**Вопросы к экзамену по дисциплине «Базы данных»**для студентов специальности

1-40 01 01 «Программное обеспечение информационных технологий»

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

База данных – совокупность взаимосвязанных данных.

Система управления базами данных – программная реализация технологии хранения, извлечения, обновления и обработки данных в базе данных.

Основные требования к информации 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

1. Модели данных, основная терминология реляционных баз данных.

Модели данных:

• Иерархическая – это модель данных, где используется представление базы данных в виде древови дной (иерархической) структуры.

• Сетевая – как древовидная, но предков может быть больше, чем один.

• Реляционная – модель данных, основанная на отношениях (на теории множеств).

Терминология реляционных баз данных:

• домен: множество;

• таблица: отношение;

• атрибут: имя столбца таблицы (имя домена);

• заголовок таблицы: множество всех атрибутов;

• кортеж: элемент отношения или строка таблицы

Операции Реляционной бд:

* **UNION** (объединение)
* **INTERSECT** (пересечение)
* **MINUS** (разность)
* **TIMES** (декартово произведение)
* **WHERE** (ограничение)
* **PROJECT** (проекция)
* **JOIN** (соединение)
* **DIVIDE BY** (реляционное деление)
* **RENAME** (переименование)
* **:=** (присваивание).

• 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

* Структурный аспект  — данные в базе данных представляют собой набор отношений
* Аспект целостности — отношения (таблицы) отвечают определенным условиям целостности
* РМД поддерживает декларативные ограничения целостности уровня домена (типа данных), уровня отношения и уровня базы данных
* Аспект обработки — РМД поддерживает операторы манипулирования отношениями (реляционная алгебра, реляционное исчисление)

1. Нормализация таблиц базы данных. Нормальные формы таблиц.

Нормализация данных – процесс преобразования таблиц базы данных к нормальной форме.

**Шесть нормальных форм** – 1NF, 2NF,...6NF.Широкое практическое применение имеют формы 1NF, 2NF, 3NF

Первая нормальная форма:

* Таблица не должна содержать повторяющихся групп данных
* Атомарность – каждый столбец должен содержать одно неделимое значение
* Устранить повторяющиеся группы в отдельных таблицах
* Создать отдельную таблицу для каждого набора связанных данных
* Идентифицировать каждый набор связанных данных с помощью первичного ключа

Чтобы таблица соответствовала 2-й нормальной форме (2NF), необходимо:

* Таблица находится в первой нормальной форме
* Каждый неключевой атрибут полностью функционально зависит от каждого возможного ключа
* Создать отдельные таблицы для наборов значений, относящихся к нескольким записям
* Связать эти таблицы с помощью внешнего ключа

Для перехода к 3-й нормальной форме (3NF), необходимо обеспечить,

* чтобы все таблицы находились во 2-й нормальной форме и
* Отсутствуют транзитивные зависимости(Транзитивные зависимости между полями базы данных существует тогда, когда значения не ключевых полей зависят от значений других не ключевых полей.)

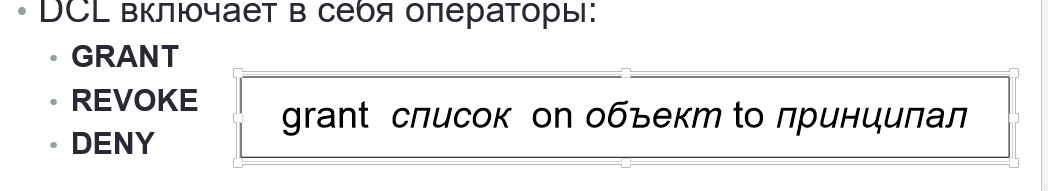
1. Язык SQL. Группы операторов SQL.

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

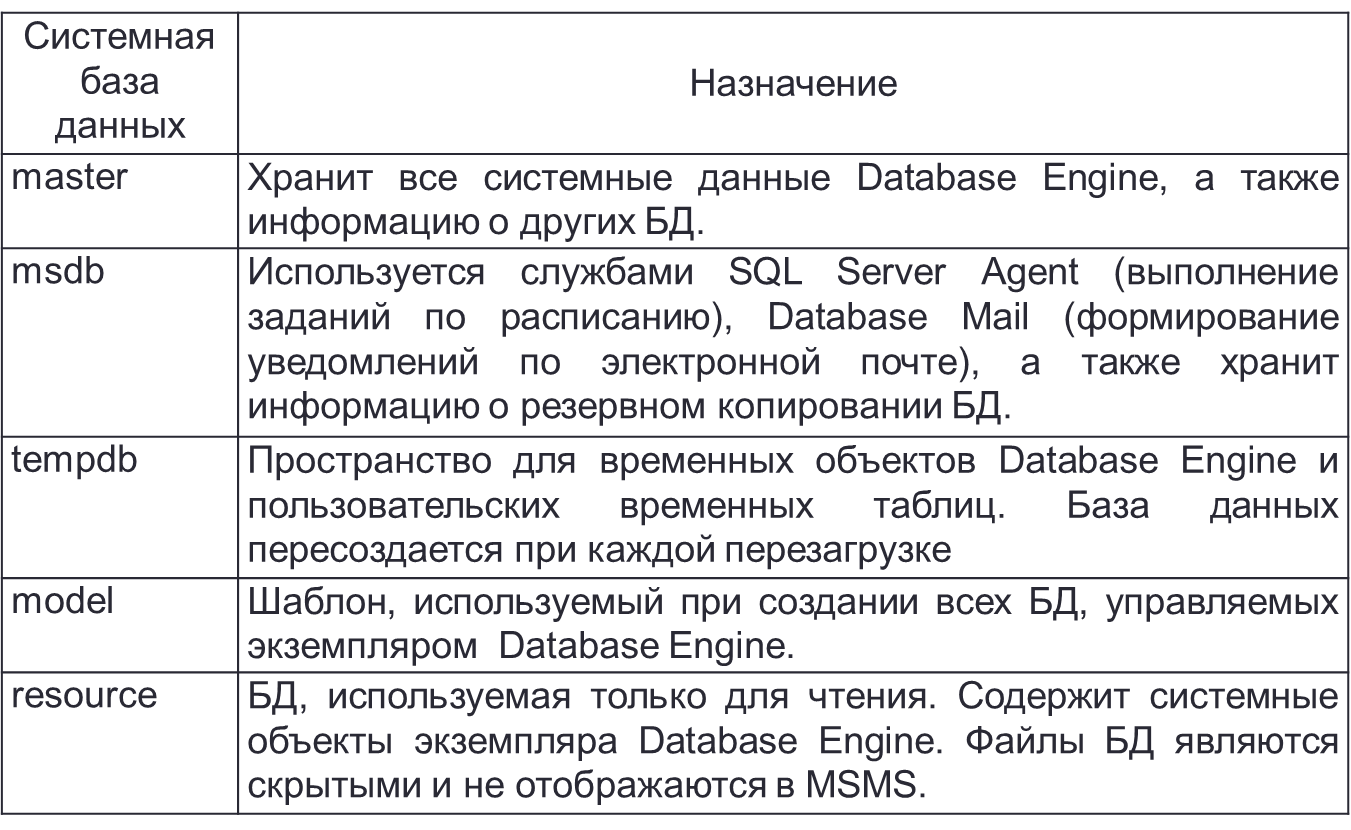
Операторы SQL:

* DDL - Data Definition Language - язык определения данных. предназначены для создания, удаления и изменения объектов БД или сервера СУБД: CREATE, DROP, ALTER.
* DML - Data Manipulation Language - язык манипулирования данными. предназначены для работы со строками таблиц: INSERT, DELETE, SELECT, UPDATE.
* TCL - Transaction Control Language - язык управления транзакциями. предназначены для управления транзакциями: BEGIN TRAN, SAVE TRAN, COMMIT TRAN, ROLLBACK TRAN.
* DCL - Data Control Language - язык управления данными. предназначены для управления процессом авторизации(**Авторизация** – это процедура проверки разрешений на выполнение определенных операций): GRANT, REVOKE, DENY (еще есть LOCK / UNLOCK , SET LOCK MODE).

**Принципал** – это объект сервера или БД, которому может быть выдано разрешение на выполнение операции, а также отобрано или запрещено разрешение



1. Системные базы данных: master, msdb, model, tempdb.



1. Структура файлов базы данных.

Структура файлов базы данных может варьироваться в зависимости от используемой системы управления базами данных (СУБД) и их конкретной реализации. Однако, я могу общим образом описать типичную структуру файлов базы данных.

Файл данных (Data File):

Этот файл содержит фактические данные, хранящиеся в базе данных.

Обычно данные организованы в виде таблиц, каждая из которых представляет собой совокупность строк и столбцов.

Каждая таблица может иметь свой отдельный файл данных, либо несколько таблиц могут совместно использовать один файл данных.

Файл журнала (Log File):

Файл журнала используется для записи операций, производимых в базе данных.

В него записываются изменения данных, такие как вставка, обновление и удаление записей.

Файл журнала обеспечивает целостность данных и восстановление базы данных в случае сбоев или сбоев системы.

Файл индекса (Index File):

Файл индекса содержит структурированные данные, используемые для ускорения поиска и доступа к данным.

Индексы создаются на основе одного или нескольких столбцов таблицы и позволяют быстро находить записи по определенным критериям.

Файлы индекса могут быть отдельными файлами или могут быть интегрированы в файлы данных.

Файл конфигурации (Configuration File):

Файл конфигурации содержит параметры и настройки базы данных.

Он определяет параметры, такие как размеры файлов, ограничения целостности данных, настройки безопасности и другие свойства базы данных.

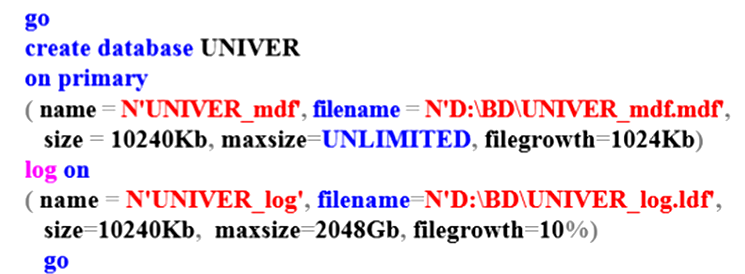
В базах данных Microsoft SQL Server используется следующая структура файлов:

Первичный файл (Primary Data File, расширение .mdf): Это основной файл базы данных, который содержит системные и пользовательские данные. Он содержит схему таблиц, индексы, представления, процедуры и другие объекты базы данных. Расширение .mdf обозначает файл данных первичного файла.

Вторичные файлы (Secondary Data Files, расширение .ndf): Вторичные файлы используются для расширения хранения данных базы данных на дополнительных дисковых устройствах или разделах. Они также содержат системные и пользовательские данные, но физически располагаются отдельно от первичного файла (.mdf). Расширение .ndf обозначает файлы данных вторичных файлов.

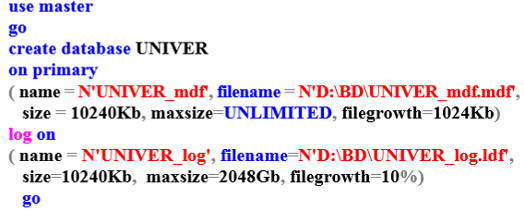
Файлы журнала транзакций (Transaction Log Files, расширение .ldf): Файлы журнала транзакций содержат информацию о всех изменениях, произведенных в базе данных. Это включает операции вставки, обновления, удаления и другие транзакции. Файлы журнала транзакций не содержат непосредственно пользовательские данные, а служат для обеспечения целостности и восстановления базы данных в случае сбоев или ошибок. Расширение .ldf обозначает файлы журнала транзакций.

Эта структура файлов позволяет эффективно организовать хранение данных и обеспечить защиту данных в базе данных SQL Server.



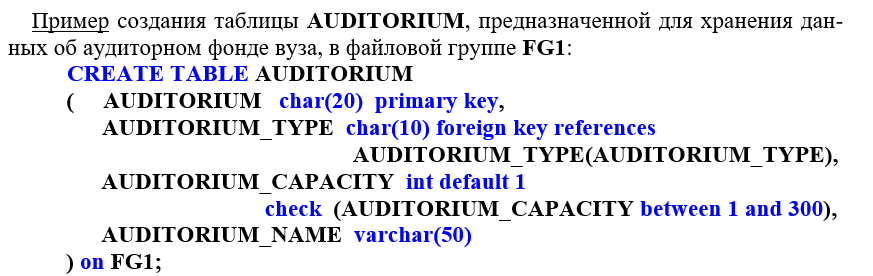
1. Создание файлов базы данных. Файловые группы. Создание таблиц в файловой группе. Дисковое хранение файлов базы данных. Страницы, экстенты.

БД представляет собой набор файлов операционной системы трех типов: первичный файл (расширение **mdf**), вторичные файлы (**ndf**) и файлы журнала транзакций (**log**). Все файлы БД, кроме файлов журнала транзакций, распределены по файловым группам. Файловые группы – это поименованный набор файлов БД.



Файловая группа, называемая первичной, является обязательной. Для ее обозначения используются ключевые слова **ON PRIMARY**. В примере создается база **UNIVER**, файл **UNIVER\_mdf.mdf** с первоначальным размером 10240Кb, максимальным размером неограниченным, приращением 1024Кb. Файл располагается на диске **D** в папке **BD** в первичной файловой группе.

Журнал транзакций в операторе **CREATE DATABASE** описывается отдельно в секции, обозначенной ключевыми словами **LOG ON**.



