УО «Полоцкий государственный университет»

Факультет информационных технологий

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

к выполнению лабораторной работы №7

по дисциплине «Базы данных»
для специальности
1-40 01 01 Программное обеспечение информационных технологий

Новополоцк 2011

Методические указания разработала:

Бураченок Ирина Брониславовна – ст. преподаватель кафедры технологий программирования

ЛАБОРАТОРНАЯ РАБОТА №7

ТЕМА: знакомство с основными особенностями программирования в MS SQL Server средствами встроенного языка Transact SQL.

ЦЕЛЬ: знакомство с особенностями построения запросов средствами встроенного языка Transact SQL при использовании переменными типа Table и Cursor. Знакомство с особенностями построения процедур и триггеров.

Результат обучения:

После успешного завершения занятия пользователь должен:

- знать правила написания программ на Transact-SQL,
- знать правила объявления переменных и их типов,
- знать правила построения идентификаторов,
- уметь строить запросы при использовании переменных типа Table и Cursor,
- уметь строить триггеры: с условиями и ветвлениями, при использовании циклов.

Используемая программа: Microsoft SQL Server 2008.

План занятия:

1. Краткие теоретические сведения.
2. Выполнение задания.
3. Оформление отчета.
4. Демонстрация скрипта.

1. КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Триггеры в Transact-SQL

Определение триггера в стандарте языка SQL

Триггеры являются одной из разновидностей хранимых процедур. Их исполнение происходит при выполнении для таблицы какого-либо оператора языка манипулирования данными (DML). Триггеры используются для проверки целостности данных, а также для отката транзакций.

Триггер – это откомпилированная SQL-процедура, исполнение которой обусловлено наступлением определенных событий внутри реляционной базы данных. Применение триггеров большей частью весьма удобно для пользователей базы данных. И все же их использование часто связано с дополнительными затратами ресурсов на операции ввода/вывода. В том случае, когда тех же результатов (с гораздо меньшими непроизводительными затратами ресурсов) можно добиться с помощью хранимых процедур или прикладных программ, применение триггеров нецелесообразно.

Триггеры – особый инструмент SQL-сервера, используемый для поддержания целостности данных в базе данных. С помощью ограничений целостности, правил и значений по умолчанию не всегда можно добиться нужного уровня функциональности. Часто требуется реализовать сложные алгоритмы проверки данных, гарантирующие их достоверность и реальность. Кроме того, иногда необходимо отслеживать изменения

значений таблицы, чтобы нужным образом изменить связанные данные. Триггеры можно рассматривать как своего рода фильтры, вступающие в действие после выполнения всех операций в соответствии с правилами, стандартными значениями и т.д.

Триггер представляет собой специальный тип хранимых процедур, запускаемых сервером автоматически при попытке изменения данных в таблицах, с которыми триггеры связаны. Каждый триггер привязывается к конкретной таблице. Все производимые им модификации данных рассматриваются как одна транзакция. В случае обнаружения ошибки или нарушения целостности данных происходит откат этой транзакции. Тем самым внесение изменений запрещается. Отменяются также все изменения, уже сделанные триггером.

Создает триггер только владелец базы данных. Это ограничение позволяет избежать случайного изменения структуры таблиц, способов связи с ними других объектов и т.п.

Триггер представляет собой весьма полезное и в то же время опасное средство. Так, при неправильной логике его работы можно легко уничтожить целую базу данных, поэтому триггеры необходимо очень тщательно отлаживать.

В отличие от обычной подпрограммы, триггер выполняется неявно в каждом случае возникновения триггерного события, к тому же он не имеет аргументов.

Приведение его в действие иногда называют запуском триггера. С помощью триггеров достигаются следующие цели:

- проверка корректности введенных данных и выполнение сложных ограничений целостности данных, которые трудно, если вообще возможно, поддерживать с помощью ограничений целостности, установленных для таблицы;
- выдача предупреждений, напоминающих о необходимости выполнения некоторых действий при обновлении таблицы, реализованном определенным образом;
- накопление аудиторской информации посредством фиксации сведений о внесенных изменениях и тех лицах, которые их выполнили;
- поддержка репликации.

