

Java/Oracle Application Migration Guide to Azure Database for PostgreSQL

The information herein is for informational purposes only and represents the current view of Microsoft Corporation as of the date of this publication. Because Microsoft must respond to changing market conditions, it should not be interpreted to be a commitment on the part of Microsoft, and Microsoft cannot guarantee the accuracy of any information provided after the date of this presentation. MICROSOFT MAKES NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, AS TO THE INFORMATION IN THIS PRESENTATION.

© 2020 Microsoft. All rights reserved. This document is for informational purposes only. Microsoft makes no warranties, express or implied, with respect to the information presented here

Contents

[About the guide 6](#_Toc36617824)

[Scenario overview 7](#_Toc36617825)

[Legacy on-premises application architecture 8](#_Toc36617826)

[Oracle Database ER Diagram 10](#_Toc36617827)

[Get the document artifacts from Git repo 11](#_Toc36617828)

[Tour of the application 13](#_Toc36617829)

[Application target goal 14](#_Toc36617830)

[Database migration process 16](#_Toc36617831)

[Migrations types 17](#_Toc36617832)

[PostgreSQL Introduction 18](#_Toc36617833)

[PostgreSQL database options 19](#_Toc36617834)

[Database migration tool options 19](#_Toc36617835)

[Setting up your migration server 21](#_Toc36617836)

[Choosing your migration server 21](#_Toc36617837)

[Hardware resources 22](#_Toc36617838)

[Securing the data during migration 22](#_Toc36617839)

[Download and install the Oracle database client library 23](#_Toc36617840)

[Set up the environment variables 23](#_Toc36617841)

[Set up the pgAdmin PostgreSQL database client 23](#_Toc36617842)

[Download and install the ora2pg utility 24](#_Toc36617843)

[Discovering and assessing the source database with ora2pg 25](#_Toc36617844)

[Prepping your database for export 25](#_Toc36617845)

[Data and column refactoring 26](#_Toc36617846)

[Create your ora2pg conf structure 28](#_Toc36617847)

[Add your Oracle and PostgreSQL DSN configuration to the conf file 29](#_Toc36617848)

[Testing your database connections and permissions 31](#_Toc36617849)

[Separating the constraints and indexes into files 32](#_Toc36617850)

[Evaluate the data type conversions 34](#_Toc36617851)

[Layering on the sequences and triggers 34](#_Toc36617852)

[Convert the procedures 35](#_Toc36617853)

[Default ora2pg code conversion 39](#_Toc36617854)

[Object conversion errors that do not produce runtime errors 43](#_Toc36617855)

[Objects that cannot be converted automatically 45](#_Toc36617856)

[Other useful ora2pg configurations 46](#_Toc36617857)

[PostgreSQL workarounds for Oracle objects and features 46](#_Toc36617858)

[Assessing database complexity and time to import 46](#_Toc36617859)

[Comparing the Oracle and PostgreSQL instance schema 49](#_Toc36617860)

[Migration 50](#_Toc36617861)

[Sample application modifications 50](#_Toc36617862)

[Copying the data over to Azure PostgreSQL 50](#_Toc36617863)

[Cutting over 52](#_Toc36617864)

[Post-migration 53](#_Toc36617865)

[Should you convert Stored Procedure and Functions to application code? 53](#_Toc36617866)

[Architecture strategies 54](#_Toc36617867)

[Have questions? 54](#_Toc36617868)

## About the guide

Azure Database for PostgreSQL is a managed service that you use to run, manage, and scale highly available PostgreSQL databases in the cloud. This guide walks through the steps of migrating an on-premises legacy open source application using an Oracle database to an application using a cloud hosted Azure PostgreSQL database.

Every application is different and has its own level of complexity. Very complex applications and databases require extra time and resources to analyze, assess the level of effort, and implement a solution. This guide focuses on simple to moderately complex web applications that need to switch the database to PostgreSQL.

The application was architected to be simple, but realistic. Building a full application would take away from the process of understanding a database migration. The reader should appreciate the few changes required to retool the application in order to take advantage of the Azure Database for PostgreSQL .

Applications that take advantage of ORM tools like Hibernate and Spring are great candidates for easy migrations. They abstract the mapping of data and types. The sample Oracle database fields, objects, and data chosen represent common objects found in community projects. More complex Oracle types would put this application into the advanced migration scenario.

Some of the database schema objects are contrived and meant to exercise the process of migration, not advocate best application architecture guidelines. The reader should have some basic familiarity with the development tools referenced and their usage. Step by step debugging instructions will not be provided. At the conclusion of this document, the reader will be able to understand clearly the steps required to host the migrated database in Azure and the small Java application adjustments necessary to connect to PostgreSQL.

It is recommended the reader build up the knowledge of a Oracle to PostgreSQL database migration using a small database. Jumping into a large migration first will most likely lead to project failure or major delays by experiencing common errors first hand.

## Scenario overview

Contoso Tech specializes in providing leading edge training and technical evangelism. They host multiple conferences around the world with packed audiences. The website used for promoting the conferences, marketing session tracks, and registering attendees has been in production for several years.

Contoso has been modernizing their infrastructure and the operations team would like to move this application from on-premises to the cloud as it is one of the last applications in their shrinking datacenter. The operations team recently had to move the application and database to newer hardware. During the migration, it was discovered the database configuration was hard-coded in the application configuration. The application failed to start after migration and developers were called in to resolve the problem in the early morning hours. Since then, the development team has made great improvements to the web and API stack preparing for the move to the cloud. However, there is a reluctance to modernize or change the database feeding the web site due to the amount objects layered on over the years. A lot has changed in the database marketplace since this application was originally developed and there might be new database options to consider.

Management has heard about the benefits of the PostgreSQL database. Below are some of the benefits:

* High performance and scalability.
* Popular programming languages have a PostgreSQL provider.
* It can store many of the data types required by applications.
* It has an open-source licensing model.

Steps required to migrate to this database platform need to be explored by the development team.

The development team is motivated, but has communicated firmly migration is not as easy as changing a connection string in the application configuration. PostgreSQL has very similar capabilities to the existing Oracle database, but special considerations would be required for database objects that did not convert transparently. Downtime, functionality degradation, and loss of data are the biggest risks for this project. The development team has been given the task of documenting the migration and risk mitigation plan.

The web application consists of an Angular frontend, Java Spring Maven web API, and an Oracle database.

This guide will cover the migration path for hosting an existing Java Spring Oracle application in Azure utilizing App Services and managed PostgreSQL PaaS service. The topics of application scaling and containerization are important, but will not be covered as the focus of this guide is the challenges of database migration.

## Legacy on-premises application architecture

The sample legacy application is a simple application. Since the application scenario is hosted on-premises, the developer could run this Angular Java application locally.



