

---

# AWS Database Migration Service

## Step-by-Step Migration Guide

API Version 2016-01-01

---

## **AWS Database Migration Service: Step-by-Step Migration Guide**

Copyright © 2021 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon's trademarks and trade dress may not be used in connection with any product or service that is not Amazon's, in any manner that is likely to cause confusion among customers, or in any manner that disparages or discredits Amazon. All other trademarks not owned by Amazon are the property of their respective owners, who may or may not be affiliated with, connected to, or sponsored by Amazon.

## Table of Contents

AWS Database Migration Service Step-by-Step Walkthroughs .....	1
Migrating Databases to Amazon Web Services (AWS) .....	2
AWS Migration Tools .....	2
Walkthroughs in this Guide .....	2
Migrating an On-Premises Oracle Database to Amazon Aurora MySQL .....	4
Costs .....	4
Migration High-Level Outline .....	5
Step 1: Prepare Your Oracle Source Database .....	5
Step 2: Launch and Prepare Your Aurora MySQL Target Database .....	5
Step 3: Launch a Replication Instance .....	6
Step 4: Create a Source Endpoint .....	6
Step 5: Create a Target Endpoint .....	6
Step 6: Create and Run a Migration Task .....	7
Migration Step-by-Step Guide .....	8
Step 1: Configure Your Oracle Source Database .....	8
Step 2: Configure Your Aurora Target Database .....	10
Step 3: Creating a Replication Instance .....	11
Step 4: Create Your Oracle Source Endpoint .....	13
Step 5: Create Your Aurora MySQL Target Endpoint .....	16
Step 6: Create a Migration Task .....	18
Step 7: Monitor Your Migration Task .....	23
Troubleshooting .....	23
Working with the Sample Database for Migration .....	24
Migrating an Amazon RDS Oracle Database to Amazon Aurora MySQL .....	25
Costs .....	25
Prerequisites .....	26
Migration Architecture .....	27
Step-by-Step Migration .....	28
Step 1: Launch the RDS Instances in a VPC by Using the CloudFormation Template .....	29
Step 2: Install the SQL Tools and AWS Schema Conversion Tool on Your Local Computer .....	36
Step 3: Test Connectivity to the Oracle DB Instance and Create the Sample Schema .....	38
Step 4: Test the Connectivity to the Aurora MySQL DB Instance .....	41
Step 5: Use the AWS Schema Conversion Tool (AWS SCT) to Convert the Oracle Schema to Aurora MySQL .....	43
Step 6: Validate the Schema Conversion .....	50
Step 7: Create a AWS DMS Replication Instance .....	52
Step 8: Create AWS DMS Source and Target Endpoints .....	53
Step 9: Create and Run Your AWS DMS Migration Task .....	55
Step 10: Verify That Your Data Migration Completed Successfully .....	58
Step 11: Delete Walkthrough Resources .....	60
Next Steps .....	61
Migrating a SQL Server Database to Amazon Aurora MySQL .....	62
Prerequisites .....	62
Step-by-Step Migration .....	63
Step 1: Install the SQL Drivers and AWS Schema Conversion Tool on Your Local Computer .....	63
Step 2: Configure Your Microsoft SQL Server Source Database .....	64
Step 3: Configure Your Aurora MySQL Target Database .....	66
Step 4: Use AWS SCT to Convert the SQL Server Schema to Aurora MySQL .....	66
Step 5: Create an AWS DMS Replication Instance .....	74
Step 6: Create AWS DMS Source and Target Endpoints .....	75
Step 7: Create and Run Your AWS DMS Migration Task .....	79
Step 8: Cut Over to Aurora MySQL .....	82
Troubleshooting .....	83
Migrating an Oracle Database to PostgreSQL .....	84

Prerequisites .....	84
Step-by-Step Migration .....	85
Step 1: Install the SQL Drivers and AWS Schema Conversion Tool on Your Local Computer .....	85
Step 2: Configure Your Oracle Source Database .....	86
Step 3: Configure Your PostgreSQL Target Database .....	88
Step 4: Use the AWS Schema Conversion Tool (AWS SCT) to Convert the Oracle Schema to PostgreSQL .....	89
Step 5: Create an AWS DMS Replication Instance .....	96
Step 6: Create AWS DMS Source and Target Endpoints .....	98
Step 7: Create and Run Your AWS DMS Migration Task .....	101
Step 8: Cut Over to PostgreSQL .....	104
Rolling Back the Migration .....	105
Troubleshooting .....	105
Migrating an Amazon RDS for Oracle Database to Amazon Redshift .....	106
Prerequisites .....	106
Migration Architecture .....	107
Step-by-Step Migration .....	108
Step 1: Launch the RDS Instances in a VPC by Using the CloudFormation Template .....	109
Step 2: Install the SQL Tools and AWS Schema Conversion Tool on Your Local Computer .....	113
Step 3: Test Connectivity to the Oracle DB Instance and Create the Sample Schema .....	116
Step 4: Test the Connectivity to the Amazon Redshift Database .....	119
Step 5: Use AWS SCT to Convert the Oracle Schema to Amazon Redshift .....	121
Step 6: Validate the Schema Conversion .....	127
Step 7: Create an AWS DMS Replication Instance .....	128
Step 8: Create AWS DMS Source and Target Endpoints .....	129
Step 9: Create and Run Your AWS DMS Migration Task .....	132
Step 10: Verify That Your Data Migration Completed Successfully .....	136
Step 11: Delete Walkthrough Resources .....	138
Next Steps .....	139
Migrating MySQL-Compatible Databases to AWS .....	140
Migrating a MySQL-Compatible Database to Amazon Aurora MySQL .....	141
Migrating Data from an External MySQL Database to an Amazon Aurora MySQL Using Amazon S3 .....	141
Prerequisites .....	141
Step 1: Backing Up Files to be Restored as a DB Cluster .....	144
Step 2: Copying Files to an Amazon S3 Bucket .....	145
Step 3: Restoring an Aurora MySQL DB Cluster from an Amazon S3 Bucket .....	145
Migrating MySQL to Amazon Aurora MySQL by Using mysqldump .....	150
Migrating Data from an Amazon RDS MySQL DB Instance to an Amazon Aurora MySQL DB Cluster .....	150
Migrating an RDS MySQL Snapshot to Aurora MySQL .....	150
Migrating a MariaDB Database to Amazon RDS for MySQL or Amazon Aurora MySQL .....	158
Set up MariaDB as a source database .....	158
Set up Aurora MySQL as a target database .....	161
Set up an AWS DMS replication instance .....	162
Test the endpoints .....	163
Create a migration task .....	163
Validate the migration .....	164
Cut over .....	164
Migrating from MongoDB to Amazon DocumentDB .....	166
Launch an Amazon EC2 instance .....	166
Install and configure MongoDB community edition .....	167
Create an AWS DMS replication instance .....	168
Create source and target endpoints .....	169
Create and run a migration task .....	171

# AWS Database Migration Service Step-by-Step Walkthroughs

You can use AWS Database Migration Service (AWS DMS) to migrate your data to and from most widely used commercial and open-source databases such as Oracle, PostgreSQL, Microsoft SQL Server, Amazon Redshift, Amazon Aurora, MariaDB, and MySQL. The service supports homogeneous migrations such as Oracle to Oracle, and also heterogeneous migrations between different database platforms, such as Oracle to MySQL or MySQL to Amazon Aurora MySQL-Compatible Edition. The source or target database must be on an AWS service.

In this guide, you can find step-by-step walkthroughs that go through the process of migrating sample data to AWS:

- [Migrating Databases to Amazon Web Services \(AWS\) \(p. 2\)](#)
- [Migrating an On-Premises Oracle Database to Amazon Aurora MySQL \(p. 4\)](#)
- [Migrating an Amazon RDS Oracle Database to Amazon Aurora MySQL \(p. 25\)](#)
- [Migrating a SQL Server Database to Amazon Aurora MySQL \(p. 62\)](#)
- [Migrating an Oracle Database to PostgreSQL \(p. 84\)](#)
- [Migrating an Amazon RDS for Oracle Database to Amazon Redshift \(p. 106\)](#)
- [Migrating MySQL-Compatible Databases to AWS \(p. 140\)](#)
- [Migrating a MySQL-Compatible Database to Amazon Aurora MySQL \(p. 141\)](#)
- [Migrating a MariaDB Database to Amazon RDS for MySQL or Amazon Aurora MySQL \(p. 158\)](#)
- [Migrating from MongoDB to Amazon DocumentDB \(p. 166\)](#)

In the DMS User Guide, you can find additional resources:

- [Migrating Large Data Stores Using AWS Database Migration Service and AWS Snowball Edge](#)

# Migrating Databases to Amazon Web Services (AWS)

## AWS Migration Tools

You can use several AWS tools and services to migrate data from an external database to AWS. Depending on the type of database migration you are doing, you may find that the native migration tools for your database engine are also effective.

AWS Database Migration Service (AWS DMS) helps you migrate databases to AWS efficiently and securely. The source database can remain fully operational during the migration, minimizing downtime to applications that rely on the database. AWS DMS can migrate your Oracle data to the most widely used commercial and open-source databases on AWS.

AWS DMS migrates data, tables, and primary keys to the target database. All other database elements are not migrated. If you are migrating an Oracle database to Amazon Aurora MySQL-Compatible Edition, for example, you would want to use the AWS Schema Conversion Tool in conjunction with AWS DMS.

The AWS Schema Conversion Tool (SCT) makes heterogeneous database migrations easy by automatically converting the source database schema and a majority of the custom code, including views, stored procedures, and functions, to a format compatible with the target database. Any code that cannot be automatically converted is clearly marked so that it can be manually converted. You can use this tool to convert your source Oracle databases to an Amazon Aurora MySQL, MySQL, or PostgreSQL target database on either Amazon RDS or EC2.

It is important to understand that DMS and SCT are two different tools and serve different needs and they don't interact with each other in the migration process. As per the DMS best practice, migration methodology for this tutorial is outlined as below:

- AWS DMS takes a minimalist approach and creates only those objects required to efficiently migrate the data for example tables with primary key – therefore, we will use DMS to load the tables with data without any foreign keys or constraints. (We can also use the SCT to generate the table scripts and create it on the target before performing the load via DMS).
- We will leverage SCT:
  - To identify the issues, limitations and actions for the schema conversion
  - To generate the target schema scripts including foreign key and constraints
  - To convert code such as procedures and views from source to target and apply it on target

The size and type of Oracle database migration you want to do greatly determines the tools you should use. For example, a heterogeneous migration, where you are migrating from an Oracle database to a different database engine on AWS, is best accomplished using AWS DMS. A homogeneous migration, where you are migrating from an Oracle database to an Oracle database on AWS, is best accomplished using native Oracle tools.

## Walkthroughs in this Guide

[Migrating an On-Premises Oracle Database to Amazon Aurora MySQL \(p. 4\)](#)

[Migrating an Amazon RDS Oracle Database to Amazon Aurora MySQL \(p. 25\)](#)

[Migrating a SQL Server Database to Amazon Aurora MySQL \(p. 62\)](#)

[Migrating an Oracle Database to PostgreSQL \(p. 84\)](#)

[Migrating an Amazon RDS for Oracle Database to Amazon Redshift \(p. 106\)](#)

[Migrating MySQL-Compatible Databases to AWS \(p. 140\)](#)

[Migrating a MySQL-Compatible Database to Amazon Aurora MySQL \(p. 141\)](#)

[Migrating a MariaDB Database to Amazon RDS for MySQL or Amazon Aurora MySQL \(p. 158\)](#)

[Migrating from MongoDB to Amazon DocumentDB \(p. 166\)](#)

# Migrating an On-Premises Oracle Database to Amazon Aurora MySQL

Following, you can find a high-level outline and also a complete step-by-step walkthrough that both show the process for migrating an on-premises Oracle database (the source endpoint) to an Amazon Aurora MySQL-Compatible Edition (the target endpoint) using AWS Database Migration Service (AWS DMS) and the AWS Schema Conversion Tool (AWS SCT).

AWS DMS migrates your data from your Oracle source into your Aurora MySQL target. AWS DMS also captures data manipulation language (DML) and data definition language (DDL) changes that happen on your source database and apply these changes to your target database. This way, AWS DMS helps keep your source and target databases in synch with each other. To facilitate the data migration, DMS creates tables and primary key indexes on the target database if necessary.

However, AWS DMS doesn't migrate your secondary indexes, sequences, default values, stored procedures, triggers, synonyms, views and other schema objects not specifically related to data migration. To migrate these objects to your Aurora MySQL target, use the AWS Schema Conversion Tool.

We highly recommend that you follow along using the Amazon sample database. To find a tutorial that uses the sample database and instructions on how to get a copy of the sample database, see [Working with the Sample Database for Migration \(p. 24\)](#).

If you've used AWS DMS before or you prefer clicking a mouse to reading, you probably want to work with the high-level outline. If you need the details and want a more measured approach (or run into questions), you probably want the step-by-step guide.

<b>Topic: Migration from On-Premises Oracle to Aurora MySQL or MySQL on Amazon RDS</b>
<b>Time:</b>
<b>Cost:</b>
<b>Source Database:</b> Oracle
<b>Target Database:</b> Amazon Aurora MySQL/MySQL
<b>Restrictions:</b>
<b>Oracle Edition:</b> Enterprise, Standard, Express and Personal
<b>Oracle Version:</b> 10g (10.2 and later), 11g, 12c, (On Amazon Relational Database Service (Amazon RDS), 11g or higher is required.)
<b>MySQL or Related Database Version:</b> 5.5, 5.6, 5.7, MariaDB, Amazon Aurora MySQL

## Costs

Because AWS DMS isn't incorporated into the calculator yet, see the following table for a pricing estimate.



In addition to the setup on your own PC, you must create several AWS components to complete the migration process. The AWS components include:

AWS Service	Type	Description
Amazon Aurora MySQL DB instance	db.r3.large	Single AZ, 10 GB storage, 1 million I/O
AWS DMS replication instance	T2.large	50 GB of storage for keeping replication logs included
AWS DMS data transfer	Free, based on the amount of data transferred for the sample database.	
Data transfer out	First 1 GB per month free	

## Migration High-Level Outline

To migrate your data from Oracle to Aurora MySQL using AWS DMS, you take the following steps. If you've used AWS DMS before or prefer clicking a mouse to reading, the following summary should help you kick-start your migration. To get the details about migration or if you run into questions, see the step-by-step guide.

### Step 1: Prepare Your Oracle Source Database

To use AWS DMS to migrate data from an Oracle source database requires some preparation and we also recommend a few additional steps as best practices.

- AWS DMS account – It's a good practice to create a separate account for the specific purpose of migrating your data. This account should have the minimal set of privileges required to migrate your data. Specific details regarding those privileges are outlined below. If you are simply interested in testing AWS DMS on a non-production database, any DBA account will be sufficient.
- Supplemental logging – To capture changes, you must enable supplemental logging in order to use DMS. To enable supplemental logging at the database level issue the following command.

```
ALTER DATABASE ADD SUPPLEMENTAL LOG DATA
```

Additionally, AWS DMS requires for each table being migrated, you set at least key-level supplemental logging. AWS DMS automatically adds this supplemental logging for you if you include the following extra connection parameter for your source connection.

```
addSupplementalLogging=Y
```

- Source database – To migrate your data, the AWS DMS replication server needs access to your source database. Make sure that your firewall rules give the AWS DMS replication server ingress.

### Step 2: Launch and Prepare Your Aurora MySQL Target Database

Following are some things to consider when launching your Aurora MySQL instance:

- For best results, we recommend that you locate your Aurora MySQL instance and your replication instance in the same VPC and, if possible, the same Availability Zone.
- We recommend that you create a separate account with minimal privileges for migrating your data. The AWS DMS account needs the following privileges on all databases to which data is being migrated.

```
ALTER, CREATE, DROP, INDEX, INSERT, UPDATE, DELETE, SELECT
```

Additionally, AWS DMS needs complete access to the `awsdms_control` database. This database holds information required by AWS DMS specific to the migration. To provide access, run the following command.

```
ALL PRIVILEGES ON awsdms_control.* TO 'dms_user'
```

## Step 3: Launch a Replication Instance

The AWS DMS service connects to your source and target databases from a replication instance. Here are some things to consider when launching your replication instance:

- For best results, we recommend that you locate your replication instance in the same VPC and Availability Zone as your target database, in this case Aurora MySQL.
- If either your source or target database is outside of the VPC where you launch your replication server, the replication server must be publicly accessible.
- AWS DMS can consume a fair bit of memory and CPU. However, it's easy enough to scale up if necessary. If you anticipate running several tasks on a single replication server or if your migration involves a large number of tables, consider using one of the larger instances.
- The default storage is usually enough for most migrations.

## Step 4: Create a Source Endpoint

For AWS DMS to access your Oracle source database you'll need to create a source endpoint. The source endpoint defines all the information required for AWS DMS to connect to your source database from the replication server. Following are some requirements for the source endpoint.

- Your source endpoint needs to be accessible from the replication server. To allow this, you will likely need to modify your firewall rules to whitelist the replication server. You can find the IP address of your replication server in the AWS DMS Management Console.
- For AWS DMS to capture changes, Oracle requires supplemental logging be enabled. If you want AWS DMS to enable supplemental logging for you, add the following to the extra connection attributes for your Oracle source endpoint.

```
addSupplementalLogging=Y
```

## Step 5: Create a Target Endpoint

For AWS DMS to access your Aurora MySQL target database you'll need to create a target endpoint. The target endpoint defines all the information required for DMS to connect to your Aurora MySQL database.

- Your target endpoint needs to be accessible from the replication server. You might need to modify your security groups to make the target endpoint accessible.

- If you've pre-created the database on your target, it's a good idea to disable foreign key checks during the full load. To do so, add the following to your extra connection attributes.

```
initstmt=SET FOREIGN_KEY_CHECKS=0
```

## Step 6: Create and Run a Migration Task

A migration task tells AWS DMS where and how you want your data migrated. When creating your migration task, you should consider setting migration parameters as follows.

**Endpoints and replication server** — Choose the endpoints and replication server created above.

**Migration type** — In most cases you'll want to choose **migrate existing data and replication ongoing changes**. With this option, AWS DMS loads your source data while capturing changes to that data. When the data is fully loaded, AWS DMS applies any outstanding changes and keeps the source and target databases in sync until the task is stopped.

**Target table preparation mode \*** — **If you're having AWS DMS create your tables, choose drop tables on target. If you're using some other method to create your target tables such as the AWS Schema Conversion Tool, choose \*truncate.**

**LOB parameters \*** — **If you're just trying AWS DMS, choose include LOB columns in replication, Limited LOB mode, and set your \*max LOB size to 16** (which is 16k.) For more information regarding LOBs, read the details in the step-by-step guide.

**\*Enable logging \*** — To help with debugging migration issues, always enable logging.

**\*Table mappings \*** — When migrating from Oracle to Aurora MySQL, we recommend that you convert your schema, table, and column names to lowercase. To do so, create a custom table mapping. The following example migrates the schema DMS\_SAMPLE and converts schema, table and column names to lower case.

```
{
  "rules": [
    {
      "rule-type": "selection",
      "rule-id": "1",
      "rule-name": "1",
      "object-locator": {
        "schema-name": "DMS_SAMPLE",
        "table-name": "%"
      },
      "rule-action": "include"
    },
    {
      "rule-type": "transformation",
      "rule-id": "6",
      "rule-name": "6",
      "rule-action": "convert-lowercase",
      "rule-target": "schema",
      "object-locator": {
        "schema-name": "%"
      }
    },
    {
      "rule-type": "transformation",
      "rule-id": "7",
      "rule-name": "7",
      "rule-action": "convert-lowercase",
```

```
    "rule-target": "table",
    "object-locator": {
      "schema-name": "%",
      "table-name": "%"
    }
  },
  {
    "rule-type": "transformation",
    "rule-id": "8",
    "rule-name": "8",
    "rule-action": "convert-lowercase",
    "rule-target": "column",
    "object-locator": {
      "schema-name": "%",
      "table-name": "%",
      "column-name": "%"
    }
  }
]
}
```

## Migration Step-by-Step Guide

Following, you can find step-by-step instructions for migrating an Oracle database from an on-premises environment to Amazon Aurora MySQL. These instructions assume that you have already done the setting up steps for using AWS DMS located at [Setting Up to Use AWS Database Migration Service](#).

### Topics

- [Step 1: Configure Your Oracle Source Database \(p. 8\)](#)
- [Step 2: Configure Your Aurora Target Database \(p. 10\)](#)
- [Step 3: Creating a Replication Instance \(p. 11\)](#)
- [Step 4: Create Your Oracle Source Endpoint \(p. 13\)](#)
- [Step 5: Create Your Aurora MySQL Target Endpoint \(p. 16\)](#)
- [Step 6: Create a Migration Task \(p. 18\)](#)
- [Step 7: Monitor Your Migration Task \(p. 23\)](#)
- [Troubleshooting \(p. 23\)](#)

## Step 1: Configure Your Oracle Source Database

To use Oracle as a source for AWS Database Migration Service (AWS DMS), you must first ensure that ARCHIVELOG MODE is on to provide information to LogMiner. AWS DMS uses LogMiner to read information from the archive logs so that AWS DMS can capture changes.

For AWS DMS to read this information, make sure the archive logs are retained on the database server as long as AWS DMS requires them. If you configure your task to begin capturing changes immediately, you should only need to retain archive logs for a little longer than the duration of the longest running transaction. Retaining archive logs for 24 hours is usually sufficient. If you configure your task to begin from a point in time in the past, archive logs need to be available from that time forward. For more specific instructions for enabling ARCHIVELOG MODE and ensuring log retention for your on-premises Oracle database see the [Oracle documentation](#).

To capture change data, AWS DMS requires supplemental logging to be enabled on your source database for AWS DMS. Minimal supplemental logging must be enabled at the database level. AWS DMS also

requires that identification key logging be enabled. This option causes the database to place all columns of a row's primary key in the redo log file whenever a row containing a primary key is updated (even if no value in the primary key has changed). You can set this option at the database or table level.

If your Oracle source is in Amazon RDS, your database will be placed in ARCHIVELOG MODE if, and only if, you enable backups. The following command will ensure archive logs are retained on your RDS source for 24 hours:

```
exec rdsadmin.rdsadmin_util.set_configuration('archivelog retention hours',24);
```

To configure your Oracle source database, do the following:

### 1. Enable database-level supplemental logging

Run the following command to enable supplemental logging at the database level, which AWS DMS requires:

```
ALTER DATABASE ADD SUPPLEMENTAL LOG DATA;  
  
For RDS:  
exec rdsadmin.rdsadmin_util.alter_supplemental_logging('ADD');
```

### 2. Enable identification key supplemental logging

Use the following command to enable identification key supplemental logging at the database level. AWS DMS requires supplemental key logging at the database level unless you allow AWS DMS to automatically add supplemental logging as needed or enable key-level supplemental logging at the table level:

```
ALTER DATABASE ADD SUPPLEMENTAL LOG DATA (PRIMARY KEY) COLUMNS;  
  
For RDS:  
exec rdsadmin.rdsadmin_util.alter_supplemental_logging('ADD','PRIMARY KEY');
```

### 3. (Optional) Enable key level supplemental logging at the table level

Your source database incurs a small bit of overhead when key level supplemental logging is enabled. Therefore, if you are migrating only a subset of your tables, you might want to enable key level supplemental logging at the table level. To enable key level supplemental logging at the table level, use the following command.

```
alter table table_name add supplemental log data (PRIMARY KEY) columns;
```

If a table does not have a primary key you have two options:

- You can add supplemental logging to all columns involved in the first unique index on the table (sorted by index name.)
- You can add supplemental logging on all columns of the table.

To add supplemental logging on a subset of columns in a table, that is those involved in a unique index, run the following command.

```
ALTER TABLE table_name ADD SUPPLEMENTAL LOG GROUP example_log_group (ID,NAME)  
ALWAYS;
```

To add supplemental logging for all columns of a table, run the following command.

```
alter table table_name add supplemental log data (ALL) columns;
```

#### 4. Create or configure a database account to be used by AWS DMS

We recommend that you use an account with the minimal privileges required by AWS DMS for your AWS DMS connection. AWS DMS requires the following privileges.

```
CREATE SESSION
SELECT ANY TRANSACTION
SELECT on V_$ARCHIVED_LOG
SELECT on V_$LOG
SELECT on V_$LOGFILE
SELECT on V_$DATABASE
SELECT on V_$THREAD
SELECT on V_$PARAMETER
SELECT on V_$NLS_PARAMETERS
SELECT on V_$TIMEZONE_NAMES
SELECT on V_$TRANSACTION
SELECT on ALL_INDEXES
SELECT on ALL_OBJECTS
SELECT on ALL_TABLES
SELECT on ALL_USERS
SELECT on ALL_CATALOG
SELECT on ALL_CONSTRAINTS
SELECT on ALL_CONS_COLUMNS
SELECT on ALL_TAB_COLS
SELECT on ALL_IND_COLUMNS
SELECT on ALL_LOG_GROUPS
SELECT on SYS.DBA_REGISTRY
SELECT on SYS.OBJ$
SELECT on DBA_TABLESPACES
SELECT on ALL_TAB_PARTITIONS
SELECT on ALL_ENCRYPTED_COLUMNS
* SELECT on all tables migrated
```

If you want to capture and apply changes (CDC) you also need the following privileges.

```
EXECUTE on DBMS_LOGMNR
SELECT on V_$LOGMNR_LOGS
SELECT on V_$LOGMNR_CONTENTS
LOGMINING /* For Oracle 12c and higher. */
* ALTER for any table being replicated (if you want DMS to add supplemental logging)
```

For Oracle versions before 11.2.0.3, you need the following privileges. If views are exposed, you need the following privileges.

```
SELECT on DBA_OBJECTS /* versions before 11.2.0.3 */
SELECT on ALL_VIEWS (required if views are exposed)
```

## Step 2: Configure Your Aurora Target Database

As with your source database, it's a good idea to restrict access of the user you're connecting with. You can also create a temporary user that you can remove after the migration.

```
CREATE USER 'dms_user'@'%' IDENTIFIED BY 'dms_user';
GRANT ALTER, CREATE, DROP, INDEX, INSERT, UPDATE, DELETE,
```

```
SELECT ON <target database(s)>.* TO 'dms_user'@'%';
```

AWS DMS uses some control tables on the target in the database `awsdms_control`. The following command ensures that your `dms_user` has the necessary access to the `awsdms_control` database:

```
GRANT ALL PRIVILEGES ON awsdms_control.* TO 'dms_user'@'%';  
flush privileges;
```

## Step 3: Creating a Replication Instance

An AWS DMS replication instance performs the actual data migration between source and target. The replication instance also caches the changes during the migration. How much CPU and memory capacity a replication instance has influences the overall time required for the migration. Use the following procedure to set the parameters for a replication instance.

To create an AWS DMS replication instance, do the following:

1. Sign in to the AWS Management Console, and open the AWS DMS console at <https://console.aws.amazon.com/dms/> and choose **Replication instances**. If you are signed in as an AWS Identity and Access Management (IAM) user, you must have the appropriate permissions to access AWS DMS. For more information on the permissions required, see [IAM Permissions Needed to Use AWS DMS](#).
2. Choose **Create replication instance**.
3. On the **Create replication instance** page, specify your replication instance information as shown following.

For This Parameter	Do This
<b>Name</b>	If you plan to launch multiple replication instances or share an account, choose a name that helps you quickly differentiate between the different replication instances.
<b>Description</b>	A good description gives others an idea of what the replication instance is being used for and can prevent accidents.
<b>Instance class</b>	AWS DMS can use a fair bit of memory and CPU. If you have a large database (many tables) or use a number of LOB data types, setting up a larger instance is probably better. As described following, you might be able to boost your throughput by running multiple tasks. Multiple tasks consume more resources and require a larger instance. Keep an eye on CPU and memory consumption as you run your tests. If you find you are using the full capacity of the CPU or swap space, you can easily scale up.
<b>VPC</b>	Here you can choose the VPC where your replication instance will be launched. We recommend that, if possible, you select the same VPC where either your source or target database is (or both). AWS DMS needs to access your source and target database from within this VPC. If either or both of your database endpoints are

For This Parameter	Do This
	outside of this VPC, modify your firewall rules to allow AWS DMS access.
<b>Multi-AZ</b>	If you choose Multi-AZ, AWS DMS launches a primary and secondary replication instance in separate Availability Zones. In the case of a catastrophic disk failure, the primary replication instance automatically fails over to the secondary, preventing an interruption in service. In most situations, if you are performing a migration, you won't need Multi-AZ. If your initial data load takes a long time and you need to keep the source and target databases in sync for a significant portion of time, you might consider running your migration server in a Multi-AZ configuration.
<b>Publicly accessible</b>	If either your source or your target database are outside of the VPC where your replication instance is, you need to make your replication instance publicly accessible.

4. In the **Advanced** section, set the **Allocated storage (GB)** parameter, and then choose **Next**.

For This Option	Do This
<b>Allocated storage (GB)</b>	<p>Storage is primarily consumed by log files and cached transactions. For caches transactions, storage is used only when the cached transactions need to be written to disk. Therefore, AWS DMS doesn't use a significant amount of storage. Some exceptions include the following:</p> <ul style="list-style-type: none"> <li>* <i>Very large tables that incur a significant transaction load.</i> Loading a large table can take some time, so cached transactions are more likely to be written to disk during a large table load.</li> <li>* <i>Tasks that are configured to pause prior to loading cached transactions.</i> In this case, all transactions are cached until the full load completes for all tables. With this configuration, a fair amount of storage might be consumed by cached transactions.</li> <li>* <i>Tasks configured with tables being loaded into Amazon Redshift.</i> However, this configuration isn't an issue when Aurora MySQL is the target.</li> </ul> <p>In most cases, the default allocation of storage is sufficient. However, it's always a good idea to pay attention to storage related metrics and scale up your storage if you find you are consuming more than the default allocation.</p>



For This Option	Do This
<b>Replication Subnet Group</b>	If you run in a Multi-AZ configuration, you need at least two subnet groups.
<b>Availability Zone</b>	If possible, locate your primary replication server in the same Availability Zone as your target database.
<b>VPC Security group(s)</b>	With security groups you can control ingress and egress to your VPC. With AWS DMS you can associate one or more security groups with the VPC where your replication server launches.
<b>KMS key</b>	With AWS DMS, all data is encrypted at rest using a KMS encryption key. By default, AWS DMS creates a new encryption key for your replication server. However, you can use an existing key if desired.

## Step 4: Create Your Oracle Source Endpoint

While your replication instance is being created, you can specify the Oracle source endpoint using the AWS Management Console. However, you can only test connectivity after the replication instance has been created, because the replication instance is used to test the connection.

To specify source or target database endpoints, do the following:

1. In the AWS DMS console, choose **Endpoints** on the navigation pane.
2. Choose **Create endpoint**. The **Create database endpoint page** appears, as shown following.

## Create database endpoint

A database endpoint is used by the replication server to connect to a database. The database specified in the endpoint can be on-premise, on RDS, in EC2 or in the cloud. Details should be specified in the form below. It is recommended that you test your endpoint connections here to avoid errors during processing.

Endpoint type\*  Source  Target ⓘ

Endpoint identifier\*  ⓘ

Source engine\*  ⓘ

Server name\*

Port\*

SSL mode\*  ⓘ

User name\*

Password\*

▶ Advanced

- Specify your connection information for the source Oracle database. The following table describes the source settings.

For This Parameter	Do This
<b>Endpoint type</b>	Choose <b>Source</b> .
<b>Endpoint Identifier</b>	Type an identifier for your Oracle endpoint. The identifier for your endpoint must be unique within an AWS Region.
<b>Source Engine</b>	Choose <b>oracle</b> .
<b>Server name</b>	If your database is on-premises, type an IP address that AWS DMS can use to connect to your database from the replication server. If your database is running on Amazon Elastic Compute Cloud (Amazon EC2) or Amazon RDS, type the public Domain Name Service (DNS) address.
<b>Port</b>	Type the port which your database is listening for connections (the Oracle default is 1521).
<b>SSL mode</b>	Choose a Secure Sockets Layer (SSL) mode if you want to enable connection encryption for this endpoint. Depending on the mode you select,

For This Parameter	Do This
	you might need to provide certificate and server certificate information.
<b>Username</b>	Type the AWS account user name. We recommend that you create an AWS account specific to your migration.
<b>Password</b>	Provide the password for the user name preceding.

4. Choose the **Advanced** tab to set values for extra connection strings and the encryption key.

For This Option	Do This
<b>Extra connection attributes</b>	<p>Here you can add values for extra attributes that control the behavior of your endpoint. A few of the most relevant attributes are listed here. For the full list, see the documentation. Separate multiple entries from each other by using a semi-colon (;).</p> <ul style="list-style-type: none"> <li>* <b>addSupplementalLogging:</b> AWS DMS will automatically add supplemental logging if you enable this option (addSupplementalLogging=Y).</li> <li>* <b>useLogminerReader:</b> By default AWS DMS uses Oracle LogMiner to capture change data from the logs. AWS DMS can also parse the logs using its proprietary technology. If you use Oracle 12c and need to capture changes to tables that include LOBS, set this to No (useLogminerReader=N).</li> <li>* <b>numberDataTypeScale:</b> Oracle supports a NUMBER data type that has no precision or scale. By default, NUMBER is converted to a number with a precision of 38 and scale of 10, number(38,10). Valid values are 0—38 or -1 for FLOAT.</li> <li>* <b>archivedLogDestId:</b> This option specifies the destination of the archived redo logs. The value should be the same as the DEST_ID number in the \$archived_log table. When working with multiple log destinations (DEST_ID), we recommend that you specify a location identifier for archived redo logs. Doing so improves performance by ensuring that the correct logs are accessed from the outset. The default value for this option is 0.</li> </ul>
<b>KMS key</b>	Choose the encryption key to use to encrypt replication storage and connection information. If you choose <b>(Default) aws/dms</b> , the default

For This Option	Do This
	AWS KMS key associated with your account and region is used.

Before you save your endpoint, you can test it. To do so, select a VPC and replication instance from which to perform the test. As part of the test AWS DMS refreshes the list of schemas associated with the endpoint. (The schemas are presented as source options when creating a task using this source endpoint.)

## Step 5: Create Your Aurora MySQL Target Endpoint

Next, you can provide information for the target Amazon Aurora MySQL database by specifying the target endpoint settings. The following table describes the target settings.

To specify a target database endpoint, do the following:

1. In the AWS DMS console, choose **Endpoints** on the navigation pane.
2. Choose **Create endpoint**. The **Create database endpoint page** appears, as shown following.

### Create database endpoint

A database endpoint is used by the replication server to connect to a database. The database specified in the endpoint can be on-premise, on RDS, in EC2 or in the cloud. Details should be specified in the form below. It is recommended that you test your endpoint connections here to avoid errors during processing.

**Endpoint type\***  Source  Target ?

**Endpoint identifier\***  ?

**Source engine\***  ?

**Server name\***

**Port\***

**SSL mode\***  ?

**User name\***

**Password\***

▶ **Advanced**

3. Specify your connection information for the target Aurora MySQL database. The following table describes the target settings.

For This Parameter	Do This
<b>Endpoint type</b>	Choose <b>Target</b> .
<b>Endpoint Identifier</b>	Type an identifier for your Aurora MySQL endpoint. The identifier for your endpoint must be unique within an AWS Region.
<b>Target Engine</b>	Choose <b>aurora</b> .
<b>Servername</b>	Type the writer endpoint for your Aurora MySQL instance. The writer endpoint is the primary instance.
<b>Port</b>	Type the port assigned to the instance.
<b>SSL mode</b>	Choose an SSL mode if you want to enable connection encryption for this endpoint. Depending on the mode you select, you might need to provide certificate and server certificate information.
<b>Username</b>	Type the user name for the account you are using for the migration. We recommend that you create an account specific to your migration.
<b>Password</b>	Provide the password for the user name preceding.

4. Choose the **Advanced** tab to set values for extra connection strings and the encryption key if you need them.

For This Option	Do This
<b>Extra connection attributes</b>	<p>Here you can enter values for additional attributes that control the behavior of your endpoint. A few of the most relevant attributes are listed here. For the full list, see the documentation. Separate multiple entries from each other by using a semi-colon (;).</p> <p>* <b>targetDbType</b>: By default, AWS DMS creates a different MySQL database for each schema being migrated. Sometimes you might want to combine objects from several schemas into a single database. To do so, set this option to <code>specific_database</code> (<code>targetDbType=SPECIFIC_DATABASE</code>).</p> <p>* <b>initstmt</b>: You use this option to invoke the MySQL <code>initstmt</code> connection parameter and accept anything <code>mysql initstmt</code> accepts. When working with an Aurora MySQL target, it's often useful to disable foreign key checks. To do so, use the <code>initstmt</code> parameter as follows:</p> <pre>initstmt=SET FOREIGN_KEY_CHECKS=0</pre>

For This Option	Do This
KMS key	Choose the encryption key to use to encrypt replication storage and connection information. If you choose <b>(Default) aws/dms</b> , the default AWS KMS key associated with your account and region is used.

Prior to saving your endpoint, you have an opportunity to test it. To do so you'll need to select a VPC and replication instance from which to perform the test.

## Step 6: Create a Migration Task

When you create a migration task you tell AWS DMS exactly how you want your data migrated. Within a task you define which tables you'd like migrated, where you'd like them migrated, and how you'd like them migrated. If you're planning to use the change capture and apply capability of AWS DMS it's important to know transactions are maintained within a single task. In other words, you should migrate all tables that participate in a single transaction together in the same task.

Using an AWS DMS task, you can specify what schema to migrate and the type of migration. You can migrate existing data, migrate existing data and replicate ongoing changes, or replicate data changes only. This walkthrough migrates existing data only.

To create a migration task, do the following:

1. On the navigation pane, choose **Tasks**.
2. Choose **Create Task**.
3. On the **Create Task** page, specify the task options. The following table describes the settings.

For This Option	Do This
<b>Task name</b>	It's always a good idea to give your task a descriptive name that helps organization.
<b>Task description</b>	Type a description for the task.
<b>Source endpoint</b>	Select your source endpoint.
<b>Target endpoint</b>	Select your target endpoint.
<b>Replication instance</b>	Select a replication instance on which to run the task. Remember, your source and target endpoints must be accessible from this instance.
<b>Migration type</b>	<p>You can use three different migration types with AWS DMS.</p> <p><i>1. Migrate existing data</i></p> <p>If you select this option, AWS DMS migrates only your existing data. Changes to your source data aren't captured and applied to your target. If you can afford taking an outage for the duration of the full load, migrating with this option is simple and straight forward. This method is also good to use when creating test copies of your database.</p>

For This Option	Do This
	<p><i>2. Migrate existing data and replicate ongoing changes</i></p> <p>With this option, AWS DMS captures changes while migrating your existing data. AWS DMS continues to capture and apply changes even after the bulk data has been loaded. Eventually the source and target databases will be in sync, allowing for a minimal downtime migration. To do this, take the following steps:</p> <ul style="list-style-type: none"> <li>* Shut the application down</li> <li>* Let the final change flow through to the target</li> <li>* Perform any administrative tasks such as enabling foreign keys and triggers</li> <li>* Start the application pointing to the new target database</li> </ul> <p>Note that AWS DMS loads the bulk data table-by-table, &lt;n&gt; tables at a time. As the full load progresses, AWS DMS begins applying cached changes to the target tables as soon as possible. During the bulk load, referential integrity is violated, therefore existing foreign keys must be disabled for the full load. Once the full load is complete, your target database has integrity and changes are applied as transactions.</p> <p><i>3. Replicate data changes only</i></p> <p>In some cases you might choose to load bulk data using a different method. This approach generally only applies to homogeneous migrations.</p>
<b>Start task on create</b>	<p>In most situations having the task start immediately is fine. Sometimes you might want to delay the start of a task, for instance, to change logging levels.</p>

4. Next, set the Advanced settings as shown following.

For This Option	Do This
<b>Target table preparation mode</b>	<p>AWS DMS allows you to specify how you would like your target tables prepared prior to loading.</p> <p><b>Do nothing</b> - When you select this option, AWS DMS does nothing to prepare your tables. Your table structure remains as is and any existing data is left in the table. You can use this method to consolidate data from multiple systems.</p>

For This Option	Do This
	<p><b>Drop tables on target</b> - Typically you use this option when you want AWS DMS to create your target table for you. When you select this option, AWS DMS drops and recreates the tables to migrate before migration.</p> <p><b>Truncate</b> - Select this option if you want to pre-create some or all of the tables on your target system, maybe with the AWS Schema Conversion Tool. When you select this option, AWS DMS truncates a target table prior to loading it. If the target table doesn't exist, AWS DMS creates the table for you.</p>
<p><b>Include LOB columns in replication</b></p>	<p>Large objects, (LOBs) can sometimes be difficult to migrate between systems. AWS DMS offers a number of options to help with the tuning of LOB columns. To see which and when datatypes are considered LOBS by AWS DMS, see the AWS DMS documentation.</p> <p><b>Don't include LOB columns</b> - When you migrate data from one database to another, you might take the opportunity to rethink how your LOBs are stored, especially for heterogeneous migrations. If you want to do so, there's no need to migrate the LOB data.</p> <p><b>Full LOB mode</b> - In <b>full LOB mode</b> AWS DMS migrates all LOBs from source to target regardless of size. In this configuration, AWS DMS has no information about the maximum size of LOBs to expect. Thus, LOBs are migrated one at a time, piece by piece. Full LOB mode can be quite slow.</p> <p><b>Limited LOB mode</b> - In <b>limited LOB mode</b>, you set a maximum size LOB that AWS DMS should accept. Doing so allows AWS DMS to pre-allocate memory and load the LOB data in bulk. LOBs that exceed the maximum LOB size are truncated and a warning is issued to the log file. In <b>limited LOB mode</b> you get significant performance gains over <b>full LOB mode</b>. We recommend that you use <b>limited LOB mode</b> whenever possible.</p> <p>Note that with Oracle, LOBs are treated as VARCHAR data types whenever possible. This approach means AWS DMS fetches them from the database in bulk, which is significantly faster than other methods. The maximum size of a VARCHAR in Oracle is 64K, therefore a limited LOB size of less than 64K is optimal when Oracle is your source database.</p>



For This Option	Do This
<b>Max LOB size (K)</b>	When a task is configured to run in <b>limited LOB mode</b> , this option determines the maximum size LOB that AWS DMS accepts. Any LOBs that are larger than this value will be truncated to this value.
<b>LOB chunk size (K)</b>	When a task is configured to use <b>full LOB mode</b> , AWS DMS retrieves LOBs in pieces. This option determines the size of each piece. When setting this option, pay particular attention to the maximum packet size allowed by your network configuration. If the LOB chunk size exceeds your maximum allowed packet size, you might see disconnect errors.
<b>Custom CDC start time</b>	This parameter pertains to tasks configured to replicate data changes only. It tells AWS DMS where to start looking for changes in the change stream.
<b>Enable logging</b>	Always enable logging.

5. Set additional parameters.

For This Option	Do This
<b>Create control table(s) in target schema</b>	AWS DMS requires some control tables in the target database. By default those tables are created in the same database as your data. This parameter allows you to tell AWS DMS to put those artifacts somewhere else.
<b>Maximum number of tables to load in parallel</b>	AWS DMS performs a table-by-table load of your data. This parameter allows you to control how many tables AWS DMS will load in parallel. The default is 8, which is optimal in most situations.

6. Specify any table mapping settings.

Table mappings tell AWS DMS which tables a task should migrate from source to target. Table mappings are expressed in JSON, though some settings can be made using the AWS Management Console. Table mappings can also include transformations such as changing table names from upper case to lower case.

AWS DMS generates default table mappings for each (non-system) schema in the source database. In most cases you'll want to customize your table mapping. To customize your table mapping select the custom radio button. For details on creating table mappings see the AWS DMS documentation. The following table mapping does these things:

- It includes the DMS\_SAMPLE schema in the migration.
- It excludes the tables NFL\_DATA, MLB\_DATA, NAME\_DATE, and STADIUM\_DATA.
- It converts the schema, table, and column names to lower case.

```
{
  "rules": [
    {
      "rule-type": "selection",
```

AWS Database Migration Service  
Step-by-Step Migration Guide  
Step 6: Create a Migration Task

```
"rule-id": "1",
"rule-name": "1",
"object-locator": {
  "schema-name": "DMS_SAMPLE",
  "table-name": "%"
},
"rule-action": "include"
},
{
  "rule-type": "selection",
  "rule-id": "2",
  "rule-name": "2",
  "object-locator": {
    "schema-name": "DMS_SAMPLE",
    "table-name": "MLB_DATA"
  },
  "rule-action": "exclude"
},
{
  "rule-type": "selection",
  "rule-id": "3",
  "rule-name": "3",
  "object-locator": {
    "schema-name": "DMS_SAMPLE",
    "table-name": "NAME_DATA"
  },
  "rule-action": "exclude"
},
{
  "rule-type": "selection",
  "rule-id": "4",
  "rule-name": "4",
  "object-locator": {
    "schema-name": "DMS_SAMPLE",
    "table-name": "NFL_DATA"
  },
  "rule-action": "exclude"
},
{
  "rule-type": "selection",
  "rule-id": "5",
  "rule-name": "5",
  "object-locator": {
    "schema-name": "DMS_SAMPLE",
    "table-name": "NFL_STADIUM_DATA"
  },
  "rule-action": "exclude"
},
{
  "rule-type": "transformation",
  "rule-id": "6",
  "rule-name": "6",
  "rule-action": "convert-lowercase",
  "rule-target": "schema",
  "object-locator": {
    "schema-name": "%"
  }
},
{
  "rule-type": "transformation",
  "rule-id": "7",
  "rule-name": "7",
  "rule-action": "convert-lowercase",
  "rule-target": "table",
```

```
    "object-locator": {  
      "schema-name": "%",  
      "table-name": "%"  
    }  
  },  
  {  
    "rule-type": "transformation",  
    "rule-id": "8",  
    "rule-name": "8",  
    "rule-action": "convert-lowercase",  
    "rule-target": "column",  
    "object-locator": {  
      "schema-name": "%",  
      "table-name": "%",  
      "column-name": "%"  
    }  
  }  
]  
}
```

## Step 7: Monitor Your Migration Task

Three sections in the console provide visibility into what your migration task is doing:

- Task monitoring – The **Task Monitoring** tab provides insight into your full load throughput and also your change capture and apply latencies.
- Table statistics – The **Table Statistics** tab provides detailed information on the number of rows processed, type and number of transactions processed, and also information on DDL operations.
- Logs – From the **Logs** tab you can view your task's log file, (assuming you turned logging on.) If for some reason your task fails, search this file for errors. Additionally, you can look in the file for any warnings. Any data truncation in your task appears as a warning in the log file. If you need to, you can increase the logging level by using the AWS Command Line Interface (CLI).

## Troubleshooting

The two most common areas people have issues with when working with Oracle as a source and Aurora MySQL as a target are: supplemental logging and case sensitivity.

- Supplemental logging – With Oracle, in order to replication change data supplemental logging must be enabled. However, if you enable supplemental logging at the database level, it sometimes still need to enable it when creating new tables. The best remedy for this is to allow DMS to enable supplemental logging for you using the extra connection attribute:

```
addSupplementalLogging=Y
```

- Case sensitivity: Oracle is case-insensitive (unless you use quotes around your object names). However, text appears in uppercase. Thus, AWS DMS defaults to naming your target objects in uppercase. In most cases, you'll want to use transformations to change schema, table and column names to lower case.

For more tips, see the AWS DMS troubleshooting section in the [AWS DMS User Guide](#).

To troubleshoot issues specific to Oracle, see the Oracle troubleshooting section:

[https://docs.aws.amazon.com/dms/latest/userguide/CHAP\\_Troubleshooting.html#CHAP\\_Troubleshooting.Oracle](https://docs.aws.amazon.com/dms/latest/userguide/CHAP_Troubleshooting.html#CHAP_Troubleshooting.Oracle)

To troubleshoot Aurora MySQL issues, see the MySQL troubleshooting section:

[https://docs.aws.amazon.com/dms/latest/userguide/CHAP\\_Troubleshooting.html#CHAP\\_Troubleshooting.MySQL](https://docs.aws.amazon.com/dms/latest/userguide/CHAP_Troubleshooting.html#CHAP_Troubleshooting.MySQL)

## Working with the Sample Database for Migration

We recommend working through the preceding outline and guide by using the sample Oracle database provided by Amazon. This database mimics a simple sporting event ticketing system. The scripts to generate the sample database can be found at <https://github.com/aws-samples/aws-database-migration-samples/tree/master/mysql/sampledb/v1>.

To build the sample database, go to the `oracle/sampledb/v1` folder and follow the instructions in the `README.md` file.

The sample creates approximately 8-10 GB of data. The sample database also includes a *ticketManagement* package, which you can use to generate some transactions. To generate transactions, log into SQL\*Plus or SQL Developer and run the following as **dms\_sample** :

```
SQL>call generateTicketActivity(1000,0.01);
```

The first parameter is the transaction delay in seconds, the second is the number of transactions to generate. The procedure preceding simply "sells tickets" to people. You'll see updates to the tables: `sporting_event_ticket`, and `ticket_purchase_history`.

Once you've "sold" some tickets, you can transfer them using the command following:

```
SQL>call generateTransferActivity(100,0.1);
```

The first parameter is the transaction delay in seconds, the second is the number of transactions to generate. This procedure also updates `sporting_event_ticket` and `ticket_purchase_history`.

# Migrating an Amazon RDS Oracle Database to Amazon Aurora MySQL

This walkthrough gets you started with heterogeneous database migration from Amazon RDS Oracle to Amazon Aurora MySQL-Compatible Edition using AWS Database Migration Service and the AWS Schema Conversion Tool. This is an introductory exercise so does not cover all scenarios but will provide you with a good understanding of the steps involved in executing such a migration.

It is important to understand that AWS DMS and AWS SCT are two different tools and serve different needs. They don't interact with each other in the migration process. At a high level, the steps involved in this migration are:

1. Using the AWS SCT to:
  - Run the conversion report for Oracle to Aurora MySQL to identify the issues, limitations, and actions required for the schema conversion.
  - Generate the schema scripts and apply them on the target before performing the data load via AWS DMS. AWS SCT will perform the necessary code conversion for objects like procedures and views.
2. Identify and implement solutions to the issues reported by AWS SCT. For example, an object type like Oracle Sequence that is not supported in the Amazon Aurora MySQL can be handled using the `auto_increment` option to populate surrogate keys or develop logic for sequences at the application layer.
3. Disable foreign keys or any other constraints which may impact the AWS DMS data load.
4. AWS DMS loads the data from source to target using the Full Load approach. Although AWS DMS is capable of creating objects in the target as part of the load, it follows a minimalistic approach to efficiently migrate the data so it doesn't copy the entire schema structure from source to target.
5. Perform post-migration activities such as creating additional indexes, enabling foreign keys, and making the necessary changes in the application to point to the new database.

