Oracle to Azure PostgreSQL Migration Workarounds



Prepared by

**Paula Berenguel**

Solution Architect, DMJ Engineering Program

[pabereng@microsoft.com](mailto:pabereng@microsoft.com)

Dec-2018

**Disclaimer**

The High-Level Architecture, Migration Dispositions and guidelines in this document is developed in consultation and collaboration with Microsoft Corporation technical architects. This document developed was based on the information provided to Microsoft by <customer> and represents current Microsoft high level architectural guidelines and practices. Because Microsoft must respond to changing market conditions, this document should not be interpreted as an invitation to contract or a commitment on the part of Microsoft.

Microsoft has provided generic high-level guidance in this document with the understanding that <customer> would undertake detailed design and comprehensive reviews of the overall solution before the solution would be implemented/delivered. Microsoft is not responsible for the final implementation undertaken by <customer>. MICROSOFT MAKES NO WARRANTIES, EXPRESS OR IMPLIED, TO <customer> WITH RESPECT TO THE INFORMATION CONTAINED HEREIN

Contents

[Introduction 4](#_Toc532511446)

[List of Workarounds 4](#_Toc532511447)

[Find below the list of workarounds available in this document: 4](#_Toc532511448)

[1. Database Link 4](#_Toc532511449)

[2. External Tables 4](#_Toc532511450)

[3. Synonyms 5](#_Toc532511451)

[4. Global Temporary Tables 6](#_Toc532511452)

[5. Virtual Column 7](#_Toc532511453)

[6. Connect By – Hierarchical query 8](#_Toc532511454)

[7. Reverse Index 9](#_Toc532511455)

[8. Index Organized Table 9](#_Toc532511456)

[Known Unsupported features: 10](#_Toc532511457)

Introduction

This document purpose is to provide Architects, Consultants, DBAs and related roles with a guide for quick fixing / working around issues while migrating workloads from Oracle to Azure Database for PostgreSQL.

List of Workarounds

Find below the list of workarounds available in this document:

|  |  |
| --- | --- |
| Oracle | PostgreSQL |
| Database Link | Foreign Data Wrapper |
| External Table | Foreign Table |
| Synonym | View / Set search\_path |
| Global Temporary Table | Unlogged Table / Temp Table |
| Virtual column | View / Function / Trigger |
| Connect by | With Recursive |
| Reverse Index | Functional Index |
| Index Organized Table (IOT table) | Cluster the table according to an Index |

1. Database Link

Oracle syntax:

CREATE PUBLIC DATABASE LINK remote\_service USING 'remote\_db';

SELECT \* FROM employees@remote\_service;

Azure PostgreSQL syntax:

CREATE SERVER remote\_service FOREIGN DATA WRAPPER oracle\_fdw OPTIONS (dbserver 'remote\_db');

CREATE USER MAPPING FOR current\_user SERVER remote\_service OPTIONS (user 'scott', password 'tiger');

CREATE FOREIGN TABLE employees\_fdw (<columns\_list>) SERVER remote\_service OPTIONS(schema 'HR', table 'EMPLOYEES');

# External Tables

Oracle Syntax:

CREATE OR REPLACE DIRECTORY ext\_dir AS '/data/ext/';

CREATE TABLE ext\_table (empno VARCHAR2(4), firstname VARCHAR2(20), lastname VARCHAR2(20),

age VARCHAR2(2)) ORGANIZATION EXTERNAL (DEFAULT DIRECTORY ext\_dir ACCESS PARAMETERS (… LOCATION ('file\_ext.csv')));

cat /data/ext/file\_ext.csv

1234,ALBERT,GRANT,21

1235,ALFRED,BLUEOS,

26 1236,BERNY,JOLYSE,34

Azure PostgreSQL syntax:

CREATE FOREIGN TABLE ext\_table

(empno VARCHAR(4),

firstname VARCHAR(20),

lastname VARCHAR(20),

age VARCHAR(2) )

SERVER ext\_dir OPTIONS (filename '/data/ext/file\_ext.csv', format 'csv', delimiter ',');

# Synonyms

A synonym is an alias name for objects. They are used to make access to an object from another schema or a remote database simpler. Synonyms are not supported in PostgreSQL.

