1. *Основные понятия теории баз данных: база данных, система управления базами данных, основные требования к информации в БД, модели данных, логическая схема БД, основная терминология реляционных баз данных.*

База данных - совокупность связанных данных, организованных по определенным правилам, предусматривающим общие принципы описания, хранения и манипулирования, независимая от прикладных программ.

Система управления базами данных (СУБД) — совокупность программных средств, обеспечивающих управление созданием и использованием баз данных.

Требования инфы к 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

**Модели данных**:

Иерархическая модель данных. Иерархическая модель данных, имеет иерархическую структуру, т.е. каж­дый из элементов связан только с одним стоящим выше элементом, но в то же время на него могут ссылаться один или несколь­ко стоящих ниже элементов.

Сетевая модель данных. Единственное отличие между иерархической и сетевой моделями данных заключается в том, что в последней каждый элемент данных (узел) может быть связан с любым другим элементом (узлом).

Реляционная модель данных. Основная идея реляционной модели данных заключается в том, чтобы представить любой набор данных в виде двумерного массива — таблицы.

**Логическая схема** базы данных является отображением полученной информационно-логической модели исследуемой предметной области.

**Основные понятия** реляционных баз данных: тип данных, домен, атрибут, кортеж, отношение, первичный ключ.

Определения:

- домен: множество допустимых значений;

- таблица: отношение;

- атрибут: имя столбца таблицы (имя домена);

- заголовок таблицы: множество всех атрибутов;

- кортеж: элемент отношения или строка таблицы

1. *Язык SQL.*

SQL - специализированный язык, предназначенный для написания запросов к реляционной БД.

Группы операторов:

**DDL** - Data Definition Language - язык определения данных

* CREATE
* ALTER
* DROP

**DML** - Data Manipulation Language - язык манипулирования данными

* SELECT
* INSERT
* DELETE
* UPDATE

**TCL** - Transaction Control Language - язык управления транзакциями

* BEGIN TRAN
* SAVE TRAN
* COMMIT TRAN
* ROLLBACK TRAN

**DCL** - Data Control Language - язык управления данными

* GRANT
* REVOKE
* DENY

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

|  |  |
| --- | --- |
| Системная  база данных | Назначение |
| master | Хранит все системные данные Database Engine, а также информацию о других БД. |
| msdb | Используется службами SQL Server Agent (выполнение заданий по расписанию), Database Mail (формирование уведомлений по электронной почте), а также хранит информацию о резервном копировании БД. |
| tempdb | Пространство для временных объектов Database Engine и пользовательских временных таблиц. База данных пересоздается при каждой перезагрузке |
| model | Шаблон, используемый при создании всех БД, управляемых экземпляром Database Engine. |
| resource | БД, используемая только для чтения. Содержит системные объекты экземпляра Database Engine. Файлы БД являются скрытыми и не отображаются в MSMS. |

1. Структура файла базы данных.

Первичный файл (.mdf),

Вторичный файл (.ndf).

Сначала запись идет в журнал транзакций, потом в файл данных. Файл транзакций используется для восстановления данных в БД.

SQL Server имеют три типа файлов.

|  |  |
| --- | --- |
| Файл | Description |
| Первичная | Первичный файл данных содержит сведения, необходимые для запуска базы данных, и ссылки на другие файлы в базе данных. Данные и объекты пользователя могут храниться в данном файле или во вторичном файле данных. В каждой базе данных имеется один первичный файл данных. Для имени первичного файла данных рекомендуется расширение MDF. |
| Вторичная | Вторичные файлы данных не являются обязательными; это пользовательские файлы, в которых хранятся данные пользователя. Вторичные файлы могут быть использованы для распределения данных на несколько дисков, в этом случае каждый файл записывается на отдельный диск. Кроме того, если размер базы данных превышает максимальный размер для одного файла Windows, можно использовать вторичные файлы данных, таким образом база данных сможет расти дальше.   Для имени вторичного файла данных рекомендуется расширение NDF. |
| Журнал транзакций | Файлы журнала транзакций содержат сведения, используемые для восстановления базы данных. Для каждой базы данных должен существовать хотя бы один файл журнала. Для файлов журнала транзакций рекомендуется расширение LDF. |

У каждой базы данных есть первичная файловая группа. Эта файловая группа содержит первичный файл данных и все вторичные файлы, не входящие в другие файловые группы. Пользовательские файловые группы могут создаваться для удобства администрирования, распределения и размещения данных.

1. Дисковое хранение файлов базы данных. Страницы, экстенты.

Основной единицей хранилища данных является страница. Размер страницы постоянен и составляет *8 Кбайт*.

Страница состоит из:

* заголовка страницы;
* пространства для данных;
* таблицы смещений строк.

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.

Экстент - физическая единица дискового пространства, используемая для выделения памяти для таблиц и индексов.

Размер экстента составляет восемь последовательно расположенных страниц или 64 Кбайт.

Существует два следующих типа экстентов:

* однородные экстенты - содержат данные одной таблицы или индекса;
* смешанные экстенты - могут содержать данные до восьми таблиц или индексов.

1. Создание файлов базы данных. Файловые группы. Создание таблиц в файловой группе.

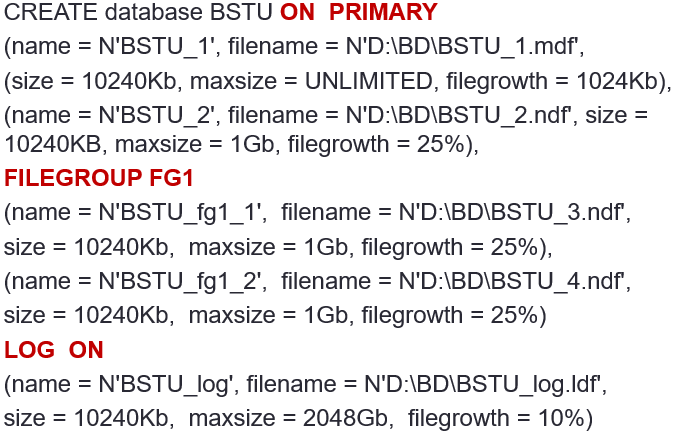
Для создания пользовательской БД применяется оператор CREATE DATABASE, который должен выполняться в контексте системной БД MASTER.

Все файлы БД, кроме файлов журнала транзакций, распределены по файловым группам.

Файловые группы – это поименованный набор файлов БД.

* Первичные файловые группы. Сюда входят первичный файл и все файлы, которые явным образом не были помещены в другую группу.
* Пользовательские файловые группы – это любая группа создаваемая пользователем в БД.

Файлы журналов не входят ни в одну файловую группу, они обрабатываются отдельно от обычных файлов.



**Создание таблиц в файловой группе**

USE MyDB;  
CREATE TABLE MyTable  
 ( cola int PRIMARY KEY,  
 colb char(8) )  
ON MyDB\_FG1;  
GO

1. Группы операторов SQL: DDL, DML, DCL, TCL.

**DDL** - Data Definition Language - язык определения данных

* *CREATE тип имя дополнение (create function Func1 ...)*
* *ALTER тип имя дополнение (alter table Table1 …)*
* *DROP тип имя (drop table Table1)*

**DML** - Data Manipulation Language - язык манипулирования данными

* *SELECT список дополнение*
* *INSERT into таблица дополнение (insert into Table1(поле1, …) values ..)*
* *DELETE from таблица дополнение (delete from Table1)*
* *UPDATE таблица дополенение (update Table1 set …)*

**TCL** - Transaction Control Language - язык управления транзакциями

* *BEGIN TRAN дополнение*
* *SAVE TRAN дополнение*
* *COMMIT TRAN дополнение*
* *ROLLBACK TRAN дополнение*

**DCL** - Data Control Language - язык управления данными

* *GRANT список on объект to принципал*
* *REVOKE список on объект from принципал*
* *DENY список on объект to принципал*

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

1NF:

* Таблица не должна содержать повторяющихся групп данных
* Атомарность – каждый столбец должен содержать одно неделимое значение

2NF:

* Таблица находится в первой нормальной форме
* Каждый неключевой атрибут полностью функционально зависит от каждого возможного ключа

3NF:

* Таблица находится во второй нормальной форме
* Отсутствуют транзитивные зависимости, то есть каждый столбец, не являющийся ключом, должен зависеть **только** от столбца, который является ключом.

1. Типы данных Microsoft SQL Server.

* Точные числа
* Символьные строки в Юникоде
* Приблизительные числа (float, real)
* Двоичные данные (binary, varbinary, image)
* Дата и время
* Прочие типы данных(курсор и те что перечислены ниже)
* Символьные строки

**ROWVERSION** - синоним TIMESTAMP. Необходимо установить хронологию изменения данных занимает 8 байт. Значения могут вводиться и изменяться только сервером

**UNIQUEIDENTIFIER** - 16-байтовый идентификатор GUID. Главная особенность – способность генерировать уникальные значения, которые с очень малой вероятностью могут быть независимо получены еще раз. Могут быть получены при помощи встроенной функции NEWID.

**Hierarchyid** - Системный тип данных переменной длины. Используется для представления положения в иерархии

**geography** хранит эллиптические данные, такие как координаты широты и долготы GPS

**geometry** представляет данные в эвклидовом пространстве (плоской системе координат)

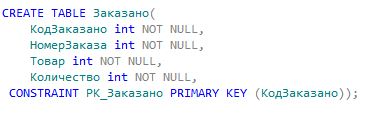
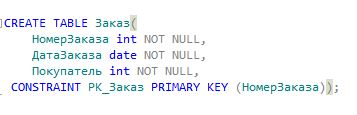
**sqlvariant -** Тип данных, хранящий значения различных типов данных. Максимальная длина значения типа sql\_variant составляет 8016 байт.

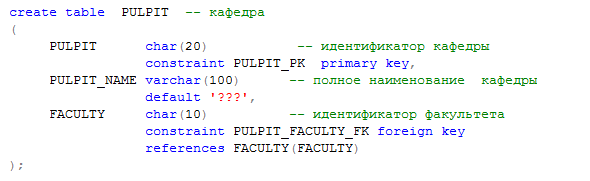
| **Тип данных** | **Описание** | **Тип данных** | **Описание** |
| --- | --- | --- | --- |
| **integer**  **(int)** | Целочисленные значения, занимают 4 байта. | **date** | Дата, занимает 3 байта |
| **smallint** | Целочисленные значения, занимают 2 байта | **time[(p)]**  **0 ≤ p ≤ 7** | Время, занимает от 3 до 5 байтов. **p** − колич. знаков после точки в секундах |
| **Tinyint** | Неотрицательные целочисленные значения, занимают 1 байт | **smalldatetime** | Дата и время, занимает 2 байта |
| **Bigint** | Целочисленные значения, занимают 8 байт | **datetime** | Дата и время, занимает 4 байта |

| **Тип данных** | **Описание** | **Тип данных** | **Описание** |
| --- | --- | --- | --- |
| **Real** | Вещественные числа с плавающей точкой | **datetime2** | Дата и время, занимает 8 байтов |
| **decimal (p[,s] )**  **(dec)**  или  **numeric (p[,s] )** | Вещественные значения с фикс. точкой, **р** − общее количество цифр, **s** − количество цифр после точки. Занимает от 5 до 17 байт. | **сhаr[(n)]** | Строки фиксированной длины из однобайтовых симво­лов, n − количество символов |
| **float[(p)]** | Вещественные числа с плавающей точкой. Если **р** < 25 − одинарная точность (4 байта), при **р** > 25 − двойная точность (8 байтов) | **nchar[(n)]** | Строки фиксированной длины символов Unicode. Каждый символ занимает 2 байта. |
| **Money** | Денежные значения, занимают 8 байтов | **varchar[(n)]** | Строки переменной длины из однобайтовых симво­лов |
| **smallmoney** | Денежные значения, занимают 4 байта | **nvarchar[(n)]** | Строки переменной длины символов Unicode − 2 байта |
| **binary(n)** | Задает битовую строку, длиной ровно **n** байтов | **varbinary(n)** | Задает битовую строку, длиной не более **n** байтов |

1. Создание таблиц.

Таблицы можно создавать используя Management Studio через контекстное меню или написав sql запрос





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

|  |  |
| --- | --- |
| **Условное обозначение** | **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**  **Значение по умолчанию** | устанавливает значение в столбце по умолчанию при выполнении операции INSERT |
| **primary key**  **первичный ключ** | предотвращает появление в столбце (группе столбцов) повторяющихся значений и пустого значения |
| **foreign key**  **внешний ключ** | устанавливает связь между таблицей со столбцом, имеющим свойство foreign key (FK внешний ключ) и таблицей, имеющей столбец со свойством primary key (PK – первичный ключ);  предотвращает несогласованные операции между PK и FK |
| **unique**  **уникальное значение** | аналогично primary key, но допускает пустые значения и не может быть использован для связи с foreign key |
| **check**  **проверка значений** | предотвращает появление в столбце значения, не удовлетворяющего логическому условию, записанному после check |

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

Data Manipulation Language - язык манипулирования данными

* *SELECT список дополнение*
* *INSERT into таблица дополнение (insert into Table1(поле1, …) values ..)*
* *DELETE from таблица дополнение (delete from Table1)*
* *UPDATE таблица дополенение (update Table1 set …)*

1. Операторы DDL: Create, alter, drop.

**DDL** - Data Definition Language - язык определения данных

* *CREATE тип имя дополнение (create function Func1 ...)*
* *ALTER тип имя дополнение (alter table Table1 …)*
* *DROP тип имя (drop table Table1)*
* Язык определения данных используется для создания и изменения структуры базы данных и ее составных частей - таблиц, индексов, представлений (виртуальных таблиц), а также триггеров и сохраненных процедур. Основными его командами являются:

|  |  |
| --- | --- |
| CREATE DATABASE | (создать базу данных) |
| CREATE TABLE | (создать таблицу) |
| CREATE VIEW | (создать виртуальную таблицу) |
| CREATE INDEX | (создать индекс) |
| CREATE TRIGGER | (создать триггер) |
| CREATE PROCEDURE | (создать сохраненную процедуру) |
| ALTER DATABASE | (модифицировать базу данных) |
| ALTER TABLE | (модифицировать таблицу) |
| ALTER VIEW | (модифицировать виртуальную таблицу) |
| ALTER INDEX | (модифицировать индекс) |
| ALTER TRIGGER | (модифицировать триггер) |
| ALTER PROCEDURE | (модифицировать сохраненную процедуру) |
| DROP DATABASE | (удалить базу данных) |
| DROP TABLE | (удалить таблицу) |
| DROP VIEW | (удалить виртуальную таблицу) |
| DROP INDEX | (удалить индекс) |
| DROP TRIGGER | (удалить триггер) |
| DROP PROCEDURE | (удалить сохраненную процедуру) |

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

Оператор select является часть группы операторов DML.

