Зарезервированное слово CONSTRAINT языка [SQL](http://www.firebirdsql.su/doku.php?id=raznovidnosti_jazyka_sql) сервера Firebird служит для работы с ограничениями ссылочной целостности создаваемых в базе данных таблиц. Ограничения ссылочной целостности бывают двух уровней: ограничения, накладываемые на отдельный столбец, и ограничения, накладываемые на всю таблицу. Одноименный оператор CONSTRAINT служит для работы с ограничениями ссылочной целостности, накладываемых на таблицу.

Оператор ограничения ссылочной целостности CONSTRAINT не употребляется самостоятельно, а выполняется в рамках операторов [CREATE TABLE](http://www.firebirdsql.su/doku.php?id=create_table) и [ALTER TABLE](http://www.firebirdsql.su/doku.php?id=alter_table),

1. **Основные понятия теории БД: БД, СУБД, основные требования к информации в БД, мера информации в БД, модель данных БД, логическая схема БД, основная терминология реляционной базы данных (таблица, домен, кортеж, атрибут, заголовок таблицы), операции реляционной алгебры Кодда.**

**База данных:** хранилище систематизированных данных.

**компьютерная база данных:** хранилище систематизированных и динамически обновляемых данных; данные хранятся на электронных носителях; доступ к данным обеспечивается СУБД;

**СУБД** - специальные программные средства для доступа к данным.

**Требования к информации в базе данных**: 1)полезность;2) полнота информации; 3) точность; 4)достоверность; 5)непротиворечивость; 6) актуальность.

**Мера информации в базе данных:** 1) синтаксическая (в символах, в B, KB, MB, GB, TB); 2)семантическая мера информации (количество информации на символ); 3) прагматическая мера информации (полезность для управления).

**модели данных:** иерархическая, сетевая, реляционная;

как правило, создается для многих пользователей. Каждый пользователь имеет свое представление о базе данных. Совокупность всех представлений – это **логическая схема данных**.

- **домен**: множество;

- **таблица**: отношение;

- **атрибут**: имя столбца таблицы (имя домена);

- **заголовок таблицы**: множества всех атрибутов;

- **кортеж**: элемент отношения или строка таблицы;

- **строка таблицы**: кортеж.

**Операции реляционной алгебры (алгебра Кодда)**: **UNION** (объединение), **INTERSECT** (пересечение), **MINUS** (разность), **TIMES** (декартово произведение), **WHERE** (ограничение), **PROJECT** (проекция), **JOIN** (соединение), **DIVIDE BY** (реляционное деление).

1. **Язык SQL. Декларативность языка SQL. Основные стандарты языка SQL. Процедурные расширения языка SQL. Формы языка SQL. Понятие диалекта SQL. Группы операторов SQL: DDL, DML, DCL, TCL.**

SQL — универсальный компьютерный язык, применяемый для создания, модификации и управления данными в реляционных базах данных. Вопреки существующим заблуждениям, SQL является информационно-логическим языком, а не языком программирования.

**SQL (Structured Query Language)**: 1970гг впервые разработан IBM для System R, назывался **SEQUEL**; первый стандарт ANSI/SQL 1986г; первая коммерческая СУБД, поддерживающая SQL, была Oracle V2 на машинах VAX.

**SQL 92**: SQL 2, ISO 9075

**SQL 99**: SQL 3, объектно-ориентированные возможности.

**SQL 2003, SQL 2006, SQL 2009:** XML.

**Декларативность**: с помощью SQL программист описывает только какие данные нужно извлечь или модифицировать. То, каким образом это сделать решает СУБД непосредственно при обработке SQL запроса.

**SQL**: DDL, DML, DCL, TCL.

Операторы SQL делятся на:

* операторы определения данных (*Data Definition Language,* [*DDL*](http://ru.wikipedia.org/wiki/DDL))
  + [**CREATE**](http://ru.wikipedia.org/wiki/CREATE) создает объект БД (саму базу, таблицу, [представление](http://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%B5%D0%B4%D1%81%D1%82%D0%B0%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5_(%D0%B1%D0%B0%D0%B7%D1%8B_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85)), пользователя и т. д.)
  + [**ALTER**](http://ru.wikipedia.org/w/index.php?title=ALTER&action=edit&redlink=1) изменяет объект
  + [**DROP**](http://ru.wikipedia.org/w/index.php?title=DROP&action=edit&redlink=1) удаляет объект
* операторы манипуляции данными (*Data Manipulation Language,* [*DML*](http://ru.wikipedia.org/wiki/DML))
  + [**SELECT**](http://ru.wikipedia.org/wiki/SELECT) считывает данные, удовлетворяющие заданным условиям
  + [**INSERT**](http://ru.wikipedia.org/wiki/Insert_(SQL)) добавляет новые данные
  + [**UPDATE**](http://ru.wikipedia.org/wiki/UPDATE) изменяет существующие данные
  + [**DELETE**](http://ru.wikipedia.org/wiki/DELETE) удаляет данные
* операторы определения доступа к данным (*Data Control Language,* [*DCL*](http://ru.wikipedia.org/wiki/DCL))
  + [**GRANT**](http://ru.wikipedia.org/w/index.php?title=GRANT&action=edit&redlink=1) предоставляет пользователю (группе) разрешения на определенные операции с объектом
  + [**REVOKE**](http://ru.wikipedia.org/w/index.php?title=REVOKE&action=edit&redlink=1) отзывает ранее выданные разрешения
  + [**DENY**](http://ru.wikipedia.org/w/index.php?title=DENY&action=edit&redlink=1) задает запрет, имеющий приоритет над разрешением
* операторы управления [транзакциями](http://ru.wikipedia.org/wiki/%D0%A2%D1%80%D0%B0%D0%BD%D0%B7%D0%B0%D0%BA%D1%86%D0%B8%D1%8F) (*Transaction Control Language,* [*TCL*](http://ru.wikipedia.org/wiki/TCL_(SQL)))
  + [**COMMIT**](http://ru.wikipedia.org/wiki/COMMIT) применяет транзакцию.
  + [**ROLLBACK**](http://ru.wikipedia.org/wiki/ROLLBACK) откатывает все изменения, сделанные в контексте текущей транзакции.
  + [**SAVEPOINT**](http://ru.wikipedia.org/wiki/SAVEPOINT) делит транзакцию на более мелкие участки.

**Транза́кция** ([англ.](http://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *transaction*) — в [информатике](http://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0), группа последовательных операций, которая представляет собой логическую единицу работы с данными.

**Процедурные расширения SQL:** T-SQL, PL/SQL.

**Формы SQL**: 1)интерактивный; 2)встроенный SQL (embedded); 3) динамический SQL.

**Диалекты SQL**: расширения стандарта SQL для конкретных СУБД.

1. **Нормализация таблиц базы данных. Нормальные формы таблиц. Процедуры построения 1ой, 2ой и 3ей нормальных форм. Назначение и принцип построения 4ой, 5ой и 6ой нормальных форм.**

*Нормализация отношений* – это формальный аппарат ограничений на формирование отношений, который позволяет устранить дублирование и потенциальную противоречивость хранимых данных, уменьшает трудозатраты на ведение БД. Процесс нормализации заключается в декомпозиции исходных отношений на более простые отношения. Цель нормализации – получение такого проекта БД, в котором «каждый факт появляется лишь в одном месте».

**Нормализация данных**: процесс преобразования таблиц базы данных к нормальной форме.

**Шесть нормальных форм**: 1NF, 2NF,...6NF. Широкое практическое применение имеют формы 1NF, 2NF, 3NF. Редко применяется форма 4NF. 5NF имеет теоретическое значение. 6NF – для баз данных специального назначения.

**1NF**: Любое отношение находится в первой нормальной форме**.**  Каждый кортеж в отношении всегда в единственном экземпляре.

**2NF:** 1NF+любой атрибут не входящий в состав потенциального ключа функционально полно зависит от каждого возможного потенциального ключа.

Ключи таблицы, обладающие свойством минимальности, называются **потенциальными ключами этой таблицы**.

**3NF:** 2NF+ отсутствуют транзитивные функциональные зависимости.



BCNF – Нормальная форма Бойса-Кодда.

Признак отсутствия BCNF при 3NF: наличие 2х возможных составных ключей, имеющих общий атрибут

4NF – четвертая нормальная форма

4NF: BCNF+в отношении отсутствуют нетривиальные многозначные зависимости не являющиеся функциональными зависимостями.

5NF: 4NF+любая зависимость соединения является тривиальной

DKNF – Доменно-ключевая нормальная форма

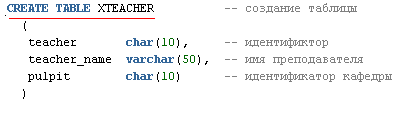
6NF – шестая нормальная форма

6NF - обобщение 5NF для темпоральных баз данных (данные связанные со временем)

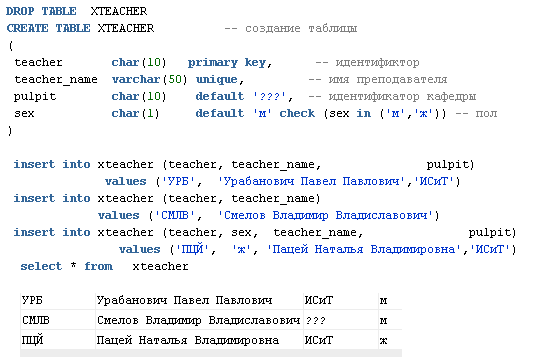
1. **Типы данных MSSQL. Создание таблиц. Специальные типы identity, uniqueidentifier. Разреженные таблицы.**

| **Вер. MSSQL** | **Тип данных** | **Тип Firebird** | **Определение MSSQL и пояснения** |
| --- | --- | --- | --- |
| 6.5 | **bigint** | **INT64** | 8-байтные целочисленные данные. |
| 6.5 | **binary** | **CHAR** | Двоичные данные фиксированной длины. Максимальная длина 8000 байт. В версии 6.5 максимальная длина была 255 байт. |
| 6.5 | **bit** | **CHAR(1)** | Целочисленные данные со значениями только 1 или 0. Обычно заменяются константами 'T' и 'F'. (В СУБД Firebird так же можно использовать тип SMALLINT со значениями 0 и 1. - прим. перев.) |
| 6.5 | **char** | **CHAR** | Текстовые данные фиксированной длины (не-Unicode). Максимальная длина 8000 символов. В версии 6.5 максимальная длина была 255 байт. СУБД Firebird способна хранить до 32767 символов. |
| 6.5 | **cursor** |  | Ссылка на курсор. Этот тип используется только в хранимых процедурах и триггерах; этот тип не может использоваться при объявлении структуры таблиц. |
| 6.5 | **datetime** | **TIMESTAMP** | Дата и время с 1 января 1753 года до 31 декабря 9999 года, точность 3/100 секунды (3.33 мс). |
| 6.5 | **decimal** | **DECIMAL** | Числовые данные с фиксированной точностью. Диапазон от -1038-1 до 1038-1. |
| 6.5 | **float** | **FLOAT** | Вещественные числовые данные. Диапазон от -1.79E+38 до 1.79E+38. |
| 6.5 | **image** | **BLOB** | Двоичные данные переменной длины. Максимальная длина 231-1 (2147483647) байт. |
| 6.5 | **int** | **INTEGER** | 4-байтные целочисленные данные. Диапазон от -231 (-2147483648) до 231-1 (2147483647). |
| 6.5 | **money** | **DECIMAL(18,4)** | Денежные данные. Диапазон от -263 (-922337203685477.5808) до 263-1 (+922337203685477.5807), точность до 1/10000 денежной единицы. |
| 7 | **nchar** | **CHAR(x) CHARACTER SET UNICODE\_FSS** | Символные данные фиксированной длины (Unicode). Максимальная длина 4000 символов. |
| 7 | **ntext** | **BLOB SUB\_TYPE TEXT** | Символьные данные переменной длины (Unicode). Максимальная длина 230-1 (1073741823) символов. |
| 6.5 | **numeric** | **NUMERIC** | В СУБД MS SQL decimal и numeric эквивалентны. |
| 7 | **nvarchar** | **VARCHAR(x) CHARACTER SET UNICODE\_FSS** | Символьные данные переменной длины. Максимальная длина 4000 символов. |
| 6.5 | **real** | **DOUBLE PRECISION** | Вещественные данные. Диапазон от -3.40E+308 до 3.40E+308. |
| 6.5 | **smalldatetime** | **TIMESTAMP** | Дата и время с 1 января 1900 года до 6 июня 2079 года, точность 1 минута. Тип данных Firebird имеет бОльшие диапазон и точность. |
| 6.5 | **smallint** | **SMALLINT** | 2-байтные целочисленные данные. Диапазон от -215 (-32768) до 215-1 (32767). |
| 6.5 | **smallmoney** | **DECIMAL(10,4)** | Денежные данные. Диапазон от -214748.3648 до +214748.3647, точность 1/10000 денежной единицы. Диапазон СУБД Firebird больше при указанной замене. |
| 2000 | **sql\_variant** | **BLOB** | Данные различных типов. |
| 2000 | **table** | нет эквивалента | Промежуточные результаты выполнения запроса для последующего использования. |
| 6.5 | **text** | **BLOB SUB\_TYPE TEXT** | Символьные данные переменной длины (не-Unicode). Максимальная длина 231-1 (2147483647) символов. |
| 6.5 | **timestamp** | **INTEGER или BIGINT** | Уникальное для базы данных число. В СУБД Firebird Вам необходимо использовать механизм генераторов для этих целей. |
| 6.5 | **tinyint** | **SMALLINT** | 1-байтовое целочисленное значение без знака. Диапазон от 0 до 255. В СУБД Firebird нет эквивалентного типа. |
| 6.5 | **varbinary** | **VARCHAR** | Двоичные данные переменной длины. Максимальная длина 8000 байт. |
| 6.5 | **varchar** | **VARCHAR** | Символьные данные переменной длины (не-Unicode). Максимальная длина 8000 символов. СУБД Firebird способна хранить до 32765 символов. В СУБД MS SQL 6.5 максимум был 255 символов. |
| 7 | **uniqueidentifier** | **CHAR(36)** | Глобально уникальный идентификатор (GUID). В СУБД Firebird Вам неоходимо использовать функции, определяемые пользователем (UDF), для генерации значения идентификатора. (Если Вы собираетесь индексировать поле с глобально уникальными идентификаторами, то лучше использовать UUID - это другой формат представления GUID, представляемый как CHAR(22). - прим. перев.) |

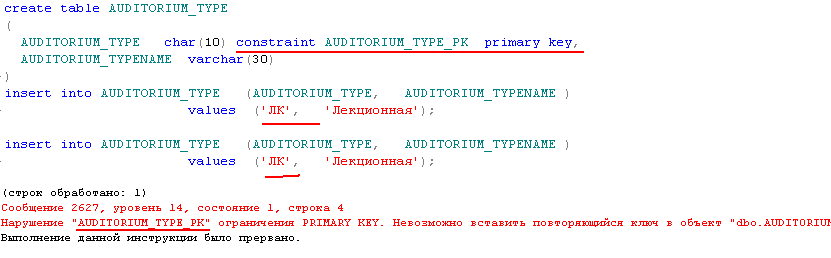
Разреженная таблица — двумерная структура данных ST[i, j], для которой выполнено следующее: ST[i,j]=\min\left(A[i], A[i+1], ..., A[i+2^{j}-1]\right),\quad j \in [0 \log N]. Иначе говоря, в этой таблице хранятся минимумы на всех отрезках, длины которых равны степеням двойки. Объём, занимаемый таблицей, равен O(N \log N), и заполненными являются только те элементы, для которых i+2^j \le N.

