**1. Основные понятия теории баз данных: база данных, система управления базами данных, основные требования к информации в БД,**

База данных – это совокупность взаимосвязанных данных. (лк Блиновой)

Субд - Программная реализация технологии хранения, извлечения, обновления и обработки данных в базе данных (лк).

Основные требования к информации в БД:

* Полезность - уменьшает информационную энтропию системы
* Полнота информации - информации должно быть достаточно, чтобы осуществить качественное управление
* Точность
* Достоверность - заведомо ошибочные данные не должны храниться в базе данных
* Непротиворечивость
* Актуальность

**2. Модели данных, основная терминология реляционных баз данных.**

* Иерархическая модель данных — это модель данных, где используется представление базы данных в виде древовидной (иерархической) структуры
* Сетевая модель данных - как древовидная, но предков может быть больше, чем один
* Реляционная модель данных - модель данных, основанная на отношениях (на теории множеств)

Основная терминология (взял из лк, вроде то)

* Relation – отношение
* Отношение может быть представлено в виде двумерной таблицы
* Реляционная база данных представляет собой набор взаимосвязанных таблиц
* Все объекты разделяются на типы
* Объекты одного и того же типа имеют свой набор атрибутов
* Один из атрибутов однозначно идентифицирует объект в таблице – первичный ключ

Определения:

* + - **домен**: множество;
  + - **таблица**: отношение;
  + - **атрибут**: имя столбца таблицы (имя домена);
  + - **заголовок таблицы**: множество всех атрибутов;
  + - **кортеж**: элемент отношения или строка таблицы

**3. Нормализация таблиц базы данных. Нормальные формы таблиц.**

Нормализация данных – процесс преобразования таблиц базы данных к нормальной форме (из лекций)

Чтобы таблица соответствовала 1-й нормальной форме (1NF), необходимо, чтобы все значения ее полей были неделимыми и не вычисляемыми, а все записи – уникальными (не должно быть полностью совпадающих строк).

Чтобы таблица соответствовала 2-й нормальной форме (2NF), необходимо, чтобы она находилась в 1-й нормальной форме и все не ключевые поля полностью зависели от ключевого.

Для перехода к 3-й нормальной форме (3NF), необходимо обеспечить, чтобы все таблицы находились во 2-й нормальной форме и все не ключевые поля в таблицах не зависели взаимно друг от друга.

**4. Язык SQL. Группы операторов SQL: DDL, DML, DCL, TCL.**

Язык **SQL** (**Structured Query Language**, язык структурированных запросов) – специализированный язык, предназначенный для написания запросов к реляционной БД

* Операторы **DDL** предназначены для создания, удаления и изменения объектов БД или сервера СУБД: **CREATE, DROP, ALTER**
* Операторы **DML** предназначены для работы со строками таблиц: **INSERT, DELETE, SELECT, UPDATE**
* Операторы TCL предназначены для управления транзакциями: **BEGIN TRAN, SAVE TRAN, COMMIT TRAN, ROLLBACK TRAN** (Не забудьте блин tran, пидоры)
* Операторы **DCL** предназначены для управления процессом авторизации: **GRANT, REVOKE, DENY** (Еще есть LOCK / UNLOCK , SET LOCK MODE, но она их не дает, так что хз, надо ли)

**5. Системные базы данных: master, msdb, model, tempdb.**

База данных **master** В этой базе данных хранятся все данные системного уровня для экземпляра SQL Server.

База данных **msdb** Используется агентом SQL Server для планирования предупреждений и задач.

Шаблон базы данных (**model)** Используется в качестве шаблона для всех баз данных, создаваемых в экземпляре SQL Server. Изменение размера, параметров сортировки, модели восстановления и других параметров базы данных model приводит к изменению соответствующих параметров всех баз данных, создаваемых после изменения.

База данных **tempdb** Рабочее пространство для временных объектов или взаимодействия результирующих наборов.

6. Структура файлов базы данных.

7. Создание файлов базы данных. Файловые группы. Создание таблиц в файловой группе. Дисковое хранение файлов базы данных. Страницы, экстенты.

|  |  |
| --- | --- |
| **Файл** | **Описание** |
| Первичная | Содержит сведения, необходимые для запуска базы данных, и ссылки на другие файлы в базе данных. В каждой базе данных имеется один первичный файл данных. Для имени первичного файла данных рекомендуется расширение ***MDF***. |
| Вторичная | Необязательные определяемые пользователем файлы данных. Данные могут быть распределены на несколько дисков, в этом случае каждый файл записывается на отдельный диск. Для имени вторичного файла данных рекомендуется расширение ***NDF***. |
| Журнал транзакций | Журнал содержит информацию для восстановления базы данных. Для каждой базы данных должен существовать хотя бы один файл журнала. Для файлов журнала транзакций рекомендуется расширение ***LDF***. |

Файловая группа — это способ организации файлов базы данных. По умолчанию для любой базы данных создается файловая группа PRIMARY, и все создаваемые файлы базы данных по умолчанию будут относиться именно к ней. При создании таблиц и индексов дисковая память для них автоматически отводится в файловой группе по умолчанию. Для размещения в другой файловой группе следует явно указывать ее имя в операторе CREATE, создающем таблицу или индекс.

create database DEZH\_UNIVER on primary

•Основной единицей хранилища данных является страница

•Размер страницы постоянен и составляет 8 Кб

•Каждая страница имеет заголовок размером в 96 байтов для системная информации

•Строки данных размещаются на странице сразу же после заголовка

Виды страниц:

•страницы данных

•страницы индексов

•Содержимое базы данных хранится в одном или нескольких файлах

•Каждый файл разделен на несколько страниц

•Каждую страницу таблицы или индекса можно однозначно идентифицировать, используя идентификатор базы данных, идентификатор файла базы данных, номер страницы

•Физическая единица дискового пространства, используемая для выделения памяти для таблиц и индексов, называется экстентом

•Размер экстента составляет 8 последовательно расположенных страниц или иначе 64 Кбайт

•Все страницы данных имеют фиксированный размер (8 Кб) и состоят из следующих частей:

• заголовка страницы (96 байтов)

• пространства для данных

• таблицы смещений строк

**СТРАНИЦЫ**

•Основной единицей хранилища данных является страница.

•Размер страницы постоянен и составляет 8 Кбайт.

•Каждая страница имеет заголовок 96 байтов, в котором хранится системная информация.

•Строки данных размещаются на странице сразу же после заголовка.

•Виды страниц:

•страницы данных;

•страницы индексов.

**ЭКСТЕНТЫ**

•Физическая единица дискового пространства, используемая для выделения памяти для таблиц и

•индексов, называется экстентом.

•Размер экстента составляет восемь последовательно расположенных страниц или 64 Кбайт.

•Существует два следующих типа экстентов:

• однородные экстенты;

• смешанные экстенты.

•Однородные экстенты содержат данные одной таблицы или индекса

•Смешанные экстенты могут содержать данные до восьми таблиц или индексов.

**8. Типы данных Microsoft SQL Server.**

* **BIT**: хранит значение 0 или 1. Фактически является аналогом булевого типа в языках программирования. Занимает 1 байт.
* **TINYINT**: хранит числа от 0 до 255. Занимает 1 байт. Хорошо подходит для хранения небольших чисел.
* **SMALLINT**: хранит числа от –32 768 до 32 767. Занимает 2 байта
* **INT**: хранит числа от –2 147 483 648 до 2 147 483 647. Занимает 4 байта. Наиболее используемый тип для хранения чисел.
* **BIGINT**: хранит очень большие числа от -9 223 372 036 854 775 808 до 9 223 372 036 854 775 807, которые занимают в памяти 8 байт.
* **DECIMAL**: хранит числа c фиксированной точностью. Занимает от 5 до 17 байт в зависимости от количества чисел после запятой.  
  Данный тип может принимать два параметра precision и scale: DECIMAL(precision, scale).  
  Параметр precision представляет максимальное количество цифр, которые может хранить число. Это значение должно находиться в диапазоне от 1 до 38. По умолчанию оно равно 18.  
  Параметр scale представляет максимальное количество цифр, которые может содержать число после запятой. Это значение должно находиться в диапазоне от 0 до значения параметра precision. По умолчанию оно равно 0.
* **NUMERIC**: данный тип аналогичен типу DECIMAL.
* **SMALLMONEY**: хранит дробные значения от -214 748.3648 до 214 748.3647. Предназначено для хранения денежных величин. Занимает 4 байта. Эквивалентен типу DECIMAL(10,4).
* **MONEY**: хранит дробные значения от -922 337 203 685 477.5808 до 922 337 203 685 477.5807. Представляет денежные величины и занимает 8 байт. Эквивалентен типу DECIMAL(19,4).
* **FLOAT**: хранит числа от –1.79E+308 до 1.79E+308. Занимает от 4 до 8 байт в зависимости от дробной части.  
  Может иметь форму опредеения в виде FLOAT(n), где n представляет число бит, которые используются для хранения десятичной части числа (мантиссы). По умолчанию n = 53.
* **REAL**: хранит числа от –340E+38 to 3.40E+38. Занимает 4 байта. Эквивалентен типу FLOAT(24).

#### **Типы данных, представляющие дату и время**

* **DATE**: хранит даты от 0001-01-01 (1 января 0001 года) до 9999-12-31 (31 декабря 9999 года). Занимает 3 байта.
* **TIME**: хранит время в диапазоне от 00:00:00.0000000 до 23:59:59.9999999. Занимает от 3 до 5 байт.  
  Может иметь форму TIME(n), где n представляет количество цифр от 0 до 7 в дробной части секунд.
* **DATETIME**: хранит даты и время от 01/01/1753 до 31/12/9999. Занимает 8 байт.
* **DATETIME2**: хранит даты и время в диапазоне от 01/01/0001 00:00:00.0000000 до 31/12/9999 23:59:59.9999999. Занимает от 6 до 8 байт в зависимости от точности времени.  
  Может иметь форму DATETIME2(n), где n представляет количество цифр от 0 до 7 в дробной части секунд.
* **SMALLDATETIME**: хранит даты и время в диапазоне от 01/01/1900 до 06/06/2079, то есть ближайшие даты. Занимает от 4 байта.
* **DATETIMEOFFSET**: хранит даты и время в диапазоне от 0001-01-01 до 9999-12-31. Сохраняет детальную информацию о времени с точностью до 100 наносекунд. Занимает 10 байт.

#### **Строковые типы данных**

* **CHAR**: хранит строку длиной от 1 до 8 000 символов. На каждый символ выделяет по 1 байту. Не подходит для многих языков, так как хранит символы не в кодировке Unicode.  
  Количество символов, которое может хранить столбец, передается в скобках. Например, для столбца с типом CHAR(10) будет выделено 10 байт. И если мы сохраним в столбце строку менее 10 символов, то она будет дополнена пробелами.
* **VARCHAR**: хранит строку. На каждый символ выделяется 1 байт. Можно указать конкретную длину для столбца - от 1 до 8 000 символов, например, VARCHAR(10). Если строка должна иметь больше 8000 символов, то задается размер MAX, а на хранение строки может выделяться до 2 Гб: VARCHAR(MAX).  
  Не подходит для многих языков, так как хранит символы не в кодировке Unicode.  
  В отличие от типа CHAR если в столбец с типом VARCHAR(10) будет сохранена строка в 5 символов, то в столце будет сохранено именно пять символов.
* **NCHAR**: хранит строку в кодировке Unicode длиной от 1 до 4 000 символов. На каждый символ выделяется 2 байта. Например, NCHAR(15)
* **NVARCHAR**: хранит строку в кодировке Unicode. На каждый символ выделяется 2 байта.Можно задать конкретный размер от 1 до 4 000 символов: . Если строка должна иметь больше 4000 символов, то задается размер MAX, а на хранение строки может выделяться до 2 Гб.

#### **Бинарные типы данных**

* **BINARY**: хранит бинарные данные в виде последовательности от 1 до 8 000 байт.
* **VARBINARY**: хранит бинарные данные в виде последовательности от 1 до 8 000 байт, либо до 2^31–1 байт при использовании значения MAX (VARBINARY(MAX)).

**9. Таблицы. Создание, изменение и удаление таблиц.**

Процесс проектирования логической схемы реляционной БД заключается в представлении данных в виде набора логически связанных таблиц. Таблицы нормализованы (см. вопрос Нормализация)

**1. Создание.** Таблица в БД создается с помощью DDL-оператора CREATE TABLE.

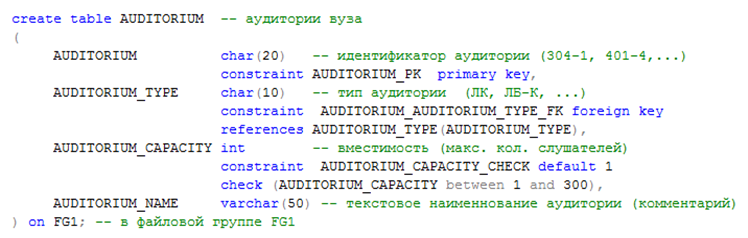
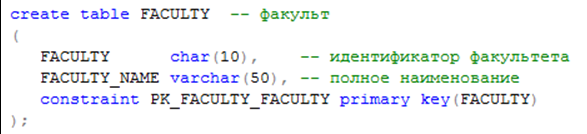
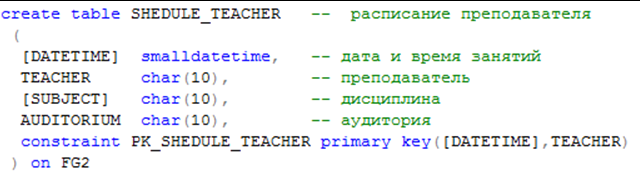


Таблица AUDITORIUM включает четыре столбца с именами: AUDITORIUM, AUDITORIUM\_TYPE, AUDITORIUM\_CAPACITY, AUDITORIUM\_NAME. Следует обратить внимание на то, что таблица будет размещена в файловой группе с именем FG1.

