PostgreSQL

Разработка приложений

Часть вторая.

Зачем нужны СУБД

- Унифицированный доступ к унифицированным данным из разных систем, платформ и языков.
  - LAMP, C#, Java EE…
  - Единое представление разнородных данных.
  - Разделение представления данных и методов доступа: единый язык запросов (SQL)/оптимизатор.
- Средство обеспечения целостности данных.
  - Внутри СУБД
  - **Независимо от приложений**.

PostgreSQL

- Стандартные объекты СУБД (таблицы, представления, процедуры), но не только:
  - Массивы, типы
  - Временные таблицы. Таблицы как переменные.
    `create temporary table temptable as select…`
  - Богатый набор встроенных индексируемых типов. (hstore, json, range)
  - Функциональные индексы
  - Условные индексы
- Богатый диалект SQL (CTE, LATERAL …)
- Транзакционный DDL (!)
- PL/pgSQL
  - Полноценный ЯП
  - Несколько СУБД-ориентированный
- Другие языки (PL/perl, PL/Java, PL/Python, PL/V8, PL/R, PL/Ruby – ну и C)
- Расширения (extensions)

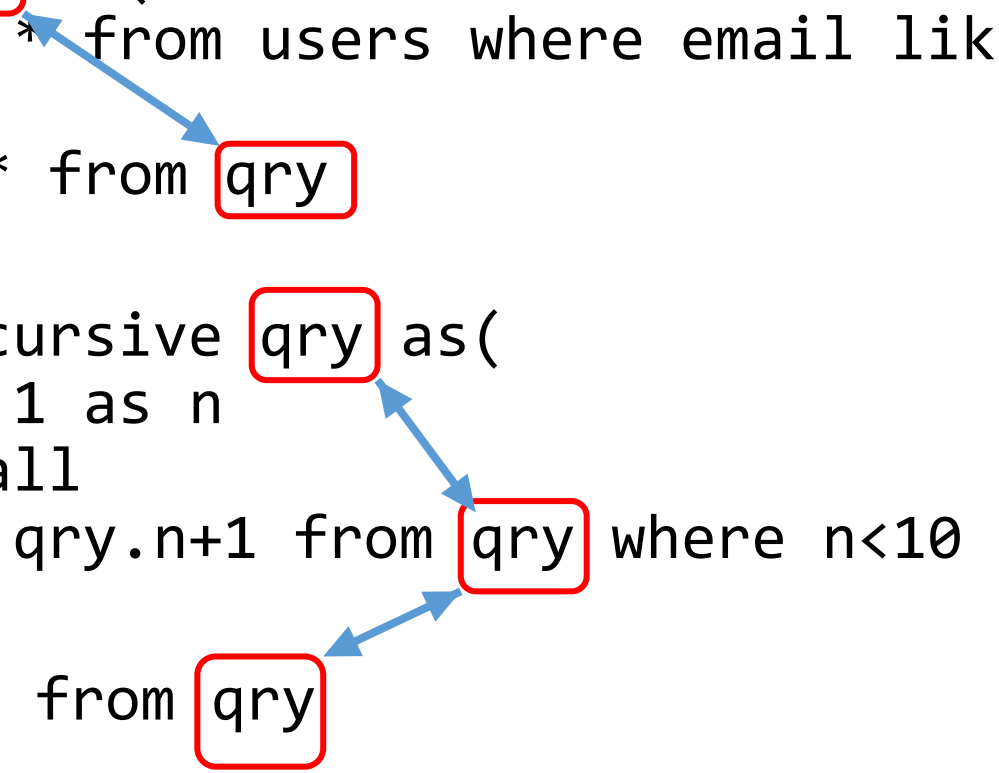
SQL

- Common table expressions (c DML!)
- WINDOW functions
- User-defined aggregation functions
- MERGE нет ☹
  - Не надо пытаться изобразить MERGE с помощью DML в CTE – не получится.
- MODEL нет ☹

