Язык Transact SQL

Объекты БД:

- Tables (таблицы)
- Views (представления)
- Stored procedures (хранимые процедуры)
- Triggers (триггеры)
- User Defined function (пользовательские функции)
- Indexes (индексы)
- User Defined Data Types (пользовательские типы данных)
- Constraints (ограничения)
- Keys (ключи)
- Users (пользователи), Roles (роли), Rules (правила), Defaults (умолчания)

Язык Transact SQL

Transact SQL содержит в своем составе:

- язык определения данных (DDL – Data Definition Language)
- язык манипулирования данными (DML – Data Manipulation Language)
- язык управления данными (Data Control Language), содержащий операторы для разграничения доступа пользователей к объектам БД
- алгоритмическую составляющую языка, включающую в себя процедуры, переменные, циклы, условные конструкции и т.п.

Схема базы данных

База данных WorkDataBase

Схема Teacher

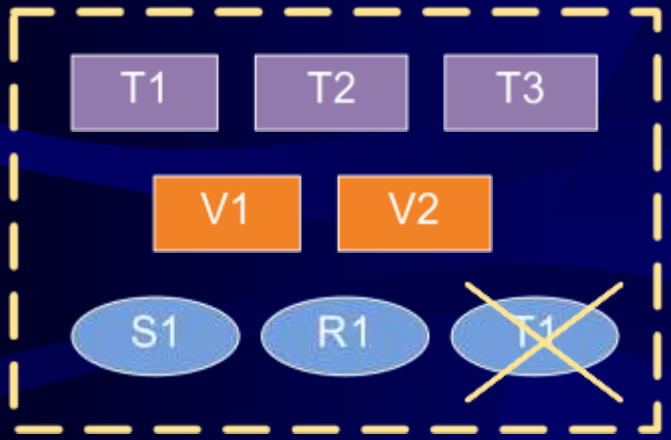


Схема Stud1

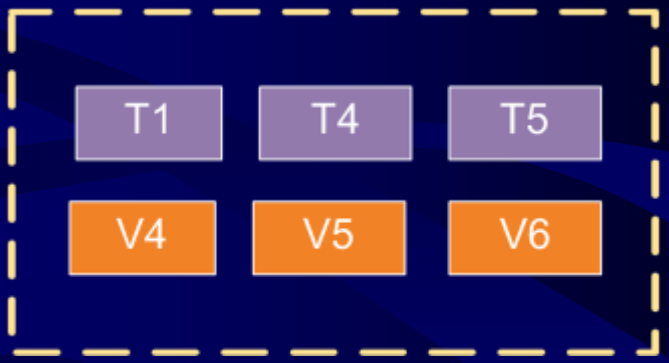


Схема Stud2

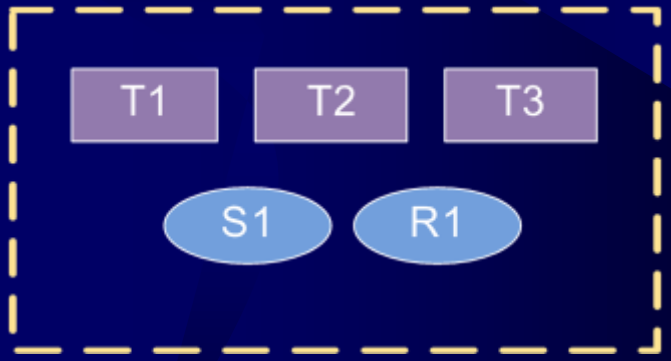


Схема базы данных

Преимущества применения схем:

- каждый объект базы (кроме объектов безопасности) относится к какой-либо схеме
- для каждого пользователя определяется своя схема по умолчанию
- нескольким пользователям можно назначить одну и ту же схему по умолчанию
- один пользователь может являться владельцем сразу нескольких схем
- при создании объекта можно явно указать схему, в которую его нужно поместить

