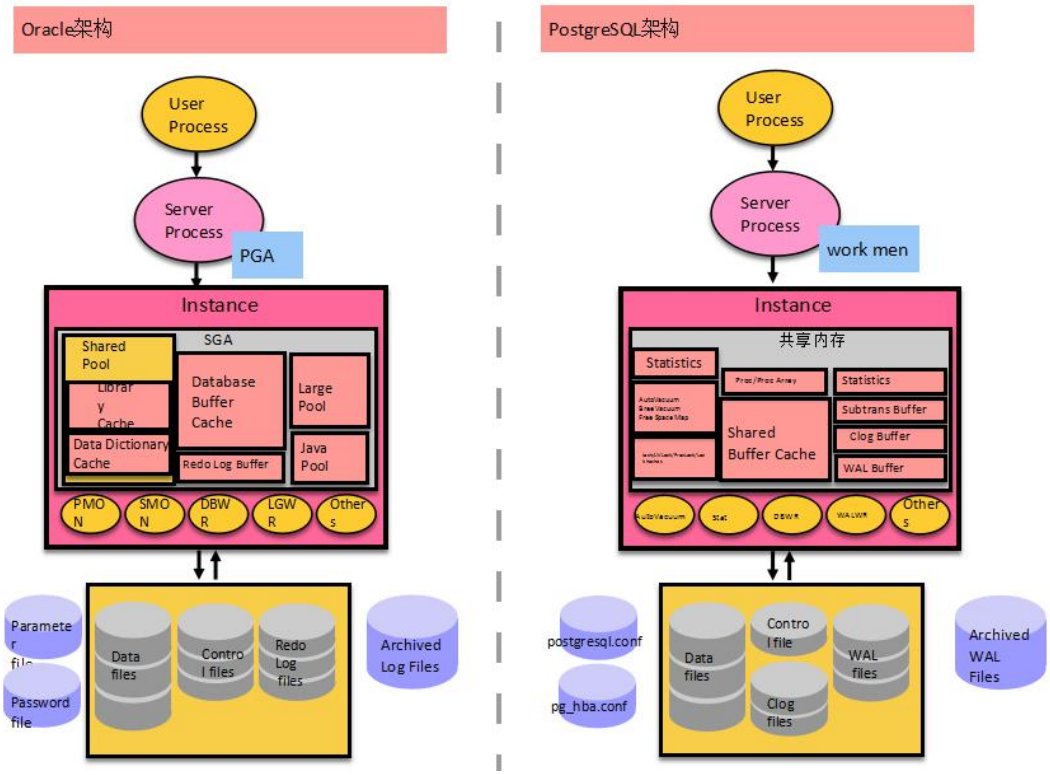
oracle 迁移到 pg 全流程

postgres 限制

Limit	Value
Maximum Database Size	Unlimited
Maximum Table Size	32TB
Maximum Row Size	1.6TB
Maximum Field Size	1GB
Maximum Rows/Table	Unlimited
Maximum Columns/Table	250~1600
Maximum Indexes/Table	Unlimited

DBA 的概念映射

Postgres 的架构和 **oracle** 对比



1,MVCC 的原理实现有差异:
oracle 采用 **rollback segment** 的方式实现
Postgres 采用事务 **id xmin xmax** 的方式实现

2,SGA-> shared_buffers
3,PGA-> work_mem
4,PMON-> postmaster
5,TNS Listener -> postmaster
6,grant/revoke -> 几乎一样的语法

Oracle pg 的数据类型映射

Oracle Type	postgresL Type	Comment
Varchar、varchar2、nvarchar、nvarchar2	**char, varchar, text**	
char nchar	**char, varchar, text**	
cblog, long	**Varchar,text,jsonb**	
Number	**Bigint,int,small,real,double presion:**性能很好,精度不好控制 **Numeric:**精度很高，性能略差	
Binary_integer,binary_float, BINARY_DOUBLE	**Integer,float,numeric**	
Blob,raw,log_raw	**Bytea** 如果大对象是 **json** 可以换做 **jsonb**	
Date	**Date or timestamp Timestamp with timezone**	
Date 加减	**Date + inteval ' N day/minute'**	
Nls_date_format	**To_char to_date**	
TIMESTAMP	**date,timestamp, timestamptz, char,varchar, text**	
TIMESTAMP WITH TIME ZONE	**date, timestamp, timestamptz, char, varchar, text**	
TIMESTAMP WITH LOCAL TIME ZONE	**date, timestamp, timestamptz, char, varchar, text**	
INTERVAL YEAR TO MONTH	**interval, char, varchar, text**	
INTERVAL DAY TO SECOND	**interval, char, varchar, text**	
MDSYS.SDO_GEOMETRY	**geometry (see "PostGIS support")**	

oracle 迁移 postgres 转换

项目	Oracle	Postgres
当前时间	SYSDATE	now(),clock_timestamp(),current_time,current_date,current_time,current_timestamp,localtime,localtimestamp
序列	SEQNAME.NEXTVAL	NEXTVAL('SEQNAME')
固定值列	SELECT '1' AS COL1	SELECT CAST('1' AS TEXT) as col
NVL	NVL 函数	用 COALESCE 函数替换
INSTR 函数	instr('str1','str2')	strpos('str1','str2')
外连接	Oracle 可简写为(+)	用 LEFT JOIN 等语句替换
层次查询	START WITH 语句 CONNECT BY 语句	用 WITH RECURSIVE 语句
数据库对象大小写	统一大写,""包起来的除外	统一小写,""包起来的除外
GOTO 语句	GOTO 语句	pgsql 不支持
同义词	Oracle 支持同义词	用视图代替
trunc	trunc(时间)	date_trunc()
DUAL	SELECT 1+1 FROM DUAL	SELECT 1+1 或者 CREATE VIEW dual AS SELECT current_timestamp
ROWNUM	ROWNUM 关键字	两种情况： 1.限制结果集数量，用于翻页等： SELECT * FROM T LIMIT 5 OFFSET 0 2.生成行号： ROW_NUMBER() OVER()
DECODE 等判断函数	DECODE()	用标准的 CASE WHEN THEN ELSE END 语句替换
TO_CHAR	TO_CHAR(COL,FMT)，格式化字符串可以为空	TO_CHAR(COL1,'FM999999')，9 的个数为字段长度，详细定义见：https://www.postgresql.org/docs/10/static/functions-formatting.html
TO_NUMBER	TO_NUMBER(COL,FMT)，格式化字符串可以为空	TO_NUMBER(COL1,'999999')，9 的个数为字段长度，详细定义见：http://www.postgresql.org/docs/10/static/functions-formatting.html
NULL 和''	ORACLE 认为 '' 等同于 NULL,'a'||null 结果是'a'	NULL 和''不同,'a'||null 结果是 null,用 concat()函数替代
NULL 和''	LENGTH('')为 NULL	LENGTH('')为 0
NULL 和''	TO_DATE('','YYYYMMDD') 为空	TO_DATE('','YYYYMMDD')为 0001-01-01 BC
NULL 和''	TO_NUMBER('',1)为 NULL	TO_NUMBER('',1)，报错
ADD_MONTHS	ADD_MONTHS(DATE,INT)	CREATE FUNCTION add_months(date, int) RETURNS date AS 'SELECT ($1 +($2::text||'' month'')::interval)::date' LANGUAGE 'sql'

		或 SQL: SELECT ($1 +($2::text||' month')::interval)
LAST_DAY	LAST_DAY(DATE)	创建函数来解决 CREATE OR REPLACE FUNCTION last_day(date) RETURNS date AS $$ SELECT (date_trunc('MONTH', $1) + INTERVAL '1 MONTH - 1 day')::date; $$ LANGUAGE 'sql'; 或 SQL: SELECT (date_trunc('MONTH', $1) + interval '1 month - 1 day')::date;
MONTHS_BETWEEN	MONTHS_BETWEEN(DATE, DATE)	创建函数来解决 CREATE FUNCTION MONTH_BETWEEN (d1 timestamp,d2 timestamp) RETURNS NUMERIC AS 'SELECT (extract(year from age(d1,d2))*12 + extract(month from age(d1,d2)))::integer' LANGUAGE 'sql';
BITAND	BITAND(A,B)	A & B
MINUS	MINUS 语句	以 EXCEPT 语句来替代
BIN_	SELECT BIN_TO_NUM(1,0,1,0) AS VALUE1 FROM DUAL	SELECT CAST(B'1010' AS INTEGER) AS VALUE1
UPDATE 语句列列表	UPDATE accounts SET (contact_last_name, contact_first_name) = (SELECT last_name,first_name FROM salesmen WHERE salesmen.id =accounts.sales_id);	UPDATE accounts a SET contact_last_name=blast_name, contact_first_name=b.first_name From salesmen b b.id =a.sales_id);
SUBSTR 函数	如果从第一个开始取子串，可以从 0 开始，也可以从 1 开始，如果不是第一个开始，则从 1 开始计数，可以为负值，从字符串结尾计数，用于取最后几位。	从 1 开始计数。如果要取最后几位，可以用 RIGHT 函数解决
子查询别名	子查询别名	必须有别名
列(别)名为关键字	Oracle 中比如 name，type 这样的关键字可以直接作为列的别名 ,比如 :select xx name from t	需要加 as，比如 select xx as name from t
当前登录用户	SELECT USER FROM DUAL	select current_user

ALL_COL_COMMENTS	通过 SELECT * FROM ALL_COL_COMMENTS 可以获得列注释信息	select s.column_name as COLUMN_NAME, coalesce(col_description(c.oid,ordinal_position) ,s.column_name) as COMMENTS from information_schema.columns s,pg_class c where s.table_name = 'ac01_si' and s.table_name = c.relname and s.table_schema = current_schema() PG 需要通过 col_description 获得列注释信息
修改表字段类型	1.如果字段无数据，可直接修改 2.如果有数据且新类型和原类型兼容，也可以直接修改 3.如果不兼容，可通过对原字段改名，然后增加新字段，再通过 UPDATE 语句对数据进行处理	1.如果新类型和原类型兼容，可直接修改 2.如果不兼容，需要使用 USING 关键字然后提供一个类型转换的表达式
储存过程函数包	Function,procedure package	pgsql 不支持 procedure 和 package，都需要改写成 function,当 package 有全局变量的情况修改起来比较麻烦，我们是用临时表传递的。
cursor 的属性	%FOUND %NOTFOUND %ISOPEN %ROWCOUNT	%FOUND → found %NOTFOUND → not found %ISOPEN → pgsql 不支持 %ROWCOUNT → pgsql 不支持 另外关于 cursor 还发现了其他差异

oracle 与 postgres 对象

Schema

oracle 是按照每个用户为独立的 schema,postgres 是可以独立创建 schema,和用户无关

标识符

Schema、表、列、函数、视图...
oracle 的是大写，除非是双引号括起来
Postgres 统一转换为小写，除非是双引号括起来
关键还是要保持一致

表

创建表一般都兼容，除了
Global temporay table
　　使用 **local temp** 表
分区表
　　使用 **inherent trigger rule** 和 **check constraint pg_pathman**
Initrans,maxextents 存储参数
　　删除他们
Pctfree: 使用 **fillfactor** 填充因子

列

虚拟列：使用视图
数据类型：根据类型映射

Constraint:

主键、外键、唯一键、条件约束、非空约束 都支持

索引：
Btree/descending/ascending :pg 都支持
Reverse key/bitmap/join:pg 没实现

Partition:
Hash、**List**、**range:**都兼容 **pg_pathman** 或触发器实现 **pg10** 自带分区功能

Tablespace:
原理不一样，但工作的效果是一样的

迁移方案

数据
如果类型转换顺利
数据类型字节长度大小正常

使用 **ETL** 方式

1,可以采用自定义导出到 plain-text,csv 固定分隔符的文件
2,采用 copy from 的方式加载
3,或者采用 pg_bulkload 的方式进行加载