Синтаксис оператора CREATE TABLE допускает другую форму записи ограничения:



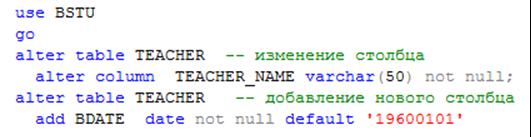
Пример создания таблицы с составным первичным ключом. В этом случае вторая форма записи ограничения является единственно возможной:



**2. Изменение.** Модификация таблицы — изменение ее структуры. Можно добавлять/удалять столбцы и ограничения, для числовых данных изменять точность/тип, а для символьных — их размерность. Вообще можно просто удалить таблицу (DROP TABLE) и создать ее (CREATE TABLE) с новой структурой. Но иногда это нерационально, например, когда уже распределены права доступа или когда в таблице дохуя данных (но не всякая модификация сохранит эти данные).

Перед модификацией таблицы целесообразно ознакомиться с ее структурой. Обычно для этого применяют системную процедуру **SP\_HELP**, при вызове которой указывают в качестве параметра имя исследуемой таблицы: *exec SP\_HELP TEACHER*

Изменяем таблицы при помощи Alter Table. В примере оператор ALTER изменяет существующий столбец **TEACHER\_NAME**: уменьшает максимальный размер столбца до 50 символов и добавляет ограничение NOT NULL. Второй оператор добавляет новый столбец с именем **BDATE** типа DATE, не допускающий значений NULL и с заданным значением по умолчанию.



**3. Удаление.** Таблицу можно удалить с помощью оператора DROP TABLE:

*drop table TEACHERS*

Проблема с удалением таблицы может быть в трех следующих случаях.

1. Пользователь не имеет достаточных прав на удаление таблицы.

2. Таблица заблокирована транзакцией другого сеанса. (нужно переждать, пока транзакция отработает)

3. На первичный ключ удаляемой таблицы ссылается внешний ключ другой таблицы. (нужно удалить ограничение целостности или эти самые таблицы)

**10. Таблицы. Локальные и глобальные временные таблицы.**

Основное отличие временных таблиц от постоянных в том, что они хранятся в системной БД **TEMPDB** и не могут иметь внешние ключи. Как правило, временные таблицы создаются для временного хранения результатов SELECT-запросов. При применении временных таблиц следует помнить, что БД **TEMPDB** снова создается при каждом перезапуске сервера, поэтому сохранить или восстановить временную таблицу в случае сбоя, приведшего к перезапуску сервера СУБД, невозможно.

Существует два вида временных таблиц: локальные и глобальные. Они отличаются друг от друга форматом имени, областью видимости и жизненным циклом.

**Локальные временные таблицы** имеют имена, начинающиеся с символа #, доступны только создавшему ее пользователю и могут быть удалены с помощью оператора DROP TABLE. Если пользователь временную таблицу не удалил сам, то она удалится автоматически при его отключении. На рис. 5.79 и 5.80 приведен пример, демонстрирующий создание, использование и удаление локальной временной таблицы с именем **#TEACHER**.

Следует обратить внимание: несмотря на то, что временная таблица #TEACHER создается в контексте БД BSTU (оператор USE), размещена она будет в системной базе данных TEMPDB.

Как правило, локальные временные таблицы применяются для временного хранения результатов трудоемких SELECT-запросов.

**Глобальные временные таблицы** имеют имена, начинающиеся с символа ##, доступны всем пользователям, подключенным к серверу, и могут быть удалены с помощью оператора DROP TABLE. Если глобальная временная таблица не удалена одним из пользователей, то она удалится автоматически при отключении всех пользователей, которые работали с этой таблицей. Если таблица использовалась только создавшим ее пользователем, то она будет удалена сразу после его отключения. Обычно глобальные временные таблицы применяются для обмена данными между несколькими сеансами.

11**. Ограничения целостности. Создание, изменение и удаление ограничений целостности.**

При создании таблиц используются различные ограничения. Ограничения, накладываемые на столбцы таблиц баз данных, предотвращают появление данных, не соответствующих предварительно заданным свойствам таблиц. Эти ограничения называются ограничениями целостности. В таблице приведены типы ограничений.

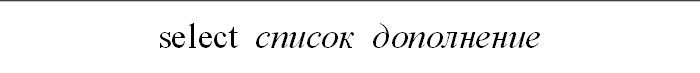
|  |  |
| --- | --- |
| Условное обозначение ограничения | Действие ограничения  целостности |
| **data type**  тип данных | Предотвращает появление в столбце значений, не соответствующих типу данных |
| **not null**  запрет значений **null** | Предотвращает появление в столбце значений null |
| **default**  знач. по умолчанию | Устанавливает значение в столбце по умолчанию при выполнении операции INSERT |
| **primary key**  первичный ключ | Предотвращает появление в столбце повторяющихся значений и пустого значения |
| **foreign key**  внешний ключ | Устанавливает связь между таблицей со столбцом, имеющим свойство **foreign key** и таблицей, имеющей столбец со свойством **primary key**; предотвращает не согласованные операции между этими таблицами |
| **unique**  уникальное значение | Не допускает пустые и повторяющиеся значения, не может быть использовано для связи с полем другой таблицы |
| **check**  проверка значений | Предотвращает появление в столбце значения, не удовлетворяющего логическому условию |

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

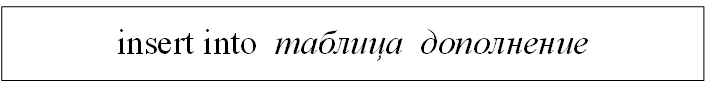
Операторы DML предназначены для работы с одним (наиболее важным) типом объектов БД – таблицами. DML включает четыре оператора: **SELECT**, **INSERT**, **DELETE**, **UPDATE** (рис. 3.13). Иногда к этой группе относят оператор **TRUNCATE** (операция мгновенного удаления всех строк в таблице).

Наиболее мощным DML-оператором является **SELECT**. Он позволяет выбрать множество строк из одной или нескольких таблиц. При успешном выполнении этого оператора формируется результирующий набор, представляющий собой множество однотипных (с одинаковыми столбцами) строк. В общем случае результирующий набор может содержать ни одной, одну или более строк.

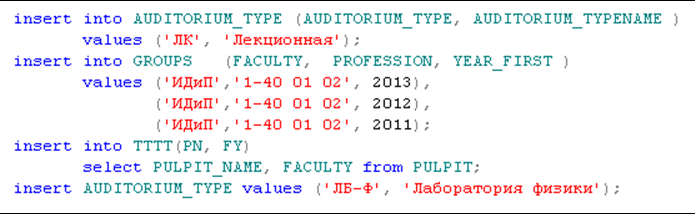
Любой оператор SELECT содержит список (на рис. 3.14 обозначен словом «список»), определяющий перечень столбцов (в общем случае и содержимое) результирующего набора. Дополнительная часть оператора (на рис. 3.14 обозначена словом «дополнение») описывает множество строк из одной или нескольких таблиц, служащее источником для формирования результирующего набора.



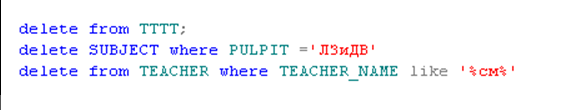
Добавить одну или несколько строк в существующую таблицу можно с помощью оператора **INSERT**. Ключевое слова **INTO** указывает на то, что далее следует имя таблицы, в которую будут добавляться строки. Далее структура оператора и примеры использования

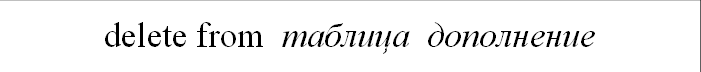


Можно вставлять данные только в определенные столбцы, можно во все столбцы и не указывать, куда вставляем. Можно вставлять одну строчку, а можно много. Можно вставлять при помощи select.



Для удаления строк из таблицы предназначен оператор **DELETE**. Можно удалить все, а можно указать условие в секции where





Для изменения строк таблицы предназначен оператор **UPDATE.** Структура и примеры использования:

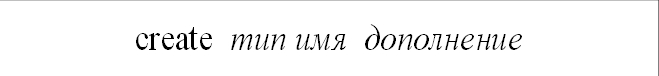
****

****

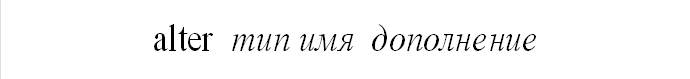
**13. Операторы DDL: Create, alter, drop.**

Операторы DDL предназначены для создания, удаления и изменения объектов БД (таблицы, функции, процедуры, индексы, курсоры и т.д.) или сервера СУБД. DDL включает три оператора: **CREATE**, **ALTER**, **DROP**

Оператор **CREATE** предназначен для создания объектов БД или сервера СУБД. Структура оператора:



Для модификации существующих объектов БД или сервера СУБД применяется оператор **ALTER**. Структура оператора:



Удалить существующий объект сервера или БД можно с помощью оператора **DROP**. Структура оператора :



**14. Оператор select. Секции from, where, group by, having, order by, distinct, top.**

Секции Select в порядке выполнения:

FROM,

WHERE

GROUP BY

HAVING

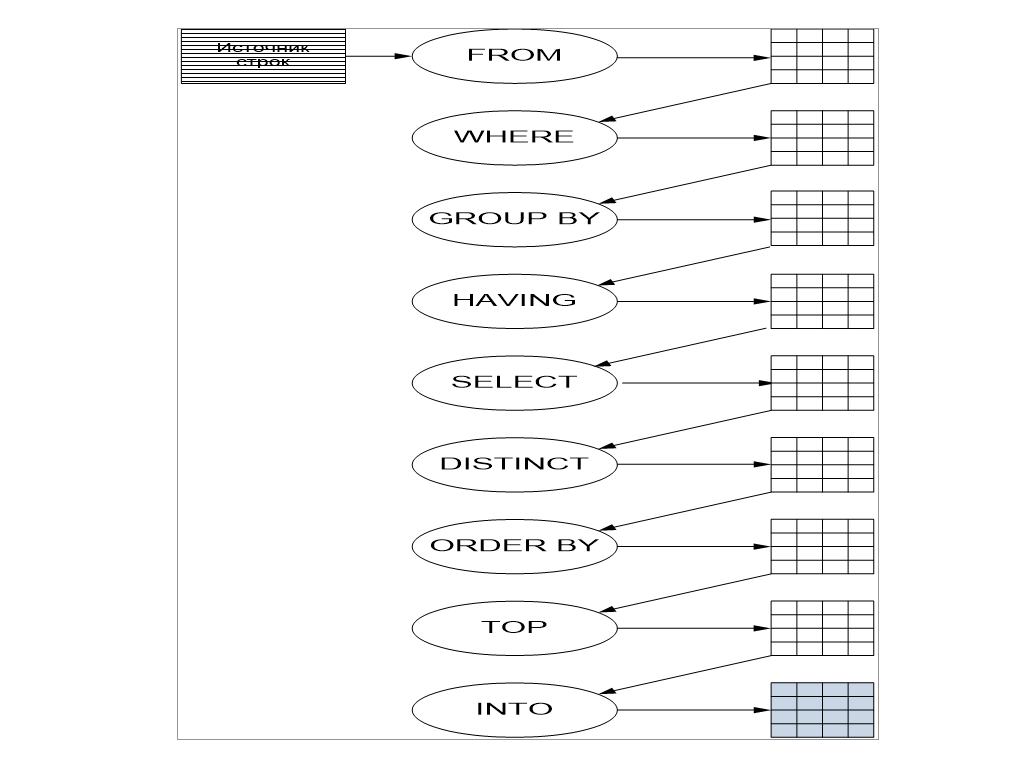
SELECT

DISTINCT

ORDER BY

TOP

INTO



* **SELECT определяет список возвращаемых столбцов** (как существующих, так и вычисляемых), их имена, ограничения на уникальность строк в возвращаемом наборе, ограничения на количество строк в возвращаемом наборе;
* **FROM задаёт табличное выражение, которое определяет базовый набор данных** для применения операций, определяемых в других предложениях оператора;
* **WHERE задает ограничение** на строки табличного выражения из предложения **FROM**;
* **GROUP BY** **объединяет ряды, имеющие одинаковое свойство** с применением агрегатных функций
* **HAVING** выбирает среди групп, определённых параметром GROUP BY
* **DISTINCT** позволяет не выводить повторяющиеся строки
* **ORDER BY** задает критерии сортировки строк; отсортированные строки передаются в точку вызова.
* **TOP** ограничивает количество результирующих строк
* **INTO** позволяет создать новую таблицу и заполнить ее результатом SQL запроса

**15. Подзапросы. Конструкции in, exists, all, any, NOT.**

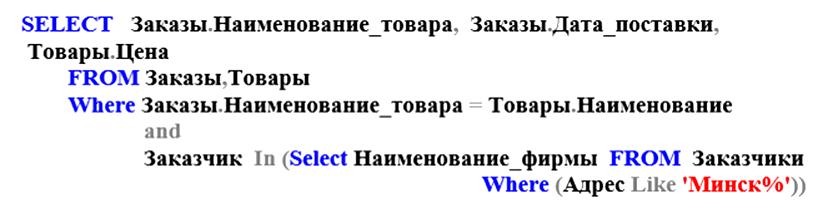
**Подзапрос** – это SELECT-запрос, который выполняется в рамках другого запроса. Подзапросы могут применяться в секции WHERE. Подзапросы бывают двух видов: коррелируемые и независимые.