Файлы базы данных:

• первичный файл (.mdf) - содержит сведения, необходимые для запуска базы данных, и ссылки на другие файлы в базе данных. В каждой базе данных имеется один первичный файл данных.

• вторичные файлы (.ndf) - необязательные определяемые пользователем файлы данных. Данные могут быть распределены на несколько дисков, в этом случае каждый файл записывается на отдельный диск.

• файлы журнала транзакций (.log) - журнал содержит информацию для восстановления базы данных. Для каждой базы данных должен существовать хотя бы один файл журнала.

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

1. 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 определения 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)).

1. Таблицы. Создание, изменение и удаление таблиц.

При проектировании реляционной базы данных необходимо исследовать предметную область с целью определения объектов, нормализовать данные и установить связи между ними. **Нормализация** данных − это процесс, в результате выполнения которого таблицы базы данных проверяются на наличие зависимостей между столбцами и, если необходимо, то исходная таблица разделяется на несколько таблиц.

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

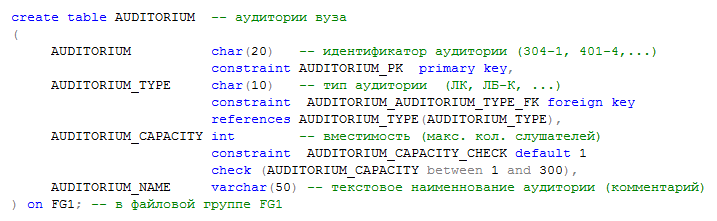
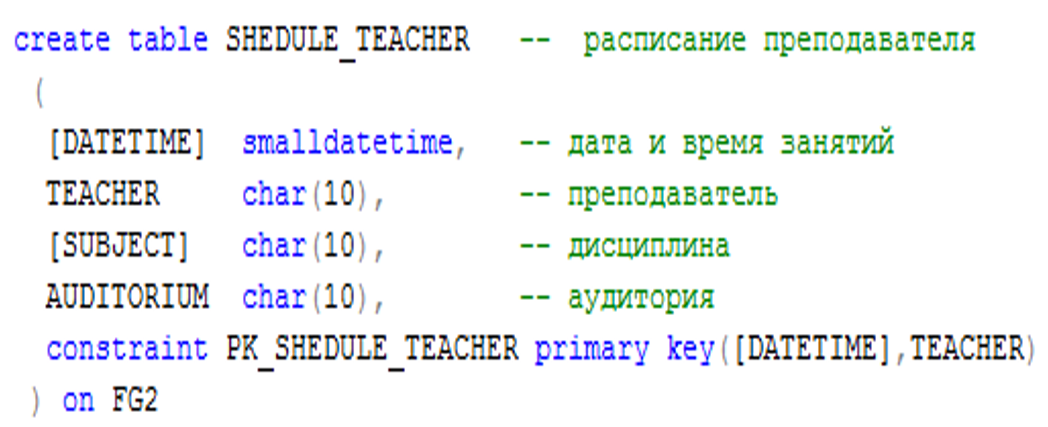


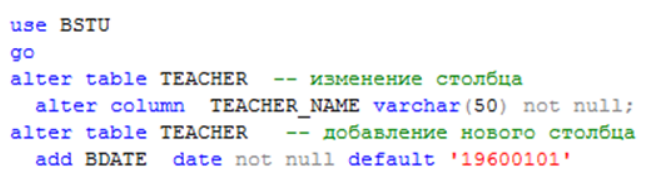
Таблица AUDITORIUM включает четыре столбца с именами: AUDITORIUM, AUDITORIUM\_TYPE, AUDITORIUM\_CAPACITY, AUDITORIUM\_NAME. Следует обратить внимание на то, что таблица будет размещена в файловой группе с именем FG1. Синтаксис оператора CREATE TABLE допускает другую форму записи ограничения:



**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.



1. **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 подождать, пока транзакция отработает)

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

1. 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.

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.

1. **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. Может быть задано имя. Если это имя не задано, при создании таблицы сервер назначает ограничениям собственные имена.

|  |  |  |
| --- | --- | --- |
| Условное | Действие ограничения целостности |  |
| обозначение |  |  |
| ограничения |  |  |
|  |  |  |
| **data type** | Предотвращает появление в столбце значений, не |  |
| тип данных | соответствующих типу данных |  |
|  |  |
|  |  |  |

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

Alter table => Drop, alter, add constraint. Constraint в столбцах таблицы.



**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. Иногда к этой группе относят оператор TRUNCATE (операция мгновенного удаления всех строк в таблице).

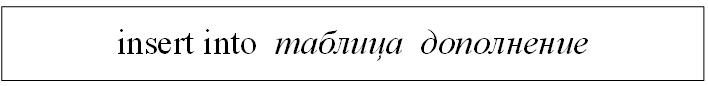
Наиболее мощным DML-оператором является SELECT. Он позволяет выбрать множество строк из одной или нескольких таблиц. При успешном выполнении этого оператора формируется результирующий набор, представляющий собой множество однотипных (с одинаковыми столбцами) строк. В общем случае результирующий набор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

Любой оператор SELECT содержит список, определяющий перечень столбцов (в общем случае и содержимое) результирующего набора. Дополнительная часть оператора описывает множество строк из одной или

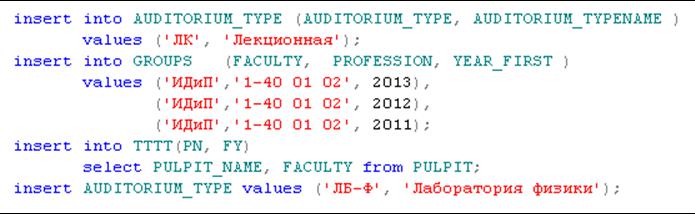
нескольких таблиц, служащее источником для формирования результирующего набора.



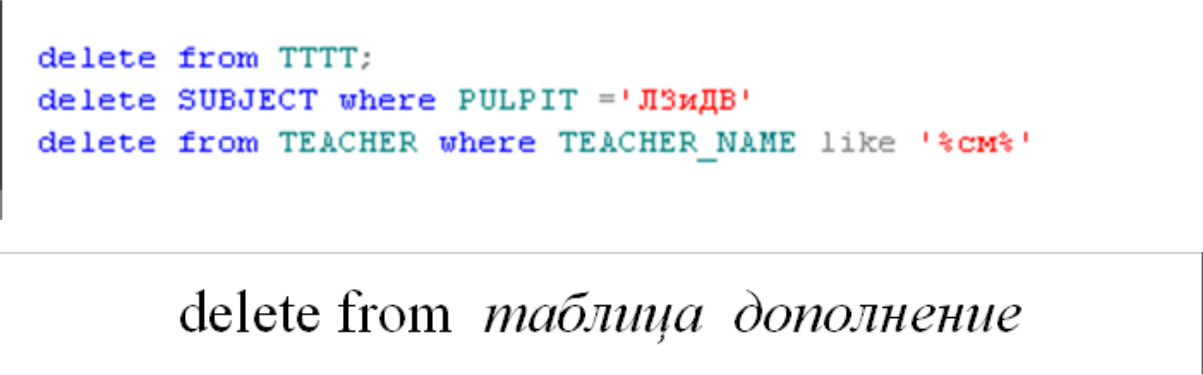
Добавить одну или несколько строк в существующую таблицу можно с помощью оператора **INSERT**. Ключевое слова **INTO** указывает на то, что далее следует имя таблицы, в которую будут добавляться строки. Далее структура оператора и примеры использования



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 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:

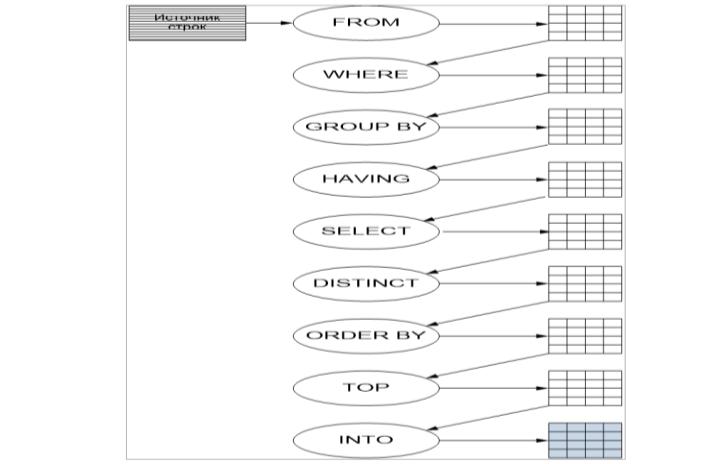


Удалить существующий объект сервера или БД можно с помощью оператора **DROP**. Структура оператора :



1. **Оператор SELECT. Секции FROM, WHERE, GROUP BY,**

**HAVING, ORDER BY, DISTINCT, TOP.**



* **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
  1. **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.

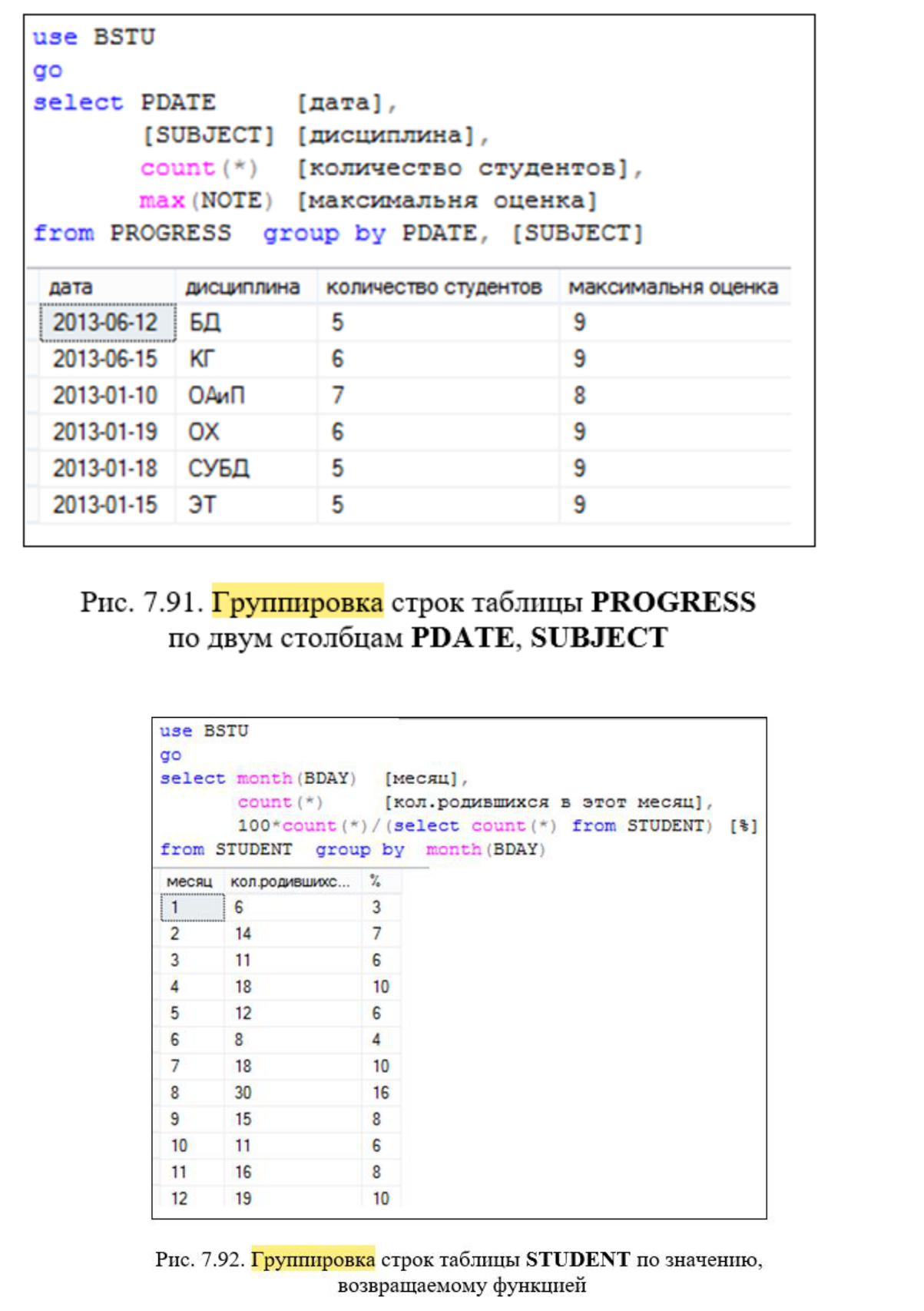
Операция >=ANY формирует истинное значение в том случае, если значение стоящее слева, больше или равно хотя бы одному значению в списке, указанном справа.

Not — инвертирует логическое значение.

1. **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 суммызначений).

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(вопрос 18)

1. **Оператор SELECT. Использование агрегатных функций.**

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

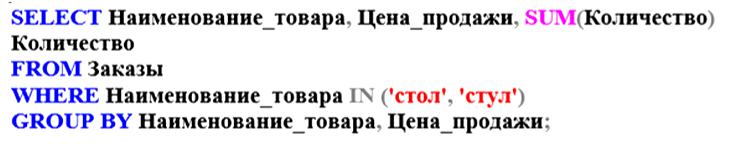
**Наиболее часто применяемые агрегатные функции**

|  |  |
| --- | --- |
| Наименование | Назначение |
| функции |  |
|  |  |
| 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.



