Лабораторная работа № 8

Использование хранимых процедур и триггеров при работе с базами данных

Цель работы: изучить и использовать возможности хранимых процедур и триггеров.

Продолжительность работы - 4 часа.

Теоретитческие сведения

Хранимые процедуры

Хранимая процедура представляет собой оформленный особым образом пакет, который хранится в базе данных. Хранимые процедуры отличаются от пакетов тем, что в них допускается использование входных и выходных параметров, а также возвращаемых значений.

Сведения о правилах создания и использования хранимых процедур были рассмотрены на лекциях.

В качестве примеров далее приведены описания хранимых процедур для учебной базы данных со сведениями о жителях Зеленограда. Эти хранимые процедуры предназначены для регистрации сведений об источнике дохода, используемом или потерянном жителем:

- `pAdd_D` добавляет строку в таблицу Have_D и корректирует общий доход;
- `pGetMoney` определяет размер источника дохода по его идентификатору;
- `pAddMoney` к общему доходу жителя прибавляет размер источника дохода (с заданным идентификатором);
- `pDel_D` удаляет строку из таблицы Have_D и корректирует общий доход;
- `pSubMoney` из общего дохода жителя вычитает размер источника дохода (с заданным идентификатором).

В программном коде процедур использована системная переменная `@@RowCount`, в которой запоминается число строк выбранных, добавленных, обновленных или удаленных предшествующим SQL-оператором. После заголовка каждой

процедуры помещены комментарии с пояснением назначения ее параметров и примером использования.

Хранимая процедура pGetMoney

```
CREATE PROCEDURE pGetMoney(
   @Id  INT,
   @Money MONEY  OUTPUT
   )
AS
-- Определяет размер источника дохода по его идентификатору
-- Входной параметр @Id нужен для поиска размера
-- Выходной параметр @Money будет содержать размер
-- Процедура возвращает код –100, если значение идентификатора
--                                в таблице PROFIT отсутствует
-- Пример использования этой хранимой процедуры:
-- Declare @Qty money,
--         @RetCode int;
-- EXEC @RetCode = pGetMoney 5, @Qty OUTPUT;
-- SELECT @RetCode, @Qty;
SET NOCOUNT ON
SELECT @Money = MONEYS
  FROM PROFIT
  WHERE Id = @Id;
IF @@RowCount=0
   RETURN -100;
ELSE
   RETURN 0;
```

Хранимая процедура pAddMoney

```
CREATE PROCEDURE pAddMoney(
   @Nom  INT,
   @Id   INT
   )
AS
 -- Прибавляет к общему доходу SumD жителя размер источника
дохода
 -- Входной параметр @Nom нужен для поиска жителя
 -- Входной параметр @Id нужен для поиска размера
 -- Процедура возвращает код –100, если источник дохода с
 --                            идентификатором @Id отсутствует
 -- Процедура возвращает код –101, если житель с номером @Nom
 --                                отсутствует
 -- Пример использования этой хранимой процедуры:
 -- Declare @RetCode int;
 -- EXEC @RetCode = pAddMoney 1, 9;
-- SELECT @RetCode;
SET NOCOUNT ON
Declare @Qty     money,
        @RetCode int;
EXEC @RetCode = pGetMoney @Id, @Qty OUTPUT;
IF @RetCode<>0
```

```
   RETURN -100;
UPDATE PERSON SET SumD = SumD + @Qty
  WHERE Nom = @Nom;
IF @@RowCount=0
   RETURN -101;
RETURN;
```