*Коррелируемый* подзапрос зависит от внешнего запроса и выполняется для каждой строки результирующего набора.

*Независимый* подзапрос не зависит от внешнего запроса и выполняется только один раз, но результат его выполнения подставляется в каждую строку результирующего набора.

Операция **IN** формирует логическое значение «истина» в том случае, если значение, указанное слева от ключевого слова IN, равно хотя бы одному из значений списка, указанного справа. (В скобках может быть запрос, а может быть просто вручную заданное множество:

*in (‘man’, ‘women’))*



Операция EXISTS формирует значение «истина», если результирующий набор подзапроса содержит хотя бы одну строку, в противоположном случае - значение «ложь».

Операция >=ALL формирует истинное значение в том случае, если значение стоящее слева больше или равно каждому значению в списке, указанном справа.

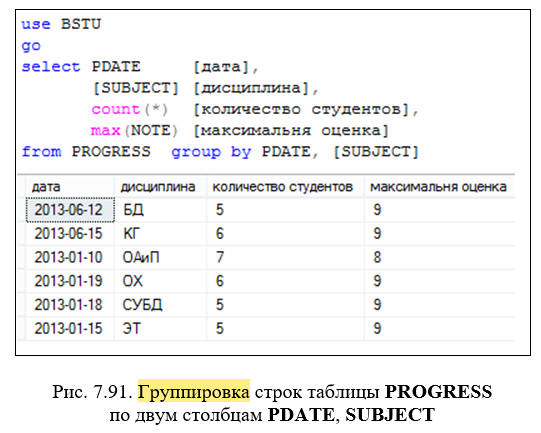
Операция >=ANY формирует истинное значение в том случае, если значение стоящее слева, больше или равно хотя бы одному значению в списке, указанном справа.

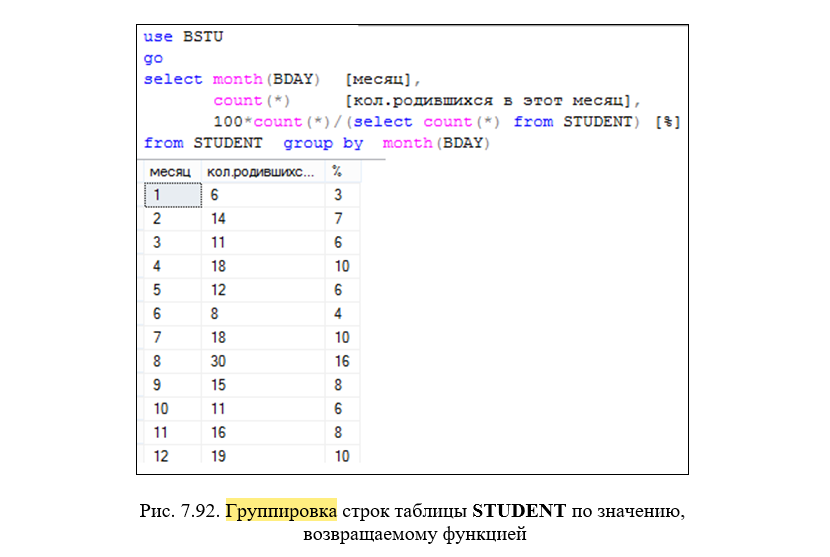
Not — инвертирует логическое значение

**16. Оператор select. Группировка данных.**

Основное назначение **группировки** с помощью секции GROUP BY – разбиение множества строк, сформированных секциями FROM и WHERE, на группы в соответствии со значениями в заданных столбцах, а также выполнение вычислений над группами строк с помощью наиболее часто используемых функций: **AVG** (вычисление среднего значения), **COUNT** (вычисление количества строк), **MAX** (вычисление максимального значения), **MIN** (вычисление минимального значения), **SUM** (вычисление суммы значений).

При использовании секции **GROUP BY** в SELECT-списке допускается указывать **только** те столбцы, по которым осуществляется группировка.





Следует помнить, что если в SELECT-запросе применяются секции WHERE и GROUP BY, первой исполняется секция WHERE. Затем выполняется группировка результата фильтрации в соответствии с выражением, указанным в секции GROUP BY

Если в SELECT-запросе указана опция TOP, ограничивающая количество строк в окончательном результирующем наборе, следует помнить, что действие опции осуществляется после группировки.

Также можно использовать rollup, cube (см. вопрос 18)

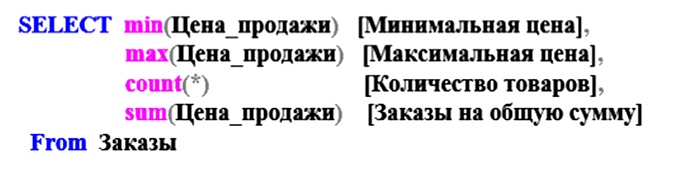
**17. Оператор select. Использование агрегатных функций.**

Агрегатные функции SQL нужны, чтобы получать результирующее значение из нескольких строк

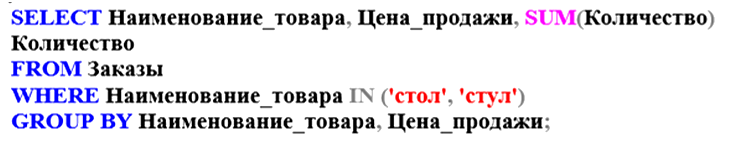
**Наиболее часто применяемые агрегатные функции**

|  |  |
| --- | --- |
| Наименование функции | Назначение |
| AVG | Вычисление среднего значения |
| COUNT | Вычисление количества строк |
| MAX | Вычисление максимального значения |
| MIN | Вычисление минимального значения |
| SUM | Вычисление суммы значений |

Можно использовать агрегатные функции просто, тогда мы будем получать значение, вычисленное для всех строк результирующего набора.



Можно использовать агрегатные функции вместе с Group By, тогда значение будет вычисляться отдельно для каждой группы строк.



**18. Оператор select. Группировка данных с использованием cube, rollup**

ROLLUP – оператор Transact-SQL, который формирует промежуточные итоги для каждого указанного элемента и общий итог.

SELECT otdel, god, sum(summa) as itog

FROM test\_table

GROUP BY ROLLUP(otdel, god)

CUBE — оператор Transact-SQL, который формирует результаты для всех возможных перекрестных вычислений.

SELECT otdel, god, sum(summa) as itog

FROM test\_table

GROUP BY CUBE(otdel, god)

**19. Представления. Создание, изменение и удаление представлений.**

Представление (View) – это объект базы данных, представляющий собой поименованный SELECT-запрос, который хранится в базе данных. Представление создается с помощью оператора CREATE, удаляется с помощью оператора DROP и изменяется с помощью ALTER.

Создание:

CREATE VIEW [Название] AS SELECT \* FROM TEST\_TABLE

Удаление:

DROP VIEW [Название]

Изменение:

ALTER VIEW [Название] AS SELECT \* FROM TEST\_TABLE2

Использование:

SELECT \* FROM [Название]

20**. Представления. Операции DML над представлениями. Использование представлений с указанием with check option.**

При создании представлений, позволяющих выполнять операции

INSERT, DELETE и UPDATE, базовый SELECT-запрос должен

удовлетворять правилам:

\* запрос не должен содержать секцию группировки GROUP BY;

\* запрос не должен применять агрегатные функции, опции DISTINCT и TOP, операторы UNION, INTERSECT и EXCEPT;

\* в SELECT-списке запроса не должно быть вычисляемых значений;

\* в секции FROM запроса должна указываться только одна таблица.

Чтобы операция вставки не могла осуществиться в том случае, когда

информация не удовлетворяет условию, записанному в секции Where, то

следует создавать представление с опцией WITH CHECK OPTION.

CREATE VIEW [Название] AS SELECT \* FROM Test\_table WHERE price > 200 with check option;

INSERT [Название] VALUES (‘хуй’, 201)

**21. Представления. Использование представлений с указанием schemabinding.**

Опция SCHEMABINDING устанавливает запрещение на операции с

таблицами и представлениями, которые могут привести к нарушению

работоспособности представления.

При использовании опции SCHEMABINDING требуется использовать в

SELECT-запросе для имен таблиц и представлений двухкомпонентный

формат (в имени присутствует наименование схемы).

Любая попытка модифицировать структуру представлений или таблиц, на которые ссылается созданное таким образом представление, будет неудачной. Чтобы такие таблицы или представления можно было модифицировать (инструкцией ALTER) или удалять (инструкцией DROP), нужно удалить это представление или убрать из него предложение SCHEMABINDING

**22. Функции преобразования типов данных cast и convert.**

CAST ([Аргумент] as [Тип\_Данных])

PRINT ‘Кол-во говна: ’ + CAST(12 as varchar(3))

CONVERT([Тип\_Данных], [Аргумент])

PRINT ‘Кол-во говна: ’ + CONVERT(varchar(3), 12)

convert более крутой! он позволяет еще стили накладывать после преобразования

**23. Операторы работы с множествами union (all), intersect, except.**

UNION - объединение без копий

UNION ALL - объединение с копиями

SELECT \* FROM Товары WHERE Название = ‘Коля’

UNION

SELECT \* FROM Товары WHERE Название = ‘Хуеля’

INTERSECT - пересечение двух исходных наборов

EXCEPT - разность двух исходных наборов

**24. Соединение таблиц. Внутреннее соединение inner join.**

## **INNER JOIN**

Самый простой вид соединения INNER JOIN – **внутреннее соединение**. Этот вид джойна выведет только те строки, если условие соединения выполняется (является истинным, т.е. TRUE). В запросах необязательно прописывать INNER – если написать только JOIN, то СУБД по умолчанию выполнить именно внутреннее соединение.

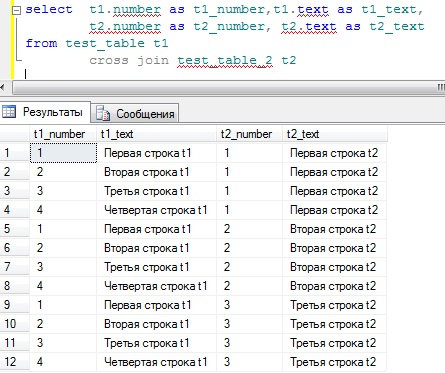
Пример

SELECT \* FROM student LEFT OUTER JOIN teacher ON teacher.Id = student.Teacher

**25. Ортогональное соединение cross join.**

## **Объединение SQL CROSS JOIN**

CROSS JOIN – это объединение SQL по которым каждая строка одной таблицы объединяется с каждой строкой другой таблицы.



26. Внешние соединения: left(right) outer join, full outer join.

Внешнее соединение OUTER JOIN двух таблиц формирует набор строк, состоящий из двух частей: результат внутреннего соединения двух таблиц и строки из одной из двух таблиц, которые не смогли соединиться. Значения в столбцах, соответствующих незаполненной (несоединенной) части строки будет NULL.

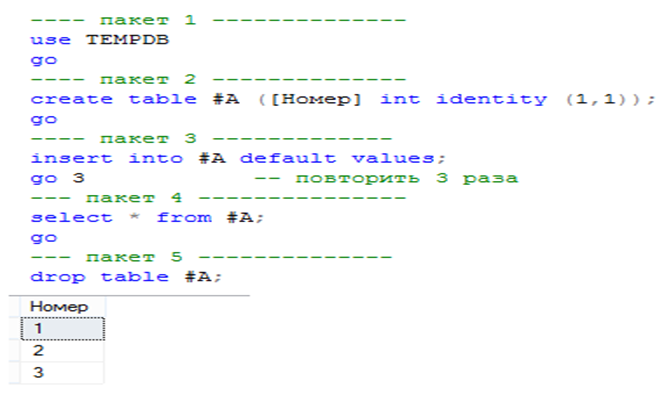
Имеется два вида внешнего соединения: LEFT OUTER JOIN – левое внешнее соединение и RIGHT OUTER JOIN – правое внешнее соединение.

Левое внешнее соединение включает в набор несоединенные строки таблицы, имя которой записано слева от ключевых слов LEFT OUTER JOIN, а правое внешнее соединение – несоединенные строки таблицы, имя которой записано справа от RIGHT OUTER JOIN.

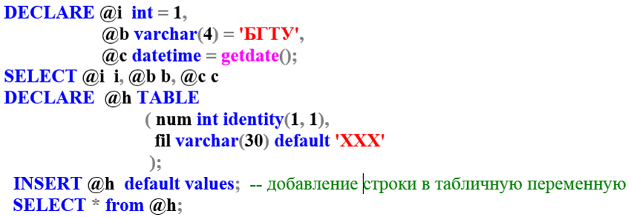
FULL OUTER JOIN определяет объединение правого и левого соединения

**27. Язык T-SQL. Пакеты. Объявление переменных.**

Пакет - это группа операторов T-SQL, которая обрабатывается сервером СУБД вместе



Для объявления переменных, используемых в программах, предназначен оператор DECLARE:



Имя переменной должно начинаться с символа @.