Кроме того, что перечислено в вопросе в секции SELECT могут использоваться опции TOP и/или DISTINCT, а в секции ORDERBY опции ASC или DESC. Обязательной является только секция SELECT.

Результатом работы запроса является результирующий набором.

Основное назначение секции SELECT – указать порядок и имена столбцов результирующего набора данных.

Секция INTO позволяет на основе структуры результирующего набора (заданную SELECT-списком) создать новую таблицу и заполнить ее строками результирующего набора.

Секция FROM в общем случае содержит выражение, результатом которого является набор однородных строк с явно поименованными столбцами. В простейшем случае – это таблица БД, а в общем случае, может быть достаточно сложное выражение, содержащее несколько таблиц, SELECT-запросов, представлений и табличных функций.

Секция WHERE применяется для фильтрации строк, сформированных в секции FROM. Как правило, секция содержит логическое выражение, которое вычисляется для каждой строки. В том случае, если выражение принимает истинное значение – строка отбирается.

GROUPBY – секция, позволяющая сгруппировать данные по значениям, хранящимся в столбцах однородного набора строк, сформированного в секции FROM. Секция GROUPBY ограничивает содержимое SELECT-списка.

Секция HAVING имеет такое же назначение, что и WHERE, но применяется после выполнения секции GROUPBY.

Для сортировки результирующего набора применяется секция ORDERBY. Выражения дают возможность задать порядок строк в результирующем наборе. Опции ASC и DESC позволяют указать порядок сортировки строк (по возрастанию или по убыванию).

**WHERE** содержит логическое выражение, которое вычисляется для каждой строки. В том случае, если выражение принимает истинное значение – строка отбирается.

**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 агрегируемых функций

**HAVING** определяется условие, которое применяется к группе строк

**DISTINCT** позволяет не выводить повторяющиеся строки

**ORDER BY** позволяет отсортировать результаты

**TOP** ограничивает количество результирующих строк

**INTO** указывает на то, что далее следует имя таблицы, в которую будут добавляться строки (вывод результата в таблицу)

SELECT column\_list

[INTO новая таблица]  
FROM table\_name   
[WHERE условие]   
[GROUP BY условие группировки]   
[HAVING условие для группировки]   
[ORDER BY условие [ASC|DESC]]

Порядок выполнения оператора SELECT – FROM, WHERE, GROUP BY, HAVING, SELECT, DISTINCT, ORDER BY, TOP, INTO.

1. Подзапросы. Конструкции in, exists, all, any, some.

**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.

В SELECT-списке допускается применять только такие подзапросы, которые формируют скалярный результирующий набор (набор, состоящий из одной строки и одного столбца).

**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.

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

SELECT DepartmentID, Name   
FROM HumanResources.Department   
WHERE EXISTS (SELECT NULL)

**ALL** используется совместно с операциями сравнения и предназначена для сравнения со списком значений, который задается как результат выполнения подзапроса.

WHERE price > ALL (SELECT price

FROM PC );

**ANY или SOME** формирует истинное значение в том случае, если значение, стоящее слева, больше(меньше) или равно хотя бы одному значению в списке, указанном справа.

model = ANY (SELECT model

FROM PC );

16 Представления. Создание представлений.

Представление– это объект БД, представляющий собой поименованный SELECT-запрос. При этом следует подчеркнуть: в БД хранится именно SELECT-запрос, а не результат его выполнения. Как и любой объект БД, представление создается с помощью оператора CREATE, удаляется с помощью оператора DROP и изменяется с помощью ALTER.

При создании представления к SELECT-запросу предъявляются следующие требования:

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

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.

CREATE VIEW [аудитории]

AS SELECT AUDITORIUM [номер],

AUDITORIUM\_NAME [наименование] FROM AUDITORIUM

WHERE AUDITORIUM\_TYPE='ËÊ'

SELECT \* FROM [аудитории]; DROP VIEW [аудитории];

17 Операции DML с представлениями. Представления with check option и schemabinding

Для наиболее простых представлений допустимо выполнение DML-операций INSERT, DELETE и UPDATE. Следует помнить, что любая операция добавления, удаления или изменения строк представления все равно транслируется в соответствующую операцию с базовой таблицей. Как следствие этого, все существующие ограничения целостности базовой таблицы наследуются представлением.

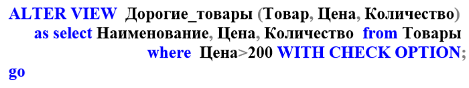
При создании представлений, позволяющих выполнять операции INSERT, DELETE и UPDATE, базовый SELECT-запрос должен удовлетворять следующим условиям:

1. не содержать секцию группировки GROUP BY;
2. не применять агрегатные функции;
3. не использовать опции DISTINCT и TOP;
4. не применять операторы UNION, INTERSECT и EXCEPT;
5. в SELECT-списке запроса не должно быть вычисляемых значений.
6. 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.

**WITH CHECK OPTION** — при использовании данной конструкции все добавляемые или изменяемые строки будут проверяться на соответствие определению представления. В случае несоответствия данное изменение не будет выполнено. Иными словами, нельзя добавить или изменить данные в представлении таким образом, чтобы они не были доступны через представление

Например, можно изменить представление **Дорогие\_товары**:

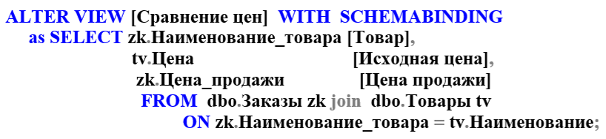


**SCHEMABINDING**

Привязывает представление к схеме базовой таблицы или таблиц. Если аргумент SCHEMABINDING указан, нельзя изменить базовую таблицу или таблицы таким способом, который может повлиять на определение представления. Все указанные в инструкции объекты должны находиться в одной базе данных.

Представления или таблицы, входящие в представление, созданное при помощи предложения SCHEMABINDING, не могут быть сброшены, пока это представление не будет удалено или изменено таким образом, чтобы оно более не было привязано к схеме. Кроме того, выполнение инструкций ALTER TABLE для таблиц, которые входят в представления, привязанные к схемам, завершается ошибкой, если эти инструкции влияют на определение представления.

Опция SCHEMABINDING устанавливает запрещение на операции с таблицами и представлениями, которые могут привести к нарушению работоспособности представления.



18 Агрегатные функции: count, sum, avg, min, max.

Агрегатные функции вычисляют некоторые скалярные значения в наборе строк. В MySQL есть следующие агрегатные функции:

* **AVG**: вычисляет среднее значение
* **SUM**: вычисляет сумму значений
* **MIN**: вычисляет наименьшее значение
* **MAX**: вычисляет наибольшее значение
* **COUNT**: вычисляет количество строк в запросе

Все агрегатные функции принимают в качестве параметра выражение, которое представляет критерий для определения значений. Зачастую, в качестве выражения выступает название столбца, над значениями которого надо проводить вычисления.

Выражения в функциях **AVG** и **SUM** должно представлять числовое значение (например, столбец, который хранит числовые значения). Выражение в функциях **MIN**, **MAX** и **COUNT** может представлять числовое или строковое значение или дату.

Все агрегатные функции за исключением COUNT(\*) игнорируют значения NULL.

По умолчанию все вышеперечисленных пять функций учитывают все строки выборки для вычисления результата. Но выборка может содержать повторяющие значения. Если необходимо выполнить вычисления только над уникальными значениями, исключив из набора значений повторяющиеся данные, то для этого применяется оператор **DISTINCT**.

|  |  |
| --- | --- |
| 1 | SELECT COUNT(DISTINCT Manufacturer) FROM Products |

19 Группировка данных с использованием cube, rollup

Конструкции ROLLUP и CUBE применяются в секции GROUP BY и служат для вычисления агрегатных значений над подмножествами строк.

Сформулируем правило формирования результирующего набора SELECT-запроса, применяющего в секции GROUP BY конструкцию ROLLUP.

* 1. Результирующий набор содержит *n* + 1 групп строк, где *n* – количество выражений для группировки, указанных в скобках за ключевым словом ROLLUP.
  2. Первая группа строк является результатом выполнения группировки по всем *n* выражениям.
  3. Вторая группа строк является результатом группировки первой группы **строк по *n* – 1 первым выражениям. Причем столбцы, по которым не выполнялась группировка,** заполняются значениями NULL.
  4. Группа строк k является группировкой группы строк, полученной на предыдущем этапе по *n* – *k* + 1 первым выражениям.
  5. Последняя (*n* + 1)-я группа содержит одну строку. Значения во всех столбцах, соответствующих выражениям в ROLLUP, равны NULL. Значение в столбце, соответствующем агрегатной функции, вычисляется по всем строкам исходного для группировки набора строк.

Сформулируем правило формирования результирующего набора SELECT-запроса, применяющего в секции GROUP BY конструкцию CUBE.

* 1. Формируется множество всех подмножеств выражений, указанных в CUBE-списке.
  2. Для каждого непустого подмножества, сформированного в п. 1 выполняется группировка. Если количество элементов подмножества меньше количества элементов CUBE-списка, то соответствующие значения в строках заполняются NULL (как это делалось конструкцией ROLLUP). Сформированные строки помещаются в результирующий набор секции GROUP BY.
  3. Для пустого подмножества, сформированного в п. 1 выполняется группировка аналогичная той, что выполнялась в п. 5 правил для ROLLUP.

20 Функция преобразования типов данных cast.

Преобразует выражение одного типа данных в другой.

CAST ( expression AS data\_type [ ( length ) ] )

**CAST(выражение AS тип\_данных)**

*Expression* Любое допустимое [выражение](https://docs.microsoft.com/ru-ru/sql/t-sql/language-elements/expressions-transact-sql?view=sql-server-2017).

*data\_type* Целевой тип данных. Это может быть xml, bigint и sql\_variant. Псевдонимы типов данных недопустимы.

*Length* Указываемое дополнительно целое число, обозначающее длину целевого типа данных.Значение по умолчанию — 30.

21 Функция преобразования типов данных convert

Преобразует выражение одного типа данных в другой.

CONVERT ( data\_type [ ( length ) ] , expression [ , style ] )

**CONVERT(тип\_данных, выражение [, стиль])**

*expression*

Любое допустимое [выражение](https://docs.microsoft.com/ru-ru/sql/t-sql/language-elements/expressions-transact-sql?view=sql-server-2017).

*data\_type*

Целевой тип данных. Это может быть xml, bigint и sql\_variant. Псевдонимы типов данных недопустимы.

*length*

Указываемое дополнительно целое число, обозначающее длину целевого типа данных.Значение по умолчанию — 30.

*style*

Целочисленное выражение, определяющее, как функция CONVERT преобразует значение аргумента *expression*. Для значения стиля NULL возвращается NULL. Аргумент *data\_type*определяет диапазон.

 Третий необязательный параметр задает стиль форматирования данных. Этот параметр представляет числовое значение, которое для разных типов данных имеет разную интерпретацию.

Например, некоторые значения для форматирования дат и времени:

• 0 или 100 - формат даты "Mon dd yyyy hh:miAM/PM" (значение по умолчанию)

• 1 или 101 - формат даты "mm/dd/yyyy"

• 3 или 103 - формат даты "dd/mm/yyyy"

• 7 или 107 - формат даты "Mon dd, yyyy hh:miAM/PM"

• 8 или 108 - формат даты "hh:mi:ss"

• 10 или 110 - формат даты "mm-dd-yyyy"

• 14 или 114 - формат даты "hh:mi:ss:mmmm" (24-часовой формат времени)

Некоторые значения для форматирования данных типа money в строку:

• 0 - в дробной части числа остаются только две цифры (по умолчанию)

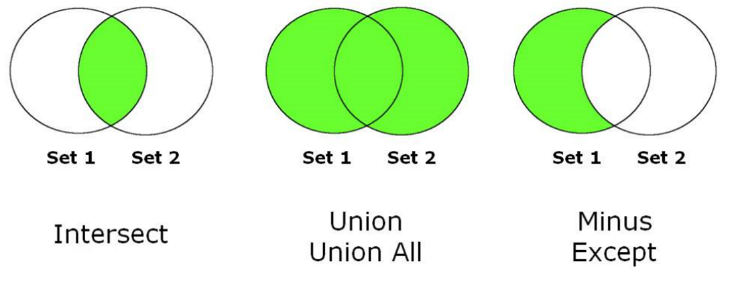
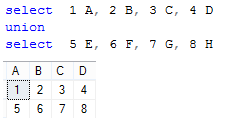
• 1 - в дробной части числа остаются только две цифры, а для разделения разрядов применяется запятая

• 2 - в дробной части числа остаются только четыре цифры

22 Операторы union (all), intersect, except

Оператор UNION позволяет объединить результирующие наборы, сформированные двумя SELECT-запросами в общий результирующий набор. Единственным требованием, предъявляемым к SELECT-запросам, является совместимость типов соответствующих столбцов сформированных результирующих наборов.

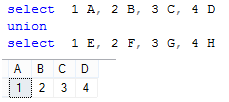
Приведен простейший пример использования оператора UNION для объединения результирующих наборов двух SELECT-запросов.



Каждый оператор SELECT формирует результирующий набор, состоящий из одной строки. Заметим следующее.

* 1. Результирующий набор оператора UNION содержит объединение строк результирующих наборов первого и второго SELECT-запроса.
  2. Имена столбцов результирующего набора оператора UNION совпадают с именами столбцов результирующего набора, сформированного первым SELECT-запросом.
  3. Количество столбцов результирующих наборов первого и второго SELECT-запросов должно быть одинаковое.
  4. Типы соответствующих столбцов должны быть совместимы.
  5. Оператор UNION является коммутативным: результат не зависит от порядка объединяемых наборов.

Оператор UNION выполняет теоретико-множественную операцию объединения, т. е. результатом является множество строк, в котором строки не могут повторяться. Обратите внимание на то, что результатом объединения является только одна строка.



Если требуется механическое объединение строк, можно применить оператор UNION ALL.