Хранимая процедура pAdd_D

```
CREATE PROCEDURE pAdd_D(
   @Nom INT,
   @Id  INT,
   @Comment CHAR(30)
  )
AS
-- Добавляет строку  в таблицу Have_D и корректирует общий доход
-- Входной параметр @Nom задает номер жителя
-- Входной параметр @Id задает индентификатор источника дохода
-- Входной параметр @Comment содержит комментарий
-- Процедура возвращает код –100, если источник дохода с
--                             идентификатором @Id отсутствует
-- Процедура возвращает код –101, если житель с номером @Nom
--                                  отсутствует
-- Пример использования этой хранимой процедуры:
-- Declare @RetCode int;
-- EXEC @RetCode = pAdd_D 1, 9, NULL;
-- SELECT @RetCode;
SET NOCOUNT ON
Declare @RetCode int;
EXEC @RetCode = pAddMoney @Nom, @Id;
IF @RetCode<>0
   RETURN @RetCode;
INSERT INTO Have_D
   VALUES (@Nom, @Id, @Comment);
RETURN;
```

Хранимая процедура pSubMoney

```
CREATE PROCEDURE pSubMoney(
   @Nom  INT,
   @Id   INT
  )
AS
 -- Вычитает из общего дохода SumD жителя размер источника дохода
 -- Входной параметр @Nom нужен для поиска жителя
 -- Входной параметр @Id нужен для поиска размера
 -- Процедура возвращает код –100, если источник дохода с
 --                            идентификатором @Id отсутствует
 -- Процедура возвращает код –101, если житель с номером @Nom
 --                                  отсутствует
 -- Пример использования этой хранимой процедуры:
 -- Declare @RetCode int;
 -- EXEC @RetCode = pSubMoney 1, 9;
```

```
-- SELECT @RetCode;
SET NOCOUNT ON
Declare @Qty     money,
        @RetCode int;
EXEC @RetCode = pGetMoney @Id, @Qty OUTPUT;
IF @RetCode<>0
   RETURN -100;
UPDATE PERSON SET SumD = SumD - @Qty
  WHERE Nom = @Nom;
IF @@RowCount=0
   RETURN -101;
RETURN;
```

Хранимая процедура pDel_D

```
CREATE PROCEDURE pDel_D(
   @Nom INT,
   @Id  INT
   )
AS
-- Удаляет строку из таблицы Have_D и корректирует общий доход
-- Входной параметр @Nom задает номер жителя
-- Входной параметр @Id задает индентификатор источника дохода
-- Процедура возвращает код –100, если источник дохода с
--                              идентификатором @Id отсутствует
-- Процедура возвращает код –101, если житель с номером @Nom
--                                 отсутствует
-- Процедура возвращает код –102, если в таблице Have_D
--                  отсутствует строка со значениями @Nom и @Id
-- Пример использования этой хранимой процедуры:
-- Declare @RetCode int;
-- EXEC @RetCode = pDel_D 1, 9;
-- SELECT @RetCode;
SET NOCOUNT ON
Declare @RetCode int;
DELETE FROM Have_D
   WHERE (Nom = @Nom) AND (Id = @Id)
IF @@RowCount=0
   RETURN -102;
EXEC @RetCode = pSubMoney @Nom, @Id;
IF @RetCode<>0
   RETURN @RetCode;
RETURN;
```

Триггеры и их особенности в СУБД SQL Server

Триггер — это хранимая процедура особого типа, вызываемая на выполнение в ответ на определенные события. Триггеры подразделяются на два основных класса: триггеры языка определения данных (Data Definition Language — DDL) и триггеры языка манипулирования данными (Data Manipulation Language — DML).

Триггеры DDL активизируются в ответ на внесение каких-либо изменений в структуру базы данных с помощью операторов CREATE, ALTER, DROP и т.п.

Триггеры DML, более широко применяемые на практике, представляют собой фрагменты кода, закрепленные за конкретной таблицей или представлением.

В отличие от хранимых процедур, при использовании которых требуется их явный вызов на выполнение, триггеры вызываются на выполнение автоматически при обнаружении события (или событий), связанного с выполнением операций над таблицей, за которой закреплен триггер. Триггеры не могут быть вызваны явно; единственный способ обеспечения вызова состоит в выполнении требуемого действия над таблицей, за которой закреплен триггер.

