ALIBABA CLOUD

阿里云

云原生数据仓库 AnalyticDB PostgreSQL 版

应用迁移

文档版本：20210713

阿里云

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. 阿里云网站上所有内容，包括但不限于著作、产品、图片、档案、资讯、资料、网站架构、网站画面的安排、网页设计，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
危险	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。	危险 重置操作将丢失用户配置数据。
警告	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。
注意	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。
说明	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	命令或代码。	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 PostgreSQL

AnalyticDB 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 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 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Orafce插件

AnalyticDB PostgreSQL中提供了Orafce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 PostgreSQL中使用。

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

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

Orafce插件提供的兼容函数如下表所示。

Orafce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)

函数	描述	示例
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)

函数	描述	示例

months_between(date1 date, date2 date)	返回date1和date2之间的月数。
	• 如果date1晚于date2，结果为正。
	• 如果date1早于date2，结果为负。
		postgres=# select months_between('2019-01-01', '2018-11-01');
		months_between
		---------- 	
		2
		(1 row)
		postgres=# select months_between('2018-11-01', '2019-01-01');
		months_between
		---------- 	
		-2
		(1 row)

trunc(value timestamp with time zone, fmt text)	• 参数一：要被截断的timestamp。
	• 参数二：应用于截断的度量单位，如年，月，日，周，时，分，秒等。
	  ○ Y：截断成日期年份的第一天。
	  ○ Q：返回季度的第一天。
		postgres=# SELECT TRUNC(current_date,'Q');
		trunc
		---------- 	
		2019-04-01
		(1 row)
		postgres=# SELECT TRUNC(current_date,'Y');
		trunc
		---------- 	
		2019-01-01
		(1 row)

trunc(value timestamp with time zone)	截断timestamp，默认截断时分秒。	postgres=# SELECT TRUNC('2019-12-11'::timestamp); trunc ---------- 2019-12-11 00:00:00+08 (1 row)
trunc(value date)	截断日期。	postgres=# SELECT TRUNC('2019-12-11'::timestamp,'Y'); trunc ---------- 2019-01-01 00:00:00+08

函数	描述	示例
round(value timestamp with time zone, fmt text)	将timestamp圆整到最近的unit_of_measure（日，周等）。	postgres=# SELECT round('2018-10-06 13:11:11'::timestamp, 'YEAR'); round --- 2019-01-01 00:00:00+08 (1 row)
round(value timestamp with time zone)	默认圆整到天。	postgres=# SELECT round('2018-10-06 13:11:11'::timestamp); round --- 2018-10-07 00:00:00+08 (1 row)

round(value date, fmt text)	参数类型为date。	postgres=# SELECT round(TO_DATE('27-OCT-00','DD-MON-YY'), 'YEAR');
		round
		--- 	
		2001-01-01
		(1 row)
		postgres=# SELECT round(TO_DATE('27-FEB-00','DD-MON-YY'), 'YEAR');
		round
		--- 	
		2000-01-01
		(1 row)

round(value date)	参数类型为date。	ostgres=# SELECT round(TO_DATE('27-FEB-00','DD-MON-YY')); round --- 2000-02-27 (1 row)

函数	描述	示例

instr(str text, patt text, start int, nth int)	在一个string中搜索一个substring，若搜索到则返回substring在string中位置，若没有搜索到，则返回0。
	• start：搜索的起始位置。
	• nth：搜索第几次出现的位置。
		postgres=# SELECT instr('Greenplum', 'e',1,2);
		instr
		------- 	
		4
		(1 row)
		postgres=# SELECT instr('Greenplum', 'e',1,1);
		instr
		------- 	
		3
		(1 row)

instr(str text, patt text, start int)	未提供nth参数，默认是第一次出现的位置。	postgres=# SELECT instr('Greenplum', 'e',1); instr ------- 3 (1 row)
instr(str text, patt text)	未提供start参数，默认从头开始搜索。	postgres=# SELECT instr('Greenplum', 'e'); instr ------- 3 (1 row)
plvstr.rvrs(str text, start int, end int)	str为输入的字符串start；end分别为对字符串从start到end这一段进行逆序。	postgres=> select plvstr.rvrs('adb4pg', 5,6); reverse --------- gp
plvstr.rvrs(str text, start int)	从start开始到字符串结束进行逆序。	ostgres=> select plvstr.rvrs('adb4pg', 4); reverse --------- gp4

