Лекция 11

Взаимодействие приложений и базы данных на языке SQL

Курсоры

Доступ к БД

может быть выполнен в двух режимах:

1. В интерактивном режиме;
2. В режиме выполнения прикладных программ (приложений).

В случае если данные с которыми работает программа отличаются сложной логикой взаимодействия, простых текстовых файлов XML или JASON уже недостаточно. Приходится использовать специальные программы-сервера — системы управления базами данных (СУБД).

Эта двойственность SQL создает ряд преимуществ:

- Все возможности интерактивного языка запросов доступны и в прикладном программировании. Все языки программирования имеют ***встроенный*** SQL.
- Можно в интерактивном режиме отладить основные алгоритмы обработки информации, которые в дальнейшем могут быть готовыми вставлены в работающие приложения.

Два способа применения SQL в прикладных программах

Встроенный(статический) SQL.

При таком подходе операторы SQL встраиваются непосредственно в исходный текст программы на базовом языке.

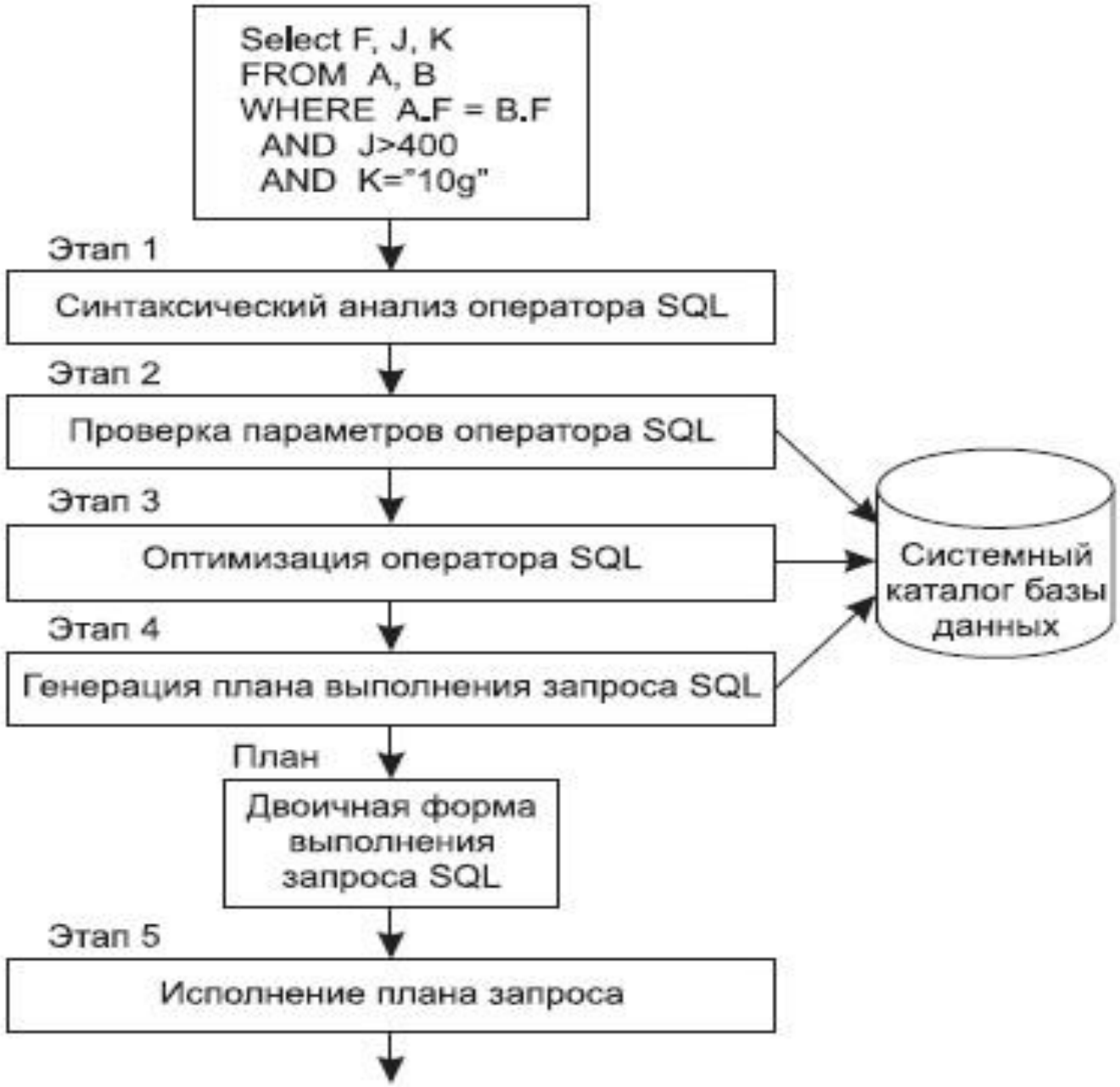
При компиляции программы со встроенными операторами SQL используется специальный препроцессор SQL, который преобразует исходный текст в исполняемую программу.