Oracle Syntax:

CREATE PUBLIC SYNONYM emp\_table FOR hr.employees [@ dblink];

Azure PostgreSQL syntax:

There are two options for migrating synonyms: use search\_path or use views:

--search path – session level – no permanent effect, it needs to be set for every connection

SET search\_path TO other\_schema;

--search path – role or database level – it takes permanent effect

--@postgresql

alter database <database\_name> set search\_path = "other\_schema";

--@database\_name

alter role <role\_name> set search\_path = "other\_schema";

--view:

CREATE VIEW public.emp\_table AS SELECT \* FROM hr.employees;

ALTER VIEW public.emp\_table OWNER TO hr;

GRANT ALL ON public.emp\_table TO PUBLIC;

Note that with @DBLINK clause, the creation of a foreign table HR.EMPLOYEES using a foreign server is required (Ora2Pg will warn you to see DBLINK and FDW export type).

# Global Temporary Tables

There are two options for migrating global temporary tables to Azure Database for PostgreSQL: use Unlogged tables or Temp Tables.

Option 1: Unlogged Table:

Oracle Syntax:

CREATE GLOBAL TEMPORARY TABLE MY\_CONTRACT\_MONTH

(ID NUMBER(10),

CMONTH DATE

)

ON COMMIT DELETE ROWS;

Azure PostgreSQL Syntax:

CREATE UNLOGGED TABLE MY\_CONTRACT\_MONTH

(

ID number,

CMONTH timestamp,

pid bigint default pg\_backend\_pid()

);

ALTER TABLE MY\_CONTRACT\_MONTH ENABLE ROW LEVEL SECURITY;

ALTER TABLE MY\_CONTRACT\_MONTH FORCE ROW LEVEL SECURITY;

CREATE POLICY cm\_pid ON MY\_CONTRACT\_MONTH TO <role\_name> USING (pid = (select pg\_backend\_pid()));

The Role is also the User. Please read PostgreSQL Users and Roles implementations if questions arise.

With Unlogged table, row level security must be implemented, to prevent sessions from querying other sessions data.

There is also a need of implementing a job that eliminates the data on the unlogged table for the inactive sessions.

DELETE FROM smdr.contract\_months cm WHERE not exists (select 1 from pg\_stat\_activity psa where psa.pid = cm.pid);

Option 2 – Temp table

Oracle Syntax:

CREATE GLOBAL TEMPORARY TABLE MY\_CONTRACT\_MONTH

(ID NUMBER(10),

CMONTH DATE

)

ON COMMIT DELETE ROWS;

Oracle stores the definitions of temporary tables permanently like the definitions of regular tables.

Azure PostgreSQL Syntax:

CREATE GLOBAL TEMPORARY TABLE MY\_CONTRACT\_MONTH

(ID NUMERIC,

CMONTH TIMESTAMP

)

ON COMMIT DELETE ROWS;

CREATE TEMPORARY TABLE statement creates a temporary table that is automatically dropped at the end of a session, or the current transaction (ON COMMIT DROP option).

During the conversion, you need to extract CREATE TEMPORARY TABLE statement from application code, stored procedures, triggers etc. and execute them once to create the temporary table definition.

* Oracle does not support ON COMMIT DROP, so if this option is required, you need to explicitly execute DROP TABLE statement after each COMMIT
* ON COMMIT PRESERVE ROWS is the default in PostgreSQL, while ON COMMIT DELETE ROWS is the default in Oracle

# Virtual Column

One way of implementing the equivalent of a virtual column in PostgreSQL is by creating a view, as following:

Oracle Syntax

CREATE TABLE VIRT\_COL\_TABLE (

id NUMBER,

first\_name VARCHAR2(10),

last\_name VARCHAR2(10),

salary NUMBER(9,2),

comm1 NUMBER(3),

comm2 NUMBER(3),

salary1 AS (ROUND(salary\*(1+comm1/100),2)),

salary2 NUMBER GENERATED ALWAYS AS (ROUND(salary\*(1+comm2/100),2)) VIRTUAL,

);

