Миграция с Oracle на Postgresql

Игнатов А.В.

Миграция с Oracle на Postgresql

Что помогает при миграции с Oracle на PostgreSQL:

- Похожесть языка PL/SQL и pgplsql
- Расширяемость PostgreSQL
- Поддержка PostgreSQL многих языков программирования — pl/perl, pl/python, pl/tcl
- Холодная голова. Поначалу может показаться что миграция невозможна. Это не так. Ведь всегда можно полностью переписать код. Шутка ;)

Основные инструменты для миграции с Oracle

- ora2pg
- Секретный инструмент

ora2pg

- Позволяет сделать очень многое для миграции в автоматическом режиме с Oracle в PostgreSQL

  Осуществляет конвертацию:
- базовых типов (**varchar2**, **number** и пр.)
- составных типов (as **record**)
- последовательностей

ora2pg

- таблиц и данные этих таблиц(**insert** или **copy** по вкусу)
- представлений(**view**)
- триггеров
- хранимых процедур PL/SQL (базовый уровень, иногда требуется приличная работа руками)
- автономных транзакций(посредством использования dblink)

ora2pg

- Автоматическая конвертация PL/SQL в PL/PGSQL
- Автоматическое создание проекта миграции
- Создание отчёта по времени миграции

ora2pg

Что пока не умеет ora2pg

- Конвертирование глобальных переменных в пакетах. Требуются обходные пути
- Функции определяемые внутри других функций. Необходимо определять как отдельные функции
- Конвертировать составные триггера

ora2pg

Специфический код Oracle

- Внешние модули(DBMS, UTL и прочие). Частичная замена модуль orafce
- CONNECT BY. Необходимо переписывать с использованием рекурсивных CTE
- Outer join определенный посредством оператора (+)
- Различные функцкии вроде DECODE. Базовая поддержка.

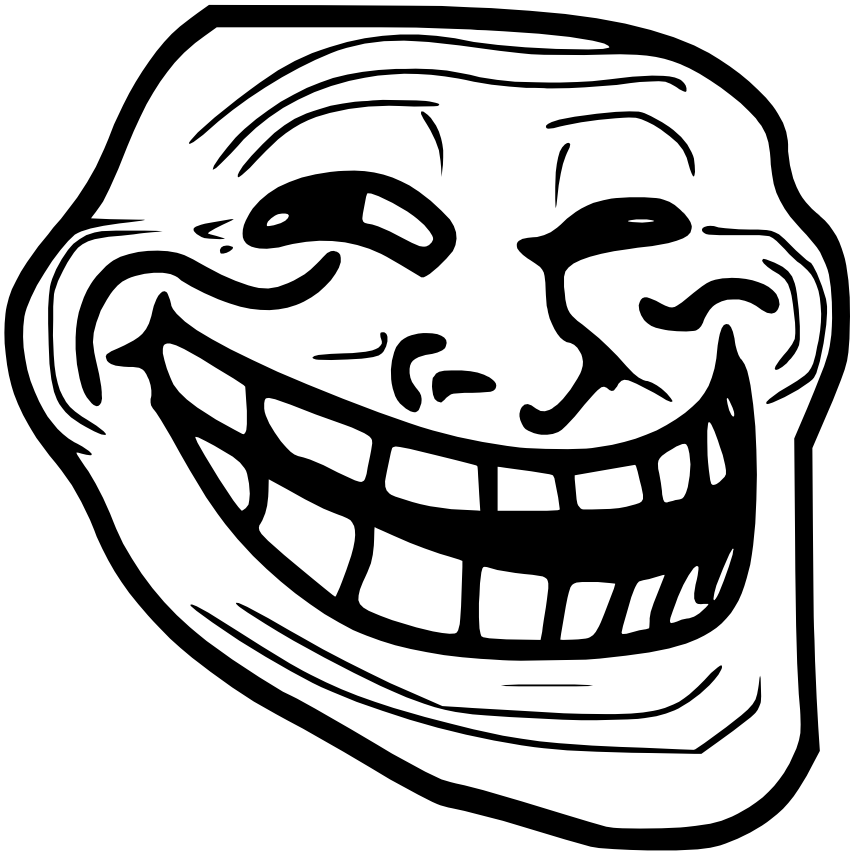
ora2pg

Ознакомится с полными возможностями ora2pg можно здесь:

- http://pgconf.ru/static/presentations/2015/darold.pdf
- http://ora2pg.darold.net/

Так а что же насчёт секретного инструмента??

