阿里云
云原生数据仓库 AnalyticDB PostgreSQL 版

应用迁移

文档版本：20200624

法律声明

阿里云提醒您在阅读或使用本文档之前仔细阅读、充分理解本法律声明各条款的内容。如果您阅读或使用本文档，您的阅读或使用行为将被视为对本声明全部内容的认可。

1. 您应当通过阿里云网站或阿里云提供的其他授权通道下载、获取本文档，且仅能用于自身的合法合规的业务活动。本文档的内容视为阿里云的保密信息，您应当严格遵守保密义务；未经阿里云事先书面同意，您不得向任何第三方披露本手册内容或提供给任何第三方使用。
2. 未经阿里云事先书面许可，任何单位、公司或个人不得擅自摘抄、翻译、复制本文档内容的部分或全部，不得以任何方式或途径进行传播和宣传。
3. 由于产品版本升级、调整或其他原因，本文档内容有可能变更。阿里云保留在没有任何通知或者提示下对本文档的内容进行修改的权利，并在阿里云授权通道中不时发布更新后的用户文档。您应当实时关注用户文档的版本变更并通过阿里云授权渠道下载、获取最新版的用户文档。
4. 本文档仅作为用户使用阿里云产品及服务的参考性指引，阿里云以产品及服务的“现状”、“有缺陷”和“当前功能”的状态提供本文档。阿里云在现有技术的基础上尽最大努力提供相应的介绍及操作指引，但阿里云在此明确声明对本文档内容的准确性、完整性、适用性、可靠性等不作任何明示或暗示的保证。任何单位、公司或个人因为下载、使用或信赖本文档而发生任何差错或经济损失的，阿里云不承担任何法律责任。在任何情况下，阿里云均不对任何间接性、后果性、惩戒性、偶然性、特殊性或刑罚性的损害，包括用户使用或信赖本文档而遭受的利润损失，承担责任（即使阿里云已被告知该等损失的可能性）。
5. 阿里云文档中所有内容，包括但不限于图片、架构设计、页面布局、文字描述，均由阿里云和/或其关联公司依法拥有其知识产权，包括但不限于商标权、专利权、著作权、商业秘密等。非经阿里云和/或其关联公司书面同意，任何人不得擅自使用、修改、复制、公开传播、改变、散布、发行或公开发表阿里云网站、产品程序或内容。此外，未经阿里云事先书面同意，任何人不得为了任何营销、广告、促销或其他目的使用、公布或复制阿里云的名称（包括但不限于单独为或以组合形式包含“阿里云”、“Aliyun”、“万网”等阿里云和/或其关联公司品牌，上述品牌的附属标志及图案或任何类似公司名称、商号、商标、产品或服务名称、域名、图案标示、标志、标识或通过特定描述使第三方能够识别阿里云和/或其关联公司）。
6. 如若发现本文档存在任何错误，请与阿里云取得直接联系。

通用约定

格式	说明	样例
	该类警示信息将导致系统重大变更甚至故障，或者导致人身伤害等结果。	 **禁止：** 重置操作将丢失用户配置数据。
	该类警示信息可能会导致系统重大变更甚至故障，或者导致人身伤害等结果。	 **警告：** 重启操作将导致业务中断，恢复业务时间约十分钟。
	用于警示信息、补充说明等，是用户必须了解的内容。	 **注意：** 权重设置为0，该服务器不会再接受新请求。
	用于补充说明、最佳实践、窍门等，不是用户必须了解的内容。	 **说明：** 您也可以通过按Ctrl + A选中全部文件。
>	多级菜单递进。	单击**设置** > **网络** > **设置网络类型**。
粗体	表示按键、菜单、页面名称等UI元素。	在**结果确认**页面，单击**确定**。
Courier字体	命令。	执行cd /d C:/window命令，进入Windows系统文件夹。
斜体	表示参数、变量。	bae log list --instanceid Instance_ID
[]或者[a|b]	表示可选项，至多选择一个。	ipconfig [-all|-t]
{}或者{a|b}	表示必选项，至多选择一个。	switch {active|stand}

目录

1 Oracle应用迁移至AnalyticDB for PostgreSQL

AnalyticDB for PostgreSQL对Oracle 语法有着较好的兼容，本文介绍如何将Oracle应用迁移到AnalyticDB for PostgreSQL。

基于ora2pg完成初步转换工作

可以使用开源工具ora2pg进行最初的Oracle应用转换。您可以使用ora2pg将Oracle的表DDL，view，package等语法转换成PostgreSQL兼容的语法。具体操作方法请参见ora2pg的用户文档。

说明：

由于脚本转换后的PG语法版本比AnalyticDB for PostgreSQL使用的PG内核版本高，而且ora2pg依赖规则进行转换，难免会存在偏差，因此您还需要手动对转换后的SQL脚本做纠正。

使用Orafunc插件

AnalyticDB for PostgreSQL中提供了Orafunc插件，该插件提供了一些兼容Oracle的函数。对于这些函数，您无需任何修改转换即可在AnalyticDB for PostgreSQL中使用。

在使用Orafunc插件前，只需执行`create extension orafunc;`命令即可。

```
postgres=> create extension orafunc;
CREATE EXTENSION
```

Orafunc插件提供的兼容函数如下表所示。

表 1-1: Orafunc插件兼容函数表

函数	描述	示例

nvl(anyelement, anyelement)	• 若第一个参数为null，则会返回第二个参数。
	• 若第一个参数不为null，则返回第一个参数。
	说明:
	两个参数必须是相同类型。
		postgres=# select nvl(null,1);
		nvl
		----- 	
		1
		(1 row)
		
		postgres=# select nvl(0,1);
		nvl
		----- 	
		0
		(1 row)
		
		postgres=# select nvl(0,null);
		nvl
		----- 	
		0
		(1 row)

add_months(day date, value int) RETURNS date	在第一个月份参数上加上第二个月份参数，返回类型为date。	postgres=# select add_months(current_date, 2); add_months ------------ 2019-08-31 (1 row)
last_day(value date)	返回某年某个月份的最后一天，返回类型为date。	postgres=# select last_day('2018-06-01'); last_day ------------ 2018-06-30 (1 row)
next_day(value date, weekday text)	• 参数一：开始的日期。 • 参数二：包含星期几的英文字符串，如Friday。 返回开始日期后的第二个星期几的日期，如第二个Friday。	postgres=# select next_day(current_date, 'FRIDAY'); next_day ------------ 2019-07-05 (1 row)

next_day(value date, weekday integer)	• 参数一：开始的日期。
	• 参数二：星期几的数字，取值为1到7，1为星期日，2为星期一，以此类推。
	返回开始日期加天数之后的日期。
		postgres=# select next_day('2019-06-22', 1);
		next_day
		------------ 	
		2019-06-23
		(1 row)
		
		postgres=# select next_day('2019-06-22', 2);
		next_day
		------------ 	
		2019-06-24
		(1 row)

months_between(date1 date, date2 date)	返回date1和date2之间的月数。 • 如果date1晚于date2，结果	postgres=# select months_between('2019-01-01', '2018-11-01'); months_between

