Курсоры T-SQL

Операции в реляционной базе данных выполняются над множеством строк. Набор строк, возвращаемый инструкцией SELECT, содержит все строки, которые удовлетворяют условиям, указанным в предложении WHERE. Такой полный набор строк, возвращаемых инструкцией, называется результирующим набором. Приложения, особенно интерактивные, не всегда эффективно работают с результирующим набором как с единым целым. Им нужен механизм, позволяющий обрабатывать одну строку или небольшое их число за один раз. Курсоры предоставляют такой механизм.

Microsoft SQL Server поддерживает три способа реализации курсоров:

1. **Курсоры T-SQL**. Основываются на синтаксисе DECLARE CURSOR и используются главным образом в сценариях T-SQL, хранимых процедурах и триггерах. Курсоры T-SQL реализуются на сервере и управляются инструкциями T-SQL, направляемыми клиентом серверу. Они также могут содержаться в пакетах, хранимых процедурах или триггерах.
2. **Серверные курсоры интерфейса прикладного программирования (API)** . Поддерживают функции курсоров API в OLE DB и ODBC. Курсоры API реализуются на сервере. Всякий раз, когда клиентское приложение вызывает функцию курсора API, поставщик OLE DB или драйвер ODBC для собственного клиента SQL передает требование на сервер для выполнения действия в отношении серверного курсора API.
3. **Клиентские курсоры**. Реализуются внутренне драйвером ODBC для собственного клиента SQL и DLL, реализующим ADO API. Клиентские курсоры реализуются посредством кэширования всех строк результирующего набора на клиенте. Всякий раз, когда клиентское приложение вызывает функцию курсора API, драйвер ODBC для собственного клиента SQL или ADO DLL выполняет операцию курсора на строках результирующего набора, кэшированных на клиенте.

Курсоры T-SQL и серверные курсоры API собирательно называются серверными курсорами.

Методика использования курсора языка Transact-SQL

Методика использования курсора языка T-SQL такова:

1. С помощью инструкции DECLARE необходимо объявить переменные T-SQL, которые будут содержать данные, возвращенные курсором. Для каждого столбца результирующего набора надо объявить по одной переменной. Переменные должны быть достаточно большого размера для хранения значений, возвращаемых в столбце и имеющих тип данных, к которому могут быть неявно преобразованы данные столбца.
2. С помощью инструкцией DECLARE CURSOR необходимо связать курсор T-SQL с инструкцией SELECT. Инструкция DECLARE CURSOR определяет также характеристики курсора, например имя курсора и тип курсора (read-only или forward-only).
3. С помощью инструкции OPEN выполнить инструкцию SELECT и заполнить курсор.
4. С помощью инструкции FETCH INTO организовать перемещение по курсору для выборки отдельных строк; значение каждого столбца отдельной строки заносится в указанную переменную. После этого другие инструкции T-SQL могут ссылаться на эти переменные для доступа к выбранным значениям данных. Курсоры T-SQL не поддерживают выборку группы строк.
5. После завершения работы с курсором необходимо применить инструкцию CLOSE. Закрытие курсора освобождает некоторые ресурсы, например результирующий набор курсора и его блокировки на текущей строке, однако структура курсора будет доступна для обработки, если снова выполнить инструкцию OPEN. Поскольку курсор все еще существует на этом этапе, повторно использовать его имя нельзя.
6. Наконец инструкция DEALLOCATE полностью освобождает все ресурсы, выделенные курсору, в том числе имя курсора. После освобождения курсора его необходимо строить заново с помощью инструкции DECLARE CURSOR.

Пример

```
USE dbSPJ
GO
-- Объявляем переменную
DECLARE @TableName varchar(255)
-- Объявляем курсор
DECLARE TableCursor CURSOR FOR
   SELECT TABLE_NAME FROM INFORMATION_SCHEMA.TABLES
      WHERE TABLE_TYPE = 'BASE TABLE'
-- Открываем курсор и выполняем извлечение первой записи
OPEN TableCursor
FETCH NEXT FROM TableCursor INTO @TableName
-- Проходим в цикле все записи из множества
WHILE @@FETCH_STATUS = 0
```

```
BEGIN
   PRINT @TableName
   FETCH NEXT FROM TableCursor INTO @TableName
END
-- Убираем за собой «хвосты»
CLOSE TableCursor
DEALLOCATE TableCursor
```

Рекомендация. Примените этот курсор к своей базе данных и проанализируйте результаты его исполнения.

