

A Guide to Migrating a Java Oracle Application 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](#_Toc37177927)

[Scenario overview 7](#_Toc37177928)

[Legacy on-premises application architecture 8](#_Toc37177929)

[Sample Oracle Database ER Diagram 10](#_Toc37177930)

[Getting the document artifacts from Git repo 11](#_Toc37177931)

[Tour of the application 13](#_Toc37177932)

[Application target goal 15](#_Toc37177933)

[Database migration process 16](#_Toc37177934)

[Migrations types 17](#_Toc37177935)

[Azure hosting options 18](#_Toc37177936)

[Azure PostgreSQL Introduction 19](#_Toc37177937)

[Measure performance and plan for optimization tasks 20](#_Toc37177938)

[Database migration tool options 21](#_Toc37177939)

[Setting up your migration server 23](#_Toc37177940)

[Choosing your migration server 23](#_Toc37177941)

[Hardware resources 24](#_Toc37177942)

[Securing the data during migration 24](#_Toc37177943)

[Getting started: Download and install the Oracle database client library 25](#_Toc37177944)

[Set up the environment variables 25](#_Toc37177945)

[Set up the pgAdmin PostgreSQL database client 25](#_Toc37177946)

[Download and install the ora2pg utility 26](#_Toc37177947)

[Discovering and assessing the source database with ora2pg 27](#_Toc37177948)

[Prepping your database for export 27](#_Toc37177949)

[Checking for invalid Oracle objects 28](#_Toc37177950)

[Before migration data trimming and column refactoring 30](#_Toc37177951)

[Create your ora2pg conf structure 31](#_Toc37177952)

[Add your Oracle and PostgreSQL DSN configuration to the conf file 32](#_Toc37177953)

[Testing your database connections and permissions 34](#_Toc37177954)

[Separating the constraints and indexes into files 35](#_Toc37177955)

[Exporting each of the tables as a separate files 39](#_Toc37177956)

[Evaluate the data type conversions 39](#_Toc37177957)

[Layering on the sequences and triggers 40](#_Toc37177958)

[Converting the procedures 40](#_Toc37177959)

[Other default ora2pg code conversions to consider 45](#_Toc37177960)

[Object conversion errors that do not produce runtime errors 49](#_Toc37177961)

[Objects that cannot be converted automatically 50](#_Toc37177962)

[Other useful ora2pg configurations 51](#_Toc37177963)

[PostgreSQL workarounds for Oracle objects and features 52](#_Toc37177964)

[Assessing database complexity and time to import 52](#_Toc37177965)

[Comparing the Oracle and PostgreSQL instance schema 54](#_Toc37177966)

[Migration 55](#_Toc37177967)

[Filtering your source data 55](#_Toc37177968)

[Sample application modifications for the PostgreSQL database 56](#_Toc37177969)

[Copying the data over to Azure PostgreSQL 58](#_Toc37177970)

[Data synchronization 62](#_Toc37177971)

[Post-migration 64](#_Toc37177972)

[Should you convert Stored Procedure and Functions to application code? 64](#_Toc37177973)

[Architecture strategies 65](#_Toc37177974)

[Have questions? 65](#_Toc37177975)

## 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 a simple to moderately complex web applications that need to switch the database to PostgreSQL. Several real-world migration warnings and tasks are listed in this document.

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 are great candidates for database migrations. ORM tools abstract the mapping of data and types. The sample Oracle database fields, objects, and data were 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 the 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 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 an Oracle to PostgreSQL database migration using a small database like the one described in this document. 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 its 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 evolved 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 Azure Database for 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.
* Built-in high availability
* Added enterprise security: Advanced threat protection, bring your own key (BYOK), AD integration, Private Link
* Other benefits: automatic backups and point-in-time-restore (better SLA).
* Automatic patching.
* Total cost of ownership is very attractive. There is no a la carte model licensing necessary. All of the features are available for one price.
* Cost of licensing and support is pay-as-you-go for the open source software. All-inclusive pricing from one vendor.

Steps required to migrate to this database platform need to be explored by the development team. Initial comparisons of PL/SQL and PL/PgSQL reveal the syntax is very similar between the two databases. Given the similarities, the ramp-up time should be lower compared to other database options.

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 most significant 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.



The Angular framework provides a light HTML frontend that calls the Java API using RESTful calls. The Java layer contains all of the business and data access logic.

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/)

## Sample 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.

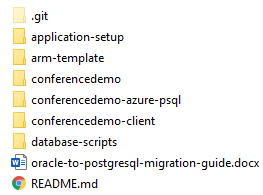


## Getting the document artifacts from Git repo

The reference application and database artifacts can be found in the Git repo.

<TODO: Microsoft 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 you have a contextual understanding of the application, we will move on to the process of assessment, migration, and conversion.

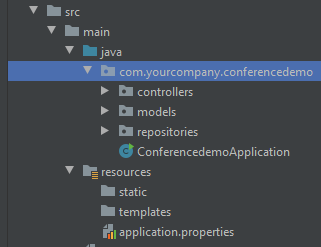
If you are utilizing a database server other than a local instance, the migration user account will need elevated permissions to the source database in order to properly capture the schema and data information for PostgreSQL export. Grant all the rights for a local testing database instance. It makes the learning experience easier.



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

## Tour of the sample application

The Java API project structure follows the Model View Controller (MVC) pattern.



Web requests are handled by the controllers, and the repositories map the database calls.

Each of these layers could have been separated out into a modules within the Maven project, but for simplicity purposes they are in one module with different namespaces.

The Spring framework was chosen for the sample project because it makes creating Java applications easy, and it is a very popular among the development community. The Hibernate ORM framework allows the application to handle the calls to the database and the mapping of the database query results to Java POJOs. Developers spend more time providing application solutions instead of basic data access plumbing.

The entire solution goal was to exercise basic CRUD actions.