This walkthrough uses a custom AWS CloudFormation template to create an Amazon RDS DB instances for Oracle and Amazon Aurora MySQL. It then uses a SQL command script to install a sample schema and data onto the Amazon RDS Oracle DB instance that you then migrate to Amazon Aurora MySQL.

This walkthrough takes approximately two hours to complete. The estimated cost to complete it, using AWS resources, is about \$5.00. Be sure to follow the instructions to delete resources at the end of this walkthrough to avoid additional charges.

## Topics

- [Costs \(p. 25\)](#)
- [Prerequisites \(p. 26\)](#)
- [Migration Architecture \(p. 27\)](#)
- [Step-by-Step Migration \(p. 28\)](#)
- [Next Steps \(p. 61\)](#)

## Costs

For this walkthrough, you provision Amazon Relational Database Service (Amazon RDS) resources by using AWS CloudFormation and also AWS Database Migration Service (AWS DMS) resources. Provisioning

these resources will incur charges to your AWS account by the hour. The AWS Schema Conversion Tool incurs no cost; it is provided as a part of AWS DMS.

Although you'll need only a minimum of resources for this walkthrough, some of these resources are not eligible for AWS Free Tier. At the end of this walkthrough, you'll find a section in which you delete the resources to avoid additional charges. Delete the resources as soon as you complete the walkthrough.

To estimate what it will cost to run this walkthrough on AWS, you can use the AWS Simple Monthly Calculator. However, the AWS DMS service is not incorporated into the calculator yet. The following table shows both AWS DMS and Amazon RDS Oracle Standard Edition Two pricing.

AWS Service	Instance Type	Storage and I/O
Amazon RDS Oracle DB instance, License Included (Standard Edition Two), Single AZ	db.m3.medium	Single AZ, 10 GB storage, GP2
Amazon Aurora MySQL DB instance	db.r3.large	Single AZ, 10 GB storage, 1 million I/O
AWS DMS replication instance	t2.small	50 GB of storage for keeping replication logs included
AWS DMS data transfer	Free—data transfer between AWS DMS and databases in RDS instances in the same Availability Zone is free	
Data transfer out	First 1 GB per month free	

Assuming you run this walkthrough for two hours, we estimate the following pricing for AWS resources:

- Amazon Aurora MySQL + 10 GB storage pricing estimated by using the link to the Simple Monthly Calculator that you can access from the [pricing site](#) is \$1.78.
- Amazon RDS Oracle SE2 (license included) + 10 GB GP2 storage cost, estimated as per the [aws.amazon.com](#) at  $(\$0.226) * 2 \text{ hours} + (\$0.115) * 10 \text{ GB}$ , is \$1.602.
- AWS DMS service cost for the t2.small instance with 50 GB GP2 storage, estimated as per the [pricing site](#) at  $(\$0.036) * 2 \text{ hours}$ , is \$0.072.

Total estimated cost to run this project =  $\$1.78 + \$1.602 + \$0.072 = \$3.454$ —approximately \$5.00.

This pricing is based on the following assumptions:

- We assume the total data transfer to the Internet is less than a gigabyte. The preceding pricing estimate assumes that data transfer and backup charges associated with the RDS and DMS services are within Free Tier limits.
- Storage consumed by the Aurora MySQL database is billed in per GB-month increments, and I/Os consumed are billed in per-million request increments.
- Data transfer between DMS and databases in RDS instances in the same Availability Zone is free.

## Prerequisites

The following prerequisites are also required to complete this walkthrough:

- Familiarity with Amazon RDS, the applicable database technologies, and SQL.

- The custom scripts that include creating the tables to be migrated and SQL queries for confirming the migration, as listed following:
  - `Oracle-HR-Schema-Build.sql`--SQL statements to build the HR schema.
  - `Oracle_Aurora_For_DMSDemo.template`--an AWS CloudFormation template.

These scripts are available at the following link: [dms-sbs-RDSOracle2Aurora.zip](#)

Each step in the walkthrough also contains a link to download the file involved or includes the exact query in the step. \* An AWS account with AWS Identity and Access Management (IAM) credentials that allow you to launch Amazon Relational Database Service (Amazon RDS) and AWS Database Migration Service (AWS DMS) instances in your AWS Region. For information about IAM credentials, see [Creating an IAM User](#). \* Basic knowledge of the Amazon Virtual Private Cloud (Amazon VPC) service and of security groups. For information about using Amazon VPC with Amazon RDS, see [Virtual Private Clouds \(VPCs\) and Amazon RDS](#). For information about Amazon RDS security groups, see [Amazon RDS Security Groups](#). \* An understanding of the supported features and limitations of AWS DMS. For information about AWS DMS, see [What Is AWS Database Migration Service?](#). \* Knowledge of the supported data type conversion options for Oracle and Amazon Aurora MySQL. For information about data types for Oracle as a source, see [Using an Oracle Database as a Source for AWS Database Migration Service](#). For information about data types for Amazon Aurora MySQL as a target, see [Using a MySQL-Compatible Database as a Target for AWS Database Migration Service](#).

For more information on AWS DMS, see [the AWS DMS documentation](#).

## Migration Architecture

This walkthrough uses AWS CloudFormation to create a simple network topology for database migration that includes the source database, the replication instance, and the target database in the same VPC. For more information on AWS CloudFormation, see [the CloudFormation documentation](#).

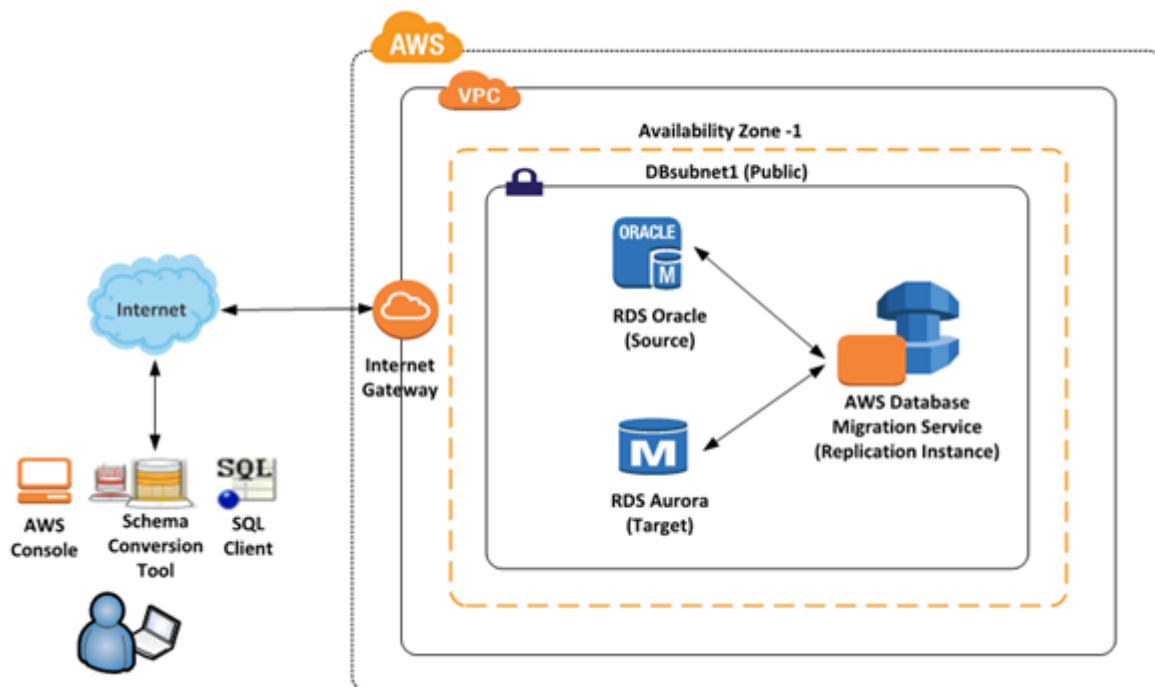
We will provision the AWS resources that are required for this AWS Database Migration Service (AWS DMS) walkthrough through AWS CloudFormation. These resources include a VPC and Amazon Relational Database Service (Amazon RDS) instances for Oracle and Amazon Aurora MySQL. We provision through AWS CloudFormation because it simplifies the process, so we can concentrate on tasks related to data migration. When you create a stack from the AWS CloudFormation template, it provisions the following resources:

- A VPC with CIDR (10.0.0.0/24) with two public subnets in your region, DBSubnet1 at the address 10.0.0.0/26 in Availability Zone 1 (AZ 1) and DBSubnet2 at the address 10.0.0.64/26, in AZ 2.
- A DB subnet group that includes DBSubnet1 and DBSubnet2.
- Oracle RDS Standard Edition Two with these deployment options:
  - License Included
  - Single-AZ setup
  - db.m3.medium or equivalent instance class
  - Port 1521
  - Default option and parameter groups
- Amazon Aurora MySQL DB instance with these deployment options:
  - No replicas
  - db.r3.large or equivalent instance class
  - Port 3306
  - Default option and parameter groups
- A security group with ingress access from your computer or 0.0.0.0/0 (access from anywhere) based on the input parameter

We have designed the CloudFormation template to require few inputs from the user. It provisions the necessary AWS resources with minimum recommended configurations. However, if you want to change some of the configurations and parameters, such as the VPC CIDR block and Amazon RDS instance types, feel free to update the template.

We will use the AWS Management Console to provision the AWS DMS resources, such as the replication instance, endpoints, and tasks. You will install client tools such as SQL Workbench/J and the AWS Schema Conversion Tool (AWS SCT) on your local computer to connect to the Amazon RDS instances.

Following is an illustration of the migration architecture for this walkthrough.



## Step-by-Step Migration

In the following sections, you can find step-by-step instructions for migrating an Amazon Relational Database Service (Amazon RDS) Oracle database to Amazon Aurora MySQL. These steps assume that you have already prepared your source database as described in preceding sections.

### Topics

- [Step 1: Launch the RDS Instances in a VPC by Using the CloudFormation Template \(p. 29\)](#)
- [Step 2: Install the SQL Tools and AWS Schema Conversion Tool on Your Local Computer \(p. 36\)](#)
- [Step 3: Test Connectivity to the Oracle DB Instance and Create the Sample Schema \(p. 38\)](#)
- [Step 4: Test the Connectivity to the Aurora MySQL DB Instance \(p. 41\)](#)
- [Step 5: Use the AWS Schema Conversion Tool \(AWS SCT\) to Convert the Oracle Schema to Aurora MySQL \(p. 43\)](#)
- [Step 6: Validate the Schema Conversion \(p. 50\)](#)
- [Step 7: Create a AWS DMS Replication Instance \(p. 52\)](#)
- [Step 8: Create AWS DMS Source and Target Endpoints \(p. 53\)](#)



- [Step 9: Create and Run Your AWS DMS Migration Task \(p. 55\)](#)
- [Step 10: Verify That Your Data Migration Completed Successfully \(p. 58\)](#)
- [Step 11: Delete Walkthrough Resources \(p. 60\)](#)

## Step 1: Launch the RDS Instances in a VPC by Using the CloudFormation Template

Before you begin, you'll need to download an AWS CloudFormation template. Follow these instructions:

1. Download the following archive to your computer: <http://docs.aws.amazon.com/dms/latest/sbs/samples/dms-sbs-RDSOracle2Aurora.zip>
2. Extract the CloudFormation template (`Oracle_Aurora_For_DMSDemo.template`) from the archive.
3. Copy and paste the `Oracle_Aurora_For_DMSDemo.template` file into your current directory.

Now you need to provision the necessary AWS resources for this walkthrough. Do the following:

1. Sign in to the AWS Management Console and open the AWS CloudFormation console at <https://console.aws.amazon.com/cloudformation>.
2. Choose **Create stack** and then choose **With new resources (standard)**.
3. On the **Specify template** section of the **Create stack** page, choose **Upload a template file**.
4. Click **Choose file**, and then choose the `Oracle_Aurora_For_DMSDemo.template` file that you extracted from the `dms-sbs-RDSOracle2Aurora.zip` archive.
5. Choose **Next**. On the **Specify Details** page, provide parameter values as shown following.

For This Parameter	Do This
<b>Stack Name</b>	Type <code>DMSdemo</code> .
<b>OracleDBName</b>	Provide a unique name for your database. The name should begin with a letter. The default is <code>ORCL</code> .
<b>OracleDBUsername</b>	Specify the admin (DBA) user for managing the Oracle instance. The default is <code>oraadmin</code> .
<b>OracleDBPassword</b>	Provide the password for the admin user. The default is <code>oraadmin123</code> .
<b>AuroraDBUsername</b>	Specify the admin (DBA) user for managing the Aurora MySQL instance. The default is <code>auradmin</code> .
<b>AuroraDBPassword</b>	Provide the password for the admin user. The default is <code>auradmin123</code> .
<b>ClientIP</b>	Specify the IP address in CIDR ( <code>x.x.x.x/32</code> ) format for your local computer. You can get your IP address from <a href="http://whatsmyip.org">whatsmyip.org</a> . Your RDS instances' security group will allow ingress to this IP address. The default is access from anywhere ( <code>0.0.0.0/0</code> ), which is not recommended; you should use your IP address for this walkthrough.

## Specify stack details

### Stack name

Stack name

Stack name can include letters (A-Z and a-z), numbers (0-9), and dashes (-).

### Parameters

Parameters are defined in your template and allow you to input custom values when you create or update a stack.

#### Source Oracle Database Configuration

OracleDBName

Enter Oracle Database name

OracleDBUsername

Enter database Admin username for Oracle

OracleDBPassword

Enter password for Oracle Admin user

#### Target Aurora Database Configuration

AuroraDBUsername

Enter database Admin username for Aurora

AuroraDBPassword

Enter password for Aurora Admin user

#### Enter IP address for DB Security group Configuration

ClientIP

API Version 2016-01-01

The IP address range that can be used to connect to the RDS instances from your local machine. It must be a valid range of the form x.x.x.x/x. Pls get your address using [checkip.amazonaws.com](http://checkip.amazonaws.com) or [whatsmyip.org](http://whatsmyip.org)

6. Choose **Next**. On the **Configure stack options** page, shown following, choose **Next**.

## Configure stack options

### Tags

You can specify tags (key-value pairs) to apply to resources in your stack. You can add up to 50 unique tags for each stack. [Learn more](#)

<input type="text" value="Key"/>	<input type="text" value="Value"/>
<input type="button" value="Add tag"/>	

### Permissions

Choose an IAM role to explicitly define how CloudFormation can create, modify, or delete resources in the stack. If you don't choose a role, CloudFormation uses permissions based on your user credentials. [Learn more](#)

#### IAM role - optional

Choose the IAM role for CloudFormation to use for all operations performed on the stack.

<input type="text" value="IAM role ..."/>	<input type="text" value="Sample-role-name"/>
---	---

### Stack failure options

#### Behavior on provisioning failure

Specify the roll back behavior for a stack failure. [Learn more](#)

- Roll back all stack resources**  
Roll back the stack to the last known stable state.
- Preserve successfully provisioned resources**  
Preserves the state of successfully provisioned resources, while rolling back failed resources to the last known stable state. A last known stable state will be deleted upon the next stack operation.

## Advanced options

You can set additional options for your stack, like notification options and a stack policy. [Learn more](#)

### ► Stack policy

Defines the resources that you want to protect from unintentional updates during a stack update.

### ► Rollback configuration 32

Specify alarms for CloudFormation to monitor when creating and updating the stack. If the operation breaches an alarm, CloudFormation rolls it back. [Learn more](#)

AWS Database Migration Service  
Step-by-Step Migration Guide  
Step 1: Launch the RDS Instances in a VPC  
by Using the CloudFormation Template

---

7. On the **Review** page, review the details, and if they are correct, scroll down and choose **Create stack**. You can get the estimated cost of running this CloudFormation template by choosing **Estimate cost** at the **Template** section on top of the page.

## Review DMSdemo

### Step 1: Specify template

#### Template

Template URL

[https://s3-external-1.amazonaws.com/cf-templates-grtw7rybb40t-us-east-1/20212514ci-Oracle\\_Aurora\\_For\\_DMSDemo.template](https://s3-external-1.amazonaws.com/cf-templates-grtw7rybb40t-us-east-1/20212514ci-Oracle_Aurora_For_DMSDemo.template)

Stack description

This CloudFormation sample template OracletoAurora\_DMS creates Oracle and Aurora instances in a VPC to test the migration using AWS DMS service. You will be billed for the AWS resources used if you create a stack from this template

[Estimate cost](#) 

### Step 2: Specify stack details

#### Parameters (6)

Key	Value
AuroraDBPassword	*****
AuroraDBUsername	auradmin
ClientIP	54.239.6.187/32
OracleDBName	ORCL
OracleDBPassword	*****
OracleDBUsername	oraadmin

AWS Database Migration Service  
 Step-by-Step Migration Guide  
 Step 1: Launch the RDS Instances in a VPC  
 by Using the CloudFormation Template

8. AWS can take about 20 minutes or more to create the stack with Amazon RDS Oracle and Amazon Aurora MySQL instances.

The screenshot shows the AWS CloudFormation console. At the top, there are buttons for 'Create Stack', 'Actions', and 'Design template'. Below these is a filter section with 'Filter: Active' and a search box 'By Name:'. The main area displays a table of stacks:

Stack Name	Created Time	Status	Description
<input checked="" type="checkbox"/> DMSdemo	2016-08-18 14:57:51 UTC-0700	CREATE_IN_PROGRESS	This CloudFormation sample template c
<input type="checkbox"/> [Stack Name]	2016-08-10 13:56:20 UTC-0700	CREATE_COMPLETE	Basic Cluster Substack
<input type="checkbox"/> [Stack Name]	2016-08-09 13:51:47 UTC-0700	CREATE_COMPLETE	Create instances ready for CloudMap
<input type="checkbox"/> [Stack Name]	2016-08-04 15:06:43 UTC-0700	ROLLBACK_COMPLETE	Create instances ready for CloudMap
<input type="checkbox"/> [Stack Name]	2016-08-01 10:56:18 UTC-0700	CREATE_COMPLETE	Create instances ready for CloudMap
<input type="checkbox"/> [Stack Name]	2016-07-11 10:07:14 UTC-0700	CREATE_COMPLETE	Builds a VPC subnet, a VPC w/ 1 public s
<input type="checkbox"/> [Stack Name]	2016-07-01 17:24:20 UTC-0700	CREATE_COMPLETE	Adds an IAM user to an IAM group
<input type="checkbox"/> [Stack Name]	2016-07-01 17:18:56 UTC-0700	CREATE_COMPLETE	Creates an IAM user
<input type="checkbox"/> [Stack Name]	2016-07-01 17:10:18 UTC-0700	CREATE_COMPLETE	Assigns access permissions to an IAM

Below the stack list is an 'Events' section with a table showing the creation event:

2016-08-18	Status	Type	Logical ID	Status reason
14:57:51 UTC-0700	CREATE_IN_PROGRESS	AWS::CloudFormation::Stack	DMSdemo	User Initiated

9. After the stack is created, choose **Stack**, select the DMSdemo stack, and then choose **Outputs**. Record the JDBC connection strings, **OracleJDBCConnectionString** and **AuroraJDBCConnectionString**, for use later in this walkthrough to connect to the Oracle and Aurora MySQL DB instances.

AWS Database Migration Service  
Step-by-Step Migration Guide  
Step 2: Install the SQL Tools and AWS Schema  
Conversion Tool on Your Local Computer

CloudFormation > Stack List > Stack Detail: DMSdemo

## DMSdemo

Other Actions ▾

Update Stack

Stack name: DMSdemo

Stack ID: arn:aws:cloudformation:us-west-2:100137374868:stack/DMSdemo/23aa30e0-6628-11e6-bd48-50d5ca0184d2

Status: CREATE\_COMPLETE

Status reason:

Description:

### ▼ Outputs

Key	Value	Description
OracleJDBCConnectionString	<code>jdbc:oracle:thin:@do1xaskn0mfp18y.cqiw4tcs0mg7.us-west-2.rds.amazonaws.com:1521:ORCL</code>	JDBC connection string for Oracle database
Regionname	us-west-2	
StackName	DMSdemo	
AuroraJDBCConnectionString	<code>jdbc:mysql://dmsdemo-auroracluster-1u1oynqy35jvw.cluster-cqiw4tcs0mg7.us-west-2.rds.amazonaws.com:3306</code>	JDBC connection string for Aurora database

### Note

Oracle 12c SE Two License version 12.1.0.2.v4 is available in all regions. However, Amazon Aurora MySQL is not available in all regions. Amazon Aurora MySQL is currently available in US East (N. Virginia), US West (Oregon), EU (Ireland), Asia Pacific (Tokyo), Asia Pacific (Mumbai), Asia Pacific (Sydney), and Asia Pacific (Seoul). If you try to create a stack in a region where Aurora MySQL is not available, creation fails with the error `Invalid DB Engine for AuroraCluster`.

## Step 2: Install the SQL Tools and AWS Schema Conversion Tool on Your Local Computer

Next, you need to install a SQL client and the AWS Schema Conversion Tool (AWS SCT) on your local computer.

This walkthrough assumes you will use the SQL Workbench/J client to connect to the RDS instances for migration validation. A few other software tools you might want to consider are the following:

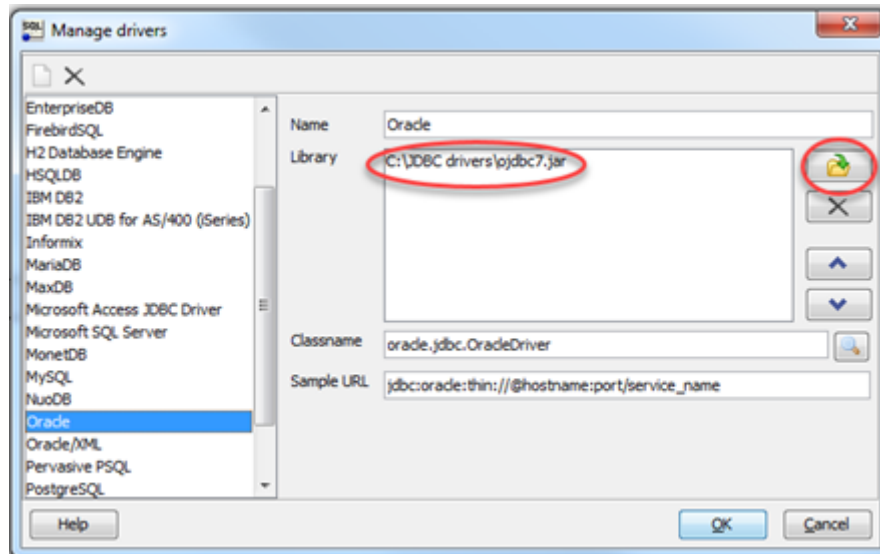
- [JACK DB](#), an online web interface to work with RDS databases (Oracle and Aurora MySQL) over JDBC
- [DBVisualizer](#)
- [Oracle SQL Developer](#)

To install the SQL client software, do the following:

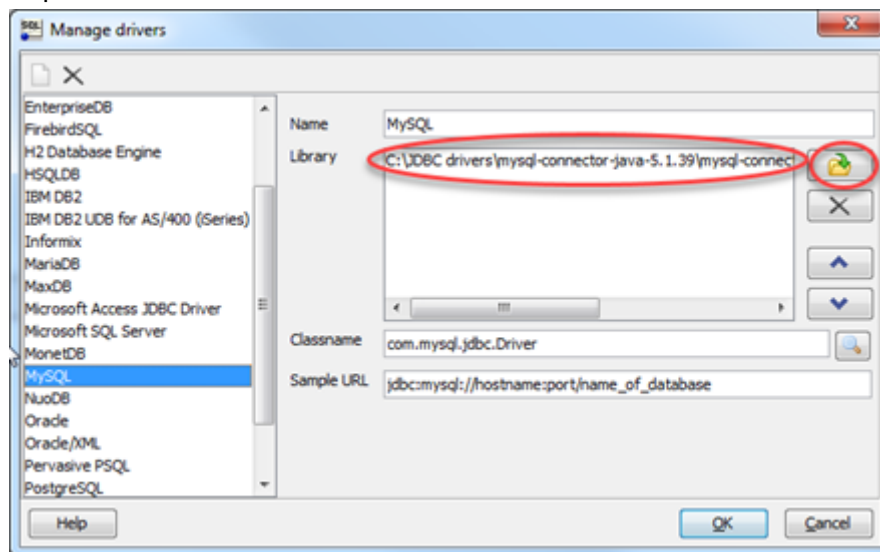


AWS Database Migration Service  
Step-by-Step Migration Guide  
Step 2: Install the SQL Tools and AWS Schema  
Conversion Tool on Your Local Computer

1. Download SQL Workbench/J from [the SQL Workbench/J website](#), and then install it on your local computer. This SQL client is free, open-source, and DBMS-independent.
2. Download the JDBC driver for your Oracle database release. For more information, go to <https://www.oracle.com/jdbc>.
3. Download the MySQL JDBC driver (#0#jar file). For more information, go to <https://dev.mysql.com/downloads/connector/j/>.
4. Using SQL Workbench/J, configure JDBC drivers for Oracle and Aurora MySQL to set up connectivity, as described following.
  - a. In SQL Workbench/J, choose **File**, then choose **Manage Drivers**.
  - b. From the list of drivers, choose **Oracle**.
  - c. Choose the Open icon, then choose the #0#jar file for the Oracle JDBC driver that you downloaded in the previous step. Choose **OK**.



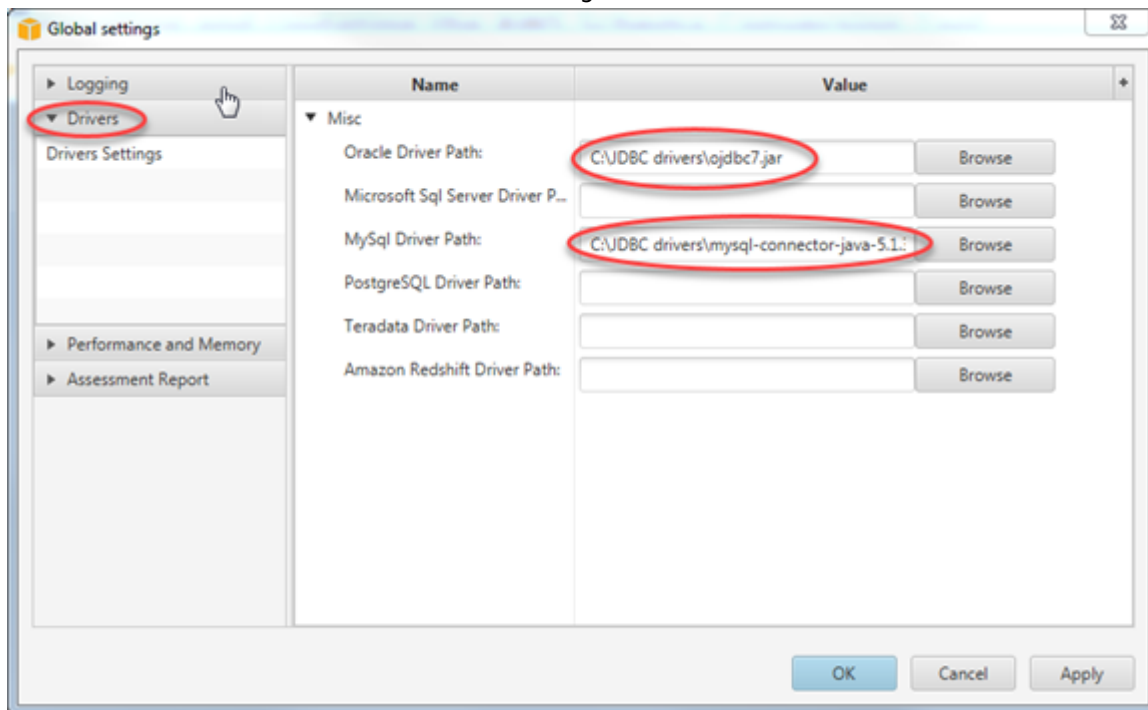
- d. From the list of drivers, choose MySQL.
    - e. Choose the Open icon, then choose the MySQL JDBC driver that you downloaded in the previous step. Choose **OK**.



AWS Database Migration Service  
Step-by-Step Migration Guide  
Step 3: Test Connectivity to the Oracle DB  
Instance and Create the Sample Schema

To install the AWS Schema Migration Tool and the required JDBC drivers, do the following:

1. Download the AWS Schema Conversion Tool from [Installing and Updating the AWS Schema Conversion Tool](#) in the *AWS Schema Conversion Tool User Guide*. By default, the tool is installed in the "C:\Program Files\AWS Schema Conversion Tool\AWS directory.
2. Launch the AWS Schema Conversion Tool.
3. In the AWS Schema Conversion Tool, choose **Global Settings** from **Settings**.
4. In **Global Settings**, choose **Driver**, and then choose **Browse** for **Oracle Driver Path**. Locate the JDBC Oracle driver and choose **OK**. Next, choose **Browse** for **MySQL Driver Path**. Locate the JDBC MySQL driver and choose **OK**. Choose **OK** to close the dialog box.



## Step 3: Test Connectivity to the Oracle DB Instance and Create the Sample Schema

After the CloudFormation stack has been created, test the connection to the Oracle DB instance by using SQL Workbench/J and then create the HR sample schema.

To test the connection to your Oracle DB instance and create the sample schema, do the following:

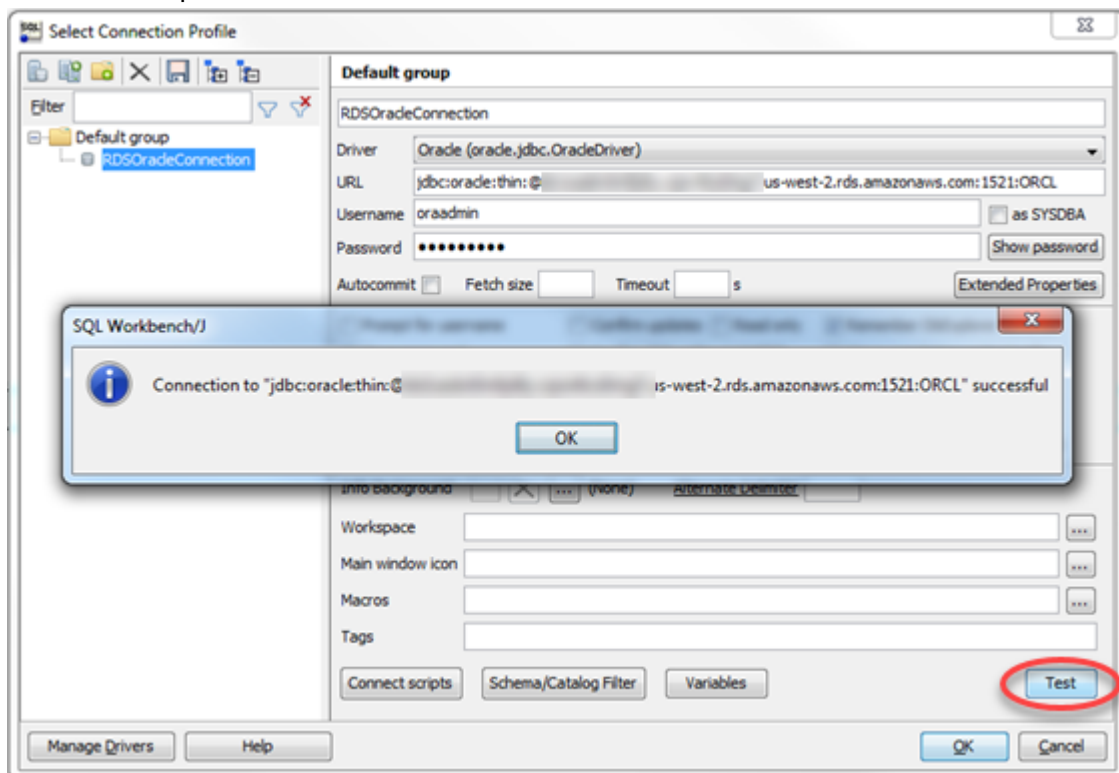
1. In SQL Workbench/J, choose **File**, then choose **Connect window**. Create a new connection profile using the following information as shown following

For This Parameter	Do This
New profile name	Type RDSOracleConnection.
Driver	Choose Oracle (oracle.jdbc.OracleDriver).

AWS Database Migration Service  
 Step-by-Step Migration Guide  
 Step 3: Test Connectivity to the Oracle DB  
 Instance and Create the Sample Schema

For This Parameter	Do This
<b>URL</b>	Use the <b>OracleJDBCConnectionString</b> value you recorded when you examined the output details of the DMSdemo stack in a previous step.
<b>Username</b>	Type oraadmin.
<b>Password</b>	Provide the password for the admin user that you assigned when creating the Oracle DB instance using the AWS CloudFormation template.

2. Test the connection by choosing **Test**. Choose **OK** to close the dialog box, then choose OK to create the connection profile.



**Note**

If your connection is unsuccessful, ensure that the IP address you assigned when creating the CloudFormation template is the one you are attempting to connect from. This is the most common issue when trying to connect to an instance.

3. Create the HR schema you will use for migration using a custom SQL script (Oracle-HR-Schema-Build.sql). To obtain this script, do the following:
  - a. Download the following archive to your computer: <http://docs.aws.amazon.com/dms/latest/sbs/samples/dms-sbs-RDSOracle2Aurora.zip>
  - b. Extract the SQL script(Oracle-HR-Schema-Build.sql) from the archive.
  - c. Copy and paste the Oracle-HR-Schema-Build.sql file into your current directory.
4. Open the provided SQL script in a text editor. Copy the entire script.
5. In SQL Workbench/J, paste the SQL script in the Default.wksp window showing **Statement 1**.
6. Choose **SQL**, then choose **Execute All**.

AWS Database Migration Service  
Step-by-Step Migration Guide  
Step 3: Test Connectivity to the Oracle DB  
Instance and Create the Sample Schema

When you run the script, you will get an error message indicating that user HR does not exist. You can ignore this error and run the script. The script drops the user before creating it, which generates the error.

7. Verify the object types and count in HR Schema were created successfully by running the following SQL query.

```
Select OBJECT_TYPE, COUNT(*) from dba_OBJECTS where owner='HR'  
GROUP BY OBJECT_TYPE;
```

The results of this query should be similar to the following:

OBJECT_TYPE	COUNT(*)
INDEX	8
PROCEDURE	2
SEQUENCE	3
TABLE	7
VIEW	1

8. Verify the number of constraints in HR schema by running the following SQL query:

```
Select CONSTRAINT_TYPE, COUNT(*) from dba_constraints where owner='HR'  
AND (CONSTRAINT_TYPE IN ('P', 'R') OR SEARCH_CONDITION_VC NOT LIKE '%NOT NULL%')  
GROUP BY CONSTRAINT_TYPE;
```

The results of this query should be similar to the following:

CONSTRAINT_TYPE	COUNT(*)
R	10
P	7
C	1

9. Analyze the HR schema by running the following:

```
BEGIN  
  dbms_stats.gather_schema_stats('HR');  
END;  
/
```

10. Verify the total number of tables and number of rows for each table by running the following SQL query:

```
SELECT table_name, num_rows from dba_tables where owner='HR' order by 1;
```

The results of this query should be similar to the following:

TABLE_NAME	NUM_ROWS
COUNTRIES	25
DEPARTMENTS	27
EMPLOYEES	107
JOBS	19
JOB_HISTORY	10
LOCATIONS	23
REGIONS	4

11. Verify the relationships of the tables. Check the departments with employees greater than 10 by running the following SQL query:

AWS Database Migration Service  
Step-by-Step Migration Guide  
Step 4: Test the Connectivity to  
the Aurora MySQL DB Instance

```
Select b.department_name,count(*) from HR.Employees a,HR.departments b where  
a.department_id=b.department_id  
group by b.department_name having count(*) > 10  
order by 1;
```

The results of this query should be similar to the following:

DEPARTMENT_NAME	COUNT(*)
Sales	34
Shipping	45

## Step 4: Test the Connectivity to the Aurora MySQL DB Instance

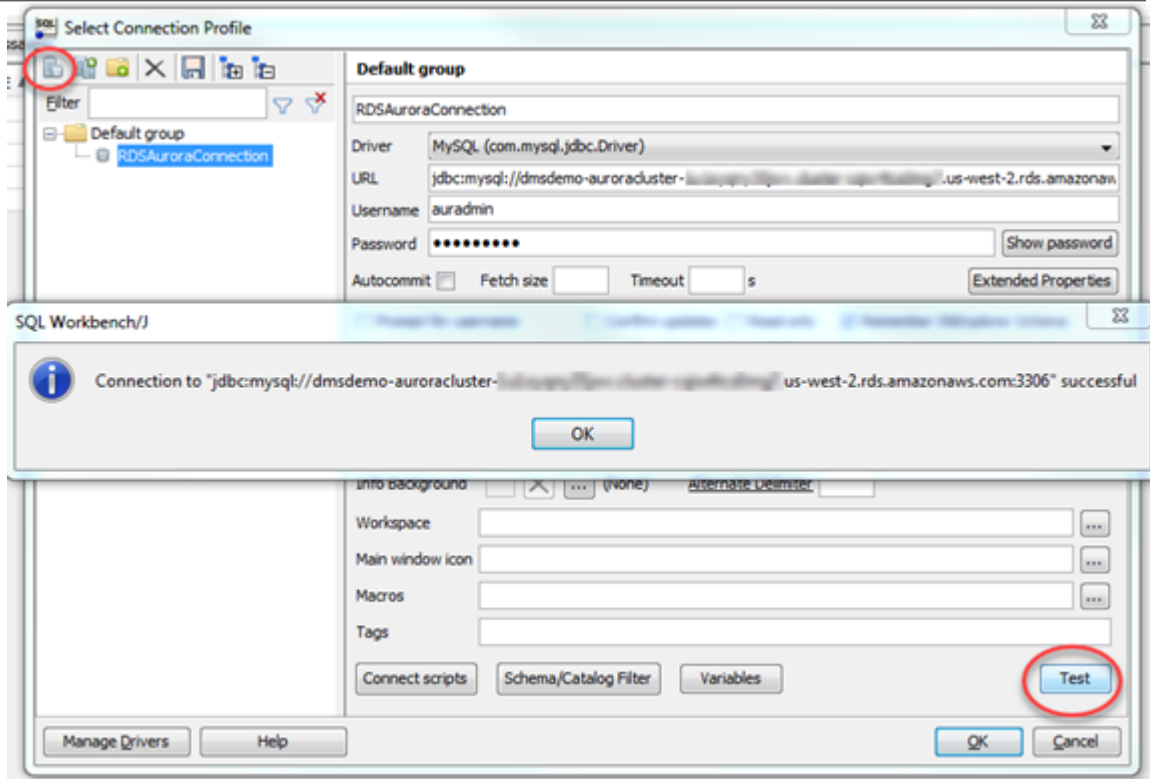
Next, test your connection to your Aurora MySQL DB instance.

1. In SQL Workbench/J, choose **File**, then choose **Connect window**. Choose the Create a new connection profile icon. using the following information: Connect to the Aurora MySQL DB instance in SQL Workbench/J by using the information as shown following

For This Parameter	Do This
<b>New profile name</b>	Type <code>RDSAuroraConnection</code> .
<b>Driver</b>	Choose <code>MySQL (com.mysql.jdbc.Driver)</code> .
<b>URL</b>	Use the <code>AuroraJDBCConnectionString</code> value you recorded when you examined the output details of the DMSdemo stack in a previous step.
<b>Username</b>	Type <code>auradmin</code> .
<b>Password</b>	Provide the password for the admin user that you assigned when creating the Aurora MySQL DB instance using the AWS CloudFormation template.

2. Test the connection by choosing **Test**. Choose **OK** to close the dialog box, then choose **OK** to create the connection profile.

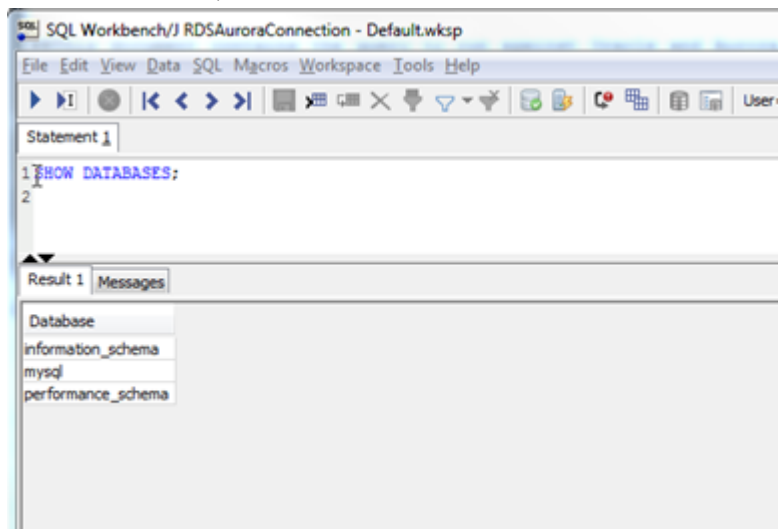
AWS Database Migration Service  
Step-by-Step Migration Guide  
Step 4: Test the Connectivity to  
the Aurora MySQL DB Instance



**Note**

If your connection is unsuccessful, ensure that the IP address you assigned when creating the CloudFormation template is the one you are attempting to connect from. This is the most common issue when trying to connect to an instance.

3. Log on to the Aurora MySQL instance by using the master admin credentials.
4. Verify your connectivity to the Aurora MySQL DB instance by running a sample SQL command, such as `SHOW DATABASES ;`.



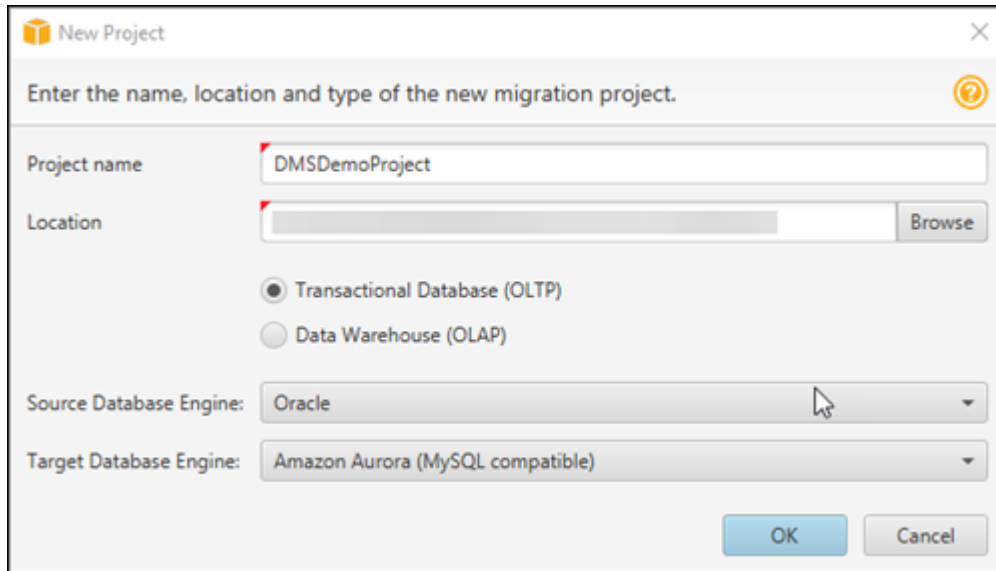
## Step 5: Use the AWS Schema Conversion Tool (AWS SCT) to Convert the Oracle Schema to Aurora MySQL

Before you migrate data to Aurora MySQL, you convert the Oracle schema to an Aurora MySQL schema.

To convert an Oracle schema to an Aurora MySQL schema using AWS Schema Conversion Tool (AWS SCT), do the following:

1. Launch the AWS Schema Conversion Tool (AWS SCT). In the AWS SCT, choose **File**, then choose **New Project**. Create a new project called `DMSDemoProject`. Enter the following information in the New Project window and then choose **OK**.

For This Parameter	Do This
<b>Project Name</b>	Type <code>DMSDemoProject</code> .
<b>Location</b>	Use the default <b>Projects</b> folder and the default <b>Transactional Database (OLTP)</b> option.
<b>Source Database Engine</b>	Choose <b>Oracle</b> .
<b>Target Database Engine</b>	Choose <b>Amazon Aurora (MySQL Compatible)</b> .

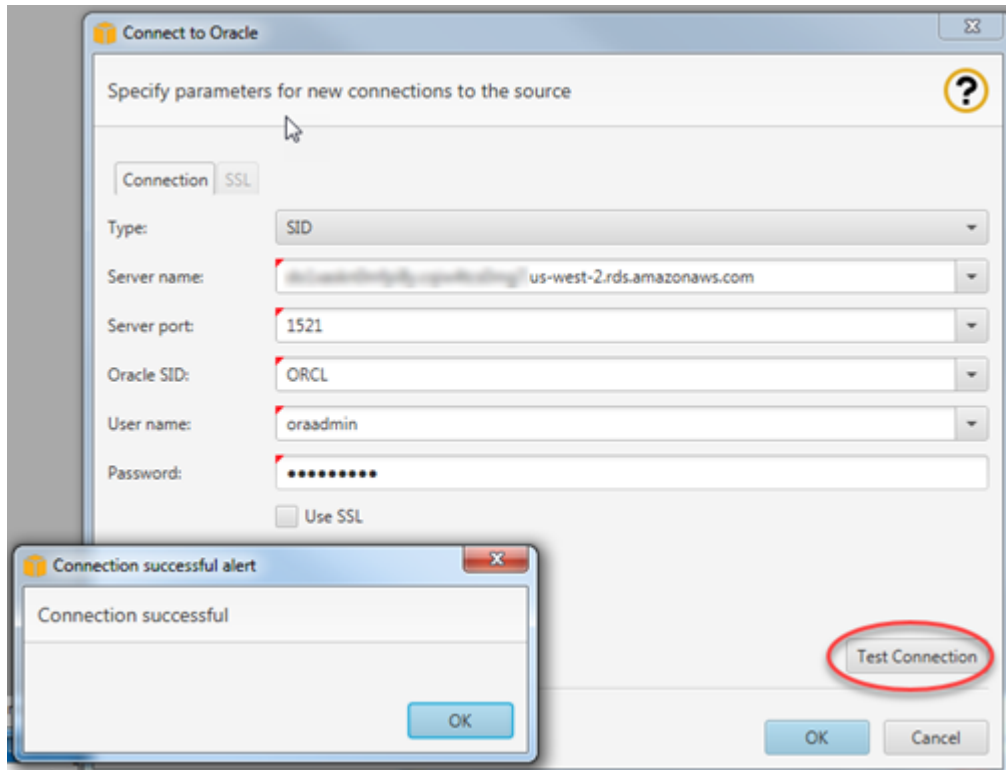


2. Choose **Connect to Oracle**. In the **Connect to Oracle** dialog box, enter the following information, and then choose **Test Connection**.

For This Parameter	Do This
<b>Type</b>	Choose <b>SID</b> .
<b>Server name</b>	Use the <b>OracleJDBCConnectionString</b> value you used to connect to the Oracle DB instance, but remove the JDBC prefix information. For example, a sample connection string

AWS Database Migration Service  
 Step-by-Step Migration Guide  
 Step 5: Use the AWS Schema Conversion Tool (AWS  
 SCT) to Convert the Oracle Schema to Aurora MySQL

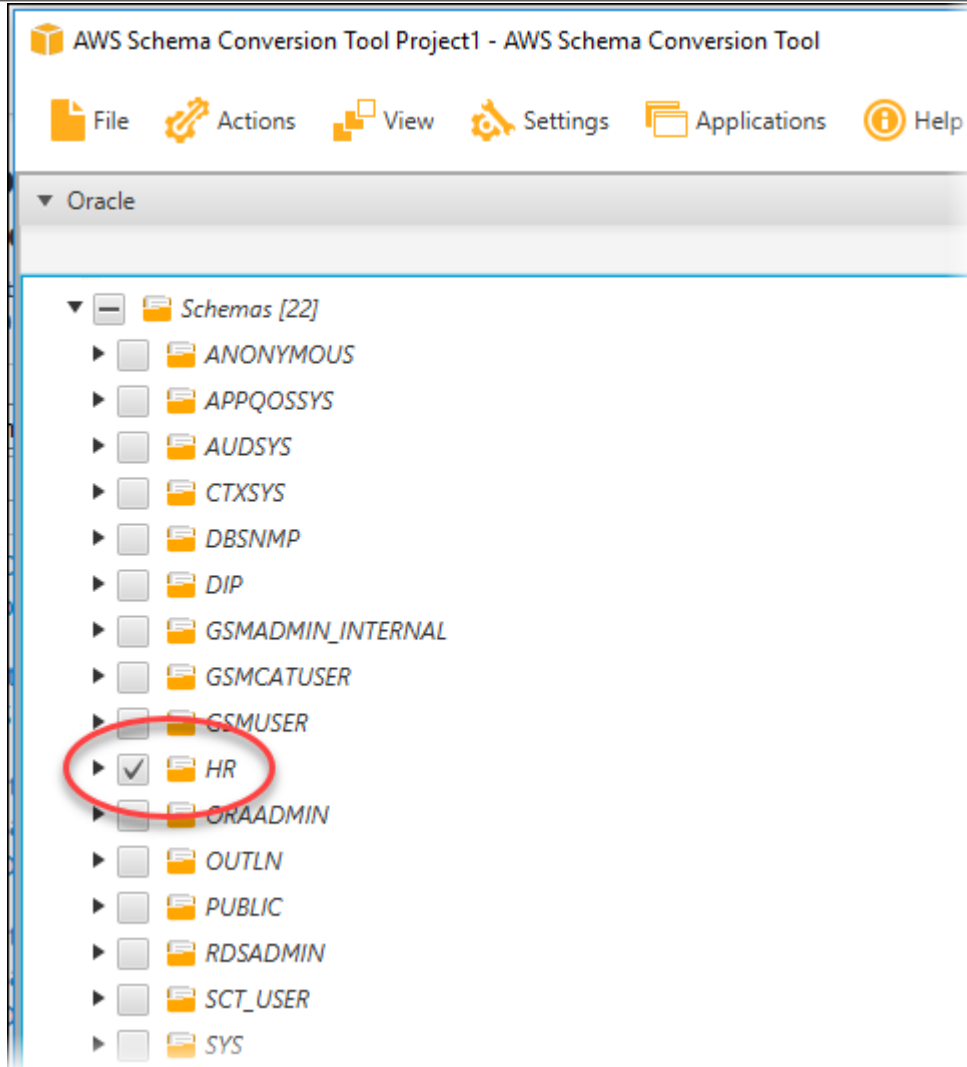
For This Parameter	Do This
	you use with SQL Workbench/J might be "jdbc:oracle:thin:@do1xa4grferti8y.cqiw4tcs0mg7.us-west-2.rds.amazonaws.com:1521:ORCL". For the AWS SCT <b>Server name</b> , you remove "jdbc:oracle:thin:@/" and ":1521" to use just the server name: "do1xa4grferti8y.cqiw4tcs0mg7.us-west-2.rds.amazonaws.com"
<b>Server port</b>	Type 1521.
<b>Oracle SID</b>	Type ORCL.
<b>User name</b>	Type oraadmin.
<b>Password</b>	Provide the password for the admin user that you assigned when creating the Oracle DB instance using the AWS CloudFormation template.



