MIGRATION PLAYBOOK · DATABASES

# How to migrate from Oracle to PostgreSQL

Assess the Oracle schema, convert tables and PL/SQL to PostgreSQL, move the data with a bulk load or change data capture, prove the result row by row, then cut the application over with a tested way back.

Updated 4 Nov 2026  7 phases  31 detailed steps  11 official sources

START HERE

## How do you migrate from Oracle to PostgreSQL?

SHORT ANSWER

This moves an Oracle database and the application that uses it onto PostgreSQL. Tables convert almost automatically, but PL/SQL code (Oracle's built-in programming language) and the SQL inside the application must be converted and tested, and that work sets the timeline.

- **Recommended route:** Your target cloud's migration service, checked with an Ora2Pg assessment
- **Order:** Assess first, then convert the schema and the code before any real data moves, because the data copy is the easier, repeatable part.
- **7 phases:** Assess, Prepare PostgreSQL, Convert the schema, Convert and test the code, Move the data, Validate and rehearse and Cut over.
- **You're done when:** Object and row counts match for every schema, and row contents or column totals match.

The recommended route uses your target cloud's migration service (AWS DMS on AWS, or Google Database Migration Service on Google Cloud), which can keep PostgreSQL in sync with Oracle so the final outage is short. On Azure or your own servers, the free Ora2Pg tool moves the data instead, during an outage. Use the Ora2Pg assessment as a second opinion on any route.

RECOMMENDED ROUTE

### Your target cloud's migration service, checked with an Ora2Pg assessment

AWS DMS (for Aurora or RDS) and Google Database Migration Service (for Cloud SQL or AlloyDB) convert the schema and keep copying changes until cutover. Ora2Pg costs an afternoon and flags the hardest objects.

▼ **Other options and when to use them**

- **VS Code schema conversion plus Ora2Pg for data**
  : Your target is Azure Database for PostgreSQL flexible server and you can take an outage for the data copy.
- **Ora2Pg and native PostgreSQL tools only**
  : Self-managed or any other PostgreSQL, you want full control of the generated DDL, and the data copy fits in an outage.

**TOOLS FOR THIS PLAN**

| Are you ready? | Plan your dates | New to the jargon? |
|---|---|---|
| 22-question readiness check | Dated countdown you can add to your calendar | Plain-English migration glossary |

**ORDER TO MIGRATE IN, AND WHY**

Assess first, then convert the schema and the code before any real data moves, because the data copy is the easier, repeatable part. Rehearse the load until the times are known, then switch the application in one agreed outage.

1. **Assessment and route choice per schema**
   Tells you the target, whether you need change data capture or can take an outage, and how much code needs rewriting by hand. That sets every later date.
2. **Oracle logging changes, if you chose change data capture**
   ARCHIVELOG mode and supplemental logging may need a DBA change window and a restart, so book them early.
3. **Target PostgreSQL, roles and extensions**
   Converted code must compile against the version and the extensions the real target allows, not a test server.
4. **Schema: tables, sequences and primary keys only**
   Data needs tables to land in. Other indexes, foreign keys and triggers slow the load and can make it fail, so they come later.
5. **Code conversion and side-by-side testing**

The longest piece of work. It runs against test loads while the data copy is rehearsed.

6 **Rehearsal: full load, then indexes, constraints and triggers, then validation**

Gives real times for the runsheet and proves the checks pass before the outage.

7 **Cutover: final load or catch-up, sequences, then connection strings**

Sequences can only be set after the last row lands, and the application switches only when counts match.

8 **Oracle kept read-only and licensed until the rollback window closes**

Pointing the application back at Oracle is the fastest rollback, and only works while Oracle is still there.

**SETTINGS CHECKLIST: DECIDE THESE BEFORE YOU COPY**

Built-in and third-party tools treat these differently. Tick each one off once you have decided.

Download this checklist (Excel)

- [ ] **Outage or change data capture**

  Whether the application can stop for the whole data copy, or you need change data capture (CDC: copying changes while the application keeps running) so the outage is only the final catch-up.

  ▼ How each tool handles it

  | | |
  |---|---|
  | **Built-in method** | AWS DMS: full load plus CDC from Oracle redo logs. CDC needs ARCHIVELOG mode and supplemental logging on Oracle. Google Database Migration Service: offers continuous migration from Oracle, and does not support DDL changes (changes to tables and other schema objects) during the migration. |
  | **Ora2Pg** | No CDC. You stop writes, export with COPY and load the files, so the application is down for the whole copy. |

  **Tip:** Time a test export of the largest schema first. If it does not fit the outage, you need a CDC route.

  AWS DMS: Using an Oracle database as a source

- [ ] **Which converter is the source of truth**

  Which tool generates the PostgreSQL schema you deploy, and where its output is kept.

▼ How each tool handles it

| | |
|---|---|
| **Built-in method** | AWS DMS Schema Conversion: converts schemas and most code objects, for Aurora and RDS for PostgreSQL targets only. Google Database Migration Service: conversion workspaces convert packages, functions, procedures and table triggers, with Gemini-assisted conversion, for Cloud SQL and AlloyDB. |
| **Ora2Pg** | Generates DDL for any PostgreSQL target, and its assessment report classes each schema by how much manual rewriting it needs. |
| **Visual Studio Code schema conversion (Azure)** | Converts the schema for Azure Database for PostgreSQL flexible server and compiles it against a scratch database. It generates scripts and does not move data. It drops and recreates matching schemas in the scratch database. |

**Tip:** Keep the chosen output in version control, and run the Ora2Pg assessment as a second opinion whichever converter you use.

AWS DMS: Converting database schemas using DMS Schema Conversion

- [ ] **Tables without a primary key**

Whether to add a key to each table that has none, or copy and check those tables another way.

▼ How each tool handles it

| | |
|---|---|
| **Built-in method** | Google Database Migration Service: migrates only tables that have a primary key. AWS DMS: data validation needs a primary key or unique index on every table. |
| **Ora2Pg** | TEST_COUNT compares row counts and TEST_DATA compares rows on both sides, whichever tool loaded the data. How TEST_DATA handles tables without a key: not stated in Ora2Pg's documentation: test it in your pilot. |

**Tip:** List keyless tables in Oracle before the first load, not after validation fails.

Google Cloud: Known limitations, Oracle to PostgreSQL

- [ ] **Data types, empty strings and dates**

Your type rules: NUMBER columns to integer, bigint or numeric; Oracle DATE to timestamp or date; and one rule for empty strings, which Oracle treats as NULL.

▼ How each tool handles it

| | |
|---|---|
| **Built-in method** | AWS DMS Schema Conversion: transformation rules apply your type choices every time you regenerate the DDL. Google Database Migration Service: does not support ANYDATA, BFILE, INTERVAL, LONG and LONG RAW, SDO_GEOMETRY, user-defined types, UROWID or XMLTYPE columns, and needs UTF8 encoding on the target. |
| **Ora2Pg** | Type mapping directives in its configuration make the same choices repeatable each time you regenerate the DDL. |

**Tip:** Map Oracle DATE to timestamp unless you have checked the column never holds a time.

Google Cloud: Known limitations, Oracle to PostgreSQL

- [ ] **LOB columns**

How to copy LOB columns (large text and binary values such as CLOB, BLOB and LONG): with what size limit, and in which task.

▼ **How each tool handles it**

| | |
|---|---|
| **Built-in method** | AWS DMS: limited LOB mode, the default, copies values up to the size you set (32 KB by default). Full LOB mode copies every value but is much slower. In limited LOB mode LONG and LONG RAW are cut to 64 KB, and with LogMiner on Oracle 12 LOB changes are not replicated. Google Database Migration Service: rows are limited to 100 MB, and LONG, LONG RAW and XMLTYPE are not supported. |

**Tip:** Measure the largest value in each LOB column, and move tables with very large values in their own task.

AWS DMS User Guide: Recommendations for migrations

- [ ] **Indexes, foreign keys and triggers during the load**

Which objects exist on the target before the load, and which are created after it.