对于以下的四个Oracle函数，在AnalyticDB for PostgreSQL中，无需安装Orafunc插件就可以提供兼容支持。

函数	描述	示例
sinh(float)	双曲正弦值。	postgres=# select sinh(0.1); sinh ------------------ 0.100166750019844 (1 row)
tanh(float)	双曲正切值。	postgres=# select tanh(3); tanh ----------------- 0.99505475368673 (1 row)
cosh(float)	双曲余弦值。	postgres=# select cosh(0.2); cosh ----------------- 1.02006675561908 (1 row)

decode(expression, value , return [,value, return]... [, default])	在表达式中寻找一个搜索值，搜索到则返回指定的值。如果没有搜索到，返回默认值。	create table t1(id int, name varchar(20));
		postgres=# insert into t1 values(1,' alibaba');
		postgres=# insert into t1 values(2,' adb4pg');
		postgres=# select decode(id, 1, ' alibaba', 2, 'adb4pg', 'not found') from t1;
		case
		--------- 	
		alibaba
		adb4pg
		(2 rows)
		
		postgres=# select decode(id, 3, ' alibaba', 4, 'adb4pg', 'not found') from t1;
		case
		----------- 	
		not found
		not found
		(2 rows)

数据类型转换对照表

Oracle	AnalyticDB for PostgreSQL
VARCHAR2	varchar or text
DATE	timestamp

Oracle	AnalyticDB for PostgreSQL
LONG	text
LONG RAW	bytea
CLOB	text
NCLOB	text
BLOB	bytea
RAW	bytea
ROWID	oid
FLOAT	double precision
DEC	decimal
DECIMAL	decimal
DOUBLE PRECISION	double precision
INT	int
INTERGE	integer
REAL	real
SMALLINT	smallint
NUMBER	numeric
BINARY_FLOAT	double precision
BINARY_DOUBLE	double precision
TIMESTAMP	timestamp
XMLTYPE	xml
BINARY_INTEGER	integer
PLS_INTEGER	integer
TIMESTAMP WITH TIME ZONE	timestamp with time zone
TIMESTAMP WITH LOCAL TIME ZONE	timestamp with time zone

系统函数转换对照表

Oracle	AnalyticDB for PostgreSQL
sysdate	current timestamp
trunc	trunc/ date trunc
dbms_output.put_line	raise 语句

Oracle	AnalyticDB for PostgreSQL
decode	转成case when/直接使用decode
NVL	coalesce

PL/SQL迁移指导

PL/SQL（Procedural Language/SQL）是一种过程化的SQL语言，是Oracle对SQL语句的拓展。PL/SQL使得SQL的使用可以具有一般编程语言的特点，可以用来实现复杂的业务逻辑。PL/SQL对应了AnalyticDB for PostgreSQL中的PL/PGSQL。

Package

PL/PGSQL不支持Package，需要把Package转换成schema，同时Package里面的所有procedure和function也需要转换成AnalyticDB for PostgreSQL的function。

例如：

```
create or replace package pkg is
...
end;
```

转换成：

```
create schema pkg;
```

- Package定义的变量

  procedure/function的局部变量保持不变，全局变量在AnalyticDB for PostgreSQL中可以使用临时表进行保存。

- Package初始化块

  请删除，若无法删除请使用function封装，在需要的时候主动调用该function。

- Package内定义的procedure/function

  Package内定义的procedure和function需要转成AnalyticDB for PostgreSQL的function，并把function定义到package对应的schema内。

  例如，有一个Package名为pkg中有如下函数：

  ```
  FUNCTION test_func (args int) RETURN int is
  var number := 10;
  BEGIN
  ... ...
  ```

```
END;
```

转换成如下AnalyticDB for PostgreSQL的function：

```
CREATE OR REPLACE FUNCTION pkg. test_func(args int) RETURNS int AS
$$

 ... ...

$$
 LANGUAGE plpgsql;
```

Procedure/function

Oracle中的procedure和function，不论属于package的还是属于全局，都需要转换成AnalyticDB for PostgreSQL的function。

例如：

```
CREATE OR REPLACE FUNCTION test_func (v_name varchar2, v_version varchar2)
RETURN varchar2 IS
   ret varchar(32);
BEGIN
   IF v_version IS NULL THEN
      ret := v_name;
ELSE
   ret := v_name || '/' || v_version;
   END IF;
   RETURN ret;
END;
```

转化成：

```
CREATE OR REPLACE FUNCTION test_func (v_name varchar, v_version varchar)
RETURNS varchar AS
$$

DECLARE
   ret varchar(32);
BEGIN
   IF v_version IS NULL THEN
      ret := v_name;
ELSE
   ret := v_name || '/' || v_version;
   END IF;
   RETURN ret;
END;

$$
 LANGUAGE plpgsql;
```

Procedure/function转换的注意事项有以下几点：

- RETURN关键字转成RETURNS。
- 函数体使用$\$ ... $\$进行封装。
- 函数语言声明。

- Subprocedure需要转换成AnalyticDB for PostgreSQL的function。

PL statement

- **For语句**

  PL/SQL和PL/PGSQL在带有REVERSE的整数FOR循环中的工作方式不同：

  - PL/SQL中是从第二个数向第一个数倒数。
  - 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，如下所示：

```
FOR i IN REVERSE 1..3 LOOP
   DBMS_OUTPUT.PUT_LINE (TO_CHAR(i));
END LOOP;
```

  转换成：

```
FOR i IN REVERSE 3..1 LOOP
   RAISE '%' ,i;
END LOOP;
```

- **PRAGMA语句**

  AnalyticDB for PostgreSQL中没有PRAGMA语句，删除该类语句。

- **事务处理**

  AnalyticDB for PostgreSQL的function 内部无法使用事务控制语句，如begin，commit，rollback等。

  修改方法如下：

  - 删除函数体内的事务控制语句，把事务控制放在函数体外。
  - 把函数按照commit/rollback 拆分成多个。

- **EXECUTE语句**

  AnalyticDB for PostgreSQL支持类似Oracle的动态SQL语句，不同之处如下：

  - 不支持using语法，可通过把参数拼接到SQL串中来解决。
  - 数据库标识符使用quote_ident包裹，数值使用quote_literal包裹。

  示例：

```
EXECUTE 'UPDATE employees_temp SET commission_pct = :x' USING a_null;
```

  转换成：

```
EXECUTE 'UPDATE employees_temp SET commission_pct = ' || quote_literal(a_null);
```

- **Pipe row**

  Pipe row函数，可使用AnalyticDB for PostgreSQL的table function函数来替换。

  示例：

```
TYPE pair IS RECORD(a int, b int);
TYPE numset_t IS TABLE OF pair;

FUNCTION f1(x int) RETURN numset_t PIPELINED IS
DECLARE
   v_p pair;
BEGIN
   FOR i IN 1..x LOOP
     v_p.a := i;
     v_p.b := i+10;
     PIPE ROW(v_p);
   END LOOP;
   RETURN;
END;

select * from f1(10);
```

  转换成：

```
create type pair as (a int, b int);

create or replace function f1(x int) returns setof pair as
$$

declare
rec pair;
begin
   for i in 1..x loop
      rec := row(i, i+10);
      return next rec;
   end loop;
   return ;
end

$$
 language 'plpgsql';
```

```
select * from f1(10);
```

