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

Служба Database Engineявляется ядром системы управления реляционной БД. Один сервер Database Engine может обеспечить доступ к нескольким БД. При этом каждый сервер в своей работе использует пять системных БД.

|  |  |
| --- | --- |
| Системная  база данных | Назначение |
| master | Хранит все системные данные экземпляра Database Engine, а также информацию обо всех других БД, управляемых экземпляром |
| 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 экземпляра Database Engine и пользовательских временных таблиц. База данных пересоздается при каждой перезагрузке экземпляра |
| 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 |

**2.Группы операторов SQL: DDL, DML, DCL, TCL**

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

Операторы DDL предназначены для создания, удаления и изменения объектов БД или сервера СУБД

• CREATE

• ALTER

• DROP

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

Операторы DML предназначены для работы с таблицами

• SELECT

• INSERT

• DELETE

• UPDATE

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

Операторы TCL предназначены для управления транзакциями

Транзакция – это несколько DML-операторов, которые либо все выполняются, либо все не выполняются.

• BEGIN TRAN

• SAVE TRAN создание контрольной точки

• COMMIT TRAN фиксация

• ROLLBACK TRAN откат

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

Операторы DCL предназначены для управления процессом авторизации

Авторизация – это процедура проверки разрешений на выполнение определенных операций

• GRANT выдать разрешения

• REVOKE запретить

• DENY отобрать разрешения, выданные ранее

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

Нормализация данных – процесс преобразования таблиц базы данных к нормальной форме. Шесть нормальных форм – 1NF, 2NF,...6NF.

Широкое практическое применение имеют формы 1NF, 2NF, 3NF.

**Первая нормальная форма**

• Таблица не должна содержать повторяющихся групп данных.

• Атомарность – каждый столбец должен содержать одно неделимое значение.

**Вторая нормальная форма**

• Таблица находится в первой нормальной форме

• Каждый неключевой атрибут полностью функционально зависит от каждого возможного ключа

• Простой и составной ключ

**Третья нормальная форма**

• Таблица находится во второй нормальной форме

• Отсутствуют транзитивные зависимости

**4. Типы данных Microsoft SQL Server**

Каждый столбец таблицы, создаваемой с помощью оператора CREATE TABLE, характеризуется типом данных, для хранения которых он предназначен. Microsoft SQL Server 2008 поддерживает следующие типы данных: числовые, символьные, для хранения даты и времени, денежные, двоичные и специальные.

**Числовые данные.** Числовые данные могут быть точными и приближенными. В табл. 5.1 сведены точные числовые типы. Все, кроме DECIMAL и NUMERIC, типы являются целочисленными. Тип NUMERIC используется для хранения данных с фиксированной точкой точности *p* (максимальное количество цифр в числе) и масштабом *s* (количество цифр после десятичной точки). Тип DECIMAL является синонимом NUMERIC.

**Точные числовые типы**

|  |  |  |
| --- | --- | --- |
| Тип данных | Диапазон значений | Количество байт |
| Tinyint | 0–255 | 1 |
| smallint | –32 768–32 768 | 2 |
| int | –231–(231 – 1) | 4 |
| Bigint | –263–(263 – 1) | 8 |
| Bit | 0 или 1 | 1 |
| decimal(*p*,*s*)  numeric(*p*,*s*)  1 ≤ *p* ≤ 38,  0 ≤ *s* < *p* | (–1038 + 1) – (1038 + 1) | 5–17 |

В табл. 5.2 сведены приближенные числовые типы. Для типа FLOAT можно указать точность *p*, но поддерживается только два значения точности: 24 (для всех 1 ≤ *p* ≤ 24) и 53 (*p* > 24). По сути, приближенный тип только один – FLOAT, тип REAL является синонимом типа FLOAT(24).

**Приближенные числовые типы**

|  |  |  |
| --- | --- | --- |
| Тип данных | Диапазон значений | Количество байт |
| float(*p*)  1 ≤ *p* ≤ 53 | От –1,79·10308 до –2,23·10–308;  0;  от 2,23·10–308 до 1,79·10308 | 4 или 8 |
| real  float(24) | От –3,4·1038 до –1,18·10–38;  0;  от 1,18·10–38 до 3,4·1038 | 4 |

**Символьные данные**.Для хранения символьных данных в таблицах БД применяется шесть типов, перечисленных в табл. 5.3.

**Символьные типы**

|  |  |  |
| --- | --- | --- |
| Тип данных | Размер в символах | Количество байт |
| char(n) | 1–8000 | *n* |
| varchar(n) | 1–4000 | Количество символов + 2 |
| varchar(max) | 1–(231 – 1) | Количество символов + 2 |
| nchar(n) | 1–8000 | 2*n* |
| nvarchar(n) | 1–4000 | 2·количество символов + 2 |
| nvarchar(max) | 1–(230 – 1) | 2·количество символов + 2 |

Все символьные типы данных можно разбить на две группы: обычные символьные данные (CHAR, VARCHAR), символьные данные в формате Юникода (NCHAR, NVARCHAR). Обычные символьные данные для хранения одного символа используют один байт. Символьные данные в формате Юникода для хранения одного символа использует два байта.