****

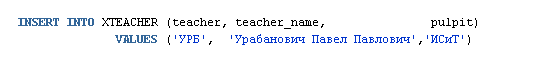
1. **Ограничения целостности: primary key, unique, check, default, (not)null, foreign key. Констрейнт.**

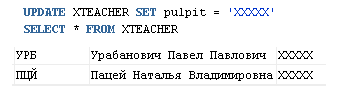
****

ALTER TABLE LINK ADD CONSTRAINT UNQ UNIQUE(ID1, ID2);

****

1. **Операторы DML: select, insert, update, delete.**

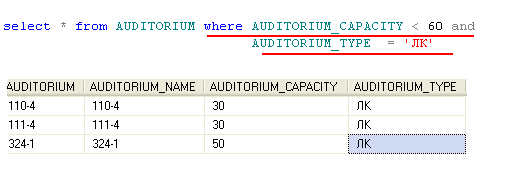
****

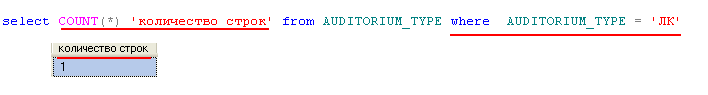
****

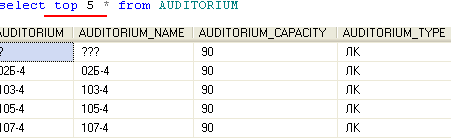
**12**

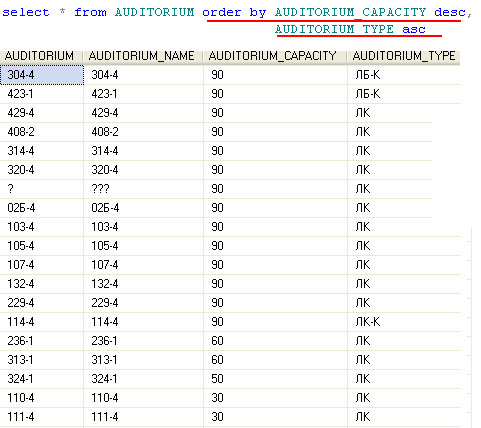
**18**

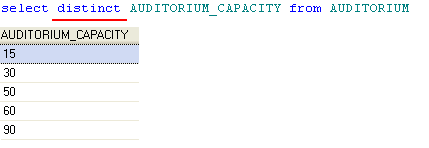
1. **Оператор select. Секции from, where, group by (all, cube, rollup), having, order by (asc, desc), distinct, top, compute.**

****

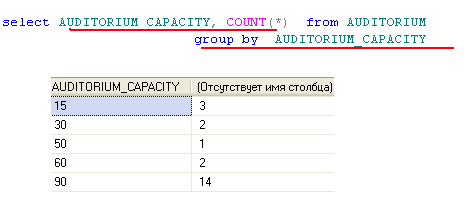


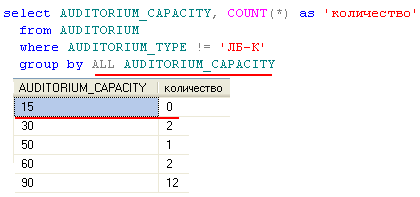
****

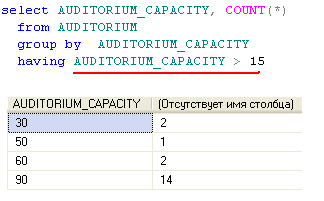
****

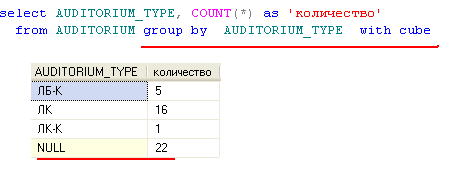
****

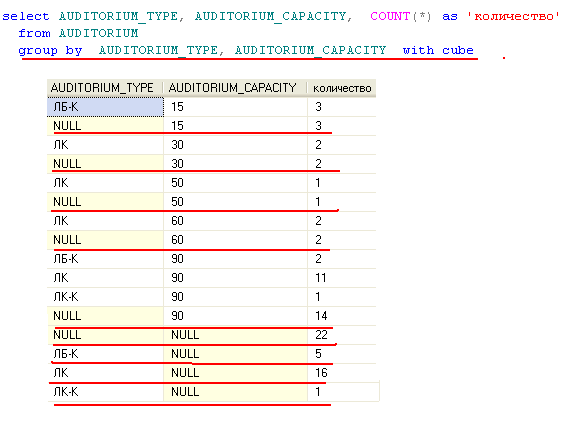
Distinct - выбираемые значения не повторяются

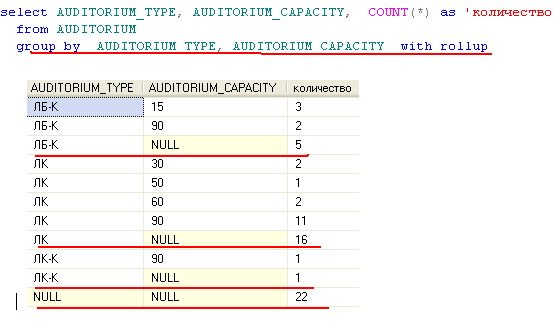


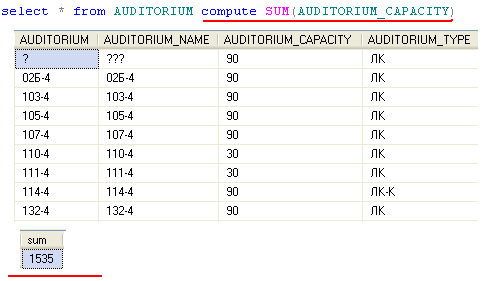


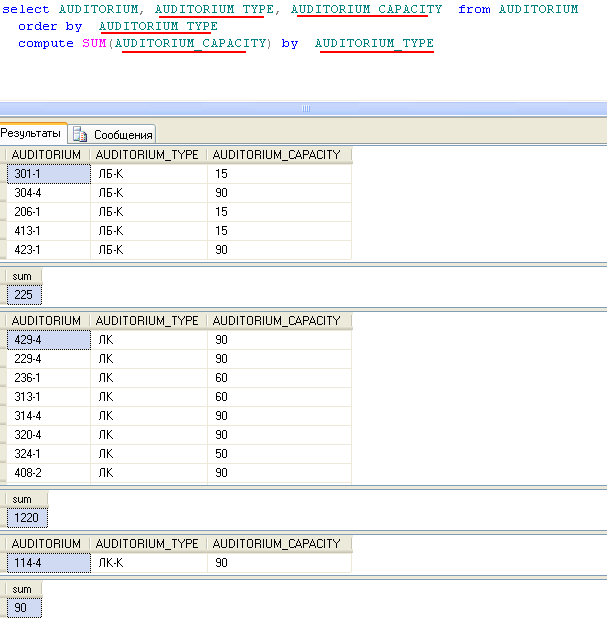
 аналог where

****

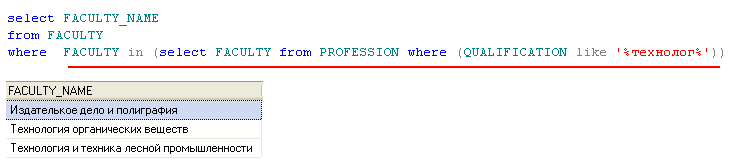
****

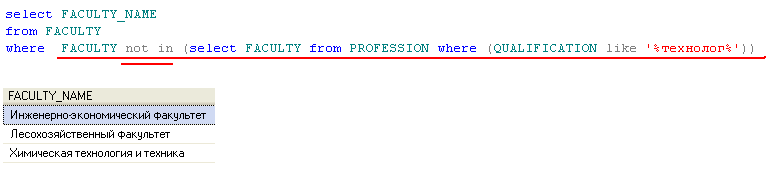
****





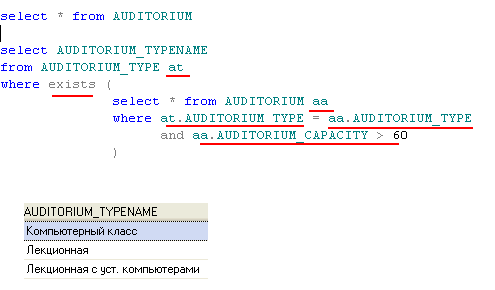
1. **Подзапросы. Конструкции (not)in, not exists, all, any, some.**

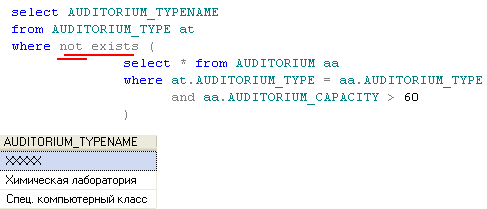
****

****

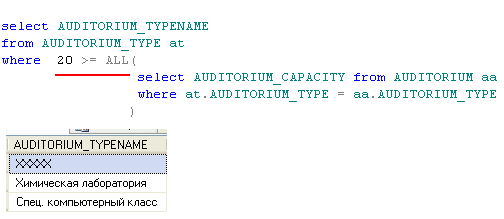
In-если есть хоть один элемент, то истина

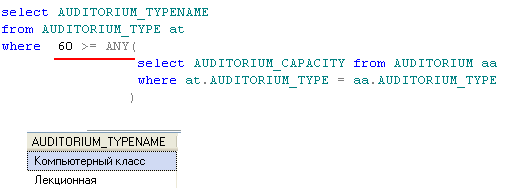
Not in-наоборот

****

****

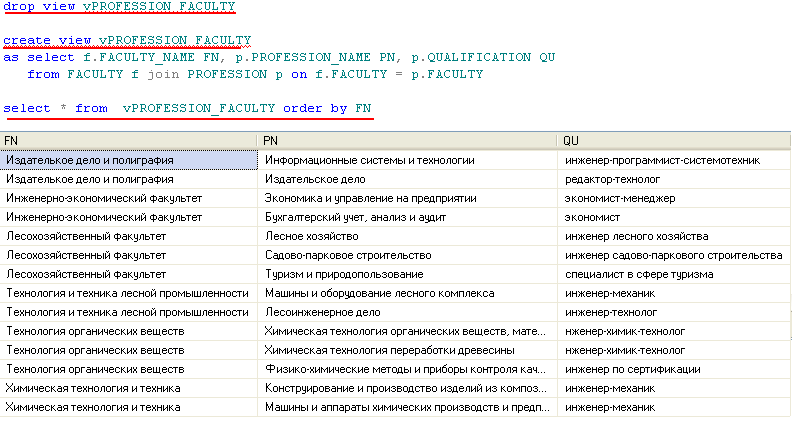
exists - выдает истину, если выбрана хотя бы одна строка

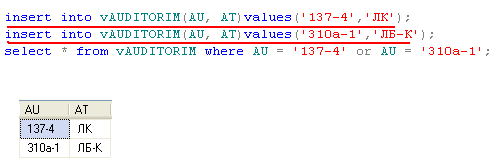
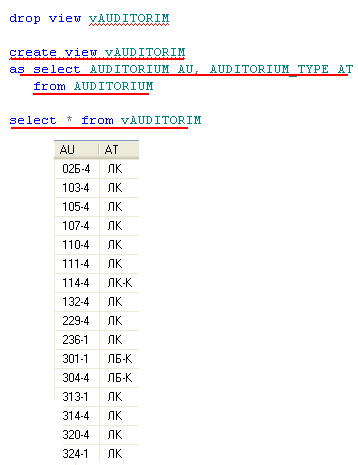
****

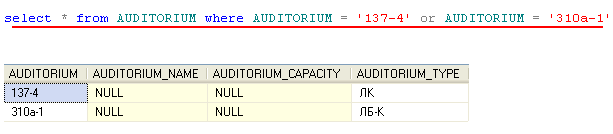
****

1. **Представления (view) БД. Создание представлений БД. Операции DML с представлениями. Представления с with check option.**

Представление (VIEW) — объект базы данных, являющийся результатом выполнения запроса к базе данных, определенного с помощью оператора SELECT, в момент обращения к представлению.  
  
Представления иногда называют «виртуальными таблицами». Такое название связано с тем, что представление доступно для пользователя как таблица, но само оно не содержит данных, а извлекает их из таблиц в момент обращения к нему. Если данные изменены в базовой таблице, то пользователь получит актуальные данные при обращении к представлению, использующему данную таблицу; кэширования результатов выборки из таблицы при работе представлений не производится. При этом, механизм кэширования запросов (query cache) работает на уровне запросов пользователя безотносительно к тому, обращается ли пользователь к таблицам или представлениям.  
  
Представления могут основываться как на таблицах, так и на других представлениях, т.е. могут быть вложенными (до 32 уровней вложенности).