加载加速

不要开启 wal 归档
数据导入完毕后在创建索引
唯一键和主键也可以考虑在导入完成后在创建

业务逻辑存储过程迁移

1,return 改为 returns
2,Execute immediate 改为 execute
3,select 没有 into 该为 perform
4,选择一种储存过程语言
create or replace function fn(a inout) returns int as $$ declare ... begin ... end;$$language;
5,%type,%rowtype:能正常功能
6,cursor_name%rowtype:不工作，使用为类型 record
7,refcursors:没有替代方案,使用 returning 特性
8,匿名块： Postrges 不支持
9,在事务中 commit/rollback, pg11 支持事务自治
10,reverse loop:可以采用调换 start/end 的条件解决
For i in reverse 1..10 loop
For i in reverse 10..1 loop

11,触发器
https://www.postgresql.org/docs/11/static/plpgsql-trigger.html
改写为出发函数和触发器的方式解决
Create or replace function trg_fn() returns trigger as $$ ... $$ language xx;
Create trigger tbl_trg before update on table execute procedure trg_fn();
:NEW,:OLD
　　代表触发器使用时捕获的新值和旧值
Updating,insert -> 通过 TG_OP;TG_*等变量获取
在 before trigger 记得返回 return NEW;

12,条件触发器

达到某个条件才执行触发器
pg 可以采用事件触发器

13,储存过程

postgres 只有函数,采用 returns void 的返回值

14,函数

1,Return 改为 returns
2,对于函数的空参数，需要提供双括号()
Create function fn() returns ...
3,默认值 default ,postgres 支持
4,可以返回为类型 record,但是调用者需要知道列的名字
5,可以返回 set of record: returns setof type
oracle 有 table functions

15,Oracle postrges 储存过程迁移注意事项

https://www.postgresql.org/docs/9.6/static/plpgsql-porting.html

1,如果一个 SQL 命令中使用的名字可能是一个表的列名或者是对一个函数中变量的引用，那么 PL/SQL 会将它当作一个列名
2,在 PostgreSQL 中，函数体必须写成字符串文本。因此你需要使用美元符引用或者转义函数体中的单引号
3,数据类型名称常常需要翻译
4,应该用模式把函数组织成不同的分组，而不是用包
5,因为没有包，所以也没有包级别的变量。可以在临时表里保存会话级别的状态
6,带有 REVERSE 的整数 FOR 循环的工作方式不同：PL/SQL 中是从第二个数向第一个数倒数，而 PL/pgSQL 是从第一个数向第二个数倒数，因此在移植时需要交换循环边界
7,查询上的 FOR 循环（不是游标）的工作方式同样不同：目标变量必须已经被声明，而 PL/SQL 总是会隐式地声明它们。但是这样做的优点是在退出循环后，变量值仍然可以访问
8,在使用游标变量方面，存在一些记法差异

16,package

1,一组变量，函数和储存过程
2,采用 schema 对函数分组
3,使用(临时)表替换包内的变量
4,对于 private 函数和变量,没有替代方案
5,包的初始代码,可以在每次调用函数调用一个初始函数
6,local function 函数里面递归调用函数
postgres 不支持，采用正常的函数替换

17,synonyms

1,postgres 不支持这个特性
采用视图解决或包装成函数

18,database links

1,不支持这个特性
2,采用 dblink 插件 和视图解决

19,connect by

采用 with recursive by 改写

20,物化视图

Postgres 支持

21,分区

可以采用 inherent 触发器 规则 条件约束 和 constraint_exlusion pg_pathman 来解决

22,sequence 序列

1,和 oracle 一样的机制
2,nocache 改为 cache 1(或者 remove 这个参数)
3,maxvalue 999999999999999999999999
　　减少限制 最大 9223372036854775807
4,.next,.currval
　　nextval('sequence')
5,order/noorder
　　oracle 需要这个做 cluster/rac 的设置
　　Postgres 没有
6,no {cache|minvalue|maxvalue|cycle}
　　通 no{*} 代替
　　nominvalue 改为 minvalue

23,关联语法
Postgres 提供{left|right|full|out} join oracle 也提供

024,集合操作

UNION 并集
INTERSECT 交集
EXCEPT 差集

25,使用参数名进行函数调用

=>改为 :=
var = fn(c=>10,a=>'xyz',b=>2.5)
改为
var = fn(c := 10,a :='xyz',b:=2.5)

26,Dual

Orafce 兼容 oracle 相关函数
https://postgres.cz/wiki/Oracle_functionality_(en)
https://postgres.cz/wiki/Napi%C5%A1te_si_debugger_PL/pgSQL_aneb_pokro%C4%8Dil%C3%A9_techniky_programov%C3%A1n%C3%AD_v_PostgreSQL

1,Orafce
很多兼容的功能
Dbms_alert
Dbms_pipe
Utl_file
Dbms_output
Dbms_random
Date operations
Dual
To_char() 支持多不同的数据类型
(需要安装插件包,虽然可以兼容,但更建议直接改写,减少依赖**)**

27,Rownum

Row_number()窗口函数
https://www.postgresql.org/docs/11/static/functions-window.html

28,rowid

使用 **ctid** 系统列
不能用作分区键，空间回收 **ctid** 会变化
使用 **oid** 列

迁移工具

1,Ora2pg
2,oracle_fdw

1,Ora2pg 特性介绍

Features included:
- Export full database schema (tables, views, sequences, indexes), with unique, primary, foreign key and check constraints.
- Export grants/privileges for users and groups.
- Export range/list partitions andi sub partitions.
- Export a table selection (by specifying the table names).

- Export Oracle schema to a PostgreSQL 8.4+ schema.
- Export predefined functions, triggers, procedures, packages and package bodies.
- Export full data or following a WHERE clause.
- Full support of Oracle BLOB object as PG BYTEA.
- Export Oracle views as PG tables.
- Export Oracle user defined types.
- Provide some basic automatic conversion of PLSQL code to PLPGSQL.
- Works on any plateform.
- Export Oracle tables as foreign data wrapper tables.
- Export materialized view.
- Show a detailled report of an Oracle database content.
- Migration cost assessment of an Oracle database.
- Migration difficulty level assessment of an Oracle database.
- Migration cost assessment of PL/SQL code from a file.
- Migration cost assessment of Oracle SQL queries stored in a file.
- Generate XML ktr files to be used with Penthalo Data Integrator (Kettle)
- Export Oracle locator and spatial geometries into PostGis.
- Export DBLINK as Oracle FDW.
- Export SYNONYMS as views.
- Export DIRECTORY as external table or directory for external_file extension.
- Full MySQL export just like Oracle database.
- Dispatch a list of SQL orders over multiple PostgreSQL connections
- Perform a diff between Oracle and PostgreSQL database for test purpose.

2,oracle_fdw 特性介绍

特性 **Features:**

1,Uses the standard compliant SQL/MED environment of PostgreSQL 9.1 and above

2,Supports translation of Oracle data types to similar PostgreSQL data types

3,WHERE conditions and ORDER BY expressions are propagated to Oracle where possible

4,Only the required Oracle table columns are fetched

5,EXPLAIN shows the remote query, EXPLAIN VERBOSE the Oracle execution plan

6,Should compile and run on all platforms supported by Oracle Client and PostgreSQL

7,Works with the regular Oracle client and Oracle Instant Client

8,Installable with a single CREATE EXTENSION command

9,Allows foreign tables based on arbitrary Oracle queries

10,Oracle connections are cached for the duration of the PostgreSQL session

11,Supports Oracle external authentication to avoid storing passwords in the database
12,Uses Oracle prefetching for high performance
13,Supports gathering statistics with ANALYZE from PostgreSQL 9.2 on
14,Supports INSERT, UPDATE and DELETE from PostgreSQL 9.3 on
15,Efficient mapping between MDSYS.SDO_GEOMETRY and PostGIS geometry
16,Supports IMPORT FOREIGN SCHEMA from PostgreSQL 9.5 on
17,Propagates 2-way inner joins between foreign tables to Oracle from PostgreSQL 9.6 on

迁移工具使用

Ora2pg 案例

https://github.com/darold/ora2pg
http://ora2pg.darold.net/documentation.html

安装 oracle 略

安装 pg 略

环境

[postgres@DD_DB2 ~]$ cat /etc/redhat-release
CentOS release 6.5 (Final)

[postgres@DD_DB2 ~]$ uname -a
Linux DD_DB2 2.6.32-573.22.1.el6.x86_64 #1 SMP Wed Mar 23 03:35:39 UTC 2016 x86_64 x86_64 x86_64 GNU/Linux

依赖安装

1、安装 perl 依赖

yum install -y perf cpan perl-Time-HiRes

2、安装 oracle-instantclient 安装包

oracle 官网下载 **basic**、**devel**、**sqlplus** 三个 **rpm** 包。
http://www.oracle.com/technetwork/database/database-technologies/instant-client/downloads/index.html

直接下载
http://download.oracle.com/otn/linux/instantclient/122010/oracle-instantclient12.2-basic-12.2.0.1.0-1.x86_64.rpm
http://download.oracle.com/otn/linux/instantclient/121020/oracle-instantclient12.1-devel-12.1.0.2.0-1.x86_64.rpm
http://download.oracle.com/otn/linux/instantclient/122010/oracle-instantclient12.2-sqlplus-12.2.0.1.0-1.x86_64.rpm

安装
rpm -ivh oracle-instantclient*.rpm
rpm -ivh *.rpm
Preparing... ### [100%]
package oracle-instantclient12.2-basic-12.2.0.1.0-1.x86_64 is already installed
package oracle-instantclient12.2-devel-12.2.0.1.0-1.x86_64 is already installed
package oracle-instantclient12.2-sqlplus-12.2.0.1.0-1.x86_64 is already installed

[root@10-0-98-60 ~]# echo "/usr/lib/oracle/12.2/client64/lib" > /etc/ld.so.conf.d/oracle_client.conf
[root@10-0-98-60 ~]# ldconfig
[root@10-0-98-60 ~]#
[root@10-0-98-60 ~]# ldconfig -p|grep oracle
libsqlplusic.so (libc6,x86-64) => /usr/lib/oracle/12.2/client64/lib/libsqlplusic.so
libsqlplus.so (libc6,x86-64) => /usr/lib/oracle/12.2/client64/lib/libsqlplus.so
liboramysql12.so (libc6,x86-64) => /usr/lib/oracle/12.2/client64/lib/liboramysql12.so

```
    libons.so (libc6,x86-64) => /usr/lib/oracle/12.2/client64/lib/libons.so
    libocijdbc12.so (libc6,x86-64) =>
/usr/lib/oracle/12.2/client64/lib/libocijdbc12.so
    libociei.so (libc6,x86-64) => /usr/lib/oracle/12.2/client64/lib/libociei.so
    libocci.so.12.1 (libc6,x86-64) =>
/usr/lib/oracle/12.2/client64/lib/libocci.so.12.1
    libocci.so (libc6,x86-64) => /usr/lib/oracle/12.2/client64/lib/libocci.so
    libnnz12.so (libc6,x86-64) => /usr/lib/oracle/12.2/client64/lib/libnnz12.so
    libmql1.so (libc6,x86-64) => /usr/lib/oracle/12.2/client64/lib/libmql1.so
    libipc1.so (libc6,x86-64) => /usr/lib/oracle/12.2/client64/lib/libipc1.so
    libclntshcore.so.12.1 (libc6,x86-64) =>
/usr/lib/oracle/12.2/client64/lib/libclntshcore.so.12.1
    libclntshcore.so (libc6,x86-64) =>
/usr/lib/oracle/12.2/client64/lib/libclntshcore.so
    libclntsh.so.12.1 (libc6,x86-64) =>
/usr/lib/oracle/12.2/client64/lib/libclntsh.so.12.1
    libclntsh.so (libc6,x86-64) => /usr/lib/oracle/12.2/client64/lib/libclntsh.so
```