函数	描述	示例
plvstr.rvrs(str text)	逆序整个字符串。	postgres=> select plvstr.rvrs('adb4pg'); reverse --------- gp4bda (1 行记录)
concat(text, text)	将两个字符串拼接在一起。	postgres=> select concat('adb','4pg'); concat -------- adb4pg (1 行记录)

concat(text, anyarray)/concat(anyarray, text)/concat(anyarray, anyarray)	用于拼接任意类型的数据。	postgres=> select concat('adb4pg', 6666);
		concat
		------------ 	
		adb4pg6666
		(1 行记录)
		postgres=> select concat(6666, 6666);
		concat
		---------- 	
		66666666
		(1 行记录)
		postgres=> select concat(current_date, 6666);
		concat
		---------------- 	
		2019-06-306666
		(1 行记录)

nanvl(float4, float4)/nanvl(float4, float4)/nanvl(numeric, numeric)	如果第一个参数为数值类型，则返回第一个参数；如果不为数值类型，则返回第二参数。	postgres=> select nanvl('NaN', 1.1);
		nanvl
		------- 	
		1.1
		(1 行记录)
		postgres=> select nanvl('1.2', 1.1);
		nanvl
		------- 	
		1.2
		(1 行记录)

函数	描述	示例

bitand(bigint, bigint)	将两个整形的二进制进行and操作，并返回and之后的结果，只输出一行。	postgres=# select bitand(1,3);
		bitand
		-------- 	
		1
		(1 row)
		postgres=# select bitand(2,6);
		bitand
		-------- 	
		2
		(1 row)
		postgres=# select bitand(4,6);
		bitand
		-------- 	
		4
		(1 row)

listagg(text)	将文本值聚集成一个串。	postgres=> SELECT listagg(t) FROM (VALUES('abc'), ('def')) as l(t); listagg --------- abcdef (1 行记录)
listagg(text, text)	将文本值聚集成一个串，第二个参数执行了分割符。	postgres=> SELECT listagg(t, '.') FROM (VALUES('abc'), ('def')) as l(t); listagg --------- abc.def (1 行记录)

nvl2(anyelement, anyelement, anyelement)	如果第一个参数不为null，那么返回第二个参数，如果第一个参数为null，则返回第三个参数。	postgres=> select nvl2(null, 1, 2);
		nvl2
		------ 	
		2
		(1 行记录)
		postgres=> select nvl2(0, 1, 2);
		nvl2
		------ 	
		1
		(1 行记录)

函数	描述	示例

lnnvl(bool)	如果参数为null或者false，则返回true；如果为true，则返回false。	postgres=> select lnnvl(null);
		lnnvl
		------- 	
		t
		(1 行记录)
		postgres=> select lnnvl(false);
		lnnvl
		------- 	
		t
		(1 行记录)
		postgres=> select lnnvl(true);
		lnnvl
		------- 	
		f
		(1 行记录)

dump("any")	返回一个文本值，该文本中包含第一个参数的数据类型代码、以字节计的长度和内部表示。	postgres=> select dump('adb4pg'); dump ---------- Typ=705 Len=7: 97,100,98,52,112,103,0 (1 行记录)

dump("any", integer)	第二个参数表示返回文本值的内部表示是使用10进制还是16进制，目前仅支持10进制和16进制。	postgres=> select dump('adb4pg', 10);
		dump
		---------- 	
		Typ=705 Len=7: 97,100,98,52,112,103,0
		(1 行记录)
		postgres=> select dump('adb4pg', 16);
		dump
		---------- 	
		Typ=705 Len=7: 61,64,62,34,70,67,0
		(1 行记录)
		postgres=> select dump('adb4pg', 2);
		ERROR: unknown format (others.c:430)

函数	描述	示例

nlssort(text, text)	指定排序规则的排序数据函数。	create table t1 (name text);
		INSERT INTO t1 VALUES('Anne'), ('anne'), ('Bob'), ('bob');
		postgres=> select from t1 order by nlssort(name, 'en_US.UTF-8');
		name
		------ 	
		anne
		Anne
		bob
		Bob
		(4 行记录)
		postgres=> select from t1 order by nlssort(name, 'C');
		name
		------ 	
		Anne
		Bob
		anne
		bob
		(4 行记录)

substr(str text, start int)	获取参数一中字符串的子串。参数二表示获取到子串的起始位置 >= start。	postgres=> select substr('adb4pg', 1);
		substr
		-------- 	
		adb4pg
		(1 行记录)
		postgres=> select substr('adb4pg', 4);
		substr
		-------- 	
		4pg
		(1 行记录)

substr(str text, start int, len int)	参数三会指定子串的结束位置 >= start and <= end 。	postgres=> select substr('adb4pg', 5,6); substr -------- pg (1 行记录)

