ТЕМА 15. КУРСОРЫ

Курсор – это механизм, инкапсулирующий заданный запрос и получающий результат запроса по нескольку строк за раз.

Назначение:

– для возможности избежать переполнения памяти, когда результат содержит большое количество строк;

– возврат из функции ссылки на курсор (эффективный способ получать большие наборы строк из функций).

1. Объявление курсорных переменных

Доступ к курсорам в PL/pgSQL осуществляется через курсорные переменные, которые всегда имеют специальный тип данных `refcursor`.

– объявление переменной типа `refcursor;`

– *имя* [[NO] SCROLL] CURSOR [(*аргументы*)] FOR *запрос;*

С указанием `SCROLL` курсор можно будет прокручивать назад. При `NO SCROLL` прокрутка назад не разрешается. Если ничего не указано, то возможность прокрутки назад зависит от запроса. Если указаны *аргументы*, то они должны представлять собой пары *имя тип_данных*, разделённые через запятую. Эти пары определяют имена, которые будут заменены значениями параметров в данном запросе. Фактические значения для замены этих имён появятся позже, при открытии курсора. Примеры:

```
DECLARE
    curs1 refcursor;
    curs2 CURSOR FOR SELECT * FROM tenk1;
    curs3 CURSOR (key integer) FOR SELECT * FROM tenk1 WHERE unique1 = key;
```

2. Открытие курсора

OPEN *несвязанная_переменная_курсора* [[NO] SCROLL] FOR *запрос;*

OPEN *несвязанная_переменная_курсора* [[NO] SCROLL] FOR EXECUTE *строка_запроса*
[USING *выражение* [, ...]];

Открытие связанного курсора

OPEN *связанная_переменная_курсора* [([*имя_аргумента* :=] *значение_аргумента* [, ...])];

```
OPEN curs2;
OPEN curs3(42);
OPEN curs3(key := 42);
```

3. Использование курсоров

FETCH [*направление* { FROM | IN }] *курсор* INTO *цель;*

Извлекает следующую строку из курсора в *цель*. В качестве *цели* может быть строковая переменная, переменная типа `record`, или разделённый запятыми список простых переменных, как и в `SELECT INTO`. Если следующей строки нет, *цели* присваивается NULL.

направление: `NEXT`, `PRIOR`, `FIRST`, `LAST`, `ABSOLUTE` *число*, `RELATIVE` *число*, `FORWARD` или `BACKWARD`. Без указания *направления* используется значение `NEXT`. Значения *направления*, которые требуют перемещения назад, приведут к ошибке, если курсор не был объявлен или открыт с указанием `SCROLL`.

MOVE [*направление* { FROM | IN }] *курсор;*

Перемещает курсор без извлечения данных. `MOVE` работает точно также как и `FETCH`, но при этом только перемещает курсор и не извлекает строку, к которой переместился.

```
UPDATE таблица SET ... WHERE CURRENT OF курсор;
DELETE FROM таблица WHERE CURRENT OF курсор;
```

Изменяет или удаляет строку, на которой позиционирован курсор

```
CLOSE курсор;
```

CLOSE закрывает связанный с курсором портал. Используется для того, чтобы освободить ресурсы раньше, чем закончится транзакция, или чтобы освободить курсорную переменную для повторного открытия.

4. Возврат курсора из функции

Используется, когда нужно вернуть множество строк и столбцов, особенно если выборки очень большие. Для этого, в функции открывается курсор и его имя возвращается вызывающему. Вызывающий затем может извлекать строки из курсора. Курсор может быть закрыт вызывающим или он будет автоматически закрыт при завершении транзакции.

5. Обработка курсора в цикле

```
[ <<метка>> ]
FOR переменная-запись IN связанная_переменная_курсора [ ( [ имя_аргумента := ]
значение_аргумента [, ...] ) ] LOOP
    операторы
END LOOP [ метка ];
```

Команда FOR автоматически открывает курсор и автоматически закрывает при завершении цикла. Список фактических значений аргументов должен присутствовать только в том случае, если курсор объявлялся с параметрами. Данная *переменная-запись* автоматически определяется как переменная типа record и существует только внутри цикла (другие объявленные переменные с таким именем игнорируется в цикле). Каждая возвращаемая курсором строка последовательно присваивается этой переменной и выполняется тело цикла.