说明：

- 自定义类型pair转换成AnalyticDB for PostgreSQL的复合类型pair。
- Table of类型不需要定义，使用AnalyticDB for PostgreSQL的setof替换。
- Pipe row 语句转换成下面两个语句：

```
 rec := row(i);
 return next rec;
```

oracle function还可以转换成如下语句：

```
create or replace function f1(x int) returns setof record as
$$

declare
rec record;
begin
   for i in 1..x loop
      rec := row(i, i+10);
      return next rec;
   end loop;
   return ;
end

$$
language 'plpgsql';
```

与第一种改法的不同的是，这种改法不需要提前定义数据类型numset_t，所以在查询的时候需要指定返回的类型，例如：`select * from f1(10) as (a int, b int);`。

- **异常处理**
  - 使用raise抛出异常。
  - Catch异常后，不能rollback事务，只能在udf外做rollback。
  - AnalyticDB for PostgreSQL支持的error，请参考：https://www.postgresql.org/docs/8.3/errcodes-appendix.html
- **function中同时有Return和OUT参数**

在AnalyticDB for PostgreSQL中，fucntion无法同时存在return和out参数，需要把返回的参数改写成out类型参数。

示例：

```
CREATE OR REPLACE FUNCTION test_func(id int, name varchar(10), out_id out int)
returns varchar(10)
AS $body$
BEGIN
    out_id := id + 1;
    return name;
end
```

```
$body$
LANGUAGE PLPGSQL;
```

转换成：

```
CREATE OR REPLACE FUNCTION test_func(id int, name varchar(10), out_id out int,
out_name out varchar(10))
AS $body$
BEGIN
    out_id := id + 1;
    out_name := name;
end
$body$
LANGUAGE PLPGSQL;
```

`select * from test_func(1,'1') into rec;`从rec中取对应字段的返回值即可。

- **字符串连接中变量含有单引号**

  在如下示例中，变量param2是一个字符串类型。假设param2的值为`adb'-'pg`。下面的sql_str直接放到AnalyticDB for PostgreSQL中使用会将`-`识别成一个operator而报错。需要使用quote_literal函数来进行转换。

  示例：

  ```
  sql_str := 'select * from test1 where col1 = ' || param1 || ' and col2 = '''|| param2 || '''and
  col3 = 3';
  ```

  转换成：

  ```
  sql_str := 'select * from test1 where col1 = ' || param1 || ' and col2 = '|| quote_literal(
  param2) || 'and col3 = 3';
  ```

- **获取两个timestamp相减后的天数**

  示例：

  ```
  SELECT to_date('2019-06-30 16:16:16') - to_date('2019-06-29 15:15:15') + 1 INTO
  v_days from dual;
  ```

  转换成：

  ```
  SELECT extract('days' from '2019-06-30 16:16:16'::timestamp - '2019-06-29 15:15:15
  '::timestamp + '1 days'::interval)::int INTO v_days;
  ```

PL数据类型

- **Record**

  使用AnalyticDB for PostgreSQL的复合数据类型替换。

  示例：

  ```
  TYPE rec IS RECORD (a int, b int);
  ```

  转换成：

  ```
  CREATE TYPE rec AS (a int, b int);
  ```

- **Nest table**

  - Nest table作为PL变量，可以使用ADB for PG的array类型替换。

    示例：

    ```
    DECLARE
      TYPE Roster IS TABLE OF VARCHAR2(15);
      names Roster :=
      Roster('D Caruso', 'J Hamil', 'D Piro', 'R Singh');
    BEGIN
      FOR i IN names.FIRST .. names.LAST
      LOOP
         IF names(i) = 'J Hamil' THEN
           DBMS_OUTPUT.PUT_LINE(names(i));
         END IF;
      END LOOP;
    END;
    ```

    转换成：

    ```
    create or replace function f1() returns void as
    $$

    declare
       names varchar(15)[] := '{"D Caruso", "J Hamil", "D Piro", "R Singh"}';
       len int := array_length(names, 1);
    begin
       for i in 1..len loop
          if names[i] = 'J Hamil' then
             raise notice '%', names[i];
          end if;
       end loop;
       return ;
    end

    $$
     language 'plpgsql';

    select f();
    ```

  - 作为function返回值，可以使用table function替换。

- **Associative Array**

  无替换类型。

- **Variable-Size Arrays**

  与Nest table类似，使用array类型替换。

- **Global variables**

  目前AnalyticDB for PostgreSQL不支持global variables，可以把package中的所有global variables存入一张临时表（temporary table）中，然后修改定义，获取global variables的函数。

  示例：

```
create temporary table global_variables (
      id int,
      g_count int,
      g_set_id varchar(50),
      g_err_code varchar(100)
);

insert into global_variables values(0, 1, null，null);

CREATE OR REPLACE FUNCTION get_variable() returns setof global_variables AS

$$

DECLARE
   rec global_variables%rowtype;
BEGIN
   execute 'select * from global_variables' into rec;
   return next rec;
END;

$$
 LANGUAGE plpgsql;

CREATE OR REPLACE FUNCTION set_variable(in param varchar(50), in value anyelement
) returns void AS

$$

BEGIN
   execute 'update global_variables set ' ‖ quote_ident(param) ‖ ' = ' ‖ quote_literal(
value);
END;

$$
 LANGUAGE plpgsql;
```

  临时表global_variables中，字段id为这个表的分布列，由于AnalyticDB for PostgreSQL中不允许对于分布列的修改，需要加一个tmp_rec record;字段。

  修改一个全局变量时，使用：`select * from set_variable('g_error_code', 'error'::varchar) into tmp_rec;`。

  获取一个全局变量时，使用：`select * from get_variable() into tmp_rec; error_code := tmp_rec.g_error_code;`。

SQL

- **Connect by**

  Oracle层次查询，AnalyticDB for PostgreSQL没有等价替换的SQL语句。可以使用循环按层次遍历这样的转换思路。

  示例：

```
create table employee(
      emp_id numeric(18),
      lead_id numeric(18),
      emp_name varchar(200),
      salary numeric(10,2),
      dept_no varchar(8)
);
insert into employee values('1',0,'king','1000000.00','001');
insert into employee values('2',1,'jack','50500.00','002');
insert into employee values('3',1,'arise','60000.00','003');
insert into employee values('4',2,'scott','30000.00','002');
insert into employee values('5',2,'tiger','25000.00','002');
insert into employee values('6',3,'wudde','23000.00','003');
insert into employee values('7',3,'joker','21000.00','003');
insert into employee values('3',7,'joker','21000.00','003');
```

```
select emp_id,lead_id,emp_name,prior emp_name as lead_name,salary
    from employee
    start with  lead_id=0
    connect by prior emp_id =  lead_id
```