函数	描述	示例
pg_catalog.substrb(varchar2, integer, integer)	varchar2类型的获取子串函数；参数二为start pos，参数三为end pos。	postgres=> select substr('adb4pg'::varchar2, 5,6) ; substr -------- pg (1 行记录)
pg_catalog.substrb(varchar2, integer)	varchar2类型的获取子串函数；参数二为start pos，从start pos一直取到字符串结束。	postgres=> select substr('adb4pg'::varchar2, 4) ; substr -------- 4pg (1 行记录)
pg_catalog.lengthb(varchar2)	获取varchar2类型字符串占的字节数。若输入为null返回null，输入为空字符，则返回0。	ostgres=> select lengthb('adb4pg'::varchar2) ; lengthb --------- 6 (1 行记录) postgres=> select lengthb('分析型'::varchar2) ; lengthb --------- 9 (1 行记录)

Orafce插件除了提供上述兼容函数，还对Oracle的Varchar2数据类型提供了兼容。

对于以下的四个Oracle函数，在AnalyticDB PostgreSQL中，无需安装Orafce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)

函数	描述	示例
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 PostgreSQL
VARCHAR2	varchar or text
DATE	timestamp
LONG	text
LONG RAW	bytea
CLOB	text

Oracle	AnalyticDB PostgreSQL
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 PostgreSQL
sysdate	current timestamp
trunc	trunc/ date trunc

Oracle	AnalyticDB PostgreSQL
dbms_output.put_line	raise 语句
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 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 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 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 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 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 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 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 PostgreSQL中没有PRAGMA语句，删除该类语句。

- 事务处理

  AnalyticDB 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 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 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

- 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 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 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 varc
har(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 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 col
  3 = 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 d
  ays'::interval)::int INTO v_days;
  ```

PL数据类型

- Record

  使用AnalyticDB 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AnalyticDB PostgreSQL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 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 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 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
$$
declare
  idx int := 0;
  res_tbl varchar(265) := 'result_table';
  prev_tbl varchar(265) := 'tmp_prev';
  curr_tbl varchar(256) := 'tmp_curr';
  current_result_sql varchar(4000);
  tbl_count int;
  rec record;
begin
  execute 'truncate ' || prev_tbl;
  execute 'truncate ' || curr_tbl;
  execute 'truncate ' || res_tbl;
  loop
    -- 查询当前层次结果，并插入到tmp_curr表
    current_result_sql := 'insert into ' || curr_tbl || ' select t1.* from ' || tablename || ' t1';
    if idx > 0 then
      current_result_sql := current_result_sql || ', ' || prev_tbl || ' t2 where t1.lead_id = t2.emp_id';
    else
      current_result_sql := current_result_sql || ' where t1.lead_id = ' || lead_id;
    end if;
    execute current_result_sql;
    -- 如果有环，删除已经遍历过的数据
    if nocycle is false then
      execute 'delete from ' || curr_tbl || ' where (lead_id, emp_id) in (select lead_id, emp_id from ' || re
s_tbl || ') ';
    end if;
    -- 如果没有数据，则退出
    execute 'select count(*) from ' || curr_tbl into tbl_count;
    exit when tbl_count = 0;
    -- 把tmp_curr数据保存到result表
    execute 'insert into ' || res_tbl || ' select * from ' || curr_tbl;
    execute 'truncate ' || prev_tbl;
    execute 'insert into ' || prev_tbl || ' select * from ' || curr_tbl;
    execute 'truncate ' || curr_tbl;
    idx := idx + 1;
  end loop;
  -- 返回结果
  current_result_sql := 'select * from ' || res_tbl;
  for rec in execute current_result_sql loop
    return next rec;
  end loop;
  return;
end
$$
 language plpgsql;
```

- Rownum

  i. 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;
```

ii. 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
  i. 去掉dual。

  示例：

  ```
  select sysdate from dual;
  ```

  转换成：

  ```
  select current_timestamp;
  ```

  ii. 创建一个叫dual的表。
- Select中的udf

  AnalyticDB 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 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.hi
gh,cte.low);
```

- Merge into

  对于merge into语法的转换，要先在AnalyticDB 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 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 PostgreSQL