3. Choose **OK** to close the alert box, then choose OK to close the dialog box and to start the connection to the Oracle DB instance. The database structure of the Oracle DB instance is shown. Select only the HR schema.



AWS Database Migration Service  
 Step-by-Step Migration Guide  
 Step 5: Use the AWS Schema Conversion Tool (AWS  
 SCT) to Convert the Oracle Schema to Aurora MySQL

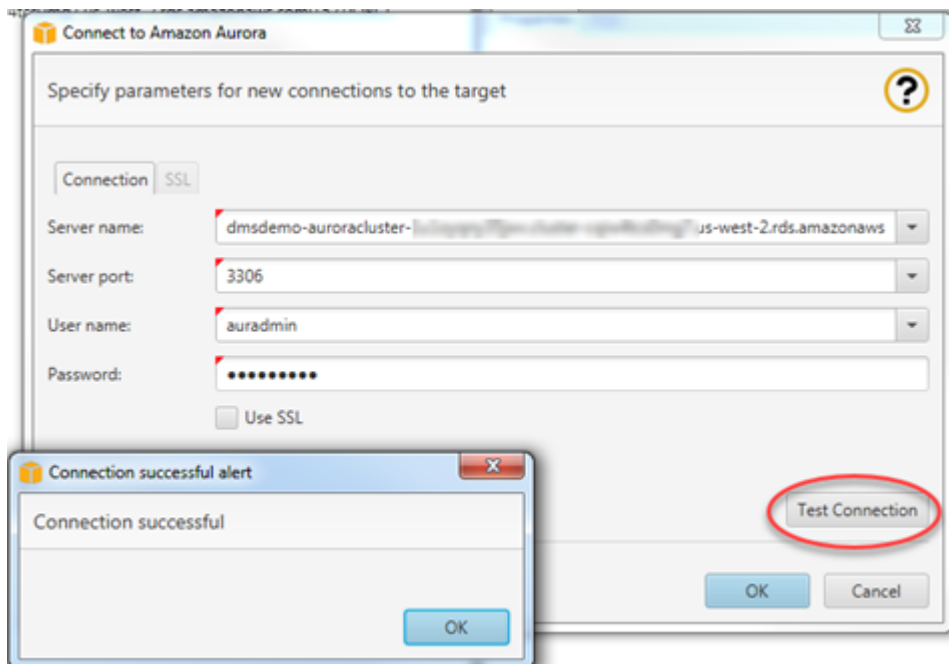


4. Choose **Connect to Amazon Aurora**. In the **Connect to Amazon Aurora** dialog box, enter the following information and then choose **Test Connection**.

For This Parameter	Do This
Type	Choose <b>SID</b> .
Server name	Use the <b>AuroraJDBCConnectionString</b> value you used to connect to the Aurora MySQL DB instance, but remove the JDBC prefix information and the port suffix. For example, a sample connection string you use with SQL Workbench/J might be "jdbc:mysql://dmsdemo-auroracluster-1u1ogdfg35v.cluster-cqiw4tcs0mg7.us-west-2.rds.amazonaws.com:3306". For the AWS SCT <b>Server name</b> , you remove "jdbc:mysql://" and ":3306" to use just the server name:

AWS Database Migration Service  
 Step-by-Step Migration Guide  
 Step 5: Use the AWS Schema Conversion Tool (AWS  
 SCT) to Convert the Oracle Schema to Aurora MySQL

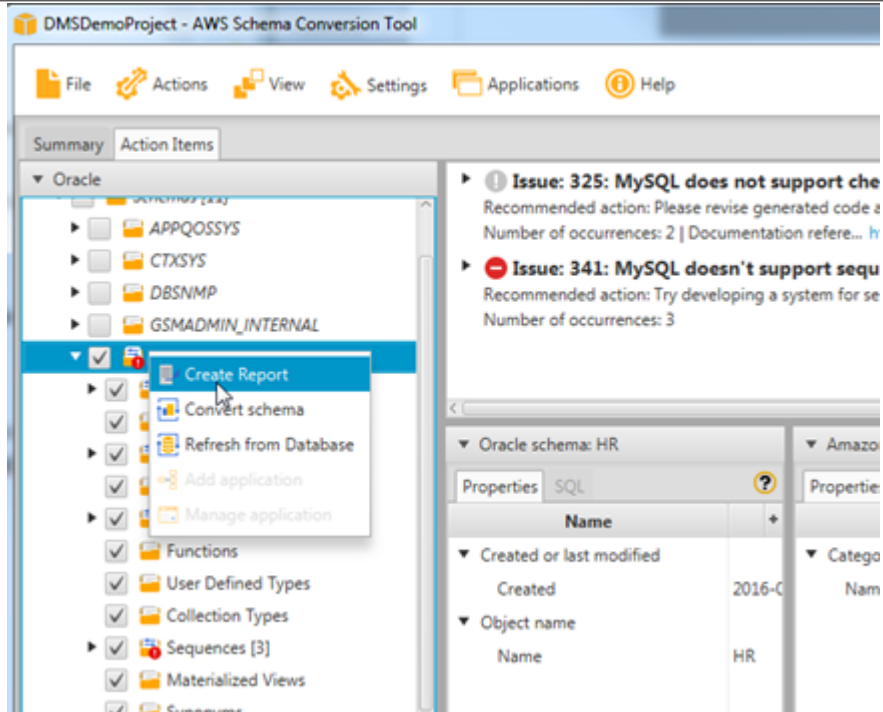
For This Parameter	Do This
	"dmsdemo-auroracluster-1u1ogdfg35v.cluster-cqiw4tcs0mg7.us-west-2.rds.amazonaws.com"
<b>Server port</b>	Type 3306.
<b>User name</b>	Type auradmin.
<b>Password</b>	Provide the password for the admin user that you assigned when creating the Oracle DB instance using the AWS CloudFormation template.



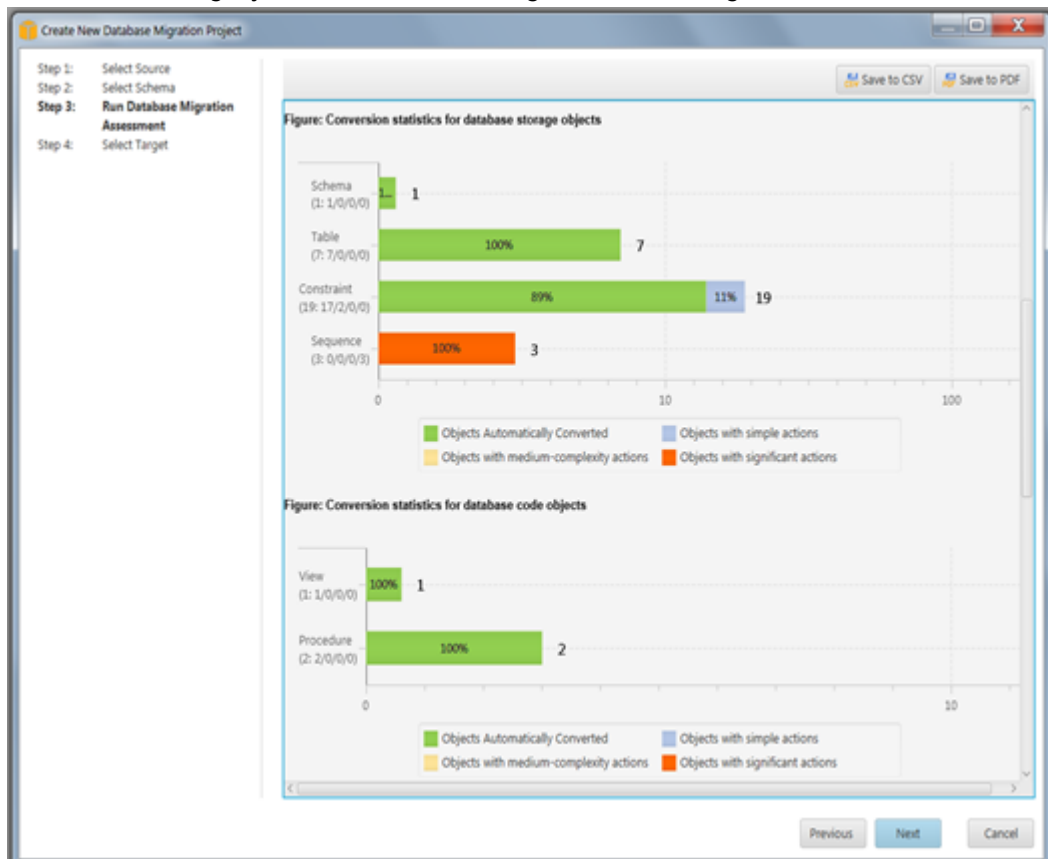
AWS SCT analyses the HR schema and creates a database migration assessment report for the conversion to Amazon Aurora MySQL.

5. Choose **OK** to close the alert box, then choose **OK** to close the dialog box to start the connection to the Amazon Aurora MySQL DB instance.
6. Right-click the HR schema and select **Create Report**.

AWS Database Migration Service  
 Step-by-Step Migration Guide  
 Step 5: Use the AWS Schema Conversion Tool (AWS  
 SCT) to Convert the Oracle Schema to Aurora MySQL



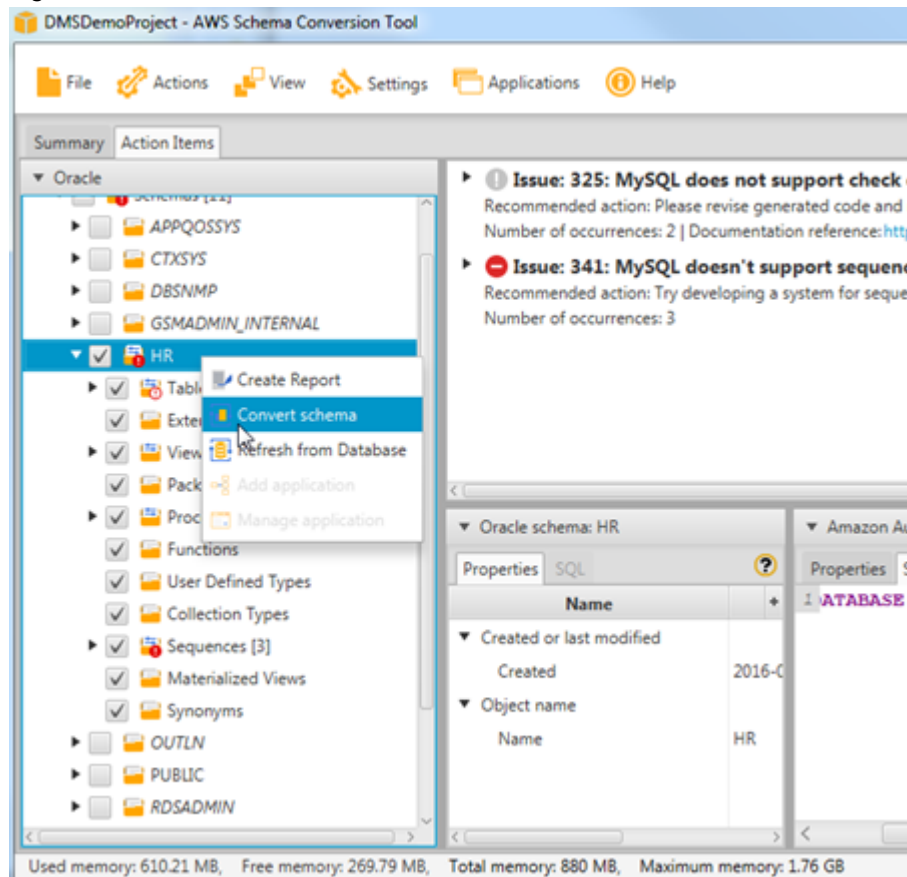
7. Check the report and the action items it suggests. The report discusses the type of objects that can be converted by using AWS SCT, along with potential migration issues and actions to resolve these issues. For this walkthrough, you should see something like the following:



AWS Database Migration Service  
Step-by-Step Migration Guide  
Step 5: Use the AWS Schema Conversion Tool (AWS  
SCT) to Convert the Oracle Schema to Aurora MySQL

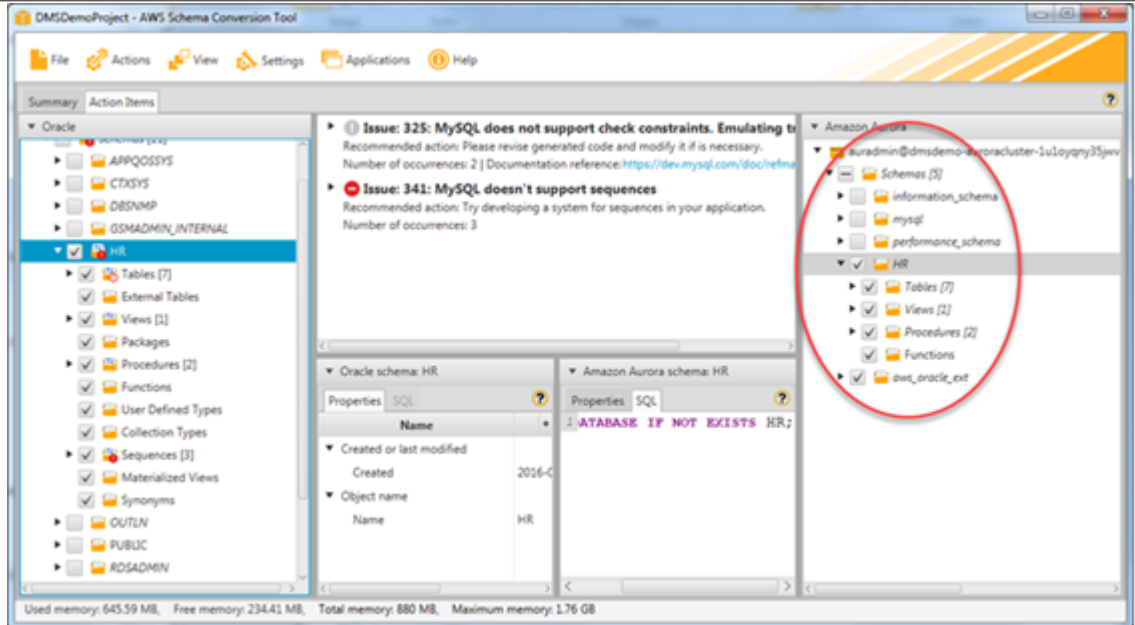
You can optionally save the report as .csv or .pdf format for later analysis.

8. Choose the **Action Items** tab, and review any recommendations that you see.
9. Right-click the HR schema, and then choose **Convert schema**.

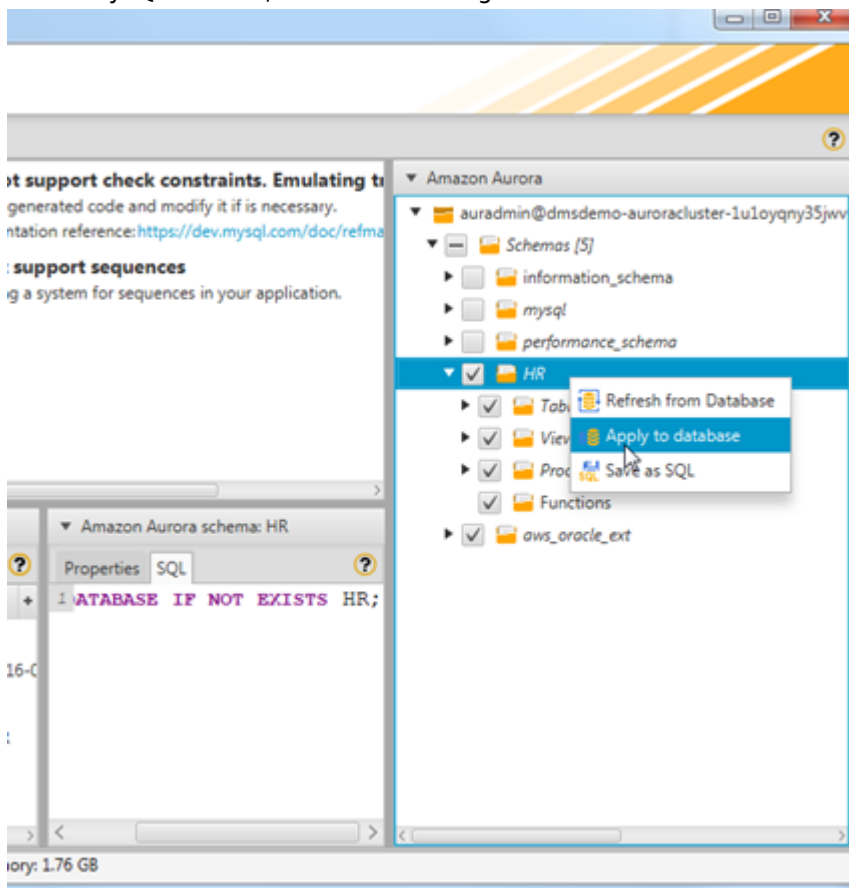


10. Choose **Yes** for the confirmation message. AWS SCT then converts your schema to the target database format.

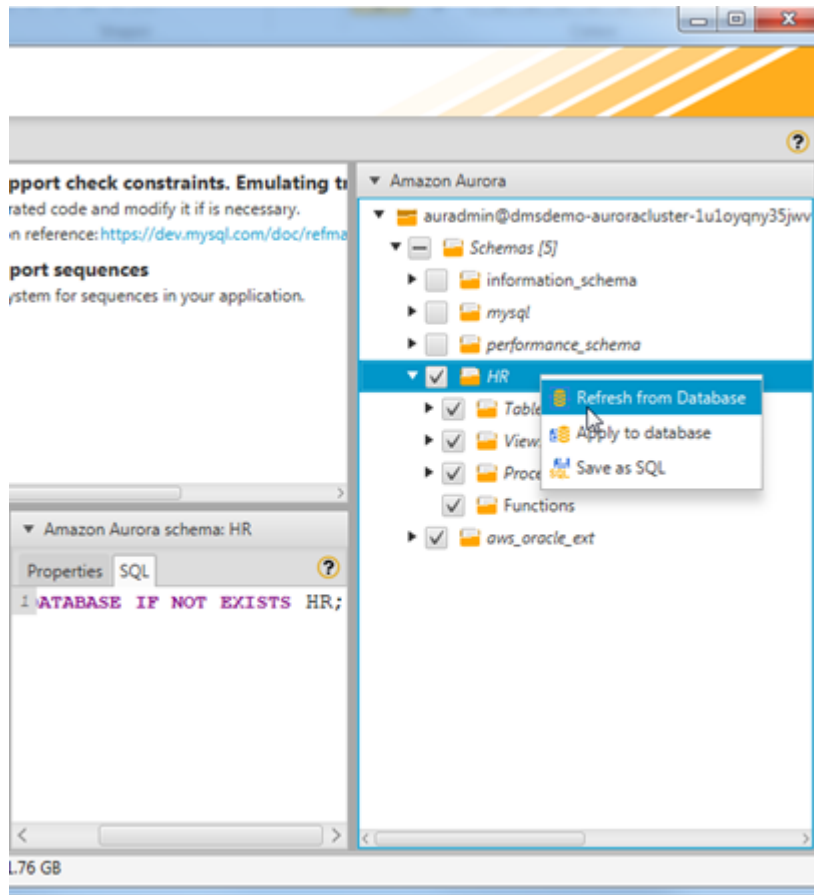
AWS Database Migration Service  
Step-by-Step Migration Guide  
Step 5: Use the AWS Schema Conversion Tool (AWS  
SCT) to Convert the Oracle Schema to Aurora MySQL



11 Choose the HR schema, and then choose **Apply to database** to apply the schema scripts to the target Aurora MySQL instance, as shown following.



12 Choose the HR schema, and then choose **Refresh from Database** to refresh from the target database, as shown following.



The database schema has now been converted and imported from source to target.

## Step 6: Validate the Schema Conversion

To validate the schema conversion, you compare the objects found in the Oracle and Aurora MySQL databases using SQL Workbench/J.

1. In SQL Workbench/J, choose **File**, then choose **Connect window**. Choose the RDSAuroraConnection you created in an earlier step. Click **OK**.
2. Run the following script to verify the number of object types and count in HR schema in the target Aurora MySQL database. These values should match the number of objects in the source Oracle database:

```
SELECT a.OBJECT_TYPE, COUNT(*)
FROM
(
SELECT OBJECT_TYPE
,OBJECT_SCHEMA
,OBJECT_NAME
FROM (
SELECT 'TABLE' AS OBJECT_TYPE
,TABLE_NAME AS OBJECT_NAME
,TABLE_SCHEMA AS OBJECT_SCHEMA
FROM information_schema.TABLES
where TABLE_TYPE='BASE TABLE'
UNION
```

AWS Database Migration Service  
Step-by-Step Migration Guide  
Step 6: Validate the Schema Conversion

---

```
SELECT 'VIEW' AS OBJECT_TYPE
, TABLE_NAME AS OBJECT_NAME
, TABLE_SCHEMA AS OBJECT_SCHEMA
FROM information_schema.VIEWS
UNION

SELECT 'INDEX' AS OBJECT_TYPE
, CONCAT (
CONSTRAINT_TYPE
, ' : '
, CONSTRAINT_NAME
, ' : '
, TABLE_NAME
) AS OBJECT_NAME
, TABLE_SCHEMA AS OBJECT_SCHEMA
FROM information_schema.TABLE_CONSTRAINTS
where constraint_type='PRIMARY KEY'
UNION

SELECT ROUTINE_TYPE AS OBJECT_TYPE
, ROUTINE_NAME AS OBJECT_NAME
, ROUTINE_SCHEMA AS OBJECT_SCHEMA
FROM information_schema.ROUTINES
UNION

SELECT 'TRIGGER' AS OBJECT_TYPE
, CONCAT (
TRIGGER_NAME
, ' : '
, EVENT_OBJECT_SCHEMA
, ' : '
, EVENT_OBJECT_TABLE
) AS OBJECT_NAME
, TRIGGER_SCHEMA AS OBJECT_SCHEMA
FROM information_schema.triggers
) R
WHERE R.OBJECT_SCHEMA = 'HR'
order by 1) a
GROUP BY a.OBJECT_TYPE;
```

The output from this query should be similar to the following:

OBJECT_TYPE	COUNT(*)
INDEX	7
PROCEDURE	2
TABLE	7
TRIGGER	10
VIEW	1

Next, run the following query to get table constraints information:

```
SELECT CONSTRAINT_TYPE, COUNT(*)
FROM information_schema.TABLE_CONSTRAINTS where constraint_schema='HR'
GROUP BY CONSTRAINT_TYPE;
```

The output from this query should be similar to the following:

CONSTRAINT_TYPE	COUNT(*)
FOREIGN KEY	10
PRIMARY KEY	7
UNIQUE	7

## Step 7: Create a AWS DMS Replication Instance

After we validate the schema structure between source and target databases, as described preceding, we proceed to the core part of this walkthrough, which is the data migration. The following illustration shows a high-level view of the migration process.



A DMS replication instance performs the actual data migration between source and target. The replication instance also caches the transaction logs during the migration. How much CPU and memory capacity a replication instance has influences the overall time required for the migration.

To create an AWS DMS replication instance, do the following:

1. Sign in to the AWS Management Console, select [Database Migration Service \(AWS DMS\)](#) and choose **Create replication instance**. If you are signed in as an AWS Identity and Access Management (IAM) user, you must have the appropriate permissions to access AWS DMS. For more information on the permissions required, see [IAM Permissions Needed to Use AWS DMS](#).
2. On the **Create replication instance** page, specify your replication instance information as shown following.

For This Parameter	Do This
<b>Name</b>	Type <code>DMSdemo-repserver</code> .
<b>Descriptive Amazon Resource Name (ARN)</b>	Skip this optional field.
<b>Description</b>	Type a brief description, such as <code>DMS demo replication server</code> .
<b>Instance class</b>	Choose <code>dms.t3.medium</code> . This instance class is large enough to migrate a small set of tables.
<b>Engine version</b>	Choose <code>3.4.5</code> . This is the latest AWS DMS version, which includes all new features and enhancements.
<b>Allocated storage (GiB)</b>	Choose <code>50</code> . This storage space is enough for your migration project.
<b>VPC</b>	Choose <code>DMSDemoVPC</code> , which is the VPC that was created by the CloudFormation stack.
<b>Multi-AZ</b>	Choose <code>Dev or test workload (Single-AZ)</code> .
<b>Publicly accessible</b>	Leave this item selected.

3. For the **Advanced**, **Maintenance**, and **Tags** sections, leave the default settings as they are, and choose **Create**.



## Step 8: Create AWS DMS Source and Target Endpoints

While your replication instance is being created, you can specify the source and target database endpoints using the AWS Management Console. However, you can only test connectivity after the replication instance has been created, because the replication instance is used in the connection.

1. Specify your connection information for the source Oracle database and the target Amazon Aurora MySQL database. The following table describes the source settings.

For This Parameter	Do This
<b>Endpoint Identifier</b>	Type <code>Orasource</code> (the Amazon RDS Oracle endpoint).
<b>Source Engine</b>	Choose <b>oracle</b> .
<b>Server name</b>	Provide the Oracle DB instance name. This is the <b>Server name</b> you used for AWS SCT, such as "do1xa4grferti8y.cqiw4tcs0mg7.us-west-2.rds.amazonaws.com".
<b>Port</b>	Type 1521.
<b>SSL mode</b>	Choose <b>None</b> .
<b>Username</b>	Type <code>oraadmin</code> .
<b>Password</b>	Provide the password for the Oracle DB instance.
<b>SID</b>	Provide the Oracle database name.

The following table describes the target settings.

For This Parameter	Do This
<b>Endpoint Identifier</b>	Type <code>Aurtarget</code> (the Amazon Aurora MySQL endpoint).
<b>Target Engine</b>	Choose <b>aurora</b> .
<b>Servername</b>	Provide the Aurora MySQL DB instance name. This is the <b>Server name</b> you used for AWS SCT, such as "dmsdemo-auroracluster-1u1oyqny35jvw.cluster-cqiw4tcs0mg7.us-west-2.rds.amazonaws.com".
<b>Port</b>	Type 3306.
<b>SSL mode</b>	Choose <b>None</b> .
<b>Username</b>	Type <code>auradmin</code> .
<b>Password</b>	Provide the password for the Aurora MySQL DB instance.

The completed page should look like the following:

## Connect source and target database endpoints

✔ Replication instance created successfully.

Your database endpoint can be on-premise, in EC2, RDS or in the cloud. Define the connection details below. It is recommended that you test your endpoint connections here to avoid errors later.

### Source database connection details

Endpoint identifier\*  ⓘ

Source engine\*  ⓘ

Server name\*

Port\*  ⓘ

SSL mode\*  ⓘ

User name\*

Password\*

SID\*

▶ Advanced

### Target database connection details

Endpoint identifier\*  ⓘ

Target engine\*  ⓘ

Server name\*

Port\*  ⓘ

SSL mode\*  ⓘ

User name\*

Password\*

▶ Advanced

Run test

2. In order to disable foreign key checks during the initial data load, you must add the following commands to the target Aurora MySQL DB instance. In the **Advanced** section, shown following, type the following commands for **Extra connection attributes**: `initstmt=SET FOREIGN_KEY_CHECKS=0;autocommit=1`

The first command disables foreign key checks during a load, and the second command commits the transactions that DMS executes.

SID\*

▶ **Advanced**

✔ Connection tested successfully

▼ **Advanced**

Extra connection attributes

KMS master key  ⓘ

**Description** Default master key that protects my DMS replication instance volumes when no other key is defined

**Account** 100137374868

**Key ARN** arn:aws:kms:us-west-2:100137374868:key/ef0d4b5e-a9ad-74830e851659

✔ Connection tested successfully

3. Choose **Next**.

## Step 9: Create and Run Your AWS DMS Migration Task

Using a AWS DMS task, you can specify what schema to migrate and the type of migration. You can migrate existing data, migrate existing data and replicate ongoing changes, or replicate data changes only. This walkthrough migrates existing data only.

1. On the **Create Task** page, specify the task options. The following table describes the settings.

For This Parameter	Do This
<b>Task name</b>	Type migratehrschem.
<b>Task description</b>	Type a description for the task.
<b>Source endpoint</b>	Shows orasource (the Amazon RDS Oracle endpoint).
<b>Target endpoint</b>	Shows aurtarget (the Amazon Aurora MySQL endpoint).

For This Parameter	Do This
Replication instance	Shows <code>DMSdemo-repserver</code> (the AWS DMS replication instance created in an earlier step).
Migration type	Choose the option <b>Migrate existing data</b> .
Start task on create	Select this option.

The page should look like the following:

## Create task

A task can contain one or more table mappings which define what data is moved from the source to the target. If a table does not exist on the target, it can be created automatically.

Task name\*  ⓘ

Task description\*  ⓘ

Source endpoint orasource

Target endpoint aurtarget

Replication instance dmsdemo-repserver

Migration type\*  ⓘ

Start task on create

▶ Task Settings

▶ Table mappings

[Cancel](#) [Previous](#) [Create task](#)

2. Under **Task Settings**, choose **Do nothing** for **Target table preparation mode**, because you have already created the tables through Schema Migration Tool. Because this migration doesn't contain any LOBs, you can leave the LOB settings at their defaults.

Optionally, you can select **Enable logging**. If you enable logging, you will incur additional Amazon CloudWatch charges for the creation of CloudWatch logs. For this walkthrough, logs are not necessary.

▼ Task Settings

Target table preparation mode\*  Do nothing ⓘ  
 Drop tables on target  
 Truncate

Include LOB columns in replication\*  Don't include LOB columns ⓘ  
 Full LOB mode  
 Limited LOB mode

Max LOB size (kb)\* 32 ⓘ

Enable logging

[Advanced Settings](#)

▶ Table mappings

[Cancel](#) [Previous](#) [Create task](#)

3. Leave the Advanced settings at their default values.
4. Choose **Table mappings**, choose **Default** for **Mapping method**, and then choose **HR** for **Schema to migrate**.

The completed section should look like the following.

Enable logging

[Advanced Settings](#)

▼ Table mappings

Mapping method\*  Default  Custom ⓘ

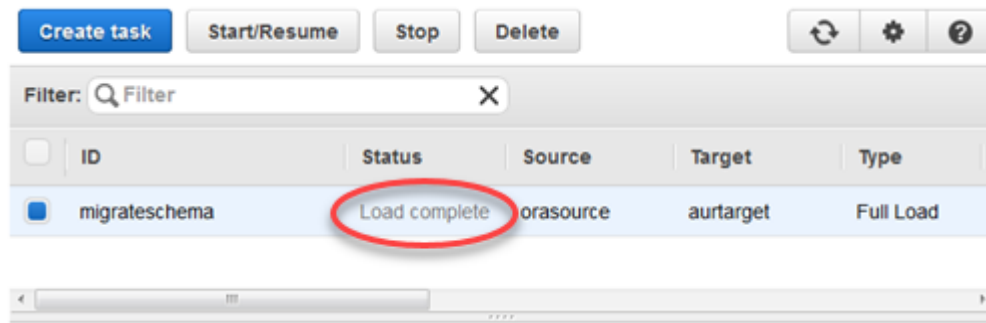
Schema to migrate **HR** ⓘ  
DMS will create the schema on the target if it does not already exist

[Show JSON](#)

[Cancel](#) [Previous](#) [Create task](#)

5. Choose **Create task**. The task will begin immediately.

The Tasks section shows you the status of the migration task.



The screenshot shows the AWS Database Migration Service console. At the top, there are buttons for 'Create task', 'Start/Resume', 'Stop', and 'Delete'. Below these are three icons: a refresh icon, a settings icon, and a help icon. A search bar labeled 'Filter: Filter' is present. Below the search bar is a table with columns: ID, Status, Source, Target, and Type. The table contains one row with the following data: ID: migrateschema, Status: Load complete (circled in red), Source: orasource, Target: aurtarget, Type: Full Load. A scrollbar is visible at the bottom of the table.

ID	Status	Source	Target	Type
migrateschema	Load complete	orasource	aurtarget	Full Load

You can monitor your task if you choose **Enable logging** when you set up your task. You can then view the CloudWatch metrics by doing the following:

1. On the navigation pane, choose **Tasks**.
2. Choose your migration task (`migratehrschem`).
3. Choose the **Task monitoring** tab, and monitor the task in progress on that tab.

## Step 10: Verify That Your Data Migration Completed Successfully

When the migration task completes, you can compare your task results with the expected results.

1. On the navigation pane, choose **Tasks**.
2. Choose your migration task (`migratehrschem`).
3. Choose the **Table statistics** tab, shown following.

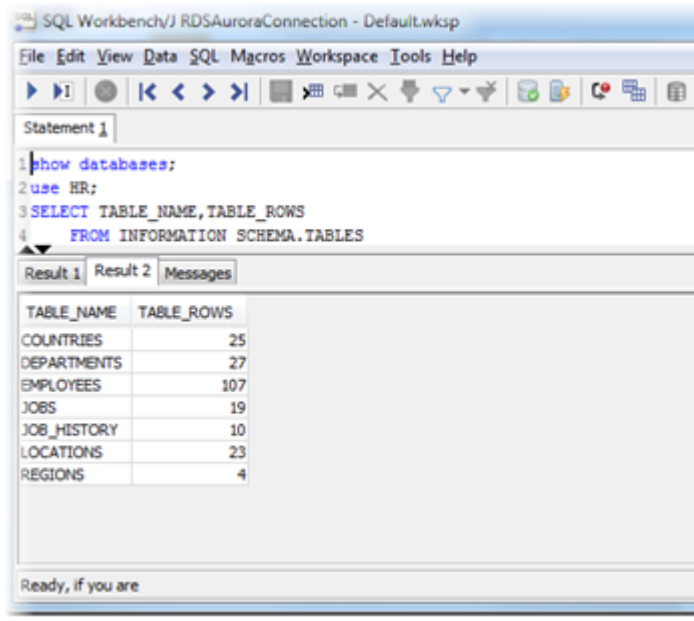
AWS Database Migration Service  
Step-by-Step Migration Guide  
Step 10: Verify That Your Data  
Migration Completed Successfully

The screenshot displays the AWS Database Migration Service (DMS) console. At the top, there are buttons for 'Create task', 'Start/Resume', 'Stop', and 'Delete'. Below these is a search filter. The main table shows a single task named 'migrateschema' with a status of 'Load complete', source 'orasource', target 'aurtarget', and type 'Full Load'. Below this, the 'migrateschema' task details are shown, with tabs for 'Overview', 'Task monitoring', 'Table statistics', and 'Logs'. The 'Table statistics' tab is active, showing a table with columns: 'Table', 'State', 'Inserts', 'Deletes', 'Updates', 'DDLs', and 'Full Load Rows'. The data rows are as follows:

Table	State	Inserts	Deletes	Updates	DDLs	Full Load Rows
COUNTRIES	Table completed	0	0	0	0	25
DEPARTMENTS	Table completed	0	0	0	0	27
EMPLOYEES	Table completed	0	0	0	0	107
JOBS	Table completed	0	0	0	0	19
JOB_HISTORY	Table completed	0	0	0	0	10
LOCATIONS	Table completed	0	0	0	0	23
REGIONS	Table completed	0	0	0	0	4

4. Connect to the Amazon Aurora MySQL instance by using SQL Workbench/J, and then check if the database tables were successfully migrated from Oracle to Aurora MySQL by running the SQL script shown following.

```
SELECT TABLE_NAME, TABLE_ROWS
FROM INFORMATION_SCHEMA.TABLES
WHERE TABLE_SCHEMA = 'HR' and TABLE_TYPE='BASE TABLE' order by 1;
```



5. Run the following query to check the relationship in tables; this query checks the departments with employees greater than 10.

```
SELECT B.DEPARTMENT_NAME, COUNT(*)
FROM HR.EMPLOYEES A, HR.DEPARTMENTS B
WHERE A.DEPARTMENT_ID=B.DEPARTMENT_ID
GROUP BY B.DEPARTMENT_NAME HAVING COUNT(*) > 10
ORDER BY 1;
```

The output from this query should be similar to the following.

```
department_name count(*)
Sales            34
Shipping        45
```

Now you have successfully completed a database migration from an Amazon RDS Oracle DB instance to Amazon Aurora MySQL.

## Step 11: Delete Walkthrough Resources

After you have completed this walkthrough, perform the following steps to avoid being charged further for AWS resources used in the walkthrough. It's necessary that you do the steps in order, because some resources cannot be deleted if they have a dependency upon another resource.

1. On the navigation pane, choose **Tasks**, choose your migration task (`migratehrschema`), and then choose **Delete**.
2. On the navigation pane, choose **Endpoints**, choose the Oracle source endpoint (`orasource`), and then choose **Delete**.
3. Choose the Amazon Aurora MySQL target endpoint (`aurtarget`), and then choose **Delete**.
4. On the navigation pane, choose **Replication instances**, choose the replication instance (`DMSdemo-repserver`), and then choose **Delete**.



Next, you must delete your AWS CloudFormation stack, `DMSdemo`.

1. Sign in to the AWS Management Console and open the AWS CloudFormation console at <https://console.aws.amazon.com/cloudformation>.

Note that if you are signed in as an AWS Identity and Access Management (IAM) user, you must have the appropriate permissions to access AWS CloudFormation.

2. Choose your CloudFormation stack, `DMSdemo`.
3. For **Actions**, choose **Delete stack**.

The status of the stack changes to `DELETE_IN_PROGRESS` while AWS CloudFormation cleans up the resources associated with the `DMSdemo` stack. When AWS CloudFormation is finished cleaning up resources, it removes the stack from the list.

## Next Steps

You can explore several other features of AWS DMS that were not included in this walkthrough, including the following:

- The AWS DMS change data capture (CDC) feature, for ongoing replication of data.
- Transformation actions that let you specify and apply transformations to the selected schema or table as part of the migration process.

For more information, see [the AWS DMS documentation](#).

# Migrating a SQL Server Database to Amazon Aurora MySQL

Using this walkthrough, you can learn how to migrate a Microsoft SQL Server database to an Amazon Aurora MySQL-Compatible Edition database using the AWS Schema Conversion Tool (AWS SCT) and AWS Database Migration Service (AWS DMS). AWS DMS migrates your data from your SQL Server source into your Aurora MySQL target.

AWS DMS doesn't migrate your secondary indexes, sequences, default values, stored procedures, triggers, synonyms, views, and other schema objects that aren't specifically related to data migration. To migrate these objects to your Aurora MySQL target, use AWS SCT.

## Topics

- [Prerequisites \(p. 62\)](#)
- [Step-by-Step Migration \(p. 63\)](#)
- [Troubleshooting \(p. 83\)](#)

## Prerequisites

The following prerequisites are required to complete this walkthrough:

- Understand Amazon Relational Database Service (Amazon RDS), the applicable database technologies, and SQL.
- Create an AWS account with AWS Identity and Access Management (IAM) credentials that allows you to launch Amazon RDS and AWS Database Migration Service (AWS DMS) instances in your AWS Region. For information about IAM credentials, see [Create an IAM User](#).
- Understand the Amazon Virtual Private Cloud (Amazon VPC) service and security groups. For information about using Amazon VPC with Amazon RDS, see [Amazon Virtual Private Cloud \(VPCs\) and Amazon RDS](#). For information about Amazon RDS security groups, see [Amazon RDS Security Groups](#).
- Understand the supported features and limitations of AWS DMS. For information about AWS DMS, see [What Is AWS Database Migration Service?](#)
- Understand how to work with Microsoft SQL Server as a source and Amazon Aurora MySQL as a target. For information about working with SQL Server as a source, see [Using a SQL Server Database as a Source for AWS Database Migration Service](#). Aurora MySQL is a MySQL-compatible database. For information about working with Aurora MySQL as a target, see [Using a MySQL-Compatible Database as a Target for AWS Database Migration Service](#).
- Understand the supported data type conversion options for SQL Server and Aurora MySQL. For information about data types for SQL Server as a source, see [Source Data Types for Microsoft SQL Server](#). For information about data types for Aurora MySQL; as a target, see [Target Data Types for MySQL](#).
- Size your target Aurora MySQL database host. DBAs should be aware of the load profile of the current source SQL Server database host. Consider CPU, memory, and IOPS. With Amazon RDS, you can size up the target database host, or reduce it, after the migration. If this is the first time that you're migrating to Aurora MySQL, we recommended that you have extra capacity to account for performance issues and tuning opportunities.

- Audit your source SQL Server database. For each schema and all the objects under each schema, determine whether any of the objects are no longer being used. Deprecate these objects on the source SQL Server database, because there's no need to migrate them if they aren't being used.
- Decide between these migration options: migrate existing data only or migrate existing data and replicate ongoing changes.
  - If you migrate existing data only, the migration is a one-time data transfer from a SQL Server source database to the Aurora MySQL target database. If the source database remains open to changes during the migration, these changes must be applied to the target database after the migration is complete.

**Note**

If the SQL Server database is an Amazon RDS database, replication is not supported, and you must use the option to migrate existing data only.

- If you migrate existing data and replicate ongoing changes, one option is to replicate the source database changes. Replication keeps the source and target databases in sync with each other during the migration process and can reduce database downtime. With this option, you complete an initial sync operation and then configure MS-REPLICATION. This option requires the Standard, Enterprise, or Developer SQL Server edition. You enable MS-REPLICATION for each SQL Server instance that you want to use as a database source.
- If you want to migrate existing data and replicate ongoing changes, another option is change data capture (CDC) instead of replication. This option allows AWS DMS to perform ongoing migration of data. In the case of CDC, AWS DMS uses the CDC tables to enable ongoing database migration. This option requires the Enterprise or Developer edition of SQL Server.

For more information about AWS DMS, see [the AWS DMS User Guide](#).

## Step-by-Step Migration

The following steps provide instructions for migrating a Microsoft SQL Server database to an Amazon Aurora MySQL database. These steps assume that you have already prepared your source database as described in [Prerequisites](#) (p. 62).

**Topics**

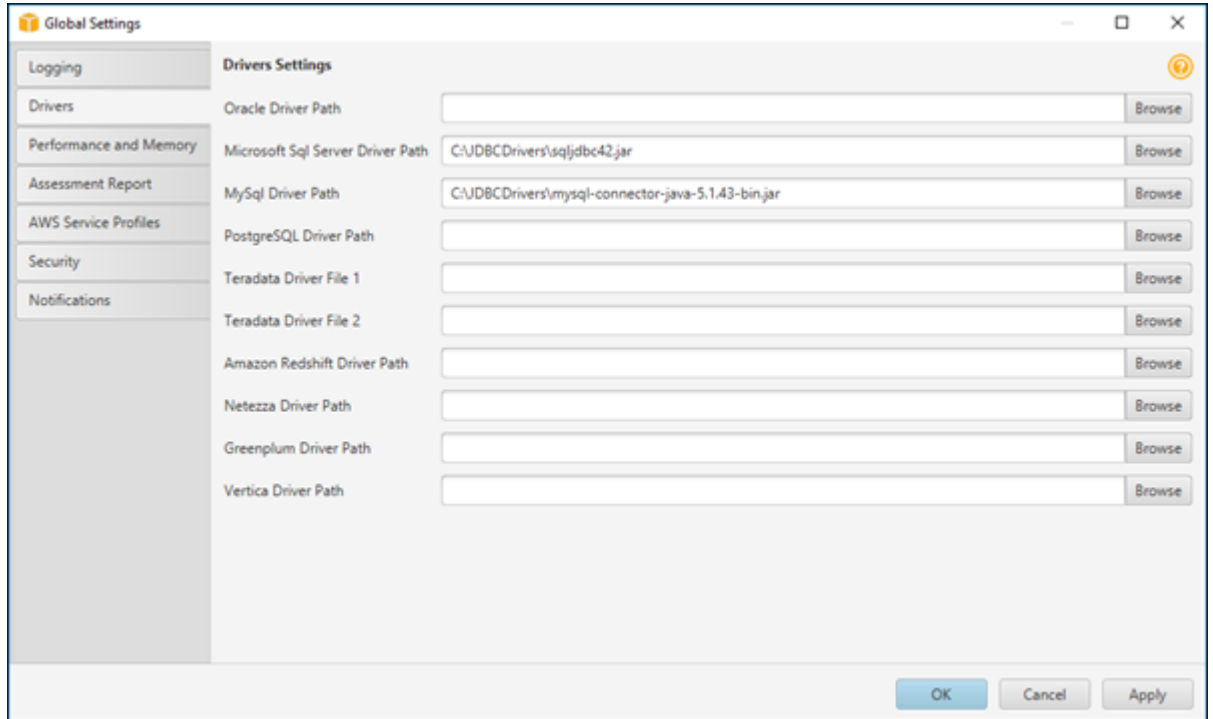
- [Step 1: Install the SQL Drivers and AWS Schema Conversion Tool on Your Local Computer](#) (p. 63)
- [Step 2: Configure Your Microsoft SQL Server Source Database](#) (p. 64)
- [Step 3: Configure Your Aurora MySQL Target Database](#) (p. 66)
- [Step 4: Use AWS SCT to Convert the SQL Server Schema to Aurora MySQL](#) (p. 66)
- [Step 5: Create an AWS DMS Replication Instance](#) (p. 74)
- [Step 6: Create AWS DMS Source and Target Endpoints](#) (p. 75)
- [Step 7: Create and Run Your AWS DMS Migration Task](#) (p. 79)
- [Step 8: Cut Over to Aurora MySQL](#) (p. 82)

## Step 1: Install the SQL Drivers and AWS Schema Conversion Tool on Your Local Computer

First, install the SQL drivers and the AWS Schema Conversion Tool (AWS SCT) on your local computer. Do the following:

1. Download the [JDBC driver for Microsoft SQL Server](#).

2. Download the [JDBC driver for Aurora MySQL](#). Amazon Aurora MySQL uses the MySQL driver.
3. Install AWS SCT and the required JDBC drivers.
  - a. See [Installing and Updating the AWS Schema Conversion Tool](#) in the *AWS Schema Conversion Tool User Guide*, and choose the appropriate link to download the AWS SCT.
  - b. Start AWS SCT, and choose **Settings, Global Settings**.
  - c. In **Global Settings**, choose **Drivers**, and then choose **Browse for Microsoft Sql Server Driver Path**. Locate the JDBC driver for SQL Server, and choose **OK**.
  - d. Choose **Browse for MySql Driver Path**. Locate the JDBC driver you downloaded for Aurora MySQL, and choose **OK**.



- e. Choose **OK** to close the **Global Settings** dialog box.

## Step 2: Configure Your Microsoft SQL Server Source Database

After installing the SQL drivers and AWS Schema Conversion Tool, you can configure your Microsoft SQL Server source database using one of several options, depending on how you plan to migrate your data.

When configuring your source database, you can choose to migrate existing data only, migrate existing data and replicate ongoing changes, or migrate existing data and use change data capture (CDC) to replicate ongoing changes. For more information about these options, see [Prerequisites \(p. 62\)](#).

### Migrating existing data only

No configuration steps are necessary for the SQL Server database. You can move on to [Step 3: Configure Your Aurora MySQL Target Database \(p. 66\)](#).

#### Note

If the SQL Server database is an Amazon RDS database, replication is not supported, and you must use the option for migrating existing data only.

### Migrating existing data and replicating ongoing changes

To configure MS-REPLICATION, complete the following steps:

1. In Microsoft SQL Server Management Studio, open the context (right-click) menu for the **Replication** folder, and then choose **Configure Distribution**.
2. In the **Distributor** step, choose **db\_name will act as its own distributor**. SQL Server creates a distribution database and log.

For more information, see the [Microsoft documentation](#).

When the configuration is complete, your server is enabled for replication. Either a distribution database is in place, or you have configured your server to use a remote distribution database.

#### Note

Replication requires a primary key for all tables that are being replicated. If your tables don't have primary keys defined, consider using CDC instead.

### Migrating existing data and using change data capture (CDC) to replicate ongoing changes

To configure MS-CDC, complete the following steps:

1. Connect to SQL Server with a login that has SYSADMIN role membership.
2. For each database containing data that is being migrated, run the following command within the database context:

```
use [DBname]
EXEC sys.sp_cdc_enable_db
```

3. For each table that you want to configure for ongoing migration, run the following command:

```
EXEC sys.sp_cdc_enable_table @source_schema = N'schema_name', @source_name =
N'table_name', @role_name = NULL;
```

For more information, see the [Microsoft documentation](#).

#### Note

- If you are migrating databases that participate in an AlwaysOn Availability Group, it is best practice to use replication for migration. To use this option, publishing must be enabled, and a distribution database must be configured for each node of the AlwaysOn Availability Group. Additionally, ensure you are using the name of the availability group listener for the database rather than the name of the server currently hosting the availability group database for the target server name. These requirements apply to each instance of SQL Server in the cluster and must not be configured using the availability group listener.
- If your database isn't supported for MS-REPLICATION or MS-CDC (for example, if you are running the Workgroup Edition of SQL Server), some changes can still be captured, such as INSERT and DELETE statements, but other DML statements such as UPDATE and TRUNCATE TABLE will not be captured. Therefore, a migration with continuing data replication is not recommended in this configuration, and a static one time migration (or repeated one time full migrations) should be considered instead.

For more information about using MS-REPLICATION and MS-CDC, see [Configuring a Microsoft SQL Server Database as a Replication Source for AWS Database Migration Service](#).

## Step 3: Configure Your Aurora MySQL Target Database

AWS DMS migrates the data from the SQL Server source into an Amazon Aurora MySQL target. In this step, you configure the Aurora MySQL target database.

1. Create the AWS DMS user to connect to your target database, and grant Superuser or the necessary individual privileges (or for Amazon RDS, use the master username).

Alternatively, you can grant the privileges to an existing user.

```
CREATE USER 'aurora_dms_user' IDENTIFIED BY 'password';  
  
GRANT ALTER, CREATE, DROP, INDEX, INSERT, UPDATE, DELETE,  
SELECT ON target_database.* TO 'aurora_dms_user';
```

2. AWS DMS uses control tables on the target in the database `awsdms_control`. Use the following command to ensure that the user has the necessary access to the `awsdms_control` database:

```
GRANT ALL PRIVILEGES ON awsdms_control.* TO 'aurora_dms_user';  
FLUSH PRIVILEGES;
```

## Step 4: Use AWS SCT to Convert the SQL Server Schema to Aurora MySQL

Before you migrate data to Amazon Aurora MySQL, convert the Microsoft SQL Server schema to an Aurora MySQL schema using the AWS Schema Conversion Tool (AWS SCT).

To convert a SQL Server schema to an Aurora MySQL schema, do the following:

1. In AWS SCT, choose **File, New Project**. Create a new project named `AWS Schema Conversion Tool SQL Server to Aurora MySQL`.
2. In the **New Project** dialog box, enter the following information, and then choose **OK**.

Parameter	Description
<b>Project Name</b>	Type <code>AWS Schema Conversion Tool SQL Server to Aurora MySQL</code> .
<b>Location</b>	Use the default <b>Projects</b> folder and the default <b>Transactional Database (OLTP)</b> option.
<b>Source Database Engine</b>	Choose <b>Microsoft SQL Server</b> .
<b>Target Database Engine</b>	Choose <b>Amazon Aurora (MySQL compatible)</b> .