Типы данных VARCHAR(MAX) и NVARCHAR(MAX) применяются для хранения больших символьных данных объемом до 4 Гб.

**Типы для хранения даты и времени.** Для хранения даты и/или времени применяются типы данных, представленные в табл. 5.4. Для каждого типа указывается диапазон, точность и формат строки, который может быть автоматически преобразован к соответствующему типу.

**Типы для даты и времени**

|  |  |  |
| --- | --- | --- |
| Тип данных | Диапазон, точность, формат | Количество байт |
| date | 01.01.1753–31.12.9999;  1 день;  YYYYMMDD | 3 |
| time(*p*)  0 ≤ *p* ≤ 7 | 00:00:00.0000000–23:59:59.9999999;  100 нс;  hh:mm:ss.nnnnnnn | 3–5 |
| smalldatetime | 01.01.1900 00:00–06.06.2079 23:59,  1 мин;  YYYYMMDD hh:mm | 4 |
| datetime | 01.01.1753 00:00:00.000–31.12.9999 23:59:59.999;  0.003 с;  YYYYMMDD hh:mm:ss.nnn | 8 |
| datetime2(*p*)  0 ≤ *p* ≤ 7 | 01.01.0001 00:00:00.00000000–31.12.9999 23:59:59.9999999;  100 нс;  YYYYMMDD hh:mm:ss.nnnnnnnn | 6–8 |
| datetimeoffset(*p*)  0 ≤ *p* ≤ 7 | 01.01.0001.00:00:00:00000000+00:00–31.12.9999.23:59:59:9999999+23:59;  100 нс;  YYYYMMDDhh:mm:ss:nnnnnnnn ± hh:mm | 8–10 |

**Двоичные данные.** В некоторых случаях в БД необходимо хранить просто последовательность битов безотносительно их дальнейшего применения. Другими словами Microsoft SQL Server 2008 ничего не «знает» о формате таких данных и рассматривает их как последовательность (часто говорят – поток) битов. О том, как использовать (интерпретировать) эти данные, решает пользователь, извлекающий эти данные из соответствующего столбца таблицы. Типы для хранения двоичных данных (говорят также – бинарных данных) часто применяются для хранения изображений (например, фотографий, фильмов) и звука. В то же время в двоичном виде можно хранить любые данные, например, файлы форматов **PDF**, **DOC**, **XLS**.

**Типы для хранения двоичных данных**

|  |  |  |
| --- | --- | --- |
| Тип данных | Размер в байтах | Количество байт |
| binary(n) | 1–8000 | n |
| varbinary(n) | 1–8000 | Количество символов + 2 |
| varbinary(max) | 1–(231 – 1) | Количество символов + 2 |

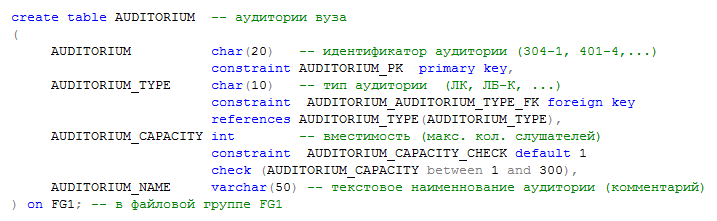
Тип BINARY применяется в том случае, если размер двоичных данных фиксирован, заранее известен и не превышает 8000 байт. В тех случаях, когда нет предположений относительно размера двоичных данных, применяют тип VARBINARY.

**Редко используемые типы данных**

|  |  |
| --- | --- |
| Тип данных | Назначение |
| smallmoney, money | Денежный тип, разновидность числового типа с фиксированной точкой, предназначен для хранения значения денежных величин |
| xml | XML-тип предназначен для хранения данных, представленных в формате XML |
| hierarhied | Иерархический тип данных, используется для отражения иерархических (древовидных) связей между строками таблицы |
| geography, geometry | Пространственные данные, предназначенные для хранения географических координат, геометрических координат и геометрических (2D) объектов |
| sqlvariant | Универсальный тип, способный хранить числовые, символьные данные, а также данные для хранения даты и времени. Непосредственно перед использованием, данные могут быть преобразованы к необходимому типу |
| text, ntext, image | Устаревшие типы данных, поддерживаются для совместимости с предыдущими версиями сервера, заменены на varchar(max) и varbinary(max) |