В SQL Server поддерживаются четыре типа серверных курсоров:

- *Статические курсоры* (STATIC). Создается временная копия данных для использования курсором. Все запросы к курсору обращаются к указанной временной таблице в базе данных tempdb, поэтому изменения базовых таблиц не влияют на данные, возвращаемые выборками для данного курсора, а сам курсор не позволяет производить изменения.
- *Динамические курсоры* (DYNAMIC). Отображают все изменения данных, сделанные в строках результирующего набора при просмотре этого курсора. Значения данных, порядок, а также членство строк в каждой выборке могут меняться. Параметр выборки ABSOLUTE динамическими курсорами не поддерживается.
- *Курсоры, управляемые набором ключей* (KEYSET). Членство или порядок строк в курсоре не изменяются после его открытия. Набор ключей, однозначно определяющих строки, встроен в таблицу в базе данных tempdb с именем keyset.
- *Быстрые последовательные курсоры* (FAST_FORWARD). Параметр FAST_FORWARD указывает курсор FORWARD_ONLY, READ_ONLY, для которого включена оптимизация производительности. Параметр FAST_FORWARD не может указываться вместе с параметрами SCROLL или FOR_UPDATE.

Статическими курсорами обнаруживаются лишь некоторые изменения или не обнаруживаются вовсе, но при этом в процессе прокрутки такие курсоры потребляют сравнительно мало ресурсов. Динамические курсоры обнаруживают все изменения, но потребляют больше ресурсов при прокрутке. Управляемые набором ключей курсоры имеют промежуточные свойства, обнаруживая большинство изменений, но потребляя меньше ресурсов, чем динамические курсоры.

По области видимости имени курсора различают:

- *Локальные курсоры* (LOCAL). Область курсора локальна по отношению к пакету, хранимой процедуре или триггеру, в которых этот курсор был создан. Курсор неявно освобождается после завершения выполнения пакета, хранимой процедуры или триггера, за исключением случая, когда курсор был передан параметру OUTPUT. В этом случае курсор освобождается при освобождении всех ссылающихся на него переменных или при выходе из области видимости.
- *Глобальные курсоры* (GLOBAL). Область курсора является глобальной по отношению к соединению. Имя курсора может использоваться любой хранимой процедурой или пакетом, которые выполняются соединением. Курсор неявно освобождается только в случае разрыва соединения. Глобальность курсора позволяет создавать его в рамках одной хранимой процедуры, а вызывать его из совершенно другой процедуры, причем передавать его в эту процедуру необязательно.

По способу перемещения по курсору различают:

- *Последовательные курсоры* (FORWARD_ONLY). Курсор может просматриваться только от первой строки к последней. Поддерживается только параметр выборки FETCH NEXT. Если параметр FORWARD_ONLY указан без ключевых слов STATIC, KEYSET или DYNAMIC, то курсор работает как DYNAMIC. Если не указан ни один из параметров FORWARD_ONLY или SCROLL, а также не указано ни одно из ключевых слов STATIC, KEYSET или DYNAMIC, то по умолчанию задается параметр FORWARD_ONLY. Курсоры STATIC, KEYSET и DYNAMIC имеют значение по умолчанию SCROLL.
- *Курсоры прокрутки* (SCROLL). Перемещение осуществляется по группе записей как вперед, так и назад. В этом случае доступны все параметры выборки (FIRST, LAST, PRIOR, NEXT, RELATIVE, ABSOLUTE). Параметр SCROLL не может указываться вместе с параметром для FAST_FORWARD.