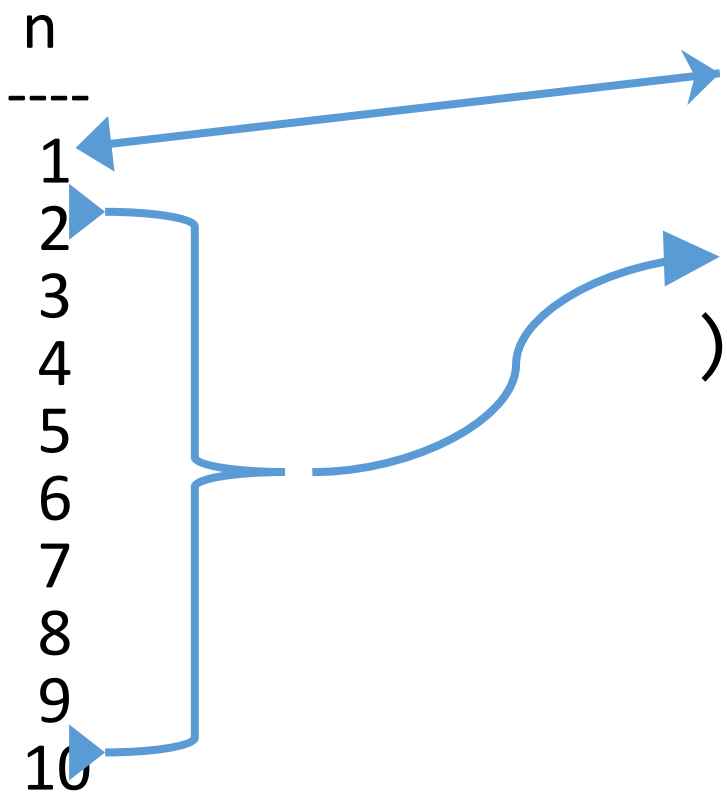
CTE

- ```
  with qry as(
    select * from users where email like '%@gmail.com‘
  )
  select * from qry
  ```

- ```
  with recursive qry as(
    select 1 as n
    union all
    select qry.n+1 from qry where n<10
  )
  select * from qry
  ```



Результат



```
 n
----
  1
  2
  3
  4
  5
  6
  7
  8
  9
 10
(10 строк)
```

```
with recursive qry as(
 select 1 as n
 union all
 select qry.n+1 from qry where n<10
)
 select * from qry
```

WINDOW functions

```
with recursive qry as(
 select 1 as n
 union all
 select qry.n+1 from qry where n<10
)
select *,
       row_number() over(order by n desc),
       sum(n) over(order by n),
       n/3,
       sum(n) over(partition by n/3 order by n)
  from qry
order by n
```

Результат

n	row_number	sum	?column?	sum
1	10	1	0	1
2	9	3	0	3
3	8	6	1	3
4	7	10	1	7
5	6	15	1	12
6	5	21	2	6
7	4	28	2	13
8	3	36	2	21
9	2	45	3	9
10	1	55	3	19

```
select *,
row_number() over(order by n desc),
sum(n) over(order by n),
n/3,
sum(n) over(partition by n/3 order by n)
  from qry
order by n
```

Массивы и типы

- ```
  create table atable(
      id int,
      vals text[]
  )
  ```
- ```
  create type auser as(
      name text,
      email text
  )
  ```
- ```
  declare var auser;
  var.name:=’Vasya’;
  ```
- Другие типы – enum, range, базовый
  ```

Временные таблицы

- Временные таблицы как глобальные табличные переменные.

```
create or replace function fun1() returns void as
$code$
begin
 create temporary table if not exists t(id int) on commit drop;
end; $code$
language plpgsql;
```

```
explain analyze
  select count(fun1()) from generate_series(1,1000);
```

Отдельная транзакция

```
"Aggregate  (cost=262.50..262.51 rows=1 width=0) (actual time=5.511..5.512 rows=1 loops=1)“

insert into t select n from generate_series(1,1000) as gs(n);