Основной формат команды **CREATE TRIGGER** показан ниже:

```
<Определение_триггера>::=
  CREATE TRIGGER имя_триггера
  BEFORE | AFTER <триггерное_событие>
  ON <имя_таблицы>
  [REFERENCING
    <список_старых_или_новых_псевдонимов>]
  [FOR EACH { ROW | STATEMENT}]
  [WHEN(условие_триггера)]
  <тело_триггера>
```

Триггерные события состоят из вставки, удаления и обновления строк в таблице. В последнем случае для триггерного события можно указать конкретные имена столбцов таблицы. Время запуска триггера определяется с помощью ключевых слов **BEFORE** (триггер запускается до выполнения связанных с ним событий) или **AFTER** (после их выполнения).

Выполняемые триггером действия задаются для каждой строки (**FOR EACH ROW**), охваченной данным событием, или только один раз для каждого события (**FOR EACH STATEMENT**).

Обозначение <список_старых_или_новых_псевдонимов> относится к таким компонентам, как старая или новая строка (**OLD** / **NEW**) либо старая или новая таблица

(**OLD TABLE** / **NEW TABLE**). Ясно, что старые значения не применимы для событий вставки, а новые – для событий удаления.

При условии правильного использования триггеры могут стать очень мощным механизмом. Основное их преимущество заключается в том, что стандартные функции сохраняются внутри базы данных и согласованно активизируются при каждом ее обновлении. Это может существенно упростить приложения. Тем не менее следует упомянуть и о присущих триггеру недостатках:

- сложность: при перемещении некоторых функций в базу данных усложняются задачи ее проектирования, реализации и администрирования;
- скрытая функциональность: перенос части функций в базу данных и сохранение их в виде одного или нескольких триггеров иногда приводит к сокрытию от пользователя некоторых функциональных возможностей. Хотя это в определенной степени упрощает его работу, но, к сожалению, может стать причиной незапланированных, потенциально нежелательных и вредных побочных эффектов, поскольку в этом случае пользователь не в состоянии контролировать все процессы, происходящие в базе данных;
- влияние на производительность: перед выполнением каждой команды по изменению состояния базы данных СУБД должна проверить триггерное условие с целью выяснения необходимости запуска триггера для этой команды. Выполнение подобных вычислений сказывается на общей производительности СУБД, а в моменты пиковой нагрузки ее снижение может стать особенно заметным. Очевидно, что при возрастании количества триггеров увеличиваются и накладные расходы, связанные с такими операциями.

Неправильно написанные триггеры могут привести к серьезным проблемам, таким, например, как появление "мертвых" блокировок. Триггеры способны длительное время блокировать множество ресурсов, поэтому следует обратить особое внимание на сведение к минимуму конфликтов доступа.

2. РЕАЛИЗАЦИЯ ТРИГГЕРОВ В СРЕДЕ MS SQL SERVER

В реализации СУБД MS SQL Server используется следующий оператор создания или изменения триггера:

```
<Определение_триггера>::=
{CREATE | ALTER} TRIGGER имя_триггера
ON {имя_таблицы | имя_просмотра }
[WITH ENCRYPTION ]
{
{ { FOR | AFTER | INSTEAD OF }
{ [ DELETE] [,] [ INSERT] [,] [ UPDATE] }
[ WITH APPEND ]
[ NOT FOR REPLICATION ]
AS
  sql_оператор[...n]
} |
{ {FOR | AFTER | INSTEAD OF } { [INSERT] [,]
  [UPDATE] }
[ WITH APPEND]
[ NOT FOR REPLICATION]
AS
```

```
{ IF UPDATE(имя_столбца)
[ {AND | OR} UPDATE(имя_столбца)] [...n]
|
IF (COLUMNS_UPDATES(){оператор_бит_обработки}
  бит_маска_изменения)
{оператор_бит_сравнения }бит_маска [...n]}
sql_оператор [...n]
}
}
```

Триггер может быть создан только в текущей базе данных, но допускается обращение внутри триггера к другим базам данных, в том числе и расположенным на удаленном сервере.

Рассмотрим назначение аргументов из команды **CREATE** | **ALTER TRIGGER**.

Имя триггера должно быть уникальным в пределах базы данных. Дополнительно можно указать имя владельца.

При указании аргумента **WITH ENCRYPTION** сервер выполняет шифрование кода триггера, чтобы никто, включая администратора, не мог получить к нему доступ и прочитать его. Шифрование часто используется для скрытия авторских алгоритмов обработки данных, являющихся интеллектуальной собственностью программиста или коммерческой тайной.

