AWS Database Migration Service

Step-by-Step Migration Guide

API Version 2016-01-01

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

Table of Contents

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. 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)

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, 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, 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 Using AWS Database Migration Service

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 (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 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 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 .

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.

Topic: Migration from On-Premises Oracle to Aurora or MySQL on Amazon RDS
Time:
Cost:
Source Database: Oracle
Target Database: Amazon Aurora/MySQL

Topic: Migration from On-Premises Oracle to Aurora or MySQL on Amazon RDS	

Restrictions:

Oracle Edition: Enterprise, Standard, Express and Personal

Oracle Version: 10g (10.2 and later), 11g, 12c, (On Amazon Relational Database Service (Amazon RDS), 11g or higher is required.)

MySQL or Related Database Version: 5.5, 5.6, 5.7, MariaDB, Amazon Aurora

Character Set: utf8mb4 is not currently supported

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 RDS Aurora 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	

Assuming, you run this tutorial for 2 hours, following is an estimate of pricing:

Migration High-Level Outline

To migrate your data from Oracle to Aurora 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 Target Database

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

- For best results, we recommend that you locate your Aurora 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 databse. 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.
- 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
- 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 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 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, 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. 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. 9)

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

1. 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. 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. 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

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
Name	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.
Description	A good description gives others an idea of what the replication instance is being used for and can prevent accidents.
Instance class	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. Multple 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.
VPC	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 outside of this VPC, modify your firewall rules to allow AWS DMS access.
Multi-AZ	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

For This Parameter	Do This
	time, you might consider running your migration server in a Muti-AZ configuration.
Publicly accessible	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 following parameters, and then choose **Next**.

For This Option	Do This
Allocated storage (GB)	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: • Very large tables that incur a significant transaction load. Loading a large table can take some time, so cached transactions are more likely to be written to disk during a large table load. • Tasks that are configured to pause prior to loading cached transactions. 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. • Tasks configured with tables being loaded into Amazon Redshift. However, this configuration isn't an issue when Aurora is the target. 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.
Replication Subnet Group	If you run in a Multi-AZ configuration, you need at least two subnet groups.
Availability Zone	If possible, locate your primary replication server in the same Availability Zone as your target database.
VPC Security group(s)	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.
KMS master key	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 using the AWS console

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, or 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*	e.g. ProdEndpoint
Source engine*	- Select One -
Server name*	
Port*	
SSL mode*	- Select One -
User name*	
Password*	

▸ Advanced

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

For This Parameter	Do This
Endpoint type	Choose **Source**.
Endpoint Identifier	Type an identifier for your Oracle endpoint. The identifier for your endpoint must be unique within an AWS Region.
Source Engine	Choose **oracle**.
Server name	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

For This Parameter	Do This
	on Amazon Elastic Compute Cloud (Amazon EC2) or Amazon RDS, type the public Domain Name Service (DNS) address.
Port	Type the port which your database is listening for connections (the Oracle default is 1521).
SSL mode	Choose a Secure Sockets Layer (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.
Username	Type the AWS account user name. We recommend that you create an AWS account specific to your migration.
Password	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
Extra connection attributes	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 (;).
	• **addSupplementalLogging:** AWS DMS will automatically add supplemental logging if you enable this option (addSupplementalLogging=Y).
	• **useLogminerReader:** 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).
	• **numberDataTypeScale:** 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.
	• **archivedLogDestId:** 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.
KMS master key	Choose the encryption key to use to encrypt replication storage and connection information. If you choose **(Default) aws/dms**, the default 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 Target Endpoint

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

To specify a target database endpoints using the AWS Management Console

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, or 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*	e.g. ProdEndpoint
Source engine*	- Select One -
Server name*	
Port*	
SSL mode*	- Select One -
User name*	
Password*	

▸ Advanced

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

For This Parameter	Do This
Endpoint type	Choose **Target**.
Endpoint Identifier	Type an identifier for your Aurora endpoint. The identifier for your endpoint must be unique within an AWS Region.
Target Engine	Choose **aurora**.

For This Parameter	Do This
Servername	Type the writer endpoint for your Aurora instance. The writer endpoint is the primary instance.
Port	Type the port assigned to the instance.
SSL mode	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.
Username	Type the user name for the account you are using for the migration. We recommend that you create an account specific to your migration.
Password	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
Extra connection attributes	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 (;). • **targetDbType:** 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 specific_database (targetDbType=SPECIFIC_DATABASE). • **initstmt:** You use this option to invoke the MySQL initstmt connection parameter and accept anything mysql initstmt accepts. When working with an Aurora target, it's often useful to disable foreign key checks. To do so, use the initstmt parameter as follows: `initstmt=SET FOREIGN_KEY_CHECKS=0`
KMS master key	Choose the encryption key to use to encrypt replication storage and connection information. If you choose **(Default) aws/dms**, 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

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
Task name	It's always a good idea to give your task a descriptive name that helps organization.
Task description	Type a description for the task.
Source endpoint	Select your source endpoint.
Target endpoint	Select your target endpoint.
Replication instance	Select a replication instance on which to run the task. Remember, your source and target endpoints must be accessible from this instance.
Migration type	You can use three different migration types with AWS DMS. • **Migrate existing data:** 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. • **Migrate existing data and replicate ongoing changes:** 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: • Shut the application down

For This Option	Do This
	• Let the final change flow through to the target • Perform any administrative tasks such as enabling foreign keys and triggers • Start the application pointing to the new target database **Note** AWS DMS loads the bulk data table-by-table, \<n\> 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. • **Replicate data changes only:** In some cases you might choose to load bulk data using a different method. This approach generally only applies to homogeneous migrations.
Start task on create	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.

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

For This Option	Do This
Target table preparation mode	AWS DMS allows you to specify how you would like your target tables prepared prior to loading. **Do nothing** - 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. **Drop tables on target** - 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. **Truncate** - 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.
Include LOB columns in replication	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.

For This Option	Do This
	Don't include LOB columns - 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. **Full LOB mode** - In **full LOB mode** 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. **Limited LOB mode** - In **limited LOB mode**, 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 **limited LOB mode** you get significant performance gains over **full LOB mode**. We recommend that you use **limited LOB mode** whenever possible. **Note** 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.
Max LOB size (K)	When a task is configured to run in **limited LOB mode**, 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.
LOB chunk size (K)	When a task is configured to use **full LOB mode**, 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.
Custom CDC start time	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.
Enable logging	Always enable logging.

5. Set additional parameters.

For This Option	Do This
Create control table(s) in target schema	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 puts those artifacts somewhere else.
Maximum number of tables to load in parallel	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",
      "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 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:

http://docs.aws.amazon.com/dms/latest/userguide/
CHAP_Troubleshooting.html#CHAP_Troubleshooting.Oracle

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

http://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 are part of the .tar file located here: https://github.com/awslabs/aws-database-migration-samples.

To build the sample database, extract the .tar file and follow the instructions in the README and install files.

The sample includes approximately 8-10 GB of data. The sample database also includes the ticketManagment 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>exec  ticketManagement.generateTicketActivity(0.01,1000);
```

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>exec ticketManagement.generateTransferActivity(1,100);
```

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 Using AWS Database Migration Service

This walkthrough gets you started with heterogeneous database migration from Amazon RDS Oracle to Amazon Aurora 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 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 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. 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.

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

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 RDS Aurora 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 + 10 GB storage pricing estimated by using the AWS Simple Monthly Calculator is $1.78.

