включение триггера
ENABLE TRIGGER trg_Employees_Audit ON Employees;

-- Отключение всех триггеров на таблице
DISABLE TRIGGER ALL ON Employees;

-- Включение всех триггеров на таблице

ENABLE TRIGGER ALL ON Employees;
```

Просмотр информации о триггерах

```
-- Список всех триггеров в базе

SELECT
```

```
    name AS TriggerName,
    OBJECT_NAME(parent_id) AS TableName,
    is_disabled AS IsDisabled,
    type_desc AS TriggerType
FROM sys.triggers
ORDER BY TableName, TriggerName;

-- Получение определения триггера
SELECT definition
FROM sys.sql_modules
WHERE object_id = OBJECT_ID('trg_Employees_Audit');

-- Зависимости триггера
SELECT
    OBJECT_NAME(referencing_id) AS TriggerName,
    referenced_entity_name AS ReferencedObject
FROM sys.sql_expression_dependencies
WHERE referencing_id = OBJECT_ID('trg_Employees_Audit');
```

Изменение и удаление триггеров

```
-- Изменение триггера
ALTER TRIGGER trg_Employees_Audit
ON Employees
AFTER INSERT, UPDATE, DELETE
AS
BEGIN
    -- Новая логика триггера
END;

-- Удаление триггера
DROP TRIGGER trg_Employees_Audit;
```

Контрольные вопросы

1. Что такое триггер и каково его назначение в базе данных?
2. Какие типы триггеров существуют и в чем их различия?
3. Как работают таблицы inserted и deleted?
4. Что такое прямая и косвенная рекурсия в триггерах?

5. Как предотвратить бесконечную рекурсию в триггерах?
6. Приведите пример практического использования триггера для реализации бизнес-правил.

Литература

1. Дейт К. Дж. Введение в системы баз данных. - Глава 9.
2. Коннолли Т., Бегг К. Базы данных: проектирование, реализация и сопровождение. - Глава 8.
3. Microsoft SQL Server Documentation: CREATE TRIGGER
4. PostgreSQL Documentation: CREATE TRIGGER