Различия между хранимыми процедурами и триггерами не ограничиваются тем, что невозможен явный вызов триггера. Хранимые процедуры не только вызываются явно, но и отличаются двумя особенностями, которыми не обладают триггеры: (1) принимают и передают параметры и (2) возвращают коды завершения.

Триггеры не имеют параметров, но в коде триггера может использоваться определенный механизм, позволяющий выяснить, на какие строки должно распространяться действие триггера. Еще одна особенность триггеров состоит в том, что в них допускается применение команды RETURN, но они не позволяют возвращать конкретные значения кода завершения (поскольку явный вызов триггера не предусмотрен, то не определена точка вызова, в которую можно было бы возвратить код завершения).

Возможность закрепления триггеров DML за конкретными операторами определяется тем, что в языке SQL предусмотрены три типа запросов, предназначенных для внесения изменений в данные. В связи с этим предусмотрены три основных типа триггеров, а также дополнительные типы, создаваемые с учетом одновременного возникновения и совпадения событий, обусловленных выполнением запросов этих типов. Кроме того, выбор типа триггера зависит от того, как связаны во времени запуск триггера и возникновение соответствующих событий. Основные типы триггеров DML перечислены ниже.

1. Триггеры INSERT.
2. Триггеры DELETE.
3. Триггеры UPDATE.

4. Триггеры, создаваемые с учетом одновременного возникновения и совпадения событий.

В СУБД SQL Server существуют два подкласса триггеров DML: INSTEAD OF (триггер замены операции) и AFTER (триггер, выполняющийся сразу после операции), отличающиеся своим назначением, моментом выполнения и производимым эффектом (табл. 1).

Таблица 1

Сравнение характеристик подклассов триггеров DML

Характеристика	Триггер INSTEAD OF	Триггер AFTER
Эффект оператора DML	Автоматически откатывается	Выполняется, если триггер сам не откатит транзакцию
Момент выполнения триггера	Перед проверкой ограничений первичного и внешнего ключей	После выполнения транзакции, но перед ее подтверждением
Количество возможных событий таблицы	Одно	Несколько
Возможность применения к представлениям	Есть	Отсутствует
Рекурсивное срабатывание	Отсутствует	Зависит от параметров настройки СУБД

Оператор DML не может повлиять на срабатывание триггера, однако триггер можно временно отключить оператором ALTER TABLE с параметром DISABLE TRIGGER:

```
ALTER TABLE имя_таблицы DISABLE TRIGGER имя_триггера
```

Для включения триггера применяется тот же оператор, но с параметром ENABLE TRIGGER:

```
ALTER TABLE имя_таблицы ENABLE TRIGGER имя_триггера
```

Триггер AFTER

Триггер AFTER создается оператором следующего вида:

```
CREATE TRIGGER имя_триггера ON имя_таблицы
AFTER операция, ...
AS
   программный_код_триггера
```

Созданный триггер выполняется в ответ на событие, возникающее после завершения указанной за ключевым словом AFTER операции добавления (Insert), удаления (Delete) или обновления (Update), но до

подтверждения фиксации результатов модификации. Например, триггер, созданный оператором

```
CREATE TRIGGER ai_PROFIT_trig ON PROFIT
AFTER Insert
AS
   PRINT('Добавлена строка');
```

выводит сообщение `'Добавлена строка'` при добавлении каждой строки в таблицу `PROFIT`, в чем можно убедиться, если воспользоваться оператором

```
INSERT INTO PROFIT (Id, Source, Moneys)
       VALUES (9, 'Менеджер', 20000.00)
```

Предложение AFTER

Предложение AFTER позволяет указать операцию (или операции), приводящие к запуску триггера. Может быть предусмотрен запуск триггера при выполнении оператора INSERT, UPDATE или DELETE либо любого сочетания этих трех операторов, например:

```
AFTER INSERT, DELETE
```

или

```
AFTER UPDATE, INSERT
```

или

```
AFTER DELETE
```

Триггеры, объявляемые с предложением AFTER, могут быть закреплены только за таблицами; закрепление этих триггеров за представлениями не допускается.