Секретный инструмент



Тип record

- Неявное объявление в Oracle типа **record** (переменная **rec**)
- В PostgreSQL обязательно нужно объявлять переменную **rec** с типом **record** для её использования в циклах вида **for rec in (...)**

Тип record

Oracle	PostgreSQL

for rec in (select * from ...)
loop
...
end loop;
	declare
	  rec record;
	  ...
	begin
	  ...
	  for rec in (select * from ...)
	  loop
	  ...
	  end loop;
	  ...

Тип refcursor

- В Oracle требуется описание данного типа. Много лишних слов и телодвижений ;)
- В PostgreSQL нужно лишь объявить переменную типа refcursor и тут же можно работать с ней

Тип refcursor

Oracle	Postgres

TYPE cursor_type IS REF CURSOR;
TYPE stmt_type IS RECORD (
...
);
...
declare
  cur cursor_type;
  stmt clob;
  stmt_rec stmt_type ;
begin
  stmt:='select * from ...';
  open cur for stmt;
  loop
  fetch cur into stmt_rec;
  exit when cur%NOTFOUND;
  ...
  end loop;
...
	...
	declare
	  cur refcursor;
	  rec record;
	  stmt text;
	begin
	  stmt:='select * from...';
	  open cur for execute stmt;
	  loop
	  fetch cur into rec;
	  if not found
	  then
	  exit;
	  end if;
	  ...
	  end loop;
	...

rownum

- В Oracle есть псевдоколонка для получения номера строки в результате выполнения запроса
- В Oracle значение выставляется перед сортировкой
- Ораклиный **rownum** можно получить путем использования оконной функции row_number() over().
- Если используется сортировка в select, то необходимо использовать row_number() over (сортировка)

rownum

Oracle	Postgresql
`select rownum as rn, ... from ...`	`select row_number() over([order by...]) as rn, ... from...[order by ...]`
`SELECT ROWNUM, foo FROM places ORDER BY place`	`select row_number() over(order by place) as rn, place from places order by place`
`select * from ( select * from places order by place ) where ROWNUM <= 10;`	`SELECT * FROM places ORDER BY place LIMIT 10`

ROWCOUNT

- Данная переменная используется для получения количества задействованных (измененных, полученных, вставленных, удаленных) строк последнего запроса
- У Oracle есть «волшебнае комбинация слов» SQL %ROWCOUNT
- У PostgreSQL тоже есть, но не комбинация а конструкция

ROWCOUNT

Oracle	Postgresql

DECLARE
count number;
BEGIN
update places set place ='НЬЮ-ЙОРК' where place = 'Нью-Йорк';
DBMS_OUTPUT.PUT_LINE(to_char(SQL %ROWCOUNT));
END;
/
	do $$
	DECLARE
	count integer;
	BEGIN
	update places set place ='НЬЮ-ЙОРК' where place = 'Нью-Йорк';
	get diagnostics count:=ROW_COUNT;
	raise notice 'updated % rows',count;
	END;
	$$

Функции trim, instr,substr

- Очень часто используются
- У Oracle есть эти функции
- У PostgreSQL тоже есть, но могут вызываются и по SQL стандарту

Функции trim, instr,substr

Oracle	Postgresql
trim(a)	trim (from a)
instr(a,b)	position(b in a) strpos(a, b)
substr(a,1)	substring(a from 1) или substr(a,1)