Типы триггеров

В SQL Server существует два параметра, определяющих поведение триггеров:

AFTER. Триггер выполняется после успешного выполнения вызвавших его команд. Если же команды по какой-либо причине не могут быть успешно завершены, триггер не выполняется. Следует отметить, что изменения данных в результате выполнения запроса пользователя и выполнение триггера осуществляется в теле одной транзакции: если произойдет откат триггера, то будут отклонены и пользовательские изменения. Можно определить несколько **AFTER**-триггеров для каждой операции (**INSERT**, **UPDATE**, **DELETE**). Если для таблицы предусмотрено выполнение нескольких **AFTER**-триггеров, то с помощью системной хранимой процедуры **sp_settriggerorder** можно указать, какой из них будет выполняться первым, а какой последним. По умолчанию в SQL Server все триггеры являются **AFTER**-триггерами.

INSTEAD OF. Триггер вызывается вместо выполнения команд. В отличие от **AFTER**-триггера **INSTEAD OF**-триггер может быть определен как для таблицы, так и для просмотра. Для каждой операции **INSERT**, **UPDATE**, **DELETE** можно определить только один **INSTEAD OF**-триггер.

Триггеры различают по типу команд, на которые они реагируют.

Существует три типа триггеров:

INSERT TRIGGER – запускаются при попытке вставки данных с помощью команды **INSERT**.

UPDATE TRIGGER – запускаются при попытке изменения данных с помощью команды **UPDATE**.

DELETE TRIGGER – запускаются при попытке удаления данных с помощью команды **DELETE**.

Конструкции [DELETE] [,] [INSERT] [,] [UPDATE] и FOR | AFTER | INSTEAD OF } { [INSERT] [,] [UPDATE] определяют, на какую команду будет реагировать триггер. При его создании должна быть указана хотя бы одна команда. Допускается создание триггера, реагирующего на две или на все три команды.

Аргумент **WITH APPEND** позволяет создавать несколько триггеров каждого типа.

При создании триггера с аргументом **NOT FOR REPLICATION** запрещается его запуск во время выполнения модификации таблиц механизмами репликации.

Конструкция AS sql_оператор[...n] определяет набор SQL- операторов и команд, которые будут выполнены при запуске триггера.

Отметим, что внутри триггера не допускается выполнение ряда операций, таких, например, как:

- создание, изменение и удаление базы данных;
- восстановление резервной копии базы данных или журнала транзакций.

Выполнение этих команд не разрешено, так как они не могут быть отменены в случае отката транзакции, в которой выполняется триггер. Это запрещение вряд ли может каким-то образом сказаться на функциональности создаваемых триггеров. Трудно найти такую ситуацию, когда, например, после изменения строки таблицы потребуется выполнить восстановление резервной копии журнала транзакций.

Программирование триггера

При выполнении команд добавления, изменения и удаления записей сервер создает две специальные таблицы: **inserted** и **deleted**. В них содержатся списки строк, которые будут вставлены или удалены по завершении транзакции. Структура таблиц **inserted** и **deleted** идентична структуре таблиц, для которой определяется триггер. Для каждого триггера создается свой комплект таблиц **inserted** и **deleted**, поэтому никакой другой триггер не сможет получить к ним доступ. В зависимости от типа операции, вызвавшей выполнение триггера, содержимое таблиц **inserted** и **deleted** может быть разным:

команда INSERT – в таблице **inserted** содержатся все строки, которые пользователь пытается вставить в таблицу; в таблице **deleted** не будет ни одной строки; после завершения триггера все строки из таблицы **inserted** переместятся в исходную таблицу;

команда DELETE – в таблице **deleted** будут содержаться все строки, которые пользователь попытается удалить; триггер может проверить каждую строку и определить, разрешено ли ее удаление; в таблице **inserted** не окажется ни одной строки;

команда UPDATE – при ее выполнении в таблице **deleted** находятся старые значения строк, которые будут удалены при успешном завершении триггера. Новые значения строк содержатся в таблице **inserted**. Эти строки добавятся в исходную таблицу после успешного выполнения триггера.

Для получения информации о количестве строк, которое будет изменено при успешном завершении триггера, можно использовать функцию **@@ROWCOUNT**; она возвращает количество строк, обработанных последней командой. Следует подчеркнуть, что триггер запускается не при попытке изменить конкретную строку, а в момент выполнения команды изменения. Одна такая команда воздействует на множество строк, поэтому триггер должен обрабатывать все эти строки.