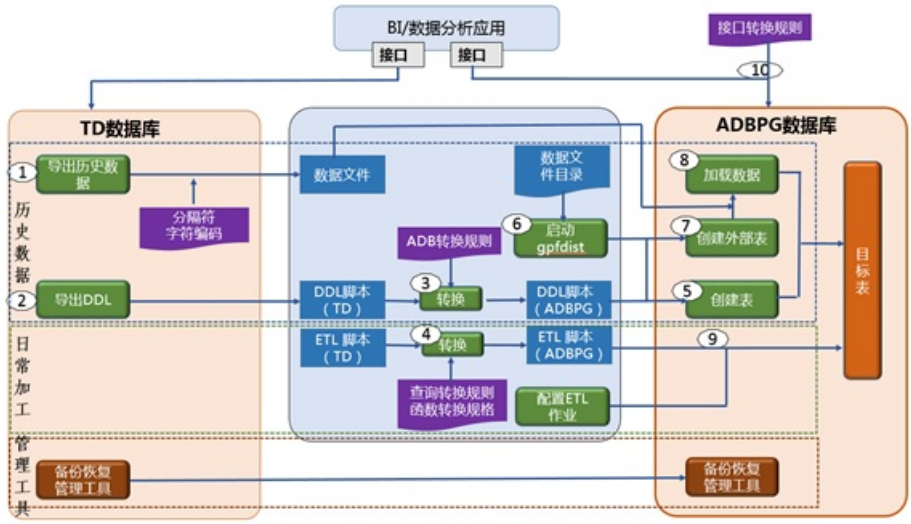
本文介绍如何将Teradata数据和应用迁移到云原生数据仓库AnalyticDB PostgreSQL版。

迁移原则

云原生数据仓库AnalyticDB 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AnalyticDB PostgreSQL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AnalyticDB PostgreSQL数据库网络相通的ECS服务器本地磁盘或云存储OSS上，确保AnalyticDB PostgreSQL数据库通过gpfdist协议的外部表后AnalyticDB PostgreSQL的OSS外部表能读取数据文件。之后从TD导出DDL脚本，按AnalyticDB PostgreSQL语法批量修改脚本，确保在AnalyticDB PostgreSQL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AnalyticDB PostgreSQL的DML语法进行转换（AnalyticDB PostgreSQL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AnalyticDB PostgreSQL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：AnalyticDB PostgreSQL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AnalyticDB PostgreSQL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AnalyticDB PostgreSQL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	AnalyticDB PostgreSQL
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AnalyticDB PostgreSQL函数转换对照表

TD函数	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.Amazon Redshift应用和数据迁移至AnalyticDB PostgreSQL

本文介绍从Amazon Redshift迁移数据到云原生数据仓库AnalyticDB PostgreSQL版的过程。

准备工作

- 需要迁移的Amazon Redshift实例。
- 用于导出Amazon Redshift数据的Amazon S3服务。
- 已开通阿里云对象存储服务（OSS），OSS的详细信息，请参见什么是对象存储OSS。
- 已创建云原生数据仓库AnalyticDB PostgreSQL版实例，如何选择实例规格，请参见规格选型。

迁移大纲

具体迁移步骤，请参见数据迁移步骤。

规格选型

以下内容将为您介绍如何选择根据原有Amazon Redshift实例判断需要的云原生数据仓库AnalyticDB PostgreSQL版的规格。

Amazon Redshift实例由**Leader Node**和多个**Compute Node**组成。

- **Leader Node**：相当于AnalyticDB PostgreSQL版的Master节点，负责与客户端通信，分析和制定执行计划，实施数据库操作。
- **Compute Node**：相当于AnalyticDB PostgreSQL版的一个计算组，由多个Node Slices组成。每个Node Slices都相当于AnalyticDB PostgreSQL版的单个Segment节点，负责实际的数据存储和查询计算。

因此，当您创建AnalyticDB PostgreSQL版实例时，如果不知道如何选择规格，可以通过Amazon Redshift中每个Node Slice的规格来选择合适的AnalyticDB PostgreSQL版节点规格。

示例如下：

Amazon Redshift实例包含4个计算节点，每个节点规格为2核16 GB，存储为1 TB；每个节点下包含两个Node Slices。

根据源Amazon Redshift实例规格，您在创建AnalyticDB PostgreSQL版实例时，可以选择创建8个Segment节点，节点规格为2C16GB；每个Segment节点的存储容量选择1000 GB。

> **说明**
> - 创建AnalyticDB PostgreSQL版实例具体操作，请参见创建实例。
> - 存储磁盘类型建议选择**ESSD云盘**，I/O性能比高效云盘更佳。
> - 按量付费实例在创建后支持升降配。

