Order by: desc – порядок убывания; Asc – порядок возрастания

Оператор **EXISTS** позволяет проверить, возвращает ли подзапрос какое-либо значение.

Представление **выполняет функцию фильтра базовых таблиц, на которые оно ссылается**.

**Лаба 6**

1. FROM
2. ON
3. JOIN
4. WHERE
5. GROUP BY
6. WITH CUBE или WITH ROLLUP
7. HAVING
8. SELECT
9. DISTINCT
10. ORDER BY
11. TOP

**Агрегатные функции** (англ. aggregate function) — это математические функции, применяемые к набору входных данных и возвращающие по ним одно результирующее значение.

* Аналитические функции
* Функции поразрядной операции
* Ранжирующие функции
* Функции наборов строк
* Скалярные функции

**Лаба 8**

**Transact-SQL** — процедурное расширение языка SQL, предназначена для работы сервера от микрософт

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

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

**Хранимая процедура** – поименованный блок (BEGIN…END) операторов, хранящийся в базе данных в откомпилированном виде

**Операторы присваивания**:

* инициализировать в DECLARE
* присвоить значение SET
* присвоить значение SELECT

**DECLARE** - объявление используемых переменных

С помощью оператора **SET** можно переменной присвоить значение и выполнять вычисления.

Оператор **SELECT** позволяет нескольким переменным присвоить значения.

Здесь @h - переменная типа TABLE. Этот тип позволяет создавать таблицы в памяти и использовать их для хранения промежуточных данных.

Вывод данных в T-SQL возможен двумя способами:

* ‌оператором SELECT можно сформировать выходной результирующий набор
* ‌и с помощью оператора PRINT можно вывести строку в стандартный выходной поток.

Если одновременно выводятся данные, сформированные оператором SELECT и данные, сформированные оператором PRINT, то посмотреть последние можно на вкладке Messages.

Функция CAST используется для преобразования типов.

Функция CONVERT также используется для преобразования типов. Третий параметр этой функции применяется при преобразованиях, использующих типы данных для хранения даты и времени, и задает стиль представления этих данных.

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

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.

Сформировать временную локальную таблицу можно с помощью оператора CREATе

**Глобальные временные** таблицы имеют имена, начинающиеся с символов ## и доступны всем пользователям, подключенным к серверу

**Оператор RETURN** служит для немедленного завершения работы пакета

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

**Лаба 9**

Для каждой индексируемой строки имеется элемент индекса (index entry), который сохраняется на странице индексов. Каждый элемент индекса состоит из ключа индекса и указателя.

Индексы создаются для столбцов таблиц и представлений. Индексы предоставляют путь для быстрого поиска данных на основе значений в этих столбцах

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

Как и любой другой объект базы данных, индекс может быть создан с помощью оператора CREATE, модифицирован с помощью ALTER и удален с помощью оператора DROP.

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

План запроса можно посмотреть, выполнив в контекстном меню запроса **команду Display Estimated Execution Plan** (Показать предполагаемый план выполнения) или нажав соответствующую кнопку на панели инструментов.

Для одной таблицы возможно построение нескольких индексов.

**Фильтрующий индекс** создается только для строк таблицы, которые удовлетворяют логическому условию

**Индексы бывают**: кластеризованные, некластеризованные, уникальные, неуникальные, Простые и составные, XML-индексы, Пространственные индексы

***Кластеризованный индекс*** определяет физический порядок данных в таблице.

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

В таблице может быть только один кластеризованный индекс.

**Некластеризованные (кучи) индексы** не влияют на физический порядок строк в таблице.

MSS допускает создавать индексы по нескольким столбцам – такие **индексы называются составными**.

**Уникальным (Unique) называют индекс**, обеспечивающий уникальное значение всех строк по определенному ключу и гарантирующий, что в ключе индекса не будет значений одинаковых, повторяющихся.

**Индексы XML** представляют собой специальный тип индекса, который может быть создан на столбцах типа XML.

**Пространственный индекс** представляет собой тип расширенного индекса, позволяющий индексировать пространственные столбцы. Пространственный столбец представляет собой столбец таблицы, в котором содержатся данные пространственного типа, например, geometry или geography.

**INCLUDE п**озволяет указать столбцы, которые добавляются к страницам листьев некластеризованного индекса

**Покрывающий индекс** (covering index). Некластеризованный индекс, когда все столбцы запроса включены в индекс

**Фрагментацией** называют такое состояние, когда для некоторых страниц индекса логический порядок, основанный на значении ключа, не совпадает с физическим порядком страниц индексов.