По способу распараллеливания курсоров различают:

- READ_ONLY. Содержимое курсора можно только считывать.
- SCROLL_LOCKS. При редактировании данной записи вами никто другой вносить в нее изменения не может. Такую блокировку прокрутки иногда еще называют «пессимистической» блокировкой. При блокировке прокрутки важным фактором является продолжительность действия блокировки. Если курсор не входит в состав некоторой транзакции, то блокировка распространяется только на текущую запись в курсоре. В противном случае все зависит от того, какой используется уровень изоляции транзакций.
- OPTIMISTIC. Означает отсутствие каких бы то ни было блокировок. «Оптимистическая» блокировка предполагает, что даже во время выполнения вами редактирования данных, другие пользователи смогут к ним обращаться. Если

кто-то все-таки попытается выполнить модификацию данных одновременно с вами, генерируется ошибка 16394. В этом случае вам придется повторно выполнить доставку данных из курсора или же осуществить полный откат транзакции, а затем попытаться выполнить ее еще раз.

Для выявления ситуаций с преобразованием типа курсора используется опция TYPE_WARNING, которая указывает, что клиенту посылается предупреждающее сообщение при неявном преобразовании типа курсора из запрошенного типа в другой тип.

По умолчанию курсор, разрешающий изменение данных, позволяет выполнять модификацию любых столбцов в своем составе. Опция FOR UPDATE указывает столбцы курсора, которые можно изменять. Все остальные столбцы становятся доступными только для чтения. Если опция FOR UPDATE используется без списка столбцов, то обновление возможно для всех столбцов, за исключением случая, когда был указан параметр параллелизма READ_ONLY.

На содержательном уровне синтаксис инструкции DECLARE CURSOR будет иметь вид:

DECLARE *имя-курсора* CURSOR
[*область-видимости-имени-курсора*]
[*возможность-перемещения-по-курсору*]
[*типы-курсоров*]
[*опции-распараллеливания-курсоров*]
[*выявление-ситуаций-с-преобразованием-типа-курсора*]
FOR *инструкция_select*
[*опция-FOR-UPDATE*]

Формальный синтаксис инструкции DECLARE CURSOR имеет вид:

DECLARE *имя-курсора* CURSOR
[LOCAL | GLOBAL]
[FORWARD_ONLY | SCROLL]
[STATIC | KEYSET | DYNAMIC | FAST_FORWARD]
[READ_ONLY | SCROLL_LOCKS | OPTIMISTIC]
[TYPE_WARNING]
FOR *инструкция_select*
[FOR UPDATE [OF *список-имен_столбцов*]]

Использование переменной типа cursor

Microsoft SQL Server поддерживает переменные с типом данных CURSOR. Курсор может быть связан с переменной типа CURSOR двумя способами, например:

```
1.    DECLARE @MyVariable CURSOR
      DECLARE MyCursor CURSOR FOR
      SELECT LastName FROM AdventureWorks.Person.Contact
      SET @MyVariable = MyCursor;

2.    DECLARE @MyVariable CURSOR
      SET @MyVariable = CURSOR SCROLL KEYSET FOR
      SELECT LastName FROM AdventureWorks.Person.Contact;
```

После связи курсора с переменной типа CURSOR эта переменная может использоваться в инструкциях курсора языка T-SQL вместо имени курсора. Кроме того, выходным параметрам хранимой процедуры можно назначить тип данных CURSOR и связать их с курсором. Это позволяет управлять локальными курсорами из хранимых процедур.

Перемещение внутри курсора (прокрутка курсора)

Основой всех операций прокрутки курсора является ключевое слово FETCH. Оно может использоваться для перемещения по курсору в обоих направлениях, в том числе и для перехода к заданной позиции. В качестве аргументов оператора FETCH могут выступать:

- NEXT – возвращает строку результата сразу же за текущей строкой и перемещает указатель текущей строки на возвращенную строку. Если инструкция FETCH NEXT выполняет первую выборку в отношении курсора, она возвращает первую строку в результирующем наборе. NEXT является параметром по умолчанию выборки из курсора.