Microsoft provides full support for Java applications. On the Microsoft Azure and Azure Stack, you can build and run production Java applications by using [Azul Zulu Enterprise for Azure](https://www.azul.com/downloads/azure-only/zulu/) without incurring additional support costs. You can use any Java runtime you want on Azure, but when you use Zulu, you get free maintenance updates and you can resolve support issues with Microsoft.

Azul Zulu Enterprise builds of OpenJDK are a no-cost, multi-platform, production-ready distribution of the OpenJDK for Azure and Azure Stack that's backed by Microsoft and Azul Systems.

As always, as a Java developer, you can bring to Azure your own Java runtimes, including the Oracle JDK and the Red Hat JDK. You can also use the secure infrastructure and feature-rich services. The production edition of Oracle Java SE is available to you for running Java workloads in Windows or Linux virtual machines on Azure.

**Additional resources**

[Java long-term support for Azure and Azure Stack](https://docs.microsoft.com/en-us/java/azure/jdk/?view=azure-java-stable)

[Azul Product Support Lifecycle](https://www.azul.com/products/azul-support-roadmap/)

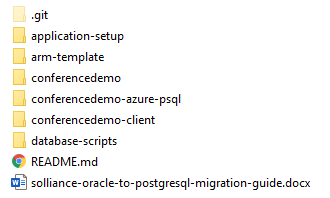
## Oracle Database ER Diagram

Below is the ER diagram. The design is simple but the most common found fields are incorporated into the design. As you review the document and the examples, this will provide some reference as to the database structure.



## Get the document artifacts from Git repo

* Run git clone to download the application locally. You should have the structure below.



We are going to focus on setting up the Angular and Java application with a Oracle backend to provide the reader with an understanding of the legacy application before the PostgreSQL migration tasks start.

* **application-setup –** step by step instructions on how to set up the sample application.
* **arm-template -** template file to set up the entire Azure migration environment.
* **conferencedemo** - folder contains Java API application.
* **conferencedemo-client -** contains the Angular application
* **conferencedemo-azure-psql** - contains the same Java API application with minor changes to connect to PostgreSQL.
* **database-scripts** - contains the Oracle scripts to set up the database objects and sample data.

Once they have a contextual understanding of the application, we will move on to the process of assessment, migration, and conversion.

\*\* If you are utilizing a server other than a local copy, the migration user account will need elevated permissions to the source database in order to properly capture the schema and data information for PostgreSQL export. If you are using a local Oracle copy, grant all the rights.



It will help demonstrate your typical challenges with assessing the entire database.

## Tour of the application

Landing page



After selecting an event, the session list is presented. Attendees can register for the sessions on this page.



Speaker bio details

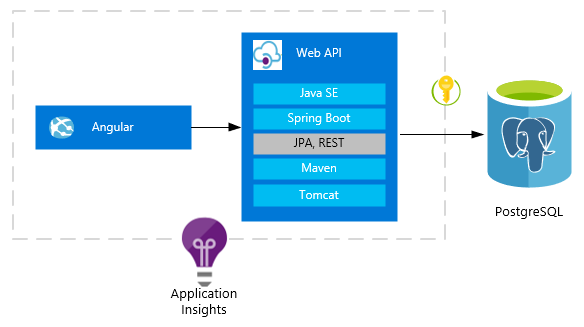


## Application target goal

Weigh the existing investment in your on-premises data center and consider the costs of moving to the cloud to determine whether it’s right for your organization. For many smaller or younger companies, migrating to the cloud can be a no-brainer. With cloud computing, you can quickly gain access to enterprise-class resources on a pay-as-you-go basis, resources that may otherwise be cost-prohibitive.

Moving this application to the cloud provides several advantages. You can scale the environment resources up or down for your applications.

Secure key management is essential to protect data in the cloud. Azure Key Vault will encrypt keys and small secrets like passwords that use keys stored in hardware security modules (HSMs). Administrators can update application configuration variables and maintain a secure environment without involving the development team.



Application Insights, a feature of [Azure Monitor](https://docs.microsoft.com/en-us/azure/azure-monitor/overview), is an extensible Application Performance Management (APM) service for developers and DevOps professionals. Use it to monitor your live applications. It will automatically detect performance anomalies, and includes powerful analytics tools to help you diagnose issues and to understand what users actually do with your app. It's designed to help you continuously improve performance and usability. It works for apps on a wide variety of platforms including .NET, Node.js and Java EE, hosted on-premises, hybrid, or any public cloud. It integrates with your DevOps process, and has connection points to a variety of development tools. It can monitor and analyze telemetry from mobile apps by integrating with Visual Studio App Center.

**Additional resources**

[Azure Key Vault](https://azure.microsoft.com/en-us/services/key-vault/)

[What is Application Insights?](https://docs.microsoft.com/en-us/azure/azure-monitor/app/app-insights-overview)

[Azure Monitor](https://azure.microsoft.com/en-us/services/monitor/)

## Database migration process

At this point, you should have the sample application running and be able to identify with the familiar architecture. We will be referencing parts of the project as we progress through the migration process.

The entire migration process can be broken down into these discrete phases.



**Discover**: Customers discover all the databases they own, instances hosting them, source database type and version, other database metadata, etc.

**Assess**: In this phase, the goal is to identify which Azure database target a customer can migrate their database to and how much work is involved.

* Target and SKU recommendation – Identifies which Azure database target and SKU is right for a database.
* Readiness assessment – Compares the features used on the source vs the target and give an overview of work involved in migrating a database to Azure.
* Performance assessment – Compares the query correctness and performance of a database on premise and in Azure.

**Convert**: Convert the schema from the source database type to target database type. This phase is valid only for heterogenous migrations.

**Migrate**: Migrating a database involves migrating schema, data and objects usually in that order. There are 2 types of database migration methods:

* Offline migrations – In this method, customers plan a downtime for the database, mark it read-only, perform the migration and switch over their applications to the new database. Prepared by Data Migration Jumpstart Engineering
* Online migrations – Also known as minimal downtime, this method requires a short or no downtime for migrations. This method involves doing a front load of data to the target, enabling data sync so that data continues to be replicated from source to target until customer is ready to cut-over.

**Validate**: After the migration is complete, it is important to ensure that target database has the same schema, data and objects as source database. Also, it is important to validate if applications using the database behaves the same way on both source and target databases.

## Migrations types

Each application migration needs to be evaluated and estimated based on its own merits. The types of projects fall into these categories generally.

* Little to no code changes. Migration works without issues.
* Some effort and code changes. Some schema objects require review and adjustments.
* Difficult and time consuming. The migration team is most likely rearchitecting and rebuilding the application.

Keep in mind, you will always need to review and convert the source database programming objects to the target PostgreSQL environment. The process for exporting objects and the common problems will be discussed later in the document.

<TODO: Benefits of moving to the cloud>

The process of moving an application to Azure should follow this maturity process.



