




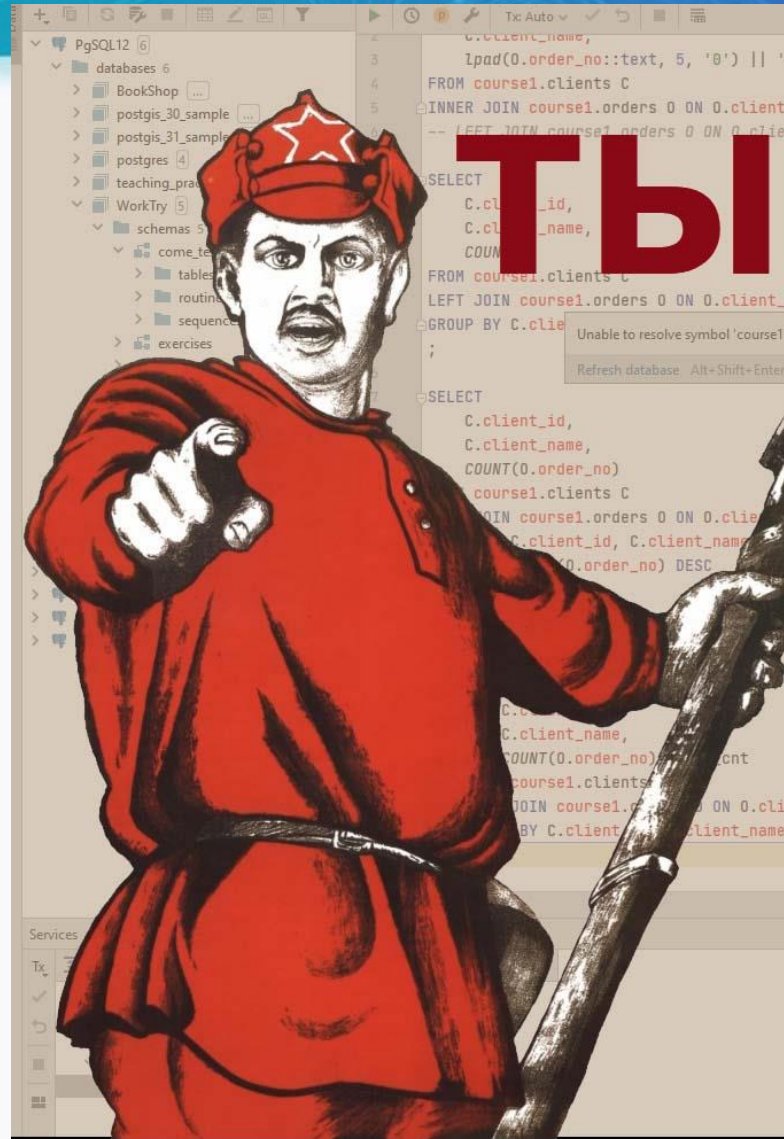
Онлайн-образование



Меня хорошо видно && слышно?

Ставьте  , если все хорошо
Напишите в чат, если есть проблемы
заодно проверяем, включена ли запись занятия

...а включена ли запись?



ВКЛЮЧИЛ ЗАПИСЬ?



Триггеры



ЗОЛОТОВ АНТОН

telegram @AVZolotov

Правила вебинара



Активно участвуем



Задаем вопрос в чат или голосом



Off-topic обсуждаем вSlack



Вопросы вижу в чате, могу ответить не сразу

Маршрут вебинара



Цели вебинара | После занятия вы сможете

- 1 понять как работают хранимые функции и процедуры и для чего они нужны
- 2 объяснить назначение триггеров и курсоров
- 3 обрабатывать ошибки в хранимых функциях и процедурах

Смысл | Зачем вам это уметь, в результате:

1 настроить безопасность

2 настроить оптимальную производительность

3 научитесь использовать хранимые функции и процедуры



Триггеры

Триггеры. Определение.

Предоставляемый СУБД механизм, позволяющий автоматически выполнять действия СУБД при возникновении соответствующих условий.