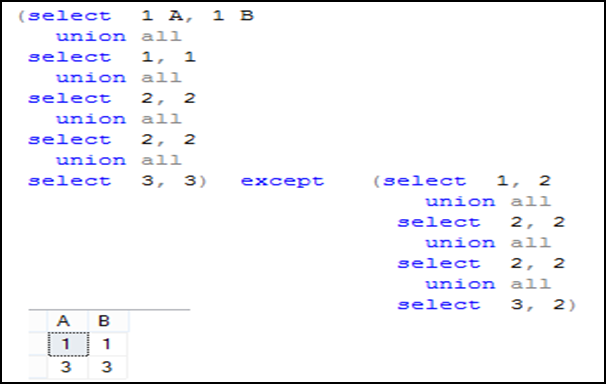
Результатом оператора INTERSECT является набор строк, представляющий собой пересечение двух исходных наборов строк. Как и для оператора UNION, к SELECT-запросам предъявляется требование совместимости типов соответствующих столбцов.

Заметим следующее.

* 1. В исходных наборах строк должно быть одинаковое количество столбцов и совместимый тип соответствующих столбцов.
  2. Результирующий набор оператора INTERSECT содержит множество строк, являющееся пересечением двух исходных множеств строк. Другими словами, результирующий набор оператора INTERSECT содержит строки входящие одновременно и в первый во второй исходные наборы данных.
  3. Результатом оператораINTERSECTявляется множество неповторяющихся строк. В этом смысле, оператором выполняется теоретико-множественная операция пересечения множеств.
  4. Имена и типы столбцов результирующего набора оператора INTERSECTсовпадают с именами и типами столбцов первого результирующего набора.
  5. Оператор INTERSECT является коммутативным: результат не зависит от порядка исходных наборов.

Оператор INTERSECT имеет более высокий приоритет, чем оператор UNION.

Результатом оператора EXCEPT является набор строк, представляющий собой разность двух исходных наборов строк. Как и для операторов UNION и INTERSECT к SELECT-запросам предъявляется требование совместимости типов соответствующих столбцов.



Заметим следующее.

* 1. В исходных наборах строк должно быть одинаковое количество столбцов и совместимый тип соответствующих столбцов.
  2. Результирующий набор оператора EXCEPT содержит множество строк, являющееся разностью двух исходных множеств строк. Другими словами, результирующий набор оператора EXCEPT содержит строки, которые входят в первый набор строк, но не входят во второй.
  3. Результатом оператора EXCEPT является множество неповторяющихся строк. В этом смысле оператором выполняется теоретико-множественная операция вычитания множеств.
  4. Имена и типы столбцов результирующего набора оператора EXCEPT совпадают с именами и типами столбцов первого результирующего набора.
  5. Оператор EXCEPT не является коммутативным: результат зависит от порядка исходных наборов.

Изменение порядка исходных наборов строк приводит к другому результату оператора EXCEPT.

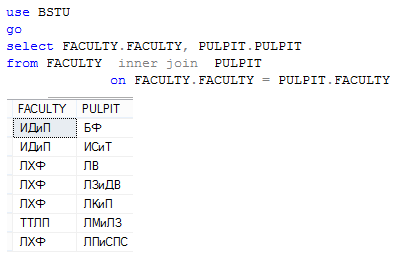
Наивысшим приоритетом обладает оператор INTERSECT, а операторы UNION и EXCEPT имеют одинаковый приоритет.

**INTERSECT** — пересечение — набор строк, которые принадлежат к обеим таблицам.

**EXCEPT** — разность двух таблиц — все значения, которые принадлежат к первой таблице и не присутствуют во второй.

23 Соединение таблиц. Внутреннее соединение inner join

Соединение INNER JOIN наиболее часто используемый вид соединения реляционных таблиц.



* Внутреннее соединение содержит только те строки одной таблицы, для которых имеются соответствующие строки в другой таблице

выражение INNER JOIN, указываемое между двумя соединяемыми таблицами; логическое выражение, записываемое после ключевого слова ON.

операция INNER JOIN является коммутативной – формируемый результирующий набор не зависит от порядка, в котором указаны таблицы.

Внутреннее соединение можно выполнить для двух и более таблиц.

Допускается применение подзапросов.

С помощью INNER JOIN можно соединять таблицу саму с собой. Обычно такие запросы применяются для получения информации об иерархических связях между строками одной таблицы.

Логическое условие, указываемое после ключевого слова ON, не обязательно должно содержать условие, записываемое в виде равенства – это может любое логическое выражение.

24 Ортогональное соединение cross join

Оператор SQL CROSS JOIN формирует таблицу перекрестным соединением (декартовым произведением) двух таблиц. При использовании оператора SQL CROSS JOIN каждая строка левой таблицы сцепляется с каждой строкой правой таблицы. В результате получается таблица со всеми возможными сочетаниями строк обеих таблиц. Оператор SQL CROSS JOIN имеет следующий синтаксис: SELECT column\_names [,... n] FROM Table\_1 CROSS JOIN Table\_2

Обратите внимание, что в операторе за ненадобностью отсутствует условие (ON).

**Примеры оператора SQL CROSS JOIN.**Имеются две таблицы:

**Authors** — содержит в себе информацию об авторах книг:

|  |  |
| --- | --- |
| **AuthorID** | **AuthorName** |
| 1 | Bruce Eckel |
| 2 | Robert Lafore |
| 3 | Andrew Tanenbaum |

|  |  |
| --- | --- |
| **BookID** | **BookName** |
| 3 | Modern Operating System |
| 1 | Thinking in Java |
| 3 | Computer Architecture |
| 4 | Programming in Scala |

**Books** — содержит в себе информацию о названии книг:

В таблице Books поле BookID являются внешним ключом и ссылаются на таблицу Authors.

**Пример 1.**Произвести декартово произведение обеих таблиц с помощью оператора SQL CROSS JOIN: SELECT \* FROM Authors CROSS JOIN Books

|  |  |  |  |
| --- | --- | --- | --- |
| **Authors.AuthorID** | **Authors.AuthorName** | **Books.BookID** | **Books.BookName** |
| 1 | Bruce Eckel | 3 | Modern Operating System |
| 1 | Bruce Eckel | 1 | Thinking in Java |
| 1 | Bruce Eckel | 3 | Computer Architecture |
| 1 | Bruce Eckel | 4 | Programming in Scala |
| 2 | Robert Lafore | 3 | Modern Operating System |
| 2 | Robert Lafore | 1 | Thinking in Java |
| 2 | Robert Lafore | 3 | Computer Architecture |
| 2 | Robert Lafore | 4 | Programming in Scala |
| 3 | Andrew Tanenbaum | 3 | Modern Operating System |
| 3 | Andrew Tanenbaum | 1 | Thinking in Java |
| 3 | Andrew Tanenbaum | 3 | Computer Architecture |
| 3 | Andrew Tanenbaum | 4 | Programming in Scala |

Декартово произведение (перекрестное соединение) соединяет каждую строку первой таблицы с каждой строкой второй. Результатом декартово произведения первой таблицы с n строками и второй таблицы с m строками будет таблица с n × m строками.

SELECT employee.\*, department.\*

FROM employee

CROSS JOIN department;

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

Внешнее соединение двух таблиц формирует набор строк, состоящий из двух частей:

1)результат внутреннего соединения двух таблиц;

2)строки из двух таблиц, которые не смогли соединиться; если в SELECT-списке указаны столбцы двух таблиц, то значения в столбцах, соответствующих незаполненной (несоединенной) части строки, будут NULL.

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

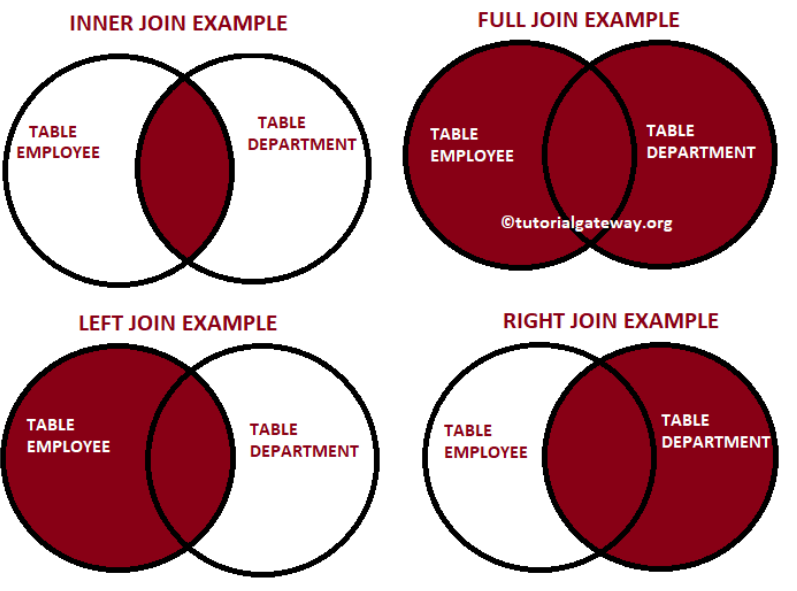
Левое внешнее соединение, в соответствии записано слева от ключевых слов LEFT OUTER JOIN, а правое внешнее соединение – несоединенные строки таблицы, имя которой записано справа от RIGHT OUTER JOIN.

**FULL OUTER JOIN** определяет объединение правого и левого соединения

 Внешнее соединение позволяет в отличие от внутреннего извлечь не только строки с одинаковыми значениями соединяемых столбцов, но и строки без совпадений из одной или обеих таблиц.

Выделяют три вида внешних соединений:

* **левое внешнее соединение** — в результирующий набор попадают все строки из таблицы с левой стороны оператора сравнения (независимо от того имеются ли совпадающие строки с правой стороны), а из таблицы с правой стороны — только строки с совпадающими значениями столбцов.
* **правое внешнее соединение** — аналогично левому внешнему соединению, но таблицы меняются местами
* **полное внешнее соединение** — композиция левого и правого внешнего соединения: результирующий набор состоит из всех строк обеих таблиц. Если для строки одной из таблиц нет соответствующей строки в другой таблице, всем ячейкам строки второй таблицы присваивается значение NULL.



26 Язык T-SQL. Пакеты. Объявление переменных.

Язык SQL – является декларативным языком. В нем отсутствуют инструкции, позволяющие управлять последовательностью выполнения операторов. Другими словами, в языке SQL нет операторов перехода, цикла, ветвления и других механизмов, с помощью которых можно было бы управлять ходом выполнения программы (записать алгоритм).

Пакет – это группа операторов T-SQL, которая обрабатывается сервером СУБД вместе. Понятие пакета необходимо при клиент-серверной архитектуре приложения. В этом смысле, пакет – это группа операторов T-SQL, которую программа клиент отправляет серверу на исполнение.

Разграничиваются пакеты с помощью специальной инструкции GO. Инструкция GO интерпретируется как команда отправки на сервер порции операторов T-SQL. В качестве операнда инструкции GO можно указать целое положительное число, которое будет восприниматься как количество повторений пакета.

**Контекст выполнения пакета**

Каждый пакет выполняется в контексте какой-то БД. Контекст БД переключается с помощью инструкции USE. За ключевым словом USE указывается имя БД, в контекст которой осуществляется переключение. Обычно переключение контекста выполняется в отдельном пакете. Как и GO инструкция USE применяется стандартными программами-клиентами, входящими в состав программного обеспечения MSS. Переключение контекста БД позволяет указывать короткие имена объектов БД, используемых в операторах T-SQL. Если в контексте одной БД требуется сослаться на объект другой БД в рамках одного сервера, то необходимо использовать трехкомпонентное имя объекта, включающего имя БД, имя схемы и собственно имя объекта.

**Объявление переменных**

Оператор DECLARE предназначен для объявления используемых в программе переменных. Для каждой переменной указывается имя и тип. Имя должно обязательно начинаться с символа **@**, но не c **@@** , так как это сочетание символов зарезервировано для системных переменных и функций.

 Перечень допустимых типов переменных совпадает с перечнем типов данных БД MSS. Переменная может быть проинициализирована в операторе DECLARE с помощью символа **=**. В одном сценарии допускается несколько операторов DECLARE.

Тип TABLE позволяет создавать таблицы в памяти и использовать их для хранения промежуточных данных. Жизненный цикл всех объявленных переменных заканчивается вместе с завершением пакета, в котором находится соответствующий оператор DECLARE.

DECLARE @name NVARCHAR(20), @age INT;

SET @name='Tom';

SET @age = 18;

27 Язык T-SQL. Операторы присвоения.

Переменной, объявленной в операторе DELCLARE, можно присвоить значение тремя способами: инициализировать в операторе DECLARE (рис.13.2), присвоить значение с помощью операторов SET или SELECT (рис 13.3).

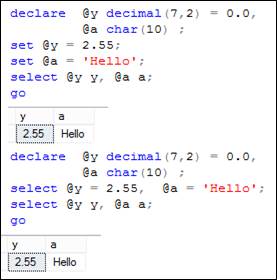


Рис. 13.3. Присваивание значений с помощью операторов SET и SELECT

Единственное отличие операторов присваивания SET и SELECT заключается в возможности SELECT присваивать значения одновременно нескольким переменным.

28 Язык T-SQL. Операторы print, if-else, case.

PRINT (Transact-SQL)

PRINT msg\_str | @local\_variable | string\_expr

Аргументы

*msg\_str*

Символьная строка или строковая константа Юникода.

@ *local\_variable*

Переменная любого допустимого символьного типа данных. Аргумент @ *local\_variable* должен иметь тип char, nchar, varchar или nvarchar либо должен неявно преобразовываться в эти типы данных.

*string\_expr*

Выражение, возвращающее строку. Может содержать объединенные буквенные значения, функции и переменные.

IF...ELSE (Transact-SQL)

Задает условия для выполнения инструкции Transact-SQL. Инструкция языка Transact-SQL, следующая за ключевым словом IF и его условием, выполняется только в том случае, если логическое выражение возвращает TRUE. Необязательное ключевое слово ELSE представляет другую инструкцию языка Transact-SQL, которая выполняется, если условие IF не удовлетворяется и логическое выражение возвращает FALSE.

IF Boolean\_expression   
 { sql\_statement | statement\_block }   
[ ELSE   
 { sql\_statement | statement\_block } ]

Аргументы

*Boolean\_expression*

Выражение, возвращающее значение TRUE или FALSE. Если логическое выражение содержит инструкцию SELECT, инструкция SELECT должна быть заключена в скобки.

{ *sql\_statement*| *statement\_block* }

Любая инструкция или группа инструкций языка Transact-SQL, указанная с помощью блока инструкций. Без использования блока инструкций условия IF и ELSE могут повлиять на выполнение только одной инструкции языка Transact-SQL.

Для определения блока инструкций используйте ключевые слова потока управления BEGIN и END.