**5. Создание таблиц**

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



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

Ограничения, накладываемые на столбцы таблицы, предотвращающие появление в БД данных, не соответствующих предварительно заданным свойствам таблицы, называются ограничениями целостности.

|  |  |
| --- | --- |
| Условное обозначение  ограничения целостности | 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 (комбинации значений) и пустого значения (комбинации пустых значений) |
| foreign key  внешний ключ | Устанавливает связь между таблицей со столбцом, имеющим свойство foreign key (FK) и таблицей, имеющей столбец со свойством primary key (PK); предотвращает не согласованные 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 | Предотвращается появление в столбце значения, не удовлетворяющего логическому условию |

**7. Операторы DML: SELECT, INSERT, UPDATE, DELETE**

**8. Операторы DDL: CREATE, ALTER, DROP**

**9. Оператор SELECT. Секции FROM, WHERE, GROP BY, HAVING, ORDER BY, DISTINCT, TOP**

Секция DISTINCT позволяет не выводить повторяющиеся строки.



**10. 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).

Для работы с результирующими наборами, представляющими собой множество однородных значений (значений одного типа), можно использовать операции IN, ANY и ALL.

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 от IN.

Операция >ALL сформирует истинное значение в том случае, если значение, стоящее слева больше каждого значения в списке, указанном справа (значения в столбце, сформированном подзапросом).

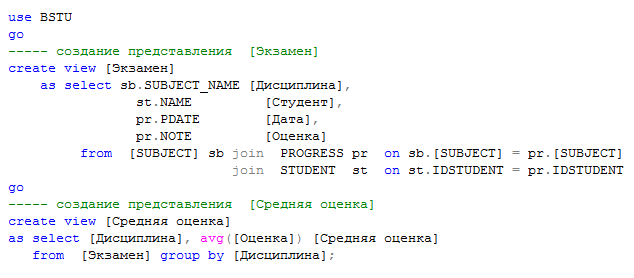
При выполнении ANY-сравнения стоящее слева значение сравнивается (операция >=) с каждым значением списка, записанным справа (содержимое столбца, сформированного подзапросом). Если хотя бы для одного значения из списка операция сравнения принимает значение «истина», то вся операция (>= ANY) принимает значение «истина».

Часто в подзапросе, который применяется в секции WHERE, бывает достаточно выяснить только сам факт наличия строк в результирующем наборе. В таких случаях удобно использовать операцию EXISTS. В скобках, следующих за ключевым словом EXISTS, записывается подзапрос. Результатом операции является «истина», если результирующий набор подзапроса содержит хотя бы одну строку, в противоположном случае операция EXISTS формирует значение «ложь».

**11. Представления. Создание представлений**

Представление (от англ. View) – 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.



В операторе CREATE после ключевого слова VIEW следует имя представления и далее после ключевого слова AS – текст SELECT-запроса, лежащего в основе представления. При этом к SELECT-запросу предъявляются следующие ограничения:

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

**12. Операции DML с представлениями. Представления с 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.

Удаление или изменение таблиц или представлений, используемых в SELECT-запросе, на котором основано представление, может повлечь за собой потерю работоспособности представления. Для предотвращения проблемы применяется специальная опция SCHEMABINDING, после указания которой при создании представления запрещаются операции с таблицами и представлениями, которые могут привести к нарушению его работоспособности.

Для того чтобы предотвратить несогласованность, при создании или изменении представления может быть указана специальная опция CHECK OPTION. В представлении с этой опцией помимо унаследованных от базовой таблицы ограничений целостности появляется еще одно ограничение, основанное на логическом выражении в секции WHERE базового SELECT-оператора.

**13. Агрегатные функции: COUNT, SUM, AVG, MIN, MAX**

Секция GROUP BY выполняет обработку строк в три этапа: 1) формирование строковых групп; 2) вычисление агрегатных функций над каждой группой строк; 3) формирование результирующего набора. В секции GROUP BY задается одно или несколько выражений (формирующих значения), задающих признак объединения строк в группы. На втором этапе для вычислений могут применяться агрегатные функции.

**Наиболее часто применяемые агрегатные функции**

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

**14. Группировка данных с использованием CUBE, ROLLUP**

ROLLUP ( )

Формирует статистические строки простого предложения GROUP BY и строки подытогов или строки со статистическими вычислениями высокого уровня, а также строки общего итога.

Количество возвращаемых группирований равно количеству выражений в <составном списке элементов> плюс один. Например, рассмотрим следующую инструкцию.