**Внутренняя фрагментация** - объем данных, хранящихся в каждой странице

**Внешняя фрагментация** - нарушение логического порядка страниц

Параметр FILLFACTOR указывает процент заполнения индексных страниц нижнего уровня.

* При высоком FillFactor больше строк помещается в одну страницу данных, но может стать больше разбиений страниц
* При низком FillFactor на странице данных меньше записей, что снизит число разбиений страниц, но потребуется больше операций чтения, так как данные будут распределены по большему числу страниц

Можно создать составной не уникальный, некластеризованный индекс #EX\_NONCLU по двум столбцам TKEY и CC таблицы #EX с помощью оператора: CREATE index #EX\_NONCLU on #EX(TKEY, CC)

Этот индекс не применяется оптимизатором ни при фильтрации, ни при сортировке строк таблицы #EX

Но, если хотя бы одно из индексируемых значений зафиксировать (задать одно значение), то оптимизатор применит индекс.

‌Некластеризованный 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 IN-CLUDE): 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.

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

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

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

Фрагментация индексов снижает эффект от их применения.

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

**Реорганизация (REORGANIZE)** выполняется быстро, но после нее фрагментация будет убрана только на самом нижнем уровне.

Пусть выполнена реорганизация с помощью оператора ALTER для индекса #EX\_TKEY.

ALTER index #EX\_TKEY on #EX reorganize

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

**Операция перестройки (REBUILD**) затрагивает все узлы дерева, поэтому после ее выполнения степень фрагментации равна нулю.

Пусть выполнена перестройка с помощью оператора ALTER для индекса #EX\_TKEY в режиме OFFLINE.

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

Выполнением запроса о фрагментации можно оценить ее уровень.

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

Опция **PAD\_INDEX** указывает процент, до которого заполняется не листовые индексные страницы

Параметр **FILLFACTOR** указывает процент заполнения индексных страниц нижнего уровня.

REBUILD - пересоздание индекса

REORGANIZE - реорганизация страниц листьев индекса

DISABLE - отключение индекса

**Лаба 10**

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

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

1. Курсор объявляется в операторе DECLARE.
2. Курсор открывается с помощью оператора OPEN.
3. С помощью оператора FETCH считывается одна или несколько строк результирующего набора, связанного с курсором SELECT-оператора, и обрабатывается нужным образом. Результат каждого считывания проверяется с помощью системной функции @@FETCH\_STATUS.
4. Курсор закрывается оператором CLOSE.

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

Курсоры:

* Глобальные курсоры

***Глобальный* курсор** может быть объявлен, открыт и использован в разных пакетах

* Локальные курсоры

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

Типы курсоров:

* Динамические

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

* Статические

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

Асинхронное заполнение статических курсоров оптимизирует производительность

Статические курсоры используют рабочие таблицы базы данных **tempdb** для хранения строк, составляющих курсор

Если в соответствии с прогнозом оптимизатора запросов SQL Server ожидаемое число строк, возвращаемых в курсоре, превысит значение параметра **sp\_configure cursor threshold**, сервер запускает отдельный поток для заполнения рабочей таблицы

**Функция @@CURSOR\_ROWS** сообщает число строк в курсоре.

@@CURSOR\_ROWS для курсора, рабочая таблица которого продолжает заполняться, возвращается отрицательное число. Абсолютное значение возвращенного числа дает число строк в рабочей таблице, заполненных на данный момент времени.

Например, если функция @@CURSOR\_ROWS выбрана, пока идет заполнение набора ключей или курсора, управляемого набором ключей, но в наборе ключей уже имеется 143 ключа, функция возвращает значение -143.

@@CURSOR\_ROWS

* + -n – количество записей при асинхронной выборке,
  + n – количество записей при синхронной выборке,
  + 0 – курсор не открыт

**SCROLL** – Указывает, что доступны все параметры выборки (FIRST, LAST, PRIOR, NEXT, RELATIVE, ABSOLUTE). Если в инструкции DECLARE CURSOR стандарта ISO не указан параметр SCROLL, то поддерживается только параметр выборки NEXT.