1. **Оператор SELECT. Группировка данных с использованием**

**CUBE, ROLLUP**

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

|  |
| --- |
|  |

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

|  |
| --- |
|  |

1. **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]

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.

В приведенном выше примере представление не удовлетворяет одному из этих правил.

1. **Представления. Операции DML над представлениями. Использование представлений с указанием WITH CHECK OPTION.**

**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.

В операторе CREATE после ключевого слова **VIEW** следует **имя представления** и далее после ключевого слова **AS** текст SELECT-запроса, лежащего в основе представления.

При этом к SELECT-запросу представления предъявляется следующие **требования**:

* секцию ORDER BY можно использовать только совместно с опцией TOP;
* не допускается применение секции INTO;
* все столбцы результирующего набора, формируемого SELECT-запросом, должны быть **поименованы**.

При создании представлений, позволяющих выполнять операции INSERT, DELETE и UPDATE (DML), 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;

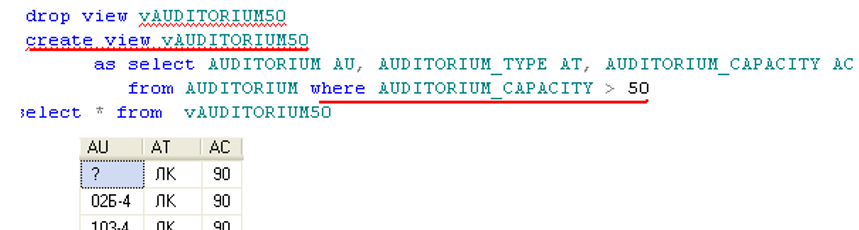
− в секции FROM запроса должна указываться только одна таблица.

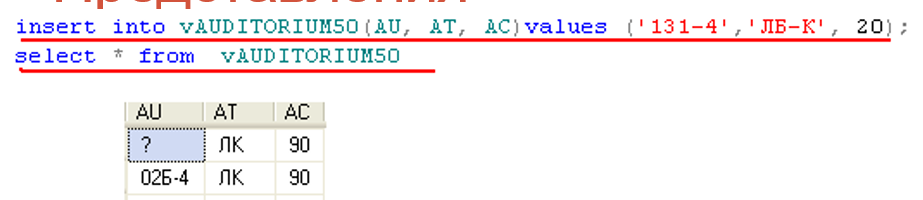
Особенности:

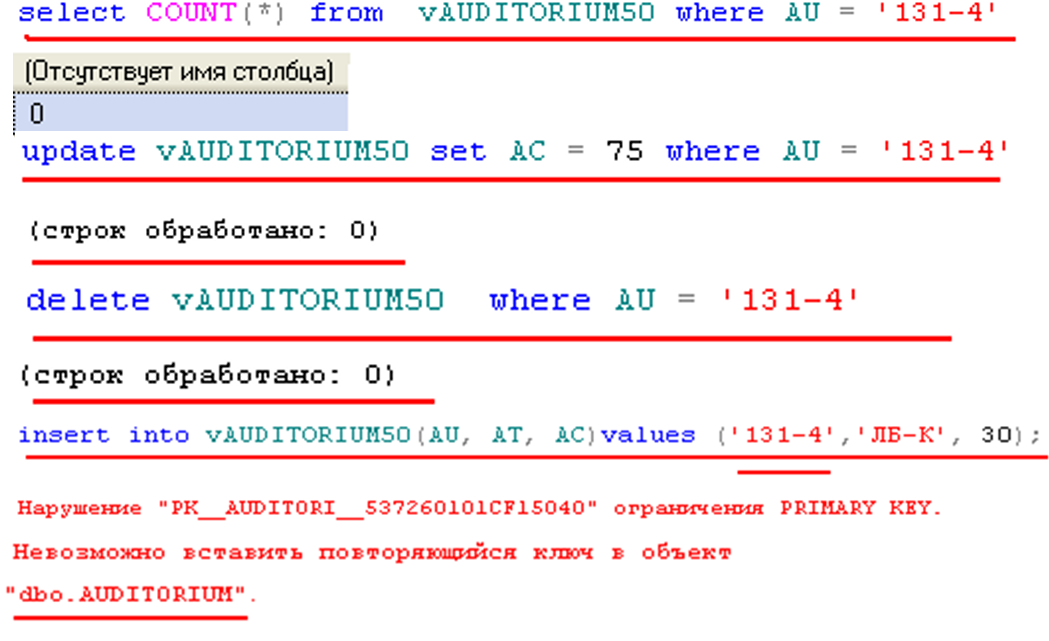
* Любая операция выполняется над базовой таблицей
* Все существующие **ограничения** **целостности** базовой таблицы наследуются представлением

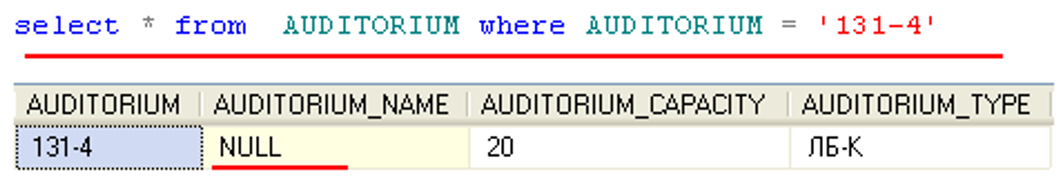
|  |  |
| --- | --- |
|  |  |











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. Выполнение INSERT и UPDATE допускается, но с учетом ограничения, задаваемого опцией WITH CHECK OPTION.

1. **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.

Схема – это поименованный контейнер объектов БД, позволяющий разграничить объекты с одинаковыми именами.

1. **Функции преобразования типов данных CAST и CONVERT.**

CAST ([Аргумент] as [Тип\_Данных])

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

CONVERT([Тип\_Данных], [Аргумент])

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

Хотя обе функции выполняют одну и ту же задачу - преобразование данных, между ними есть несколько отличий. В некоторых преобразованиях функция CAST обеспечивает более быструю и простую производительность или наоборот. Функция CONVERT лучше при попытке преобразовать значения даты и времени, дробные числа и денежные знаки. Однако функция CAST соответствует стандарту ANSI и является более переносимой по сравнению с функцией CONVERT и может использоваться для других приложений баз данных, не требуя от пользователя значительных изменений. CAST также удобнее, когда дело доходит до преобразования десятичных и числовых значений, так как функция может сохранить десятичные дроби из исходных выражений. Многие эксперты предлагают использовать CAST и CONVERT вместе, используя CAST в первую очередь для преобразования и CONVERT для других задач, которые намного лучше в функции CONVERT.

1. **Операторы работы с множествами UNION (ALL), INTERSECT,**

**EXCEPT**

UNION - объединение без копий

UNION ALL - объединение с копиями

Оператор **UNION** выполняет теоретико-множественную операцию объединения результирующих наборов SELECT-запросов, в котором строки не могут повторяться. Если требуется механическое объединение строк, можно применить оператор **UNION ALL**.

SELECT \* FROM Товары WHERE Название = ‘Коля’

UNION

SELECT \* FROM Товары WHERE Название = ‘Николя’

INTERSECT - пересечение двух исходных наборов EXCEPT - разность двух исходных наборов

Результатом оператора **INTERSECT** является набор строк, являющийся пересечением двух исходных результирующих наборов SELECT-запросов.

Результатом оператора **EXCEPT** является набор строк, являющийся разностью двух исходных результирующих наборов SELECT-запросов (т.е. в результат включаются те строки, которые есть в первом запросе, но отсутствуют во втором).

1. **Соединение таблиц. Внутреннее соединение INNER JOIN**

Оператор языка SQL JOIN предназначен для соединения двух или более таблиц базы данных по совпадающему условию. Этот оператор существует только в реляционных базах данных. Именно благодаря JOIN реляционные базы данных обладают такой мощной функциональностью, которая позволяет вести не только хранение данных, но и их, хотя бы простейший, анализ с помощью запросов.

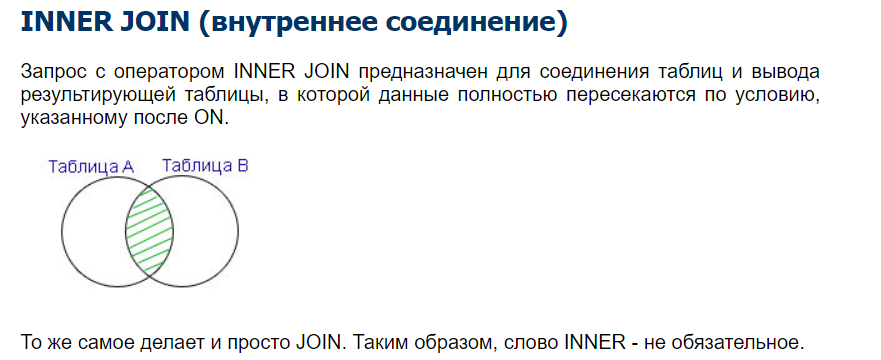
Для соединения двух таблиц оператор SQL JOIN имеет 2 синтаксиса: явный и неявный.

|  |  |
| --- | --- |
|  |  |

Неявный синтаксис устарел. Соединяется каждый с каждым. Максимальное количество таблиц, которые возможно соединить 64.

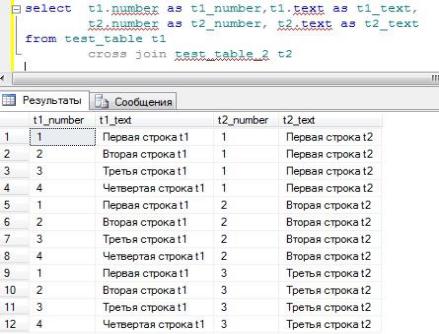
Соединение таблиц INNER JOIN (внутреннее соединение) наиболее часто используемый вид соединения реляционных таблиц.

Результирующий набор создается следующим образом: выполняется декартово произведение (каждая строка одной таблицы соединяется с каждой строкой другой); из полученного результата выбираются строки, удовлетворяющие указанному условию; из всех столбцов результирующего набора выбираются столбцы, указанные в списке SELECT.



1. **Ортогональное соединение CROSS JOIN**

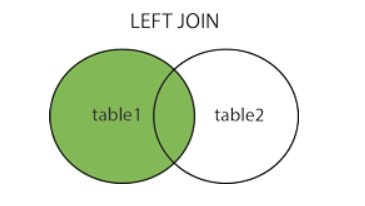
CROSS JOIN (декартово произведение) – это объединение SQL по которым каждая строка одной таблицы объединяется с каждой строкой другой таблицы. Количество строк n × m.



1. **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.



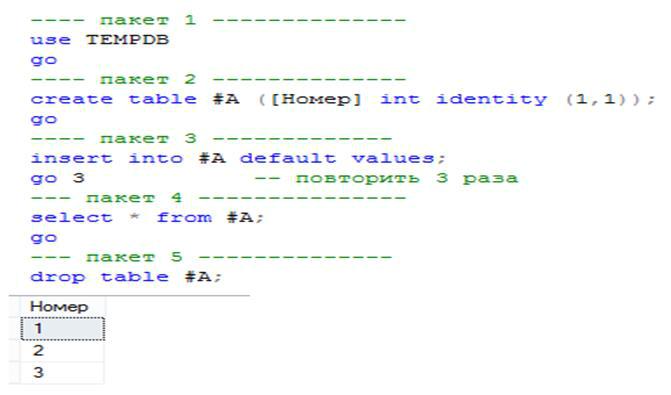
Левое внешнее соединение включает в набор несоединенные строки таблицы, имя которой записано слева от ключевых слов LEFT OUTER JOIN, а правое внешнее соединение – несоединенные строки таблицы, имя которой записано справа от RIGHT OUTER JOIN.

FULL OUTER JOIN определяет объединение правого и левого соединения.

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

Transact-SQL — процедурное расширение языка SQL, созданное компанией Microsoft и Sybase. SQL был расширен такими дополнительными возможностями как: управляющие операторы, локальные и глобальные переменные, различные дополнительные функции для обработки строк, дат, математики и т. п.

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





Для объявления переменных, используемых в программах, предназначен оператор

Имя переменной должно начинаться с символа @.

Областью видимости переменной являются все инструкции между ее объявлением и концом пакета или хранимой процедуры, где она объявлена.

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

T-SQL – это расширение языка SQL, реализованное в Microsoft SQL Server.

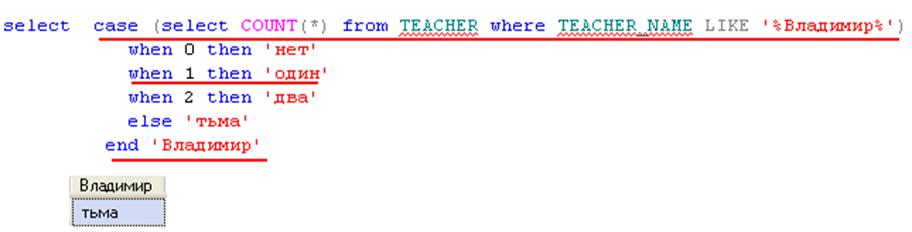
T-SQL – это сокращенное название языка, а полное название Тransact-SQL, т.е. транзакционный SQL.

