Оператор INSERT

Оператор *INSERT* имеет две формы:

1) Вставка одной записи

INSERT [INTO] <имя таблицы>[(список имен полей)]

VALUES (список значений)

Предполагается взаимнооднозначное соответствие между списком имен полей и списком значений, то есть k-му полю соответствует k-е значение.

Например:

```
INSERT INTO Tovar(TovarName, IsTovar, Parent_ID)
VALUES('Макаронные изделия', 0, null)
```

Если список имён полей опущен (не рекомендуется), то подразумевается, что поля следуют в том порядке, определенном операторами *CREATE TABLE* и *ALTER TABLE*.

2) Вставка результата оператора *SELECT* в таблицу.

INSERT [INTO] <имя таблицы>[(список имен полей)]

SELECT <список выражений> FROM…

Пример: поместить в таблицу *Tovar1* список товаров, ранее покупавшихся организацией с идентификатором Org_ID=14. Предполагается, что таблица *Tovar1* уже существует.

```
INSERT INTO Tovar1(TovarName, IsTovar, Parent_ID)
SELECT TovarName, IsTovar, Parent_ID FROM Tovar
WHERE Tovar_ID in
   (SELECT Tovar_ID
    FROM SostNakl, Nakl
    WHERE SostNakl_ID=Nakl.Nakl_ID
     and Nakl.Org_ID=14)
```

Вставка значений полей обладающих свойством identity допустима только тогда, когда используется список полей и выполнена установка

set identity_insert <имя таблицы> on

Оператор *UPDATE*

Оператор *UPDATE* изменяет значения одного или более полей в существующей записи.

```
UPDATE <имя таблицы> SET
  <имя поля> = {<выражение|DEFAULT|NULL>}
  [,<имя поля> = <выражение|DEFAULT|NULL>]...
[FROM <список источников данных>]
[WHERE <логическое выражение>]
```

Пример 1:

```
UPDATE Nakl SET
  Dat='20121231',
  Numb=1234
WHERE Nakl_ID=12
```

Пример 1:

```
UPDATE Nakl SET
   Dat='20041231',
   Numb=1234
WHERE Nakl_ID=12
```

Пример 2.

Поступила накладная на приход товара на склад. Значение Nakl_ID=234. Необходимо изменить значение количества товара на складе в связи с этим новым поступлением.

```
UPDATE Tovar
SET Amount=Tovar.Amount+SostNakl.Amount
FROM Tovar, SostNakl
WHERE Tovar.Tovar_ID=SostNakl.Tovar_ID
  AND Nakl_ID=234
```

Оператор *DELETE*

Оператор *DELETE* удаляет одну или более записей из одной таблицы.

DELETE FROM <имя таблицы>

WHERE <логическое выражение>

Например:

DELETE FROM SostNakl WHERE Nakl_ID=23456

Оператор *TRUNCATE TABLE*

Оператор *TRUNCATE* удаляет все записи таблицы.

TRUNCATE TABLE <имя таблицы>

Оператор *TRUNCATE TABLE TabName* работает много быстрее, чем *DELETE FROM TabName*, особенно при больших размерах таблицы.

Представления (VIEW)

Представление (*VIEW*) это виртуальная таблица, являющаяся результатом запроса (*SELECT*) к базе данных.

Как и обычная таблица, *VIEW* состоит из строк и именованных столбцов.

Содержимое *VIEW* не хранится постоянно, записи из *VIEW* порождаются динамически тогда, когда к нему происходит обращение.

VIEW хранится в базе данных как её объект.

VIEW, однажды будучи создано, в дальнейшем может выступать как источник данных в SQL операторах наравне с таблицами (правда, с некоторыми ограничениями).

VIEW создается оператором *CREATE VIEW*. Его синтаксис:

CREATE VIEW [<имя базы данных>.][<имя владельца VIEW>.]
<имя VIEW [(список имён полей)]
[WITH <атрибут VIEW>[,атрибут VIEW]]
AS <оператор SELECT>
[WITH CHECK OPTION]

VIEW может иметь атрибуты:

- ENCRYPTION – хранимый в базе данных текст оператора *CREATE VIEW* будет зашифрован

- SCHEMABINDING – означает, что источники данных, используемые в операторе *SELECT* не могут быть удалены или модифицированы до тех пор, пока существует *VIEW* или пока его связь со схемой (SCHEMABINDING) не будет разорвана.