  转换成：

```
create or replace function f1(tablename text, lead_id int, nocycle boolean) returns
setof employee as
$$

declare
   idx int := 0;
   res_tbl varchar(265) := 'result_table';
   prev_tbl varchar(265) := 'tmp_prev';
   curr_tbl varchar(256) := 'tmp_curr';

   current_result_sql varchar(4000);
   tbl_count int;

   rec record;
begin

   execute 'truncate ' || prev_tbl;
   execute 'truncate ' || curr_tbl;
   execute 'truncate ' || res_tbl;
   loop
      -- 查询当前层次结果，并插入到tmp_curr表
      current_result_sql := 'insert into ' || curr_tbl || ' select t1.* from ' || tablename || ' t1';

      if idx > 0 then
         current_result_sql := current_result_sql || ', ' || prev_tbl || ' t2 where t1.lead_id = t2
.emp_id';
      else
```

```
        current_result_sql := current_result_sql || ' where t1.lead_id = ' || lead_id;
    end if;
    execute current_result_sql;

    -- 如果有环，删除已经遍历过的数据
    if nocycle is false then
        execute 'delete from ' || curr_tbl || ' where (lead_id, emp_id) in (select lead_id,
emp_id from ' || res_tbl || ') ';
    end if;

    -- 如果没有数据，则退出
    execute 'select count(*) from ' || curr_tbl into tbl_count;
    exit when tbl_count = 0;

    -- 把tmp_curr数据保存到result表
    execute 'insert into ' || res_tbl || ' select * from ' || curr_tbl;
    execute 'truncate ' || prev_tbl;
    execute 'insert into ' || prev_tbl || ' select * from ' || curr_tbl;
    execute 'truncate ' || curr_tbl;
    idx := idx + 1;
  end loop;

  -- 返回结果
  current_result_sql := 'select * from ' || res_tbl;
  for rec in execute current_result_sql loop
    return next rec;
  end loop;
  return;
end

$$
```

```
language plpgsql;
```

- **Rownum**

  1. 限定查询结果集大小，可以使用limit替换。

     示例：

     ```
     select * from t where rownum < 10;
     ```

     转换成：

     ```
     select * from t limit 10;
     ```

  2. 使用row_number() over()生成rownum。

     示例：

     ```
     select rownum, * from t;
     ```

     转换成：

     ```
     select row_number() over() as rownum, * from t;
     ```

- **Dual表**

  1. 去掉dual。

     示例：

     ```
     select sysdate from dual;
     ```

     转换成：

     ```
     select current_timestamp;
     ```

  2. 创建一个叫dual的表。

- **Select中的udf**

  AnalyticDB for PostgreSQL支持在select中调用udf，但是udf中不能有SQL语句，否则会收到如下的错误信息：

  ```
  ERROR: function cannot execute on segment because it accesses relation "public.t2" (
  functions.c:155) (seg1 slice1 127.0.0.1:25433 pid=52153) (cdbdisp.c:1326)
  DETAIL:
  SQL statement "select b from t2 where a = $1 "
  ```

  可以把select中的udf转换成SQL表达式或者子查询等方法来转换。

  示例：

  ```
  create or replace FUNCTION f1(arg int) RETURN int IS
    v int;
  ```

```
BEGIN
   select b into v from t2 where a = arg;
   return v;
END;

select a, f1(b) from t1;
```

转换成：

```
select t1.a, t2.b from t1, t2 where t1.b = t2.a;
```

- **（+）多表外链接**

  AnalyticDB for PostgreSQL不支持(+)语法形式，需要转换成标准的outer join语法。

  示例：

  ```
  oracle
  select * from a,b where a.id=b.id(+)
  ```

  转换成：

  ```
  select * from a left join b on a.id=b.id
  ```

  若在(+)中有三表的join，需要先用wte做两表的join，再用+号那个表跟wte表做outer join。

  示例：

  ```
  Select * from test1 t1, test2 t2, test3 t3 where t1.col1(+) between NVL(t2.col1, t3.col1)
  and NVL(t3.col1, t2.col1);
  ```

  转换成：

  ```
  with cte as (select t2.col1 as low, t2.col2, t3.col1 as high, t3.col2 as c2 from t2, t3)
  select * from t1 right outer join cte on t1.col1 between coalesce(cte.low, cte.high) and
  coalesce(cte.high,cte.low);
  ```

- **Merge into**

  对于merge into语法的转换，要先在AnalyticDB for PostgreSQL中使用update进行更新，然后使用`GET DIAGNOSTICS rowcount := ROW_COUNT;`语句获取update更新的行数，若update更新的行数为0，再使用insert语句进行插入。

  ```
  MERGE INTO test1 t1
          USING (SELECT t2.col1 col1, t3.col2 col2,
                FROM test2 t2, test3 t3) S
          ON S.col1 = 1 and S.col2 = 2
  WHEN MATCHED THEN
            UPDATE
            SET test1.col1 = S.col1+1,
                test1.col2 = S.col2+2
  WHEN NOT MATCHED THEN
            INSERT (col1, col2)
            VALUES
  ```

```
            (S.col1+1, S.col2+2);
```

转换成：

```
Update test1 t1 SET t1.col1 = test2.col1+1, test3.col2 = S.col2+2 where test2.col1 = 1
and test2.col2 = 2;
GET DIAGNOSTICS rowcount := ROW_COUNT;
if rowcount = 0 then
   insert into test1 values(test2.col1+1, test3.col2+2);
end if;
```

- **Sequence**

  示例：

  ```
  create sequence seq1;
  select seq1.nextval from dual;
  ```

  转换成：

  ```
  create SEQUENCE seq1;
  select nextval('seq1');
  ```

- **Cursor的使用**

  - 在Oracle中，可以使用下面的语句对cursor进行遍历。

    示例：

    ```
    FUNCTION test_func() IS
       Cursor data_cursor IS SELECT * from test1;
    BEGIN
       FOR I IN data_cursor LOOP
          Do something with I;
    END LOOP;
    END;
    ```

    转换成：

    ```
    CREATE OR REPLACE FUNCTION test_func()
    AS $body$
    DECLARE
    data_cursor cursor for select * from test1;
    I record;
    BEGIN
       Open data_cursor;
       LOOP
         Fetch data_cursor INTO I;
        If not found then
             Exit;
        End if;
        Do something with I;
       END LOOP;
       Close data_cursor;
    END;
    $body$
    ```

```
LANGUAGE PLPGSQL;
```

- Oracle可以在递归调用的函数里重复打开名字相同的cursor。但是在AnalyticDB for PostgreSQL中则无法重发重复打开，需要改写成for I in query的形式。

  示例：

```
FUNCTION test_func(level IN numer) IS
   Cursor data_cursor IS SELECT * from test1;
BEGIN
If level > 5 then
      return;
  End if;

   FOR I IN data_cursor LOOP
      Do something with I;
      test_func(level + 1);
END LOOP;
END;
```

  转换成：

```
CREATE OR REPLACE FUNCTION test_func(level int) returns void
AS $body$
DECLARE
data_cursor cursor for select * from test1;
I record;
BEGIN
   If level > 5 then
      return;
   End if;
   For I in select * from test1 LOOP
     Do something with I;
      PERFORM test_func(level+1);
   END LOOP;
END;
$body$
LANGUAGE PLPGSQL;
```