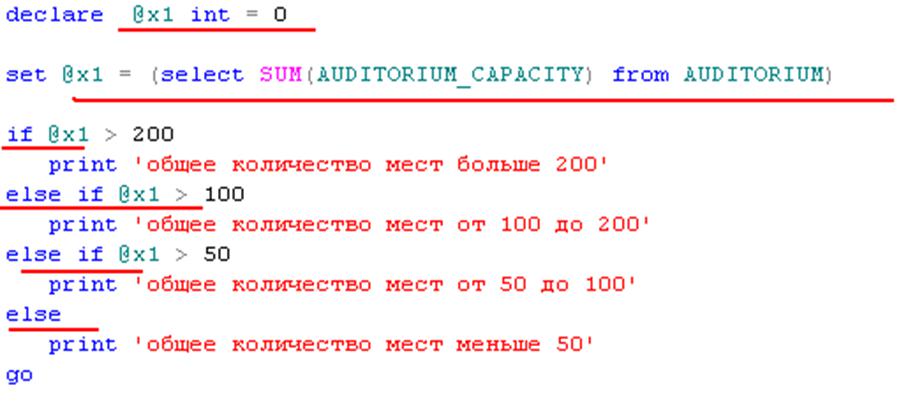
T-SQL обладает всеми возможностями языка SQL, однако предназначен он для решения задач программирования, не связанных с реляционными данными. Иными словами, на T-SQL Вы можете полноценно программировать, используя переменные, циклы, условные конструкции и другие возможности. T-SQL позволяет нам реализовывать сложные алгоритмы бизнес-логики в виде хранимых процедур и функций, тем самым создавая так называемые «программы» внутри базы данных.

Кроме этого, T-SQL упрощает написание SQL запросов за счет огромного количества встроенных системных функций, а также позволяет администрировать SQL Server и получать системную информацию с помощью системных представлений, процедур и функций.

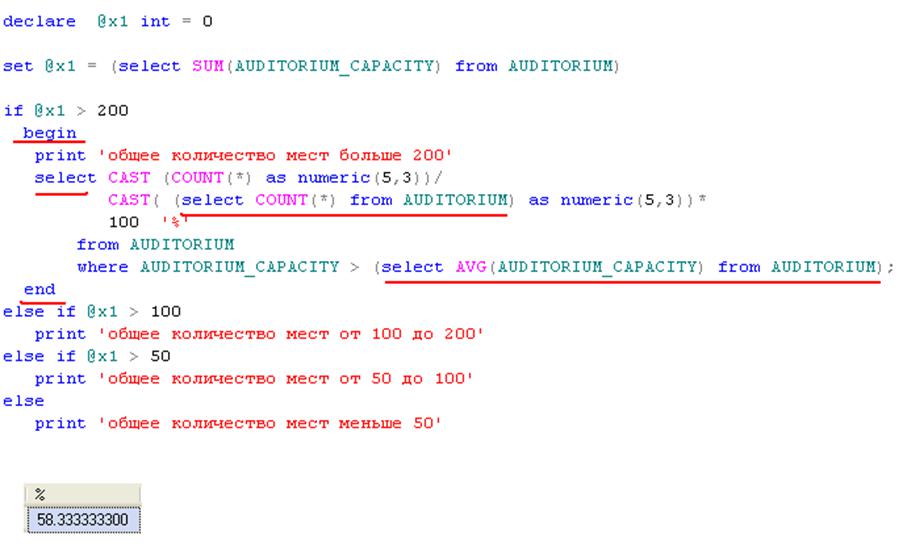
Операторы присвоения: set, select. Разница set для одного select для нескольких.

* 1. **Язык T-SQL. Операторы print, if-else, case**
* помощью оператора PRINT можно вывести строку в стандартный выходной поток.
* 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

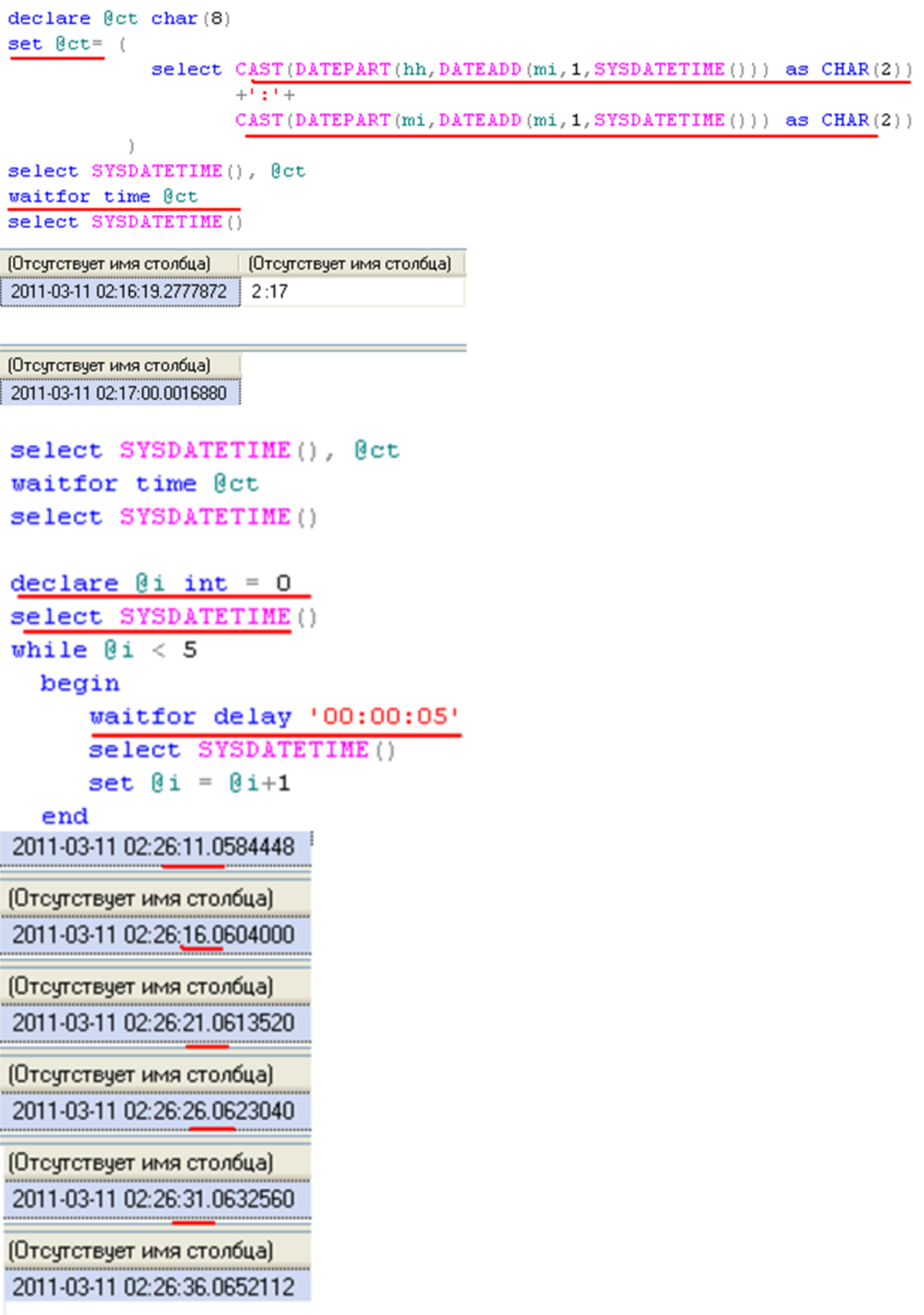




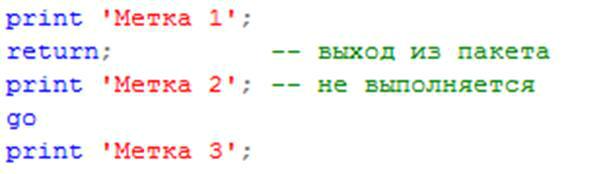
* + 1. **Язык T-SQL. Операторы begin-end, waitfor и return.**
  + помощью операторных скобок BEGIN END можно объединять операторы
* группы.



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



1. **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



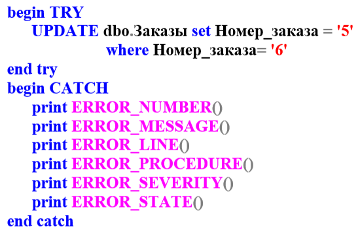
или более операторов, которые выполняются в том случае и до тех пор, пока логическое выражение принимает значение «истина».

1. **Язык T-SQL. Обработка ошибок в конструкциях try-catch.**

**Функция RAISERROR.**

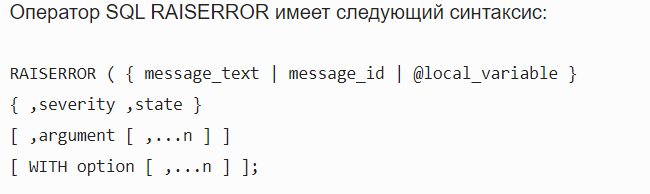
Для обработки ошибок выполнения в сценарии T-SQL предусмотрена конструкция, состоящая из двух блоков: TRY и CATCH. Блок TRY содержит код T-SQL, в котором могут возникнуть ошибки, а блок CATCH – код, предназначенный для обработки ошибок.

Ошибка, возникающая в охраняемом коде, приводит к передаче управления в блок обработки ошибок:



RAISERROR – это системная функция, позволяющая создавать сообщение об ошибке и возвращать его как сообщение об ошибке сервера вызывающему приложению.

С помощью RAISERROR мы можем сослаться на уже определенное сообщение, которое находится в системном представлении sys.messages, либо самостоятельно динамически создавать сообщение.



**message\_text**- сообщение, которое вы хотите показать при ошибке. Замечание: вы можете добавлять пользовательские сообщения для вывода информации об ошибке. Смотрите следующий раздел статьи.  
**message\_id**- id сообщения об ошибке. Если вы хотите вывести пользовательское сообщение, вы можете определить этот идентификатор.

severity - серьезность ошибки. Тип данных переменной severity - smallint, значения находятся в диапазоне от 0 до 25. Допустимыми значениями серьезности ошибки являются:

0-10- информационные сообщения

11-18- ошибки

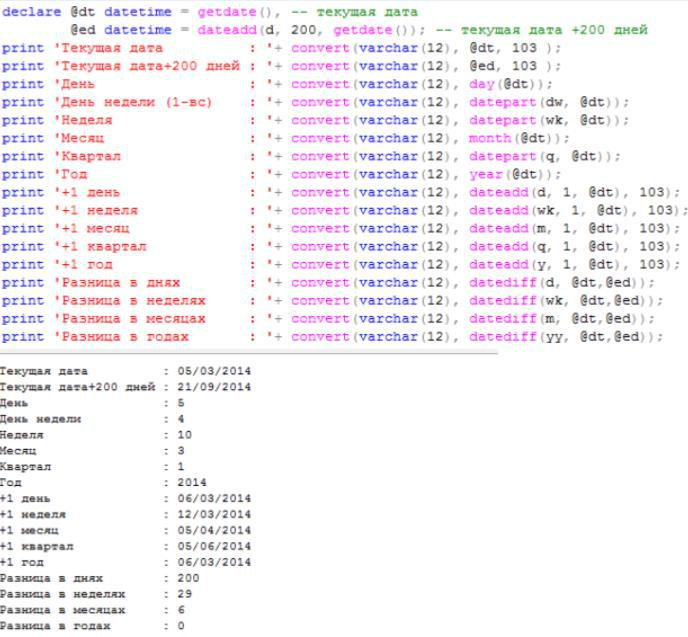
19-25- фатальные ошибки

**state**- уникальное идентификационное число, которое может использоваться для раздела кода, вызывающего ошибку.

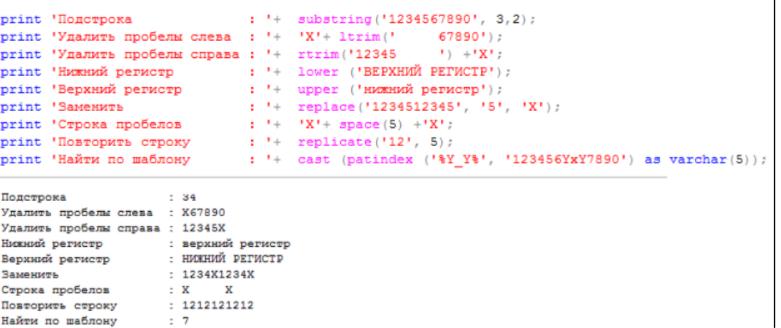


1. **Язык T-SQL. Встроенные функции работы с датами**

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



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



1. **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.