Функция instr

Если нужна функция прямо как у Oracle то можно создать аналогичную по синтаксису:

http://www.postgresql.org/docs/9.5/static/plpgsql-porting.html

Раздел 40.12.3. Appendix

ORAFCE

Модуль содержит несколько полезных функций которые могут помочь при миграции с Oracle на Postgres

- Таблица dual
- concat, nvl, nvl2, lnnvl, decode...
- dbms_output
- utl_file
- dbms_pipe
- dbms_alert
- Множество функций для работы с датой -add_months,last_date, next_day …
- https://github.com/orafce/orafce

Quotation(цитирование)

Используется:

- Лентяями, которые не хотят экранировать символы. Шутка ;)
- Например, для написания удобоваримых динамических SQL выражений =)

Quotation(цитирование)

Oracle	Postgresql
`q'{This is 'string'}'`	`$$This is 'string'$$`

DECLARE
v VARCHAR2(1024);
BEGIN
v := q'{It's a string with embedded quotes…}';
DBMS_OUTPUT.PUT_LINE(v);
END;
/
	do $$
	DECLARE
	v text;
	BEGIN
	v := s[It's a string with embedded quotes...]'s;
	raise notice '%',v;
	END;
	$$

Глобальные сессионные переменные

- Как у них (Oracle) — они есть
- Как у нас (PostgreSQL) — их нет =(!! Кто-то скажет больно надо - это же потенциальные баги!!
- Отвечаем — возможно, но тогда придётся переписать код при миграции с Oracle. Может быть даже очень много.
- Возможное использование — передача между хранимыми процедурами некоторых параметров в пределах текущей сессии
- Являются потенциальными ошибками, которые ждут не дождутся чтобы проявить себя

Так как же определить эти глобальные сессионные переменные?

Глобальные сессионные переменные

Определить можно:

- Через глобальные переменные pl/perl или pl/python
- Через временные таблицы

Глобальные сессионные переменные

Через глобальные переменные языков pl/perl(переменная %_SHARED) или pl/python(переменная SD):

```
CREATE OR REPLACE FUNCTION set_var(name text, val text) RETURNS text AS $$
    if ($_SHARED{$_[0]} = $_[1]) {
        return 'ok';
    } else {
        return "cannot set shared variable $_[0] to $_[1]";
    }
$$ LANGUAGE plperl;

CREATE OR REPLACE FUNCTION get_var(name text) RETURNS text AS $$
    return $_SHARED{$_[0]};
$$ LANGUAGE plperl;

SELECT set_var('sample', 'Hello, PL/Perl!  How''s tricks?');
SELECT get_var('sample');
```

Глобальные сессионные переменные

- Через использование временных таблиц

Глобальные сессионные переменные

```
CREATE OR REPLACE FUNCTION set_variable( IN p_var TEXT, IN p_val TEXT ) RETURNS void as $$
DECLARE
    v_var TEXT;
BEGIN
    execute 'CREATE temp TABLE IF NOT exists sys_variables ( variable TEXT PRIMARY KEY, value TEXT );';
    LOOP
        execute 'UPDATE sys_variables SET value = $1 WHERE variable = $2 returning variable' INTO v_var USING p_val,
p_var;
        IF v_var IS NOT NULL THEN
            RETURN;
        END IF;
        BEGIN
            execute 'INSERT INTO sys_variables ( variable, value ) VALUES ( $1, $2 )' USING p_var, p_val;
            RETURN;
        EXCEPTION WHEN unique_violation THEN
            -- ignore, re-process the loop
        END;
    END LOOP;
END;
$$ language plpgsql;
CREATE OR REPLACE FUNCTION get_variable( IN p_var TEXT ) RETURNS TEXT as $$
DECLARE
    v_val TEXT;
BEGIN
    execute 'CREATE temp TABLE IF NOT exists sys_variables ( variable TEXT PRIMARY KEY, value TEXT );';
    execute 'SELECT value FROM sys_variables WHERE variable = $1' INTO v_val USING p_var;
    RETURN v_val;
END;
$$ language plpgsql;
```

http://www.depesz.com/2013/02/25/variables-in-sql-what-how-when

Глобальные сессионные переменные

```
$ select set_variable('xx', 'qq');
$ select get_variable('xx');
```