Интерфейс программирования приложений (API application program interface).

При использовании данного метода прикладная программа взаимодействует с СУБД путем применения специальных функций.

Вызывая эти функции, программа передает СУБД операторы SQL и получает обратно результаты запросов. В этом случае не требуется специализированный препроцессор.

Процесс выполнения операторов SQL



5 этапов

1. На первом этапе выполняется синтаксический анализ оператора SQL. На этом этапе проверяется корректность записи SQL-оператора в соответствии с правилами синтаксиса.
2. На этом этапе проверяется корректность параметров оператора SQL: имен отношений, имен полей данных, привилегий пользователя по работе с указанными объектами. Здесь обнаруживаются семантические ошибки.
3. На этом этапе проводится оптимизация запроса. СУБД проводит разделение целостного запроса на ряд минимальных операций и оптимизирует последовательность их выполнения с точки зрения стоимости выполнения запроса.
4. На этом этапе строится несколько планов выполнения запроса и выбирается из них один — оптимальный для данного состояния БД.
5. На четвертом этапе СУБД генерирует двоичную версию оптимального плана запроса, подготовленного на этапе 3. Двоичный план выполнения запроса в СУБД фактически является эквивалентом объектного кода программы.
6. И наконец, только на пятом этапе СУБД реализует (выполняет) разработанный план, тем самым выполняя оператор SQL.

Объединение операторов SQL с базовым языком программирования

1. Операторы SQL включаются непосредственно в текст программы на исходном языке программирования. Исходная программа поступает на вход препроцессора SQL, который компилирует операторы SQL.
2. Встроенные операторы SQL могут ссылаться на переменные базового языка программирования.
3. Встроенные операторы SQL получают результаты SQL-запросов с помощью переменных базового языка программирования.
4. Для присвоения неопределенных значений (**NULL**) атрибутам отношений БД используются специальные функции.
5. Для обеспечения построчной обработки результатов запросов во встроенный SQL добавляются несколько новых операторов, которые отсутствуют в интерактивном SQL.

Запросы делятся на 2 типа

- **Однострочные запросы**, где ожидаемые результаты соответствуют одной строке данных. Эта строка может содержать значения нескольких столбцов.
- **Многострочные запросы**, результатом которых является получение целого набора строк. При этом приложение должно иметь возможность проработать все полученные строки. Значит, должен существовать механизм, который поддерживает просмотр и обработку полученного набора строк.

Однострочный запрос во встроенном SQL

вызвал модификацию оператора SQL:

SELECT [{ALL | DISTINCT}] <список возвращаемых столбцов>

INTO <список переменных базового языка> FROM <список исходных таблиц>

[WHERE <условия соединения и поиска>]

Пример:

```
CREATE TABLE READERS
(
    READER_ID Smallint(4) PRIMARY
    KEY,FIRST_NAME char(30) NOT NULL,
    LAST_NAME char(30) NOT NULL,
    ADRES char(50) ,
    HOME_PHON char(12) ,
    WORK_PHON char(12) ,
    BIRTH_DAY date CHECK( DateDiff(year, GetDate(),
    BIRTH_DAY) >=17 )
);
```