测试 **sqlplus username/password@ip:port/sid**

```
sqlplus64 scott/iger@10.0.98.60:1521/orcl
```

3、安装 DBI

```
cpan install DBI
```

4、安装 DBD-Oracle

```
export ORACLE_HOME=/usr/lib/oracle/12.2/client64
export PATH=$ORACLE_HOME/bin:$PATH
export LD_LIBRARY_PATH=$ORACLE_HOME/lib:$LD_LIBRARY_PATH

wget
http://search.cpan.org/CPAN/authors/id/P/PY/PYTHIAN/DBD-Oracle-1.74.tar.gz
tar -zxvf DBD-Oracle-1.74.tar.gz
cd DBD-Oracle-1.74
perl Makefile.PL -l
make && make test
make install
```

5,DBD:PG 安装

下载
http://search.cpan.org/~turnstep/DBD-Pg/
wget http://www.cpan.org/authors/id/T/TU/TURNSTEP/DBD-Pg-3.7.4.tar.gz
tar -zxf DBD-Pg-3.7.4.tar.gz
cd DBD-Pg-3.7.4
perl Makefile.PL
make
make install
history

6、Ora2pg install

wget https://github.com/darold/ora2pg/archive/v18.2.tar.gz
tar -zxvf v18.2.tar.gz
cd ora2pg-18.2/
perl Makefile.PL
make && make install

perl -MCPAN -e shell
cpan> get DBD::mysql
cpan> quit
cd ~/.cpan/build/DBD-mysql*
perl Makefile.PL
make
make install

7、检查

cat check.pl
```
#!/usr/bin/perl
use strict;
use ExtUtils::Installed;

my $inst=ExtUtils::Installed->new();

my @modules = $inst->modules();

foreach(@modules){
```

```
        my $ver = $inst->version($_) || "???";
        printf("%-12s -- %s\n",$_,$ver);
}

perl check.pl
DBD::Oracle   -- 1.74
DBD::Pg       -- 3.7.4
DBD::mysql    -- 4.046
DBI           -- 1.641
Ora2Pg        -- 18.2
Perl          -- 5.10.1
Test::Simple -- 1.302136
```

8、配置 ora2pg

复制
```
cp /etc/ora2pg/ora2pg.conf.dist /etc/ora2pg/ora2pg.conf
```

编辑导出配置文件
```
cd /etc/ora2pg/

cat ora2pg.conf
ORACLE_HOME /usr/local/oracle/product/11.2.0/db_1

#Set Oracle database connection (data source, user, password)
ORACLE_DSN dbi:Oracle:host=10.0.98.60;sid=orcl
ORACLE_USER scott
ORACLE_PWD tiger
SCHEMA    scott
DEBUG     1
ORA_INITIAL_COMMAND
EXPORT_SCHEMA     0
CREATE_SCHEMA     1
COMPILE_SCHEMA    0
TYPE          TABLE   #导出标结果
OUTPUT        output.sql #导出的文件名
```

9、查看 oracle 的表数据

```
SQL> show user;
USER is "SCOTT"
```

```
SQL>          select          SYS_CONTEXT('USERENV','CURRENT_SCHEMA')
CURRENT_SCHEMA from dual;

CURRENT_SCHEMA
--------------------------------------------------------------------------------
SCOTT

SQL> select table_name,tablespace_name from user_tables;

TABLE_NAME                     TABLESPACE_NAME
------------------------------ ------------------------------
DEPT                           USERS
EMP                                 USERS
BONUS                          USERS
SALGRADE                       USERS
TMP                                 USERS
TEST                           USERS
T1                             USERS
```

10、导出表结构

```
[oracle@10-0-98-60 ora2pg]$ ora2pg -c ora2pg.conf
Ora2Pg version: 18.2
Trying to connect to database: dbi:Oracle:host=10.0.98.60;sid=orcl
Isolation level: SET TRANSACTION ISOLATION LEVEL READ COMMITTED
Looking forward functions declaration in schema SCOTT.
Retrieving table information...
[1] Scanning table BONUS (0 rows)...
[2] Scanning table DEPT (4 rows)...
[3] Scanning table EMP (14 rows)...
[4] Scanning table SALGRADE (5 rows)...
[5] Scanning table T1 (1 rows)...
[6] Scanning table TEST (1 rows)...
[7] Scanning table TMP (1 rows)...
Dumping table T1...
Dumping table TEST...
Dumping table DEPT...
Dumping table BONUS...
Dumping table EMP...
Dumping table SALGRADE...
Dumping table TMP...
Dumping RI EMP...
```

问题

```
[oracle@10-0-98-60 ora2pg]$ ora2pg -c ora2pg.conf
Ora2Pg version: 18.2
Trying to connect to database: dbi:Oracle:host=10.0.98.60;sid=orcl
Isolation level: SET TRANSACTION ISOLATION LEVEL READ COMMITTED
Looking forward functions declaration in schema SCOTT.
DBD::Oracle::db prepare failed: ORA-00942: table or view does not exist (DBD
ERROR: error possibly near
```

解决方案

oracle 授予用户 **DBA** 权限

```
SQL> grant dba to scott;
Grant succeeded.
```

11、查看表结构

自动做了表结构数据类型转换

```
[oracle@10-0-98-60 ora2pg]$ more output.sql
-- Generated by Ora2Pg, the Oracle database Schema converter, version 18.2
-- Copyright 2000-2017 Gilles DAROLD. All rights reserved.
-- DATASOURCE: dbi:Oracle:host=10.0.98.60;sid=orcl

SET client_encoding TO 'UTF8';

\set ON_ERROR_STOP ON

SET check_function_bodies = false;
CREATE TABLE dept (
    deptno smallint NOT NULL,
    dname varchar(14),
    loc varchar(13)
) ;
ALTER TABLE dept ADD PRIMARY KEY (deptno);

CREATE TABLE bonus (
    ename varchar(10),
    job varchar(9),
    sal bigint,
    comm bigint
) ;

CREATE TABLE emp (
```

```
    empno smallint NOT NULL,
    ename varchar(10),
    job varchar(9),
    mgr smallint,
    hiredate timestamp,
    sal decimal(7,2),
    comm decimal(7,2),
    deptno smallint
) ;
ALTER TABLE emp ADD PRIMARY KEY (empno);
```

12、更改 ora2pg 配置导出数据

```
cat ora2pg.conf
ORACLE_HOME /usr/local/oracle/product/11.2.0/db_1

#Set Oracle database connection (data source, user, password)
ORACLE_DSN dbi:Oracle:host=10.0.98.60;sid=orcl
ORACLE_USER scott
ORACLE_PWD tiger
SCHEMA    scott
DEBUG          1
ORA_INITIAL_COMMAND
EXPORT_SCHEMA     0
CREATE_SCHEMA     1
COMPILE_SCHEMA    0
TYPE           DATA   #导出数据内容
OUTPUT          data.sql #导出的数据文件名
```

13、导出表内容

```
[oracle@10-0-98-60 ora2pg]$ ora2pg -c ora2pg.conf
Ora2Pg version: 18.2
Trying to connect to database: dbi:Oracle:host=10.0.98.60;sid=orcl
Isolation level: SET TRANSACTION ISOLATION LEVEL READ COMMITTED
Looking forward functions declaration in schema SCOTT.
Retrieving table information...
[1] Scanning table BONUS (0 rows)...
[2] Scanning table DEPT (4 rows)...
[3] Scanning table EMP (14 rows)...
[4] Scanning table SALGRADE (5 rows)...
```

```
[5] Scanning table T1 (1 rows)...
[6] Scanning table TEST (1 rows)...
[7] Scanning table TMP (1 rows)...
Trying to connect to database: dbi:Oracle:host=10.0.98.60;sid=orcl
Isolation level: SET TRANSACTION ISOLATION LEVEL READ COMMITTED
Retrieving partitions information...
Looking how to retrieve data from BONUS...
Fetching all data from BONUS tuples...
DEBUG: Formatting bulk of 10000 data for PostgreSQL.
DEBUG: Creating output for 10000 tuples
Dumping data from BONUS to file: data.sql
Extracted records from table BONUS: total_records = 0 (avg: 0 recs/sec)
[>                                ]  0/26 total rows (0.0%) - (0 sec., avg: 0 recs/sec).
Looking how to retrieve data from DEPT...
Fetching all data from DEPT tuples...
DEBUG: Formatting bulk of 10000 data for PostgreSQL.
DEBUG: Creating output for 10000 tuples
Dumping data from DEPT to file: data.sql
Extracted records from table DEPT: total_records = 4 (avg: 4 recs/sec)
....
[========================>] 51/26 total rows (196.2%) - (1 sec., avg: 51 recs/sec).
Looking how to retrieve data from TMP...
Fetching all data from TMP tuples...
DEBUG: Formatting bulk of 10000 data for PostgreSQL.
DEBUG: Creating output for 10000 tuples
Dumping data from TMP to file: data.sql
Extracted records from table TMP: total_records = 14 (avg: 14 recs/sec)

[========================>] 65/26 total rows (250.0%) - (1 sec., avg: 65 recs/sec).
Restarting sequences
```


14、导入表结构和数据

导入表结构

```
[oracle@10-0-98-60 ora2pg]$ psql -h 127.0.0.1 -U postgres < output.sql
SET
SET
CREATE TABLE
CREATE TABLE
CREATE TABLE
ALTER TABLE
```

CREATE TABLE
CREATE TABLE
ALTER TABLE
CREATE TABLE
CREATE TABLE
ALTER TABLE

导入数据

```
[oracle@10-0-98-60 ora2pg]$ psql -h 127.0.0.1 -U postgres < data.sql
SET
SET
BEGIN
INSERT 0 1
INSERT 0 1
...
INSERT 0 1
INSERT 0 1
COMMIT
```

15、数据校验

oracle

```
SQL> select * from emp;

     EMPNO ENAME      JOB              MGR HIREDATE         SAL       COMM
DEPTNO
---------- ---------- --------- ---------- --------- ---------- ----------
----------
      7369 SMITH      CLERK           7902 17-DEC-80        800
20
      7499 ALLEN      SALESMAN        7698 20-FEB-81                  1600
300       30
      7521 WARD       SALESMAN        7698 22-FEB-81                  1250
500       30
      7566 JONES      MANAGER         7839 02-APR-81       2975
20
      7654 MARTIN     SALESMAN        7698 28-SEP-81                  1250
1400           30
      7698 BLAKE      MANAGER         7839 01-MAY-81       2850
30
      7782 CLARK      MANAGER         7839 09-JUN-81       2450
10
      7788 SCOTT      ANALYST         7566 19-APR-87       3000
20
      7839 KING       PRESIDENT            17-NOV-81       5000
10
```

```
      7844 TURNER      SALESMAN            7698 08-SEP-81       1500         0
       30
      7876 ADAMS       CLERK          7788 23-MAY-87       1100              20
      7900 JAMES       CLERK          7698 03-DEC-81        950              30
      7902 FORD        ANALYST             7566 03-DEC-81       3000
20
      7934 MILLER      CLERK          7782 23-JAN-82       1300              10

postgres
postgres=# select * from emp;
 empno | ename  |      job      | mgr  |         hiredate          |   sal   |  comm
| deptno
-------+--------+-----------+------+---------------------+---------+---------+--------
  7369 | SMITH  | CLERK         | 7902 | 1980-12-17 00:00:00 |  800.00 |         |
20
  7499 | ALLEN  | SALESMAN      | 7698 | 1981-02-20 00:00:00 | 1600.00 |  300.00 |
30
  7521 | WARD   | SALESMAN      | 7698 | 1981-02-22 00:00:00 | 1250.00 |  500.00 |
30
  7566 | JONES  | MANAGER       | 7839 | 1981-04-02 00:00:00 | 2975.00 |         |
20
  7654 | MARTIN | SALESMAN      | 7698 | 1981-09-28 00:00:00 | 1250.00 | 1400.00 |
30
  7698 | BLAKE  | MANAGER       | 7839 | 1981-05-01 00:00:00 | 2850.00 |         |
30
  7782 | CLARK  | MANAGER       | 7839 | 1981-06-09 00:00:00 | 2450.00 |         |
10
  7788 | SCOTT  | ANALYST       | 7566 | 1987-04-19 00:00:00 | 3000.00 |         |
20
  7839 | KING   | PRESIDENT |          | 1981-11-17 00:00:00 | 5000.00 |         |
10
  7844 | TURNER | SALESMAN      | 7698 | 1981-09-08 00:00:00 | 1500.00 |    0.00 |
30
  7876 | ADAMS  | CLERK         | 7788 | 1987-05-23 00:00:00 | 1100.00 |         |
20
  7900 | JAMES  | CLERK         | 7698 | 1981-12-03 00:00:00 |  950.00 |         |
30
  7902 | FORD   | ANALYST       | 7566 | 1981-12-03 00:00:00 | 3000.00 |         |
20
  7934 | MILLER | CLERK         | 7782 | 1982-01-23 00:00:00 | 1300.00 |         |
10
(14 rows)
```