▼ **How each tool handles it**

| | |
|---|---|
| **Built-in method** | AWS DMS: creates tables and primary keys only, not secondary indexes or foreign keys. AWS recommends turning off foreign keys and triggers for a full load, adding secondary indexes before the CDC phase, and enabling |

| | |
|---|---|
| | triggers right before cutover. With a pre-created schema, set target table preparation mode to Do nothing. |
| **Ora2Pg** | Microsoft's Ora2Pg guidance recommends exporting with COPY rather than INSERT, and loading without foreign keys, constraints and indexes. |

**Tip:** Apply the index, foreign key and trigger scripts as a timed step in the rehearsal.

AWS DMS User Guide: Recommendations for migrations

- [ ] **Sequence values**

Who sets each sequence (the counter that generates new key numbers) at cutover, and from which value.

▼ **How each tool handles it**

| | |
|---|---|
| **Built-in method** | AWS DMS: copies table rows, not current sequence values. Google Database Migration Service: migrates sequence values, but Oracle keeps advancing them until writes stop, so update them on the target after the final sync. |
| **Ora2Pg** | Export type SEQUENCE_VALUES writes the statements that set each sequence to its last Oracle value. Run it after writes stop. |

**Tip:** Check after cutover that every sequence is at or above the highest key it feeds.

Google Cloud: Known limitations, Oracle to PostgreSQL

- [ ] **Users, roles and permissions**

Which PostgreSQL roles you create for schema owners and applications, and which permissions each gets.

▼ **How each tool handles it**

| | |
|---|---|
| **Built-in method** | AWS DMS: does not create user accounts or copy privileges. Google Database Migration Service: users and permissions are not migrated. |
| **Ora2Pg** | Export type GRANT exports roles and privileges from Oracle for review. |

**Tip:** Create roles with lower-case, unquoted names before the first load, and grant only what each application needs.

AWS DMS User Guide: Recommendations for migrations

- [ ] **Extensions on a managed target**

  Which PostgreSQL extensions and helper schemas the converted code may use.

  ▼ **How each tool handles it**

  | | |
  |---|---|
  | **Built-in method** | AWS DMS Schema Conversion: can add an extension pack (helper functions that emulate Oracle features). Each managed service, including Amazon RDS and Aurora, Azure Database for PostgreSQL and Cloud SQL, allows only its own published list of extensions. |

  **Tip:** Check every extension the converted DDL creates against your target's supported list before you build on it.

  AWS DMS: Converting database schemas using DMS Schema Conversion

- [ ] **SQL inside the application**

  Who changes the SQL and database driver in each application, report and batch job, and how it is tested.

  ▼ **How each tool handles it**

  | | |
  |---|---|
  | **Built-in method** | AWS DMS and Google Database Migration Service convert and copy the database. Neither describes converting SQL that lives in your application code, so plan that as your own work. |
  | **AWS Schema Conversion Tool (application conversion)** | Converts SQL embedded in C#, C++, Java and Pro*C applications and SQL*Plus scripts, and lists what you must convert by hand. |

  **Tip:** Search the code for CONNECT BY, ROWNUM, DECODE, NVL, (+) outer joins and SYSDATE to size the work.

  AWS SCT: Converting application SQL

- [ ] **How you prove the data matches**

  Which checks you run before go-live: row counts, row contents, or totals of key columns.

  ▼ **How each tool handles it**

  | | |
  |---|---|
  | **Built-in method** | AWS DMS: built-in row-level validation, which needs a primary key or unique index on every table and fails on text keys unless their collation (sort order setting) is C. |

| | |
|---|---|
| **Ora2Pg** | TEST compares objects, TEST_COUNT compares row counts and TEST_DATA compares rows, whichever tool loaded the data. |

**Tip:** Also compare totals of key numeric and date columns, which catch type mapping mistakes that counts miss.

AWS DMS: Data validation

- [ ] **Rollback after go-live**

How long Oracle stays available, and how changes made in PostgreSQL would get back to it.

▼ **How each tool handles it**

| | |
|---|---|
| **Built-in method** | Once real data is written to PostgreSQL, going back means copying those changes to Oracle, with a reverse task you set up and test yourself or by re-entering them. Keep Oracle running and licensed until the rollback window closes. |

**Tip:** Agree the rollback window with the business before cutover, and do not book the Oracle decommission inside it.

## THE PLAN, PHASE BY PHASE

### 1 Assess — Before you set a date

List each schema with its size, object counts and the applications, reports and jobs that use it.

Run the Ora2Pg assessment report with a cost estimate on each schema.

On AWS, also run the DMS Schema Conversion assessment.

Time a test export of the largest schema to see whether it fits the outage the business will accept.

Record per schema the target, route, CDC or outage, and an owner for manual code work.

**Watch out:** Converters handle tables well, but packages, procedures, triggers and application SQL usually need rewriting by hand.

**WAYS AROUND IT**

- Plan the dates from the objects flagged for manual work, not from the object count.

**Watch out:** Ora2Pg has no CDC (change data capture, which keeps copying changes while the app runs), so the app is down for the whole copy.

**WAYS AROUND IT**

- Use a CDC route (AWS DMS or Google Database Migration Service), or date-filtered incremental loads for large tables.

**Watch out:** AWS DMS CDC needs Oracle in ARCHIVELOG mode with supplemental logging, which may need a DBA change window and a restart.

**WAYS AROUND IT**

- Run SELECT log_mode FROM v$database; now and book the change early.

**Watch out:** Google Database Migration Service migrates only tables that have a primary key.

**WAYS AROUND IT**

- List tables without a primary key now, and add keys or plan another way to copy them.

**Show the detailed steps (5) →**

## 2 Prepare PostgreSQL

Create the target at the PostgreSQL version you will run in production, plus a separate test database.

Place it where the migration server, and DMS if used, can reach it.

Create an owner role

per schema and the application login roles, keeping names lower case and unquoted.

Install only the extensions your platform supports.

> **Watch out:** Managed services allow only a published list of extensions, so converted code that works on a test server can fail there.
>
> **WAYS AROUND IT**
>
> - Check every extension and helper schema the converted DDL creates against your target's supported list.

**Show the detailed steps (3) →**

## 3 Convert the schema

Pick one converter as the source of truth and keep

its output in version control.

Set type rules first, such as NUMBER key columns to bigint where the data holds no decimals.

Map Oracle DATE to timestamp unless the column never holds a time.

Apply tables, sequences and primary keys

only. Hold back other indexes, foreign keys and triggers.

> **Watch out:** Oracle treats an empty string as NULL, its DATE holds a time, and it folds names to upper case. Results change without any error.
>
> **WAYS AROUND IT**
>
> - Agree one rule for empty strings, and keep PostgreSQL names unquoted and lower case.

> **Watch out:** The VS Code conversion drops and recreates matching schemas in its scratch database.
>
> **WAYS AROUND IT**
>
> - Point it only at an empty scratch database, never one with data you need.

**Show the detailed steps (5) →**

## 4 Convert and test the code

Log every object the tool flagged, with an owner and a status.

Rewrite each one in

PL/pgSQL (PostgreSQL's equivalent of PL/SQL) and compile it on the test database.

Change the application's SQL and database driver to PostgreSQL.

Review AI-converted code like any other converted code.

Run the same transactions, reports and batch jobs on both

databases and compare the results.

Fix differences in the code, not by editing data.

> **Watch out:** Finding Oracle-only SQL in the application by hand is slow and easy to get wrong.
>
> **WAYS AROUND IT**
>
> - Search the code for CONNECT BY, ROWNUM, DECODE, NVL, (+) outer joins and SYSDATE.
>
> **TOOLS THAT CAN DO IT**
>
> **AWS Schema Conversion Tool (application conversion)**
> Converts SQL embedded in C#, C++, Java and Pro*C applications and SQL*Plus scripts, and lists what you must convert by hand.