Пример:базовые переменные

```
DECLARE @READER_ID intDECLARE
@FIRS_NAME Char(30),
@LAST_NAME Char(30),
@ADRES Char(50)
DECLARE @HOME_PHON Char(12),@WORK_PHON Char(12)
/* зададим уникальный номер читательского билета */
SET @READER_ID = 4
/* теперь выполним запрос и поместим полученные сведения в
   определенныеранее переменные */
SELECT READERS.FIRST_NAME, READERS.LAST_NAME,
   READERS.ADRES, READERS.HOME_PHON,
   READERS.WORK_PHON
INTO @FIRS_NAME, @LAST_NAME, @ADRES,
   @HOME_PHON,@WORK_PHON
FROM READERS WHERE READERS.READER_ID = @READER_ID
```

многострочные запросы

Для реализации многострочных запросов вводится новое понятие — курсора или указателя набора записей. Для работы с курсором добавляется несколько новых операторов SQL:

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.

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

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 адресацию, так и относительную адресацию, позволяет установить курсор на первую или последнюю запись виртуального набора.

Оператор **CLOSE** закрывает курсор и прекращает доступ к виртуальному набору записей. Он фактически ликвидирует связь между курсором и результатом выполнения базового запроса.

Оператор определения курсора

DECLARE <имя_курсора>

CURSOR FOR <спецификация_курсора>

<спецификация курсора>::= <выражение_запроса **SELECT**>

Имя курсора — это допустимый идентификатор в базовом языке программирования.

В объявлении курсора могут быть использованы базовые переменные.

На момент выполнения оператора **OPEN** значения всех базовых переменных, используемых в качестве входных переменных, связанных с условиями фильтрации значений в базовом запросе, должны быть уже заданы.

стандарт SQL2 Transact SQL

расширенное определение курсора

DECLARE <имя_курсора> [**INSENSITIVE**] [**SCROLL**] **CURSOR**

FOR <оператор выбора **SELECT**>

[FOR {**READ ONLY** | **UPDATE** [OF <имя_столбца 1> [,...n]]}]

Параметр **INSENSITIVE** (нечувствительный) определяет режим создания набора строк

Ключевое слово **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 доступной только стандартное перемещение вперед: спецификация NEXT в операторе **FETCH**.

Если указана спецификация **READ ONLY** (только для чтения), то изменения и обновления исходных таблиц не будут выполняться с использованием данного курсора.

При использовании параметра

UPDATE [**OF** <имя столбца 1> [,...<имя столбца n>]]

мы задаем перечень столбцов, в которых допустимы изменения в процессе нашей работы с курсором.

Такое ограничение упростит и ускорит работу СУБД. Если этот параметр не указан, то предполагается, что допустимы изменения всех столбцов курсора.

Пример:

Вывести список должников, их данные и названия книг

```
DECLARE Debtor_reader_cursor CURSOR FOR
  SELECT readers.first_name, readers.last_name,
  readers.adres, readers.home_phon, readers.work_phon,
  books.title
  FROM readers, books, exemplar
  WHERE readers.reader_id = exemplar.reader_id
  AND books.isbn = exemplare.isbn
  AND exemplar.data_out < Getdate()
  ORDER BY readers.first_name
```

Оператор открытия курсора

Оператор открытия курсора имеет следующий синтаксис:

OPEN <имя_курсора>

[**USING** <список базовых переменных>]

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

Оператор чтения очередной строки курсора

После открытия указатель текущей строки установлен перед первой строкой курсора.

Стандартно оператор **FETCH** перемещает указатель текущей строки на следующую строку и присваивает базовым переменным значение столбцов, соответствующее текущей строке.