#### Landing page

This landing page assumes the user has been authenticated and is ready to choose an event. Users can choose from the list of events. Once the event is selected, users can see a list of event sessions.



#### Event session list

The session list is presented after selecting an event. Attendees can register for the sessions on this page by selecting the ‘Register Here’ button. The register action saves the user and session data to the database. Users can select the speaker link to see the bio details. This page utilizes database views for content.



#### Speaker bio details

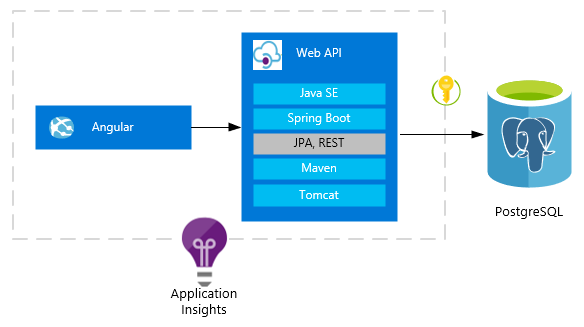
The speaker bio contains a picture (blob) and background information (clob).



## Application target goal

Weighing 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. Also, you can scale the environment resources up or down, depending on your applications’ needs.

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 familiar with the 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 with small conversion 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.

## Azure hosting options

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



When it comes time to migrate application from your on premise environment, you will need to make some important decisions. You could purchase, manually configure, and secure multiple VMs to meet this architecture need, known as infrastructure as a service (IaaS). Another option to consider is the platform as a service (PaaS) approach.

PaaS is a complete development and deployment environment in the cloud, with resources that enable you to deliver everything from simple cloud-based apps to sophisticated, cloud-enabled enterprise applications. You purchase the resources you need from a cloud service provider on a pay-as-you-go basis and access them over a secure Internet connection.

Like IaaS, PaaS includes infrastructure—servers, storage, and networking—but also middleware, development tools, business intelligence (BI) services, database management systems, and more. PaaS is designed to support the complete web application lifecycle: building, testing, deploying, managing, and updating.

PaaS allows you to avoid the expense and complexity of buying and managing software licenses, the underlying application infrastructure and middleware, container orchestrators such as Kubernetes, or the development tools and other resources. You manage the applications and services you develop, and the cloud service provider typically manages everything else.

Organizations typically use PaaS for these scenarios:

**Development framework.** PaaS provides a framework that developers can build upon to develop or customize cloud-based applications. Similar to the way you create an Excel macro, PaaS lets developers create applications using built-in software components. Cloud features such as scalability, high-availability, and multi-tenant capability are included, reducing the amount of coding that developers must do.

**Analytics or business intelligence.** Tools provided as a service with PaaS allow organizations to analyze and mine their data, finding insights and patterns and predicting outcomes to improve forecasting, product design decisions, investment returns, and other business decisions.

**Additional services.** PaaS providers may offer other services that enhance applications, such as workflow, directory, security, and scheduling.

**Additional resources**

[What is PaaS?](https://azure.microsoft.com/en-us/overview/what-is-paas/)

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

## Azure 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. The goal should be better performance after the migration completion. Get agreement on 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 the current environment performance in a way the test 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
* CPU and memory utilization

See: [Performance best practices for using Azure Database for PostgreSQL](https://azure.microsoft.com/en-us/blog/performance-best-practices-for-using-azure-database-for-postgresql/)

You can build up realistic tests in a UI testing tool, like [Selenium](https://docs.microsoft.com/en-us/azure/devops/pipelines/test/continuous-test-selenium?view=azure-devops). This tool allows you to replay the same test cases in each environment. You can use the [PostgreSQL logging](https://www.postgresql.org/docs/current/runtime-config-logging.html) to capture SQL statements as well for future load generation.

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. The tests must be significant and take several minutes or hours to complete in order to get reproducible numbers. A testing script that completes in seconds most likely contains inaccurate numbers and should not be relied upon as a reference point.

Testing results may lead to an Azure environment hardware configuration and sizing change. 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**

[UI test with Selenium](https://docs.microsoft.com/en-us/azure/devops/pipelines/test/continuous-test-selenium?view=azure-devops)

[Optimize performance using Azure Database for PostgreSQL Recommendations](https://azure.microsoft.com/en-us/blog/optimize-performance-using-azure-database-for-postgresql-recommendations/)

[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)

[Performance best practices for using Azure Database for PostgreSQL](https://azure.microsoft.com/en-us/blog/performance-best-practices-for-using-azure-database-for-postgresql/)

<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. The option will depend on the timeline, budget, and database complexity.

**Azure Database Migration Services**

For online data 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.

[Known issues/migration limitations with online migrations from Oracle to Azure DB for PostgreSQL-Single server](https://docs.microsoft.com/en-us/azure/dms/known-issues-azure-postgresql-online)

**Other commercial utilities**

Other commercial data transfer utilities exist in the marketplace. For example, [Qlik Replicate](https://www.attunity.com/products/replicate/attunity-replicate-for-microsoft-migration/) and [Striim](https://www.striim.com/change-data-capture/) facilitate and simplify 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 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 due to data clean up and migration restarts. The cost of delays may exceed the cost of utilizing the proper Azure SKU from the beginning of the project.

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

### 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

### Getting started: 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 to administrate and query the PostgreSQL database. The utility can be downloaded 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.

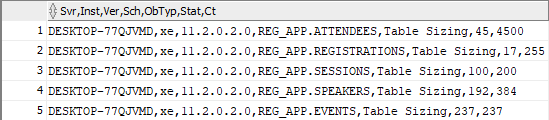


### Checking for invalid Oracle objects

One of the important steps before migration, is checking for invalid database objects. The Data Migration Team at Microsoft wrote an Oracle select statement 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 the ora2pg 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, unexpected results may occur. Also, an invalid object may not be used anymore and maybe a candidate for deprecation.

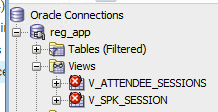
This Oracle query will produce an inventory of database objects.