- Amazon RDS Oracle SE2 (license included) + 10 GB GP2 storage cost, estimated as per the pricing site at ($0.226) * 2 hours + ($0.115) * 10 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 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 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. The scripts and queries are available at the following links. Each step in the walkthrough also contains a link to download the file or includes the exact query in the step.
  - SQL statements to build the HR schema— https://dms-sbs.s3.amazonaws.com/Oracle-HR-Schema-Build.sql.
  - SQL queries to validate the schema contents — (text) https://dms-sbs.s3.amazonaws.com/AWSDMSDemoStats.txt and (spreadsheet) https://dms-sbs.s3.amazonaws.com/AWSDMSDemoStats.xlsx.
  - AWS CloudFormation template — https://dms-sbs.s3.amazonaws.com/Oracle_Aurora_RDS_For_DMSDemo.template.
- 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. 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 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. 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 (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

- Aurora RDS 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.
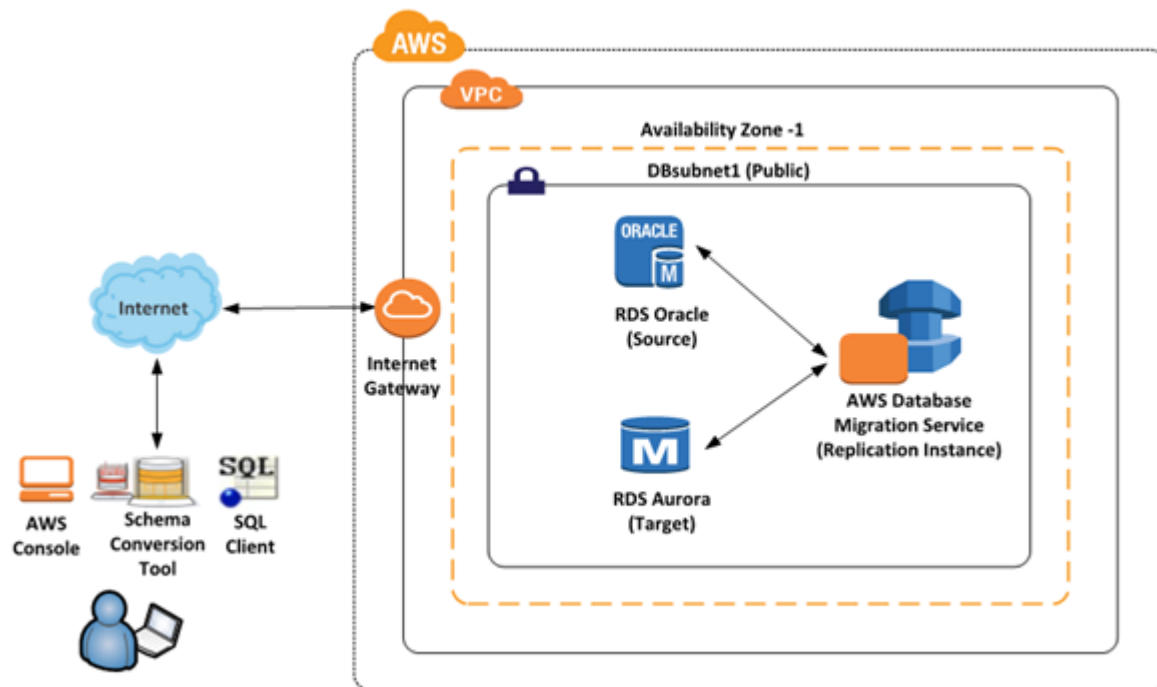
Step-by-Step Migration

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. 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

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

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

To use AWS CloudFormation to create Amazon RDS 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 **Specify an Amazon S3 template URL** and paste the following URL into the adjacent text box:

   https://dms-sbs.s3.amazonaws.com/Oracle_Aurora_RDS_For_DMSDemo.template.

4. Choose **Next**. On the **Specify Details** page, provide parameter values as shown following.

For This Parameter	Do This
Stack Name	Type **DMSdemo**.
OracleDBName	Provide any unique name for your database. The name should begin with a letter. The default is ORCL.
OracleDBUsername	Specify the admin (DBA) user for managing the Oracle instance. The default is oraadmin.
OracleDBPassword	Provide the password for the admin user.
AuroraDBUsername	Specify the admin (DBA) user for managing the Aurora instance. The default is auradmin.

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
AuroraDBPassword	Provide the password for the admin user.
ClientIP	Specify the IP address in CIDR (x.x.x.x/32) format for your local computer. You can get your IP address from whatsmyip.org. 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.

Specify Details

Specify a stack name and parameter values. You can use or change the default parameter values, which are defined in the AWS CloudFormation template. Learn more.

Stack name DMSdemo

Parameters

Source Oracle Database Configuration

OracleDBName ORCL   Enter Oracle Database name

OracleDBUsername oraadmin   Enter database Admin username for Oracle

OracleDBPassword •••••••••   Enter password for Oracle Admin user

Target Aurora Database Configuration

AuroraDBUsername auradmin
Enter database Admin username for Aurora RDS

AuroraDBPassword •••••••••   Enter password for Aurora Admin user

Enter IP address for DB Security group Configuration

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

5.   Choose **Next**. On the **Options** page, shown following, choose **Next**.

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

Options

Tags

You can specify tags (key-value pairs) for resources in your stack. You can add up to 10 unique key-value pairs for each stack. Learn more.

	Key (127 characters maximum)	Value (255 characters maximum)	
1			+

▶ **Advanced**

You can set additional options for your stack, like notification options and a stack policy. Learn more.

Cancel   Previous   Next

6. On the **Review** page, review the details, and if they are correct choose **Create Stack**. You can get the estimated cost of running this CloudFormation template by choosing **Cost**.

Review

Template

Template URL	https://s3-us-west-2.amazonaws.com/awsdmsdemo /Oracle_Aurora_RDS_For_DMSDemo.template
Description	This CloudFormation sample template Oracle_Aurora_RDS_In_VPC creates a Oracle and Aurora RDS instances in a VPC which can be used to test the database migration using AWS DMS service. You will be billed for the AWS resources used if you create a stack from this template
Estimate cost	Cost

Details

Stack name   DMSdemo

Source Oracle Database Configuration

OracleDBName   ORCL
OracleDBUsername   oraadmin
OracleDBPassword   ·····

Target Aurora Database Configuration

AuroraDBUsername   auradmin
AuroraDBPassword   ·····

Enter IP address for DB Security group Configuration

ClientIP

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

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

Create Stack	Actions ▼	Design template			C

Filter: Active ▼   By Name: ☐   Showing 66 sta

	Stack Name	Created Time	Status	Description
☑	DMSdemo	2016-08-18 14:57:51 UTC-0700	CREATE_IN_PROGRESS	This CloudFormation sample templat
☐		2016-08-10 13:56:20 UTC-0700	CREATE_COMPLETE	
☐		2016-08-09 13:51:47 UTC-0700	CREATE_COMPLETE	
☐		2016-08-04 15:06:43 UTC-0700	ROLLBACK_COMPLETE	
☐		2016-08-01 10:56:18 UTC-0700	CREATE_COMPLETE	
☐		2016-07-11 10:07:14 UTC-0700	CREATE_COMPLETE	
☐		2016-07-01 17:24:20 UTC-0700	CREATE_COMPLETE	
☐		2016-07-01 17:18:56 UTC-0700	CREATE_COMPLETE	
☐		2016-07-01 17:10:18 UTC-0700	CREATE_COMPLETE	

Events ▼

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

8. 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 DB instances.