Простой оператор FETCH имеет следующий синтаксис:

FETCH <имя_курсора>

INTO <список переменных базового языка >

Расширенный оператор FETCH

синтаксис:

FETCH [NEXT | PRIOR | FIRST | LAST

ABSOLUTE {n | <имя_переменной>}

|RELATIVE {n|<имя_переменной>}]

FROM<имя_курсора>

INTO <список базовых переменных>

Оператор закрытия курсора

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

CLOSE <имя_курсора>

В PostgreSQL:

CLOSE { имя | ALL }

Удаление и обновление данных с использованием курсора

DELETE FROM <имя_таблицы> WHERE CURRENT OF <имя курсора>

UPDATE <имя_таблицы>

SET <имя_столбца1>= {<значение> | NULL}

[{,<имя_столбца_N>= {<значение> | NULL}}...]

WHERE CURRENT OF <имя_курсора>

Стандарт, требования к курсору:

- Запрос, связанный с курсором, должен считывать данные из одной исходной таблицы, то есть в предложении **FROM** запроса **SELECT**, связанного с определением курсора (**DECLARE CURSOR**), должна быть задана только одна таблица.
- В запросе не может присутствовать параметр упорядочения **ORDER BY**. Для того чтобы сохранялось взаимно однозначное соответствие строк курсора и исходной таблицы, курсор не должен идентифицировать упорядоченный набор данных.
- В запросе не должно присутствовать ключевое слово **DISTINCT**.
- Запрос не должен содержать операций группировки, то есть в нем не должно присутствовать предложение **GROUP BY** или **HAVING**.
- Пользователь, который хочет применить операции позиционного удаления или обновления, должен иметь соответствующие права на выполнение данных операций над базовой таблицей.

Минимум количества требуемых блокировок

- Необходимо делать транзакции как можно короче.
- Необходимо выполнять оператор завершения **COMMIT** после каждого запроса и как можно скорее после изменений, сделанных программой.
- Необходимо избегать программ, в которых осуществляется интенсивное взаимодействие с пользователем или осуществляется просмотр очень большого количества строк данных.
- Если возможно, то лучше не применять **прокручиваемые курсоры** (**SCROLL**), потому что они требуют блокирования всех строк выборки, связанных с открытым курсором.
- Использование простого последовательного курсора позволит системе разблокировать текущую строку, как только будет выполнена операция **FETCH**, что минимизирует блокировки других пользователей, работающих параллельно с вами и использующих те же таблицы.
- Если возможно, определяйте курсор как **READ ONLY**.

Курсоры в PostgreSQL

Курсор, это такой объект, который позволяет работать с одной или несколькими строками запроса, не перегружая память. Это очень похоже на итераторы в программировании.

По другому можно сказать, что курсор это временная таблица, в которой установлен порядок. Есть последующие и предыдущие строки. Можно последовательно переходить от одной строки к другой.

Это противоречит общим принципам реляционных баз данных, но зато, в частности, позволяет интегрировать алгоритмический язык программирования с языком запросов SQL.

Курсор как объект может существовать и на стороне сервера и на стороне клиента.

Курсоры в PostgreSQL

Курсоры PostgreSQL могут инкапсулировать запросы и обрабатывать каждую строку записей по отдельности.

Курсоры можно использовать, когда мы хотим выполнить пакетную обработку большого количества наборов результатов, поскольку однократная обработка может вызвать переполнение памяти.

Кроме того, мы можем определить функцию, которая возвращает переменную типа курсора, которая является эффективным способом для функции возвращать большой набор данных.

Вызывающая функция обрабатывает результат в соответствии с возвращенным курсором.

Курсоры в PostgreSQL

Оператор **DECLARE** позволяет пользователю создавать курсоры, с помощью которых можно выбирать по очереди некоторое количество строк из результата большого запроса.

DECLARE ***имя*** [BINARY] [ASENSITIVE | INSENSITIVE] [[NO] SCROLL] CURSOR [{ WITH | WITHOUT } HOLD] FOR ***запрос***

В PostgreSQL все курсоры являются «нечувствительными», поэтому данные ключевые слова не действуют и принимаются только для совместимости со стандартом SQL.