AWS Database Migration Service  
Step-by-Step Migration Guide  
Step 4: Use AWS SCT to Convert the  
SQL Server Schema to Aurora MySQL

**New Project**

Enter the name, location and type of the new migration project.

Project name: AWS Schema Conversion Tool SQL Server to Aurora

Location: [Empty text box] Browse

Transactional Database (OLTP)  
 Data Warehouse (OLAP)

Source Database Engine: Microsoft SQL Server

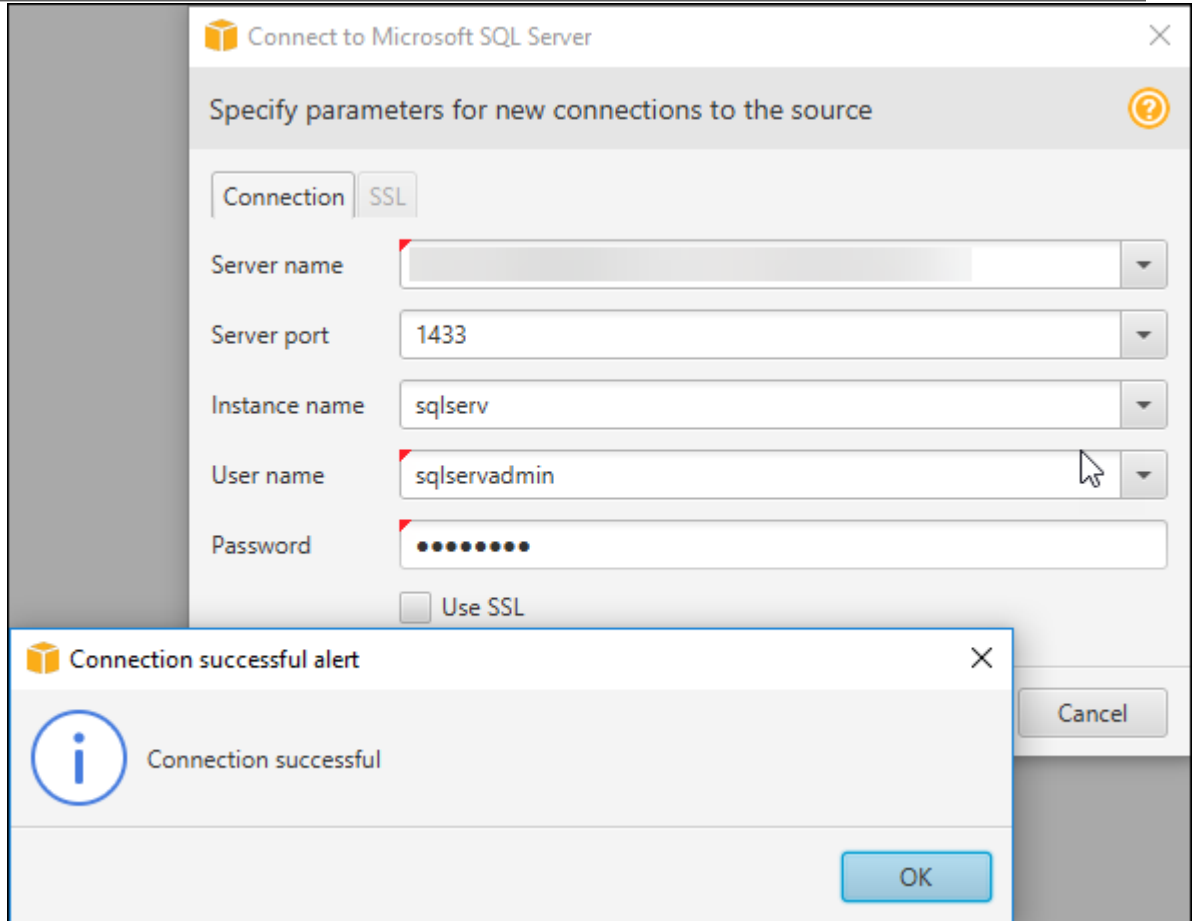
Target Database Engine: Amazon Aurora (MySQL compatible)

OK Cancel

3. Choose **Connect to Microsoft SQL Server**. In the **Connect to Microsoft SQL Server** dialog box, enter the following information, and then choose **Test Connection**.

Parameter	Description
<b>Server name</b>	Type the server name.
<b>Server port</b>	Type the SQL Server port number. The default is 1433.
<b>Instance name</b>	Type the SQL Server database instance name.
<b>User name</b>	Type the SQL Server admin user name.
<b>Password</b>	Provide the password for the admin user.

AWS Database Migration Service  
Step-by-Step Migration Guide  
Step 4: Use AWS SCT to Convert the  
SQL Server Schema to Aurora MySQL

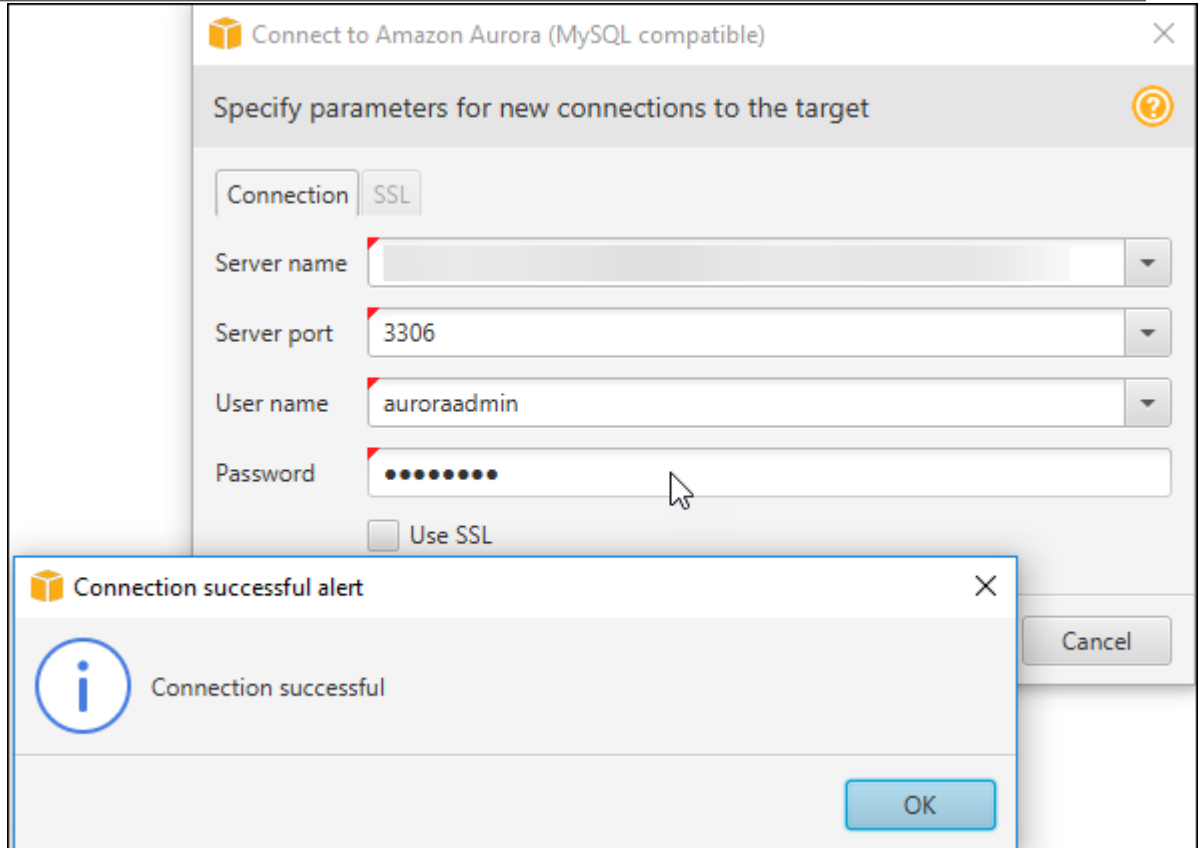


4. Choose **OK** to close the alert box. Then choose **OK** to close the dialog box and start the connection to the SQL Server DB instance. The database structure on the SQL Server DB instance is shown.
5. Choose **Connect to Amazon Aurora (MySQL compatible)**. In the **Connect to Amazon Aurora (MySQL compatible)** dialog box, enter the following information, and then choose **Test Connection**.

Parameter	Description
<b>Server name</b>	Type the server name.
<b>Server port</b>	Type the SQL Server port number. The default is 3306.
<b>User name</b>	Type the Aurora MySQL admin user name.
<b>Password</b>	Provide the password for the admin user.

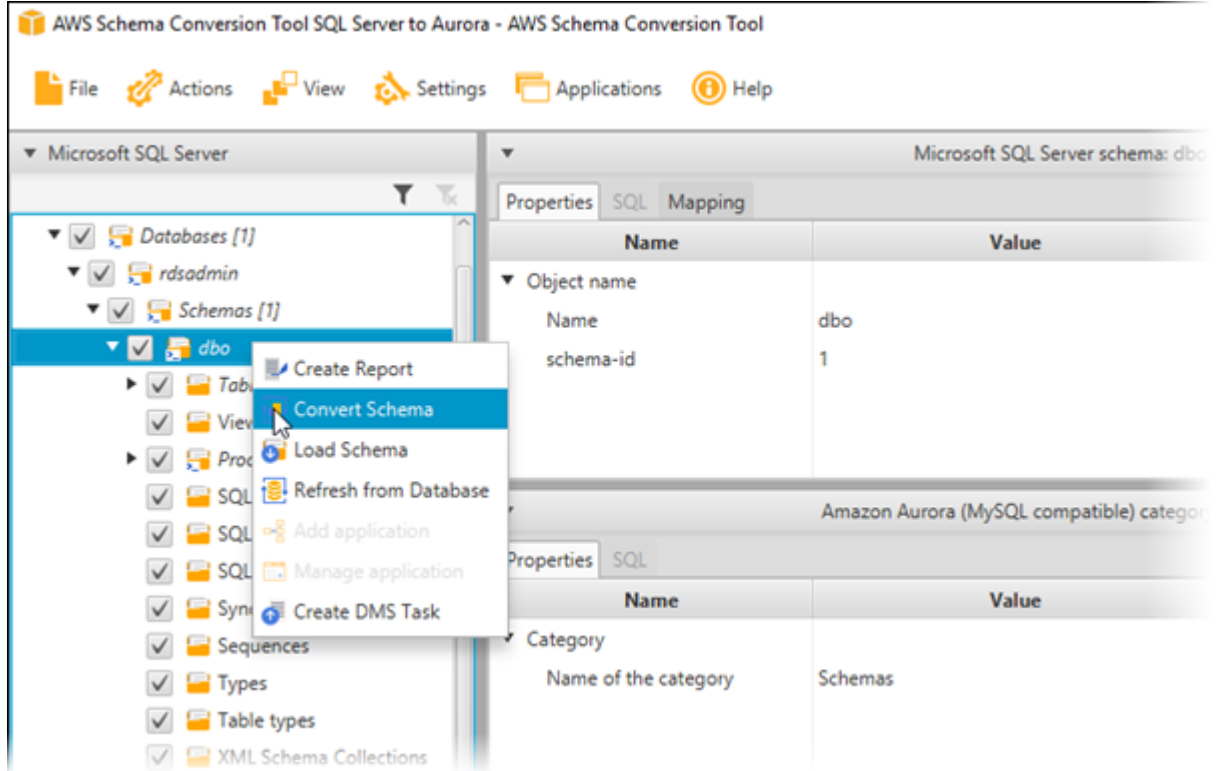


AWS Database Migration Service  
Step-by-Step Migration Guide  
Step 4: Use AWS SCT to Convert the  
SQL Server Schema to Aurora MySQL



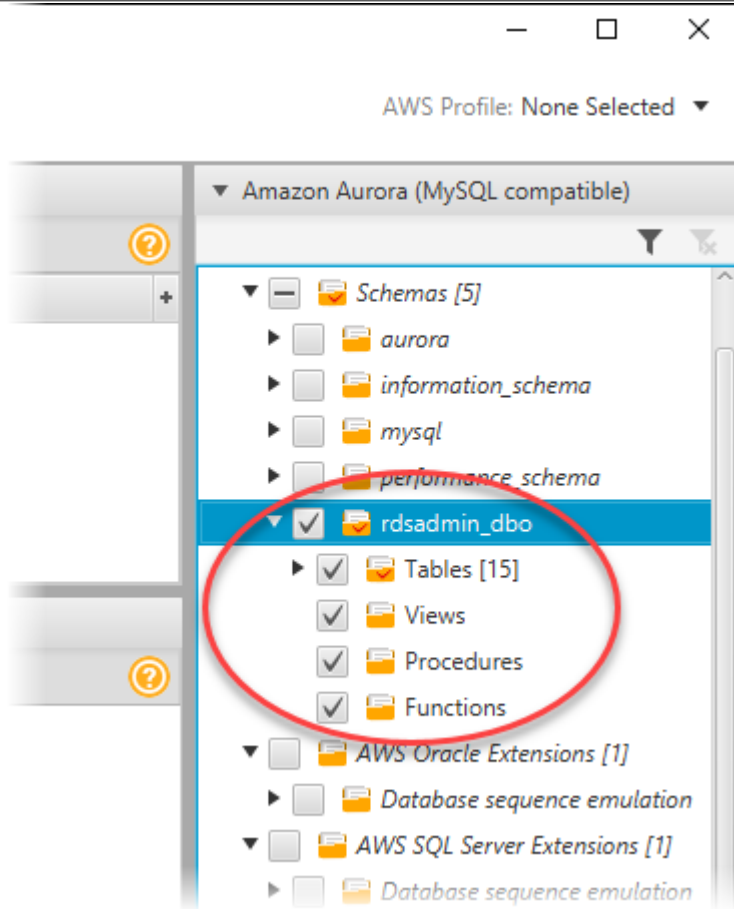
6. Choose **OK** to close the alert box. Then choose **OK** to close the dialog box and start the connection to the Aurora MySQL DB instance.
7. Open the context (right-click) menu for the schema to migrate, and then choose **Convert schema**.

AWS Database Migration Service  
Step-by-Step Migration Guide  
Step 4: Use AWS SCT to Convert the  
SQL Server Schema to Aurora MySQL



8. Choose **Yes** for the confirmation message. AWS SCT then converts your schemas to the target database format.

AWS Database Migration Service  
Step-by-Step Migration Guide  
Step 4: Use AWS SCT to Convert the  
SQL Server Schema to Aurora MySQL

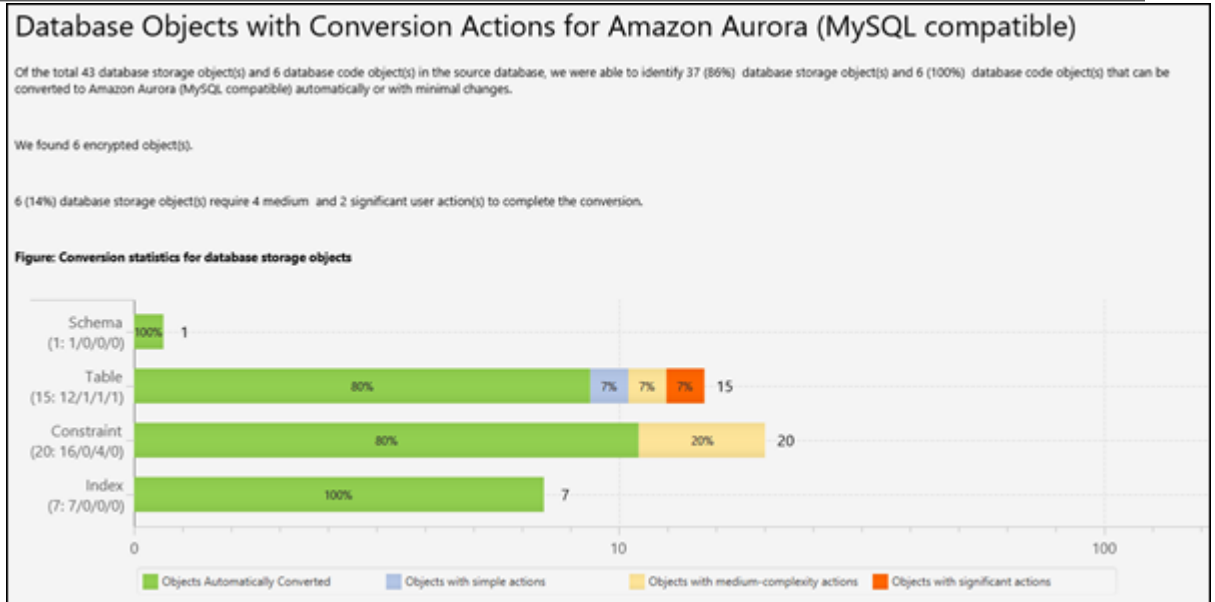


AWS SCT analyzes the schema and creates a database migration assessment report for the conversion to Aurora MySQL.

9. Choose **Assessment Report View** from **View** to check the report.

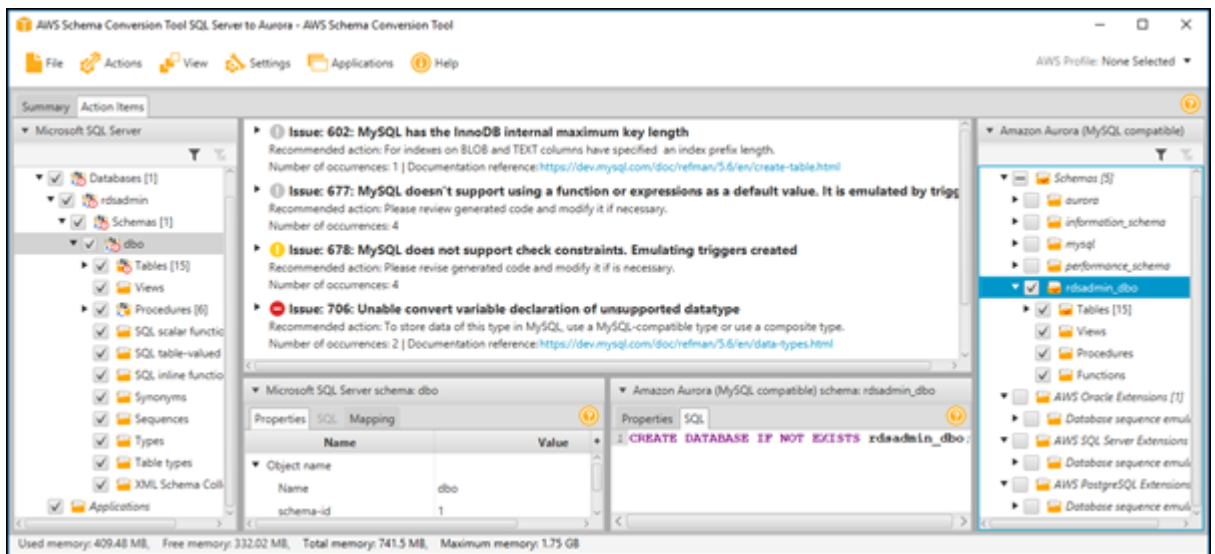
The report breaks down by each object type and by how much manual change is needed to convert it successfully.

## AWS Database Migration Service Step-by-Step Migration Guide Step 4: Use AWS SCT to Convert the SQL Server Schema to Aurora MySQL



Generally, packages, procedures, and functions are more likely to have some issues to resolve because they contain the most custom PL/SQL code. AWS SCT also provides hints about how to fix these objects.

### 10 Choose the **Action Items** tab.



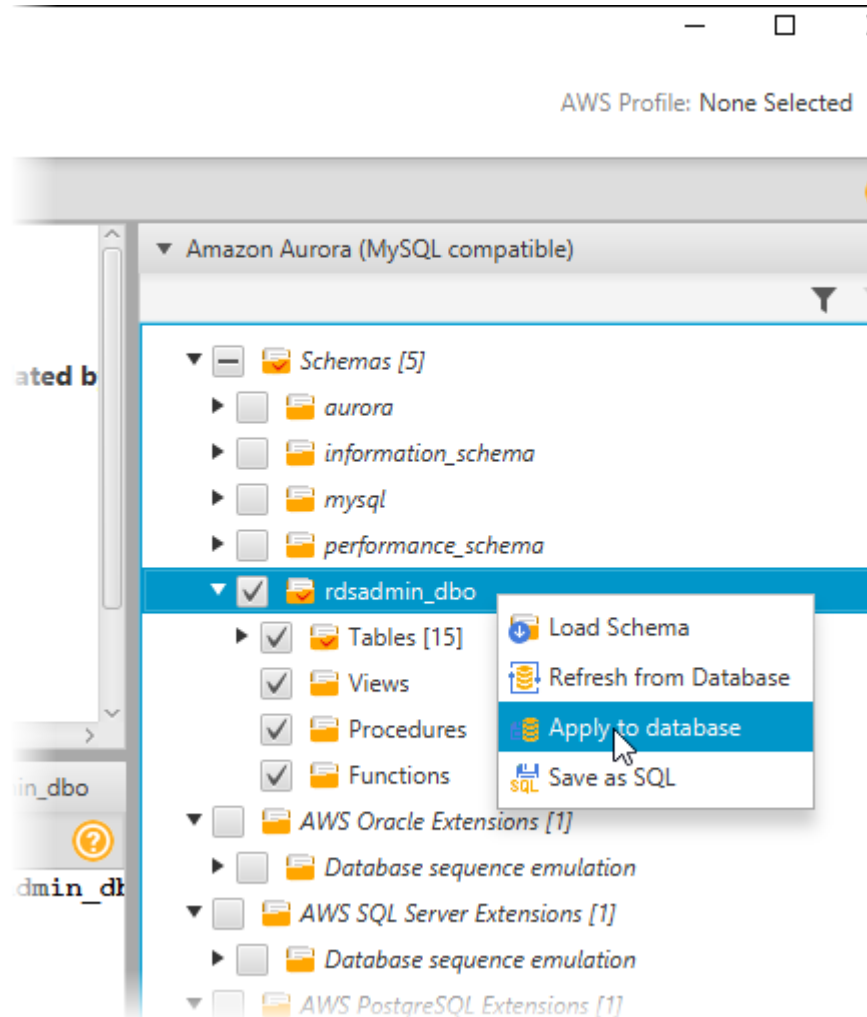
The **Action Items** tab shows each issue for each object that requires attention.

For each conversion issue, you can complete one of the following actions:

- a. Modify the objects on the source SQL Server database so that AWS SCT can convert the objects to the target Aurora MySQL database.
  - i. Modify the objects on the source SQL Server database.
  - ii. Repeat the previous steps to convert the schema and check the assessment report.
  - iii. If necessary, repeat this process until there are no conversion issues.

AWS Database Migration Service  
Step-by-Step Migration Guide  
Step 4: Use AWS SCT to Convert the  
SQL Server Schema to Aurora MySQL

- iv. Choose **Main View** from **View**. Open the context (right-click) menu for the target Aurora MySQL schema, and choose **Apply to database** to apply the schema changes to the Aurora MySQL database, and confirm that you want to apply the schema changes.



- b. Instead of modifying the source schema, modify scripts that AWS SCT generates before applying the scripts on the target Aurora MySQL database.
- i. Choose **Main View** from **View**. Open the context (right-click) menu for the target Aurora MySQL schema name, and choose **Save as SQL**. Next, choose a name and destination for the script.
- ii. In the script, modify the objects to correct conversion issues.

You can also exclude foreign key constraints, triggers, and secondary indexes from the script because they can cause problems during the migration. After the migration is complete, you can create these objects on the Aurora MySQL database.

- iii. Run the script on the target Aurora MySQL database.

For more information, see [Converting Database Schema to Amazon RDS by Using the AWS Schema Conversion Tool](#) in the *AWS Schema Conversion Tool User Guide*.

11(Optional) Use AWS SCT to create mapping rules.

- a. Under **Settings**, select **Mapping Rules**.
- b. Create additional mapping rules that are required based on the action items.
- c. Save the mapping rules.

- d. Choose **Export script for DMS** to export a JSON format of all the transformations that the AWS DMS task will use. Choose **Save**.

## Step 5: Create an AWS DMS Replication Instance

After validating the schema structure between source and target databases, continue with the core part of this walkthrough, which is the data migration. The following illustration shows a high-level view of the migration process.



An AWS DMS replication instance performs the actual data migration between source and target. The replication instance also caches the transaction logs during the migration. The amount of CPU and memory capacity a replication instance has influences the overall time that is required for the migration.

For information about best practices for using AWS DMS, see [AWS Database Migration Service Best Practices](#).

To create an AWS DMS replication instance, do the following:

1. Sign in to the AWS Management Console, and open the AWS DMS console at <https://console.aws.amazon.com/dms/>.
2. In the console, choose **Create migration**. If you are signed in as an AWS Identity and Access Management (IAM) user, you must have the appropriate permissions to access AWS DMS. For more information about the permissions required, see [IAM Permissions Needed to Use AWS DMS](#).
3. On the Welcome page, choose **Next** to start a database migration.
4. On the **Create replication instance** page, specify your replication instance information.

Parameter	Description
<b>Name</b>	Select a name for your replication instance. If you are using multiple replication servers or sharing an account, choose a name that helps you quickly differentiate between the different servers.
<b>Description</b>	Type a brief description.
<b>Instance class</b>	Select the type of replication server to create. Each size and type of instance class has increasing CPU, memory, and I/O capacity. Generally, <code>t2</code> instances are for lower load tasks, and the <code>c4</code> instances are for higher load and more tasks.
<b>VPC</b>	Choose the virtual private cloud (VPC) in which your replication instance will launch. If possible,

Parameter	Description
	select the same VPC in which either your source or target database resides (or both).
<b>Multi-AZ</b>	If you choose <b>Yes</b> , AWS DMS creates a second replication server in a different Availability Zone for failover if there is a problem with the primary replication server.
<b>Publicly accessible</b>	If either your source or target database resides outside of the VPC in which your replication server resides, you must make your replication server policy publicly accessible.

5. For the **Advanced** section, specify the following information.

Parameter	Description
<b>Allocated storage (GB)</b>	Amount of storage on the replication server for the AWS DMS task logs, including historical tasks logs. AWS DMS also uses disk storage to cache certain data while it replicates it from the source database to the target. Additionally, more storage generally enables better IOPS on the server.
<b>Replication Subnet Group</b>	If you are running in a Multi-AZ configuration, you need at least two subnet groups.
<b>Availability zone</b>	Generally, performance is better if you locate your primary replication server in the same Availability Zone as your target database.
<b>VPC Security Group(s)</b>	Security groups enable you to control ingress and egress to your VPC. AWS DMS lets you associate one or more security groups with the VPC in which your replication server is launched.
<b>KMS key</b>	With AWS DMS, all data is encrypted at rest using a KMS encryption key. By default, AWS DMS creates a new encryption key for your replication server. However, you might choose to use an existing key.

For information about the KMS key, see [Setting an Encryption Key and Specifying KMS Permissions](#).

6. Click **Next**.

## Step 6: Create AWS DMS Source and Target Endpoints

While your replication instance is being created, you can specify the source and target database endpoints using the AWS Management Console. However, you can test connectivity only after the replication instance has been created, because the replication instance is used in the connection.

1. In the AWS DMS console, specify your connection information for the source SQL Server database and the target Aurora MySQL database. The following table describes the source settings.

Parameter	Description
<b>Endpoint Identifier</b>	Type a name, such as <code>SQLServerSource</code> .
<b>Source Engine</b>	Choose <code>sqlserver</code> .
<b>Server name</b>	Provide the SQL Server DB instance server name.
<b>Port</b>	Type the port number of the database. The default for SQL Server is 1433.
<b>SSL mode</b>	Choose an SSL mode if you want to enable encryption for your connection's traffic.
<b>User name</b>	Type the name of the user you want to use to connect to the source database.
<b>Password</b>	Provide the password for the user.
<b>Database name</b>	Provide the SQL Server database name.

The following table describes the advanced source settings.

Parameter	Description
<b>Extra connection attributes</b>	<p>Extra parameters that you can set in an endpoint to add functionality or change the behavior of AWS DMS. A few of the most relevant attributes are listed here. Use a semicolon (;) to separate multiple entries.</p> <ul style="list-style-type: none"> <li>* <code>safeguardpolicy</code> - Changes the behavior of SQL Server by opening transactions to prevent the transaction log from being truncated while AWS DMS is reading the log. Valid values are <code>EXCLUSIVE_AUTOMATIC_TRUNCATION</code> or <code>RELY_ON_SQL_SERVER_REPLICATION_AGENT</code> (default).</li> <li>* <code>useBCPFullLoad</code> - Directs AWS DMS to use BCP (bulk copy) for data loading. Valid values are <code>Y</code> or <code>N</code>. When the target table contains an identity column that does not exist in the source table, you must disable the use of BCP for loading the table by setting the parameter to <code>N</code>.</li> <li>* <code>BCPPacketSize</code> - If BCP is enabled for data loads, then enter the maximum packet size used by BCP. Valid values are 1 – 100000 (default 16384).</li> <li>* <code>controlTablesFileGroup</code> - Specifies the file group to use for the control tables that the AWS DMS process creates in the database.</li> </ul>



Parameter	Description
KMS key	Enter the KMS key if you choose to encrypt your replication instance's storage.

The following table describes the target settings.

Parameter	Description
Endpoint Identifier	Type a name, such as <code>AuroraTarget</code> .
Target Engine	Choose <b>aurora</b> .
Server name	Provide the Aurora MySQL DB server name for the primary instance.
Port	Type the port number of the database. The default for Aurora MySQL is <code>3306</code> .
SSL mode	Choose <b>None</b> .
User name	Type the name of the user that you want to use to connect to the target database.
Password	Provide the password for the user.

The following table describes the advanced target settings.

Parameter	Description
Extra connection attributes	<p>Extra parameters that you can set in an endpoint to add functionality or change the behavior of AWS DMS. A few of the most relevant attributes are listed here. Use a semicolon to separate multiple entries.</p> <p>* <code>targetDbType</code> - By default, AWS DMS creates a different database for each schema that is being migrated. If you want to combine several schemas into a single database, set this option to <code>targetDbType=SPECIFIC_DATABASE</code>.</p> <p>* <code>initstmt</code> - Use this option to invoke the MySQL <code>initstmt</code> connection parameter and accept anything MySQL <code>initstmt</code> accepts. For an Aurora MySQL target, it's often useful to disable foreign key checks by setting this option to <code>initstmt=SET FOREIGN_KEY_CHECKS=0</code>.</p>
KMS key	Enter the KMS key if you choose to encrypt your replication instance's storage.

The following is an example of the completed page.

AWS Database Migration Service  
Step-by-Step Migration Guide  
Step 6: Create AWS DMS Source and Target Endpoints

**Connect source and target database endpoints**

✓ Replication instance created successfully.

Your database endpoint can be on-premise, in EC2, RDS or in the cloud. Define the connection details below. It is recommended that you test your endpoint connections here to avoid errors later.

Source database connection details	Target database connection details
Endpoint identifier* SQLServSource ⓘ	Endpoint identifier* AuroraTarget ⓘ
Source engine* sqlserver ⓘ	Target engine* aurora ⓘ
Server name*	Server name*
Port* 1433 ⓘ	Port* 3306 ⓘ
SSL mode* none ⓘ	SSL mode* none ⓘ
User name* sqlservadmin ⓘ	User name* auroraadmin ⓘ
Password* ..... ⓘ	Password* ..... ⓘ
Database name* sqlserv	Advanced
Advanced	Run test
Run test	

For information about extra connection attributes, see [Using Extra Connection Attributes with AWS Database Migration Service](#).

2. After the endpoints and replication instance are created, test the endpoint connections by choosing **Run test** for the source and target endpoints.
3. Drop foreign key constraints and triggers on the target database.

During the full load process, AWS DMS does not load tables in any particular order, so it might load the child table data before parent table data. As a result, foreign key constraints might be violated if they are enabled. Also, if triggers are present on the target database, they might change data loaded by AWS DMS in unexpected ways.

```
ALTER TABLE 'table_name' DROP FOREIGN KEY 'fk_name';  
  
DROP TRIGGER 'trigger_name';
```

4. If you dropped foreign key constraints and triggers on the target database, generate a script that enables the foreign key constraints and triggers.

Later, when you want to add them to your migrated database, you can just run this script.

5. (Optional) Drop secondary indexes on the target database.

Secondary indexes (as with all indexes) can slow down the full load of data into tables because they must be maintained and updated during the loading process. Dropping them can improve the

performance of your full load process. If you drop the indexes, you must to add them back later, after the full load is complete.

```
ALTER TABLE 'table_name' DROP INDEX 'index_name';
```

6. Choose **Next**.

## Step 7: Create and Run Your AWS DMS Migration Task

Using an AWS DMS task, you can specify what schema to migrate and the type of migration. You can migrate existing data, migrate existing data and replicate ongoing changes, or replicate data changes only.

1. In the AWS DMS console, on the **Create task** page, specify the task options. The following table describes the settings.

Parameter	Description
<b>Task name</b>	Type a name for the migration task.
<b>Task description</b>	Type a description for the task.
<b>Source endpoint</b>	Shows the SQL Server source endpoint.  If you have more than one endpoint in the account, choose the correct endpoint from the list.
<b>Target endpoint</b>	Shows the Aurora MySQL target endpoint.
<b>Replication instance</b>	Shows the AWS DMS replication instance.
<b>Migration type</b>	Choose an option.  * <b>Migrate existing data</b> - AWS DMS migrates only your existing data. Changes to your source data aren't captured and applied to your target. If you can afford to take an outage for the duration of the full load, then this is the simplest option. You can also use this option to create test copies of your database. If the source SQL Server database is an Amazon RDS database, you must choose this option.  * <b>Migrate existing data and replicate ongoing changes</b> - AWS DMS captures changes while migrating your existing data. AWS DMS continues to capture and apply changes even after the bulk data has been loaded. Eventually the source and target databases are in sync, allowing for a minimal downtime.  * <b>Replicate data changes only</b> - Bulk load data using a different method. This approach

AWS Database Migration Service  
 Step-by-Step Migration Guide  
 Step 7: Create and Run Your AWS DMS Migration Task

Parameter	Description
	generally applies only to homogeneous migrations.
<b>Start task on create</b>	In most situations, you should choose this option. Sometimes, you might want to delay the start of a task, for example, if you want to change logging levels.

The page should look similar to the following:

### Create task

A task can contain one or more table mappings which define what data is moved from the source to the target. If a table does not exist on the target, it can be created automatically.

**Task name\***  ⓘ

**Task description\***  ⓘ

**Source endpoint** sqlservsource

**Target endpoint** auroratarget

**Replication instance** sqlserver2aurora

**Migration type\***  ⓘ

**Start task on create**

2. Under **Task settings**, specify the settings. The following table describes the settings.

Parameter	Description
<b>Target table preparation mode</b>	<p>Choose an option.</p> <ul style="list-style-type: none"> <li>* <b>Do nothing</b> - AWS DMS does nothing to prepare your tables. Your table structure remains the same, and any existing data remains in the table. You can use this method to consolidate data from multiple systems.</li> <li>* <b>Drop tables on target</b> - AWS DMS creates your target tables for you. AWS DMS drops and re-creates the tables to migrate before migration. AWS DMS creates the table and a primary key only for heterogeneous migrations.</li> <li>* <b>Truncate</b> - AWS DMS truncates a target table before loading it. If the target table doesn't exist, then AWS DMS creates it.</li> </ul>

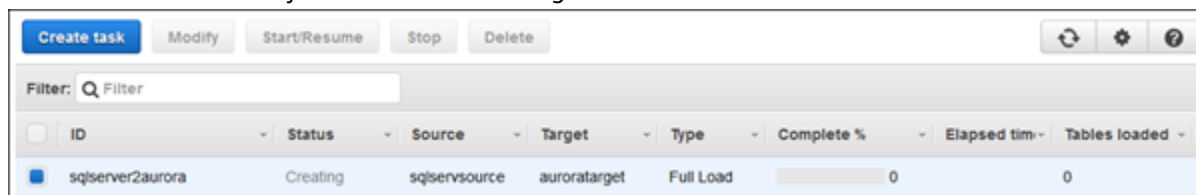
Parameter	Description
	IMPORTANT: If the AWS Schema Conversion Tool already created the tables on the target, choose <b>Do nothing</b> or <b>Truncate</b> .
<b>Include LOB columns in replication</b>	<p>Choose an option.</p> <p>* <b>Don't include LOB columns</b> - Do not migrate LOB data.</p> <p>* <b>Full LOB mode</b> - AWS DMS migrates all LOBs (large objects) from the source to the target regardless of size. In this configuration, AWS DMS has no information about the maximum size of LOBs to expect. Thus, LOBs are migrated one at a time, piece by piece. Full LOB mode can be relatively slow.</p> <p>* <b>Limited LOB mode</b> - You set a maximum size LOB that AWS DMS accepts. This option enables AWS DMS to pre-allocate memory and load the LOB data in bulk. LOBs that exceed the maximum LOB size are truncated, and a warning is issued to the log file. In limited LOB mode, you get significant performance gains over full LOB mode. We recommend that you use limited LOB mode whenever possible.</p>
<b>Max LOB size (kb)</b>	When <b>Limited LOB mode</b> is selected, this option determines the maximum LOB size that AWS DMS accepts. Any LOBs that are larger than this value are truncated to this value.
<b>Enable logging</b>	It's best to select <b>Enable logging</b> . If you enable logging, you can see any errors or warnings that the task encounters, and you can troubleshoot those issues.

3. Leave the Advanced settings at their default values.
4. If you created and exported mapping rules with AWS SCT in the last step in [Step 4: Use AWS SCT to Convert the SQL Server Schema to Aurora MySQL \(p. 66\)](#), choose **Table mappings**, and select the **JSON** tab. Then select **Enable JSON editing**, and enter the table mappings you saved.

If you did not create mapping rules, then proceed to the next step.

5. Choose **Create task**. The task starts immediately.

The **Tasks** section shows you the status of the migration task.



If you chose **Enable logging** during setup, you can monitor your task. You can then view the Amazon CloudWatch metrics.

1. On the navigation pane, choose **Tasks**.
2. Choose your migration task.
3. Choose the **Task monitoring** tab, and monitor the task in progress on that tab.

When the full load is complete and cached changes are applied, the task stops on its own.

4. On the target Aurora MySQL database, if you disabled foreign key constraints and triggers, enable them using the script that you saved previously.
5. On the target Aurora MySQL database, re-create the secondary indexes if you removed them previously.
6. If you chose to use AWS DMS to replicate changes, in the AWS DMS console, start the AWS DMS task by choosing **Start/Resume** for the task.

Important replication instance metrics to monitor include the following:

- CPU
- FreeableMemory
- DiskQueueDepth
- CDCLatencySource
- CDCLatencyTarget

The AWS DMS task keeps the target Aurora MySQL database up to date with source database changes. AWS DMS keeps all the tables in the task up to date until it's time to implement the application migration. The latency is zero, or close to zero, when the target has caught up to the source.

For more information, see [Monitoring AWS Database Migration Service Tasks](#).

## Step 8: Cut Over to Aurora MySQL

To move connections from your Microsoft SQL Server database to your Amazon Aurora MySQL database, do the following:

1. End all SQL Server database dependencies and activities, such as running scripts and client connections. Ensure that the SQL Server Agent service is stopped.

The following query should return no results other than your connection:

```
SELECT session_id, login_name from sys.dm_exec_sessions where session_id > 50;
```

2. Kill any remaining sessions (other than your own).

```
KILL session_id;
```

3. Shut down the SQL Server service.
4. Let the AWS DMS task apply the final changes from the SQL Server database on the Amazon Aurora MySQL database.
5. In the AWS DMS console, stop the AWS DMS task by choosing **Stop** for the task, and then confirming that you want to stop the task.

## Troubleshooting

When you work with Microsoft SQL Server as a source database and Amazon Aurora MySQL as a target database, the two most common problem areas are SQL Server change data capture (CDC) and foreign keys.

- **MS-CDC:** If you are using MS-CDC with SQL Server for the migration, errors that are related to permissions or errors during change data capture are common. These types of errors usually result when one of the prerequisites was not met. For example, the most common overlooked prerequisite is a full database backup.
- **Foreign keys:** During the full load process, AWS DMS does not load tables in any particular order, so it might load the child table data before parent table data. As a result, foreign key constraints might be violated if they are enabled. You should disable foreign keys on the Aurora MySQL target database. You can enable the foreign keys on the target after the migration is complete.

For more tips, see the AWS DMS troubleshooting section in the [AWS DMS User Guide](#).

To troubleshoot issues specific to SQL Server, see the SQL Server troubleshooting section:

- [Troubleshooting Microsoft SQL Server Specific Issues](#)

To troubleshoot Aurora MySQL issues, see the Aurora MySQL troubleshooting section and the MySQL troubleshooting section:

- [Troubleshooting Amazon Aurora MySQL Specific Issues](#)
- [Troubleshooting MySQL Specific Issues](#)

# Migrating an Oracle Database to PostgreSQL

Using this walkthrough, you can learn how to migrate an Oracle database to a PostgreSQL database using AWS Database Migration Service (AWS DMS) and the AWS Schema Conversion Tool (AWS SCT).

AWS DMS migrates your data from your Oracle source into your PostgreSQL target. AWS DMS also captures data manipulation language (DML) and [supported data definition language \(DDL\)](#) changes that happen on your source database and applies these changes to your target database. This way, AWS DMS keeps your source and target databases in sync with each other. To facilitate the data migration, AWS SCT creates the migrated schemas on the target database, including the tables and primary key indexes on the target if necessary.

AWS DMS doesn't migrate your secondary indexes, sequences, default values, stored procedures, triggers, synonyms, views, and other schema objects not specifically related to data migration. To migrate these objects to your PostgreSQL target, use AWS SCT.

## Topics

- [Prerequisites \(p. 84\)](#)
- [Step-by-Step Migration \(p. 85\)](#)
- [Rolling Back the Migration \(p. 105\)](#)
- [Troubleshooting \(p. 105\)](#)

## Prerequisites

The following prerequisites are required to complete this walkthrough:

- Understand Amazon Relational Database Service (Amazon RDS), the applicable database technologies, and SQL.
- Create an AWS account with AWS Identity and Access Management (IAM) credentials that allows you to launch Amazon RDS and AWS Database Migration Service (AWS DMS) instances in your AWS Region. For information about IAM credentials, see [Create an IAM User](#).
- Understand the Amazon Virtual Private Cloud (Amazon VPC) service and security groups. For information about using Amazon VPC with Amazon RDS, see [Amazon Virtual Private Cloud \(VPCs\) and Amazon RDS](#). For information about Amazon RDS security groups, see [Amazon RDS Security Groups](#).
- Understand the supported features and limitations of AWS DMS. For information about AWS DMS, see [What Is AWS Database Migration Service?](#)
- Understand the supported data type conversion options for Oracle and PostgreSQL. For information about data types for Oracle as a source, see [Using an Oracle Database as a Source for AWS Database Migration Service](#). For information about data types for PostgreSQL as a target, see [Using a PostgreSQL Database as a Target for AWS Database Migration Service](#).
- Size your target PostgreSQL database host. DBAs should be aware of the load profile of the current source Oracle database host. Consider CPU, memory, and IOPS. With RDS, you can size up the target database host, or reduce it, after the migration. If this is the first time you are migrating to



PostgreSQL, then we recommend that you have extra capacity to account for performance issues and tuning opportunities.

- Audit your source Oracle database. For each schema and all the objects under each schema, determine if any of the objects are no longer being used. Deprecate these objects on the source Oracle database, because there's no need to migrate them if they are not being used.
- If load capacity permits, then get the max size (kb) for each LOB type on the source database, and keep this information for later.
- If possible, move columns with BLOB, CLOB, NCLOB, LONG, LONG RAW, and XMLTYPE to Amazon S3, Dynamo DB, or another data store. Doing so simplifies your source Oracle database for an easier migration. It will also lower the capacity requirements for the target PostgreSQL database.

For more information on AWS DMS, see [the AWS DMS documentation](#).

## Step-by-Step Migration

The following steps provide instructions for migrating an Oracle database to a PostgreSQL database. These steps assume that you have already prepared your source database as described in [Prerequisites \(p. 84\)](#).

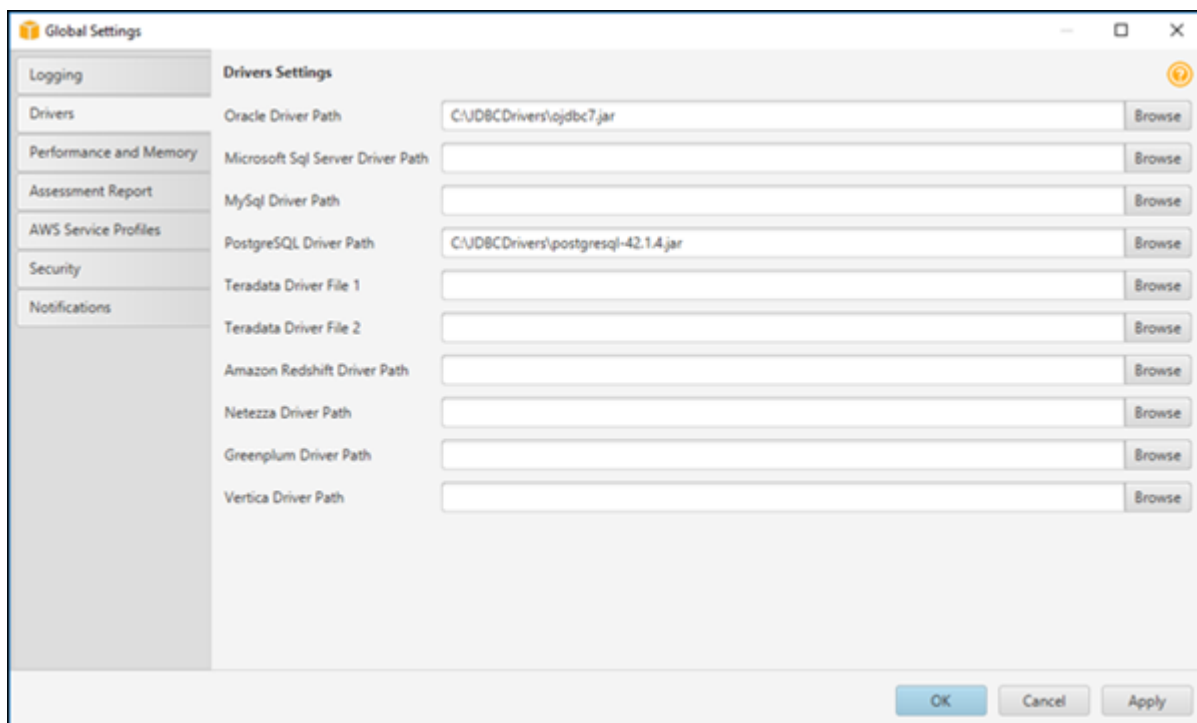
### Topics

- [Step 1: Install the SQL Drivers and AWS Schema Conversion Tool on Your Local Computer \(p. 85\)](#)
- [Step 2: Configure Your Oracle Source Database \(p. 86\)](#)
- [Step 3: Configure Your PostgreSQL Target Database \(p. 88\)](#)
- [Step 4: Use the AWS Schema Conversion Tool \(AWS SCT\) to Convert the Oracle Schema to PostgreSQL \(p. 89\)](#)
- [Step 5: Create an AWS DMS Replication Instance \(p. 96\)](#)
- [Step 6: Create AWS DMS Source and Target Endpoints \(p. 98\)](#)
- [Step 7: Create and Run Your AWS DMS Migration Task \(p. 101\)](#)
- [Step 8: Cut Over to PostgreSQL \(p. 104\)](#)

## Step 1: Install the SQL Drivers and AWS Schema Conversion Tool on Your Local Computer

To install the SQL drivers and the AWS Schema Conversion Tool (AWS SCT) on your local computer, do the following:

1. Download the JDBC driver for your Oracle database release. For more information, go to <https://www.oracle.com/jdbc>.
2. Download the PostgreSQL driver ([postgresql-42.1.4.jar](#)).
3. Install AWS SCT and the required JDBC drivers.
  - a. Download AWS SCT from [Installing and Updating the AWS Schema Conversion Tool](#) in the *AWS Schema Conversion Tool User Guide*.
  - b. Launch AWS SCT.
  - c. In AWS SCT, choose **Global Settings** from **Settings**.
  - d. In **Global Settings**, choose **Driver**, and then choose **Browse** for **Oracle Driver Path**. Locate the JDBC Oracle driver and choose **OK**.
  - e. Choose **Browse** for **PostgreSQL Driver Path**. Locate the JDBC PostgreSQL driver and choose **OK**.



f. Choose **OK** to close the dialog box.

## Step 2: Configure Your Oracle Source Database

To use Oracle as a source for AWS Database Migration Service (AWS DMS), you must first ensure that ARCHIVELOG MODE is on to provide information to LogMiner. AWS DMS uses LogMiner to read information from the archive logs so that AWS DMS can capture changes.

For AWS DMS to read this information, make sure the archive logs are retained on the database server as long as AWS DMS requires them. If you configure your task to begin capturing changes immediately, then you should only need to retain archive logs for a little longer than the duration of the longest running transaction. Retaining archive logs for 24 hours is usually sufficient. If you configure your task to begin from a point in time in the past, then archive logs must be available from that time forward. For more specific instructions about enabling ARCHIVELOG MODE and ensuring log retention for your Oracle database, see the [Oracle documentation](#).

To capture change data, AWS DMS requires supplemental logging to be enabled on your source database. Minimal supplemental logging must be enabled at the database level. AWS DMS also requires that identification key logging be enabled. This option causes the database to place all columns of a row's primary key in the redo log file whenever a row containing a primary key is updated. This result occurs even if no value in the primary key has changed. You can set this option at the database or table level.

