**Работа с курсорами в MySQL**

**Оператор создания курсора**

Курсор объявляется оператором DECLARE.

Оператор DECLARE CURSOR определяет выполняемый запрос, задает имя курсора и связывает результаты запроса с заданным курсором. Этот оператор не является исполняемым для запроса, он только определяет структуру будущего множества записей и связывает ее с уникальным именем курсора. Этот оператор подобен операторам описания данных в языках программирования.

DECLARE <имя\_курсора> CURSOR

[LOCAL | GLOBAL]

[FORWARD\_ONLY | SCROLL]

[STATIC | KEYSET | DYNAMIC | FAST\_FORWARD]

[READ\_ONLY | SCROLL\_LOCKS | OPTIMISTIC]

[TYPE\_WARNING]

FOR <оператор выбора SELECT>

[FOR UPDATE [OF <имя\_столбца 1> [,... n]]]

* Опция LOCAL говорит о том, что курсор является локальным для того блока (пакета, хранимой процедуры или триггера), в котором он определен. По окончании работы этого блока курсор будет неявно уничтожен.
* Опция GLOBAL является противоположностью опции LOCAL. Она говорит, что объявленный курсор продолжает существовать и по окончании блока, в котором он был создан, если только он не был удален явно. По умолчанию используются установки сервера для конкретной БД.
* SCROLL определяет, что допустимы любые режимы перемещения по курсору (FIRST, LAST, PRIOR, NEXT, RELATIVE, ABSOLUTE) в операторе FETCH. Если не указано ключевое слово SCROLL, то считается доступным только стандартное перемещение вперед — спецификация NEXT в операторе FETCH.
* FORWARD\_ONLY — это опция является противоположностью опции SCROLL, описанной выше, и говорит, что для перемещения по курсору можно использовать только опцию NEXT команды FETCH.
* STATIC определяет режим создания набора строк, соответствующего определяемому курсору. Этот тип курсора соответствует некоторому мгновенному слепку с базы данных. При открытии такого курсора все данные, на которых он строится, копируются во временную таблицу в базе данных tempdb. Таким образом, все изменения в исходных таблицах, произведенные после открытия курсора другими пользователями, в нем не видны. Такой набор данных нечувствителен ко всем изменениям, которые производятся другими пользователями в исходных таблицах. Поскольку данные копируются во временную таблицу, такой курсор не позволяет делать изменений в данных, на которых он строится.
* При использовании опции KEYSET в момент открытия курсора во временной таблице будет создан набор ключей набора строк, получаемого при помощи оператора выбора SELECT. В дальнейшем при перемещении по курсору будет происходить перемещение по ключам из временной таблицы, а остальные поля будут браться из исходных полей. Такой подход позволит лучше отслеживать изменения, сделанные с момента открытия курсора. Однако отследить все изменения он не позволяет.
* При указании опции DYNAMIC в курсоре будут отображаться все изменения, сделанные с момента открытия курсора. При ее использовании недоступна опция ABSOLUTE команды FETCH.
* FAST\_FORWARD — по сути, обозначает совокупность опций FORWARD\_ONLY и READ\_ONLY с добавлением оптимизации производительности. Эта опция несовместима с опциями SCROLL, FOR\_UPDATE и FORWARD\_ONLY.
* READ\_ONLY — изменения и обновления исходных таблиц не будут выполняться с использованием данного курсора. Курсор с данной спецификацией может быть самым быстрым в обработке.
* SCROLL\_LOCKS — гарантирует возможность удаления и изменения строки, на которую в данный момент указывает курсор. При указании этой опции после позиционирования на какой-либо строке данная строка блокируется, и другие пользователи не могут с ней ничего делать с того момента, как строка была прочитана в курсор, и до тех пор пока она не будет освобождена (при переходе на другую строку или закрытии курсора).
* OPTIMISTIC — при указании этой опции при чтении строки в курсор ее блокировка не производится.
* TYPE\_WARNING — при задании этой опции пользователю могут быть выданы специальные сообщения о некоторых нарушениях.
* FOR UPDATE [OF <имя столбца 1> [, ...<имя столбца n>]] — здесь мы задаем перечень столбцов, в которых допустимы изменения в процессе нашей работы с курсором. Такое ограничение упростит и ускорит работу СУБД.

Пример:

Опишем курсор, который содержит список должников нашей библиотеки. Мы не собираемся использовать данный курсор для удаления или обновления строк, поэтому определим его как курсор только для чтения:

DECLARE Deptor Cursor Local for

SELECT DISTINCT Num\_reader FROM Exemplar

WHERE Yes\_No ='0' AND Data\_RETURN < Getdate()

FOR READ\_ONLY

**Оператор открытия курсора**

Оператор OPEN дает команду СУБД выполнить описанный запрос, создать виртуальный набор строк, который соответствует заданному запросу. Оператор OPEN устанавливает указатель записей (курсор) перед первой строкой виртуального набора строк результата.

Оператор открытия курсора имеет следующий синтаксис:

OPEN {{[GLOBAL] <имя\_курсора>} | | <имя переменной типа курсор>}

Именно оператор открытия курсора инициирует выполнение базового запроса, соответствующего описанию курсора, заданному в операторе DECLARE … CURSOR.

SQL Server возвращает код завершения операции в специальной системной переменной @@Error. При неудачном выполнении операции открытия курсора СУБД возвращает ненулевое значение @@Error.