数据迁移步骤

1. 将Amazon Redshift的数据导出到Amazon S3中。您可以使用UNLOAD命令进行导出，如何导出，请参见UNLOAD。

```
UNLOAD ('select-statement')
TO 's3://object-path/name-prefix'
authorization
[ option [ ... ] ]
where option is
{ [ FORMAT [ AS ] ] CSV | PARQUET
| PARTITION BY ( column_name [, ... ] ) [ INCLUDE ]
| MANIFEST [ VERBOSE ]
| HEADER
| DELIMITER [ AS ] 'delimiter-char'
| FIXEDWIDTH [ AS ] 'fixedwidth-spec'
| ENCRYPTED [ AUTO ]
| BZIP2
| GZIP
| ZSTD
| ADDQUOTES
| NULL [ AS ] 'null-string'
| ESCAPE
| ALLOWOVERWRITE
| CLEANPATH
| PARALLEL [ { ON | TRUE } | { OFF | FALSE } ]
| MAXFILESIZE [AS] max-size [ MB | GB ]
| REGION [AS] 'aws-region' }
```

> **说明**
> - 导出数据时，建议使用FORMAT AS PARQUET或CSV格式。
> - 导出数据时，建议使用并行导出（PARALLEL ON），提高导出效率，生成更多文件分片。
> - 导出数据时，建议通过MAXFILESIZE参数设置合适的文件段大小，尽可能生成多个文件段，推荐数量为AnalyticDB PostgreSQL版实例的节点个数的整数倍，便于后续OSS外表并行导入AnalyticDB PostgreSQL版，提高效率。

2. 将Amazon S3数据同步到OSS中。
   i. 创建OSS存储空间，请参见创建存储空间。

   > **说明** 建议OSS存储空间与AnalyticDB PostgreSQL版实例在同一地域，便于后续数据导入数据库中。

ii. 下载并安装单机模式的ossimport，如何下载并安装，请参见说明及配置。

单机模式的ossimport软件的文件结构如下：

```
ossimport
├── bin
│   └── ossimport2.jar # 包括Master、Worker、TaskTracker、Console四个模块的总jar
├── conf
│   ├── local_job.cfg  # 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
```

iii. 配置单机模式的ossimport，请参考如下示例修改配置文件*conf/local_job.cfg*，关于ossimport配置的详细信息，请参见说明及配置。

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

> **说明** 仅需修改以上示例中的参数。

iv. 运行ossimport将数据同步至OSS，使用单机版ossimport的具体操作，请参见单机部署。

3. 将OSS数据导入AnalyticDB PostgreSQL版实例。

i. 修改DDL定义，修改SCHEMA、TABLE、FUNCTION、VIEW等对象信息，具体信息，请参见DDL语法转换。

ii. 使用COPY FROM OSS命令将数据导入到AnalyticDB PostgreSQL版实例，使用方法，请参见使用COPY/UNLOAD导入/导出数据到OSS。

```
COPY table_name [ ( column_name [, ...] ) ]
  FROM 'data_source_url'
  ACCESS_KEY_ID 'access_key_id'
  SECRET_ACCESS_KEY 'secret_access_key'
  [ [ FORMAT ] [ AS ] data_format ]
  [ MANIFEST ]
  [ [ option value] ... ]
where data_source_url should be in format:
  oss://{bucket_name}/{path_prefix}
```

- 示例一：使用COPY命令从OSS上导入PARQUET格式文件

```
COPY tp
FROM  'oss://adbpg-regress/test_parquet/'
ACCESS_KEY_ID 'id'
SECRET_ACCESS_KEY 'key'
FORMAT AS PARQUET
ENDPOINT 'oss-****.aliyuncs.com'
FDW 'oss_fdw';
```

- 示例二：使用COPY命令从OSS上导入CSV格式文件，只导入a和c两列，b列默认为NULL。

```
COPY local_t2 (a, c)
FROM 'oss://adbpg-regress/local_t/'
ACCESS_KEY_ID 'id'
SECRET_ACCESS_KEY 'key'
FORMAT AS CSV
ENDPOINT 'oss-****.aliyuncs.com'
FDW 'oss_fdw';
```

DDL语法转换

Amazon Redshift DDL语法与AnalyticDB PostgreSQL版DDL语法略有不同，迁移前您需要对这些语法进行转换。

- **CREATE SCHEMA**

  按照AnalyticDB PostgreSQL版语法标准创建Schema，示例如下：

```
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 PostgreSQL版不兼容Amazon Redshift的部分SQL函数，因此您需要定制或者重写这些函数。示例如下：

  - `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
  ```

- 用户定义函数（UDF）替换或优化。Redshift示例函数如下：

  ```
  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;
  ```