Пример оператора *CREATE VIEW*:

```
CREATE VIEW FirstView AS
SELECT SostNakl_ID, Nakl_ID, Tovar.Tovar_ID,
  TovarName, Tovar.Amount, SostNakl.Price,
  Tovar.IsTovar
FROM Tovar, SostNakl
WHERE Tovar.Tovar_ID=SostNakl.Tovar_ID
```

После того, как такое VIEW создано, мы получаем право обращаться к нему как к таблице, например:

```
SELECT * FROM FirstView WHERE Nakl_ID=456
```

Как и таблица, *VIEW* может быть квалифицировано именем базы данных и именем владельца.

Необязательный список имен полей содержит имена столбцов, по которым к ним следует обращаться. Если список имён полей отсутствует, то столбцы получают имена полей, возвращаемых оператором *SELECT*.

Если по отношению к *VIEW* применима какая-нибудь из операций *INSERT, UPDATE, DELETE*, то *VIEW* является модифицируемым. Это возможно в следующих случаях:

- для *VIEW* существуют триггеры *INSTEAD OF*

- операторы *INSERT* или *UPDATE* изменяют только одну из базовых таблиц, упомянутых во фразе *FROM* оператора *SELECT*, составляющего основу *VIEW*.

- Оператор *DELETE* применим к *VIEW* только если фраза *FROM* ссылается только на одну базовую таблицу

Пример 1. В результате выполнения следующего оператора будет добавлена одна запись в таблицу Tovar.

```
INSERT INTO FirstView(TovarName,IsTovar,Amount)
VALUES('Яблоки',1,32)
```

Пример 2. К названию товара для уровней классификации добавить символ '?'.

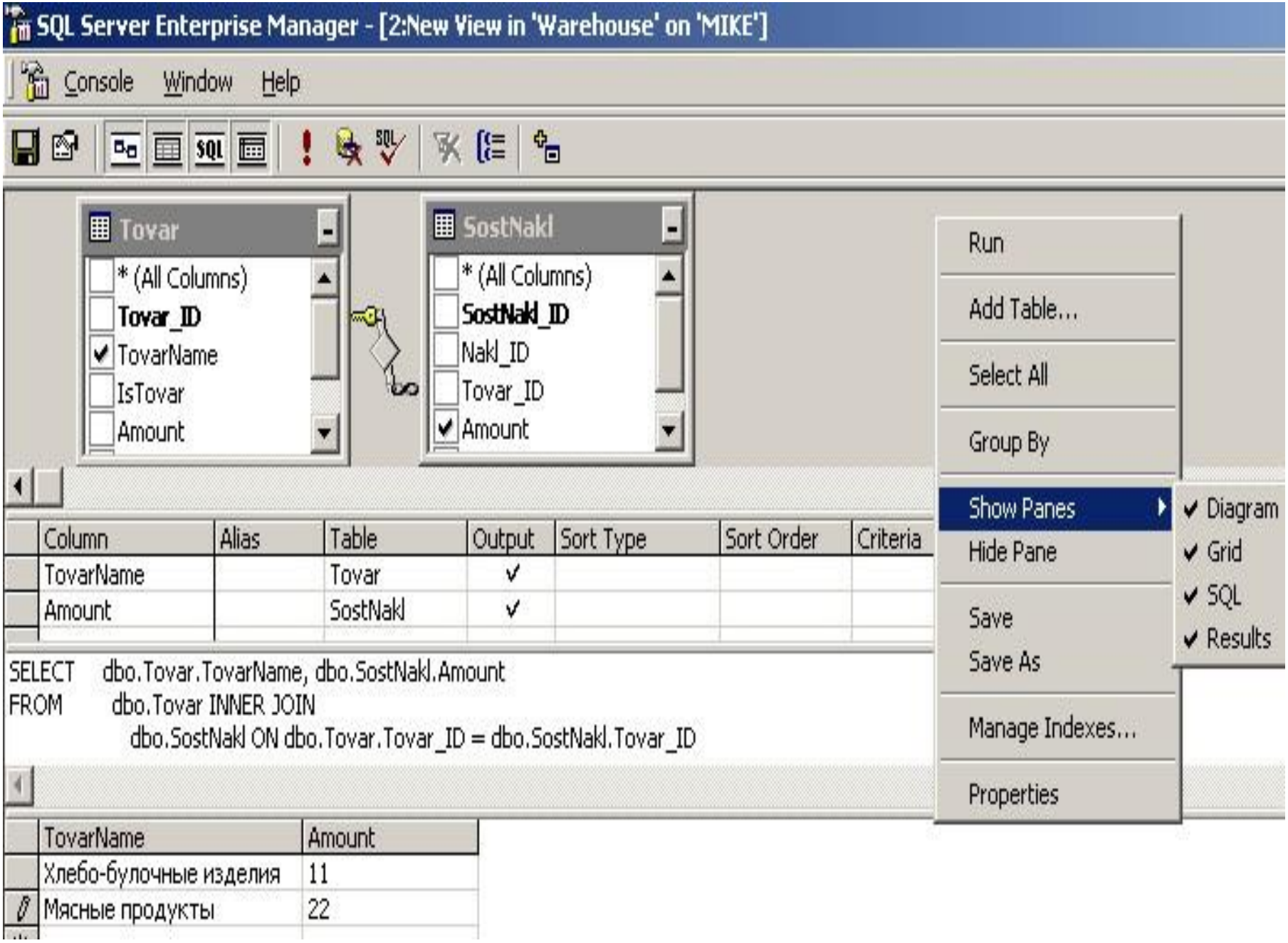
```
UPDATE FirstView SET TovarName=TovarName+'?'
  WHERE IsTovar=0
```