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



### Before migration data trimming 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. Are there old temp tables lying around in the database? 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 objects, 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, sign off from management and possibly the records management team will be required 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. This approach will ensure the highest likelihood of overall success. 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. For example, if a team member changes a flag in the conf file that filters export data to one schema, that change may need to be reviewed. Steps to separate out your scripts will be described in subsequent sections.

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

Example output



Now that your base structure has been created, place your **ora2pg conf** file into the config folder. If you want to test conf files configurations, create a copy. Copy of the ‘ora2pg.conf’ was created for testing configuration parameters. Once you are comfortable with the parameter changes, you can transfer them to your master conf file.



### 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 for your Oracle instance, try running the ‘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 exported?! What happened? Why are the apex tables in the output?



You need to specify the schema/namespace. All schemas the user has access to are exported by default.



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.



When migrating a large database, the indexes and constraints should be applied once the tables have been created and the data has been imported. The import performance will be significantly better compared to a database with all of the indexes and constraints applied.

**Notice all of the table names appear in lower case?**  If previous naming conventions makes use of upper case names for objects, consider adopting a new appreciation for lower case named objects. Otherwise, referenced tables and columns with upper case letters will require quotes. The quotes workaround can lead to possible coding errors.

See: [Table name case matters](#_Table_name_case)

### Separating the constraints and indexes into files

To separate the indexes and constraints into separate 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.



Given the command in the example above, all the tables schemas can be found in the ‘reg\_app-psql.sql’ file. Notice only the tables are shown and the indexes and constraints are missing.

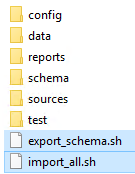


A database with many tables or tables with many fields may require a different migration strategy. You could create a script for each table. This will allow team members, like a database administrator, to evaluate and optimize the conversions before migration.

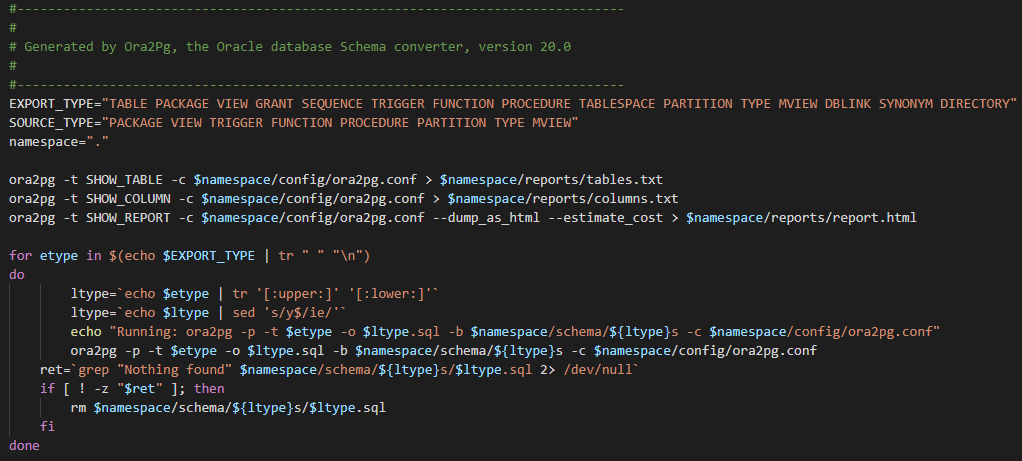




If you are using a Linux migration server, consider utilizing the helpful Bash shell scripts to export the objects into separate files.

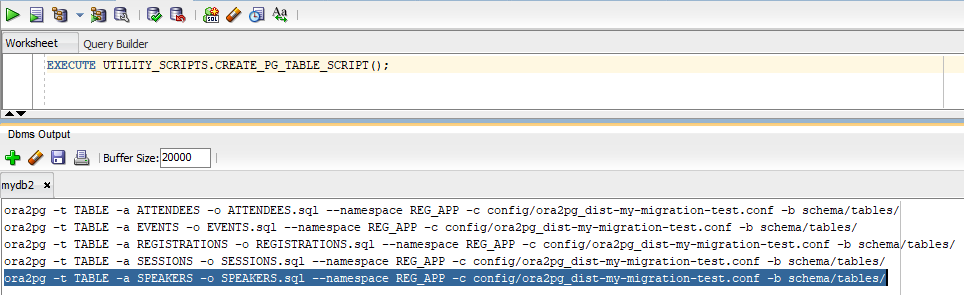


This snippet from the shell script exports each of the types to the schema folder and provides some useful reports.

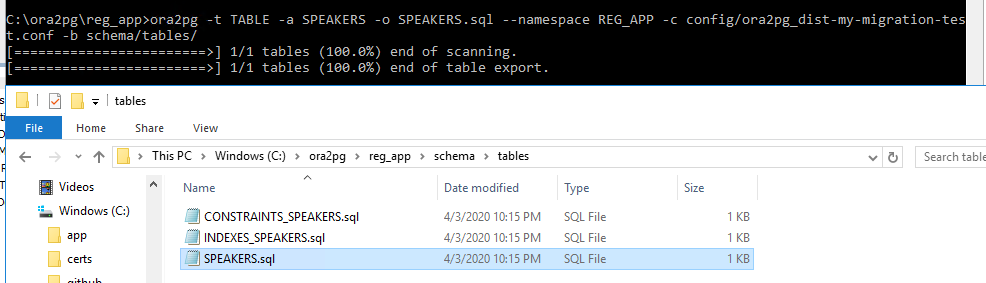


### Exporting each of the tables as a separate files

As part of the sample Oracle database included in the Git repo, there is a sample script to export all of the tables as separate ora2pg scripts.



This allows the migration team to focus on one table at a time. Teams tasked with a large database conversion will appreciate this approach, especially if the table has many columns. Table column data types can be evaluated. You will need to ensure your conf file is configured with the separate file parameters set.