16、连接 Postgres

PG_DSN dbi:Pg:dbname=postgres;host=10.0.98.60;port=5432
PG_USER postgres
PG_PWD 123456
OUTPUT output.sql

17、更多用法 创建迁移模板

创建迁移模板需要 **--project_base and --init_project** 这两个参数
ora2pg --project_base /app/migration/ --init_project test_project

```
Creating project test_project.
/app/migration/test_project/
        schema/
                dblinks/
                directories/
                functions/
                grants/
                mviews/
                packages/
                partitions/
                procedures/
                sequences/
                synonyms/
                tables/
                tablespaces/
                triggers/
                types/
                views/
        sources/
                functions/
                mviews/
                packages/
                partitions/
                procedures/
                triggers/
                types/
                views/
        data/
        config/
        reports/
```

Generating generic configuration file
Creating script export_schema.sh to automate all exports.
Creating script import_all.sh to automate all imports.

Oracle_fdw 案例(略)

特性 Features:

1,Uses the standard compliant SQL/MED environment of PostgreSQL 9.1 and above
2,Supports translation of Oracle data types to similar PostgreSQL data types
3,WHERE conditions and ORDER BY expressions are propagated to Oracle where possible
4,Only the required Oracle table columns are fetched
5,EXPLAIN shows the remote query, EXPLAIN VERBOSE the Oracle execution plan
6,Should compile and run on all platforms supported by Oracle Client and PostgreSQL
7,Works with the regular Oracle client and Oracle Instant Client
8,Installable with a single CREATE EXTENSION command
9,Allows foreign tables based on arbitrary Oracle queries
10,Oracle connections are cached for the duration of the PostgreSQL session
11,Supports Oracle external authentication to avoid storing passwords in the database
12,Uses Oracle prefetching for high performance
13,Supports gathering statistics with ANALYZE from PostgreSQL 9.2 on
14,Supports INSERT, UPDATE and DELETE from PostgreSQL 9.3 on
15,Efficient mapping between MDSYS.SDO_GEOMETRY and PostGIS geometry
16,Supports IMPORT FOREIGN SCHEMA from PostgreSQL 9.5 on
17,Propagates 2-way inner joins between foreign tables to Oracle from PostgreSQL 9.6 on

oracle_fdw 安装使用

oracle_fdw 安装之前需要先安装 oracle client (略)
https://github.com/laurenz/oracle_fdw/wiki/Help-for-installing-and-configuring-Oracle

将 oracle11 的库加入路径，否则会出错

```
echo "/usr/local/oracle/product/11.2.0/db_1/lib" >> /etc/ld.so.conf.d/oracle_client.conf
ldconfig
[root@10-0-98-60 ~]# ldconfig -p |grep oracle
    libxdb.so (libc6,x86-64) => /usr/local/oracle/product/11.2.0/db_1/lib/libxdb.so
    libuini11.so (libc6,x86-64) => /usr/local/oracle/product/11.2.0/db_1/lib/libuini11.so
    libsrvm11.so0 (libc6,x86-64) => /usr/local/oracle/product/11.2.0/db_1/lib/libsrvm11.so0
    libsrvm11.so (libc6,x86-64) => /usr/local/oracle/product/11.2.0/db_1/lib/libsrvm11.so
    libsrvmocr11.so (libc6,x86-64) => /usr/local/oracle/product/11.2.0/db_1/lib/libsrvmocr11.so
    libsrvmhas11.so (libc6,x86-64) => /usr/local/oracle/product/11.2.0/db_1/lib/libsrvmhas11.so
    libsqora.so.11.1 (libc6,x86-64) => /usr/local/oracle/product/11.2.0/db_1/lib/libsqora.so.11.1
    libsqlplusic.so (libc6,x86-64) => /usr/lib/oracle/12.2/client64/lib/libsqlplusic.so
    libsqlplus.so (libc6,x86-64) => /usr/lib/oracle/12.2/client64/lib/libsqlplus.so
    libsqlplus.so (libc6,x86-64) => /usr/local/oracle/product/11.2.0/db_1/lib/libsqlplus.so
    libskvol11.so (libc6,x86-64) => /usr/local/oracle/product/11.2.0/db_1/lib/libskvol11.so
    libskgxp11.so (libc6,x86-64) => /usr/local/oracle/product/11.2.0/db_1/lib/libskgxp11.so
    libskgxpr.so (libc6,x86-64) => /usr/local/oracle/product/11.2.0/db_1/lib/libskgxpr.so
    libskgxpg.so (libc6,x86-64) => /usr/local/oracle/product/11.2.0/db_1/lib/libskgxpg.so
    libskgxpd.so (libc6,x86-64) => /usr/local/oracle/product/11.2.0/db_1/lib/libskgxpd.so
    libskgxpcompat.so (libc6,x86-64) => /usr/local/oracle/product/11.2.0/db_1/lib/libskgxpcompat.so
    libskgxn2.so (libc6,x86-64) => /usr/local/oracle/product/11.2.0/db_1/lib/libskgxn2.so
```

```
    libskgxns.so                    (libc6,x86-64)                  =>
/usr/local/oracle/product/11.2.0/db_1/lib/libskgxns.so
    librdjni11.so                   (libc6,x86-64)                  =>
/usr/local/oracle/product/11.2.0/db_1/lib/librdjni11.so
    libqsmashr.so                   (libc6,x86-64)                  =>
/usr/local/oracle/product/11.2.0/db_1/lib/libqsmashr.so
    libowm2.so                      (libc6,x86-64)                  =>
/usr/local/oracle/product/11.2.0/db_1/lib/libowm2.so
    libosbws11.so                   (libc6,x86-64)                  =>
/usr/local/oracle/product/11.2.0/db_1/lib/libosbws11.so
    libordim11.so                   (libc6,x86-64)                  =>
/usr/local/oracle/product/11.2.0/db_1/lib/libordim11.so
```

下载

https://github.com/laurenz/oracle_fdw/releases/tag/ORACLE_FDW_2_0_0

pg 命令 **pg_config** 的环境变量要配置好

安装

```
unzip oracle_fdw-ORACLE_FDW_2_0_0.zip
cd oracle_fdw-ORACLE_FDW_2_0_0
make
make install
```

登陆查看可用插件

```
[postgres@10-0-98-60 oracle_fdw-ORACLE_FDW_2_0_0]$ psql -h 127.0.0.1
psql (9.6.1)
Type "help" for help.

postgres=# select * from pg_available_extensions;
    name      | default_version | installed_version |                        comment
--------------+-----------------+-------------------+-----------------------------------------
 oracle_fdw | 1.1             |                   | foreign data wrapper for
```

Oracle access

```
postgres=# create extension oracle_fdw ;
CREATE EXTENSION
```

查看 oracle_fdw 相关函数

```
postgres=# \df *oracle*
                                        List of functions
 Schema |             Name             | Result data type |     Argument data types     |  Type
--------+------------------------------+------------------+-----------------------------+--------
 public | oracle_close_connections     | void             |                             | normal
 public | oracle_diag                  | text             | name DEFAULT NULL::name     | normal
 public | oracle_fdw_handler           | fdw_handler      |                             | normal
 public | oracle_fdw_validator         | void             | text[], oid                 | normal

 FUNCTION oracle_fdw_handler() RETURNS fdw_handler
 FUNCTION oracle_fdw_validator(text[], oid) RETURNS void
These functions are the handler and the validator function necessary to create
a foreign data wrapper.

FUNCTION oracle_close_connections() RETURNS void

This function can be used to close all open Oracle connections in this session.
FUNCTION oracle_diag(name DEFAULT NULL) RETURNS text

This function is useful for diagnostic purposes only.
It will return the versions of oracle_fdw, PostgreSQL server and Oracle client.
If called with no argument or NULL, it will additionally return the values of
some environment variables used for establishing Oracle connections.
If called with the name of a foreign server, it will additionally return
the Oracle server version.
```

使用案例

```
CREATE EXTENSION oracle_fdw;
```

```
CREATE SERVER pgoradb FOREIGN DATA WRAPPER oracle_fdw OPTIONS (dbserver '//10.0.98.60:1521/orcl');


GRANT USAGE ON FOREIGN SERVER pgoradb TO postgres;

CREATE USER MAPPING FOR postgres SERVER pgoradb OPTIONS (user 'scott', password 'tiger');

drop FOREIGN TABLE foreign_emp;
CREATE FOREIGN TABLE foreign_emp
(
   empno smallint NOT NULL,
   ename character varying(10),
   job character varying(9),
   mgr smallint,
   hiredate timestamp without time zone,
   sal numeric(7,2),
   comm numeric(7,2),
   deptno smallint
) SERVER pgoradb OPTIONS (schema 'SCOTT', table 'EMP');

drop FOREIGN TABLE foreign_dept;
CREATE FOREIGN TABLE foreign_dept
(
    deptno smallint NOT NULL,
   dname character varying(14),
   loc character varying(13)
) SERVER pgoradb OPTIONS (schema 'SCOTT', table 'EMP');
```


