Практика №16

1. Напишите функцию add_author для добавления новых авторов. Функция должна принимать три параметра (фамилия, имя, отчество) и возвращать идентификатор нового автора.

Проверьте, что приложение позволяет добавлять авторов.

2. Напишите функцию buy_book для покупки книги. Функция принимает идентификатор книги и уменьшает количество таких книг на складе на единицу. Возвращаемое значение отсутствует.

Проверьте, что в «Магазине» появилась возможность покупки книг.

Решение

1.

FUNCTION add_author(last_name text, first_name text, surname text)
RETURNS integer

3.

FUNCTION buy_book(book_id integer)
RETURNS void

Вы можете обратить внимание, что при покупке книг приложение позволяет «уйти в минус». Если бы количество книг хранилось в столбце, простым и хорошим решением было бы сделать ограничение CHECK. Но в нашем случае количество рассчитывается, и мы отложим написание проверки до темы «Триггеры».

1. Функция add_author

```
=> CREATE OR REPLACE FUNCTION add_author(
    last_name text,
    first_name text,
    middle_name text
) RETURNS integer
AS $$
DECLARE
    author_id integer;
```

```
BEGIN
   INSERT INTO authors(last_name, first_name, middle_name)
      VALUES (last_name, first_name, middle_name)
      RETURNING authors.author_id INTO author_id;
   RETURN author_id;
END;


$$ VOLATILE LANGUAGE plpgsql;
```

CREATE FUNCTION

2. Функция buy_book

```
=> CREATE OR REPLACE FUNCTION buy_book(book_id integer)
RETURNS void
AS $$
BEGIN
   INSERT INTO operations(book_id, qty_change)
      VALUES (book_id, -1);
END;
$$ VOLATILE LANGUAGE plpgsql;
```

CREATE FUNCTION

Практика №17

Напишите игру, в которой сервер пытается угадать загаданное пользователем животное, задавая последовательные уточняющие вопросы, на которые можно отвечать «да» или «нет».

Если сервер предложил неправильный вариант, он запрашивает у пользователя имя животного и отличающий вопрос. Эта новая информация запоминается и используется в следующих играх.

1. Создайте таблицу для представления информации.

2. Придумайте интерфейс и реализуйте необходимые функции.

3. Проверьте реализацию.

Решение

Пример диалога (между людьми):

— Это млекопитающее?	— Да.
— Это слон?	— Нет.
— Сдаюсь. Кто это?	— Кит.
— Как отличить кита от слона?	— Он живет в воде.

1. Информацию удобно представить в виде двоичного дерева. Внутренние узлы хранят вопросы, листовые узлы — названия животных. Один из дочерних узлов соответствует ответу «да», другой — ответу «нет».

2. Между вызовами функций надо передавать информацию о том, на каком узле дерева мы остановились («контекст» диалога). Функции могут быть, например, такими:

- начать игру (нет входного контекста)

```
FUNCTION start_game(OUT context integer, OUT question text)
```

- продолжение игры (получаем ответ, выдаем следующий вопрос)

```
FUNCTION continue_game(
    INOUT context integer, IN answer boolean,
    OUT you_win boolean, OUT question text)
```

- завершение игры (внесение информации о новом животном)

```
FUNCTION end_game(
    IN context integer, IN name text, IN question text)


RETURNS void
```

1. Таблица

```
=> CREATE TABLE animals(
    id     integer PRIMARY KEY GENERATED ALWAYS AS IDENTITY,
    yes_id integer REFERENCES animals(id),
    no_id  integer REFERENCES animals(id),
    name   text
);
```

```
CREATE TABLE
```

```
=> INSERT INTO animals(name) VALUES
    ('млекопитающее'), ('слон'), ('черепаха');
```

```
INSERT 0 3
```

```
=> UPDATE animals SET yes_id = 2, no_id = 3 WHERE id = 1;
```

```
UPDATE 1
```

```
=> SELECT * FROM animals ORDER BY id;
```

```
 id | yes_id | no_id |     name
----+--------+-------+--------------
1 | 2 | 3 |
(3 rows)
```

Первая строка считается корнем дерева.