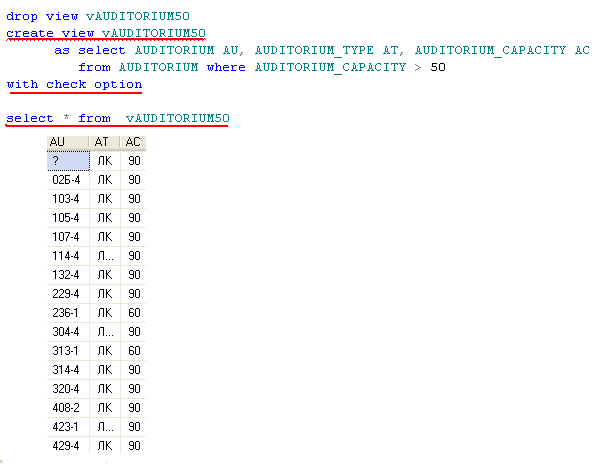
****

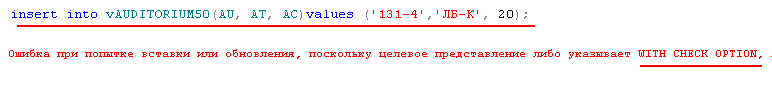
****

****

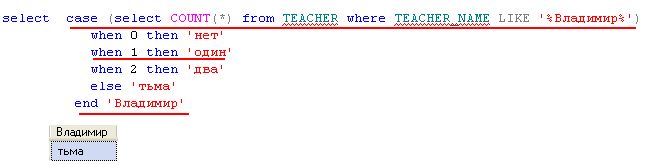
**25**

**26**

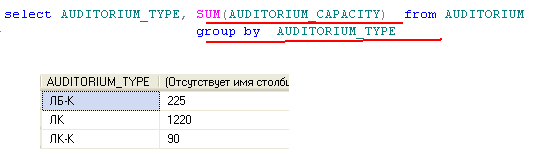
****

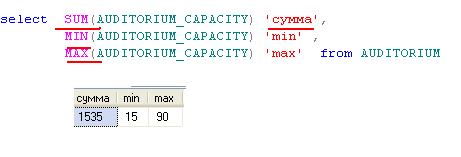
****

1. **Оператор case. Применение оператора case в секции order by оператора select.**

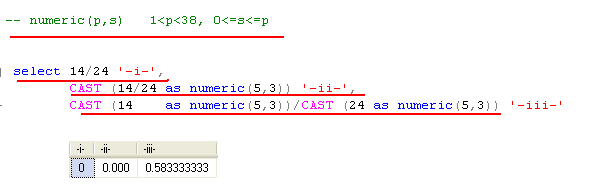
****

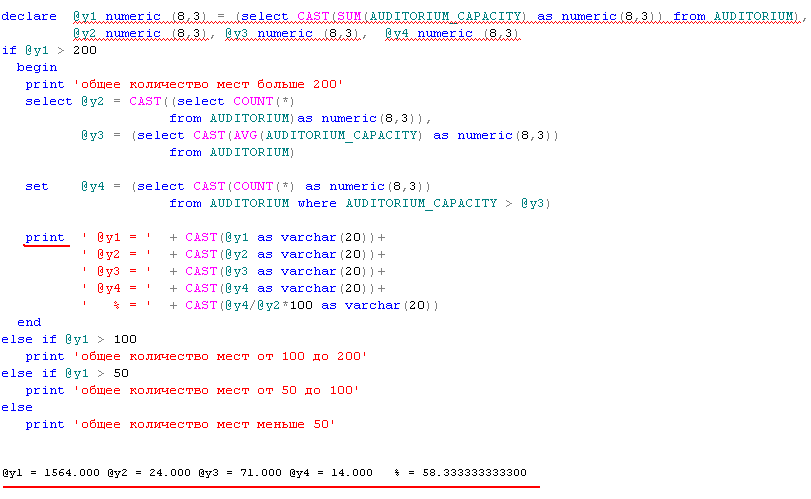
1. **Агрегатные функции: count, sum, avg, min, max.**



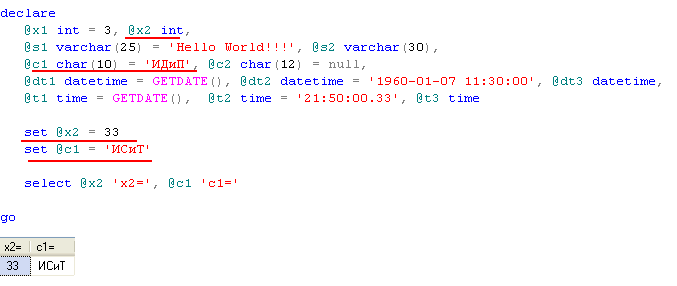


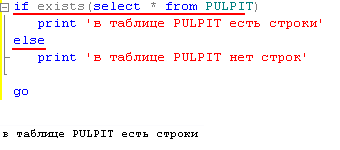
1. **Функция преобразования типов данных cast.**

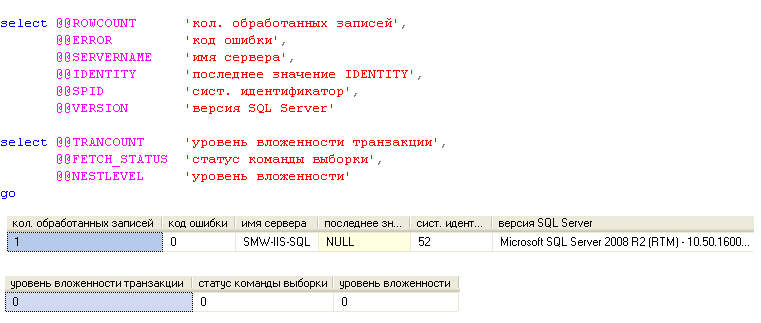
****

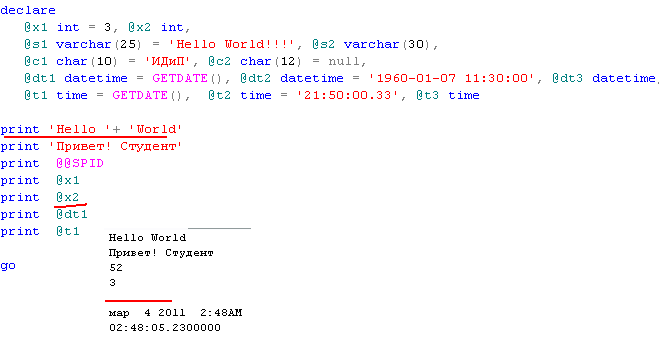
****

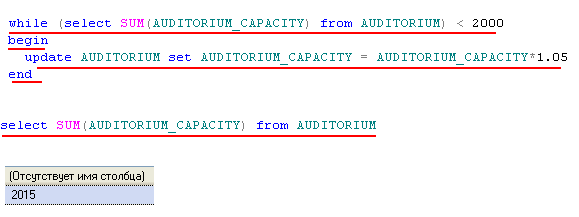
1. **Язык T-SQL. Объявление локальных переменных. Глобальные переменные. Операторы присвоения. Операторы print, if-else, begin-end, конкатенации строк, while, waitfor, return, try-catch.**

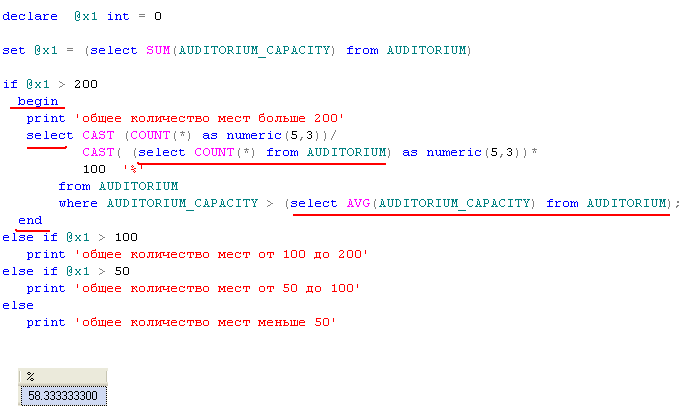
****

****

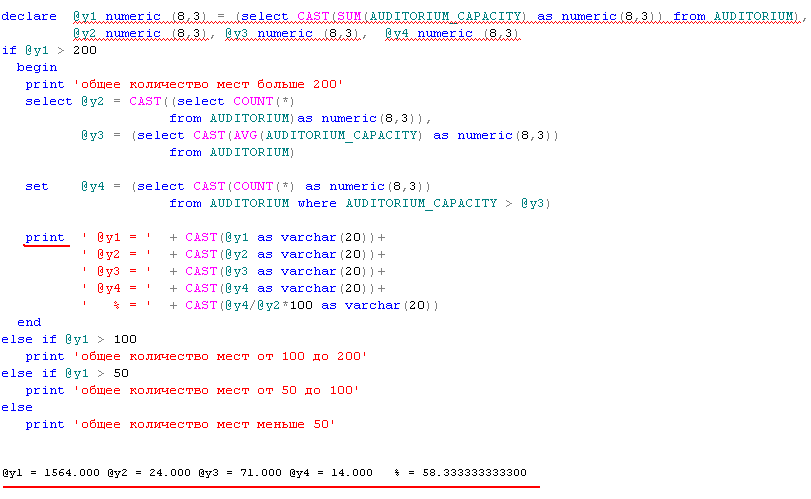
****

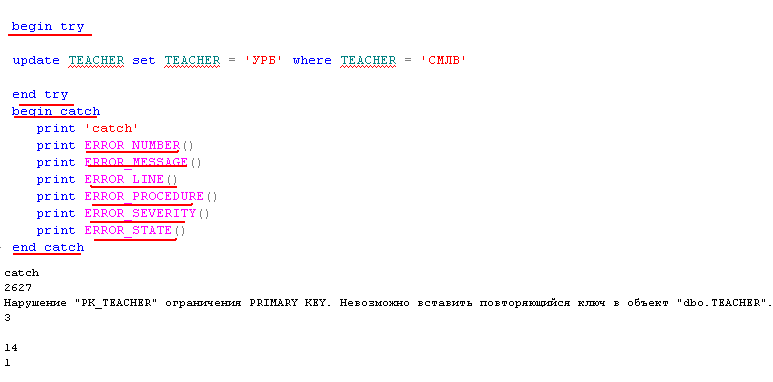
****

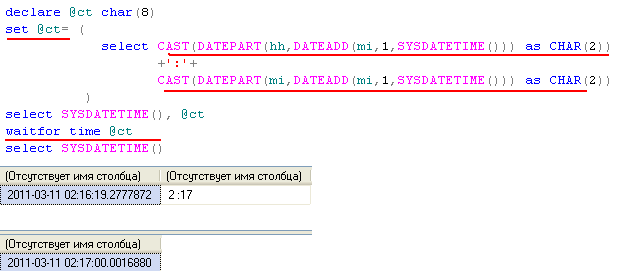
****

****

конкатенация строк

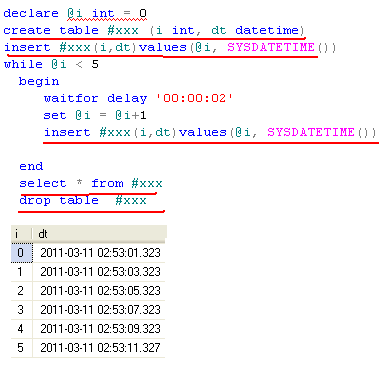
****

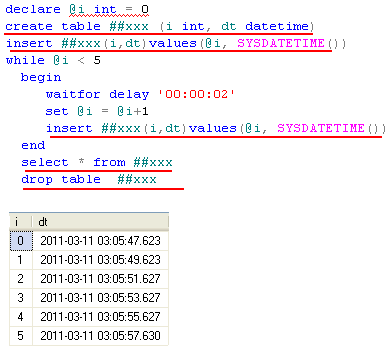
****

****

1. **Локальные и глобальные временные таблицы.**

Два типа временных таблиц, локальные и глобальные, отличаются друг от друга своими именами, своей видимостью и своей доступностью. Локальные временные таблицы имеют префикс в имени в виде символа номера (#); они видны только в текущем соединении пользователя и удаляются  после того, как пользователь отсоединится от экземпляра (instance) microsoft® sql server™ 2000. Глобальные временные таблицы имеют префикс имени в виде двойного символа #; они видны для любого пользователя после их создания и удаляются  тогда, когда все пользователи, ссылающиеся на таблицу, отсоединятся от sql server."

****

****

1. **Курсоры. Объявление курсора. Общая схема работы с курсором: declare, open, fetch, close, deallocate. Типы курсоров: global/local, read\_only, static/dynamic, scroll. Способы позиционирования в курсоре: relative/absolute, next/prior, first/last. Применение переменных запросе курсора. Функция fetch\_status. Применение секции where current of в операторах update, delete. Возврат кода процедурой.**

**Курсор:** механизм, позволяющий обрабатывать отдельные строки полученные в результате select-запроса.

**Курсор:** область памяти сервера, предназначенная для хранения и обработки результата select-запроса и характеризующаяся атрибутами.

**Типы курсоров:** динамические, статические, ключевые.

**Динамический курсор:** изменения данных отображаются в динамике.

**Статический курсор:** данные выбраны один раз и произошедшие изменения «не видны».

**Ключевые курсоры:**  большая часть данных отображается в динамике, «не видны» insert-строки, при обращении к delete-строкам возникает ошибка.

При использовании ключевого слова LOCAL будет создан локальный курсор, который виден только в пределах создавшего его пакета, триггера, хранимой процедуры или пользовательской функции. По завершении работы пакета, триггера, процедуры или функции курсор неявно уничтожается. Чтобы передать содержимое курсора за пределы создавшей его конструкции, необходимо присвоить его параметру аргумент OUTPUT.

Если указано ключевое слово GLOBAL, создается глобальный курсор; он существует до закрытия текущего соединения.

При указании FORWARD\_ONLY создается последовательный курсор; выборку данных можно осуществлять только в направлении от первой строки к последней.

При указании SCROLL создается прокручиваемый курсор; обращаться к данным можно в любом порядке и в любом направлении.

При указании STATIC создается статический курсор.

При указании KEYSET создается ключевой курсор.

При указании DYNAMIC создается динамический курсор.

При указании FIRST будет возвращена самая первая строка полного результирующего набора курсора, которая становится текущей строкой.

При указании LAST возвращается самая последняя строка курсора. Она же становится текущей строкой.

При указании NEXT возвращается строка, находящаяся в полном результирующем наборе сразу же после текущей. Теперь она становится текущей. По умолчанию команда FETCH использует именно этот способ выборки строк.

Ключевое слово PRIOR возвращает строку, находящуюся перед текущей. Она и становится текущей.

Аргумент ABSOLUTE {номер\_строки | @переменная\_номера\_строки} возвращает строку по ее абсолютному порядковому номеру в полном результирующем наборе курсора. Номер строки можно задать с помощью константы или как имя переменной, в которой хранится номер строки. Переменная должна иметь целочисленный тип данных. Указываются как положительные, так и отрицательные значения. При указании положительного значения строка отсчитывается от начала набора, отрицательного – от конца. Выбранная строка становится текущей. Если указано нулевое значение, строка не возвращается.

Аргумент RELATIVE {кол\_строки | @переменная\_кол\_строки} возвращает строку, находящуюся через указанное количество строк после текущей. Если указать отрицательное значение числа строк, то будет возвращена строка, находящаяся за указанное количество строк перед текущей. При указании нулевого значения возвратится текущая строка. Возвращенная строка становится текущей.