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

@@FETCH\_STATUS

* + 0 – успешная выборка,
  + -1 – вышли за диапазон таблицы,
  + -2 – запись удалена после открытия курсора

**CURSOR\_STATUS** – cкалярная функция, позволяющая при вызове хранимой процедуры определить, вернула ли она курсор и результирующий набор для данного параметра

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

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

Курсоры с установленным свойством FOR UPDATE помимо чтения данных из строк с помощью оператора FETCH, могут эти строки изменять или удалять с помощью операторов UPDATE и DELETE, если в секции WHERE эти операторы используют операцию CURRENT OF, для которой указывается имя курсора (в примере ниже Primer2). Оператор WHERE CURRENT OF позволяет обновить или удалить запись, которая была последней выбранной курсором.

**Лаба 11**

**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.

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

**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).

Модели конкурентного доступа:

* пессимистический одновременный конкурентный доступ

(Предполагается, что между процессами в любое время может возникнуть конфликт и ресурс блокируется)

* оптимистический одновременный конкурентный доступ

(Предполагается, что одновременное изменение данных маловероятно)

**Виды транзакций:**

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

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

**SET IMPLICIT\_TRANSACTIONS ON**

Обратное переключение осуществляется с использованием ключевого слова OFF вместо ON.

Неявная транзакция начинается, если выполняется один из следующих операторов:

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

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

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

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

**Системная функция @@TRANCOUNT** возвращает уровень вложенности транзакции. (если значение больше нуля, то транзакция не завершена).

**Встроенная функция PATINDEX** определяет в строке позицию первого символа подстроки, заданную шаблоном. С помощью этой функции в тексте сообщения об ошибке отыскивается имя ограничения целостности.

Изменения в таблице будут осуществлены только в случае отсутствия ошибок в операторах удаления и вставки.

Здесь **системная функция @@SPID** возвращает системный идентификатор процесса, назначенный сервером текущему подключению.

При параллельных транзакциях могут возникать три проблемы.

***Неподтвержденное чтение.*** До момента t1 транзакцией B выполняются два оператора: INSERT и UPDATE. Эти операторы изменяют таблицы БД, но до момента времени t2 не фиксируют и не откатывают эти изменения. После момента t1 транзакция A считывает содержимое таблиц, измененных транзакцией B и «видит» измененные или добавленные строки. При этом изменения остаются до момента t2 в неподтвержденном состоянии, т. е. могут быть как зафиксированными, так и отмененными.

***Неповторяющееся чтение.*** Одна транзакция читает данные несколько раз, а другая изменяет те же данные между двумя операциями чтения в первом процессе. По этой причине данные, прочитанные в различных операциях, будут разными.

***Фантомное чтение****.* Две последовательные операции чтения могут получать различные значения, т. к. дополнительные строки, называемые фантом­ными, могут добавляться другими транзакциями.

Чтобы такие проблемы не возникали, определяются различные уровни изолированности: READ COMMITED, REPEATABLE READ, SERIALIZABLE и др.

Проблемы:

* Потеря обновлений

Несколько транзакций одновременно могут считывать и обновлять одни и те же данные

Теряются все обновления данных, за исключением обновлений, выполненных последней транзакцией

* **Грязное чтение** – Происходит чтение несуществующих данных или потеря модифицированных данных

**Или**

чтение, при котором могут быть считаны добавленные или изменённые данные из другой транзакции, которая впоследствии не подтвердится (откатится).

* **Неповторяемое чтение** – Один процесс считывает данные несколько раз, а другой процесс изменяет эти данные между двумя операциями чтения первого процесса

Значения двух чтений будут разными

Или

повторно читает те же данные, что и раньше, и обнаруживает, что они были изменены другой транзакцией (которая завершилась после первого чтения).

* **Фантомное чтение** – Последовательные операции чтения могут возвратить разные значения

Считывание разного числа строк при каждом чтении

Возникают дополнительные фантомные строки, которые вставляются другими транзакциями

**Уровни изоляции:**

Пессимистическая модель:

* + READ COMMITTED – решает проблему грязного чтения
  + READ UNCOMMITTED – ничего не решает
  + REPEATABLE READ – решает проблемы грязного чтения и неповторяющегося
  + SERIALIZABLE – решает все проблемы

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

При работе с вложенными транзакциями нужно учитывать следующее:

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

− оператор ROLLBACK внешней транзакции отменяет зафиксированные операции внутренней транзакции;

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

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

Инструкции:

* BEGIN TRANSACTION
* BEGIN DISTRIBUTED TRANSACTION
* COMMIT [WORK]
* ROLLBACK [WORK]
* SAVE TRANSACTION
* SET IMPLICIT\_TRANSACTIONS

