

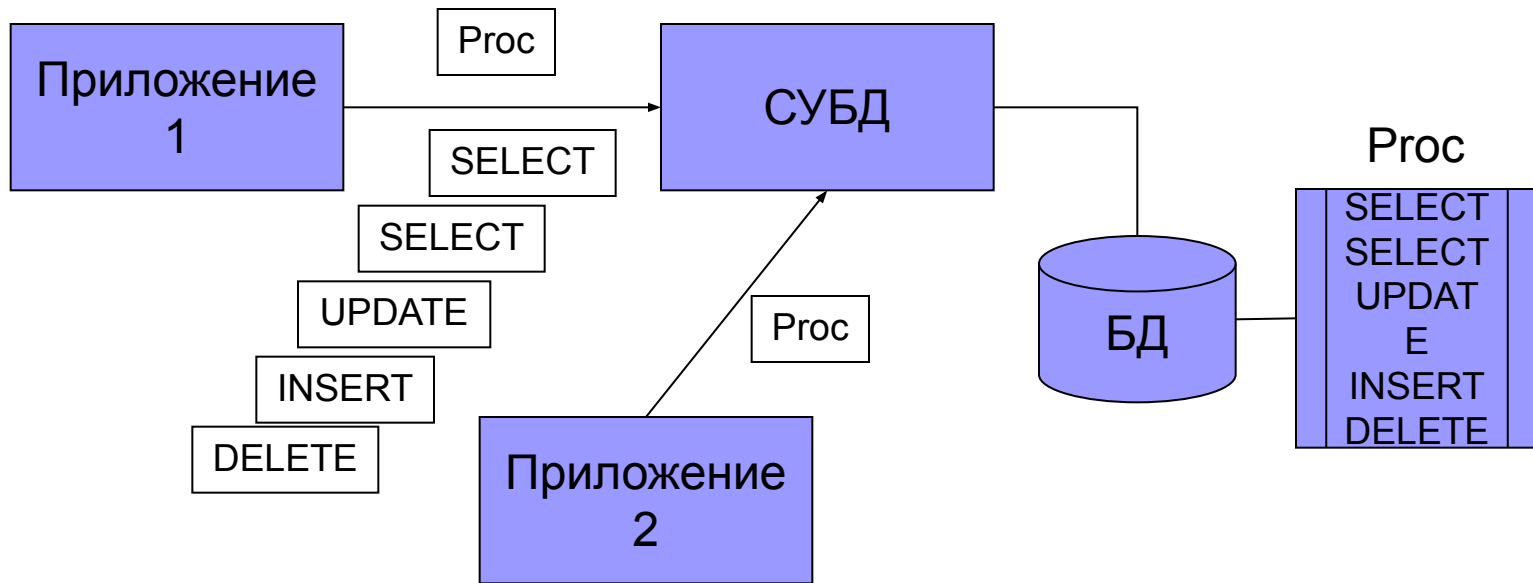


Базы данных

Хранимые процедуры и
триггеры

Хранимые процедуры

Хранимая процедура (Stored Procedure) – это именованный набор команд языка Transact-SQL, хранящийся на сервере в качестве самостоятельного объекта БД



- SP хранится на сервере (пакет – на клиенте)
- Для вызова используется имя (у пакета нет имени)
- SP могут вызвать друг друга (пакеты не могут)
- Изменения вносятся в одном месте, на сервере (в пакеты – на всех клиентах)
- Реализуется модульность
- Уменьшается объем передаваемой информации по сети

Типы хранимых процедур

- **Системные ХП** – входят в состав SQL Server, реализуют все действия администрирования сервера, начинаются с sp_, хранятся в БД master. Контекст выполнения любой
- **Пользовательские ХП (User-Defined SP)** – размещаются в пользовательских БД и выполняются в контексте только одной БД
- **Временные ХП** – существуют только в рамках одного соединения с сервером (временно хранятся в tempDB)

Создание хранимых процедур

```
CREATE PROC[EDURE] <имя процедуры>  
  [{@<параметр> <тип>} [= <значение по умолчанию>]  
  [OUTPUT] ] [,...n]  
AS <SQL оператор> [,...n]
```

- <имя процедуры> - начинается с «sp_» для системных ХП, «#» - для локальных временных ХП, «##» - для глобальных временных ХП.
- Параметры и локальные переменные начинаются с @. Параметры разделяются запятыми, количество параметров до 1024.
- OUTPUT помечает выходной параметр (он же является и входным)
- <значение по умолчанию> присваивается входному параметру, если при вызове процедуры ему не присваивается значение
- Возвращаемое значение процедуры (как для функции) задается в теле процедуры оператором RETURN