База данных WorkDataBase

Имя_сервера. Имя_БД. Имя_схемы. Имя_объекта

Схема Teacher

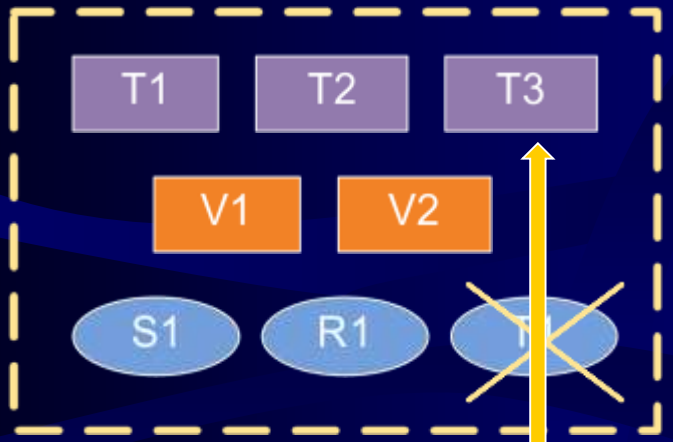
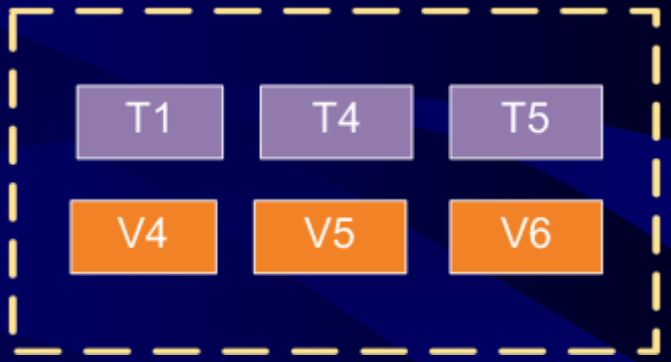
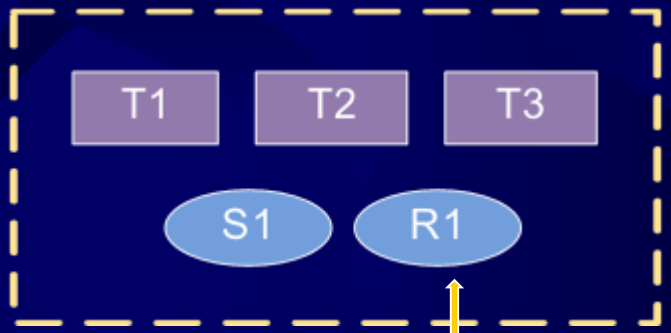


Схема Stud1



WorkDataBase.Teacher.T3

Схема Stud2



WorkDataBase.Stud2.R1

Локальные переменные

Локальные переменные используются для:

- проведения различных вычислений
- передачи данных в другие процедуры и функции и возвращения данных из процедур и функций
- хранения промежуточных данных и т.п.

```
DECLARE @a INT, @i AS REAL
DECLARE @s CHAR(15), @name CHAR(15)
SET @name = 'Ирина'
SELECT @name = 'Ирина'
```

Задание. Присвоить переменной @name значение имени пользователя с номером 1

```
-- Объявление переменных
DECLARE @id INT, @name VARCHAR(40)
-- Задание значения переменной @id
SET @id = 1
-- Задание значения переменной @name
SELECT @name = UserName FROM Users WHERE
UserID = @id
/* Вывод переменной @name в результат запроса (в
форме таблицы) */
SELECT @name AS [Имя пользователя]
-- или печать на экран
PRINT ‘Имя пользователя - ’+ @name
```

Язык Transact SQL

Арифметические операторы: +, -, *, /, %

Операторы сравнения: =, >, <, >=, <=, <>

Логические операторы, возвращающие TRUE/FALSE:
NOT, ALL, OR, AND, ANY, IN, BETWEEN, EXISTS, LIKE

BEGIN ... END

Конструкция ветвления (условные конструкции)

IF <лог. выражение>
    Оператор
[ELSE
    Оператор]