SELECT ROUND(1342.345, 2) -- 1342.350

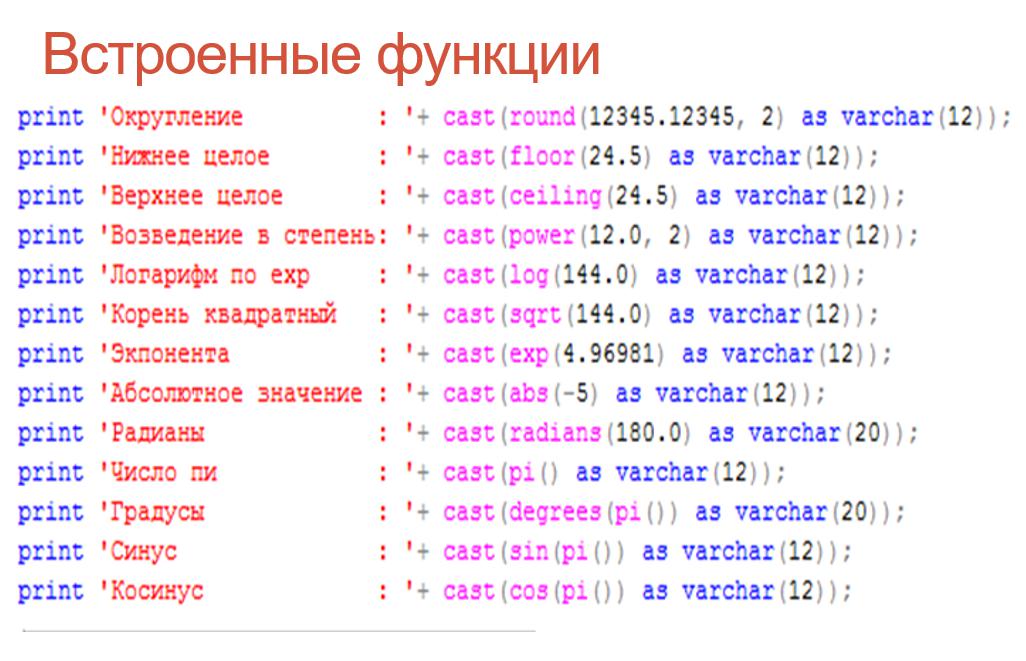
SELECT ROUND(1342.345, -2) -- 1300.000

**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.

|  |  |  |
| --- | --- | --- |
| SELECT ISNUMERIC(1342.345) | | -- 1 |
| SELECT ISNUMERIC('SQL') | | -- 0 |
| **SQUARE**: возводит число в квадрат. | |  |
| SELECT SQUARE(5) | -- 25 |  |
| **SQRT**: получает квадратный корень числа. | | |
| SELECT SQRT(225) | -- 15 |  |

**RAND**: генерирует случайное число с плавающей точкой в диапазоне от 0 до 1.

|  |  |
| --- | --- |
| SELECT RAND() | -- 0.707365088352935 |
| SELECT RAND() | -- 0.173808327956812 |



1. **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. Механизм, позволяющий обрабатывать отдельные строки, полученные в результате select-запроса. Область памяти сервера, предназначенная для хранения и обработки результата select-запроса.

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

* помощью системной функции @@FETCH\_STATUS.

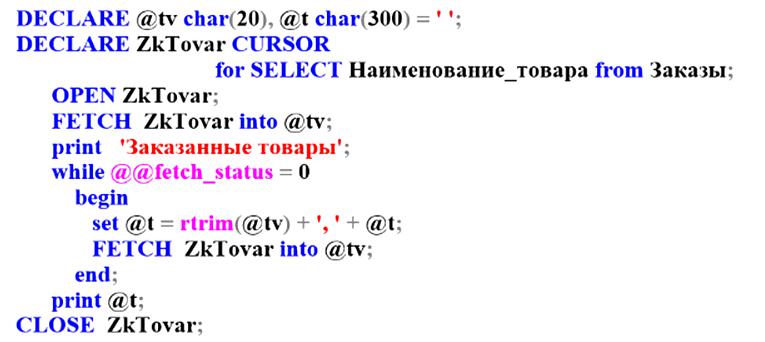
Оператор FETCH считывает одну строку из результирующего набора и продвигает указатель на следующую строку. Количество переменных в списке после ключевого слова INTO должно быть равно количеству столбцов результирующего набора, а порядок их должен соответствовать порядку перечисления столбцов в SELECT-списке.

После выполнения FETCH проверяется значение функции **@@fetch\_status**, которая возвращает значение 0, если оператор FETCH выполнен успешно; −1, если достигнут конец результирующего набора и строка не считывается; −2, если выбранная строка отсутствует в БД. В зависимости от полученного результата цикл продолжается и считывается следующая строка, или цикл заканчивается.

4.Курсор закрывается оператором CLOSE.

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

Оператор FETCH считывает одну строку из результирующего набора и продвигает указатель на следующую строку. Количество переменных в списке после ключевого слова INTO должно быть равно количеству столбцов результирующего набора, а порядок их должен соответствовать порядку перечисления столбцов в SELECT-списке.



1. **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.

1. **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.

SROLL



|  |
| --- |
|  |

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).

1. **Курсоры. Функция fetch\_status**

После выполнения FETCH проверяется значение функции **@@fetch\_status**.

* зависимости от полученного результата цикл продолжается и считывается следующая строка, или цикл заканчивается.

@@FETCH\_STATUS

0 – успешная выборка,

-1 – вышли за диапазон таблицы,

-2 – запись удалена после открытия курсора

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

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

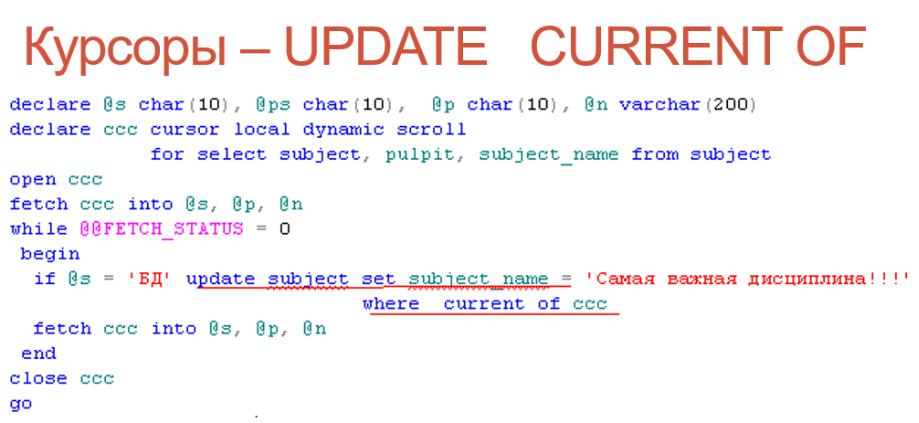
close tmpcursor

print @str

select \* from #temp1

drop table #temp1

go

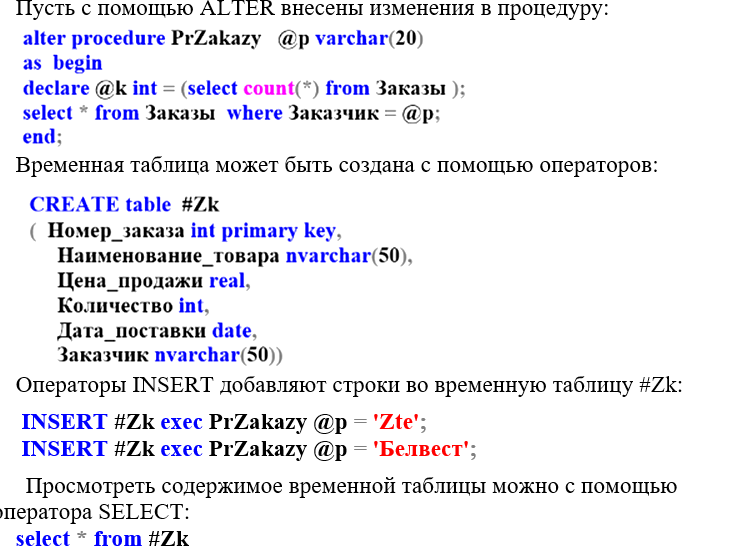


1. **Хранимые процедуры. Создание, изменение и удаление хранимых процедур. Вызов хранимых процедур. Передача параметров**

Хранимая процедура – это объект БД, представляющий собой поименованный код T-SQL, хранящийся в откомпилированном виде. 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.



В хранимых процедурах допускается применение основных DDL, DML и TCL-операторов, конструкций TRY/CATCH, курсоров, временных таблиц.

1. **Хранимые процедуры. DML в процедурах. Входные и выходные параметры.**

Допускается применение :

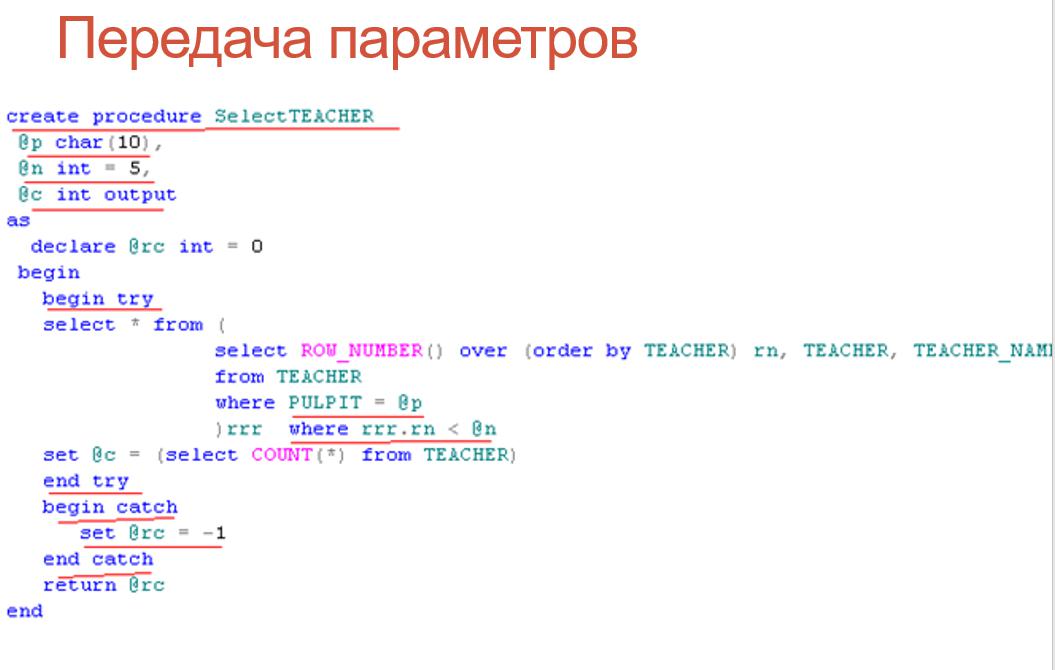
Основных DDL, DML и TCL-операторов Конструкций TRY/CATCH

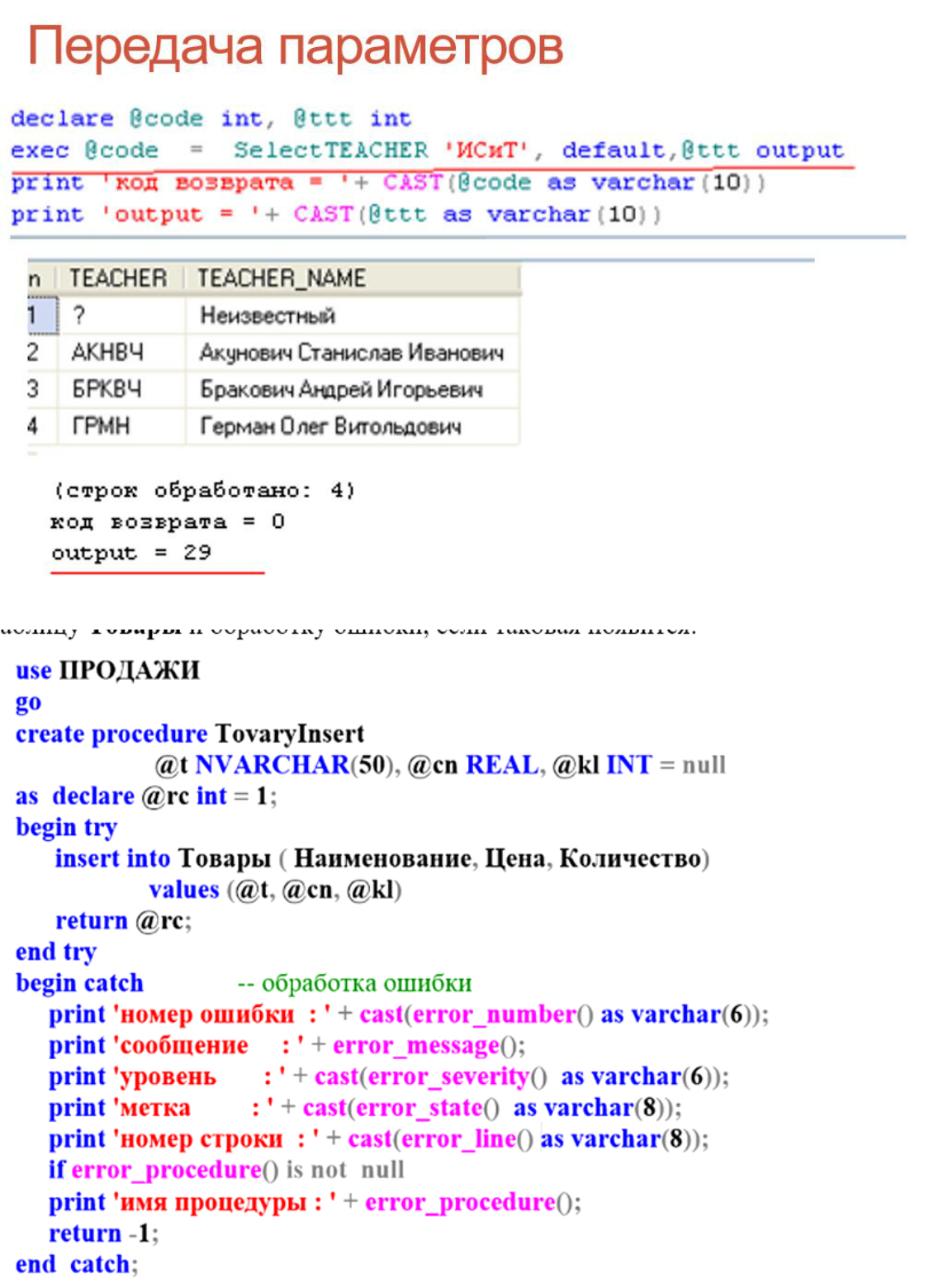
Курсоров

Временных таблиц

Не допускается применение : CREATE or ALTER FUNCTION CREATE or ALTER TRIGGER CREATE or ALTER PROCEDURE CREATE or ALTER VIEW