ИНДЕКСЫ

Индексы в PostgreSQL — специальные объекты базы данных, предназначенные в основном для ускорения доступа к данным.

При использовании индексов устанавливается соответствие между ключом (например, значением проиндексированного столбца) и строками таблицы, в которых этот ключ встречается. Строки идентифицируются с помощью TID (tuple id), который состоит из номера блока файла и позиции строки внутри блока. Тогда, зная ключ или некоторую информацию о нем, можно быстро прочитать те строки, в которых может находиться интересующая нас информация, не просматривая всю таблицу полностью.

Важно понимать, что индекс, ускоряя доступ к данным, взамен требует определенных затрат на свое поддержание. При любой операции над проиндексированными данными — будь то вставка, удаление или обновление строк таблицы, — индексы, созданные для этой таблицы, должны быть перестроены, причем в рамках той же транзакции.

PostgreSQL поддерживает несколько типов индексов: B-дерево, хеш, GiST, SP-GiST, GIN и BRIN. Для разных типов индексов применяются разные алгоритмы, ориентированные на определённые типы запросов. По умолчанию команда `CREATE INDEX` создаёт индексы типа B-дерево, эффективные в большинстве случаев.

1. Создание индекса

```
CREATE [ UNIQUE ] INDEX [ [ IF NOT EXISTS ] имя ] ON имя_таблицы [ USING метод ]
    ( { имя_столбца | ( выражение ) }
    [ ASC | DESC ] [ NULLS { FIRST | LAST } ] [, ...] )
    [ WHERE предикат ]
```

`CREATE INDEX` создаёт индексы по:

– указанному столбцу(ам) заданной таблицы или материализованного представление. Д допускается указание нескольких полей;

– выражению, вычисляемым из значений одного или нескольких столбцов в строке таблицы. Это может быть полезно для получения быстрого доступа к данным по некоторому преобразованию исходных значений. Например, индекс, построенный по выражению `upper(col)`, позволит использовать поиск по индексу в предложении `WHERE upper(col) = 'JIM';`

Если в команде присутствует предложение `WHERE`, она создаёт *частичный индекс*. Такой индекс содержит записи только для части таблицы, обычно более полезной для индексации, чем остальная таблица. Частичные индексы могут быть полезны тем, что позволяют избежать индексирования распространённых значений (исключив их из индекса, можно уменьшить его размер, а значит и ускорить запросы, использующие этот индекс. Это также может ускорить операции изменения данных в таблице, так как индекс будет обновляться не всегда).

UNIQUE

Указывает, что система должна контролировать повторяющиеся значения в таблице при создании индекса (если в таблице уже есть данные) и при каждом добавлении данных. Попытки вставить или изменить данные, при которых будет нарушена уникальность индекса, будут завершаться ошибкой. Значения NULL считаются не равными друг другу. Когда для таблицы определяется ограничение уникальности или первичный ключ, PostgreSQL автоматически создаёт уникальный индекс по всем столбцам, составляющим это ограничение или первичный ключ (индекс может быть составным).

ASC

Указывает порядок сортировки по возрастанию (подразумевается по умолчанию).

DESC

Указывает порядок сортировки по убыванию.

NULLS FIRST

Указывает, что значения NULL после сортировки оказываются перед остальными. Это поведение по умолчанию с порядком сортировки `DESC`.

`NULLS LAST`

Указывает, что значения NULL после сортировки оказываются после остальных. Это поведение по умолчанию с порядком сортировки `ASC`.

Помимо простого поиска строк для выдачи в результате запроса, индексы также могут применяться для сортировки строк в определённом порядке. Это позволяет учесть предложение `ORDER BY` в запросе, не выполняя сортировку дополнительно. Из всех типов индексов, которые поддерживает PostgreSQL, сортировать данные могут только B-деревья — индексы других типов возвращают строки в неопределённом, зависящем от реализации порядке.