**Журнал транзакций** применяется для отката или восстановления транзакции

Для каждой базы данных собственный журнал транзакций

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

Свойства:

* + Длительность блокировки
  + Режим блокировки
  + Гранулярность блокировки

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

**Режим блокировки:**

* Разделяемая (shared lock)

Разделяемая блокировка резервирует ресурс только для чтения

Другие процессы не могут изменять заблокированный ресурс

Может быть несколько разделяемых блокировок

* Монопольная (exclusive lock)

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

* Обновления (update lock)

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

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

**Гранулярность блокировки** определяет, какой объект блокируется:

* + строки
  + страницы
  + индексный ключ или диапазон индексных ключей
  + таблицы
  + экстент
  + база данных

СУБД выбирает гранулярность блокировки автоматически

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)

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

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

Оптимистическая модель:

* + SNAPSHOT

Уровень изоляции на уровне транзакций

Любая другая транзакция будет читать подтвержденные значения в том виде, в каком они существовали непосредственно перед началом выполнения этой транзакции

Обе модели:

**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

Любая другая транзакция будет читать значения зафиксированные на момент начала этой транзакции

* ALTER DATABASE
* SET ISOLATION LEVEL
* READ COMMITTED SNAPSHOT

**Лаба 12**

**Хранимая процедура** – это поименованный код на языке Transact-SQL.

Хранимая процедура может быть создана с помощью CREATE, изменена с помощью ALTER и удалена с помощью оператора DROP.

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

Процедура может принимать входные и формировать выходные параметры.

Входные параметры объявляются после создания процедуры

Выходные параметры объявляются с помощью слова OUTPUT

Вызов процедуры осуществляется оператором EXECUTE (EXEC).

**RETURN –** может возвращать атомарные значения (строка, число), одну или несколько результирующих таблиц.

**Нельзя применять** – функции, тригеры, представления, процедуры.

@@TRANCOUNT– Возвращает количество инструкций BEGIN TRANSACTION, которые произошли в текущем соединении.

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.

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

Здесь для формирования сообщения об ошибке применяется встроенная функция RAISERROR, которая содержит три параметра: текстовое сообщение об ошибке, уровень серьезности ошибки и метку. Если уровень серьезности равен 11, то управление передается в блок обработки ошибок.

Пакет — это последовательность инструкций, которые отправляются системе базы данных для совместного их выполнения

Преимущество – одновременное исполнение всех инструкций позволяет получить улучшение производительности

Ограничение на единственную инструкцию:

* CREATE VIEW
* CREATE PROCEDURE
* CREATE TRIGGER

Хранимая процедура – объект базы данных

Хранимая процедура – поименованный блок (BEGIN…END) операторов, хранящийся в базе данных в откомпилированном виде

Принимает входные параметры

Принимаете и формирует выходные параметры

Целочисленное значение, возвращаемое к точке вызова с помощью оператора RETURN

Один или более результирующих наборов, сформированных операторами SELECT

Содержимое стандартного выходного потока, полученного при выполнении операторов PRINT

Не допускается применение :

* CREATE or ALTER FUNCTION
* CREATE or ALTER TRIGGER
* CREATE or ALTER PROCEDURE
* CREATE or ALTER VIEW
* USE databasename

**Лаба 13**

**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,
* 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

При вызове функции надо указывать ее имя с точностью до схемы БД. Если при создании функции имя схемы не указано, то она размещается по умолчанию в схеме **DBO**.

Если передается в функцию параметр, равный NULL, то выводится сообщение об общем количестве заказов

Функции:

1. Встроенные :
   1. Математические
   2. Строковые
   3. Работа с датами
2. Пользовательские.
3. Скалярные:
   1. возвращает одно значение
   2. нельзя timestamp, cursor, text, ntext, image, table

**Лаба 14**

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

**DML-**триггеры бывают двух типов: AFTER-триггеры и INSTEAD OF-триггеры.

**Триггеры типа AFTER** исполняются *после* выполнения оператора, вызвавшего соответствующее событие. При этом создаются автоматически две псевдотаблицы INSERTED и DELETED.

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

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

Важной особенностью AFTER-триггера является то, что он вызывается после выполнения активизирующего его оператора. Поэтому, если оператор нарушает ограничение целостности, то возникшая ошибка не допускает выполнения этого оператора и соответствующих триггеров.