数据校验

```
postgres=# select * from foreign_emp;
 empno | ename  |     job      | mgr  |        hiredate         |   sal   |  comm   | deptno
-------+--------+--------------+------+-------------------------+---------+---------+--------
  7369 | SMITH  | CLERK        | 7902 | 1980-12-17 00:00:00 |  800.00 |         |     20
  7499 | ALLEN  | SALESMAN     | 7698 | 1981-02-20 00:00:00 | 1600.00 |  300.00 |     30
  7521 | WARD   | SALESMAN     | 7698 | 1981-02-22 00:00:00 | 1250.00 |  500.00 |     30
  7566 | JONES  | MANAGER      | 7839 | 1981-04-02 00:00:00 | 2975.00 |         |
```

```
20
  7654 | MARTIN | SALESMAN  | 7698 | 1981-09-28 00:00:00 | 1250.00 | 1400.00 |
30
  7698 | BLAKE  | MANAGER   | 7839 | 1981-05-01 00:00:00 | 2850.00 |         |
30
  7782 | CLARK  | MANAGER   | 7839 | 1981-06-09 00:00:00 | 2450.00 |         |
10
  7788 | SCOTT  | ANALYST   | 7566 | 1987-04-19 00:00:00 | 3000.00 |         |
20
  7839 | KING   | PRESIDENT |      | 1981-11-17 00:00:00 | 5000.00 |         |
10
  7844 | TURNER | SALESMAN  | 7698 | 1981-09-08 00:00:00 | 1500.00 |    0.00 |
30
  7876 | ADAMS  | CLERK     | 7788 | 1987-05-23 00:00:00 | 1100.00 |         |
20
  7900 | JAMES  | CLERK     | 7698 | 1981-12-03 00:00:00 |  950.00 |         |
30
  7902 | FORD   | ANALYST   | 7566 | 1981-12-03 00:00:00 | 3000.00 |         |
20
  7934 | MILLER | CLERK     | 7782 | 1982-01-23 00:00:00 | 1300.00 |         |
10
(14 rows)

Time: 6.600 ms
postgres=# select * from emp;
 empno | ename  |    job    | mgr  |      hiredate       |   sal   |  comm
| deptno
-------+--------+-----------+------+---------------------+---------+---------+--------
  7369 | SMITH  | CLERK     | 7902 | 1980-12-17 00:00:00 |  800.00 |         |
20
  7499 | ALLEN  | SALESMAN  | 7698 | 1981-02-20 00:00:00 | 1600.00 |  300.00 |
30
  7521 | WARD   | SALESMAN  | 7698 | 1981-02-22 00:00:00 | 1250.00 |  500.00 |
30
  7566 | JONES  | MANAGER   | 7839 | 1981-04-02 00:00:00 | 2975.00 |         |
20
  7654 | MARTIN | SALESMAN  | 7698 | 1981-09-28 00:00:00 | 1250.00 | 1400.00 |
30
  7698 | BLAKE  | MANAGER   | 7839 | 1981-05-01 00:00:00 | 2850.00 |         |
30
  7782 | CLARK  | MANAGER   | 7839 | 1981-06-09 00:00:00 | 2450.00 |         |
10
  7788 | SCOTT  | ANALYST   | 7566 | 1987-04-19 00:00:00 | 3000.00 |         |
20
```

```
  7839 | KING     | PRESIDENT |          | 1981-11-17 00:00:00 | 5000.00 |              |
10
  7844 | TURNER | SALESMAN    | 7698 | 1981-09-08 00:00:00 | 1500.00 |      0.00 |
30
  7876 | ADAMS    | CLERK       | 7788 | 1987-05-23 00:00:00 | 1100.00 |              |
20
  7900 | JAMES    | CLERK       | 7698 | 1981-12-03 00:00:00 |   950.00 |              |
30
  7902 | FORD     | ANALYST     | 7566 | 1981-12-03 00:00:00 | 3000.00 |              |
20
  7934 | MILLER | CLERK        | 7782 | 1982-01-23 00:00:00 | 1300.00 |              |
10
(14 rows)

Time: 0.856 ms
postgres=# select * from foreign_emp where empno=7369;
 empno | ename |   job   | mgr  |         hiredate          |   sal    | comm | deptno
-------+-------+-------+------+---------------------+--------+------+--------
  7369 | SMITH | CLERK | 7902 | 1980-12-17 00:00:00 | 800.00 |      |      20
(1 row)

Time: 6.688 ms
postgres=# explain select * from foreign_emp where empno=7369;

QUERY PLAN
------------------------------------------------------------------------------------------------------------------------
------------------
------------------------------------------------------------------
 Foreign Scan on foreign_emp  (cost=10000.00..20000.00 rows=1000
width=116)
   Oracle query: SELECT /*441e3d24c66c24e41956dfbc10b97948*/ r1."EMPNO",
r1."ENAME", r1."JOB", r1."MGR", r1."HIREDATE", r1."SAL", r1."CO
MM", r1."DEPTNO" FROM "SCOTT"."EMP" r1 WHERE (r1."EMPNO" = 7369)
(2 rows)

Time: 1.615 ms

postgres=# select a.*,b.* from emp a inner join dept b on a.deptno=b.deptno
where a.empno=7369;
 empno | ename |   job   | mgr  |         hiredate          |   sal    | comm | deptno |
deptno |   dname    |   loc
-------+-------+-------+------+---------------------+--------+------+--------+--------+----------+--------
  7369 | SMITH | CLERK | 7902 | 1980-12-17 00:00:00 | 800.00 |      |      20 |
20 | RESEARCH | DALLAS
```

(1 row)

Time: 1.683 ms

postgres=# SELECT oracle_diag('pgoradb');
oracle_diag
---
oracle_fdw 2.0.0, PostgreSQL 9.6.1, Oracle client 11.2.0.1.0, Oracle server 11.2.0.1.0

oracle 转 postgres 函数兼容

1 Connect by

PostgreSQL 实现 Oracle 的 connect_by

with 实现层级数据的查询
PostgreSQL 不支持 Oracle 中的 connect by 语法，而即使是 edb 对 connect by 的兼容有限，edb 不支持的点：
函数 sys_connect_by_path
在 SELECT 表达式中使用 PRIOR 限定符
CONNECT BY 有多个表达式
CONNECT_BY_ROOT 表达式
Oracle 中还有以下：
connect_by_is_leaf :connect_by_isleaf is a new operator that comes with Oracle 10g and enhances the ability to perform hierarchical queries.
connect_by_iscycle: connect_by_is_cycle is a new operator that comes with Oracle 10g and enhances the ability to perform hierarchical queries.
解决方案如下，
假设 Oracle 中有表：

```
CREATE TABLE sys_cbp_test
  (id INTEGER NOT NULL PRIMARY KEY,parent_id INTEGER
  );
```

生成测试数据：

```
INSERT INTO sys_cbp_test
SELECT 1, NULL FROM dual
UNION ALL
SELECT 2, 1 FROM dual
UNION ALL
SELECT 3, 2 FROM dual
UNION ALL
SELECT 4, 3 FROM dual
```

```
UNION ALL
SELECT 5, 1 FROM dual
UNION ALL
SELECT 6, 5 FROM dual
UNION ALL
SELECT 7, 2 FROM dual
UNION ALL
SELECT 20, NULL FROM dual
UNION ALL
SELECT 21, 20 FROM dual
UNION ALL
SELECT 22, 21 FROM dual;
```

表的内容如下：

```
SQL> select * from sys_cbp_test;

 ID   PARENT_ID
--- ----------
  1
  2          1
  3          2
  4          3
  5          1
  6          5
  7          2
 20
 21         20
 22         21

10 rows selected.
```

Oracle 中的查询语句为：

```
SELECT id,
  prior id,
  parent_id,
  Level,
  sys_connect_by_path (TO_CHAR (id), '/') AS Path,
  CONNECT_BY_ROOT   id AS root
FROM sys_cbp_test
  START WITH parent_id     IS NULL
  CONNECT BY prior id = parent_id;
```

Oracle 中查询的结果如下：

```
 ID      PRIORID   PARENT_ID         LEVEL PATH          ROOT
--- ---------- ---------- ---------- ---------- ----------
  1                     1 /1                 1
  2         1          1         2 /1/2              1
```

```
  3        2        2        3 /1/2/3          1
  4        3        3        4 /1/2/3/4            1
  7        2        2        3 /1/2/7          1
  5        1        1        2 /1/5            1
  6        5        5        3 /1/5/6          1
 20                 1 /20             20
 21       20           20       2 /20/21          20
 22       21           21       3 /20/21/22           20
```

10 rows selected.

PostgreSQL 中生成测试数据的 SQL 如下：

```
INSERT INTO sys_cbp_test VALUES (1, NULL),(2, 1),(3, 2),(4, 3),(5, 1),(6, 5),(7,
2),(20, NULL),(21, 20),(22, 21);
```

PostgreSQL 中的 SQL 如下：

```
WITH RECURSIVE x (id, prior_id, parent_id, level, path, root) AS
(SELECT id, NULL::INT AS prior_id, NULL::INT AS parent_id, 1, array[id], id as
root
    FROM sys_cbp_test
  WHERE parent_id IS NULL
  UNION ALL
 SELECT b.id, x.id AS prior_id, b.parent_id, level+1, x.path|| b.id, x.root
    FROM x, sys_cbp_test b
  WHERE x.id = b.parent_id
)
SELECT id, prior_id, parent_id, level, '/'|| array_to_string (path, '/') AS path, root
  FROM x;
```

PostgreSQL 看到的结果如下：

```
 id | prior_id | parent_id | level |     path      | root
----+----------+-----------+-------+-----------+------
  1 |          |           |     1 | /1            |    1
 20 |          |           |     1 | /20           |   20
  2 |        1 |         1 |     2 | /1/2          |    1
  5 |        1 |         1 |     2 | /1/5          |    1
 21 |       20 |        20 |     2 | /20/21        |   20
  3 |        2 |         2 |     3 | /1/2/3        |    1
  6 |        5 |         5 |     3 | /1/5/6        |    1
  7 |        2 |         2 |     3 | /1/2/7        |    1
 22 |       21 |        21 |     3 | /20/21/22     |   20
  4 |        3 |         3 |     4 | /1/2/3/4      |    1
(10 rows)
```

行的顺序与 Oracle 不一样，但对于关系型数据库一般是不保证行的顺序的，如果要保证，需要排序，如：

```
WITH RECURSIVE x (id, prior_id, parent_id, level, path, root) AS
```

```
(SELECT id, NULL::INT AS prior_id, NULL::INT AS parent_id, 1, array[id], id as
root
    FROM sys_cbp_test
   WHERE parent_id IS NULL
   UNION ALL
 SELECT b.id, x.id AS prior_id, b.parent_id, level+1, x.path|| b.id, x.root
    FROM x, sys_cbp_test b
   WHERE x.id = b.parent_id
)
SELECT id, prior_id, parent_id, level, '/'|| array_to_string (path, '/') AS path, root
   FROM x
 ORDER BY id NULLS FIRST
   ;

 id | prior_id | parent_id | level |     path       | root
----+----------+-----------+-------+-----------+------
  1 |          |           |     1 | /1             |    1
  2 |        1 |         1 |     2 | /1/2           |    1
  3 |        2 |         2 |     3 | /1/2/3         |    1
  4 |        3 |         3 |     4 | /1/2/3/4       |    1
  5 |        1 |         1 |     2 | /1/5           |    1
  6 |        5 |         5 |     3 | /1/5/6         |    1
  7 |        2 |         2 |     3 | /1/2/7         |    1
 20 |          |           |     1 | /20            |   20
 21 |       20 |        20 |     2 | /20/21         |   20
 22 |       21 |        21 |     3 | /20/21/22      |   20
(10 rows)
```

2 ratio

Oracle 的分析函数 RATIO_TO_REPORT()是用于计算当前值在分组内的占比的
RATIO_TO_REPORT is an analytic function. It computes the ratio of a value to the sum of a set of values.
If expr evaluates to null, then the ratio-to-report value also evaluates to null.
PostgreSQL 也支持窗口查询，但是没有提供这个分析函数，不过我们知道它是干什么的，当然就知道如何写 SQL 来实现同样的目的了。

Oracle 例子

```
SELECT last_name, salary, RATIO_TO_REPORT(salary) OVER () AS rr
    FROM employees
    WHERE job_id = 'PU_CLERK';
```

```
LAST_NAME                 SALARY          RR
------------------------- ---------- ----------
Khoo                       3100 .223021583
Baida                     2900 .208633094
Tobias                    2800 .201438849
Himuro                     2600   .18705036
Colmenares                 2500 .179856115


drop table t1;
create table t1(
id serial not null,
val integer not null,
category character varying(1)
);
insert into t1(val,category) values(10,'a'),(10,'a'),(20,'a'),(20,'b');

select id,val,1.0*val/nullif(sum(val) over(),0) as ratio_to_report from t1;
1;10;0.16666666666666666667
2;10;0.16666666666666666667
3;20;0.33333333333333333333
4;20;0.33333333333333333333

select id,val,category,1.0*val/nullif(sum(val) over(partition by category),0) as
ratio_to_report from t1;
1;10;"a";0.25000000000000000000
2;10;"a";0.25000000000000000000
3;20;"a";0.50000000000000000000
4;20;"b";1.00000000000000000000
```

3,eval

4,decode

```
Oracle:
SELECT product_id,
DECODE (warehouse_id, 1, 'Southlake',
2, 'San Francisco',
3, 'New Jersey',
4, 'Seattle',
```