1. Create or configure a database account to be used by AWS DMS. We recommend that you use an account with the minimal privileges required by AWS DMS for your AWS DMS connection. AWS DMS requires the following privileges.

```
CREATE SESSION
SELECT ANY TRANSACTION
SELECT on V_$ARCHIVED_LOG
SELECT on V_$LOG
SELECT on V_$LOGFILE
```

AWS Database Migration Service  
Step-by-Step Migration Guide  
Step 2: Configure Your Oracle Source Database

---

```
SELECT on V_$DATABASE
SELECT on V_$THREAD
SELECT on V_$PARAMETER
SELECT on V_$NLS_PARAMETERS
SELECT on V_$TIMEZONE_NAMES
SELECT on V_$TRANSACTION
SELECT on ALL_INDEXES
SELECT on ALL_OBJECTS
SELECT on ALL_TABLES
SELECT on ALL_USERS
SELECT on ALL_CATALOG
SELECT on ALL_CONSTRAINTS
SELECT on ALL_CONS_COLUMNS
SELECT on ALL_TAB_COLS
SELECT on ALL_IND_COLUMNS
SELECT on ALL_LOG_GROUPS
SELECT on SYS.DBA_REGISTRY
SELECT on SYS.OBJ$
SELECT on DBA_TABLESPACES
SELECT on ALL_TAB_PARTITIONS
SELECT on ALL_ENCRYPTED_COLUMNS
* SELECT on all tables migrated
```

If you want to capture and apply changes (CDC), then you also need the following privileges.

```
EXECUTE on DBMS_LOGMNR
SELECT on V_$LOGMNR_LOGS
SELECT on V_$LOGMNR_CONTENTS
LOGMINING /* For Oracle 12c and higher. */
* ALTER for any table being replicated (if you want AWS DMS to add supplemental logging)
```

For Oracle versions before 11.2.0.3, you need the following privileges.

```
SELECT on DBA_OBJECTS /* versions before 11.2.0.3 */
SELECT on ALL_VIEWS (required if views are exposed)
```

2. If your Oracle database is an AWS RDS database, then connect to it as an administrative user, and run the following command to ensure that archive logs are retained on your RDS source for 24 hours:

```
exec rdsadmin.rdsadmin_util.set_configuration('archivelog retention hours',24);
```

If your Oracle source is an AWS RDS database, it will be placed in ARCHIVELOG MODE if, and only if, you enable backups.

3. Run the following command to enable supplemental logging at the database level, which AWS DMS requires:
  - a. In Oracle SQL:

```
ALTER DATABASE ADD SUPPLEMENTAL LOG DATA;
```

- b. In RDS:

```
exec rdsadmin.rdsadmin_util.alter_supplemental_logging('ADD');
```

4. Use the following command to enable identification key supplemental logging at the database level. AWS DMS requires supplemental key logging at the database level. The exception is if you allow AWS DMS to automatically add supplemental logging as needed or enable key-level supplemental logging at the table level:

- a. In Oracle SQL:

AWS Database Migration Service  
Step-by-Step Migration Guide  
Step 3: Configure Your PostgreSQL Target Database

---

```
ALTER DATABASE ADD SUPPLEMENTAL LOG DATA (PRIMARY KEY) COLUMNS;
```

b. In RDS:

```
exec rdsadmin.rdsadmin_util.alter_supplemental_logging('ADD', 'PRIMARY KEY');
```

Your source database incurs a small bit of overhead when key level supplemental logging is enabled. Therefore, if you are migrating only a subset of your tables, then you might want to enable key level supplemental logging at the table level.

5. To enable key level supplemental logging at the table level, use the following command.

```
ALTER TABLE table_name ADD SUPPLEMENTAL LOG DATA (PRIMARY KEY) COLUMNS;
```

If a table does not have a primary key, then you have two options.

- You can add supplemental logging on all columns involved in the first unique index on the table (sorted by index name).
- You can add supplemental logging on all columns of the table.

To add supplemental logging on a subset of columns in a table, such as those involved in a unique index, run the following command.

```
ALTER TABLE table_name  
ADD SUPPLEMENTAL LOG GROUP example_log_group (column_list) ALWAYS;
```

To add supplemental logging for all columns of a table, run the following command.

```
ALTER TABLE table_name ADD SUPPLEMENTAL LOG DATA (ALL) COLUMNS;
```

6 Create a user for AWS SCT.

```
CREATE USER oracle_sct_user IDENTIFIED BY password;  
  
GRANT CONNECT TO oracle_sct_user;  
GRANT SELECT_CATALOG_ROLE TO oracle_sct_user;  
GRANT SELECT ANY DICTIONARY TO oracle_sct_user;
```

## Step 3: Configure Your PostgreSQL Target Database

1. If the schemas you are migrating do not exist on the PostgreSQL database, then create the schemas.
2. Create the AWS DMS user to connect to your target database, and grant Superuser or the necessary individual privileges (or use the master username for RDS).

```
CREATE USER postgresql_dms_user WITH PASSWORD 'password';  
ALTER USER postgresql_dms_user WITH SUPERUSER;
```

3. Create a user for AWS SCT.

```
CREATE USER postgresql_sct_user WITH PASSWORD 'password';
```

AWS Database Migration Service  
Step-by-Step Migration Guide  
Step 4: Use the AWS Schema Conversion Tool (AWS  
SCT) to Convert the Oracle Schema to PostgreSQL

```
GRANT CONNECT ON DATABASE database_name TO postgresql_sct_user;  
GRANT USAGE ON SCHEMA schema_name TO postgresql_sct_user;  
GRANT SELECT ON ALL TABLES IN SCHEMA schema_name TO postgresql_sct_user;  
GRANT ALL ON SEQUENCES IN SCHEMA schema_name TO postgresql_sct_user;
```

## Step 4: Use the AWS Schema Conversion Tool (AWS SCT) to Convert the Oracle Schema to PostgreSQL

Before you migrate data to PostgreSQL, you convert the Oracle schema to a PostgreSQL schema. Do the following:

1. Launch AWS SCT. In AWS SCT, choose **File**, then choose **New Project**. Create a new project called AWS Schema Conversion Tool Oracle to PostgreSQL. Enter the following information in the New Project window and then choose **OK**.

Parameter	Description
<b>Project Name</b>	Type AWS Schema Conversion Tool Oracle to PostgreSQL.
<b>Location</b>	Use the default <b>Projects</b> folder and the default <b>Transactional Database (OLTP)</b> option.
<b>Source Database Engine</b>	Choose <b>Oracle</b> .
<b>Target Database Engine</b>	Choose <b>Amazon RDS for PostgreSQL</b> .

**New Project**

Enter the name, location and type of the new migration project.

Project name: AWS Schema Conversion Tool Oracle to PostgreSQL

Location: [Browse]

Transactional Database (OLTP)  
 Data Warehouse (OLAP)

Source Database Engine: Oracle

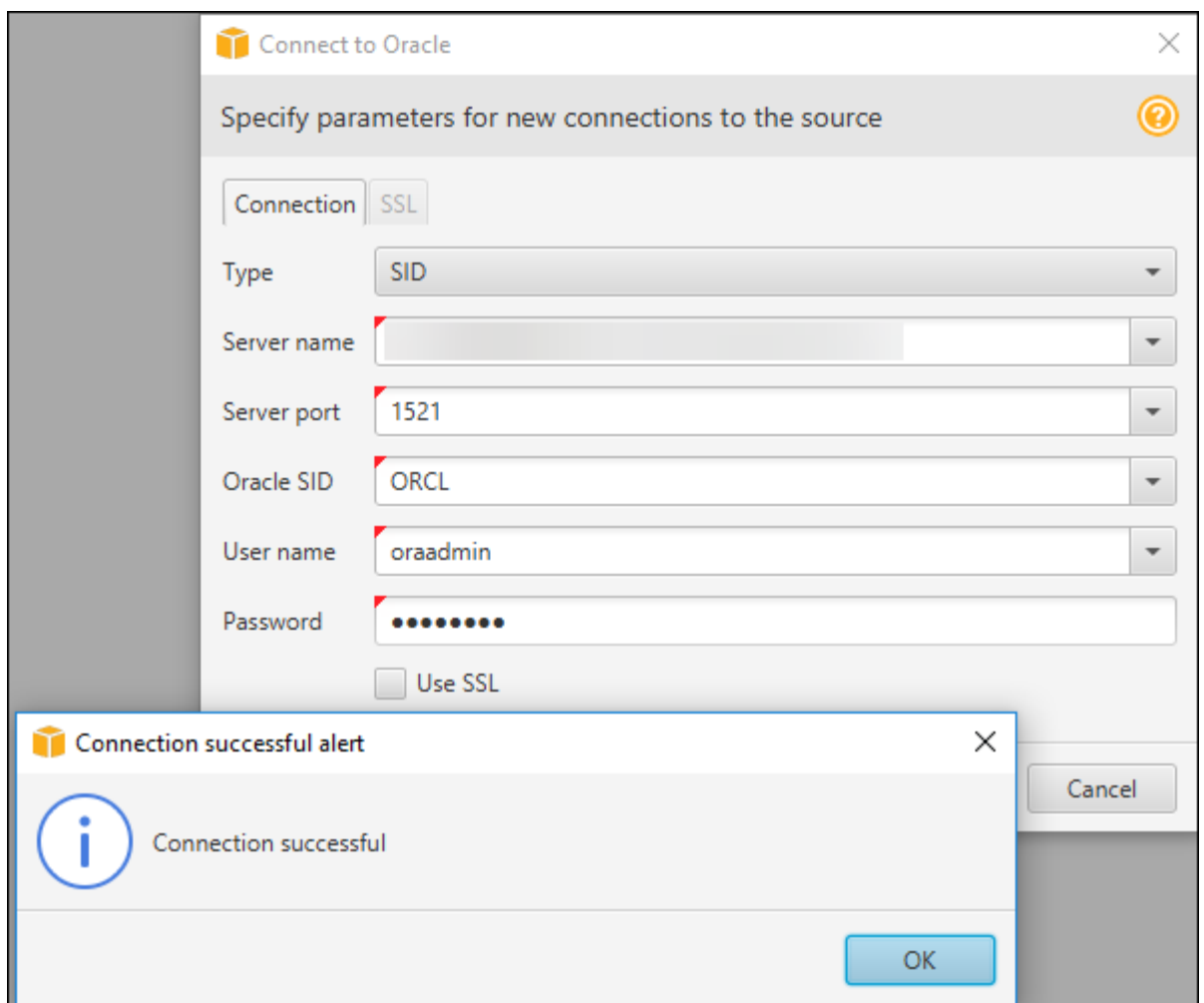
Target Database Engine: Amazon RDS for PostgreSQL

OK Cancel

2. Choose **Connect to Oracle**. In the **Connect to Oracle** dialog box, enter the following information, and then choose **Test Connection**.

AWS Database Migration Service  
Step-by-Step Migration Guide  
Step 4: Use the AWS Schema Conversion Tool (AWS  
SCT) to Convert the Oracle Schema to PostgreSQL

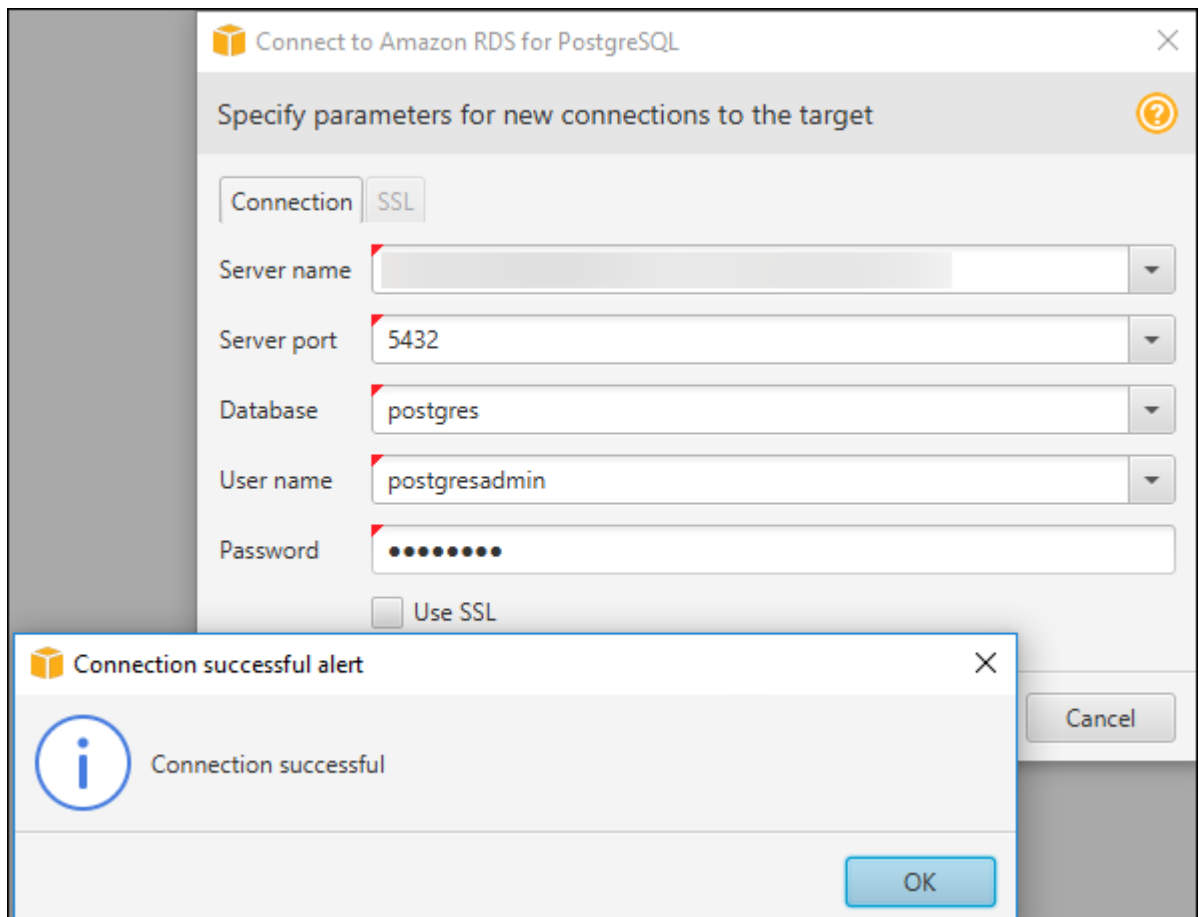
Parameter	Description
Type	Choose <b>SID</b> .
Server name	Type the server name.
Server port	Type the Oracle port number. The default is 1521.
Oracle SID	Type the database SID.
User name	Type the Oracle admin username.
Password	Provide the password for the admin user.



3. Choose **OK** to close the alert box, then choose **OK** to close the dialog box and to start the connection to the Oracle DB instance. The database structure on the Oracle DB instance is shown.
4. Choose **Connect to Amazon RDS for PostgreSQL**. In the **Connect to Amazon PostgreSQL** dialog box, enter the following information and then choose **Test Connection**.

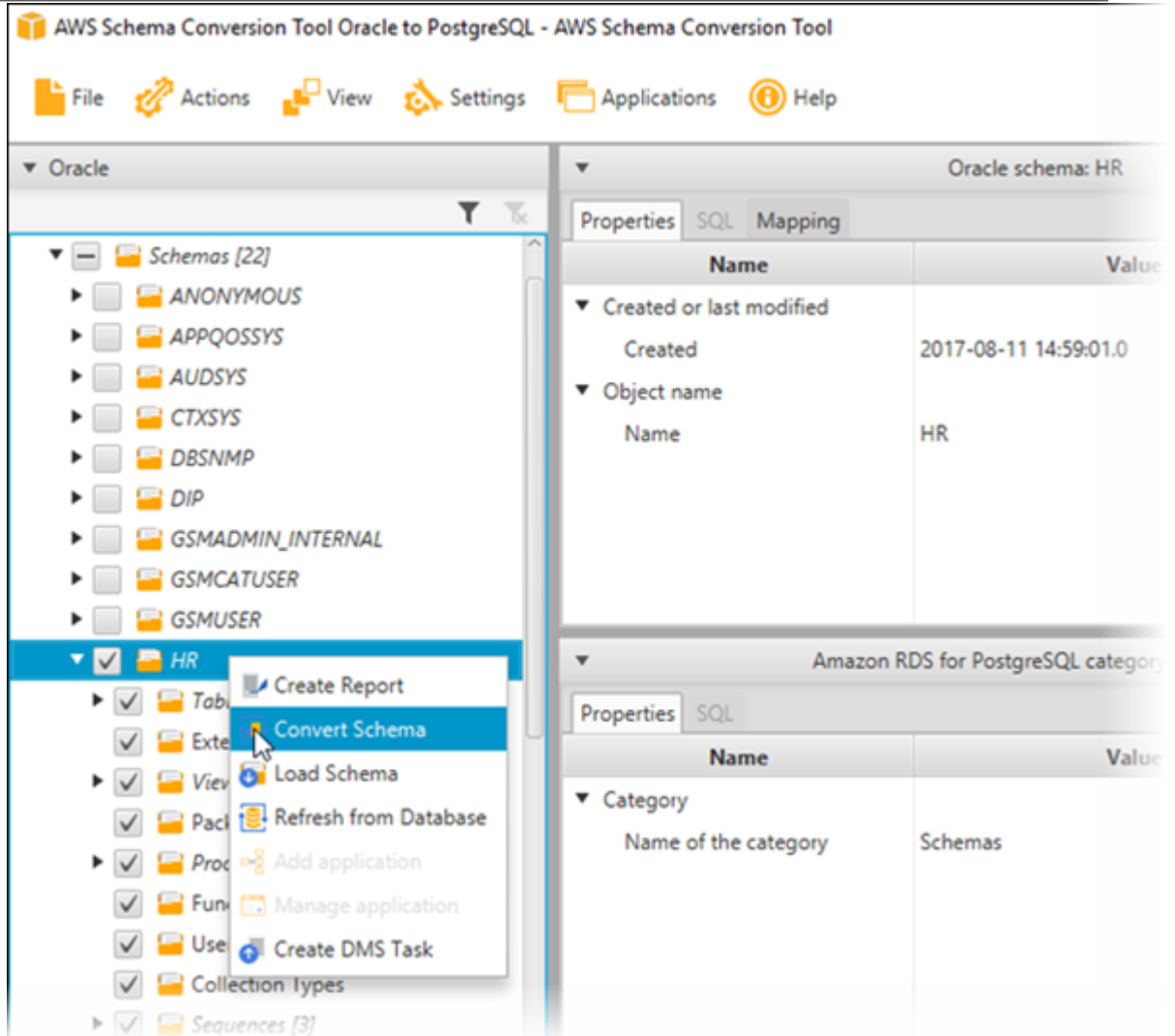
AWS Database Migration Service  
Step-by-Step Migration Guide  
Step 4: Use the AWS Schema Conversion Tool (AWS  
SCT) to Convert the Oracle Schema to PostgreSQL

Parameter	Description
Server name	Type the server name.
Server port	Type the PostgreSQL port number. The default is 5432.
Database	Type the database name.
User name	Type the PostgreSQL admin username.
Password	Provide the password for the admin user.



5. Choose **OK** to close the alert box, then choose **OK** to close the dialog box to start the connection to the PostgreSQL DB instance.
6. Open the context (right-click) menu for the schema to migrate, and then choose **Convert schema**.

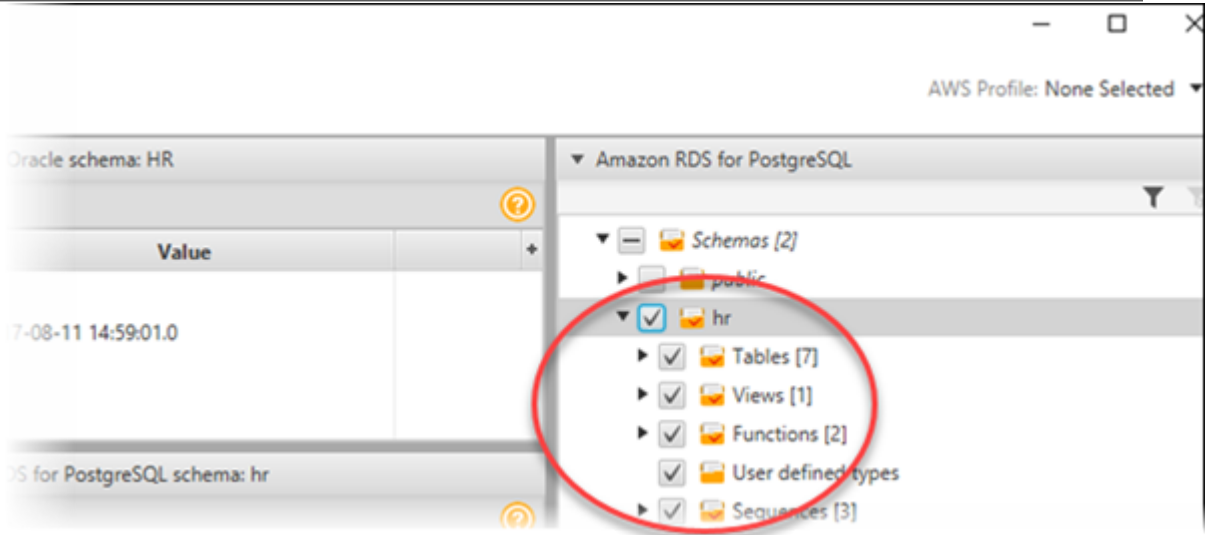
AWS Database Migration Service  
Step-by-Step Migration Guide  
Step 4: Use the AWS Schema Conversion Tool (AWS  
SCT) to Convert the Oracle Schema to PostgreSQL



7. Choose **Yes** for the confirmation message. AWS SCT then converts your schemas to the target database format.



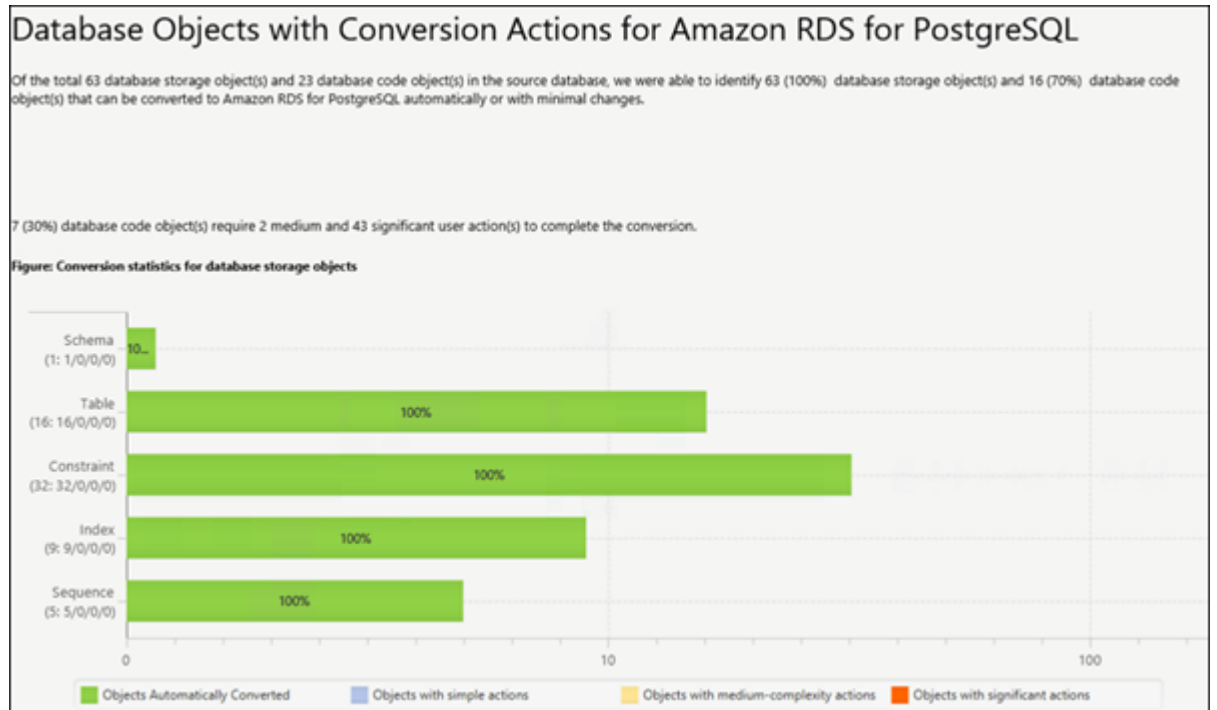
AWS Database Migration Service  
Step-by-Step Migration Guide  
Step 4: Use the AWS Schema Conversion Tool (AWS  
SCT) to Convert the Oracle Schema to PostgreSQL



AWS SCT analyses the schema and creates a database migration assessment report for the conversion to PostgreSQL.

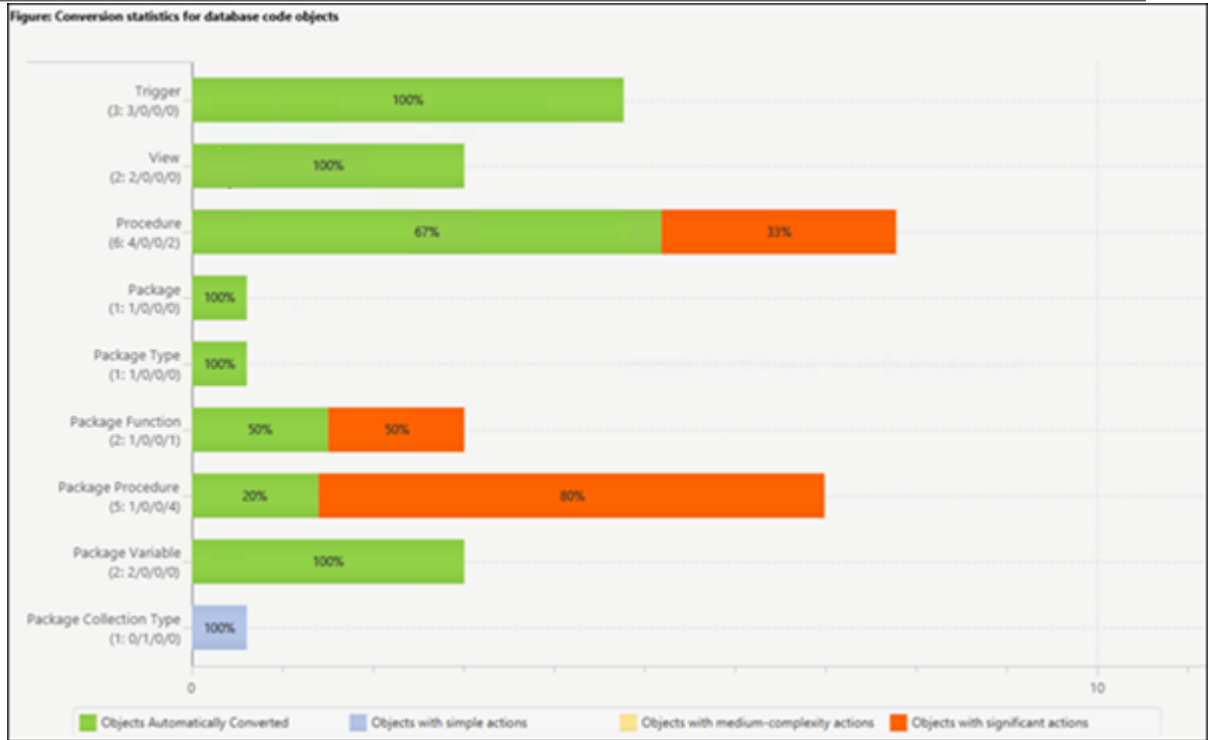
8. Select **Assessment Report View** from **View** to check the report.

The report breaks down by each object type and by how much manual change is needed to successfully convert it.

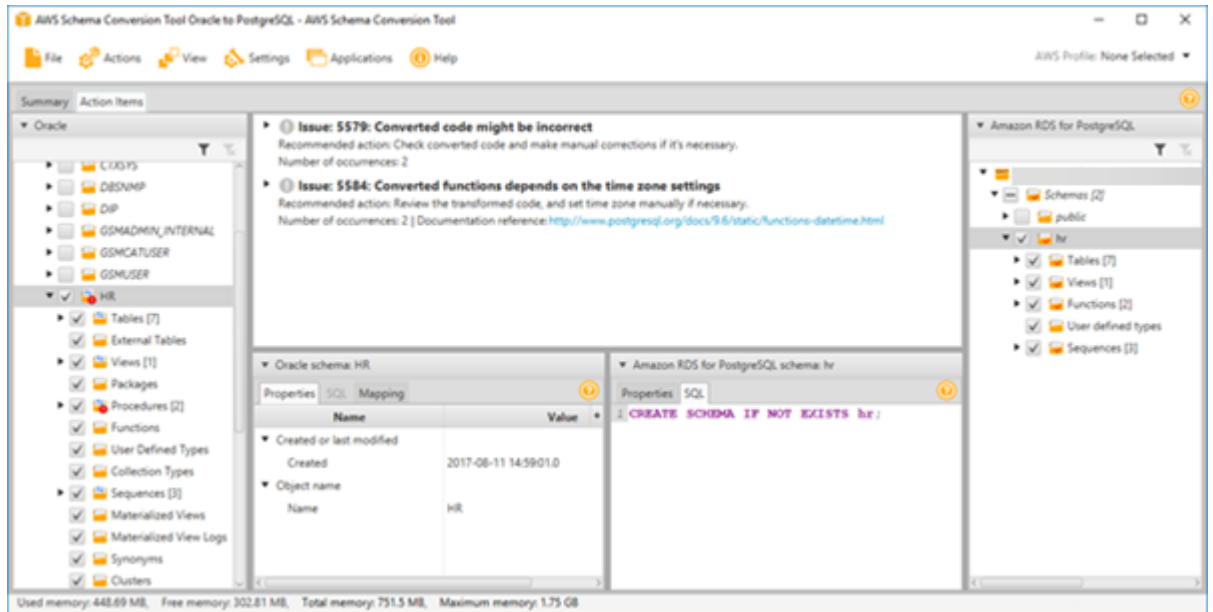


Generally packages, procedures, and functions are most likely to have some issues to resolve because they contain the most custom PL/SQL code. AWS SCT also provides hints about how to fix these objects.

AWS Database Migration Service  
 Step-by-Step Migration Guide  
 Step 4: Use the AWS Schema Conversion Tool (AWS  
 SCT) to Convert the Oracle Schema to PostgreSQL



9. Choose the **Action Items** tab.



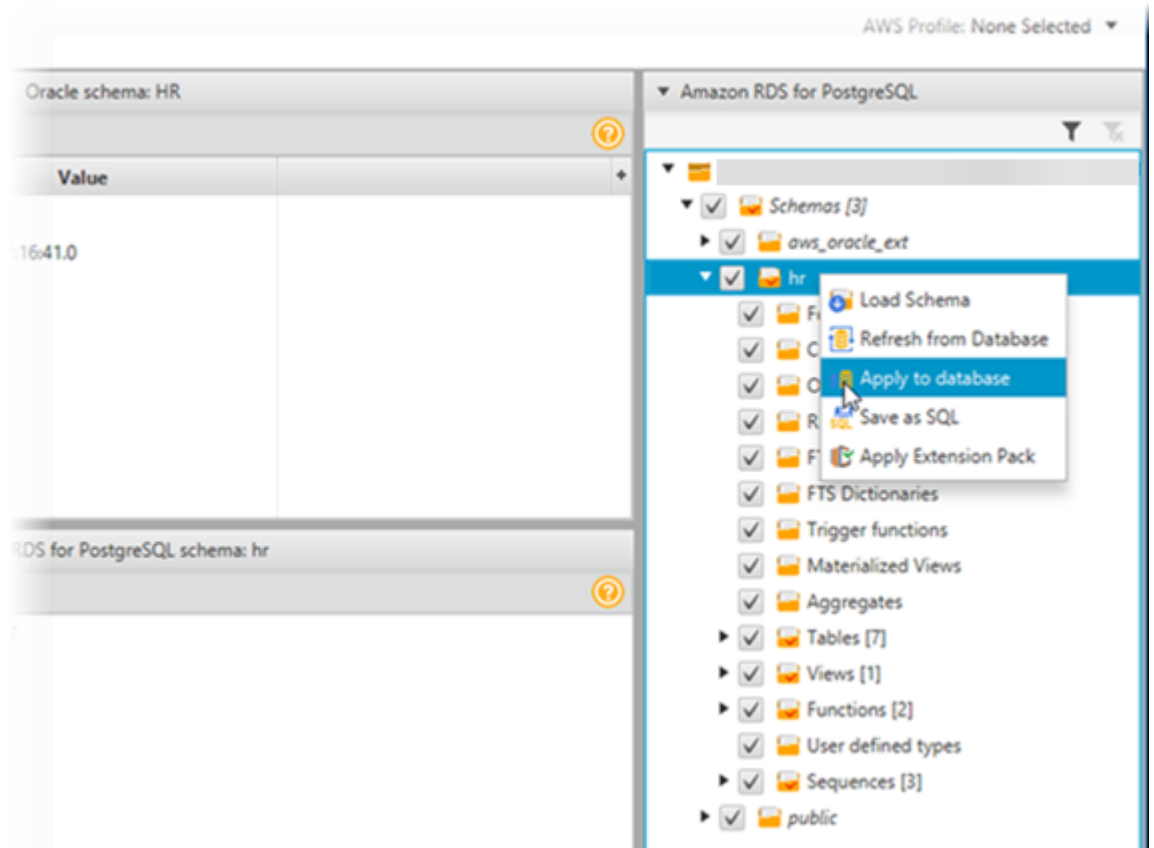
The **Action Items** tab shows each issue for each object that requires attention.

For each conversion issue, you can complete one of the following actions:

- a. Modify the objects on the source Oracle database so that AWS SCT can convert the objects to the target PostgreSQL database.
  - i. Modify the objects on the source Oracle database.
  - ii. Repeat the previous steps to convert the schema and check the assessment report.

AWS Database Migration Service  
Step-by-Step Migration Guide  
Step 4: Use the AWS Schema Conversion Tool (AWS  
SCT) to Convert the Oracle Schema to PostgreSQL

- iii. If necessary, repeat this process until there are no conversion issues.
- iv. Choose **Main View** from **View**, and open the context (right-click) menu for the target PostgreSQL schema, and choose **Apply to database** to apply the schema changes to the PostgreSQL database.

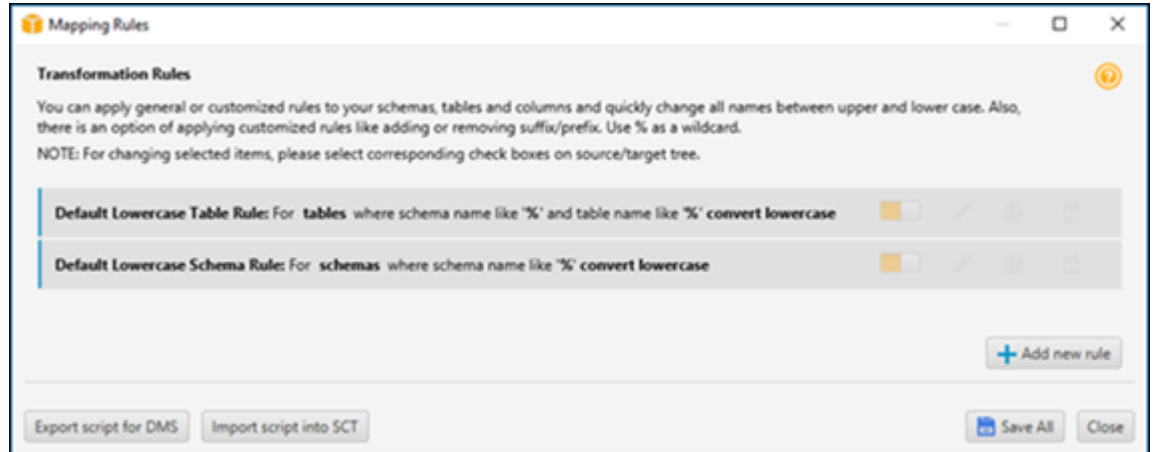


- b. Instead of modifying the source schema, modify scripts generated by AWS SCT before applying the scripts on the target PostgreSQL database.
  - i. Open the context (right-click) menu for the target PostgreSQL schema name, and select **Save as SQL**. Next, choose a name and destination for the script.
  - ii. In the script, modify the objects to correct conversion issues.
  - iii. Run the script on the target PostgreSQL database.

For more information, see [Converting Database Schema to Amazon RDS by Using the AWS Schema Conversion Tool](#) in the *AWS Schema Conversion Tool User Guide*.

10 Use AWS SCT to create mapping rules.

- a. Under **Settings**, select **Mapping Rules**.
- b. In addition to the two default mapping rules that convert schema names and table names to lower case, create additional mapping rules that are required based on the action items.
- c. Save the mapping rules.



- d. Click **Export script for DMS** to export a JSON format of all the transformations that the AWS DMS task will use to determine which object from the source corresponds to which object on the target. Click **Save**.

## Step 5: Create an AWS DMS Replication Instance

After validating the schema structure between source and target databases, continue with the core part of this walkthrough, which is the data migration. The following illustration shows a high-level view of the migration process.



An AWS DMS replication instance performs the actual data migration between source and target. The replication instance also caches the transaction logs during the migration. How much CPU and memory capacity a replication instance has influences the overall time required for the migration.

1. Sign in to the AWS Management Console, and select AWS DMS at <https://console.aws.amazon.com/dms/>. Next, choose **Create Migration**. If you are signed in as an AWS Identity and Access Management (IAM) user, then you must have the appropriate permissions to access AWS DMS. For more information on the permissions required, see [IAM Permissions Needed to Use AWS DMS](#).
2. Choose **Next** to start a database migration from the console's Welcome page.
3. On the **Create replication instance** page, specify your replication instance information.

Parameter	Description
<b>Name</b>	Select a name for your replication instance. If you will be using multiple replication servers or sharing an account, then choose a name that will help you quickly differentiate between the different servers.

Parameter	Description
<b>Description</b>	Type a brief description.
<b>Instance class</b>	Select the type of replication server to create. Each size and type of instance class will have increasing CPU, memory, and I/O capacity. Generally, the <code>t2</code> instances are for lower load tasks, and the <code>c4</code> instances are for higher load and more tasks.
<b>VPC</b>	Choose the VPC in which your replication instance will be launched. If possible, select the same VPC in which either your source or target database resides (or both).
<b>Multi-AZ</b>	When <b>Yes</b> is selected, AWS DMS creates a second replication server in a different Availability Zone for failover if there is a problem with the primary replication server.
<b>Publicly accessible</b>	If either your source or target database resides outside of the VPC in which your replication server resides, then you must make your replication server policy publicly accessible.

4. For the **Advanced** section, specify the following information.

Parameter	Description
<b>Allocated storage (GB)</b>	Amount of storage on the replication server for the AWS DMS task logs, including historical tasks logs. AWS DMS also uses disk storage to cache certain data while it replicates it from the source to the target. Additionally, more storage generally enables better IOPS on the server.
<b>Replication Subnet Group</b>	If you are running in a Multi-AZ configuration, then you will need at least two subnet groups.
<b>Availability zone</b>	Generally, performance is better if you locate your primary replication server in the same Availability Zone as your target database.
<b>VPC Security Group(s)</b>	Security groups enable you to control ingress and egress to your VPC. AWS DMS allows you to associate one or more security groups with the VPC in which your replication server is launched.
<b>KMS key</b>	With AWS DMS, all data is encrypted at rest using a KMS encryption key. By default, AWS DMS will create a new encryption key for your replication server. However, you may choose to use an existing key.

For information about the KMS key, see [Setting an Encryption Key and Specifying KMS Permissions](#).

5. Click **Next**.

## Step 6: Create AWS DMS Source and Target Endpoints

While your replication instance is being created, you can specify the source and target database endpoints using the AWS Management Console. However, you can only test connectivity after the replication instance has been created, because the replication instance is used in the connection.

1. Specify your connection information for the source Oracle database and the target PostgreSQL database. The following table describes the source settings.

Parameter	Description
<b>Endpoint Identifier</b>	Type a name, such as <code>Orasource</code> .
<b>Source Engine</b>	Choose <b>oracle</b> .
<b>Server name</b>	Provide the Oracle DB instance server name.
<b>Port</b>	The port of the database. The default for Oracle is 1521.
<b>SSL mode</b>	Choose an SSL mode if you want to enable encryption for your connection's traffic.
<b>Username</b>	The user you want to use to connect to the source database.
<b>Password</b>	Provide the password for the user.
<b>SID</b>	Provide the Oracle database name.

The following table describes the advanced source settings.

Parameter	Description
<b>Extra connection attributes</b>	<p>Extra parameters that you can set in an endpoint to add functionality or change the behavior of AWS DMS. Some of the most common and convenient parameters to set for an Oracle source database are the following. Separate multiple entries from each other by using a semi-colon (;).</p> <p>* <code>addSupplementalLogging</code> - This parameter automatically configures supplemental logging when set to <code>Y</code>.</p> <p>* <code>useLogminerReader</code> - By default, AWS DMS uses Logminer on the Oracle database to capture all of the changes on the source database. The other mode is called Binary Reader. When using Binary Reader instead of Logminer, AWS DMS copies the archived redo log from the source Oracle database to the replication server and reads the entire log in order to capture changes.</p>

Parameter	Description
	<p>The Binary Reader option is recommended if you are using ASM since it has performance advantages over Logminer on ASM. If your source database is 12c, then the Binary Reader option is currently the only way to capture CDC changes in Oracle for LOB objects.</p> <p>To use Logminer, enter the following:  <code>useLogminerReader=Y</code></p> <p>To use Binary Reader, enter the following:  <code>useLogminerReader=N; useBfile=Y`</code></p>
KMS key	Enter the KMS key if you choose to encrypt your replication instance's storage.

For information about extra connection attributes, see [Using Extra Connection Attributes with AWS Database Migration Service](#).

The following table describes the target settings.

Parameter	Description
<b>Endpoint Identifier</b>	Type a name, such as <code>Postgrestarget</code> .
<b>Target Engine</b>	Choose <b>postgres</b> .
<b>Servername</b>	Provide the PostgreSQL DB instance server name.
<b>Port</b>	The port of the database. The default for PostgreSQL is 5432.
<b>SSL mode</b>	Choose <b>None</b> .
<b>Username</b>	The user you want to use to connect to the target database.
<b>Password</b>	Provide the password for the PostgreSQL DB instance.

The following is an example of the completed page.

### Connect source and target database endpoints

✔ Replication instance created successfully.

Your database endpoint can be on-premise, in EC2, RDS or in the cloud. Define the connection details below. It is recommended that you test your endpoint connections here to avoid errors later.

Source database connection details	Target database connection details
<b>Endpoint identifier*</b> <input type="text" value="Orasource"/>	<b>Endpoint identifier*</b> <input type="text" value="Postgrestarget"/>
<b>Source engine*</b> <input type="text" value="oracle"/>	<b>Target engine*</b> <input type="text" value="postgres"/>
<b>Server name*</b> <input type="text"/>	<b>Server name*</b> <input type="text"/>
<b>Port*</b> <input type="text" value="1521"/>	<b>Port*</b> <input type="text" value="5432"/>
<b>SSL mode*</b> <input type="text" value="none"/>	<b>SSL mode*</b> <input type="text" value="none"/>
<b>User name*</b> <input type="text" value="oraadmin"/>	<b>User name*</b> <input type="text" value="postgresadmin"/>
<b>Password*</b> <input type="password" value="*****"/>	<b>Password*</b> <input type="password" value="*****"/>
<b>SID*</b> <input type="text" value="ORCL"/>	<b>Database name*</b> <input type="text" value="postgres"/>
<a href="#">▶ Advanced</a>	<a href="#">▶ Advanced</a>
<input type="button" value="Run test"/>	<input type="button" value="Run test"/>

2. After the endpoints and replication instance have been created, test each endpoint connection by choosing **Run test** for the source and target endpoints.
3. Drop foreign key constraints and triggers on the target database.

During the full load process, AWS DMS does not load tables in any particular order, so it may load the child table data before parent table data. As a result, foreign key constraints might be violated if they are enabled. Also, if triggers are present on the target database, then it may change data loaded by AWS DMS in unexpected ways.

4. If you do not have one, then generate a script that enables the foreign key constraints and triggers.

Later, when you want to add them to your migrated database, you can just run this script.

5. (Optional) Drop secondary indexes on the target database.

Secondary indexes (as with all indexes) can slow down the full load of data into tables since they need to be maintained and updated during the loading process. Dropping them can improve the



performance of your full load process. If you drop the indexes, then you will need to add them back later after the full load is complete.

6. Choose **Next**.

## Step 7: Create and Run Your AWS DMS Migration Task

Using an AWS DMS task, you can specify which schema to migrate and the type of migration. You can migrate existing data, migrate existing data and replicate ongoing changes, or replicate data changes only. This walkthrough migrates existing data and replicates ongoing changes.

1. On the **Create Task** page, specify the task options. The following table describes the settings.

Parameter	Description
<b>Task name</b>	Type a name for the migration task.
<b>Task description</b>	Type a description for the task.
<b>Source endpoint</b>	Shows the Oracle source endpoint.  If you have more than one endpoint in the account, then choose the correct endpoint from the list.
<b>Target endpoint</b>	Shows the PostgreSQL target endpoint.
<b>Replication instance</b>	Shows the AWS DMS replication instance.
<b>Migration type</b>	Choose the option <b>Migrate existing data and replicate ongoing changes</b> .
<b>Start task on create</b>	Select this option.

The page should look like the following:

AWS Database Migration Service  
Step-by-Step Migration Guide  
Step 7: Create and Run Your AWS DMS Migration Task

The screenshot shows the 'Create task' form in the AWS DMS console. The form includes the following fields and values:

- Task name\***: MigrateSchematoPostgres
- Task description\***: Migrate a schema from Oracle to PostgreSQL
- Source endpoint**: orasource
- Target endpoint**: postgrestarget
- Replication instance**: oracle2postgressql
- Migration type\***: Migrate existing data and replicate ongoing changes

A yellow warning box contains the following text:

Your source database is Oracle. Replicating ongoing changes requires supplemental logging to be turned on. Please ensure your archive logs are retained on the server for a sufficient amount of time, (24 hours is usually enough.) To set your archive log retention on RDS databases you can use the following command: `exec rdsadmin.rdsadmin_util.set_configuration('archive_log_retention_hours', 24);`

At the bottom, there is a checkbox for **Start task on create** which is checked.

2. Under **Task Settings**, choose **Do nothing** or **Truncate** for **Target table preparation mode**, because you have already created the tables using the AWS Schema Conversion Tool.

If the Oracle database has LOBs, then for **Include LOB columns in replication**, select **Full LOB mode** if you want to replicate the entire LOB for all tables. Select **Limited LOB mode** if you want to replicate the LOBs only up to a certain size. You specify the size of the LOB to migrate in **Max LOB size (kb)**.

It is best to select **Enable logging**. If you enable logging, then you can see any errors or warnings that the task encounters, and you can troubleshoot those issues.

The screenshot shows the 'Task Settings' section in the AWS DMS console. The settings are as follows:

- Target table preparation mode\***:  Do nothing,  Drop tables on target,  Truncate
- Include LOB columns in replication\***:  Don't include LOB columns,  Full LOB mode,  Limited LOB mode
- Max LOB size (kb)\***: 32
- Enable logging**:

At the bottom, there is a link for **Advanced Settings**.

3. Leave the Advanced settings at their default values.

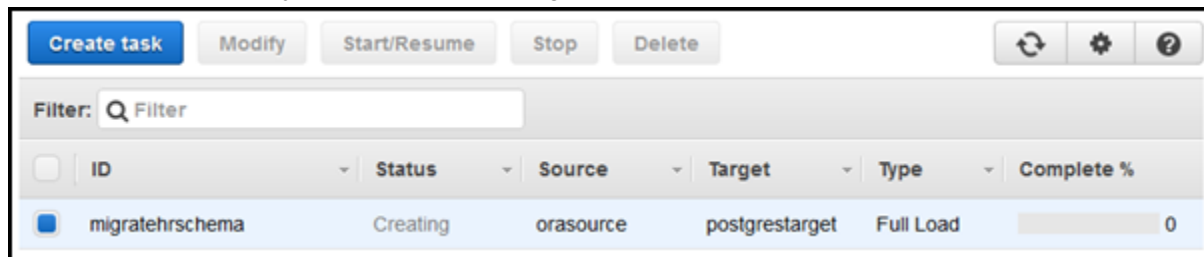
4. Choose **Table mappings**, and select the **JSON** tab. Next, select **Enable JSON editing**, and enter the table mappings you saved in the last step in [Step 4: Use the AWS Schema Conversion Tool \(AWS SCT\) to Convert the Oracle Schema to PostgreSQL \(p. 89\)](#).

The following is an example of mappings that convert schema names and table names to lowercase.

```
{
  "rules": [
    {
      "rule-type": "transformation",
      "rule-id": "100000",
      "rule-name": "Default Lowercase Table Rule",
      "rule-action": "convert-lowercase",
      "rule-target": "table",
      "object-locator": {
        "schema-name": "%",
        "table-name": "%"
      }
    },
    {
      "rule-type": "transformation",
      "rule-id": "100001",
      "rule-name": "Default Lowercase Schema Rule",
      "rule-action": "convert-lowercase",
      "rule-target": "schema",
      "object-locator": {
        "schema-name": "%"
      }
    }
  ]
}
```

5. Choose **Create task**. The task will begin immediately.

The Tasks section shows you the status of the migration task.



You can monitor your task if you chose **Enable logging** when you set up your task. You can then view the CloudWatch metrics by doing the following:

1. On the navigation pane, choose **Tasks**.
2. Choose your migration task.
3. Choose the **Task monitoring** tab, and monitor the task in progress on that tab.

When the full load is complete and cached changes are applied, the task will stop on its own.

4. On the target PostgreSQL database, enable foreign key constraints and triggers using the script you saved previously.
5. On the target PostgreSQL database, re-create the secondary indexes if you removed them previously.
6. In the AWS DMS console, start the AWS DMS task by clicking **Start/Resume** for the task.

The AWS DMS task keeps the target PostgreSQL database up-to-date with source database changes. AWS DMS will keep all of the tables in the task up-to-date until it is time to implement the application migration. The latency will be zero, or close to zero, when the target has caught up to the source.

## Step 8: Cut Over to PostgreSQL

To move connections from your Oracle database to your PostgreSQL database, do the following:

1. End all Oracle database dependencies and activities, such as running scripts and client connections.

The following query should return no results:

```
SELECT MACHINE, COUNT FROM V$SESSION GROUP BY MACHINE;
```

2. List any remaining sessions, and kill them.

```
SELECT SID, SERIAL#, STATUS FROM V$SESSION;  
  
ALTER SYSTEM KILL 'sid, serial_number' IMMEDIATE;
```

3. Shut down all listeners on the Oracle database.
4. Let the AWS DMS task apply the final changes from the Oracle database on the PostgreSQL database.

```
ALTER SYSTEM CHECKPOINT;
```

5. In the AWS DMS console, stop the AWS DMS task by clicking **Stop** for the task, and confirm that you want to stop the task.
6. (Optional) Set up a rollback.

You can optionally set up a rollback task, in case you run into a show stopping issue, by creating a task going in the opposite direction. Because all tables should be in sync between both databases, you only need to set up a CDC task. Therefore, you do not have to disable any foreign key constraints. Now that the source and target databases are reversed, you must follow the instructions in the following sections:

- [Using a PostgreSQL Database as a Source for AWS Database Migration Service](#)
- [Using an Oracle Database as a Target for AWS Database Migration Service](#)
  - a. Disable triggers on the source Oracle database.

```
SELECT 'ALTER TRIGGER' || owner || '.' || trigger_name || 'DISABLE';  
FROM DBA_TRIGGERS WHERE OWNER = 'schema_name';
```

You do not have to disable the foreign key constraints. During the CDC process, foreign key constraints are updated in the same order as they are updated by application users.

- b. Create a new CDC-only AWS DMS task with the endpoints reversed (source PostgreSQL endpoint and target Oracle endpoint database). See [Step 7: Create and Run Your AWS DMS Migration Task](#) (p. 101).