"Aggregate  (cost=262.50..262.51 rows=1 width=0) (actual time=702.723..702.723 rows=1
loops=1)"
```

Ограничения (constraints)

- Являются ЧРЕЗВЫЙЧАЙНО критичной частью и единственным критерием оценки верности введенных данных.
  - Пример: проверка валидности телефона произвоится в приложении, загрузка данных с неверным телефоном – в итоге невозможно из приложения изменить, скажем, фамилию (телефон не редактируется), а то и показать данные.
- Все стандартные (NOT NULL, PRIMARY KEY, UNIQUE, CHECK, FOREIGN KEY с каскадированием)
- EXCLUDE – более общая версия уникальности (можно, например, запретить пересекающиеся интервалы)
- DEFERRABLE/NOT DEFERRABLE

Триггеры

- Триггерные функции и почему это хорошо
  - Одна функция для нескольких триггеров.
  - Более удачная модель, чем в некоторых других СУБД – исключение в триггере откатывает всю транзакцию.
- FOR EACH ROW/STATEMENT
- CONSTRAINT TRIGGER может быть DEFERRABLE/NOT DEFERRABLE
- Несколько триггеров – вызываются в алфавитном порядке.
- Не надо опасаться триггеров – они нередко незаменимы для контроля целостности

Event triggers

- Триггеры на CREATE/ALTER/DROP
  - ```
    CREATE EVENT TRIGGER name
    ON event
    [ WHEN filter_variable IN (filter_value [, ... ]) [ AND ... ] ]
    EXECUTE PROCEDURE function_name()
    ```
- Транзакционный DDL позволяет делать интересные вещи.
- Пока только в будущем – мало чего доступно, если писать на PL/pgSQL.
    ```

Представления

- Все как обычно, можно UNION
- Обновляемые VIEW
- INSTEAD OF триггеры – вызываются при попытке использовать DML со VIEW

hstore, json, range

- hstore – плоский key->value
- JSON – JSON он и есть JSON
- Range types (int4range, int8range, numrange, tsrange, tstzrange, daterange)
- Индексы
  - `create index ”tbl[fn(col)]” on tbl(fn(col))`
  - `create index ”tbl[fn(col->’key’)]” on tbl(fn(col))`
  - `create index ”tbl[fn(col)]” on tbl(fn(col)) where age>18`

Схемы

- create schema ”schema_name”
  - Таблицы
  - Представления
  - Функции
  - и т.д.
  - search_path
- Да, нет пакетов. Зато в пакетах нет таблиц.
  - А по-хорошему было бы неплохо иметь и то, и другое.
- Схема как логическая единица.
- Extensions в схемах
  - create extension with schema <schema_name>

PL/PgSQL

- Описываются функции. Тело функции задается строкой.

```
CREATE [ OR REPLACE ] FUNCTION
    name ( [ [ argmode ] [ argname ] argtype [ { DEFAULT | = }
default_expr ] [, ...] ] )
    [ RETURNS rettype
      | RETURNS TABLE ( column_name column_type [, ...] ) ]
  { LANGUAGE lang_name
    | WINDOW
    | IMMUTABLE | STABLE | VOLATILE | [ NOT ] LEAKPROOF
    | CALLED ON NULL INPUT | RETURNS NULL ON NULL INPUT | STRICT
    | [ EXTERNAL ] SECURITY INVOKER | [ EXTERNAL ] SECURITY DEFINER
    | COST execution_cost
    | ROWS result_rows
    | SET configuration_parameter { TO value | = value | FROM CURRENT }
    | AS 'definition‘
    | AS 'obj_file', 'link_symbol‘
  } ...
    [ WITH ( attribute [, ...] ) ]
```

PL/pgSQL

- Параметры
  - Просто параметры
    - `create or replace function([IN] email text);`
  - Значения по умолчанию
    - `create or replace function([IN] email text default ‘root@localhost’);`
  - Переменный список
- Вызов
  - `select func(p1,p2,p3) [ INTO var1, var2,… ];`
  - `perform func(p1,p2,p3);`
  - `perform/select func(par1:=p1, par2:=p2, par3:=p3);`
  - `perform/select func(par1:=p1, variadic array[1,2,3]);`