### Evaluate the data type conversions

Most of the ora2pg data type conversions 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 for storage and performance reasons. Do I need a bigint or is the int data type sufficient?



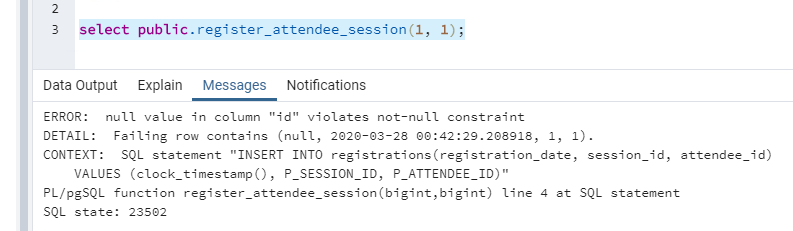
### Layering on the sequences and triggers

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:



This document only covers some of the basic commands. The ora2ph help command will show you all of the features and their descriptions.

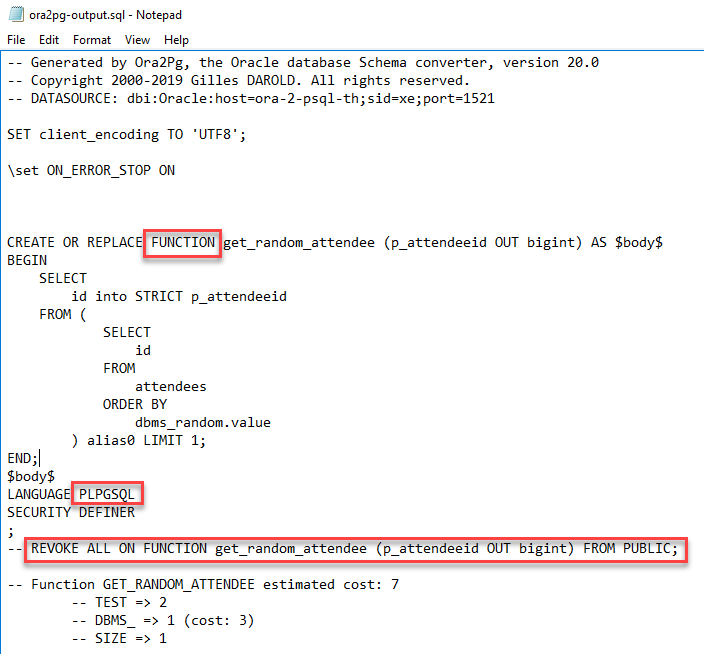
ora2og --help

### Converting 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 tables, 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](#_Checking_for_invalid) 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. Will this work?



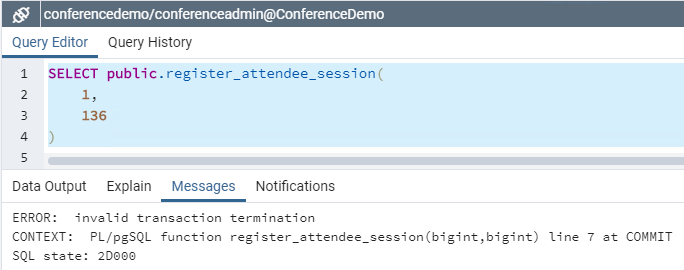
This is a contrived example. The SQL statement could have been written without nesting, etc. Part of the goal for this example is to show how ora2pg will convert nested queries.

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.

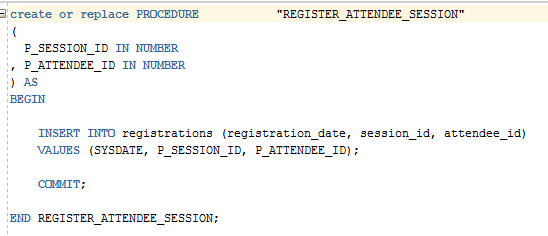
The ora2pg script converted the procedures to functions. Are functions and procedures in PostgreSQL 11 the same? No! The main difference is a function returns a value, whereas a procedure does not. Also, do you plan on changing values in the tables during function processing? Does your function need to see changes as the function processes? Any function with side-effects must be labeled VOLATILE.

See: [Function Volatility Categories](https://www.postgresql.org/docs/current/xfunc-volatility.html)

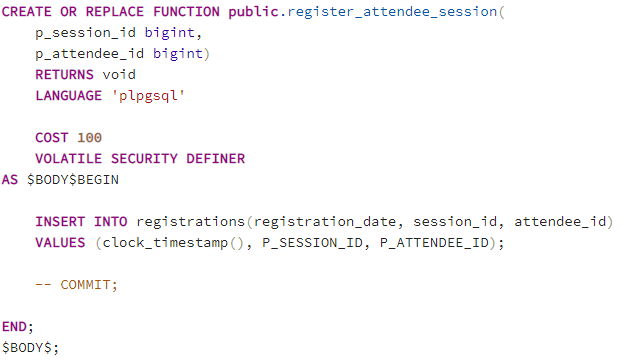
Why did we get this error message?



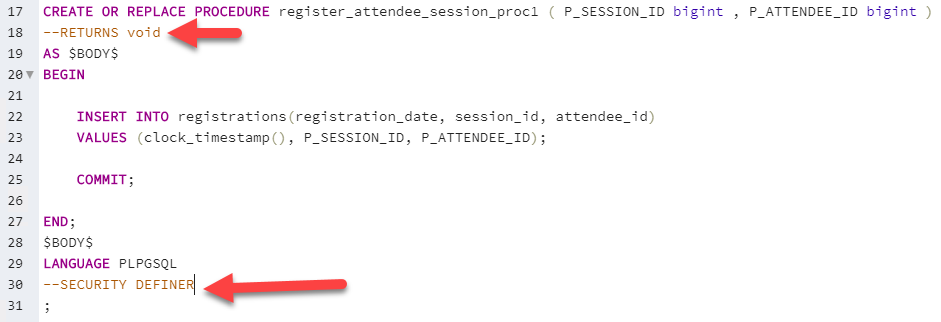
Below is the original Oracle procedure. Hibernate could have handled this transaction in the application. However, the focus of this exercise is to determine the conversion options we have available. Let’s assume this procedure had several lines of business logic ending with a COMMIT or ROLLBACK and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



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. Remember, procedures do not return anything. Remove unnecessary comments for production deployments. Also, procedures and functions cannot have the same name and arguments. If you are testing changing a function into a procedure, you may have to rename or drop the function first. Otherwise, you will get an error upon saving.