Note
Oracle 12c SE Two License version 12.1.0.2.v4 is available in all regions. However, Amazon
Aurora is not available in all regions. Amazon Aurora 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
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) over JDBC
• DBVisualizer
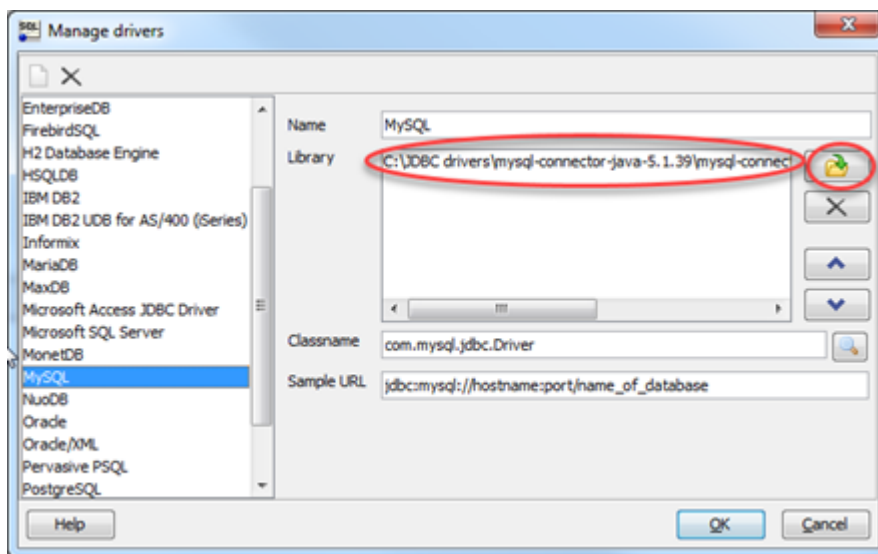• Oracle SQL Developer

To install the SQL client software

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 Oracle Database 12.1.0.2 JDBC driver (ojdbc7.jar).

3. Download the MySQL driver (mysql-connector-java-5.1.39-bin.jar).

4. Using SQL Workbench/J, configure JDBC drivers for Oracle and Aurora to set up connectivity, as described following.

   1. In SQL Workbench/J, choose **File**, then choose **Manage Drivers**.

   2. From the list of drivers, choose **Oracle**.

   3. Choose the Open icon, then choose the `ojdbc.jar` file that you downloaded in the previous step. Choose **OK**.
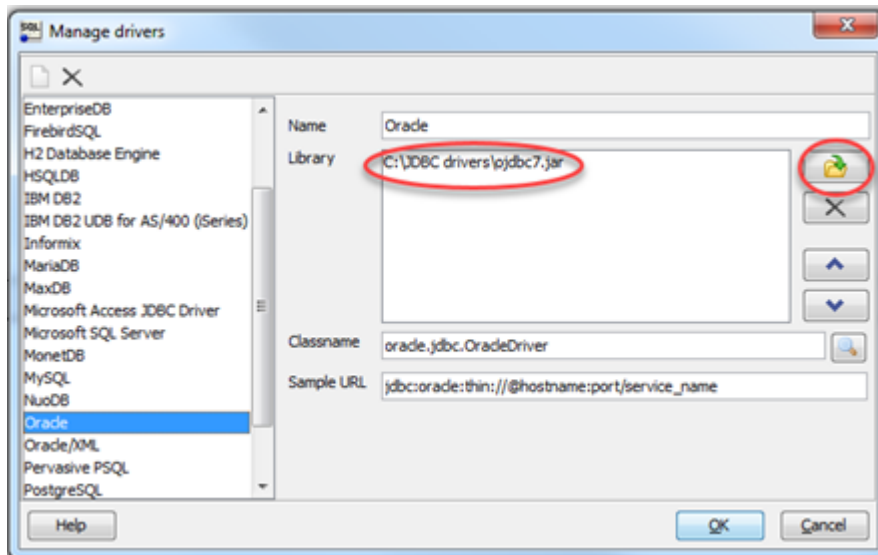

   4. From the list of drivers, choose MySQL.

   5. Choose the Open icon, then choose the MySQL JDBC driver that you downloaded in the previous step. Choose **OK**.

Next, install the AWS Schema Migration Tool and the required JDBC drivers.

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 JDBC drivers

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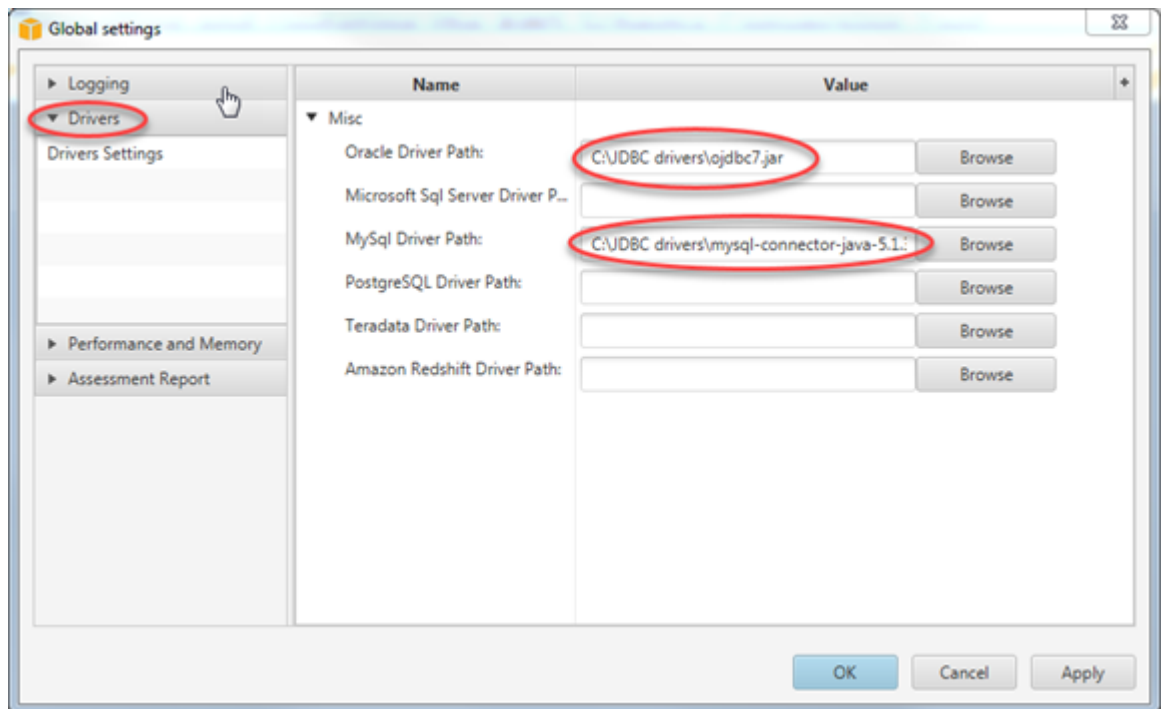
Step 3: Test Connectivity to the Oracle DB Instance and Create the Sample Schema

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 using SQL Workbench/J and create the sample schema