Remarks

Конструкция IF...ELSE может быть использована в пакетах, хранимых процедурах и нерегламентированных запросах. При использовании в хранимой процедуре эта конструкция часто применяется для проверки существования некоторого параметра.

Проверки IF могут находиться внутри другого IF или следующего ELSE. Ограничение количества вложенных уровней зависит от свободной памяти.

Пример

IF DATENAME(weekday, GETDATE()) IN (N'Saturday', N'Sunday')  
 SELECT 'Weekend';  
ELSE   
 SELECT 'Weekday';

Выражение CASE (Transact-SQL)

Выражение CASE, как правило, применяется в операторе SELECT (в SELECT-списке, секциях WHERE, ORDER BY, HAVING) и служит для формирования одного из нескольких возможных значений. Может применяться в операторах UPDATE (в секциях SET и WHERE), DELETE (в секции WHERE). Выражение CASE имеет два формата: первый сравнивает результат одного выражения со списком заданных значений, второй – вычисляет набор логических выражений.

На рис. 13.20 представлен составной SELECT-запрос, использующий выражение CASE первого формата в SELECT-списке и в секции GROUP BY внутреннего запроса, а также в секции ORDER BY внешнего запроса.

Сразу за ключевым словом CASE (рис. 13.20) следует вычисляемое выражение (в нашем случае – это просто значения из столбца **NOTE** таблицы **PROGRESS** или из столбца **Категория оценки** результирующего набора внутреннего запроса), а далее одно или более предложений WHEN, содержащих значения для сравнений. Ключевое слово THEN, присутствующее в каждом предложении WHEN, указывает на один из возможных (соответствующих значению) результатов, формируемых CASE. Список WHEN-предложений завершается предложением ELSE, содержащим результат выражения CASE, формируемого в случае безуспешного сравнения значения исходного выражения со всеми предшествующими WHEN-значениями.

Обратите внимание на применение выражения CASE в секциях GROUP BY и ORDER BY.

Применение CASE в GROUP BY позволяет формировать группы строк на основе заданного в этом выражении условия. В примере на рис. 13.20 в зависимости от значений в столбце **NOTE** строки таблицы **PROGRESS** разбиваются на четыре группы (в первой группе нет ни одной строки), соответствующие четырем возможным результатам выражения CASE.

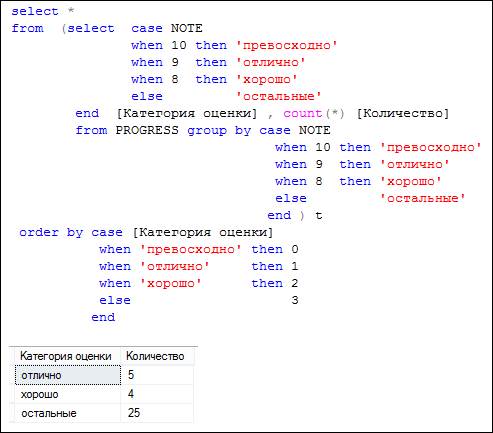


Рис. 13.20. Пример использования выражения CASE первого формата

Выражение CASE в ORDER BY позволяет динамически формировать значения ключей сортировки. В примере на рис. 13.20 с помощью CASE в результате сортировки строка результирующего набора внутреннего запроса со значением **превосходно** в столбце **Категория оценки** будет первой (ключ сортировки **0**), со значением **отлично** (ключ сортировка **1**) и т. д.

Пример на рис. 13.21 демонстрирует применение выражения CASE второго формата в SELECT-списке и секции GROUP BY запроса SELECT.

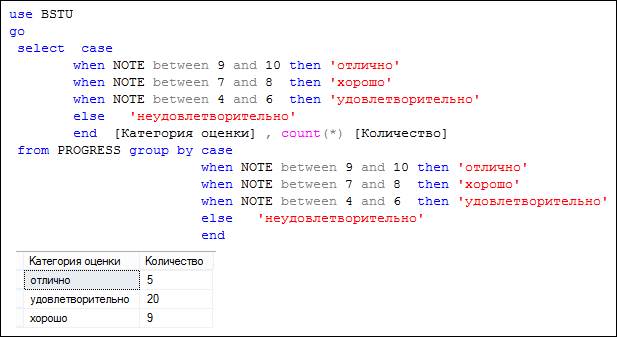


Рис. 13.21. Пример использования выражения CASE второго формата

Обратите внимание: каждое предложение WHEN содержит логическое выражение. Логические 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.

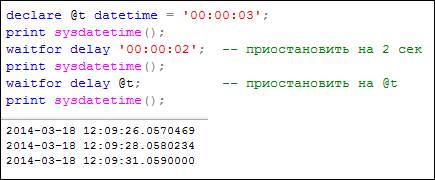
29 Язык T-SQL. Операторы begin-end, waitfor и return.

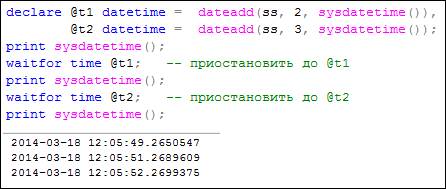
С помощью оператора BEGIN END указывается, что группа следующих друг за другом операторов должна выполняться как единое целое.

Объединение операторов T-SQL в блоки с помощью BEGIN END никак не влияет на действие оператора RETURN (в любом случае осуществляется выход из пакета) и на область видимости переменных (область видимости распространяется строго сверху вниз).

С помощью оператора WAITFOR можно приостановить выполнение пакета. Оператор работает в двух режимах: DELAY и TIME.

В режиме DELAY оператор WAITFOR приостанавливает выполнение пакета на заданный интервал времени.В режиме TIME можно приостановить выполнение потока до наступления заданного значения системного времени на сервере.





Немедленно завершить работу пакета можно с помощью оператора RETURN.

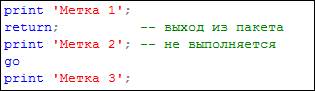


Рис. 13.10. Пример использования оператора RETURN

Кроме выхода из текущего пакета оператор RETURN применяется для завершения работы процедур, функций и триггеров

30 Язык T-SQL. Оператор цикла while.

Оператор WHILE предназначен для организации программного цикла.

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».

На рис. 13.2 приведен сценарий, демонстрирующий работу оператора WHILE. Заметим, что тело цикла содержит только один оператор.

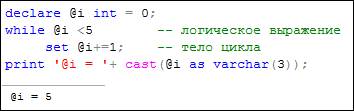


Рис. 13.12.

В том случае, если тело цикла содержит более, чем один оператор, следует применять операторные скобки BEGIN END (рис. 13.13).

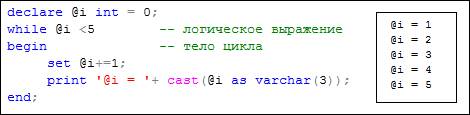


Рис. 13.12.

Управление выполнением WHILE-цикла может осуществляться с помощью инструкций BREAK и CONTINUE. На рис. 13.14 представлен пример, демонстрирующий использование этих инструкций.

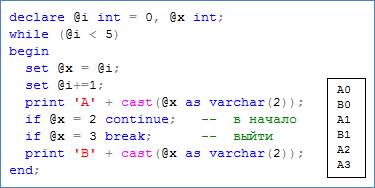


Рис. 13.14. Пример использования инструкций BREAK и CONTINUE

Инструкция CONTINUE (рис. 13.14) прерывает выполнение блока операторов в теле WHILE и передает управление в начало цикла. Если в теле цикла выполнилась инструкция BREAK, работа цикла прекращается и управление передается следующему за WHILE оператору.

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

Язык SQL – является декларативным языком. В нем отсутствуют инструкции, позволяющие управлять последовательностью выполнения операторов.

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. В блоке CATCH можно использовать несколько системных функций, позволяющих диагностировать возникшую в cохраняемом коде ошибку.

|  |  |
| --- | --- |
| Наименование функции | Функция возвращает |
| ERROR\_NUMBER | Код (целочисленное значение) последней ошибки |
| ERROR\_MESSAGE | Сообщение (символьная строка) об ошибке |
| ERROR\_SEVERITY | Уровень серьезности (целочисленное значение) ошибки |
| ERROR\_STATE | Метку (целочисленное значение) ошибки |
| ERROR\_LINE | Код (целочисленное значение) последней ошибки |
| ERROR\_PROCEDURE | Имя (символьная строка) процедуры или NULL |

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.

Инструкция RAISERROR применяется дважды: в первом случае для предупреждения об ошибке при делении на нуль, во втором – об ошибке нарушения ограничения PRIMARY KEY. Обратите внимание: инструкция ведет себя по-разному в зависимости от заданного параметром уровня серьезности ошибки. В первом случае (уровень серьезности 10) выполнение инструкции свелось к выводу в стандартный поток сообщения и выполнение охраняемого кода продолжилось. Во втором (уровень серьезности 11) – управление было передано в блок обработки ошибок.

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

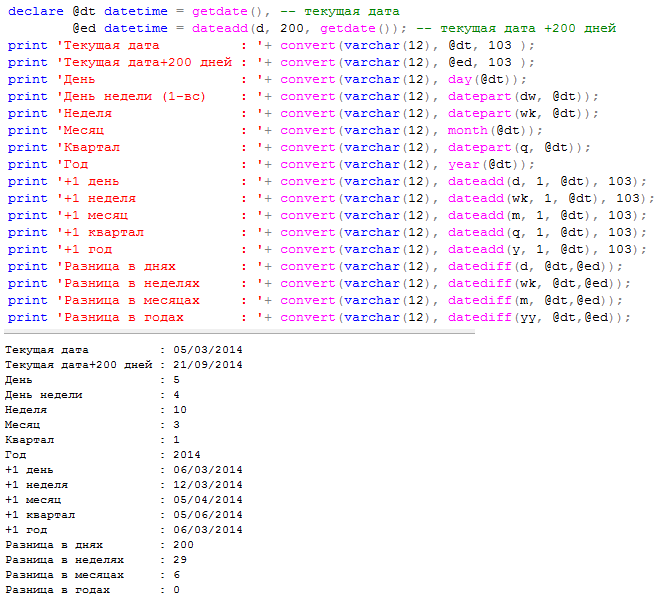
Для работы с датами и временем в языке T-SQL предусмотрен специальный набор встроенных функций. На рис. 13.8 представлен сценарий, демонстрирующий наиболее часто применяемые функции для работы с датами.

Рис. 13.8. Примеры использования встроенных функций T-SQL  
 для работы с датами

Обратите внимание на третий числовой параметр функции CONVERT, используемой здесь для преобразования даты к строке. Этот параметр определяет строчный формат даты.

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

Помимо операции конкатенации строк, упоминаемой ранее, язык T-SQL обеспечивает широкий набор встроенных функций для работы со строковыми данными. На рис.13.7 приведены примеры использования встроенных строковых функций.

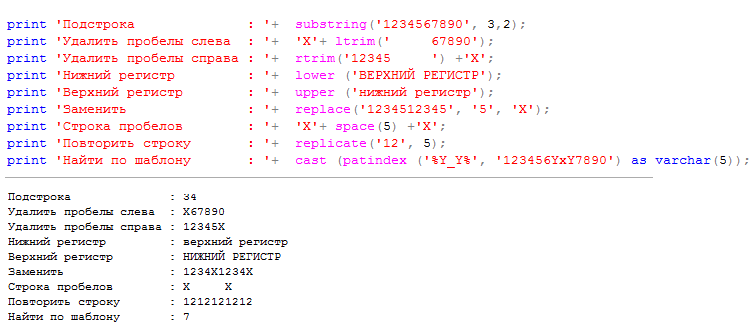
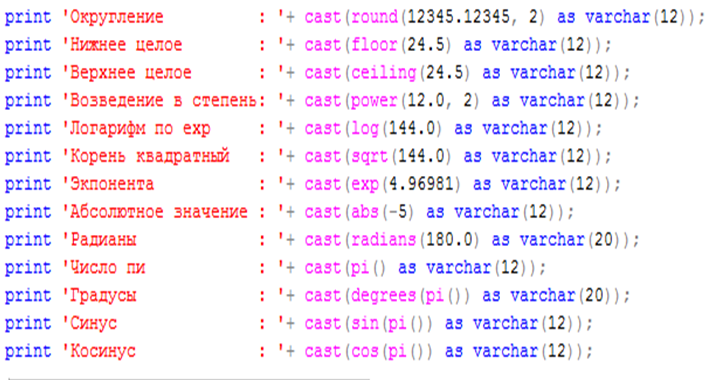
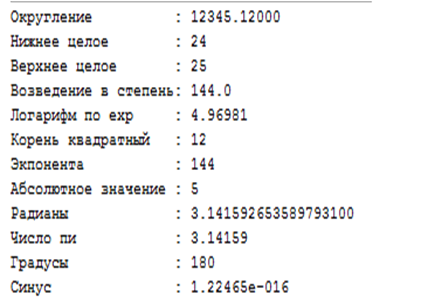


Рис. 13.7. Примеры использования встроенных строковых функций T-SQL

**34. Язык T-SQL. Встроенные функции работы с числовыми данными.**





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

**Временный табл** хранять во временной системе данных TEMPDB. Не могут иметь внешнего ключа. Создаются каждый раз заново, хранятся во время сеанса.

**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.

use P\_Bstu; create table #VRTABL( NAME nvarchar(100), BDAY date);

insert into #VRTABL(NAME, BDAY)

values ('Петров Игорь Васильевич,'11.03.1995')

select\*from #VRTABL;

drop table #VRTABL;

Следует обратить внимание: несмотря на то, что временная таблица **#VRTABL** создается в контексте БД **P\_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.

create table ##VR ( Adress nvarchar(100), telefon nvarchar(100));insert into ##VR (Adress, telefon)

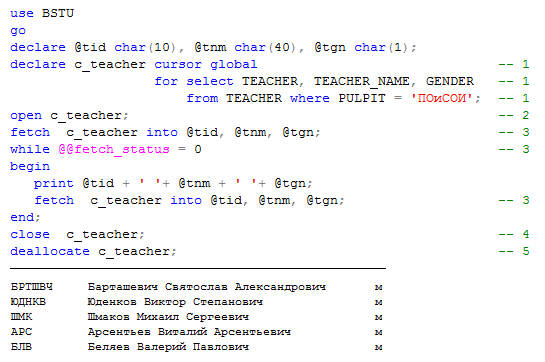
values (“Ершова 15”,'25658974)

select\*from #VR;

drop table ##VR;

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

**Курсор** является программной конструкцией, которая служит для хранения результата запроса и дает возможность пользователю обрабатывать строки результирующего набора запись за записью. Курсоры бывают локальные и глобальные (по умолчанию), статические и динамические (по умолчанию). Работа с курсором осуществляется в следующей последовательности: курсор объявляется в операторе DECLARE, открывается с помощью оператора OPEN, с помощью оператора FETCH считывается одна или несколько строк результирующего набора. Результат каждого считывания проверяется с помощью системной функции @@FETCH\_STATUS. Затем курсор закрывается с помощью оператора CLOSE. Если курсор глобальный, то он должен быть освобожден с помощью оператора DEALLOCATE.