Операция *DELETE* для *VIEW FirstView* невозможна.

Создание *VIEW* в *Enterprise Manager*

VIEW может быть создано как результат выполнения пакета, так и с помощью построителя запросов в Enterprise Manager (EM).

Для создания нового *VIEW* нужно выделить узел дерева *View* в левой панели EM и выполнить команду *New View…* контекстного меню. В результате будет вызван диалог построителя запросов:


SQL Server Enterprise Manager - [2:New View in 'Warehouse' on 'MIKE']
Console
Window
Help
Tovar
* (All Columns)
Tovar_ID
TovarName
IsTovar
Amount
SostNakl
* (All Columns)
SostNakl_ID
Nakl_ID
Tovar_ID
Amount
Run
Add Table...
Select All
Group By
Show Panes
Hide Pane
Save
Save As
Manage Indexes...
Properties
Diagram
Grid
SQL
Results
Column
Alias
Table
Output
Sort Type
Sort Order
Criteria
TovarName
Tovar
Amount
SostNakl
SELECT dbo.Tovar.TovarName, dbo.SostNakl.Amount
FROM dbo.Tovar INNER JOIN
dbo.SostNakl ON dbo.Tovar.Tovar_ID = dbo.SostNakl.Tovar_ID
TovarName
Amount
Хлебо-булочные изделия
11
Мясные продукты
22

Запрос создается следующим образом.

на верхнюю панель, содержащую диаграмму, поместите таблицы, участвующие в запросе.

перетащите поля таблиц в поля *VIEW*.

добавьте сортировку (*ORDER BY)* и фильтрацию (фраза *WHERE*), если требуется.

Существующее *VIEW* может быть удалено оператором *DROP VIEW*:

DROP VIEW <имя VIEW>

ПОТОК УПРАВЛЕНИЯ

Программные единицы

Программной единицей (модулем) может быть пакет (batch), хранимая процедура (*stored procedure*), триггер, функция.

Пакетом называется группа из одного или более операторов Transact SQL, присылаемых клиентским приложением на SQL Server для исполнения.

Одновременно может быть прислано несколько пакетов; признаком окончания пакета является ключевое слово *GO.*

Если некоторый текст содержит единственный пакет, то наличие слова *GO* необязательно. Слово *GO* должно быть единственным в строке.