**Additional resources**

[Application Modernization on Azure](https://medius.studios.ms/Embed/Video/BRK2102?sid=BRK2102)

## PostgreSQL Introduction

Azure Database for PostgreSQL is a relational database service based on the open-source Postgres database engine. It's a fully managed database-as-a-service offering that can handle mission-critical workloads with predictable performance, security, high availability, and dynamic scalability. It's available in two deployment options, as a single server and as a Hyperscale (Citus) cluster.

Single server is best for workloads that can perform well with the compute, memory, and storage of a single node. Hyperscale (Citus) is best for applications that have demanding performance & concurrency requirements and need to scale out Postgres horizontally. With Hyperscale (Citus) you can take advantage of the aggregate compute, memory, and storage of a multi-node database cluster. ​The Hyperscale (Citus) option horizontally scales queries across multiple machines using sharding, and serves applications that require greater scale and performance.

Azure Database for PostgreSQL Hyperscale is now Azure Arc-enabled. You can run this service on premises on infrastructure of your choice with cloud benefits like automation, hyperscale, unified management, and a cloud billing model with reserved capacity pricing now available.

For details on pricing: [Azure Database for PostgreSQL pricing](https://azure.microsoft.com/en-us/pricing/details/postgresql/server/)

**Additional resources**

[Oracle to Postgres Conversion](https://wiki.postgresql.org/wiki/Oracle_to_Postgres_Conversion)

## Measure performance and plan for optimization tasks

Successfully migrating the data and the schema objects to the new database platform is not the only measurement of project success. If the application appears to perform slower than before migration, users will complain and there will be a question about the migration choice. Your goal should be better performance after migration completion. Get agreement on the phase one’s acceptable performance. Performance tuning and workload optimization are required and should be factored into the project planning.

How do we achieve acceptable database performance? One of the first steps is to measure your current environment performance in a way it can be repeated in the new environment. What type of load is the current system under? Can you measure it? Examples of measurements to be taken:

* Queries completed per second
* Average time taken for completion

When it is time to test your PostgreSQL environment you will want to determine if your converted schema objects scale. Generate query workloads against the target environment and capture the performance metrics. Documenting and comparing your measurements provides an objective discussion point for further evaluation.

One tool that can assist in database load generation is [PostgreSQL pgbench](https://www.postgresql.org/docs/devel/pgbench.html). pgbench has the capability to run batches of queries repeatedly and capture the associated performance metrics. Make sure your tests are significant and take several minutes or hours to complete in order to get reproduceable numbers. Testing that completes in seconds most likely contains inaccurate numbers and should not be relied on.

Your testing results may lead to the need for new environment hardware configuration and sizing. Also, the new environment should have enough resources allocated to handle the users’ projected usage and data storage needs. For more information on performance optimization, see the additional resources.

**Additional resources**

[PostgreSQL pgbench](https://www.postgresql.org/docs/devel/pgbench.html)

[PostgreSQL reclaiming space from delete rows with VACUUM](https://www.postgresql.org/docs/9.5/sql-vacuum.html)

<https://wiki.postgresql.org/wiki/Performance_Optimization>

<https://www.postgresql.org/docs/current/performance-tips.html>

[Using PostgreSQL EXPLAIN](https://wiki.postgresql.org/wiki/Using_EXPLAIN)

[Oracle Database Tuning Overview](https://docs.oracle.com/database/121/TGDBA/pfgrf_perf_overview.htm#TGDBA025)

[Gathering Oracle Database Statistics](https://docs.oracle.com/database/121/TGDBA/gather_stats.htm#TGDBA167)

## Database migration tool options

There are few options for migrating an Oracle database to Azure Database for PostgreSQL. Your choice will depend on your timeline, budget, and database complexity.

**Azure Database Migration Services**

For online migrations, with [Azure Data Migration Services](https://docs.microsoft.com/en-us/azure/dms/tutorial-oracle-azure-postgresql-online) you can migrate your oracle databases hosted on-premises or on a virtual machine to Azure Database for PostgreSQL. It enables resilient migrations of Oracle databases at scale and with high reliability. Provision an instance of Database Migration Service from the Azure portal or via Azure PowerShell and create a migration project to perform the migration.

For an optimal migration experience, Microsoft recommends creating an instance of Azure Database Migration Service in the same Azure region as the target database. Moving data across regions or geographies can slow down the migration process and introduce errors.

When you migrate databases to Azure by using Azure Database Migration Service, you can do an offline or an online migration. With an offline migration, application downtime starts when the migration starts. With an online migration, downtime is limited to the time to cut over at the end of migration. We suggest that you test an offline migration to determine whether the downtime is acceptable; if not, do an online migration.

Azure Database Migration Service creates only the table schema, not other database objects such as stored procedures, packages, and indexes.

**Other commercial utilities**

Other commercial data transfer utilities exist in the marketplace. For example, [Attunity Replicate for Microsoft Migrations](https://www.attunity.com/products/replicate/attunity-replicate-for-microsoft-migration/) facilitates and simplifies migrations from a broad range of commercial and open-source databases, including Oracle.

**ora2pg utility**

A popular method of database migration is using the open source utility, ora2pg. The utility automates many of the migration tasks required to migrate schema and data into PostgreSQL. Utility tasks and configuration details are discussed later in [Discovering and assessing the source database with ora2pg](#_Discovering_and_assessing).

This document will focus on ora2pg utility v20 and its feature set.

**Additional resources**

<https://docs.microsoft.com/en-us/azure/dms/dms-tools-matrix>

## Setting up your migration server

This next section provides information related to setting up a server for database migration and the choices to consider.

### Choosing your migration server

#### Run locally or use a migration server?

You could run the ora2pg migration utility on your local development machine or the database server. The Oracle administrator will not appreciate the extra software installed on the server as well as the resources consumed during migration. Also, you would need to repeat this installation and configuration for each server. This approach is not recommended. Usually, a migration effort requires multiple team members. Running the migration locally on your machine would require you to create a setup document for the rest of the team in order for you to run the process in a similar fashion. This might not be efficient as the project configuration and processing would need to be kept in sync. Also, running locally will consume significant resources causing the hardware to be tied to the migration process until completion. An alternative to running locally, would be to use a dedicated migration server(s). Multiple team members can check a central standardized migration server for progress and exception handling. It can be secured using best practices.



### Hardware resources

Your migration server should be reasonably configured with enough processing power and memory to handle the load. Memory is the key issue with migrating large amounts of data, especially records containing blobs. You may have to reduce your data limit (rows processed per batch) significantly if you do not provide enough migration server resources. Receiving an out of memory error could cause unwanted project delays. The cost of delays may exceed the cost of utilizing the proper Azure SKU.

The migration server needs access to the Oracle and the Azure PostgreSQL instances. Depending on your project timelines, you may need to increase your throughput between the source environment and the Azure PostgreSQL network.