****

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

*Локальный* курсор может применяться в рамках одного пакета и ресурсы, выделенные ему при объявлении, освобождаются сразу после завершения работы пакета.

*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.

Открытие *статического* курсора приводит к выгрузке результирующего набора во временную таблицу системной БД **TEMPDB**, и все дальнейшие операции осуществляются с этой таблицей, т. е. после открытия курсора все текущие изменения в исходных таблицах не будут отражаться в курсорном результирующем наборе.

*Динамический курсор* поддерживает данные в "живом" состоянии, но это требует затрат сетевых и программных ресурсов. При использовании динамических курсоров не создается полная копия исходных данных, а выполняется динамическая выборка из исходных таблиц только при обращении пользователя к тем или иным данным. На время выборки сервер блокирует строки, а все изменения, вносимые пользователем в полный результирующий набор курсора, будут видны в курсоре. Однако если другой пользователь внес изменения уже после выборки данных курсором, то они не отразятся в курсоре.

Курсору может быть установлено свойство READ\_ONLY, запрещающее применение операции CURRENT OF в секции WHERE операторов DELETE и UPDATE. Однако свойство READ\_ONLY не мешает применять эти операторы без использования CURRENT OF.

Следует отметить, что атрибут READ\_ONLY имеет смысл только для динамических и ключевых курсоров, статические курсоры по определению применяются только для чтения.

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

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

**Scroll**– тип курсора, в котором мы можем двигаться в любом направлении. Работа со скролл курсорами:

* last-позиция переходит на последнюю строку;
* first-позиция переходит на первую строку;
* absolute 5 – на 5 строку относительно начала;
* absolute -5 – на 5 строку относительно конца;
* relative 5 – на строку, от текущей +5 (вниз на 5);
* relative -5 на строку, от текущей -5 (вверх на 5)
* next- позиция переходит на следующую строку;
* prior-позиция переходит на предыдущую строку.

**Where current ofcursor** — изменение текущей строки, очень быстрая операция. работает быстрее обычного where.

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

**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 |

* @@FETCH\_STATUS – возвращает состояние последней инструкции FETCH, вызванной в любом курсоре, открытом в данном соединении

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

SQL позволяет использовать в командах UPDATE и DELETE специальную конструкцию WHERE CURRENT OF, облегчающую процесс изменения последней выбранной из курсора строки данных.

Обновление столбцов последней выбранной строки можно выполнить следующей командой:

UPDATE имя\_таблицы SET

предложение\_set WHERE CURRENT

OF имя\_курсора;

Аналогичным образом производится удаление последней выбранной строки:

DELETE

FROM имя\_таблицы WHERE

CURRENT OF имя\_курсора;

Обратите внимание: в секции WHERE CURRENT OF указывается курсор, а не запись, в которую была помещена очередная строка.

Главное преимущество использования секции WHERE CURRENT OF при модификации или удалении последней извлеченной строки из курсора заключается в том, что один критерий поиска строки не нужно задавать в двух (и более) местах программы. Не будь его, пришлось бы ввести секцию WHERE в определение курсора, а затем повторить его в соответствующих командах UPDATE и DELETE. Если бы в будущем структура таблицы изменилась способом, влияющим на формирование первичного ключа, нам пришлось бы вносить изменения во всех командах SQL, в которых оно используется. С другой стороны, с where current of изменяется только секция WHERE команды SELECT.

declare   
 cursor C is  
 select \*  
 from tab11 t  
 FOR UPDATE OF object\_name, owner;   
begin  
 FOR i in C loop  
 update tab11 set object\_name = 'update1' , owner = 'OW'   
 where current of C;  
 end loop;   
 commit;   
end;

**41. Хранимые процедуры и функции T-SQL.**

**Хранимая процедура** – это объект базы данных, представляющий собой поименованный код T-SQL. Хранимая процедура может быть создана с помощью CREATE, изменена с помощью ALTER и удалена с помощью оператора DROP. Процедура может принимать входные и формировать выходные параметры. Результатом ее выполнения может быть целочисленное значение, которое возвращается к точке вызова оператором RETURN, либо один или более результирующих наборов, сформированных операторами SELECT, либо содержимое стандартного выходного потока, полученного при выполнении операторов PRINT. Вызов процедуры осуществляется оператором EXECUTE (EXEC).

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

**CREATE** **PROCEDURE** IncreaseBudget (@percent INT=5)

**AS** **UPDATE** Project

**SET** Budget = Budget + Budget \* @percent/100;

**EXECUTE** IncreaseBudget 10;

Отличия функций от хранимых процедур в:

* Возвращаемое значение;
* Способ вызова;
* Не допускают применения DDL-операторов (CREATE, ALTER, DROP), DML-операторов (INSERT, UPDATE, DELETE), TRY/CATCH, транзакции.

**Функция** – это объект БД, представляющий собой поименованный код T-SQL. Для создания, удаления и изменения функций надо использовать операторы 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.

Результатом выполнения функции является возвращаемое к точке вызова значение. 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.

Встроенные табличные функции возвращают к точке вызова результирующий набор SELECT-запроса и по своим возможностям очень напоминают представления.

Многооператорные табличные функции возвращают к точке вызова результирующий набор и допускают более сложную логику работы.

**CREATE FUNCTION**, которая имеет следующий синтаксис:

**CREATE** **FUNCTION** ComputeCosts (@percent INT = 10)  
 **RETURNS** DECIMAL(16, 2)  
 **BEGIN**  
 **DECLARE** @addCosts DEC (14,2), @sumBudget DEC(16,2)  
 **SELECT** @sumBudget = **SUM** (Budget) **FROM** Project  
 **SET** @addCosts = @sumBudget \* @percent/100  
 **RETURN** @addCosts  
 **END**;

Вызов в селекте - dbo.ComputeCosts(25)

**42. Создание хранимых процедур. Передача параметров. Входные и выходные параметры.**

**Хранимая процедура** – это объект базы данных, представляющий собой поименованный код T-SQL. Хранимая процедура может быть создана с помощью CREATE, изменена с помощью ALTER и удалена с помощью оператора DROP. Процедура может принимать входные и формировать выходные параметры. Результатом ее выполнения может быть целочисленное значение, которое возвращается к точке вызова оператором RETURN, либо один или более результирующих наборов, сформированных операторами SELECT, либо содержимое стандартного выходного потока, полученного при выполнении операторов PRINT. Вызов процедуры осуществляется оператором EXECUTE (EXEC).

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

CREATE PROCEDURE PSUBJECT -- Создаём процедуру

@p varchar(20), -- С 2 параметрами: р и с

@c int output

AS BEGIN

DECLARE @k int = (SELECT count(\*) FROM SUBJECT1 );

PRINT 'Параметры: @p='+@p+', @c='+cast(@c as varchar(3));

SELECT \* FROM SUBJECT1 WHERE SUBJECT=@p; -- находим кол строк

SET @c = @@rowcount;

RETURN @k;

END;

DECLARE @k INT =0, @r INT =0, @p VARCHAR(20);

EXEC @k= PSUBJECT @p='ОАиП', @c=@r output; -- вызов процедуры

DROP PROCEDURE PSUBJECT -- Удаление процедуры

Возвращает полное название предмета, кафедру на которой он читается и количество строк в результирующем наборе.

Для передачи входных и выходных данных в создаваемой хранимой процедуре имена параметров должны начинаться с символа @. В одной хранимой процедуре можно задать множество параметров, разделенных запятыми. В теле процедуры не должны применяться локальные переменные, чьи имена совпадают с именами параметров этой процедуры. Наличие ключевого слова OUTPUT означает, что соответствующий параметр предназначен для возвращения данных из хранимой процедуры. Однако это вовсе не означает, что параметр не подходит для передачи значений в хранимую процедуру.

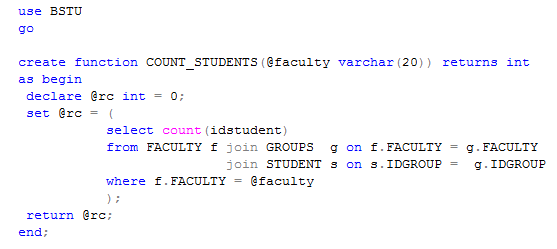
**43. Виды функций. Создание функций. Передача параметров.**

**Функция** – это объект БД, представляющий собой поименованный код T-SQL. Для создания, удаления и изменения функций надо использовать операторы 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.

Результатом выполнения функции является возвращаемое к точке вызова значение. 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.

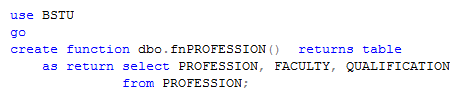
**Скалярные функции**

Скалярная функция возвращает единственное значение и принимает любое количество параметров. Допускается возвращать и принимать все типы данных, за исключением TIMESTAMP и устаревших IMAGE, TEXT и NTEXT.



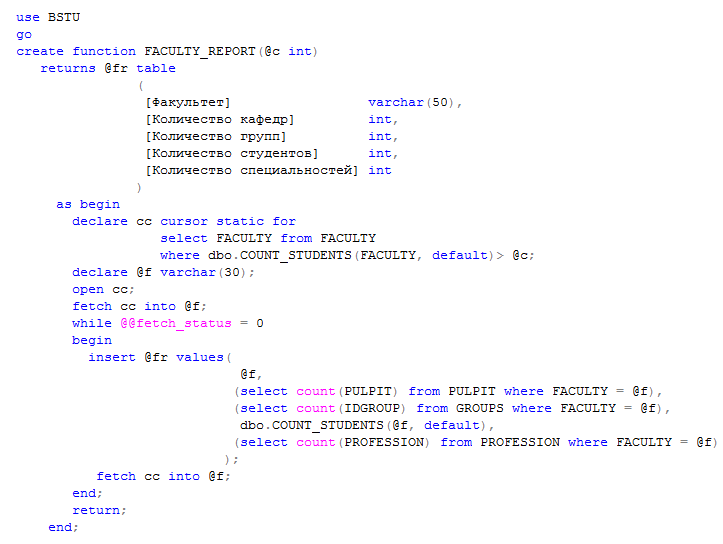
**Встроенные табличные функции**

Встроенные табличные функции возвращают к точке вызова результирующий набор SELECT-запроса и по своим возможностям очень напоминают представления.



**Многооператорные табличные функции**

Многооператорная табличная функция возвращает таблицу, построенную с помощью одного или более операторов Transact-SQL и подобны хранимой процедуре. В отличие от хранимой процедуры, на мультиоператорную табличную функцию можно сослаться в выражении FROM оператора SELECT, как на представление или таблицу.



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

**Индекс** – это объект базы данных, позволяющий **ускорить поиск** в определенной таблице. 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.

Обычно **кластеризованные** индексы создаются автоматически при создании таб-лицы. Наличие первичного ключа (ограничение PRIMARY KEY) влечет создание индекса при создании таблицы. Если при этом в свойствах столбца первичного ключа специально не указано ключевое слово NONCLUSTERED, то создается кластеризованный индекс. Такие индексы физически упорядочены в соответствии со значениями индексируемых столбцов. В таблице может быть только один кластеризованный индекс.

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

use BSTU

exec SP\_HELPINDEX 'AUDITORIUM\_TYPE'

Основное отличие *временных* таблиц от постоянных в том, что они хранятся в системной базе данных TEMPDB и не могут иметь внешние ключи. Локальные временные таблицы имеют имена, начинающиеся с символа # и доступны только пользователю, ее создавшему.

**Некластеризованные** индексы в отличие от кластеризованных никак не влияют на физический порядок строк в таблице. Как и в случае кластеризованных индексов, данные индекса организованы в виде сбалансированного дерева. С таблицей некластеризованный индекс связан указателями на ее строки. Для одной таблицы БД допускается создавать до 1023 некластеризованных индексов.

Индексы, соответствующие ограничению PRIMARY KEY, не обязательно должны быть кластеризованными.

Индексы, созданные по одному столбцу таблицы. Такие индексы называются простыми. MSS допускает создавать индексы по нескольким столбцам – такие индексы называются составными. Составными могут быть кластеризованные и некластеризованные индексы. Например, составному первичному ключу по умолчанию соответствует составной кластерный индекс.

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

**Индекс** – это объект базы данных, позволяющий **ускорить поиск** в определенной таблице. 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.

**Индекс покрытия** запроса позволяет включить в состав индексной строки значения одного или нескольких неиндексируемых столбцов. Например, индекс покрытия #EX\_TKEY\_X включает значения 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

**Некластеризованные индексы**, как и кластеризованные, могут быть **уникальными**. Например, уникальные индексы создаются автоматически при применении ограничения целостности UNIQUE. Установка такого свойства влечет за собой автоматическое создание некластеризованного уникального индекса с именем, заданным с ключевым словом CONSTRAINT.

В некоторых случаях бывает важным, чтобы часть запросов выполнялась особенно быстро. Если эти запросы основаны на WHERE-фильтрации строк, то может быть эффективным применение **фильтруемых некластеризованных индексов**.

Обратите внимание на ключевое слово WHERE и следующее за ним логическое условие в операторе CREATE INDEX: фильтруемый индекс создается только для строк таблицы #EXPLORE, которые удовлетворяют этому логическому условию. Кроме того, заметим, что WHERE-выражение при создании индекса описывает подмножество строк, являющееся надмножеством для всех подмножеств строк, выбираемых запросами.

**Составной индекс**

Такой индекс может содержать более одного столбца. Вы можете включить до 16 столбцов в индекс, но их общая длина ограничена 900 байтами. Как кластеризованный, так и некластеризованный индексы могут быть составными. Составной индекс следует использовать в том случае, когда нужно вести поиск по комбинации значений в нескольких столбцах. Например, предположим, что в таблице *friends* нужно связать составной индекс со столбцами *pname* и *sname*.

**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. Уникальный индекс автоматически создается, когда вы определяете ограничения столбца: первичный ключ или ограничение на уникальность значений.