Создание хранимых процедур

```
CREATE PROC DeleteReader  
    @ReaderID INT
```

```
AS
```

```
DELETE FROM Readers  
    WHERE reader_id = @ReaderID
```

```
-----  
CREATE PROC GetReaderName
```

```
    @ReaderID INT,  
    @ReaderName VARCHAR(50) OUTPUT
```

```
AS
```

```
SELECT @ReaderName = first_name+' ' + last_name  
    FROM Readers  
    WHERE reader_id = @ReaderID
```

```
-----  
CREATE PROC ReaderExists
```

```
    @Surname VARCHAR(50)
```

```
AS
```

```
DECLARE @Count INT  
SET @Count = (SELECT COUNT(*)  
    FROM Readers  
    WHERE last_name = @Surname)
```

```
IF @Count = 0  
    RETURN 0
```

```
ELSE  
    RETURN 1
```

```
-- Без параметров
```

```
CREATE PROC GetReaders
```

```
AS
```

```
SELECT * FROM Readers
```

Передача параметров

Вызов хранимой процедуры:

EXEC[UTE]

[@<переменная> =] <имя процедуры>

[[@<параметр> =] {<значение> |

@<переменная> [OUT[PUT]] |

[DEFAULT] }] [,...n]

Передача параметров

Способы передачи параметров:

- *Позиционные параметры*

```
DECLARE @R CHAR(50)
EXEC GetReaderName 5, @R OUTPUT
PRINT @R
```

- *Ключевые параметры*

```
DECLARE @R CHAR(50)
EXEC GetReaderName
    @ReaderID = 5, @ReaderName = @R OUTPUT
-- или в другом порядке
EXEC GetReaderName
    @ReaderName = @R OUTPUT, @ReaderID = 5
```

Возвращение данных из хранимой процедуры

- **Стандартный набор строк или набор данных** (RecordSet, DataSet – несколько RecordSet). Если в теле процедуры выполняется SELECT, то возвращается набор строк (таблица), если несколько операторов SELECT, то набор данных (набор таблиц).
EXEC GetReaders
- **OUTPUT-параметры** (пример выше)
- **Код завершения** возвращается всегда. По умолчанию он равен 0. Изменяется оператором RETURN:
DECLARE @R INT
EXEC @R = ReaderExists 'Петров'
PRINT @R

Управление хранимыми процедурами

- Изменение (кроме имени). Процедура заменяется полностью:
ALTER PROCEDURE <имя процедуры>
далее как в CREATE PROCEDURE
- Изменение имени – при помощи системной хранимой процедуры
sp_rename '<старое имя>', '<новое имя>' [, 'object']
(она же используется для переименования таблиц и столбцов в таблице) например,
exec sp_rename 'GetReader', 'GetAllReaders', 'object'
- Удаление хранимой процедуры
DROP PROCEDURE <имя процедуры>

Управляющие конструкции Transact-SQL

- BEGIN...END – блок
- BREAK – выход из цикла
- CONTINUE – переход к началу цикла
- GOTO – безусловный переход
- IF...ELSE – ветвление
- RETURN – возврат из хранимой процедуры
- WHILE – цикл с предусловием
- TRY...CATCH – обработка исключения

Управляющие конструкции Transact-SQL

```
DECLARE @MyCounter INT;  
SET @MyCounter = 1;
```

```
-- Все строки будут заполнены одинаково:  
SELECT FirstColumnHeading = 'xyz',  
       SecondColumnHeading = ProductID  
FROM Production.Product;
```

```
DECLARE @compareprice money, @cost money  
EXECUTE usp_GetList '%Bikes%', 700,  
       @compareprice OUT,  
       @cost OUTPUT  
IF @cost <= @compareprice  
BEGIN  
    PRINT 'These products can be purchased for less than  
    $'+RTRIM(CAST(@compareprice AS varchar(20)))+'.'  
END  
ELSE  
    PRINT 'The prices for all products in this category exceed $'+  
    RTRIM(CAST(@compareprice AS varchar(20)))+'.'
```

Управляющие конструкции Transact-SQL

```
WHILE (SELECT AVG(ListPrice) FROM Production.Product) < $300
BEGIN
    UPDATE Production.Product
        SET ListPrice = ListPrice * 2
    SELECT MAX(ListPrice) FROM Production.Product
    IF (SELECT MAX(ListPrice) FROM Production.Product) > $500
        BREAK
    ELSE
        CONTINUE
END
PRINT 'Too much for the market to bear';
```

Триггеры

Триггер – процедура, связанная с таблицей или представлением, которая **автоматически** выполняется при выполнении операции вставки, изменения или удаления строки этой таблицы или представления

Назначение триггеров:

- Обеспечение целостности данных (например, не стандартное каскадное удаление)
- Сокращение затрат на программирование приложений (общие действия – в триггер)
- Автоматическое ведение журнала изменений базы данных
- Автоматическое предупреждение об изменении данных в БД

Классификация по типу действия:

- INSERT TRIGGER – запускаются при выполнении команды INSERT
- UPDATE TRIGGER – запускаются при выполнении команды UPDATE
- DELETE TRIGGER – запускаются при выполнении команды DELETE

Классификация по типу поведения:

- AFTER – триггер выполняется **после успешного выполнения** команды
- INSTEAD OF – триггер вызывается **вместо выполнения** команды. Для представлений можно использовать только триггер INSTEAD OF.