  使用如下语句替换：

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

- 其他Amazon Redshift标准的函数。

  您可以在Functions and Operators中查看标准的PostgreSQL函数的用法，修改或自行实现Amazon Redshift和AnalyticDB PostgreSQL版不兼容的函数，例如：

  - DATEADD()
  - DATEDIFF()
  - REGEXP_COUNT()

- **CREATE TABLE**

  Amazon Redshift的表名最大有效长度为127个字节，而AnalyticDB PostgreSQL版的表名默认最大有效长度为63字节。若表、函数、视图等对象名称超过63个有效字符时，需要裁剪名称。

  - 修改压缩编码

    您需要删除Amazon Redshift建表语句中的`ENCODE XXX`，用如下子句代替。

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

    AnalyticDB PostgreSQL版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

- 修改分布键

  Amazon Redshift支持三种分布键（分配）。您需按照如下规则修改分布键：

  - EVEN分配（DISTSTYLE EVEN）：可用 `DISTRIBUTED RANDOMLY` 代替。
  - KEY分配（DISKEY）：可用 `DISTRIBUTED BY (column, [ ... ])` 代替。
  - ALL分配（ALL）：可用 `DISTRIBUTED REPLICATED` 代替。

- 修改排序键（SortKey）

  删除Amazon 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。

  ```
  order by (column, [ ... ])
  ```

示例如下：

- 示例一：

  Amazon 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 PostgreSQL版的CREATE TABLE语句：

  ```
  CREATE TABLE schema1.table1
  (
    filed1 VARCHAR(100) ,
    filed3 INTEGER,
    filed5 INTEGER
  )
  WITH(APPENDONLY=true,ORIENTATION=column,COMPRESSTYPE=zstd)
  DISTRIBUTED BY (filed2)
  ORDER BY (filed1, filed2);
  -- 排序
  SORT schema2.table1;
  MULTISORT schema2.table1;
  ```

- 示例二：

  Amazon 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 PostgreSQL版的CREATE TABLE语句：

```
CREATE TABLE schema2.table2
(
  filed1 VARCHAR(50),
  filed2 VARCHAR(50),
  filed3 VARCHAR(20),
)
WITH(APPENDONLY=true, ORIENTATION=column, COMPRESSTYPE=zstd)
DISTRIBUTED randomly
ORDER BY (filed1);
-- 排序
SORT schema2.table2;
MULTISORT schema2.table2;
```

  > **说明** 更多sortkey信息，请参见列存表使用排序键和粗糙集索引加速查询。

- **CREATE VIEW**

  您需要将Amazon Redshift的CREATE VIEW语句转换成符合AnalyticDB PostgreSQL版语法的SQL语句。CREATE VIEW与CREATE TABLE的转换规则基本一致。

  > **说明** 不支持WITH NO SCHEMA BINDING子句。

4.自建Greenplum迁移到AnalyticDB PostgreSQL版

AnalyticDB PostgreSQL 6.0版基于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版Greenplum。整体迁移分为应用迁移和数据迁移，应用层可以实现平滑迁移，数据迁移提供了多种方案。

自建Greenplum迁移到AnalyticDB PostgreSQL版整个迁移流程如下：

1. 评估迁移风险，指定迁移方案和计划。
2. 采购AnalyticDB PostgreSQL版测试实例，验证迁移方案（包含数据迁移验证和应用迁移验证）。
3. 采购AnalyticDB PostgreSQL版生产实例，迁移自建Greenplum生产库。
4. 业务连接到AnalyticDB PostgreSQL版生产实例，验证业务。

自建Greenplum的网络环境（AnalyticDB PostgreSQL版实例节点能否访问自建Greenplum节点）和版本（4X、5X、6X）的不同，则数据迁移方案也会有所不同，数据迁移主体步骤分为如下三步：

1. DDL Schema迁移。
   - library：pg_dump不能直接导出Schema，需要您手动在AnalyticDB PostgreSQL版实例中创建Schema。
   - 插件（extension）：您需要确认自建Greenplum与AnalyticDB PostgreSQL版支持的插件差异，AnalyticDB PostgreSQL版支持的插件，请参见扩展插件列表。
   - 语法兼容：AnalyticDB PostgreSQL版内核版本为PG 9.4，与Greenplum 4X版本的语法存在部分语法不兼容的情况，需要您手动修改。
2. 表数据迁移。
   - 全量迁移和增量迁移：建议您通过业务层面来设计迁移，例如对表增加字段（时间戳或TAG）来区别增量数据。
   - 分区表或INHERIT表：建议从子表粒度迁移。
3. 数据校验与业务验证。
   - 关于数据校验，除了表COUNT总数校验外，SUM、MIN、MAX等计算字段也需要验证。
   - 关注视图、存储过程、业务SQL运行的正确性。