WITH HOLD, WITHOUT HOLD - Указание WITH HOLD определяет, что курсор можно продолжать использовать после успешной фиксации создавшей его транзакции. WITHOUT HOLD определяет, что курсор нельзя будет использовать за рамками транзакции, создавшей его. Если не указано ни WITHOUT HOLD, ни WITH HOLD, по умолчанию подразумевается WITHOUT HOLD.

Когда курсор создан, через него можно получать строки, применяя команду FETCH.

FETCH [***направление***] [FROM | IN] ***имя_курсора***

NEXT PRIOR FIRST; LAST ABSOLUTE ***число;*** RELATIVE ***число число;***
ALL; FORWARD; FORWARD ***число;*** FORWARD ALL; BACKWARD;
BACKWARD; ***число;*** BACKWARD ALL.

Курсоры в PostgreSQL

Один из способов создать курсорную переменную, просто объявить её как переменную типа **refcursor**.

Другой способ заключается в использовании синтаксиса объявления курсора, который в общем виде выглядит так:

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

Примеры:

DECLARE

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

Все три переменные имеют тип данных refcursor.

Первая может быть использована с любым запросом, вторая связана (bound) с полностью сформированным запросом, а последняя связана с параметризованным запросом. (key будет заменён целочисленным значением параметра при открытии курсора.)

Про переменную curs1 говорят, что она является несвязанной (unbound), так как к ней не привязан никакой запрос.

Курсоры в PostgreSQL

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

Курсорная переменная открывается и получает конкретный запрос для выполнения. Курсор не может уже быть открытым, а курсорная переменная обязана быть несвязанной (переменной типа refcursor).

Запрос должен быть командой SELECT или любой другой, которая возвращает строки. Запрос обрабатывается так же, как и другие команды SQL в PL/pgSQL: имена переменных PL/pgSQL заменяются на значения, план запроса кешируется для повторного использования.

Подстановка значений переменных PL/pgSQL проводится при открытии курсора командой OPEN, последующие изменения значений переменных не влияют на работу курсора.

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

Пример:

OPEN curs1 FOR SELECT * FROM foo WHERE key = mykey;

Закрытие курсора

CLOSE { имя | ALL } освобождает ресурсы, связанные с открытым курсором. Когда курсор закрыт, никакие операции с ним невозможны. Закрывать курсор следует, когда он становится ненужным.

Простейший пример с курсором в PostgreSQL

- Определить курсор в разделе *DECLARE*.
- При определении указывается запрос, на основе которого будет формироваться курсор.
- Далее курсор должен быть открыт.
- Использование происходит внутри цикла *LOOP*.
- Команда *FETCH* выбирает очередную строку таблицы - курсора.