*“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: [PostgreSQL documentation](https://www.postgresql.org/docs/11/)

**Additional resources**

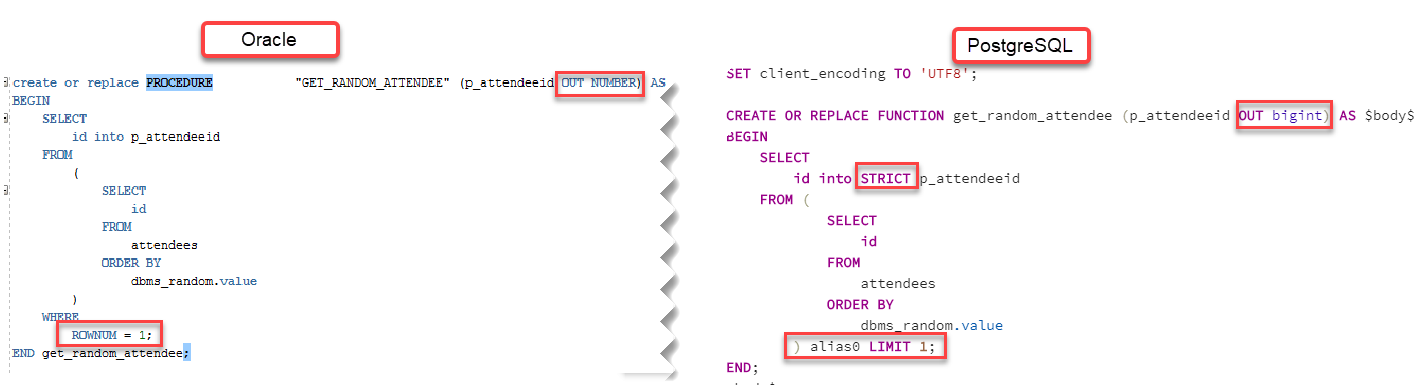
<https://www.postgresql.org/docs/11/>

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

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

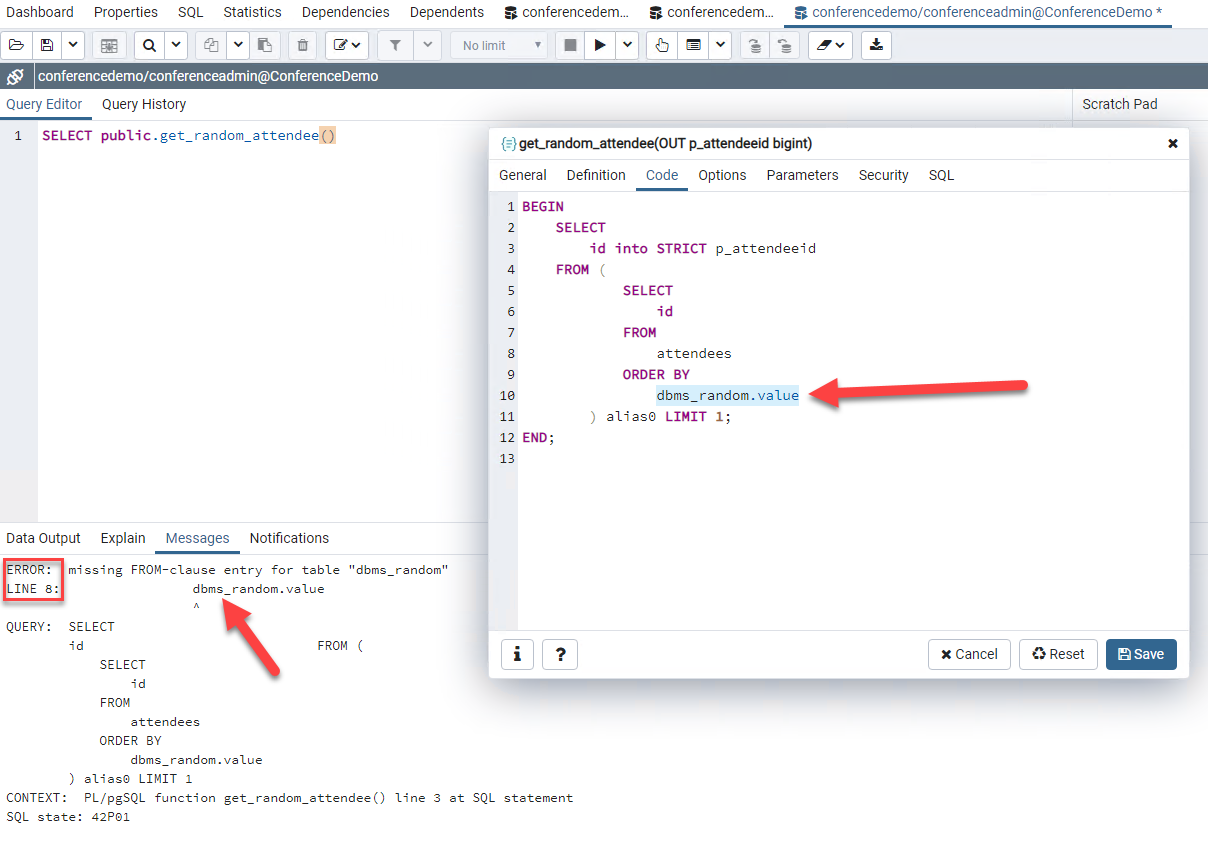
### Other default ora2pg code conversions to consider

ora2pg does a good job of converting Oracle PL/SQL syntax to PostgreSQL PL/pgSQL. Most of the syntax looks similar. Additional syntax maybe inserted that is not necessary by the conversion process. For example, STRICT is not necessary for the command to run properly.