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

Перестройка и реорганизация индексов

Правильное применение индексов позволяет во многих случаях существенно сократить продолжительность выполнения SELECT-запроса. Однако следует понимать, что эффективность SELECT-запросов достигается снижением эффективности выполнения других DML-операторов. Ясно, что, например, при добавлении строки в таблицу, имеющую некластеризованные индексы, автоматически выполняется изменение в индексах.

Чтобы повысить эффективность выполнения операций изменения таблиц БД, алгоритмы изменения индексов реализованы таким образом, чтобы отложить наиболее трудоемкую их часть на более позднее время. Например, удаление строки в таблице БД приводит к необходимости удалить соответствующие записи во всех индексах данной таблицы. Удаление записей индекса не выполняется физически, а данные страницы индекса при этом не уплотняются с целью ликвидации «дырки», образовавшейся после удаления этой записи – запись просто помечается специальным маркером. Аналогично операции добавления и изменения строк БД могут повлечь образование неиспользуемых фрагментов в области памяти индекса. Поэтому индексы таблиц, часто подвергающихся операциям изменения фрагментированными. Процесс образования неиспользуемых фрагментов памяти называют фрагментацией. Фрагментация индексов снижает эффект от их применения и в некоторых случаях может привести к значительным потерям производительности сервера БД.

Степень фрагментации индекса обычно измеряется в процентах (процентная доля неиспользуемой памяти диска) и может быть получена с помощью запроса

Select name [index], avg\_fragmentation\_in\_percent [fragmentation]  
from sys.dm\_db\_index\_physical\_stats(DB\_I(N’TEMPDB’), object\_id(N’#EXPLORE’), 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;

От фрагментации индекса можно избавиться с помощью простого пересоздания индекса, но таким способом пользуются редко.

Для избавления от фрагментации индекса предусмотрены две специальные операции: реорганизация и перестройка индекса. Обе операции могут выполняться в режиме ONLINE, позволяющем не блокировать таблицу на время построения индекса и допускающем реализацию других запросов к этой таблице. Главное отличие между реорганизацией и перестройкой индекса в глубине перестройки индекса.

**Реорганизация** (REORGANIZE) выполняется значительно быстрее, но после этой операции фрагментация будет убрана только из листовых узлов индексного дерева (на самом нижнем уровне). Поэтому после реорганизации может оставаться некоторая степень фрагментации индекса. Операцию реорганизации индексов рекомендуется выполнять при уровне фрагментации 10–20%. Реорганизация индекса всегда выполняется в режиме ONLINE.

Alter index #index\_name on #table-name reorganize;

Операция **перестройки** (REBUILD) затрагивает все узлы дерева, поэтому после ее выполнения степень фрагментации равна нулю. Перестройку индексов рекомендуется выполнять при уровне фрагментации, превышающем 20%. По умолчанию операция перестройки индекса выполняется в режиме OFFLINE и полностью блокирует индексируемую таблицу для доступа других запросов, но с помощью специального параметра может быть установлен режим ONLINE.

Alter index #index\_name on #table rebuild with (online=off);

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

Включение и отключение

**Отключение** индекса препятствует доступу пользователя к индексу и для кластеризованных индексов к данным базовой таблицы. Определение индекса остается в метаданных, а статистика индекса хранится в некластеризованных индексах. Отключение некластеризованного или кластеризованного индекса в представлении физически удаляет данные индекса. Отключение кластеризованного индекса в таблице препятствует доступу к данным; данные по-прежнему остаются в таблице, но недоступны для операций на языке манипулирования данными (DML) до тех пор, пока индекс не будет удален или перестроен.

ALTER INDEX имя\_индекса\_из\_sys.indexes.name ON имя\_таблицы\_из\_sys.tables DISABLE;

-- Disables all indexes on the HumanResources.Employee table.

ALTER INDEX ALL ON HumanResources.Employee

DISABLE;

После того, как индекс отключен, он остается в отключенном состоянии, пока не будет перестроен или удален

ALTER INDEX IX\_Employee\_OrganizationLevel\_OrganizationNode ON HumanResources.Employee

REBUILD;

ALTER INDEX ALL ON HumanResources.Employee REBUILD;

**47. План запроса. Оптимизация запросов. Подсказки оптимизатора.**

**Оптимизация**. Специальная компоненте сервера – оптимизатор. Основная задача оптимизатора – построение плана запроса План запроса представляет собой алгоритм выполнения SQL-запроса.

**План запроса**. Для каждого шага вычисляется стоимость – величина, пропорциональная продолжительности выполнения шага. Суммарная стоимость шагов плана составляет стоимость всего запроса, является величиной, пропорциональной продолжительности выполнения запроса.

Критерием оптимизации при построении плана является минимизация общей стоимости запроса. Оптимизатор, формирующий план запроса, учитывает статистику, собираемую сервером СУБД, а также наличие специальных объектов БД, предназначенных для ускорения выполнения запросов и называемых индексами.

План выполнения

* Compute Scalar вычисляет выражение, выдавая в результате скалярное значение
* Clustered Index Seek выполняет поиск строк по кластеризованным индексам
* Index Scan - обрабатываются все листья страницы дерева индексов

Оптимизация запроса

* анализ запроса
* выбор индекса
* выбор порядка выполнения операций соединения
* выбор метода выполнения операций соединения

**Подсказки оптимизации** являются частью инструкции SELECT, которые указывают оптимизатору запросов, что нужно выполнять данную инструкцию определенным образом

Hints

* табличные подсказки (INDEX, NOEXPAND, FORCESEEK)
* подсказки соединения (FORCE ORDER, LOOP, HASH, MERGE)
* подсказки запросов (FAST n, OPTIMIZE FOR, OPTIMIZE FOR UNKNOWN, USE PLAN)
* структуры планов (sp\_create\_plan\_guide, sys.plan\_guides)

Анализ запроса

* Наличие аргументов поиска
* Использование оператора OR
* Существование критериев соединения

Аргумент поиска — это часть запроса, которая ограничивает промежуточный результирующий набор запроса:

* name = ‘Иванов С.П.'
* capacity >= 100
* name = ‘Иванов С.П.‘ AND idgroup = 512

Нельзя использовать в качестве аргументов поиска:

* Выражение с оператором отрицания NOT
* <>

Выражение

* NOT IN('d1', 'd2')
* aud\_no <> 9031
* capacity \* 0.6 > 100

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



Обработка SQL-запроса, поступившего серверу MSS, проходит в несколько **этапов**: разбор, разрешение имен, оптимизация, компиляция и выполнение

Обработка любого SQL-запроса начинается с синтаксического разбора текста запроса для проверки его на соответствие правилам языка. На этапе разрешения имен осуществляется проверка наличия (возможность доступа) используемых в запросе объектов БД (таблиц, столбцов таблиц, пользовательских и встроенных функций и пр.).

Запрос, прошедший этапы разбора и разрешения имен, поступает на обработку специальной компоненте сервера, называемой оптимизатором. Основная задача оптимизатора – построение плана запроса. План запроса представляет собой алгоритм выполнения SQL-запроса.

При построении плана запроса для каждого шага вычисляется числовая величина, пропорциональная продолжительности выполнения шага, называемая стоимостью. Суммарная стоимость шагов плана составляет **стоимость всего запроса** и, соответственно, является величиной, пропорциональной продолжительности выполнения запроса. Критерием оптимизации при построении плана является минимизация общей стоимости запроса. Оптимизатор, формирующий план запроса, учитывает статистику, собираемую сервером СУБД, а также наличие специальных объектов БД, предназначенных для ускорения выполнения запросов и называемых индексами.

После построения плана запроса осуществляется его компиляция. Откомпилированный запрос (точнее план запроса), помещается в специальную область памяти, называемую библиотечным кэшем. Кэш потом используется для ускорения выполнения будущих аналогичных запросов. Откомпилированный план выполняется (точнее интерпретируется) сервером СУБД.

Predicate: предикат - это условие, которое оценивается как истина или ложь. Предикаты используются в предложении WHERE или в JOIN запросов к базе данных.

Selectivity: **селективность** - это доля строк в получаемом предикатом наборе данных, которые удовлетворяют условию этого предиката. Также встречаются более сложные определения селективности, необходимые для оценки числа строк, вовлечённых в объединения, DISTINCT и другие операторы.

Индекс успешно работает при <= 5%.

Не нужен индекс при 80% или более

**Плотность** - это информация о числе дубликатов в анализируемом столбце или комбинации столбцов, и она вычисляется, 1 / (число различающихся значений).

Плотность запроса – количество возвращаемых строк запроса

**49. Триггеры. Типы триггеров. Создание after-триггера.**

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

Различают **два вида DDL-триггеров**: триггеры уровня сервера (ALLSERVER), триггеры уровня БД (DATABASE). Триггеры уровня сервера могут обрабатывать события сервера СУБД (создание, изменение или удаление объектов сервера, подключение к серверу и пр.). Триггеры уровня базы данных предназначены для обработки событий, происходящих в рамках БД.

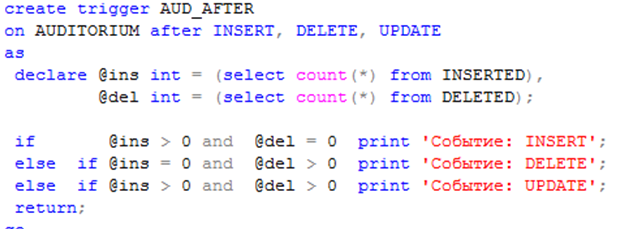
*DDL-триггеры уровня сервера*.Триггеры уровня сервера предназначаются, прежде всего, для отслеживания настроек сервера, а также для обеспечения его безопасности. Условно триггеры уровня сервера можно разбить на две группы: триггеры для обработки событий, связанных с созданием, изменением или удалением объектов сервера, и триггеры, обрабатывающие событие подключения к серверу.

Каждый **DML-триггер** связан с таблицей или представлением и предназначен для обработки одного или нескольких событий, соответствующих трем операторам, изменяющим содержимое таблицы: INSERT, UPDATE и DELETE. MSS поддерживает два типа DML-триггеров: AFTER и INSTEAD OF.

*AFTER-триггеры*.Триггеры типа AFTER исполняются после выполнения оператора, вызвавшего соответствующее событие.

при создании триггера указывается: 1) имя таблицы (после ключевого слова ON), с которой этот триггер связан; 2) событие, обработку которого осуществляет триггер.

Один AFTER-триггер может реагировать на несколько событий. В этом случае после ключевого слова AFTER должны быть перечислены все необходимые события.



Этот триггер будет вызван после выполнения одного из операторов INSERT, UPDATE или DELETE.

В рамках выполнения триггера разработчику доступны две псевдотаблицы с именами INSERTED и DELETED. В зависимости от типа события, активизировавшего триггер, содержимое таблиц разное.

Событие INSERT приводит к тому, что в таблицу INSERTED помещаются строки, добавленные оператором INSERT, вызвавшим это событие. При этом таблица DELETED остается пустой.

При событии DELETE в таблицу DELETED копируются удаленные строки, а таблица INSERTED остается пустой.

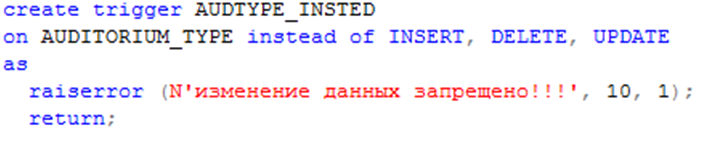
При изменении строк таблицы с помощью оператора UPDATE заполняются обе псевдотаблицы. При этом INSERTED содержит обновленные версии строк, а таблица DELETED версию строк до их изменения.

AFTER-триггер вызывается после выполнения активизирующего его оператора. Поэтому, если оператор нарушает ограничение целостности, то возникшая ошибка не допускает выполнения этого оператора и соответствующих триггеров. В AFTER-триггере допускается применения функции RAISERROR и TCL-операторов COMMIT и ROLLBACK. Для нескольких AFTER-триггеров, реагирующих на одно и то же событие, имеется возможность частично упорядочить их выполнение с помощью системной процедуры SP\_SETTRIGGERORDER.

*INSTEAD OF-триггеры.* Триггеры типа INSTEAD OF исполняются вместо оператора, вызвавшего соответствующее событие.

**50. Триггеры. Создание и назначение instead of-триггеров. Таблицы inserted, deleted.**

*INSTEAD OF-триггеры*.Триггеры типа INSTEAD OF исполняются вместо оператора, вызвавшего соответствующее событие.



для таблицы или представления допускается создание только по одному INSTEAD OF-триггеру, реагирующему на каждое событие. Кроме того, выполнение INSTEAD OF-триггера предшествует проверке установленных для таблицы ограничений целостности.

Если для таблицы или представления созданы INSTEADOF и AFTER-триггеры, реагирующие на одно и то же событие, то выполниться только INSTEAD OF- триггер.

В рамках выполнения триггера разработчику доступны две **псевдотаблицы** с именами INSERTED и DELETED. В зависимости от типа события, активизировавшего триггер, содержимое таблиц разное.

Событие INSERT приводит к тому, что в таблицу INSERTED помещаются строки, добавленные оператором INSERT, вызвавшим это событие. При этом таблица DELETED остается пустой.

При событии DELETE в таблицу DELETED копируются удаленные строки, а таблица INSERTED остается пустой.

При изменении строк таблицы с помощью оператора UPDATE заполняются обе псевдотаблицы. При этом INSERTED содержит обновленные версии строк, а таблица DELETED версию строк до их изменения.

**51. Транзакции. Свойства ACID.**

**Транзакция** – механизм БД, позволяющий объединять несколько операторов, изменяющих БД таким образом, чтобы при выполнении этой совокупности операторов они или все выполнились, или все не выполнились.

Логическая единица работы, обеспечивающая переход базы данных из одного согласованного состояния в другое согласованное состояние

Обычно при определении понятия транзакции используют перечень требований к этому механизму, сформулированный в конце 70-х гг. Джимом Греем в виде аббревиатуры ACID: Atomicity – атомарность, Consistency – согласованность, Isolation – изолированность, Durability – долговечность.

**Атомарность** транзакции уже пояснялась выше на примере банковской операции. Она заключается в гарантии того, что никакая транзакция не будет зафиксирована в БД частично: операторы изменения БД, включенные в транзакцию, либо выполнятся все, либо не выполнится ни один. В БД это свойство реализуется с помощью механизма отката, позволяющего отменить выполненные, но незафиксированные изменения БД.

Свойство **согласованности** заключатся в том, что транзакция должна фиксировать новое согласованное состояние БД. Это свойство тоже пояснялось на примере банковской операции. Здесь лишь еще раз отметим, что понятие согласованности зависит от предметной области, состояние которой отражается в БД.

