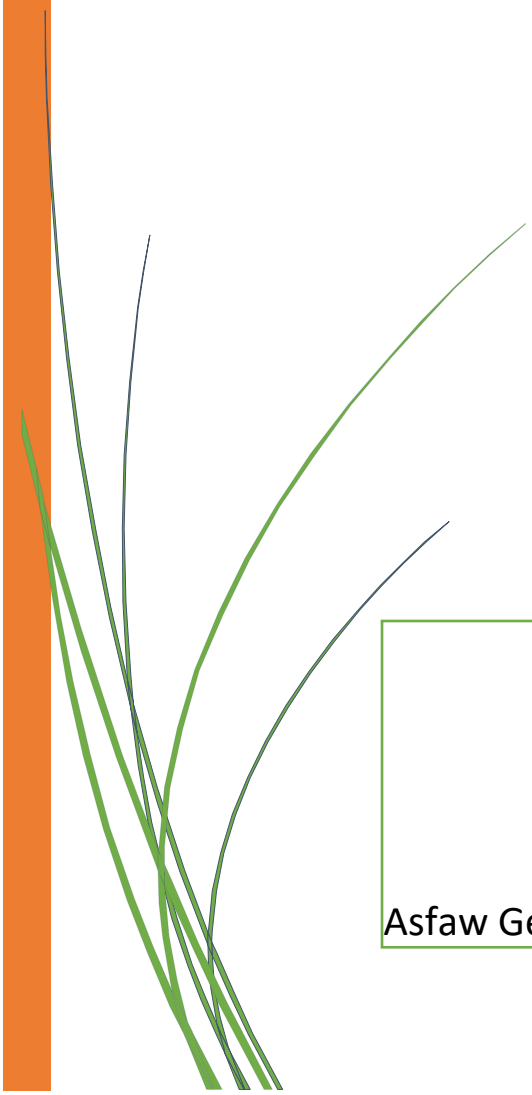
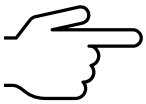
7/20/2023

How to Migrate from Oracle to PostgreSQL



Asfaw Gedamu



Caution: Please use the commands with care, try them on test environments first.

Migrating from Oracle Database to PostgreSQL can be a strategic move for organizations seeking cost savings, open-source flexibility, and robust community support. This document outlines the essential steps involved in the migration process, providing practical guidance and best practices.

Outline

1. **Introduction**
    - Understanding the Motivation for Migration
    - Key Benefits of PostgreSQL
2. **Assessment and Planning**
    - Analyzing Existing Oracle Environment
    - Identifying Dependencies and Constraints
    - Defining Migration Goals
3. **Data Migration**
    - Extracting Data from Oracle
    - Transforming Data for PostgreSQL
    - Loading Data into PostgreSQL
4. **Schema Migration**
    - Converting Oracle Schema Objects to PostgreSQL
    - Handling Data Types, Constraints, and Indexes
    - Addressing Differences in SQL Syntax
5. **Application Code Migration**
    - Adapting SQL Queries and Stored Procedures
    - Handling PL/SQL to PL/pgSQL Conversion
    - Testing Application Compatibility
6. **Testing and Validation**
    - Unit Testing
    - Integration Testing
    - Performance Testing
7. **Post-Migration Tasks**
    - Optimizing PostgreSQL Configuration

    - Monitoring and Tuning
    - User Training and Documentation
8. **Conclusion**
    - Celebrating a Successful Migration
    - Continuous Improvement

This guide aims to empower database administrators, developers, and IT teams with the knowledge needed to navigate the transition smoothly. Remember that each migration is unique, so adapt these guidelines to your specific environment. Good luck on your journey from Oracle to PostgreSQL!

There are several methods for migrating data from an Oracle database into PostgreSQL. For example, we can use the following methods:

Method 1: Ora2Pg

As the name suggests, Ora2Pg is used to migrate Oracle objects into PostgreSQL. This tool will connect to an Oracle database and generate SQL scripts that contain table structures and data that can be executed against PostgreSQL.