### Securing the data during migration

On-premises migration server

Encrypting your data during migration is critical. This can be done utilizing a few methods:

* Database provider connection
* VPN gateway
* ExpressRoute

Azure hosted VM

After provisioning the VM and Azure Database for PostgreSQL, two configurations are needed for enabling connectivity between them: “Allow Azure Services” and “Enforce SSL Connection”, depicted as follows:

* “Connection Security” blade -> Allow access to Azure Services -> ON
* “Connection Security” blade -> SSL Settings -> Enforce SSL Connection -> DISABLED

### Download and install the Oracle database client library

If you are running the Oracle XE database locally, then you can skip the install of the database client.

### Set up the environment variables

#### Windows server

ORACLE\_HOME = <Your client or server install path>

LD\_LIBRARY\_PATH = %ORACLE%\lib

\*\* If you installed the database client only, then set your LD\_LIBRARY\_PATH path to <client install path>.

Example



### Set up the pgAdmin PostgreSQL database client

You will need to install the pgAdmin admin client. You can download the utility from <https://www.pgadmin.org/download/> . Connect to the Azure database server with the server information captured earlier in the Application Setup Guide.

Create the database **conferencedemo** database.



Create a user **reg\_app**. Assign the **conferenceadmin** role.



### Download and install the ora2pg utility

The configuration of the ora2pg environment can take up to a few hours. For quick testing in a Docker environment, consider using this image:

docker pull georgmoser/ora2pg-docker

This will provide an environment which will allow you to understand the basics of the utility without having to spend a lot of time installing and configuring.

For a full migration server, please review the [Step-by-Step Guide to Install ora2pg on Linux & Windows](https://github.com/microsoft/DataMigrationTeam/blob/master/Whitepapers/Steps%20to%20Install%20ora2pg%20on%20Windows%20and%20Linux.pdf).

**Additional resources**

[Migrate Oracle to Azure Database for PostgreSQL](https://datamigration.microsoft.com/scenario/oracle-to-azurepostgresql?step=1)

[What is VPN Gateway?](https://docs.microsoft.com/en-us/azure/vpn-gateway/vpn-gateway-about-vpngateways)

[Virtual machine network bandwidth](https://docs.microsoft.com/en-us/azure/virtual-network/virtual-machine-network-throughput)

[Optimize network throughput for Azure virtual machines](https://docs.microsoft.com/bs-cyrl-ba/azure/virtual-network/virtual-network-optimize-network-bandwidth)

[ExpressRoute overview](https://docs.microsoft.com/en-au/azure/expressroute/expressroute-introduction)

## Discovering and assessing the source database with ora2pg



### Prepping your database for export

Before running the ora2pg utility against your source database, the database statistics will need to be updated. Statistics can become stale over time because of changing data volumes or changes in column values. Statistics can be inaccurate after lots of data and schema changes.

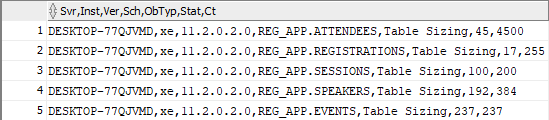


### Checking for invalid Oracle objects

Also, you need to check for invalid objects. The Data Migration Team at Microsoft wrote an Oracle procedure that queries the database and shows the count of objects and their validity. Invalid objects will not be converted and exported by the ora2pg utility by default. Setting the EXPORT\_INVALID = 1 in conf file configuration will export invalid objects. It is recommended to fix any errors and compile the objects before starting the migration process. Otherwise, when it comes time for application testing, you may have unexpected results. Also, an invalid object may not be actually used anymore and maybe a candidate for deprecation.

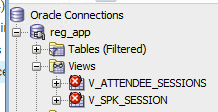
This is Oracle query will produce an inventory of database objects.





This query is located at: [Oracle\_PreSSMA\_v12\_Plus.sql](https://github.com/microsoft/DataMigrationTeam/blob/master/Oracle%20Inventory%20Script%20Artifacts/Oracle%20Inventory%20Script%20Artifacts/Oracle_PreSSMA_v12_Plus.sql)

Example of invalid objects



### Data and column refactoring

Before blindly exporting the schema and data, meet with the migration team to determine if large tables with many columns can be trimmed or refactored. Is all the data required? Maybe the last seven years of data is all that is required. Orphaned columns do not provide value and may not be needed in the target database. Making this decision to drop unneeded columns or data may increase the application testing time but will reduce the migration time. Do you have legal requirements to hold this data? Depending on the data, you will want to get sign off from your management and possibly the records management team before implementing changes.

### Create your ora2pg conf structure

For small projects, running the ora2pg utility with the defaults will allow you to export all objects in one giant script. For larger more realistic projects, you will be running the ora2pg utility several times. It is important to separate your scripts into easily maintainable directories. You will want to import database objects and data into PostgreSQL in stages. It will be a rare project where you can accept all the defaults and run the data migration process.

Using Git, the team can track changes to the schema and make development decisions based on those changes. Separating your scripts into directories makes it easier to review the changes.

To create your base project directory structure using the ora2pg tool:

1. Navigate to a base directory.
2. Run this command

ora2pg --init\_project reg\_app



Now that your base structure has been created, place your **ora2pg conf** file into the config folder.



### Add your Oracle and PostgreSQL DSN configuration to the conf file

Configure Oracle: ORACLE\_HOME, ORACLE\_DSN, ORACLE\_USER, and ORACLE\_PWD information.

Example of DSN settings



If you are having trouble finding your DSN settings, try running ‘lsnrctl status’ command in the console.

You can confirm:

1. ORACLE\_HOME value
2. Oracle SID
3. Oracle host and port



Configure the Azure Database for PostgreSQL:

PG\_DSN, PG\_USER, PG\_PWD



### Testing your database connections and permissions

#### Setting your schema

Run the following command **if you created a local Oracle XE test server**. If you are using a database on a shared server with several other databases, skip ahead.



434 tables?! What happened? Why are the apex tables in the output?



You need to specify the schema/namespace.



If you are going to work with one schema, you can set it in the conf file. Remember, the command line parameters override the conf file parameters at runtime.



This is the output we were expecting, but we still have a problem. All of the constraints and indexes are in the table creation scripts.



If you have a lot of data, the indexes and constraints should be applied once the tables have been created and the data has imported. You will have much better data import performance.

### Separating the constraints and indexes into files

Update your conf file with the following configurations.





This type of schema output allows the database tables to be created first. Data could be imported and then the indexes and constraints can be applied at a later time.



All the tables schemas can be found in the ‘reg\_app-psql.sql’ file. Notice the indexes and constraints are missing.



A database with many tables or tables with many fields may require a different migration strategy. You could create a script for each table. This will allow team members, like the database administrator, to evaluate the conversions before migration.

<TODO: Talk about the import and export .sh script>





### Evaluate the data type conversions

Most of the ora2pg type conversions suggestions make sense. There are times the migration team will need to adjust the schema data types based on intimate knowledge of the data. Evaluating each table and its dependent objects makes sense. Do I need a bigint or is int good enough?