**Show the detailed steps (4) →**

## 5 Move the data

Rehearse first, then repeat for real

Outage route: stop writes, export with Ora2Pg COPY, and load the files

with psql.

CDC route: enable ARCHIVELOG and supplemental logging, then run a DMS full load and CDC task with validation on.

Set the DMS target table preparation mode to Do nothing, because the schema already exists.

After the load, create

secondary indexes, foreign keys and triggers, then run ANALYZE.

> **Watch out:** LOB columns (large text and binary values) have the most limits. In AWS DMS limited LOB mode, LONG values are cut to 64 KB. Google Database Migration Service does not support LONG, LONG RAW or XMLTYPE columns.
>
> **WAYS AROUND IT**
>
> - List LOB, LONG and XMLTYPE columns, measure the largest value, and size limited LOB mode to fit or move those tables in their own task.
> - Consider Binary Reader if you need LOB CDC on Oracle 12c.

**Show the detailed steps (6) →**

## 6 Validate and rehearse

Run Ora2Pg TEST on each schema to compare objects and row counts.

On AWS, check every table shows

Validated in the DMS Table statistics tab.

Compare totals and checksums of key numeric and date columns on both sides.

Rehearse the whole cutover on a copy and write the real times into the runsheet.

> **Watch out:** AWS DMS validation needs a primary key on every table, and fails on text keys unless their collation (sort order setting) is C.
>
> **WAYS AROUND IT**
>
> - Add keys, or set COLLATE "C" on text key columns in the converted DDL.

**TOOLS THAT CAN DO IT**

**Ora2Pg**
TEST_COUNT compares row counts, TEST compares objects, and TEST_DATA checks rows on both sides, whichever tool loaded the data.

**Show the detailed steps (3) →**

## 7 Cut over

In the agreed outage

- Stop the application and batch jobs, and confirm no sessions remain on Oracle.
- Wait for CDC latency to reach zero and stop the task, or run the final COPY export and load.

Set every sequence (the counter that generates new key numbers) to the Oracle value, and enable triggers.

Switch connection strings to PostgreSQL and run the smoke tests.

Lock the application users on Oracle, and hand backups and monitoring to operations.

**Watch out:** AWS DMS and an Ora2Pg COPY load copy rows, not current sequence values, so the first new inserts fail with duplicate key errors. Google Database Migration Service copies sequence values, but Oracle keeps advancing them until writes stop.

**WAYS AROUND IT**

- Set each sequence with setval() from the Oracle value or MAX() of the key.

**TOOLS THAT CAN DO IT**

**Ora2Pg**
Export type SEQUENCE_VALUES writes the statements that set each sequence to its last Oracle value. Run it after writes stop.

**Watch out:** Once real data is written to PostgreSQL, going back to Oracle means copying those changes back.

**WAYS AROUND IT**

- Agree a rollback window, and keep Oracle running and licensed until it closes.
- Decide in advance between a reverse DMS task and manual re-entry.

**Turn this phase into dated tasks with the cutover planner**

**Show the detailed steps (5) →**

### You are finished when

- Object and row counts match for every schema, and row contents or column totals match.
- Every sequence is at or above the highest key it feeds.
- UAT scripts, batch jobs and integrations pass against PostgreSQL.
- The application runs on PostgreSQL and no application sessions remain on Oracle.

Tools named in this plan are independent third-party products, not endorsed by the platform vendor and not paid placements. Check the tool's own documentation and run a pilot first.

## CLICK BY CLICK

# What does each step involve?

Open a part when you reach it in the plan. Each step says where to go, what to do and what you should see.

## About this migration

What it covers, limits at a glance

Moving from Oracle to PostgreSQL is a heterogeneous migration: the data types, procedural language, SQL dialect and many built-in packages differ, so the schema and code must be converted before any data moves. The data itself is the easier part. The work that decides the timeline is converting PL/SQL and the SQL embedded in the application, then testing that the converted code returns the same results.

This guide follows the order the work is done: assess, prepare the target, convert the schema, convert and test the code, move the data (one bulk load, or a full load followed by change data capture), validate, and cut over. It gives a target-neutral path with Ora2Pg and native PostgreSQL tools, and the cloud-native routes on AWS, Azure and Google Cloud.

If your source is SQL Server or MySQL rather than Oracle, see the database to Amazon RDS guide or the SQL Server to Azure SQL guide.

TARGET-NEUTRAL TOOL
**Ora2Pg (open source): assessment report, schema export, COPY data export and a TEST comparison**

AWS
**DMS Schema Conversion for Aurora PostgreSQL and RDS for PostgreSQL, then AWS DMS full load and CDC**

AZURE
**Schema conversion in the PostgreSQL extension for Visual Studio Code; Microsoft documents Ora2Pg for data**

GOOGLE CLOUD
**Database Migration Service conversion workspaces for Cloud SQL for PostgreSQL and AlloyDB**

CDC PREREQUISITE
**Oracle in ARCHIVELOG mode with supplemental logging**

HARDEST PART
**PL/SQL packages, procedures and triggers, plus the application SQL that calls them**

## 1 What you need before step one

| Need | Details | Where to get it |
| --- | --- | --- |
| Oracle access | A user that can read the dictionary for assessment. For CDC, AWS lists grants such as SELECT ANY TRANSACTION, LOGMINING (12c and later) and EXECUTE on DBMS_LOGMNR. | Your Oracle DBA. AWS DMS Oracle source page lists every grant. |
| Source change window | Turning on ARCHIVELOG mode or supplemental logging may need a DBA change and, for ARCHIVELOG, a restart. | Your change advisory board |
| PostgreSQL target | A server or managed instance at the version you will run in production, plus a separate test database. | Amazon RDS or Aurora console, Azure portal, Google Cloud console, or your own server |
| Migration server | A Linux or Windows host near the source with network access to both databases. Ora2Pg needs Perl, DBI and DBD::Oracle with an Oracle client. | Ora2Pg download and documentation: https://ora2pg.darold.net/ |
| AWS (AWS route) | IAM permissions for AWS DMS, a VPC that can reach Oracle, and AWS Secrets Manager secrets for both databases. | Install the AWS CLI; AWS DMS console (https://console.aws.amazon.com/dms/v2/) |
| Azure (Azure route) | Visual Studio Code 1.95.2 or later with the PostgreSQL extension (ms-ossdata.vscode-pgsql), a Microsoft Foundry model deployment, GitHub Copilot Pro+, Business or Enterprise, and an empty scratch database on flexible server version 15 or later. | Visual Studio Code Marketplace; Install the Azure CLI |

| Need | Details | Where to get it |
|---|---|---|
| Google Cloud (Google route) | A project with Database Migration Service enabled and a Cloud SQL for PostgreSQL or AlloyDB destination. | Install the gcloud CLI |
| psql client | To apply DDL files and run checks against PostgreSQL. | Your PostgreSQL distribution or package manager |
| Test plan | The queries, reports and batch jobs that prove the application works, with expected results captured from Oracle. | Test strategy and UAT test scripts templates |

## 2 Assess the Oracle schemas

5 steps

### 1 Inventory schemas, objects and sizes

**Where: SQL*Plus or SQL Developer connected to the source Oracle database as a DBA**

List the application schemas, count objects by type and measure segment sizes. Record the results in your **Application inventory and dependency map**, including which applications, reports and ETL jobs connect to each schema.

```
SELECT owner, object_type, COUNT(*) FROM dba_objects WHERE owner IN ('HR','SALES') GROUP BY owner,
object_type ORDER BY 1,2;
SELECT owner, ROUND(SUM(bytes)/1024/1024/1024,1) AS gb FROM dba_segments WHERE owner IN
('HR','SALES') GROUP BY owner;
SELECT * FROM v$version;
```

✓ **You should see:** A count per object type (TABLE, PACKAGE BODY, PROCEDURE, TRIGGER and so on) and a size in GB for each schema, plus the Oracle version.