Глобальные сессионные переменные

- Приведеные выше способы определения глобальных сессионных переменных все таки являются костылями и использоваться должны лишь в крайних случаях
- При использовании pl/perl каждый серверный процесс запускает свою версию perl => использование памяти порядка 25 мегабайт на процесс
- При использовании временных таблиц возможно разрастание системного каталога и деградация производительности со временем при ненадлежащем его вакууме

Глобальные сессионные переменные

Где почитать:

- http://www.depesz.com/2013/02/25/variables-in-sql-what-how-when
- http://www.postgresql.org/docs/9.5/static/plperl-global.html
- http://www.postgresql.org/docs/9.5/static/plpython-sharing.html

Булки(bulk collect)

- В Oracle используются для быстрой загрузки данных из одной таблицы в другую. Снижение количества контекстных переключений. Перемещение нескольких записей за раз. Использование памяти
- В PostgreSQL — нет такого, необходимо придумывать обходные пути.
- Что первое приходит на ум - использование массивов
- В версиях до версии 9.5 надо очень осторожно принимать решение использовать массивы если нужен будет доступ(изменение) к произвольному элементу этого массива и добавление элемента массива

Булки(bulk collect)

В PostgreSQL:

Используем пару трюков:

- Имя таблицы может использоваться как имя типа
- Имя таблицы может использоваться как фиктивный столбец, который содержит в себе все поля таблицы

Это он о чём говорит вообще??

Булки(bulk collect)

Oracle	Postgresql

create or replace procedure bulk_test is
declare
type tb_person_rt is table of tb_person%rowtype;
a tb_person_rt;
begin
select * bulk collect into a from tb_person;
end;
	create or replace function bulk_test() returns void as $$
	declare
	a tb_person[] ;
	begin
	a : = array(select tb_person from tb_person);
	...
	end;
	$$ language plpgsql;

Массивы

- До версии PostgreSQL 9.5 работа с массивами типов переменной длины(text, составные типы) была осложнена тем фактом, что такие массивы были очень медленны при операциях: добавления, изменение элемента, доступ к элементу
- В 9.5 эта проблема решена
- http://git.postgresql.org/gitweb/?p=postgresql.git;a=commit;h=1dc5ebc9077ab742079ce5dac9a6664248d42916

Пользовательские глобальные сессионные константы

- В Oracle они есть!
- В PostgreSQL их нет =(…
- Так же широко используются как и глобальные сессионные переменные, может быть даже чаще

Postgres PROFESSIONAL

Можем ли мы всё-таки определить пользовательские глобальные сессионные константы?

Postgres PROFESSIONAL

Да мы можем

Пользовательские константы

- Определение константы возможно через создание функции
- Функцию нужно сделать IMMUTABLE, так как она возвращает всегда одно и тоже значение и, поэтому, планировщик это сможет учитывать при построении плана выполнения
- Получение значения такой константы работает достаточно быстро ~ 4 млн присвоений значения в секунду на core i7

Пользовательские константы(пример)

```
create or replace function pi()
returns double precision
immutable
language sql as
'select 3.14159265358979::double precision';
do $$
   declare
      r double precision:=1;
      s double precision;
   begin
      s:=pi()*r*r;
      raise notice 'Площадь круга радиуса % = %',r,s;
   end;
$$
```

Пользовательские исключения

- База данных не может заранее знать о всевозможных ошибках при обработки вводимой инфомации
- Поэтому если что-то пошло не по плану нужно суметь это обработать
- У Oracle — они есть!!! И активно используются
- В PostgreSQL — их нет =(…
- В PostgreSQL исключение можно вызвать с неким номером(**errcode**) и потом по этому номеру отловить это исключение