- DML-триггеры – срабатывают при модификации данных
- DDL-триггеры – срабатывают при изменениях объектов БД
(событийные триггеры)

Триггеры. Назначение.

- Протоколирование выполняемых действий
- Обеспечение целостности данных
- Реализация сложной бизнес-логики
- Повышение производительности системы*

* Но не во всех случаях!

DML Триггеры.

В PostgreSQL триггеры представлены двумя объектами: триггерной функцией и триггером, как таковым

Собственно триггер	Объект, изменения в котором вызовут срабатывание	Таблица или представление
	Событие вызывающее срабатывание	INSERT, UPDATE, DELETE, TRUNCATE
	Момент срабатывания	BEFORE, AFTER, INSTEAD OF
	Возможность отложенного выполнения	Для CONSTRAINT триггеров
	Построчное или пооператорное срабатывание	
	Спецификация переходных отношений	
Триггерная функция	Условие выполнения	
	Реализует алгоритм действий, которые надо выполнить при срабатывании	
	Выполняется в той же транзакции, что и основная (вызвавшая срабатывание) операция	
	Не принимает параметры (!)	
	Возвращает значение типа trigger (record, структура соответствует строке таблицы)	

DML Триггеры. Создание триггера и триггерной функции.

```
CREATE TRIGGER триггер  
  BEFORE | AFTER | INSTEAD OF { событие [ OR событие ] } ON таблица  
  FOR EACH { ROW | STATEMENT }  
  [ WHEN (condition) ]  
  EXECUTE PROCEDURE функция ()
```

```
CREATE FUNCTION функция ()  
  RETURNS trigger AS  
  DECLARE  
    объявления;  
  BEGIN  
    команды;  
  END;  
  LANGUAGE plpgsql;
```

*Любой язык, кроме чистого SQL

DML Триггеры. Использование.

Момент срабатывания	Событие	Построчный	Пооператорный
BEFORE	INSERT/UPDATE/DELETE	Таблицы, сторонние (foreign) таблицы	Таблицы, сторонние таблицы, представления
	TRUNCATE	-	Таблицы
AFTER	INSERT/UPDATE/DELETE	Таблицы, сторонние таблицы	Таблицы, сторонние таблицы, представления
	TRUNCATE	-	Таблицы
INSTEAD OF	INSERT/UPDATE/DELETE	Представления	-
	TRUNCATE	-	-

DML Триггеры. Специальные переменные триггерных функций.

Если функция на PL/PgSQL вызвана сработавшим триггером (т.е. как триггерная функция) – в блоке верхнего уровня автоматически создаются несколько специальных переменных:

ИМЯ	ТИП	описание
NEW	RECORD	Новые значения полей записи базы данных, созданной командой INSERT или обновленной командой UPDATE, при срабатывании триггера уровня записи (FOR EACH ROW). Переменная используется для модификации новых записей.
OLD	RECORD	Старые значения полей записи базы данных, содержащиеся в записи перед выполнением команды DELETE или UPDATE при срабатывании триггера уровня записи (FOR EACH ROW)
TG_NAME	name	Имя сработавшего триггера.
TG_WHEN	text	Строка BEFORE или AFTER в зависимости от момента срабатывания триггера, указанного в определении (до или после операции)

DML Триггеры. Специальные переменные триггерных функций.

ИМЯ	ТИП	описание
TG_LEVEL	text	Строка ROW или STATEMENT в зависимости от уровня триггера, указанного в определении.
TG_OP	text	Строка INSERT, UPDATE или DELETE в зависимости от операции, вызвавшей срабатывание триггера.
TG_RELID	oid	Идентификатор объекта таблицы, в которой сработал триггер.
TG_RELNAME	name	Имя таблицы, в которой сработал триггер (устарело!).
TG_TABLE_NAME	name	Имя таблицы, в которой сработал триггер.
TG_TABLE_SCHEMA	name	Имя схемы, содержащей таблицу, для которой сработал триггер
TG_NARGS	integer	Число аргументов в команде CREATE TRIGGER, которые передаются в триггерную функцию
TG_ARGV[]	text[]	Аргументы от оператора CREATE TRIGGER. Индекс массива начинается с 0. Для недопустимых значений индекса (< 0 или >= tg_nargs) возвращается NULL