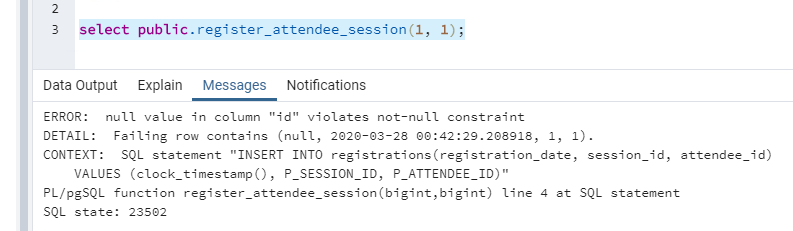
### Layering on the sequences and triggers

Loading existing data does not require the sequences or triggers to be applied. However, if you plan on performing a smoke test by adding **new** data, you may want to apply sequences and then the triggers.

Sample export calls:

1. ora2pg -c config/ora2pg.conf --namespace reg\_app --type SEQUENCE
2. ora2pg -c config/ora2pg.conf --namespace reg\_app --type TRIGGER

Failing to apply the sequences and triggers will result in PostgreSQL error messages like this:

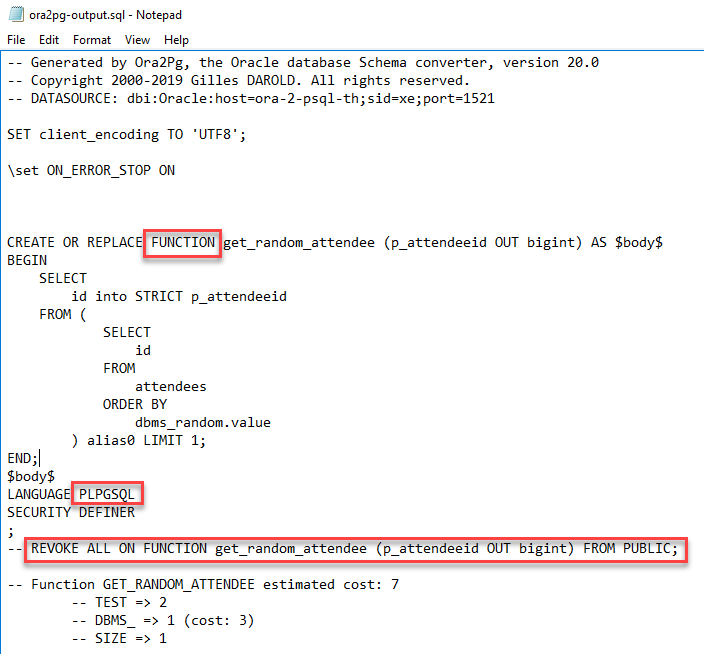


### Convert the procedures

Use the ora2pg utility as a basic guide for procedure and function conversion. Users will need to review and update the exported code to work in their environment. Once dependencies like table, sequences, and view objects are transferred, it is time to try exporting the procedures. As mentioned previously, it is important to remember to check for Oracle invalid objects first. Those objects will need to be fixed and compiled for correct export using the default configuration.



What did we get? The original Oracle object was a procedure, but I got a function.

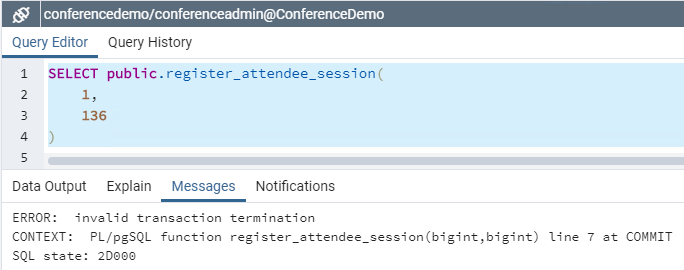


This is a contrived example. You could write the SQL to not use nesting, etc. Part of the goal for this example is to show how ora2pg will convert nested queries.

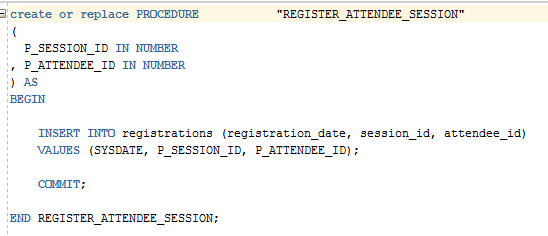
PostgreSQL 11 introduced procedures. The ora2pg v20 utility exports to functions and this may work depending on the complexity of the original stored procedure. Something else to keep in mind. The PUBLIC role has default access to newly created functions. If you have a sensitive function, you should consider uncommenting the REVOKE call before granting access to other roles to ensure the PUBLIC role does not have unintended access.

<TODO: Differences between a function and a procedure in PostgreSQL 11>

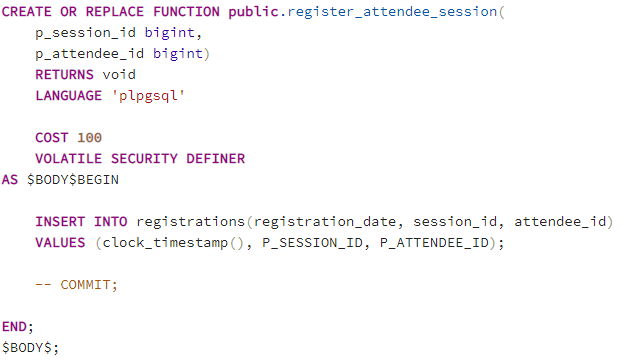
The ora2pg script converted the procedures to functions. Are functions and procedures in PostgreSQL 11 the same? No! Why did we get this error message?



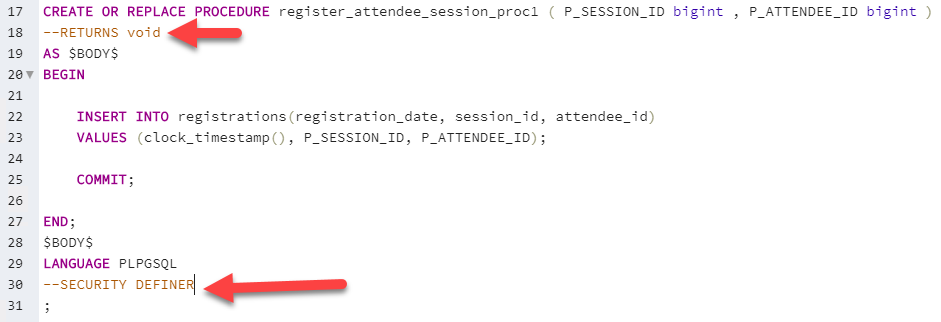
Below is the original Oracle procedure. Hibernate could have handled this transaction in the application. However, the focus of this exercise is to determine the conversion options we have available. Let’s assume this procedure had several lines of business logic ending with a COMMIT or ROLLBACK and the team decided to keep the object in the database layer.