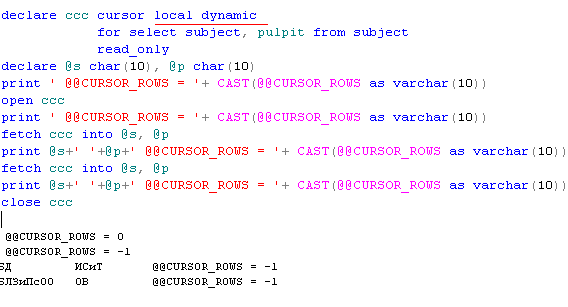
Для контроля достижения конца курсора рекомендуется применять функцию: @@FETCH\_STATUS

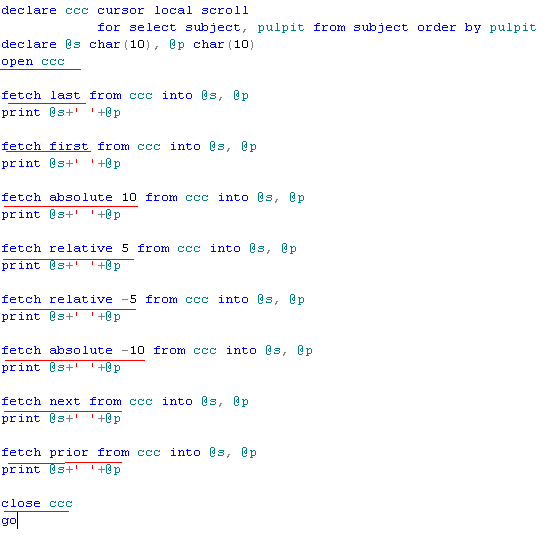
Функция @@FETCH\_STATUS возвращает:

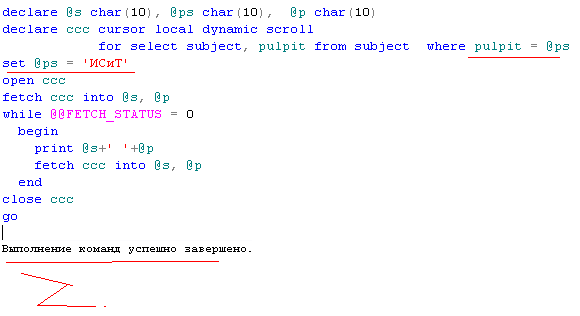
0, если выборка завершилась успешно;

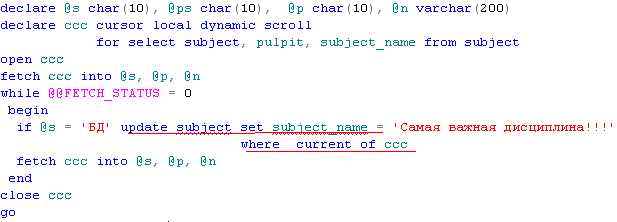
-1, если выборка завершилась неудачно вследствие попытки выборки строки, находящейся за пределами курсора;

-2, если выборка завершилась неудачно вследствие попытки обращения к удаленной или измененной строке.

****

****

****

****

1. **Хранимые процедуры и функции T-SQL. Создание хранимых процедур и функций (скалярные, табличные). Передача параметров. Входные и выходные параметры,**

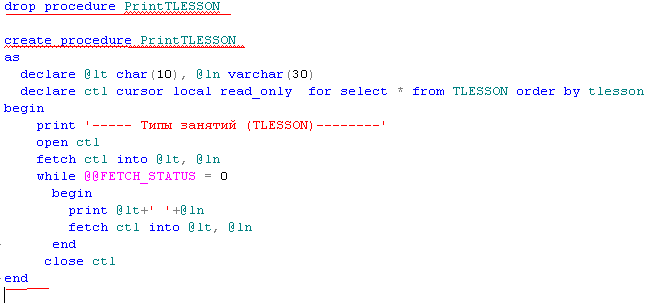
Хранимая процедура - это подпрограмма, состоящая из SQL операторов и команд T-SQL. Хранимая процедура сохраняется на сервере и выполняется по команде пользователя или вызывается из блока T-SQL. План выполнения процедуры подготавливается во время запуска процедуры, поэтому собственно выполнение процедуры происходит очень быстро. Хранимая процедура может:

Содержать параметры (аргументы); Вызывать другие процедуры; Возвращать свой статус вызывающей процедуре или модулю T\_SQL, указывающий на успешное окончание или ошибку, и в случае ошибки на ее причину; Возвращать значения параметров вызывающей процедуре или модулю T-SQL.

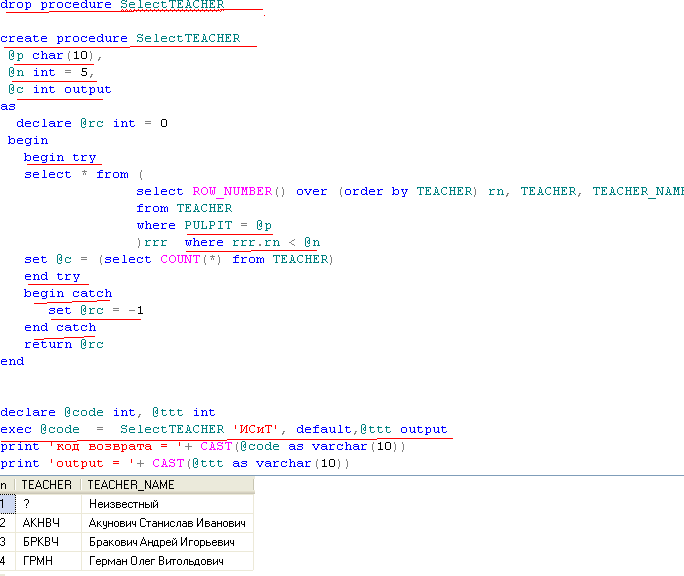
**Хранимая процедура:** объект базы данных

**Хранимая процедура:** поименованный блок (BEGIN/END) операторов, хранящийся в базе данных в откомпилированном виде.

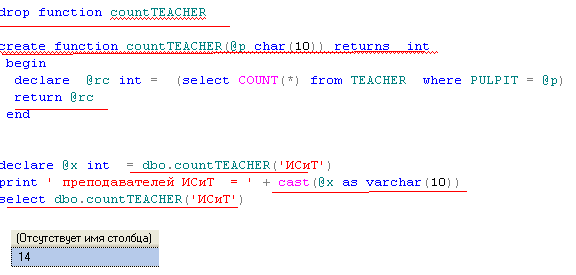
**Хранимая процедура:** системные процедуры, пользовательские процедуры.

****

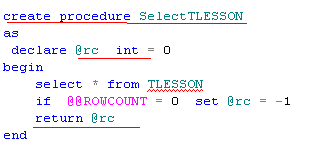
передача праметров

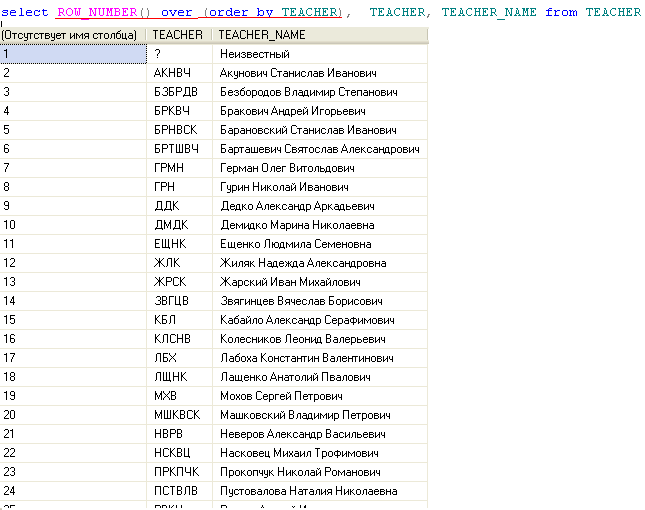


1. **Пользовательские функции: скалярные** (возвращает одно значение, нельзя timestamp, cursor, text, ntext, image, table),

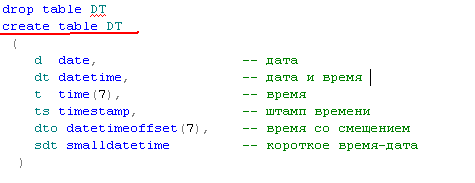
****

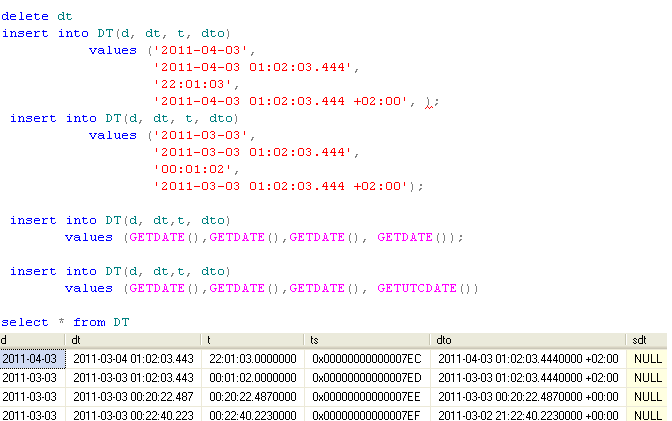
1. **Встроенные функции: rowcount, row\_number over,**

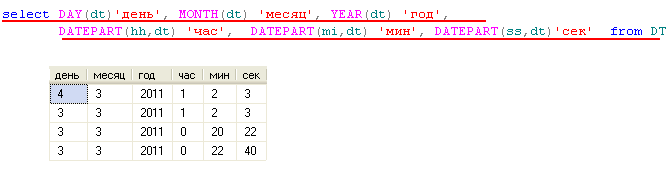




1. **Функции работы со временем: day, month, year, datepart.**

****

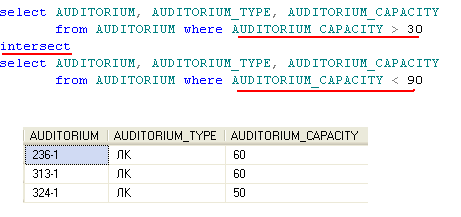
****

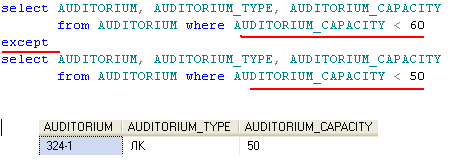
****

1. **Операторы union (all, order by), intersect, except.**

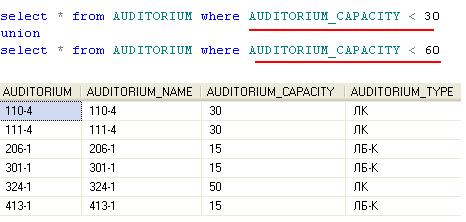
Оператор EXCEPT возвращает все различные значения, возвращенные левым запросом и отсутствующие в результатах выполнения правого запроса.

Оператор INTERSECT возвращает все различные значения, входящие в результаты выполнения, как левого, так и правого запроса.

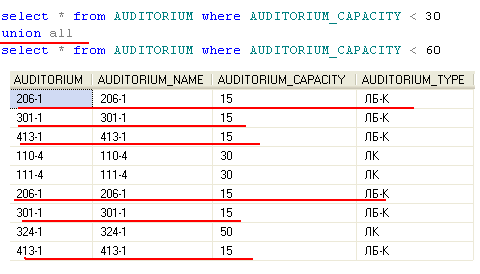
****

****

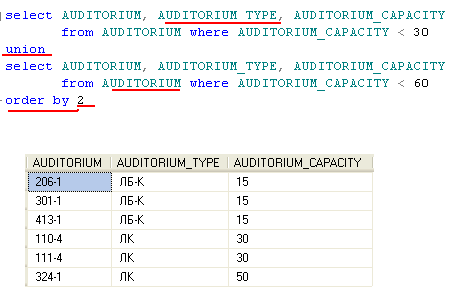
union all – сохраняет дубликаты



**Оператор UNION ALL**

****

**UNION ORDER BY**

****

1. **Соединение таблиц. Естественное соединение с помощью секции where. Внутреннее соединение inner join. Ортогональное соединение cross join. Внешние соединения: left(right) outer join, full outer join.**

Тип соединения "внутреннее". Внутренний тип соединения используется по умолчанию, когда тип явно не задан (то есть, опущено ключевое слово INNER). Если не приведено ни одного условия совпадения, то будет возвращено декартово произведение, в котором каждая строка одной таблицы будет сопоставлена с каждой строкой другой таблицы. То есть будут получены всевозможные сочетания из обеих таблиц. Такое соединение в разделе FROM может быть заменено списком таблиц через запятую.

|  |
| --- |
| [Выделить всё](javascript:%20selectAll('sql_TVkYTU'))**код SQL** |
| |  |  | | --- | --- | | 1: | **select**\* from sport inner join music; | |

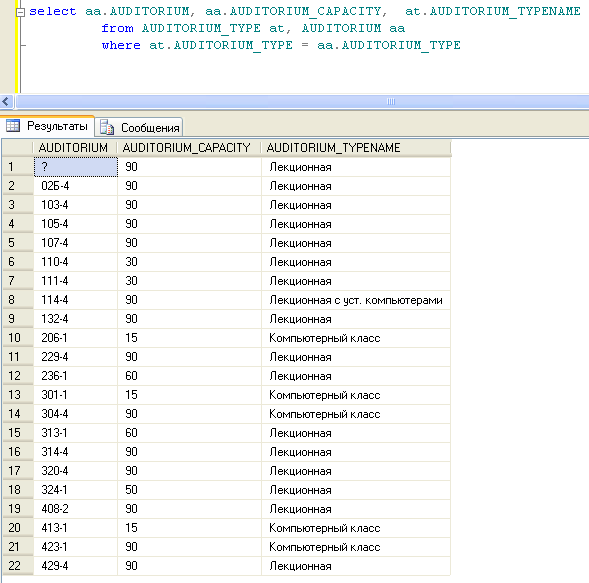
LEFT (OUTER) - Тип соединения "левое (внешнее)". Левое соединение таблиц А и В включает в себя все строки из левой таблицы А и те строки из правой таблицы В, для которых обнаружено совпадение. Для строк из таблицы А, для которых не найдено соответствия в таблице В, в столбцы, извлекаемые из таблицы В, заносятся значения NULL.

|  |
| --- |
| [Выделить всё](javascript:%20selectAll('sql_JjZTA5'))**код SQL** |
| |  |  | | --- | --- | | 1: | **select**\* from sport left outer join music on sport.name=music.name; | |