Some of the features include:

- Exports full database schema (such as tables, views including materialized view, sequences, and indexes).
- Exports specific tables.
- Exports user-defined functions, triggers, procedures, and packages.
- Migrates Oracle user-defined types.
- Supports Oracle BLOB object as BYTEA.
- Supports DBLINK as Oracle FDW.
- Supports SYNONYMS as views.

A sample database migration report can be found here: http://ora2pg.darold.net/report.html.
For more information please visit https://ora2pg.darold.net/start.html.

Method 2: Oracle foreign data wrappers (Oracle_fdw)

PostgreSQL can link to other systems to fetch data via foreign data wrappers (FDWs). When we fire a query (e.g., SELECT) against a foreign table, the FDW will fetch the result from the external data source and print the output via foreign table.

Oracle_fdw is used to perform DML operations (SELECT, INSERT, UPDATE, DELETE) on Oracle servers in PostgreSQL, which is covered under the SQL Management of External Data (SQL/MED) standard.

For more information please visit https://github.com/laurenz/oracle_fdw.

Here, we will discuss how to migrate data from Oracle to PostgreSQL using the first method(Ora2Pg).

Prerequisites:

- You will need to have a PostgreSQL database installed and running.
- You will need to have the necessary SQL scripts to migrate your data from Oracle to PostgreSQL.
- You will need to have a plan for how to handle any data that is not compatible with PostgreSQL.

  If not yet, please follow these steps:

  1. **Install PostgreSQL and Oracle client libraries on the target server.**

  sudo apt-get install postgresql postgresql-client-[oracle_version]

  2. **Create a new PostgreSQL database.**

  createdb my_database

  3. **Install the ora2pg tool on the target server.**

  pip install ora2pg

  4. **Configure ora2pg to connect to the Oracle database.**

```
vim /etc/ora2pg.conf
```

In the /etc/ora2pg.conf file, you need to specify the following information:

- The hostname and port of the Oracle database.
- The username and password of a user with sufficient privileges to access the Oracle database.
- The name of the Oracle database that you want to migrate. You can use the following script to begin with.

```
# The hostname and port of the Oracle database.
ORACLE_HOST=localhost
ORACLE_PORT=1521

# The username and password of a user with sufficient privileges to access the Oracle database.
ORACLE_USER=oracle
ORACLE_PASSWORD=password

# The name of the Oracle database that you want to migrate.
ORACLE_DATABASE=my_database

# The name of the PostgreSQL database that you want to create.
POSTGRES_DATABASE=my_postgresql_database

# The directory where you want to store the migrated data.
MIGRATED_DATA_DIR=/tmp/ora2pg

# The level of logging that you want to use.
LOG_LEVEL=INFO
```

- This is just a sample configuration file, and you may need to modify it to fit your specific needs. For example, you may need to change the ORACLE_HOST, ORACLE_PORT, ORACLE_USER, ORACLE_PASSWORD, ORACLE_DATABASE, POSTGRES_DATABASE, and MIGRATED_DATA_DIR variables to match your own environment.
- You can also specify additional configuration options in the /etc/ora2pg.conf file. For a complete list of configuration options, please refer to the ora2pg documentation: https://ora2pg.darold.net/documentation.html.

5. Extract the Oracle schema and data using ora2pg.

```
ora2pg -d my_database -i my_oracle_database
```

This will create a directory called my_database in the current directory. The directory will contain the schema and data from the Oracle database.

6. Generate SQL scripts from the extracted data.

```
ora2pg -d my_database -s my_database.sql
```

This will create a file called my_database.sql in the current directory. The file will contain SQL scripts that can be used to load the data into the PostgreSQL database.

7. **Load the SQL scripts into the PostgreSQL database.**

```
psql -d my_database -f my_database.sql
```

This will load the data from the SQL scripts into the PostgreSQL database.