Пользовательские исключения Oracle

```
EXCEPTION1 EXCEPTION;
--PRAGMA EXCEPTION_INIT(EXCEPTION1,-06502);
EXCEPTION2 EXCEPTION;
--PRAGMA EXCEPTION_INIT(EXCEPTION1,-06503);
...
BEGIN
...
RAISE EXCEPTION EXCEPTION1;
...
RAISE EXCEPTION EXCEPTION2;
EXCEPTION
   WHEN EXCEPTION1 THEN
   ...
   WHEN EXCEPTION2 THEN
   ...
```

А как же быть в PostgreSQL???

Пользовательские исключения PostgreSQL

- Раз их нет надо что-то придумать
- В сообществе PostgreSQL регулярно всплывает этот вопрос

Пользовательские исключения PostgreSQL

- Выше мы говорили про определения пользовательских глобальных констант
- Это нам пригодится для описания пользовательских исключений и их обработки

Пользовательские исключения PostgreSQL

- EXCEPTION1 имя первого исключения
- EXCEPTION2 имя второго исключения
- Сделаем соответствие их имени и номера исключения
- Делаем это как при определении констант через функции
- Используем переменную SQLSTATE

Пользовательские исключения PostgreSQL(пример)

```
create or replace function EXCEPTION1()
   returns text as
$body$
   begin
      return 06502;
   end;
$body$
language plpgsql
Immutable;

create or replace function EXCEPTION2()
   returns text as
$body$
   begin
      return 06503;
   end;
$body$
language plpgsql
Immutable;
```

Пользовательские исключения PostgreSQL(пример)

```
begin
   ...
   raise exception using errcode=EXCEPTION1();
   ...
   raise exception using errcode=EXCEPTION2();
   ...
exception
   ...
   when OTHERS then
      case SQLSTATE
         when EXCEPTION1() then
         ...
         when EXCEPTION2() then
         ...
      end case;
```

Пользовательские исключения PostgreSQL

- При отлове исключений должны использоваться константные выражения
- Поэтому используем **when OTHERS** для отлова пользовательских исключений и далее проверяем переменную **SQLSTATE**

Пользовательские исключения PostgreSQL(пример)

```
begin
    ...
    raise exception using errcode=EXCEPTION1();
    ...
    raise exception using errcode=EXCEPTION2();
    ...
exception
          when EXCEPTION1() then
          ...
          when EXCEPTION2() then
          ...
```

Rollback в функциях plpgsql

- В Oracle есть возможность откатить транзакцию в процедурах pl/sql.
- Для этого используются savepoint
- В Postgres функция есть часть транзакции, которая запустила данную функцию
- Rollback в функциях plpgsql делается не так как в Oracle

Rollback в функциях plpgsql(пример)

Oracle	Postgres

CREATE OR REPLACE PROCEDURE proc
AS
BEGIN
--Создаем savepoint1
SAVEPOINT savepoint1;
insert into test(id,text) (1,'abcd');
update test set text='xyz' where id=1;
-- If any exception occurs
EXCEPTION
WHEN OTHERS THEN
-- We roll back to the savepoint.
ROLLBACK TO savepoint1;
-- And of course we raise again,
-- since we don't want to hide the error.
-- Not raising here is an error!
RAISE;
END;
	create or replace function proc()
	returns void
	as $body$
	begin
	insert into test(id,text) (1,'abcd');
	update test set text='xyz'
	where id=1;
	--Создаем 'скрытый' savepoint
	begin
	insert into test(id,text)
	(2,'efgh');
	update test set text='qwert'
	where id=2;
	--Тут происходит исключение
	exception
	when others then
	--откатываемся к 'скрытому' savepoint
	raise;
	end;
	end;
	$body$
	language plpgsql
	security definer;

Commit в функциях plpgsql

- В Postgres нет rollback сегментов или табличного пространства undo, которые могут кончится. В Postgres кончится может только диск %)!!!
- Поэтому commit в функциях не нужны =)
- Необходимо пересматривать логику работы программы