For the rollback task, set **Migration type** to **Replicate data changes only** and **Target table preparation mode** to **Do nothing**.

- c. Start the AWS DMS task to enable you to push changes back to the original source Oracle database from the new PostgreSQL database if rollback is necessary.
7. Connect to the PostgreSQL database, and enable triggers.

```
ALTER TABLE table_name ENABLE TRIGGER ALL;
```

8. If you set up a rollback, then complete the rollback setup.
  - a. Start the application services on new target PostgreSQL database (including scripts, client software, and so on).
  - b. Add Cloudwatch monitoring on your new PostgreSQL database. See [Monitoring Amazon RDS](#).

## Rolling Back the Migration

If there are major issues with the migration that cannot be resolved in a timely manner, you can roll back the migration. These steps assume that you have already prepared for the rollback as described in [Step 8: Cut Over to PostgreSQL \(p. 104\)](#).

1. Stop all application services on the target PostgreSQL database.
2. Let the AWS DMS task replicate remaining changes back to the source Oracle database.
3. Stop the PostgreSQL to Oracle AWS DMS task.
4. Start all applications back on the source Oracle database.

## Troubleshooting

The two most common problem areas when working with Oracle as a source and PostgreSQL as a target are: supplemental logging and case sensitivity.

- Supplemental logging – With Oracle, in order to replicate change data, supplemental logging must be enabled. However, if you enable supplemental logging at the database level, it sometimes still needs to be enabled when new tables are created. The best remedy for this is to allow AWS DMS to enable supplemental logging for you by using the extra connection attribute:

```
addSupplementalLogging=Y
```

- Case sensitivity: Oracle is case-insensitive (unless you use quotes around your object names). However, text appears in uppercase. Thus, AWS DMS defaults to naming your target objects in uppercase. In most cases, you'll want to use transformations to change schema, table, and column names to lower case.

For more tips, see the AWS DMS troubleshooting section in the [AWS DMS User Guide](#).

To troubleshoot issues specific to Oracle, see the Oracle troubleshooting section:

[Troubleshooting Oracle Specific Issues](#)

To troubleshoot PostgreSQL issues, see the PostgreSQL troubleshooting section:

[Troubleshooting PostgreSQL Specific Issues](#)

# Migrating an Amazon RDS for Oracle Database to Amazon Redshift

This walkthrough gets you started with heterogeneous database migration from Amazon RDS for Oracle to Amazon Redshift using AWS Database Migration Service (AWS DMS) and the AWS Schema Conversion Tool (AWS SCT). This introductory exercise doesn't cover all scenarios but provides you with a good understanding of the steps involved in such a migration.

It is important to understand that AWS DMS and AWS SCT are two different tools and serve different needs. They don't interact with each other in the migration process. At a high level, the steps involved in this migration are the following:

1. Using the AWS SCT to do the following:
  - Run the conversion report for Oracle to Amazon Redshift to identify the issues, limitations, and actions required for the schema conversion.
  - Generate the schema scripts and apply them on the target before performing the data load by using AWS DMS. AWS SCT performs the necessary code conversion for objects like procedures and views.
2. Identify and implement solutions to the issues reported by AWS SCT.
3. Disable foreign keys or any other constraints that might impact the AWS DMS data load.
4. AWS DMS loads the data from source to target using the Full Load approach. Although AWS DMS is capable of creating objects in the target as part of the load, it follows a minimalistic approach to efficiently migrate the data so that it doesn't copy the entire schema structure from source to target.
5. Perform postmigration activities such as creating additional indexes, enabling foreign keys, and making the necessary changes in the application to point to the new database.

This walkthrough uses a custom AWS CloudFormation template to create RDS DB instances for Oracle and Amazon Redshift. It then uses a SQL command script to install a sample schema and data onto the RDS Oracle DB instance that you then migrate to Amazon Redshift.

This walkthrough takes approximately two hours to complete. Be sure to follow the instructions to delete resources at the end of this walkthrough to avoid additional charges.

## Topics

- [Prerequisites \(p. 106\)](#)
- [Migration Architecture \(p. 107\)](#)
- [Step-by-Step Migration \(p. 108\)](#)
- [Next Steps \(p. 139\)](#)

## Prerequisites

The following prerequisites are also required to complete this walkthrough:

- Familiarity with Amazon RDS, Amazon Redshift, the applicable database technologies, and SQL.
- The custom scripts that include creating the tables to be migrated and SQL queries for confirming the migration, as listed following:
  - `Oracle_Redshift_For_DMSDemo.template`--an AWS CloudFormation template
  - `Oraclesalesstarschema.sql`--SQL statements to build the SH schema

These scripts are available at the following link: [dms-sbs-RDSOracle2Redshift.zip](#)

Each step in the walkthrough also contains a link to download the file involved or includes the exact query in the step.

- An AWS account with AWS Identity and Access Management (IAM) credentials that allow you to launch RDS, AWS Database Migration Service (AWS DMS) instances, and Amazon Redshift clusters in your AWS Region. For information about IAM credentials, see [Creating an IAM User](#).
- Basic knowledge of the Amazon Virtual Private Cloud (Amazon VPC) service and of security groups. For information about using Amazon VPC with Amazon RDS, see [Virtual Private Clouds \(VPCs\) and Amazon RDS](#). For information about Amazon RDS security groups, see [Amazon RDS Security Groups](#). For information about using Amazon Redshift in a VPC, see [Managing Clusters in an Amazon Virtual Private Cloud \(VPC\)](#).
- An understanding of the supported features and limitations of AWS DMS. For information about AWS DMS, see [What Is AWS Database Migration Service?](#)
- Knowledge of the supported data type conversion options for Oracle and Amazon Redshift. For information about data types for Oracle as a source, see [Using an Oracle Database as a Source for AWS Database Migration Service](#). For information about data types for Amazon Redshift as a target, see [Using an Amazon Redshift Database as a Target for AWS Database Migration Service](#).

For more information about AWS DMS, see [the AWS DMS documentation](#).

## Migration Architecture

This walkthrough uses AWS CloudFormation to create a simple network topology for database migration that includes the source database, the replication instance, and the target database in the same VPC. For more information on AWS CloudFormation, see [the CloudFormation documentation](#).

We provision the AWS resources that are required for this AWS DMS walkthrough through AWS CloudFormation. These resources include a VPC and Amazon RDS instance for Oracle and an Amazon Redshift cluster. We provision through CloudFormation because it simplifies the process, so we can concentrate on tasks related to data migration. When you create a stack from the CloudFormation template, it provisions the following resources:

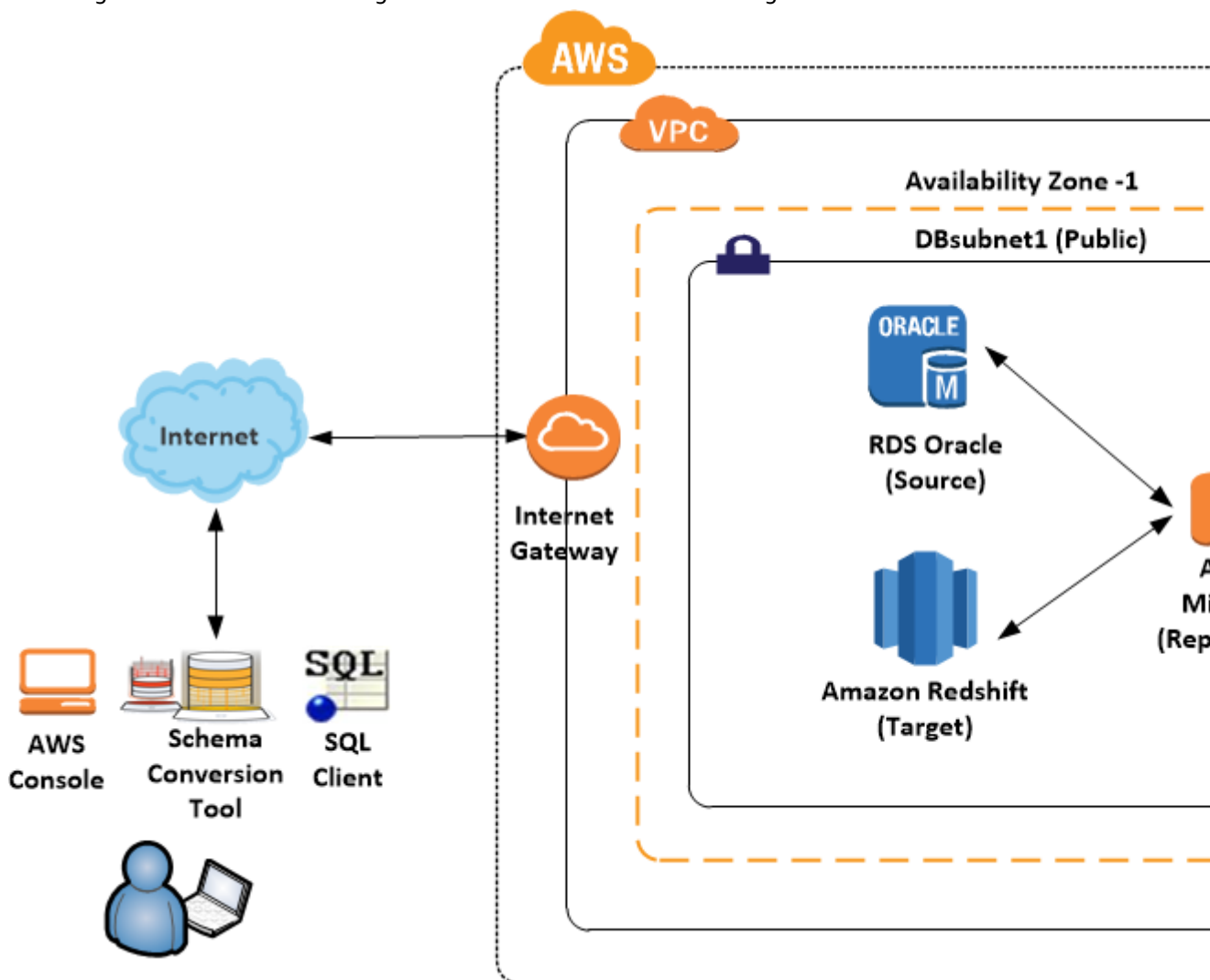
- A VPC with CIDR (10.0.0.0/24) with two public subnets in your region, DBSubnet1 at the address 10.0.0.0/26 in Availability Zone (AZ) 1 and DBSubnet2 at the address 10.0.0.64/26, in AZ 12.
- A DB subnet group that includes DBSubnet1 and DBSubnet2.
- Oracle RDS Standard Edition Two with these deployment options:
  - License Included
  - Single-AZ setup
  - db.m3.medium or equivalent instance class
  - Port 1521
  - Default option and parameter groups
- Amazon Redshift cluster with these deployment options:
  - dc1.large
  - Port 5439
  - Default parameter group
- A security group with ingress access from your computer or 0.0.0.0/0 (access from anywhere) based on the input parameter

We have designed the CloudFormation template to require few inputs from the user. It provisions the necessary AWS resources with minimum recommended configurations. However, if you want to change

some of the configurations and parameters, such as the VPC CIDR block and Amazon RDS instance types, feel free to update the template.

We use the AWS Management Console to provision the AWS DMS resources, such as the replication instance, endpoints, and tasks. You install client tools such as SQL Workbench/J and the AWS Schema Conversion Tool (AWS SCT) on your local computer to connect to the Amazon RDS instances.

Following is an illustration of the migration architecture for this walkthrough.



## Step-by-Step Migration

In the following sections, you can find step-by-step instructions for migrating an Amazon RDS for Oracle database to Amazon Redshift. These steps assume that you have already prepared your source database as described in preceding sections.

### Topics

- [Step 1: Launch the RDS Instances in a VPC by Using the CloudFormation Template \(p. 109\)](#)
- [Step 2: Install the SQL Tools and AWS Schema Conversion Tool on Your Local Computer \(p. 113\)](#)



- [Step 3: Test Connectivity to the Oracle DB Instance and Create the Sample Schema \(p. 116\)](#)
- [Step 4: Test the Connectivity to the Amazon Redshift Database \(p. 119\)](#)
- [Step 5: Use AWS SCT to Convert the Oracle Schema to Amazon Redshift \(p. 121\)](#)
- [Step 6: Validate the Schema Conversion \(p. 127\)](#)
- [Step 7: Create an AWS DMS Replication Instance \(p. 128\)](#)
- [Step 8: Create AWS DMS Source and Target Endpoints \(p. 129\)](#)
- [Step 9: Create and Run Your AWS DMS Migration Task \(p. 132\)](#)
- [Step 10: Verify That Your Data Migration Completed Successfully \(p. 136\)](#)
- [Step 11: Delete Walkthrough Resources \(p. 138\)](#)

## Step 1: Launch the RDS Instances in a VPC by Using the CloudFormation Template

Before you begin, you'll need to download an AWS CloudFormation template. Follow these instructions:

1. Download the following archive to your computer: <http://docs.aws.amazon.com/dms/latest/sbs/samples/dms-sbs-RDSOracle2Redshift.zip>
2. Extract the CloudFormation template (`Oracle_Redshift_For_DMSDemo.template`) from the archive.
3. Copy and paste the `Oracle_Redshift_For_DMSDemo.template` file into your current directory.

Now you need to provision the necessary AWS resources for this walkthrough.

1. Sign in to the AWS Management Console and open the AWS CloudFormation console at <https://console.aws.amazon.com/cloudformation>.
2. Choose **Create Stack**.
3. On the **Select Template** page, choose **Upload a template to Amazon S3**.
4. Click **Choose File**, and then choose the `Oracle_Redshift_For_DMSDemo.template` file that you extracted from the `dms-sbs-RDSOracle2Redshift.zip` archive.
5. Choose **Next**. On the **Specify Details** page, provide parameter values as shown following.

For This Parameter	Do This
*Stack Name *	Type <code>OracletoRedshiftDWusingDMS</code> .
<b>OracleDBName</b>	Provide a unique name for your database. The name should begin with a letter. The default is <code>ORCL</code> .
<b>OracleDBUsername</b>	Specify the admin (DBA) user for managing the Oracle instance. The default is <code>oraadmin</code> .
<b>OracleDBPassword</b>	Provide the password for the admin user. The default is <code>oraadmin123</code>
<b>RedshiftDBName</b>	Provide any unique name for your database. The name should begin with a letter. The default is <code>test</code> .
<b>RedshiftDBUsername</b>	Provide the password for the master user. The default is <code>Redshift#123</code> .

AWS Database Migration Service  
Step-by-Step Migration Guide  
Step 1: Launch the RDS Instances in a VPC  
by Using the CloudFormation Template

For This Parameter	Do This
<b>ClientIP</b>	Specify the IP address in CIDR (x.x.x.x/32) format for your local computer. You can get your IP address from <a href="http://whatsmyip.org">whatsmyip.org</a> . Your RDS instances' security group will allow ingress to this IP address. The default is access from anywhere (0.0.0.0/0), which is not recommended; you should use your IP address for this walkthrough.

## Create stack

- Select Template
- Specify Details**
- Options
- Review

### Specify Details

Specify a stack name and parameter values. You can use or change the default parameter values.

Stack name

### Parameters

#### Source Oracle Database Configuration

OracleDBName  Enter Oracle database name

OracleDBUsername  Enter database username

OracleDBPassword  Enter database password

#### Target Redshift Database Configuration

RedshiftDBName  Enter Redshift database name

RedshiftDBUsername  Enter master user name

RedshiftDBPassword  Enter master user password

#### Enter IP address for DB Security group Configuration

ClientIP

The IP address range that can be used to connect to the RDS instance. For more information, see [checkip.amazonaws.com](http://checkip.amazonaws.com) or [whatsmyip.org](http://whatsmyip.org).

- Choose **Next**. On the **Options** page, choose **Next**.
- On the **Review** page, review the details, and if they are correct choose **Create**.

## Create stack

- Select Template
- Specify Details
- Options
- Review**

## Review

### Template

Template URL	<a href="https://s3.amazonaws.com/154124131215/stacks/OracleDWtoRedshift_DMS.template">https://s3.amazonaws.com/154124131215/stacks/OracleDWtoRedshift_DMS.template</a>
Description	This CloudFormation sample template OracleDWtoRedshift_DMS used to test the datawarehouse migration using AWS DMS service from this template
Estimate cost	Link is not available

### Details

Stack name OracletoRedshiftDWusingDMS

#### Source Oracle Database Configuration

OracleDBName	ORCL
OracleDBUsername	oraadmin
OracleDBPassword	.....

#### Target Redshift Database Configuration

RedshiftDBName	test
RedshiftDBUsername	masteruser
RedshiftDBPassword	.....

#### Enter IP address for DB Security group Configuration

ClientIP 0.0.0.0/0

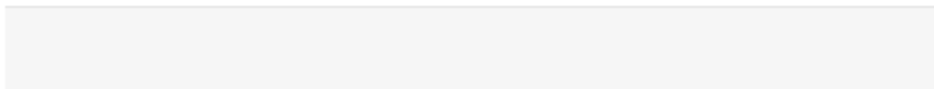
### Options

#### Tags

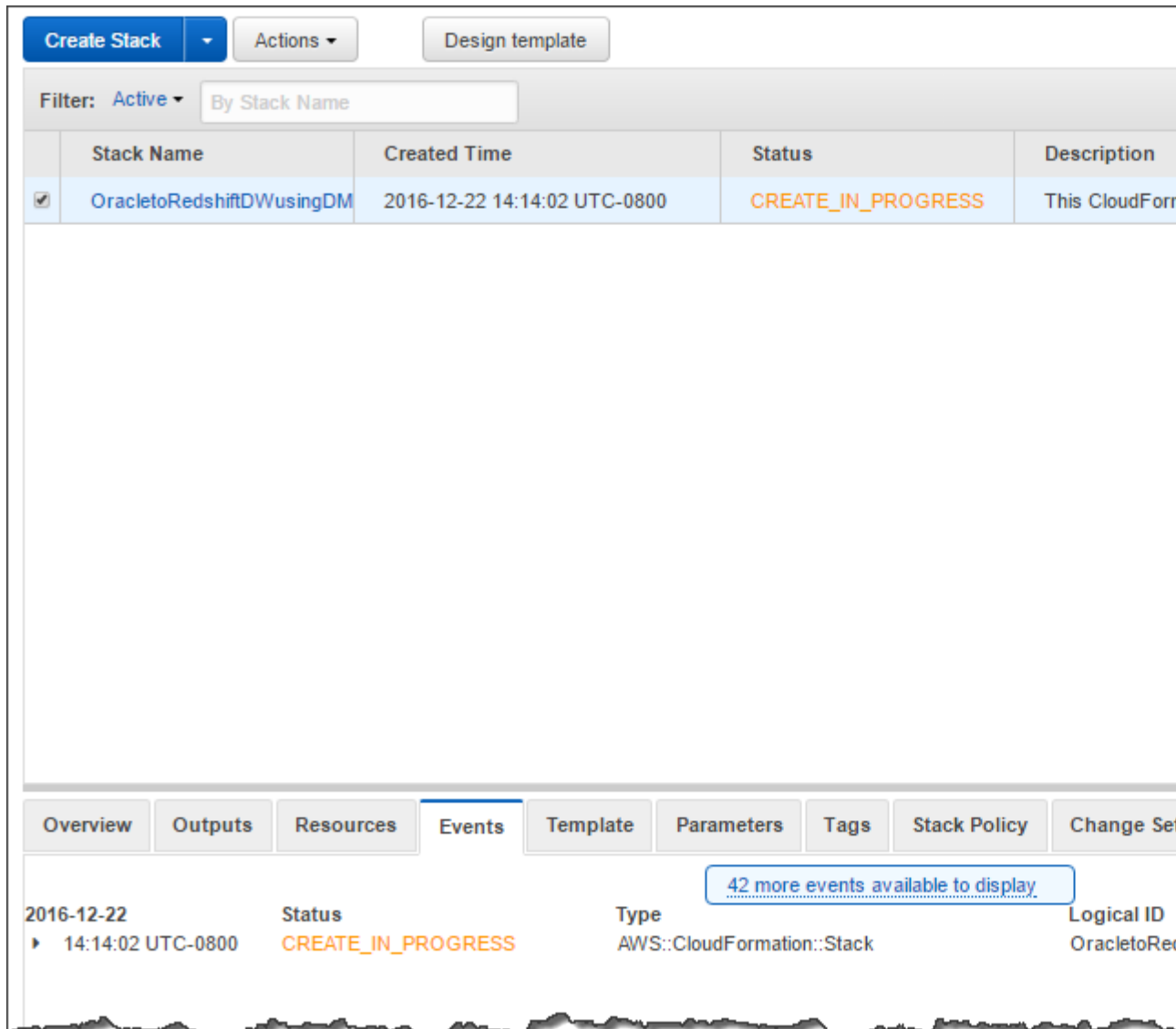
No tags provided

#### Advanced

Notification	
Timeout	none
Rollback on failure	Yes



8. AWS can take about 20 minutes or more to create the stack with an Amazon RDS Oracle instance and an Amazon Redshift cluster.



9. After the stack is created, select the **OracletoRedshiftDWusingDMS** stack, and then choose the **Outputs** view. Record the JDBC connection strings, **OracleJDBCConnectionString** and **RedshiftJDBCConnectionString**, for use later in this walkthrough to connect to the Oracle and Amazon Redshift databases.

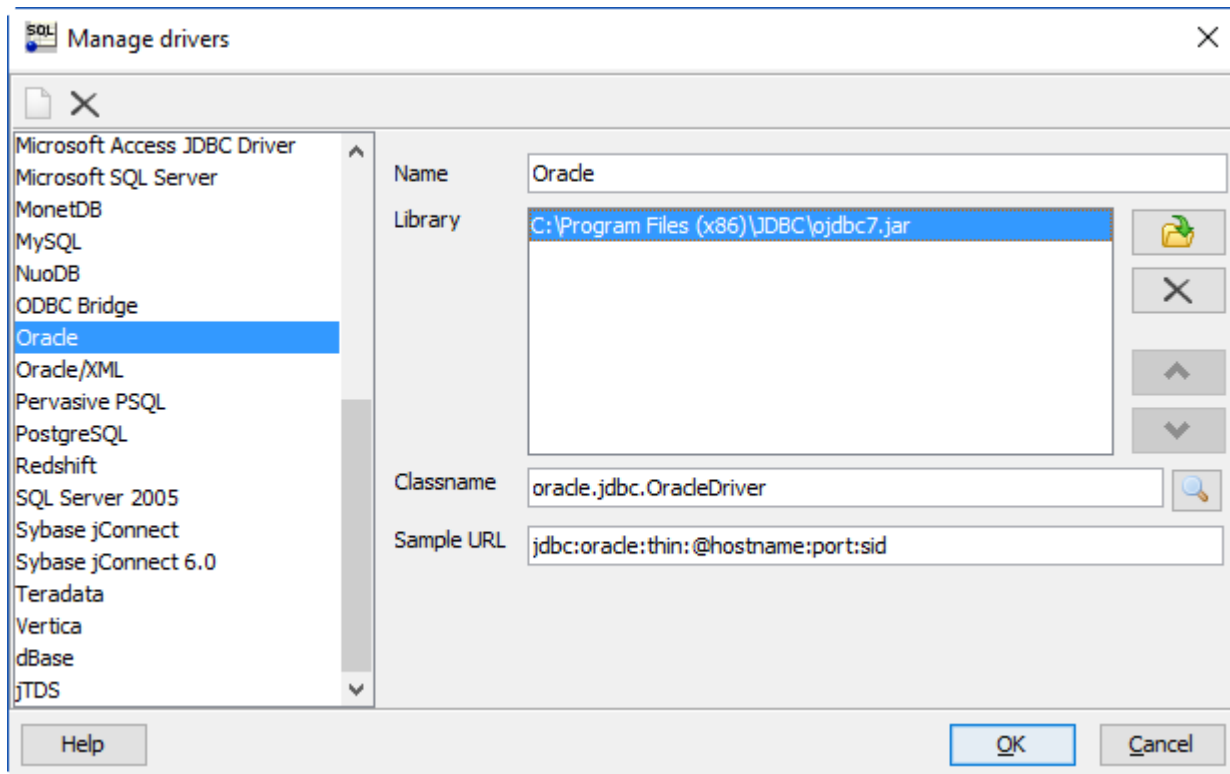
## Step 2: Install the SQL Tools and AWS Schema Conversion Tool on Your Local Computer

Next, you need to install a SQL client and AWS SCT on your local computer.

This walkthrough assumes you will use the SQL Workbench/J client to connect to the RDS instances for migration validation.

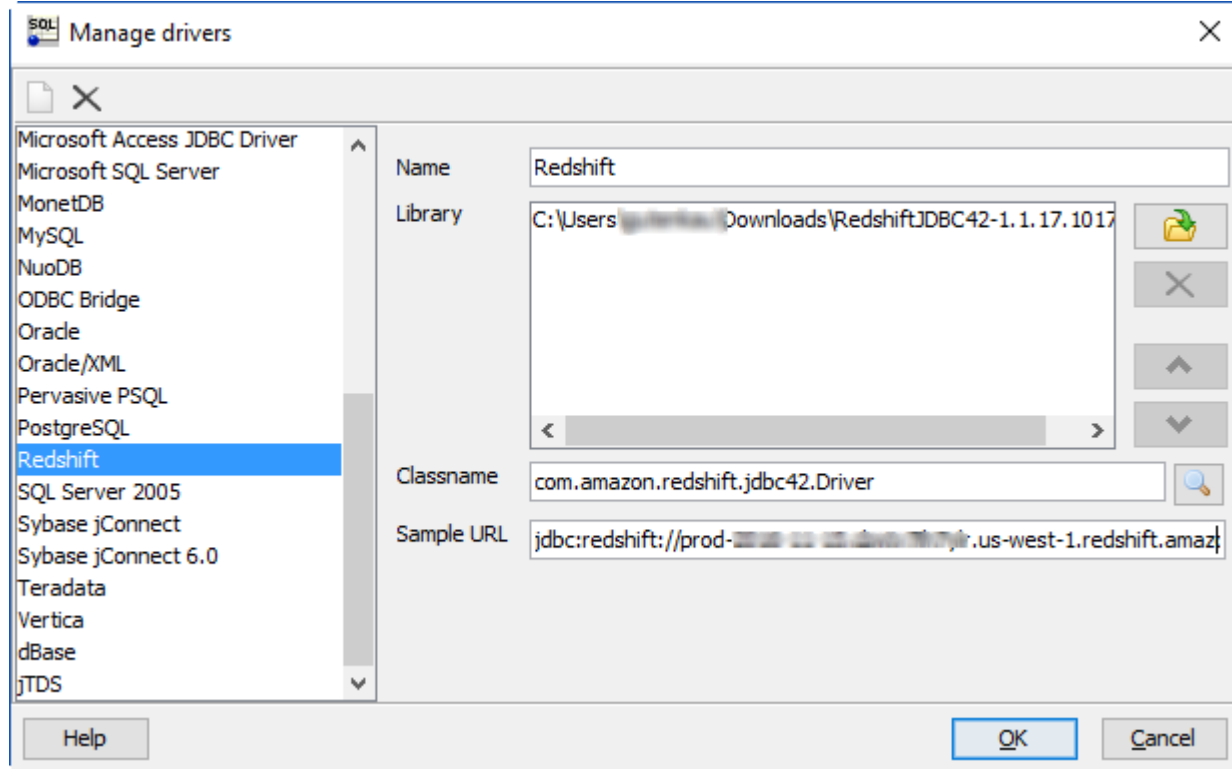
AWS Database Migration Service  
Step-by-Step Migration Guide  
Step 2: Install the SQL Tools and AWS Schema  
Conversion Tool on Your Local Computer

1. Download SQL Workbench/J from [the SQL Workbench/J website](#), and then install it on your local computer. This SQL client is free, open-source, and DBMS-independent.
2. Download the JDBC driver for your Oracle database release. For more information, go to <https://www.oracle.com/jdbc>.
3. Download the Amazon Redshift driver file, `RedshiftJDBC41-1.1.17.1017.jar`, as described following.
  - a. Find the Amazon S3 URL to the file in [Previous JDBC Driver Versions](#) of the *Amazon Redshift Cluster Management Guide*.
  - b. Download the driver as described in [Download the Amazon Redshift JDBC Driver](#) of the same guide.
4. Using SQL Workbench/J, configure JDBC drivers for Oracle and Amazon Redshift to set up connectivity, as described following.
  - a. In SQL Workbench/J, choose **File**, then choose **Manage Drivers**.
  - b. From the list of drivers, choose **Oracle**.
  - c. Choose the **Open** icon, then choose the `ojdbc7.jar` file that you downloaded in the previous step. Choose **OK**.



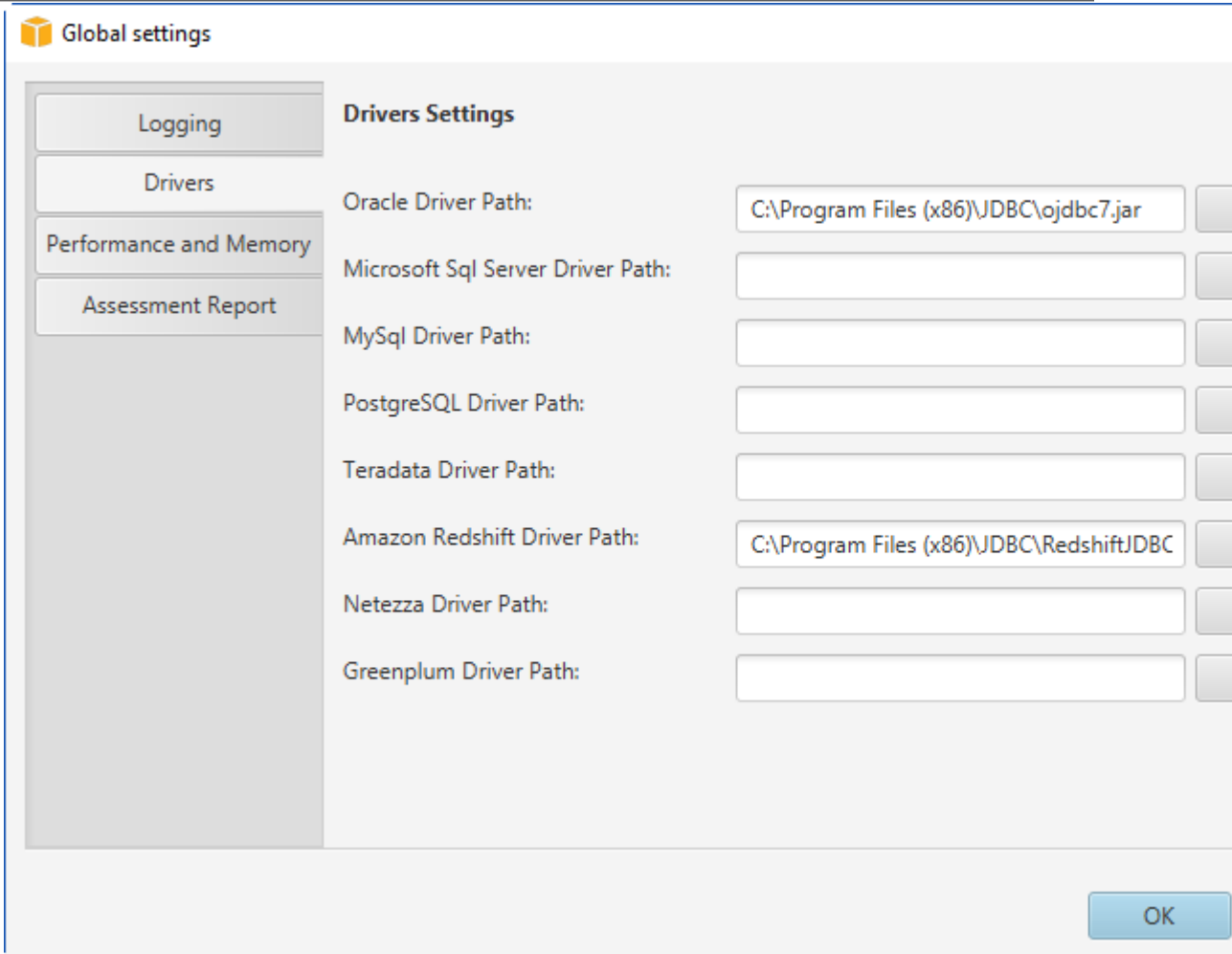
- d. From the list of drivers, choose **Redshift**.
- e. Choose the **Open** icon, then choose the Amazon Redshift JDBC driver that you downloaded in the previous step. Choose **OK**.

AWS Database Migration Service  
Step-by-Step Migration Guide  
Step 2: Install the SQL Tools and AWS Schema  
Conversion Tool on Your Local Computer



Next, install AWS SCT and the required JDBC drivers.

1. Download AWS SCT from [Installing and Updating the AWS Schema Conversion Tool](#) in the *AWS Schema Conversion Tool User Guide*.
2. Follow the instructions to install AWS SCT. By default, the tool is installed in the `C:\Program Files\AWS Schema Conversion Tool\AWS` directory.
3. Launch AWS SCT.
4. In AWS SCT, choose **Global Settings** from **Settings**.
5. Choose **Settings, Global Settings**, then choose **Drivers**, and then choose **Browse for Oracle Driver Path**. Locate the Oracle JDBC driver and choose **OK**.
6. Choose **Browse for Amazon Redshift Driver Path**. Locate the Amazon Redshift JDBC driver and choose **OK**. Choose **OK** to close the dialog box.



## Step 3: Test Connectivity to the Oracle DB Instance and Create the Sample Schema

After the CloudFormation stack has been created, test the connection to the Oracle DB instance by using SQL Workbench/J and then create the HR sample schema.

1. In SQL Workbench/J, choose **File**, then choose **Connect window**. Create a new connection profile using the following information.

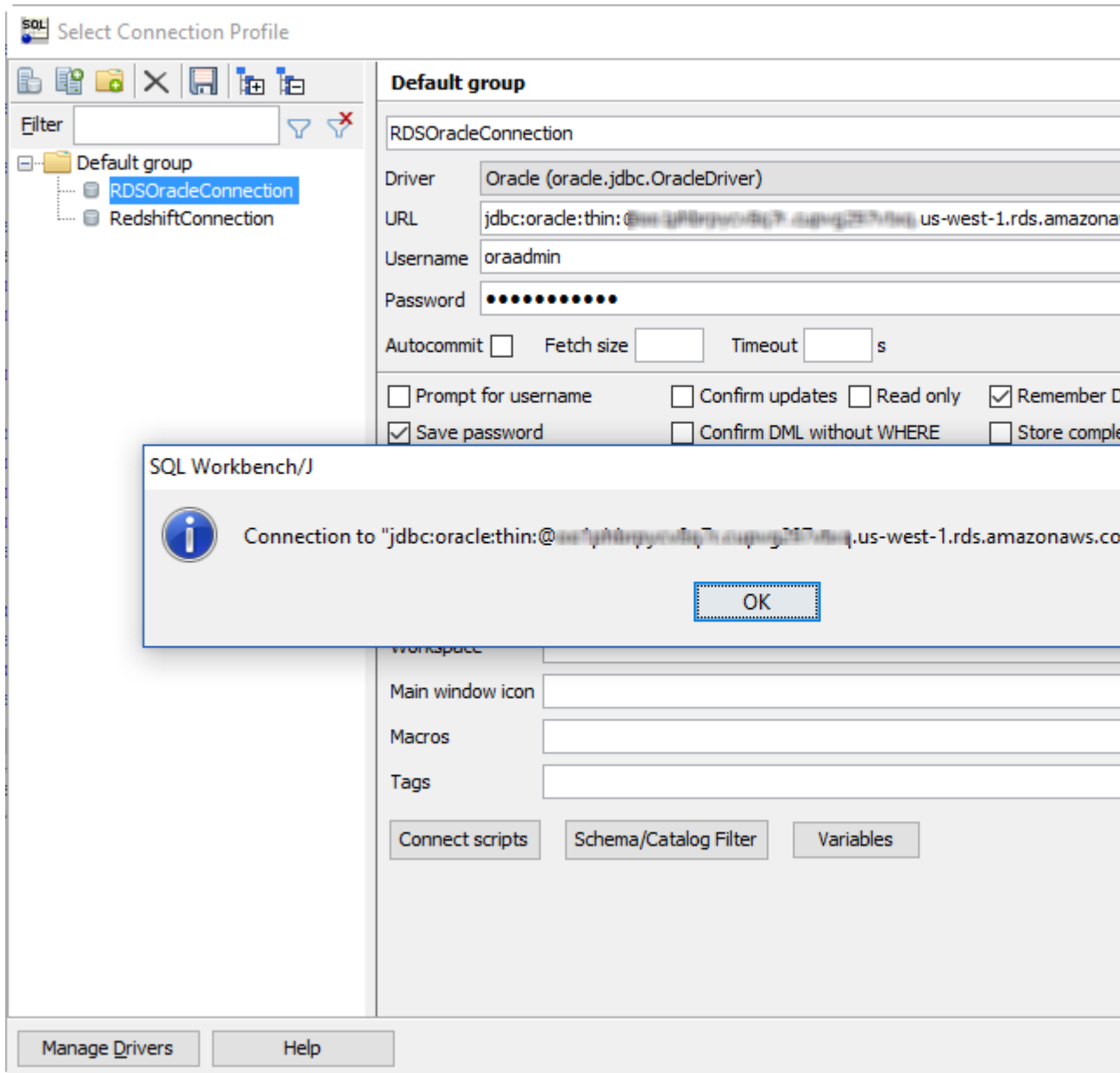
For This Parameter	Do This
<b>New profile name</b>	Type <code>RDSOracleConnection</code> .
<b>Driver</b>	Choose <code>Oracle</code> ( <code>oracle.jdbc.OracleDriver</code> ).
<b>URL</b>	Use the <b>OracleJDBCConnectionString</b> value you recorded when you examined the output details of the DMSdemo stack in a previous step.



AWS Database Migration Service  
 Step-by-Step Migration Guide  
 Step 3: Test Connectivity to the Oracle DB  
 Instance and Create the Sample Schema

For This Parameter	Do This
<b>Username</b>	Type oraadmin.
<b>Password</b>	Type oraadmin123.

- Test the connection by choosing **Test**. Choose **OK** to close the dialog box, then choose **OK** to create the connection profile.



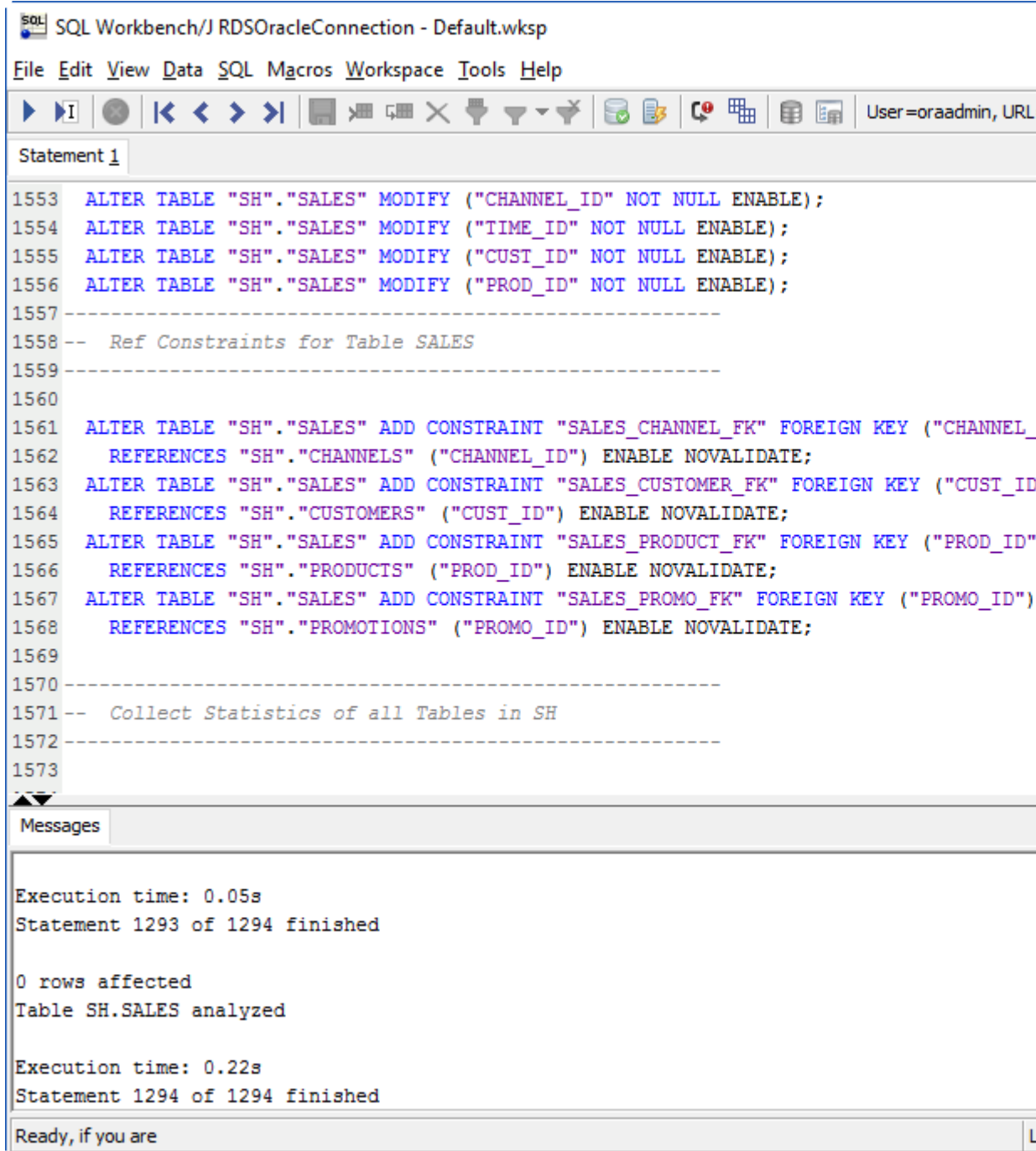
**Note**

If your connection is unsuccessful, ensure that the IP address you assigned when creating the CloudFormation template is the one you are attempting to connect from. This issue is the most common one when trying to connect to an instance.

- Create the SH schema you will use for migration using a custom SQL script (Oraclesalesstarschema.sql). To obtain this script, do the following:

AWS Database Migration Service  
Step-by-Step Migration Guide  
Step 3: Test Connectivity to the Oracle DB  
Instance and Create the Sample Schema

- Download the following archive to your computer: <http://docs.aws.amazon.com/dms/latest/sbs/samples/dms-sbs-RDSOracle2Redshift.zip>
- Extract the SQL script(Oraclesalesstarschema.sql) from the archive.
- Copy and paste the Oraclesalesstarschema.sql file into your current directory.
  - a. Open the SQL script in a text editor. Copy the entire script.
  - b. In SQL Workbench/J, paste the SQL script in the Default.wksp window showing **Statement 1**.
  - c. Choose **SQL**, then choose **Execute All**.



The screenshot shows the SQL Workbench/J interface. The main window displays the following SQL script for Statement 1:

```
1553 ALTER TABLE "SH"."SALES" MODIFY ("CHANNEL_ID" NOT NULL ENABLE);
1554 ALTER TABLE "SH"."SALES" MODIFY ("TIME_ID" NOT NULL ENABLE);
1555 ALTER TABLE "SH"."SALES" MODIFY ("CUST_ID" NOT NULL ENABLE);
1556 ALTER TABLE "SH"."SALES" MODIFY ("PROD_ID" NOT NULL ENABLE);
1557 -----
1558 -- Ref Constraints for Table SALES
1559 -----
1560
1561 ALTER TABLE "SH"."SALES" ADD CONSTRAINT "SALES_CHANNEL_FK" FOREIGN KEY ("CHANNEL_
1562 REFERENCES "SH"."CHANNELS" ("CHANNEL_ID") ENABLE NOVALIDATE;
1563 ALTER TABLE "SH"."SALES" ADD CONSTRAINT "SALES_CUSTOMER_FK" FOREIGN KEY ("CUST_ID
1564 REFERENCES "SH"."CUSTOMERS" ("CUST_ID") ENABLE NOVALIDATE;
1565 ALTER TABLE "SH"."SALES" ADD CONSTRAINT "SALES_PRODUCT_FK" FOREIGN KEY ("PROD_ID"
1566 REFERENCES "SH"."PRODUCTS" ("PROD_ID") ENABLE NOVALIDATE;
1567 ALTER TABLE "SH"."SALES" ADD CONSTRAINT "SALES_PROMO_FK" FOREIGN KEY ("PROMO_ID")
1568 REFERENCES "SH"."PROMOTIONS" ("PROMO_ID") ENABLE NOVALIDATE;
1569
1570 -----
1571 -- Collect Statistics of all Tables in SH
1572 -----
1573
```

The Messages pane at the bottom shows the execution results:

```
Execution time: 0.05s
Statement 1293 of 1294 finished

0 rows affected
Table SH.SALES analyzed

Execution time: 0.22s
Statement 1294 of 1294 finished

Ready, if you are
```

- Verify the object types and count in SH Schema were created successfully by running the following SQL query.

```
Select OBJECT_TYPE, COUNT(*) from dba_OBJECTS where owner='SH'  
GROUP BY OBJECT_TYPE;
```

The results of this query should be similar to the following.

OBJECT_TYPE	COUNT(*)
INDEX PARTITION	40
TABLE PARTITION	8
TABLE	5
INDEX	15

- Verify the total number of tables and number of rows for each table by running the following SQL query.

```
Select table_name, num_rows from dba_tables where owner='SH' order by 1;
```

The results of this query should be similar to the following.

TABLE_NAME	NUM_ROWS
CHANNELS	5
CUSTOMERS	8
PRODUCTS	66
PROMOTIONS	503
SALES	553

- Verify the integrity in tables. Check the number of sales made in different channels by running the following SQL query.

```
Select b.channel_desc,count(*) from SH.SALES a,SH.CHANNELS b where  
a.channel_id=b.channel_id  
group by b.channel_desc  
order by 1;
```

The results of this query should be similar to the following.

CHANNEL_DESC	COUNT(*)
Direct Sales	710
Internet	52
Partners	344

#### Note

The preceding examples are representative of validation queries. When you perform actual migrations, you should develop similar queries to validate the schema and the data integrity.

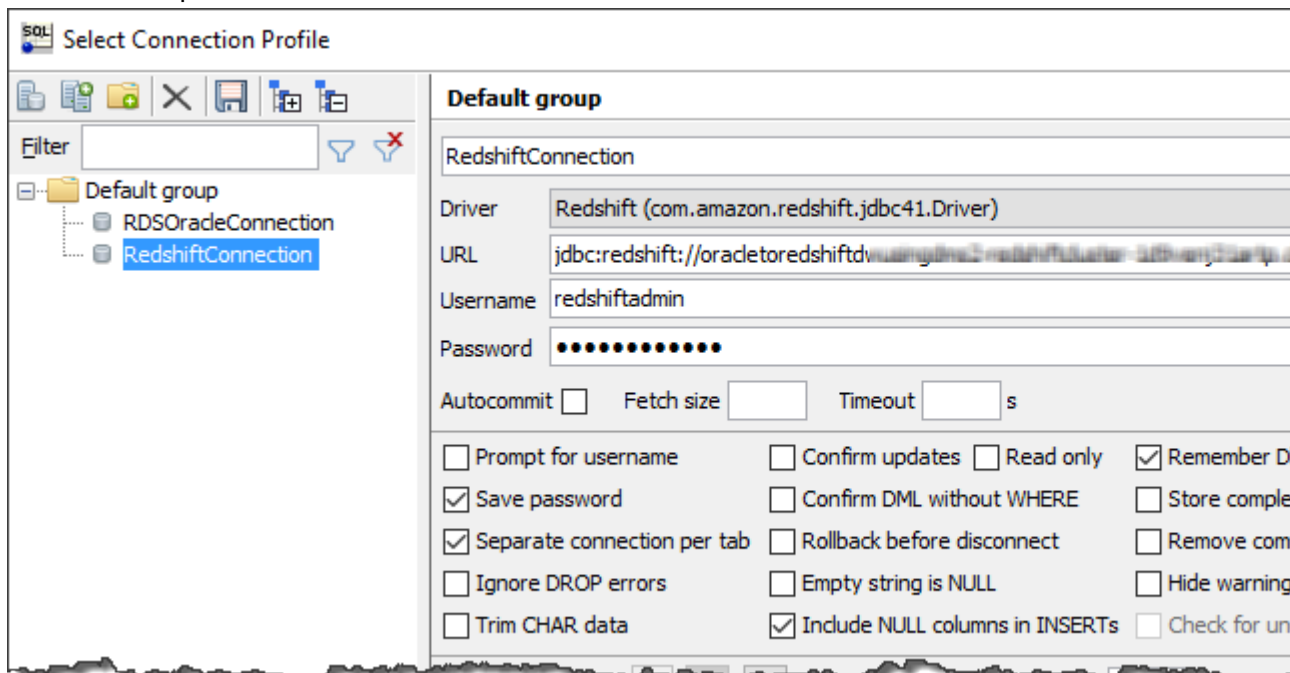
## Step 4: Test the Connectivity to the Amazon Redshift Database

Next, test your connection to your Amazon Redshift database.

1. In SQL Workbench/J, choose **File**, then choose **Connect window**. Choose the **Create a new connection profile** icon. Connect to the Amazon Redshift database in SQL Workbench/J by using the information shown following.

For This Parameter	Do This
<b>New profile name</b>	Type RedshiftConnection.
<b>Driver</b>	Choose Redshift (com.amazon.redshift.jdbc42.Driver).
<b>URL</b>	Use the <b>RedshiftJDBCConnectionString</b> value you recorded when you examined the output details of the DMSdemo stack in a previous step.
<b>Username</b>	Type redshiftadmin.
<b>Password</b>	Type Redshift#123.

2. Test the connection by choosing **Test**. Choose **OK** to close the dialog box, then choose **OK** to create the connection profile.



**Note**

If your connection is unsuccessful, ensure that the IP address you assigned when creating the CloudFormation template is the one you are attempting to connect from. This issue is the most common one when trying to connect to an instance.

3. Verify your connectivity to the Amazon Redshift DB instance by running a sample SQL command, such as `select current_date;`

## Step 5: Use AWS SCT to Convert the Oracle Schema to Amazon Redshift

Before you migrate data to Amazon Redshift, you convert the Oracle schema to an Amazon Redshift schema as described following.

1. Launch AWS SCT. In AWS SCT, choose **File**, then choose **New Project**. Create a new project called `DWSchemaMigrationDemoProject`. Enter the following information in the New Project window, and then choose **OK**.

For This Parameter	Do This
<b>Project Name</b>	Type <code>DWSchemaMigrationDemoProject</code> .
<b>Location</b>	Use the default <b>Projects</b> folder and the default <b>Data Warehouse (OLAP)</b> option.
<b>Source Database Engine</b>	Choose <b>Oracle DW</b> .
<b>Target Database Engine</b>	Choose <b>Amazon Redshift</b> .