Если триггер обнаружил, что из 100 вставляемых, изменяемых или удаляемых строк только одна не удовлетворяет тем или иным условиям, то никакая строка не будет вставлена, изменена или удалена. Такое поведение обусловлено требованиями транзакции – должны быть выполнены либо все модификации, либо ни одной.

Триггер выполняется как неявно определенная транзакция, поэтому внутри триггера допускается применение команд управления транзакциями. В частности, при обнаружении нарушения ограничений целостности для прерывания выполнения триггера и отмены всех изменений, которые пытался выполнить пользователь, необходимо использовать команду **ROLLBACK TRANSACTION**.

Для получения списка столбцов, измененных при выполнении команд **INSERT** или **UPDATE**, вызвавших выполнение триггера, можно использовать функцию **COLUMNS_UPDATED()**. Она возвращает двоичное число, каждый бит которого, начиная с младшего, соответствует одному столбцу таблицы (в порядке следования

столбцов при создании таблицы). Если бит установлен в значение "1", то соответствующий столбец был изменен. Кроме того, факт изменения столбца определяет и функция **UPDATE** (имя_столбца).

Для удаления триггера используется команда

```
DROP TRIGGER {имя_триггера} [,...n]
```

Приведем примеры использования триггеров

Пример 1. Триггер для реализации ограничений на значение.

В добавляемой в таблицу **Сделка** записи количество проданного товара должно быть не меньше, чем его остаток из таблицы **Склад**.

Команда вставки записи в таблицу **Сделка** может быть, например, такой:

```
INSERT INTO Сделка VALUES (3, 1, -299, '01/08/2015')
```

Создаваемый триггер должен отреагировать на ее выполнение следующим образом: необходимо отменить команду, если в таблице Склад величина остатка товара оказалась меньше продаваемого количества товара с введенным кодом (в примере код товара=3). Во вставляемой записи количество товара указывается со знаком "+", если товар поставляется, и со знаком "-", если он продается. Представленный триггер настроен на обработку только одной добавляемой записи.

Листинг 1

```
CREATE TRIGGER Триггер_ins
ON Сделка FOR INSERT
AS
IF @@ROWCOUNT=1
BEGIN
        IF NOT EXISTS
    (SELECT *
    FROM inserted
    WHERE -inserted.количество<=ALL
            (SELECT Склад.Остаток
      FROM Склад,Сделка
      WHERE Склад.КодТовара=Сделка.КодТовара))
    BEGIN
            ROLLBACK TRAN
            PRINT 'Отмена поставки: товара на складе нет'
    END
END
```

Пример 2. Триггер для сбора статистических данных.

Создать триггер для обработки операции вставки записи в таблицу **Сделка**, например, такой команды:

```
INSERT INTO Сделка VALUES (3, 1, 200, '01/08/2015')
```

поставляется товар с кодом 3 от клиента с кодом 1 в количестве 200 единиц.

При продаже или получении товара необходимо соответствующим образом изменить количество его складского запаса. Если товара на складе еще нет, необходимо добавить соответствующую запись в таблицу **Склад**. Триггер обрабатывает только одну добавляемую строку.

Листинг 2

```
ALTER TRIGGER Триггер_ins
ON Сделка FOR INSERT
AS
DECLARE @x INT, @y INT
IF @@ROWCOUNT=1
--в таблицу Сделка добавляется запись о поставке товара
BEGIN
--количество проданного товара должно быть не
--меньше, чем его остаток из таблицы Склад
IF NOT EXISTS
        (SELECT *
        FROM inserted
        WHERE -inserted.количество<=ALL
                (SELECT Склад.Остаток
                FROM Склад,Сделка
                WHERE Склад.КодТовара=Сделка.КодТовара))
  BEGIN
        ROLLBACK TRAN
        PRINT 'откат товара нет '
  END
--если записи о поставленном товаре еще нет,
--добавляется соответствующая запись в таблицу Склад
 IF NOT EXISTS
        (SELECT *
        FROM Склад c, inserted i
        WHERE c.КодТовара=i.КодТовара )
    INSERT INTO Склад (КодТовара,Остаток)
 ELSE
--если запись о товаре уже была в таблице
--Склад, то определяется код и количество
--товара из добавленной в таблицу Сделка записи
  BEGIN
        SELECT @y=i.КодТовара, @x=i.Количество
        FROM Сделка c, inserted i
        WHERE c.КодТовара=i.КодТовара
--и производится изменения количества товара в таблице Склад
        UPDATE Склад
        SET Остаток=остаток+@x
        WHERE КодТовара=@y
  END
END
```