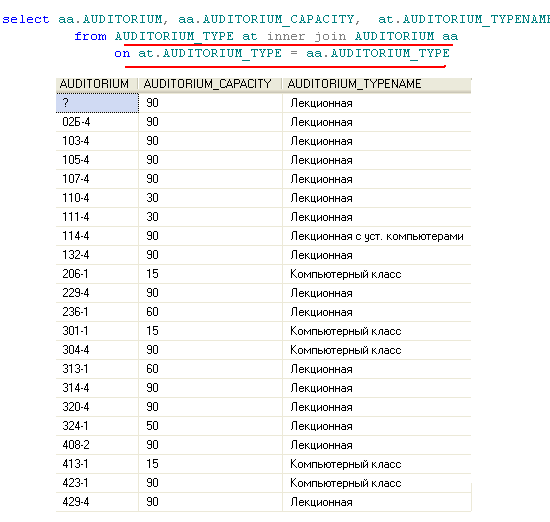
Тип соединения "правое (внешнее)". Правое соединение таблиц А и В включает в себя все строки из правой таблицы В и те строки из левой таблицы А, для которых обнаружено совпадение. Для строк из таблицы В, для которых не найдено соответствия в таблице А, в столбцы, извлекаемые из таблицы А заносятся значения NULL.

|  |
| --- |
| [Выделить всё](javascript:%20selectAll('sql_DEzZmE'))**код SQL** |
| |  |  | | --- | --- | | 1: | **select**\* from sport right outer join music on sport.name=music.name; | |

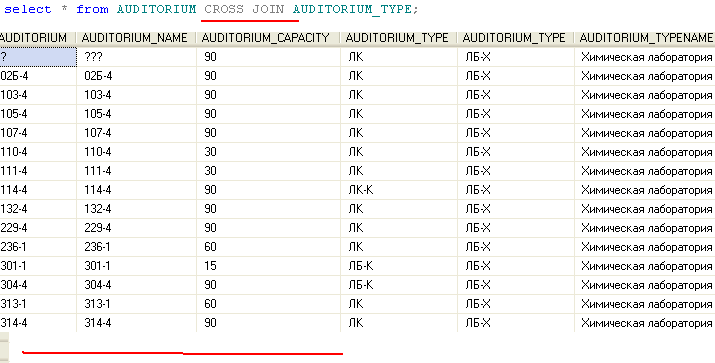
1. **Естественное соединение таблиц**

****

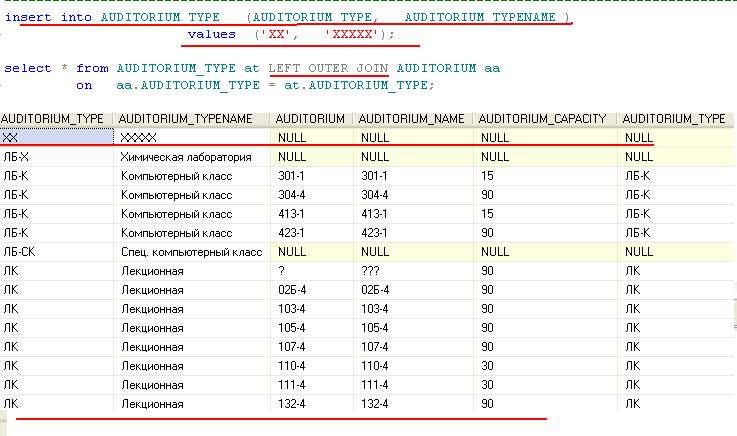
1. **Внутреннее соединение INNER JOIN**

****

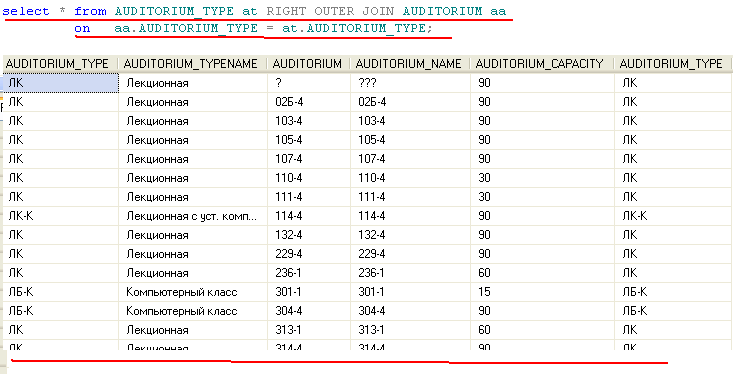
1. **Ортогональное соединение: CROSS JOIN**

****

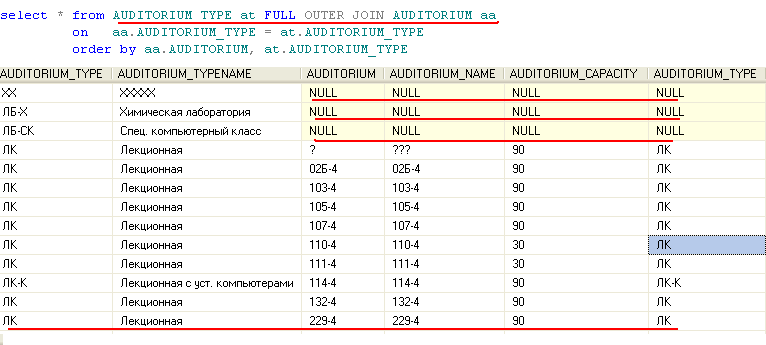
1. **LEFT OUTER JOIN**



1. **RIGHT OUTER JOIN**

****

1. **FULL OUTER JOIN**

****

1. **Триггеры. Типы триггеров. Создание after-триггера. Создание и назначение instead of-триггеров. Таблицы inserted, deleted. Назначение и применение системной процедуры sp\_settriggerorder. Вложенные триггеры.**

Триггер представляет собой специальный тип хранимых процедур, запускаемых сервером автоматически при попытке изменения данных в таблицах, с которыми триггеры связаны. Каждый триггер привязывается к конкретной таблице. Все производимые им модификации данных рассматриваются как одна транзакция. В случае обнаружения ошибки или нарушения целостности данных происходит откат этой транзакции. Тем самым внесение изменений запрещается. Отменяются также все изменения, уже сделанные триггером.

Вложенность триггеров может достигать максимум 32 уровня. Если триггер изменяет таблицу, для которой определен другой триггер, то запускается второй триггер, вызывающий срабатывание третьего и т.д. Если любой из триггеров в цепочке отключает бесконечный цикл, то уровень вложенности превышает допустимый предел, и срабатывание триггера отменяется. Если триггер на языке Transact-SQL выполняет управляемый код с помощью ссылки на метод, тип или статистическую функцию среды CLR, эта ссылка считается одним из допустимых 32 уровней вложенности. Методы, вызываемые из управляемого кода, не учитываются в этом ограничении.

В таблице deleted находятся копии строк, с которыми работали инструкции DELETE или UPDATE. При выполнении инструкции DELETE или UPDATE происходит удаление строк из таблицы триггера и их перенос в таблицу deleted. У таблицы deleted обычно нет общих строк с таблицей триггера.

В таблице inserted находятся копии строк, с которыми работали инструкции INSERT или UPDATE. При выполнении транзакции вставки или обновления происходит одновременное добавление строк в таблицу триггера и в таблицу inserted. Строки таблицы inserted являются копиями новых строк таблицы триггера.

Если для таблицы предусмотрено выполнение нескольких AFTER-триггеров, то с помощью системной хранимой процедуры sp\_settriggerorder можно указать, какой из них будет выполняться первым, а какой последним.

**Триггер:** специальный вид хранимых процедур, выполняющихся при некоторых событиях базы данных.

**Триггер:** назначение и применение.

**Триггер:** DML-триггеры, DDL-триггеры.

**DML-триггеры:** создаются для таблицы или представления.

**DML-триггеры:** реагируют на события INSERT, DELETE, UPDATE.

**DML-триггеры:** для одной таблицы или представления может быть несколько триггеров. Частично (первый и последний) порядок триггеров может быть установлен ч помощью процедуры sp\_settriggerorder (не работает для INSTEAD OF).

**DML-триггеры:** AFTER-триггеры, INSTEAD OF-триггеры.

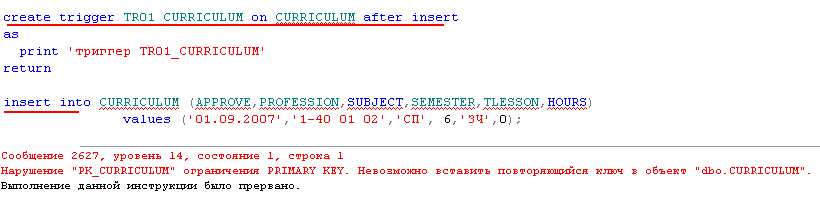
**AFTER-триггеры:** триггеры уровня оператора (выполняются по одному разу для каждого оператора), выполняются после наступления события.

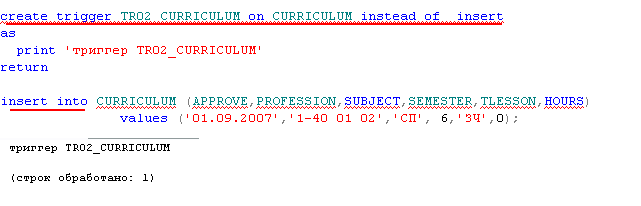
**INSTEAD OF-триггеры**: триггеры уровня оператора (выполняются по одному разу для каждого оператора), выполняются вместо операции (сама операция не выполняется). INSTEAD OF-триггеры не могут вызываться рекурсивно (если в триггере сработает операция, снова вызвавшая работу триггера). Если образуется рекурсия вызовов триггеров, то будет сделана попытка выполнить оператор.

**DDL-триггеры:** AFTER-триггеры: CREATE, ALTER, DROP...., LOGON (при подключении).

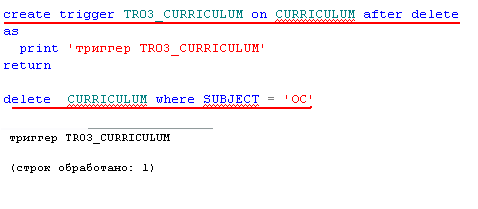
**Ограничения:** во всех триггерах запрещено ряд команд: CREATE, DROP, ALTER TABLE/DATABASE и ...

**Пример DML:**

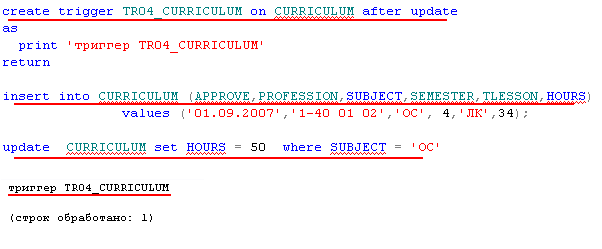
****

****

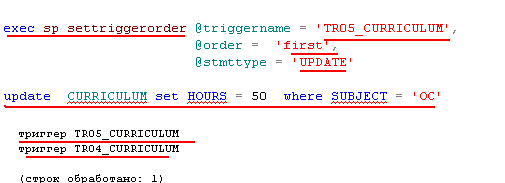
**AFTER DELETE**

****

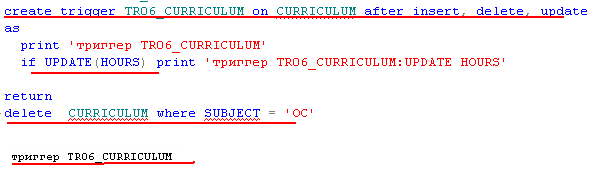
**AFTER UPDATE**

****

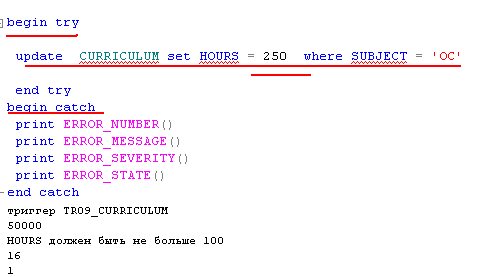
**sp\_settriggerorder**

****

**AFTER UPDATE, DELETE, UPDATE**

****

1. **Обработка исключений конструкцией try/catch. Функция raiseerror.**

****

**Формирование сообщений об ошибках. Функция RAISERROR().**

Данная функция позволяет формировать сообщение об ошибках.

Формат:

RAISERROR(‘Сообщение’, степень\_серьезности, состояние)

Параметр ‘Сообщение’ – это текст, который будет читать пользователь при нарушении правил триггера.

Параметр ‘степень\_серьезности’ указывает системе минимальный уровень серьезности ошибки.

Ошибкам можно задавать разную степень серьезности, начиная с 1 и заканчивая 25(самая критическая ошибка).

Параметр состояние используется в том случае, если отдельная ошибка была вызвана из нескольких точек триггера.

**пример**: допустим, у нас есть таблица «Sotrydnik» БД “Ludi”. Необходимо запретить удалять запись о сотруднике оклад которого более 50000 рублей.

IF (SELECT OKLAD FROM DELETED)=’50000’

BEGIN

RASEERROR(‘Нельзя удалять запись о сотруднике, с окладом более 50000’,10,1)

ROLLBACK

END

1. **Индексы БД. Назначение и применение индексов. Кластерные и некластерные индексы.**
2. **Индекс:** объект базы данных (CREATE, DROP).
3. **Индекс:** всегда связан с таблицей(с подмножеством столбцов таблицы).
4. **Индекс:** предназначен для ускорения поиска строк в таблице по индексируемым столбцам
5. **Индекс:**  Microsoft SQL Server бывают кластерные, некластерные (просто индексы).
6. **Некластерный индекс:** физически находится отдельно от таблицы, список значений индексируемого столбца (столбцов в определенном порядке) с указателем на строку в таблице; список, как правило, бинарное дерево поиска.
7. **Некластерный индекс:** может быть несколько для одной таблицы.
8. **Кластерный индекс:** физически совпадает с таблицей, таблица физически перестраивается в порядке индекса.
9. **Кластерный индекс:** может только один для одной таблицы
10. **Индекс:** кластерный индекс следует создавать раньше некластерных, т.к. при создании кластерный индекс перестраивает таблицу.
11. **Индекс:** бывают UNIQ (запрещается повторение значений индексного выражения), не UNIQ.
12. **Индекс:** для первичных ключей и для UNIQ-столбцов автоматически создается индекс.
13. **Индекс:** можно посмотреть в SMS.

**Индексы** представляют собой структуру, позволяющую выполнять ускоренный доступ к строкам таблицы на основе значений одного или более ее столбцов. Наличие индекса может существенно повысить скорость выполнения некоторых запросов и сократить время поиска необходимых данных за счет физического или логического их упорядочивания. Индекс – это набор ссылок, упорядоченных по определенному столбцу таблицы, который в данном случае будет называться индексированным столбцом. Хотя индекс и связан с конкретным столбцом (или столбцами) таблицы, все же он является самостоятельным объектом базы данных.

****

1. **Транзакции. Свойства ACID. Блокировки. Операторы TCL. Уровни изоляции. Функция trancount.**

**Транзакция:** одна или несколько команд SQL, которые либо успешно выполняются как единое целое, либо отменяются как единое целое.

**Транзакция:** логическая единица работы, обеспечивающая переход базы данных из одного согласованного (непротиворечивого) состояния в другое согласованное состояние.