It is important to remember that some procedures will need to be fixed manually before they will compile. One of the main differences between procedures and functions is PostgreSQL 11 **functions** do not handle programmatic transactions. Commenting out the COMMIT allows the function to proceed as intended. Simple procedures could use this type of solution. Executing the call using a SELECT command performs the desired action.



This solution may not work for your situation. If your team decides the logic needs to remain the same, converting the function to a procedure is easy in this example. The commented code demonstrates the changes required. Remove unnecessary comments for production.



*“A SECURITY DEFINER procedure cannot execute transaction control statements (for example, COMMIT and ROLLBACK, depending on the language).”*

Commenting or removing code requires you to research the consequences and should be done carefully. See the [PostgreSQL documentation](https://www.postgresql.org/docs/11/) for more information.

**Additional resources**

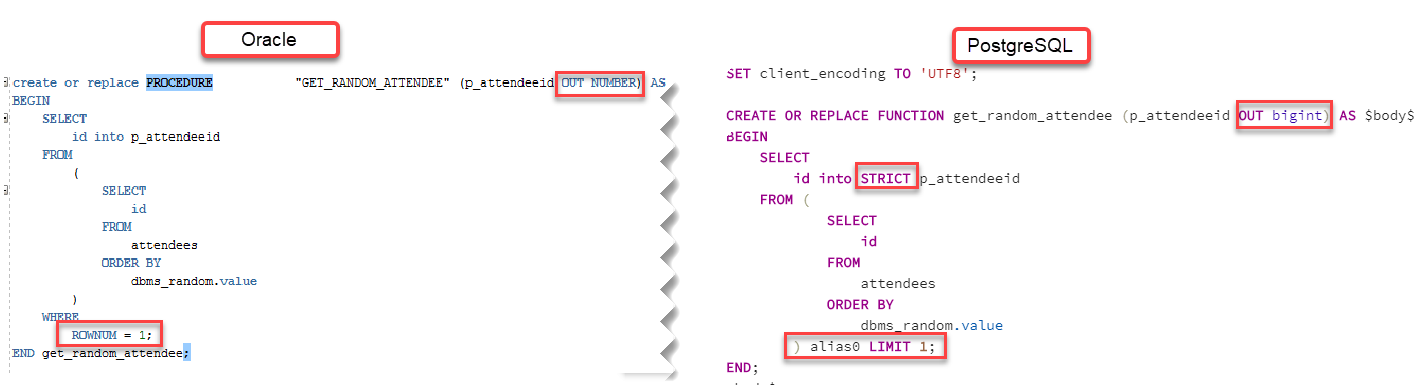
<https://www.postgresql.org/docs/11/>

<https://www.postgresql.org/docs/current/sql-createprocedure.html>

<https://www.postgresql.org/docs/current/sql-createfunction.html>

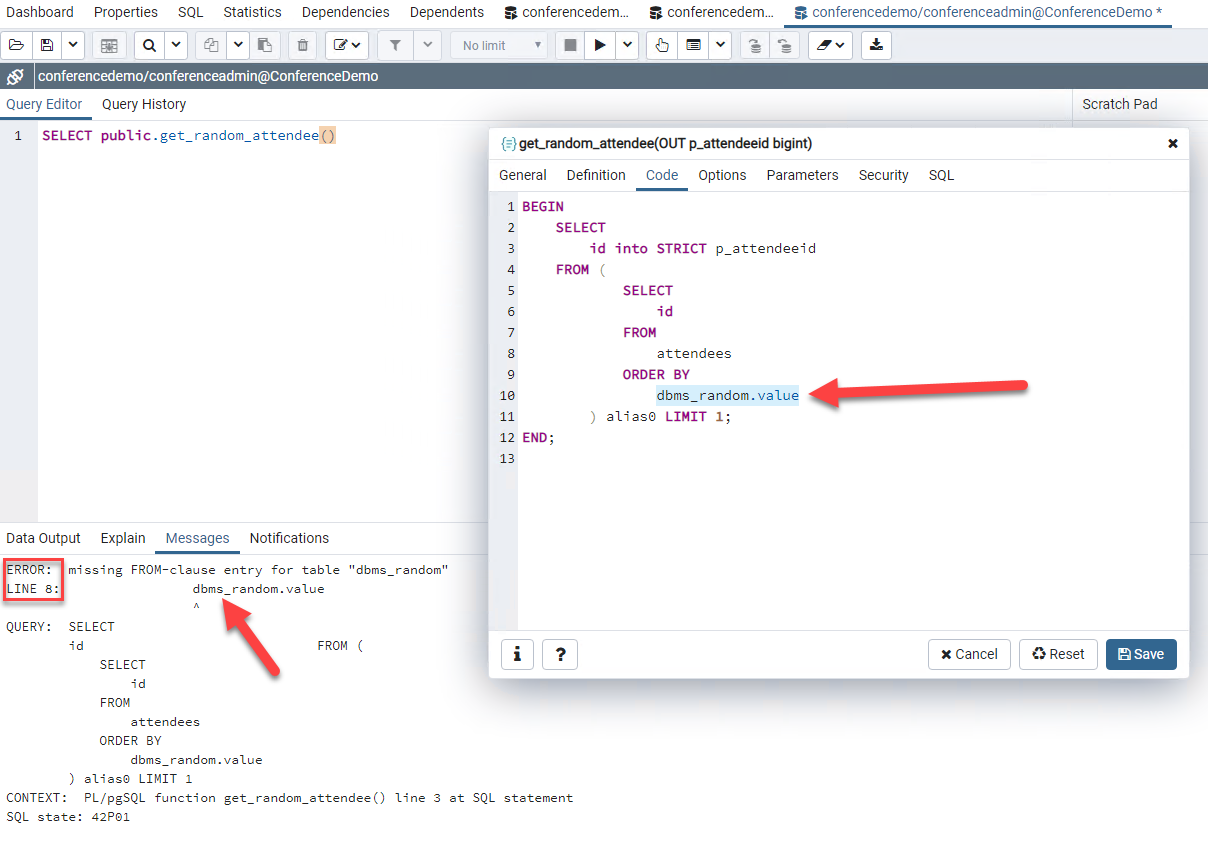
### Default ora2pg code conversion

ora2pg does a good job of converting Oracle PL/SQL syntax to PostgreSQL PL/pgSQL. Most of the syntax looks similar.