PL/pgSQL

- Возвращаемые значения
  - Скалярные типы (в т.ч. и hstore или json)
  - Массивы
  - Таблицы
- INOUT и OUT или TABLE(…)
  - ```
    Create or replace function returnstable()
       returns table(col1 text, col2 int) as
    $code$
    …
    col1:=“text”; col2:=100;
    return next;
    return query select “text”, 200;
    return query execute $Q$select '---'::text, 0::int$Q$;
    ```
    ```

Атрибуты функции

- IMMUTABLE
  - Не модифицирует данные и всегда возвращает один и тот же результат для тех же самых данных
- STABLE
  - Не модифицирует данные и в течении одного сканирования таблицы ведет себя как IMMUTABLE
- VOLATILE
  - Ничего из перечисленного
- Надо внимательно смотреть, может негативно повлиять на производительность.
- SECURITY INVOKER
  - Выполняется с правами вызывающего пользователя
- SECURITY DEFINER
  - С правами создателя
- И т.д.

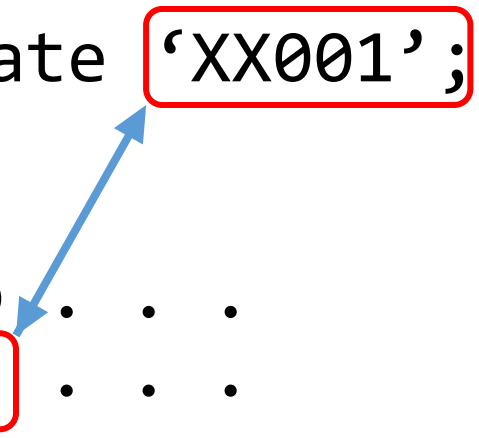
Управляющие структуры

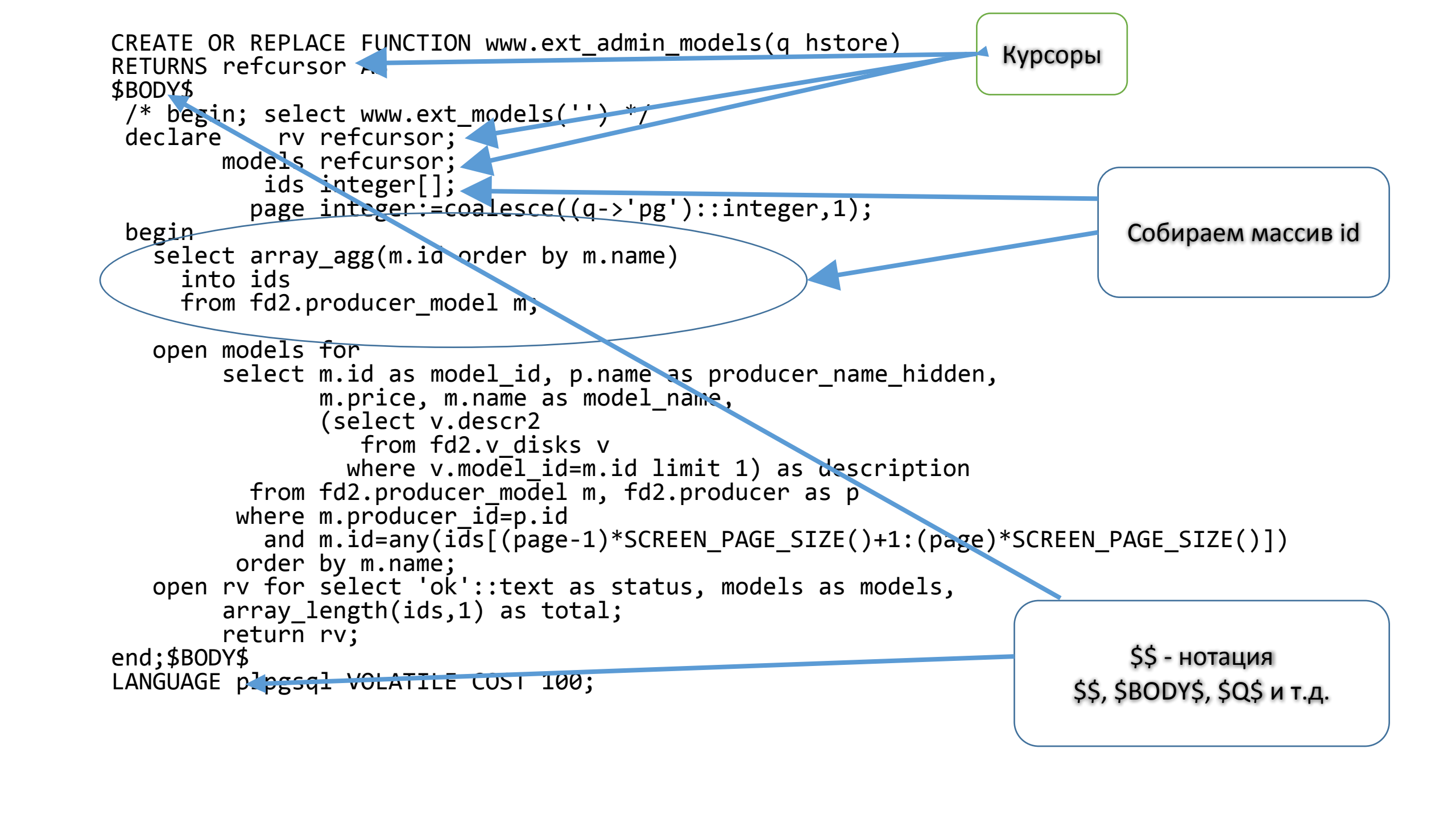
- Ничего необычного или странного
- IF-ELSE-END IF, CASE-END CASE, LOOP – END LOOP и т.п.
- SELECT INTO/PERFORM

Исключения

- <<code>>