DML Триггеры. Срабатывание: BEFORE

STATEMENT	Сработает однократно перед выполнением оператора, независимо от того, сколько строк будет затронуто оператором (возможно - ни одной).
	Возвращаемое значение будет игнорироваться. Можно вернуть NULL.
	Если в процессе выполнения триггерной функции возникла ошибка - операция отменяется.
	Доступны переменные TG_*
ROW	Срабатывает перед выполнением действия со строкой.
	Возвращаемое значение - строка. Возможен возврат измененной строки. Возврат NULL воспринимается, как отмена действия со строкой
	Доступны переменные TG_*

DML Триггеры. Срабатывание: AFTER

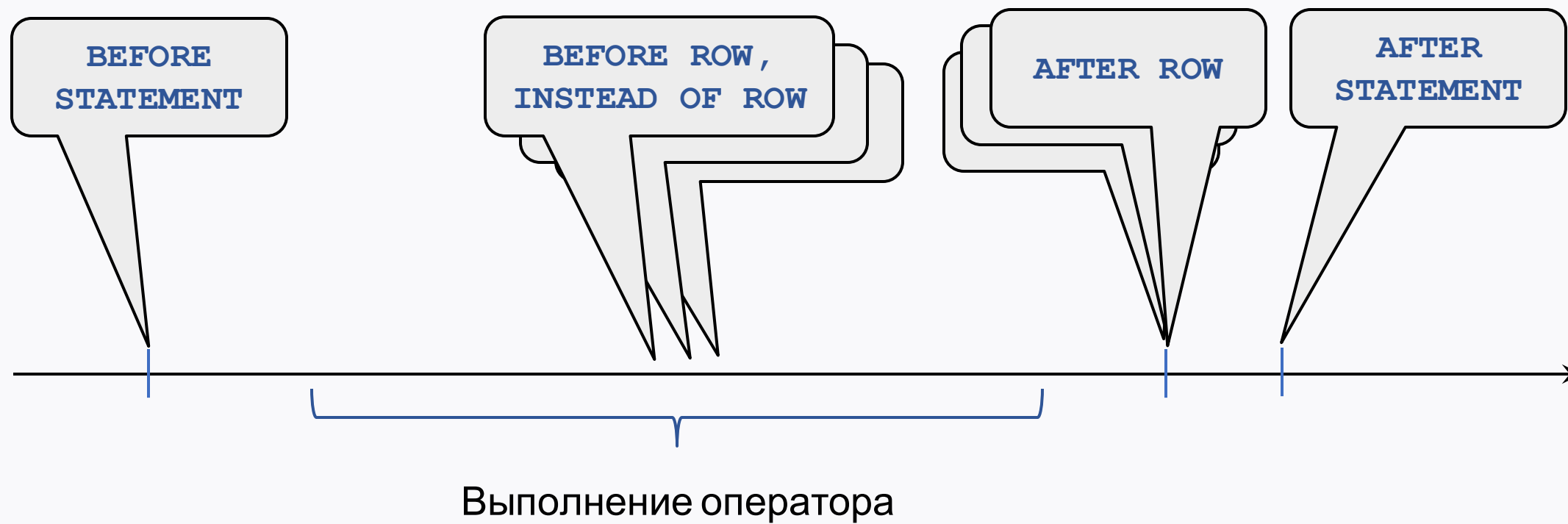
ROW	Срабатывает после выполнения оператора (!) - события помещаются в очередь и обрабатываются после окончания операции.
	Возвращаемое значение игнорируется - de-facto операция уже выполнена.
	Доступны переменные TG_*
	Доступны переходные таблицы (хотя обычно transition tables используются триггерами AFTER STATEMENT)
STATEMENT	Срабатывает однократно после того, как выполнение оператора завершится и сработают все триггеры AFTER ROW, независимо от того, сколько строк будет затронуто оператором (возможно - ни одной).
	Возвращаемое значение будет игнорироваться. Можно вернуть NULL.
	Контекст вызова может быть передан с помощью переходных таблиц (OLD TABLE, NEW TABLE)
	Доступны переменные TG_*
	Доступны переходные таблицы

DML Триггеры. Срабатывание: INSTEAD OF

Триггеры INSTEAD OF определяются только для представлений и только построчные (FOR EACH ROW). Срабатывают не до, не после, а вместо операции. Обычно используются для операций INSERT, UPDATE или DELETE над данными в базовых таблицах представления.