SELECT a, b, c, SUM ( <expression> )

FROM T

GROUP BY ROLLUP (a,b,c);

Для каждого уникального сочетания значений (a, b, c), (a, b) и (a) формируется одна строка с подытогом. Вычисляется также строка общего итога.

Столбцы свертываются справа налево. Последовательность расположения столбцов влияет на выходное группирование ROLLUP и может отразиться на количестве строк в результирующем наборе.

CUBE ( )

Формирует статистические строки простого предложения GROUP BY, строки со статистическими вычислениями высокого уровня конструкции ROLLUP и строки с результатами перекрестных вычислений.

Выходные данные CUBE являются группированием для всех перестановок выражений в <составном списке элементов>.

Количество формируемых группирований равно (2n), где n — количество выражений в <составном списке элементов>. Например, рассмотрим следующую инструкцию.

SELECT a, b, c, SUM (<expression>)

FROM T

GROUP BY CUBE (a,b,c);

Формируется одна строка для каждого уникального сочетания значений (a, b, c), (a, b), (a, c), (b, c), (a), (b) и (c) с подытогом для каждой строки и строкой общего итога.

Выходные данные CUBE не зависят от порядка столбцов.

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**.

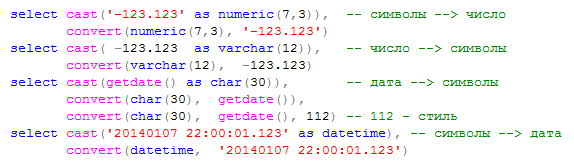
1. Результирующий набор содержит *n* + 1 групп строк, где *n* – количество выражений для группировки, указанных в скобках за ключевым словом ROLLUP.
2. Первая группа строк является результатом выполнения группировки по всем *n* выражениям.
3. Вторая группа строк является результатом группировки первой 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.

**15. Функция преобразования типов данных CAST**

Для преобразования типов предусмотрены две встроенные функции **CAST** и **CONVERT**. По своим возможностям функции практически не отличаются, но функция **CAST** соответствует стандарту SQL-92, поэтому ее применение предпочтительнее.



**16. Функция преобразования типов данных CONVERT**

Обратите внимание, что в предпоследнем SELECT-запросе в функции **CONVERT** есть третий параметр, который задается числовым значением. Этот параметр применяется при преобразованиях, использующих типы данных для хранения даты и времени, и задает стиль представления этих данных.

**17. 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.

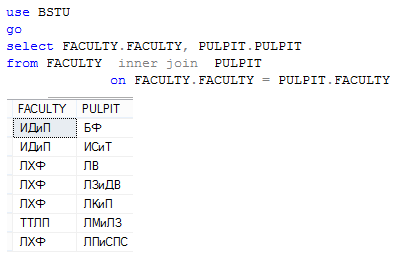
Результатом оператора INTERSECT является набор строк, представляющий собой пересечение (в теоретико-множественном смысле) двух исходных наборов строк. Как и для оператора UNION, к SELECT-запросам предъявляется требование совместимости типов соответствующих столбцов

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

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

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

Обратите внимание, что выражение, описывающее внутреннее соединение, состоит из двух частей: собственно выражение INNER JOIN, указываемое между двумя соединяемыми таблицами; логическое выражение, записываемое после ключевого слова ON.



Результирующий набор SELECT-запроса на рис. 7.27 создается по следующим правилам:

1. выполняется декартово произведение для таблиц **FACULTY** и **PULPIT**;
2. из результирующего набора, полученного на предыдущем шаге, выбираются строки, удовлетворяющие условию логического выражения, указанного после ON;
3. из всех столбцов результирующего набора «вырезаются» столбцы, указанные в списке SELECT.

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

**19. Ортогональное соединение cross join**

Каждая строка одной таблицы соединяется с каждой строкой другой таблицы. Такое соединение называют декартовым произведением таблиц.

Для выполнения декартова произведения двух или более таблиц следует просто перечислить их через запятую в секции FROM. Существует другой способ записи этой же операции с помощью выражения CROSS JOIN.

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

**21. Язык T-SQL. Пакеты. Объявление локальных переменных. Операторы присвоение**

T-SQL – это процедурное расширение языка SQL. Иначе говоря, это надстройка над SQL, дополняющая его возможностями алгоритмического языка программирования.

Пакет – это группа операторов T-SQL, которая обрабатывается сервером СУБД вместе. Пакет – это группа операторов T-SQL, которую программа клиент отправляет серверу на исполнение.Пакеты разграничиваются с помощью специальной инструкции GO. Инструкция GO интерпретируется как команда отправки на сервер порции операторов T-SQL.

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. В одном сценарии допускается несколько операторов DECLARE. Жизненный цикл всех объявленных переменных заканчивается вместе с завершением пакета, в котором находится соответствующий оператор DECLARE.