1. **ACID:**  требование к транзакционной системе

**Atomicity (Атомарность)** –транзакция либо завершается полностью и успешно, либо полностью не выполняется (аварийно завершается);

**Consistency(Согласованность)** – транзакция (успешно или аварийно завершенная) всегда оставляет базу данных в согласованном состоянии;

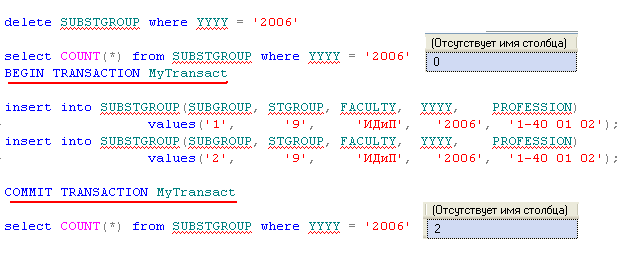
**Isolation(Изолированность)** - дофиксацииизменения видит только сеанс, выполнивший эти изменения; есть различные уровни изолированности;

**Durability (Устойчивость, долговечность)** - после фиксациитранзакцию нельзя отменить.

**Транзакция:** TCL: BEGIN TRANSACTION, COMMIT TRANSACTION, ROLLBACK TRANSACTION.

**commit** сохраняет изменения

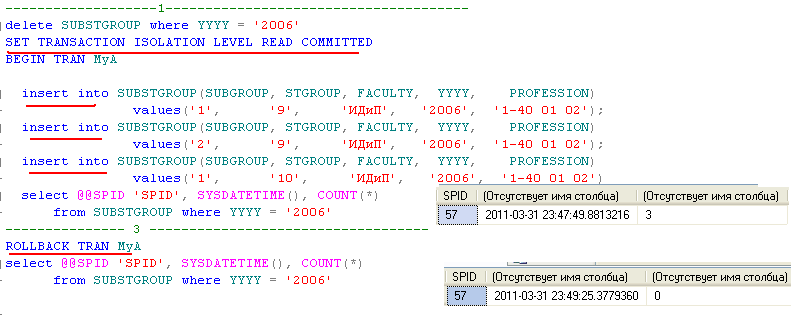
**rollback** откатывает транзакцию



1. **Транзакция:** уровни изоляции (ANSI SQL-92), читатели писатели
2. **READ UNCOMMITED** (неподтвержденное (грязное) чтение). Читатель может прочитать незафиксированные строки. Читатель не может изменить незафиксированные строки (ожидает);
3. **READ COMMITED** (подтвержденное чтение). Читатель не может прочитать неподтвержденные данные (ожидает), писатель может изменить и удалить уже прочитанные читателем данные;
4. **REPETABLE READ** (повторяемое чтение). В начале транзакции и в конце транзакции результат одного и того же запроса читателя будет одинаков; писатели могут вводить данные, что приводит к появлению ***фантомных записей***;
5. **SERIALIZABLE** (сериализуемость). Высший уровень изоляции. Данные не могут не прочитаны ни изменены до окончания транзакции.

**SNAPSHOT (**снимок**).** Читатель считывает версию данных на момент начала транзакции писателя.

1. **READ UNCOMMITED**

****

может быть наложена как на отдельную строку таблицы, так и на всю базу данных. Управлением блокировками на сервере занимается менеджер блокировок, контролирующий их применение и разрешение конфликтов. Транзакции и блокировки тесно связаны друг с другом. Транзакции накладывают блокировки на данные, чтобы обеспечить выполнение требований ACID. Без использования блокировок несколько транзакций могли бы изменять одни и те же данные.

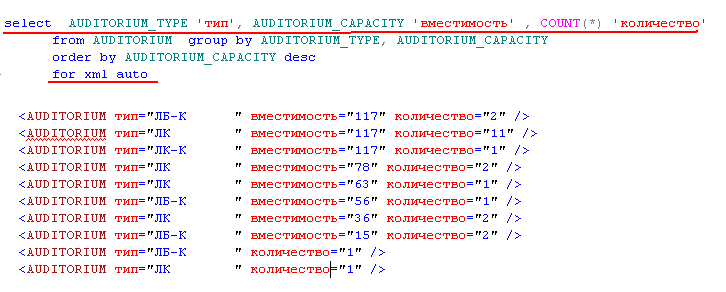
Блокировка представляет собой метод управления параллельными процессами, при котором объект БД не может быть модифицирован без ведома транзакции, т.е. происходит блокирование доступа к объекту со стороны других транзакций, чем исключается непредсказуемое изменение объекта. Различают два вида блокировки:

* блокировка записи – транзакция блокирует строки в таблицах таким образом, что запрос другой транзакции к этим строкам будет отменен;
* блокировка чтения – транзакция блокирует строки так, что запрос со стороны другой транзакции на блокировку записи этих строк будет отвергнут, а на блокировку чтения – принят.

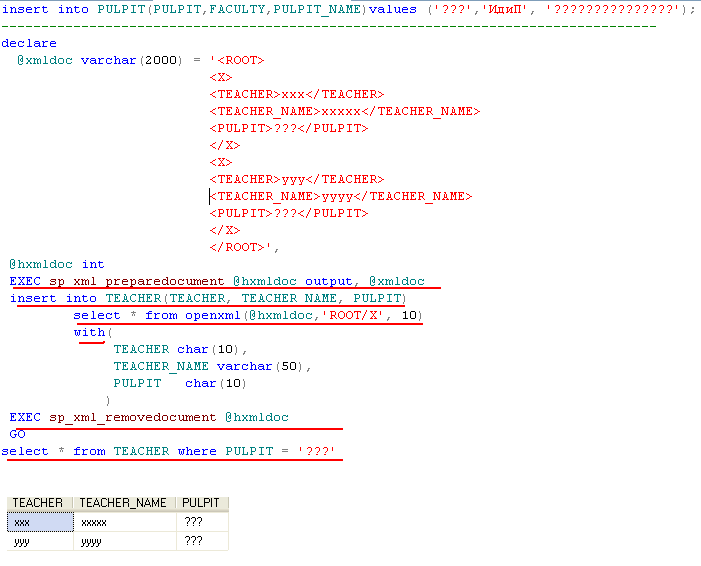
**функция** @@**TRANCOUNT** используется для определения числа открытых транзакций.

1. **Язык XML. Секция for XML оператора select. Применение системных процедур sp\_xml\_preparedocument, sp\_xml\_removedocument. Применение системной функции openxml.**

**FOR XML**Этот оператор предназначен для представления результирующего набора строк в виде XML-документа

****

Поскольку XML как таковой является строкой, то для того чтобы SQL Server мог работать с XML как с набором данных, нужно ему об этом рассказать. Сделать это можно при помощи системной хранимой процедуры **sp\_xml\_preparedocument**. Процедура **sp\_xml\_preparedocument** создает представление указанного XML-документа (по сути, таблица) и передает его дескриптор (идентификатор, числовой код). Последующая работа с XML происходит именно через дескриптор, посредствам функции **OPENXML**. После завершения работы, документ можно удалить при помощи процедуры **sp\_xml\_removedocument**, хотя это делать вовсе и не обязательно, поскольку данные и так удалятся по завершению работы текущей сессии.

****

1. **Системные базы данных MSSQL: master, msdb, model, tempdb.Структура файла базы данных.**

**Базы данных**: системные, пользовательские.

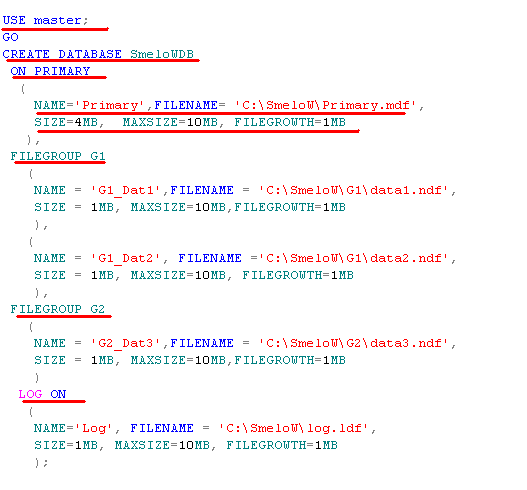
**Системные базы данных**

* **master**: информация системного уровня: параметры настройки, о других базах данных (о файлах базы данных), учетные записи и др.
* **msdb:** для планирования работы заданий;
* **model:** шаблонбазы данных;
* **tempdb**:временные объекты (таблицы, процедуры, курсоры).

1. **База данных содержит файлы трех типов:**
   * **первичные файлы данных**: это один файл, метаинформация, расположение других файлов, имя **\*.MDF**;
   * **вторичные файлы данных:** ни одного или несколько файлов, данные, имя **\*.NDF**;
   * **файлы журналов:** один или несколько файлов, информация для восстановления базы данных, имя **\*.LDF**;

**Структура файла базы данных:** страница 8K, 1MB = 128 страниц, 4MB = 512 страниц

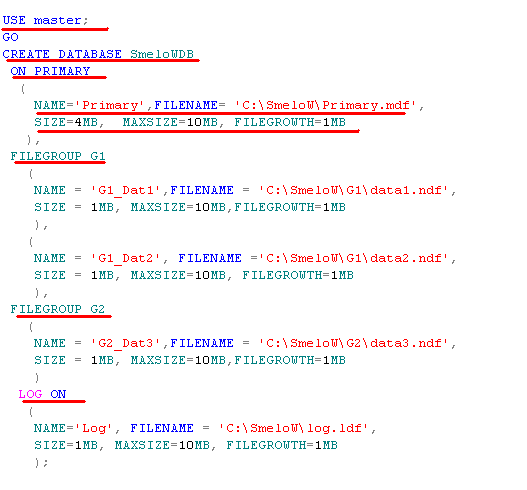
1. **Типы страницы:** данные, индексы, текст/изображение, карты распределения экстентов, свободное место на страницах, карта распределения индекса, схема массовых изменений, схема разностных изменений.
2. **Экстенты:** 8непрерывных страниц; однородные, смешанные.

****

1. **Создание баз данных. Страницы, экстенты. Создание базы данных. Файловые группы. Создание таблиц в файловой группе.**

**Структура файла базы данных:** страница 8K, 1MB = 128 страниц, 4MB = 512 страниц

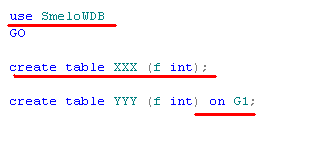
1. **Типы страницы:** данные, индексы, текст/изображение, карты распределения экстентов, свободное место на страницах, карта распределения индекса, схема массовых изменений, схема разностных изменений.
2. **Экстенты:** 8непрерывных страниц; однородные, смешанные.
3. **Создание базы данных**

****

1. **Назначение файловой группы по умолчанию.** До установки по умолчанию считается файловая группа PRIMARY

**11**

1. **Создать таблицу в файловой группе**

****

Из объектов баз данных и файлов можно формировать файловые группы, используемые для решения задач распределения и административного управления. Существует два типа файловых групп.

Первичные файловые группы

В первичную файловую группу входит первичный файл данных и все прочие файлы, не назначенные специально в другие файловые группы. Все страницы для системных таблиц размещаются в первичной файловой группе.

Пользовательские файловые группы

К пользовательским файловым группам относятся все файловые группы, которые задаются с помощью ключевого слова FILEGROUP в инструкции CREATE DATABASE или ALTER DATABASE.

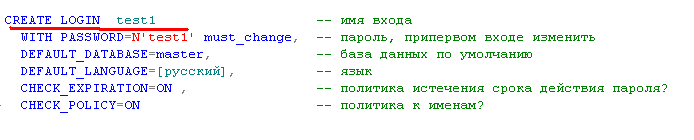
Файлы журналов не могут входить в состав файловых групп. Управление пространством журнала отделено от управления пространством данных.

Файл не может входить в состав нескольких файловых групп. Таблицы, индексы и данные больших объектов могут быть ассоциированы с указанной файловой группой. В этом случае все их страницы будут размещены внутри файловой группы; либо таблицы и индексы могут быть секционированы. Данные секционированных таблиц и индексов разделяются на блоки, каждый из которых может быть помещен в отдельную файловую группу базы данных.

В каждой базе данных одна файловая группа назначается файловой группой по умолчанию. Если при создании таблицы или индекса файловая группа не указывается, предполагается, что все страницы будут распределяться из файловой группы по умолчанию. В каждый момент времени лишь одна файловая группа может быть файловой группой по умолчанию. Элементы фиксированной роли базы данных **db\_owner** могут присваивать той или иной файловой группе статус файловой группы по умолчанию. Если файловая группа по умолчанию не указана, в качестве таковой выступает первичная файловая группа.

1. **Аутентификация в MSSQL. Создание имени входа (объект login). Создание пользователя (объект user). Понятие роли в MSSQL.**

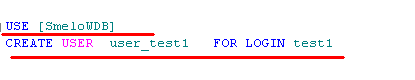
**Создание имени входа:** имя входа SQL Server, аутентификация SQL Server

****

**ALTER DISABLE/ENABLE**

**23**

**CREATE USER**

****

login — имя учетной записи, которую необходимо связать с именем пользователя базы данных;

user — имя пользователя базы данных, с которым ассоциируется данная учетная запись (в базе данных заранее не должно существовать пользователя с указанным именем);

role — этот параметр определяет роль, в которую данный пользователь будет включен (подробнее о ролях будет рассказано позже).

Роль — это мощный инструмент, добавленный в SQL Server 7.0, чтобы заменить группы, которые использовались в предыдущих версиях. Роль позволяет объединять пользователей, выполняющих одинаковые функции, для упрощения администрирования системы безопасности MS SQL Server 2000.

аутентификация -- средство защиты, определяющее подлинность пользователя и законность его работы

Строго говоря, этот процесс состоит из двух частей – идентификации и аутентификации. Идентификация – это предъявление пользователем какого-то уникального, присущего только ему признака-идентификатора. Это может быть пароль, какая-то биометрическая информация, например отпечаток пальца, персональный электронный ключ или смарт-карта и т.д. Аутентификация – это процедура, проверяющая, имеет ли пользователь с предъявленным идентификатором право на доступ к ресурсу.

1. **Аутентификация:**

- Windows-аутентификация;

- аутентификация SQL-сервера;

- смешанная аутентификация.

1. **Имена входа (учетные записи):**

**- имя входа SQL Server**;

**-** имя входа Windows;

**-** сертифицированное имя входа;

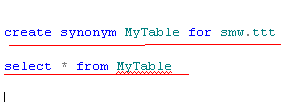
**-** имена входа для симметричного ключа.