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
URL	Use the **OracleJDBCConnectionString** value you recorded when you examined the output details of the DMSdemo stack in a previous step.
Username	Type `oraadmin`.
Password	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.
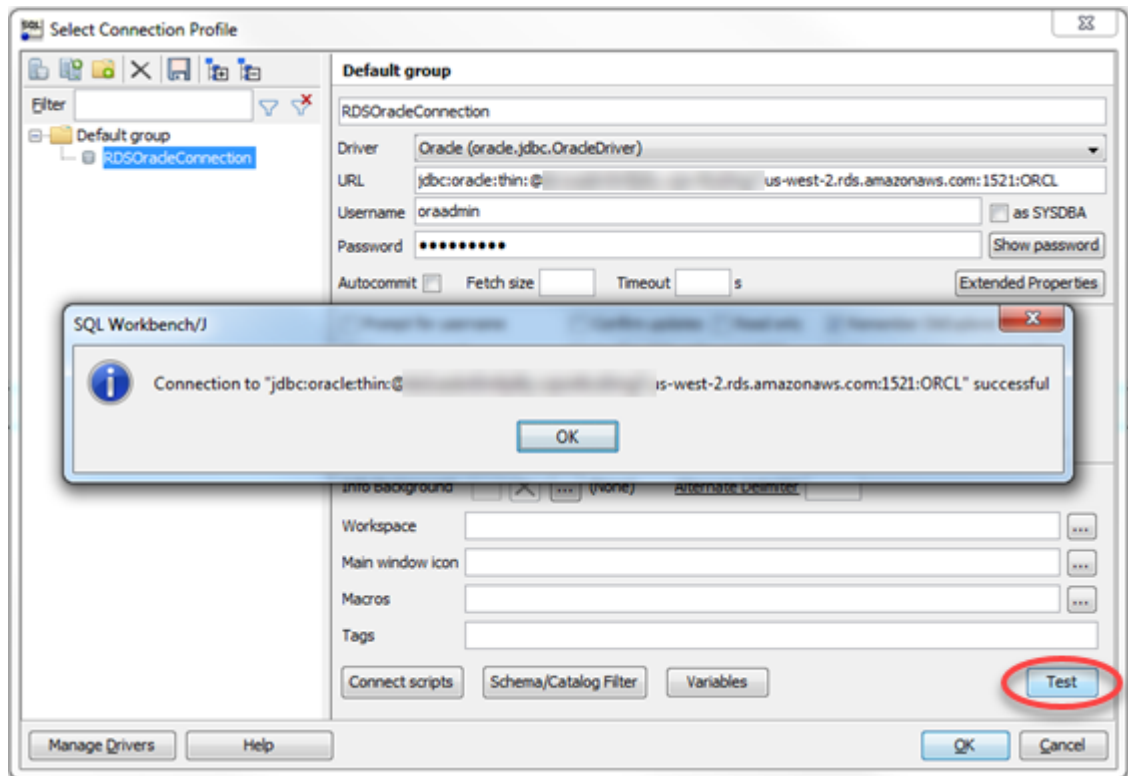
> **Note**
> If your connection is unsuccessful, ensure that the IP address you assigned when creating the CloudFormation template is the one you are attempting to connect from. This is the most common issue when trying to connect to an instance.

3. Create the HR schema you will use for migration using a custom script. The SQL script provided by AWS is located at this site.

1. Open the provided SQL script in a text editor. Copy the entire script.

2. In SQL Workbench/J, paste the SQL script in the Default.wksp window showing **Statement 1**.

3. Choose **SQL**, then choose **Execute All**.

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

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

4. Verify the object types and count in HR Schema were created successfully by running the following SQL query. You can also compare the results from the following queries with the results listed in the spreadsheet provided by AWS at this site.

```
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 7
 PROCEDURE 2
 SEQUENCE 3
 TABLE 7
 VIEW 1
```

AWS Database Migration Service
Step-by-Step Migration Guide
Step 3: Test Connectivity to the Oracle DB
Instance and Create the Sample Schema
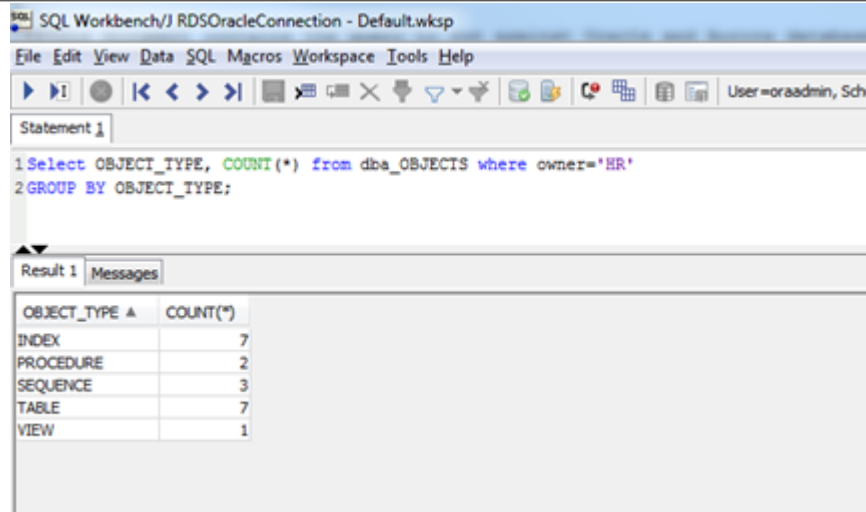
5. Verify the number of constraints in HR schema by running the following SQL query:

```
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             2
```

6. 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
```

7. Verify the relationship in tables. Check the departments with employees greater than 10 by running the following SQL query:

```
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 DB Instance

Next, test your connection to your Aurora DB instance.

To test the connection to your Aurora DB instance using SQL Workbench/J

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 DB instance in SQL Workbench/J by using the information as shown following

For This Parameter	Do This
New profile name	Type `RDSAuroraConnection`.
Driver	Choose `MySQL (com.mysql.jdbc.Driver)`.
URL	Use the **AuroraJDBCConnectionString** value you recorded when you examined the output details of the DMSdemo stack in a previous step.
Username	Type `auradmin`.
Password	Provide the password for the admin user that you assigned when creating the Aurora 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. Log on to the Aurora instance by using the master admin credentials.

4. Verify your connectivity to the Aurora DB instance by running a sample SQL command, such as `SHOW DATABASES;`.
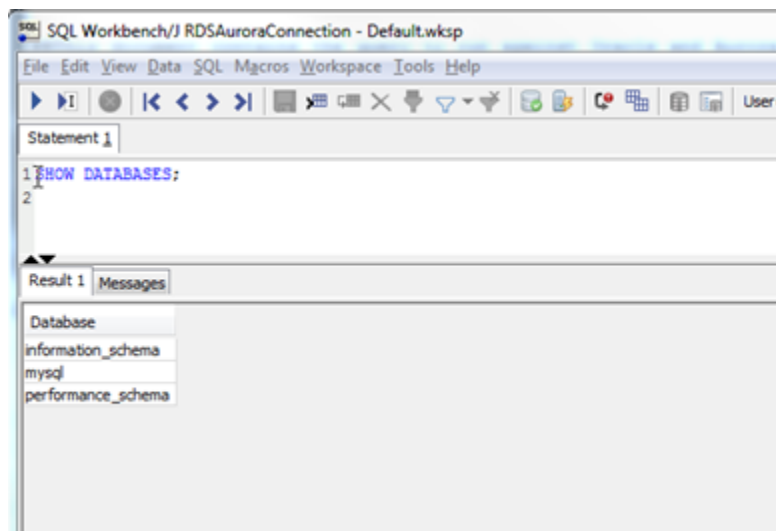
Step 5: Use the AWS Schema Conversion Tool (AWS SCT) to Convert the Oracle Schema to Aurora

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

To convert an Oracle schema to an Aurora schema using AWS Schema Conversion Tool (AWS SCT)