2 Teradata应用迁移至AnalyticDB for PostgreSQL

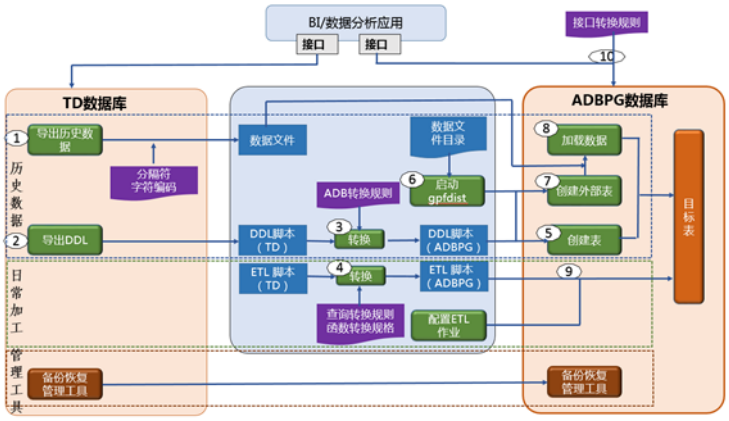
本文介绍如何将Teradata数据和应用迁移到云原生数据仓库PostgreSQL版。

迁移原则

云原生数据仓库PostgreSQL版对Teradata语法有着很好的兼容。本指南在将TD数仓应用迁移至ADB PG云化数仓过程中，秉承充分复用旧系统架构、ETL算法、数据结构和工具的原则，需对原加工脚本进行转换，另外，需对历史数据进行迁移，并保证数据的准确性，完整性。

- 对数据仓库基础数据平台的完整迁移；
- 对数据仓库系统上已部署应用的平滑迁移；
- 业务外观透明迁移，保持新旧系统业务操作一致性；
- 充分保证数据仓库迁移后的性能；
- 可接受的系统迁移周期及良好的迁移可操作性；
- 充分复用旧系统架构、ETL算法、数据结构和工具。

迁移流程



1. 历史数据迁移，首先从TD数据库按规定分隔符及字符编码将历史数据导成文本文件，存放于ADB PG数据库网络相通的ECS服务器本地磁盘或云存储OSS上，确保ADB PG数据库通过gpfdist协议的外部表后ADBPG的OSS外部表能读取数据文件。之后从TD导出DDL脚本，按ADB PG语法批量修改脚本，确保在ADB PG能成功创建所有用户表。
2. 日常加工流程迁移：对ETL查询加工语句按ADB PG的DML语法进行转换（ADBPG构建了相关基于脚本的自动化转化工具，可以对语法进行自动mapping转换），并根据TD与ADB PG函数对照表替换相关函数，转换ETL连接数据库方式。重新配置加工作业，历史数据迁移成功后，启动日常ETL作业。
3. 应用接口迁移：ADB PG数据库支持ODBC/JDBC，BI前端展现等工具可通过ODBC或JDBC标准访问DW，改动网络连接IP等即可。

4. 管理工具迁移：部署ADB PG备份及恢复工具，定期备份数据及定期进行恢复演练。

数据类型

云原生数据仓库PostgreSQL版和Teradata的核心数据类型是互相兼容的，仅部分数据类型需要进行修改，通过ADBPG 的自动化转化工具，可以批量进行TD建表DDL语句的转换。详情请参见下表：

Teradata	ADB PG
char	char
varchar	varchar
long varchar	varchar(64000)
varbyte(size)	bytea
byteint	无，可用bytea替代
smallint	smallint
integer	integer
decimal(size,dec)	decimal(size,dec)
numeric(precision,dec)	numeric(precision,dec)
float	float
real	real
double precision	double precision
date	date
time	time
timestamp	timestamp

建表语句

我们通过一个建表语句的例子来比较云原生数据仓库PostgreSQL版和Teradata。

Teradata建表SQL语句如下：

```
CREATE MULTISET TABLE test_table,NO FALLBACK ,
    NO BEFORE JOURNAL,
    NO AFTER JOURNAL,
    CHECKSUM = DEFAULT,
    DEFAULT MERGEBLOCKRATIO
    (
     first_column DATE FORMAT 'YYYYMMDD' TITLE '第一列' NOT NULL,
     second_column INTEGER TITLE '第二列' NOT NULL ,
     third_column CHAR(6) CHARACTER SET LATIN CASESPECIFIC TITLE '第三列' NOT NULL ,
     fourth_column CHAR(20) CHARACTER SET LATIN CASESPECIFIC TITLE '第四列' NOT NULL,
     fifth_column CHAR(1) CHARACTER SET LATIN CASESPECIFIC TITLE '第五列' NOT NULL,
     sixth_column CHAR(24) CHARACTER SET LATIN CASESPECIFIC TITLE '第六列' NOT NULL,
```

```
    seventh_column VARCHAR(18) CHARACTER SET LATIN CASESPECIFIC TITLE '第七列' NOT
NULL,
    eighth_column DECIMAL(18,0) TITLE '第八列' NOT NULL ,
    nineth_column DECIMAL(18,6) TITLE '第九列' NOT NULL )
PRIMARY INDEX ( first_column ,fourth_column )
PARTITION BY RANGE_N(first_column  BETWEEN DATE '1999-01-01' AND DATE '2050-12-31
' EACH INTERVAL '1' DAY );

CREATE INDEX test_index (first_column, fourth_column) ON test_table;
```

云原生数据仓库PostgreSQL版的建表语句如下：

```
CREATE TABLE test_table
   (
    first_column DATE NOT NULL,
    second_column INTEGER NOT NULL ,
    third_column CHAR(6) NOT NULL ,
    fourth_column CHAR(20) NOT NULL,
    fifth_column CHAR(1) NOT NULL,
    sixth_column CHAR(24) NOT NULL,
    seventh_column VARCHAR(18) NOT NULL,
    eighth_column DECIMAL(18,0) NOT NULL ,
    nineth_column DECIMAL(18,6) NOT NULL )
DISTRIBUTED BY ( first_column ,fourth_column )
PARTITION BY RANGE(first_column)
(START (DATE '1999-01-01')  INCLUSIVE
END (DATE '2050-12-31')  INCLUSIVE
EVERY (INTERVAL '1 DAY' ) );

create index test_index on test_table(first_column, fourth_column);
```

通过以上例子，我们可以清晰地发现云原生数据仓库PostgreSQL版和Teradata建表语句的异同：

- 核心数据类型互相兼容，数据类型无需修改。
- 均支持分布列，但语法不同，Teradata使用的是primary index，云原生数据仓库PostgreSQL版使用的是distributed by。
- 均支持PARTITION BY二级分区，语义相同但语法不同。
- 均支持对表创建索引，但语法不同。
- 云原生数据仓库PostgreSQL版不支持TITLE关键字，但是支持单独对列添加注释COMMENT，语法为`COMMENT ON COLUMN table_name.column_name IS 'XXX';`
- 云原生数据仓库PostgreSQL版不支持在定义char/varchar时声明编码类型，可以在连接数据库时，通过执行`SET client_encoding = latin1;`来声明编码类型。

导入导出数据格式

云原生数据仓库PostgreSQL版和Teradata均支持txt、csv格式的数据导入导出，与Teradata的区别在于数据文件的分隔符。

- Teradata支持双分隔符。
- 云原生数据仓库PostgreSQL版支持单分隔符。

SQL语句

云原生数据仓库PostgreSQL版和Teradata的大部分SQL语法都是兼容的，仅有部分Teradata语法需要进行修改。需要修改的语法如下所示：

- **cast**

  Teradata支持如下的cast语法：

  ```
  cast(XXX as int format '999999')
  cast(XXX as date format 'YYYYMMDD')
  ```