**New Project**

Enter the name, location and type of the new migration project.

Project Name:

Location:

Transactional Database (OLTP)
  Data Warehouse (OLAP)

Source Database Engine:

Target Database Engine:

2. Choose **Connect to Oracle**. In the **Connect to Oracle** dialog box, enter the following information, and then choose **Test Connection**.

For This Parameter	Do This
<b>Type</b>	Choose <b>SID</b> .
<b>Server name</b>	Use the <code>OracleJDBCConnectionString</code> value you used to connect to the Oracle DB instance, but remove the JDBC prefix information and the port and database name suffix. For example, a sample connection string

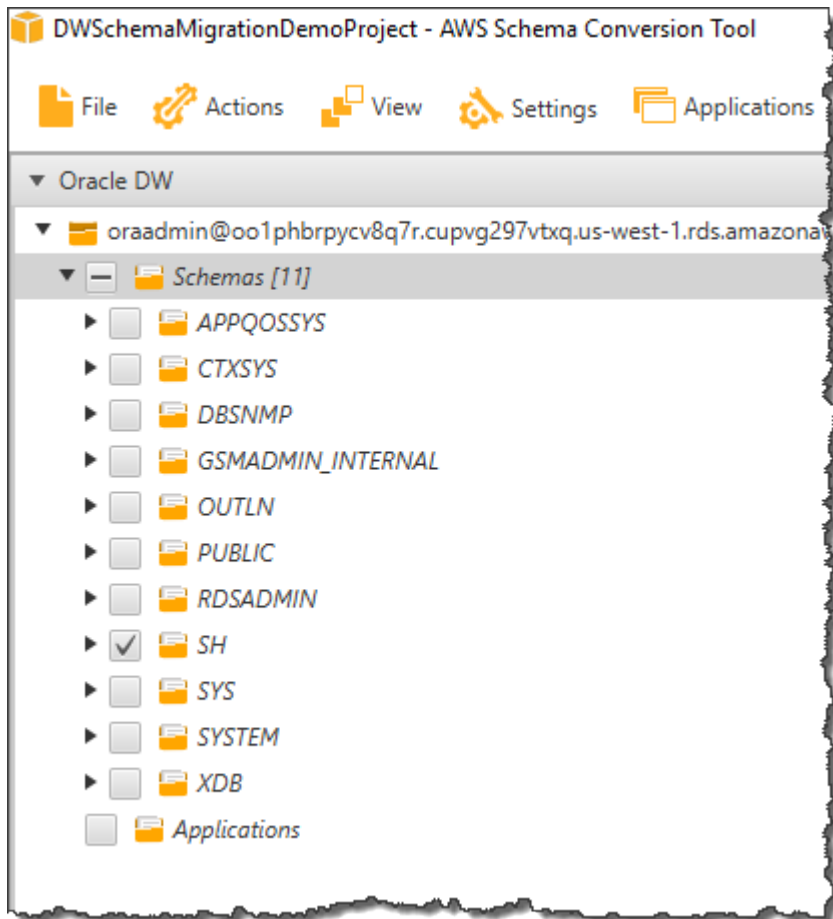
AWS Database Migration Service  
 Step-by-Step Migration Guide  
 Step 5: Use AWS SCT to Convert the  
 Oracle Schema to Amazon Redshift

For This Parameter	Do This
	you use with SQL Workbench/J might be "jdbc:oracle:thin:@abc12345678.cqi87654abc.us-west-2.rds.amazonaws.com:1521:ORCL". For the AWS SCT <b>Server name</b> , you remove "jdbc:oracle:thin:@" and ":1521:ORCL" and use just the server name: "abc12345678.cqi87654abc.us-west-2.rds.amazonaws.com".
<b>Server port</b>	Type 1521.
<b>Oracle SID</b>	Type ORCL.
<b>User name</b>	Type oraadmin.
<b>Password</b>	Type oraadmin123.

- Choose **OK** to close the alert box, then choose **OK** to close the dialog box and to start the connection to the Oracle DB instance. The database structure on the Oracle DB instance is shown following. Select only the SH schema.

**Note**

If the SH schema does not appear in the list, choose **Actions**, then choose **Refresh from Database**.



4. Choose **Connect to Amazon Redshift**. In the **Connect to Amazon Redshift** dialog box, enter the following information and then choose **Test Connection**.

For This Parameter	Do This
<b>Type</b>	Choose <b>SID</b> .
<b>Server name</b>	Use the <b>RedshiftJDBCConnectionString</b> value you used to connect to the Amazon Redshift cluster, but remove the JDBC prefix information and the port suffix. For example, a sample connection string you use with SQL Workbench/J might be "jdbc:redshift://oracletoredshiftdwusingdms-redshiftcluster-abc123567.abc87654321.us-west-2.redshift.amazonaws.com:5439/test". For the AWS SCT <b>Server name</b> , you remove "jdbc:redshift://" and ":5439/test" to use just the server name: "oracletoredshiftdwusingdms-redshiftcluster-abc123567.abc87654321.us-west-2.redshift.amazonaws.com"
<b>Server port</b>	Type 5439.
<b>User name</b>	Type redshiftadmin.
<b>Password</b>	Type Redshift#123.

AWS SCT analyzes the SH schema and creates a database migration assessment report for the conversion to Amazon Redshift.

5. Choose **OK** to close the alert box, then choose **OK** to close the dialog box to start the connection to the Amazon Redshift DB instance.
6. In the **Oracle DW** view, open the context (right-click) menu for the **SH** schema and select **Create Report**.
7. Review the report summary. To save the report, choose either **Save to CSV** or **Save to PDF**.

The report discusses the type of objects that can be converted by using AWS SCT, along with potential migration issues and actions to resolve these issues. For this walkthrough, you should see something like the following.

# Database Migration Assessment Report

Source Database: SH.oraadmin@oo1phbrpycv8q7r.cupvg297vtxq.us-west-1.rds.amazonaws.com:1521:ORCL  
Oracle Database 12c Enterprise Edition 12.1.0.2.0 (64bit Production)

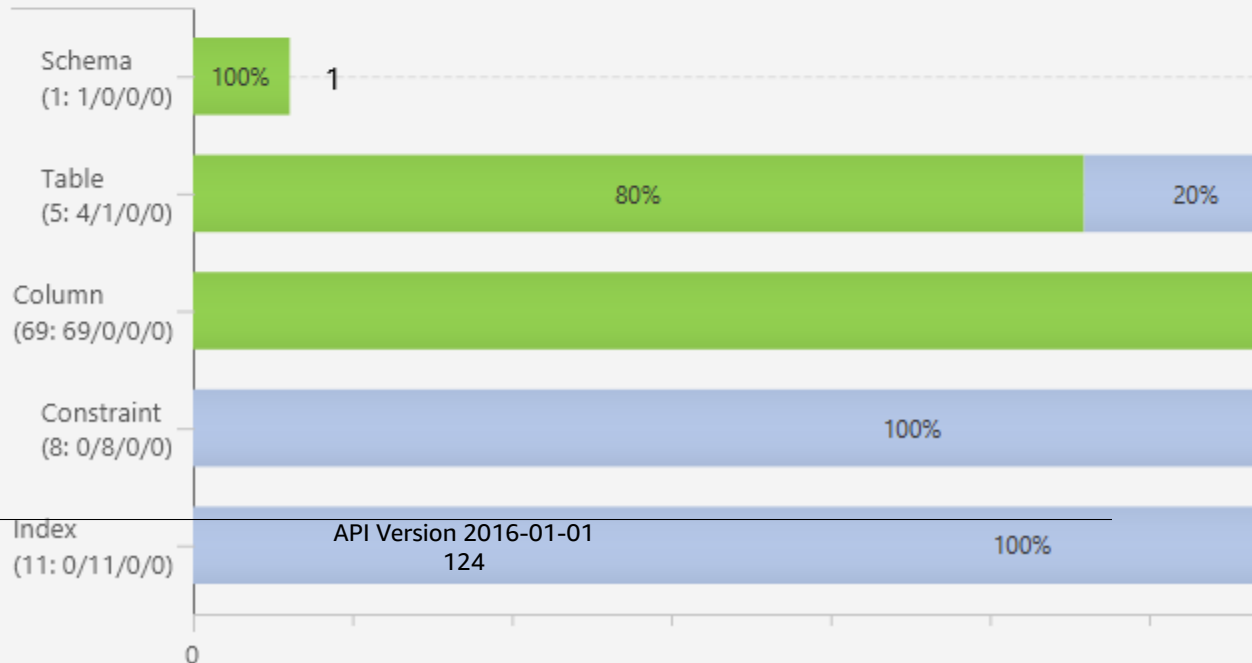
## Executive Summary

We completed the analysis of your Oracle DW source database and estimate that 100% of the database storage external-tables, constraints, indexes, sequences, synonyms, table types, public types, private types, user defined attributes, variables, constants, cursors, exceptions and other objects. Based on our analysis of SQL syntax elements ranging from simple tasks to medium-complexity actions to significant conversion actions.

## Database Objects with Conversion Actions for A

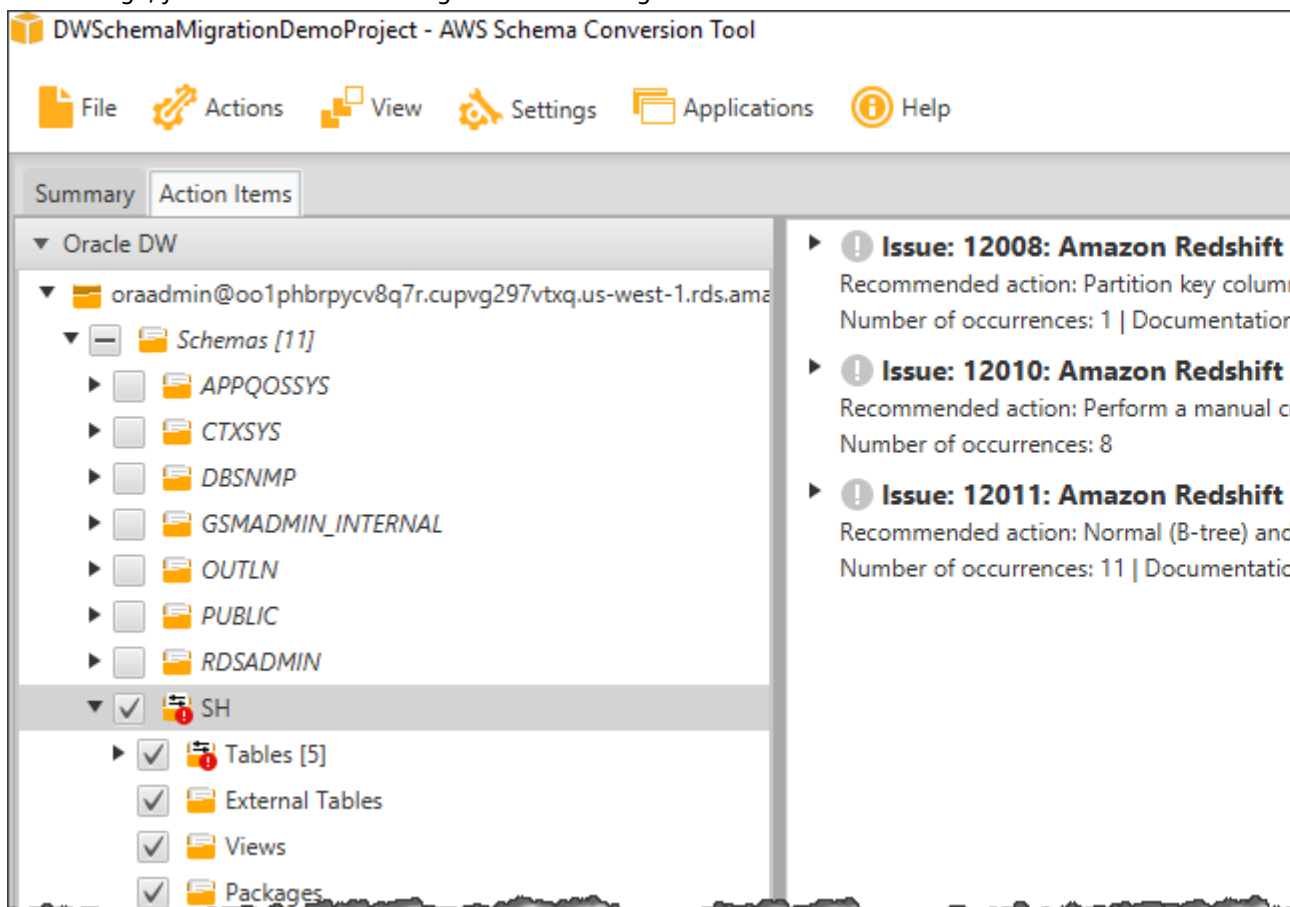
Of the total 94 database storage object(s) in the source database, we were able to identify 94 (100%) database

Figure: Conversion statistics for database storage objects

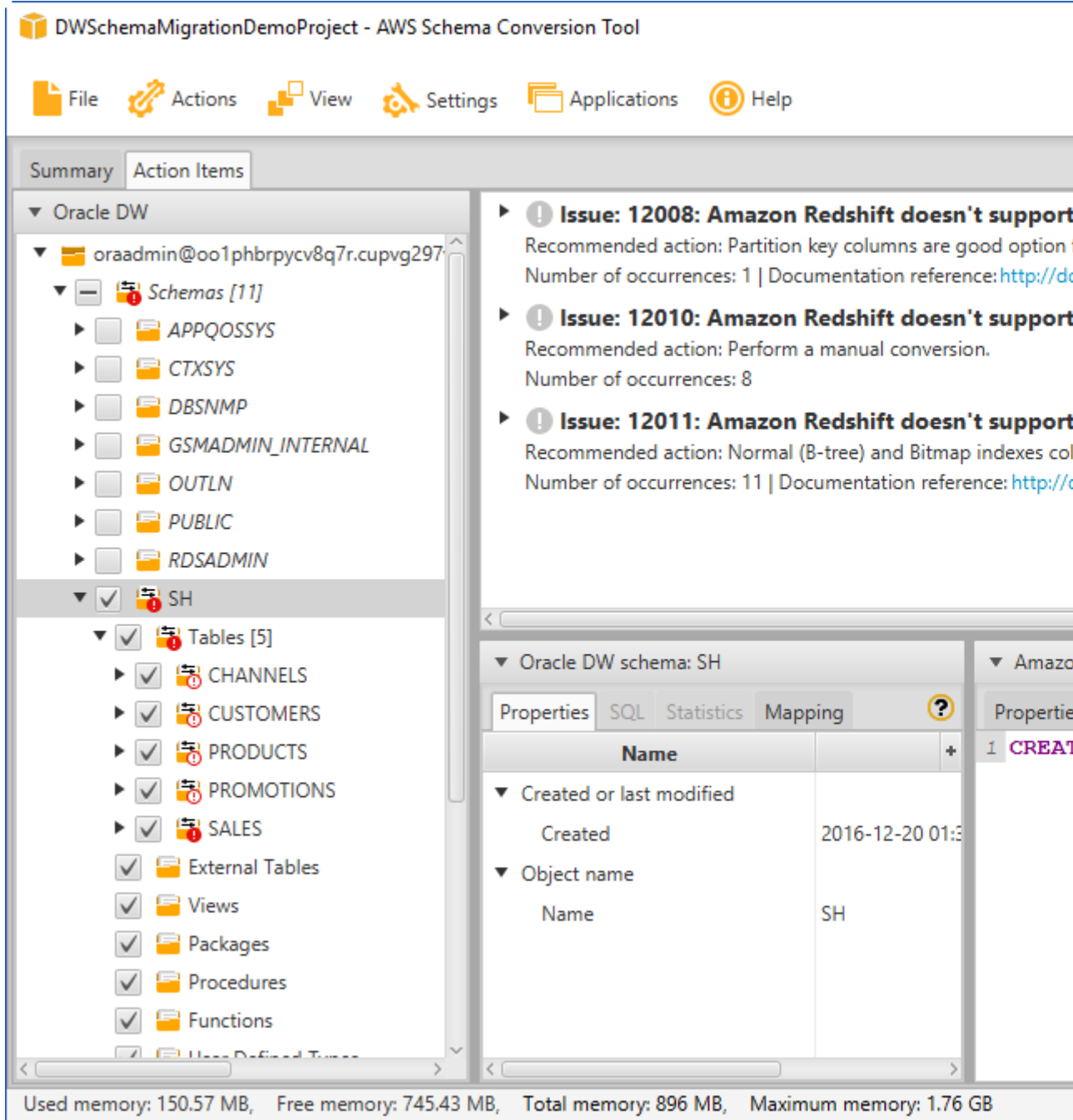




8. Choose the **Action Items** tab. The report discusses the type of objects that can be converted by using AWS SCT, along with potential migration issues and actions to resolve these issues. For this walkthrough, you should see something like the following.



9. Open the context (right-click) menu for the **SH** item in the **Schemas** list, and then choose **Collect Statistics**. AWS SCT analyzes the source data to recommend the best keys for the target Amazon Redshift database. For more information, see [Collecting or Uploading Statistics for the AWS Schema Conversion Tool](#).
10. Open the context (right-click) menu for the **SH** schema, and then choose **Convert schema**.
11. Choose **Yes** for the confirmation message. AWS SCT then converts your schema to the target database format.



**Note**

The choice of the Amazon Redshift sort keys and distribution keys is critical for optimal performance. You can use key management in AWS SCT to customize the choice of keys. For this walkthrough, we use the defaults recommended by AWS SCT. For more information, see [Optimizing Amazon Redshift by Using the AWS Schema Conversion Tool](#).

12 In the Amazon Redshift view, open the context (right-click) menu for the **SH** schema, and then choose **Apply to database** to apply the schema scripts to the target Amazon Redshift instance.

13 Open the context (right-click) menu for the **SH** schema, and then choose **Refresh from Database** to refresh from the target database.

The database schema has now been converted and imported from source to target.

## Step 6: Validate the Schema Conversion

To validate the schema conversion, you compare the objects found in the Oracle and Amazon Redshift databases using SQL Workbench/J.

1. In SQL Workbench/J, choose **File**, then choose **Connect window**. Choose the **RedshiftConnection** you created in an earlier step. Choose **OK**.
2. Run the following script to verify the number of object types and count in SH schema in the target Amazon Redshift database. These values should match the number of objects in the source Oracle database.

```
SELECT 'TABLE' AS OBJECT_TYPE,  
       TABLE_NAME AS OBJECT_NAME,  
       TABLE_SCHEMA AS OBJECT_SCHEMA  
FROM information_schema.TABLES  
WHERE TABLE_TYPE = 'BASE TABLE'  
AND OBJECT_SCHEMA = 'sh';
```

The output from this query should be similar to the following.

```
object_type | object_name | object_schema  
-----+-----+-----  
TABLE      | channels    | sh  
TABLE      | customers   | sh  
TABLE      | products    | sh  
TABLE      | promotions  | sh  
TABLE      | sales       | sh
```

3. Verify the sort and distributions keys that are created in the Amazon Redshift cluster by using the following query.

```
set search_path to '$user', 'public', 'sh';  
  
SELECT tablename,  
       "column",  
       TYPE,  
       encoding,  
       distkey,  
       sortkey,  
       "notnull"  
FROM pg_table_def  
WHERE (distkey = TRUE OR sortkey <> 0);
```

The results of the query reflect the distribution key (*distkey*) and sort key (*sortkey*) choices made by using AWS SCT key management.

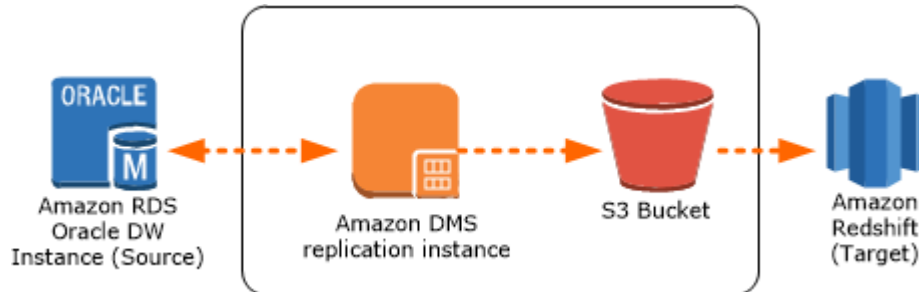
```
tablename | column          | type                | encoding | distkey |  
sortkey | notnull  
-----+-----+-----+-----+-----  
+-----+-----  
channels  | channel_id     | numeric(38,18)     | none    | true    |  
1 | true  
customers | cust_id        | numeric(38,18)     | none    | false   |  
4 | true  
customers | cust_gender    | character(2)       | none    | false   |  
1 | true
```

AWS Database Migration Service  
Step-by-Step Migration Guide  
Step 7: Create an AWS DMS Replication Instance

customers	cust_year_of_birth	smallint	none	false
3				true
customers	cust_marital_status	character varying(40)	none	false
2				false
products	prod_id	integer	none	true
4				true
products	prod_subcategory	character varying(100)	none	false
3				true
products	prod_category	character varying(100)	none	false
2				true
products	prod_status	character varying(40)	none	false
1				true
promotions	promo_id	integer	none	true
1				true
sales	prod_id	numeric(38,18)	none	false
4				true
sales	cust_id	numeric(38,18)	none	false
3				true
sales	time_id	timestamp without time zone	none	true
1				true
sales	channel_id	numeric(38,18)	none	false
2				true
sales	promo_id	numeric(38,18)	none	false
5				true

## Step 7: Create an AWS DMS Replication Instance

After we validate the schema structure between source and target databases, as described preceding, we proceed to the core part of this walkthrough, which is the data migration. The following illustration shows a high-level view of the migration process.



A DMS replication instance performs the actual data migration between source and target. The replication instance also caches the transaction logs during the migration. How much CPU and memory capacity a replication instance has influences the overall time required for the migration.

To create an AWS DMS replication instance, do the following:

1. Sign in to the AWS Management Console, open the AWS DMS console at <https://console.aws.amazon.com/dms/>, and choose **Create Migration**. If you are signed in as an AWS Identity and Access Management (IAM) user, you must have the appropriate permissions to access AWS DMS. For more information on the permissions required, see [IAM Permissions Needed to Use AWS DMS](#).
2. Choose **Create migration** to start a database migration.
3. On the **Welcome** page, choose **Next**.
4. On the **Create replication instance** page, specify your replication instance information as shown following.

For This Parameter	Do This
<b>Name</b>	Type <code>DMSdemo-repserver</code> .
<b>Description</b>	Type a brief description, such as <code>DMS demo replication server</code> .
<b>Instance class</b>	Choose <b>dms.t2.medium</b> . This instance class is large enough to migrate a small set of tables.
<b>VPC</b>	Choose <code>OracletoRedshiftusingDMS</code> , which is the VPC that was created by the CloudFormation stack.
<b>Multi-AZ</b>	Choose <code>No</code> .
<b>Publicly accessible</b>	Leave this item selected.

5. For the **Advanced** section, leave the default settings as they are, and choose **Next**.

## Step 8: Create AWS DMS Source and Target Endpoints

While your replication instance is being created, you can specify the source and target database endpoints using the AWS Management Console. However, you can only test connectivity after the replication instance has been created, because the replication instance is used in the connection.

1. Specify your connection information for the source Oracle database and the target Amazon Redshift database. The following table describes the source settings.

For This Parameter	Do This
<b>Endpoint Identifier</b>	Type <code>Orasource</code> (the Amazon RDS Oracle endpoint).
<b>Source Engine</b>	Choose <b>oracle</b> .
<b>Server name</b>	Provide the Oracle DB instance name. This name is the <b>Server name</b> value that you used for AWS SCT, such as <code>"abc123567.abc87654321.us-west-2.rds.amazonaws.com"</code> .
<b>Port</b>	Type <code>1521</code> .
<b>SSL mode</b>	Choose <b>None</b> .
<b>Username</b>	Type <code>oraadmin</code> .
<b>Password</b>	Type <code>oraadmin123</code> .
<b>SID</b>	Type <code>ORCL</code> .

The following table describes the target settings.

For This Parameter	Do This
Endpoint Identifier	Type <code>`Redshifttarget`</code> (the Amazon Redshift endpoint).
Target Engine	Choose <b>redshift</b> .
Servername	Provide the Amazon Redshift DB instance name. This name is the <b>Server name</b> value that you used for AWS SCT, such as "oracleredshiftdwusingdms-redshiftcluster-abc123567.abc87654321.us-west-2.redshift.amazonaws.com"..
Port	Type 5439.
SSL mode	Choose <b>None</b> .
Username	Type redshiftadmin.
Password	Type Redshift#123.
Database name	Type test.

The completed page should look like the following.

✔ Replication instance created successfully.

Your database endpoint can be on-premise, in EC2, RDS or in the cloud. Define the connection details for the source and target endpoints. You can test your endpoint connections here to avoid errors later.

### Source database connection details

Endpoint identifier*	<input type="text" value="Orasource"/>	
Source engine*	<input type="text" value="oracle"/>	
Server name*	<input type="text" value="661pR8dpyCwDq7V.CuPwG297v"/>	
Port*	<input type="text" value="1521"/>	
SSL mode*	<input type="text" value="none"/>	
User name*	<input type="text" value="oraadmin"/>	
Password*	<input type="password" value="....."/>	
SID*	<input type="text" value="ORCL"/>	

▶ Advanced

### Target database connection details

Endpoint identifier*	<input type="text" value="Redsh"/>	
Target engine*	<input type="text" value="redshif"/>	
Server name*	<input type="text" value="oraclet"/>	
Port*	<input type="text" value="5439"/>	
SSL mode*	<input type="text" value="none"/>	
User name*	<input type="text" value="redshif"/>	
Password*	<input type="password" value="....."/>	
Database name*	<input type="text" value="test"/>	

▶ Advanced

2. Wait for the status to say **Replication instance created successfully..**
3. To test the source and target connections, choose **Run Test** for the source and target connections.
4. Choose **Next**.

## Step 9: Create and Run Your AWS DMS Migration Task

Using an AWS DMS task, you can specify what schema to migrate and the type of migration. You can migrate existing data, migrate existing data and replicate ongoing changes, or replicate data changes only. This walkthrough migrates existing data only.

1. On the **Create Task** page, specify the task options. The following table describes the settings.

For This Parameter	Do This
<b>Task name</b>	Type <code>migrateSHschema</code> .
<b>Replication instance</b>	Shows <code>DMSdemo-repserver</code> (the AWS DMS replication instance created in an earlier step).
<b>Source endpoint</b>	Shows <code>orasource</code> (the Amazon RDS for Oracle endpoint).
<b>Target endpoint</b>	Shows <code>redshifttarget</code> (the Amazon Redshift endpoint).
<b>Migration type</b>	Choose the option <b>Migrate existing data</b> .
<b>Start task on create</b>	Choose this option.

The page should look like the following.



## Create task

A task can contain one or more table mappings which define what data is moved from the source to the target automatically.

**Task name\***

migrateSHschema

**Replication instance\***

dmsdemo-repserver - vpc-809665e4

**Source endpoint\***

orasource

**Target endpoint\***

redshifftarget

**Migration type\***

Migrate existing data

**Start task on create**



2. On the **Task Settings** section, specify the settings as shown in the following table.

For This Parameter	Do This
Target table preparation mode	Choose <b>Do nothing</b> .
Include LOB columns in replication	Choose <b>Limited LOB mode</b> .
Max LOB size (kb)	Accept the default (32).

The section should look like the following.

▼ Task Settings

**Target table preparation mode\***  Do nothing  
 Drop tables on target  
 Truncate

**Include LOB columns in replication\***  Don't include LOB columns  
 Limited LOB mode

**Max LOB size (kb)\***

**Enable logging**

[Advanced Settings](#)

3. In the **Selection rules** section, specify the settings as shown in the following table.

For This Parameter	Do This
Schema name is	Choose <code>Enter a schema</code> .
Schema name is like	Type <code>SH%</code> .
Table name is like	Type <code>%</code> .
Action	Choose <code>Include</code> .

The section should look like the following:

▼ Table mappings

**Guided** JSON

**Selection rules** ⓘ

At least one selection rule with an include action is required. Once you have one or more selection rules, you can add filters to the rules.

**Where** ⓘ

Schema name is

Schema name is like

Table name is like

Use % as a wildcard.

**Action**

**Filter** ⓘ

[Add column filter](#)

4. Choose **Add selection rule**.
5. Choose **Create task**. The task begins immediately. The **Tasks** section shows you the status of the migration task.

## DMS

Dashboard

Get started

**Tasks**

Endpoints

Certificates

Replication instances

Subnet groups

Create task

Modify

Start/Resume

Stop

Filter:

<input type="checkbox"/>	ID	Status	Source
<input type="checkbox"/>	migrateshschema2	Creating	orasou
<input checked="" type="checkbox"/>	migrateshschema	Ready	orasou

## Step 10: Verify That Your Data Migration Completed Successfully

When the migration task completes, you can compare your task results with the expected results.

1. On the navigation pane, choose **Tasks**.
2. Choose your migration task (`migrateshschema`).
3. Choose the **Table statistics** tab, shown following.

**Create task**   **Modify**   **Start/Resume**   **Stop**   **Delete**

Filter:  ✕

<input type="checkbox"/>	ID	Status	Source	Target	Type
<input checked="" type="checkbox"/>	migrateshschema	Modifying	orasource	redshifttarget	Full

## migrateshschema

**Overview**   **Task monitoring**   **Table statistics**   **Logs**

Filter:  ✕

Schema	Table	State	Inserts	Deletes	Updates	DDL
SH	CHANNELS	Table completed	0	0	0	0
SH	CUSTOMERS	Table completed	0	0	0	0
SH	PRODUCTS	Table completed	0	0	0	0
SH	PROMOTIONS	Table completed	0	0	0	0
SH	SALES	Table completed	0	0	0	0

4. Connect to the Amazon Redshift instance by using SQL Workbench/J, and then check whether the database tables were successfully migrated from Oracle to Amazon Redshift by running the SQL script shown following.

```
select "table", tbl_rows
from svv_table_info
where
SCHEMA = 'sh'
```

```
order by 1;
```

Your results should look similar to the following.

table	tbl_rows
channels	5
customers	8
products	66
promotions	503
sales	1106

5. To verify whether the output for tables and number of rows from the preceding query matches what is expected for RDS Oracle, compare your results with those in previous steps.
6. Run the following query to check the relationship in tables; this query checks the departments with employees greater than 10.

```
Select b.channel_desc,count(*) from SH.SALES a,SH.CHANNELS b where  
a.channel_id=b.channel_id  
group by b.channel_desc  
order by 1;
```

The output from this query should be similar to the following.

channel_desc	count
Direct Sales	355
Internet	26
Partners	172

7. Verify column compression encoding.

DMS uses an Amazon Redshift COPY operation to load data. By default, the COPY command applies automatic compression whenever loading to an empty target table. The sample data for this walkthrough is not large enough for automatic compression to be applied. When you migrate larger data sets, COPY will apply automatic compression.

For more details about automatic compression on Amazon Redshift tables, see [Loading Tables with Automatic Compression](#).

To view compression encodings, run the following query.

```
SELECT *  
FROM pg_table_def  
WHERE schemaname = 'sh';
```

Now you have successfully completed a database migration from an Amazon RDS for Oracle DB instance to Amazon Redshift.

## Step 11: Delete Walkthrough Resources

After you have completed this walkthrough, perform the following steps to avoid being charged further for AWS resources used in the walkthrough. It's necessary that you do the steps in order, because some resources cannot be deleted if they have a dependency upon another resource.

To delete AWS DMS resources, do the following:

1. On the navigation pane, choose **Tasks**, choose your migration task (`migratehrschema`), and then choose **Delete**.
2. On the navigation pane, choose **Endpoints**, choose the Oracle source endpoint (`orasource`), and then choose **Delete**.
3. Choose the Amazon Redshift target endpoint (`redshifttarget`), and then choose **Delete**.
4. On the navigation pane, choose **Replication instances**, choose the replication instance (`DMSdemo-repserver`), and then choose **Delete**.

Next, you must delete your AWS CloudFormation stack, `DMSdemo`. Do the following:

1. Sign in to the AWS Management Console and open the AWS CloudFormation console at <https://console.aws.amazon.com/cloudformation>.

If you are signed in as an IAM user, you must have the appropriate permissions to access AWS CloudFormation.

2. Choose your CloudFormation stack, `OracletoRedshiftDWusingDMS`.
3. For **Actions**, choose **Delete stack**.

The status of the stack changes to `DELETE_IN_PROGRESS` while AWS CloudFormation cleans up the resources associated with the `OracletoRedshiftDWusingDMS` stack. When AWS CloudFormation is finished cleaning up resources, it removes the stack from the list.

## Next Steps

You can explore several other features of AWS DMS that were not included in this walkthrough, including the following:

- The AWS DMS change data capture (CDC) feature, for ongoing replication of data.
- Transformation actions that let you specify and apply transformations to the selected schema or table as part of the migration process.

For more information, see [the AWS DMS documentation](#).

# Migrating MySQL-Compatible Databases to AWS

Amazon Web Services (AWS) has several services that allow you to run a MySQL-compatible database on AWS. Amazon Relational Database Service (Amazon RDS) supports MySQL-compatible databases including MySQL, MariaDB, and Amazon Aurora MySQL. AWS Elastic Cloud Computing Service (EC2) provides platforms for running MySQL-compatible databases.

Migrating From	Solution
An RDS MySQL DB instance	You can migrate data directly from an Amazon RDS MySQL DB snapshot to an Amazon Aurora MySQL DB cluster. For details, see <a href="#">Migrating Data from an Amazon RDS MySQL DB Instance to an Amazon Aurora MySQL DB Cluster (p. 150)</a> .
A MySQL database external to Amazon RDS	<p>If your database supports the InnoDB or MyISAM tablespaces, you have these options for migrating your data to an Amazon Aurora MySQL DB cluster:</p> <ul style="list-style-type: none"><li>* You can create a dump of your data using the <code>mysqldump</code> utility, and then import that data into an existing Amazon Aurora MySQL DB cluster.</li><li>* You can copy the source files from your database to an Amazon Simple Storage Service (Amazon S3) bucket, and then restore an Amazon Aurora MySQL DB cluster from those files. This option can be considerably faster than migrating data using <code>mysqldump</code>.</li></ul> <p>For details, see <a href="#">Migrating MySQL to Amazon Aurora MySQL by Using mysqldump (p. 150)</a>.</p> <p>However, for very large databases, you can significantly reduce the amount of time that it takes to migrate your data by copying the source files for your database and restoring those files to an Amazon Aurora MySQL DB instance as described in <a href="#">Migrating Data from an External MySQL Database to an Amazon Aurora MySQL Using Amazon S3 (p. 141)</a>.</p>
A database that is not MySQL-compatible	You can also use AWS Database Migration Service (AWS DMS) to migrate data from a not MySQL-compatible database. For more information on AWS DMS, see <a href="#">What Is AWS Database Migration Service?</a>



# Migrating a MySQL-Compatible Database to Amazon Aurora MySQL

If your database supports the InnoDB or MyISAM tablespaces, you have these options for migrating your data to an Amazon Aurora MySQL DB cluster:

- You can create a dump of your data using the `mysqldump` utility, and then import that data into an existing Amazon Aurora MySQL DB cluster. For more information, see [Migrating MySQL to Amazon Aurora MySQL by Using `mysqldump`](#) (p. 150).
- You can copy the source files from your database to an Amazon S3 bucket, and then restore an Amazon Aurora MySQL DB cluster from those files. This option can be considerably faster than migrating data using `mysqldump`. For more information, see [Migrating Data from an External MySQL Database to an Amazon Aurora MySQL Using Amazon S3](#) (p. 141).

## Migrating Data from an External MySQL Database to an Amazon Aurora MySQL Using Amazon S3

You can copy the source files from your source MySQL version 5.5, 5.6, or 5.7 database to an Amazon S3 bucket, and then restore an Amazon Aurora MySQL DB cluster from those files.

This option can be considerably faster than migrating data using `mysqldump`, because using `mysqldump` replays all of the commands to recreate the schema and data from your source database in your new Amazon Aurora MySQL DB cluster. By copying your source MySQL data files, Amazon Aurora MySQL can immediately use those files as the data for DB cluster.

### Note

Restoring an Amazon Aurora MySQL DB cluster from backup files in an Amazon S3 bucket is not supported for the Asia Pacific (Mumbai) region.

Amazon Aurora MySQL does not restore everything from your database. You should save the database schema and values for the following items from your source MySQL or MariaDB database and add them to your restored Amazon Aurora MySQL DB cluster after it has been created.

- User accounts
- Functions
- Stored procedures
- Time zone information. Time zone information is loaded from the local operating system of your Amazon Aurora MySQL DB cluster.

## Prerequisites

Before you can copy your data to an Amazon S3 bucket and restore a DB cluster from those files, you must do the following:

- Install Percona XtraBackup on your local server.
- Permit Amazon Aurora MySQL to access your Amazon S3 bucket on your behalf.

## Installing Percona XtraBackup

Amazon Aurora MySQL can restore a DB cluster from files that were created using Percona XtraBackup. You can install Percona XtraBackup from the Percona website at <https://www.percona.com/doc/percona-xtrabackup/2.4/installation>.

## Required Permissions

To migrate your MySQL data to an Amazon Aurora MySQL DB cluster, several permissions are required:

- The user that is requesting that Amazon RDS create a new cluster from an Amazon S3 bucket must have permission to list the buckets for your AWS account. You grant the user this permission using an AWS Identity and Access Management (IAM) policy.
- Amazon RDS requires permission to act on your behalf to access the Amazon S3 bucket where you store the files used to create your Amazon Aurora MySQL DB cluster. You grant Amazon RDS the required permissions using an IAM service role.
- The user making the request must also have permission to list the IAM roles for your AWS account.
- If the user making the request will create the IAM service role, or will request that Amazon RDS create the IAM service role (by using the console), then the user must have permission to create an IAM role for your AWS account.

For example, the following IAM policy grants a user the minimum required permissions to use the console to both list IAM roles, create an IAM role, and list the S3 buckets for your account.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iam:ListRoles",
        "iam:CreateRole",
        "iam:CreatePolicy",
        "iam:AttachRolePolicy",
        "s3:ListBucket",
        "s3:ListObjects"
      ],
      "Resource": "*"
    }
  ]
}
```

Additionally, for a user to associate an IAM role with an S3 bucket, the IAM user must have the `iam:PassRole` permission for that IAM role. This permission allows an administrator to restrict which IAM roles a user can associate with S3 buckets.

For example, the following IAM policy allows a user to associate the role named `S3Access` with an S3 bucket.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowS3AccessRole",
      "Effect": "Allow",
      "Action": "iam:PassRole",
      "Resource": "arn:aws:iam::123456789012:role/S3Access"
    }
  ]
}
```

}

## Creating the IAM Service Role

You can have the Amazon RDS Management Console create a role for you by choosing the **Create a New Role** option (shown later in this topic). If you select this option and specify a name for the new role, then Amazon RDS will create the IAM service role required for Amazon RDS to access your S3 bucket with the name that you supply.

As an alternative, you can manually create the role using the following procedure.

1. Sign in to the AWS Management Console and open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the left navigation pane, choose **Roles**.
3. Choose **Create New Role**, specify a value for **Role Name** for the new role, and then choose **Next Step**.
4. Under **AWS Service Roles**, find **Amazon RDS** and choose **Select**.
5. Do not select a policy to attach in the **Attach Policy** step. Instead, choose **Next Step**.
6. Review your role information, and then choose **Create Role**.
7. In the list of roles, choose the name of your newly created role. Choose the **Permissions** tab.
8. Choose **Inline Policies**. Because your new role has no policy attached, you will be prompted to create one. Click the link to create a new policy.
9. On the **Set Permissions** page, choose **Custom Policy** and then choose **Select**.
10. Type a **Policy Name** such as `S3-bucket-policy`. Add the following code for **Policy Document**, replacing `<bucket name>` with the name of the S3 bucket that you are allowing access to.

As part of the policy document, you can also include a file name prefix. If you specify a prefix, then Amazon Aurora MySQL will create the DB cluster using the files in the S3 bucket that begin with the specified prefix. If you don't specify a prefix, then Amazon Aurora MySQL will create the DB cluster using all of the files in the S3 bucket.

To specify a prefix, replace `<prefix>` following with the prefix of your file names. Include the asterisk (\*) after the prefix. If you don't want to specify a prefix, specify only an asterisk.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:ListBucket",
        "s3:GetBucketLocation"
      ],
      "Resource": [
        "arn:aws:s3:::<bucket name>"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject"
      ],
      "Resource": [
        "arn:aws:s3:::<bucket name>/<prefix>*"
      ]
    }
  ]
}
```

11 Choose **Apply Policy**.

## Step 1: Backing Up Files to be Restored as a DB Cluster

To create a backup of your MySQL database files that can be restored from S3 to create an Amazon Aurora MySQL DB cluster, use the Percona Xtrabackup utility (`innobackupex`) to back up your database.

For example, the following command creates a backup of a MySQL database and stores the files in the `/s3-restore/backup` folder.

```
innobackupex --user=myuser --password=<password> --no-timestamp /s3-restore/backup
```

If you want to compress your backup into a single file (which can be split, if needed), you can use the `--stream` option to save your backup in one of the following formats:

- Gzip (.gz)
- tar (.tar)
- Percona xstream (.xstream)

For example, the following command creates a backup of your MySQL database split into multiple Gzip files. The parameter values shown are for a small test database; for your scenario, you should determine the parameter values needed.

```
innobackupex --user=myuser --password=<password> --stream=tar \  
/mydata/s3-restore/backup | split -d --bytes=512000 \  
- /mydata/s3-restore/backup3/backup.tar.gz
```

For example, the following command creates a backup of your MySQL database split into multiple tar files.

```
innobackupex --user=myuser --password=<password> --stream=tar \  
/mydata/s3-restore/backup | split -d --bytes=512000 \  
- /mydata/s3-restore/backup3/backup.tar
```

For example, the following command creates a backup of your MySQL database split into multiple xstream files.

```
innobackupex --stream=xstream \  
/mydata/s3-restore/backup | split -d --bytes=512000 \  
- /mydata/s3-restore/backup/backup.xstream
```

Amazon S3 limits the size of a file uploaded to a bucket to 5 terabytes (TB). If the backup data for your database exceeds 5 TB, then you must use the `split` command to split the backup files into multiple files that are each less than 5 TB.

Amazon Aurora MySQL does not support partial backups created using Percona Xtrabackup. You cannot use the `--include`, `--tables-file`, or `--databases` options to create a partial backup when you backup the source files for your database.

For more information, see the [The innobackupex Script](#).

Amazon Aurora MySQL consumes your backup files based on the file name. Be sure to name your backup files with the appropriate file extension based on the file format—for example, `#0#xstream` for files stored using the Percona xstream format.

Amazon Aurora MySQL consumes your backup files in alphabetical order as well as natural number order. Always use the `split` option when you issue the `innobackupex` command to ensure that your backup files are written and named in the proper order.

## Step 2: Copying Files to an Amazon S3 Bucket

Once you have backed up your MySQL database using the Percona Xtrabackup utility, then you can copy your backup files to an Amazon S3 bucket.

For information on creating and uploading a file to an Amazon S3 bucket, see [Getting Started with Amazon Simple Storage Service](#) in the *Amazon S3 Getting Started Guide*.

## Step 3: Restoring an Aurora MySQL DB Cluster from an Amazon S3 Bucket

You can restore your backup files from your Amazon S3 bucket to a create new Amazon Aurora MySQL DB cluster by using the Amazon RDS console.

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. In the RDS Dashboard, choose **Restore Aurora MySQL DB Cluster from S3**.
3. In the **Create database by restoring from S3** page, specify the following settings in the following sections:
  - a. In the **S3 Destination** section, specify the following:

For This Option	Do This
<b>S3 Bucket</b>	Select the Amazon S3 bucket where your backup files are stored.
<b>S3 Prefix (Optional)</b>	<p>Specify a file path prefix for the files stored in your Amazon S3 bucket. The <b>S3 Bucket Prefix</b> is optional. If you don't specify a prefix, then Amazon Aurora MySQL will create the DB cluster using all of the files in the root folder of the S3 bucket. If you specify a prefix, then Amazon Aurora MySQL will create the DB cluster using the files in the S3 bucket where the full path for the file begins with the specified prefix.</p> <p>Amazon Aurora MySQL does not traverse subfolders in your S3 bucket looking for backup files. Only the files from the folder identified by the <b>S3 Bucket Prefix</b> are used. If you store your backup files in a subfolder in your S3 bucket, then you must specify a prefix that identifies the full path to the folder where the files are stored.</p> <p>For example, if you store your backup files in a subfolder of your S3 bucket named <code>backups</code>, and you have multiple sets of backup files, each in its own directory (<code>gzip_backup1</code>, <code>gzip_backup2</code>, and so on), then you would</p>

AWS Database Migration Service  
 Step-by-Step Migration Guide  
 Step 3: Restoring an Aurora MySQL  
 DB Cluster from an Amazon S3 Bucket

For This Option	Do This
	specify a prefix of <code>backups/gzip_backup1</code> to restore from the files in the <code>gzip_backup1</code> folder.

b. In the **Engine Options** section, specify the following:

For This Option	Do This
<b>Engine Type</b>	Leave Amazon Aurora selected.
<b>Edition</b>	Leave Amazon Aurora with MySQL compatibility selected.
<b>Version</b>	Specify the version of the MySQL database that the backup files were created from, for example 5.7. MySQL version 5.6 and 5.7 are supported.

c. In the **IAM role** section, specify the following:

For This Option	Do This
<b>IAM Role</b>	Choose the IAM role that you created to authorize Amazon Aurora MySQL to access Amazon S3 on your behalf. If you have not created an IAM role, you can choose <b>Create a New Role</b> to create one.

d. In the **Settings** section, specify the following:

For This Option	Do This
<b>DB cluster identifier</b>	<p>Type a name for your DB cluster. This identifier will be used in the endpoint address for the primary instance of your DB cluster.</p> <p>The DB instance identifier has the following constraints:</p> <ul style="list-style-type: none"> <li>* It must contain from 1 to 63 alphanumeric characters or hyphens.</li> <li>* Its first character must be a letter.</li> <li>* It cannot end with a hyphen or contain two consecutive hyphens.</li> <li>* It must be unique for all DB instances per AWS account, per region.</li> </ul>

AWS Database Migration Service  
 Step-by-Step Migration Guide  
 Step 3: Restoring an Aurora MySQL  
 DB Cluster from an Amazon S3 Bucket

For This Option	Do This
<b>Master Username</b>	Type a name using alphanumeric characters that you will use as the master user name to log on to your DB cluster. The default privileges granted to the master user name account include: <code>create</code> , <code>drop</code> , <code>references</code> , <code>event</code> , <code>alter</code> , <code>delete</code> , <code>index</code> , <code>insert</code> , <code>select</code> , <code>update</code> , <code>create temporary tables</code> , <code>lock tables</code> , <code>trigger</code> , <code>create view</code> , <code>show view</code> , <code>alter routine</code> , <code>create routine</code> , <code>execute</code> , <code>create user</code> , <code>process</code> , <code>show databases</code> , <code>grant option</code> .
<b>Auto generate a password</b>	Leave unchecked.
<b>Master Password</b>	Type a password that contains from 8 to 41 printable ASCII characters (excluding <code>/</code> , <code>,</code> , and <code>@</code> ) for your master user password.
<b>Confirm Password</b>	Retype the Master Password.

e. In the **DB Instance Class** section, specify the following:

For This Option	Do This
<b>DB Instance Class</b>	Select a DB instance class that defines the processing and memory requirements for each instance in the DB cluster. Aurora MySQL supports the <code>db.r3.large</code> , <code>db.r3.xlarge</code> , <code>db.r3.2xlarge</code> , <code>db.r3.4xlarge</code> , and <code>db.r3.8xlarge</code> DB instance classes. For more information about DB instance class options, see the <a href="#">Amazon RDS documentation</a> .

f. In the **Availability & durability** section, specify the following:

For This Option	Do This
<b>Multi-AZ Deployment</b>	Determine if you want to create Aurora MySQL Replicas in other Availability Zones for failover support. For more information about multiple Availability Zones, see the <a href="#">Amazon RDS documentation</a> .

g. In the **Connectivity** section, specify the following:

For This Option	Do This
<b>Virtual private cloud (VPC)</b>	Select the VPC that will host the DB cluster. Select <b>Create a New VPC</b> to have Amazon RDS create a VPC for you. For more information, see earlier in this topic.

AWS Database Migration Service  
 Step-by-Step Migration Guide  
 Step 3: Restoring an Aurora MySQL  
 DB Cluster from an Amazon S3 Bucket

For This Option	Do This
<b>Subnet group</b>	Select the DB subnet group to use for the DB cluster. Select <b>Create a New DB Subnet Group</b> to have Amazon RDS create a DB subnet group for you. For more information, see earlier in this topic.
<b>Public access</b>	Select <b>Yes</b> to give the DB cluster a public IP address; otherwise, select <b>No</b> . The instances in your DB cluster can be a mix of both public and private DB instances. For more information about hiding instances from public access, see the <a href="#">Amazon RDS documentation</a> .
<b>VPC Security Group(s)</b>	Select one or more VPC security groups to secure network access to the DB cluster. Select <b>Create a New VPC Security Group</b> to have Amazon RDS create a VPC security group for you. For more information, see earlier in this topic.
<b>Availability Zone</b>	Determine if you want to specify a particular Availability Zone. For more information about Availability Zones, see the <a href="#">the Amazon RDS documentation</a> .
<b>Database Port</b>	Specify the port that applications and utilities will use to access the database. Aurora MySQL DB clusters default to the default MySQL port, 3306. The firewalls at some companies block connections to the default MySQL port. If your company firewall blocks the default port, choose another port for the new DB cluster.

h. In the **Database authentication** section, specify the following:

For This Option	Do This
<b>Database Authentication</b>	Leave <b>Password authentication</b> selected.

i. In the **Additional configuration** section, specify the following:

For This Option	Do This
<b>Initial Database Name</b>	Type a name for your database of up to 8 alphanumeric characters. If you don't provide a name, Amazon RDS will not create a database on the DB cluster you are creating.
<b>DB cluster parameter Group</b>	Select a parameter group for the cluster. Aurora MySQL has a default parameter group you can use, or you can create your own parameter group. For more information about parameter groups, see the <a href="#">Amazon RDS documentation</a> .
<b>DB parameter Group</b>	Select a parameter group for the database.



AWS Database Migration Service  
 Step-by-Step Migration Guide  
 Step 3: Restoring an Aurora MySQL  
 DB Cluster from an Amazon S3 Bucket

For This Option	Do This
<b>Option Group</b>	Select an option group. Aurora MySQL has a default option group you can use, or you can create your own option group. For more information about option groups, see the <a href="#">Amazon RDS documentation</a> .
<b>Failover Priority</b>	Choose a failover priority for the instance. If you don't select a value, the default is <b>tier-1</b> . This priority determines the order in which Aurora MySQL Replicas are promoted when recovering from a primary instance failure. For more information, see the <a href="#">Amazon RDS documentation</a> .
<b>Backup Retention Period</b>	Select the length of time, from 1 to 35 days, that Aurora MySQL will retain backup copies of the database. Backup copies can be used for point-in-time restores (PITR) of your database, timed down to the second.
<b>Copy tags to snapshots</b>	Leave checked.
<b>Enable Encryption</b>	Check the box to enable encryption at rest for this DB cluster. Leave <b>AWS KMS Key</b> set to <b>(default) aws/rds</b> . For more information, see the <a href="#">Amazon RDS documentation</a> .
<b>Backtrack</b>	Leave unchecked.
<b>Enable Performance insights</b>	Leave checked. Leave <b>Retention Period</b> and <b>AWS KMS Key</b> as they are.
<b>Enable Enhanced Monitoring</b>	Choose <b>Yes</b> to enable gathering metrics in real time for the operating system that your DB cluster runs on. For more information, see the <a href="#">Amazon RDS documentation</a> .
<b>Granularity</b>	This option is only available if <b>Enable Enhanced Monitoring</b> is set to <b>Yes</b> . Set the interval, in seconds, between times at which metrics are collected for your DB cluster.
<b>Monitoring role</b>	Leave as <b>default</b> .
<b>Log exports</b>	Leave unchecked.
<b>Enable auto Minor Version Upgrade</b>	<p>Check this box if you want to enable your Aurora MySQL DB cluster to receive minor MySQL DB engine version upgrades automatically when they become available.</p> <p>The <b>Auto Minor Version Upgrade</b> option only applies to upgrades to MySQL minor engine versions for your Amazon Aurora MySQL DB cluster. It doesn't apply to regular patches applied to maintain system stability.</p>

For This Option	Do This
Maintenance Window	Select the weekly time range during which system maintenance can occur.
Enable deletion protection	Leave unchecked.

4. Choose **Launch DB Instance** to launch your Aurora MySQL DB instance, and then choose **Close** to close the wizard.

On the Amazon RDS console, the new DB instance appears in the list of DB instances. The DB instance has a status of **creating** until the DB instance is created and ready for use. When the state changes to **available**, you can connect to the primary instance for your DB cluster. Depending on the DB instance class and store allocated, it can take several minutes for the new instance to be available.

To view the newly created cluster, choose the **Clusters** view in the Amazon RDS console. For more information, see the [Amazon RDS documentation](#).

Note the port and the endpoint of the cluster. Use the endpoint and port of the cluster in your JDBC and ODBC connection strings for any application that performs write or read operations.

## Migrating MySQL to Amazon Aurora MySQL by Using mysqldump

You can create a dump of your data using the `mysqldump` utility, and then import that data into an existing Amazon Aurora MySQL DB cluster.

Because Amazon Aurora MySQL is a MySQL-compatible database, you can use the `mysqldump` utility to copy data from your MySQL or MariaDB database to an existing Amazon Aurora MySQL DB cluster.

## Migrating Data from an Amazon RDS MySQL DB Instance to an Amazon Aurora MySQL DB Cluster

You can migrate (copy) data to an Amazon Aurora MySQL DB cluster from an Amazon RDS snapshot, as described following.

### Note

Because Amazon Aurora MySQL is compatible with MySQL, you can migrate data from your MySQL database by setting up replication between your MySQL database, and an Amazon Aurora MySQL DB cluster. We recommend that your MySQL database run MySQL version 5.5 or later.

## Migrating an RDS MySQL Snapshot to Aurora MySQL

You can migrate a DB snapshot of an Amazon RDS MySQL DB instance to create an Aurora MySQL DB cluster. The new DB cluster is populated with the data from the original Amazon RDS MySQL DB instance. The DB snapshot must have been made from an Amazon RDS DB instance running MySQL 5.6.

You can migrate either a manual or automated DB snapshot. After the DB cluster is created, you can then create optional Aurora MySQL Replicas.

The general steps you must take are as follows:

1. Determine the amount of space to provision for your Amazon Aurora MySQL DB cluster. For more information, see the [Amazon RDS documentation](#).
2. Use the console to create the snapshot in the region where the Amazon RDS MySQL 5.6 instance is located
3. If the DB snapshot is not in the region as your DB cluster, use the Amazon RDS console to copy the DB snapshot to that region. For information about copying a DB snapshot, see the [Amazon RDS documentation](#).
4. Use the console to migrate the DB snapshot and create an Amazon Aurora MySQL DB cluster with the same databases as the original DB instance of MySQL 5.6.

**Warning**

Amazon RDS limits each AWS account to one snapshot copy into each region at a time.

## How Much Space Do I Need?

When you migrate a snapshot of a MySQL DB instance into an Aurora MySQL DB cluster, Aurora MySQL uses an Amazon Elastic Block Store (Amazon EBS) volume to format the data from the snapshot before migrating it. In some cases, additional space is needed to format the data for migration. When migrating data into your DB cluster, observe the following guidelines and limitations:

- Although Amazon Aurora MySQL supports storage up to 64 TB in size, the process of migrating a snapshot into an Aurora MySQL DB cluster is limited by the size of the EBS volume of the snapshot. Thus, the maximum size for a snapshot that you can migrate is 6 TB.
- Tables that are not MyISAM tables and are not compressed can be up to 6 TB in size. If you have MyISAM tables, then Aurora MySQL must use additional space in the volume to convert the tables to be compatible with Aurora MySQL. If you have compressed tables, then Aurora MySQL must use additional space in the volume to expand these tables before storing them on the Aurora MySQL cluster volume. Because of this additional space requirement, you should ensure that none of the MyISAM and compressed tables being migrated from your MySQL DB instance exceeds 3 TB in size.

## Reducing the Amount of Space Required to Migrate Data into Amazon Aurora MySQL

You might want to modify your database schema prior to migrating it into Amazon Aurora MySQL. Such modification can be helpful in the following cases:

- You want to speed up the migration process.
- You are unsure of how much space you need to provision.
- You have attempted to migrate your data and the migration has failed due to a lack of provisioned space.

You can make the following changes to improve the process of migrating a database into Amazon Aurora MySQL.

**Important**

Be sure to perform these updates on a new DB instance restored from a snapshot of a production database, rather than on a production instance. You can then migrate the data from the snapshot of your new DB instance into your Amazon Aurora MySQL DB cluster to avoid any service interruptions on your production database.

Table Type	Limitation or Guideline
MyISAM tables	<p>Amazon Aurora MySQL supports InnoDB tables only. If you have MyISAM tables in your database, then those tables must be converted before being migrated into Amazon Aurora MySQL. The conversion process requires additional space for the MyISAM to InnoDB conversion during the migration procedure.</p> <p>To reduce your chances of running out of space or to speed up the migration process, convert all of your MyISAM tables to InnoDB tables before migrating them. The size of the resulting InnoDB table is equivalent to the size required by Amazon Aurora MySQL for that table. To convert a MyISAM table to InnoDB, run the following command:</p> <pre>alter table &lt;schema&gt;.&lt;table_name&gt; engine=innodb, algorithm=copy;</pre>
Compressed tables	<p>Amazon Aurora MySQL does not support compressed tables (that is, tables created with ROW_FORMAT=COMPRESSED).</p> <p>To reduce your chances of running out of space or to speed up the migration process, expand your compressed tables by setting ROW_FORMAT to DEFAULT, COMPACT, DYNAMIC, or REDUNDANT. For more information, see <a href="https://dev.mysql.com/doc/refman/5.6/en/innodb-row-format.html">https://dev.mysql.com/doc/refman/5.6/en/innodb-row-format.html</a>.</p>

You can use the following SQL script on your existing MySQL DB instance to list the tables in your database that are MyISAM tables or compressed tables.

