

PL/pgSQL Обработка ошибок



Авторские права

© Postgres Professional, 2017 год.

Авторы: Егор Рогов, Павел Лузанов

Использование материалов курса

Некоммерческое использование материалов курса (презентации, демонстрации) разрешается без ограничений. Коммерческое использование возможно только с письменного разрешения компании Postgres Professional. Запрещается внесение изменений в материалы курса.

Обратная связь

Отзывы, замечания и предложения направляйте по адресу:

edu@postgrespro.ru

Отказ от ответственности

Компания Postgres Professional не несет никакой ответственности за любые повреждения и убытки, включая потерю дохода, нанесенные прямым или косвенным, специальным или случайным использованием материалов курса. Компания Postgres Professional не предоставляет каких-либо гарантий на материалы курса. Материалы курса предоставляются на основе принципа «как есть» и компания Postgres Professional не обязана предоставлять сопровождение, поддержку, обновления, расширения и изменения.

Обработка ошибки

Как происходит поиск обработчика

Передача ошибки на клиент и в журнал сообщений сервера

Надо ли обрабатывать ошибки и каком уровне это делать

```
[ DECLARE
    объявления ]
BEGIN
    операторы
EXCEPTION
    WHEN условие [OR условие ...] THEN
        операторы_обработчика
    [ WHEN условие [OR условие ...] THEN
        операторы_обработчика
    ... ]
END;
```

Условие определяет тип перехватываемой ошибки

имя или код ошибки — двухуровневая иерархия
OTHERS перехватывает любую (почти) ошибку

Если внутри блока кода происходит ошибка времени выполнения, то обычно программа (блок, функция) аварийно прерывается, а текущая транзакция переходит в состояние сбоя: ее нельзя зафиксировать и можно только откатить.

Но ошибку можно перехватить и обработать. Для этого в блоке можно указать дополнительную секцию `EXCEPTION`, внутри которой перечислить условия, соответствующие ошибке, и операторы для обработки каждой такой ситуации. Эта конструкция работает аналогично `CASE`: условия просматриваются сверху вниз, выбирается первая подходящая ветвь и выполняются ее операторы.

Тонкий момент: если ошибка произойдет в секции `DECLARE` или `EXCEPTION`, то в этом блоке ее перехватить не получится.

`EXCEPTION` в целом соответствует конструкции `try-catch` в некоторых языках программирования (за исключением особенностей, связанных с обработкой транзакций, конечно).

Каждой возможной ошибке соответствует код (строка из пяти символов) и имя. В условии можно указывать как одно, так и другое. Кроме обычных ошибок можно использовать и категории, которые группируют ряд «однотипных» ошибок (например, все ошибки, связанные с нарушением ограничений целостности).

Можно также использовать специальное имя `OTHERS` для того, чтобы перехватить любую ошибку.

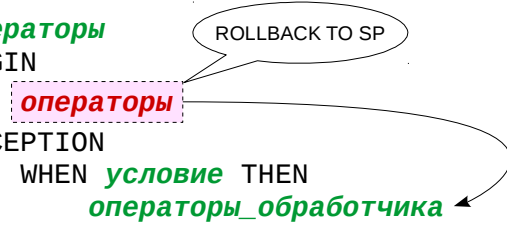
```
BEGIN
  операторы
BEGIN
  операторы
EXCEPTION
  WHEN условие THEN
    операторы_обработчика
END;
операторы
EXCEPTION
  WHEN условие THEN
    операторы_обработчика
END;
операторы
```

Рассмотрим несколько примеров. Если при выполнении блока не происходит никаких ошибок, выполняются все операторы от BEGIN до EXCEPTION. Операторы из секции EXCEPTION не выполняются.