Иерархические запросы

- В Oracle с давних пор (1977 год?) используется оператор CONNECT BY(и так же функции, например, как SYS_CONNECT_BY_PATH)
- В Postgres такого оператора нет. Зато есть возможность написания рекурсивных запросов. В Oracle они тоже появились с некоторых пор, но запросы с CONNECT BY выглядят изящнее

Иерархические запросы

place	is_in
Россия	
Ленинградская область	Россия
Санкт-Петербург	Ленинградская область
Васильевский остров	Санкт-Петербург
Московская область	Россия
Москва	Московская область
ЦАО	Москва
Кремль	ЦАО
Пушкинская площадь	ЦАО
США	
Нью-Йорк	США
Манхэттен	Нью-Йорк
Таймс-сквер	Манхэттен
Эмпайр-стейт-билдинг	Манхэттен
Вашингтон	США
Белый дом	Вашингтон

Что находится в России?

?

Иерархические запросы

- Задача - найти в даной таблице все места, которые есть в России
- Сделать это без использования каких-либо других таблиц и страшных запросов

Иерархические запросы(пример)

Oracle	Postgres
SELECT p.places, p.is_in FROM places p START WITH p.is_in = 'Россия' CONNECT BY PRIOR p.place = p.is_in	WITH RECURSIVE cte AS(SELECT place, is_in FROM places WHERE is_in = 'Россия' UNION ALL SELECT prev.place, prev.is_in FROM places prev JOIN cte ON cte.place =prev.is_in) SELECT place, is_in FROM cte

Иерархические запросы(пример)

Postgres

```
          place          |          is_in
-------------------------+-------------------------
 Ленинградская область   | Россия
 Московская область      | Россия
 Санкт-Петербург         | Ленинградская область
 Москва                  | Московская область
 Васильевский остров     | Санкт-Петербург
 ЦАО                     | Москва
 Кремль                  | ЦАО
 Пушкинская площадь      | ЦАО
```

Oracle

place	is_in
Ленинградская область	Россия
Санкт-Петербург	Ленинградская область
Васильевский остров	Санкт-Петербург
Московская область	Россия
Москва	Московская область
ЦАО	Москва
Кремль	ЦАО
Пушкинская площадь	ЦАО

Иерархические запросы SYS_CONNECT_BY_PATH

- Иногда требуется проследить иерархию подчинения от корневой записа до конечного элемента
- В Oracle используется оператор SYS_CONNECT_BY_PATH в иерархических запросах
- В Postgres SYS_CONNECT_BY_PATH нет, но как всегда есть решение. И выглядит оно солидно ;)

Иерархические запросы SYS_CONNECT_BY_PATH(пример)

Oracle	Postgres
SELECT place, SYS_CONNECT_BY_PATH(place, '/') "PATH", level FROM places START WITH is_in is null CONNECT BY PRIOR place= is_in	WITH RECURSIVE cte AS (SELECT pl.place, pl.is_in, pl.place as path ,1 as level FROM places pl WHERE pl.is_in is null UNION ALL SELECT t2.place, t2.is_in, cte.path ||'/'|| t2.place,level +1 FROM places as t2 inner join cte on (cte.place=t2.is_in)) SELECT place,'/'||path as path,level FROM cte

Иерархические запросы SYS_CONNECT_BY_PATH(пример)

Oracle	Postgres
SELECT place, SYS_CONNECT_BY_PATH(place, '/') "PATH", level FROM places START WITH is_in is null CONNECT BY PRIOR place= is_in	WITH RECURSIVE cte AS (SELECT place, 1 as level, ARRAY[place] AS path FROM places where places.is_in is null UNION ALL SELECT next.place, prev.level + 1 as level, prev.path || next.place as path FROM cte prev, places next WHERE prev.place = next.is_in) SELECT place, '/'|| array_to_string(path, '/') as path,level from cte order by path