2. Функции

```
=> CREATE FUNCTION start_game(
    OUT context integer,
    OUT question text
)
AS $$
DECLARE
```

```
    root_id CONSTANT integer := 1;
BEGIN
    SELECT id, name||'?'
    INTO context, question
    FROM animals
    WHERE id = root_id;
END;
$$ LANGUAGE plpgsql;

CREATE FUNCTION

=> CREATE FUNCTION continue_game(
    INOUT context integer,
    IN answer boolean,
    OUT you_win boolean,
    OUT question text
)

AS $$

DECLARE

    new_context integer;

BEGIN
    SELECT CASE WHEN answer THEN yes_id ELSE no_id END
    INTO new_context
    FROM animals
    WHERE id = context;
    IF new_context IS NULL THEN
        you_win := NOT answer;
        question := CASE
            WHEN you_win THEN 'Сдаюсь'
            ELSE 'Вы проиграли'
        END;
    ELSE
        SELECT id, null, name||'?'
        INTO context, you_win, question
        FROM animals
        WHERE id = new_context;

END IF;
END;
$$ LANGUAGE plpgsql;

CREATE FUNCTION
```

```
=> CREATE FUNCTION end_game(
   IN context integer,
   IN name text,
   IN question text
) RETURNS void
AS $$
DECLARE
   new_animal_id integer;
   new_question_id integer;
BEGIN
   INSERT INTO animals(name) VALUES (name)
      RETURNING id INTO new_animal_id;
   INSERT INTO animals(name) VALUES (question)
      RETURNING id INTO new_question_id;
   UPDATE animals SET yes_id = new_question_id
   WHERE yes_id = context;
   UPDATE animals SET  no_id = new_question_id
   WHERE  no_id = context;
   UPDATE animals SET yes_id = new_animal_id, no_id = context
   WHERE id = new_question_id;
END;
$$ LANGUAGE plpgsql;

CREATE FUNCTION
```

3. Пример сеанса игры

Загадываем слово «кит».

```
=> SELECT * FROM start_game();
 context |    question
---------+---------------
       1 | млекопитающее?
(1 row)

=> SELECT * FROM continue_game(1,true);
 context | you_win | question
---------+---------+----------
       2 |         | слон?
(1 row)

=> SELECT * FROM continue_game(2,false);

 context | you_win | question
---------+---------+----------
       2 | t       | Сдаюсь
```

```
(1 row)

=> SELECT * FROM end_game(2,'кит','живет в воде');

end_game

----------
(1 row)
```

Теперь в таблице:

```
=> SELECT * FROM animals ORDER BY id;
 id | yes_id | no_id |     name
----+--------+-------+--------------
```

1. 1| 5 |
1. 2| |
1. 3| |
1. 4| |
1. 5| 4 |

```
(5 rows)

3 | млекопитающее
  | слон

| черепаха

  | кит
2 | живет в воде
```

Снова загадали «кит».

```
=> SELECT * FROM start_game();
 context |    question
---------+----------------
       1 | млекопитающее?
(1 row)

=> SELECT * FROM continue_game(1,true);
 context | you_win |   question
---------+---------+--------------
       5 |         | живет в воде?
(1 row)
```

```
=> SELECT * FROM continue_game(5,true);
 context | you_win | question
---------+---------+----------
       4 |         | кит?
(1 row)

=> SELECT * FROM continue_game(4,true);
 context | you_win |   question
---------+---------+--------------
       4 | f       | Вы проиграли
(1 row)
```

Практика №18

1. Измените функцию book_name: если у книги больше двух авторов, то в названии указываются только первые два
и в конце добавляется «и др.».
Проверьте работу функции в SQL и в приложении.
2. Попробуйте написать функцию book_name на SQL. Какой вариант нравится больше — PL/pgSQL или SQL?

Решение

1. Например:

Хрестоматия. Пушкин А. С., Толстой Л. Н., Тургенев И. С. →

→ Хрестоматия. Пушкин А. С., Толстой Л. Н. и др.

1. Функция book_name (сокращение авторов)

Напишем более универсальную функцию с дополнительным параметром — максимальное число авторов в названии.

Поскольку функция меняет сигнатуру (число и/или типы входных параметров), ее необходимо сначала удалить, а потом создать заново. В данном случае у функции есть зависимый объект — представление catalog_v, в котором она используется. Представление тоже придется пересоздать (в реальной работе все эти действия надо выполнять в одной транзакции, чтобы изменения вступили в силу атомарно).

```
=> DROP FUNCTION book_name(integer,text) CASCADE;
```

```
NOTICE: drop cascades to view catalog_v

DROP FUNCTION

=> CREATE OR REPLACE FUNCTION book_name(
   book_id integer,
   title text,
   maxauthors integer DEFAULT 2
)
RETURNS text
AS $$
DECLARE

   r record;

   res text;
BEGIN
   res := shorten(title) || '. ';
   FOR r IN (
      SELECT a.last_name, a.first_name, a.middle_name, ash.seq_num
      FROM   authors a
           JOIN authorship ash ON a.author_id = ash.author_id
      WHERE  ash.book_id = book_name.book_id
      ORDER BY ash.seq_num

)

   LOOP
      EXIT WHEN r.seq_num > maxauthors;
      res := res || author_name(r.last_name, r.first_name, r.middle_name) || ', ';
   END LOOP;
   res := rtrim(res, ', ');
   IF r.seq_num > maxauthors THEN
      res := res || ' и др.';
   END IF;
   RETURN res;
END;

$$ STABLE LANGUAGE plpgsql;

CREATE FUNCTION

=> CREATE OR REPLACE VIEW catalog_v AS
SELECT b.book_id,
      b.title,
      b.onhand_qty,
      book_name(b.book_id, b.title) AS display_name,
```

```
      b.authors
FROM   books b
ORDER BY display_name;

CREATE VIEW

=> SELECT book_id, display_name FROM catalog_v;

 book_id |                  display_name
---------+-----------------------------------------------------
       4 | Война и мир. Толстой Л. Н.
       2 | Муму. Тургенев И. С.
       5 | Путешествия в некоторые удаленные страны.... Свифт Д.
       1 | Сказка о царе Салтане. Пушкин А. С.
       3 | Трудно быть богом. Стругацкий А. Н., Стругацкий Б. Н.
       6 | Хрестоматия. Пушкин А. С., Толстой Л. Н. и др.


(6 rows)
```