Обработка внутри блока

Обработчик выбирается в порядке вложенности блоков

```
BEGIN
  операторы
  BEGIN
    операторы
  EXCEPTION
    WHEN условие THEN
      операторы_обработчика
  END;
  операторы
EXCEPTION
  WHEN условие THEN
    операторы_обработчика
END;
операторы
```



5

Если же при выполнении одного из операторов блока, содержащего секцию EXCEPTION, произошла ошибка, то:

- Во-первых, текущая транзакция откатывается до начала блока. Фактически, в начале блока с обработчиком неявно и автоматически ставится точка сохранения, и происходит откат до этой точки.

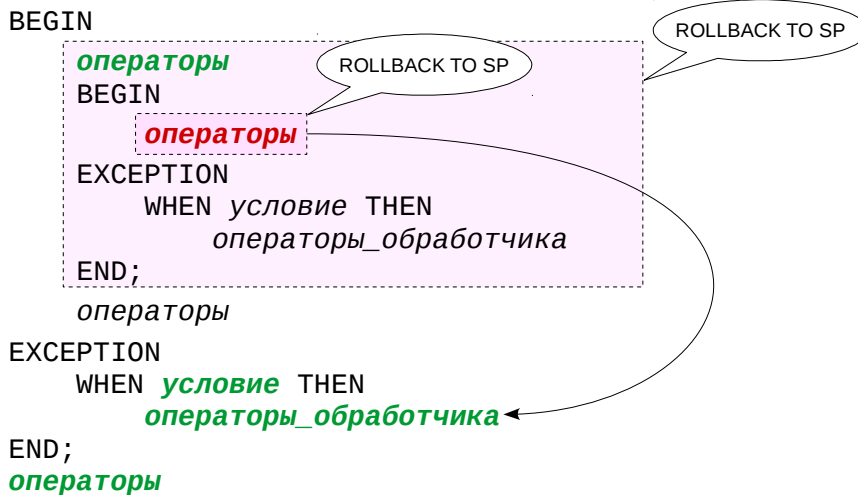
При этом состояние переменных не изменяется — откат действует только на изменения, сделанные в базе данных.

- Во-вторых, в секции EXCEPTION выбирается первый обработчик, условия которого соответствуют возникшей ошибке. Если такой обработчик найден, управление передается на соответствующие операторы.

- После того, как операторы обработчика отработают, продолжается выполнение операторов, следующих за блоком — как это происходило бы, если бы ошибка не возникла.

Обработчик выбирается в порядке вложенности блоков

```
BEGIN
  операторы
  BEGIN
    операторы
  EXCEPTION
    WHEN условие THEN
      операторы_обработчика
  END;
  операторы
EXCEPTION
  WHEN условие THEN
    операторы_обработчика
END;
операторы
```



6

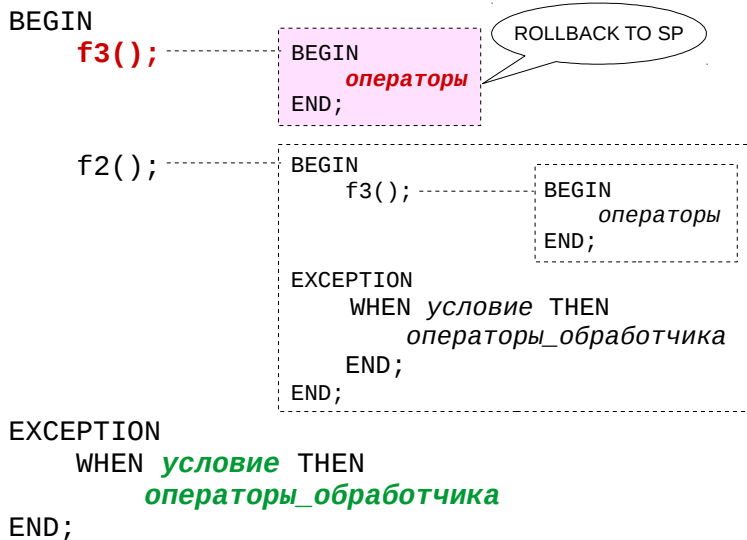
Что, если ни одно из условий, перечисленных в секции EXCEPTION блока, не работает?