Конструкция ветвления

```
IF EXISTS (SELECT * FROM Sotr WHERE Fio LIKE
                                        ‘Агеев%’)
        PRINT  'Такой сотрудник имеется'
ELSE
BEGIN
        INSERT INTO Sotr (Sotr_id, Fio) VALUES (114,
‘Агеев’)
         PRINT ‘Сотрудник был добавлен'
END
```

Конструкция CASE

1. С входным выражением

```
CASE <входное выражение>
        WHEN <выражение 1> THEN <результат 1>
      [ WHEN <выражение 2> THEN <результат 2>
        ...]
        [ ELSE <результат> ]
END
```

```
SELECT Fio, CASE Gender
                WHEN 'м' THEN 'Мужчина'
                 WHEN 'ж' THEN 'Женщина'
                 ELSE 'Не указан'
        END
FROM Sotr
```

Конструкция CASE

2. Без входного выражения

```
CASE
        WHEN <лог. выражение 1> THEN <результат 1>
  [ WHEN < лог. выражение 2> THEN <результат 2>
        ...]
  [ ELSE <результат> ]
END
```

```
SELECT Fio, CASE
        WHEN Salary < 8000 THEN 'Низкая'
        WHEN Salary BETWEEN 8000 AND 15000 THEN
                                        'Средняя'
        WHEN Salary > 15000 THEN 'Высокая'
                END
FROM Sotr
```

Конструкция CASE

```
-- определение уровня зарплаты
DECLARE @salary INT, @text_salary VARCHAR (15)
SET @salary = 12000
CASE  SET @text_salary = CASE
        WHEN @salary < 8000 THEN
                        SET @text_salary = 'Низкая'
        WHEN @salary BETWEEN 8000 AND 15000 THEN
                        SET @text_salary = 'Средняя'
        WHEN @salary > 15000 THEN
                        SET @text_salary = 'Высокая'
END
```

Конструкция CASE

Увеличить размер зарплаты женщинам - на 20%, мужчинам - на 25%

```
UPDATE Sotr SET Salary =
CASE
        WHEN Gender = 'м' THEN Salary*1.25
        WHEN Gender = 'ж' THEN Salary*1.2
        ELSE Salary
END
```

Конструкция IIF

IIF (<логическое выражение>, <результат 1>,
<результат 2>)

Вывести список сотрудников с указанием их пола

```
SELECT Fio, IIF (Gender = 'м', 'Мужчина', 'Женщина')
        FROM Sotr
```

Изменить зарплату сотрудникам в зависимости от пола

```
UPDATE Sotr SET Salary =
    IIF (Gender = 'м', Salary*1.25, Salary*1.2)
```

Циклическая конструкция

```
WHILE <логическое выражение>
        Оператор
        [BREAK | CONTINUE]
```

```
DECLARE @n INT, @i INT, @x BIGINT
SET  @i = 1
SET  @x = 1
SET  @n = 10
WHILE @i < @n
      BEGIN
                SET  @x = @x * @i
                IF @x > 1000000 BREAK
                SET  @i = @i + 1
      END
SELECT @i AS i, @x AS x
```

Табличные переменные

CREATE TABLE #temp_t или ##temp_tt

DECLARE @имя_табличной_переменной TABLE
<определение таблицы>

Можно задать:

- описание столбцов таблицы (имя, тип данных, значение по умолчанию, ограничения на столбец, возможность NULL значений, первичный ключ, уникальность и т.п.)
- описание ограничений на уровне таблицы

Опция IDENTITY (<начальное значение>, <шаг>)

Табличные переменные

```
-- определить переменную типа таблица
DECLARE @my_table TABLE
(       id INT IDENTITY (0, 1) NOT NULL,
        Fam VARCHAR (20),
        Birthday DATE,
        PRIMARY KEY (id) )
-- добавление записи
INSERT INTO @my_table (Fam, Birthday) VALUES
                                ('Зайцев','01.04.1980')
-- загрузка данных из базы данных
INSERT INTO @my_table SELECT Fio, Birthday FROM Sotr
-- вывести содержимое переменной типа таблица
SELECT * FROM @my_table
```