  而云原生数据仓库PostgreSQL版支持如下cast语法：

  ```
  cast(XXX as int)
  cast(XXX as date)
  ```

  云原生数据仓库PostgreSQL版不支持在cast中声明format。

  - 对于`cast(XXX as int format '999999')`，需要编写函数来实现相同功能。
  - 对于`cast(XXX as date format 'YYYYMMDD')`，云原生数据仓库PostgreSQL版支持date的显示格式为`'YYYY-MM-DD'`，不影响正常使用。

- **qualify**

  Teradata的qualify关键字，用与根据用户的条件，进一步过滤前序排序计算函数得到的结果。

  例如，Teradata的qualify关键字如下所示：

  ```
  SELECT itemid, sumprice, RANK() OVER (ORDER BY sumprice DESC)
      FROM (SELECT a1.item_id, SUM(a1.sale)
          FROM sales AS a1
          GROUP BY a1.itemID) AS t1 (itemid, sumprice)
      QUALIFY RANK() OVER (ORDER BY sum_price DESC) <=100;
  ```

  而云原生数据仓库PostgreSQL版不支持qualify关键字，需要将带qualify的SQL语句，修改为嵌套子查询：

  ```
  SELECT itemid, sumprice, rank from
  (SELECT itemid, sumprice, RANK() OVER (ORDER BY sumprice DESC) as rank
      FROM (SELECT a1.item_id, SUM(a1.sale)
          FROM sales AS a1
          GROUP BY a1.itemID) AS t1 (itemid,sumprice)
  )  AS a
  where rank <=100;
  ```

- **macro**

  Teradata通过macro来执行一组SQL语句，如下所示：

  ```
  CREATE MACRO Get_Emp_Salary(EmployeeNo INTEGER) AS (
    SELECT
    EmployeeNo,
    NetPay
  ```

```
  FROM
  Salary
  WHERE EmployeeNo = :EmployeeNo;
);
```

云原生数据仓库PostgreSQL版不支持macro，但是可以使用function语句来完成Teradata的macro功能：

```
CREATE OR REPLACE FUNCTION Get_Emp_Salary(
      EmployeeNo INTEGER,
      OUT EmployeeNo INTEGER,
      OUT NetPay FLOAT
) returns setof record AS
$$

      SELECT EmployeeNo,NetPay
      FROM Salary
      WHERE EmployeeNo = $1

$$
 LANGUAGE SQL;
```

函数转化

TD与ADB PG函数转换对照表

TD函数	ADBPG函数	说明
Zeroifnull	Coalesce	对数据作累计处理时，将空值作零处理
NULLIFZERO	Coalesce	对数据作累计处理时，忽略零值
Index	Position	字符串定位函数
Add_months	To_date	从某日期增加或减少指定月份的日期
format	To_char/to_date	函数定义数据格式
csum	可通过子查询方式实现	计算一列的连续的累计的值
MAVG	可通过子查询方式实现	基于预定的行数(查询宽度)计算一列的移动平均值
MSUM	可通过子查询方式实现	基于预定的查询宽度计算一列的移动汇总值
MDIFF	可通过子查询方式实现	基于预定的查询宽度计算一列的移动差分值
qualify	可通过子查询方式实现	QUALIFY子句限制排队输出的最终结果
Char/characters	length	字符个数

3 AWS Redshift应用和数据迁移至AnalyticDB for PostgreSQL

本文描述从AWS Redshift迁移数据到AnalyticDB for PostgreSQL的整体过程。

总体步骤

从Redshift迁移数据到AnalyticDB for PostgreSQL包含如下步骤：

1. 资源和环境准备，执行操作前需提前准备Amazon Redshift、Amazon S3（Amazon Simple Storage Service）、AnalyticDB for PostgreSQL和阿里云对象存储服务（OSS）的相关资源。
2. 将Redshift的数据导入到Amazon S3中。
3. 使用OSSImport将Amazon S3中CSV格式的数据文件导入到OSS。
4. 在目标AnalyticDB for PostgreSQL中创建和源Redshift对应的对象，包括模式（Schema）、表（Table）、视图（View）和函数（Function）。
5. 使用OSS外部表将数据导入到AnalyticDB for PostgreSQL。

整体迁移路径如下：

AWS上的准备工作

准备用户访问S3的安全凭证

包括如下信息：

- 访问密钥ID（AccessKeyID）和秘密访问密钥（Secret AccessKey）。
- S3的Endpoint，例如s3.ap-southeast-2.amazonaws.com。
- S3的Bucket名称，例如alibaba-hybrid-export。

导出数据格式约定

- 导出文件为CSV格式。
- 导出文件不得大于50MB。
- 导出文件中列的顺序必须和建表语句中列的顺序一致。
- 导出文件的数量最好和AnalyticDB for PostgreSQL计算组的数量一致或者是计算组数量的整数倍。

推荐的Redshift UNLOAD命令选项

经过大量的实践，我们建议使用类似如下的Redshift UNLOAD选项将数据导入到S3中：

```
unload ('select * from test')
to 's3://xxx-poc/test_export_'
access_key_id '<Your access key id>'
secret_access_key '<Your access key secret>'
DELIMITER AS ','
ADDQUOTES
ESCAPE
NULL AS 'NULL'
MAXFILESIZE 50 mb;
```

在上述样例中，推荐使用如下选项：

```
DELIMITER AS ','
ADDQUOTES
ESCAPE
NULL AS 'NULL'
MAXFILESIZE 50 mb
```

从Redshift导出DDL语句

从Redshift导出所有的DDL语句，包括但不限于创建模式、创建表、创建视图和创建函数的语句。

阿里云上的准备工作

准备阿里云RAM账户

- RAM账户ID
- RAM账户密码
- RAM账户AccessKeyID
- RAM账户AccessKeySecret

创建OSS存储空间（Bucket）

在AWS S3 Bucket所在地域，比如华北2（北京），创建一个OSS存储空间。OSS存储空间创建完成后，可以从OSS的控制台获取存储空间的访问域名，本文中会使用到**ESC的VPC网络访问（内网）**的访问域名信息。使用内网传输，可以保障数据传输的速度和安全性。

下载安装OSSImport

1. 在AWS S3 Bucket所在地域，创建一个ECS实例。我们选择创建带宽为100Mbps，系统镜像为Windows X64的ECS实例。
2. 在ECS系统中下载并安装单机模式的OSSImport。OSSImport的最新版本，可从#unique_6获取。
3. 单机模式的OSSImport软件包解压后，软件的文件结构如下：

```
ossimport
```

```
├──── bin
│   └──── ossimport2.jar  # 包括Master、Worker、Tracker、Console四个模块的总jar
├──── conf
│   ├──── local_job.cfg   # 单机Job配置文件
│   └──── sys.properties  # 系统运行参数配置文件
├──── console.bat        # Windows命令行，可以分布执行调入任务
├──── console.sh         # Linux命令行，可以分布执行调入任务
├──── import.bat         # Windows一键导入，执行配置文件为conf/local_job.cfg配置的数据迁移任务，包括启动、迁移、校验、重试
├──── import.sh          # Linux一键导入，执行配置文件为conf/local_job.cfg配置的数据迁移任务，包括启动、迁移、校验、重试
├──── logs               # 日志目录
└──── README.md          # 说明文档，强烈建议使用前仔细阅读
```