Иерархические запросы SYS_CONNECT_BY_PATH(пример)

Oracle

```
PLACE                     PATH                                                                  LEVEL
------------------------- ---------------------------------------------------------------------- ----------
Россия                    /Россия                                                                        1
Ленинградская область     /Россия/Ленинградская область                                                  2
Санкт-Петербург           /Россия/Ленинградская область/Санкт-Петербург                                  3
Васильевский остров       /Россия/Ленинградская область/Санкт-Петербург/Васильевский остров              4
Московская область        /Россия/Московская область                                                     2
Москва                    /Россия/Московская область/Москва                                              3
ЦАО                       /Россия/Московская область/Москва/ЦАО                                          4
Кремль                    /Россия/Московская область/Москва/ЦАО/Кремль                                   5
Пушкинская площадь        /Россия/Московская область/Москва/ЦАО/Пушкинская площадь                        5
США                       /США                                                                           1
Вашингтон                 /США/Вашингтон                                                                 2
Белый дом                 /США/Вашингтон/Белый дом                                                       3
Нью-Йорк                  /США/Нью-Йорк                                                                  2
Манхэттен                 /США/Нью-Йорк/Манхэттен                                                        3
Таймс-сквер               /США/Нью-Йорк/Манхэттен/Таймс-сквер                                            4
Эмпайр-стейт-билдинг      /США/Нью-Йорк/Манхэттен/Эмпайр-стейт-билдинг                                   4
```

Иерархические запросы SYS_CONNECT_BY_PATH(пример)

Postgres

place	path	level
Россия	/Россия	1
США	/США	1
Ленинградская область	/Россия/Ленинградская область	2
Московская область	/Россия/Московская область	2
Нью-Йорк	/США/Нью-Йорк	2
Вашингтон	/США/Вашингтон	2
Санкт-Петербург	/Россия/Ленинградская область/Санкт-Петербург	3
Москва	/Россия/Московская область/Москва	3
Манхэттен	/США/Нью-Йорк/Манхэттен	3
Белый дом	/США/Вашингтон/Белый дом	3
Васильевский остров	/Россия/Ленинградская область/Санкт-Петербург/Васильевский остров	4
ЦАО	/Россия/Московская область/Москва/ЦАО	4
Таймс-сквер	/США/Нью-Йорк/Манхэттен/Таймс-сквер	4
Эмпайр-стейт-билдинг	/США/Нью-Йорк/Манхэттен/Эмпайр-стейт-билдинг	4
Кремль	/Россия/Московская область/Москва/ЦАО/Кремль	5
Пушкинская площадь	/Россия/Московская область/Москва/ЦАО/Пушкинская площадь	5

Иерархические запросы
CONNECT BY NOCYCLE

А что если сделать дополнительно

insert into places values ('Москва','ЦАО')?

- Oracle ORA-01436: CONNECT BY loop in user data
- Postgres - бесконечный цикл
- Для Postgres есть решение

Иерархические запросы
CONNECT BY NOCYCLE(пример)

Oracle	Postgres
SELECT place, SYS_CONNECT_BY_PATH(place, '/') "PATH", level as "LEVEL" FROM places START WITH is_in is null CONNECT BY NOCYCLE PRIOR place= is_in;	WITH RECURSIVE cte AS (SELECT place, 1 as level, ARRAY[place] AS path,false AS cycle FROM places where places.is_in is null UNION ALL SELECT next.place, prev.level + 1 as level, prev.path || next.place as path ,next.place=any(prev.path) FROM cte prev, places next WHERE prev.place = next.is_in AND cycle = false) SELECT place, '/'|| array_to_string(path, '/') as path,level from cte order by path

Интерфейсы к Postgres

- ODBC
- JDBC. Нужно помнить о наличии параметра autocommit
- .NET
  Npgsql

Postgres PROFESSIONAL

Вопросы?

Спасибо за внимание!

Контакты:

a.ignatov@postgrespro.ru