Современные СУБД допускают одновременную (часто говорят – параллельную) работу нескольких транзакций. При этом параллельные транзакции могут изменять одни и те же данные в БД. Свойство **изолированности** гарантирует с точностью до определенного уровня отсутствие взаимного влияния параллельных транзакций на результаты их выполнения. Современные СУБД могут обеспечивать несколько уровней изолированности транзакций. Уровни изолированности транзакций, поддерживаемые MSS, будут рассматриваться позже.

Любая транзакция завершается либо фиксацией (оператор COMMIT), либо откатом (оператор ROLLBACK). Откат – это отмена всех изменений в БД, выполненных в рамках транзакции. Фиксация – подтверждение успешного завершения транзакции. Свойство **долговечности** заключается в том, что изменения в БД, выполненные и зафиксированные транзакцией, не могут быть отменены, кроме как с помощью новой транзакции.

**52. Транзакции. Уровни изолированности транзакций. Функция TRANCOUNT.**

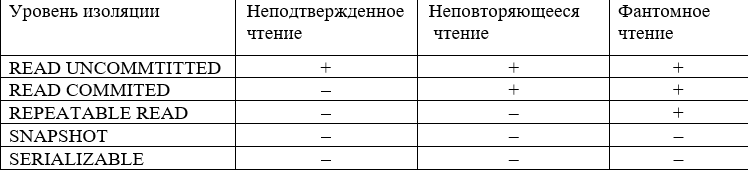
Транзакция – механизм БД, позволяющий объединять несколько операторов, изменяющих БД таким образом, чтобы при выполнении этой совокупности операторов они или все выполнились, или все не выполнились.

Логическая единица работы, обеспечивающая переход базы данных из одного согласованного состояния в другое согласованное состояние

Под **изолированностью** подразумеваются ограничения на одновременное чтение или изменение данных в параллельно работающих транзакциях.

MSS поддерживает пять уровней изолированности транзакций: READ UNCOMMITTED, READ COMMITTED, REPEATABLE READ, SNAPSHOT и SERIALIZABLE. Для установки уровня изолированности текущей для транзакций в текущем подключении служит инструкция SET TRANSACTION ISOLATION LEVEL.

Обычно уровни изолированности транзакций рассматриваются относительно трех проблем параллельного доступа: неподтвержденное чтение, неповторяющееся чтение, фантомное чтение.



Пессимистическая модель:

READ UNCOMMITTED

* Не изолирует операции чтения других транзакций
* Транзакция не задает и не признает блокировок
* проблемы: Грязное чтение 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

Оптимистическая модель:

SNAPSHOT

Обе модели:

READ COMMITTED

* Транзакция выполняет проверку только на наличие монопольной блокировки для данной строки
* Является уровнем изоляции по умолчанию
* Проблемы: Неповторяемое чтение Фантомное чтение

**@@TRANCOUNT** содержит число активных транзакций для текущего пользователя

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

**Блокировки** – механизм обеспечения согласованности данных в случае одновременного обращения к данным нескольких пользователей

Свойства:

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

Режим блокировки

* Разделяемая (shared lock)
* Монопольная (exclusive lock)
* Обновления (update lock)

Гранулярность блокировки определяет, какой объект блокируется (строки, страницы, индексный ключ или диапазон индексных ключей, таблицы, экстент, база данных). СУБД выбирает гранулярность блокировки автоматически

Режим блокировки

СУБД автоматически выбирает соответствующий режим блокировки, в зависимости от типа операции (чтение или запись)

**Разделяемая** блокировка резервирует ресурс только для чтения

Другие процессы не могут изменять заблокированный ресурс

Может быть несколько разделяемых блокировок

**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 **обновлений**

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

Процесс преобразования большого числа блокировок уровня строки, страницы или индекса в одну блокировку уровня таблицы называется **эскалацией** блокировок (lock escalation)

ALTER TABLE

SET (LOCK\_ESCALATION = {TABLE | AUTO | DISABLE})

Подсказки блокировок (locking hints)

SET LOCK\_TIMEOUT - период в миллисекундах, в течение которого транзакция будет ожидать снятия блокировки с объекта (-1 по умолчанию, не установлен)

Sys.dm\_tran\_locks

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

**54. Операторы TCL. Назначение привилегий.**

Операторы TCL предназначены для создания транзакций.

TCL SQL включает четыре оператора: BEGIN TRAN, SAVE TRAN, COMMIT TRAN и ROLLBACK TRAN

Оператор BEGIN TRAN указывает на начало транзакции.

Предполагается, что за оператором BEGIN TRAN будут следовать один или несколько DML-операторов, входящих в транзакцию.

Снизу группу DML-операторов, входящую в транзакцию, ограничивает один из операторов: COMMIT TRAN или ROLLBACK TRAN.

Оператор COMMIT TRAN фиксирует транзакцию – подтверждает ее успешное завершение.

Оператор ROLLBACK TRAN откатывает транзакцию – отменяет все изменения, произведенные DML-операторами в таблицах БД в рамках данной транзакции.

Оператор SAVE TRAN применяется для формирования контрольной точки – промежуточного состояния транзакции, к которому может быть осуществлен откат. В тексте SQL-скрипта оператор SAVE TRAN должен находиться между операторами BEGIN TRAN и ROLLBACK TRAN

Роль – именованный набор привилегий. **Привилегия** – право на выполнение.

Разрешения пользовательских ролей базы данных можно настроить с помощью операторов GRANT, DENY и REVOKE (DCL).

Оператор GRANT предназначен для выдачи разрешений. После ключевого слова GRANT следует список разрешений, который будет назначен принципалу, имя которого указывается после ключевого слова TO.

Grant список on объект to принципал

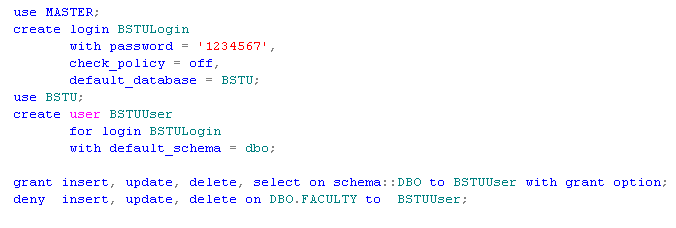
Отобрать разрешение у принципала можно с помощью оператора REVOKE. После ключевого слова REVOKE следует список разрешений, который будет отобран у принципала, имя которого указывается после ключевого слова FROM.

Revoke список on объект from

Оператор DENY не входит в состав стандарта SQL и может применяться только в рамках Microsoft SQL Server. Он позволяет запретить разрешения, выданные ранее. После ключевого слова DENY следует список разрешений, который будет запрещен у принципала, имя которого указывается после ключевого слова TO.

Deny список on объект to принципал

Дальше вода/хуйня

****

с помощью оператора CREATE LOGIN создается объект типа Login (учетная запись сервера, принципал) с именем **BSTULogin.** с помощью оператора CREATE USER создается объект типа USER (пользователь БД, принципал) с именем **BSTUUser**, который с помощью ключевых слов FOR LOGIN ссылается на объект типа Login c именем **BSTULogin**. Оператор GRANT разрешает выполнять пользователю BSTUUser все DML-операторы применительно ко всем объектам схемы с именем DBO.

Следующий за оператором GRANT оператор DENY запрещает выполнять DML-операторы применительно к таблице DBO.FACULTY, находящейся в схеме DBO.

**55. Привилегии. Роли. Аутентификация и авторизация**

**Роль** – именованный набор привилегий. **Привилегия** – право на выполнение.

В SQL Server роли используются для группирования пользователей с одинаковыми разрешениями. Любой пользователь отдельной роли наследует все разрешения данной роли, а изменения разрешений роли изменяют разрешения всех пользователей данной роли. Поэтому, чтобы одновременно изменить разрешения для большого количества пользователей, следует изменить разрешения их роли.

используются два типа ролей: серверные и роли базы данных. Серверные роли управляют доступом к операциям, которые влияют на сервер SQL Server, например, запуск и остановка сервера, управление системой безопасности и создание баз данных. Роли баз данных управляют операциями и доступом к данным указанной базы данных.

Безопасность данных является одной из важнейших задач при хранении информации в СУБД. В языке SQL используются следующие основные принципы защиты данных:

* Все операции над данными выполняются от имени конкретного пользователя. СУБД может отказаться выполнить запрашиваемые действия в зависимости от того, какой пользователь запрашивает это действие.
* СУБД разрешает пользователям осуществлять действия над одними объектами и запрещает делать это с другими.
* В SQL используется система привилегий, то есть прав пользователя на проведение тех или иных действий над определенным объектом БД.

В базу данных вводится новый тип – пользователи. Каждому пользователю присваивается уникальный идентификатор. Пользователи могут быть объединены в группы. Один пользователь может входить в несколько групп. В стандарте вводится понятие группы PUBLIC, для которой должен быть определен стандартный минимальный набор прав. По умолчанию предполагается, что каждый новый пользователь, если специально не указано иное, относится к группе PUBLIC.

Роль – это поименованный набор полномочий.

Существует ряд стандартных ролей, которые определены в момент установки сервера БД. Имеется возможность создавать новые роли, группируя в них произвольные полномочия. Введение ролей позволяет упростить управление привилегиями пользователей, структурировать этот процесс.

В самом общем виде концепция обеспечения безопасности БД заключается в следующем: необходимо поддерживать проверку подлинности (**аутентификацию**) пользователя и проверку полномочий или прав пользователя. Проверка подлинности (**аутентификация**) означает достоверное подтверждение того, что пользователь, пытающийся выполнить некоторое действие действительно тот, за кого себя выдает.

Проверка полномочий основана на том, что каждому пользователю информационной системы соответствует набор действий, которые он может выполнять по отношению к определенным объектам.

Операторы DCL (GRANT, REVOKE и DENY) предназначены для управления процессом авторизации. **Авторизация** – это процедура проверки разрешений (permission) на выполнения определенных операций. Авторизация может выполняться на двух уровнях: на уровне сервера или на уровне БД. Каждому уровню соответствует свой перечень операций, которыми можно управлять с помощью DCL.

Разрешениями могут обладать специальные объекты сервера или БД, которые объединены под общим названием – принципалы. Или иначе, принципал – это объект сервера или БД, которому может быть выдано разрешение на выполнение операции, а также отобрано или запрещено разрешение.

Все операции на уровне сервера или БД выполняются от имени принципалов. Например, при подключении к серверу любой пользователь сервера обязан пройти процедуру аутентификации (проверку подлинности), которая в простейшем случае сводится к вводу имени и пароля. После этого все действия пользователя на уровне сервера выполняются от имени объекта сервера, имеющего тип Login (учетная запись), а на уровне БД от имени объекта, имеющего тип User (пользователь БД). Объекты типов Login и User – принципалы, которым могут быть выданы разрешения.

**56. Основы безопасности в SQL Server. Встроенные роли сервера и базы данных.**

Безопасность данных является одной из важнейших задач при хранении информации в СУБД. В языке SQL используются следующие основные принципы защиты данных:

* Все операции над данными выполняются от имени конкретного пользователя. СУБД может отказаться выполнить запрашиваемые действия в зависимости от того, какой пользователь запрашивает это действие.
* СУБД разрешает пользователям осуществлять действия над одними объектами и запрещает делать это с другими.
* В SQL используется система привилегий, то есть прав пользователя на проведение тех или иных действий над определенным объектом БД.

Роль – именованный набор привилегий. Привилегия – право на выполнение.

В базу данных вводится новый тип объектов – пользователи. Каждому пользователю присваивается уникальный идентификатор.

Пользователи могут быть объединены в группы. Один пользователь может входить в несколько групп. В стандарте вводится понятие группы PUBLIC, для которой должен быть определен стандартный минимальный набор прав. По умолчанию предполагается, что каждый новый пользователь, если специально не указано иное, относится к группе PUBLIC.

По умолчанию роль public разрешает пользователям выполнять следующие действия:

* просматривать системные таблицы и отображать информацию из системной базы данных master, используя определенные системные процедуры;
* выполнять инструкции, для которых не требуются разрешения, например, PRINT.

Существует ряд стандартных ролей, которые определены в момент установки сервера БД. Имеется возможность создавать новые роли, группируя в них произвольные полномочия. Введение ролей позволяет упростить управление привилегиями пользователей, структурировать этот процесс.

В самом общем виде концепция обеспечения безопасности БД заключается в следующем: необходимо поддерживать проверку подлинности (аутентификацию) пользователя и проверку полномочий или прав пользователя. Проверка подлинности (аутентификация) означает достоверное подтверждение того, что пользователь, пытающийся выполнить некоторое действие действительно тот, за кого себя выдает.

Проверка полномочий основана на том, что каждому пользователю информационной системы соответствует набор действий, которые он может выполнять по отношению к определенным объектам.

СУБД в своих системных каталогах хранит как описание самих пользователей, так и описание их привилегий по отношению ко всем объектам.

Членами роли базы данных могут быть любые из следующих:

* группы и учетные записи Windows;
* регистрационные имена входа в SQL Server;
* другие роли.

**Фиксированные серверные роли** определяются на уровне сервера и поэтому находятся вне баз данных, принадлежащих серверу баз данных.

Sysadmin – Выполняет любые действия в системе баз данных

serveradmin Конфигурирует параметры сервера

setupadmin Устанавливает репликацию и управляет расширенными процедурами

securityadmin Управляет регистрационными именами и разрешениями для инструкции CREATE DATABASE и чтением журналов логов

processadmin Управляет системными процессами

dbcreator Создает и модифицирует базы данных

diskadmin Управляет файлами на диске

Членов фиксированной серверной роли можно добавлять и удалять двумя способами:

* используя среду Management Studio (папку "Security", а в ней разверните папку "Logins");
* используя инструкции языка Transact-SQL (CREATE SERVER ROLE и DROP SERVER ROLE, ALTER SERVER ROLE)

CREATE SERVER ROLE program\_admin;

ALTER SERVER ROLE program\_admin ADD MEMBER Vasya;

Фиксированные серверные роли нельзя добавлять, удалять или переименовывать. Кроме этого, только члены фиксированных серверных ролей могут выполнять системные процедуры для добавления или удаления регистрационного имени в роли.

Регистрационное имя sa является регистрационным именем системного администратора. Это регистрационное имя всегда является членом фиксированной серверной роли sysadmin и его нельзя удалить из этой роли. Регистрационное имя sa следует использовать только в тех случаях, когда нет другого способа войти в систему базы данных.