Триггеры

Создание триггера:

```
CREATE TRIGGER <имя триггера>  
ON <имя таблицы> | <имя представления>  
{FOR | AFTER | INSTEAD OF}  
{ [DELETE] [,] [INSERT] [,] [UPDATE] }  
AS  
<SQL оператор> [...n]
```

FOR и AFTER – синонимы.

Триггеры

AFTER триггер выполняется в транзакции

Сравнение AFTER и INSTEAD OF триггеров: AFTER триггер выполняется после того, как действие команды было завершено. Поэтому, если необходимо отменить действие команды, то в AFTER триггере надо использовать конструкцию ROLLBACK TRANSACTION. В этой же ситуации в INSTEAD OF триггере не надо отменять действие, т.к. оно не выполняется (т.е. не надо использовать ROLLBACK TRANSACTION). Но для фиксации операции сам триггер должен выполнить соответствующую операцию (INSERT, DELETE, UPDATE).

Триггеры

```
CREATE TRIGGER ExemplarsUpdateTrigger
  ON Exemplars
  AFTER UPDATE
AS
BEGIN
```

```
IF (EXISTS (select e.reader_id
  FROM Exemplars e WHERE e.reader_id IS NOT NULL
  GROUP BY e.reader_id
  HAVING count(*) > 2))
```

```
BEGIN
  ROLLBACK TRANSACTION
  PRINT 'Попытка взять более 2 книг'
END
```

```
END
```

-- Взять все свободные экземпляры

```
UPDATE Exemplars SET reader_id = 1 WHERE reader_id IS NULL
```


Триггеры

Результат выполнения триггера

	inv	isbn	reader_id	date_out
	1	123	1	NULL
	2	123	1	NULL
	3	123	NULL	NULL
..	4	123	1	NULL
	5	123	NULL	NULL
	6			
*	NULL			



Триггеры

Таблицы `inserted` и `deleted`

- Команда `INSERT`: `inserted` содержит все вставляемые строки, `deleted` – пустая
- Команда `DELETE`: `inserted` – пустая, `deleted` содержит удаленные строки
- Команда `UPDATE`: `inserted` содержит новые значения строк, `deleted` – старые (заменяемые) значения

```
CREATE TRIGGER ExemplarsDeleteTrigger
ON Exemplars
AFTER DELETE
AS
BEGIN
    IF EXISTS(SELECT * FROM deleted WHERE reader_id IS NOT NULL)
    BEGIN
        ROLLBACK TRANSACTION
        PRINT 'Запрещено удалять не возвращенные книги'
    END
END
```

`DELETE FROM Exemplars WHERE inv = 2`

Триггеры

UPDATE(<имя столбца>) – логическая функция для проверки изменения значения столбца (только для команд INSERT и UPDATE)

COLUMNS_UPDATED () - функция возвращает результат типа VARBINARY, каждому разряду соответствует номер столбца (0 – не изменен, 1 – изменен)

```
CREATE TRIGGER reminder ON Person.Address AFTER UPDATE
AS
  IF ( UPDATE (StateProvinceID) OR UPDATE (PostalCode) )
BEGIN
  RAISERROR (50009, 16, 10)
END;
```

Триггеры

```
CREATE TRIGGER updEmployeeData ON employeeData AFTER UPDATE
AS
```

```
/*Check whether columns 2, 3 or 4 have been updated. If any or all columns 2, 3 or 4 have
been changed, create an audit record. The bitmask is:
```

```
power(2,(2-1))+power(2,(3-1))+power(2,(4-1)) = 14. To test whether all columns 2, 3, and 4
are updated, use = 14 instead of >0 (below).*/
```

```
IF (COLUMNS_UPDATED() & 14) > 0 /*Use IF (COLUMNS_UPDATED() & 14) = 14 to see
whether all columns 2, 3, and 4 are updated.*/
```

```
BEGIN
```

```
-- Audit OLD record.
```

```
INSERT INTO auditEmployeeData (audit_log_type, audit_emp_id,
audit_emp_bankAccountNumber, audit_emp_salary, audit_emp_SSN)
```

```
SELECT 'OLD', del.emp_id, del.emp_bankAccountNumber, del.emp_salary, del.emp_SSN
FROM deleted del
```

```
-- Audit NEW record.
```

```
INSERT INTO auditEmployeeData (audit_log_type, audit_emp_id,
audit_emp_bankAccountNumber, audit_emp_salary, audit_emp_SSN)
```

```
SELECT 'NEW', ins.emp_id, ins.emp_bankAccountNumber, ins.emp_salary, ins.emp_SSN
FROM inserted ins
```

```
END;
```



Триггеры

Предупреждение: объекты, на которые ссылаются триггеры, могут быть удалены или переименованы без ограничения. В результате таких изменений триггер будет работать некорректно