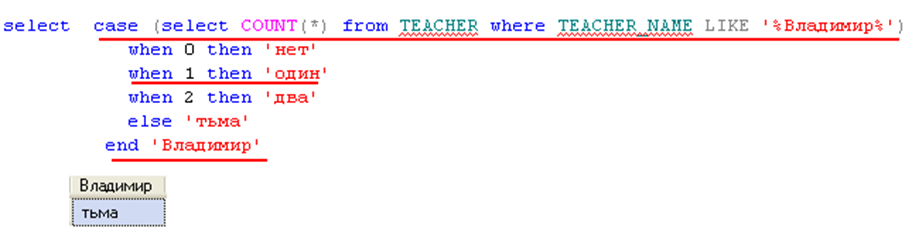
**28. Язык T-SQL. Операторы присвоения.**

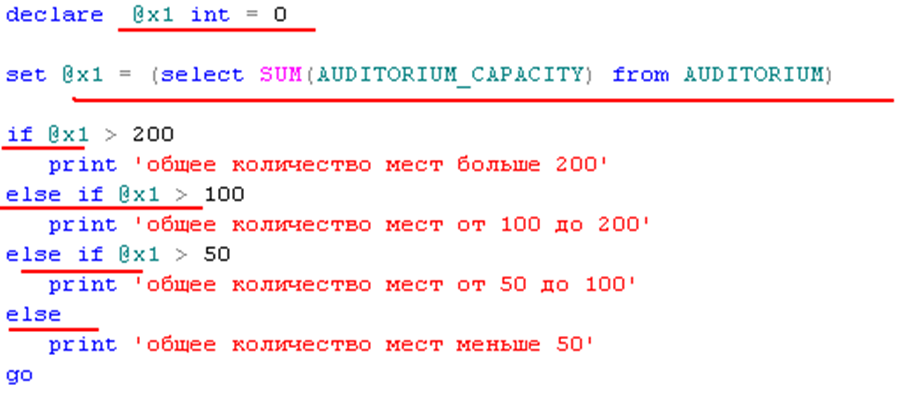
Переменную можно инициализировать в DECLARE. С помощью оператора SET можно переменной присвоить значение и выполнять вычисления. Оператор SELECT позволяет нескольким переменным присвоить значения.

**29. Язык T-SQL. Операторы print, if-else, case.**

С помощью оператора PRINT можно вывести строку в стандартный выходной поток.

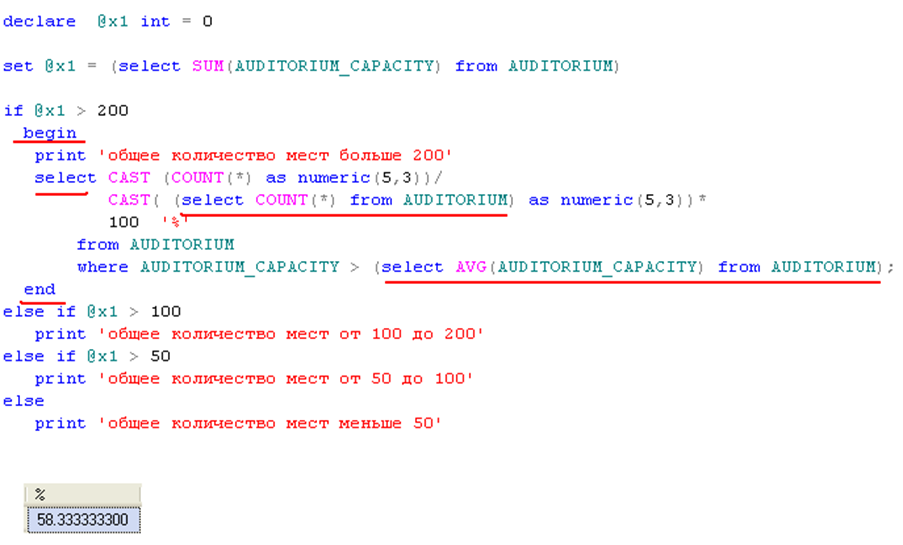
В выражении CASE каждое предложение WHEN содержит логическое выражение. Эти выражения проверяются на истинность сверху вниз, и при первом успешном сравнении формируется результирующее значение, указанное за ключевым словом THEN. В том случае, если ни одно из логических WHEN-выражений не принимает истинного значения, в качестве результата CASE формируется значение, указанное в предложении ELSE



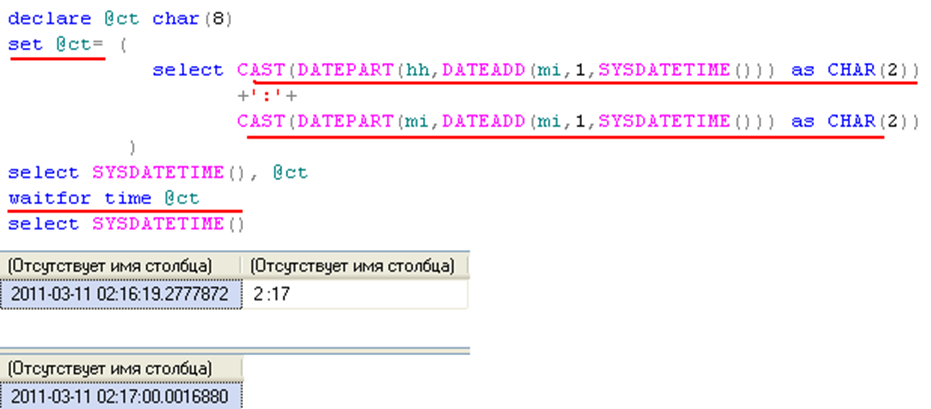


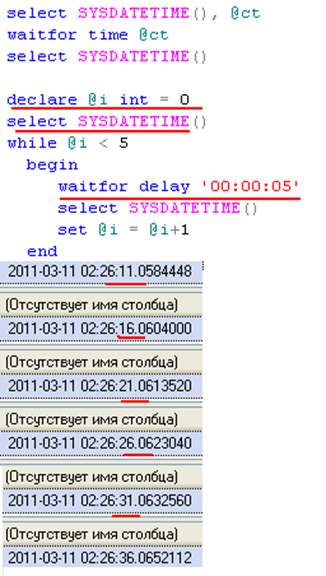
**30. Язык T-SQL. Операторы begin-end, waitfor и return.**

С помощью операторных скобок BEGIN END можно объединять операторы в группы.

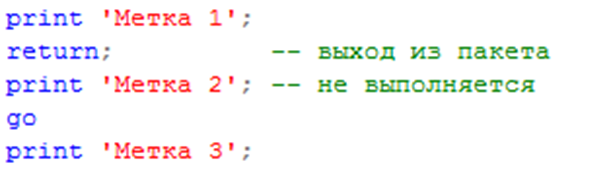


WAITFOR блокирует выполнение пакета, хранимой процедуры или транзакции до тех пор, пока не истечет заданное время или интервал времени, либо указанная инструкция не изменит или не вернет хотя бы одну строку.





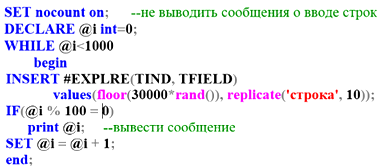
Оператор RETURN служит для немедленного завершения работы пакета



**31. Язык T-SQL. Оператор цикла while.**

Оператор WHILE предназначен для организации программного цикла. Принцип работы оператора WHILE такой же, как в большинстве алгоритмических языков (например C++, Java).

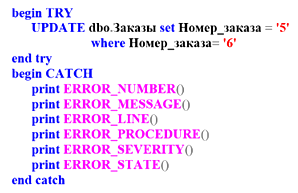
Оператор WHILE содержит две составляющие: логическое выражение и тело цикла. Логическое выражение определяет условие выполнения тела цикла. Тело цикла содержит один



или более операторов, которые выполняются в том случае и до тех пор, пока логическое выражение принимает значение «истина».

**32. Язык T-SQL. Обработка ошибок в конструкциях try-catch. Функция RAISERROR.**

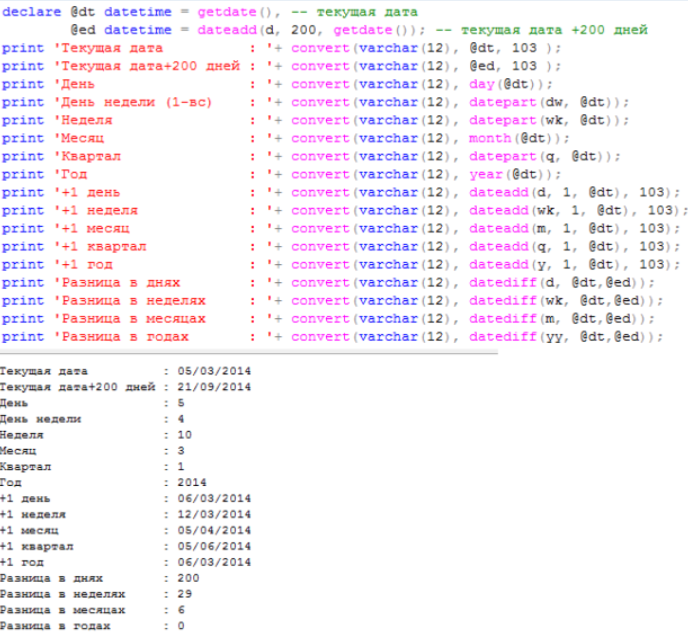
Для обработки ошибок выполнения в сценарии T-SQL предусмотрена конструкция, состоящая их двух блоков: TRY и CATCH. При этом блок TRY содержит код T-SQL, в котором могут возникнуть ошибки (говорят – охраняемый код), а блок CATCH – код, предназначенный для обработки ошибок. Ошибка, возникающая в охраняемом коде, приводит к немедленной передаче управления в блок обработки ошибок.



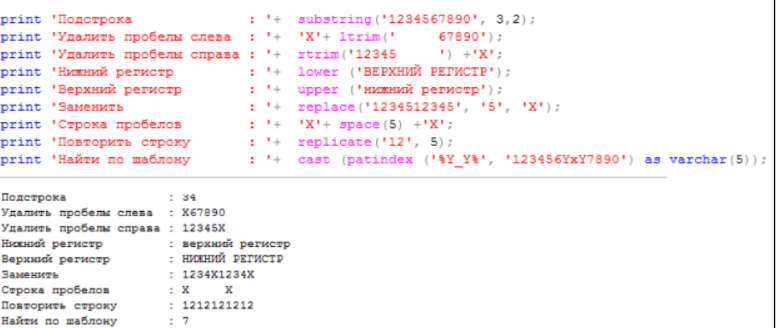
Разработчик сценария на языке T-SQL может сам сгенерировать ошибку с помощью специальной инструкции RAISERROR. В простейшем случае, при вызове инструкции можно передать три параметра: текстовое сообщение об ошибке, уровень серьезности ошибки и метку

33. Язык T-SQL. Встроенные функции работы с датами.

Для работы с датами и временем в языке T-SQL предусмотрен специальный набор встроенных функций. На рис. 13.8 представлен сценарий, демонстрирующий наиболее часто применяемые функции для работы с датами.



**34. Язык T-SQL. Встроенные функции работы со строками.**



35. Язык T-SQL. Встроенные функции работы с числовыми данными.

**ROUND**: округляет число. В качестве первого параметра передается число. Второй параметр указывает на длину. Если длина представляет положительное число, то оно указывает, до какой цифры после запятой идет округление. Если длина представляет отрицательное число, то оно указывает, до какой цифры с конца числа до запятой идет округление

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1  2 | SELECT ROUND(1342.345, 2) -- 1342.350  SELECT ROUND(1342.345, -2) -- 1300.000  **ISNUMERIC**: определяет, является ли значение числом. В качестве параметра функция принимает выражение. Если выражение является числом, то функция возвращает 1. Если не является, то возвращается 0.   |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | | 1  2 | SELECT ISNUMERIC(1342.345) -- 1  SELECT ISNUMERIC('SQL') -- 0  **SQUARE**: возводит число в квадрат.   |  |  | | --- | --- | | 1 | SELECT SQUARE(5) -- 25 |   **SQRT**: получает квадратный корень числа.   |  |  |  |  | | --- | --- | --- | --- | | 1 | SELECT SQRT(225) -- 15  **RAND**: генерирует случайное число с плавающей точкой в диапазоне от 0 до 1.   |  |  | | --- | --- | | 1  2 | SELECT RAND() -- 0.707365088352935  SELECT RAND() -- 0.173808327956812 | | | |

36**. Курсоры. Объявление курсора. Общая схема работы с курсором: declare, open, fetch, close, deallocate.**

Курсор – программная конструкция, которая служит для хранения результата запро­са и для обработки строк результирующего набора запись за записью

1.Курсор объявляется в операторе DECLARE.

2.Курсор открывается с помощью оператора OPEN.

3.С помощью оператора FETCH считывается одна или несколько строк результирующего набора, связанного с курсором SELECT-оператора, и обрабатывается нужным образом. Результат каждого считывания проверяется с помощью системной функции @@FETCH\_STATUS.