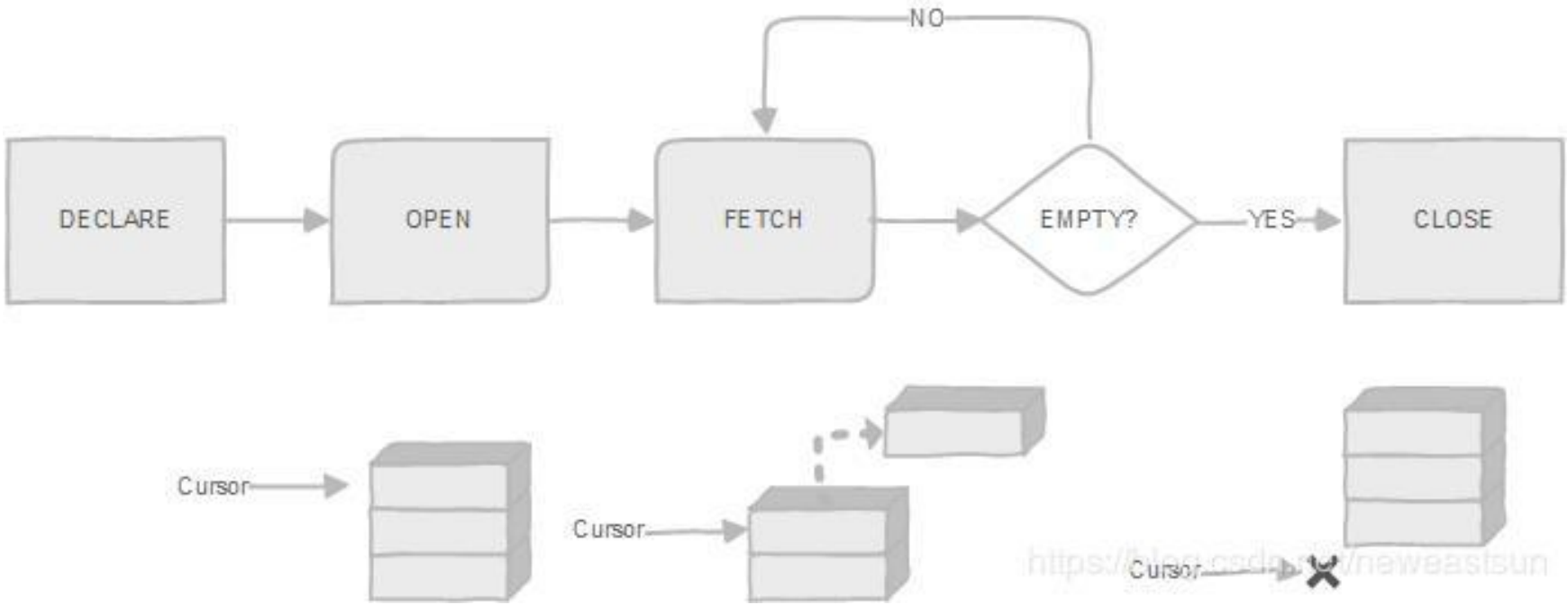
При этом происходит присвоение поля (полей) переменной.

- После этой команды следует проверить, не закончился ли набор строк (*IF NOT FOUND THEN EXIT;END IF;*).
- По выходу из цикла курсор должен быть закрыт.
- Хранимая функция *curs1()* возвращает среднюю оценку по таблице *marks*.
- Вызвать функцию можно просто командой *select*, как обычно

select * from curs1()

```
CREATE FUNCTION curs1()
   RETURNS real
AS $$
DECLARE
      cr1 CURSOR FOR SELECT mark FROM
     marks;
      s real := 0.0;
      mk integer;
      n real := 0.0;
BEGIN
      OPEN cr1;
      LOOP
        FETCH cr1 INTO mk;
        IF NOT FOUND THEN EXIT;END IF;
      s := s + mk;
      n := n + 1.0;
      END LOOP;
      CLOSE cr1;
      s := s/n;
      RETURN s;
END;
$$
LANGUAGE 'plpgsql';
```

Как использовать курсоры PostgreSQL



1. Первый шаг - объявить курсор.
2. Затем откройте курсор.
3. Затем возьмите строку из результата в целевую переменную.
4. После этого проверьте, есть ли еще строки для выборки. Как вернуться к шагу три, иначе перейти к шагу пять.
5. Наконец, закройте курсор.

Как использовать курсоры PostgreSQL(1)

Чтобы получить доступ к курсору, переменная курсора должна быть объявлена в блоке объявления. Используется специальный тип **REFCURSOR** для объявления переменных курсора. Объявляется несвязанный курсор:

DECLARE my_cursor **REFCURSOR**;

Другой способ объявить привязку переменной курсора и связать оператор запроса при объявлении, синтаксис выглядит следующим образом:

cursor_name [[NO] SCROLL] **CURSOR** [(name datatype, name data type, ...)] **FOR** query;

Разделенное запятыми имя данных параметра (имя типа данных), используется для определения параметров запроса. Эти параметры заменяются при открытии курсора.

В примере показано, как объявить переменную курсора:

DECLARE

cur_films **CURSOR FOR** SELECT * FROM film;

cur_films2 **CURSOR** (year integer) **FOR** SELECT * FROM film WHERE release_year = year;

Как использовать курсоры PostgreSQL(2)

cur_films - это переменная курсора, которая инкапсулирует все записи в таблице фильмов.

cur_films2 - это переменная курсора, которая инкапсулирует таблицу фильмов с определенной записью года выпуска.

