Oracle to Azure Database for PostgreSQL Migration Workarounds

Prepared by

Data SQL Ninja Engineering Team (datasqlninja@microsoft.com)

Note: The detail provided in this document has been harvested as part of a customer engagement sponsored through the Data SQL Ninja Engineering.

Table of Contents

1 Introduction

The purpose of this document is to provide Architects, Consultants, DBAs and related roles with a guide for remediating issues when migrating objects from Oracle to Azure Database for PostgreSQL. This guide is not limited to the PL/SQL to PG/PLSQL conversion; therefore, the instructions may touch on wider Azure ecosystem.

2 List of Workarounds

Find below the list of workarounds available in this document:

Oracle	Azure Database for PostgreSQL
Database Link	Non-Azure PostgreSQL or PostgreSQL: ADF pipeline For Azure Database for PostgreSQL and PostgreSQL : use the extension postgres_fdw
External Table	Blob Storage + ADF v2 pipeline use psql on Cloud Shell
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

3. Workarounds in detail

3.1 Database Link

Oracle supports heterogeneous services to allow data in non-Oracle database to be queried using SQL. This support has been in the form of transparent gateways, which are vendor specific, or generic connectivity which uses ODBC or OLEDB to make the connections. The functionality supported by generic connectivity is typically more limited than that possible when using vendor specific gateways, but it is quick and simple to configure. The bottom line is that one can connect Oracle to any ODBC compliant database (MS Access, SQL Server, MySQL etc.). Typical database link configuration below using Oracle syntax:

CREATE PUBLIC DATABASE LINK remote_service USING 'remote_db';

SELECT * FROM employees@remote_service;

Migration scenario I: Oracle is replaced by Azure Database for PostgreSQL and *remote_service* is another Azure Database for PostgreSQL

The solution is to use postgres_fdw extension:

```
CREATE EXTENSION postgres_fdw;

CREATE SERVER demoserver
FOREIGN DATA WRAPPER postgres_fdw
OPTIONS (host 'myreceivingserver.postgres.database.azure.com',
         dbname 'postgres', port '5432');

CREATE USER MAPPING FOR demouser
      SERVER demoserver
      OPTIONS (user 'demouser@myreceivingserver', password
'secretpassword');

CREATE FOREIGN TABLE inventory (
       id serial,
       name VARCHAR(50),
       quantity INTEGER
)
SERVER demoserver;
```

```
SELECT * FROM inventory;
```

Migration scenario II: Oracle is replaced by Azure Database for PostgreSQL and *remote_service* is another PostgreSQL server:

Same as migration scenario I. Use the extension postgres_fdw.

In case the PostgreSQL community server is not running in an Azure VM (PostgreSQL IaaS) it is important to understand what type of networking is running between the PostgreSQL server and Azure Database for PostgreSQL as performance can be penalized.

Migration scenario III: Oracle is replaced by Azure Database for PostgreSQL and *remote_service* is another RDBMS server:

In this scenario, the recommendation is to leverage whenever possible, Azure Data Factory v2 for moving the data asynchronously from *remote_service* to Azure Database for PostgreSQL in a schedule basis.

The solution involves having a local copy of the table inside Azure Database for PostgreSQL and having an scheduled ADFv2 pipeline that copies the data in a frequency that is acceptable to the business, into Azure Database for PostgreSQL.

3.2 External Tables

Oracle Syntax:

```
CREATE OR REPLACE DIRECTORY ext_dir AS '/data/ext/';

CREATE EXTERNAL 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 Database for PostgreSQL:

There are various solutions that can accommodate the behavior of an Oracle external table. Solutions will, inevitably, rely on creating a PostgreSQL regular table with some sort of load mechanism.

```
CREATE TABLE ext_table
(empno VARCHAR(4),
 firstname VARCHAR(20),
 lastname VARCHAR(20),
  age VARCHAR(2)
);
```

Load options:

a) Use psql or Cloud Shell to import data into the table manually – less preferred solution.
b) Leverage an Azure Storage Account and an ADFv2 pipeline to load data into the table.

   The Azure Storage account will be the host of csv, json, etc. within a container called landing zone, for example. Azure Data Factory v2 runs on a triggered event or against a schedule to consume the file from Storage account and load it into Azure Database for PostgreSQL.

3.3 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: search_path and views:

--search path – session level – no permanent effect, it must 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;
```

3.4 Global Temporary Tables

There are a few options for migrating global temporary tables to Azure Database for PostgreSQL. These are Unlogged tables and 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()));
```

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 MY_CONTRACT_MONTH 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 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 Oracle to Azure PostgreSQL conversion, you need to extract the DDL of the Oracle Global Temporary Table and convert it into "CREATE TEMPORARY TABLE" statement; once this is done, the application code has to be changed in order to execute the command.

- 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.

3.5 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, one of most popular Oracle to PostgreSQL conversion and migration tools, implements this 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();
```

3.6 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;
```

3.7 Reverse Index

This workaround is valid only when the reverse index is applied against an “string” 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));
```

3.8 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.

```
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);

CLUSTER IOT_TEMP USING IOT_TEMP_N1;
```

3.9 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.

4 Appendices

Azure Database for PostgreSQL documentation:

https://docs.microsoft.com/en-us/azure/postgresql/

Scaling out Azure Database for PostgreSQL using Foreign Data Wrappers:

https://techcommunity.microsoft.com/t5/azure-database-for-postgresql/scaling-out-azure-database-for-postgresql-using-foreign-data/ba-p/788518

5 Feedback and suggestions

If you have feedback or suggestions for improving this data migration asset, please contact the Data SQL Ninja Engineering Team (datasqlninja@microsoft.com). Thanks for your support!

Note: For additional information about migrating various source databases to Azure, see the Azure Database Migration Guide.