4.Курсор закрывается оператором CLOSE.

5.Если курсор глобальный, то он должен быть освобожден с использованием оператора DEALLOCATE.

declare specialtiesCursor cursor for

select SUBJECT.SUBJECT from PULPIT inner join SUBJECT on PULPIT.PULPIT = SUBJECT.PULPIT

where PULPIT.PULPIT like 'ИСиТ'

go

declare @a int, @b int

open specialitiesCursor

fetch first from specialitiesCursor into @a, @b

close specialitiesCursor

**37. Курсоры. Типы курсоров: global/local, static/dynamic.**

Курсор – программная конструкция, которая служит для хранения результата запро­са и для обработки строк результирующего набора запись за записью

Курсоры могут быть глобальными и локальными.

*Локальный курсор* может применяться в рамках одного пакета и ресурсы, выделенные ему при объявлении, освобождаются сразу после завершения работы пакета. Признаком того, что курсор является локальным, служит атрибут LOCAL, указанный при объявлении курсора

declare specialtiesCursor cursor local for....

*Глобальный* курсор может быть объявлен, открыт и использован в разных пакетах. Выделенные ему при объявлении ресурсы освобождаются только после выполнения оператора DEALLOCATE или при завершении сеанса пользователя. Для того чтобы объявить глобальный курсор, следует применить атрибут GLOBAL

declare specialtiesCursor cursor global for....

Курсоры могут быть статическими и динамическими.

•**Статический курсор –** данные выбраны один раз и произошедшие изменения не видны

•**Динамический курсор –** изменения данных отображаются в динамике

При объявлении *статического* курсора должен указываться атрибут STATIC. Открытие статического курсора приводит к выгрузке результирующего набора строк в динамически созданную временную таблицу системной БД **TEMPDB**, и все дальнейшие курсорные операции осуществляются с этой таблицей. После открытия курсора изменения в исходных таблицах, которые осуществляются в рамках этого или других сеансов, не будут отражаться в курсорном результирующем наборе.

**38. Курсоры. Опция scroll. Способы позиционирования в курсоре: relative/absolute, next/prior, first/last.**

Курсор – программная конструкция, которая служит для хранения результата запро­са и для обработки строк результирующего набора запись за записью

По умолчанию для курсора установлен атрибут SCROLL, позволяющий применять оператор FETCH с дополнительными опциями позиционирования.

**DECLARE @tc int, @rn char(50);**

**DECLARE Primer1 cursor local dynamic SCROLL**

**for SELECT row\_number() over (order by Наименование\_товара) N,**

**Наименование\_товара FROM dbo.Заказы**

**where Заказчик = 'Луч'**

**OPEN Primer1;**

**FETCH Primer1 into @tc, @rn;**

**print 'следующая строка : ' + cast(@tc as varchar(3))+ rtrim(@rn);**

**FETCH LAST from Primer1 into @tc, @rn;**

**print 'последняя строка : ' + cast(@tc as varchar(3))+ rtrim(@rn);**

**. . . . . . . . . . . . . . . . . . . . . . . . .**

**CLOSE Primer1;**

Можно дописать этот пример, используя другие ключевые слова: FIRST (первая строка), NEXT (следующая строка за текущей), PRIOR (предыдущая строка от текущей), ABSOLUTE 3 (третья строка от начала), ABSOLUTE -3 (третья строка от конца), RELATIVE 5 (пятая строка вперед от текущей), RELATIVE -5 (пятая строка назад от текущей).

**39. Курсоры. Функция fetch\_status.**

Курсор – программная конструкция, которая служит для хранения результата запро­са и для обработки строк результирующего набора запись за записью

После выполнения FETCH проверяется значение функции **@@fetch\_status**. В зависимости от полученного результата цикл продолжается и считывается следующая строка, или цикл заканчивается.

**Коды возврата функции @@FETCH\_STATUS**

|  |  |
| --- | --- |
| Возвращаемое значение | Описание |
| 0 | Оператор FETCH выполнен успешно. Строка считана из результирующего набора |
| –1 | Достигнут конец результирующего набора. Строка не считывается |
| –2 | Выбранная строка отсутствует в БД |

**40. Курсоры. Применение секции where current of в операторах update, delete.**

Курсор – программная конструкция, которая служит для хранения результата запро­са и для обработки строк результирующего набора запись за записью

Курсоры с установленным свойством FOR UPDATE помимо чтения данных из строк с помощью оператора FETCH, могут эти строки изменять или удалять с помощью операторов UPDATE и DELETE, если в секции WHERE эти операторы используют операцию CURRENT OF, для которой указывается имя курсора. Такой формат операторов позволяет удалять или изменять строки в таблице, соответствующих *текущей* позиции курсора в результирующем наборе.

declare tmpcursor cursor local dynamic

for select val from #temp1 for update -- изменяем или удаляем строки

declare @str varchar(100) = ''

declare @line varchar(30) = ''

open tmpcursor

fetch tmpcursor into @line

set @str = @str + @line

while @@fetch\_status = 0

begin

fetch tmpcursor into @line

set @str = @str + @line

delete #TEMP1 where current of tmpcursor

end

close tmpcursor

print @str

select \* from #temp1

drop table #temp1

go

**41. Хранимые процедуры. Создание, изменение и удаление хранимых процедур. Вызов хранимых процедур. Передача параметров. Входные и выходные параметры.**

Хранимая процедура – это объект БД, представляющий собой поименованный код T-SQL. Как и любой объект БД, хранимая процедура может быть создана с помощью CREATE, изменена с помощью ALTER и удалена с помощью оператора DROP.

Хранимая процедура может принимать входные и формировать выходные параметры, а результатом ее выполнения может быть целочисленное значение, возвращаемое к точке вызова с помощью оператора RETURN один или более результирующих наборов, сформированных операторами SELECT, а также содержимое стандартного выходного потока, полученного при выполнении операторов PRINT.

Вызов процедуры осуществляется с помощью оператора EXECUTE. Кроме того, результирующий набор хранимой процедуры может быть использован в качестве исходного источника строк для оператора INSERT.

use ФАЙЛОВАЯ\_ГРУППА

go

create procedure PSUBJECT as

begin

declare @count int = (select count(\*) from SUBJECT)

select SUBJECT, SUBJECT\_NAME, PULPIT from SUBJECT

return @count

end

go

declare @count int

exec @count = PSUBJECT

print @count

42. **Функции, определенные пользователем. Создание, изменение и удаление функций. Виды функций. Обращение к функции. Передача параметров.**

Функция – это объект БД, представляющий собой поименованный код T-SQL. Для создания, удаления и изменения функций необходимо применять CREATE, DROP и ALTER соответственно. Отличие функций от хранимых процедур в ограничениях, накладываемых на код функции, в форме представления результата работы, а также в способе вызова.

В функции не допускается применение DDL-операторов, DML-операторов, изменяющих БД (INSERT, DELETE, UPDATE), конструкций TRY/CATCH, а также использование транзакций.

Если функция возвращает единственное значение (число, строка, дата, время и пр.), то она называется *скалярной*. Функция, возвращающая таблицу, называется *табличной*. В зависимости от структуры кода, различают *встроенные* функции и *многооператорные* табличные функции.

-- многооператорная функция

--CREATE FUNCTION имя (параметры)

--RETURNS имя\_переменной TABLE

-- (описание вида таблицы,

-- в которой будет представлен результат)

--AS

-- BEGIN

-- Выполнение любого количества операций

-- RETURN

-- END

43.  **План запроса. Этапы обработки select запроса. Понятие стоимости запроса. Понятия селективности и плотности. Оптимизация запросов.**

**Этапы выполнения select-запроса:**  
- Разбор  
 Синтаксический разбор текста запроса для проверки его на соответствие правилам языка.  
- Разрешение имен  
 Проверка наличия используемых в запросе объектов БД (таблиц, представлений, столбцов, функций и тд).  
- Оптимизация  
 Производится на Специальной компоненте сервера – оптимизатор​е. Основная задача оптимизатора – построение плана запроса​. **План запроса** представляет собой алгоритм выполнения SQL-запроса. Для каждого шага вычисляется стоимость – *величина, пропорциональная продолжительности выполнения шага​.*Суммарная стоимость шагов плана составляет стоимость всего запроса​. Задача – минимизация общей стоимости запроса.  
- Компиляция  
 Откомпилированный план запроса помещается в специальную область памяти, называемую библиотечным кэшем​. Кэш используется для ускорения выполнения будущих аналогичных запросов.  
- Выполнение  
 Откомпилированный план выполняется сервером СУБД.

**Селективность запроса** – соотношение количества строк, удовлетворяющих условию, к общему количеству строк в таблице​.  
 - Индекс успешно работает при <= 5%.​  
 - Не нужен индекс при 80% или более

**Плотность запроса** – количество возвращаемых строк запроса.

**саси гуд (с) блинова завтра денису**

**Оптимизация запроса:**  
 - анализ запроса​  
 - выбор индекса​  
 - выбор порядка выполнения операций соединения​  
 - выбор метода выполнения операций соединения

**44. Индексы. Назначение и применение индексов.**

**Индекс** представляет собой отдельную физическую структуру данных, которая позволяет получать быстрый доступ к одной или нескольким строкам данных. Индексы сохраняются в страницах индексов​. Для каждой индексируемой строки имеется элемент индекса, который сохраняется на странице индексов​. Каждый элемент индекса состоит из ключа индекса и указателя. Индексы создаются по сбалансированному дереву B+​. B+-дерево имеет древовидную структуру, в которой все листья находятся на расстоянии одинакового количества уровней от вершины дерева​. Это свойство поддерживается при добавлении или удалении данных в индексированном столбце. Индекс всегда связан с таблицей или с подмножеством столбцов таблицы.

**Свойства индексов:**  
- Индекс может иметь максимум 900 байтов и не более 16 столбцов​.  
- Разрешено максимум 249 некластеризованных индексов для таблицы.  
- В UNIQUE составном индексе - однозначная комбинация значений всех столбцов каждой строки​.  
- Если UNIQUE не указывается, то повторяющиеся значения разрешаются​.  
- Параметр NONCLUSTED по умолчанию

**45. Индексы. Виды индексов. Применение различных видов индексов.**

**Виды индексов:** - Кластеризованные и некластеризованные ​  
 - Уникальные и неуникальные​  
 - Простые и составные​  
 - XML-индексы​  
 - Пространственные индексы

**- Кластеризованные**  
 Определяет физический порядок данных в таблице. Может только один для одной таблицы​. Таблица перестраивается в порядке индекса​. Листья дерева индекса содержат страницы данных. Создается по умолчанию для каждой таблицы, для которой определен первичный ключ​. Он уникальный – в столбце, для которого определен кластеризованный индекс, каждое значение данных может встречаться только один раз​. Если кластеризованный индекс создается для столбца, содержащего повторяющиеся значения, СУБД принудительно добавляет четырехбайтовый идентификатор к строкам, содержащим дубликаты значений  
 **- Некластеризованные**  
 Физически находится отдельно от таблицы. страницы листьев состоят из ключей индекса и закладок​. Может быть несколько для одной таблицы. Не изменяет физическое упорядочивание строк таблицы. Если есть кластеризованный индекс, то закладка некластеризованного индекса показывает B+-дерево кластеризованного индекса таблицы​. Если нет кластеризованного индекса, закладка идентична RID — Row Identifier, состоящего из: (Адреса файла, в котором хранится таблица; Адреса физического блока (страницы), в котором хранится строка; Смещения строки в странице).   
 Поиск данных с использованием некластеризованного индекса в зависимости от типа таблицы:  
 -Куча — прохождение при поиске по структуре некластеризованного индекса, после чего строка извлекается, используя идентификатор строки.​  
 -Кластеризованная таблица — прохождение при поиске по структуре некластеризованного индекса, после чего следует прохождение по соответствующему кластеризованному индексу.

**Применение индексов:**  
 Индекс занимает определенный объем дискового пространства​. Индекс используется для выборки данных​. Для вставки и удаления данных необходимо обслуживания индекса​. Чем больше индексов имеет таблица, тем больше объем работы по их реорганизации​

**Правила:​**  
 - Выбирать индексы для частых запросов​  
 - Затем оценивать их использование​  
 - Не индексировать столбцы, по которым нет поиска​

**46. Индексы. Реорганизация, перестроение, включение и отключение индексов.**

Процесс образования неиспользуемых фрагментов памяти называется *фрагментацией*.