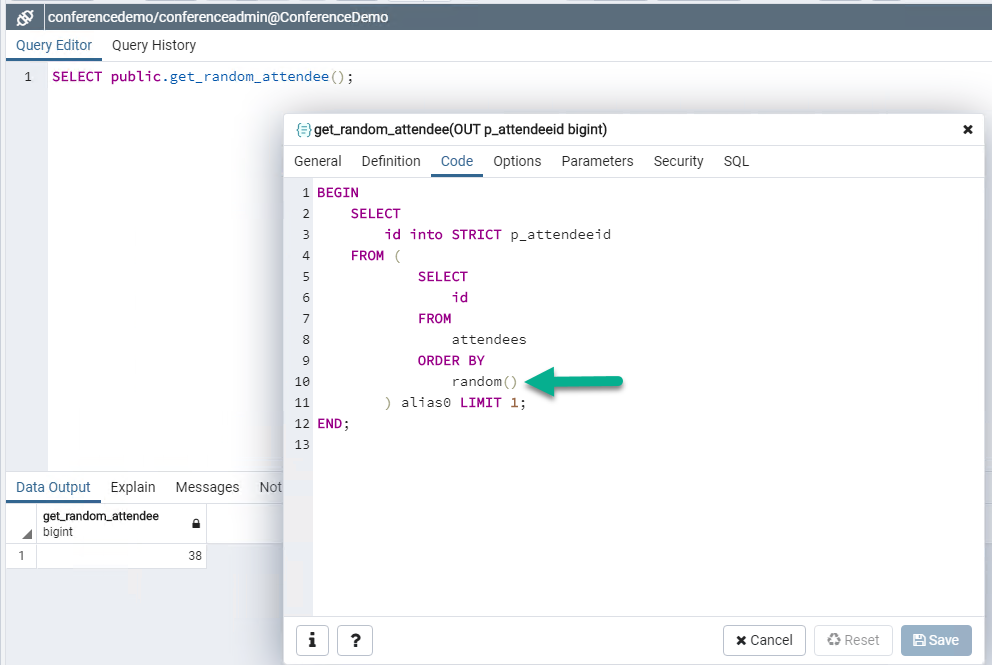


Looks good to me. Let’s run it!

What happened?! The call failed. This conversion script works for a different version of PostgreSQL, but not for Azure Database for PostgreSQL 11.



Migration team members still need to test the converted scripts for issues and refactor when necessary. The correct function call for version 11 of PostgreSQL is ‘random()’, not dbms\_random.value.

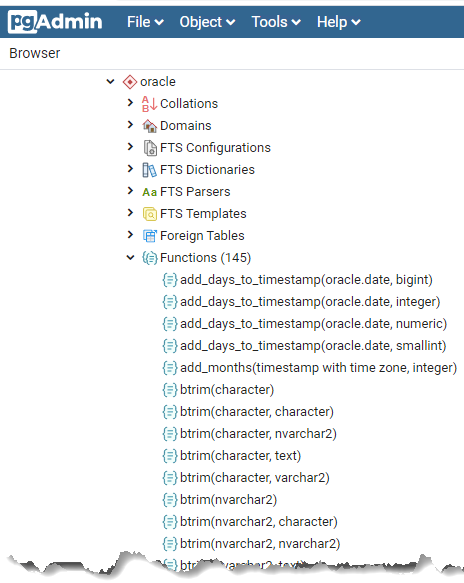


There is another option for handling Oracle function call conversions, the orafce extension.

#### PostgreSQL orafce extension

PostgreSQL provides the ability to extend the functionality of your database by using extensions. Extensions bundle multiple related SQL objects together in a single package that can be loaded or removed from your database with a single command. After being loaded in the database, extensions function like built-in features.

The **orafce** extension for PostgreSQL should be an option to consider. The orafce extension contains some useful functions that can help with porting Oracle application to PostgreSQL. This extension package will save time on code conversion. For example, once installed, the function call to *dbms\_random.value()* should work. In addition to fixing the ‘random’ function error, the extension has several date functions that are typical used in Oracle stored procedures. Also, the capability to use the dummy table “dual” has been added as well.



Some teams may adopt a strategy to move completely to PostgreSQL and convert all Oracle objects. It depends on the resources available and the timelines. The project constraints may not allow for a removing all Oracle like functionality. In this case, the orafce extension can be a bridge until the next project phase.

**Additional resources**

[The Orafce extension on Azure Database for PostgreSQL is now available](https://azure.microsoft.com/en-gb/updates/the-orafce-extension-on-azure-database-for-postgresql-is-now-available/)

[PostgreSQL extensions in Azure Database for PostgreSQL - Single Server](https://docs.microsoft.com/en-us/azure/postgresql/concepts-extensions)

[PostgreSQL Extension Network](https://pgxn.org/dist/orafce/)

[orafce extension GitHub repo](https://github.com/orafce/orafce/)

### Object conversion errors that do not produce runtime errors

Just because a migrated object compiles and executes without runtime errors does not mean there are not underlying issues. Depending on the Oracle version, the evaluation of empty strings may provide different query results.

*“Oracle Database currently treats a character value with a length of zero as null. However, this may not continue to be true in future releases, and Oracle recommends that you do not treat empty strings the same as nulls.”*

*Oracle*

UPDATE sessions SET other\_info = '' WHERE id = 1; COMMIT;

UPDATE sessions SET other\_info = NULL WHERE id = 2; COMMIT;

-- zero rows returned

SELECT COUNT(\*) FROM sessions WHERE other\_info = '';

-- two rows returned

SELECT COUNT(\*) FROM sessions WHERE other\_info IS NULL;

Oracle and PostgreSQL evaluate these statements differently. You will need to review your code to check for the existence of these types of expressions and make conversions where necessary.

*PostgreSQL*

-- one row returned

SELECT COUNT(\*) FROM sessions WHERE other\_info = '';

-- one row returned

SELECT COUNT(\*) FROM sessions WHERE other\_info IS NULL;

By looking at the comments, you can see the query results returned are very different. Can you imagine if this was a financial application? Care must be taken to evaluate the SQL found in the functions and procedures to ensure the expected results are returned.

Empty string vs NULL evaluation is one of handful issues to remember.

**Additional resources**

[Porting from Oracle PL/SQL](https://www.postgresql.org/docs/11/plpgsql-porting.html#id-1.8.8.15.6)

<https://docs.oracle.com/cd/B28359_01/server.111/b28286/sql_elements005.htm>

### Objects that cannot be converted automatically

Stored Procedures

Oracle allows you to write stored procedures in other languages. Below is a Java example.

**PROCEDURE add\_item (stock\_no NUMBER, description VARCHAR2, price NUMBER) AS LANGUAGE JAVA NAME 'LOBManager.addStockItem(int, java.lang.String, float)';**

This stored procedure cannot be automatically converted by ora2pg. The user will need to make a conversion decision, write a new PostgreSQL procedure or convert to application logic.