Триггеры INSERT

Код любого триггера, который объявлен с ключевыми словами AFTER INSERT, вызывается на выполнение каждый раз, когда кто-либо вставляет новую строку в таблицу, за которой закреплен триггер. Для каждой вставляемой строки СУБД SQL Server создает копию этой новой строки и вставляет ее в специальную таблицу, существующую только в области определения данного триггера. Эта таблица называется INSERTED. Наиболее важная особенность таблицы INSERTED состоит в том, что она сохраняется только до тех пор, пока действует триггер, с которым она связана, т.е. таблица INSERTED существует только с момента запуска триггера и до момента завершения его выполнения.

Триггеры DELETE

Триггеры DELETE действуют во многом аналогично триггерам INSERT, не считая того, что связанная с ними таблица INSERTED пуста

(и это очевидно, поскольку осуществляется удаление, а не вставка строк, поэтому отсутствуют строки, копия которых должна находиться в таблице INSERTED). Вместо этого копия каждой удаленной строки вставляется в другую таблицу, называемую DELETED. Эта таблица, как и таблица INSERTED, имеет область определения, которая ограничивается исключительно продолжительностью существования триггера.

Триггеры UPDATE

Триггеры UPDATE имеют свою специфику. Запуск кода триггера, объявленного с ключевыми словами AFTER UPDATE, происходит при каждом внесении изменения в строку, существующую в таблице. А особенность триггера UPDATE состоит в том, что отсутствует такая таблица, как UPDATED. Вместо этого в СУБД SQL Server операция модификации каждой строки трактуется так, как будто удаляется существующая строка и вставляется полностью новая. Вполне очевидно, что из этого следует такой вывод: триггер, объявленный с ключевыми словами AFTER UPDATE, обслуживается с помощью не одной, а двух специальных таблиц, INSERTED и DELETED, количество строк в которых одинаково.

Виртуальные таблицы Inserted и Deleted

Упомянутые ранее специальные таблицы INSERTED и DELETED можно рассматривать как представления журнала транзакций, т.е. виртуальные таблицы. Их содержимое для каждого оператора DML в краткой форме отражено в табл.2, которая показывает, что таблица DELETED содержит строки в состоянии до применения оператора DML, а таблица INSERTED — после применения.

Как уже отмечалось, эти виртуальные таблицы имеют очень ограниченную область определения. К ним можно обращаться только внутри триггера; хранимые процедуры, вызываемые триггером, не имеют доступа к этим таблицам.

В следующем примере таблица Inserted используется для создания отчета об изменении места проживания гражданина:

```
CREATE TRIGGER aiu_PERSON_trig ON Person
AFTER Insert, Update
AS
SET NoCount ON
IF Update(Adr)
   SELECT Inserted.FIO + ' изменил место проживания на '
                       + Inserted.Adr FROM Inserted;
```

Таблица 2

Содержимое виртуальных таблиц Inserted и Deleted

Оператор DML	Виртуальная таблица	
	Inserted	Deleted
INSERT	Вставленные строки	Пустая
UPDATE	Строки базы данных после обновления	Строки базы данных до обновления
DELETE	Пустая	Строки, подлежащие удалению

В триггере использована функция `Update()`, которая возвращает значение true для указанного в качестве аргумента столбца, если на него повлиял оператор DML. При выполнении операторов

```
UPDATE Person SET Adr = 'Зеленоград, 801-1'
  WHERE Adr = 'Зеленоград, 903-9';
INSERT INTO Person (Nom, FIO, Rdate, Pol, SumD, Adr)
  VALUES(100,'Ильин Илья Ильич','15.05.2005','M', 0,
          'Зеленоград, 801-1');
```

триггером будут выводится сообщения

```
Федоров Федор Иванович     изменил место проживания на Зеленоград, 801-1
Ильин Илья Ильич           изменил место проживания на Зеленоград, 801-1
```

Триггер INSTEAD OF

Триггер INSTEAD OF создается оператором следующего вида:

```
CREATE TRIGGER имя_триггера ON имя_таблицы
INSTEAD OF операция
AS
   программный_код_триггера
```

Созданный триггер выполняется в ответ на событие, возникающее в начале операции, указанной за ключевым словом INSTEAD OF. Триггер INSTEAD OF заменяет транзакцию (т.е. выполняется вместо нее). Это подобно тому, как если бы триггер автоматически откатил операцию, для которой был создан.