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
Project Name	Type `DMSDemoProject`.
Location	Use the default **Projects** folder and the default **Transactional Database (OLTP)** option.
Source Database Engine	Choose **Oracle**.
Target Database Engine	Choose **Amazon Aurora**.
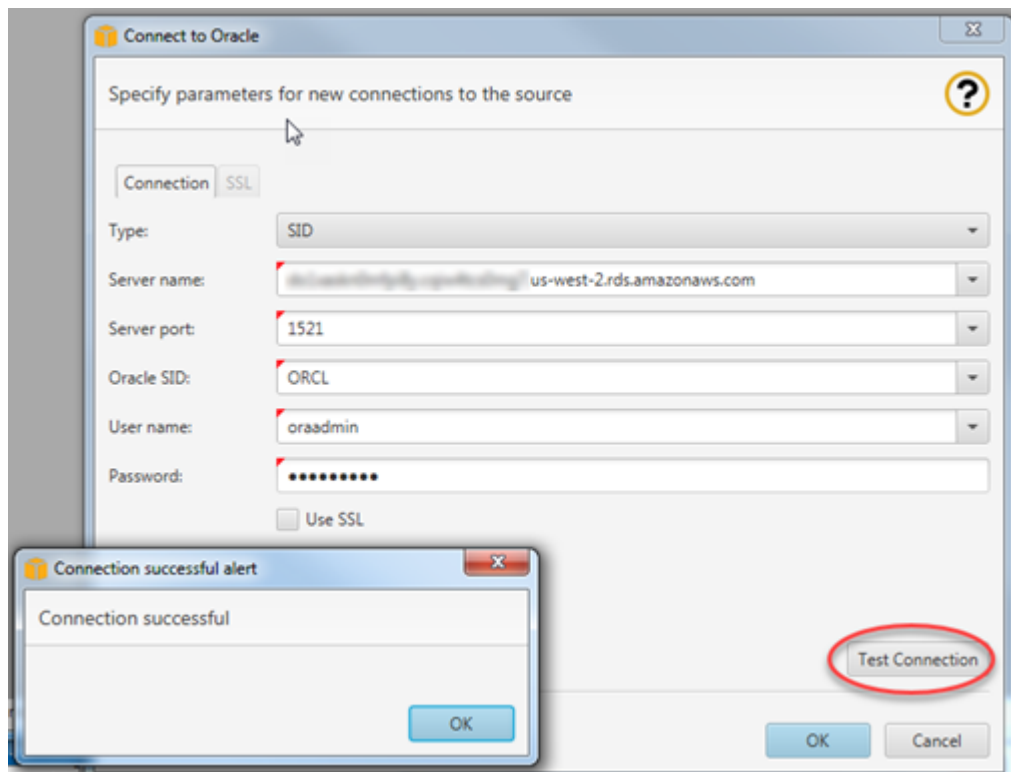
2. Choose **Connect to Oracle**. In the **Connect to Oracle** dialog box, enter the following information, and then choose **Test Connection**.

For This Parameter	Do This
Type	Choose **SID**.
Server name	Use the **OracleJDBCConnectionString** value you used to connect to the Oracle DB instance, but remove the JDBC prefix information. For example, a sample connection string 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 **Server name**, you remove "jdbc:oracle:thin:@" and use just the server name: "do1xa4grferti8y.cqiw4tcs0mg7.us-west-2.rds.amazonaws.com"
Server port	Type `1521`.
Oracle SID	Type `ORCL`.
User name	Type `oraadmin`.
Password	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 on the Oracle DB instance is shown. Select only the HR schema.
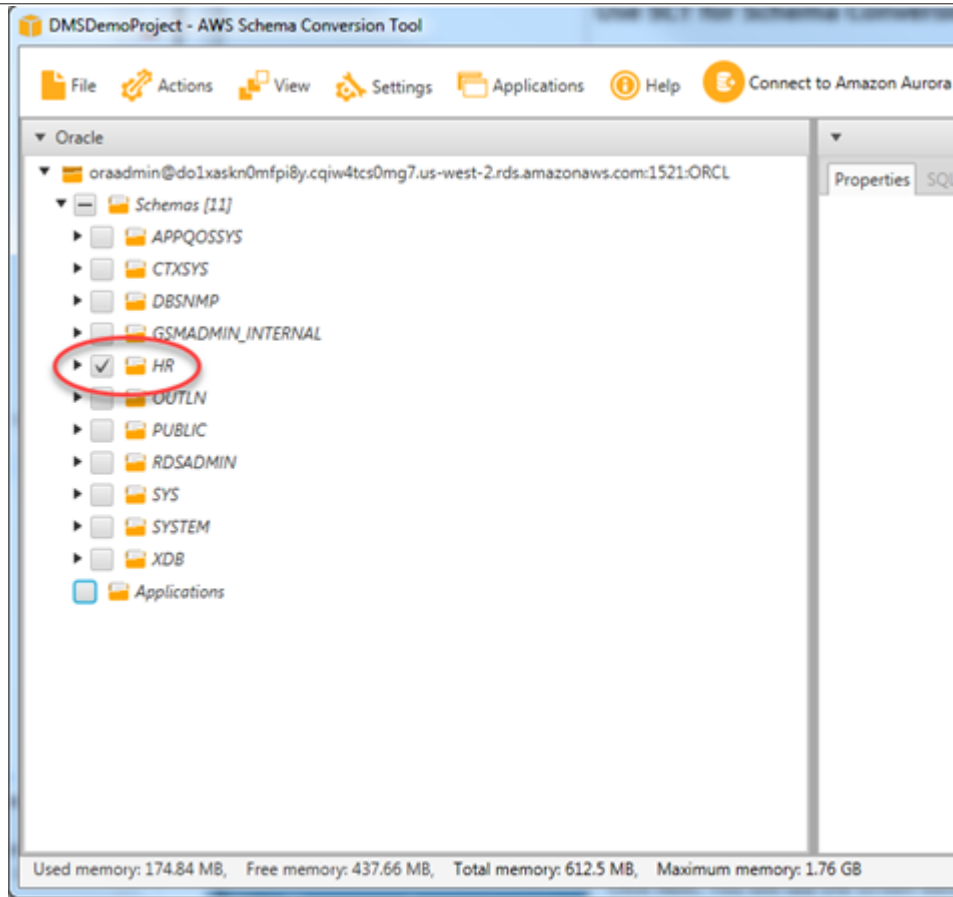
4. Choose **Connect to Amazon Aurora**. In the **Connect to Amazon Aurora** dialog box, enter the following information and then choose **Test Connection**.

For This Parameter	Do This
Type	Choose **SID**.
Server name	Use the **AuroraJDBCConnectionString** value you used to connect to the Aurora 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 **Server name**, you remove "jdbc:oracle:thin:@" and ":3306" to use just the server name: "dmsdemo-auroracluster-1u1ogdfg35v.cluster-cqiw4tcs0mg7.us-west-2.rds.amazonaws.com"
Server port	Type `3306`.
User name	Type `auradmin`.
Password	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.

5. Choose **OK** to close the alert box, then choose **OK** to close the dialog box to start the connection to the Amazon Aurora DB instance.