- PRIOR – возвращает строку результата, находящуюся непосредственно перед текущей строкой и перемещает указатель текущей строки на возвращенную строку. Если инструкция FETCH PRIOR выполняет первую выборку из курсора, не возвращается никакая строка и положение курсора остается перед первой строкой.
- FIRST – возвращает первую строку в курсоре и делает ее текущей.
- LAST – возвращает последнюю строку в курсоре, и делает ее текущей.
- ABSOLUTE { n | @nvar}. Если n или @nvar имеют положительное значение, возвращает строку, стоящую дальше на n строк от передней границы курсора, и делает возвращенную строку новой текущей строкой. Если n или @nvar имеют отрицательное значение, возвращает строку, отстоящую на n строк от задней границы курсора, и делает возвращенную строку новой текущей строкой. Если n или @nvar равны 0, не возвращается никакая строка. n должно быть целым числом, а @nvar должна иметь тип данных smallint, tinyint или int.
- RELATIVE { n | @nvar}. Если n или @nvar имеют положительное значение, возвращает строку, отстоящую на n строк вперед от текущей строки, и делает возвращенную строку новой текущей строкой. Если n или @nvar имеют отрицательное значение, возвращает строку, отстоящую на n строк назад от текущей строки, и делает возвращенную строку новой текущей строкой. Если n или @nvar равны 0, возвращает текущую строку. Если при первой выборке из курсора инструкция FETCH RELATIVE указывается с отрицательными или равными нулю параметрами n или @nvar, то никакая строка не возвращается. n должно быть целым числом, а @nvar должна иметь тип данных smallint, tinyint или int.

Опциями оператора FETCH являются:
- GLOBAL – указывает, что параметр *имя-курсора* ссылается на глобальный курсор.
- INTO – позволяет поместить данные из столбцов выборки в локальные переменные. Каждая переменная из списка, слева направо, связывается с соответствующим столбцом в результирующем наборе курсора. Тип данных каждой переменной должен соответствовать типу данных соответствующего столбца результирующего набора, или должна обеспечиваться поддержка неявного преобразования в тип данных этого столбца. Количество переменных должно совпадать с количеством столбцов в списке выбора курсора.

Синтаксис инструкции FETCH имеет вид:

FETCH
[[NEXT | PRIOR | FIRST | LAST
| ABSOLUTE { *n* | *@nvar* }
| RELATIVE { *n* | *@nvar* }
]
FROM
]
{ { [GLOBAL] *имя-объявленного-курсора* } | *@имя-переменной-курсора* }
[INTO *список-имен-переменных*]

После выполнения каждой инструкции FETCH функция @@FETCH_STATUS обновляется, отражая состояние последней выборки. Функция @@FETCH_STATUS показывает такие состояния, как выборка за пределами первой или последней строк курсора. Функция @@FETCH_STATUS глобальна для соединения и обновляется после любой выборки в любом курсоре, открытом во время соединения. Если состояние нужно посмотреть позже, то перед выполнением следующей инструкции в пределах соединения необходимо сохранить значение функции @@FETCH_STATUS в пользовательской переменной. Даже если следующей инструкцией будет не FETCH, а INSERT, UPDATE или DELETE, сработает триггер, содержащий инструкции FETCH, что приведет к обновлению значения функции @@FETCH_STATUS.