8. **Port stored procedures, packages, functions, and application code to work with PostgreSQL.**

This step is optional. If your application depends on stored procedures, packages, functions, or other code that is specific to Oracle, you will need to port this code to work with PostgreSQL. If this steps is a must to, use the following walk through:

Stored procedures

Stored procedures are a type of code that is stored in a database and can be executed by a user. To port a stored procedure from Oracle to PostgreSQL, you can use the following steps:

1. Identify the stored procedure in the Oracle database.
2. Write a PostgreSQL equivalent of the stored procedure.
3. Create the stored procedure in the PostgreSQL database.

Here is an example of how to port a stored procedure from Oracle to PostgreSQL:

```
-- Oracle stored procedure
CREATE PROCEDURE my_stored_procedure (
  p_in_param int,
  p_out_param int
) AS
BEGIN
  -- Do something with the parameters.
  p_out_param := p_in_param * 2;
END;
/

-- PostgreSQL equivalent
CREATE FUNCTION my_stored_procedure (
  p_in_param int
) RETURNS int AS
$$
BEGIN
  -- Do something with the parameter.
  RETURN p_in_param * 2;
END;
```

```
$$ LANGUAGE plpgsql;
```

Packages

Packages are a collection of stored procedures, functions, and other code that are grouped together. To port a package from Oracle to PostgreSQL, you can use the following steps:

1. Identify the package in the Oracle database.
2. Write a PostgreSQL equivalent of the package.
3. Create the package in the PostgreSQL database.

Here is an example of how to port a package from Oracle to PostgreSQL:

```
-- Oracle package
CREATE PACKAGE my_package AS
PROCEDURE my_stored_procedure (
  p_in_param int,
  p_out_param int
);
FUNCTION my_function (
  p_in_param int
) RETURNS int;
END;
/

-- PostgreSQL equivalent
CREATE PACKAGE my_package AS
PROCEDURE my_stored_procedure (
  p_in_param int
);
FUNCTION my_function (
  p_in_param int
) RETURNS int;
END;
$$ LANGUAGE plpgsql;
```

Functions

Functions are a type of code that is executed when it is called. To port a function from Oracle to PostgreSQL, you can use the following steps:

1. Identify the function in the Oracle database.
2. Write a PostgreSQL equivalent of the function.
3. Create the function in the PostgreSQL database.

Here is an example of how to port a function from Oracle to PostgreSQL:

```
-- Oracle function
CREATE FUNCTION my_function (
  p_in_param int
) RETURNS int AS
```

```
BEGIN
  -- Do something with the parameter.
  RETURN p_in_param * 2;
END;
/

-- PostgreSQL equivalent
CREATE FUNCTION my_function (
  p_in_param int
) RETURNS int AS
$$
BEGIN
  -- Do something with the parameter.
  RETURN p_in_param * 2;
END;
$$ LANGUAGE plpgsql;
```

Application code

Application code is code that is used to interact with a database. To port application code from Oracle to PostgreSQL, you can use the following steps:

1. Identify the application code that interacts with the database.
2. Modify the code to use the PostgreSQL syntax.
3. Test the code to make sure that it works correctly.

Here is an example of how to port application code from Oracle to PostgreSQL:

```
-- Oracle code
DECLARE
  v_result int;
BEGIN
  v_result := my_function(10);
  dbms_output.put_line(v_result);
END;
/

-- PostgreSQL code
DECLARE
  v_result int;
BEGIN
  v_result := my_function(10);
  print(v_result);
END;
```

9. Test the migrated database thoroughly.

Once you have migrated the database, you should test it thoroughly to make sure that it is working correctly. You can use the psql command to query the PostgreSQL database and to run your application.

Here are some prerequisites you should consider before migrating:

- Ensure that your PostgreSQL version is compatible with your application.
- Ensure that your Oracle database is compatible with PostgreSQL.
- Ensure that you have enough disk space on your target server to store the migrated data.