Возвращаемое значение – запись NEW, возможно – изменённая (в этом случае выражение RETURNING выдаст уже именно эти, измененные значения)

DML Триггеры. Порядок срабатывания.



DML Триггеры. Что плохого?

- В незадокументированной (или плохо задокументированной) системе бывает сложно отследить логику выполнения. Проблемы с отладкой неизбежны
- Перегруженную триггерами систему сложно поддерживать и развивать.
- Возможно каскадное срабатывание триггеров, не исключена рекурсия
- Неоправданное применение триггеров может значительно снизить производительность системы.

Событийные триггеры.

Срабатывают при выполнении не DML, а DDL операторов:

- CREATE ***
- ALTER ***
- DROP ***
- COMMENT ***
- GRANT ***
- REVOKE***

Для одного события может быть создано несколько триггеров, в этом случае они будут вызываться в алфавитном порядке

Событийные триггеры. События.

`ddl_command_start`

Событие происходит непосредственно перед выполнением команд CREATE, ALTER, DROP, SECURITY LABEL, COMMENT, GRANT и REVOKE. Проверка на существование объекта перед срабатыванием триггера не производится.

Событие не происходит для команд DDL, обращающихся к общим объектам кластера базы данных — базам данных, табличным пространствам, ролям, а также к самим триггерам событий. Команда SELECT INTO (равнозначна CREATE TABLE AS) также порождает событие `ddl_command_start`

Событийные триггеры. События.

`ddl_command_end`

Событие происходит непосредственно после выполнения команд CREATE, ALTER, DROP, SECURITY LABEL, COMMENT, GRANT и REVOKE.

NB: этот триггер срабатывает после того, как эти действия имели место (но до фиксации транзакции), так что в системных каталогах можно увидеть уже изменённое состояние.

Получение дополнительной информации об операторах, вызвавших событие: функция `pg_event_trigger_ddl_commands()`

<https://postgrespro.ru/docs/postgresql/13/functions-event-triggers#PG-EVENT-TRIGGER-DDL-COMMAND-END-FUNCTIONS>

Событийные триггеры. События.

sql_drop

Событие происходит непосредственно перед событием `ddl_command_end` для команд, которые удаляют объекты базы данных.

Получение дополнительной информации об операторах, вызвавших событие: функция `pg_event_trigger_dropped_objects ()`

<https://postgrespro.ru/docs/postgresql/13/functions-event-triggers#PG-EVENT-TRIGGER-SQL-DROP-FUNCTIONS>

Событийные триггеры. События.

table_rewrite

Событие происходит после перезаписи таблиц командами ALTER TABLE и ALTER TYPE.

Если перезапись таблицы вызвана другими командами (CLUSTER, VACUUM) – событие table_rewrite не происходит

Событийные триггеры.

Полный список команд SQL, вызывающих срабатывание триггеров событий – в документации (табл. 39.1)

<https://postgrespro.ru/docs/postgresql/13/event-trigger-matrix>

Событийные триггеры. Использование

<https://www.enterprisedb.com/postgres-tutorials/how-use-event-triggers-postgresql>

The image features a high-angle, aerial view of a dense urban skyline, likely New York City, with numerous skyscrapers and buildings. The entire image is overlaid with a semi-transparent blue and green gradient. A network of thin, light blue lines connects various points across the gradient, creating a digital or technological feel. In the center of the image, the word "Вопросы?" is written in a large, bold, white sans-serif font.

Вопросы?

The background of the entire image is an aerial photograph of a dense city skyline, likely New York City, with numerous skyscrapers. The image is overlaid with a semi-transparent blue and green gradient. A network of white lines and dots, resembling a data or communication network, is visible across the entire background.