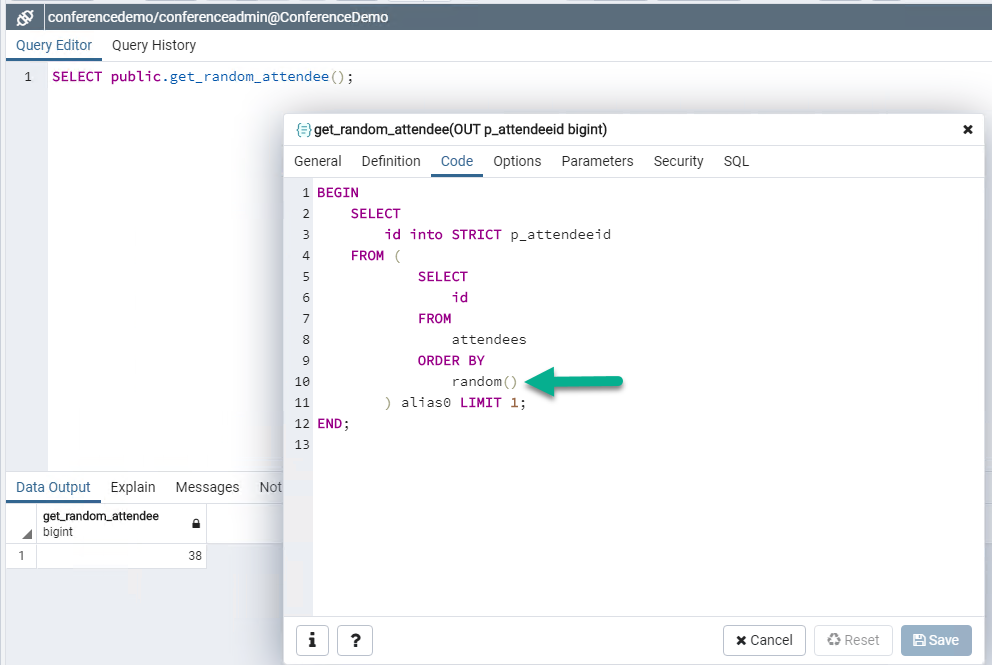


Looks good to me. Let’s run it!

What happened?! The call failed. This conversion script works for a specific commercial version of PostgreSQL, but not for Azure Database for PostgreSQL 11. Azure Database for PostgreSQL is built on the community edition. The ora2pg utility does a good job of converting most of the syntax, but some updates may be required.



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

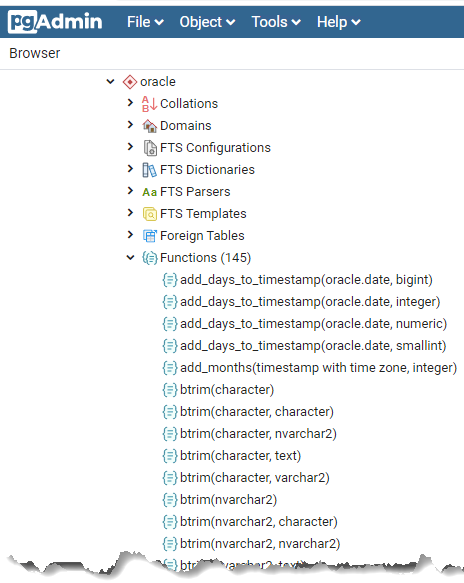
PostgreSQL provides the ability to extend the functionality of your database by using extensions. Extensions bundle multiple related SQL objects together in a single package that can be loaded or removed from your database with a single command. After being loaded in the database, extensions function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

CREATE EXTENSION orafce IF NOT EXISTS

In addition to fixing the ‘random’ function error, the extension has several date functions that are typically used in Oracle stored procedures. Also, the capability to use the dummy table “dual” has been added as well.

Example of the installed extension package.



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 removing all Oracle like functionality. In this case, the orafce extension can be a bridge until the next project phase.

Other helpful extensions:

* hstore
* PostGIS
* Pg\_partman
* Timescale
* Postgres FDW

**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 a handful issues to remember.

By default, Ora2Pg replaces all conditions with a test on NULL by a call to the coalesce() function.

**Additional resources**

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

[Database SQL Language Reference - NULL](https://docs.oracle.com/cd/B28359_01/server.111/b28286/sql_elements005.htm)

### Objects that cannot be converted automatically

Unfortunately, not all objects can be converted by the utility or even provide a basis for the conversion process. For example, Oracle allows you to write stored procedures in languages other than SQL.

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 review manually and make a conversion decision, write a new PostgreSQL procedure or convert to application logic. These efforts will happen for some of the Oracle objects.

**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

The ora2pg conf file has other useful configurations. The parameter descriptions are found 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 | Enable this directive if you want to continue direct data import on error. |
| 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 | Force Ora2Pg to set the object owner to be the one used in the Oracle database. |
| 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. |
| REPLACE\_QUERY | ? | Helps filter down the exported data. Could be used for cutover tasks. |
| WHERE | ? | Helps filter down the exported data. |

### 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. Below are some types of conversion problems you might encounter.

|  |
| --- |
| Synonyms - exported as VIEWS |
| External table - use file\_fdw. |
| Database link - use FDW. |
| Virtual columns – use a view instead. |
| Table partitioning |
| BFILE data type – use config parameter DATA\_TYPE = BFILE:TEXT or DATA\_TYPE = BFILE:EFILE |
| Global variables in packages, use dedicated tables instead or SET/SHOW |
| DECODE – only simple conversions |
| Jobs – use pgAgent |
| CURSOR BULK COLLECT |
| global temp tables migrated to postgres as unlogged tabled with RLS |
| pragma autonomous\_transaction |

This is not a complete list.