Пример 3. Использование триггера для обработки операции удаления записи из таблицы

Создать триггер для обработки операции удаления записи из таблицы **Сделка**, например, такой команды:

```
DELETE FROM Сделка WHERE КодСделки=4
```

Для товара, код которого указан при удалении записи, необходимо откорректировать его остаток на складе. Триггер обрабатывает только одну удаляемую запись.

Листинг 3

```
CREATE TRIGGER Триггер_del
ON Сделка FOR DELETE
AS
IF @@ROWCOUNT=1 -- удалена одна запись
BEGIN
   DECLARE @y INT, @x INT
--определяется код и количество товара из
--удаленной из таблицы Склад записи
            SELECT @y=КодТовара, @x=Количество
            FROM deleted
--в таблице Склад корректируется количество товара
            UPDATE Склад
            SET Остаток=Остаток-@x
            WHERE КодТовара=@y
END
```

Пример 4. Использование триггера для обработки операции изменения записи из таблицы

Создать триггер для обработки операции изменения записи в таблице **Сделка**, например, такой командой:

```
UPDATE Сделка SET количество=количество-10
WHERE КодТовара=3
```

во всех сделках с товаром, имеющим код, равный 3, уменьшить количество товара на 10 единиц.

Указанная команда может привести к изменению сразу нескольких записей в таблице **Сделка**. Поэтому покажем, как создать триггер, обрабатывающий не одну запись. Для каждой измененной записи необходимо для старого (до изменения) кода товара уменьшить остаток товара на складе на величину старого (до изменения) количества товара и для нового (после изменения) кода товара увеличить его остаток на складе на величину нового (после изменения) значения. Чтобы обработать все измененные записи, введем курсоры, в которых сохраним все старые (из таблицы **deleted**) и все новые значения (из таблицы **inserted**).

Листинг 4

```
CREATE TRIGGER Триггер_upd
ON Сделка FOR UPDATE
AS
DECLARE @x INT, @x_old INT, @y INT, @y_old INT
```

```
-- курсор с новыми значениями
DECLARE CUR1 CURSOR FOR
            SELECT КодТовара,Количество
            FROM inserted
-- курсор со старыми значениями
DECLARE CUR2 CURSOR FOR
            SELECT КодТовара,Количество
            FROM deleted
OPEN CUR1
OPEN CUR2
-- перемещаемся параллельно по обоим курсорам
  FETCH NEXT FROM CUR1 INTO @x, @y
  FETCH NEXT FROM CUR2 INTO @x_old, @y_old
  WHILE @@FETCH_STATUS=0
BEGIN
--для старого кода товара уменьшается его
--количество на складе
            UPDATE Склад
            SET Остаток=Остаток-@y_old
            WHERE КодТовара=@x_old
--для нового кода товара, если такого товара
--еще нет на складе, вводится новая запись
    IF NOT EXISTS
            (SELECT * FROM Склад
            WHERE КодТовара=@x)
                        INSERT INTO Склад(КодТовара,Остаток)
                        VALUES (@x,@y)
    ELSE
--иначе для нового кода товара увеличивается
--его количество на складе
            UPDATE Склад
            SET Остаток=Остаток+@y
            WHERE КодТовара=@x
  FETCH NEXT FROM CUR1 INTO @x, @y
  FETCH NEXT FROM CUR2 INTO @x_old, @y_old
END
CLOSE CUR1
CLOSE CUR2
DEALLOCATE CUR1
DEALLOCATE CUR2
```

В рассмотренном триггере отсутствует сравнение количества товара при изменении записи о сделке с его остатком на складе.

Пример 5. Триггер с генератором сообщения об ошибке

Исправим этот недостаток. Для генерирования сообщения об ошибке используем в теле триггера команду MS SQL Server **RAISERROR**, аргументами которой являются текст сообщения, уровень серьезности и статус ошибки.