Переменной, объявленной в операторе DELCLARE, можно присвоить значение тремя способами: инициализировать в операторе DECLARE, присвоить значение с помощью операторов SET или SELECT. Единственное отличие операторов присваивания SET и SELECT заключается в возможности SELECT присваивать значения одновременно нескольким переменным.

**22. Глобальные переменные**

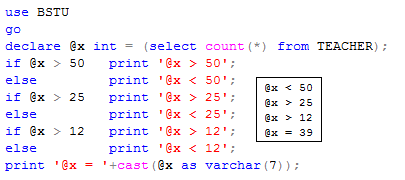
Глобальные переменные: @@ROWCOUNT (число обработанных строк), @@VERSION (версия SQL Server), @@SPID (возвращает системный идентификатор процесса, назначенный сервером текущему подключению), @@ERROR (код последней ошибки), @@SERVERNAME, @@TRANCOUNT (возвращает уровень вложенности транзакции), @@FETCH\_STATUS (проверка результата считывания строк результирующего набора), @@NESTLEVEL (уровень вложенности текущей процедуры).

**23. Операторы print, if-else, case**

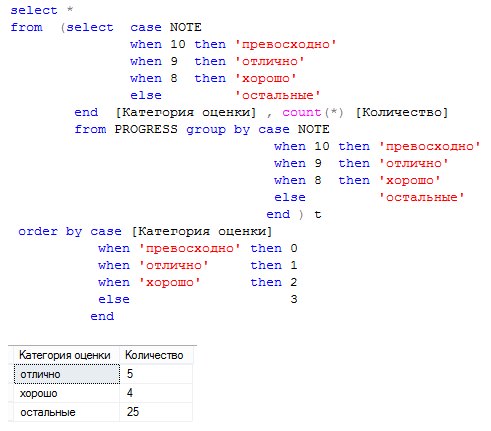
Вывод данных в T-SQL возможен двумя способами: с помощью оператора SELECT можно сформировать выходной результирующий набор, с помощью оператора PRINT можно вывести строку в стандартный выходной поток. Каждый оператор SELECT формирует отдельный результирующий набор, а каждый оператор PRINT – строку в единственный выходной поток.

Оператор ветвления IF позволяет в зависимости от значения логического выражения выполнять или не выполнять оператор или блок операторов сценария T-SQL.

За логическим выражением оператора IF располагается оператор, выполняющийся в случае истинности значения логического выражения. Если при этом требуется выполнить оператор при альтернативном значении логического выражения, следует применить конструкцию IF ELSE .



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.



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.

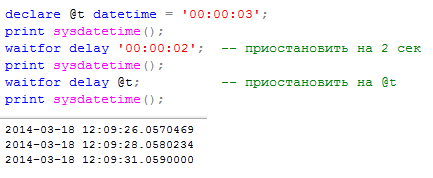
**24. Операторы begin-end, waitfor и return**

В некоторых случаях в сценарии T-SQL появляется необходимость указать, что группа следующих друг за другом операторов должна выполняться как единое целое. Это делается с помощью оператора BEGIN END.

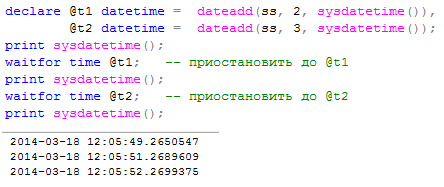
Немедленно завершить работу пакета можно с помощью оператора RETURN. На рис. 13.10 приведен сценарий, демонстрирующий работу оператора RETURN.

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 сценарии, представленном на рис. 13.22, демонстрируется выполнение оператора WAITFOR в режиме DELAY.

****

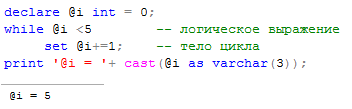
В примере на рис. 13.24 используется оператор WAIFOR в режиме TIME. В этом режиме можно приостановить выполнение потока до наступления заданного значения системного времени на сервере.

****

**25. Операторы цикла while**

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

Оператор WHILE содержит две составляющие: логическое выражение и тело цикла. Логическое выражение определяет условие выполнения тела цикла. Тело цикла содержит один или более операторов, которые выполняются в том случае и до тех пор, пока логическое выражение принимает значение «истина»..



В том случае, если тело цикла содержит более, чем один оператор, следует применять операторные скобки BEGIN END

**26. Обработка ошибок в конструкциях try-catch**

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

В блоке CATCH можно использовать несколько (табл. 13.1) системных функций, позволяющих диагностировать возникшую в охраняемом коде ошибку.