2. Вариант на чистом SQL

```
=> CREATE OR REPLACE FUNCTION book_name(
   book_id integer,
   title text,
   maxauthors integer DEFAULT 2
)
RETURNS text
AS $$
SELECT shorten(book_name.title) ||
      '. ' ||
      string_agg(
         author_name(a.last_name, a.first_name, a.middle_name), ', '
         ORDER BY ash.seq_num
      ) FILTER (WHERE ash.seq_num <= maxauthors) ||
      CASE
         WHEN max(ash.seq_num) > maxauthors THEN ' и др.'

       ELSE ''

       END

FROM   authors a
      JOIN authorship ash ON a.author_id = ash.author_id
WHERE  ash.book_id = book_name.book_id;
$$ STABLE LANGUAGE sql;

CREATE FUNCTION
```

```
=> SELECT book_id, display_name FROM catalog_v;
 book_id |                    display_name
---------+-----------------------------------------------------
       4 | Война и мир. Толстой Л. Н.
       2 | Муму. Тургенев И. С.
       5 | Путешествия в некоторые удаленные страны.... Свифт Д.
       1 | Сказка о царе Салтане. Пушкин А. С.
       3 | Трудно быть богом. Стругацкий А. Н., Стругацкий Б. Н.
       6 | Хрестоматия. Пушкин А. С., Толстой Л. Н. и др.

(6 rows)
```

Практика №19

1. Требуется распределить расходы на электроэнергию

по отделам компании пропорционально количеству сотрудников (перечень отделов находится в таблице). Напишите функцию, которая примет общую сумму расходов и запишет распределенные расходы в строки таблицы. Числа округляются до копеек; сумма расходов всех отделов должна в точности совпадать с общей суммой.

2. Напишите табличную функцию, имитирующую сортировку слиянием. Функция принимает две курсорные переменные; оба курсора уже открыты и возвращают упорядоченные по неубыванию целые числа. Требуется выдать общую упорядоченную последовательность чисел из обоих источников.

Решение

1. В качестве таблицы можно взять:

```
CREATE TABLE depts(
    id integer PRIMARY KEY GENERATED ALWAYS AS IDENTITY,
    employees integer,
    expenses numeric(10,2)
);
INSERT INTO depts(employees) VALUES (10),(10),(10);
```

Функция:

```
FUNCTION distribute_expenses(amount numeric) RETURNS void;
```

Ожидаемый результат после вызова функции с параметром 100.00:

```
 expenses
----------
  33.33
  33.34
  33.33
```

2. Функция:

FUNCTION merge(c1 refcursor, c2 refcursor) RETURNS SETOF integer;

Например, если первый курсор возвращает последовательность 1, 3, 5, а второй — 2, 3, 4, то ожидается результат:

merge

1

2

3

3

4

5

1. Распределение расходов

```
=> CREATE DATABASE plpgsql_cursors;
```

CREATE DATABASE

```
=> \c plpgsql_cursors
```

You are now connected to database "plpgsql_cursors" as user "student".Таблица:

```
=> CREATE TABLE depts(
   id integer PRIMARY KEY GENERATED ALWAYS AS IDENTITY,
   employees integer,
   expenses numeric(10,2)
);
```

CREATE TABLE

```
=> INSERT INTO depts(employees) VALUES (10),(10),(10);
```

INSERT 0 3

Функция:

```
=> CREATE FUNCTION distribute_expenses(amount numeric) RETURNS void
```

```
AS $$
DECLARE
   depts_cur CURSOR FOR
      SELECT employees FROM depts FOR UPDATE;
   total_employees numeric;
   expense numeric;
   rounding_err numeric := 0.0;
   cent numeric;
BEGIN
   SELECT sum(employees) FROM depts INTO total_employees;
   FOR dept IN depts_cur LOOP
      expense := amount * (dept.employees / total_employees);
      rounding_err := rounding_err + (expense - round(expense,2));
      cent := round(rounding_err,2);
      expense := expense + cent;
      rounding_err := rounding_err - cent;
      UPDATE depts SET expenses = round(expense,2)
      WHERE CURRENT OF depts_cur;
   END LOOP;
END;
$$ VOLATILE LANGUAGE plpgsql;
```

CREATE FUNCTION

Проверка:

```
=> SELECT distribute_expenses(100.00);
```

distribute_expenses

(1 row)

```
=> SELECT * FROM depts;
```

```
 id | employees | expenses
----+-----------+----------
```

1 | 2 | 3 |

(3 rows)

10 | 10 | 10 |

33.33
33.34
33.33

Разумеется, возможны и другие алгоритмы, например, перенос всех ошибок округления на одну строку и т. п. В курсе DEV2 рассматривается другое решение этой задачи с помощью пользовательских агрегатных функций.