Для ликвидации фрагментации используются:

**REBUILD - пересоздание индекса​** При отключенном кластеризованном индексе таблицы данные этой таблицы будут недоступны, так как все страницы данных таблицы с кластеризованным индексом хранятся в его листьях дерева. ALL - все индексы таблицы **REORGANIZE - реорганизация страниц листьев индекса​** REORGANIZE задает реорганизацию страниц листьев индекса, чтобы физический порядок страниц совпадал с их логическим порядком — слева направо **DISABLE - отключение индекса** Отключенный индекс недоступен, пока он не будет снова включен

reorganize - быстро, но фрагментация убирается только на нижнем уровне

rebuild - затрагивает все узлы дерева, поэтому фрагментация = 0

**47. Триггеры. Типы триггеров. Создание и назначение after-триггера.**

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

Типы триггеров:  
- DML-триггеры (создаются для таблицы или представления​. Реагируют на события INSERT, DELETE, UPDATE):  
 - AFTER-триггеры  
 - INSTEAD OF-триггеры

- DDL-триггеры  
 - триггеры уровня сервера (ALLSERVER)​  
 Обрабатывают события сервера СУБД (Создание объектов сервера​, Изменение объектов сервера​, Удаление объектов сервера​, Подключение к серверу)  
 - триггеры уровня базы данных (DATABASE)  
 Обработка событий, происходящих в рамках базы данных​

Триггеры **AFTER** можно создавать только для базовых таблиц. Можно использовать для создания журнала аудита действий в таблицах базы данных​, реализации бизнес-логики​, принудительного обеспечения ссылочной целостности. AFTER-триггеры - триггеры уровня оператора​. Выполняются по одному разу для каждого оператора​. Выполняются после наступления события. AFTER-триггер вызывается после выполнения активизирующего его оператора. Если оператор нарушает ограничение целостности, то возникшая ошибка не допускает выполнения этого оператора и соответствующих триггеров

**Триггеры типа AFTER исполняются *после* выполнения оператора**

**48. Триггеры. Создание и назначение instead of-триггеров.**

Триггеры уровня оператора​. Выполняются по одному разу для каждого оператора​. **Выполняются вместо операции - сама операция не выполняется.** Всегда использует таблицы inserted и deleted​. Выполняется после создания таблиц inserted и deleted​. Выполняется перед выполнением проверки ограничений целостности или каких-либо других действий​. INSTEAD OF можно создавать для таблиц и для представлений - выполняется вместо выполнения любых действий с любой таблицей. Не могут вызываться рекурсивно (если в триггере сработает операция, снова вызвавшая работу триггера)​. Если образуется рекурсия вызовов триггеров, то будет сделана попытка выполнить оператор

**49. Триггеры. Использование таблиц inserted, deleted.**

Две специальные виртуальные таблицы​:  
- **deleted** — содержит копии строк, удаленных из таблицы​  
- **inserted** — содержит копии строк, вставленных в таблицу

Структура этих таблиц эквивалентна структуре таблицы, для которой определен триггер.  
 Таблица **deleted** – в инструкции CREATE TRIGGER указывается DELETE или UPDATE​.   
 Таблица **inserted** – в инструкции CREATE TRIGGER указывается INSERT или UPDATE.

**50. Транзакции. Явные и неявные транзакции.**

(опр1) ​Одна или несколько команд SQL, которые либо успешно выполняются как единое целое, либо отменяются как единое целое.  
(опр2) Логическая единица работы, обеспечивающая переход базы данных из одного согласованного состояния в другое согласованное состояние.

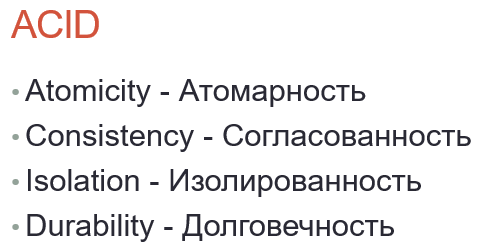
Бывают явные и неявные.

**Неявная** транзакция — задает любую отдельную инструкцию INSERT, UPDATE или DELETE как единицу транзакции.

**Явная** транзакция — группа инструкций, начало и конец которой обозначаются инструкциями:​  
 - BEGIN TRANSACTION​  
 - COMMIT​  
 - ROLLBACK

**51. Транзакции. Свойства ACID.**

**Транзакция** - это механизм базы данных, позволяющий таким образом объединять несколько операторов, изменяющих базу данных, чтобы при выполнении этой совокупности операторов они или все выполнились или все не выполнились.



Проще говоря, ACID - это основные свойства транзакции.

*Атомарность* (операторы изменения БД, включенные в транзакцию, либо выполнятся все, либо не выполнится ни один); *согласованность* (транзакция должна фиксировать новое согласованное состояние БД); *изолированность* (отсутствие взаимного влияния параллельных транзакций на результаты их выполнения); *долговечность* (изменения в БД, выполненные и зафиксированные транзакцией, могут быть отменены только с помощью новой транзакции).

**52. Транзакции. Уровни изолированности транзакций.**

Уровень изоляции задает степень защищенности данных в транзакции от возможности изменения другими транзакциями.

Уровни изоляции:

• READ UNCOMMITTED

-Не изолирует операции чтения других транзакций

-Транзакция не задает и не признает блокировок

-Допускает проблемы:

Грязное чтение

Неповторяемое чтение

Фантомное чтение

• READ COMMITTED

-Транзакция выполняет проверку только на наличие монопольной блокировки для данной строки

-Является уровнем изоляции по умолчанию

-Проблемы:

Неповторяемое чтение

Фантомное чтение

• REPEATABLE READ

-Устанавливает разделяемые блокировки на все считываемые данные и удерживает эти блокировки до тех пор, пока транзакция не будет подтверждена или отменена

-Не препятствует другим инструкциям вставлять новые строки

-Проблема:

Фантомное чтение

• SERIALIZABLE

-Устанавливает блокировку на всю область данных, считываемых соответствующей транзакцией

-Предотвращает вставку новых строк другой транзакцией до тех пор, пока первая транзакция не будет подтверждена или отменена

-Реализуется с использованием метода блокировки диапазона ключа

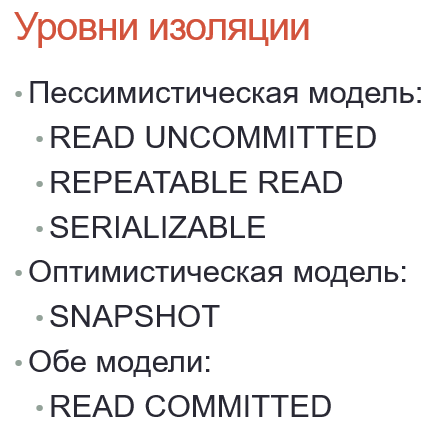
-Блокировка диапазона ключа блокирует элементы индексов

• SNAPSHOT

Уровень изоляции SNAPSHOT - отдельный уровень изоляции транзакций, который нужно явно указывать в коде. Этот уровень обеспечивать согласованность данных на уровне транзакции. Это значит, что запрос обращается к версии данных, которая была зафиксирована на момент начала транзакции.

При использовании уровня изоляции SNAPSHOT операции записи не блокируют друг друга, за исключением тех случаев, когда они меняют одни и те же строки. Это приводит либо к блокировке, либо к ошибке 3960.

*\*Я хуй знает про snapshot, так как этого не было в лабе. Информация выше взята с сайта.*

- на всякий пожарный

**53. Транзакции. Вложенные транзакции. Функция trancount.**

Транзакция, выполняющаяся в рамках другой транзакции, называется *вложенной*.

При работе с вложенными транзакциями нужно учитывать следующее:

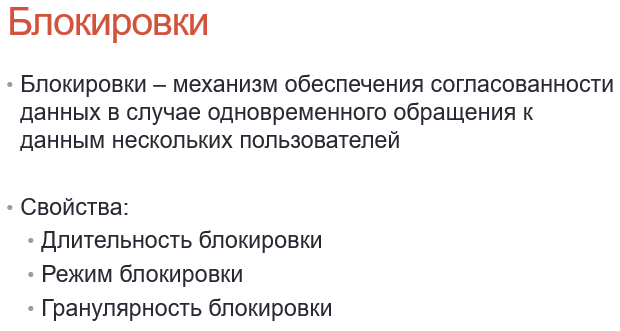
- оператор COMMIT вложенной транзакции действует только на внутренние операции вложенной транзакции;

- оператор ROLLBACK внешней транзакции отменяет зафиксированные операции внутренней транзакции;

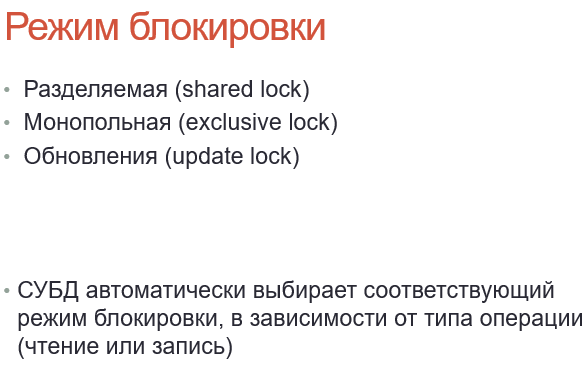
- оператор ROLLBACK вложенной транзакции действует на операции внешней и внутренней транзакции, а также завершает обе транзакции;

- уровень вложенности транзакции можно определить с помощью системной функции @@TRANCOUT.

**54. Транзакции. Блокировки. Эскалация блокировок. Взаимные блокировки.**



Длительность блокировки — это период времени, в течение которого ресурс удерживает определенную блокировку.



Разделяемая блокировка

•Разделяемая блокировка резервирует ресурс только для чтения

•Другие процессы не могут изменять заблокированный ресурс

•Может быть несколько разделяемых блокировок

Монопольная блокировка

•Монопольная блокировка резервирует страницу или строку для монопольного использования одной транзакции

•Применяется при INSERT, UPDATE и DELETE

•Монопольную блокировку нельзя установить, если на ресурс уже установлена какая-либо блокировка

Блокировка обновления

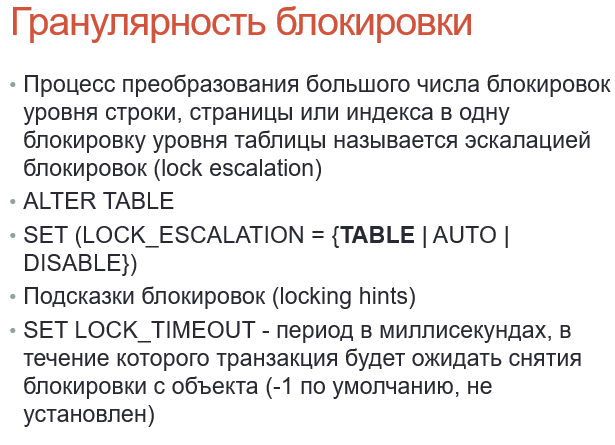
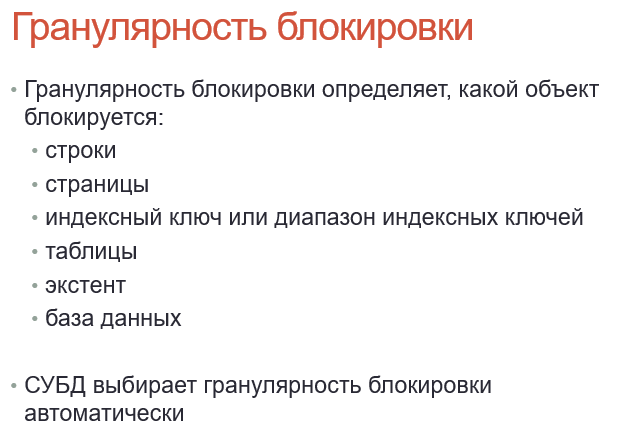
•Можно устанавливать на объекты с разделяемой блокировкой, накладывается еще одна разделяемая блокировка

•Нельзя устанавливать при наличии на нем другой блокировки обновления или монопольной блокировки

•При COMMIT транзакции обновления, блокировка обновления преобразовывается в монопольную блокировку

•У объекта может быть только одна блокировка обновления

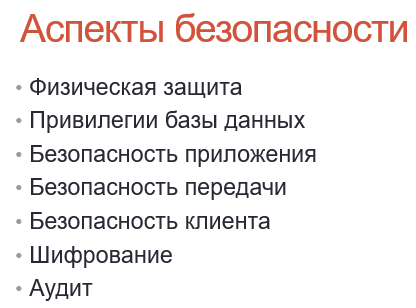
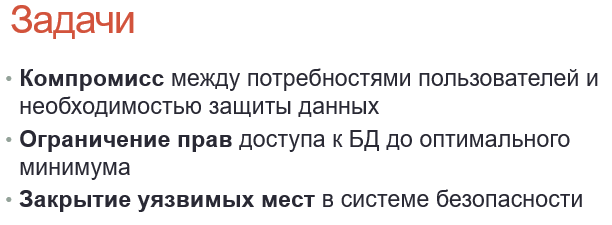
Эскалация блокировок – это процесс, при котором множество блокировок с маленькой гранулярностью, конвертируются в одну блокировку на более высоком уровне иерархии с большей *гранулярностью*.



Взаимоблокировка (deadlock) — это особая проблема одновременного конкурентного доступа, в которой две транзакции блокируют друг друга

55. Основы безопасности в SQL Server. Операторы DCL. Привилегии. Назначение и отъем привилегий.

Немного хуйни из начала презентации:



**Аутентификация** – определение, кто такой конкретный пользователь.

**Авторизация** – определение того, какие действия можно производить аутентифицированному пользователю.

К сути:

Data Control Language (DCL) – это группа операторов определения доступа к данным. Иными словами, это операторы для управления разрешениями, с помощью них мы можем разрешать или запрещать выполнение определенных операций над объектами базы данных.