使用OSSImport将数据从S3导入到OSS中

配置OSSImport

在本文中，我们采用单机模式的OSSImport。请参考如下样例修改配置文件conf/local_job.cfg，请确保仅修改本样例提及的参数。关于OSSImport配置的详细信息，请参考#unique_6。

```
srcType=s3
srcAccessKey="your AWS Access Key ID"
srcSecretKey="your AWS Access Key Secret"
srcDomain=s3.ap-southeast-2.amazonaws.com
srcBucket=alibaba-hybrid-export
srcBucket=
destAccessKey="your Alibaba Cloud Access Key ID"
destSecretKey="your Alibaba Cloud Access Key Secret"
destDomain=http://oss-ap-southeast-2-internal.aliyuncs.com
destBucket=alibaba-hybrid-export-1
destPrefix=
isSkipExistFile=true
```

启动OSSImport迁移任务

在单机模式的OSSImport中，执行import.bat批处理文件启动迁移任务。

查看迁移任务的状态

在数据迁移过程中，您可以通过命令执行窗口查看任务的执行状态。另外，你还可通过Windows系统的任务管理器查看带宽的占用情况。

在本样例中，ECS和OSS存储空间位于相同的地域，采用内网传输，不受网速的限制；S3到ECS采用外网传输，有网速限制。数据的上传速度受限于下载速度，因此从ECS到OSS存储空间的上传速度几乎和S3到ECS的下载速度相同。

任务失败重试（可选）

由于网络或者其他因素，迁移任务可能失败。在ECS Windows系统中的CMD命令窗口中执行concole.bat retry命令重试任务。失败任务重试仅仅重新执行失败的子任务，不会重试已成功的子任务。

检查OSS存储空间的文件（可选）

您可在OSS控制台检查导入的数据文件。我们推荐使用ossbrowser客户端工具来查看和管理OSS存储空间中的文件。ossbrowser可从#unique_7获取。

整理CSV文件中的数据（可选）

> **说明：**
> 本操作仅仅提供一个参考样例，为可选步骤，您可以根据您的业务需求整理CSV文件。

- 将CSV文件中的NULL替换成空格。
- 将CSV文件中的\,替换成,。

推荐在本地进行数据整理。您先通过ossbrowser将文件下载到ECS中，进行数据整理。然后再将整理后的文件上传到一个新建的OSS存储空间中，以区别于原来从S3下载的CSV文件。无论是上传还是下载CSV文件，我们都推荐使用OSS的内网Endpoint，以降低内网流量的开销。

将Redshift的DDL语句转换成AnalyticDB for PostgreSQL的DDL语句

在创建AnalyticDB for PostgreSQL数据库对象之前，我们需要做一些必要的准备工作。主要是将上述步骤导出的Redshift DDL语句转换成AnalyticDB for PostgreSQL语法的DDL语句。下面我们将简要地介绍这些转换规则。

CREATE SCHEMA

按照AnalyticDB for PostgreSQL语法标准创建模式，将其保存为create_schema.sql。如下为具体的样例：

```
CREATE SCHEMA schema1
  AUTHORIZATION xxxpoc;
GRANT ALL ON SCHEMA schema1 TO xxxpoc;
GRANT ALL ON SCHEMA schema1 TO public;
COMMENT ON SCHEMA model IS 'for xxx migration  poc test';

CREATE SCHEMA oss_external_table
  AUTHORIZATION xxxpoc;
```

CREATE FUNCTION

由于AnalyticDB for PostgreSQL不兼容Redshift的某些SQL函数，因此你需要定制或者重写这些函数。涉及的函数举例如下：

- CONVERT_TIMEZONE(a,b,c)，使用如下语句替换：

  ```
  timezone(b, timezone(a,c))
  ```

- GETDATE()，使用如下语句替换：

  ```
  current_timestamp(0):timestamp
  ```

- 替换或优化用户定义函数（UDF）。

  例如，Redshift的SQL函数如下：

  ```
  CREATE OR REPLACE FUNCTION public.f_jdate(dt timestamp without time zone)
  RETURNS character varying AS
  '    from datetime import timedelta, datetime
       if dt.hour < 4:
            d = timedelta(days=-1)
            dt = dt + d
       return str(dt.date())'
  LANGUAGE plpythonu IMMUTABLE;
  COMMIT;
  ```

  替换成如下AnalyticDB for PostgreSQL函数，可以提升性能。

  ```
  to_char(a - interval '4 hour', 'yyyy-mm-dd')
  ```

- 其他Redshift标准的函数。

  在具体的实践中，您可以在Functions and Operators in PostgreSQL8.2中查询标准的PostgreSQL函数的用法，修改或者自行实现Redshift和AnalyticDB for PostgreSQL不兼容的函数。如下为一些相关资源：

  - ISNULL()
  - DATEADD()
  - DATEDIFF()
  - REGEXP_COUNT()
  - LEFT()
  - RIGHT()

CREATE TABLE

- 修改压缩编码。AnalyticDB for PostgreSQL目前并不支持所有的Redshift压缩编码。**不**支持的压缩编码包括：
  - BYTEDICT
  - DELTA
  - DELTA32K
  - LZO
  - MOSTLY8
  - MOSTLY16
  - MOSTLY32
  - RAW (no compression)
  - RUNLENGTH
  - TEXT255
  - TEXT32K
  - ZSTD

  必须删除Redshift建表语句中的ENCODE XXX，用如下子句代替。

  ```
  with (COMPRESSTYPE={ZLIB|QUICKLZ|RLE_TYPE|NONE})
  ```

- 修改分布键。Redshift支持三种分布键（分配），具体请参见分配方式。您需按照如下规则修改分布键。
  - EVEN分配（DISTSTYLE EVEN）：用distributed randomly代替。
  - KEY分配（DISKEY）：用distributed by (colname1,...)代替。
  - ALL分配（ALL）：不支持，直接删除。
- 修改排序键（SortKey）。删除Redshift的排序键子句[COMPOUND | INTERLEAVED] SORTKEY (column_name [, ...])]中的**COMPOUND**或者**INTERLEAVED**选项，使用如下子句代替：

  ```
  with(APPENDONLY=true,ORIENTATION=column)
  sortkey (volume);
  ```

  样例1

  Redshift的CREATE TABLE语句：

  ```
  CREATE TABLE schema1.table1
  (
     filed1 VARCHAR(100) ENCODE lzo,
     filed2 INTEGER DISTKEY,
     filed3 INTEGER,
     filed4 BIGINT ENCODE lzo,
     filed5 INTEGER,
  ```

```
)
INTERLEAVED SORTKEY
(
  filed1,
  filed2
);
```

转换成AnalyticDB for PostgreSQL的CREATE TABLE语句：

```
CREATE TABLE schema1.table1
(
  filed1 VARCHAR(100) ,
  filed3 INTEGER,
  filed5 INTEGER
)
WITH(APPENDONLY=true,ORIENTATION=column,COMPRESSTYPE=zlib)
DISTRIBUTED BY (filed2)
SORTKEY
(
  filed1,
  filed2
)
```