Связанная переменная курсора инициализируется строковым значением, указывающим его имя (официальный документ становится именем портала), что в последующем согласуется.

Однако несвязанная переменная курсора изначально по умолчанию имеет нулевое значение, поэтому она примет автоматически сгенерированное уникальное имя позже.

Когда курсор определен в рекурсивной функции, он должен быть определен как несвязанный курсор, в противном случае возникнет ошибка: курсор уже используется.

Открытие курсора PostgreSQL(3)

Курсоры должны быть открыты перед использованием.

PostgreSQL предоставляет специальный синтаксис для открытия **связанных и несвязанных курсоров.**

Синтаксис открытого **несвязанного курсора**:

OPEN unbound_cursor_variable [[**NO**] **SCROLL**] **FOR** query;

Поскольку несвязанные переменные курсора не связаны ни с одним запросом при объявлении, запрос должен быть указан при открытии. Пример:

OPEN my_cursor **FOR** SELECT * FROM city WHERE counter = p_country;

PostgreSQL может открывать курсоры и связываться с динамическими запросами со следующим синтаксисом:

OPEN unbound_cursor_variable[[**NO**] **SCROLL**] **FOR EXECUTE** query_string [USING expression [, ...]];

В примере строим динамический запрос для сортировки результатов на основе параметра sort_field, затем открываем курсор и выполняем динамический запрос:

query := 'SELECT * FROM city ORDER BY $1';

Открытие курсора PostgreSQL(4)

Открытый связанный курсор

Поскольку запрос уже связан, когда объявление курсора связано, вам нужно только передать необходимые параметры при открытии:

OPEN cursor_variable[(name:=value,name:=value,...)]; 1

В следующем примере открываются курсоры привязки cur_films и cur_films2, объявленные в предыдущем слайде:

OPEN cur_films; **OPEN** cur_films2(year:=2005);

Использование курсора

После открытия курсора вы можете использовать **FETCH**, **MOVE**, чтобы манипулировать курсором и обновлять или удалять записи.
Синтаксис:

FETCH [direction { **FROM** | **IN** }] cursor_variable **INTO** target_variable;

Оператор **FETCH** получает следующую строку из курсора и присваивает ее целевой переменной target_variable, которая может быть типом записи или переменной строки или разделенным запятыми списком переменных.

Если извлекаемой строки не найдено, целевая переменная target_variable имеет значение **null**.

FETCH [direction

Значения direction : **NEXT**, **LAST**, **PRIOR**, **FIRST**, **ABSOLUTE** count, **RELATIVE** count, **FORWARD**, **BACKWARD**

При использовании SCROLL для объявления курсоров может быть **FORWARD** и **BACKWARD**, по умолчанию установлено **NEXT**

Пример:

FETCH cur_films **INTO** row_film;

FETCH LAST FROM row_film **INTO** title, release_year;

Перемещение курсора

Синтаксис выглядит следующим образом:

MOVE [direction { **FROM** | **IN** }] cursor_variable;

Если хотите только переместить курсор и не возвращать строки, вы можете использовать оператор **move**. Ключевое слово direction соответствует выражению **FETCH**.

MOVE cur_films2;

MOVE LAST FROM cur_films;

MOVE RELATIVE -1 **FROM** cur_films;

MOVE FORWARD 3 **FROM** cur_films;

Удаление или обновление строки

После определения позиции курсора можно удалить или обновить строки, используя операторы:

DELETE WHERE CURRENT OF или **UPDATE WHERE CURRENT OF**

Синтаксис:

UPDATE table_name **SET** column = value, ... **WHERE CURRENT OF** cursor_variable;

DELETE FROM table_name **WHERE CURRENT OF** cursor_variable;

Пример:

UPDATE film **SET** release_year = p_year **WHERE CURRENT OF** cur_films;

Закрытие курсора:

CLOSE cursor_variable;

Оператор **CLOSE** освобождает ресурс или освобождает переменную курсора, чтобы его можно было открыть снова.