2. Слияние отсортированных наборов

Эта реализация предполагает, что числа не могут иметь неопределенные значения NULL.

```
=> CREATE FUNCTION merge(c1 refcursor, c2 refcursor)
RETURNS SETOF integer
AS $$
DECLARE

a integer;

   b integer;
BEGIN
   FETCH c1 INTO a;
   FETCH c2 INTO b;
   LOOP
      EXIT WHEN a IS NULL AND b IS NULL;
      IF a < b OR b IS NULL THEN
         RETURN NEXT a;
         FETCH c1 INTO a;
      ELSE
         RETURN NEXT b;
         FETCH c2 INTO b;
      END IF;
   END LOOP;
END;

$$ VOLATILE LANGUAGE plpgsql;
```

CREATE FUNCTION

Проверяем.

```
=> BEGIN;
```

BEGIN

```
=> DECLARE c1 CURSOR FOR
   SELECT * FROM (VALUES (1),(3),(5)) t;
```

DECLARE CURSOR

```
=> DECLARE c2 CURSOR FOR
   SELECT * FROM (VALUES (2),(3),(4)) t;
```

DECLARE CURSOR

```
=> SELECT * FROM merge('c1','c2');
```

```
 merge
-------

1 2 3 3 4 5

(6 rows)
```

```
=> COMMIT;
```

COMMIT

Практика №20

1. Измените функцию get_catalog так, чтобы запрос к представлению catalog_v формировался динамически и содержал условия только на те поля, которые заполнены на форме поиска в «Магазине».

Убедитесь, что реализация не допускает возможности внедрения SQL-кода.

Проверьте работу функции в приложении.

Решение

1. Например, если на форме поиска не заполнены поля «Название книги» и «Имя автора», но установлен флажок «Есть на складе», должен быть сформирован запрос вида:

SELECT ... FROM catalog_v WHERE onhand_qty > 0;

Учтите, что поиск при такой реализации вовсе не обязательно будет работать эффективнее, но поддерживать его совершенно точно будет сложнее. Поэтому в реальной работе не стоит прибегать к такому приему, если для того нет веских оснований.

1. Функция get_catalog

```
=> CREATE OR REPLACE FUNCTION get_catalog(
    author_name text,
    book_title text,
    in_stock boolean
)
RETURNS TABLE(book_id integer, display_name text, onhand_qty integer)
AS $$
DECLARE
    title_cond text := '';
    author_cond text := '';
    qty_cond text := '';
BEGIN
    IF book_title != '' THEN
        title_cond := format(
            ' AND cv.title ILIKE %L', '%'||book_title||'%'
```

```
);

END IF;

   IF author_name != '' THEN
      author_cond := format(
         ' AND cv.authors ILIKE %L', '%'||author_name||'%'
      );
   END IF;
   IF in_stock THEN
      qty_cond := ' AND cv.onhand_qty > 0';
   END IF;
   RETURN QUERY EXECUTE '
      SELECT cv.book_id,
            cv.display_name,
            cv.onhand_qty
      FROM   catalog_v cv
      WHERE  true'
      || title_cond || author_cond || qty_cond || '
      ORDER BY display_name';
END;
$$ STABLE LANGUAGE plpgsql;
```

CREATE FUNCTION

Практика №21

1. Создайте функцию, которая возвращает строки матричного отчета по функциям в базе данных.

Столбцы должны содержать имена владельцев функций, строки — названия схем, а ячейки — количество функций данного владельца в данной схеме.

Как можно вызвать такую функцию?

1. Примерный вид результата:

schema total postgres student ...

information_schema 12 12 0

pg_catalog 2811 2811 0

public 3 0 3

 ...

Количество столбцов в запросе заранее не известно. Поэтому необходимо сконструировать запрос и затем динамически его выполнить. Текст запроса может быть таким:

```
SELECT pronamespace::regnamespace::text AS schema,
    COUNT(*) AS total
    ,SUM(CASE WHEN proowner = 10 THEN 1 ELSE 0 END) postgres
    ,SUM(CASE WHEN proowner = 16384 THEN 1 ELSE 0 END) student
FROM pg_proc
GROUP BY pronamespace::regnamespace
ORDER BY schema
```

Выделенные строки — динамическая часть — необходимо сформировать дополнительным запросом. Начало и конец запроса — статические.

Столбец proowner имеет тип oid, для получения имени владельца можно воспользоваться конструкцией proowner::regrole::text.