1. **Роли:** стандартные и пользовательские, уровня сервера и уровня базы данных.
2. **Понятие схемы. Создание схемы (объект schema)**
3. **Схема:** подмножество объектов базы данных, рассматриваемых как единое целое в смысле прав и владения.
4. **Схема:** для каждого пользователя определяется своя схема по умолчанию.
5. **Схема:** каждый объект (кроме объектов безопасности) относится к какой–то схеме.
6. **Схема:** полное имя объекта базы данных ***имя\_схемы.имя\_объекта***.
7. **Схема:** объектами базы данных владеет схема, а не пользователь (начиная с 2005).
8. **Схема:** с одной и той же схемой могут работать несколько пользователей.
9. **Схема:** один пользователь может работать с несколькими схемами.
10. **Схема:** пользователь может создать свою схему.

**34**

**35**

1. **Понятие синонима. Создание синонима (объект synonym).**
2. **Синонимы (2005):** объект базы данных, представляющий собой псевдоним для других объектов.

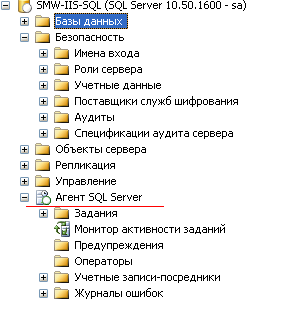
****

**44**

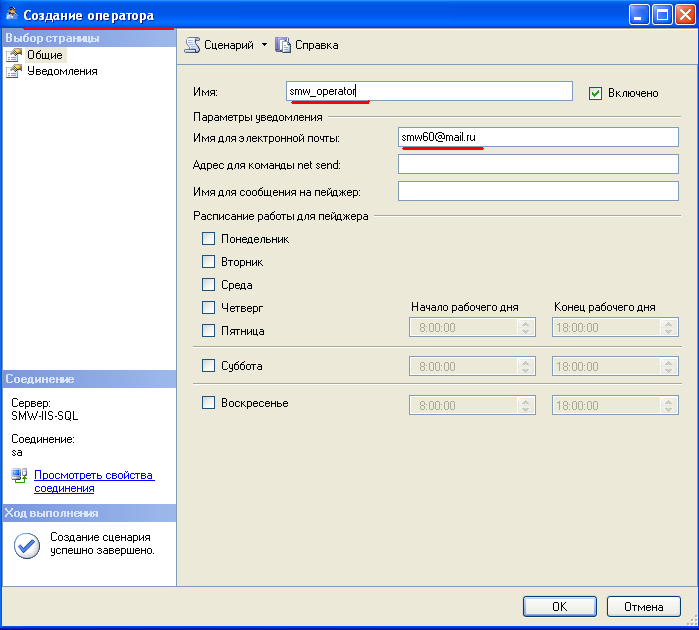
1. **SQL Server Agent. Назначение. Настройка**

**SSA:** специальная программа, предназначенная для автоматизации процесса администрирования базы данных.

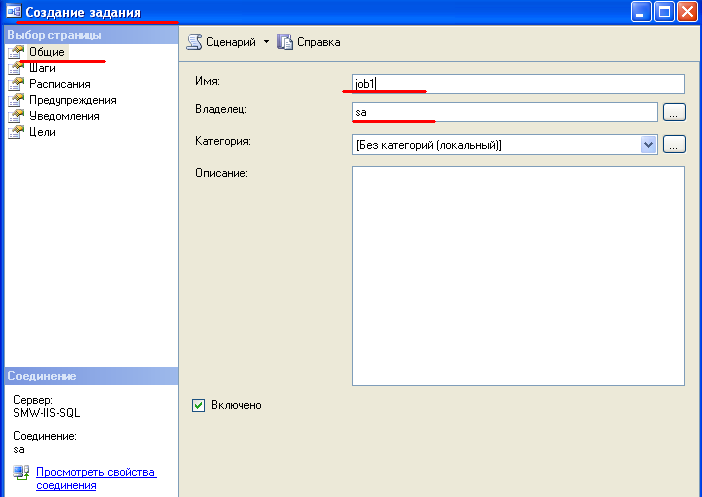
**SSA job:** SSA**-**задача, которую можно определить один раз и выполнять по расписанию.

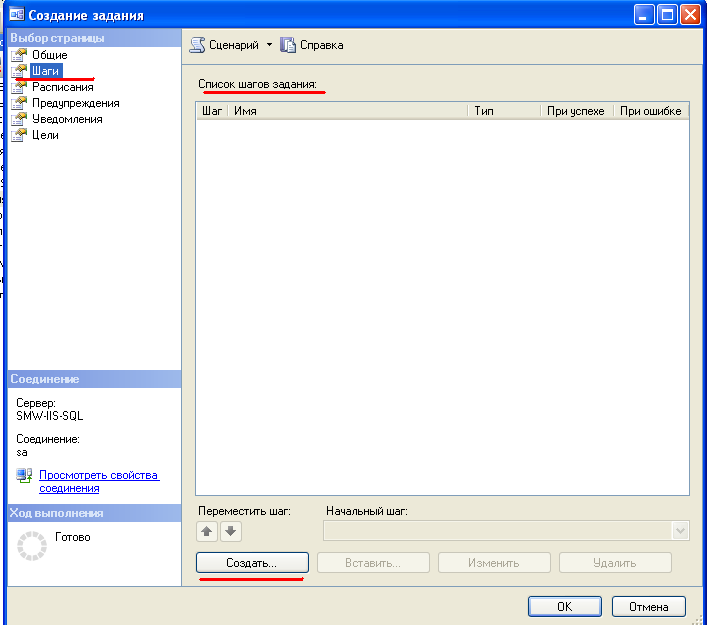
****

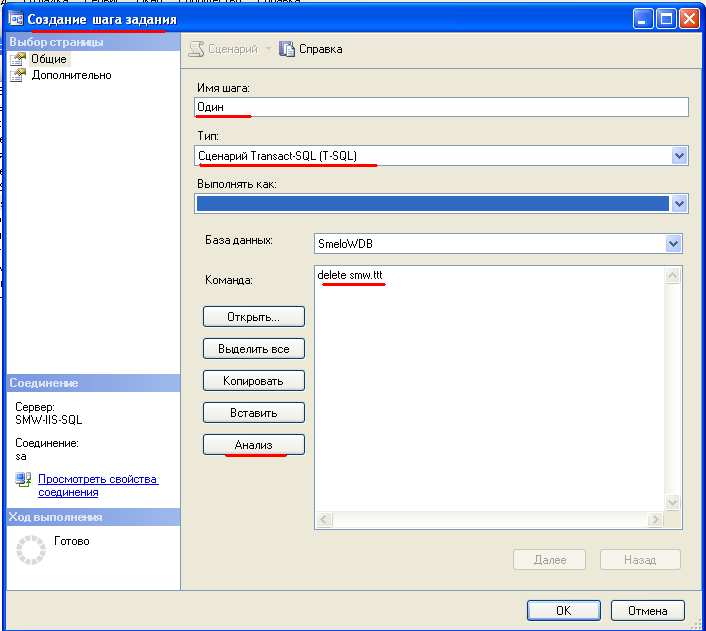
**SSA оператор**: объект безопасности от лица, которого выполняются задания.

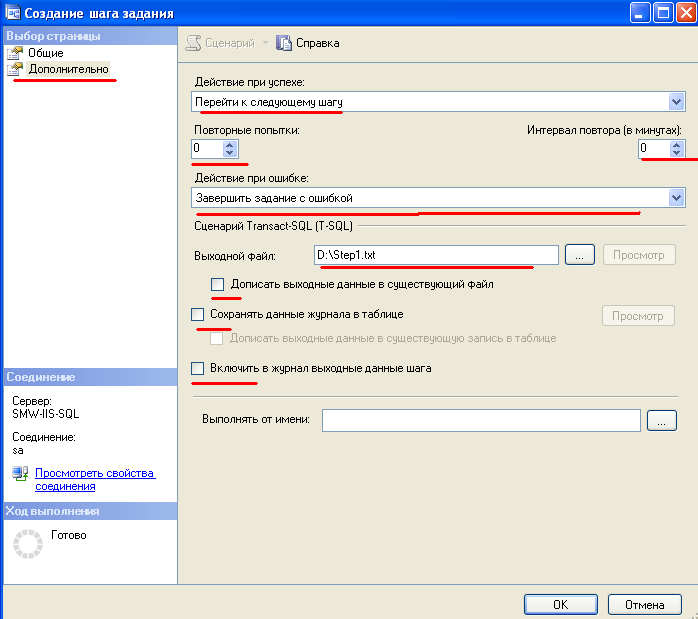


1. **Создание SSA job:**

****

****

****

****

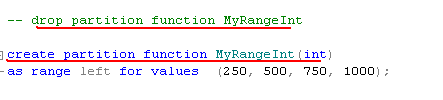
1. **Секционирование таблиц. Назначение и применение. Функция секционирования. Схема функционирования (объект schema). Создание секционированной таблицы**
2. **Секционирование:** метод хранения объектов базы данных (таблиц и индексов), при котором данные разбиваются на блоки (***секции***) и распределены между различными файловыми группами.
3. **Секционирование**: поддерживается не всеми редакциями Microsoft SQL Server 2008, а только Enterprise Edition, Developer Edition.
4. **Секционирование**: в разных СУБД реализовано по-разному; в Oracle очень развита эта технология.
5. **Секционирование**: в Microsoft SQL Server 2008 все таблицы и индексы секционированы, по умолчанию таблица или индекс находятся в одной секции; ***секции – базовая структура данных*** (совместно со страницами и экстентами).
6. **Секционирование:** в Microsoft SQL Server только ***горизонтальное***.
7. **Секционирование**: выигрыш в скорости (секции на разные диски, эффективно совместно с RAID(зеркало, стриппинг)), проще управлять данными при копировании/восстановлении и переносе.
8. **Секционирование:** пример с базой данных студентов; данные для изменения, для обновления.
9. **Секционирование:** следует применять если:

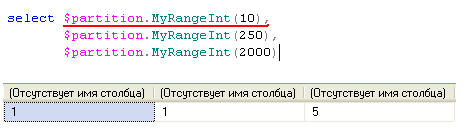
* таблица большая (много строк);
* таблица содержит множества строк, управляемые по-разному;
* таблицу можно разбить на несколько частей, и высока вероятность, что эти части будут использоваться параллельно;
* таблицу можно разбить на несколько частей и высока вероятность, что будет много запросов не выходящих за пределы этой секции.

1. **Секционирование:**  для повышения скорости join секционирование для двух таблиц должно быть согласованным.
2. **Секционирование:** для того чтобы секционировать таблицу необходимо:

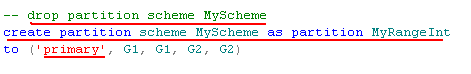
* создать ***функцию секционирования*;**
* создать ***схему секционирования***;
* создать таблицу или индекс со схемой секционирования;

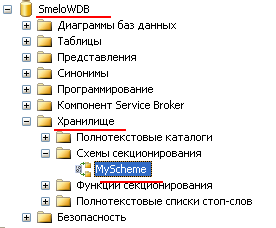
**Функция секционирования:**

****

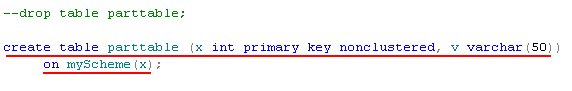
****

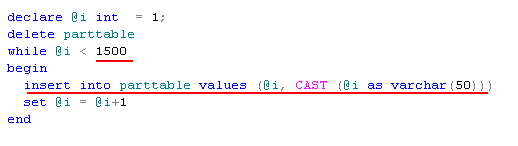
1. **Схема секционирования:**

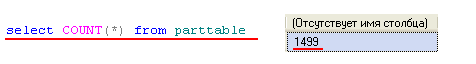
****

****

1. **Секционированная таблица:**

****

****

****

**Ограничения при секционировании:**  максимальное количество секций 999, нельзя секционировать по text, xml, timestamp, вычисляемым полям (можно по persistens)

Как только Вы создали функцию секционирования, Вы должны связать ее со схемой секционирования, для того чтобы адресовать секции определенным файловым группам. Когда Вы определяете схему секционирования убедитесь, что Вы определили файловые группы для КАЖДОЙ секции, даже если несколько секций будут располагаться в одной и той же файловой группе. Созданная ранее функция OrderDateRangePFN формирует 5 секций, последняя (пустая) будет располагаться в файловой группе PRIMARY. Нет никакой необходимости в особом размещении этой секции, поскольку она никогда не будет содержать данных.

1. **Программные интерфейсы: ODBC, JDBC, OLEDB, ADO, ADO.NET.**
2. **Программные интерфейсы**: ODBC
3. **Программные интерфейсы**: OLEDB
4. **Программные интерфейсы**: JDBC
5. **Программные интерфейсы**: ADO
6. **Программные интерфейсы:** ADO.NET
7. **Программные интерфейсы:** native-технологии
8. **ADO.NET:** основная модель доступа к данным для .NET-приложений.
9. **ADO.NET:** технология доступа к данным.
10. **ADO.NET:** набор классов в составе .NET Framework FCL.
11. **ADO.NET:** поддерживается VS 2008/2010.
12. **ADO.NET:** архитектура, модель поставщиков данных (***провайдеров***)

**ADO.NET: Data Provider** - набор классов ADO.NET, позволяющих получить доступ к базе определенного типа (MS SQL Server, Oracle, DB2, MySQL) данных (выполнять sql-команды, и извлекать данные).

**ADO.NET:** Data Providerвключает следующие классы:

**Connection**(XXXConnection, установка соединения с источником данных, реализует интерфейс **IDbConnection**);

**Command**(выполнение sql-команд и хранимых процедур);

**DataReader**(доступ к данным для чтения, извлеченным по запросу);

**DataAdapter**(наполнение **DataSet** информацией, выполнение изменений в базе данных, выполненных в DataSet)

**ODBC** ([англ.](http://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *Open Database Connectivity*) — это программный интерфейс ([API](http://ru.wikipedia.org/wiki/API)) доступа к [базам данных](http://ru.wikipedia.org/wiki/%D0%91%D0%B0%D0%B7%D0%B0_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85), разработанный фирмой [Microsoft](http://ru.wikipedia.org/wiki/Microsoft), в сотрудничестве с [Simba Technologies](http://ru.wikipedia.org/w/index.php?title=Simba_Technologies&action=edit&redlink=1) на основе спецификаций [Call Level Interface](http://ru.wikipedia.org/wiki/Call_Level_Interface) (CLI)

C помощью ODBC прикладные программисты могли разрабатывать приложения для использования одного интерфейса доступа к данным, не беспокоясь о тонкостях взаимодействия с несколькими источниками.