**Функции диагностики ошибок в сценарии T-SQL**

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

**27. Локальные и глобальные временные таблицы**

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

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

**Локальные** временные таблицы имеют имена, начинающиеся с символа #, доступны только создавшему ее пользователю и могут быть удалены с помощью оператора DROP TABLE. Если пользователь временную таблицу не удалил сам, то она удалится автоматически при его отключении.

**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.

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

При пояснении понятия курсора используют два непротиворечащих друг другу определения. Первое определение: курсор – это механизм обработки результирующего набора SELECT-запроса. Второе определение: курсор – это поименованная область памяти, содержащая результирующий набор SELECT-запроса.

**Общая схема работы с курсором**

Работа с любым курсором осуществляется в следующей последовательности.

1. Курсор объявляется в операторе DECLARE.
2. Курсор открывается с помощью оператора OPEN.
3. С помощью оператора FETCH считывается одна или несколько строк результирующего набора, связанного с курсором SELECT-оператора. Результат каждого считывания проверяется с помощью системной функции @@FETCH\_STATUS.
4. Курсор закрывается с помощью оператора CLOSE.
5. Если курсор глобальный (поясняется позже), то курсор должен быть освобожден с помощью оператора DEALLOCATE.

**29.Типы курсоров: global/local, static/dynamic**

Курсоры могут быть **глобальными и локальными**.

Локальный курсор может применяться в рамках одного пакета и ресурсы, выделенные ему при объявлении, освобождаются сразу после завершения работы пакета. Признаком того, что курсор является локальным, служит атрибут LOCAL, указанный при объявлении курсора. Доступен для использования только в том пакете, в котором был объявлен. Следующий пакет уже не «видит» курсора.

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

Курсоры могут быть **статическими и динамическими**.

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

Значение функции **@@CURSOR\_ROWS** после открытия динамического курсора всегда равно **–1**.

**30. 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.

ключевые слова: FIRST (первая строка), NEXT (следующая строка за текущей), PRIOR (предыдущая строка от текущей), ABSOLUTE 3 (третья строка от начала), ABSOLUTE −3 (третья строка от конца), RELATIVE 5 (пятая строка вперед от текущей), RELATIVE −5 (пятая строка назад от текущей).

**31. 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 |

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

Для курсоров с установленным свойством FOR UPDATE, помимо чтения данных из строк с помощью оператора FETCH, можно эти строки изменять или удалять с помощью специального формата операторов UPDATE и DELETE.

В секции WHERE операторы DELETE и UPDATE используют операцию CURRENT OF, для которой указывается имя курсора. Такой формат операторов позволяет удалять или изменять строки в таблице, соответствующие текущей позиции курсора в результирующем наборе.

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.

**33. 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.

**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.-

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

Хранимая процедура – это объект БД, представляющий собой поименованный код T-SQL. Как и любой объект БД, хранимая процедура может быть создана с помощью CREATE, изменена с помощью ALTER и удалена с помощью оператора DROP. К именованию процедуры предъявляются такие же требования, как и к именам объектов БД. Не рекомендуется использовать имена процедур, начинающиеся с символов **SP** – обычно так именуются системные хранимые процедуры. Хранимая процедура может принимать входные и формировать выходные параметры, а результатом ее выполнения может быть целочисленное значение, возвращаемое к точке вызова с помощью оператора RETURN один или более результирующих наборов, сформированных операторами SELECT, а также содержимое стандартного выходного потока, полученного при выполнении операторов PRINT. Вызов процедуры осуществляется с помощью оператора EXECUTE. Кроме того, результирующий набор хранимой процедуры может быть использован в качестве исходного источника строк для оператора INSERT.

При вызове хранимой процедуры применяется параметрическая форма передачи параметров (задаются имена параметров и их значения в произвольном порядке).

Для входных параметров можно задать значения по умолчанию. В этом случае при вызове процедуры можно не указывать значение этого параметра

Процедуры, формирующие результирующий набор и не имеющие выходных параметров, могут быть применены в операторе INSERT в качестве источника строк для добавления в целевую таблицу.

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

**35. Виды функций. Создание функций. Передача параметров**

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

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

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 (inline) функции и многооператорные табличные функции.

При вызове функции необходимо указывать ее имя с точностью до схемы БД.

**36. Индексы. Назначение и применение индексов**

Индекс – это объект БД, позволяющий ускорить поиск в определенной таблице. Для получения данных из таблицы наличие индексов не является обязательным. Если нет подходящего индекса, оптимизатор запланирует простой последовательный перебор всех строк таблицы (говорят «просканирует таблицу») для поиска необходимых данных. Иначе говоря, если индекс не позволяет ускорить процесс поиска данных – он является бесполезным.