Курсоры

Курсор – это особый временный объект SQL, который создается на основе таблиц или представлений БД, и имеет средства построчного передвижения по результирующему набору строк запроса

1. Создание курсора

DECLARE <имя_курсора> CURSOR
[<видимость>]
[<прокрутка>]
[<тип>]
[<блокировка>]
FOR <SELECT_запрос>
[FOR UPDATE [OF <имена_столбцов>]]

Создание курсора

Ограничения на команду SELECT:

- ✓ не может содержать INTO для создания новой таблицы
- ✓ не может содержать COMPUTE или COMPUTE BY, но может содержать функции агрегирования

Основные характеристики курсора:

- ➢ способность отражать изменения в исходных данных
- ➢ способность осуществлять прокрутку во множестве строк
- ➢ способность модифицировать множество строк

Создание курсора

DECLARE <имя_курсора> CURSOR
[<видимость>]
[<прокрутка>]
[<тип>]
[<блокировка>]
FOR <SELECT_запрос>
[FOR UPDATE [OF <имена_столбцов>]]

Тип курсора:

- ✓ STATIC (статический курсор)
- ✓ KEYSET (ключевой курсор)
- ✓ DYNAMIC (динамический курсор)
- ✓ FAST_FORWARD (курсор быстрого доступа)

Создание курсора

DECLARE <имя_курсора> CURSOR
[<видимость>]
[<прокрутка>]
[<тип>]
[<блокировка>]
FOR <SELECT_запрос>
[FOR UPDATE [OF <имена_столбцов>]]

READ_ONLY
SCROLL_LOCKS
OPTIMISTIC

DECLARE sotr_name_cursor CURSOR LOCAL
FORWARD_ONLY
STATIC
READ_ONLY

FOR SELECT fio FROM Sotr

Курсоры

2. Открытие курсора

OPEN [GLOBAL] <имя_курсора>

3. Чтение из курсора

FETCH <имя_курсора>

FETCH <имя_курсора> INTO <список_переменных>

```
-- открыть курсор
OPEN sotr_name_cursor

-- объявление локальных переменных
DECLARE @name VARCHAR(20)

-- чтение данных из курсора в локальные переменные
FETCH sotr_name_cursor INTO @name
```

Курсоры

FETCH <режим> FROM <имя_курсора> INTO
<список_переменных>

Режимы:

- ✓ NEXT
- ✓ PRIOR
- ✓ FIRST
- ✓ LAST
- ✓ ABSOLUTE n
- ✓ RELATIVE n

4. Закрытие курсора

CLOSE [GLOBAL] <имя_курсора>

5. Освобождение курсора

DEALLOCATE [GLOBAL] <имя_курсора>

Мониторинг курсоров

Значения глобальной переменной @@CURSOR_ROWS:

- -1 – число строк может меняться
- 0 – курсор не открыт или содержит 0 строк
- n – количество строк в курсоре равно n

Значения глобальной переменной @@FETCH_STATUS:

- ✓ 0 – команда FETCH выполнена успешно
- ✓ 1 – команда выполнена неудачно (выход за пределы курсора)
- ✓ 2 – команда выполнена неудачно (обращение к удаленной записи)

Пример

```
-- объявление курсора avg_salary_cur
DECLARE avg_salary_cur CURSOR     LOCAL
        FORWARD_ONLY   STATIC     READ_ONLY
FOR SELECT Salary FROM Sotr
-- открытие курсора
OPEN avg_salary_cur
-- объявление локальных переменных
DECLARE @kol INT, @x INT, @salary BIGINT
-- инициализация переменных
SET @kol = 0
SET @salary = 0
-- чтение данных из курсора в локальные переменные
FETCH NEXT FROM  avg_salary_cur INTO @x
```

```
-- цикл по всем записям курсора
WHILE @@FETCH_STATUS = 0
BEGIN
        SET @salary = @salary + @x
        SET @kol = @kol + 1
        -- переход к следующей записи и чтение данных
        FETCH NEXT FROM avg_salary_cur INTO @x
END
-- закрытие курсора
CLOSE  avg_salary_cur
-- освобождение курсора
DEALLOCATE  avg_salary_cur
-- вычисление среднего значения
SET @salary = @salary / @kol
-- вывод результатов
PRINT 'Средняя зарплата составляет - '+
                        CAST (@salary AS VARCHAR(8))
```