Например, приведенный ниже текст

```
DECLARE @x int, @y smallint
SET @x=3
SET @y=@x+5
SELECT @x, @y
go
DECLARE @r int
SET @r=77
SELECT 1,@r
go
```

Содержит два пакета.

Операторы записываются в свободном формате, то есть оператор может занимать более, чем одну строку, и в одной строке может находиться более чем один оператор (хотя правила хорошего стиля все же рекомендуют писать один оператор в строку и использовать отступы для вложенных операторов).

Не используется никаких символов, отделяющих один оператор от другого.

SQL Server компилирует пакет в так называемый *план исполнения* (execution plan).

Ошибка в процессе компиляции приведёт к тому, что ни один из операторов пакета не будет выполнен.

Ошибки периода исполнения, например, такие как арифметическое переполнение или нарушение ограничения могут иметь результатом одно из двух:

- большинство ошибок останавливают выполнение текущего оператора, последующие операторы не выполняются
- некоторые ошибки приводят к невыполнению только оператора, вызвавшего ошибку, остальные операторы пакета выполняются

Действие уже выполненных операторов, предшествующих тому, который вызвал ошибку, сохраняется, за исключением случая, когда группа операторов находится внутри транзакции.

Операторы

CREATE DEFAULT, CREATE PROCEDURE, CREATE RULE, CREATE TRIGGER и *CREATE VIEW*

могут быть только единственными операторами в пакете. Их нельзя объединять ни с какими другими. Нельзя в одном и том же пакете модифицировать таблицу и использовать ссылки на вновь созданные поля.

В общем случае текст программы на языке Transact SQL состоит из SQL – операторов, операторов потока управления, таких как присваивание, цикла и т.д. и комментариев, которые никак не влияют на исполняемый код. Существует два способа помещения комментариев в текст Transact SQL.

- любой текст в строке программы, который находится вслед за двумя подряд следующими дефисами и до конца строки, является комментарием -- это комментарий
- любой текст, заключенный между /* и */ является комментарием

Переменные

Локальные переменные объявляются оператором *DECLARE*.

DECLARE <имя переменной> <тип>[,<имя переменной> <тип>]…

Например: DECLARE @x int, @y varchar(30)

Имя переменной всегда начинается с символа '@'. Факторизация типа не допускается: оператор

DECLARE @x, @z int

является ошибочным.

Переменная может быть объявлена в любом месте пакета до её первого использования.

Область существования переменной – от места объявления и до конца пакета.

Помещение оператора *DECLARE* внутри операторных скобок *BEGIN, END* не приводит к локализации переменной в этом блоке.

Оператор присваивания

SET <имя переменной>=<выражение>

Например,

```
SET @x=2*@y-1
SET @z=(SELECT Tovar_Name FROM Tovar WHERE
   Tovar_ID=4)

select @z=Tovar_Name, @w=Tovar_id
FROM Tovar
WHERE Tovar_ID=4
```

Условный оператор (*IF*)

IF <логическое выражение>

<оператор1 >

[ELSE

<оператор2 >]

Если логическое выражение имеет значение *ИСТИНА*, то выполняется оператор1, иначе выполняется оператор2, если он есть.

Группа операторов может быть объединена в составной оператор (блок) с помощью операторных скобок *BEGIN* и *END*.

Пример:

```
IF @x>2
   @z=5
ELSE BEGIN
   @z=9
   @x=@z-6
END
```

Оператор перехода (*GOTO*)

GOTO <метка>

Выполняет переход на указанную метку. Метка – произвольный идентификатор.

В отличие от идентификаторов локальных переменных он не может начинаться с символа '@'.

Например:

```
GOTO mmm
.....
mmm:
SET @x=12
```

Метка, на которую передается управление не может находиться за пределами пакета, в котором находится оператор *GOTO*.

Оператор цикла

Transact SQL располагает единственным типом оператора цикла – *while*.