После того как курсор открыт, в системной переменной @@Cursor\_Rows будет содержаться количество строк в открытом курсоре.

**Оператор перемещения курсора**

После открытия указатель текущей строки установлен перед первой строкой курсора. Стандартно оператор FETCH перемешает указатель текущей строки на следующую строку и присваивает базовым переменным значение столбцов, соответствующее текущей строке.

Оператор FETCH продвигает указатель записей на следующую позицию в виртуальном наборе записей. В большинстве коммерческих СУБД оператор перемещения FETCH реализует более широкие функции перемещения. Он позволяет перемещать указатель на произвольную запись, вперед и назад; допускает как абсолютную, так и относительную адресацию; позволяет установить курсор на первую или последнюю запись виртуального набора.

Оператор FETCH имеет следующий синтаксис:

FETCH [NEXT | PRIOR | FIRST | LAST |

ABSOLUTE {n | <имя\_переменной>} | RELATIVE {n |<имя\_переменной>}]

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

INTO <список базовых переменных>

* Параметр NEXT задает выбор следующей строки после текущей из базового набора строк, связанного с курсором.
* Параметр PRIOR задает перемещение на предыдущую строку по отношению к текущей.
* Параметр FIRST задает перемещение на первую строку набора, параметр LAST — на последнюю строку набора.
* Параметр ABSOLUTE задает номер конкретной строки, а параметр RELATIVE задает относительный адрес. При абсолютной адресации отрицательное число будет трактоваться как n-ная строка с конца набора данных. Если в качестве параметра указан ноль, данные вообще не будут возвращены. При относительной адресации положительное число сдвигает указатель вперед (вниз) от текущей записи, отрицательное число сдвигает назад (вверх) от текущей записи. В случае если при первом вызове число отрицательное или равно нулю, то никаких данных возвращено не будет.

После каждого выполнения оператора FETCH изменятся состояние системной переменной @@Fetch\_status. Эта переменная показывает результат работы оператора. Переменная может иметь три состояния (табл. 9).

Таблица 9. Значения глобальной переменной @@Fetch\_status  
при работе с курсором

|  |  |
| --- | --- |
| Значение | Расшифровка значения |
| 0 | Оператор выполнен успешно. В переменные занесены значения из очередной строки набора данных |
| –1 | Строки в наборе данных закончились, либо затребованная строка находится вне его пределов, либо произошла ошибка при выполнении оператора |
| –2 | Строки в наборе данных нет. Это значение может получиться в том случае, если была указана опция KEYSET курсора. В этом случае при открытии во временную таблицу копируются ключи набора данных, а все остальные значения получаются путем обращения к реальным таблицам по известным ключам при выполнении оператора Fetch. Если после открытия курсора строка, ключ которой скопирован во временную таблицу, была удалена или если ее ключ был изменен, то найти строку по заданному ключу будет невозможно и в переменную @@Fetch\_status будет записан код –2 |

Следует помнить, что переменная @@Fetch\_Cursor является глобальной, то есть если существует несколько вложенных курсоров, то значение @@Fetch\_Cursor будет относиться к тому из них, для которого выполнен оператор FETCH.

**Оператор закрытия курсора**

Оператор CLOSE закрывает курсор и прекращает доступ к виртуальному набору записей. Он фактически ликвидирует связь между курсором и результатом выполнения базового запроса. Однако в коммерческих СУБД оператор CLOSE не всегда означает уничтожение виртуального набора записей. Мы коснемся этого вопроса далее, когда будем рассматривать работу с курсором в MS SQL Server 2000. При использовании оператора CLOSE структура данных остается доступной для повторного открытия.

Оператор закрытия курсора имеет простой синтаксис, который выглядит следующим образом:

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

Оператор закрытия курсора освобождает память, используемую под набор данных курсора, и снимает блокировки, наложенные на данные. При этом сама структура курсора, то есть его описание, остается доступной, так что курсор можно открыть вновь, не описывая его.

**Оператор уничтожения курсора**

Наряду с оператором закрытия курсора используется оператор его уничтожения:

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

Оператор DEALLOCATE удаляет взаимосвязь между курсором и именем курсора или переменной курсора. Если имя или переменная являются последними, ссылающимися на курсор, все ресурсы, выделенные курсору, освобождаются.

При выполнении оператора DEALLOCATE SQL Server освобождает разделяемую память, используемую командой описания курсора DECLARE. После выполнения этой команды невозможно выполнение команды OPEN для данного курсора.

**Пример:**

DELIMITER $$

CREATE DEFINER=`root`@`%` PROCEDURE `yearly\_streak`()

begin

declare cur\_year, max\_year, min\_year int;

select max(year), min(year) from lakers into max\_year, min\_year;

DROP TEMPORARY TABLE IF EXISTS yearly\_streak;

CREATE TEMPORARY TABLE yearly\_streak (season int, streak int, win char(1));

set cur\_year=max\_year;

year\_loop: loop

if cur\_year<min\_year then

leave year\_loop;

end if;

call streak(cur\_year, @l, @s);

insert into yearly\_streak values (cur\_year, @l, @s);

set cur\_year=cur\_year-1;

end loop;

select \* from yearly\_streak;

DROP TEMPORARY TABLE IF EXISTS yearly\_streak;

END