'Non-domestic') quantity_on_hand
FROM inventories

Postgres
SELECT a,
CASE WHEN a=1 THEN 'one'
WHEN a=2 THEN 'two'
ELSE 'other'
END
FROM test

5,Insert all

```
INSERT ALL
WHEN order_total <= 100000 THEN
INTO small_orders
WHEN order_total > 100000 AND order_total <= 200000 THEN
INTO medium_orders
WHEN order_total = 500000 THEN
INTO special_orders
WHEN order_total > 200000 THEN
INTO large_orders
ELSE
INTO large_orders
SELECT order_id, order_total, sales_rep_id, customer_id FROM orders

drop TABLE char_size_test;
CREATE TABLE char_size_test (
id integer,
size CHAR(10)
);

drop TABLE varchar_size_test;
CREATE TABLE varchar_size_test(
id integer,
size varchar(10)
);

WITH test_data AS (
SELECT t,t as i FROM generate_series (1, 10000) as t
),char_data_insert AS (
INSERT INTO char_size_test(id,size) SELECT t,i FROM test_data where t<5000
)
INSERT INTO varchar_size_test(id,size) SELECT t,i FROM test_data where
```

```
t>=5000;


postgres=# select min(id),max(id) from char_size_test;
      min        |       max
-----------+-----------
 1               | 999
(1 row)

Time: 56.353 ms
postgres=# select min(id),max(id) from varchar_size_test;
   min    | max
-------+------
 10000 | 9999
(1 row)

Time: 4.383 ms
```

6,instr

instr 函数模仿 Oracle 的对应函数

```
-- 语法: instr(string1, string2, [n], [m]) 其中 [] 表示可选参数。
--
-- 从第 n 个字符开始搜索 string2 在 string1 中的第 m 次出现。如果 n 是负的，反向搜索。
-- 如果 m 没有被传递，假定为 1（从第一个字符开始搜索）。
--

CREATE FUNCTION instr(varchar, varchar) RETURNS integer AS $$
DECLARE
    pos integer;
BEGIN
    pos:= instr($1, $2, 1);
    RETURN pos;
END;
$$ LANGUAGE plpgsql STRICT IMMUTABLE;


CREATE FUNCTION instr(string varchar, string_to_search varchar, beg_index integer)
RETURNS integer AS $$
DECLARE
    pos integer NOT NULL DEFAULT 0;
```

```
    temp_str varchar;
    beg integer;
    length integer;
    ss_length integer;
BEGIN
    IF beg_index > 0 THEN
        temp_str := substring(string FROM beg_index);
        pos := position(string_to_search IN temp_str);

        IF pos = 0 THEN
            RETURN 0;
        ELSE
            RETURN pos + beg_index - 1;
        END IF;
    ELSIF beg_index < 0 THEN
        ss_length := char_length(string_to_search);
        length := char_length(string);
        beg := length + beg_index - ss_length + 2;

        WHILE beg > 0 LOOP
            temp_str := substring(string FROM beg FOR ss_length);
            pos := position(string_to_search IN temp_str);

            IF pos > 0 THEN
                RETURN beg;
            END IF;

            beg := beg - 1;
        END LOOP;

        RETURN 0;
    ELSE
        RETURN 0;
    END IF;
END;
$$ LANGUAGE plpgsql STRICT IMMUTABLE;


CREATE FUNCTION instr(string varchar, string_to_search varchar,
                      beg_index integer, occur_index integer)
RETURNS integer AS $$
DECLARE
    pos integer NOT NULL DEFAULT 0;
    occur_number integer NOT NULL DEFAULT 0;
```

```
    temp_str varchar;
    beg integer;
    i integer;
    length integer;
    ss_length integer;
BEGIN
    IF beg_index > 0 THEN
        beg := beg_index;
        temp_str := substring(string FROM beg_index);

        FOR i IN 1..occur_index LOOP
            pos := position(string_to_search IN temp_str);

            IF i = 1 THEN
                beg := beg + pos - 1;
            ELSE
                beg := beg + pos;
            END IF;

            temp_str := substring(string FROM beg + 1);
        END LOOP;

        IF pos = 0 THEN
            RETURN 0;
        ELSE
            RETURN beg;
        END IF;
    ELSIF beg_index < 0 THEN
        ss_length := char_length(string_to_search);
        length := char_length(string);
        beg := length + beg_index - ss_length + 2;

        WHILE beg > 0 LOOP
            temp_str := substring(string FROM beg FOR ss_length);
            pos := position(string_to_search IN temp_str);

            IF pos > 0 THEN
                occur_number := occur_number + 1;

                IF occur_number = occur_index THEN
                    RETURN beg;
                END IF;
            END IF;
```

```
            beg := beg - 1;
        END LOOP;

        RETURN 0;
    ELSE
        RETURN 0;
    END IF;
END;
$$ LANGUAGE plpgsql STRICT IMMUTABLE;
```

7,rownum

Oracle ROWNUM 是一个虚拟列，每输出一行递增 **1**。
Oracle rownum
通常被用于 **LIMIT** 输出记录数。

```
SELECT ROWNUM, empno, ename, job FROM emp WHERE ROWNUM < 5 ORDER BY ename;

 rownum | empno | ename |     job
--------+-------+-------+----------
      2 |  7499 | ALLEN | SALESMAN
      4 |  7566 | JONES | MANAGER
      1 |  7369 | SMITH | CLERK
      3 |  7521 | WARD  | SALESMAN
(4 rows)
```

或者用于生成序列值。

```
ALTER TABLE jobhist ADD seqno NUMBER(3);
UPDATE jobhist SET seqno = ROWNUM;
SELECT seqno, empno, TO_CHAR(startdate,'DD-MON-YY') AS start, job FROM jobhist;

 seqno | empno |    start    |      job
-------+-------+-------------+-----------
     1 |  7369 | 17-DEC-80 | CLERK
     2 |  7499 | 20-FEB-81 | SALESMAN
     3 |  7521 | 22-FEB-81 | SALESMAN
     4 |  7566 | 02-APR-81 | MANAGER
     5 |  7654 | 28-SEP-81 | SALESMAN
     6 |  7698 | 01-MAY-81 | MANAGER
     7 |  7782 | 09-JUN-81 | MANAGER
     8 |  7788 | 19-APR-87 | CLERK
```

```
        9 |    7788 | 13-APR-88 | CLERK
       10 |    7788 | 05-MAY-90 | ANALYST
       11 |    7839 | 17-NOV-81 | PRESIDENT
       12 |    7844 | 08-SEP-81 | SALESMAN
       13 |    7876 | 23-MAY-87 | CLERK
       14 |    7900 | 03-DEC-81 | CLERK
       15 |    7900 | 15-JAN-83 | CLERK
       16 |    7902 | 03-DEC-81 | ANALYST
       17 |    7934 | 23-JAN-82 | CLERK
(17 rows)
```

PostgreSQL rownum

PostgreSQL 目前没有 **rownum** 虚拟列，但是实现同样的功能确很容易：

1、输出行号，使用临时序列

```
postgres=# create temp sequence if not exists tmp_seq;

postgres=# alter sequence tmp_seq restart with 1;

postgres=# select nextval('tmp_seq') as rownum, * from test limit 10;
 rownum | id | info |          crt_time
--------+----+------+----------------------------
      1 |  1 | test | 2018-01-24 11:06:24.882708
      2 |  2 | test | 2018-01-24 11:06:24.882708
      3 |  3 | test | 2018-01-24 11:06:24.882708
      4 |  4 | test | 2018-01-24 11:06:24.882708
      5 |  5 | test | 2018-01-24 11:06:24.882708
      6 |  6 | test | 2018-01-24 11:06:24.882708
      7 |  7 | test | 2018-01-24 11:06:24.882708
      8 |  8 | test | 2018-01-24 11:06:24.882708
      9 |  9 | test | 2018-01-24 11:06:24.882708
     10 | 10 | test | 2018-01-24 11:06:24.882708
(10 rows)
```

2、输出行号，使用窗口函数

```
postgres=# select row_number() over () as rownum, * from test limit 10;
 rownum | id | info |          crt_time
--------+----+------+----------------------------
      1 |  1 | test | 2018-01-24 11:06:24.882708
      2 |  2 | test | 2018-01-24 11:06:24.882708
      3 |  3 | test | 2018-01-24 11:06:24.882708
      4 |  4 | test | 2018-01-24 11:06:24.882708
      5 |  5 | test | 2018-01-24 11:06:24.882708
      6 |  6 | test | 2018-01-24 11:06:24.882708
```

```
  7 |   7 | test | 2018-01-24 11:06:24.882708
  8 |   8 | test | 2018-01-24 11:06:24.882708
  9 |   9 | test | 2018-01-24 11:06:24.882708
 10 | 10 | test | 2018-01-24 11:06:24.882708
(10 rows)
```

3、LIMIT，直接语法支持

```
postgres=# select * from test limit 10;
 id | info |            crt_time
----+------+----------------------------
  1 | test | 2018-01-24 11:06:24.882708
  2 | test | 2018-01-24 11:06:24.882708
  3 | test | 2018-01-24 11:06:24.882708
  4 | test | 2018-01-24 11:06:24.882708
  5 | test | 2018-01-24 11:06:24.882708
  6 | test | 2018-01-24 11:06:24.882708
  7 | test | 2018-01-24 11:06:24.882708
  8 | test | 2018-01-24 11:06:24.882708
  9 | test | 2018-01-24 11:06:24.882708
 10 | test | 2018-01-24 11:06:24.882708
(10 rows)
```


8,synonym 匿名

匿名语法如下

```
CREATE [ OR REPLACE ] [ PUBLIC ] SYNONYM
    [ schema. ]synonym
    FOR [ schema. ]object [ @ dblink ] ;
```

https://docs.oracle.com/cd/B19306_01/server.102/b14200/statements_7001.htm

1、表
代码写死了 b.tbl123：

```
create table a.tbl(id int);
```

通过视图，

```
create view b.tbl123 as select * from a.tbl;
```

这种简单视图，支持增删改查，和直接使用 a.tbl 是一样的。
相当于建立了 a.tbl 的 b.tbl123 匿名。
通过 search_path，

如果对象名没变，只是在不同的 schema 下，使用 search_path 是最通用的方法：

```
set search_path=a,"$user",public;
```

那么会先访问 a 这个 schema 下的对象。
2、函数
代码写死了 b.fun123：
```
create or replace function a.fun(int) returns int as $$
....
$$ language plpgsql strict;
```
通过函数嵌套，

```
create or replace function b.fun123(int) returns int as $$
select a.fun($1);
$$ language sql strict;
```
通过 search_path，与 1 类似。

3、视图
通过视图，与 1 类似。
通过 search_path

4、物化视图
通过视图，与 1 类似。
通过 search_path，与 1 类似。

5、DBLINK
通过重定义一样的 dblink。
通过 search_path，与 1 类似。

6、外部表
通过视图，与 1 类似。
通过 search_path，与 1 类似。

7、自定义类型
通过重定义一样的类型。
通过 search_path，与 1 类似。

9,order by INT position

```
ORDER [ SIBLINGS ] BY
{ expr | position | c_alias }
[ ASC | DESC ]
[ NULLS FIRST | NULLS LAST ]
  [, { expr | position | c_alias }
      [ ASC | DESC ]
      [ NULLS FIRST | NULLS LAST ]
  ]...
```

按 表达式、列别名、**select** 位置排序。

PostgreSQL 也支持这种语法。

PostgreSQL order by 支持
PostgreSQL 天然支持 **order by [**字段、表达式、位置**]**。

[ORDER BY expression [ASC | DESC | USING operator] [NULLS { FIRST | LAST }] [, ...]]