Critical Steps:

1. **Back up your Oracle database.** This is important in case anything goes wrong during the migration process. The following script will create a backup of your Oracle database.

   pg_dump.sql

   ```
   -- This script will create a backup of your Oracle database.

   SET ECHO OFF

   REM Connect to the Oracle database
   CONNECT userid/password@database

   REM Create a directory to store the backup
   CREATE OR REPLACE DIRECTORY backup_dir AS '/path/to/backup/directory';

   REM Back up the database
   RUN {
     pg_dump -h hostname -p port -U userid -d database | gzip > backup_dir/database.sql.gz
   ```

2. **Create a PostgreSQL database that is the same schema as your Oracle database.** You can use the pg_dump command to create a backup of your Oracle database, and then use the psql command to create a PostgreSQL database from the backup.

   This script will create a PostgreSQL database from the backup of your Oracle database.

   create_postgresql_database.sql

   ```
   -- This script will create a PostgreSQL database from the backup of your Oracle database.

   SET ECHO OFF

   REM Connect to the PostgreSQL database
   CONNECT postgres

   REM Create a directory to store the PostgreSQL database
   CREATE OR REPLACE DIRECTORY backup_dir AS '/path/to/backup/directory';

   REM Create the PostgreSQL database
   RUN {
     psql -U postgres -d template1 -f backup_dir/database.sql.gz
   ```

```
}
```

3. **Run the SQL scripts to migrate your data from Oracle to PostgreSQL.** These scripts will typically convert the Oracle data into a format that is compatible with PostgreSQL.

This script will migrate the data from Oracle to PostgreSQL.

migrate_data.sql

```
-- This script will migrate the data from Oracle to PostgreSQL.

SET ECHO OFF

REM Connect to the PostgreSQL database
CONNECT postgres

REM Migrate the data
RUN {
  psql -U postgres -d database -f migrate_data.sql
}
```

4. **Test the migrated data to make sure that it is correct.** You can use the psql command to query the PostgreSQL database to make sure that the data has been migrated correctly.

```
# Connect to the PostgreSQL database
psql -d my_postgresql_database

# Query the database to make sure that the data has been migrated correctly.
SELECT * FROM my_table;
```

This script will connect to the PostgreSQL database and query the my_table table. If the data has been migrated correctly, the query will return the data that was in the my_table table in the Oracle database.

Here is an example of the output of the script:

```
id | name | email
-- | -- | --
1 | John Doe | john.doe@example.com
2 | Jane Doe | jane.doe@example.com
```

If the output of the script is different from the data that was in the my_table table in the Oracle database, then the data has not been migrated correctly. In this case, you will need to troubleshoot the migration process to find and fix the problem.

5. **Make any necessary changes to the PostgreSQL database.** For example, you may need to change the data types of some columns to match the data types in the Oracle database.The table below illustrates Oracle to PostgreSQL safe type mapping:

Oracle	PostgreSQL
BINARY_FLOAT	REAL
BINARY_INTEGER	INTEGER
BINARY_DOUBLE	DOUBLE PRECISION
BLOB, RAW(n), LONG RAW	BYTEA (1GB limit)
CLOB, LONG	TEXT (1GB limit)
DATE	TIMESTAMP
NUMBER, NUMBER(*)	DOUBLE PRECISION or BIGINT if it is a part of Primary Key
NUMBER(n,0), NUMBER(n)	n<5 – SMALLINT5<=n<9 – INT9<=n<19 – BIGINTn>=19 – DECIMAL(n)
NUMBER(p,s)	DECIMAL(p,s)
REAL	DOUBLE PRECISION

For numeric types it is important to understand the scope of use in database. If it is focused on accuracy, Oracle numeric types must be mapped in PostgreSQL NUMERIC. If the top priority is calculation speed, the best mapping would be REAL or DOUBLE PRECISION.