Последовательности

Последовательности

<https://postgrespro.ru/docs/postgrespro/12/sql-createsequence>

The background of the image is a high-angle, aerial photograph of a dense urban skyline, likely New York City, featuring numerous skyscrapers and buildings. The image is overlaid with a semi-transparent blue and green gradient. A network of thin, light blue lines connects various points across the gradient, creating a digital or technological aesthetic. The word "Курсоры" is centered in the middle of the image in a large, white, sans-serif font.

Курсоры

Курсоры

<https://postgrespro.ru/docs/postgrespro/12/plpgsql-cursors>

Вместо того чтобы сразу выполнять весь запрос, есть возможность настроить курсор, инкапсулирующий запрос, и затем получать результат запроса по несколько строк за раз. Одна из причин так делать заключается в том, чтобы избежать переполнения памяти, когда результат содержит большое количество строк. (Пользователям PL/pgSQL не нужно об этом беспокоиться, так как циклы используют курсоры, чтобы избежать проблем с памятью.) Более интересным FOR автоматически вариантом использования является возврат из функции ссылки на курсор, что позволяет вызывающему получать строки запроса. Это эффективный способ получать большие наборы строк из функций.

Курсоры

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

(Для совместимости с Oracle, FOR можно заменять на IS.) 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.

Курсоры

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

Открытие курсора

Прежде чем получать строки из курсора, его нужно открыть. (Это эквивалентно действию SQL-команды DECLARE CURSOR.) В PL/pgSQL есть три формы оператора OPEN, две из которых используются для несвязанных курсорных переменных, а третья для связанных.

Примечание

Связанные курсорные переменные можно использовать с циклом FOR без явного открытия курсора, как описано в [Подразделе 41.7.4](#).

Курсоры

OPEN FOR *запрос*

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 (то есть просто переменной типа refcursor). Запрос должен быть командой SELECT или любой другой, которая возвращает строки (к примеру EXPLAIN). 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. SCROLL и NO SCROLL имеют тот же смысл, что и для связанного курсора.

Пример:

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

Курсоры

OPEN FOR EXECUTE

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

Переменная курсора открывается и получает конкретный запрос для выполнения. Курсор не может быть уже открыт и он должен быть объявлен как несвязанная переменная курсора (то есть, как просто переменная `refcursor`). Запрос задаётся строковым выражением, так же, как в команде EXECUTE. Как обычно, это даёт возможность гибко менять план запроса от раза к разу (см. [Подраздел 41.11.2](#)). Это также означает, что замена переменных происходит не в самой строке команды. Как и с EXECUTE, значения параметров вставляются в динамическую команду, используя `format()` и USING. Параметры SCROLL и NO SCROLL здесь действуют так же, как и со связанным курсором.

Пример:

```
OPEN curs1 FOR EXECUTE format('SELECT * FROM %I WHERE col1 = $1',tabname) USING keyvalue;
```

В этом примере в текст запроса вставляется имя таблицы с применением `format()`. Значение, сравниваемое с `col1`, вставляется посредством параметра USING, так что заключать его в апострофы не нужно.

Курсоры

Открытие связанного курсора

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

Эта форма OPEN используется для открытия курсорной переменной, которая была связана с запросом при объявлении. Курсор не может уже быть открытым. Список фактических значений аргументов должен присутствовать только в том случае, если курсор объявлялся с параметрами. Эти значения будут подставлены в запрос.

План запроса для связанного курсора всегда считается кешируемым. В этом случае нет эквивалента EXECUTE. Обратите внимание, что SCROLL и NO SCROLL не могут быть указаны в этой форме OPEN, возможность прокрутки назад была определена при объявлении курсора.