**Additional resources**

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

[pgAdmin pgAgent](https://www.pgadmin.org/docs/pgadmin4/1.22.2/pgagent.html)

### 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. Is this going to be an easy migration project?

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

Migration report 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.



### Filtering your source data

For testing data migration, consider using this conf file parameter to limit exported data:

**WHERE** ROWNUM < 1000

**REPLACE\_QUERY** EMPLOYEES[SELECT e.id,e.fisrtname,lastname FROM EMPLOYEES e JOIN EMP\_UPDT u ON (e.id=u.id AND u.cdate>'2014-08-01 00:00:00')]

Limiting your source data export can be very helpful verifying your schema choices. Detecting problems with limited data is better than pouring out the database only to find an obvious error in the beginning of the export and having to restart. Also, these parameters can be used for synchronizing data across environments.

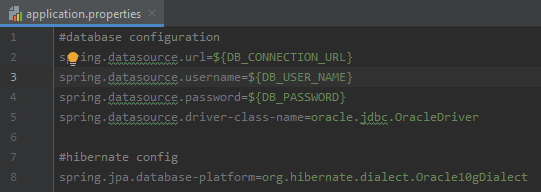
See: [Data synchronization](#_Data_synchronization).

### Sample application modifications for the PostgreSQL database

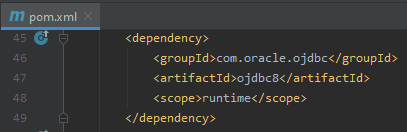
When you think about changing application databases, you might be concerned about the several required changes. Java applications leveraging ORM frameworks usually have very few changes for migration cutover.

In order to connect your Java application to Oracle, you set these configurations:

Application properties

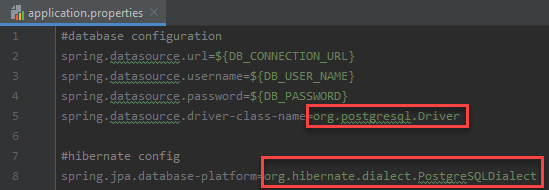


POM file



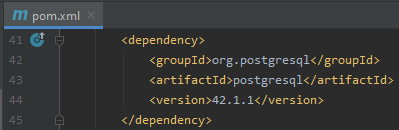
That is a pretty simple configuration. To switch the connection to Azure Database for PostgreSQL, here are the changes required:

Change your provider in the application properties



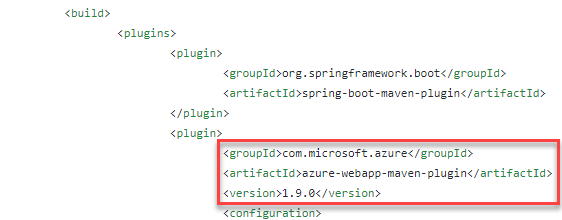
Example DB\_CONNECTION\_URL:

jdbc:postgresql://your-ora2pg-server.postgres.database.azure.com:5432/conferencedemo?ssl=true&sslmode=require

Update your POM file to pull the PostgreSQL libraries

You will need to change your build configuration in your POM file for your new Azured hosted environment. For details on the POM file changes, check the ‘conferencedemo-azure-psql’ folder in the Git repo.

The most important change to note is the addition of the [Maven Plugin for Azure App Service](https://docs.microsoft.com/en-us/java/api/overview/azure/maven/azure-webapp-maven-plugin/readme?view=azure-java-legacy). This plugin makes the deployment to the Azure hosted server seamless.



#### Table name case matters

There are some other small changes that need to be made to the models.

In PostgreSQL, the upper case ‘EVENTS’ and lower case ‘events’ tables are different. Unfortunately, ‘EVENTS’ in upper case is not recognized by default in Hibernate.

Screenshot from pgAdmin

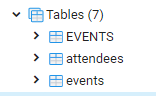


Table names: “EVENTS” <> events. Case matters!

‘SELECT \* FROM events’ is different from ‘SELECT \* FROM “EVENTS” ’.

See: [Lexical Structure](https://www.postgresql.org/docs/current/sql-syntax-lexical.html)

You will need to add some adjustments to the @Table annotations for each of the Java models.

Example of the changes made to the Event model.



### 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 the COPY data export mode. You will get a performance boost compared to using the INSERT command. The PG\_SCHEMA parameter should be set. Otherwise, the objects and data transferred will be created in the PUBLIC schema.



Example of the PostgreSQL table updated



To speed up data migration, consider copying 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.



### Data synchronization

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 the source database have been applied to the target, you can cutover from the source to the target environment.

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

select \* from table1 where create\_date < 01/01/2019

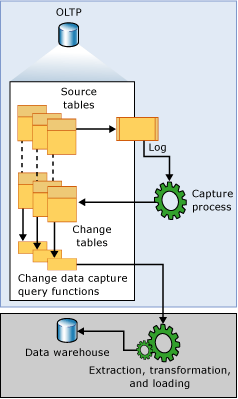
You can query the changes made since the initial migration by running a command similar to the following:

select \* from table1 where create\_date >= 01/01/2019

In this case it is recommended that the validation is enhanced by checking data parity on both sides, source and target**.** This type of data synchronization technique works well for INSERT changes. The source tables require fields capturing the create date and time for the inserted data. Tracking only inserted data might be unrealistic for your application needs.

Synchronizing source tables for all data changes (INSERT, UPDATE, DELETE) will require a Change Data Capture (CDC) or a Change Tracking (CT) solution. The migration team will need to capture all of the source table changes in log tables and use an ETL tool, like [Qlik Replicate](https://www.qlik.com/us/products/technology/qlik-microsoft-azure-migration) or [Striim](https://www.striim.com/change-data-capture/), to synchronize the data between the two environments.