**Фиксированные роли базы данных** определяются на уровне базы данных и поэтому существуют в каждой базе данных, принадлежащей серверу баз данных.

db\_owner Пользователи, которые могут выполнять почти все действия в базе данных

db\_accessadmin Пользователи, которые могут добавлять и удалять пользователей

db\_datareader Пользователи, которые могут просматривать данные во всех таблицах пользователей базы данных

db\_datawriter Пользователи, которые могут добавлять, изменять или удалять данные во всех пользовательских таблицах базы данных

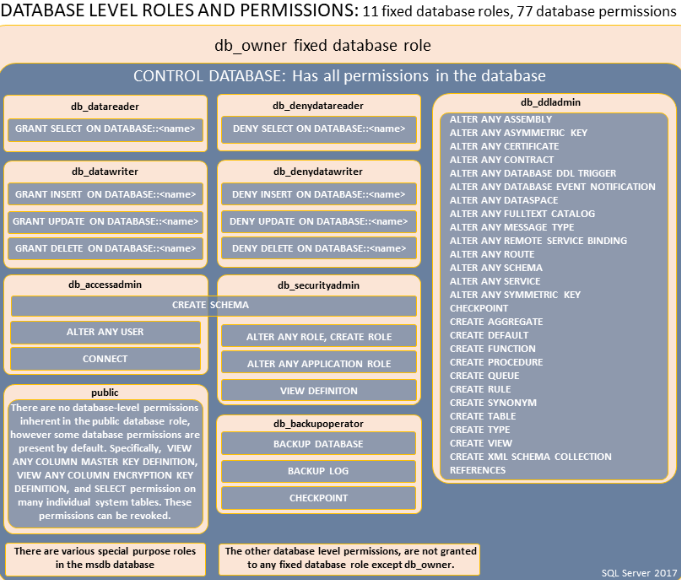
db\_ddladmin Пользователи, которые могут выполнять инструкции DDL в базе данных

db\_securityadmin Пользователи, которые могут управлять всеми действиями в базе данных, связанными разрешениями безопасности

db\_backupoperator Пользователи, которые могут выполнять резервное копирование базы данных

db\_denydatareader Пользователи, которые не могут просматривать любые данные в базе данных

db\_denydatawriter Пользователи, которые не могут изменять никакие данные в базе данных



**57. Язык XML в SQL Server. Секция for XML оператора SELECT. Директивы PATH, AUTO, RAW, ELEMENTS, ROOT.**

XML – Extensible Markup Language

Язык XML задает общие правила, по которым создаются теги (элементы языка разметки, предназначенные для разделения данных в текстовых файлах) и оформляются XML-документы. При этом XML-документ представляет собой текстовый файл, содержащий данные, разделенные (размеченные) тегами, составленными по правилам XML и расположенными в правильном (в соответствии с правилами XML) порядке. Совокупность XML-тегов, предназначенных для разметки документов определенного типа называют словарями XML или XML-реализациями.

Начинается специальным тегом с именем ?xml (объявление XML)

Каждый элемент начинается открывающим тегом и завершается закрывающим

Могут быть вложенные теги.

Tег XML - namespace:name

URI - Uniform Resource Identifier — унифицированный идентификатор ресурса

Правильно построенный документ – well-formed – соответствует синтаксическим правилам XML

Валидный документ – valid – соответствует правилам описания типа документа (Document Type Definition, DTD)

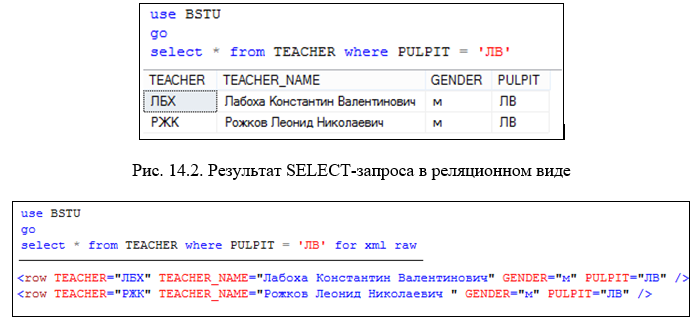
* наличие корневого элемента
* каждый открывающий тег имеет соответствующий закрывающий тег
* правильное вложение элементов документа
* атрибут должен иметь значение, которое берется в кавычки

XML-формат часто используется для обмена данными между компонентами одной информационной системы или на межсистемном уровне. Здесь проявляются главные преимущества XML: 1) все программные платформы «понимают» XML одинаково; 2) практически все программные системы имеют встроенные механизмы, позволяющие эффективно обрабатывать XML-данные.

Для преобразования результата SELECT-запроса в формат XML в операторе SELECT применяется **секция FOR XML.** При этом можно использовать один из четырех режимов: RAW, AUTO, PATH и EXPLICIT.

**Режим RAW**.По умолчанию 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 (из списка SELECT), а значения атрибутов равны их значениям.

Обратите внимание: 1) результатом выполнения SELECT-запроса является XML-фрагмент, а не XML-документ; 2) данные из столбцов таблицы представляются в виде значений атрибутов (такая форма представления данных называется атрибутной).



Для того чтобы изменить стандартное имя элемента **raw** на заданное, достаточно его указать в скобках после ключевого слова RAW

Построить корневой элемент для XML-фрагмента можно с помощью ключевого слова ROOT, которое может быть применено в инструкции FOR XML

Если в сформированных XML-элементах требуется, чтобы имена атрибутов отличались от имен столбцов исходных таблиц, следует в SELECT-списке для столбцов указать псевдонимы, которые будут использованы в качестве имен атрибутов

Для того чтобы сформировать результирующий набор в альтернативном, называемом элементном виде, необходимо в инструкции FOR XML применить ключевое слово ELEMENTS

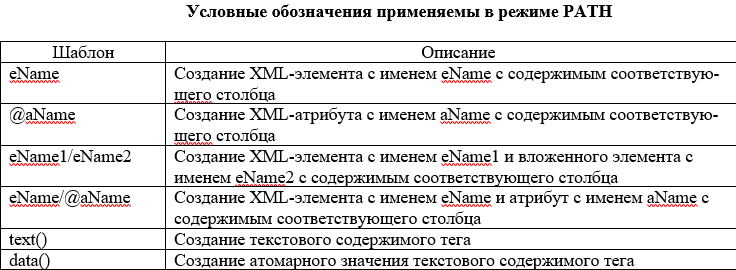
**Режим AUTO**.Результат, полученный в режиме AUTO для простых SELECT-запросов, похож на результат, полученный в режиме RAW. Основное отличие – в качестве имени элемента, соответствующего строке исходной таблицы, используется ее имя

Особенность режима AUTO проявляется в многотабличных запросах. В этом случае режим AUTO позволяет построить XML-фрагмент с применением вложенных элементов. Причем порядок вложенности основывается на порядке столбцов (слева направо), указанных в SELECT-списке.

Как и в режиме RAW, здесь допускается применение ключевых слов ROOT (указывает имя корневого элемента) и ELEMENTS (формирование элементного вида XML-фрагмента).

**Режим PATH**.Режим PATH позволяет разработчику наиболее полным образом управлять процессом формирования XML-структуры. Каждый столбец конфигурируется независимо с помощью заданного в формате XPATH [7] имени псевдонима этого столбца. В таблице 14.1 приведено несколько шаблонов, которые могут быть использованы при записи псевдонимов столбцов в режиме PATH.

Если псевдонимы задавать традиционным способом (это соответствует первому шаблону в таблице 14.1), то сформируется XML-фрагмент, состоящий из элементов с именем **row**,соответствующих строкам результирующего набора.Каждый **row**-элемент включает в себя все значения столбцов одной строки в элементной форме. При этом элементы будут иметь имена, совпадающие с заданными в SELECT-списке псевдонимами (рис. 14.13).



в этом режиме при формировании результата у разработчика появляется возможность сочетать атрибутную и элементную форму.

**58. Язык XML в SQL Server. Применение системных процедур sp\_xml\_preparedocument, sp\_xml\_removedocument. Применение OPENXML.**

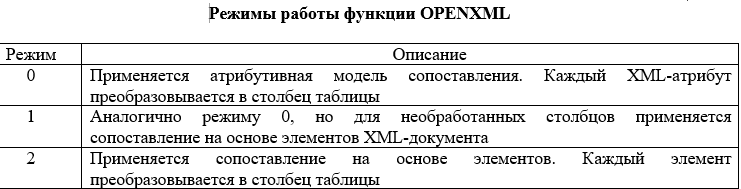
Для преобразования XML-данных в строки таблицы предназначена функция **OPENXML**. Она принимает три входных параметра: дескриптор, сформированный системной хранимой процедурой с именем SP\_XML\_PREPAREDOCUMENT; выражение XPATH [7] и целое положительное число, определяющее режим работы функции.

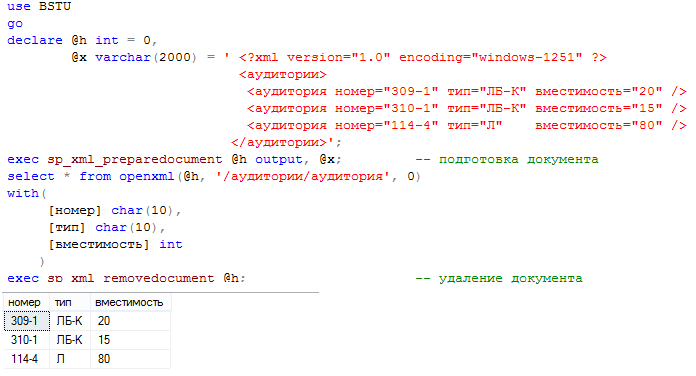
Процедура **SP\_XML\_PREPAREDOCUMENT** должна быть выполнена до SELECT-запроса, применяющего OPENXML. Процедура принимает в качестве входного параметра XML-документ (в формате строки) и возвращает дескриптор, который впоследствии применяется функцией OPENXML.

Выражение XPATH, принимаемое функцией OPENXML в качестве второго параметра, предназначено для выбора требуемых данных из исходного (введенного процедурой SP\_XML\_PREPAREDOCUMENT) XML-документа.

Последний, третий, параметр функции OPENXML указывает на тип преобразования (режим). Основные базовые типы преобразования приведены в табл. 14.2.

Кроме того, с помощью выражения WITH должна быть указана структура формируемого результирующего набора.





Обратите внимание: 1) из XML-фрагмента с помощью XPATH-выражения выделяется множество тегов с именами аудитория; 2) применяется режим работы 0, подразумевающий атрибутивное сопоставление; 3) имена столбцов в выражении WITH совпадают с именами атрибутов в XML-структуре; 4) после применения функции OPENXML должна быть выполнена системная процедура SP\_XML\_REMOVEDOCUMENT, освобождающая использованные OPENXML ресурсы.

Для того, чтобы явно указать сопоставление XML-структуры со структурой стоки результирующего набора, можно в выражении WITH каждого столбца результирующего набора указать XPATH-выражение, определяющее содержимое этого столбца

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

язык DTD - Document Type Definition

Набор правил для структурирования XML

Внутренний DTD – часть XML-документа

Внешний DTD – адрес URL

язык XML Schema

DTD – Document Type Definition

Имя DTD должно соответствовать имени тега корневого элемента XML-документа

XML-документ нужно связать с соответствующим файлом DTD

Объявления типов элементов должны начинаться с инструкции ELEMENT

Порядок элементов XML-документа

Элементы без подчиненных - #PCDATA

\* наличие элементов (от нуля и больше)

? наличие не более одного элемента

+ наличие по крайней мере одного элемента

Объявление атрибута <!ATTLIST имя атрибута и тип данных>

#IMPLIED атрибут необязательный, #REQUIRED обязательный

* Атрибут типа ID - определение уникального значения
* Атрибут типа IDREF должен ссылаться на действительный идентификатор, объявленный в этом же документе
* Атрибут типа IDREFS задает список разделенных пробелами строк, на которые ссылаются значения атрибута типа ID

XML Schema — язык описания структуры XML-документа – предназначен для определения правил, которым должен подчиняться документ

Создается модель данных документа:

словарь (названия элементов и атрибутов)

модель содержания (отношения между элементами и атрибутами и их структура)

типы данных

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.

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

**тип данных xml**

Database Engine внутренне сохраняет значения xml как большой двоичный объект. Без индекса для таких объектов во время выполнения запроса осуществляется декомпозиция, что может занять много времени. Поэтому причиной выполнять индексирование столбцов xml является увеличение производительности запроса.

Система поддерживает первичный индекс xml и три типа вторичных индексов xml. Первичный индекс xml индексирует все теги, значения и пути в экземплярах XML столбца xml. Запросы используют первичный индекс xml для получения скалярных значений или вложенных деревьев xml.

Первичный индекс xml использует экземпляр xml для генерации вне его соответствующей внутренней реляционной формы. Таким путем исключается повторяющаяся генерация внутренней формы для запросов и обновлений.

Для дальнейшего улучшения производительности выборки данных вы можете создавать вторичные индексы xml. Первичный индекс xml должен существовать до создания вторичных индексов. Вы можете создавать три типа вторичных индексов xml с использованием различных ключевых слов:

♦ for path создает вторичный индекс xml для структуры документа;

♦ for value создает вторичный индекс xml для значений элемента и атрибута столбца xml;

♦ for property создает вторичный индекс xml, который отыскивает свойство.

**XPath** (XML Path Language) — язык запросов к элементам XML-документа. Для идентификации набора элементов с помощью выражения xPath используется путь в структуре дерева к этим элементам, дескрипторы разделяются косой чертой

/ROOT/pizdec/@ebal

xPath выбирает набор элементов, а не отдельный элемент

Перед именем атрибутов стоит символ @

**Методы типа данных xml. тип данных xml**

В тех случаях, если данные являются составными (состоящими из нескольких компонент), и при этом не имеют фиксированный формат, удобно использовать XML для их структуризации. Например, если в таблице хранится информация о гражданах разных стран, то структура их почтового адреса может значительно отличаться в зависимости от страны проживания. Или, например, информация об образовании людей (учебные заведения, даты начала и завершения обучения, специальности, квалификации и пр.) будут различными у разных людей.

С помощью оператора SELECT данные XML-типа могут быть найдены и извлечены из БД. В простейшем случае извлекается полностью XML-структура

Часто требуется извлечь не полностью хранящуюся в БД XML-структуру, а лишь ее подмножество. Для этого можно воспользоваться методами XML-типа (табл. 14.3). В качестве параметра методы принимают строку, содержащую выражение на языках XPATH и XQUERY [7], позволяющие либо выделить, либо изменить фрагменты хранящегося в столбце XML-документа