USE databasename

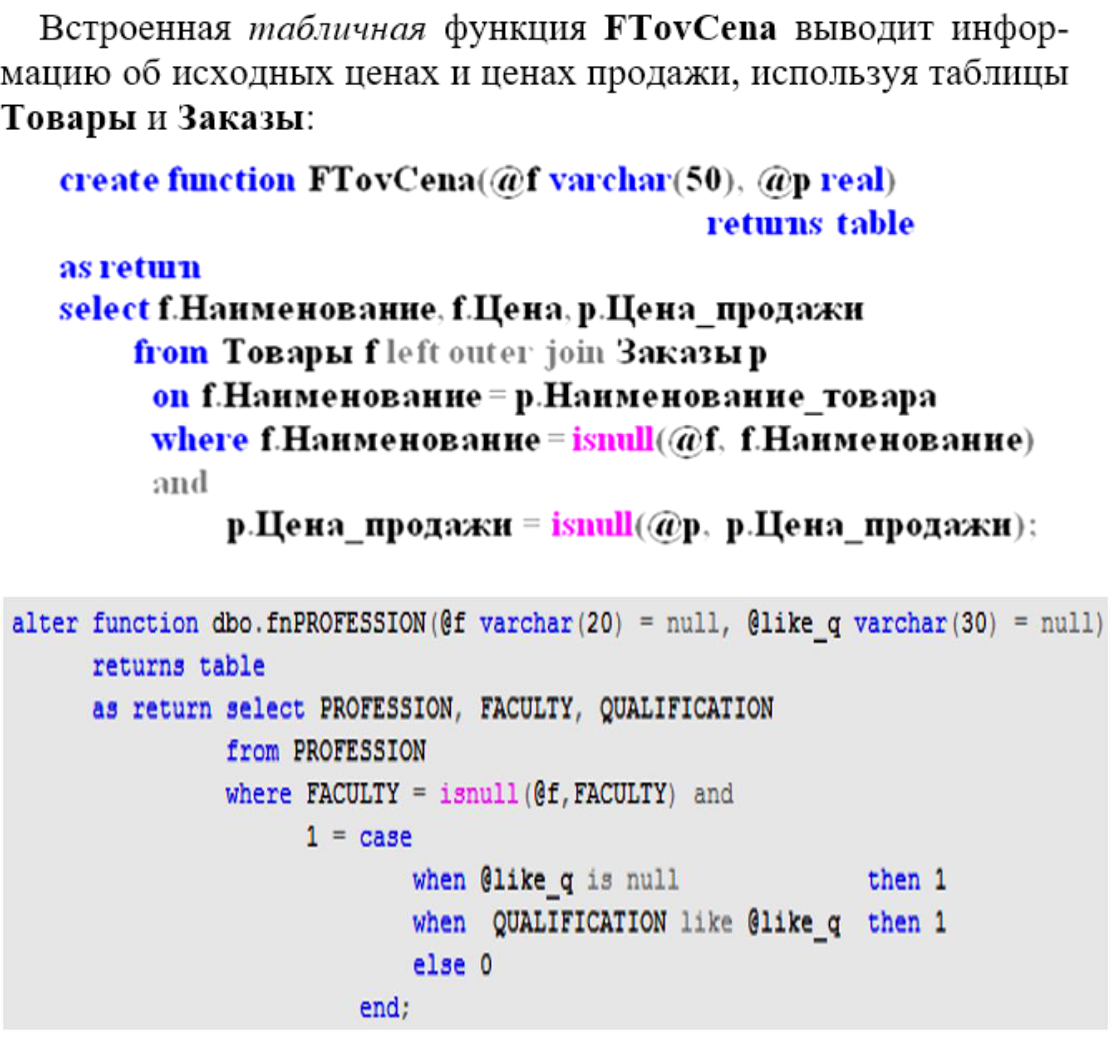




1. **Функции, определенные пользователем. Виды функций. Обращение к функции. Передача параметров.**

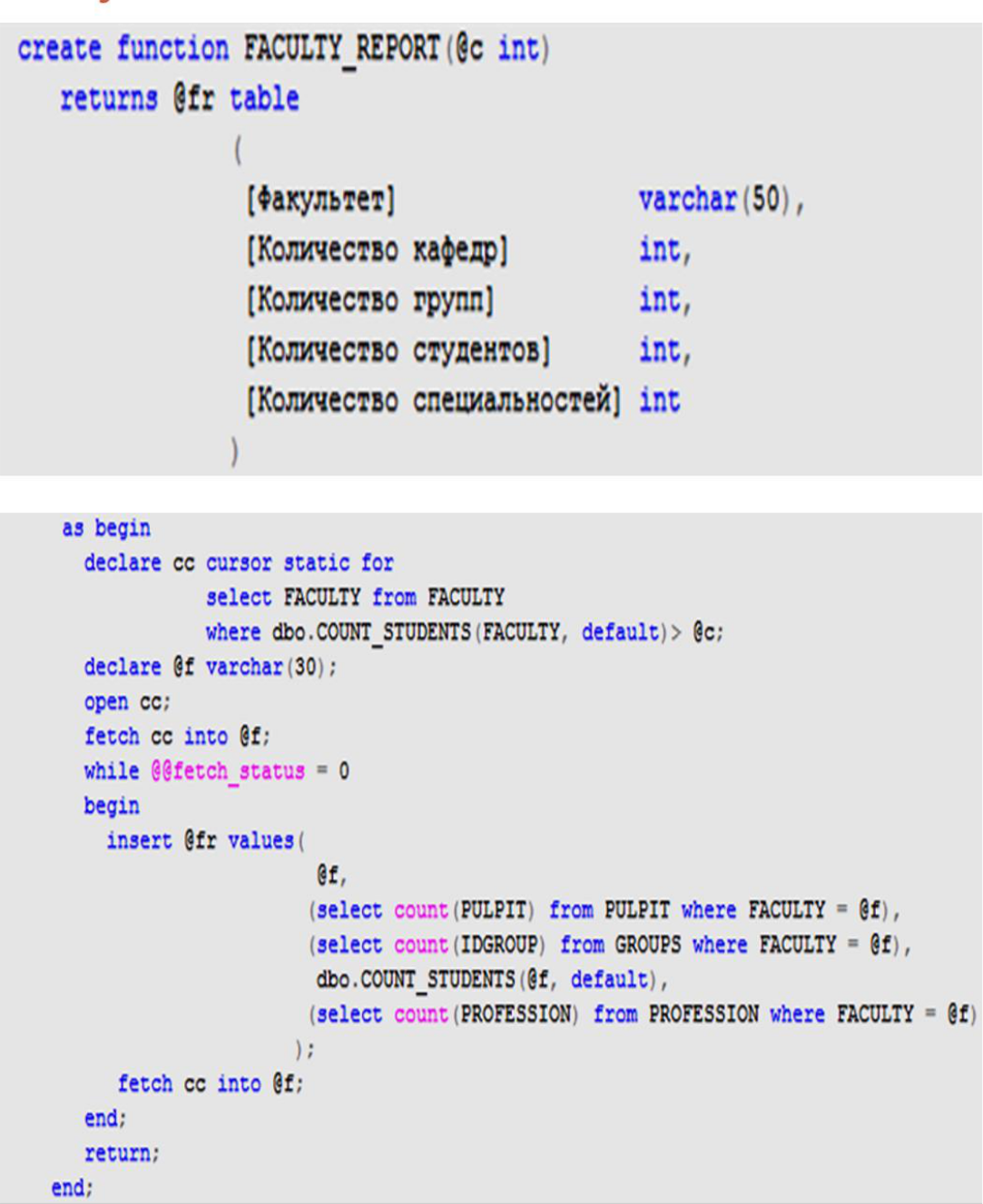
Функция – это объект БД, представляющий собой поименованный код T-SQL. Для создания, удаления и изменения функций необходимо применять CREATE, DROP и ALTER соответственно. Отличие функций от хранимых процедур в ограничениях, накладываемых на код функции, в форме представления результата работы, а также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

* 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. + Multistatement.
* зависимости от структуры кода, различают встроенные функции и многооператорные табличные функции.



* 1. **Функции, определенные пользователем. Создание, изменение и удаление функций. DML в функциях.**
* DROP FUCTION
* ALTER FUNCTION

Можно ли делать дмл в функциях? Можно, только не с таблицей.



1. **План запроса. Этапы обработки SELECT запроса. Понятие стоимости запроса. Понятия селективности и плотности. Оптимизация**

**запросов**

План запроса - это последовательность операций и доступа к данным, которую оптимизатор запроса выбирает для выполнения SELECT запроса в базе данных. План запроса определяет, какие индексы и таблицы будут использоваться, какие операции будут выполняться и в какой последовательности.

Этапы обработки SELECT запроса включают:

Анализ синтаксиса: База данных проверяет синтаксис запроса и проверяет его правильность. Если запрос содержит ошибки, он может быть отклонен.

Построение плана запроса: Оптимизатор запроса анализирует структуру запроса, доступные индексы и статистику данных, чтобы создать оптимальный план выполнения запроса. Оптимизатор учитывает стоимость операций доступа к данным и выбирает наиболее эффективный план выполнения.

Выполнение плана запроса: База данных выполняет операции, указанные в плане запроса. Это может включать чтение данных из таблиц, объединение данных из разных таблиц, фильтрацию, сортировку и другие операции.

Возвращение результирующего набора: После выполнения операций база данных возвращает результирующий набор данных, который соответствует условиям запроса. Результирующий набор может быть возвращен клиентскому приложению или использован в дальнейших операциях.

Стоимость запроса - это оценка ресурсов (время выполнения, использование памяти, объем операций ввода-вывода и т. д.), которые будут использоваться для выполнения запроса. Оптимизатор запроса старается выбрать план с наименьшей стоимостью для достижения наилучшей производительности.

Селективность - это мера количества выбранных строк в соответствии с предикатами или условиями запроса. Она определяет, насколько узкий или широкий набор данных будет возвращен запросом. Высокая селективность означает, что запрос вернет небольшое количество строк, а низкая селективность означает, что запрос вернет большое количество строк.

Плотность - это мера уникальности значений в столбце или индексе. Она определяет, насколько равномерно распределены значения в столбце или индексе. Высокая плотность означает, что значения в столбце или индексе практически уникальны, а низкая плотность означает, что значения повторяются или имеют мало уникальных значений.

Оптимизация запросов - это процесс анализа и модификации запросов с целью улучшения их производительности. Оптимизация запросов может включать выбор оптимальных индексов, переписывание запросов, использование подзапросов или объединений, устранение избыточности и другие техники для улучшения скорости выполнения запросов и снижения нагрузки на базу данных. Оптимизация запросов позволяет улучшить производительность базы данных и сделать запросы более эффективными.

**46. Индексы. Назначение и применение индексов**

**Индекс** – это объект базы данных, позволяющий **ускорить поиск** в определенной таблице, так как при этом данные организуются в виде сбалансированного бинарного дерева поиска.

Как и любой другой объект базы данных, индекс может быть создан с помощью оператора CREATE, модифицирован с помощью ALTER и удален с помощью оператора DROP. Для одной таблицы возможно построение нескольких индексов.

Индексы бывают кластеризованные, некластеризованные, уникальные, неуникальные и др.

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
* Обычно *кластеризованные* индексы создаются автоматически при создании таблицы если в ней присутствует первичный ключ (ограничение

PRIMARY KEY)

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

Виды индексов:

* + Кластеризованные и некластеризованные
  + Уникальные и неуникальные
  + Простые и составные
  + XML-индексы
  + Пространственные индексы

Обычно *кластеризованные* индексы создаются автоматически при создании таблицы если в ней присутствует первичный ключ (ограничение PRIMARY KEY).

Кластеризованные индексы физически упорядочены в соответствии со значениями индексируемых столбцов. В таблице может быть только один кластеризованный индекс.

Кластеризованный индекс хранит реальные строки данных в листьях индекса.

С помощью системной процедуры SP\_HELPINDEX можно получить перечень индексов, связанных с заданной таблицей:



Если кластеризованный индекс создается для столбца, содержащего повторяющиеся значения, СУБД принудительно добавляет четырехбайтовый идентификатор к строкам, содержащим дубликаты значений.

(на лекциях дохера раз спрашивала)

Некластеризованные

В отличие от кластеризованного индекса, листья некластеризованного индекса содержат только те столбцы (*ключевые*), по которым определен данный индекс, а также содержит указатель на строки с реальными данными в таблице. Это означает, что системе подзапросов необходима дополнительная операция для обнаружения и получения требуемых данных. Содержание указателя на данные зависит от способа хранения данных: кластеризованная таблица или куча(Таблица не имеющая кластеризованного индекса называется кучей.). Если указатель ссылается на кластеризованную таблицу, то он ведет к кластеризованному индексу, используя который можно найти реальные данные. Если указатель ссылается на кучу, то он ведет к конкретному идентификатору строки с данными. Некластеризованные индексы не могут быть отсортированы в отличие от кластеризованных, однако вы можете создать более одного некластеризованного индекса на таблице или представлении, вплоть до 999. Это не означает, что вы должны создавать как можно больше индексов. Индексы могут как улучшить, так и ухудшить производительность системы. В дополнение к возможности создать несколько некластеризованных индексов, вы можете также включить дополнительные столбцы (*included column*) в свой индекс: на листьях индекса будет храниться не только значение самих индексированных столбцов, но и значения этих не индексированных дополнительных столбцов. Этот подход позволит вам обойти некоторые ограничения, наложенные на индекс. К примеру, вы можете включить неидексируемый столбец или обойти ограничение на длину индекса (900 байт в большинстве случаев).