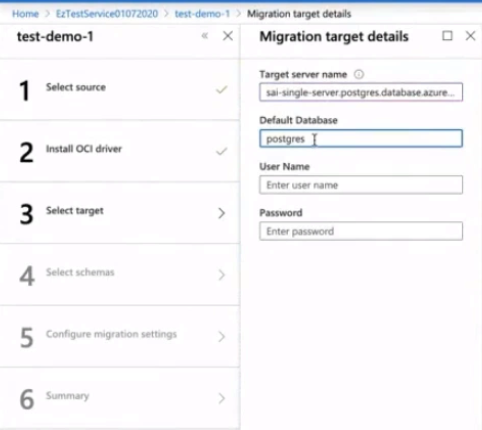
The following illustration shows the principal data flow for change data capture.



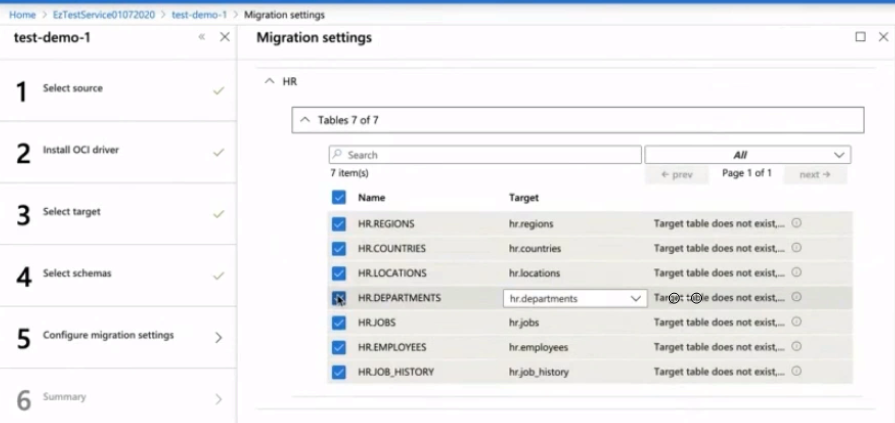
As inserts, updates, and deletes are applied to tracked source tables, entries that describe those changes are added to the log. The log serves as input to the capture process. The capture process reads the log and adds information about changes to the tracked table's associated change table. Finally, at the appropriate interval, the ETL service will execute queries and migrate the changed data to the target environment.

The implementation details of this solution go beyond the scope of this document. Review the additional resources for more information.

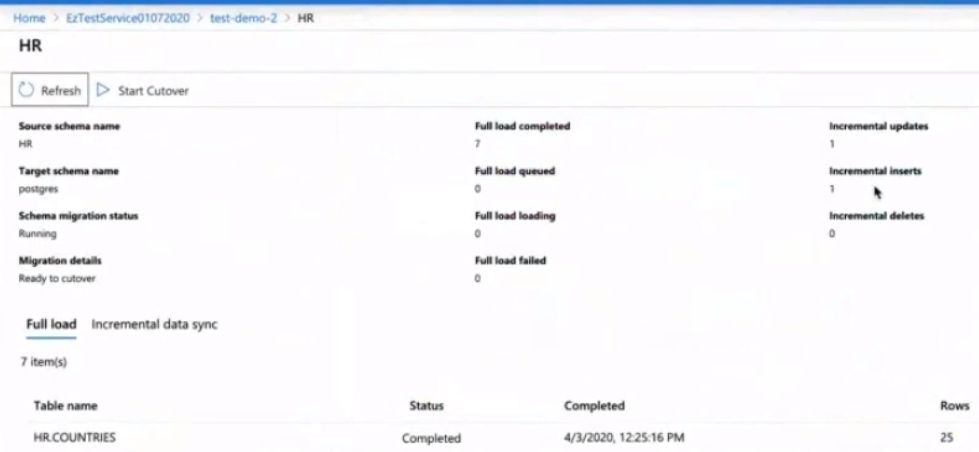
Another easy and effective solution for data transfer is Azure Database Migration Services (DMS). Azure DMS can also migrate the changed data as well between environments without the complexity of custom solution. There is no need to set up an extra migration server for the data transfer. Microsoft automates the VM setup and configuration. The workflow screen provides an easy step by step migration process.

****

The workflow is very flexible. Migration users can migrate tables in a phased approached. They can choose which tables they need to migrate.



Again, the migration first step will be to perform the full load. The bulk of the data will be transferred. Once complete, the remaining incremental changes will need to be transferred. Azure DMS can handle the full load as well as the incremental changes.



**Additional resources**

[Azure Database Migration Guide](https://datamigration.microsoft.com/scenario/oracle-to-azurepostgresql?step=1)

<https://www.striim.com/change-data-capture/>

[Change data capture](https://en.wikipedia.org/wiki/Change_data_capture)

[Streaming Data With Change Data Capture](https://www.qlik.com/us/data-streaming/data-streaming-cdc)

<https://docs.microsoft.com/en-us/azure/dms/tutorial-oracle-azure-postgresql-online>

## 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. The business logic is executed on the database server saving network calls and data transfer. However, if you are struggling to convert stored procedures and functions that are several hundreds of 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 the test results to see if there is a performance justification for the related stored procedure maintenance costs.

### Architecture strategies

Many architects are choosing to break apart their monolithic applications into smaller domain [microservices](https://martinfowler.com/articles/microservices.html). You could choose a strategy of utilizing multiple PostgreSQL databases instead of one giant database. A microservice is meant to be as small as possible: to be light when spinning up, to have a small footprint, to have a small Bounded Context (check DDD, Domain-Driven Design), to represent a small area of concerns, and to be able to start and stop fast.



Significant application development and testing time would be required to implement this design.

**Additional resources**

.[NET Microservices: Architecture for Containerized .NET Applications](https://docs.microsoft.com/en-us/dotnet/architecture/microservices/)

## 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).