Получение матричного отчета

```
=> CREATE DATABASE plpgsql_dynamic;
```

CREATE DATABASE

```
=> \c plpgsql_dynamic
```

You are now connected to database "plpgsql_dynamic" as user "student".Вспомогательная функция для формирования текста динамического запроса:

```
=> CREATE FUNCTION form_query() RETURNS text
AS $$
DECLARE
   query_text text;
   columns text := '';
   r record;

BEGIN

   -- Статическая часть запроса
   -- Первые два столбца: имя схемы и общее количество функций в ней
   query_text :=
$query$
SELECT pronamespace::regnamespace::text AS schema
    , count(*) AS total{{columns}}
FROM pg_proc
GROUP BY pronamespace::regnamespace
ORDER BY schema
$query$;
   -- Динамическая часть запроса
   -- Получаем список владельцев функций, для каждого - отдельный столбец
   FOR r IN SELECT DISTINCT proowner AS owner FROM pg_proc ORDER BY 1
   LOOP
      columns := columns || format(
         E'\n    , sum(CASE WHEN proowner = %s THEN 1 ELSE 0 END) AS %I',
         r.owner,
         r.owner::regrole

);

END LOOP;

   RETURN replace(query_text, '{{columns}}', columns);
END;

$$ STABLE LANGUAGE plpgsql;
```

CREATE FUNCTION

Итоговый текст запроса:

```
=> SELECT form_query();

                        form_query
------------------------------------------------------------------
                                                  +
 SELECT pronamespace::regnamespace::text AS schema                +
     , count(*) AS total                                  +
     , sum(CASE WHEN proowner = 10 THEN 1 ELSE 0 END) AS postgres  +
     , sum(CASE WHEN proowner = 16384 THEN 1 ELSE 0 END) AS student+
 FROM pg_proc                                             +
 GROUP BY pronamespace::regnamespace                            +
 ORDER BY schema                                        +


(1 row)
```

Теперь создаем функцию для матричного отчета:

```
=> CREATE FUNCTION matrix() RETURNS SETOF record
AS $$
BEGIN
   RETURN QUERY EXECUTE form_query();
END;

$$ STABLE LANGUAGE plpgsql;
```

CREATE FUNCTION

Простое выполнение запроса приведет к ошибке, так как не указана структура возвращаемых записей:

```
=> SELECT * FROM matrix();

ERROR:  a column definition list is required for functions returning "record"
LINE 1: SELECT * FROM matrix();

^
```

В этом состоит важное ограничение на использование функций, возвращающих произвольную выборку. В момент вызова необходимо знать и указать структуру возвращаемой записи.

В общем случае структура возвращаемой записи может быть неизвестна, но, применительно к нашему матричному отчету, можно выполнить еще один запрос, который покажет, как правильно вызвать функцию matrix.

Подготовим текст запроса:

```
=> CREATE FUNCTION matrix_call() RETURNS text
AS $$
DECLARE

cmd text;

   r record;
BEGIN
   cmd := 'SELECT * FROM matrix() AS m(
      schema text, total bigint';
   FOR r IN SELECT DISTINCT proowner AS owner FROM pg_proc ORDER BY 1
   LOOP
      cmd := cmd || format(', %I bigint', r.owner::regrole::text);
   END LOOP;
   cmd := cmd || E'\n)';
   RAISE NOTICE '%', cmd;
   RETURN cmd;
END;
$$ STABLE LANGUAGE plpgsql;
```

```
CREATE FUNCTION
```

Теперь мы можем первым запросом получить структуру матричного отчета, а вторым запросом его сформировать (и все это — одной командой psql):

```
=> BEGIN ISOLATION LEVEL REPEATABLE READ;
```

```
BEGIN
```

```
=> SELECT matrix_call() \gexec
```

```
NOTICE:  SELECT * FROM matrix() AS m(
      schema text, total bigint, postgres bigint, student bigint

)

      schema       | total | postgres | student
-------------------+-------+----------+---------


information_schema | 12 | 12 | 0 pg_catalog | 2948 | 2948 | 0 public |3|0|3
```

(3 rows)

```
=> COMMIT;
```

```
 COMMIT
```

Матричный отчет корректно формируется.

Уровень изоляции Repeatable Read гарантирует, что отчет сформируется, даже если между двумя запросами появится функция у нового владельца.
Можно было бы и напрямую выполнить запрос, возвращаемый функцией form_query. Но задача получить в клиентском приложении список возвращаемых столбцов все равно останется. Функция matrix_call показывает, как ее можно решить дополнительным запросом.

Еще один вариант решения - вместо набора записей произвольной структуры возвращать набор строк слабоструктурированного типа (такого, как JSON или XML).

Практика №22

1. Создайте функцию add_book для добавления новой книги.

Функция должна принимать два параметра — название книги и массив идентификаторов авторов — и возвращать идентификатор новой книги.

Проверьте, что в приложении появилась возможность добавлять книги.

Решение

1.

FUNCTION add_book(title text, authors integer[])

RETURNS integer

1. Функция add_book

```
=> CREATE OR REPLACE FUNCTION add_book(title text, authors integer[])
RETURNS integer
AS $$
DECLARE
   book_id integer;
   id integer;
   seq_num integer := 1;
BEGIN
   INSERT INTO books(title)
      VALUES(title)
      RETURNING books.book_id INTO book_id;
   FOREACH id IN ARRAY authors LOOP
      INSERT INTO authorship(book_id, author_id, seq_num)
         VALUES (book_id, id, seq_num);
      seq_num := seq_num + 1;
   END LOOP;
   RETURN book_id;
END;

$$ VOLATILE LANGUAGE plpgsql;
```

CREATE FUNCTION

Практика №23

1. Реализуйте функцию map, принимающую два параметра: массив вещественных чисел и название вспомогательной функции, принимающей один параметр вещественного типа.