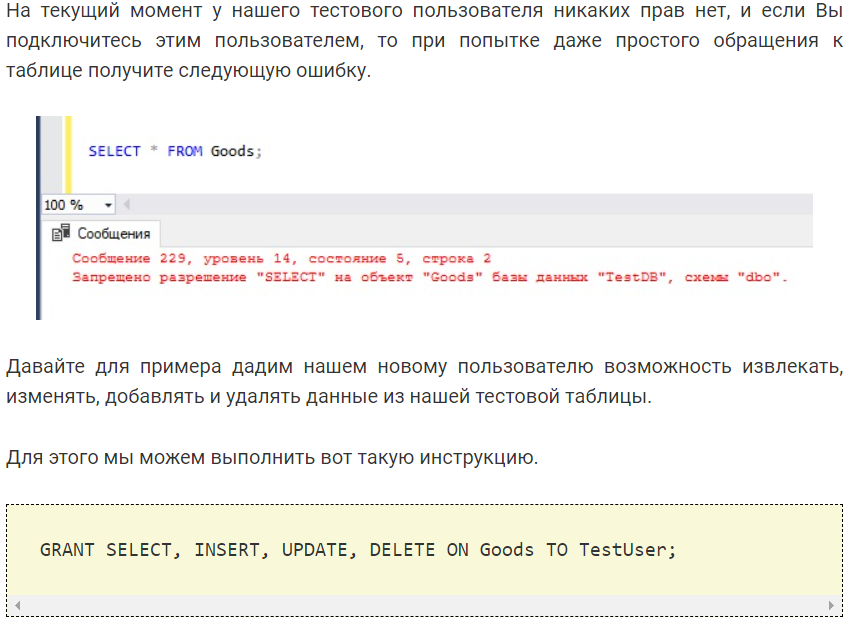
К операторам DCL относятся: GRANT, REVOKE, DENY.

GRANT – предоставляет пользователю или группе разрешения на определённые операции с объектом.

В качестве объекта может выступать: таблица, представление, функция, [хранимая процедура](https://info-comp.ru/programmirovanie/636-stored-procedures-in-t-sql.html) и т.д.

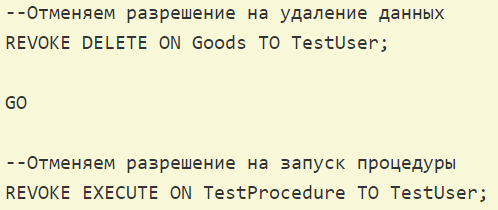
Операции над объектами могут быть разными, например, у таблицы это извлечение данных (SELECT), добавление данных (INSERT), изменение данных (UPDATE), удаление данных (DELETE), а также изменение самой таблицы (ALTER).

Пример:



GRANT EXECUTE ON TestProcedure TO TestUser; - для разрешения на выполнение процедуры.

REVOKE – отзывает выданные разрешения. Иным словами, с помощью этого оператора мы можем отменить выданное или запрещенное ранее разрешение.



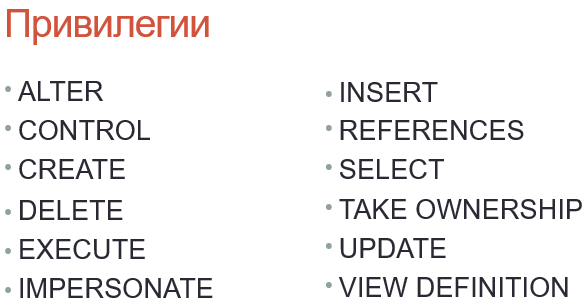
DENY– задаёт запрет, имеющий приоритет над разрешением.

С помощью DENY мы явно запрещаем выполнение определенных действий даже в том случае, если пользователь состоит в группе или роли, которая имеет разрешение на выполнение этих действий. Таким образом, мы предотвращаем наследование разрешения участником через его членство в группе или роли.

DENY имеет приоритет над всеми разрешениями, но не применяется к владельцам объектов или членам с предопределенной ролью сервера sysadmin, так как им не может быть отказано в разрешениях.

DENY EXECUTE ON TestProcedure TO TestUser; - пример

==============================================================

 Ну, назначение/отъём привилегий в примерах про операторы DCL были.

56. Основы безопасности в SQL Server. Аутентификация и авторизация. Имена входа. Пользователи. Роли.

**Аутентификация** – определение, кто такой конкретный пользователь.

**Авторизация** – определение того, какие действия можно производить аутентифицированному пользователю.

## Аутентификация в SQL Server

Аккаунт SQL Server можно разделить на 2 части: **Имя входа** и **Пользователь**.

* **Имя входа** – это глобальный логин для всего экземпляра SQL Server. С помощью него вы проходите процесс аутентификации;
* **Пользователь** – это участник базы данных, привязанный к определенному Имени Входа.

Например, ваше имя входа на сервер может быть *domain\username*, а пользователь в базе данных, привязанный к этому имени входа может называться *domain\_databaseUser*. Практически всегда имя входа и пользователь в базе данных совпадают по названию, но нужно иметь в виду что они могут и различаться, иметь разные имена.

SQL Server поддерживает 2 режима **аутентификации**:

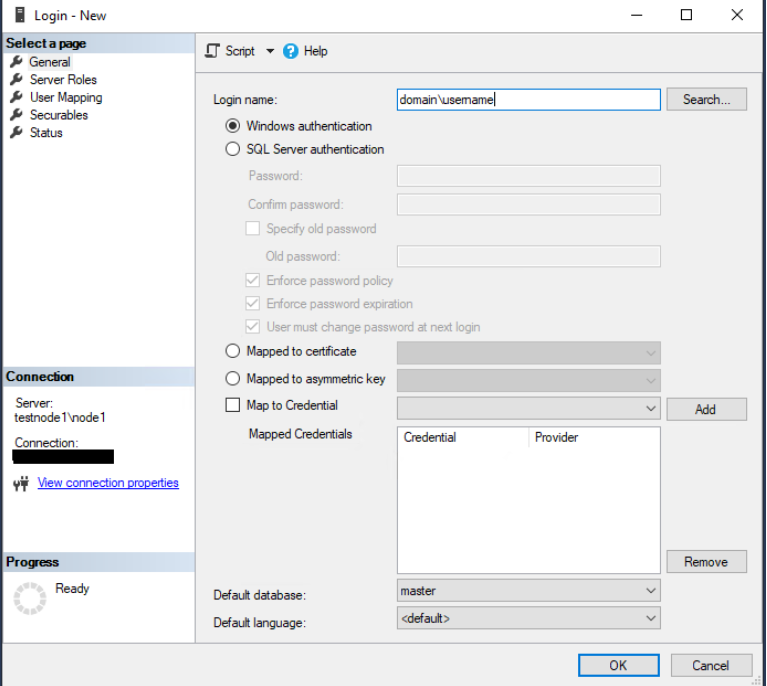
* **Аутентификация Windows** (Windows Authentication) – аутентификация осуществляется с помощью системы безопасности Windows. Пользователям, которые уже аутентифицированы в Windows и имеют права на SQL Server не нужно предоставлять дополнительные учетные данные.
* **Смешанный режим аутентификации** (Mixed Mode Authentication) – в этом режиме помимо аутентификации Windows поддерживается аутентификация самого SQL Server через логин и пароль.

Microsoft рекомендует использовать аутентификацию Windows, если есть такая возможность. Для аутентификации посредством логина и пароля, данные (логин и пароль) передаются по сети, хоть и в зашифрованном виде. При Windows аутентификации по сети передаётся серия зашифрованных сообщений, в которых не участвует пароль пользователя.

SQL Server поддерживает три типа **Login Name** (имен входа):

* **Локальная учетная запись** пользователя Windows или учетная запись **домена**/доверенного домена.
* **Группа Windows**. Предоставление доступа локальной группе Windows или группе из AD домена. Позволяет предоставить доступ ко всем пользователям, которые являются членами группы.
* **Логин SQL Server** (SQL Server authentication). SQL Server хранит имя пользователя и хэш пароля в базе данных **master**, используя методы внутренней аутентификации для проверки входа в систему.

SQL Server автоматически интегрируется с Active Directory. Если вы хотите раздать права доменной учетной записи, вам нужно использовать NetBios имя домена и логин учетной записи. Например для пользователя username в домене domain.local будет верным “domain\username”.



## Авторизация в SQL Server

Для авторизации SQL Server использует безопасность на основе ролей, которая позволяет назначать разрешения для роли или группы Windows/домена, а не отдельным пользователям. В SQL Server есть встроенные роли сервера и баз данных, у которых есть предопределенный набор разрешений.

В SQL Server есть 3 уровня безопасности, их можно представить, как иерархию от высшего к низшему:

* **Уровень сервера** – на этом уровне можно раздать права на базы данных, учетные записи, роли сервера и группы доступности;
* **Уровень базы данных** включают в себя схемы, пользователи базы данных, роли базы данных и полнотекстовые каталоги;
* **Уровень схемы** включают такие объекты, как таблицы, представления, функции и хранимые процедуры.

РОЛИ

Роли уровня сервера:

SQL Server предоставляет роли уровня сервера и помогает управлять разрешениями на сервере. Эти роли являются субъектами безопасности, группирующими других участников. Разрешения ролей уровня сервера распространяются на весь сервер.

|  |  |
| --- | --- |
| **Предопределенная роль уровня сервера** | **Описание** |
| sysadmin | Члены предопределенной роли сервера sysadmin могут выполнять любые действия на сервере. |
| serveradmin | Элементы предопределенной роли сервера serveradmin могут изменять параметры конфигурации на уровне сервера, а также выключать сервер. |
| securityadmin | Элементы предопределенной роли сервера securityadmin управляют именами входа и их свойствами. Это могут быть разрешения на уровне сервера GRANT, DENY и REVOKE. Они также могут предоставлять, запрещать и отменять разрешения на уровне базы данных (инструкции GRANT, DENY и REVOKE) при наличии доступа к базе данных. Кроме того, они могут сбрасывать пароли для имен входа SQL Server .  Важно! Возможность предоставления доступа к компоненту Компонент Database Engine и настройки разрешений пользователей позволяет администратору безопасности назначать большинство разрешений сервера. Роль securityadmin должна считаться эквивалентной роли sysadmin . |
| processadmin | Члены предопределенной роли сервера processadmin могут завершать процессы, работающие на экземпляре SQL Server. |
| setupadmin | Члены предопределенной роли сервера setupadmin могут добавлять или удалять связанные серверы с помощью инструкций Transact-SQL. (Членство в роли sysadmin необходимо при использовании службы Среда Management Studio.) |
| bulkadmin | Элементы предопределенной роли сервера bulkadmin могут выполнять инструкцию BULK INSERT. |
| diskadmin | Предопределенная роль сервера diskadmin используется для управления файлами на диске. |
| dbcreator | Члены предопределенной роли сервера dbcreator могут создавать, изменять, удалять и восстанавливать любые базы данных. |
| public | Каждое имя для входа SQL Server принадлежит к роли сервера public. Если для участника на уровне сервера не были предоставлены или запрещены конкретные разрешения на защищаемый объект, он наследует разрешения роли public на этот объект. Разрешения роли public следует назначать только тому объекту, который будет доступен всем пользователям. Нельзя изменить членство в роли public.  Примечание. Роль public реализуется не так, как другие роли. В разрешениях может быть отказано, они могут предоставляться либо отменяться для предопределенных ролей public. |

Роли уровня базы данных:

Существует два типа ролей уровня базы данных: *предопределенные роли базы данных* , являющиеся стандартными для базы данных, и *пользовательские роли базы данных* , которые можно создавать.

ПРЕДОПРЕДЕЛЁННЫЕ РОЛИ

|  |  |
| --- | --- |
| **Имя предопределенной роли базы данных** | **Описание** |
| db\_owner | Члены предопределенной роли базы данных db\_owner могут выполнять все действия по настройке и обслуживанию базы данных, а также удалять базу данных в SQL Server. (В База данных SQL и Azure Synapse некоторые операции по обслуживанию требуют наличия разрешений на уровне сервера и не могут быть выполнены членами db\_owner.) |
| db\_securityadmin | Элементы предопределенной роли базы данных db\_securityadmin могут изменять членство в роли (только для настраиваемых ролей) и управлять разрешениями. Элементы этой роли потенциально могут повышать свои права доступа, поэтому необходимо отслеживать их действия. |
| db\_accessadmin | Члены предопределенной роли базы данных db\_accessadmin могут добавлять или удалять права удаленного доступа к базе данных для имен входа и групп Windows, а также имен входа SQL Server . |
| db\_backupoperator | Члены предопределенной роли базы данных db\_backupoperator могут создавать резервные копии базы данных. |
| db\_ddladmin | Члены предопределенной роли базы данных db\_ddladmin могут выполнять любые команды языка определения данных (DDL) в базе данных. |
| db\_datawriter | Члены предопределенной роли базы данных db\_datawriter могут добавлять, удалять или изменять данные во всех пользовательских таблицах. |
| db\_datareader | Члены предопределенной роли базы данных db\_datareader могут считывать все данные из всех пользовательских таблиц и представлений. Пользовательские объекты могут существовать в любой схеме, кроме *sys* и *INFORMATION\_SCHEMA*. |
| db\_denydatawriter | Члены предопределенной роли базы данных db\_denydatawriter не могут добавлять, изменять или удалять данные в пользовательских таблицах базы данных. |
| db\_denydatareader | Члены предопределенной роли базы данных db\_denydatareader не могут считывать никакие данные из пользовательских таблиц или представлений в базе. |

ПОЛЬЗОВАЬТЕЛЬСКИЕ РОЛИ : НА САЙТЕ ПРИМЕРОВ НЕ БЫЛО (<https://docs.microsoft.com/ru-ru/sql/relational-databases/security/authentication-access/database-level-roles?view=sql-server-ver15>)

Роли уровня базы приложений:

Роль приложения — это участник базы данных, позволяющий приложению выполняться со своими, подобными пользовательским, правами доступа. Роли приложений можно использовать для разрешения доступа к определенным данным только тем пользователям, которые подключены посредством конкретного приложения. В отличие от ролей баз данных, роли приложений не содержат элементов и по умолчанию находятся в неактивном состоянии. Роли приложений активируются с помощью процедуры sp\_setapprole, которая требует указания пароля. Так как роли приложений являются участниками на уровне базы данных, они имеют доступ к другим базам данных только с разрешениями, предоставленными учетной записи пользователя guest в этих базах данных. Таким образом, любая база данных, в которой была отключена учетная запись пользователя guest , не будет доступна для ролей приложений в других базах данных.