Курсоры

CAST (<выражение> AS <тип_данных>)

Модификация и удаление строк через курсоры

А. Позиционное обновление

```
UPDATE <имя_таблицы_или_представления>
    SET <список_для_модификации>
    WHERE CURRENT OF <имя_курсора>
```

Б. Позиционное удаление

```
DELETE <имя_таблицы_или_представления>
    WHERE CURRENT OF <имя_курсора>
```

Пример

```
DECLARE add_salary_cur CURSOR LOCAL
        FORWARD_ONLY DYNAMIC
        FOR SELECT Salary FROM Sotr FOR UPDATE
OPEN add_salary_cur
FETCH next FROM  add_salary_cur
WHILE @@FETCH_STATUS = 0
BEGIN
        UPDATE Sotr SET Salary = Salary + 3000
                WHERE CURRENT OF add_salary_cur
        FETCH next FROM add_salary_cur
END
CLOSE  add_salary_cur
DEALLOCATE add_salary_cur
SELECT * FROM Sotr
```

Хранимые процедуры

Взаимодействие клиентского приложения с БД, расположенной на сервере

Хранимые процедуры

Преимущества выполнения в БД хранимых процедур по сравнению с отдельными командами Transact SQL :

- хранимые процедуры поддерживают модульное программирование
- хранимые процедуры могут вызывать другие ХП и функции
- хранимые процедуры могут быть вызваны из прикладных программ других типов
- все они прошли этап синтаксического анализа и находятся в исполняемом формате
- хранимые процедуры выполняются быстрее
- хранимые процедуры проще использовать

Хранимые процедуры

Типы хранимых процедур:

- системные хранимые процедуры
sp_addlogin или sp_statistics
- пользовательские хранимые процедуры
- временные хранимые процедуры
локальные (# имя) и глобальные (## имя)

Создание хранимых процедур

```
CREATE PROC[EDURE] <имя_процедуры>
        [ {@имя_параметра <тип_данных>
          [ = значение_по_умолчанию] [OUTPUT] }
        ]
AS <тело процедуры>
```

Хранимые процедуры

```
CREATE PROCEDURE Proc1
        @name VARCHAR(30)
AS
        SELECT Proj_n, Task FROM Sotr JOIN Project
                        ON Sotr.Sotr_id = Project.Sotr_id
        WHERE Fio = @name
```

Вызов хранимых процедур

```
EXEC[UTE] <имя_процедуры>
        [@имя_параметра=] {значение | @имя_переменной}
        [OUTPUT] | [DEFAULT]
EXECUTE Proc1 'Алексеев А.Г.'
EXEC Proc1 @name = 'Алексеев А.Г.'
```

Хранимые процедуры

```
CREATE PROCEDURE Proc2
        @proj INT, @count_sotr SMALLINT OUTPUT
AS
        SELECT @count_sotr = COUNT(*) FROM Project
                        WHERE Proj_n = @proj

DECLARE @Result SMALLINT
EXECUTE Proc2 1234, @Result OUTPUT
```

Хранимые процедуры

```
CREATE PROCEDURE NewSotr
    @fio VARCHAR(20), @birth DATE, @gender CHAR(1) = ‘м’
AS
IF NOT EXISTS (SELECT * FROM Sotr WHERE
        fio = @fio AND birthday = @birth)
BEGIN
        DECLARE @id INT
        -- определение последнего табельного номера
        SELECT @id = MAX(sotr_id) FROM Sotr
        INSERT INTO Sotr (sotr_id, fio, birthday, gender)
                        VALUES (@id+1, @fio, @birth, @gender)
END

EXECUTE NewSotr 'Антонов Б.С.', '12/12/1960'
```

```
CREATE PROCEDURE NewSotr
        @fio VARCHAR(20), @birth DATE,
        @gender CHAR(1) = ‘м’, @id INT OUTPUT
AS BEGIN
        SELECT @id = sotr_id  FROM Sotr WHERE
                        fio = @fio AND birthday = @birth
        -- если сотрудника нет в таблице
        IF @id IS NULL
        BEGIN
                SELECT @id = MAX(sotr_id) FROM Sotr
                SET @id = @id + 1
            INSERT INTO Sotr (sotr_id, fio, birthday, gender)
                                VALUES (@id, @fio, @birth,
@gender)
        END
    END
DECLARE @R INT
EXEC NewSotr 'Алов А.А.', '1/02/1960', @id = @R OUTPUT
```