Функция возвращает массив, полученный из исходного применением вспомогательной функции к каждому элементу.

2. Реализуйте функцию reduce, принимающую два параметра: массив вещественных чисел и название вспомогательной функции, принимающей два параметра вещественного типа.

Функция возвращает вещественное число, полученное последовательной сверткой массива слева направо.

3. Сделайте функции map и reduce полиморфными.

Решение

1. Например:

map(ARRAY[4.0,9.0],'sqrt') → ARRAY[2.0,3.0]

2. Например:

reduce(ARRAY[1.0,3.0,2.0,0.5],'greatest') → 3.0

В этом случае значение вычисляется как

greatest(greatest(greatest(1.0,3.0), 2.0), 0.5)

1. Функция map

```
=> CREATE DATABASE plpgsql_arrays;
```

```
CREATE DATABASE
```

```
=> \c plpgsql_arrays
```

```
You are now connected to database "plpgsql_arrays" as user "student".
```

```
=> CREATE FUNCTION map(a INOUT float[], func text)
AS $$
DECLARE

i integer;

   x float;
BEGIN
   IF cardinality(a) > 0 THEN
      FOR i IN array_lower(a,1)..array_upper(a,1) LOOP
         EXECUTE format('SELECT %I($1)',func) USING a[i] INTO x;
         a[i] := x;
      END LOOP;

END IF;

END;

$$ IMMUTABLE LANGUAGE plpgsql;
```

```
CREATE FUNCTION
```

INTO a[i] не работает, поэтому нужна отдельная переменная.

```
=> SELECT map(ARRAY[4.0,9.0,16.0],'sqrt');

   map
---------
 {2,3,4}

(1 row)
```

```
=> SELECT map(ARRAY[]::float[],'sqrt');

map -----

{}
 (1 row)
```

Другой вариант реализации с циклом FOREACH:

```
=> CREATE OR REPLACE FUNCTION map(a float[], func text) RETURNS float[]
AS $$
DECLARE

x float;

   b float[]; -- пустой массив
```

```
BEGIN
   FOREACH x IN ARRAY a LOOP
      EXECUTE format('SELECT %I($1)',func) USING x INTO x;
      b := b || x;

END LOOP;

   RETURN b;
END;

$$ IMMUTABLE LANGUAGE plpgsql;


 CREATE FUNCTION


 => SELECT map(ARRAY[4.0,9.0,16.0],'sqrt');

   map
---------
 {2,3,4}

(1 row)

=> SELECT map(ARRAY[]::float[],'sqrt');map

----- (1 row)
```

2. Функция reduce

```
=> CREATE FUNCTION reduce(a float[], func text) RETURNS float
AS $$
DECLARE

i integer;

   r float := NULL;
BEGIN
   IF cardinality(a) > 0 THEN
      r := a[array_lower(a,1)];
      FOR i IN array_lower(a,1)+1 .. array_upper(a,1) LOOP
         EXECUTE format('SELECT %I($1,$2)',func) USING r, a[i]
            INTO r;
      END LOOP;
   END IF;
   RETURN r;
END;
$$ IMMUTABLE LANGUAGE plpgsql;
```

CREATE FUNCTION

Greatest (как и least) — не функция, а встроенное условное выражение, поэтому из-за экранирования не получится использовать ее напрямую:

```
=> SELECT reduce( ARRAY[1.0,3.0,2.0], 'greatest');
```

```
ERROR:  function greatest(double precision, double precision) does not exist
LINE 1: SELECT "greatest"($1,$2)
               ^
HINT:  No function matches the given name and argument types. You might need to add explicit type casts.
QUERY:  SELECT "greatest"($1,$2)
CONTEXT:  PL/pgSQL function reduce(double precision[],text) line 9 at EXECUTE
```

Вместо нее используем реализованную в демонстрации функцию maximum.

```
=> CREATE FUNCTION maximum(VARIADIC a anyarray, maxsofar OUT anyelement)
AS $$
DECLARE
   x maxsofar%TYPE;
BEGIN
   FOREACH x IN ARRAY a LOOP
      IF x IS NOT NULL AND (maxsofar IS NULL OR x > maxsofar) THEN
         maxsofar := x;
      END IF;
   END LOOP;
END;
$$ IMMUTABLE LANGUAGE plpgsql;
```

```
 CREATE FUNCTION
```

```
 => SELECT reduce(ARRAY[1.0,3.0,2.0], 'maximum');
```

```
 reduce
--------
```

```
3 (1 row)
```

```
=> SELECT reduce(ARRAY[1.0], 'maximum');
```

```
 reduce
--------
```