Примеры, демонстрирующие различие между STATIC и DYNAMIC курсорами

```
USE Northwind
-- Создаем таблицу специально для данного примера
SELECT OrderID, CustomerID
INTO CursorTable
FROM Orders
WHERE OrderID BETWEEN 10701 AND 10705
-- Объявляем курсор
DECLARE CursorTest CURSOR
GLOBAL
SCROLL
STATIC
FOR
SELECT OrderID, CustomerID
FROM CursorTable
```

```
-- Объявляем переменные для хранения
DECLARE @OrderID      int
DECLARE @CustomerID   varchar(5)

-- Откроем курсор и запросим первую запись
OPEN CursorTest
FETCH NEXT FROM CursorTest INTO @OrderID, @CustomerID
-- Обработаем в цикле все записи курсора
WHILE @@FETCH_STATUS=0
BEGIN
     PRINT CONVERT(varchar(5),@OrderID) + '    ' +  @CustomerID
     FETCH NEXT FROM CursorTest INTO @OrderID, @CustomerID
END
-- Внесем изменеия. Далее мы увидем, что эти изменения не будут отражены в  курсоре
UPDATE CursorTable
      SET CustomerID = 'XXXXX'
      WHERE OrderID = 10703
-- Проверяем, что изменения были в действительности внесены в таблицу
SELECT OrderID, CustomerID
FROM CursorTable
-- Вернемся к началу списка
FETCH FIRST FROM CursorTest INTO @OrderID, @CustomerID
-- Еще раз пройдемся по списку записей
WHILE @@FETCH_STATUS=0
BEGIN
     PRINT CONVERT(varchar(5),@OrderID) + '    ' +  @CustomerID
     FETCH NEXT FROM CursorTest INTO @OrderID, @CustomerID
END
-- Теперь уберем за собой «хвосты»
CLOSE CursorTest
DEALLOCATE CursorTest
DROP TABLE CursorTable
```

```
Сообщения
(строк обработано: 5)
10701    HUNGO
10702    ALFKI
10703    FOLKO
10704    QUEEN
10705    HILAA

(строк обработано: 1)

(строк обработано: 5)
10701    HUNGO
10702    ALFKI
10703    FOLKO
10704    QUEEN
10705    HILAA

Результаты

10701 HUNGO
10702 ALFKI
10703 XXXXX
10704 QUEEN
10705 HILAA
```

```
USE Northwind
-- Создадим таблицу специально для данного примера
SELECT OrderID, CustomerID
INTO CursorTable
FROM Orders
WHERE OrderID BETWEEN 10701 AND 10705
```

```
-- Создадим уникальный индекс в виде первичного ключа
ALTER TABLE CursorTable
      ADD CONSTRAINT PKCursor
      PRIMARY KEY (OrderID)
/* Свойство IDENTITY автоматически устанавливается при выполнении инструкции SELECT INTO, но
поскольку в качестве уникального значения мы хотим использовать OrderID, нужно явно включить
опцию IDENTITY_INSERT, для того чтобы переопределить назначаемое по умолчанию identity-
значение. */
SET IDENTITY_INSERT CursorTable ON
-- Объявим курсор
DECLARE CursorTest CURSOR
GLOBAL               -- So we can manipulate it outside the batch
SCROLL               -- So we can scroll back and see if the changes are there
DYNAMIC              -- This is what we're testing this time
FOR
SELECT OrderID, CustomerID
FROM CursorTable
-- Объявим переменные для хранения
DECLARE @OrderID      int
DECLARE @CustomerID   varchar(5)
-- Откроем курсор и запросим первую запись
OPEN CursorTest
FETCH NEXT FROM CursorTest INTO @OrderID, @CustomerID
-- Обработаем в цикле все записи курсора
WHILE @@FETCH_STATUS=0
BEGIN
     PRINT CONVERT(varchar(5),@OrderID) + '    ' +  @CustomerID
     FETCH NEXT FROM CursorTest INTO @OrderID, @CustomerID
END
-- Внесем в таблицу изменения. We'll see that it does affect the cursor this time.
UPDATE CursorTable
       SET CustomerID = 'XXXXX'
       WHERE OrderID = 10703
-- Удалим запись, чтобы проверить, отразится ли это на курсоре
DELETE CursorTable
       WHERE OrderID = 10704
-- Вставим новую запись. Но теперь курсор ее увидит
INSERT INTO CursorTable
       (OrderID, CustomerID)
VALUES
       (99999, 'IIIII')
-- Проверим, что изменения были в действительности внесены в таблицу
SELECT OrderID, CustomerID
FROM CursorTable
-- Вернемся к началу списка
FETCH FIRST FROM CursorTest INTO @OrderID, @CustomerID
-- Еще раз пройдемся по списку записей.
WHILE @@FETCH_STATUS != -1
BEGIN
     IF @@FETCH_STATUS = -2
     BEGIN
          PRINT '  MISSING! It probably was deleted.'
     END
     ELSE
     BEGIN
          PRINT CONVERT(varchar(5),@OrderID) + '    ' +  @CustomerID
     END
     FETCH NEXT FROM CursorTest INTO @OrderID, @CustomerID
END
-- Теперь уберем за собой «хвосты»
CLOSE CursorTest
DEALLOCATE CursorTest
DROP TABLE CursorTable
```

```
Сообщения

(строк обработано: 5)
10701   HUNGO
10702   ALFKI
10703   FOLKO
10704   QUEEN
10705   HILAA

(строк обработано: 1)

(строк обработано: 1)

(строк обработано: 1)

(строк обработано: 5)
10701   HUNGO
10702   ALFKI
10703   XXXXX
10705   HILAA
99999   IIIII

Результаты

10701 HUNGO
10702 ALFKI
10703 XXXXX
10705 HILAA
99999 IIIII
```

Наблюдение за активностью курсора T-SQL

Для получения списка курсоров, видимых в текущем соединении, используется системная хранимая процедура sp_cursor_list, а для определения характеристик курсора используются процедуры sp_describe_cursor, sp_describe_cursor_columns и sp_describe_cursor_tables.