При применении ODBC требуется помнить, что данная технология доступа к данным не рассчитана на работу с большим числом клиентов[[*источник не указан 16 дней*]](http://ru.wikipedia.org/wiki/%D0%92%D0%B8%D0%BA%D0%B8%D0%BF%D0%B5%D0%B4%D0%B8%D1%8F:%D0%A1%D1%81%D1%8B%D0%BB%D0%BA%D0%B8_%D0%BD%D0%B0_%D0%B8%D1%81%D1%82%D0%BE%D1%87%D0%BD%D0%B8%D0%BA%D0%B8). В том случае, если необходимо, чтобы с БД одновременно работало много активных клиентов, требуется использовать SQL API или специальный интерфейс для взаимодействия с конкретной БД.

**JDBC** ([англ.](http://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) Java DataBase Connectivity — *соединение с базами данных на* [*Java*](http://ru.wikipedia.org/wiki/Java)) — платформенно-независимый промышленный стандарт взаимодействия Java-приложений с различными [СУБД](http://ru.wikipedia.org/wiki/%D0%A1%D0%A3%D0%91%D0%94), реализованный в виде пакета java.sql, входящего в состав [Java SE](http://ru.wikipedia.org/wiki/Java_SE).

JDBC основана на концепции так называемых драйверов, позволяющих получать соединение с базой данных по специально описанному [URL](http://ru.wikipedia.org/wiki/URL). Драйверы могут загружаться динамически (во время работы программы). Загрузившись, драйвер сам регистрирует себя и вызывается автоматически, когда программа требует [URL](http://ru.wikipedia.org/wiki/URL), содержащий протокол, за который драйвер отвечает.

**OLE DB** ([англ.](http://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *Object Linking and Embedding, Database*) — набор [интерфейсов](http://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81), основанных на [COM](http://ru.wikipedia.org/wiki/Component_Object_Model), которые позволяют [приложениям](http://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%B8%D0%BA%D0%BB%D0%B0%D0%B4%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5) обращаться к [данным](http://ru.wikipedia.org/wiki/%D0%94%D0%B0%D0%BD%D0%BD%D1%8B%D0%B5), хранимым в разных источниках информации или хранилищах данных с помощью унифицированного доступа.

**OLE DB** (связывание и внедрение объектов, базы данных, а иногда в литературе встречается как **OLEDB** или **OLE-DB**) является [API](http://ru.wikipedia.org/wiki/API) разработанной [Microsoft](http://ru.wikipedia.org/wiki/Microsoft) для доступа к различным типам данных, которые хранятся в единой форме. OLE DB отделяет хранилище данных из приложения, которое должно иметь доступ к нему через набор абстракций, которые включают DataSource, сессию, командную строку. Это было сделано потому, что различным приложениям необходим доступ к различным видам и источникам данных и не всегда нужно знать, как получить доступ к методологии функционирования конкретной технологии.

**ADO.NET** (ActiveX Data Objects .NET) — основная модель доступа к [данным](http://ru.wikipedia.org/wiki/%D0%94%D0%B0%D0%BD%D0%BD%D1%8B%D0%B5) для приложений, основанных на [Microsoft .NET](http://ru.wikipedia.org/wiki/Microsoft_.NET). Не является развитием более ранней технологии [ADO](http://ru.wikipedia.org/wiki/ADO). Скорее представляет собой совершенно самостоятельную технологию. Компоненты ADO.NET входят в поставку оболочки [.NET Framework](http://ru.wikipedia.org/wiki/.NET_Framework); таким образом, ADO.NET является одной из главных составных частей [.NET](http://ru.wikipedia.org/wiki/Microsoft_.NET).

1. **Полное резервное копирование и восстановление базы данных. Разностное резервное копирование и восстановление. Резервное копирование журналов транзакций и восстановление.**
2. **Предотвращение потери данных** – одна из самых важных задач стоящих перед администратором базы данных.
3. Потери данных возможны по следующим причинам:

* неисправность аппаратного обеспечения;
* ошибки в системном программном обеспечении;
* ошибки в пользовательском программном обеспечении;
* ошибки системного администратора;
* некорректная работа пользователей (DROP, ALTER, UPDATE, DELETE);
* аварийные ситуации (пожар, наводнение и пр.);
* вандализм, хакерские атаки, вирусы.

Администратор должен разработать ***стратегию восстановления*** данных. Стратегия должна быть: 1) спланирована; 2) протестирована.

В основе любой стратегии восстановления данных лежит создание резервных копий базы данных.

**Полное резервное копирование базы данных (ПРКБД)**

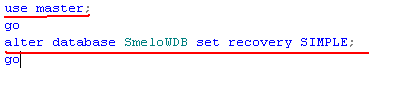
**ПРКБД:** самый простой и надежный способ сохранения данных; обычно делается периодически (например, каждую ночь).

**ПРКБД:** в результате полная копия базы данных; из полной копии восстанавливаются все файлы, данные, метаданные; данные в непротиворечивом состоянии.

**ПРКБД:** при копировании база данных находится в обычном рабочем режиме, пользователи могут выполнять транзакции, те транзакции, которые успели зафиксироваться будут отражены в копии, другие - подлежат откату.

Необходимо уведомить MS SQL Server какая ***модель восстановления*** данных будет использоваться.

**Модель восстановления SIPMLE**

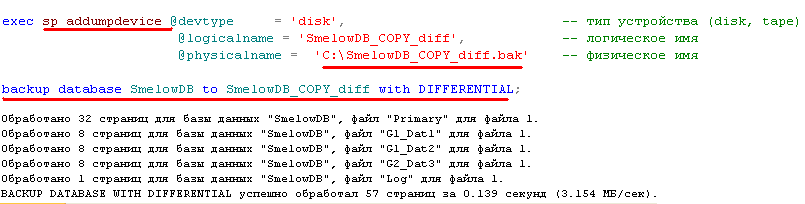
****

**Устройства для резервного копирования:** диск, стример и пр (должен видеть сервер).

**Недостатки полного резервного копирования**: восстанавливается только состояние БД соответствующее копии, часто не сделаешь (долго), мешает клиентам.

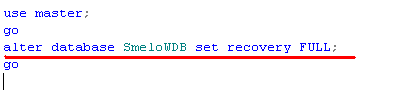
**РРКБД:** разностное резервное копирование – копирование только изменений прошедших с момента последней копии; чаще всего РРКБД сочетается с ПРКБД.

1. **Выполнение разностного резервного копирование**

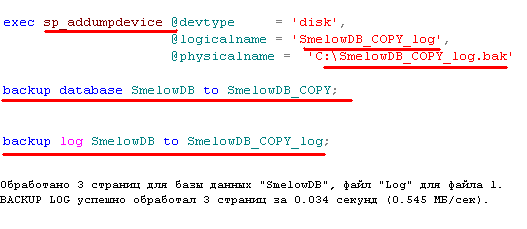
****

**Резервное копирование журнала тарнзакций**

1. **РКЖТБД**: резервное копирование журнала транзакций; необходима, как минимум одно ПРКБД; РКЖТБД может сочетаться с РРКБД.
2. **Упреждающее журналирование**
3. Требует модели восстановления **FULL** (полная)

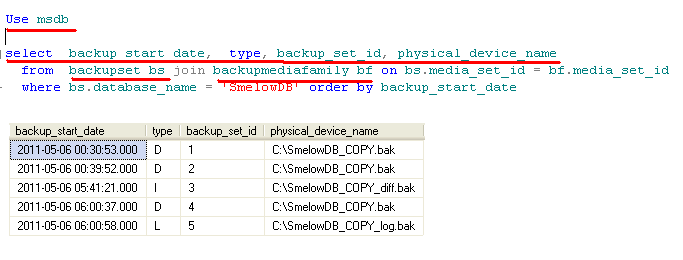
****

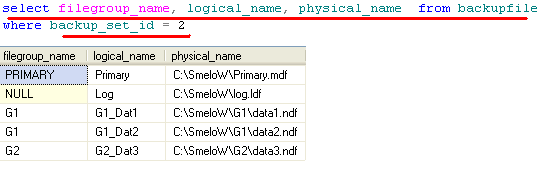
1. **Выполнение резервного копирование журнала транзакций.**

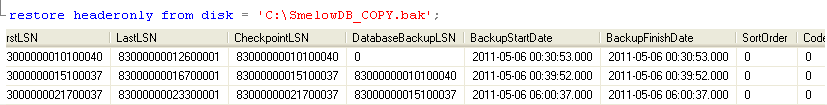
****

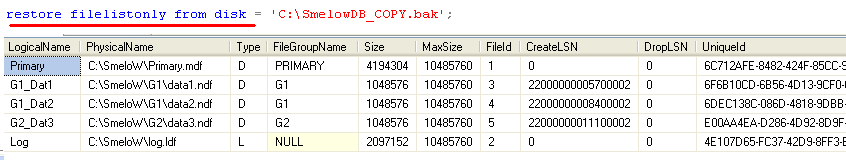
**Восстановление базы данных**

1. Получить информацию о резервном копировании

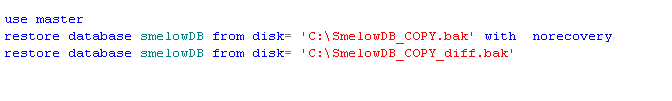




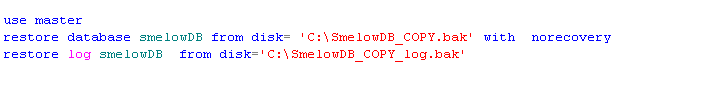


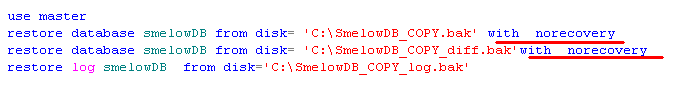


1. **Применение разностной копии**

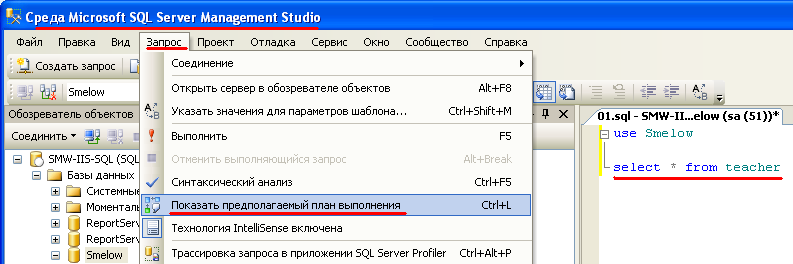
****

1. **Восстановление при стратегии FULL**

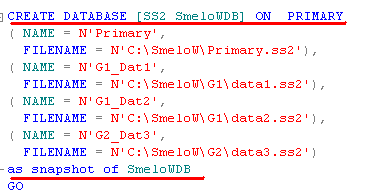
****

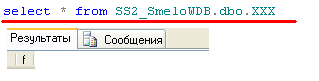
****

1. **Понятие плана запроса. Понятие стоимости запроса**.
2. **Планы запросов:** последовательность элементарных действий для исполнения запроса, планы вычисляются подсистемой MS SQL Server Query Optimizer (QO).
3. **Query Optimizer:** вычисляет несколько планов (не все) запроса на основе статистики, метаданных, информации о индексах и др.; на основе статистики предполагает стоимости запроса по различным планам и выбирает план с минимальными затратами на использование ресурсов, помещает его кэш; как правило, планы хранящиеся в кэше используются повторно.
4. **Стоимость запроса:** числовая величина, характеризующая степень использования ресурсов;
5. **Эффективность плана**: наличие индексов или сканирование; статистика о распределении данных (как правило формируются автоматически).
6. **Получение плана запроса:** Ctrl+L**,** значок – элементарная операция, стрелка – движение информации (чем толще, тем больше информации)

****

1. **Моментальный снимок базы данных (snapshot). Восстановление базы данных.**
2. **Snapshot:** моментальный снимок базы данных.
3. **Snapshot:** предназначен для хранения архивных данных
4. **Пример**

****



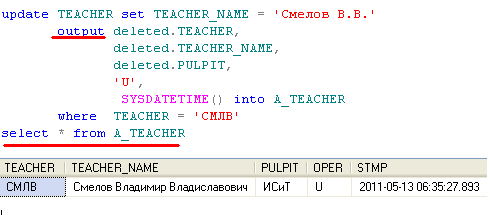
**Аудит**: мероприятия (операции) направленные на отслеживание изменений базы данных (кто, когда, как).

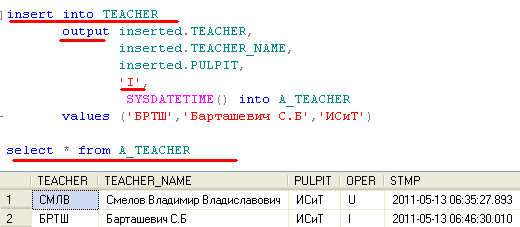
**Aудит**: 1) привилегий; 2) операций; 3) детальный аудит.

**Аудит**: в самой таблице (имитация delete, дата изменения, операция), с помощью триггеров, с помощью секции output.

1. **Применение секции output в операторах insert и update**.

**Секция output**

****

****

В предыдущих версиях MS SQL Server в триггерах использовались специальные таблицы inserted и deleted. Подобная возможность добавлена в MS SQL Server 2005 для операторов языка DML, выполняющих модификацию данных. Чтобы этим воспользоваться, необходимо в DML-выражение добавить ключевое слово OUTPUT.

Описание

OUTPUT может использоваться в операторах INSERT, UPDATE, DELETE:

* при использовании его в операторе INSERT создается временная таблица inserted;
* при использовании его в операторе DELETE создается временная таблица deleted;
* при использовании его в операторе UPDATE создаются временные таблицы inserted и deleted.

Новые операторы языка определения данных приведены в табл. 1.1, а их использование будет описано в следующих главах книги.

|  |  |  |  |
| --- | --- | --- | --- |
| **Тип объекта** | **Создание** | **Модификация** | **Удаление** |
| Синоним | CREATE SYNONYM |  | DROP SYNONYM |
| Сборка | CREATE ASSEMBLY | ALTER ASSEMBLY | DROP ASSEMBLY |
| Тип данных | CREATE TYPE |  | DROP TYPE |
| Пользователь | CREATE USER | ALTER USER | DROP USER |
| Роль | CREATE ROLE | ALTER ROLE | DROP ROLE |
| Схема | CREATE SCHEMA | ALTER SCHEMA | DROP SCHEMA |
| Контракт | CREATE CONTRACT | ALTER CONTRACT | DROP CONTRACT |
| Очередь | CREATE QUEUE | ALTER QUEUE | DROP QUEUE |
| Служба | CREATE SERVICE | ALTER SERVICE | DROP SERVICE |
| Маршрут | CREATE ROUTE | ALTER ROUTE | DROP ROUTE |
| Логин | CREATE LOGIN | ALTER LOGIN | DROP LOGIN |
| Точка доступа | CREATE ENDPOINT |  | DROP ENDPOINT |
| Сертификат | CREATE CERTIFICATE | ALTER CERTIFICATE | DROP CERTIFICATE |