1 (1 row)

```
=> SELECT reduce(ARRAY[]::float[], 'maximum');
```

```
reduce

--------
(1 row)
```

Вариант с циклом FOREACH:

```
=> CREATE OR REPLACE FUNCTION reduce(a float[], func text) RETURNS float
AS $$
DECLARE
   x float;
   r float;
   first boolean := true;
BEGIN
   FOREACH x IN ARRAY a LOOP
      IF first THEN
         r := x;
         first := false;
      ELSE
         EXECUTE format('SELECT %I($1,$2)',func) USING r, x INTO r;
      END IF;

END LOOP;

   RETURN r;
END;

$$ IMMUTABLE LANGUAGE plpgsql;
```

```
 CREATE FUNCTION
```

```
 => SELECT reduce(ARRAY[1.0,3.0,2.0], 'maximum');

 reduce
--------

3 (1 row)

=> SELECT reduce(ARRAY[1.0], 'maximum');

 reduce
--------
```

```
1 (1 row)

=> SELECT reduce(ARRAY[]::float[], 'maximum');reduce

--------
(1 row)
```

3. Полиморфные варианты функций

Функция map.

```
=> DROP FUNCTION map(float[],text);

DROP FUNCTION

=> CREATE FUNCTION map(
   a anyarray,

func text,

   elem anyelement DEFAULT NULL
)
RETURNS anyarray
AS $$
DECLARE

x elem%TYPE;

   b a%TYPE;
BEGIN
   FOREACH x IN ARRAY a LOOP
      EXECUTE format('SELECT %I($1)',func) USING x INTO x;
      b := b || x;

END LOOP;

   RETURN b;
END;

$$ IMMUTABLE LANGUAGE plpgsql;

CREATE FUNCTION
```

Требуется фиктивный параметр типа anyelement, чтобы внутри функции объявить переменную такого же типа.

```
=> SELECT map(ARRAY[4.0,9.0,16.0],'sqrt');
```

```
                          map
-----------------------------------------------------
 {2.000000000000000,3.000000000000000,4.000000000000000}


(1 row)
```

```
=> SELECT map(ARRAY[]::float[],'sqrt');
```

```
map

----- (1 row)
```

Пример вызова с другим типом данных:

```
=> SELECT map(ARRAY[' a ','  b','c  '],'btrim');
   map
---------
 {a,b,c}


(1 row)
```

Функция reduce.

```
=> DROP FUNCTION reduce(float[],text);
DROP FUNCTION
=> CREATE FUNCTION reduce(
   a anyarray,

func text,

   elem anyelement DEFAULT NULL
)
RETURNS anyelement
AS $$
DECLARE
   x elem%TYPE;
   r elem%TYPE;
   first boolean := true;
BEGIN
   FOREACH x IN ARRAY a LOOP
      IF first THEN
         r := x;
         first := false;
      ELSE
         EXECUTE format('SELECT %I($1,$2)',func) USING r, x INTO r;
      END IF;
```

```
END LOOP;

   RETURN r;
END;

$$ IMMUTABLE LANGUAGE plpgsql;
```

CREATE FUNCTION

```
=> CREATE FUNCTION add(x anyelement, y anyelement) RETURNS anyelement
AS $$
BEGIN
   RETURN x + y;
END;

$$ IMMUTABLE LANGUAGE plpgsql;
```

CREATE FUNCTION

```
=> SELECT reduce(ARRAY[1,-2,4], 'add');
```

```
 reduce
--------
```

3 (1 row)

```
=> SELECT reduce(ARRAY['a','b','c'], 'concat');
```

```
 reduce
--------
```

abc
(1 row)

Практика №24

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. Проверьте изменения в приложении.

Решение

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. Обработка повторяющихся авторов при добавлении книги

```
=> CREATE OR REPLACE FUNCTION add_book(title text, authors integer[])
RETURNS integer
AS $$
DECLARE
   book_id integer;
   id integer;
   seq_num integer := 1;
BEGIN
   INSERT INTO books(title)
      VALUES(title)
      RETURNING books.book_id INTO book_id;
   FOREACH id IN ARRAY authors LOOP
      INSERT INTO authorship(book_id, author_id, seq_num)
         VALUES (book_id, id, seq_num);
      seq_num := seq_num + 1;
```

```
    END LOOP;
    RETURN book_id;
EXCEPTION
    WHEN unique_violation THEN
        RAISE EXCEPTION 'Один и тот же автор не может быть указан дважды';
END;
$$ VOLATILE LANGUAGE plpgsql;
```

CREATE FUNCTION

Практика №25

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.

Решение

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.

1. Try-catch-finally

```
=> CREATE DATABASE plpgsql_exceptions;
```

```
CREATE DATABASE
```

```
=> \c plpgsql_exceptions
```

```
You are now connected to database "plpgsql_exceptions" as user "student".
```

Сложность состоит в том, что операторы finally должны выполняться всегда, даже в случае возникновения ошибки в операторах catch (блок EXCEPTION).