Как и любой другой объект базы данных, индекс может быть создан с помощью DDL-оператора CREATE, модифицирован с помощью ALTER и удален с помощью оператора DROP. Для одной таблицы может быть построено несколько индексов.

Структура, используемая MSS для хранения индексов в БД, представляет собой сбалансированное дерево, узлами которого являются страницы – стандартные блоки данных

MSS поддерживает несколько типов индексов: кластеризованные и некластеризованные индексы, уникальные и неуникальные, XML-индексы, пространственные индексы, а также полнотекстовые индексы.

**37. Виды индексов. Применение различных видов индексов**

типов индексов: кластеризованные и некластеризованные индексы, уникальные и неуникальные, XML-индексы, пространственные индексы, а также полнотекстовые индексы.

**Кластеризованные индексы**

Обычно кластеризованные индексы формируются автоматически при создании таблицы. Дело в том, что наличие первичного ключа (ограничение PRIMARY KEY) в таблице БД влечет автоматическое создание индекса при создании таблицы. Если при этом в свойствах столбца первичного ключа специально не указано ключевое слово NONCLUSTERED, то автоматически создастся кластеризованный индекс.

Основной особенностью кластеризованного индекса является то, что при его построении таблица становится частью индекса. Другими словами, строки таблиц, имеющих кластеризованные индексы, физически упорядочены в соответствии со значениями индексируемых столбцов. В связи с этим в таблице может быть только один кластеризованный индекс.

**Некластеризованные индексы**

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. На рис. 11.12 представлен пример создания таблицы **TEACHER**, имеющей столбец со свойством PRIMARY KEY и соответствующим некластеризованным индексом.

Во всех приводимых до сих пор примерах использовались индексы, созданные по одному столбцу таблицы. Такие индексы называются простыми. MSS допускает создавать индексы по нескольким столбцам – такие индексы называются составными

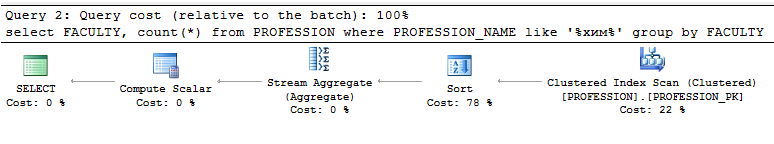
**Индексы покрытия**

Как уже отмечалось, избежать полного сканирования таблицы можно с помощью кластеризованного, некластеризованного индексов или их совместного применения.

Индекс покрытия запроса позволяет включить в состав индексной строки значения одного или нескольких неиндексируемых столбцов.

**38. План запроса**

План запроса представляет собой алгоритм выполнения SQL-запроса. На каждом шаге алгоритма выполняется элементарное действие сервера СУБД



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. На рис. 11.2 показано процентное распределение стоимости запроса (Cost) по шагам плана 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.

**39. Этапы обработки SELECT запроса**

**40. Понятие стоимости запроса**

При построении плана запроса для каждого шага вычисляется числовая величина, пропорциональная продолжительности выполнения шага, называемая стоимостью.

**41. Оптимизация стоимости 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.

**42. 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.

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.

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

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

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.

**44. Транзакции. Свойство ACID**

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

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

**45. Блокировки. Эскалация блокировок. Взаимные блокировки**

**46. Операторы TCL**

Операторы TCL предназначены для создания транзакций. Tранзакция – это несколько DML-операторов, которые либо все успешно выполнятся, либо все не выполняются. TCL SQL включает четыре оператора: BEGIN TRAN, SAVE TRAN, COMMIT TRAN и ROLLBACK TRAN (рис. 3.22).



Рис. 3.22. Операторы TCL SQL

Оператор BEGIN TRAN указывает на начало транзакции. Структура оператора представлена на рис. 3.23.



Рис. 3.23. Структура оператора BEGIN TRAN

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. Структура оператора COMMIT TRAN представлена на рис. 3.24.



Рис. 3.24. Структура оператора COMMIT

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.



Транзакция с именем T1 начинается оператором BEGIN TRAN, включает в себя пять операторов INSERT и два оператора UPDATE. В случае успешного выполнения всех DML-операторов, входящих в транзакцию, осуществляется фиксация (оператор COMMIT), иначе управление передается в блок BEGIN CATCH в котором выполняется откат.

**47. Уровни изоляции транзакций**

**Уровни изолированности транзакций**

Одним из основных свойств транзакции является изолированность**. 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.

Аргументы

READ UNCOMMITTED  
Указывает, что инструкции могут считывать строки, которые были изменены другими транзакциями, но еще не были зафиксированы.