Пользовательские функции

Виды пользовательских функций:

- скалярные функции (scalar)
- функции с одним запросом (inline-функции)
- многооператорные табличные функции (multi-statement)

Отличия функций от хранимых процедур:

1) В теле функции запрещены изменения глобальных объектов, например таблиц, курсоров и т.д.

2) Если при выполнении команды произойдет ошибка в теле процедуры, то текущая команда прерывается, и выполнится следующая команда данной процедуры.
В функции такая ошибка приведет к прерыванию выполнения всей функции.

Скалярные функции

Создание скалярной функции

```
CREATE FUNCTION <имя_функции>
        ( [ {@имя_параметра <тип_данных>
          [ = значение_по_умолчанию] } ] )
RETURNS скалярный_тип_данных
AS
BEGIN
   Операторы
   RETURN скалярное_выражение
END
```

```
CREATE FUNCTION MaxSotr ()
RETURNS INT
AS
BEGIN
        -- сохраним во временной таблице число
                сотрудников каждого отдела
        DECLARE @temp TABLE (kol INT)
        INSERT INTO @temp SELECT COUNT(*) FROM
Sotr                                        GROUP BY Dep
        -- найдем максимальное число среди всех
отделов
         DECLARE @m INT
        SELECT @m = MAX(kol) FROM @temp
        RETURN @m
END
DECLARE @Result INT
SELECT @Result = MaxSotr ()
PRINT @Result
```

Скалярные функции

```
CREATE FUNCTION CountSotrInProj (@proj INT)
RETURNS INT
AS
BEGIN
        DECLARE @c INT
        SELECT @c = COUNT(*) FROM Project WHERE
                                        Proj_n = @proj
        RETURN @c
END

PRINT CountSotrInProj (1234)
```

Функции типа *inline*

```
CREATE FUNCTION <имя_функции>
        ( [ {@имя_параметра <тип_данных>
          [ = значение_по_умолчанию] } ] )
RETURNS TABLE
AS
RETURN <SELECT-запрос>
```

```
CREATE FUNCTION SotrInDep (@dep INT)
RETURNS TABLE
AS
        RETURN
        SELECT fio FROM Sotr WHERE Dep = @dep

SELECT * FROM SotrInDep(4) ORDER BY fio
```

Многооператорные табличные функции

```
CREATE FUNCTION <имя_функции>
        ( [ {@имя_параметра <тип_данных>
          [ = значение_по_умолчанию] } ] )
RETURNS @имя_таблицы TABLE <определение_

        таблицы>
AS
BEGIN
   Операторы
   RETURN
END
```

Многооператорные табличные функции

```
CREATE FUNCTION SotrInDep_New (@dep INT)
        RETURNS @res TABLE (FioSotr VARCHAR(25))
AS
BEGIN
        INSERT INTO @res SELECT fio FROM Sotr WHERE
                                        Dep = @dep
        RETURN
END
```

Транзакции

Транзакция – это неделимая последовательность операций обработки данных, которая выполняется как единое целое и переводит БД из одного целостного состояния в другое

Свойства транзакций:

1. Атомарность
2. Согласованность
3. Изолированность
4. Устойчивость

При разработке транзакций следует:

А. Определить границы транзакции

Б. Разработать механизм управления ошибками

В. Определить уровень изолированности транзакции

Транзакции

3 вида определения транзакций:

- ✓автоматическое
- ✓явное
- ✓подразумеваемое

Транзакции с автофиксацией

```
CREATE TABLE table1 (
        id INT PRIMARY KEY,
        col VARCHAR(20) NOT NULL)

INSERT INTO table1 VALUES (1, 'Первая строка')
INSERT INTO table1 VALUES (2, NULL)
INSERT INTO table1 VALUES (3,'Третья строка')
```

Транзакции

SELECT * FROM table1

Id	col
1	Первая строка
3	Третья строка

Явные транзакции

1) BEGIN TRAN[SACTION] [<имя_транзакции>]
2) COMMIT TRAN[SACTION] [<имя_транзакции>]
3) SAVE TRAN[SACTION] <имя_точки_сохранения>
4) ROLLBACK TRAN[SACTION] [<имя_транзакции> | <имя_точки_сохранения>]

Транзакции

```
BEGIN TRAN
        INSERT INTO table1 VALUES (1, 'Первая строка')
        INSERT INTO table1 VALUES (2, NULL)
        INSERT INTO table1 VALUES (3,'Третья строка')
COMMIT TRAN

SELECT * FROM table1
```

Id	col
1	Первая строка
3	Третья строка

Обработка ошибок

```
BEGIN TRY
    { <оператор SQL> }
END TRY
```

```
BEGIN CATCH
    { <оператор SQL> }
END CATCH
```

Уровни ошибок:

- 1–10 Информационные сообщения
- 11–19 Относительно серьезные ошибки
- 20–25 Очень серьезные ошибки

Транзакции

```
-- блок команд с возможными ошибками
BEGIN TRY
    BEGIN TRAN
        INSERT INTO table1 VALUES (1, 'Первая строка')
        INSERT INTO table1 VALUES (2, NULL)
        INSERT INTO table1 VALUES (3,'Третья строка')
    COMMIT TRAN
END TRY
-- обработка ошибок
BEGIN CATCH
    RAISERROR("Ошибка в транзакции!",14,1)
    ROLLBACK TRAN
END CATCH
```

Триггеры

Триггер – это предварительно определенное действие или последовательность действий, *автоматически* осуществляемых при выполнении операций модификации данных

Цели использования триггеров:

- ❖ проверка корректности введенных данных и выполнение сложных ограничений целостности данных
- ❖ выдача предупреждений о необходимости выполнения некоторых действий с таблицей

Компоненты триггера:

1. Ограничения
2. Событие
3. Предусмотренное действие

Создание триггеров

```
CREATE TRIGGER <имя_триггера>
        ON <имя таблицы_или_представления>
        {AFTER | INSTEAD OF}
        {[INSERT] [,] [UPDATE] [,] [DELETE]}
AS <операторы_SQL>
```

Программирование триггеров

Специальные таблицы: inserted и deleted

Функция @@ROWCOUNT возвращает количество строк, обработанных последней командой

Программирование триггеров

	Содержимое таблиц	
Команды	*inserted*	*deleted*
INSERT	строки, которые пользователь вставляет в таблицу	пусто
DELETE	пусто	строки, которые пользователь пытается удалить
UPDATE	новые значения строк	старые значения строк

ROLLBACK TRANSACTION

DROP TRIGGER <имя_триггера>

Программирование триггеров

```
CREATE TRIGGER Trig_ins_after ON Sotr AFTER INSERT
   AS
BEGIN
   -- определить отдел, в который назначен сотрудник
   DECLARE @dep INT
   SELECT @dep = Dep FROM inserted
   -- если это третий отдел, то отменить добавление
   IF @dep = 3
      BEGIN
         PRINT 'Запрещено добавление в третий отдел'
         ROLLBACK TRAN
      END
END
```

Программирование триггеров

```
CREATE TRIGGER Trig_ins_instead ON Sotr
        INSTEAD OF INSERT
    AS
BEGIN
    DECLARE @dep INT
    SELECT @dep = Dep FROM inserted
    -- если это не третий отдел, то реализовать добавление
    IF @dep <> 3
        INSERT INTO Sotr SELECT * FROM inserted
    ELSE
        PRINT 'Запрещено добавление в третий отдел'
END
```