Решение может использовать два вложенных блока и фиктивное исключение, которое вызывается при нормальном завершении внутреннего блока. Это дает возможность не дублировать операторы finally, поместив их в обработчик ошибок внешнего блока.

```
=> DO $$
BEGIN
```

```
        BEGIN
                RAISE NOTICE 'Операторы try';

                --

                RAISE NOTICE '...нет исключения';

        EXCEPTION

                WHEN no_data_found THEN

                        RAISE NOTICE 'Операторы catch';

        END;

        RAISE SQLSTATE 'ALLOK';

EXCEPTION
        WHEN others THEN
                RAISE NOTICE 'Операторы finally';

                IF SQLSTATE != 'ALLOK' THEN

                        RAISE;
                END IF;

END;

$$;

NOTICE: Операторы try

NOTICE: ...нет исключения

NOTICE: Операторы finally

DO


=> DO $$

BEGIN

        BEGIN

                RAISE NOTICE 'Операторы try';

                --

                RAISE NOTICE '...исключение, которое обрабатывается';
```

```
            RAISE no_data_found;
        EXCEPTION
            WHEN no_data_found THEN
                RAISE NOTICE 'Операторы catch';
        END;
        RAISE SQLSTATE 'ALLOK';
EXCEPTION
        WHEN others THEN
            RAISE NOTICE 'Операторы finally';
            IF SQLSTATE != 'ALLOK' THEN
                RAISE;
            END IF;
END;
$$;
```

```
NOTICE: Операторы try
NOTICE:  ...исключение, которое обрабатывается
NOTICE: Операторы catch
NOTICE: Операторы finally
DO
```

```
=> DO $$
BEGIN
        BEGIN
            RAISE NOTICE 'Операторы try';
            --
            RAISE NOTICE '...исключение, которое не обрабатывается';
```

```
            RAISE division_by_zero;
        EXCEPTION
            WHEN no_data_found THEN
                RAISE NOTICE 'Операторы catch';
        END;
        RAISE SQLSTATE 'ALLOK';
EXCEPTION
        WHEN others THEN
            RAISE NOTICE 'Операторы finally';
            IF SQLSTATE != 'ALLOK' THEN
                RAISE;
            END IF;
END;
$$;
```

```
NOTICE: Операторы try
NOTICE: ...исключение, которое не обрабатывается
NOTICE: Операторы finally
ERROR: division_by_zero
CONTEXT: PL/pgSQL function inline_code_block line 7 at RAISE
```

Но в предложенном решении всегда происходит откат всех изменений, выполненных в блоке, поэтому оно не годится для команд, изменяющих состояние базы данных. Также не стоит забывать о накладных расходах на обработку исключений: это задание — не более, чем просто упражнение.

2. GET DIAGNOSTICS

```
=> DO $$
DECLARE
```

```
        ctx text;
BEGIN
        RAISE division_by_zero;
EXCEPTION    -- line 5
        WHEN others THEN
                GET STACKED DIAGNOSTICS ctx = pg_exception_context;

                RAISE NOTICE E'stacked =\n%', ctx;
                GET CURRENT DIAGNOSTICS ctx = pg_context; -- line 10

                RAISE NOTICE E'current =\n%', ctx;
END;

$$;

NOTICE: stacked =
 PL/pgSQL function inline_code_block line 5 at RAISE
 NOTICE: current =
 PL/pgSQL function inline_code_block line 10 at GET DIAGNOSTICS

DO
```

GET STACKED DIAGNOSTICS дает стек вызовов, приведший к ошибке.

GET [CURRENT] DIAGNOSTICS дает текущий стек вызовов.

3. Стек вызовов как массив

Собственно функция:

```
=> CREATE FUNCTION getstack() RETURNS text[]

AS $$
DECLARE
        ctx text;
BEGIN
        GET DIAGNOSTICS ctx = pg_context;

        RETURN (regexp_split_to_array(ctx, E'\n'))[2:];
END;
```

```
$$ VOLATILE LANGUAGE plpgsql;
```

CREATE FUNCTION

Чтобы проверить ее работу, создадим несколько функций, которые вызывают друг друга:

```
=> CREATE FUNCTION foo() RETURNS integer
AS $$
BEGIN
        RETURN bar();
END;
$$ VOLATILE LANGUAGE plpgsql;
```

CREATE FUNCTION

```
=> CREATE FUNCTION bar() RETURNS integer
AS $$
BEGIN
        RETURN baz();
END;
$$ VOLATILE LANGUAGE plpgsql;
```

CREATE FUNCTION

```
=> CREATE FUNCTION baz() RETURNS integer
AS $$
BEGIN
        RAISE NOTICE '%', getstack();
        RETURN 0;
END;
$$ VOLATILE LANGUAGE plpgsql;
```

CREATE FUNCTION

```
=> SELECT foo();
```

NOTICE: {"PL/pgSQL function baz() line 3 at RAISE","PL/pgSQL function bar() line 3 at RETURN","PL/pgSQL function foo() line 3 at RETURN"}

foo

0

(1 row)