Синтаксис:

WHILE <логическое выражение> <оператор>

Оператор, в частности, может быть блоком. Приведенный ниже пример вычисляет

$$@S = \sum_{i=1}^{50} \frac{1}{@i^2}$$

```
DECLARE @i int, @S float
SET @i=1
SET @S=0
WHILE @i<=50 BEGIN
   SET @S=@S+1.0/(@i*@i)
   SET @i=@i+1
END
```

Выражение CASE

Выражение *CASE* имеет две формы. Синтаксис первой формы:

```
CASE <выражение0>
  WHEN <выражение11> THEN <выражение12>
  [WHEN <выражение21> THEN <выражение22>]…
  [ELSE <else - выражение>]
END
```

Вычисляется *выражение0* и его значение поочередно сравнивается с выражениями *выражение11, выражение21, …выражениеN1*.

Если будет найдено такое *выражение К1*, значение которого совпадает с *выражение0*, то в качестве значения всего выражения *CASE* будет принято значение *выражение К2*.

Если же совпадение не обнаружится, то в качестве значения результата будет принято *else – выражение*, если оно есть и *null*, если его нет.

Например, выражение

```
CASE 1
   WHEN 2 THEN 3
   WHEN 4 THEN 8
   ELSE 12
END
```

равно 12.

Синтаксис второй формы:

```
CASE
   WHEN <логическое выражение1> THEN <выражение1>
   [WHEN <логическое выражение N>] THEN <выражениеN>]
   [ELSE else - выражение ]
END
```

Выражения *логическое выражение1, логическое выражения2,* … последовательно вычисляются. Как только будет обнаружено первое из них *логическое выражениеК*, имеющее значение *ИСТИНА*, в качестве результата всего выражения *CASE* буде получено значение *выражениеК*.

Если все логические выражения имеют значение *ЛОЖЬ*, то в качестве результата будет взято значение *else – выражение*, а если фраза *ELSE* отсутствует, то *null*.

Например, при @x=1 и @y=2, выражение

```
CASE
  WHEN @x<0 THEN 66
  WHEN 2*@x=2 THEN 17
  ELSE 99
END
```

равно 17.

Курсоры

Оператор *SELECT* возвращает результирующее множество строк. В некоторых случаях может возникнуть необходимость обрабатывать эти строки по одной или блоками.

Курсор предоставляет возможность перемещаться по результирующему множеству вперед и назад, каждый раз работая с очередной записью.

Курсор представляет собой некоторый ресурс на сервере, который создается оператором *DECLARE CURSOR*. Его синтаксис:

DECLARE CURSOR.

DECLARE <имя курсора> CURSOR
[LOCAL | GLOBAL]
[FORWARD_ONLY | SCROLL]
[STATIC | KEYSET | DYNAMIC | FAST_FORWARD]
[READ_ONLY | SCROLL_LOCKS | OPTIMISTIC]
[TYPE_WARNING]
FOR <оператор SELECT>
[FOR UPDATE [OF <список полей>]]

имя курсора.

Следует различать понятия *курсор* и *переменная типа курсор*.

Имя курсора ***не должно*** начинаться с символа @.

```
declare tt cursor for select * from Org
```

Имя переменной типа курсор должно, как и для других переменных, начинаться с символа @.

```
declare @w cursor
```

В дальнейшем переменной @w можно присвоить значение:

```
set @w=tt
```

LOCAL | GLOBAL.

LOCAL (локальный) курсор существует только в пределах пакета, процедуры, триггера или функции в которых он объявлен. Такой курсор автоматически уничтожается при выходе из программной единицы (модуля).

GLOBAL (глобальный курсор) видим всюду в пределах того соединения с базой данных, в котором он создан. Глобальный курсор уничтожается автоматически при закрытии соединения.

FORWARD_ONLY | SCROLL.

Курсор *FORWARD ONLY* способен перемещаться только вперед по набору данных, каждый раз перемещаясь на следующую запись.

Таким образом, он предназначен для однократного прохода по записям набора.

Курсор *SCROLL* может двигаться вперед и назад на произвольное число записей. Операции с курсором *FORWARD_ONLY* выполняются быстрее, чем с курсором *SCROLL*.

STATIC | KEYSET | DYNAMIC | FAST_FORWARD.

STATIC (статический) курсор копирует набор данных, возвращаемый оператором *SELECT* во временную таблицу в базе данных **tempdb** и, поэтому, этот вид курсора не может модифицировать данные.

Изменения данных, происходящие в базе в результате действий других пользователей, для этого типа курсора невидимы.