Откат к точке сохранения в начале блока произойдет в любом случае. Если наш блок кода вложен в другой (объемлющий) блок, то мы «поднимемся на уровень выше» и попытаемся найти подходящий обработчик в этом блоке. Если найдем — то дальше все, как в предыдущем примере; если нет — поднимемся к следующему объемлющему блоку и так далее.

То же самое произойдет и в том случае, если блок вовсе не содержит секцию EXCEPTION; но в этом случае точка сохранения в начале блока не ставится и, соответственно, откат к ней не происходит.

Ошибка внутри функции

Обработчик выбирается в порядке вызова функций



Если мы перебрали все вложенные блоки и не обнаружили подходящего обработчика возникшей ошибки, ошибка передается тому коду, который вызвал наш блок.

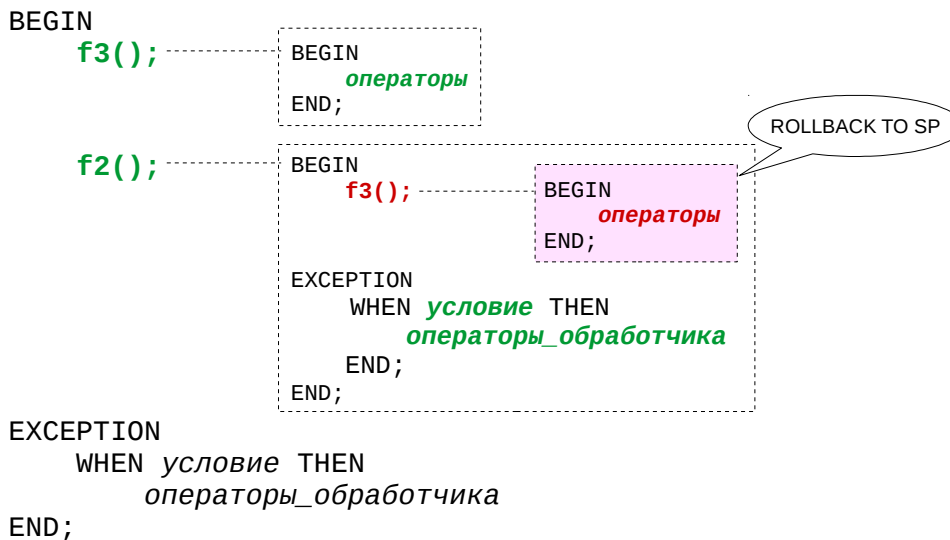
Допустим, функция `x()` вызывает функцию `y()`. При выполнении `y()` происходит ошибка и внутри `y()` она не обрабатывается. Тогда с точки зрения функции `x()` ошибка возникает в месте вызова `y()` и обрабатывается по все тем же правилам.

То есть, чтобы разобраться, в каком порядке будут просматриваться обработчики ошибки, надо проанализировать стек вызовов.

В примере на слайде ошибка возникла в функции `f3()`. Это могло произойти так: блок верхнего уровня вызвал `f3()`, которая тут же завершилась с ошибкой. Внутри `f3()` ошибка не перехватывается; значит, обработчик надо искать в блоке, который вызвал эту функцию. При этом состояние базы данных откатывается к точке сохранения в начале блока верхнего уровня.

Ошибка внутри функции

Обработчик выбирается в порядке вызова функций



8

Возможна и другая ситуация: блок верхнего уровня вызывает `f3()`, которая завершается успешно. Затем блок вызывает `f2()`, которая, в свою очередь, вызывает `f3()` — и на этот раз внутри `f3()` происходит ошибка.

Внутри `f3()` ошибка не обрабатывается. Значит, мы откатываем состояние к точке сохранения в начале блока функции `f2()` и начнем поиск обработчика в нем.