После открытия курсора столбец cursor_rows, возвращенный процедурами sp_cursor_list или sp_describe_cursor, показывают количество строк в курсоре.

После выполнения каждой инструкции FETCH сведения о состоянии последней выборки имеются в столбце fetch_status, возвращенном процедурой sp_describe_cursor. Столбец fetch_status, возвращенный процедурой sp_describe_cursor, относится только к указанному курсору и не зависит от инструкций FETCH, ссылающихся на другие курсоры. Однако процедура sp_describe_cursor зависит от инструкций FETCH, ссылающихся на тот же курсор, поэтому при ее использовании следует соблюдать осторожность.

Изменения данных непосредственно в курсоре

Для того чтобы выполнить обновление, необходимо воспользоваться специальным синтаксисом инструкций UPDATE и DELETE, в которых предложение WHERE имеет вид:

WHERE CURRENT OF *имя_курсора*

Пример

```
USE Northwind
-- Создадим таблицу специально для данного примера
SELECT OrderID, CustomerID
INTO CursorTable
FROM Orders
WHERE OrderID BETWEEN 10701 AND 10705
-- Создадим уникальный индекс в виде первичного ключа
ALTER TABLE CursorTable
      ADD CONSTRAINT PKCursor
      PRIMARY KEY (OrderID)
```

```
/* Свойство IDENTITY автоматически устанавливается при выполнении инструкции SELECT INTO, но
поскольку в качестве уникального значения мы хотим использовать OrderID, нужно явно включить
опцию IDENTITY_INSERT, для того чтобы переопределить назначаемое по умолчанию identity-
значение. */
SET IDENTITY_INSERT CursorTable ON
-- Объявим курсор
DECLARE CursorTest CURSOR
SCROLL           -- Чтобы перемещаться по курсору и видеть изменения
KEYSET
FOR
SELECT OrderID, CustomerID
FROM CursorTable
-- Объявим переменные для хранения
DECLARE @OrderID      int
DECLARE @CustomerID   varchar(5)
-- Откроем курсор и запросим первую запись
OPEN CursorTest
FETCH NEXT FROM CursorTest INTO @OrderID, @CustomerID
-- Обработаем в цикле все записи курсора
WHILE @@FETCH_STATUS=0
BEGIN
     IF (@OrderID % 2 = 0)     -- Обновим четные строки
     BEGIN
          -- Внесем изменения при помощи курсора
          UPDATE CursorTable
                 SET CustomerID = 'EVEN'
                 WHERE CURRENT OF CursorTest
     END
     ELSE                       -- Удалим нечетные строки
     BEGIN
          -- Удалим текущую строку курсора
          DELETE CursorTable
                 WHERE CURRENT OF CursorTest
     END
     FETCH NEXT FROM CursorTest INTO @OrderID, @CustomerID
END
-- Поскольку в этом курсоре разрешена прокрутка в любом направлении,
-- то можно опять перейти к началу результирующего множества
FETCH FIRST FROM CursorTest INTO @OrderID, @CustomerID
-- И вновь в цикле обработаем все строки
WHILE @@FETCH_STATUS != -1
BEGIN
     IF @@FETCH_STATUS = -2
     BEGIN
          PRINT '  MISSING! It probably was deleted.'
     END
     ELSE
     BEGIN
          PRINT CONVERT(varchar(5),@OrderID) + '    ' +  @CustomerID
     END
     FETCH NEXT FROM CursorTest INTO @OrderID, @CustomerID
END
-- Самое время удалить за собой все «хвосты»
CLOSE CursorTest
DEALLOCATE CursorTest
DROP TABLE CursorTable

Сообщения
(строк обработано: 5)
(строк обработано: 1)
(строк обработано: 1)
(строк обработано: 1)
(строк обработано: 1)
```

```
(строк обработано: 1)
  MISSING! It probably was deleted.
10702   EVEN
  MISSING! It probably was deleted.
10704   EVEN
  MISSING! It probably was deleted.
```