KEYSET

– тип курсора, управляемый набором ключей. Множество ключей (keyset) записей, возвращаемых оператором *SELECT*, копируется в таблицу базы данных **tempdb**.

При перемещении курсора по записям, данные отыскиваются в основной базе данных по ключам из keyset.

Отсюда следует, что изменения неключевых полей видимы для курсора, новые записи, добавленные другими пользователями невидимы.

Попытка извлечь запись, удалённую другим пользователем, окажется безуспешной.

Поскольку операция *UPDATE* выполняется как удаление старой записи и вставка новой, то изменение ключевого поля записи приведет к тому, что прежнюю запись невозможно извлечь, а новая невидима.

Исключением является случай, когда *UPDATE* выполняется с использованием фразы *WHERE CURRENT OF <курсор>.*

DYNAMIC (динамический)

курсор “видит” все изменения, происходящие с данными.

FAST_FORWARD курсор имеет свойства *FORWARD_ONLY, READ_ONLY* и оптимизирован по скорости выполнения операций.

Курсор типа *FAST_FORWARD* не может иметь свойств *SCROLL* и *FOR_UPDATE.* Спецификации *FAST_FORWARD* и *FORWARD_ONLY* являются взаимоисключающими.

READ_ONLY | SCROLL_LOCKS | OPTIMISTIC.

Употребление одного из этих ключевых слов определяет намерения пользователя, создающего курсор и влияет на используемые блокировки.

- *READ_ONLY* – курсор создается только для чтения и не будет модифицировать данные, то есть операторы *UPDATE* и *DELETE* с фразой *CURRENT OF <курсор>*, употребляться не будут.

- *SCROLL_LOCKS* – записи курсора блокируются. Таким образом, операции *UPDATE* или *DELETE* для текущей записи курсора гарантировано будут успешны.

OPTIMISTIC

– предполагает использование “пассивной защиты”. Записи курсора не блокируются.

При попытке модификации записи курсора выполняется её повторное чтение и, если запись не изменилась с момента чтения её курсором, то модификация выполняется, в противном случае отвергается.

Вывод о том, что запись изменилась, делается на основании значения поля типа *timestamp* если оно есть, или на основании контрольной суммы записи, если его нет.

TYPE_WARNING

– сервер может по своему усмотрению изменить свойства курсора, если сочтет, что затребованные характеристики не могут быть удовлетворены. Например, *DYNAMIC* может быть заменено на *KEYSET*.

Указание *TYPE_WARNING* требует, чтобы пользователь был уведомлен об этом.

Оператор *DECLARE CURSOR* только создает ресурс на сервере, в котором будут размещаться данные курсора, но не читает данные из базы. Наполнение набора данных курсора происходит при выполнении оператора *OPEN*.

OPEN {{[GLOBAL] <имя курсора>}|

<имя переменной типа курсор>}

Наличие ключевого слова *GLOBAL* указывает, что происходит обращение к глобальному курсору.

Функция *@@CURSOR_ROWS* возвращает число записей в последнем открытом соединением курсоре.

После открытия курсор позиционируется в “щели” (crack) перед первой записью курсора так, что последующий оператор чтения *FETCH NEXT* прочитает первую запись.

Понятие “щель” означает позицию курсора на месте несуществующей записи – перед первой, после последней или в позиции удаленной записи.

FETCH

FETCH
[[NEXT|PRIOR|FIRST|LAST|
ABSOLUTE{<номер записи>|
<переменная, содержащая номер записи>}|
RELATIVE{<номер записи>|
<переменная, содержащая номер записи>}]
FROM{{[GLOBAL]<имя курсора>}|<переменная типа курсор>}
[INTO <список переменных>]

```
Declare  ps cursor for
Select a, b, c from….
……..
Declare @a int, @b  char(11), @ss datetime
……..
Fetch next from ps into @a, @b, @ss
```

FROM – указывает курсор из которого выполняется чтение.

INTO <список переменных> - указывает в какие переменные следует поместить поля считанной записи. Порядок и число переменных в операторе *FETCH* должны соответствовать порядку и числу полей оператора *SELECT* в объявлении курсора.

Типы данных полей и переменных должны совпадать или допускать неявное преобразование одного типа в другой.