6. Right-click the HR schema and select **Create Report**.
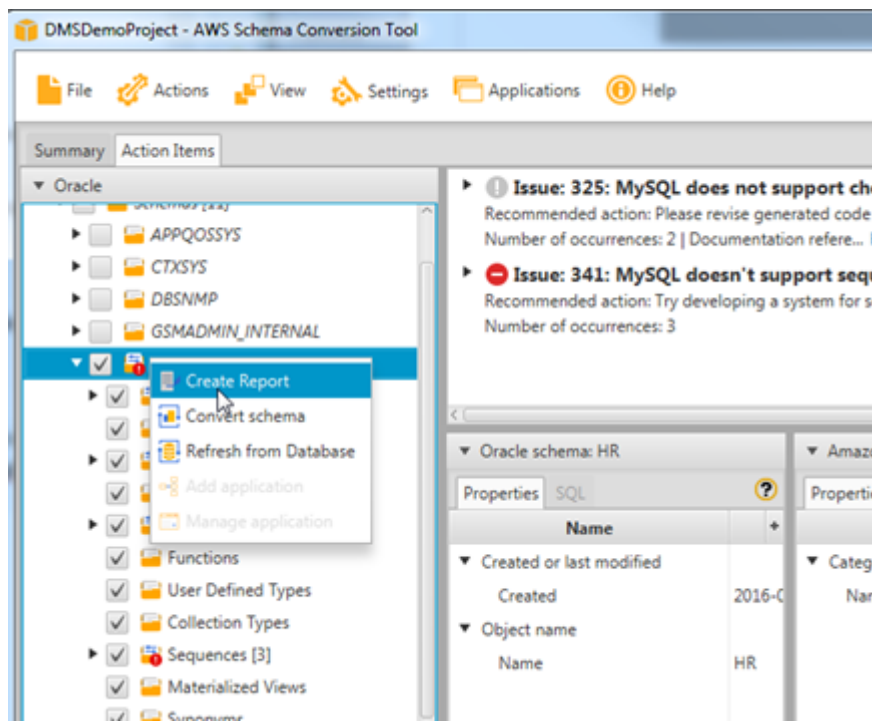
7. Check the report and the action items it suggests. The report discusses the type of objects that can be converted by using AWS SCT, along with potential migration issues and actions to resolve these issues. For this walkthrough, you should see something like the following.
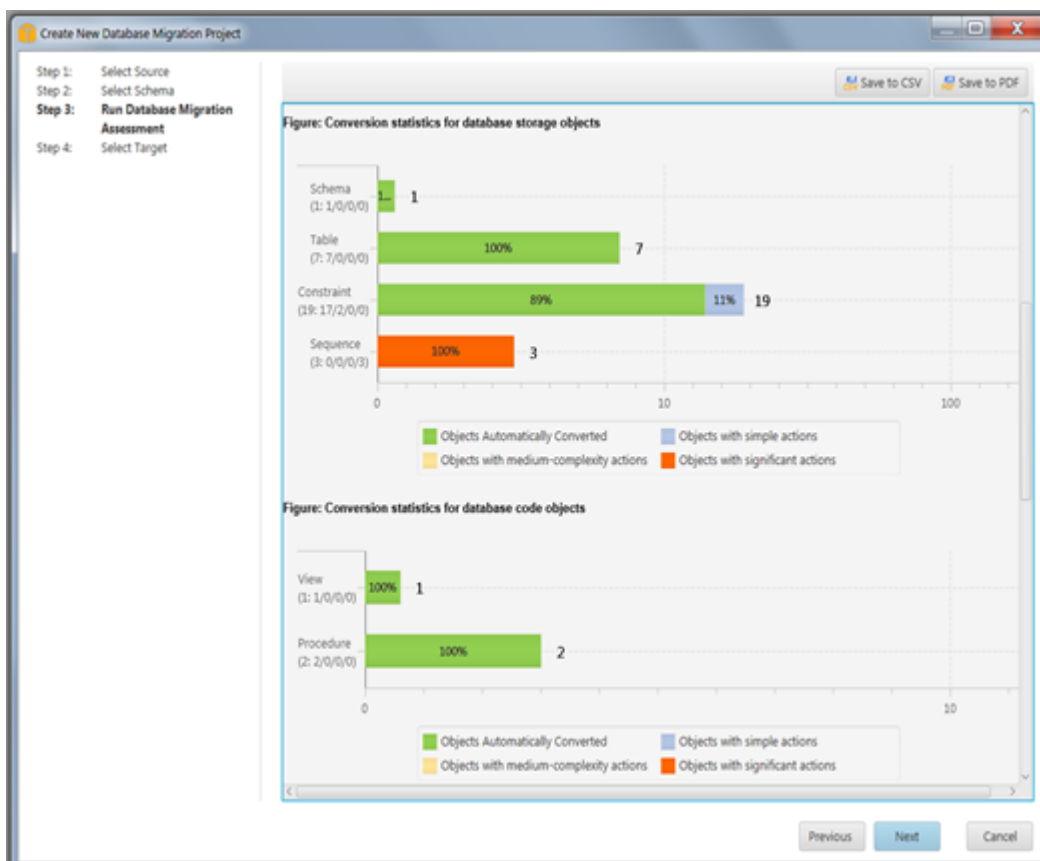
8. Save the report as .csv or .pdf format for detailed analysis, and then choose the **Action Items** tab. In the action items, you will see two issues: 1. MySQL does not support Check constraints and 2. MySQL does not support Sequences.

Regarding action item #1, SCT automatically provisions triggers to simulate check constraints in Aurora database (Emulating triggers). For example, a check constraint for SAL > 0 in the EMPLOYEES table (in Oracle) is enforced with the help of before and update trigger statements in Aurora. If you would like to have this logic handled at the application layer, then you can drop or update the triggers if required.

Regarding action item #2, there are three sequence objects in the source database that are used to generate primary keys for the EMPLOYEES (EMPLOYEE_ID), DEPARTMENTS (DEPARTMENT_ID), LOCATIONS (LOCATION_ID) tables. As mentioned earlier in this walkthrough, one alternative to using sequences for Surrogate keys in Aurora is using the auto_increment feature. To enable the auto_increment feature, you must change the settings for SCT. For brevity, the following substeps show enabling auto_increment for EMPLOYEE_ID column in the EMPLOYEES table only. The same procedure can be repeated for the other sequence objects.

Before starting, please note enabling the auto_increment option requires some additional steps via SCT due to the below reasons:

- SCT by default converts all NUMBER (Oracle) data types into DECIMAL in Aurora (http://docs.aws.amazon.com/SchemaConversionTool/latest/userguide/CHAP_SchemaConversionTool.Reference.ConversionSupport.Oracle.html#d0e50104).

- Aurora doesn't support auto_increment for the DECIMAL data type. Therefore, the data type of the primary key column and corresponding foreign key columns needs to be changed to one of the INTEGER data types such as INT, SMALLINT, MEDIUMINT or BIGINT as part of the schema conversion.