Триггеры AFTER можно создавать только для базовых таблиц

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

**DML-**триггеры:

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

INSTEAD OF можно создавать для таблиц и для представлений - выполняется вместо выполнения любых действий с любой таблицей

Ограничения на столбцы:

* вычисляемые
* с типом данных TIMESTAMP
* со свойством IDENTITY

Событие INSERT при выполнении AFTER-триггера приводит к тому, что в псевдотаблицу INSERTED помещаются строки, добавленные оператором INSERT, вызвавшим это событие. Псевдотаблица DELETED остается пустой.

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

AFTER-триггеры :

* триггеры уровня оператора
* Выполняются по одному разу для каждого оператора
* Выполняются после наступления события

Можно указать порядок выполнения для нескольких триггеров с помощью sp\_settriggerorder (), который имеет параметр @order:

* + first — указывает, что триггер является первым триггером AFTER, выполняющимся для действия
  + last — указывает, что триггер является последним триггером AFTER, выполняющимся для действия
  + none — отсутствует порядок выполнения (чтобы выполнить сброс)

Остальные триггеры выполняются в неопределенном порядке

DDL-триггеры:

* триггеры уровня сервера (ALLSERVER)
* триггеры уровня базы данных (DATABASE)

Триггеры **уровня сервера** обрабатывают события сервера СУБД:

* + Создание объектов сервера
  + Изменение объектов сервера
  + Удаление объектов сервера
  + Подключение к серверу

Триггеры **уровня базы данных** – обработка событий, происходящих в рамках базы данных

**Лаба 15**

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

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

**В режиме RAW в** результате SELECT-запроса создается XML-фрагмент, состоящий из последовательности элементов с именем **row**. Каждый элемент **row** соответствует строке результирующего набора, имена его атрибутов совпадают с именами столбцов результирующего набора, а значения атрибутов равны их значениям. Чтобы раскрыть полностью XML-фрагмент в результирующем наборе надо по фрагменту дважды щелкнуть.

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

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

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

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

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

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

С **помощью выражения WITH** должна быть указана структура формируемого результата.

XML-документ :

* Начинается специальным тегом с именем **?xml** (объявление XML
* Каждый элемент начинается открывающим тегом и завершается закрывающим
* Могут быть вложенные теги

Пространство имён:

* 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)

Xml:

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

Языки схем:

1. язык DTD - Document Type Definition:
   * Набор правил для структурирования XML
   * Внутренний DTD – часть XML-документа
   * Внешний DTD – адрес URL
2. язык XML Schema

язык DTD:

* 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

Задачи:

* преобразование XML в строки реляционных таблиц
* преобразование данных в таблицах в XML

Нельзя использовать UNIQUE, PRIMARY KEY или FOREIGN KEY

Директивы:

* TYPE – сохранять результат реляционного запроса как XML-документ или фрагмент типа данных XML
* ROOT – Добавление к результирующему набору XML одного элемента верхнего уровня
* ELEMENTS

Xml-shema:

* встроенный механизм, позволяющий проверять на корректность XML-документы
* XML SCHEMA COLLECTION
* может содержать один или более XML-SCHEMA-документов
* для XML-столбца можно указать имя коллекции схем

Первичный XML-индекс:

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

Три типа вторичных типа XML-индексов:

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

Не могут быть составными

Не могут быть кластеризованными

язык запросов Xpath– XML Path Language — язык запросов к элементам XML-документа

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

Запрос данных:

* query() – выполнение запроса
* exist() – проверка существования
* value() – возвращает значение атрибута
* nodes() – возвращает набор узлов
* modify() – изменение документа
  + insert
  + delete
  + replace value of

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

На втором уровне (внутри тега **студент**) расположено три тэга: **паспорт**, **телефон** и **адрес** (вложенные теги **element**). Причем тэг **паспорт** должен быть ровно один (атрибуты **maxOccurs** и **minOccurs**); тэг **телефон** является обязательным и может быть в количестве не более трех (атрибут **maxOccurs**); тэг **адрес** является обязательным, и количество таких тэгов не должно быть более одного.

Элементы третьего уровня (**страна**, **город**, **улица**, **дом**, **квартира**) присутствуют только внутри элемента **адрес**. Все эти элементы являются обязательными и должны присутствовать ровно один раз.

Данные в документе размещаются как значения атрибутов (теги **attribute**) или как значения, размещенные в теле элементов (**телефон**, **страна**, **город**, **улица**, **дом**, **квартира**).

Тип данных, размещаемых в атрибутах или теле элементов данных, определяется значением атрибута **type**.