样例2

Redshift的CREATE TABLE语句，包含**ENCODE**和**SORTKEY**选项：

```
CREATE TABLE schema2.table2
(
  filed1 VARCHAR(50) ENCODE lzo,
  filed2 VARCHAR(50) ENCODE lzo,
  filed3 VARCHAR(20) ENCODE lzo,
)
DISTSTYLE EVEN
INTERLEAVED SORTKEY
(
  filed1
);
```

转换成AnalyticDB for PostgreSQL的CREATE TABLE语句：

```
CREATE TABLE schema2.table2
(
  filed1 VARCHAR(50),
  filed2 VARCHAR(50),
  filed3 VARCHAR(20),
)
WITH(APPENDONLY=true, ORIENTATION=column, COMPRESSTYPE=zlib)
DISTRIBUTED randomly
SORTKEY
(
  filed1
);
```

CREATE VIEW

同样需要将Redshift的CREATE VIEW语句转换成符合AnalyticDB for PostgreSQL语法的SQL语句，转换规则和CREATE TABLE的转换规则类似。

创建和配置AnalyticDB for PostgreSQL实例

根据如下内容，创建并配置实例。

- #unique_8
- #unique_9
- #unique_10

创建数据库对象

参考#unique_11，使用psql或者pgAdmin III 1.6.3客户端，登录数据库。

按照上述转换规则，将Redshift的DDL语句转换成符合AnalyticDB for PostgreSQL语法规则的DDL语句。然后执行这些DDL语句创建数据库对象。

CREATE EXTERNAL TABLE

AnalyticDB for PostgreSQL支持通过OSS外部表（即gpossext功能），将数据并行从OSS导入或导出到OSS，并支持通过gzip进行OSS外部表文件压缩，大量地节省存储空间及成本。请参考#unique_12，创建OSS外部表。

使用INSERT INTO脚本导入数据

在完成OSS外部表和目标数据库各个对象的创建后，您需要准备INSERT脚本，用于将OSS外部表的数据插入到AnalyticDB for PostgreSQL目标表中。请将该脚本保存为insert.sql文件，并执行该脚本。

插入语句的格式为：INSERT INTO <TABLE NAME> SELECT * FROM <OSS EXTERNAL TABLE NAME>;

例如：

```
INSERT INTO schema1.table1 SELECT * FROM oss_external_table.table1;
```

导入数据后，请使用SELECT语句查询导入后的数据并和源数据进行对比分析，验证导入前后数据的一致性。

使用VACUUM脚本清理数据库

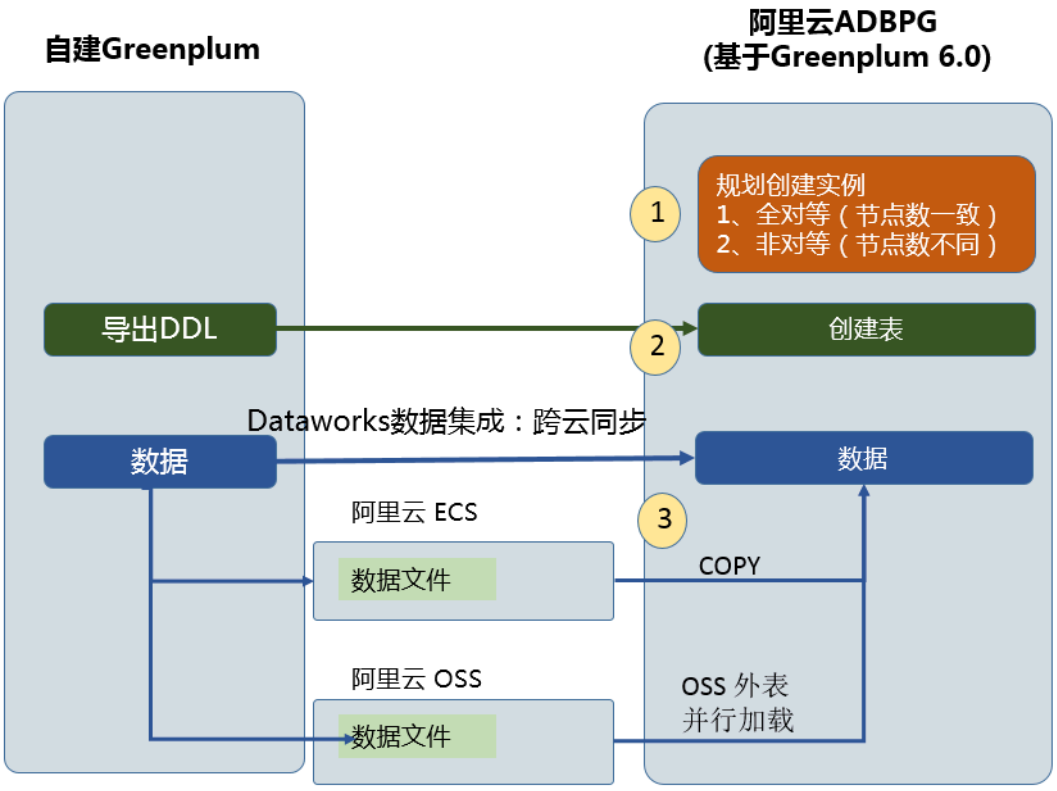
在将OSS外部表导入AnalyticDB for PostgreSQL后，您还需要使用VACUUM命令清理一下数据库，请将VACUUM脚本存储为vacuum.sql文件。关于VACUUM的用法，请参见VACUUM。

4 自建Greenplum迁移到AnalyticDB for PostgreSQL

阿里云AnalyticDB for PostgreSQL完全兼容开源Greenplum，支持应用平滑 。本文主要描述如何从自建Greenplum迁移到阿里云ADBPG 数仓平台。

迁移方案

阿里云AnalyticDB for PostgreSQL 6.0版本基于开源Greenplum 6.0构建，并深度优化演进，支持向量化计算，在Multi-Master架构下支持 事务处理，对外接口完全兼容社区Greenplum。整体迁移分为应用迁移和数据迁移。应用层迁移可以做到完全平滑实现，数据 提供多种方案。



1. 在阿里云规划并创建实例，规格设计可以参考#unique_14

2. 迁移表DDL，ADB PG完全兼容开源Greenplum，从自建GP可以基于 pg_dumpall 来导出所有表定义的DDL语句，可以在ADBPG上直接建立对应的表结构。

```
pg_dumpall --gp-syntax --schema-only > db_dump.sql
```

3. 数据迁移：数据迁移推荐如下三种方案，可以根据业务需求进行规划选择。

   - 通过Dataworks数据集成，不落地逐表进行数据 全量迁移。
   - 从自建GP导出数据，并上传到阿里云ECS，通过ADBPG 的 COPY命令工具导入数据到ADBPG实例。#unique_15
   - 从自建GP导出数据，并上传到阿里云对象存储，通过ADBPG 的OSS外表并行导入功能，高速加载数据到ADBPG实例。#unique_12

   dataworks数据同步操作简单，但相对数据同步和加载速度较慢。对于COPY和OSS两种加载方式，OSS外表并行加载 要快于COPY命令的数据加载。