2

## Create an Ora2Pg project

**Where: A shell on the migration server with Ora2Pg installed**

Create a project tree; Ora2Pg generates folders for each object type, a config file and an `export_schema.sh` script. Edit `config/ora2pg.conf` in the new folder and set `ORACLE_DSN`, `ORACLE_USER`, `ORACLE_PWD` and `SCHEMA` to the source. Set `PG_DSN`, `PG_USER` and `PG_PWD` to the test target so the TEST action can run later.

```
ora2pg --project_base /app/migration/ --init_project hr_project
cd /app/migration/hr_project
# config/ora2pg.conf (excerpt)
ORACLE_DSN  dbi:Oracle:host=ora01.contoso.com;service_name=ORCLPDB1;port=1521
ORACLE_USER ora2pg_reader
SCHEMA      HR
PG_DSN      dbi:Pg:dbname=hr;host=pgtest.contoso.com;port=5432
```

✓ **You should see:** /app/migration/hr_project contains config, schema, sources, data and reports folders and export_schema.sh.

3

## Run the Ora2Pg assessment report

**Where: The same shell, in /app/migration/hr_project**

Run the content report with a cost estimate. The output lists each object type, how many objects Ora2Pg can convert automatically, an estimated cost and a migration level (A, B or C). Run `SHOW_TABLE` and `SHOW_COLUMN` to see table sizes and the proposed column type mapping.

```
ora2pg -t SHOW_REPORT --estimate_cost -c config/ora2pg.conf > reports/assessment.txt
ora2pg -t SHOW_TABLE -c config/ora2pg.conf > reports/tables.txt
ora2pg -t SHOW_COLUMN -c config/ora2pg.conf > reports/columns.txt
```

✓ **You should see:** reports/assessment.txt ends with a migration level such as B-5 or C-3 and an estimated number of person-days.

4

### Run a DMS Schema Conversion assessment (AWS route)

**Where: AWS DMS console (https://console.aws.amazon.com/dms/v2/) > Instance profiles, Data providers and Migration projects**

Create an **instance profile** (network and security settings), a source **data provider** for Oracle and a target data provider for Aurora PostgreSQL or RDS for PostgreSQL, with credentials in AWS Secrets Manager. Then choose **Migration projects** > **Create migration project**, enter a **Name**, choose the instance profile, choose **Browse** for the **Source** and **Target**, pick each **Secret ID** and **IAM role**, and choose **Create migration project**. Open the project, start schema conversion, select the schemas and run the assessment. If no target exists yet, you can use a virtual target.

✓ **You should see:** The assessment report shows which objects convert automatically and which need manual action. Save the PDF or CSV with your project records.

5

### Agree scope and approach

**Where: Your Workload 7R decision matrix and Migration wave plan**

For each schema record: target platform, route (from **Choose your approach** above), CDC or outage, the objects that need manual rewrite and their owner, and the applications that must be retested. Drop schemas nobody uses rather than converting them.

✓ **You should see:** Each schema has a target, a route, a data method and a named owner for manual code work.

## 3 Prepare the PostgreSQL target

3 steps −

1

## Create the target and the test database

**Where: Your target platform: Amazon RDS console > Databases > Create database; Azure portal > Azure Database for PostgreSQL flexible servers > Create; Google Cloud console > SQL > Create instance; or your own server**

Choose the PostgreSQL major version you will run in production, size storage for the Oracle data plus indexes and growth, and place it in a network that the migration server (and DMS, if used) can reach. Create the application database and a separate scratch or test database for conversion work.

✓ **You should see:** You can connect with psql from the migration server: psql "host=pgtest.contoso.com dbname=hr user=pgadmin" returns a prompt.

2

## Create roles and schemas

**Where: psql connected to the target as the admin user**

Create an owner role per schema and the login roles the application uses, then create each schema owned by its role. Keep names lower case and unquoted.

```
CREATE ROLE hr_owner NOLOGIN;
CREATE ROLE hr_app LOGIN PASSWORD '<password>';
CREATE SCHEMA hr AUTHORIZATION hr_owner;
GRANT USAGE ON SCHEMA hr TO hr_app;
```

✓ **You should see:** \dn in psql lists schema hr owned by hr_owner.

3

### Enable the extensions the converted code needs

**Where: psql connected to the target, after checking the platform's supported extension list**

Install only extensions your platform supports. If you use DMS Schema Conversion, it can add its extension pack to emulate Oracle features. If you plan to use AWS DMS enhanced validation, AWS asks for the `pgcrypto` extension on PostgreSQL.

```
CREATE EXTENSION IF NOT EXISTS pgcrypto;
SELECT extname, extversion FROM pg_extension;
```

✓ **You should see:** pg_extension lists every extension the converted DDL references.

## 4 Convert the schema

5 steps

Use one converter as the source of truth and keep its output in version control. Regenerate rather than hand-edit where you can, so a late change in Oracle can be converted again.

1

### Export the schema with Ora2Pg (Ora2Pg or Azure route)

**Where: A shell on the migration server, in /app/migration/hr_project**

Run the generated script to export every object type into the `schema` folder, one file per type. Or export a single type with the second command, for example when you rerun only tables after changing type mappings.

```
./export_schema.sh
ora2pg -p -t TABLE -o table.sql -b ./schema/tables -c ./config/ora2pg.conf
```

✓ **You should see:** schema/tables/table.sql, schema/sequences, schema/views, schema/packages and the other folders contain .sql files.

2

## Convert with DMS Schema Conversion (AWS route)

**Where: AWS DMS console (https://console.aws.amazon.com/dms/v2/) > Migration projects > your project > schema conversion**

Add transformation rules for object names and data types first (for example `NUMBER` key columns to `bigint`). Select the schemas in the source pane and convert them. Compare the source and converted code side by side, edit converted SQL where needed, then apply the changes to the target database or export the converted code as SQL scripts to an Amazon S3 bucket.

✓ **You should see:** The converted objects appear in the target pane and, after you apply them, in the target database (\dt hr.* in psql lists the tables).

3

## Convert in Visual Studio Code (Azure route)

**Where: Visual Studio Code with the PostgreSQL extension > Migrations panel > Create Migration Project**

Create a connection to your flexible server in the PostgreSQL extension and open a new folder as the workspace. In the **Migrations** panel select **Create Migration Project**, name it, and enter the Oracle host, port, service name and credentials. Select **Load Schemas**, choose schemas and select **Next: Discovery**. Review the results, tick **I have reviewed the objects, exclusions and dependent schemas**, pick the scratch database, select **Verify Extensions**, configure the Microsoft Foundry endpoint and select **Create Migration Project**, then **Migrate**. The tool drops and recreates matching schemas in the scratch database, so never point it at a database with data you need.

✓ **You should see:** The project shows Schema conversion is complete. Review conversion summary and Conversion object explorer show converted and not-converted objects; Setup instruction gives the .sql files to deploy.

4

## Convert in a conversion workspace (Google Cloud route)

**Where: Google Cloud console > Database Migration > Conversion workspaces**

Create a conversion workspace (not the legacy type) for Oracle to your PostgreSQL destination and connect it to the source. Review the initial rules-based conversion, use the Gemini-assisted suggestions and the SQL editor for code objects, and add conversion mapping files where you need different type rules.

✓ **You should see:** The workspace lists each object as converted or with issues, and you can apply the converted schema to the destination.

5

## Apply tables and primary keys only

**Where: psql on the migration server, connected to the target database**

Apply table, sequence and primary key DDL now. Hold back secondary indexes, foreign keys and triggers until the data is loaded (Step 5). Ora2Pg can write indexes and constraints to separate files (see the output options in its configuration documentation); otherwise move those statements into their own file. Split DMS Schema Conversion or VS Code output the same way.

```
psql "host=pgtest.contoso.com dbname=hr user=pgadmin" -f schema/tables/table.sql
psql "host=pgtest.contoso.com dbname=hr user=pgadmin" -f schema/sequences/sequence.sql
```

✓ **You should see:** psql finishes with no ERROR lines and \dt hr.* lists every table from the inventory.