Azure Database for PostgreSQL Syntax

CREATE TABLE virt\_col\_table (

id bigint NOT NULL,

first\_name varchar(10),

last\_name varchar(10),

salary double precision,

comm1 smallint,

comm2 smallint,

salary1 bigint,

salary2 bigint

) ;

Ora2pg implements it by creating an additional export file named VIRTUAL\_COLUMNS\_(...).sql contains the trigger definition to apply the default values of the original virtual columns:

DROP TRIGGER IF EXISTS virt\_col\_VIRT\_COL\_TABLE\_trigger ON VIRT\_COL\_TABLE CASCADE;

CREATE OR REPLACE FUNCTION fct\_virt\_col\_VIRT\_COL\_TABLE\_trigger() RETURNS trigger AS $BODY$

BEGIN

NEW.SALARY2 = ROUND(NEW.SALARY\*(1+NEW.COMM2/100),2);

NEW.SALARY1 = ROUND(NEW.SALARY\*(1+NEW.COMM1/100),2);

RETURN NEW;

end

$BODY$

LANGUAGE 'plpgsql' SECURITY DEFINER;

CREATE TRIGGER virt\_col\_VIRT\_COL\_TABLE\_trigger

BEFORE INSERT OR UPDATE ON VIRT\_COL\_TABLE FOR EACH ROW

EXECUTE PROCEDURE fct\_virt\_col\_VIRT\_COL\_TABLE\_trigger();

# Connect By – Hierarchical query

Hierarchical queries in Oracle using the CONNECT BY clause are converted in PostgreSQL to queries using WITH RECURSIVE clause

Oracle Syntax

CREATE TABLE taxonomy (

key NUMBER(11) NOT NULL CONSTRAINT taxPkey PRIMARY KEY,

value VARCHAR2(255),

taxHier NUMBER(11) );

ALTER TABLE taxonomy ADD CONSTRAINT taxTaxFkey FOREIGN KEY (taxHier) REFERENCES tax(key);

SELECT

value

FROM

taxonomy

CONNECT BY

PRIOR key = taxHier

START WITH

key = 0;

Azure Database for PostgreSQL Syntax

WITH RECURSIVE cte AS (

SELECT key, value, 1 AS level

FROM taxonomy

WHERE key = 0

UNION ALL

SELECT t.key, t.value, c.level + 1

FROM cte c

JOIN taxonomy t ON t.taxHier = c.key

)

SELECT value

FROM cte

ORDER BY level;

# Reverse Index

This workaround is valid when the reverse index is applied against a TEXT column.

Oracle Syntax

CREATE TABLE REV\_TEMP (

Id NUMBER(10) NOT NULL PRIMARY KEY,

Description VARCHAR2(512) NOT NULL

) ;

CREATE INDEX REV\_TEMP\_N1 ON REV\_TEMP(Description) REVERSE;

Azure Database for PostgreSQL Syntax

CREATE TABLE REV\_TEMP (

Id NUMERIC(10) NOT NULL PRIMARY KEY,

Description VARCHAR(512) NOT NULL

) ;

CREATE INDEX REV\_TEMP\_N1 ON REV\_TEMP(REVERSE(Description));

# Index Organized Table

The Oracle database uses heap tables by default. Index-organized tables can be created using the ORGANIZATION INDEX clause:

Oracle Syntax

CREATE TABLE IOT\_TEMP (

Id NUMBER(10) NOT NULL PRIMARY KEY,

Description VARCHAR2(512) NOT NULL

) ORGANIZATION INDEX;

The Oracle database always uses the primary key as the clustering key.

Azure Database for PostgreSQL Syntax

PostgreSQL only uses heap tables. however, use the [CLUSTER clause to align the contents of the heap table with an index](https://www.postgresql.org/docs/current/static/sql-cluster.html).

CREATE TABLE IOT\_TEMP (

Id NUMERIC(10) NOT NULL PRIMARY KEY,

Description VARCHAR(512) NOT NULL

) ;

CREATE INDEX IOT\_TEMP\_N1 ON IOT\_TEMP(ID);

Known Unsupported features:

* Type inheritance and type with member method are not supported
* Global indexes over partitions are not supported
* Compound triggers are not supported