The good news is that the latest release of SCT provides a **Mapping Rules** feature that can be used to achieve the above transformation using the following steps:

1. For the EMPLOYEES table, you must identify the primary key and foreign key relationships by running the following query on the source Oracle database. Note the columns that need to be specified in the SCT Mapping rules.

```
SELECT * FROM
(SELECT
 PK.TABLE_NAME,
 C.COLUMN_NAME,
 PK.CONSTRAINT_TYPE
     FROM DBA_CONSTRAINTS PK,
   DBA_CONS_COLUMNS C
     WHERE PK.CONSTRAINT_NAME = C.CONSTRAINT_NAME
     AND PK.OWNER = 'HR' AND PK.TABLE_NAME = 'EMPLOYEES' AND
 PK.CONSTRAINT_TYPE = 'P'
UNION
 SELECT
 FK.TABLE_NAME,
 COL.COLUMN_NAME,
 FK.CONSTRAINT_TYPE
     FROM DBA_CONSTRAINTS PK,
   DBA_CONSTRAINTS FK,
   DBA_CONS_COLUMNS COL
   WHERE PK.CONSTRAINT_NAME = FK.R_CONSTRAINT_NAME
     AND FK.CONSTRAINT_TYPE = 'R'
   AND FK.CONSTRAINT_NAME = COL.CONSTRAINT_NAME
 AND PK.OWNER = 'HR' AND PK.TABLE_NAME = 'EMPLOYEES' AND
 PK.CONSTRAINT_TYPE = 'P' )
 ORDER BY 3 ASC;
```

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

```
TABLE_NAME COLUMN_NAME CONSTRAINT_TYPE
EMPLOYEES EMPLOYEE_ID P
JOB_HISTORY EMPLOYEE_ID R
EMPLOYEES MANAGER_ID R
DEPARTMENTS MANAGER_ID R
```

2. Choose **Settings**, and then choose **Mapping Rules**.

3. Specify the Mapping rule for Data type conversions for the list of identified columns in Step1. You will need to specify 4 rules, one for each column as described below.

For this Parameter	Rule1	Rule2	Rule3	Rule4
Name	EMP_SEQ1	EMP_SEQ2	JOB_SEQ1	DEPT_SEQ1
For	Select Column	Select Column	Select Column	Select Column

For this Parameter	Rule1	Rule2	Rule3	Rule4
Where (Schema Name) And (Table Name) And (Column Name)	HR EMPLOYEES EMPLOYEE_ID	HR EMPLOYEES MANAGER_ID	HR JOB_HISTORY EMPLOYEE_ID	HR DEPARTMENTS MANAGER_ID
Actions	Select **Change data type**	Select **Change data type**	Select **Change data type**	Select **Change data type**
To	SMALLINT	SMALLINT	SMALLINT	SMALLINT

Note that in a real-world scenario you would choose the data type based on your requirements.
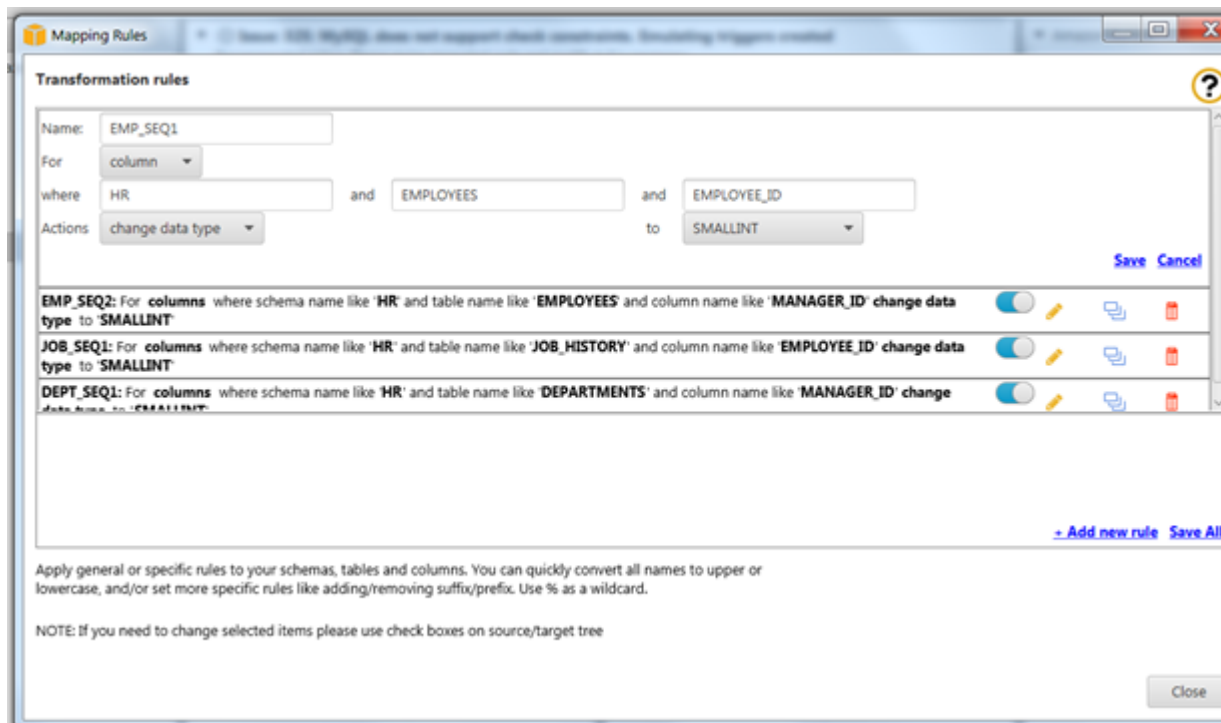
4. Choose **Yes** for "Would you like to save Mapping Rule settings?"