```
-- This script examines a MySQL database for conditions that will block
-- migrating the database into an Amazon Aurora MySQL DB.
-- It needs to be run from an account that has read permission for the
-- INFORMATION_SCHEMA database.

-- Verify that this is a supported version of MySQL.

select msg as `==> Checking current version of MySQL.`
from
(
  select
    'This script should be run on MySQL version 5.6. ' +
    'Earlier versions are not supported.' as msg,
    cast(substring_index(version(), '.', 1) as unsigned) * 100 +
    cast(substring_index(substring_index(version(), '.', 2), '.', -1)
    as unsigned)
    as major_minor
  ) as T
where major_minor <> 506;

-- List MyISAM and compressed tables. Include the table size.

select concat(TABLE_SCHEMA, '.', TABLE_NAME) as `==> MyISAM or Compressed Tables`,
```

```
round(((data_length + index_length) / 1024 / 1024), 2) "Approx size (MB)"
from INFORMATION_SCHEMA.TABLES
where
  ENGINE <> 'InnoDB'
  and
  (
    -- User tables
    TABLE_SCHEMA not in ('mysql', 'performance_schema',
                          'information_schema')

    or

    -- Non-standard system tables
    (
      TABLE_SCHEMA = 'mysql' and TABLE_NAME not in
      (
        'columns_priv', 'db', 'event', 'func', 'general_log',
        'help_category', 'help_keyword', 'help_relation',
        'help_topic', 'host', 'ndb_binlog_index', 'plugin',
        'proc', 'procs_priv', 'proxies_priv', 'servers', 'slow_log',
        'tables_priv', 'time_zone', 'time_zone_leap_second',
        'time_zone_name', 'time_zone_transition',
        'time_zone_transition_type', 'user',
        'general_log_backup', 'slow_log_backup'
      )
    )
  )
)
or
(
  -- Compressed tables
  ROW_FORMAT = 'Compressed'
);
```

The script produces output similar to the output in the following example. The example shows two tables that must be converted from MyISAM to InnoDB. The output also includes the approximate size of each table in megabytes (MB).

```
+-----+-----+
| ==> MyISAM or Compressed Tables | Approx size (MB) |
+-----+-----+
| test.name_table                 |          2102.25 |
| test.my_table                   |           65.25 |
+-----+-----+
2 rows in set (0.01 sec)
```

## Migrating a DB Snapshot by Using the Console

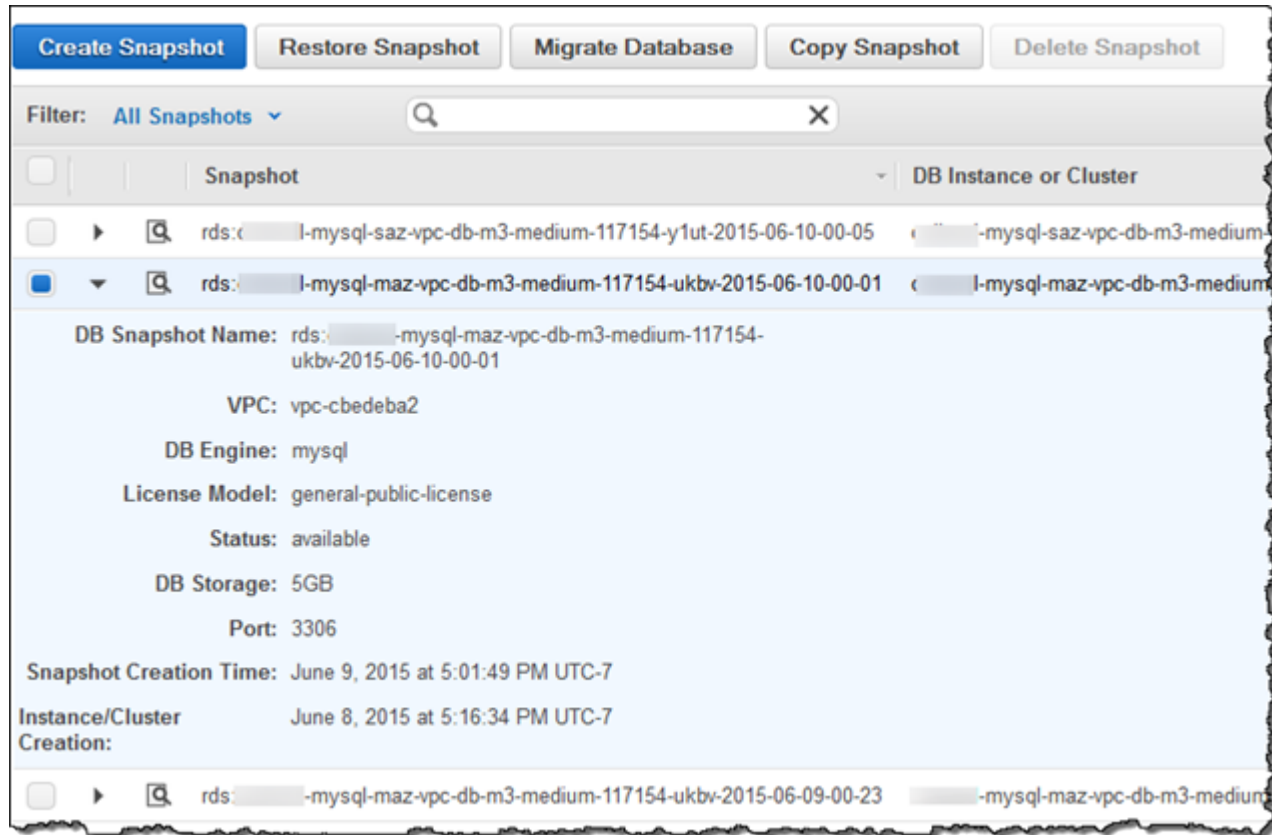
You can migrate a DB snapshot of an Amazon RDS MySQL DB instance to create an Aurora MySQL DB cluster. The new DB cluster will be populated with the data from the original Amazon RDS MySQL DB instance. The DB snapshot must have been made from an Amazon RDS DB instance running MySQL 5.6 and must not be encrypted. For information about creating a DB snapshot, see the [Amazon RDS documentation](#).

If the DB snapshot is not in the AWS Region where you want to locate your data, use the Amazon RDS console to copy the DB snapshot to that region. For information about copying a DB snapshot, see the [Amazon RDS documentation](#).

When you migrate the DB snapshot by using the console, the console takes the actions necessary to create both the DB cluster and the primary instance.

You can also choose for your new Aurora MySQL DB cluster to be encrypted "at rest" using an AWS Key Management Service (AWS KMS) encryption key. This option is available only for unencrypted DB snapshots.

1. Sign in to the AWS Management Console and open the Amazon RDS console at <https://console.aws.amazon.com/rds/>.
2. Choose **Snapshots**.
3. On the **Snapshots** page, choose the snapshot that you want to migrate into an Aurora MySQL DB cluster.
4. Choose **Migrate Database**.



5. Set the following values on the **Migrate Database** page:
  - **DB Instance Class:** Select a DB instance class that has the required storage and capacity for your database, for example `db.r3.large`. Aurora MySQL cluster volumes automatically grow as the amount of data in your database increases, up to a maximum size of 64 terabytes (TB). So you only need to select a DB instance class that meets your current storage requirements.
  - **DB Instance Identifier:** Type a name for the DB cluster that is unique for your account in the region you selected. This identifier is used in the endpoint addresses for the instances in your DB cluster. You might choose to add some intelligence to the name, such as including the region and DB engine you selected, for example `aurora-cluster1`.

The DB instance identifier has the following constraints:

- It must contain from 1 to 63 alphanumeric characters or hyphens.
  - Its first character must be a letter.
  - It cannot end with a hyphen or contain two consecutive hyphens.
  - It must be unique for all DB instances per AWS account, per AWS Region.
- **VPC:** If you have an existing VPC, then you can use that VPC with your Amazon Aurora MySQL DB cluster by selecting your VPC identifier, for example `vpc-a464d1c1`. For information on using an existing VPC, see the [Amazon RDS documentation](#).

Otherwise, you can choose to have Amazon RDS create a VPC for you by selecting **Create a new VPC**.

- **Subnet Group:** If you have an existing subnet group, then you can use that subnet group with your Amazon Aurora MySQL DB cluster by selecting your subnet group identifier, for example `gs-subnet-group1`.

Otherwise, you can choose to have Amazon RDS create a subnet group for you by selecting **Create a new subnet group**.

- **Publicly Accessible:** Select **No** to specify that instances in your DB cluster can only be accessed by resources inside of your VPC. Select **Yes** to specify that instances in your DB cluster can be accessed by resources on the public network. The default is **Yes**.

**Note**

Your production DB cluster might not need to be in a public subnet, because only your application servers will require access to your DB cluster. If your DB cluster doesn't need to be in a public subnet, set **Publicly Accessible** to **No**.

- **Availability Zone:** Select the Availability Zone to host the primary instance for your Aurora MySQL DB cluster. To have Amazon RDS select an Availability Zone for you, select **No Preference**.
- **Database Port:** Type the default port to be used when connecting to instances in the DB cluster. The default is 3306.

**Note**

You might be behind a corporate firewall that doesn't allow access to default ports such as the MySQL default port, 3306. In this case, provide a port value that your corporate firewall allows. Remember that port value later when you connect to the Aurora MySQL DB cluster.

- **Enable Encryption:** Choose **Yes** for your new Aurora MySQL DB cluster to be encrypted "at rest." If you choose **Yes**, you will be required to choose an AWS KMS encryption key as the KMS key value.
- **Auto Minor Version Upgrade:** Select **Yes** if you want to enable your Aurora MySQL DB cluster to receive minor MySQL DB engine version upgrades automatically when they become available.

The **Auto Minor Version Upgrade** option only applies to upgrades to MySQL minor engine versions for your Amazon Aurora MySQL DB cluster. It doesn't apply to regular patches applied to maintain system stability.

## Instance Specifications

**Migrate to DB Engine**

**DB Instance Class**

## Settings

**DB Snapshot ID** rds- -2016-02-22-07-42

**DB Instance Identifier\***

## Network & Security

This instance will be created with the new Certificate Authority rds-ca-2015. If you are using SSL to connect to this instance, you should use the [new certificate bundle](#). Learn more [here](#)

**VPC\***

**Subnet Group**

**Publicly Accessible**

**Availability Zone**

## Database Options

**Database Port**

**Enable Encryption**

## Maintenance

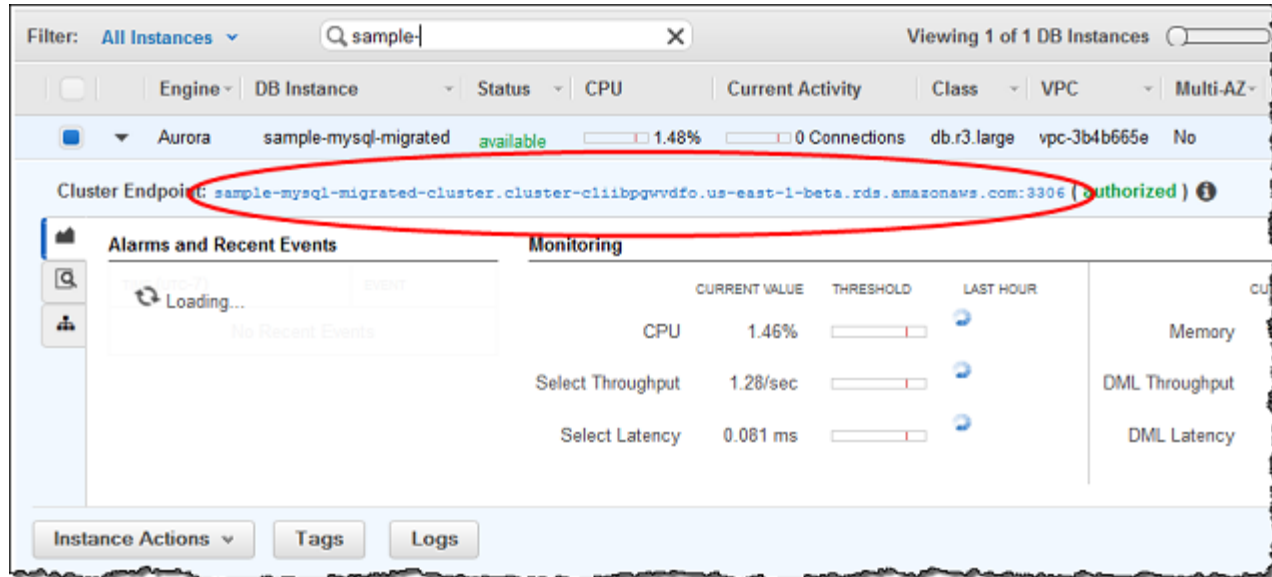
**Auto Minor Version Upgrade**

Cancel

Migrate



6. Choose **Migrate** to migrate your DB snapshot.
7. Choose **Instances**, and then choose the arrow icon to show the DB cluster details and monitor the progress of the migration. On the details page, you will find the cluster endpoint used to connect to the primary instance of the DB cluster. For more information on connecting to an Amazon Aurora MySQL DB cluster, see the [Amazon RDS documentation](#).



# Migrating a MariaDB Database to Amazon RDS for MySQL or Amazon Aurora MySQL

You can migrate data from existing on-premises MariaDB or [Amazon RDS for MariaDB](#) to [Amazon Aurora MySQL](#) using [AWS Database Migration Service \(DMS\)](#). Amazon Aurora is a MySQL and PostgreSQL-compatible relational database built for the cloud. Amazon Aurora features a distributed, fault-tolerant, self-healing storage system that auto-scales up to 64 TB per database instance. It delivers high performance and availability with up to 15 low-latency read replicas, point-in-time recovery, and continuous backup to Amazon S3, and replication across three Availability Zones (AZs).

Some key features offered by Aurora MySQL are the following:

- High throughput with low latency
- Push-button compute scaling
- Storage autoscaling
- Custom database endpoints
- Parallel queries for faster analytics

In the following sections, we demonstrate migration from MariaDB as a source database to an Aurora MySQL database as a target using AWS DMS. At a high level, the steps involved in this migration are:

- Provision MariaDB as a source DB instance and load the data
- Provision Aurora MySQL as target DB instance
- Provision DMS replication instance and create DMS endpoints
- Create DMS task, migrate data and perform validation

For the purpose of this section, we are using the CloudFormation templates for creating Amazon RDS for MariaDB, Aurora MySQL database and AWS DMS replication instance with their source and endpoints. We will be loading sample tables and data in MariaDB located on [GitHub](#).

## Topics

- [Set up MariaDB as a source database \(p. 158\)](#)
- [Set up Aurora MySQL as a target database \(p. 161\)](#)
- [Set up an AWS DMS replication instance \(p. 162\)](#)
- [Test the endpoints \(p. 163\)](#)
- [Create a migration task \(p. 163\)](#)
- [Validate the migration \(p. 164\)](#)
- [Cut over \(p. 164\)](#)

## Set up MariaDB as a source database

To provision MariaDB as a source database, download [the Mariadb\\_CF.yaml template](#). This AWS CloudFormation template creates an Amazon RDS for MariaDB instance with the required parameters.

1. On the AWS Management Console, under **Services**, choose **CloudFormation**.
2. Choose **Create Stack**.
3. For **Specify template**, choose **Upload a template file**.
4. Select **Choose File**.
5. Choose the `MariaDB.yaml` file.
6. Choose **Next**.
7. On the **Specify Stack Details** page, edit the predefined values as needed, and then choose **Next**:
  - **Stack name** – Enter a name for the stack.
  - **CIDR** – Enter the CIDR IP range to access the instance.
  - **DBAllocated Storage** – Enter the database storage size in GB. The default is 20 GB.
  - **DBBackupRetentionPeriod** – The number of days to retain backups.
  - **DBInstanceClass** – Enter the instance type of the database server.
  - **DBMonitoringInterval** – Interval to publish database logs to Amazon CloudWatch.
  - **DBSubnetGroup** – Enter the DB subnet group.
  - **MariaDBEngine** – Enter the MariaDB engine version.
  - **RDSDBName** – Enter the name of the database.
  - **VPCID** – Enter the VPC to launch your DB instance.
8. On the **Configure stack options** page, for **Tags**, specify any optional tags, and then choose **Next**.
9. On the **Review** page, select **I acknowledge that AWS CloudFormation might create IAM resources**.
10. Choose **Create Stack**.

After the Amazon RDS for MariaDB instance is created, log in to MariaDB and run the following statements to create `webdb_user` — a superuser that connects to a DMS instance for migration, and grant necessary privileges.

```
CREATE USER 'webdb_user'@'%' IDENTIFIED BY '*****';
GRANT ALL ON migrate.* TO 'webdb_user'@'%' with grant option;
grant REPLICATION SLAVE ON *.* TO webdb_user;
grant REPLICATION CLIENT ON *.* TO webdb_user;
```

In this walkthrough, we created a database called *migration* and few sample tables, along with stored procedures, triggers, functions, and so on. The below query provides the list of tables in *migration* database:

```
MariaDB [(none)]> use migration
Database changed
MariaDB [migration]> show tables;
+-----+
| Tables_in_migration |
+-----+
| animal_count        |
| animals              |
| contacts             |
| seat_type            |
| sport_location       |
| sport_team           |
| sport_type           |
+-----+
7 rows in set (0.000 sec)
```

The following query returns a list of secondary indexes.

AWS Database Migration Service  
Step-by-Step Migration Guide  
Set up MariaDB as a source database

```
MariaDB [migration]> SELECT DISTINCT TABLE_NAME, INDEX_NAME, NON_UNIQUE
-> FROM INFORMATION_SCHEMA.STATISTICS
-> WHERE TABLE_SCHEMA = 'migration' and INDEX_NAME <> 'PRIMARY';
+-----+-----+-----+
| TABLE_NAME | INDEX_NAME | NON_UNIQUE |
+-----+-----+-----+
| sport_location | city_id_sport_loc | 1 |
| sport_team | sport_team_u | 0 |
| sport_team | home_field_fk | 1 |
+-----+-----+-----+
3 rows in set (0.000 sec)
```

The following query returns a list of triggers.

```
MariaDB [migration]> select TRIGGER_SCHEMA, TRIGGER_NAME
-> from information_schema.triggers
-> where TRIGGER_SCHEMA='migration';
+-----+-----+
| TRIGGER_SCHEMA | TRIGGER_NAME |
+-----+-----+
| migration | increment_animal |
| migration | contacts_after_update |
+-----+-----+
2 rows in set (0.001 sec)
```

The following query returns a list of procedures and functions.

```
MariaDB [(none)]> select routine_schema as database_name,
-> routine_name,
-> routine_type as type,
-> data_type as return_type
-> from information_schema.routines
-> where routine_schema not in ('sys', 'information_schema',
-> 'mysql', 'performance_schema');
+-----+-----+-----+-----+
| database_name | routine_name | type | return_type |
+-----+-----+-----+-----+
| migration | CalcValue | FUNCTION | int |
| migration | loadMLBPlayers | PROCEDURE | |
| migration | loadNFLPlayers | PROCEDURE | |
+-----+-----+-----+-----+
3 rows in set (0.000 sec)
```

After all the data is loaded, use `mysqldump` to back up the database metadata. The `mysqldump` utility to dump one or more databases for backup or transfer to another database server. The dump typically contains SQL statements to create the table, populate it, or both. You can also use `mysqldump` to generate files in comma-separated value (CSV), other delimited text, or XML format.

Use the following command exports tables and index definitions:

```
$ mysqldump --no-data --no-create-db --single_transaction -u root -p migration --skip-triggers > mysql_tables_indexes.sql
```

Use following command to exports routines (stored procedures, functions, and triggers) into the file `routines.sql`:

```
$ mysqldump -u root --routines --no-create-info --no-data --no-create-db --skip-opt -p migration > routines.sql
```

The `mysqldump` utility doesn't provide the option to remove a `DEFINER` statement. Some MySQL clients provide the option to ignore the definer when creating a logical backup, but this isn't the default behavior. Use the following command in a UNIX or Linux environment to remove the `DEFINER` from `routines.sql`:

```
$ sed -i -e 's/DEFINER=`root`@`localhost`/DEFINER=`master`@`%`/g' routines.sql
```

We now have a backup of MariaDB, in two `#0#sql` files (`mysql_tables_indexes.sql` and `routines.sql`). We will use these files to load the table definition into an Aurora MySQL database.

After backups are completed into two `.sql` files (`mysql_tables_indexes.sql`, `routines.sql`), use these files to load the table definition into the Aurora MySQL database.

## Set up Aurora MySQL as a target database

To provision Aurora MySQL as a target database, download the [AuroraMySQL\\_CF.yaml template](#). This template creates an Aurora MySQL database with required parameters.

1. On the AWS Management Console, under **Services**, choose **CloudFormation**.
2. Choose **Create Stack**.
3. For **Specify template**, choose **Upload a template file**.
4. Select **Choose File**.
5. Choose the `AuroraMySQL.yaml` file.
6. Choose **Next**.
7. On the **Specify Stack Details** page, edit the predefined values as needed, and then choose **Next**:
  - **Stack name** – Enter a name for the stack.
  - **CIDR** – Enter the CIDR IP range to access the instance.
  - **DBBackupRetentionPeriod** – The number of days for backup retention.
  - **DBInstanceClass** – Enter the instance type of the database server.
  - **DBMasterUsername** – Enter the master user name for DB instance
  - **DBMasterPassword** – Enter the master user name password for DB instance.
  - **DBSubnetGroup** – Enter the DB subnet group.
  - **Engine** – Enter the Aurora engine version; the default is `5.7.mysql-aurora.2.03.4`.
  - **DBName** – Enter the name of the database.
  - **VPCID** – Enter the ID for the VPC to launch your DB instance in.
8. On the **Configure stack options** page, for **Tags**, specify any optional tags, and then choose **Next**.
9. On the **Review** page, choose **I acknowledge that AWS CloudFormation might create IAM resources**.
10. Choose **Create Stack**.

After the Aurora MySQL database is created, log in to the Aurora MySQL instance:

```
$ mysql -h mysqltrg-instance-1.xxxxxxxx.us-east-1.rds.amazonaws.com -u master -p migration -P 3306
MySQL [(none)]> show databases;
+-----+
| Database          |
+-----+
| information_schema|
| awsdms_control    |
| mysql              |
```

```
| performance_schema |  
| source             |  
| tmp                |  
| webdb              |  
+-----+  
7 rows in set (0.001 sec)  
  
MySQL [(none)]> create database migration;  
Query OK, 1 row affected (0.016 sec)  
  
MySQL [(none)]> use migration;  
Database changed  
  
MySQL [migration]> show tables;  
Empty set (0.001 sec)
```

Use `mysql_tables_indexes.sql` to create table and index structures in Aurora MySQL.

```
$ mysql -h mysqltrg-instance-1.xxxxxxxxxx.us-east-1.rds.amazonaws.com -u master -p  
migration -P 3306 < mysql_tables_indexes.sql  
Enter password:  
$
```

After the tables and indexes are successfully created, the next step is to set up and use AWS DMS.

## Set up an AWS DMS replication instance

To provision an AWS DMS replication instance, download [the DMS\\_CF.yaml template](#).

1. On the AWS Management Console, under **Services**, choose **CloudFormation**.
2. Choose **Create Stack**.
3. For **Specify template**, choose **Upload a template file**.
4. Select **Choose File**.
5. Choose the `DMS_CF.yaml` file.
6. Choose **Next**.
7. On the **Specify Stack Details** page, edit the predefined values as needed, and then choose **Next**:
  - **Stack name** – Enter a name for the stack.
  - **AllocatedStorageSize** – Enter the storage size in GB. The default is 200 GB.
  - **DMSReplicationSubnetGroup** – Enter the subnet group for DMS replication.
  - **DMSSecurityGroup** – Enter the security group for DMS replication.
  - **InstanceType** – Enter the instance type.
  - **SourceDBPort** – Enter the source database port.
  - **SourceDatabaseName** – Enter the source database name.
  - **SourceServerName** – Enter the IP address of the source database server.
  - **SourceUsername** – Enter the source database user name.
  - **SourcePassword** – Enter the source database password.
  - **TargetDBPort** – Enter the target database port.
  - **TargetDatabaseName** – Enter the target database name.
  - **TargetServerName** – Enter the IP address of the target database server.
  - **TargetUsername** – Enter the target database user name.
  - **TargetPassword** – Enter the target database password.

8. On the **Configure stack options** page, for **Tags**, specify any optional tags, and then choose **Next**.
9. On the **Review** page, choose **I acknowledge that AWS CloudFormation might create IAM resources**.
10. Choose **Create Stack**.

This AWS CloudFormation template creates a replication instance named `mariadb-mysql`. This replication instance has a source endpoint named `maria-on-prem` and a target endpoint named `mysqltrg-rds`. This target endpoint has extra connection attributes to disable foreign key constraint checks during the AWS DMS replication, as shown following.

```
ExtraConnectionAttributes : "initstmt=SET FOREIGN_KEY_CHECKS=0;parallelLoadThreads=1"
```

## Test the endpoints

1. On the navigation pane, choose **Endpoints**.
2. Choose the source endpoint name (`maria-on-prem`) and do the following:
  - a. Choose **Test connections**.
  - b. Choose the replication instance to test (`mariadb-mysql`).
  - c. Choose **Run Test** and wait for the status to be **successful**.
3. On the navigation pane, choose **Endpoints**.
4. Choose the target endpoint name (`mysqltrg-rds`) and do the following:
  - a. Choose **Test Connections**.
  - b. Choose the replication instance to test (`mariadb-mysql`).
  - c. Choose **Run Test** and wait for the status to be **successful**.

### Note

If **Run Test** returns a status other than **successful**, the reason for the failure is displayed. Make sure that you resolve the issue before proceeding further.

## Create a migration task

We've now verified that the replication instance can connect to both the source and target endpoints. The next step is to create a database migration task.

1. On the navigation pane, choose **Database Migration Tasks**.
2. Choose **Create Task**. Provide the specified values for the following, and then choose **Next**:
  - **Task identifier** – `maria-mysql`
  - **Replication instance** – Choose the replication instance, `mariadb-mysql`.
  - **Source database endpoint** – Choose the source database, `maria-on-prem`.
  - **Target database endpoint** – Choose the target database, `mysqltrg-rds`.
  - **Migration Type** – Choose **Migrate existing data and replicate ongoing changes** for CDC, or **Migrate existing data** for full load.
3. For **Task settings**, choose the following settings:
  - **Target table preparation mode** – Do nothing
  - **Stop task after full load completes** – Don't stop
  - **Include LOB columns in replication** – Limited LOB mode

- **Maximum LOB size (KB)** – 32
  - **Enable validation**
  - **Enable CloudWatch logs**
4. For **Table mappings**, choose the following settings:
    - **Schema** – Choose **migration** (assuming the schema and database to be migrated appear correctly).
    - **Table name** – Enter the table name, or % to specify all the tables in the database.
    - **Action** – Enter **Include** to include specific tables, or **Exclude** to exclude specific tables.
  5. Choose **Create Task**.

Your new AWS DMS migration task reads the data from the tables in the MariaDB source and migrates your data to the Aurora MySQL target.

## Validate the migration

AWS DMS performs data validation to confirm that your data successfully migrated the source database to the target. You can check the **Table statistics** page to determine the DML changes that occurred after the AWS DMS task started. During data validation, AWS DMS compares each row in the source with its corresponding row at the target, and verifies that those rows contain the same data. To accomplish this, AWS DMS issues the appropriate queries to retrieve the data.

After your data is loaded successfully, you can select your task on the AWS DMS page and choose **Table statistics** to show statistics about your migration. The following screen shot shows the **Table statistics** page and its relevant entries.

The following screenshot shows the table statics page and its relevant entries.

**Table statistics (8)**

Find schema

<input type="checkbox"/>	Schema name ▾	Table ▾	Load state ▾	Inserts ▾	Deletes ▾	Updates ▾	DDLs ▾	Full load rows ▾	Total ▾	Validation state ▾
<input type="checkbox"/>	migration	contacts	Table completed	0	0	0	0	0	0	Validated
<input type="checkbox"/>	migration	seat_type	Table completed	0	0	0	0	6	6	Validated
<input type="checkbox"/>	migration	player	Table completed	0	0	0	0	0	0	Validated
<input type="checkbox"/>	migration	sport_type	Table completed	0	0	0	0	2	2	Validated
<input type="checkbox"/>	migration	sport_team	Table completed	0	0	0	0	32	32	Validated
<input type="checkbox"/>	migration	sport_location	Table completed	0	0	0	0	30	30	Validated
<input type="checkbox"/>	migration	animals	Table completed	0	0	0	0	0	0	Validated
<input type="checkbox"/>	migration	animal_count	Table completed	0	0	0	0	1	1	No primary Key

AWS DMS can validate the data between source and target engines. The **Validation state** column helps us to validate the data migration. This ensures that your data was migrated accurately from the source to the target.

## Cut over

After the data validation is complete and any problems resolved, you can load the database triggers, functions, and procedures.

To do this, use the `routines.sql` file generated from MariaDB to create the necessary routines in Aurora MySQL. The following statement loads all procedures, functions, and triggers into the Aurora MySQL database.



AWS Database Migration Service  
Step-by-Step Migration Guide  
Cut over

```
$ mysql -h mysqltrg-instance-1.xxxxxxxxxx.us-east-1.rds.amazonaws.com -u master -p migration -P 3306 < routines.sql
```

After the routines are loaded, connect to the Aurora MySQL database to validate as shown following.

```
$ mysql -h mysqltrg-instance-1.xxxxxxxxxx.us-east-1.rds.amazonaws.com -u master -p migration -P 3306
Enter password:
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MySQL connection id is 957
Server version: 5.6.10 MySQL Community Server (GPL)

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MySQL [migration]> select routine_schema as database_name,
->         routine_name,
->         routine_type as type,
->         data_type as return_type
->         from information_schema.routines
->         where routine_schema not in ('sys', 'information_schema',
->                                     'mysql', 'performance_schema');
+-----+-----+-----+-----+
| database_name | routine_name | type      | return_type |
+-----+-----+-----+-----+
| migration    | CalcValue    | FUNCTION  | int          |
| migration    | loadMLBPlayers | PROCEDURE |              |
| migration    | loadNFLPlayers | PROCEDURE |              |
+-----+-----+-----+-----+
3 rows in set (0.002 sec)

MySQL [migration]> select TRIGGER_SCHEMA, TRIGGER_NAME from information_schema.triggers
where TRIGGER_SCHEMA='migration';
+-----+-----+
| TRIGGER_SCHEMA | TRIGGER_NAME |
+-----+-----+
| migration      | increment_animal |
| migration      | contacts_after_update |
+-----+-----+
2 rows in set (0.009 sec)
```

The preceding output shows that all the procedures, triggers, and functions are loaded successfully to the Aurora MySQL database.

# Migrating from MongoDB to Amazon DocumentDB

Use the following tutorial to guide you through the process of migrating from MongoDB to Amazon DocumentDB (with MongoDB compatibility). In this tutorial, you do the following:

- Install MongoDB on an Amazon EC2 instance.
- Populate MongoDB with sample data.
- Create an AWS DMS replication instance, a source endpoint (for MongoDB), and a target endpoint (for Amazon DocumentDB).
- Run an AWS DMS task to migrate the data from the source endpoint to the target endpoint.

## Important

Before you begin, make sure to launch an Amazon DocumentDB cluster in your default virtual private cloud (VPC). For more information, see [Getting started](#) in the *Amazon DocumentDB Developer Guide*.

## Topics

- [Launch an Amazon EC2 instance](#) (p. 166)
- [Install and configure MongoDB community edition](#) (p. 167)
- [Create an AWS DMS replication instance](#) (p. 168)
- [Create source and target endpoints](#) (p. 169)
- [Create and run a migration task](#) (p. 171)

## Launch an Amazon EC2 instance

For this tutorial, you launch an Amazon EC2 instance into your default VPC.

1. Open the Amazon EC2 console at <https://console.aws.amazon.com/ec2/>.
2. Choose **Launch Instance**, and do the following:
  - a. On the **Choose an Amazon Machine Image (AMI)** page, at the top of the list of AMIs, go to **Amazon Linux AMI** and choose **Select**.
  - b. On the **Choose an Instance Type** page, at the top of the list of instance types, choose **t2.micro**. Then choose **Next: Configure Instance Details**.
  - c. On the **Configure Instance Details** page, for **Network**, choose your default VPC. Then choose **Next: Add Storage**.
  - d. On the **Add Storage** page, skip this step by choosing **Next: Add Tags**.
  - e. On the **Add Tags** page, skip this step by choosing **Next: Configure Security Group**.
  - f. On the **Configure Security Group** page, do the following:
    - i. Choose **Select an existing security group**.
    - ii. In the list of security groups, choose **default**. Doing this chooses the default security group for your VPC. By default, the security group accepts inbound Secure Shell (SSH) connections on

TPC port 22. If this isn't the case for your VPC, add this rule; for more information, see [What is Amazon VPC?](#) in the *Amazon VPC User Guide*.

iii. Choose **Next: Review and Launch**.

g. Review the information, and choose **Launch**.

3. In the **Select an existing key pair or create a new key pair** window, do one of the following:

- If you don't have an Amazon EC2 key pair, choose **Create a new key pair** and follow the instructions. You are asked to download a private key file (.pem file). You need this file later when you log in to your Amazon EC2 instance.
- If you already have an Amazon EC2 key pair, for **Select a key pair** choose your key pair from the list. You must already have the private key file (.pem file) available in order to log in to your Amazon EC2 instance.

4. After you configure your key pair, choose **Launch Instances**.

In the console navigation pane, choose **EC2 Dashboard**, and then choose the instance that you launched. In the lower pane, on the **Description** tab, find the **Public DNS** location for your instance, for example: `ec2-11-22-33-44.us-west-2.compute.amazonaws.com`.

It takes a few minutes for your Amazon EC2 instance to become available.

5. Use the `ssh` command to log in to your Amazon EC2 instance, as in the following example.

```
chmod 400 my-keypair.pem
ssh -i my-keypair.pem ec2-user@public-dns-name
```

Specify your private key file (.pem file) and the public DNS name of your EC2 instance. The login ID is `ec2-user`. No password is required.

For further details about connecting to your EC instance, see [Connecting to your Linux instance using SSH](#) in the *Amazon EC2 User Guide for Linux Instances*.

## Install and configure MongoDB community edition

Perform these steps on the Amazon EC2 instance that you launched in [Launch an Amazon EC2 instance](#) (p. 166).

1. Go to [Install MongoDB community edition on Amazon Linux](#) in the MongoDB documentation and follow the instructions there.
2. By default, the MongoDB server (`mongod`) only allows loopback connections from IP address 127.0.0.1 (localhost). To allow connections from elsewhere in your Amazon VPC, do the following:
  - a. Edit the `/etc/mongod.conf` file and look for the following lines.

```
# network interfaces
net:
  port: 27017
  bindIp: 127.0.0.1 # Enter 0.0.0.0,:: to bind to all IPv4 and IPv6 addresses or,
alternatively, use the net.bindIpAll setting.
```

b. Modify the `bindIp` line so that it looks like the following.

```
bindIp: public-dns-name
```

- c. Replace `public-dns-name` with the actual public DNS name for your instance, for example `ec2-11-22-33-44.us-west-2.compute.amazonaws.com`.
- d. Save the `/etc/mongod.conf` file, and then restart `mongod`.

```
sudo service mongod restart
```

3. Populate your MongoDB instance with data by doing the following:

- a. Use the `wget` command to download a JSON file containing sample data.

```
wget http://media.mongodb.org/zips.json
```

- b. Use the `mongoimport` command to import the data into a new database (`zips-db`).

```
mongoimport --host public-dns-name:27017 --db zips-db --file zips.json
```

- c. After the import completes, use the `mongo` shell to connect to MongoDB and verify that the data was loaded successfully.

```
mongo --host public-dns-name:27017
```

- d. Replace `public-dns-name` with the actual public DNS name for your instance.  
e. At the `mongo` shell prompt, enter the following commands.

```
use zips-db
db.zips.count()

db.zips.aggregate( [
  { $group: { _id: { state: "$state", city: "$city" }, pop: { $sum: "$pop" } } },
  { $group: { _id: "$_id.state", avgCityPop: { $avg: "$pop" } } }
] )
```

The output should display the following:

- The name of the database (`zips-db`)
  - The number of documents in the `zips` collection (29353)
  - The average population for cities in each state
- f. Exit from the `mongo` shell and return to the command prompt by using the following command.

```
exit
```

## Create an AWS DMS replication instance

To perform replication in AWS DMS, you need a replication instance.

1. Open the AWS DMS console at <https://console.aws.amazon.com/dms/>.
2. In the navigation pane, choose **Replication instances**.
3. Choose **Create replication instance** and enter the following information:
  - For **Name**, enter `mongodb2docdb`.
  - For **Description**, enter `MongoDB to Amazon DocumentDB replication instance`.
  - For **Instance class**, keep the default value.
  - For **Engine version**, keep the default value.
  - For **VPC**, choose your default VPC.
  - For **Multi-AZ**, choose **No**.
  - For **Publicly accessible**, enable this option.

When the settings are as you want them, choose **Create replication instance**.

**Note**

You can begin using your replication instance when its status becomes **available**. This can take several minutes.

## Create source and target endpoints

The source endpoint is the endpoint for your MongoDB installation running on your Amazon EC2 instance.

1. Open the AWS DMS console at <https://console.aws.amazon.com/dms/>.
2. In the navigation pane, choose **Endpoints**.
3. Choose **Create endpoint** and enter the following information:
  - For **Endpoint type**, choose **Source**.
  - For **Endpoint identifier**, enter a name that's easy to remember, for example `mongodb-source`.
  - For **Source engine**, choose **mongodb**.
  - For **Server name**, enter the public DNS name of your Amazon EC2 instance, for example `ec2-11-22-33-44.us-west-2.compute.amazonaws.com`.
  - For **Port**, enter `27017`.
  - For **SSL mode**, choose **none**.
  - For **Authentication mode**, choose **none**.
  - For **Database name**, enter `zips-db`.
  - For **Authentication mechanism**, choose **default**.
  - For **Metadata mode**, choose **document**.

When the settings are as you want them, choose **Create endpoint**.

Next, you create a target endpoint. This endpoint is for your Amazon DocumentDB cluster, which should already be running. For more information on launching your Amazon DocumentDB cluster, see [Getting started](#) in the *Amazon DocumentDB Developer Guide*.

**Important**

Before you proceed, do the following:

- Have available the master user name and password for your Amazon DocumentDB cluster.
- Have available the DNS name and port number of your Amazon DocumentDB cluster, so that AWS DMS can connect to it. To determine this information, use the following AWS CLI command, replacing `cluster-id` with the name of your Amazon DocumentDB cluster.

```
aws docdb describe-db-clusters \  
  --db-cluster-identifier cluster-id \  
  --query "DBClusters[*].[Endpoint,Port]"
```

- Download a certificate bundle that Amazon DocumentDB can use to verify SSL connections. To do this, enter the following command. Here, `aws-api-domain` completes the Amazon S3 domain in your AWS Region required to access the specified S3 bucket and the `rds-combined-ca-bundle.pem` file that it provides.

```
wget https://s3.aws-api-domain/rds-downloads/rds-combined-ca-bundle.pem
```

To create a target endpoint, do the following:

1. In the navigation pane, choose **Endpoints**.
2. Choose **Create endpoint** and enter the following information:
  - For **Endpoint type**, choose **Target**.
  - For **Endpoint identifier**, enter a name that's easy to remember, for example `docdb-target`.
  - For **Target engine**, choose **docdb**.
  - For **Server name**, enter the DNS name of your Amazon DocumentDB cluster.
  - For **Port**, enter the port number of your Amazon DocumentDB cluster.
  - For **SSL mode**, choose **verify-full**.
  - For **CA certificate**, do one of the following to attach the SSL certificate to your endpoint:
    - If available, choose the existing **rds-combined-ca-bundle** certificate from the **Choose a certificate** drop down.
    - Choose **Add new CA certificate**. Then, for **Certificate identifier**, enter `rds-combined-ca-bundle`. For **Import certificate file**, choose **Choose file** and navigate to the `rds-combined-ca-bundle.pem` file that you previously downloaded. Select and open the file. Choose **Import certificate**, then choose **rds-combined-ca-bundle** from the **Choose a certificate** drop down.
  - For **User name**, enter the master user name of your Amazon DocumentDB cluster.
  - For **Password**, enter the master password of your Amazon DocumentDB cluster.
  - For **Database name**, enter `zips-db`.

When the settings are as you want them, choose **Create endpoint**.

Now that you've created the source and target endpoints, test them to ensure that they work correctly. Also, to ensure that AWS DMS can access the database objects at each endpoint, refresh the endpoints' schemas.

To test an endpoint, do the following:

1. In the navigation pane, choose **Endpoints**.
2. Choose the source endpoint (`mongodb-source`), and then choose **Test connection**.
3. Choose your replication instance (`mongodb2docdb`), and then choose **Run test**. It takes a few minutes for the test to complete, and for the **Status** to change to **successful**.

If the **Status** changes to **failed** instead, review the failure message. Correct any errors that might be present, and test the endpoint again.

**Note**

Repeat this procedure for the target endpoint (`docdb-target`).

To refresh schemas, do the following:

1. In the navigation pane, choose **Endpoints**.
2. Choose the source endpoint (`mongodb-source`), and then choose **Refresh schemas**.
3. Choose your replication instance (`mongodb2docdb`), and then choose **Refresh schemas**.

**Note**

Repeat this procedure for the target endpoint (`docdb-target`).

## Create and run a migration task

You are now ready to launch an AWS DMS migration task, to migrate the `zips` data from MongoDB to Amazon DocumentDB.

1. Open the AWS DMS console at <https://console.aws.amazon.com/dms/v2/>.
  2. In the navigation pane, choose **Database migration tasks**.
  3. Choose **Create task** and enter the following information:
    - For **Task configuration**, choose the following settings:
      - **Task identifier** — enter a name that's easy to remember, for example `my-dms-task`.
      - **Replication instance** — choose the replication instance that you created in [Create an AWS DMS replication instance \(p. 168\)](#).
      - **Source database endpoint** — choose the source endpoint that you created in [Create source and target endpoints \(p. 169\)](#).
      - **Target database endpoint** — choose the target endpoint that you created in [Create source and target endpoints \(p. 169\)](#).
      - **Migration type** — choose **Migrate existing data**.
    - For **Task settings**, choose the following settings:
      - **Target table preparation mode** — Do nothing
      - **Include LOB columns in replication** — Limited LOB mode
      - **Maximum LOB size (KB)** — 32
      - **Enable validation**
      - **Enable CloudWatch logs**
- Note**  
CloudWatch logs usage will be charged at standard rates. See [here](#) for more details.
- For **Advanced task settings**, keep all of the options at their default values.
  - For **Premigration assessment**, keep the option at its default value.
  - For **Start migration task in Migration task startup configuration**, choose **Automatically on create**.
  - For **Tags**, keep all of the options at their default values.

When the settings are as you want them, choose **Create task**.

AWS DMS now begins migrating data from MongoDB to Amazon DocumentDB. The task status changes from **Starting** to **Running**. You can monitor the progress by choosing **Tasks** in the AWS DMS console. After several minutes, the status changes to **Load complete**.

**Note**

After the migration is complete, you can use the mongo shell to connect to your Amazon DocumentDB cluster and view the `zips` data. For more information, see [Access your Amazon DocumentDB cluster using the mongo shell](#) in the *Amazon DocumentDB Developer Guide*.