**Generative AI conversion.** AWS DMS Schema Conversion offers generative AI conversion for Oracle to Aurora PostgreSQL and RDS for PostgreSQL, and the Microsoft and Google tools use AI models too. Treat AI-converted code like any other converted code: review it and test it against expected results.

## 5 Convert and test PL/SQL and application SQL

4 steps

### 1 List the objects that need manual work

**Where: The Ora2Pg assessment report, the DMS Schema Conversion assessment or the VS Code conversion summary; record them in your Issue and decision log**

Copy every object the tool flagged as not converted or converted with warnings into the log with an owner. Typical items: packages with global state, autonomous transactions, `DBMS_` and `UTL_` package calls, `CONNECT BY` hierarchies, dynamic SQL, database links and scheduler jobs.

✓ **You should see:** Every flagged object has a row, an owner and a status in the log.

### 2 Rewrite and compile each object

**Where: Your code editor and psql connected to the test database**

Rewrite each object in PL/pgSQL. Common rewrites: `NVL` to `COALESCE`, `DECODE` to `CASE`, `SYSDATE` to `LOCALTIMESTAMP` or `now()`, `CONNECT BY` to `WITH RECURSIVE`, `ROWNUM` to `LIMIT` or `row_number()`, and package state to a schema plus session settings or tables. Compile with psql and keep each object in version control.

```
psql "host=pgtest.contoso.com dbname=hr user=pgadmin" -f schema/packages/emp_pkg.sql
psql "host=pgtest.contoso.com dbname=hr user=pgadmin" -c "\df hr.*"
```

✓ **You should see:** Each file compiles with no ERROR lines and \df hr.* lists the converted functions and procedures.

3

### Convert the SQL in the application

**Where: The application source code repository and a test build of the application**

Search the application for Oracle-specific SQL, hints, outer join `(+)` syntax, bind variable styles and driver settings, and change the connection driver to PostgreSQL. The VS Code PostgreSQL extension also offers application conversion of SQL scripts, loader control files, shell scripts and Java (Microsoft labels this a preview feature).

✓ **You should see:** The application builds against the PostgreSQL driver and starts against the test database.

4

### Test results against Oracle

**Where: Your UAT test scripts, run against Oracle and against the PostgreSQL test database with the same data**

Run the same business transactions, reports and batch jobs on both sides and compare outputs: totals, row counts, rounding, sort order, NULL handling and date arithmetic. Record differences and fix them in the code, not by editing data.

✓ **You should see:** Every test script returns the same results on both databases, or each difference is explained and signed off.

## 6 Move the data

6 steps –

Use Step 5a for a one-off copy inside an outage, or Step 5b for a full load followed by change data capture on AWS. Google Cloud Database Migration Service offers a continuous migration job for Oracle sources configured as its documentation describes.

1

## 5a. Export data with Ora2Pg COPY

**Where: A shell on the migration server, in /app/migration/hr_project**

Export data as COPY statements rather than INSERT; COPY loads much faster in PostgreSQL. For a rehearsal, run it while the application is live; for the real move, stop application writes to Oracle first.

```
ora2pg -t COPY -o data.sql -b ./data -c ./config/ora2pg.conf
```

✓ **You should see:** The data folder contains data.sql (or one file per table) and the command finishes without errors.

2

## 5a. Load the data and build indexes

**Where: psql on the migration server, connected to the target**

Load the data files, then apply the held-back secondary indexes, foreign keys and triggers, and run `ANALYZE` so the planner has statistics. You can load several table files in parallel sessions to save time.

```
psql "host=pgtest.contoso.com dbname=hr user=pgadmin" -f data/data.sql
psql "host=pgtest.contoso.com dbname=hr user=pgadmin" -f schema/tables/indexes-and-constraints.sql
psql "host=pgtest.contoso.com dbname=hr user=pgadmin" -c "ANALYZE;"
```

✓ **You should see:** psql finishes with no ERROR lines; foreign keys create without violations.

3

## 5b. Enable ARCHIVELOG and supplemental logging (CDC)

**Where: SQL*Plus on the source Oracle database as SYSDBA (for RDS for Oracle, the rdsadmin procedures in the AWS page)**

Check the log mode and minimal supplemental logging, enable database-level supplemental logging, and add primary key logging. For tables without a primary key, AWS gives table-level statements for a unique index or ALL columns.

```
SELECT log_mode FROM v$database;
SELECT supplemental_log_data_min FROM v$database;
ALTER DATABASE ADD SUPPLEMENTAL LOG DATA;
ALTER DATABASE ADD SUPPLEMENTAL LOG DATA (PRIMARY KEY) COLUMNS;
```

✓ **You should see:** log_mode returns ARCHIVELOG and supplemental_log_data_min returns YES or IMPLICIT.

4

## 5b. Create the DMS user and grants

**Where: SQL*Plus on the source Oracle database as a DBA**

Create a dedicated user and grant the privileges AWS lists for your CDC method. The excerpt below covers the core and LogMiner grants; the AWS page lists the remaining dictionary views. Add `EXECUTE ON SYS.DBMS_CRYPTO` if you will use data validation.

```
CREATE USER dms_user IDENTIFIED BY "<password>";
GRANT CREATE SESSION TO dms_user;
GRANT SELECT ANY TRANSACTION TO dms_user;
GRANT SELECT ON V_$DATABASE TO dms_user;
GRANT SELECT ON V_$ARCHIVED_LOG TO dms_user;
GRANT EXECUTE ON DBMS_LOGMNR TO dms_user;
GRANT SELECT ON V_$LOGMNR_CONTENTS TO dms_user;
GRANT LOGMINING TO dms_user;
GRANT EXECUTE ON SYS.DBMS_CRYPTO TO dms_user;
```

✓ **You should see:** Each GRANT returns Grant succeeded.

5

## 5b. Create the replication instance and endpoints

**Where: AWS DMS console (https://console.aws.amazon.com/dms/v2/) > Replication instances > Create replication instance; then Endpoints > Create endpoint**

Create a replication instance (or use DMS Serverless) in a subnet that reaches both databases. Create an Oracle **Source endpoint** and a PostgreSQL **Target endpoint**, then test each connection from the replication instance. LogMiner is the default CDC method; for high change volumes or RAC, AWS recommends Binary Reader with the extra connection attributes `useLogMinerReader=N;useBfile=Y;`.

```
aws dms create-endpoint --endpoint-identifier src-hr-oracle --endpoint-type source --engine-name
oracle --server-name ora01.contoso.com --port 1521 --database-name ORCLPDB1 --username dms_user --
password "<password>"
aws dms test-connection --replication-instance-arn <replication-instance-arn> --endpoint-arn
<source-endpoint-arn>
```

✓ **You should see:** Endpoints > each endpoint > Connections shows Status successful.

6

### 5b. Run full load and CDC with validation

**Where: AWS DMS console (https://console.aws.amazon.com/dms/v2/) > Database migration tasks > Create task**

Choose the instance and endpoints and **Migrate existing data and replicate on-going changes**. Set the target table preparation mode to **Do nothing** because the schema already exists, pick a LOB mode from your LOB inventory, add selection rules for each schema, and turn on data validation. Create the secondary indexes and foreign keys after the full load completes and before cutover.

```
aws dms create-replication-task --replication-task-identifier hr-full-cdc --source-endpoint-arn
<source-endpoint-arn> --target-endpoint-arn <target-endpoint-arn> --replication-instance-arn
<replication-instance-arn> --migration-type full-load-and-cdc --table-mappings file://table-
mappings.json --replication-task-settings '{"ValidationSettings":{"EnableValidation":true}}'
aws dms start-replication-task --replication-task-arn <task-arn> --start-replication-task-type
start-replication
```

✓ **You should see:** The task shows Load complete, replication ongoing, and the Table statistics tab shows each table with ValidationState Validated.