Каждая таблица ограничена возможностью использования только одного триггера замены для каждого из своих событий. В то же время триггеры INSTEAD OF могут применяться к представлениям точно так же, как к таблицам.

Триггеры INSTEAD OF особенно полезны, когда заранее известно, что оператор DML, запустивший триггер, почти наверняка будет отменен, а вместо этого оператора должна быть реализована некоторая логика, например:

- оператор DML пытается обновить необновляемое представление. В этом случае триггер INSTEAD OF обновляет таблицы, на которых построено данное представление;
- оператор DML пытается обновить величину общего дохода жителя, которая должна вычисляться как сумма размеров отдельных источников дохода. В этом случае триггер INSTEAD OF подсчитывает эту сумму, а затем обновляет общий доход;
- оператор DML пытается удалить из главной таблицы строку, имеющую связи со строками подчененной таблицы, а триггер INSTEAD OF предварительно удаляет связанные строки подчиненной таблицы.

В следующем примере создается и тестируется триггер замены операции вставки:

```
CREATE TRIGGER ioi_PERSON_trig ON Person
INSTEAD OF Insert
AS
PRINT 'Триггер замены вставки'
go
INSERT INTO Person (Nom, FIO, Rdate, Pol, SumD, Adr)
       VALUES(101,'Петров Петр Петрович','16.10.2002','M', 0,
          'Зеленоград, 801-1');
```

Будет получен следующий результат:

```
Триггер замены вставки
(1 row(s) affected)
```

Как видно из результата, отображаются предусмотренное в триггере сообщение и отчет о том, что была обработана одна строка. Однако попытка извлечения фамилии, имени и отчества жителя с номером 101 завершится неудачей:

```
SELECT FIO FROM Person WHERE Nom = 101
                              FIO
                    ------------
(0 row(s) affected)
```

В данном случае оператор INSERT отработал так, как будто одна строка была обработана, хотя действие этого оператора было блокировано триггером INSTEAD OF. Вместо вставки строки была выполнена операция вывода на экран предусмотренного в триггере сообщения. При этом созданный ранее триггер AFTER остался в силе, однако его сообщение не было выведено на экран, поскольку операция вставки не выполнилась.

Приведенный ниже триггер замены операции удаления активизируется при попытке удалить строку из главной таблицы

PERSON и предварительно удаляет из подчиненной таблицы HAVE_D связанные строки, а затем строку из главной таблицы:

```
CREATE TRIGGER iod_PERSON_trig ON Person
INSTEAD OF Delete
AS
DECLARE @x int;
SELECT @x=Deleted.Nom from Deleted;
IF EXISTS (SELECT * from HAVE_D WHERE Nom = @x)
          DELETE FROM HAVE_D WHERE Nom = @x;
DELETE FROM PERSON WHERE Nom = @x;
```

Из используемой в этом триггере виртуальной таблицы DELETED извлекается номер жителя, сведения о котором удаляются, и сохраняется в локальной переменной. Из-за отсутствия рекурсивности (см. табл. 1) сам триггер активизируется только один раз оператором DELETE (например, `DELETE FROM PERSON WHERE Nom=5`), поэтому выполнение содержащегося в описании триггера оператора `DELETE FROM PERSON WHERE Nom = @x` происходит единожды, что и требуется.

Лабораторное задание

1. При домашней подготовке:

а) ознакомиться с описанием лабораторной работы и устно ответить на контрольные вопросы;

б) для базы данных, созданной для своего варианта в лабораторной работе № 2, запрограммировать хранимые процедуры для добавления и удаления строк одной из подчиненных таблиц по образцу процедур, приведенных в описании лабораторной работы, предусмотрев хранение в столбце NORR (Number Of Related Rows) главной таблицы количества строк подчиненной таблицы, связанных с соответствующей строкой главной таблицы;

в) написать пакеты для тестирования хранимых процедур;.