Допустим, условие в секции `EXCEPTION` выполняется. Тогда блок верхнего уровня продолжит выполнение — с его точки зрения вызов `f2()` завершился успешно.


```
db# BEGIN TRANSACTION;  
BEGIN
```

```
db# SELECT f();
```

```
BEGIN  
операторы  
END;
```

```
ERROR: query returned no rows  
CONTEXT: PL/pgSQL function f()  
         line 1 at SQL statement
```

```
db# ROLLBACK;  
ROLLBACK
```

```
журнал  
сообщений  
сервера
```

Что произойдет, если ни один из возможных обработчиков ошибки не сработает?

Во-первых, сообщение о возникшей ошибке может быть записано в журнал сообщений сервера (это зависит от настроек сервера — подробно настройки рассматриваются в теме «Отладка»).

Во-вторых, об ошибке обычно сообщается клиенту, который инициировал вызов кода в базе данных. Клиент ставится перед фактом: транзакция переходит в состояние сбоя, и единственное действие, которое с ней можно совершить — выполнить откат.

Как именно клиент будет обрабатывать возникшую ошибку, зависит от него самого. Например, `psql` выведет сообщение об ошибке и всю доступную диагностическую информацию. Клиент, ориентированный на конечного пользователя, может вывести знаменитое «обратитесь к системному администратору» и т. д.

(Если установлен параметр `exit_on_error`, то при ошибке база данных разрывает сеанс и клиенту остается только смириться с этим.)

Нет ничего плохого в том, чтобы передать возникшую ошибку клиенту. В целом, обрабатывать ошибку внутри базы данных имеет смысл только в том случае, если в возникшей ситуации мы можем сделать что-то осмысленное (например, повторить операцию при ошибке сериализации).



Поиск обработчика ошибки происходит «изнутри наружу»
в порядке вложенности блоков и вызова функций

Не перехваченная ошибка приводит к откату транзакции

В начале блока с EXCEPTION устанавливается неявная точка
сохранения; при ошибке происходит откат к этой точке

Обработка ошибок связана с накладными расходами

Сообщения отправляются клиенту и в журнал сервера



1. Если при добавлении новой книги указать одного и того же автора несколько раз, произойдет ошибка.
Измените функцию `add_book`: перехватите ошибку нарушения уникальности и вместо нее вызовите ошибку с понятным текстовым сообщением.
2. Проверьте изменения в приложении.

1. Чтобы определить название ошибки, которую необходимо перехватить, перехватите все ошибки (WHEN OTHERS) и выведите необходимую информацию (вызвав новую ошибку с соответствующим текстом).

После этого не забудьте заменить WHEN OTHERS на конкретную ошибку — пусть остальные типы ошибок обрабатываются на более высоком уровне, раз в этом месте кода нет возможности сделать что-то конструктивное.

(В реальной жизни не стоило бы обрабатывать и нарушение уникальности — лучше было бы изменить приложение так, чтобы оно не позволяло указывать двух одинаковых авторов.)

1. Ряд языков имеет конструкцию `try ... catch ... finally ...`, в которой `try` соответствует `BEGIN`, `catch` — `EXCEPTION`, а операторы из блока `finally` срабатывают всегда, независимо от того, возникло ли исключение и было ли оно обработано блоком `catch`. Предложите способ добиться подобного эффекта в PL/pgSQL.
2. Сравните стеки вызовов, получаемые конструкциями `GET STACKED DIAGNOSTICS` с элементом `pg_exception_context` и `GET [CURRENT] DIAGNOSTICS` с элементом `pg_context`.
3. Напишите функцию `getstack`, возвращающую текущий стек вызовов в виде массива строк. Сама функция `getstack` не должна фигурировать в стеке.

1. Самый простой способ — просто повторить операторы `finally` в нескольких местах. Однако попробуйте построить такую конструкцию, чтобы эти операторы можно было написать ровно один раз.