Транзакции, работающие на уровне READ UNCOMMITTED, не используют совмещаемые блокировки, чтобы предотвратить изменение считываемых текущей транзакцией данных другими транзакциями. Это наименьшее ограничение уровней изоляции.

READ COMMITTED  
Указывает, что инструкции не могут считывать данные, которые были изменены другими транзакциями, но еще не были зафиксированы. Это предотвращает чтение«грязных» данных. Данные могут быть изменены другими транзакциями между отдельными инструкциями в текущей транзакции, результатом чего будет неповторяемое чтение или фантомные данные. Этот параметр в SQL Server установлен по умолчанию.

**REPEATABLE READ**  
Указывает на то, что инструкции не могут считывать данные, которые были изменены, но еще не зафиксированы другими транзакциями, а также на то, что другие транзакции не могут изменять данные, читаемые текущей транзакцией, до ее завершения.

Это запрещает другим транзакциям изменять строки, считываемые текущей транзакцией. Другие транзакции могут вставлять новые строки, соответствующие условиям поиска инструкций, содержащихся в текущей транзакции.

**SNAPSHOT**  
Указывает на то, что данные, считанные любой инструкцией транзакции, будут согласованы на уровне транзакции с версией данных, существовавших в ее начале. Транзакция распознает только те изменения, которые были зафиксированы до ее начала. Инструкции, выполняемые текущей транзакцией, не видят изменений данных, произведенных другими транзакциями после запуска текущей транзакции. Таким образом достигается эффект получения инструкциями в транзакции моментального снимка зафиксированных данных на момент запуска транзакции.

**SERIALIZABLE**Указывает следующее.

* Инструкции не могут считывать данные, которые были изменены другими транзакциями, но еще не были зафиксированы.
* Другие транзакции не могут изменять данные, считываемые текущей транзакцией, до ее завершения.
* Другие транзакции не могут вставлять новые строки со значениями ключа, которые входят в диапазон ключей, считываемых инструкциями текущей транзакции, до ее завершения.

**48. Функция TRANCOUNT**

**Она возвращает уровень вложенности транзакции. Значение, превышающее нуль, означает, что транзакция не завершена.**

**49. Язык XML**

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

**50. Секция for XML оператора SELECT**

**. Преобразование реляционных данных в XML-структуры**

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.

**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

**. 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

**51. Применение системных процедур sp\_xml\_preparedocument, sp\_xml\_removedocument**

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.

**системная процедура SP\_XML\_REMOVEDOCUMENT**, освобождающая использованные OPENXML ресурсы

**52. 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

**Режимы работы функции OPENXML**

|  |  |
| --- | --- |
| Режим | Описание |
| 0 | Применяется атрибутивная модель сопоставления. Каждый XML-атрибут преобразовывается в столбец таблицы |
| 1 | Аналогично режиму 0, но для необработанных столбцов применяется сопоставление на основе элементов XML-документа |
| 2 | Применяется сопоставление на основе элементов. Каждый элемент преобразовывается в столбец таблицы |

**1.Системные баз данных**

**2. Группы операторов SQL**

**3. Нормализация таблиц БД**

**4.Типы данных SQL**

**5.Создание таблиц**

**6. Ограничение целостности**

**7.Операторы DML**

**8.Операторы DDL**

**9.Оператор SELECT**

**10. Подзапросы**

**11.Представления**

**12.Операции DML с представлениями**

**13.Агрегатные функции**

**14.Группировка**

**15.Функция преобразования типов данных cast**

**16.Функция преобразования типов данных convert**

**17. Оператор union, insert, except**

**18.Соединение таблиц**

**19. Ортогональное соединение**

**20.Внешнее соединение**

**21.Язык T-SQL**

**22.Глобальные переменные**

**23.Операторы print, if else, case**

**24.Операторы begin end, waitfor, return**

**25.Операторы цикла while**

**26.Обработка ошибок**

**27.Локальные глобальные временные таблицы**

**28.Курсоры**

**29.Типы курсоров**

**30.Опция scroll**

**31.Функция fetch status**

**32.Применение секции where current of**

**33.Хранимые процедуры и функции**

**34.Создание хранимых процедур**

**35.Виды функций, создание функций передача параметров**

**36.Индексы, назначение и применение**

**37.Виды индексов**

**38.План запроса**

**39.Этапы обработки SELECT-запросов**

**40.Понятие стоимости запроса**

**41.Оптимизация стоимости запроса**

**42.Триггеры**

**43.Созлание и назначение instead of триггеров**

**44.Транзакции**

**45.Блокировки**

**46.Операторы TCL**

**47.Уровни изоляции транзакции**

**48.Функция trancount**

**49.Язык XML**

**50.Функция for XML оператора SELECT**

**51. Применение системных процедур**

**52. Применение openXML**