г) запрограммировать триггеры, которые срабатывают при добавлении и удалении строк выбранной подчиненной таблицы, увеличивая и уменьшая значение столбца NORR в строках главной таблицы, а также триггер, срабатывающий при удалении строк из главной таблицы, чтобы происходило каскадное удаление подчиненных строк;

д) написать пакеты для тестирования триггеров.

2. Создать и протестировать хранимые процедуры и триггеры..

3. Разработать приложение для отображения строк главной и подчиненной таблиц и с помощью кнопок навигатора проверить выполнение операций добавления и удаления строк при включенных и отключенных триггерах.

4. Оформить отчет по результатам выполнения лабораторной работы.

Порядок выполнения работы

1. Подготовить описания хранимых процедур и триггеров для своей базы данных, а также пакеты для их тестирования.

2. На Терминале 4100 запустить утилиту SSMS.

3. Дополнить выбранную главную таблицу столбцом NORR с нулевым значением по умолчанию и заполнить его значениями, равными числу подчиненных строк.

4. Создать хранимые процедуры и убедиться в их наличии в базе данных, выбрав ветви Programability и Stored Procedures в окне Обозревателя объектов.

5. Выполнить тестирование хранимых процедур и прокомментировать результаты в отчете.

6. Создать триггеры и убедиться в их наличии в базе данных, выбрав ветвь Triggers для таблиц базы данных в окне Обозревателя объектов.

7. Выполнить тестирование триггеров и прокомментировать в отчете результаты их действия.

8. На Терминале 4100 запустить систему C++Builder командой меню Пуск | Embarcadero RAD Studio 2010 | C++Builder.

9. Создать приложение для отображения строк главной и подчиненной таблиц и с помощью кнопок навигатора проверить выполнение операций добавления и удаления строк, обращая внимание на отображение содержимого таблиц и изменение значений в столбце NORR. Учтите, что изменение содержимого таблиц может отображаться с некоторой задержкой.

10. Отметить в отчете выявленные особенности функционирования приложения и сохранить его в отдельной папке на устройстве C терминального компьютера.

11. Временно отключить срабатывание триггеров, связанных с удалением строк из подчиненной и главной таблиц.

12. С помощью имеющегося приложения проверить выполнение операций удаления строк, обращая внимание на отображение содержимого таблиц и значений в столбце NORR.

13. Отметить в отчете изменения, произошедшие в функционировании приложения, и объяснить их.

14. Включить срабатывание триггеров, связанных с удалением строк из подчиненной и главной таблиц.

15. Оформить в отчете и показать преподавателю результаты лабораторной работы и защитить ее.

Требования к отчету

Отчет должен содержать:

1) название и цель работы;
2) описания хранимых процедур и триггеров;
3) пакеты для тестирования хранимых процедур и триггеров;
4) комментарии к результатам тестирования;
5) особенности функционирования приложения, отображающего строки главной и подчиненной таблиц, с включенными и отключенными триггерами.

Контрольные вопросы

1. Что представляет собой хранимая процедура?
2. Как создать хранимую процедуру?
3. Как выполнить хранимую процедуру?
4. Как передать данные в хранимую процедуру?
5. Как получить результаты из хранимой процедуры?
6. Каково назначение команды `Return`?
7. Как получить возвращаемое значение из хранимой процедуры?
8. Что представляет собой триггер?
9. Перечислите отличия триггера от хранимой процедуры.
10. Какие классы триггеров предусмотрены?
11. Какие типы триггеров DML существуют?
12. Какие подклассы триггеров DML существуют и каковы их особенности?
13. Как временно отключить срабатывание триггера, а затем снова включить?
14. Что содержат виртуальные таблицы Inserted и Deleted?