Each expression can be the name or ordinal number of an output column (SELECT list item), or it can be an arbitrary expression formed from input-column values.
例子

1、按别名排序

```
postgres=# explain select relpages as ooo,* from pg_class order by ooo;
                            QUERY PLAN
------------------------------------------------------------------
 Sort  (cost=71.81..73.32 rows=602 width=737)
   Sort Key: relpages
   ->  Seq Scan on pg_class  (cost=0.00..44.02 rows=602 width=737)
(3 rows)
```

2、按 **SELECT** 中的位置排序

```
postgres=# explain select relpages,reltuples,relname from pg_class order by 2 limit 1;
                            QUERY PLAN
-------------------------------------------------------------------------
 Limit  (cost=47.03..47.03 rows=1 width=72)
   ->  Sort  (cost=47.03..48.54 rows=602 width=72)
         Sort Key: reltuples
         ->  Seq Scan on pg_class  (cost=0.00..44.02 rows=602 width=72)
(4 rows)
```

3、按表达式排序

```
postgres=# explain select relpages,reltuples,relname from pg_class order by reltuples+relpages limit 1;
                            QUERY PLAN
-------------------------------------------------------------------------
 Limit  (cost=50.04..50.04 rows=1 width=80)
   ->  Sort  (cost=50.04..51.55 rows=602 width=80)
         Sort Key: ((reltuples + (relpages)::double precision))
```

```
->  Seq Scan on pg_class  (cost=0.00..47.03 rows=602 width=80)
```

10,timestamp + numeric

PostgreSQL 支持时间戳与 **interval** 类型进行加减。日期支持与整型做加减。
为了兼容 **Oracle**（时间戳与数字加减），我们可以复写操作符来实现时间戳与数字的加减。
复写操作符
1、自定义几个函数，用于时间戳与数字的加减。

```
postgres=# create or replace function timestamp_add_num(timestamp, float8)
returns timestamp as $$
  select $1 + ($2||' day')::interval;
$$ language sql strict immutable;
CREATE FUNCTION

postgres=# create or replace function timestamptz_add_num(timestamptz, float8)
returns timestamptz as $$
  select $1 + ($2||' day')::interval;
$$ language sql strict immutable;
CREATE FUNCTION

postgres=# create or replace function num_add_timestamp(float8, timestamp)
returns timestamp as $$
  select $2 + ($1||' day')::interval;
$$ language sql strict immutable;
CREATE FUNCTION

postgres=# create or replace function num_add_timestamptz(float8, timestamptz)
returns timestamptz as $$
  select $2 + ($1||' day')::interval;
$$ language sql strict immutable;
CREATE FUNCTION

postgres=# create or replace function timestamp_min_num(timestamp, float8)
returns timestamp as $$
  select $1 - ($2||' day')::interval;
$$ language sql strict immutable;
CREATE FUNCTION

postgres=# create or replace function timestamptz_min_num(timestamptz, float8)
returns timestamptz as $$
  select $1 - ($2||' day')::interval;
$$ language sql strict immutable;
```

CREATE FUNCTION
2、复写操作符

```
postgres=# create operator + (procedure = timestamp_add_num, leftarg=timestamp, rightarg=float8);
CREATE OPERATOR
postgres=# create operator + (procedure = timestamptz_add_num, leftarg=timestamptz, rightarg=float8);
CREATE OPERATOR

postgres=# create operator + (procedure = num_add_timestamp, leftarg=float8, rightarg=timestamp);
CREATE OPERATOR
postgres=# create operator + (procedure = num_add_timestamptz, leftarg=float8, rightarg=timestamptz);
CREATE OPERATOR

postgres=# create operator - (procedure = timestamp_min_num, leftarg=timestamp, rightarg=float8);
CREATE OPERATOR
postgres=# create operator - (procedure = timestamptz_min_num, leftarg=timestamptz, rightarg=float8);
CREATE OPERATOR
```

3、验证测试

```
postgres=# select now()+1;
              ?column?
-------------------------------
 2017-10-25 20:03:39.256659+08
(1 row)

postgres=# select now()+1.1;
              ?column?
-------------------------------
 2017-10-25 22:27:40.925673+08
(1 row)

postgres=# select now()-1.1;
              ?column?
-------------------------------
 2017-10-23 18:35:04.419078+08
(1 row)

postgres=# select 1.1+now();
```

```
              ?column?
-------------------------------
 2017-10-25 23:23:08.842953+08
(1 row)

postgres=# select 1.1+now()::timestamp;
              ?column?
----------------------------
 2017-10-25 23:23:13.318669
```

或者 **orafce** 包

11,系统列(关键字、保留字)的处理

当我们建表时，不能使用冲突的列名，否则会报错：

```
postgres=# create table a(ctid int);
错误:  42701: 字段名 "ctid" 与系统字段名冲突
LOCATION:  CheckAttributeNamesTypes, heap.c:439
```

当 **Oracle** 用户要迁移到 **PG**，遇到这样的问题怎么办呢？让用户改程序好像不太现实。

解决办法
创建影子表（将冲突字段重命名）

```
postgres=# create table tbl_shadow(n_ctid int, n_xmin int, n_max int, n_oid int);
CREATE TABLE
```

创建视图(作为业务程序中用于交互的表名)，可以采用冲突字段，解决了兼容性问题。

```
postgres=# create view tbl1 as select n_ctid as ctid, n_xmin as xmin, n_max as xmax, n_oid as oid from tbl_shadow ;
CREATE VIEW
```

对视图进行增删改查，会自动转换为对表的增删改查。

```
postgres=# insert into tbl1 (ctid,xmin,xmax,oid) values (1,1,1,1);
INSERT 0 1

postgres=# select ctid from tbl1;;
 ctid
------
    1
(1 row)

postgres=# update tbl1 set xmax=2;
```

```
UPDATE 1

postgres=# select * from tbl1;
 ctid | xmin | xmax | oid
------+------+------+-----
    1 |    1 |    2 |   1
(1 row)
```

12,rowid

```
PostgreSQL rowid - sequence 唯一标识
create table tbl (rowid serial8 not null, c1 int, c2 int);
create unique index idx_tbl_1 on tbl(rowid);

postgres=# insert into tbl (c1,c2) values (1,2);
INSERT 0 1
postgres=# insert into tbl (c1,c2) values (1,2);
INSERT 0 1
postgres=# insert into tbl (c1,c2) values (1,2);
INSERT 0 1
postgres=# select * from tbl;
 rowid | c1 | c2
-------+----+----
     1 |  1 |  2
     2 |  1 |  2
     3 |  1 |  2
(3 rows)
PostgreSQL rowid - IDENTITY 唯一标识(适用于 PostgreSQL 10+)
create table tbl (rowid int8 GENERATED ALWAYS AS IDENTITY not null, c1 int, c2
int);
create unique index idx_tbl_1 on tbl(rowid);

postgres=# insert into tbl (c1,c2) values (1,2);
INSERT 0 1
postgres=# insert into tbl (c1,c2) values (1,2);
INSERT 0 1
postgres=# insert into tbl (c1,c2) values (1,2);
INSERT 0 1
postgres=# select * from tbl;
 rowid | c1 | c2
-------+----+----
     1 |  1 |  2
     2 |  1 |  2
```

```
      3 |   1 |   2
(3 rows)
```

PostgreSQL rowid - oids 唯一标识(oid 只有 32 位，记录数超过 40 亿的单表，不适用)

```
postgres=# \dT oid
                          List of data types
   Schema     | Name |                    Description
--------------+------+-----------------------------------------------
 pg_catalog | oid    | object identifier(oid), maximum 4 billion
(1 row)
```

例子

```
postgres=# create table tbl (c1 int, c2 int) with oids;
CREATE TABLE
postgres=# create unique index idx_tbl_oid on tbl(oid);
CREATE INDEX

postgres=# insert into tbl (c1,c2) values (1,2);
INSERT 16412 1
postgres=# insert into tbl (c1,c2) values (1,2);
INSERT 16413 1
postgres=# insert into tbl (c1,c2) values (1,2);
INSERT 16414 1
postgres=# select oid,* from tbl;
  oid   | c1 | c2
-------+----+----
 16412 |   1 |   2
 16413 |   1 |   2
 16414 |   1 |   2
(3 rows)
```


13,round interval

Oracle 可以将 interval 当成一个秒为单位的数值，并允许对其值进行 round。

PostgreSQL 的 round 没有写这个，不过我们可以自定义一个兼容函数。

```
create or replace function round(interval, int) returns float8 as $$
  select round(EXTRACT(EPOCH FROM $1)::numeric, $2)::float8;
$$ language sql strict immutable;
postgres=# select round(interval '1h 10min 1.1second',2);
 round
--------
 4201.1
```

(1 row)

14,Nvl
SELECT COALESCE(description, short_description, '(none)')

```
postgres=# select coalesce(null,23,45);
 coalesce
----------
       23
(1 row)

postgres=# select coalesce(null,null,45);
 coalesce
----------
       45
```

14,WM_SYS.WM_CONCAT
Oracle 行转列函数 WMSYS.WM_CONCAT 的使用实例 demo

```
select *   from itlife365_course a   where name= '张三';
```

name	课程	score
张三	数学	99
张三	语文	89
张三	英语	93

上面的场景可用 WMSYS.WM_CONCAT(a.name)把二行中的[课程]字段的值用","连接起来

如:

```
select name, to_char(WMSYS.WM_CONCAT(a.课程))
  from itlife365_course a
 where name= '张三'
 group by a.name;
```

注意：因为用 WMSYS.WM_CONCAT 转出的类型是 clob 的，所以我这用了 to_char 转了一下。

使用 wmsys.wm_concat 多列合成一列遇到问题

ORA-22813: 操作数值超出系统的限制

官方文档解释是总长度超过 30k

请使用其他方法替代。

PostgreSQL 不存在问题，最长可以达到 1GB。

PostgreSQL 使用 string_agg 聚合函数即可达到同样的目的：

```
select name, string_agg(a.课程, ',')
  from itlife365_course a
 where name= '张三'
 group by a.name;
```

如果用户不想改代码，可以尝试自行创建一个名为 WM_CONCAT 的聚合函数，例子如下：

```
create schema WMSYS;
create or replace function WMSYS.sf_concat(text,text) returns text as $$
  select case when $1 is not null then $1||','||$2 else $2 end;
$$ language sql called on null input;
create AGGREGATE WMSYS.wm_concat (text)
(sfunc=WMSYS.sf_concat,stype=text);
```

14,UUID

```
postgres=# create extension "uuid-ossp";
CREATE EXTENSION
postgres=# create or replace function sys_guid() returns uuid as $$
select uuid_generate_v4();
$$ language sql strict;
CREATE FUNCTION
postgres=# select sys_guid();
              sys_guid
--------------------------------------
 92bbbf05-a23c-41b3-95d4-8732c93d95dd
(1 row)

postgres=# select sys_guid();
              sys_guid
--------------------------------------
 37e34cfb-46aa-44ed-9403-9e23b6c2bfc0
(1 row)
```

15,pipelined

```
create or replace function split
(
p_list varchar2,
```

```
p_del varchar2 := ','
) return split_tbl pipelined
is
l_idx pls_integer;
l_list varchar2(32767) := p_list;
l_value varchar2(32767);
begin
loop
l_idx := instr(l_list,p_del);
if l_idx > 0 then
pipe row(trim(substr(l_list,1,l_idx-1)));
l_list := substr(l_list,l_idx+length(p_del));

else
pipe row(trim(l_list));
exit;
end if;
end loop;
return;
end split;
```

PostgreSQL：

对于以上例子的需求，可以使用现成的 PostgreSQL 函数来解决：

```
postgres=# select regexp_split_to_table('a-b-c-d','-');
 regexp_split_to_table
----------------------
 a
 b
 c
 d
(4 rows)
```