AnalyticDB PostgreSQL版实例节点无法访问自建Greenplum节点

1. **Schema迁移**
   i. 在自建Greenplum实例的Master节点，通过pg_dumpall或pg_dump工具导出DDL Schema，BASH命令示例如下：

```
export PGHOST=<源实例host>
export PGPORT=<源实例port>
export PGUSER=<源实例超级用户>
export PGPASSWORD=<源实例密码>
#-- 4X实例全部schema导出
pg_dumpall -s -q --gp-syntax > full_schema.sql
#-- 5X/6X实例全部schema导出
pg_dumpall -s --gp-syntax > full_schema.sql
```

ii. 使用psql登录自建Greenplum数据库，检查是否有自定义library，如果存在则需要手动在AnalyticDB PostgreSQL版数据库中创建library，检查语句如下：

```
select * from pg_catalog.pg_library;
```

iii. 使用psql登录AnalyticDB PostgreSQL版实例，执行DDL Schema，bash命令示例如下：

```
export PGHOST=<目标实例host>
export PGPORT=<目标实例port>
export PGUSER=<目标实例超级用户>
export PGPASSWORD=<目标实例密码>
psql postgres -f full_schema.sql > psql.log 2>&1 &
```

检查*psql.log*是否有产生错误信息，并对错误信息进行分析解决。错误信息基本都与SQL语法有关（尤其4X版本），不兼容项的相关检查，请参见4.3版本与6.0版本兼容性注意事项和4.3版升级6.0版不兼容项检查参考指南。

2. **表数据迁移**

i. 导出自建Greenplum的表数据，您可以通过COPY TO或GPFDIST外表两种方法导出：

- COPY TO

  在psql客户端使用COPY TO命令导出表数据，SQL示例如下：

  ```
  -- 4X/5x/6x
  COPY public.t1 TO '/data/gpload/public_t1.csv' FORMAT CSV ENCODING 'UTF8';
  -- 5x/6x支持 on segment导出
  -- 这里的<SEGID>不要改，系统内部会识别，<SEG_DATA_DIR>可以自定义
  COPY public.t1 TO '<SEG_DATA_DIR>/public_t1_<SEGID>.csv' FORMAT CSV ENCODING 'UTF8'
  ON SEGMENT;
  ```

  COPY命令具体使用方式，请参见COPY命令导入或导出本地数据。

- GPFDIST外表

  通过GPFDIST外表导出数据，您需要先启动GPFDIST服务（GPFDIST在自建Greenplum启动程序的bin目录下），bash命令参见：

  ```
  mkdir -p /data/gpload
  gpfdist -d /data/gpload -p 8000 &
  ```

  使用psql连接自建Greenplum数据库，导出表数据，SQL示例如下：

  ```
  -- public.t1为已知要迁移的表
  CREATE WRITABLE EXTERNAL TABLE ext_w_t1 (LIKE public.t1)
  LOCATION ('gpfdist://<host>:8000/public_t1.csv') FORMAT 'CSV' ENCODING 'UTF8';
  -- 写入外表
  insert into ext_w_t1 select * from public.t1;
  ```

ii. 将导出的数据上传到阿里云OSS上，关于阿里云OSS的详细信息，请参见什么是对象存储OSS。

> **说明** 阿里云OSS需要和AnalyticDB PostgreSQL版实例在同一地域。

iii. 使用psql客户端连接AnalyticDB PostgreSQL版实例，创建OSS Foreign Table，通过读取OSS Foreign Table将数据写入目标表，具体操作方式，请参见使用 OSS Foreign Table 访问 OSS 数据。

3. **数据验证**

   对比AnalyticDB PostgreSQL版实例内的数据与自建Greenplum内的数据，确保数据和业务没有区别。

   数据验证分为数据一致性验证和业务一致性验证：

   - 数据一致性验证：您可以全量对比数据也可以抽样对比数据，抽样对比需要重点验证表数据量是否一致和数值类型字段SUM、AVG、MIN、MAX是否一致两点。
   - 业务一致性验证：您需要将应用系统配置源指向AnalyticDB PostgreSQL版实例，检查业务功能正常性和执行结果正确性。