```
declare
    begin
     . . .
     raise exception sqlstate ‘XX001’;
     . . .
    exception
     when division_by_zero . . .
     when sqlstate ’XX000’ . . .
     else
    end;
```




CREATE OR REPLACE FUNCTION www.ext_admin_models(q hstore)
RETURNS refcursor A
$BODY$
/* begin; select www.ext_models('') */
declare rv refcursor;
models refcursor;
ids integer[];
page integer:=coalesce((q->'pg')::integer,1);
begin
select array_agg(m.id order by m.name)
into ids
from fd2.producer_model m;
open models for
select m.id as model_id, p.name as producer_name_hidden,
m.price, m.name as model_name,
(select v.descr2
from fd2.v_disks v
where v.model_id=m.id limit 1) as description
from fd2.producer_model m, fd2.producer as p
where m.producer_id=p.id
and m.id=any(ids[(page-1)*SCREEN_PAGE_SIZE()+1:(page)*SCREEN_PAGE_SIZE()])
order by m.name;
open rv for select 'ok'::text as status, models as models,
array_length(ids,1) as total;
return rv;
end;$BODY$
LANGUAGE plpgsql VOLATILE COST 100;
Курсоры
Собираем массив id
$$ - нотация
$$, $BODY$, Q и т.д.

PL/pgSQL и транзакции

- Невозможно управлять транзакциями
  - И это правильно!
- Транзакции инициируются клиентом СУБД
- Если транзакция не начата, каждый вызов функции оборачивается в транзакцию, иначе все выполняется в контексте одной транзакции.

SAVEPOINT

- SAVEPOINT <savepoint name>
- ROLLBACK TO SAVEPOINT
- Из PL/pgSQL нельзя как ни начать, ни завершить транзакцию, так и ни установить savepoint, ни откатиться к какому-либо.
- При возникновении исключения в блоке при наличии обработчика осуществятся откат к неявно установленному при входе в блок savepoint’у.

```
do $code$
 begin
   insert into t values(1);
   raise notice '1:%', (select count(*) from t);
   begin
     insert into t values(1);
     raise notice '2:%', (select count(*) from t);
     raise exception sqlstate 'ZQ001';
   exception
      when sqlstate 'ZQ000' then
        raise notice '3:%', (select count(*) from t);
   end;
   raise notice '4:%', (select count(*)from t);
end;
$code$
```

1:1

1:2

3:1

4:1

Уровни изоляции

- READ UNCOMMITED – отсутствует
- READ COMMITED
- REPEATABLE READ
- SERIALIZABLE
  - Можно получить ошибку serialization_failure
  - DEFERRABLE READ ONLY – отчеты, бекапы и т.д.

Расширения

- Достаточно большое количество стандартных расширений (42 шт.)
  - Не установлены по умолчанию.
  - dblink
  - earthdistance & PostGIS
  - pg_trgm
  - pgcrypto