**Ora2Pg does not do CDC.** Microsoft's Ora2Pg guidance suggests date-filtered incremental loads for large tables and verifying parity before cutover. If you need near-zero downtime on Azure or self-managed PostgreSQL, evaluate a CDC-capable tool before you commit to a date.

## 7 Validate the data and rehearse

3 steps

1

## Compare objects and row counts with Ora2Pg TEST

**Where: A shell on the migration server, in /app/migration/hr_project (PG_DSN set in the config)**

Run the TEST action. It reports counts of indexes, constraints, triggers, views, sequences (with current values), functions and rows per table on both sides, then an error section that explains each difference. It tests one schema per run.

```
ora2pg -t TEST -c config/ora2pg.conf > migration_diff.txt
```

✓ **You should see:** migration_diff.txt shows matching counts for every object type and table, and the error section is empty or every entry is explained.

2

## Check DMS validation results (AWS route)

**Where: AWS DMS console (https://console.aws.amazon.com/dms/v2/) > Database migration tasks > your task > Table statistics, and psql on the target**

Every table should show **Validated**. For **Mismatched records**, query the failures table on the target to see the key and the differing columns. Tables without a key show **No primary key** and need another check.

```
aws dms describe-table-statistics --replication-task-arn <task-arn>
SELECT * FROM awsdms_control.awsdms_validation_failures_v1 WHERE "TASK_NAME" = '<task resource
id>';
```

✓ **You should see:** All tables Validated, and the failures table has no rows for the task (or every row is explained).

3

### Rehearse the cutover end to end

**Where: A non-production copy, following your Cutover runsheet**

Run the whole cutover on a copy: stop writes, final load or CDC catch-up, set sequences, run validation, switch the application connection, run smoke tests. Time each step and update the runsheet with the real durations.

✓ **You should see:** The rehearsal completes inside the agreed outage and the runsheet has measured times for every step.

## 8 Cut over

5 steps

1

### Stop writes to Oracle

**Where: The application servers (maintenance mode or stopped services) and SQL*Plus on Oracle**

Put the application into maintenance mode and stop batch jobs and integrations that write to Oracle. Confirm no application sessions remain.

```
SELECT username, machine, program FROM v$session WHERE username IN ('HR_APP','SALES_APP');
```

✓ **You should see:** The query returns no application sessions.

2

## Finish the data move

**Where: AWS DMS console > Database migration tasks (CDC route), or the migration server shell (Ora2Pg route)**

For CDC, wait until the task's CDC latency reaches zero, then stop the task. For Ora2Pg, run the final COPY export and load (Step 5a). Then rerun the TEST action or a full load validation only task in DMS.

```
aws dms stop-replication-task --replication-task-arn <task-arn>
```

✓ **You should see:** The task shows Stopped with no pending changes, and the final validation matches.

3

## Set sequences and enable triggers

**Where: psql connected to the target as the schema owner**

Set every sequence to the Oracle current value or the column maximum, and enable any triggers you disabled for the load.

```
SELECT setval('hr.employees_seq', (SELECT MAX(employee_id) FROM hr.employees));
ALTER TABLE hr.employees ENABLE TRIGGER ALL;
```

✓ **You should see:** SELECT last_value FROM hr.employees_seq; returns a value at or above the highest key.

4

### Switch the application to PostgreSQL

**Where: Application configuration files or secrets store, and your DNS alias if the application uses one**

Change connection strings (or the DNS alias) to the PostgreSQL endpoint, start the application and run your smoke tests from the runsheet. Keep Oracle running and unchanged.

✓ **You should see:** Smoke tests pass, and pg_stat_activity on the target shows the application's connections.

5

### Set Oracle read-only and hand over

**Where: SQL*Plus on the source Oracle database, then your Handover to operations document**

Lock or revoke write access for the application users on Oracle so nothing writes there by mistake, and keep it for the rollback window. Hand over backups, monitoring and the PostgreSQL runbook to operations.

```
ALTER USER hr_app ACCOUNT LOCK;
```

✓ **You should see:** The application user cannot connect to Oracle, and operations has accepted the PostgreSQL service.

WORTH KNOWING

## What can go wrong, and how do you avoid it?

The detail behind the plan: options compared, and what experienced engineers check first. Tap any item.

## Compare all the ways to do this

4 options –

### Ora2Pg and native PostgreSQL tools (any target)

HIGH EFFORT

**Best for:** Self-managed PostgreSQL, any cloud or on-premises target, and teams that want full control of the generated DDL

+ Works with any PostgreSQL target
+ Detailed assessment report with an effort estimate
+ TEST mode compares object and row counts on both sides
+ Open source
– No built-in CDC, so plan an outage for the data copy or a date-filtered incremental approach
– Perl and Oracle client set-up on a migration server
– Command line and config file driven

Read more →

### AWS DMS Schema Conversion plus AWS DMS (Aurora or RDS for PostgreSQL)

MEDIUM EFFORT

**Best for:** Targets on Aurora PostgreSQL or RDS for PostgreSQL, especially when downtime must be short

+ Managed assessment and conversion in the console, with generative AI conversion for Oracle to PostgreSQL
+ Full load plus CDC from Oracle redo logs
+ Built-in row-level data validation
– Schema Conversion targets only Aurora and RDS
– Oracle needs ARCHIVELOG and supplemental logging
– LOB, LONG and XMLTYPE limitations to plan around

Read more →

### Visual Studio Code schema conversion plus Ora2Pg data (Azure Database for PostgreSQL)

HIGH EFFORT

**Best for:** Targets on Azure Database for PostgreSQL flexible server that can accept an outage for the data copy

+ Schema conversion built into Microsoft's PostgreSQL extension for VS Code
+ Converted objects compiled against a scratch database before you deploy
+ Microsoft documents the Ora2Pg path for data
– Needs a Microsoft Foundry model deployment and a GitHub Copilot subscription
– Generates scripts; it does not move data or deploy for you
– Microsoft's migration service documentation lists PostgreSQL sources, not Oracle, so plan the data copy with Ora2Pg

Read more →

### Google Cloud Database Migration Service (Cloud SQL or AlloyDB for PostgreSQL)

MEDIUM EFFORT

**Best for:** Targets on Cloud SQL for PostgreSQL or AlloyDB

+ Conversion workspaces with Gemini-assisted conversion and an SQL editor
+ Converts packages, functions, procedures and table triggers
+ Continuous migration option from Oracle redo logs
– Google Cloud targets only
– Legacy conversion workspaces lack the newer features, so create the current type

Read more →

**OUR ADVICE**

Pick the route that matches your target first: AWS DMS for Aurora or RDS, Google's Database Migration Service for Cloud SQL or AlloyDB. For Azure or self-managed PostgreSQL, use Ora2Pg for data, and on Azure add the VS Code schema conversion for the code. Whichever converter you use, run the Ora2Pg assessment as well: a second opinion on the hardest objects costs an afternoon. If the data copy cannot fit in your outage, you need a CDC-capable route.

### Size the code conversion before you promise a date

SAVES TIME

Table DDL converts almost mechanically. Packages, procedures, triggers and the SQL inside the application are where the hours go, and a schema with a few large packages can take longer than one with hundreds of simple tables. Ora2Pg classes a migration as A (can run automatically), B (code rewrite up to 5 person-days) or C (code rewrite above 5 person-days).

**WHAT TO DO**

Run `ora2pg -t SHOW_REPORT --estimate_cost` against each schema (Step 1) and, on AWS, the DMS Schema Conversion assessment report. Plan the project from the objects that need manual work, not the object count. Add the application code: grep it for Oracle-only syntax such as `CONNECT BY`, `ROWNUM`, `DECODE`, `NVL`, `(+)` outer joins and `SYSDATE`.