如果用户只是有返回多行的需求，则可以使用 returns setof 来解决。

例如：

```
postgres=# create or replace function split (text,text) returns setof text as $$
postgres$# select regexp_split_to_table($1,$2);
postgres$# $$ language sql strict;
CREATE FUNCTION

postgres=# select split('a-b-c-d','-');
 split
-------
```

```
 a
 b
 c
 d
(4 rows)

postgres=# create or replace function rsf1(id int) returns setof int as $$
postgres$# declare
postgres$# begin
postgres$#     for i in 0..abs(id) loop
postgres$#         return next i;
postgres$#     end loop;
postgres$# end;
postgres$# $$ language plpgsql strict;
CREATE FUNCTION
postgres=# select rsf1(10);
 rsf1
------
      0
      1
      2
      3
      4
      5
      6
      7
      8
      9
     10
(11 rows)
```


16,unique

Oracle: distinct unique
select unique col1, col2 from table1

Postgres
select distinct col1, col2 from table1

oracle plsql 迁移 plpgsql

PL/pgSQL 与 **PL/SQL** 在许多方面都非常类似。它是一种块结构的、命令式的语言并且所有变量必须先被声明。
赋值、循环、条件则很类似
1,在 **PostgreSQL** 中，函数体必须写成字符串文本。因此你需要使用美元符引用或者转义函数体中的单引号
2,数据类型名称常常需要翻译。例如，在 **Oracle** 中字符串值通常被声明为类型 **varchar2**，这并非 **SQL** 标准类型。在 **PostgreSQL** 中则要使用类型 **varchar** 或者 **text** 来替代。类似地，要把类型 **number** 替换成 **numeric**，或者在适当的时候使用某种其他数字数据类型。
3,应该用模式把函数组织成不同的分组，而不是用包
4,因为没有包，所以也没有包级别的变量。可以在临时表里保存会话级别的状态。
5,带有 **REVERSE** 的整数 **FOR** 循环的工作方式不同：**PL/SQL** 中是从第二个数向第一个数倒数，而 **PL/pgSQL** 是从第一个数向第二个数倒数，因此在移植时需要交换循环边界
6,查询上的 **FOR** 循环（不是游标）的工作方式同样不同：目标变量必须已经被声明，而 **PL/SQL** 总是会隐式地声明它们。但是这样做的优点是在退出循环后，变量值仍然可以访问。
7,在使用游标变量方面，存在一些记法差异

函数的教程

- **variadic parameters, default parameters**
- **RETURN QUERY, CONTINUE, FOREACH SLICE,**

- GET STACKED DIAGNOSTICS, ASSERT

- **USAGE clause in EXECUTE**
- **rich RAISE statement**
- **plpgsql_check, Orafce**

- functions: greatest, least, format, string_agg,left, right,
- \sf, \ef, \gset

PLpgSQL 案例

```
CREATE OR REPLACE FUNCTION new_customer(name text, surname text)
RETURNS int AS $$
DECLARE
  uid int;
```

```
BEGIN
IF NOT EXISTS(SELECT * FROM customers c WHERE c.name = new_customer.name
                AND c.surname = new_customer.surname)
THEN
    INSERT INTO customers(name, surname)
    VALUES(new_customer.name, new_customer.surname
    RETURNING id INTO uid;
    RETURN uid;
ELSE
    RAISE EXCEPTION "Customer exists already";
END IF;
END;
$$ LANGUAGE plpgsql STRICT;
```

plpgsql 存储过程的好处

减少网络传输
应用程序解耦，数据集编写，多个程序调用
更安全

PLpgSQL 转换 PL/SQL

1,循环是相同的
FOR i IN 1 .. 10 LOOP

2,pg 没有 **immediate**
FOR r IN SELECT * FROM ...
EXECUTE IMMEDIATE '....'

3,输出
dbms_output.put_line(...)
RAISE NOTICE '...'

4,goto 语法
PLpgSQL 没有 **GOTO**

5,支持语言
Python, Perl, Lua, Java,C

6,pg/pgsql 没有的特性

没有 oop 特性
不同于 oracle 的聚合和异常处理
只有局部变量，没有全局变量
pg 没有包,使用 pg 的 schema 代替
没有 collections,使用数组类型
dbms 包，pg 可以使用(Orafce 兼容包)
没有自治事务,通过 dblink 模拟

oracle 函数转 pg 案例

oracle 的函数

```
CREATE OR REPLACE FUNCTION cs_fmt_browser_version(v_name varchar2,
                                                  v_version varchar2)
RETURN varchar2 IS
BEGIN
    IF v_version IS NULL THEN
        RETURN v_name;
    END IF;
    RETURN v_name || '/' || v_version;
END;
/
show errors;
```

postgres 函数

```
CREATE OR REPLACE FUNCTION cs_fmt_browser_version(v_name varchar,
                                                  v_version varchar)
RETURNS varchar AS $$
BEGIN
    IF v_version IS NULL THEN
        RETURN v_name;
    END IF;
    RETURN v_name || '/' || v_version;
END;
$$ LANGUAGE plpgsql;
```

Oracle 版本：

```
CREATE OR REPLACE PROCEDURE cs_update_referrer_type_proc IS
    CURSOR referrer_keys IS
        SELECT * FROM cs_referrer_keys
        ORDER BY try_order;
    func_cmd VARCHAR(4000);
BEGIN
```

```
    func_cmd := 'CREATE OR REPLACE FUNCTION cs_find_referrer_type(v_host
IN VARCHAR2,
                    v_domain IN VARCHAR2, v_url IN VARCHAR2) RETURN
VARCHAR2 IS BEGIN';

    FOR referrer_key IN referrer_keys LOOP
        func_cmd := func_cmd ||
          ' IF v_' || referrer_key.kind
          || ' LIKE ''' || referrer_key.key_string
          || ''' THEN RETURN ''' || referrer_key.referrer_type
          || '''; END IF;';
    END LOOP;

    func_cmd := func_cmd || ' RETURN NULL; END;';

    EXECUTE IMMEDIATE func_cmd;
END;
/
show errors;
```

PostgreSQL 的版本:

```
CREATE OR REPLACE FUNCTION cs_update_referrer_type_proc() RETURNS
void AS $func$
DECLARE
    referrer_keys CURSOR IS
        SELECT * FROM cs_referrer_keys
        ORDER BY try_order;
    func_body text;
    func_cmd text;
BEGIN
    func_body := 'BEGIN';

    FOR referrer_key IN referrer_keys LOOP
        func_body := func_body ||
          ' IF v_' || referrer_key.kind
          || ' LIKE ' || quote_literal(referrer_key.key_string)
          || ' THEN RETURN ' || quote_literal(referrer_key.referrer_type)
          || '; END IF;' ;
    END LOOP;

    func_body := func_body || ' RETURN NULL; END;';

    func_cmd :=
```

```
        'CREATE OR REPLACE FUNCTION cs_find_referrer_type(v_host varchar,
                                                          v_domain varchar,
                                                          v_url varchar)
            RETURNS varchar AS '
        || quote_literal(func_body)
        || ' LANGUAGE plpgsql;' ;

    EXECUTE func_cmd;
END;
$func$ LANGUAGE plpgsql;
```

Oracle 版本：

```
CREATE OR REPLACE PROCEDURE cs_parse_url(
    v_url IN VARCHAR2,
    v_host OUT VARCHAR2,    -- 这将被传回去
    v_path OUT VARCHAR2,    -- 这个也是
    v_query OUT VARCHAR2) -- 还有这个
IS
    a_pos1 INTEGER;
    a_pos2 INTEGER;
BEGIN
    v_host := NULL;
    v_path := NULL;
    v_query := NULL;
    a_pos1 := instr(v_url, '//');

    IF a_pos1 = 0 THEN
        RETURN;
    END IF;
    a_pos2 := instr(v_url, '/', a_pos1 + 2);
    IF a_pos2 = 0 THEN
        v_host := substr(v_url, a_pos1 + 2);
        v_path := '/';
        RETURN;
    END IF;

    v_host := substr(v_url, a_pos1 + 2, a_pos2 - a_pos1 - 2);
    a_pos1 := instr(v_url, '?', a_pos2 + 1);

    IF a_pos1 = 0 THEN
        v_path := substr(v_url, a_pos2);
        RETURN;
    END IF;
```

```
    v_path := substr(v_url, a_pos2, a_pos1 - a_pos2);
    v_query := substr(v_url, a_pos1 + 1);
END;
/
show errors;
```

PL/pgSQL 的可能翻译:

```
CREATE OR REPLACE FUNCTION cs_parse_url(
    v_url IN VARCHAR,
    v_host OUT VARCHAR,   -- 这将被传递回去
    v_path OUT VARCHAR,   -- 这个也是
    v_query OUT VARCHAR) -- 以及这个
AS $$
DECLARE
    a_pos1 INTEGER;
    a_pos2 INTEGER;
BEGIN
    v_host := NULL;
    v_path := NULL;
    v_query := NULL;
    a_pos1 := instr(v_url, '//');

    IF a_pos1 = 0 THEN
        RETURN;
    END IF;
    a_pos2 := instr(v_url, '/', a_pos1 + 2);
    IF a_pos2 = 0 THEN
        v_host := substr(v_url, a_pos1 + 2);
        v_path := '/';
        RETURN;
    END IF;

    v_host := substr(v_url, a_pos1 + 2, a_pos2 - a_pos1 - 2);
    a_pos1 := instr(v_url, '?', a_pos2 + 1);

    IF a_pos1 = 0 THEN
        v_path := substr(v_url, a_pos2);
        RETURN;
    END IF;

    v_path := substr(v_url, a_pos2, a_pos1 - a_pos2);
    v_query := substr(v_url, a_pos1 + 1);
```

```
END;
$$ LANGUAGE plpgsql;
```

oracle 包函数 转 plpgsql 的案例

```
Oracle PACKAGE
CREATE PACKAGE bonus AS
PROCEDURE calc_bonus(uid int);
END
    CREATE PACKAGE BODY bonus AS
    PROCEDURE calc_bonus(uid int) IS
BEGIN
    DBMS_OUTPUT.PUT_LINE('started');
END;
END bonus;

PostgreSQL SCHEMA
DROP SCHEMA IF EXISTS bonus CASCADE;
CREATE SCHEMA bonus;
SET search_path TO bonus;
CREATE FUNCTION calc_bonus(uid int)
RETURNS void AS $$
BEGIN
    RAISE NOTICE 'started';
END;
$$ LANGUAGE plpgsql SET search_path = bonus;
```

plpgsql 储存过程检验

```
https://github.com/okbob/plpgsql_check
passive - LOAD 'plpgsql_check' (disabled by default)
active - plpgsql_check_function()
```

pldebugger 储存过程调试
git://git.postgresql.org/git/pldebugger.git

触发器

https://www.postgresql.org/docs/10/static/triggers.html

事件触发器

https://www.postgresql.org/docs/10/static/event-triggers.html

规则

https://www.postgresql.org/docs/10/static/rules.html

参考

https://postgres.cz/wiki/Napi%C5%A1te_si_debugger_PL/pgSQL_aneb_pokro%C4%8Dil%C3%A9_techniky_programov%C3%A1n%C3%AD_v_PostgreSQL

plpgsql 官方教程

https://www.postgresql.org/docs/10/static/plpgsql.html

Oracle 转 pg 的成本

后期运维差异

postgres 相比 **oracle** 运维更简单。

开发成本

1,因为 **SQL** 的语法的变动，程序可能更改的比较大
2,大对象中如果是文本可以直接使用 **text** 或 **character vaying,**
如果是 **json**，**postgres** 中可以使用 **jsonb** 类型，但代码要改动的可能就比较多了

- **PostgreSQL**
  - **SQL** 兼容性好，功能强大，扩展能力强，服务端编程能力强。
  - 数据库端可以解决非常多的程序需求，不需要 **move data**，开发成本低。

运营成本以及收益

Oracle 安核心收费，贵啊！！！
Postrges 是开源数据库中稳定性最好，功能最强，在去 **IOE** 过程中，最有可能代替 **oracle** 的数据库产品。

学习成本

Oracle 和 **postrges** 的架构类似，从 **oracle** 转到 **postgres** 还是相对比较容易的。

更多文章入口

Se7en.shi's 微信

打赏码