**Additional resources**

<https://www.postgresql.org/docs/11/sql-syntax.html>

[Oracle to PostgreSQL Wiki](https://wiki.postgresql.org/wiki/Oracle_to_Postgres_Conversion)

### Other useful ora2pg configurations

Your ora2pg conf file has other useful configurations. They are documented in the conf file. Read the descriptions carefully as they have specific valuable warnings. This may save you time as the migration process may error out on large tables if the configurations are not correct.

|  |  |  |
| --- | --- | --- |
| Parameter | Value | Other info |
| LOG\_ON\_ERROR | 1 |  |
| FILE\_PER\_CONSTRAINT | 1 | Allows the migration team to migrate the object in a deterministic way. |
| FILE\_PER\_INDEX | 1 | Allows the migration team to migrate the object in a deterministic way. |
| FILE\_PER\_FKEYS | 1 | Allows the migration team to migrate the object in a deterministic way. |
| FILE\_PER\_TABLE | 1 | Allows the migration team to migrate the object in a deterministic way. |
| DATA\_LIMIT | 2000 | Reduces the likelihood of out of memory errors. |
| FORCE\_OWNER | 1 |  |
| BLOB\_LIMIT | 500 | Reduces the likelihood of out of memory errors. |
| TRUNCATE\_TABLE | 1 | Helps reduce duplicate data insertion. Helpful during test runs. |
| SCHEMA |  | Setting this value will prevent unwanted object creation. |
| PG\_SCHEMA |  | Setting this value will prevent unwanted object creation. |
| ORAFCE | 1 | This setting could help reduce object conversion tasks. Make sure to install the extension first. |
| EXPORT\_INVALID | ? | Review the description carefully. Allows users to export invalid schema objects. |
| COMPILE\_SCHEMA | ? | Compiling the schema could take some time depending on the size of the database and the number of objects. |

### 

### PostgreSQL workarounds for Oracle objects and features

More complex Oracle database objects and features may not have a direct PostgreSQL translation. Workarounds will need to be used.

Synonyms

Database Link

Virtual Columns does not exists, use view instead.

Table partitioning

**Additional resources**

[Porting from Oracle PL/SQL](https://www.postgresql.org/docs/11/plpgsql-porting.html#PLPGSQL-PORTING-OTHER)

### Assessing database complexity and time to import

It is important to understand how the complexity involved with migrating your database. The ora2pg utility has a built-in report that can assist with this task.

Sample command. \*\* Remember to pipe to an html file \*\*\*



#### Example report





This report provides a starting point for the calculation of project complexity. The calculated value is not meant to be the exact amount of time for migration. Use the value as relative context when comparing to other projects.

The time required to migrate this database depends on the complexity of the schema objects and the amount of content. This application is considered to be an easy migration.

### Comparing the Oracle and PostgreSQL instance schema

After you create the tables in the PostgreSQL source database, it may be time to verify all of the objects were created as expected. Ora2pg provides a useful feature to easily compare the two databases at the schema level.

#### Running the migration object comparison script

ora2pg -c config/ora2pg\_dist-my-migration-test.conf -t TEST --namespace REG\_APP>migration\_diff.txt

Example



## Migration

Once your team has an opportunity to review the converted schema and decide if the default choices were acceptable, it may be time to test the migration.



### Sample application modifications

<TODO: Show the changes required to make the sample work.>

### Copying the data over to Azure PostgreSQL

In this example, only the **SESSIONS** table is being transferred to PostgreSQL. This type of functionality is helpful if you are required to carefully verify each table. Tables with millions of rows may need this type of process of progress verification. The team can focus on the schema and data transfer to ensure correctness.

When migrating data, always use COPY data export mode. You will get a performance boost compared to using the INSERT command. You should also set the PG\_SCHEMA configuration. Otherwise, the objects and data transferred will be created in the PUBLIC schema.



Example of the PostgreSQL table updated



To copy the tables from the database in parallel, use:

ora2pg -d COPY -j 8 -J 8 [-c /etc/ora2pg/ora2pg.conf]

\* j 8 specifies the number of parallel copies to Destination PostgreSQL

\* J 8 specifies the number of parallel copies from Source Oracle

Make sure you have enough memory on your migration server otherwise you might have server resource problems. The process of migration can be memory intensive.

The default ora2pg settings for COPY could be used against a simple database that does not have much data. The screen below gives you an idea of the processing messages and speed.



Notice the data was copied over to the PostgreSQL database. This included the blob and clob fields.



Tables with simple number columns can copy millions of rows quickly. Tables containing blobs will take much longer to import, especially if the blobs are large. Extra time will need to be allocated for this import process.

Running the COPY command multiple times can cause duplicate data to be inserted into the target database. If you need to run this command multiple times, you will need to truncate the data or filter the data from the watermark point. Forgetting this process could be very problematic on large data sets. Data anomalies creep in to your application with duplicate data. Depending on your constraints, you will receive errors upon constraint creation.



### Cutting over

With online (minimal downtime) migrations, the source you are migrating continues to change, drifting from the target in terms of data and schema, after the one-time migration occurs. During the Data cutover phase, you need to ensure that all changes in the source are captured and applied to the target in near real time. After you verify that all changes in source have been applied to the target, you can cutover from the source to the target environment.

As of May 2019, if you want to perform an online migration, consider using Attunity Replicate for Microsoft Migrations or Striim. For “delta/incremental” migration using ora2pg, the technique consists in applying for each table a query that applies a filter (cut) by date or time, etc., and after that finalizing the migration applying a second query which will migrate the rest of the data (leftover).

In the source data table, migrate all the historical data first. An example of that is:

select \* from table1 where filter\_data < 01/01/2019

You can query the changes made since the initial migration by running a command similar to the following:

select \* from table1 where filter\_data >= 01/01/2019

In this case it is recommended that the validation is enhanced by checking data parity on both sides, source and target**.**

## Post-migration

****

### Should you convert Stored Procedure and Functions to application code?

Stored procedures have an advantage of reducing the round trips between the application and the database. If you are struggling to convert stored procedures and functions that are several of hundreds lines of code into PostgreSQL, consider moving that logic to the application. Application logic can be tested easily via unit tests. The code can be broken down into smaller units of work making it easier to maintain and debug. Carefully weigh out the advantages of running the logic in the application layer vs at the database server. Consider creating a technical spike to measure the actual execution performance. Compare your test results to see if there is a performance justification for the related stored procedure maintenance costs.

### Architecture strategies

<TODO: Finish>

Many architects are choosing to break apart their monolithic applications into smaller domain microservices. You could choose a strategy of utilizing multiple PostgreSQL databases instead of one giant database. Significant application architecture would change the development time.



**Additional resources**

[Oracle Inventory Script Artifacts](https://github.com/microsoft/DataMigrationTeam/tree/master/IP%20and%20Scripts/Oracle%20Inventory%20Script%20Artifacts/Oracle%20Inventory%20Script%20Artifacts)

## Have questions?

For any questions or suggestions about working with Azure Database for PostgreSQL, send an email to the Azure Database for PostgreSQL Team ([@Ask Azure DB for PostgreSQL](mailto:AskAzureDBforPostgreSQL@service.microsoft.com)). This address is for general questions rather than support tickets.

In addition, consider these points of contact as appropriate:

* To contact Azure Support or fix an issue with your account, [file a ticket from the Azure portal](https://portal.azure.com/?#blade/Microsoft_Azure_Support/HelpAndSupportBlade).
* To provide feedback or to request new features, create an entry via [UserVoice](https://feedback.azure.com/forums/597976-azure-database-for-postgresql).