Если курсор открыт как *FORWARD_ONLY* или *FAST_FORWARD* то единственной допустимой операцией для него является *FETCH NEXT*.

Функция *@@FETCH_STATUS* возвращает успешность выполнения последнего оператора *FETCH* в соединении:

0 – операция выполнена успешно

1 – операцию выполнить не удалось или запись находится за пределами набора

2 – читаемая запись не существует

После того, как работа с данными курсора завершена, его следует закрыть с помощью оператора *CLOSE*.

CLOSE{{[GLOBAL]<имя курсора>}|<имя переменной типа курсор>}

Оператор *CLOSE* закрывает курсор, но не освобождает ресурсы с ним связанные. Освобождение ресурсов выполняется оператором *DEALLOCATE*:

DEALLOCATE{{[GLOBAL]<имя курсора>}|<имя переменной типа курсор>}

В накладных, поступивших от организации 'НИИЧАВО' в марте 2009 г. все цены, превышающие 2000р. увеличить на 20%. Заметим, что задача легко решается и без применения курсора.

```
DECLARE @SostNakl_ID int, @Price smallmoney
DECLARE ps cursor FORWARD_ONLY STATIC
FOR
   SELECT SostNakl_ID, Price
   FROM SostNakl, Nakl, Org
   WHERE SostNakl.Nakl_ID=Nakl.Nakl_ID
      AND Nakl.Org_ID=Org.Org_ID
      AND Org.OrgName='НИИЧАВО'
      AND month(Nakl.Dat)=3 AND year(Nakl.Dat)=2009
      AND Nakl.InOut='+'
```

```
OPEN ps
FETCH NEXT FROM ps into @SostNakl_ID, @Price
WHILE @@FETCH_STATUS=0 BEGIN
   IF @Price>2000 BEGIN
      UPDATE SostNakl SET Price=1.2*Price
      WHERE SostNakl_ID=@SostNakl_ID
   END
   FETCH NEXT FROM ps into @SostNakl_ID, @Price
END
CLOSE ps
DEALLOCATE ps
```

```
-- а это правильнее
DECLARE @SostNakl_ID int, @Price smallmoney
DECLARE ps cursor FORWARD_ONLY STATIC
FOR
   SELECT SostNakl_ID, Price
   FROM SostNakl, Nakl, Org
   WHERE SostNakl.Nakl_ID=Nakl.Nakl_ID
      AND Nakl.Org_ID=Org.Org_ID
      AND Org.OrgName='НИИЧАВО'
      AND month(Nakl.Dat)=3 AND year(Nakl.Dat)=2009
      AND Nakl.InOut='+'
OPEN ps
WHILE  1=1 begin
     FETCH NEXT FROM ps into @SostNakl_ID, @Price
     if @@FETCH_STATUS!=0 break
     IF @Price>2000 BEGIN
       UPDATE SostNakl SET Price=1.2*Price WHERE SostNakl_ID=@SostNakl_ID
    END
END
CLOSE ps
DEALLOCATE ps
```

Пример 2 (фраза WHERE CURRENT OF..)

Все телефоны организаций, начинающиеся на '63' заменить на ’263…’

```
DECLARE @Phone varchar(20)
DECLARE ps CURSOR FORWARD_ONLY FOR
   SELECT Phone FROM Org
OPEN ps
FETCH NEXT FROM ps into @Phone
WHILE @@fetch_status=0 BEGIN
   if @Phone like '63%' BEGIN
      UPDATE Org SET Phone='263'+substring(@Phone,3,20)
    WHERE CURRENT OF ps
   END
   FETCH NEXT FROM ps into @Phone
END
CLOSE ps
DEALLOCATE ps
```

TRY…CATCH

Конструкция TRY…CATCH обеспечивает перехват и обработку исключительных ситуаций, подобно тому, как это делается в C++.

Группа операторов Transact SQL помещается в блок TRY. Если внутри этого блока в период исполнения происходит ошибка, то управление будет передано в блок CATCH.

```
BEGIN TRY
   --SQL - операторы
END TRY
BEGIN CATCH
   --SQL - операторы
END CATCH
[ ; ]
```