Примеры (здесь используются ранее объявленные курсоры):

```
OPEN curs2;
```

```
OPEN curs3(42);
```

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

Курсоры

Так как для связанного курсора выполняется подстановка значений переменных, то, на самом деле, существует два способа передать значения в курсор. Либо использовать явные аргументы в OPEN, либо неявно, ссылаясь на переменные PL/pgSQL в запросе. В связанном курсоре можно ссылаться только на те переменные, которые были объявлены до самого курсора. В любом случае значение переменной для подстановки в запрос будет определяться на момент выполнения OPEN. Вот ещё один способ получить тот же результат с curs3, как в примере выше:

```
DECLARE
    key integer;
    curs4 CURSOR FOR SELECT * FROM tenk1 WHERE unique1 = key;
BEGIN
    key := 42;
    OPEN curs4;
```

Курсоры. Использование

FETCH

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

FETCH 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. Как и в SELECT INTO, проверить, была ли получена запись, можно при помощи специальной переменной FOUND.

Здесь *направление* может быть любым допустимым в SQL-команде [FETCH](#) вариантом, кроме тех, что извлекают более одной строки. А именно: NEXT, PRIOR, FIRST, LAST, ABSOLUTE *число*, RELATIVE *число*, FORWARD или BACKWARD. Без указания *направления* подразумевается вариант NEXT. Везде, где используется *число*, оно может определяться любым целочисленным выражением (в отличие от SQL-команды FETCH, допускающей только целочисленные константы). Значения *направления*, которые требуют перемещения назад, приведут к ошибке, если курсор не был объявлен или открыт с указанием SCROLL.

Курсоры. Использование

курсор это переменная с типом `refcursor`, которая ссылается на открытый портал курсора.

Примеры:

```
FETCH curs1 INTO rowvar;
```

```
FETCH curs2 INTO foo, bar, baz;
```

```
FETCH LAST FROM curs3 INTO x, y;
```

```
FETCH RELATIVE -2 FROM curs4 INTO x;
```


Курсоры. Использование

MOVE

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

MOVE перемещает курсор без извлечения данных. MOVE работает точно так же как и FETCH, но при этом только перемещает курсор и не извлекает строку, к которой переместился. Как и в SELECT INTO, проверить успешность перемещения можно с помощью специальной переменной FOUND.

Примеры:

MOVE curs1;

MOVE LAST FROM curs3;

MOVE RELATIVE -2 FROM curs4;

MOVE FORWARD 2 FROM curs4;

Курсоры. Использование

UPDATE/DELETE WHERE CURRENT OF

UPDATE *таблица* SET ... WHERE CURRENT OF *курсор*;

DELETE FROM *таблица* WHERE CURRENT OF *курсор*;

Когда курсор позиционирован на строку таблицы, эту строку можно изменить или удалить при помощи курсора. Есть ограничения на то, каким может быть запрос курсора (в частности, не должно быть группировок), и крайне желательно использовать указание FOR UPDATE. За дополнительными сведениями обратитесь к странице справки [DECLARE](#).

Пример:

```
UPDATE foo SET dataval = myval WHERE CURRENT OF curs1;
```

Курсоры. Использование

CLOSE

`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.

Пример:

```
CLOSE curs1;
```

Курсоры. Использование

Возврат курсора из функции

Курсоры можно возвращать из функции на PL/pgSQL. Это полезно, когда нужно вернуть множество строк и столбцов, особенно если выборки очень большие. Для этого, в функции открывается курсор и его имя возвращается вызывающему (или просто открывается курсор, используя указанное имя портала, каким-либо образом известное вызывающему). Вызывающий затем может извлекать строки из курсора. Курсор может быть закрыт вызывающим или он будет автоматически закрыт при завершении транзакции.

Курсоры. Использование

Обработка курсора в цикле

Один из вариантов цикла FOR позволяет перебирать строки, возвращённые курсором. Вот его синтаксис:

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

Курсорная переменная должна быть связана с запросом при объявлении. Курсор *не может* быть открытым. 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. Эти значения будут подставлены в запрос, как и при выполнении OPE