AnalyticDB PostgreSQL版实例节点可以访问自建Greenplum节点

基于GPTRANSFER工具来迁移DDL Schema和数据。

GPTRANSFER是4X和5X版本支持的数据迁移工具，由于4X版本和AnalyticDB PostgreSQL版存在部分语法差异，建议您将自建Greenplum升级至5X版本再通过GPTRANSFER工具迁移。

GPTRANSFER工具可从Greenplum源码的5X_STABLE分支的*/gpMgmt/bin*路径获取，编译部署后在bin目录下。

> **说明** 全量数据迁移过程中，建议不要往自建Greenplum实例中写入数据。

GPTRANSFER命令

将GPTRANSFER中几处代码注销后方可正常执行，需要注销的代码如下：

```
cmd = MakeDirectory('Create work directory',
            self._work_dir, REMOTE, self._options.dest_host)
self._pool.addCommand(cmd)
```

```
cmd = RemoveDirectory('Remove work directory',
             self._work_dir, REMOTE, self._options.dest_host)
self._pool.addCommand(cmd)
```

在自建Greenplum实例的master节点执行GPTRANSFER命令，全量数据迁移命令示例请参见：

```
-- 拉起源实例的linux当前用户密码
export SSHPASS=.....
python gptransfer -a --validate count --batch-size 8 --max-line-length 104857600 \
--source-host <源实例host> --source-port <源实例port> --source-user <源实例linux当前用户> \
--dest-host <ADB PG数据库链接串> --dest-port <ADB PG数据库端口> --dest-user <ADB PG数据库用户> \
--source-map-file <源实例segment ip配置文件> --work-base-dir <工作目录> \
--full --truncate --analyze --no-final-count > gptransfer.log &
```

其中参数介绍如下：

- `--batch-size`：并行度，可以并行传输多少张表，默认值为2，最大值为10。
- `--source-map-file`：源实例所有SEGMENT IP地址保存的文件（行内容唯一），格式如下：

  ```
  seg1_ip,seg1_ip
  seg2_ip,seg2_ip
  ```

GPTRANSFER也支持指定单库单表迁移，详细可执行`python gptransfer --help`命令查看帮助。

GPTRANSFER注意项

- GPTRANSFER不支持增量迁移。
- 自建Greenplum用到的Extension包含在AnalyticDB PostgreSQL版实例支持的Extension内。
- 自建Greenplum如果重度使用索引，建议先删除索引，等数据迁移完再重建，将有利于加速数据迁移。
- 自建Greenplum环境所有节点的8000 ~ 9999端口需要开放。
- AnalyticDB PostgreSQL版实例能够访问自建Greenplum环境的gpfdist服务（可在AnalyticDB PostgreSQL版创建可读外表，测试SELECT是否返回结果）。
- 如果gpfdist服务只能通过外网访问（注意外网流量是否限制），需要修改gptransfer部分逻辑方可执行。
- 在自建Greenplum实例master节点，需要psql能够免密连接到AnalyticDB PostgreSQL版实例（AnalyticDB PostgreSQL版实例控制台需要设置自建Greenplum实例master节点IP白名单）。

如果gpfdist服务只能通过外网访问，修改gptransfer部分逻辑地方请参见下图。

```
2143    def _create_dest_ext(self):
2144        """
2145        Creates the readable external table on the destination GPDB system.
2146        """
2147
2148        logger.debug('Creating external table for destination table %s...',
2149                     self._table_pair.dest)
2150        urls = ','.join(["'%s'" % url for url in self._ext_gpfdist_urls])
2151        ext_sql = \
2152            """CREATE EXTERNAL TABLE gptransfer.%s (LIKE \"%s\".\"%s\")
2153               LOCATION(%s)
2154               FORMAT '%s' """ \
2155            % (self._ext_name,
2156               self._table_pair.dest.schema,
2157               self._table_pair.dest.table,
2158               urls,
2159               self._format)
2160        if self._format.lower() == 'csv':
2161            ext_sql += """(DELIMITER AS ',' QUOTE AS E'%s')""" % self._quote
2162        elif self._format.lower() == 'text':
2163            ext_sql += """(DELIMITER AS E'%s' ESCAPE AS 'off')""" % self._delimiter
2164        cur = execSQL(self._dest_conn, ext_sql)
2165        cur.close()
```

- 修改 `urls` 参数初始化值，将内网地址改为外网地址。
- 检查gptransfer.log是否有错误信息输出，分析错误信息并解决。

其他迁移方式

您可也以使用阿里云DataWorks工具来迁移数据。

- DataWorks简介，请参见DataWorks官网。
- DataWorks集成指导，请参见DataWorks数据集成。
- DataWorks调度配置，请参见DataWorks作业调度。