**TOOLS THAT CAN GET AROUND THIS**

Instead of grepping application code by hand, AWS SCT can find the SQL embedded in the application, convert it for PostgreSQL and report the statements it could not convert.

**AWS Schema Conversion Tool (application conversion)**

For Oracle to PostgreSQL, converts SQL embedded in C#, C++, Java and Pro*C applications and SQL*Plus scripts to PSQL. Works on a local copy and lists items you must convert by hand in an assessment report.

Third-party tools, not endorsed by the platform vendor. Confirm in the tool's documentation and run a pilot first.

Ora2Pg: Migration cost assessment

### Decide early whether you need CDC or can take an outage

USER IMPACT

A one-off bulk export and load is simple but the application is down for the whole copy. A full load followed by change data capture keeps the target in sync while you test, so the outage shrinks to the final catch-up. CDC needs ARCHIVELOG mode and supplemental logging on Oracle, which your DBA may need a change window to turn on.

**WHAT TO DO**

Time a test export of the largest schema. If it does not fit inside the outage the business will accept, plan CDC with AWS DMS or Google Cloud Database Migration Service. Check `SELECT log_mode FROM v$database;` now so ARCHIVELOG is not a surprise in week six.

AWS DMS: Using an Oracle database as a source

### Treat empty strings, dates and case as data-changing differences

REDUCES RISK

Oracle treats an empty string as NULL; PostgreSQL does not. Oracle `DATE` carries a time of day; PostgreSQL `date` does not. Oracle folds unquoted names to upper case and PostgreSQL to lower case. Each of these changes query results without raising an error, which is why they slip through testing.

**WHAT TO DO**

Map Oracle `DATE` to `timestamp` unless you have checked the column never holds a time. Search the code for `= ''` and `IS NULL` tests on text columns and agree one rule. Keep object names unquoted and lower case in PostgreSQL so the application does not need quoted identifiers everywhere.

### Map NUMBER columns deliberately

PERFORMANCE

An Oracle `NUMBER` without precision can hold integers or decimals. Converting every one to PostgreSQL `numeric` is safe but slower for keys and joins; converting to `bigint` is fast but fails or truncates if a decimal ever turns up.

**WHAT TO DO**

For key and foreign key columns, check the data (for example `SELECT COUNT(*) FROM orders WHERE order_id <> TRUNC(order_id);` ) and map to `integer` or `bigint` where it returns zero. Use transformation rules in DMS Schema Conversion, or the type mapping directives in Ora2Pg, so the choice is repeatable each time you regenerate the DDL.

### Set sequences after the last data load

REDUCES RISK

Data tools copy table rows, not the current value of each sequence. If the PostgreSQL sequence still starts at 1 when the application goes live, the first inserts fail with duplicate key errors.

**WHAT TO DO**

Ora2Pg's `TEST` report lists sequences with their current values on both sides. At cutover, after writes stop and the final load or CDC catch-up finishes, set each sequence from the Oracle value (or from `MAX()` of the key) with `SELECT setval('hr.employees_seq', 120345);` .

**TOOLS THAT CAN GET AROUND THIS**

Ora2Pg can generate the statements that set each PostgreSQL sequence to the last Oracle value, so you do not have to write them by hand at cutover.

**Ora2Pg**

Export type SEQUENCE_VALUES exports DDL to set the last values of sequences. Run it after writes stop on Oracle.

Third-party tools, not endorsed by the platform vendor. Confirm in the tool's documentation and run a pilot first.

Ora2Pg: Test the migration

## Plan LOB columns separately

SAVES TIME

Large objects slow every tool down and have the most limitations. AWS documents, for example, that with LogMiner on Oracle 12 changes to LOB columns are not replicated, that LONG and LONG RAW are truncated to 64 KB in limited LOB mode, and that empty BLOB or CLOB values arrive as NULL in limited LOB mode.

**WHAT TO DO**

List LOB, LONG and XMLTYPE columns with `SELECT owner, table_name, column_name, data_type FROM dba_tab_columns WHERE data_type IN ('CLOB','BLOB','NCLOB','LONG','LONG RAW','XMLTYPE') AND owner = 'HR';` . Measure the largest value per column, then choose limited LOB mode with a size that fits, or move those tables in their own task. Consider Binary Reader if you need LOB CDC on Oracle 12c.

AWS DMS: Using an Oracle database as a source

## Use the C collation on key columns if you validate with AWS DMS

REDUCES RISK

AWS states that if the collation of the primary key column in a PostgreSQL target is not "C", the sort order differs from Oracle and DMS data validation fails to validate the records. Validation also needs a primary key or unique index on every table.

**WHAT TO DO**

Before the full load, list tables without a primary key in Oracle ( `SELECT table_name FROM dba_tables t WHERE owner = 'HR' AND NOT EXISTS (SELECT 1 FROM dba_constraints c WHERE c.owner = t.owner AND c.table_name = t.table_name AND c.constraint_type = 'P');` ) and add keys or plan another check for them. Set `COLLATE "C"` on text key columns in the converted DDL, or accept that you will validate those tables another way.

**TOOLS THAT CAN GET AROUND THIS**

Where DMS data validation fails on collation or was not used, Ora2Pg can compare the Oracle and PostgreSQL databases directly, whichever tool loaded the data. Check how it handles tables without a key on a test schema first.

**Ora2Pg**

Export type TEST_COUNT compares row counts per table, TEST compares objects between Oracle and PostgreSQL, and TEST_DATA runs a data validation check on rows on both sides.

Third-party tools, not endorsed by the platform vendor. Confirm in the tool's documentation and run a pilot first.

AWS DMS: Data validation

## Load first, then build indexes, constraints and triggers

SAVES TIME

Loading into tables that already have foreign keys, secondary indexes and triggers is slower and can fail on load order. Microsoft's Ora2Pg guidance recommends exporting with COPY rather than INSERT and avoiding foreign keys, constraints and indexes during the data load.

**WHAT TO DO**

Apply tables and primary keys first, load the data, then create secondary indexes, foreign keys and triggers. With Ora2Pg, export `TABLE` and then the `COPY` data before you run the index and constraint files. With AWS DMS, pre-create the schema and set the target table preparation mode to **Do nothing** so DMS does not recreate tables.

Microsoft Learn: Migrate Oracle to Azure Database for PostgreSQL using Ora2Pg

## Check every extension the converted code needs is allowed on the target

REDUCES RISK

Converters emulate Oracle features with helper functions. DMS Schema Conversion can add an extension pack; other tools rely on PostgreSQL extensions. Managed services only allow a published list of extensions, so code that works on a self-managed test server can fail on the managed target.

**WHAT TO DO**

List the extensions and helper schemas the converted DDL creates, and check each against your target's supported extension list (Amazon RDS and Aurora, Azure Database for PostgreSQL and Cloud SQL each publish one) before you build on them.

AWS DMS: Converting database schemas using DMS Schema Conversion

### Keep Oracle running, licensed and read-only for a while

COST LEVER

The fastest rollback is pointing the application back at Oracle. That only works if Oracle is still there, still licensed and has every change made since cutover, or you have agreed that changes in PostgreSQL after go-live are re-entered by hand.

**WHAT TO DO**

Agree a rollback window with the business, keep the Oracle licence and support until it closes, and decide in advance how post-cutover changes get back to Oracle (a reverse replication task or manual re-entry). Do not schedule Oracle decommission inside the window.