<https://info-comp.ru/obucheniest/254-cursor-in-functions.html>



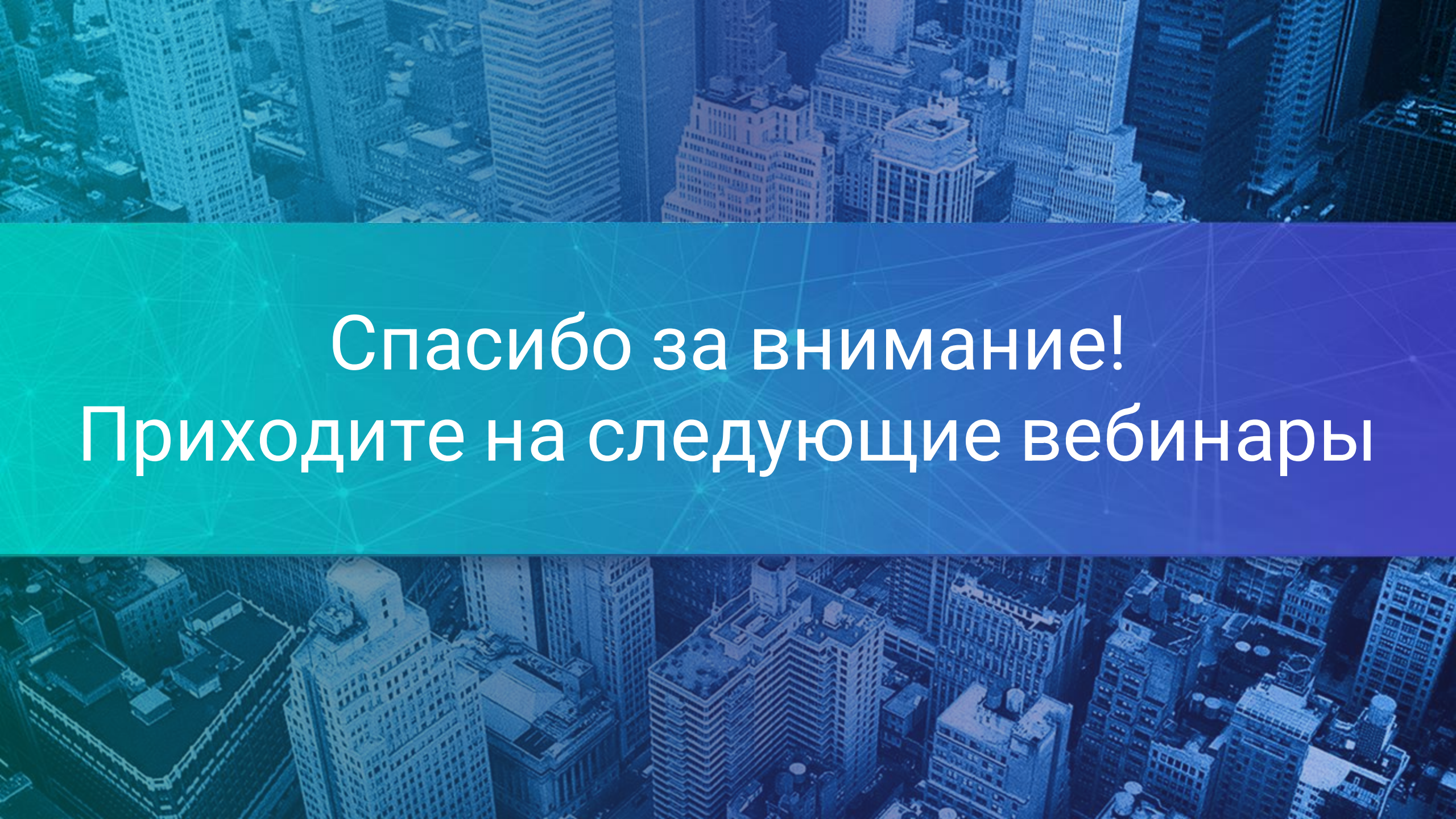
Порефлексируем

Вопросы?

- Кто что запомнил?)
- Хватило ли практики?



Заполните, пожалуйста,
опрос о занятии по ссылке в чате
<https://otus.ru/polls/32377/>

The background of the entire slide is an aerial photograph of a dense city skyline, likely New York City, with numerous skyscrapers. A semi-transparent blue overlay covers the image, featuring a subtle network or geometric pattern of lines and dots. The text is centered within this blue area.

Спасибо за внимание!
Приходите на следующие вебинары