Можно создавать индексы по нескольким столбцам – такие индексы называются *составными*. Можно создать составной неуникальный, некластеризованный индекс **#EX\_NONCLU** по двум столбцам **TKEY** и **CC** таблицы **#EX** с помощью оператора:

**CREATE index #EX\_NONCLU on #EX(TKEY, CC)**

Этот индекс не применяется оптимизатором ни при фильтрации, ни при сортировке строк таблицы #EX, в чем можно убедиться, посмотрев планы следующих запросов:

**SELECT \* from #EX where TKEY > 1500 and CC < 4500;**

**SELECT \* from #EX order by TKEY, CC**

Но, если хотя бы одно из индексируемых значений зафиксировать (задать одно значение), то оптимизатор применит индекс. Это можно проверить, выполнив запрос:

**SELECT \* from #EX where TKEY = 556 and CC > 3**

Правила:

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

*Некластеризованный индекс покрытия* запроса позволяет включить в состав индексной строки значения одного или нескольких неиндексируемых столбцов. Например, индекс покрытия **#EX\_TKEY\_X** включаетзначения столбца **CC** (ключевое слово INCLUDE):

**CREATE index #EX\_TKEY\_X on #EX(TKEY) INCLUDE (CC)**

Чтобы оценить процедуры поиска можно посмотреть планы выполнения запроса без применения индексов и с использованием индекса покрытия.

**SELECT CC from #EX where TKEY>15000**

Если запросы основаны на WHERE-фильтрации строк, то может быть эффективным применение *фильтруемых некластеризованных* индексов.

**CREATE index #EX\_WHERE on #EX(TKEY) where (TKEY>=15000 and**

**TKEY < 20000);**

Здесь фильтруемый индекс создается только для строк таблицы **#EX,** которыеудовлетворяют логическому условию. Стоимость запросов уменьшится.

**Уникальный индекс**

Такой индекс обеспечивает уникальность каждого значения в индексируемом столбце. Если индекс составной, то уникальность распространяется на все столбцы индекса, но не на каждый отдельный столбец. К примеру, если вы создадите уникальных индекс на столбцах ИМЯ и ФАМИЛИЯ, то полное имя должно быть уникально, но отдельно возможны дубли в имени или фамилии.

Применение индексов:

Индекс занимает определенный объем дискового пространства. Как было отмечено ранее индексы могут улучить производительность системы, т.к. они обеспечивают подсистему запросов быстрым путем для нахождения данных. Однако, вы должны также принять во внимание то, как часто вы собираетесь вставлять, обновлять или удалять данные. Когда вы изменяете данные, то индексы должны также быть изменены, чтобы отразить соответствующие действия над данными, что может значительно снизить производительность системы.(образуется фрагментация)

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

Операции добавления и изменения строк базы данных могут повлечь образование неиспользуемых фрагментов в области памяти индекса. Процесс образования неиспользуемых фрагментов памяти называется фрагментацией.

Фрагментация индексов снижает эффект от их применения.

Для ликвидации фрагментации используются:

Для избавления от фрагментации индекса предусмотрены две специальные операции: реорганизация и перестройка индекса.

Реорганизация (REORGANIZE) выполняется быстро, но после нее фрагментация будет убрана только на самом нижнем уровне. Пусть выполнена реорганизация с помощью оператора ALTER для индекса #EX\_TKEY.

ALTER index #EX\_TKEY on #EX reorganize;

Тогда выполнение соответствующего запроса покажет, что уровень фрагментации значительно снизился, но не до конца.

Операция перестройки (REBUILD) затрагивает все узлы дерева, поэтому после ее выполнения степень фрагментации равна нулю. Пусть выполнена перестройка с помощью оператора ALTER для индекса #EX\_TKEY в режиме

OFFLINE.

ALTER index #EX\_TKEY on #EX rebuild with (online = off);

Выполнением запроса о фрагментации можно оценить ее уровень.

Уровнем фрагментации можно в некоторой степени управлять, если при создании или изменении индекса использовать параметры FILLFACTOR и PAD\_INDEX.

Параметр FILLFACTOR указывает процент заполнения индексных страниц нижнего уровня.

Пусть индекс пересоздан со значением параметра FILLFACTOR равным 65:

**DROP index #EX\_TKEY on #EX; CREATE index #EX\_TKEY on #EX(TKEY)**

**with (fillfactor = 65);**

После добавления строк в таблицу **#EX** можно оценить уровень фрагментации:

**INSERT top(50)percent INTO #EX(TKEY, TF)**

**SELECT TKEY, TF FROM #EX;**

**SELECT name [Индекс], avg\_fragmentation\_in\_percent [Фрагментация (%)]**

**FROM sys.dm\_db\_index\_physical\_stats(DB\_ID(N'TEMPDB'), OBJECT\_ID(N'#EX'), NULL, NULL, NULL) ss JOIN sys.indexes ii**

**ON ss.object\_id = ii.object\_id and ss.index\_id =**

**ii.index\_id**

**WHERE name is not null;**

PAD\_INDEX : используется для применения процента свободного пространства, указанного FillFactor, к страницам промежуточного уровня индекса во время создания индекса.

DISABLE - отключение индекса

Отключенный индекс недоступен, пока он не будет снова включен

reorganize - быстро, но фрагментация убирается только на нижнем уровне

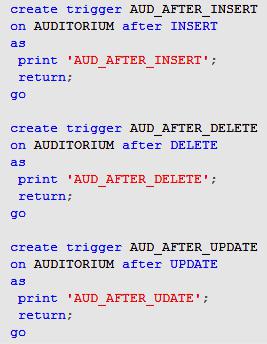
rebuild - затрагивает все узлы дерева, поэтому фрагментация = 0

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

**Триггер** – это особый вид хранимой процедуры, предназначенной для обработки событий в БД. Поддерживается два типа триггеров: *DDL-триггеры* и *DML-триггеры*. Для каждого типа определено свое семейство событий, обработку которых триггер этого типа может выполнять.

**DML(IUD)-**триггеры бывают двух типов: AFTER-триггеры и INSTEAD OF-триггеры. Триггеры типа AFTER исполняются *после* выполнения оператора, вызвавшего соответствующее событие. При этом создаются автоматически две псевдотаблицы INSERTED и DELETED.

Триггер типа INSTEAD OF выполняется *вместо* оператора, вызвавшего соответствующее событие. Выполнение INSTEAD OF триггера предшествует проверке установленных для таблицы ограничений целостности.



•

Две специальные виртуальные таблицы

deleted — содержит копии строк, удаленных из таблицы inserted — содержит копии строк, вставленных в таблицу

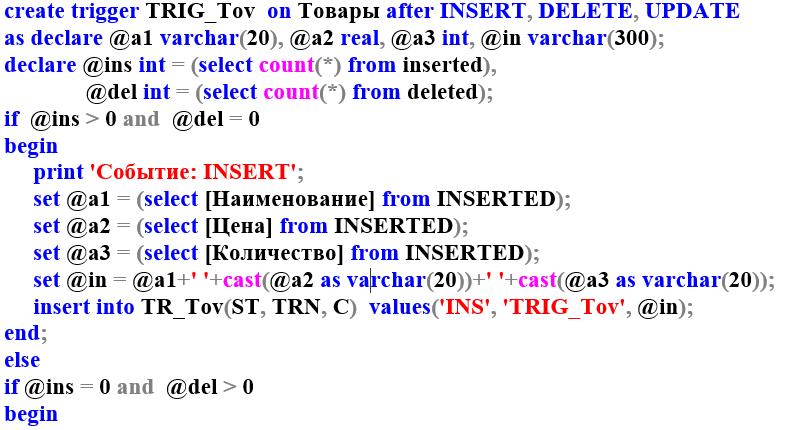
Структура этих таблиц эквивалентна структуре таблицы, для которой определен триггер.

* 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.

Ссы́лочная це́лостность — корректность значений внешних ключей реляционной базы данных.



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

Триггеры уровня оператора. Выполняются по одному разу для каждого оператора. Выполняются вместо операции - сама операция не выполняется.

Триггер типа INSTEAD OF выполняется *вместо* оператора, вызвавшего соответствующее событие. Выполнение INSTEAD OF триггера предшествует проверке установленных для таблицы ограничений целостности.

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.

|  |
| --- |
|  |

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

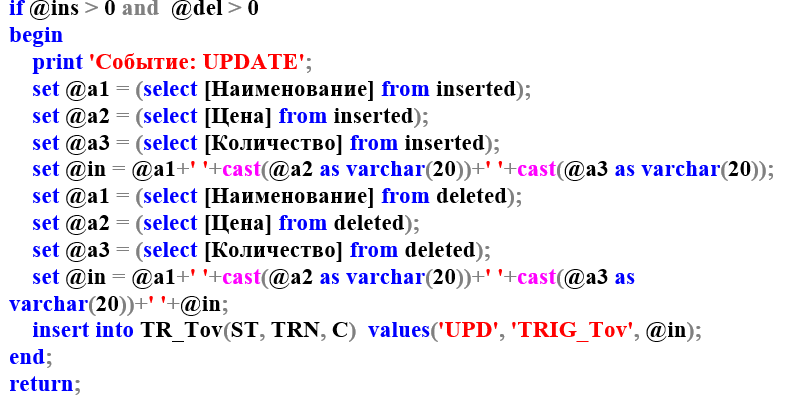
Две специальные виртуальные таблицы:

* 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.



1. **Транзакции. Явные и неявные транзакции**

**Транзакция** − это механизм базы данных, позволяющий таким образом объединять несколько операторов, изменяющих базу данных, чтобы все выполнились или все не выполнились.

Бывают явные и неявные.

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

Режим *неявной транзакции* может быть включен для текущего соединения с сервером БД с помощью специальной инструкции:

**SET IMPLICIT\_TRANSACTIONS ON**

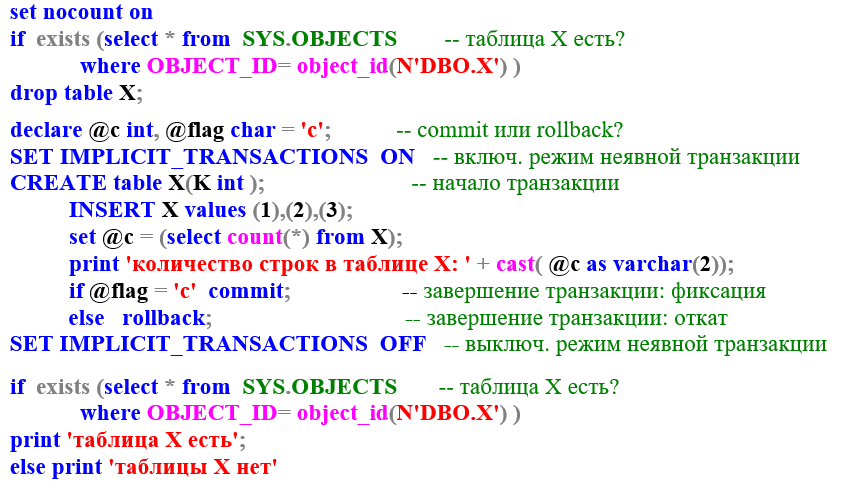
Обратное переключение осуществляется с использованием ключевого слова OFF вместо ON.

Неявная транзакция начинается, если выполняется один из следующих операторов:

CREATE, DROP; ALTER TABLE; INSERT, DELETE, UPDATE, SELECT, TRUNCATE TABLE; OPEN, FETCH; GRANT (выдача разрешений), REVOKE (запрещение разрешений).

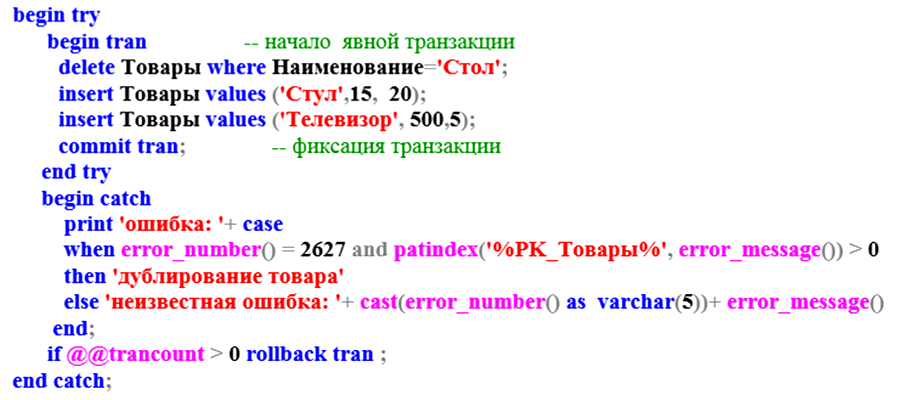
Неявная транзакция продолжается до тех пор, пока не будет выполнен оператор фиксации (COMMIT) или оператор отката (ROLLBACK) транзакции.

В примере ниже неявная транзакция стартует при выполнении оператора CREATE TABLE и завершается фиксацией изменений с помощью оператора COMMIT.



Явная транзакция — группа инструкций, начало и конец которой обозначаются инструкциями:

* BEGIN TRANSACTION
* COMMIT
* ROLLBACK



1. **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).