Листинг 5

```
ALTER TRIGGER Триггер_upd
ON Сделка FOR UPDATE
AS
DECLARE @x INT, @x_old INT, @y INT, @y_old INT, @o INT
DECLARE CUR1 CURSOR FOR
            SELECT КодТовара,Количество
            FROM inserted
DECLARE CUR2 CURSOR FOR
            SELECT КодТовара,Количество
            FROM deleted
OPEN CUR1
OPEN CUR2
  FETCH NEXT FROM CUR1 INTO @x, @y
  FETCH NEXT FROM CUR2 INTO @x_old, @y_old
  WHILE @@FETCH_STATUS=0
   BEGIN
            SELECT @o=остаток
            FROM Склад
            WHERE КодТовара=@x
    IF @o<-@y
    BEGIN
     RAISERROR('откат',16,10)
     CLOSE CUR1
     CLOSE CUR2
     DEALLOCATE CUR1
     DEALLOCATE CUR22
     ROLLBACK TRAN
     RETURN
    END
            UPDATE Склад
            SET Остаток=Остаток-@y_old
            WHERE КодТовара=@x_old
    IF NOT EXISTS
            (SELECT * FROM Склад
            WHERE КодТовара=@x)
                    INSERT INTO Склад(КодТовара,Остаток)
                    VALUES (@x, @y)
   ELSE
```

```
        UPDATE Склад
        SET Остаток=Остаток+@y
        WHERE КодТовара=@x
    FETCH NEXT FROM CUR1 INTO @x, @y
    FETCH NEXT FROM CUR2 INTO @x_old, @y_old
END
CLOSE CUR1
CLOSE CUR2
DEALLOCATE CUR1
DEALLOCATE CUR2
```

Пример 6. В примере 5 происходит отмена всех изменений при невозможности реализовать хотя бы одно из них. Создадим триггер, позволяющий отменять изменение только некоторых записей и выполнять изменение остальных.

В этом случае триггер выполняется не после изменения записей, а вместо команды изменения.

Листинг 6

```
ALTER TRIGGER Триггер_upd
ON Сделка INSTEAD OF UPDATE
AS
DECLARE @k INT, @k_old INT
DECLARE @x INT, @x_old INT, @y INT
DECLARE @y_old INT, @o INT
DECLARE CUR1 CURSOR FOR
            SELECT КодСделки, КодТовара, Количество
            FROM inserted
DECLARE CUR2 CURSOR FOR
            SELECT КодСделки, КодТовара, Количество
            FROM deleted
OPEN CUR1
OPEN CUR2
  FETCH NEXT FROM CUR1 INTO @k,@x, @y
  FETCH NEXT FROM CUR2 INTO @k_old, @x_old, @y_old
  WHILE @@FETCH_STATUS=0
BEGIN
            SELECT @o=остаток
            FROM Склад
            WHERE КодТовара=@x
    IF @o>=-@y
    BEGIN
        RAISERROR('изменение',16,10)
            UPDATE Сделка SET Количество=@y, КодТовара=@x
            WHERE КодСделки=@k
                    UPDATE Склад
```

```
                    SET Остаток=Остаток-@y_old
                    WHERE КодТовара=@x_old
        IF NOT EXISTS
                (SELECT * FROM Склад
                WHERE КодТовара=@x)
                    INSERT INTO Склад(КодТовара, Остаток)
                    VALUES (@x,@y)
        ELSE
                UPDATE Склад
                SET Остаток=Остаток+@y
                WHERE КодТовара=@x
        END
       ELSE
            RAISERROR('запись не изменена',16,10)
      FETCH NEXT FROM CUR1 INTO @k,@x, @y
      FETCH NEXT FROM CUR2 INTO @k_old, @x_old, @y_old
END
CLOSE CUR1
CLOSE CUR2
DEALLOCATE CUR1
DEALLOCATE CUR2
```

3. ВЫПОЛНЕНИЕ ЗАДАНИЯ

Задание к работе:

1. Получить у преподавателя индивидуальный вариант задания.
2. Ознакомиться с вариантом задания.

Для освоения особенностей программирования в MS SQL Server средствами встроенного языка Transact-SQL используем базу данных, которая была создана в лабораторной работе №9 (Часть I).

При выполнении заданий обращайте внимание на соответствие названий БД, таблиц и других объектов проекта.

Построить триггеры по каждому пункту согласно представленных вариантов заданий.

