ЛР № 9. Создание и применение индексов

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

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

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

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

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

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.

Общая стоимость запроса (**Estimated Subtree Cost**) появляется во всплывающем окне, если подвести курсор к компоненту **Table Scan** (она равна 0,011). Чтобы объективно оценить время выполнения следующего запроса, надо очистить буферный кэш:

**checkpoint;** --фиксация БД

**DBCC DROPCLEANBUFFERS;** --очистить буферный кэш

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

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

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

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

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

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

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

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.

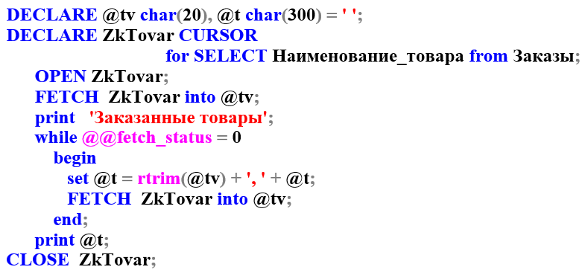
Операция ***перестройки*** (REBUILD) затрагивает все узлы дерева, поэтому после ее выполнения степень фрагментации равна нулю.

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

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

Лабораторная работа № 10. Обработка результатов запросов с помощью курсоров

**Курсор** является программной конструкцией, которая дает возможность пользователю обрабатывать строки результирующего набора запись за записью. Курсоры бывают *локальные* и *глобальные* (по умолчанию), *статические* и *динамические* (по умолчанию).

Курсор объявляется в операторе DECLARE

Курсор открывается с помощью оператора OPEN.

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

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

Курсор закрывается с помощью оператора CLOSE.

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

***Глобальный* курсор** может быть объявлен, открыт и использован в разных пакетах. Выделенные ему при объявлении ресурсы освобождаются только после выполнения оператора DEALLOCATE или при завершении сеанса пользователя.

Открытие ***статического* курсора** приводит к выгрузке результирующего набора во временную таблицу системной БД **TEMPDB**, и все дальнейшие операции осуществляются с этой таблицей.

После открытия курсора все текущие изменения в исходных таблицах не будут отражаться в результирующем наборе

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

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). Такой формат операторов позволяет удалять или изменять строки в таблице (в примере таблица **Заказы**), соответствующих *текущей* позиции курсора в результирующем наборе.

Лабораторная работа № 11. Особенности использования транзакций

**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).

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.

После этого осуществляется возврат в режим автофиксации (инструкция SET OFF).

Переключение в режим *явной транзакции* осуществляется с помощью оператора BEGIN TRANSACTION. Транзакцию должен завершать один из операторов: COMMIT TRAN или ROLLBACK TRAN. После завершения явной транзакции происходит возврат в исходный режим (автофиксации или неявной транзакции).

Транзакция начинается внутри TRY-блока и в случае успешного выполнения завершается оператором COMMIT.

Если при выполнении возникла ошибка, то в CATCH-блоке формируется и выводится соответствующее сообщение, а затем выполняется откат (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.

Если транзакция состоит из нескольких независимых блоков операторов T-SQL, изменяющих базу данных, то может быть использован оператор SAVE TRANSACTION, формирующий *контрольную точку* транзакции.

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.

Для моделирования процесса в этом и следующих заданиях следует выполнять сценарии в три этапа: последовательно оба сценария до отметки t1; последовательно оба сценария c отметки t1 до отметки t2; последовательно оба сценария c отметки t2 и до конца.

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

Лабораторная работа № 12. Разработка хранимых процедур

**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. 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, DML и TCL-операторов, конструкций TRY/CATCH, курсоров, временных таблиц.

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

Пусть с помощью ALTER внесены изменения в процедуру:

Если в операторе EXEC допустить ошибку, например, ввести повторно наименование уже имеющегося товара или неправильно использовать имена переменных, то будет выдано сообщение об ошибке.

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. Вызов процедуры:

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, а также использование транзакций.

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

* **Встроенные** 
  + **Математические**
  + **Строковые**
  + **Работа с датами**

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**.

* **Скалярные**
  + **возвращает одно значение**
  + **нельзя timestamp, cursor, text, ntext, image, table**
* **Inline**
  + **одна команда SELECT**
  + **возврат table**

**multi-statement- возвращает обычную таблицу заданной структуры в виде значения типа данных table,при этом количество команд в теле функции не ограничивается**

**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.

**Две специальные виртуальные таблицы**

* **deleted — содержит копии строк, удаленных из таблицы**
* **inserted — содержит копии строк, вставленных в таблицу**
* **Структура этих таблиц эквивалентна структуре таблицы, для которой определен триггер**
* **Таблица deleted – в инструкции CREATE TRIGGER указывается DELETE или UPDATE**
* **Таблица inserted – в инструкции CREATE TRIGGER указывается INSERT или UPDATE**
* **Триггеры AFTER можно создавать только для базовых таблиц**
* **Можно использовать для:**
  + **создания журнала аудита действий в таблицах базы данных**
  + **реализации бизнес-логики**
  + **принудительного обеспечения ссылочной целостности**
* **AFTER-триггеры - триггеры уровня оператора**
* **Выполняются по одному разу для каждого оператора**
* **Выполняются после наступления события**
* **Триггеры уровня оператора**
* **Выполняются по одному разу для каждого оператора**
* **Выполняются вместо операции - сама операция не выполняется**
* **Всегда использует таблицы inserted и deleted**
* **Выполняется после создания таблиц inserted и deleted**
* **Выполняется перед выполнением проверки ограничений целостности или каких-либо других действий**
* **INSTEAD OF можно создавать для таблиц и для представлений - выполняется вместо выполнения любых действий с любой таблицей**

Лабораторная работа № 15. Использование 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.

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.

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.

**create xml schema collection Student as**

**N'<?xml version="1.0" encoding="utf-16" ?>**

**<xs:schema attributeFormDefault="unqualified"**

**elementFormDefault="qualified"**

**xmlns:xs="http://www.w3.org/2001/XMLSchema">**

**<xs:element name="студент">**

**<xs:complexType><xs:sequence>**

**<xs:element name="паспорт" maxOccurs="1" minOccurs="1">**

**<xs:complexType>**

**<xs:attribute name="серия" type="xs:string" use="required" />**

**<xs:attribute name="номер" type="xs:unsignedInt" use="required"/>**

**<xs:attribute name="дата" use="required" >**

**<xs:simpleType> <xs:restriction base ="xs:string">**

**<xs:pattern value="[0-9]{2}.[0-9]{2}.[0-9]{4}"/>**

**</xs:restriction> </xs:simpleType>**

**</xs:attribute> </xs:complexType>**

**</xs:element>**

**<xs:element maxOccurs="3" name="телефон" type="xs:unsignedInt"/>**

**<xs:element name="адрес"> <xs:complexType><xs:sequence>**

**<xs:element name="страна" type="xs:string" />**

**<xs:element name="город" type="xs:string" />**

**<xs:element name="улица" type="xs:string" />**

**<xs:element name="дом" type="xs:string" />**

**<xs:element name="квартира" type="xs:string" />**

**</xs:sequence></xs:complexType> </xs:element>**

**</xs:sequence></xs:complexType>**

**</xs:element>**

**</xs:schema>';**

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**.