**57. XML в SQL Server. Секция for XML в select. Директивы PATH, AUTO, RAW. Директивы TYPE, ELEMENTS, ROOT.**

**XML** (Extensible Markup Language) – расширяемый язык разметки. XML-формат часто используется для обмена данными между компонентами информационных систем. При работе с базами данных важными являются две задачи: преобразование *табличных данных в XML-структуры* и преобразование *XML-структур в строки реляционной таблицы*.

Для преобразования результата SELECT-запроса в формат XML в операторе SELECT применяется секция FOR XML. При этом могут использоваться режимы RAW, AUTO, PATH.

В режиме RAW в результате SELECT-запроса создается XML-фрагмент, состоящий из последовательности элементов с именем **row**. Каждый элемент **row** соответствует строке результирующего набора, имена его атрибутов совпадают с именами столбцов результирующего набора, а значения атрибутов равны их значениям.

Особенность режима AUTO проявляется в многотабличных запросах. В этом случае режим AUTO позволяет построить XML-фрагмент с применением вложенных элементов.

PATH – сочетание атрибутной и элементной форм.

Директивы:

TYPE - сохранять результат реляционного запроса как XML-документ или фрагмент типа данных XML.

ELEMENTS - она всё бахает как елементы. Атрибутов нет.

ROOT - Добавление к результирующему набору XML одного элемента верхнего уровня.

**58. XML в SQL Server. Применение процедур sp\_xml\_preparedocument, sp\_xml\_removedocument. Применение openxml.**

Пример преобразования XML-структуры в строки реляционной таблицы:

use ПРОДАЖИ

go

declare @h int = 0,

@x varchar(2000) = ' <?xml version="1.0" encoding="windows-1251" ?>

<товары>

<товар="стол" цена="40" количество="5" />

<товар="стул" цена="10" количество="3" />

<товар="шкаф" цена="400" количество="1" />

</товары>';

exec sp\_xml\_preparedocument @h output, @x; -- подготовка документа

select \* from openxml(@h, '/товары/товар', 0)

with([товар] nvarchar(20), [цена] real, [количество] int )

exec sp\_xml\_removedocument @h; -- удаление документа

Для преобразования XML-данных в строки таблицы предназначена функция OPENXML, которая принимает три входных параметра: дескриптор, выражение XPATH и целое положительное число, определяющее режим работы функции.

Дескриптор определяется процедурой SP\_XML\_PREPAREDOCUMENT, которая должна быть выполнена до SELECT-запроса, применяющего OPENXML. Процедура принимает в качестве входного параметра XML-документ (в формате строки) и возвращает дескриптор.

Выражение XPATH предназначено для выбора требуемых данных из исходного XML-документа.

Режим работы указывает на тип преобразования (0 используется атрибутивная модель сопоставления, каждый XML-атрибут преобразовывается в столбец таблицы; 1 аналогично типу 0, но для необработанных столбцов применяется сопоставление на основе элементов XML-документа; 2 используется сопоставление на основе элементов, каждый элемент преобразовывается в столбец таблицы).

С помощью выражения WITH должна быть указана структура формируемого результата.

Для того, чтобы извлечь данные о товарах из XML-документа и добавить их в таблицу Товары, надо заменить оператор

select \* from openxml(@h, '/товары/товар', 0)

with([товар] nvarchar(20), [цена] real, [количество] int )

на

insert Товары select [товар], [цена], [количество]

from openxml(@h, '/товары/товар', 0)

with([товар] nvarchar(20), [цена] real, [количество] int )

**59. XML в SQL Server. Схема XML-документа. Коллекция XML SCHEMA.**

Сложность XML-данных требует иного подхода к механизму ограничения целостности для этого типа данных. В семействе XML-технологий существует и активно используется технология, основанная на языке XML-Schema.

XML-Schema – это одна из реализаций языка XML, поддерживаемая консорциумом W3C и предназначенная для описания структуры XML-документа. С помощью языка XML-Schema можно описать правила, которым должен подчиниться XML-документ. Файл, содержащий XML-Schema, обычно имеет расширение XSD (XML Schema definition). Большинство современных систем программирования предусматривают встроенные механизмы, позволяющие с помощью заданного XSD-файла проверять на корректность XML-документы. Для знакомства с языком XML-Schema рекомендуются источники [6, 7].

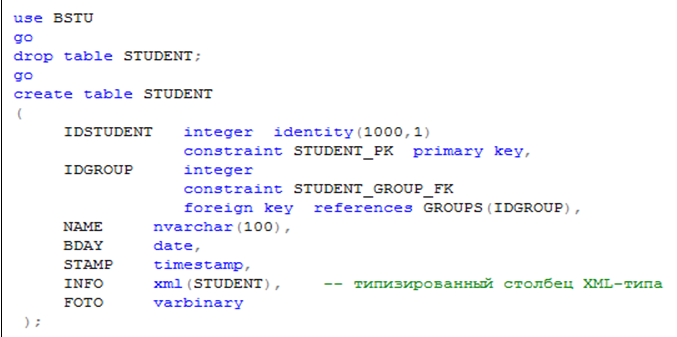
Для хранения документов XML-Schema в БД MSS предусмотрен специальный объект – XML SCHEMA COLLECTION. Каждый такой объект может содержать один или более XML-SCHEMA-документов.

На рис. 14.27 приведен пример создания объекта XML SCHEMACOLLECTION (далее коллекция схем) с именем **Student**, содержащего один документ.



Документ XML-Schema, размещенный в коллекции **Student**, описывает XML-документ с корневым элементом **студент** (первый тег **element**).

При создании таблицы для XML-столбца можно указать имя коллекции схем (рис. 14.28). Такой столбец называется типизированным столбцом. Типизированный XML-столбец должен удовлетворять хотя бы одной схеме из связанной с ним коллекции.



**60. XML в SQL Server. Индексирование XML. Инструкции xPath. Методы типа данных xml.**

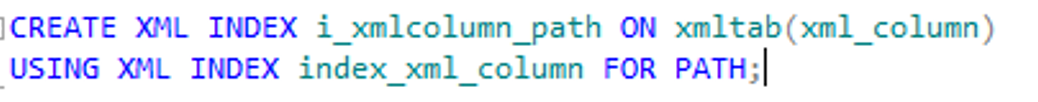
Первичный XML-индекс:

* Индексируются все теги, значения и пути
* Используется для возвращения скалярных значений или поддеревьев



Три типа вторичных типа XML-индексов:

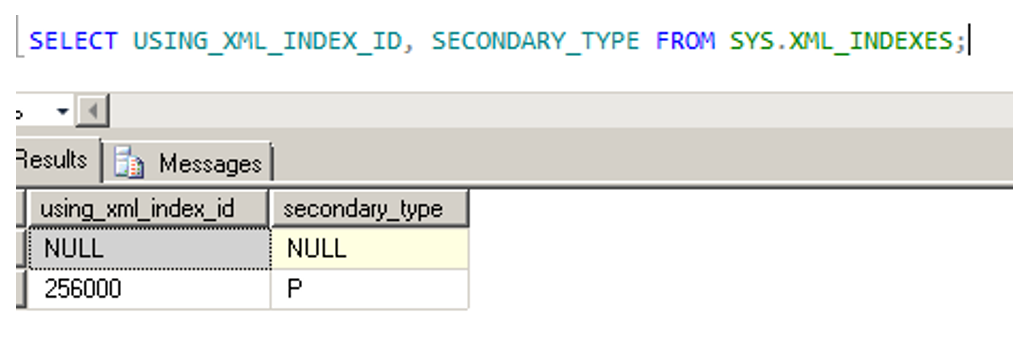
* FOR PATH — по структуре
* FOR VALUE — по значениям элементов и атрибутов
* FOR PROPERTY — по свойствам



Индексы xml:

* Не могут быть составными
* Не могут быть кластеризованными

Хранятся в таблице sys.xml\_indexes



**Xpath**

Язык запросов Xpath

* XML Path Language — язык запросов к элементам XML-документа

Язык запросов Xquery

* XQuery — язык запросов, разработанный для обработки данных в формате XML

**Запрос данных XPath:**

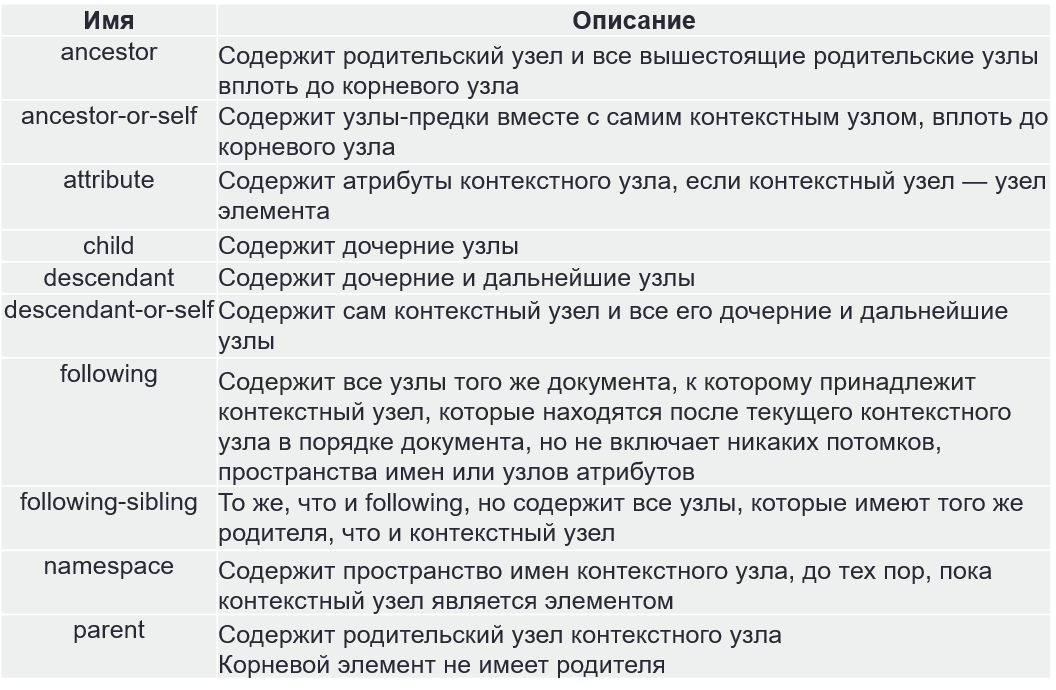
для доступа к элементам и атрибутам XML-документа

• Дочерние элементы узла /customer/\*

• Все атрибуты узла /customer/!?\*

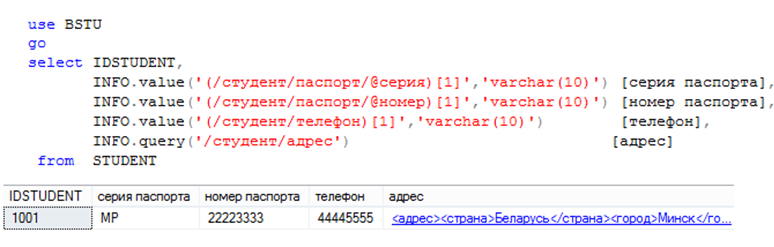
• Чтобы вернуть только покупателей из региона Dallas /customer[@region = " Dallas "]

**Оси XPath:**

****

**Методы типа данных XML**

|  |  |
| --- | --- |
| Метод | Описание |
| query | Исполняет выражения XPATH и XQUERY и возвращает XML-фрагмент |
| value | Исполняет выражения XPATH и XQUERY и возвращает скалярное значение, которое может быть преобразовано в тип SQL |
| exist | Исполняет выражения XPATH и XQUERY и возвращает 1, если узел, заданный выражением, найден |
| modify | Изменение XML-содержимого |
| nodes | Исполняет выражения XPATH и XQUERY и возвращает XML-фрагмент |



Обратите внимание (рис. 14.24): 1) XML-тип является объектным типом (имеет методы и свойства); 2) в первых двух случаях применения метода **value** извлекаются значения атрибутов **серия** и **номер**; 3) в третьем случае применения **value** извлекается текстовое значение элемента **телефон**; 4) во всех трех случаях применения метода **value** в конце заданного выражения стоит число в квадратных скобках, обозначающее номер выбираемого экземпляра; дело в том, что выражения, указанные в качестве параметров, позволяют выбрать более чем одно значение; указанная в квадратных скобках единица позволяет получить только первый экземпляр; 5) второй параметр метода **value** указывает на тип данных, к которому должно быть преобразовано выбранное значение; 6) метод **query**, применяемый в четвертом элементе SELECT-списка, позволяет выбрать XML-фрагмент.