Примеры вариантов заданий для построения триггеров обеспечивающих целостность данных:

1. Создать FOR/AFTER INSERT триггер "Добавление", выводящий на экран сообщение "Запись добавлена" при добавлении новой записи в таблицу A.
2. Создать FOR/AFTER DELETE триггер "Удаление", выводящий на экран сообщение "Запись удалена" при удалении указанной записи из таблицы A.
3. Создать FOR/AFTER UPDATE триггер "Изменение", выводящий на экран сообщение "Запись изменена" при изменении указанной записи в таблице A.
4. Создать триггер для одновременной обработки операции удаления DELETE и обновления UPDATE записи в таблице с генерированием сообщения об ошибке с помощью функции RAISERROR.

5. Создать триггер для реализации ограничения на значение и выполнить откат.
6. Создать триггер для сбора статистических данных.

Примеры вариантов заданий на условную конструкцию IF:

1. Подсчитать количество X в таблице A. Если X в таблице от 2 до 5, то ничего не сообщать, в противном случае вывести сообщение вида "В таблице A ..." (вместо многоточия указать и точное количество X).
2. Подсчитать сумму (минимальную, максимальную, среднюю стоимость X в таблице A. Если полученная сумма в диапазоне от 1000 до 5000, то ничего не сообщать, в противном случае вывести сообщение вида "Сумма (Мак, Мин) X = …" (вместо многоточия поставить точную сумму).

Примеры вариантов заданий на использование цикла WHILE:

1. Определить количество записей в таблице A. Пока записей меньше (больше) 15, делать в цикле добавление записи в таблицу с автоматическим наращиванием значения ключевого поля, а вместо (например, названия) ставить значение 'значение не известно'.

Примеры вариантов заданий на знание функций для работы со строковыми переменными:

1. Удалить в тексте лишние пробелы. Лишними считаются те, которые идут непосредственно за пробелом. Подсчитать количество исправлений.
2. Подсчитать количество встреч каждой из следующих букв: "а", "в", "и", "п" в базовом тексте.
3. Подсчитать доли процентов встречи следующих букв: "е", "о", если суммарный процент встречаемости всех этих букв равен 100% или процент встречаемости е% + о% равен 100%.
4. По правилам оформления машинописных текстов перед знаками .,!?:; пробелы не ставятся, но обязательно ставятся после этих знаков. Удалите лишние пробелы. Подсчитать количество исправлений.
5. По правилам оформления машинописных текстов перед знаками .,!?:; пробелы не ставятся, но обязательно ставятся после этих знаков. Расставьте недостающие пробелы. Подсчитать количество исправлений.
6. Найти из исходного текста второе предложение и вернуть его в переменную Perem, а также вывести на экран весь исходный текст и найденное предложение.
7. Удалить из базового текста 2, 4, 6, 8 слова.
8. Вставить в базовый текст вместо букв «а» – «АА».
9. Вставить в базовый текст вместо букв «е» и «о» – «ББ».
10. Поменять местами первое и последнее слова в базовом тексте.

4. ОФОРМЛЕНИЕ ОТЧЕТА

Содержание отчета:

1. Представить скрипт выполненных запросов для индивидуального варианта задания.

Примечание:

1. Отчет должен быть представлен в соответствии с требованиями, принятыми в ВУЗе.

2. Варианты заданий должны быть индивидуальными у каждого студента.

Продемонстрируйте Вашу работу преподавателю!

КОНТРОЛЬНЫЕ ВОПРОСЫ:

1. Что такое триггер?

2. Приведите синтаксис создания триггера в Transact-SQL.

3. Перечислите типы триггеров в Transact-SQL.

4. Что такое AFTER- и BEFORE- триггеры?

5. Как создать триггер, выполняемый после события, произошедшего с таблицей?

6. Как создать триггер, выполняемый вместо события, происходящего с таблицей?

7. Для каких целей используют команду ROLLBACK TRANSACTION в MS SQL Server?

8. Для чего предназначены таблицы inserted и deleted, создаваемые при работе триггеров?

9. Для чего предназначены операторы Transact-SQL ROLLBACK TRANSACTION и COMMIT TRANSACTION?

10. Перечислите известные вам функции для работы со строковыми переменными.

11. Для чего предназначена функция %%ROWCOUNT?

СПИСОК ЛИТЕРАТУРЫ

Основная литература:

1. Н. Вирт. Алгоритмы и структуры данных. – М.:"Мир",1989.

2. М. Сибуя, Т. Ямамото. Алгоритмы обработки данных. – М.:"Мир",1986