FINISH SAFELY

## How do you check it worked, or roll back?

### Full validation checklist

8 checks

Run these before you declare the migration complete. Record the results in the data validation report.

| Check | How to check | What good looks like |
|---|---|---|
| **Every object exists on the target** | Run `ora2pg -t TEST -c config/ora2pg.conf > migration_diff.txt` for each schema and compare counts of tables, views, sequences, indexes, constraints, triggers and functions. | Counts match for every object type, or each difference is an object you deliberately dropped and logged. |
| **Row counts match per table** | Use the row count section of the Ora2Pg TEST report, or `aws dms describe-table-statistics` (FullLoadRows plus inserts and deletes) for DMS tasks. | Identical row counts for every table after writes stopped. |

| Check | How to check | What good looks like |
|---|---|---|
| **Row contents match** | On AWS, check **Table statistics** for ValidationState and query `awsdms_control.awsdms_validation_failures_v1`. Elsewhere, compare totals and checksums of key numeric and date columns per table with the same query on both sides. | All tables Validated with no failure rows, or column totals identical on both databases. |
| **Sequences are ahead of the data** | For each sequence, compare `SELECT last_value FROM hr.employees_seq;` with `SELECT MAX(employee_id) FROM hr.employees;`. | Every sequence value is equal to or higher than the maximum key it feeds. |
| **Converted code returns the same results** | Run the UAT test scripts and key reports against PostgreSQL and compare with the outputs captured from Oracle. | All scripts signed off in the UAT test scripts and sign-off template. |
| **Application connects and performs** | Check `SELECT usename, application_name, count(*) FROM pg_stat_activity GROUP BY 1,2;` and compare response times of the top queries with the Oracle baseline. | All application services connected and key transactions within the agreed performance target. |
| **Batch jobs and integrations run** | Trigger each scheduled job, ETL feed and reporting extract from the inventory against PostgreSQL and check its log and output. | Every job completes on schedule and downstream systems receive their data. |
| **Oracle no longer receives writes** | On Oracle, run `SELECT username, program FROM v$session WHERE username IN ('HR_APP','SALES_APP');` and check the application users are locked. | No application sessions on Oracle after cutover. |

## Rollback plan

If it goes wrong —

**ROLL BACK IF**

- Data validation shows mismatches that cannot be explained or fixed inside the outage
- Smoke tests or a critical business transaction fail on PostgreSQL
- Performance of key transactions is outside the agreed limit and cannot be fixed quickly
- A batch job or integration that the business cannot run without fails on PostgreSQL

**POINT OF NO RETURN**

Rollback is simple until the application writes real business data to PostgreSQL. After that, going back to Oracle means copying those changes back (a reverse replication task or manual re-entry). Once the rollback window closes and Oracle is decommissioned or its licence lapses, rollback means a new migration in the other direction.

1

## Stop the application

**Where: The application servers (maintenance mode or stopped services)**

Put the application into maintenance mode so no more changes reach PostgreSQL. Note the time.

✓ **You should see:** pg_stat_activity shows no application sessions on the target.

2

## Decide what happens to changes made in PostgreSQL

**Where: psql on the target and your Rollback plan**

If users already wrote data to PostgreSQL, export the affected rows (for example rows with a created or modified time after cutover) so they can be re-entered or loaded back into Oracle. If you set up a reverse AWS DMS task in advance (PostgreSQL source, Oracle target), let it catch up and stop it.

✓ **You should see:** A list or file of post-cutover changes, signed off by the business owner, or a stopped reverse task with zero latency.

3

## Unlock Oracle

**Where: SQL*Plus on the source Oracle database as a DBA**

Unlock the application users and re-enable any jobs you disabled.

```
ALTER USER hr_app ACCOUNT UNLOCK;
```

✓ **You should see:** The application user can connect to Oracle.

4

## Point the application back at Oracle

**Where: Application configuration files or secrets store, and your DNS alias**

Restore the Oracle connection strings or alias and the Oracle driver configuration from the pre-cutover backup, start the application and run the smoke tests.

✓ **You should see:** Smoke tests pass against Oracle and v$session shows the application connections.

5

### Record and reschedule

**Where: Your Issue and decision log and Migration wave plan**

Record the cause, keep the PostgreSQL target for diagnosis, fix the issue and plan a new cutover date. Drop and reload the target data before the next attempt.

✓ **You should see:** A logged root cause and a new date approved by the business owner.

Agree the triggers and who decides before the go / no-go meeting. Use the rollback plan template.

### Common problems and fixes

7 errors

`ORA-01031: insufficient privileges (DMS or Ora2Pg)`

**Cause:** The migration user lacks a dictionary view or LogMiner grant.

**Fix:** Grant the privileges listed on the AWS DMS Oracle source page (or the Ora2Pg docs for read access) and test the endpoint again.

`DMS task error: supplemental logging is not enabled`

**Cause:** Database-level or table-level supplemental logging is missing.

**Fix:** Run ALTER DATABASE ADD SUPPLEMENTAL LOG DATA and add PRIMARY KEY logging, or table-level log groups for tables without a key, then resume the task.

`duplicate key value violates unique constraint after go-live`

**Cause:** Sequences still start from their initial value on PostgreSQL.

**Fix:** Set each sequence with setval() from the Oracle value or MAX() of the key column.

`Table shows No primary key in DMS Table statistics`

**Cause:** DMS validation needs a primary key or unique index.

**Fix:** Add a key in Oracle before the load, or validate that table with Ora2Pg TEST counts and your own checksums.

`Mismatched records on text key columns`

**Cause:** PostgreSQL collation on the key column sorts differently from Oracle.

**Fix:** Use COLLATE "C" on text key columns, as AWS advises, then revalidate the table from the Table statistics tab.

`LOB columns arrive empty or truncated`

**Cause:** Limited LOB mode with a maximum size smaller than the data, or LogMiner on Oracle 12 not replicating LOB changes.

**Fix:** Increase the limited LOB size, move large LOB tables to a separate task, or use Binary Reader as AWS documents.

`function does not exist (for example nvl or decode)`

**Cause:** Application or converted code still calls an Oracle function.

**Fix:** Rewrite the call (COALESCE, CASE) or install the emulation the converter provides, if your platform allows it.

Search the full error library →

FAQ

# Frequently asked questions

### What is the best tool to migrate Oracle to PostgreSQL? −

It depends on the target. AWS DMS Schema Conversion and AWS DMS suit Aurora and RDS, Database Migration Service suits Cloud SQL and AlloyDB, and Ora2Pg works with any PostgreSQL. Many teams run Ora2Pg's assessment whichever converter they use.

### Can I migrate Oracle to PostgreSQL with no downtime?

Near-zero downtime needs change data capture: a full load followed by ongoing replication from Oracle redo logs, as AWS DMS and Google's continuous migration jobs provide. Ora2Pg on its own copies data in a single pass.

### Does PL/SQL convert automatically?

Partly. Converters handle common syntax, but packages with state, autonomous transactions, Oracle-supplied packages and dynamic SQL usually need manual rewriting. The assessment reports show which objects need work.

### Does AWS DMS migrate stored procedures?

No. AWS DMS moves data. Schema and code conversion is done by DMS Schema Conversion (or AWS SCT), and then applied to the target.

### Can Azure Database Migration Service move Oracle to Azure Database for PostgreSQL?

Microsoft's migration service documentation lists PostgreSQL sources only. Microsoft documents Ora2Pg for Oracle data and the VS Code PostgreSQL extension for schema conversion.

### How do I prove the data is the same?

Compare row counts and objects with Ora2Pg TEST, use AWS DMS data validation for row-level comparison where you can, and run your business reports on both databases.

SOURCES

## Official sources

### 11 official pages this guide is based on

Vendors change limits and features, so confirm details on the source before you rely on them. Last checked 4 Nov 2026.

- AWS DMS: Converting database schemas using DMS Schema Conversion
- AWS DMS: Create a migration project in DMS Schema Conversion
- AWS DMS: Using an Oracle database as a source
- AWS DMS: Data validation
- Ora2Pg: Migration cost assessment
- Ora2Pg: Test the migration
- Microsoft Learn: Migrate Oracle to Azure Database for PostgreSQL using Ora2Pg
- Microsoft Learn: Oracle to PostgreSQL schema conversion tutorial (VS Code)
- Microsoft Learn: Oracle migration in the PostgreSQL extension for VS Code
- Microsoft Learn: Migration service in Azure Database for PostgreSQL
- Google Cloud: Conversion workspaces (Oracle to PostgreSQL)