We used Oracle-to-PostgreSQL converter to automate migration of table definitions, indexes and constrains for this project. It maps Oracle types into the most appropriate PostgreSQL equivalents and allows to customize particular type mapping.

6. **Promote the PostgreSQL database to production.** Once you are satisfied that the migration has been successful, you can promote the PostgreSQL database to production.Here is a script to promote the PostgreSQL database to production:

```
# Connect to the PostgreSQL database
psql -d my_postgresql_database

# Promote the database to production
ALTER DATABASE my_postgresql_database SET LOGICAL_REPLICATION_MODE='REPLICA';
```

This script will connect to the PostgreSQL database and promote the database to production. The ALTER DATABASE command will set the LOGICAL_REPLICATION_MODE to REPLICA. This means that the PostgreSQL database will no longer be a master database, but will instead be a replica database.

Once the database has been promoted to production, you can start using it to serve your application.

Here are some additional things to consider when promoting the PostgreSQL database to production:

- Make sure that you have a backup of the PostgreSQL database.
- Make sure that you have tested the PostgreSQL database in a staging environment.
- Make sure that you have communicated with your users about the upcoming change.

Note: These are just examples, and you may need to modify them to fit your specific needs. For example, you may need to change the hostname, port, userid, and database variables to match your own environment.

Rollback procedure:

If anything goes wrong during the migration process, you can roll back the migration by restoring the backup of your Oracle database.

Here are some rollback procedures you can follow:

- Restore a backup of your original Oracle database.
- Revert to a previous version of your application.

Here is a detailed procedure to restore a backup of original Oracle database:

1. **Identify the backup file.** The backup file is typically a file with the .dmp extension. It should be stored in a safe location, such as a network drive or an external hard drive.
2. **Connect to the Oracle database.** You can use the sqlplus command to connect to the Oracle database.
3. **Restore the backup file.** Use the restore command to restore the backup file. For example, the following command would restore the backup file called my_backup.dmp to the database called my_database:

```
restore database my_database from my_backup.dmp;
```

4. **Verify the restore.** Once the restore is complete, you should verify that the database has been restored correctly. You can do this by running some basic queries against the database.

Here are some additional tips for restoring a backup of an Oracle database:

- Make sure that you have the correct backup file. If you have multiple backup files, make sure that you restore the correct one.

- Make sure that you have enough disk space to restore the backup file. The backup file can be quite large, so you need to make sure that you have enough disk space to store it.
- Make sure that you are connected to the correct database. If you are not connected to the correct database, the restore will fail.

Use the following procedure to revert to a previous version of an application, including scripts and commands:

1. **Identify the previous version of the application.** The previous version of the application is typically stored in a backup directory. It should be stored in a safe location, such as a network drive or an external hard drive.
2. **Stop the application.** If the application is running, you need to stop it before you can revert to a previous version.

```
# Stop the application
sudo service my_application stop
```

3. **Copy the previous version of the application to the production directory.** The production directory is the directory where the application is currently running. You can use the `cp` command to copy the previous version of the application to the production directory.

```
# Copy the previous version of the application to the production directory
cp -r /path/to/previous/version /path/to/production
```

4. **Start the application.** Once the previous version of the application has been copied to the production directory, you can start the application.

```
# Start the application
sudo service my_application start
```

Here are some additional tips for reverting to a previous version of an application:

- Make sure that you have the correct backup of the application. If you have multiple backups, make sure that you restore the correct one.
- Make sure that you have enough disk space to copy the previous version of the application. The previous version of the application can be quite large, so you need to make sure that you have enough disk space to store it.
- Make sure that you stop the application before you copy the previous version. If you do not stop the application, the copy will fail.

References

1. How to Migrate from Oracle to PostgreSQL
2. Porting from Oracle PL/SQL
3. Documentation of pgTAP
4. Github Repository of oracle_fdw
5. Github Repository of plpgsql_check
6. Github Repository of plprofiler
7. Github Repository of orafce