1. **Транзакции. Уровни изолированности транзакций**

Уровень изолированности 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

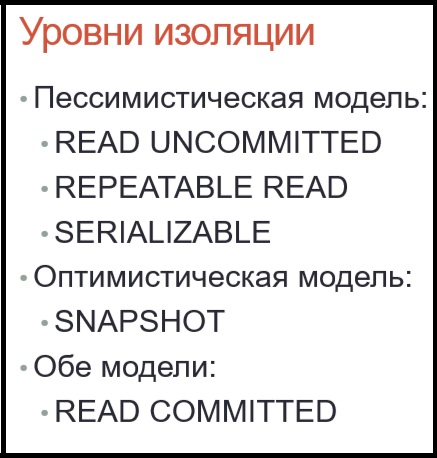
-Устанавливает блокировку на всю область данных, считываемых соответствующей транзакцией

-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 обеспечивает 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.

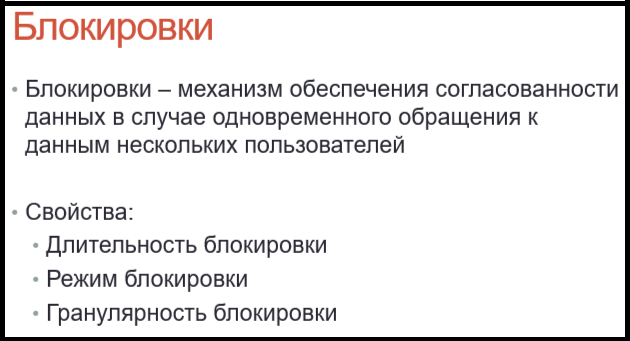


1. **Транзакции. Вложенные транзакции. Функция TRANCOUNT**

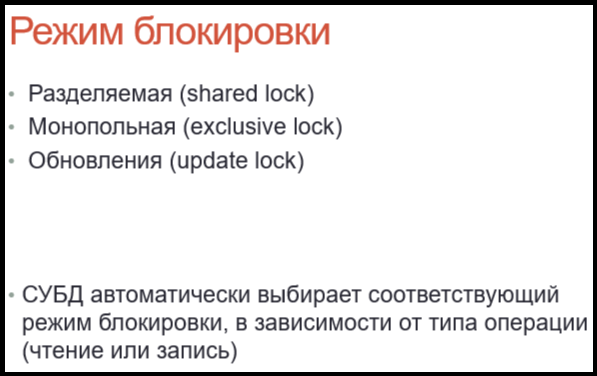
Транзакция, выполняющаяся в рамках другой транзакции, называется вложенной.

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 @@TRANCOUNT.
  1. **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*.

|  |
| --- |
|  |

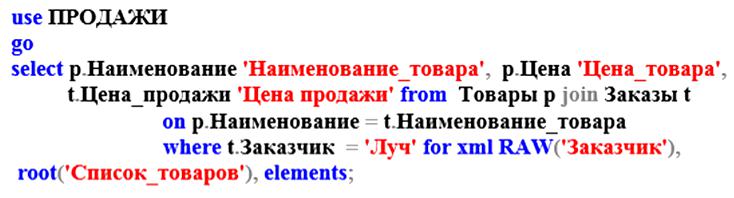
Взаимоблокировка (deadlock) — это особая проблема одновременного конкурентного доступа, в которой две транзакции блокируют друг друга

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

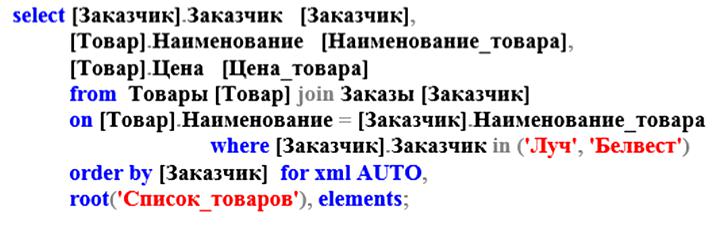
XML (Extensible Markup Language) – расширяемый язык разметки. XML-формат часто используется для обмена данными между компонентами информационных систем. При работе с базами данных важными являются две задачи: преобразование табличных данных в XML-структуры и преобразование XML-структур в строки реляционной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

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

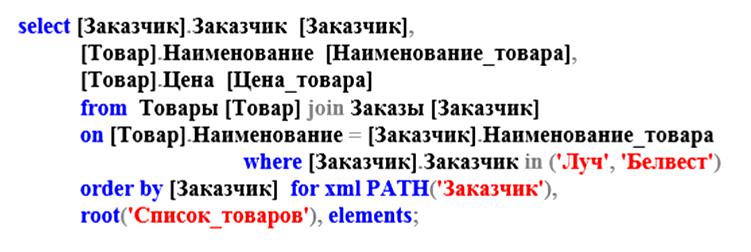
* режиме RAW в результате SELECT-запроса создается XML-фрагмент, состоящий из последовательности элементов с именем row. Каждый элемент row соответствует строке результирующего набора, имена его атрибутов совпадают с именами столбцов результирующего набора, а значения атрибутов равны их значениям.



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. При использовании режима PATH каждый столбец конфигурируется независимо с помощью псевдонима этого столбца.

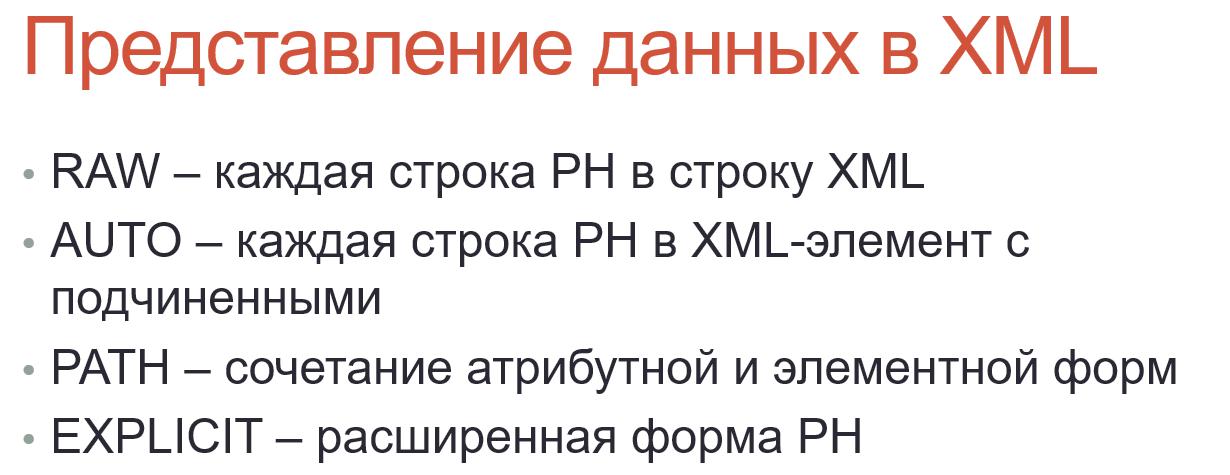


Директивы:

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

ELEMENTS - она всё бахает как элементы. Атрибутов нет.

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



1. **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).

* 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 )

1. **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.

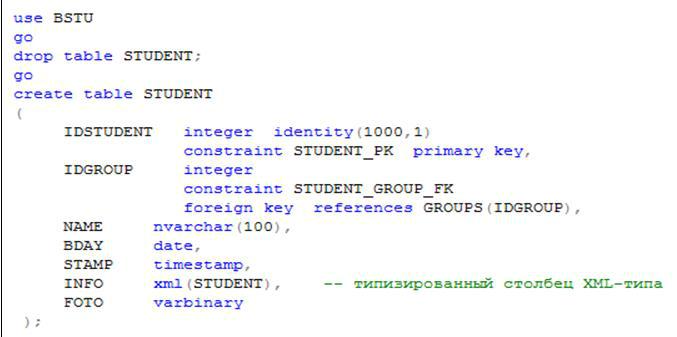
Для хранения документов XML-Schema в БД MSS предусмотрен специальный объект – XML SCHEMA COLLECTION. Каждый такой объект может содержать один или более XML-SCHEMA-документов.

На рис. 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. Такой столбец называется типизированным столбцом. Типизированный XML-столбец должен удовлетворять хотя бы одной схеме из связанной с ним коллекции.



1. **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

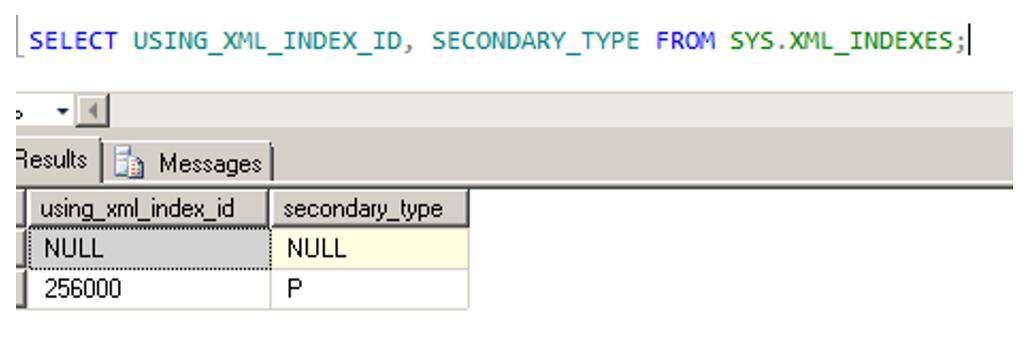
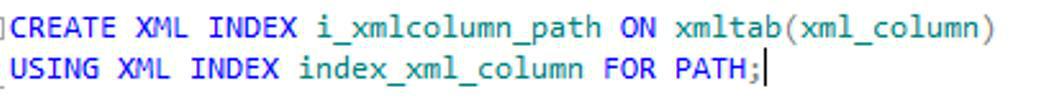


Три типа вторичных типа XML-индексов:

* 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

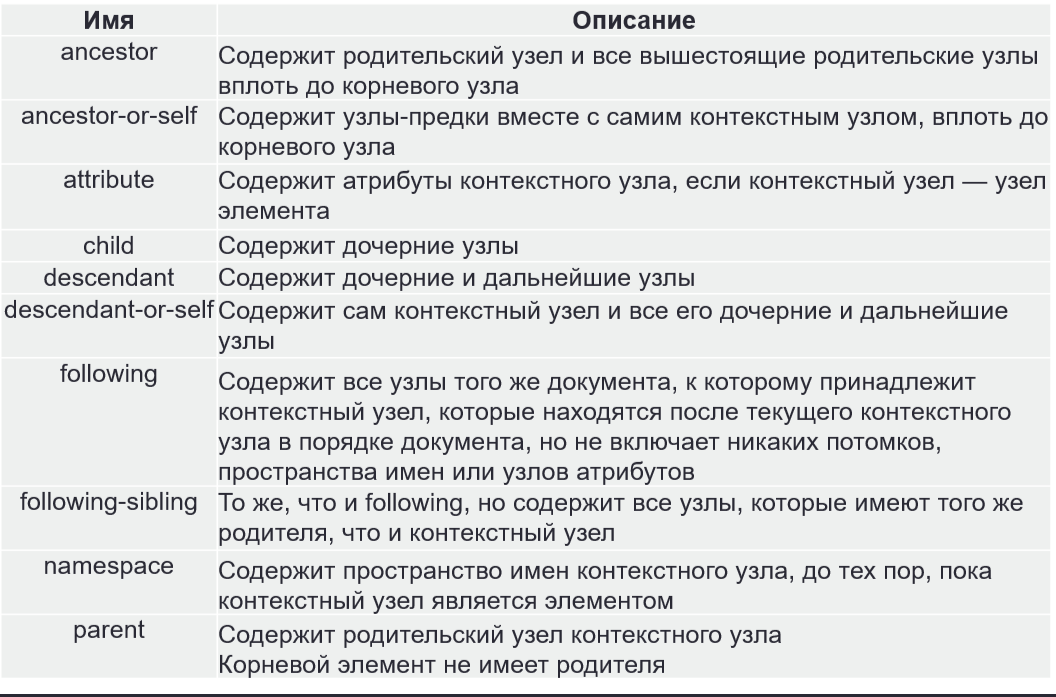
* XML Path Language — язык запросов к элементам XML-документа Язык запросов Xquery
* XQuery — язык запросов, разработанный для обработки данных в формате XML

**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 "]

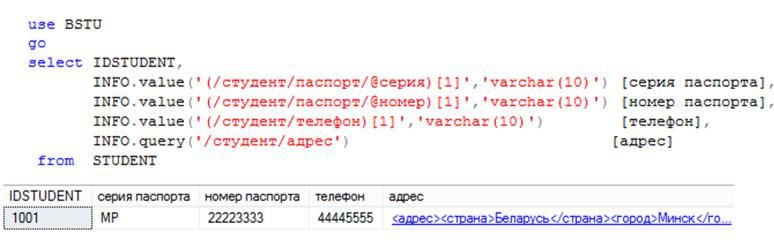
**Оси XPath:**



**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: 1) XML-тип является объектным типом (имеет методы и свойства);

* 1. в первых двух случаях применения метода **value** извлекаются значения атрибутов **серия** и **номер**;
  2. в третьем случае применения **value** извлекается текстовое значение элемента **телефон**;

1. 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;
2. второй параметр метода **value** указывает на тип данных, к которому должно быть преобразовано выбранное значение;
3. метод **query**, применяемый в четвертом элементе SELECT-списка, позволяет выбрать XML-фрагмент.