9. Right-click the HR schema, and then choose **Convert schema**.
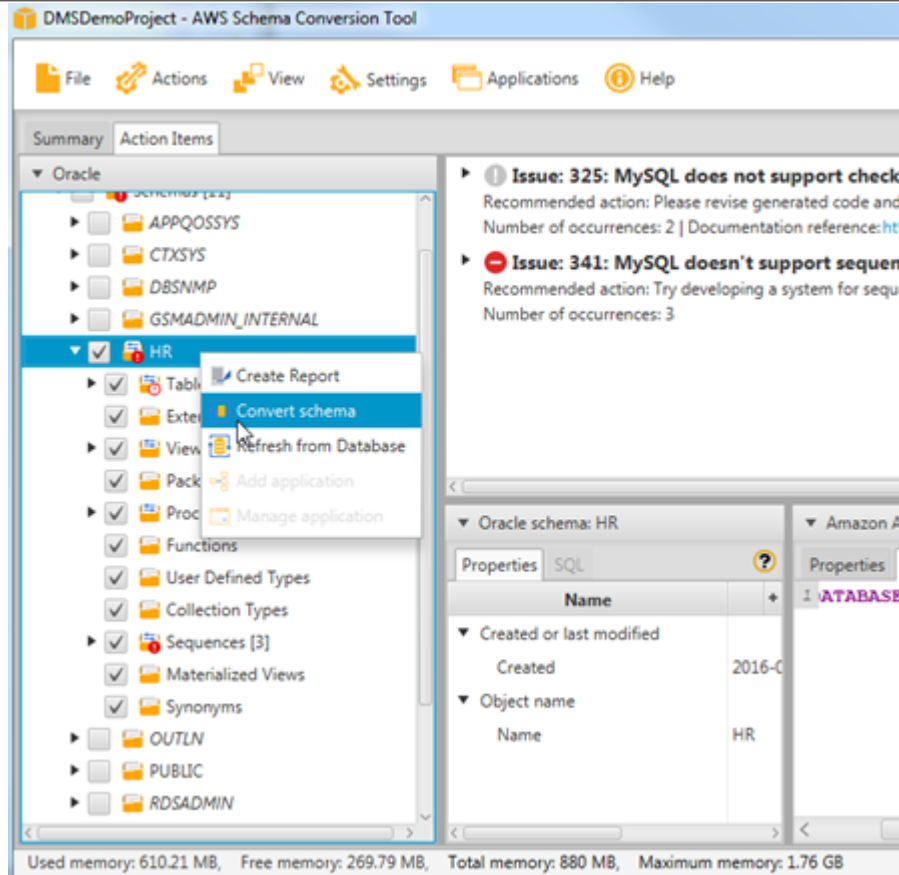
10. Choose **Yes** for the confirmation message. AWS SCT then converts your schema to the target database format.
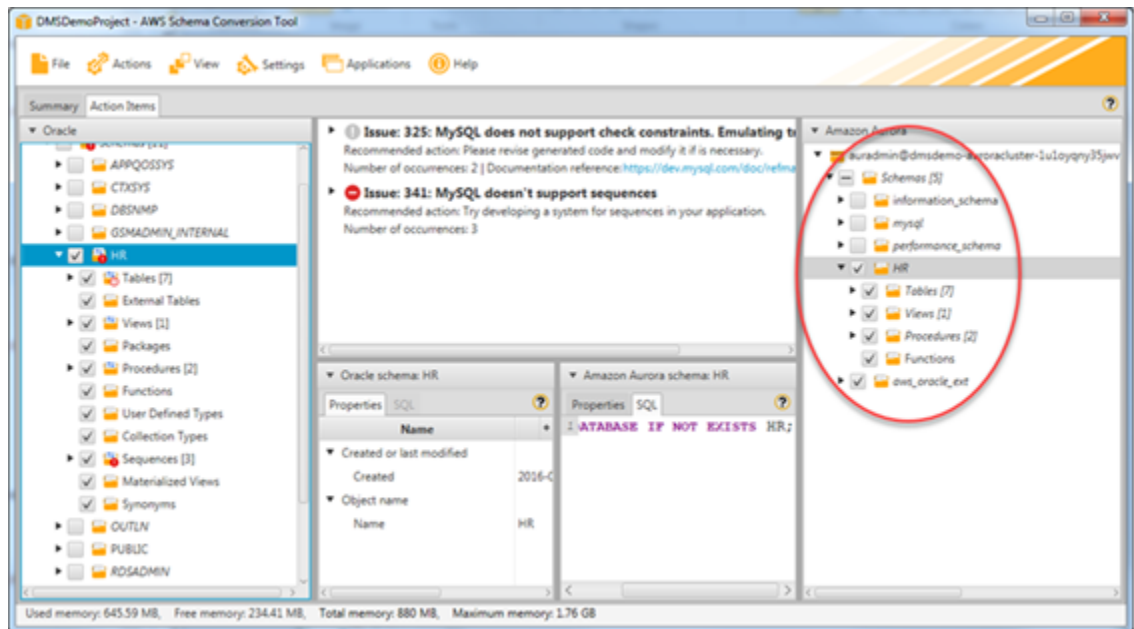
11. Choose the HR schema, and then choose **Apply to database** to apply the schema scripts to the target Aurora instance, as shown following.
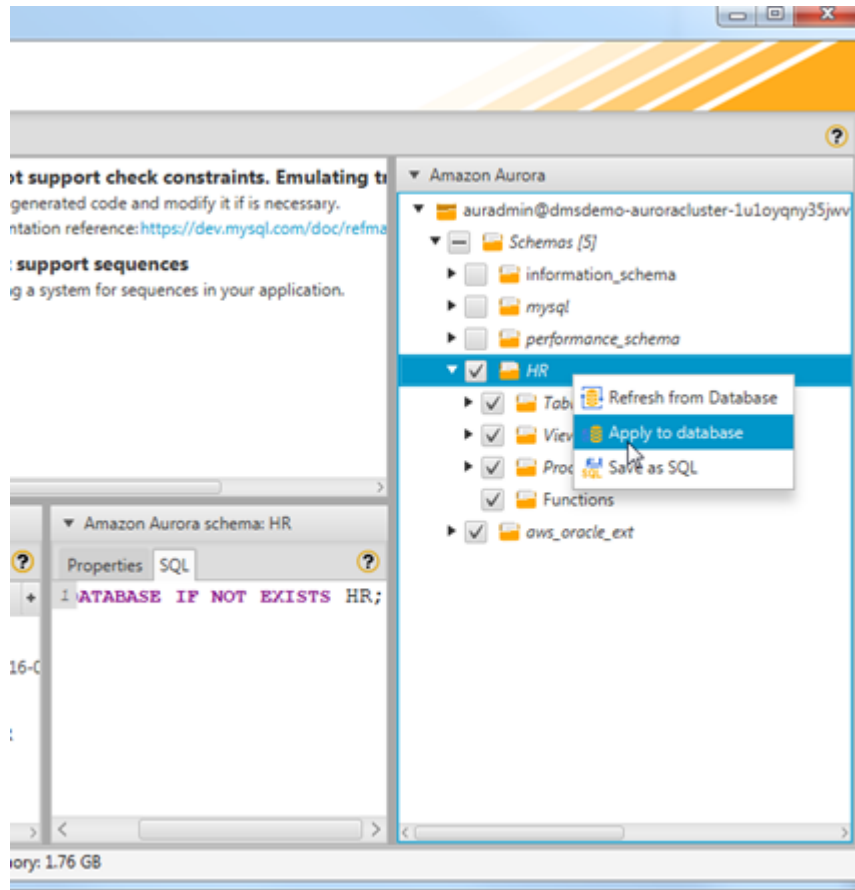
12. Choose the HR schema, and then choose **Refresh from Database** to refresh from the target database, as shown following.
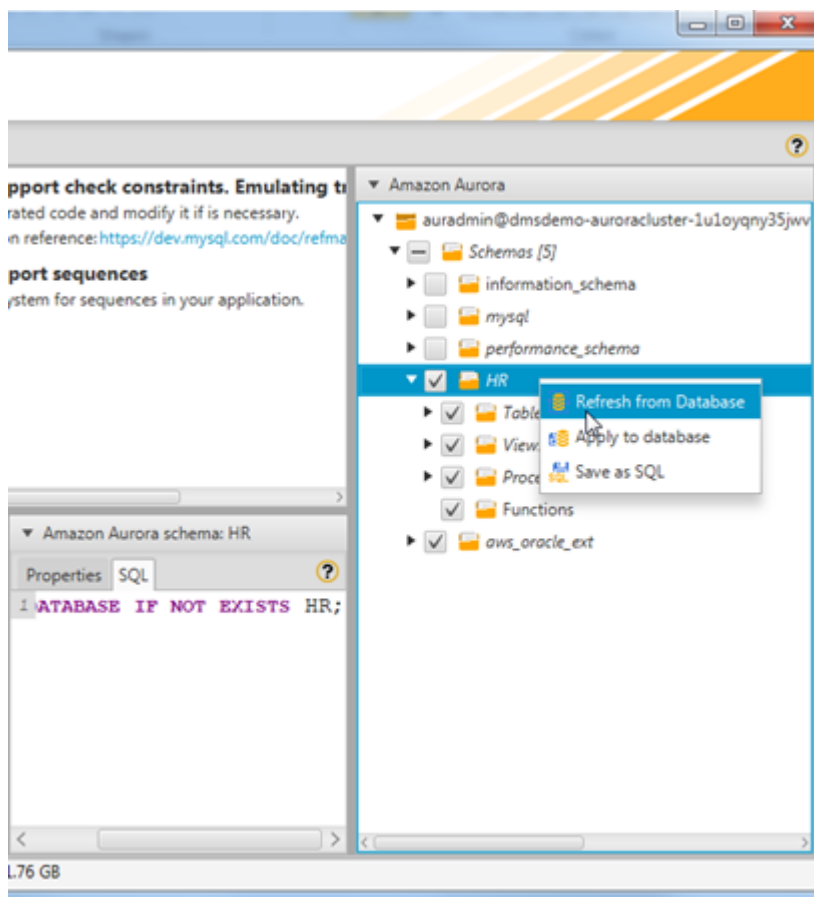
The database schema has now been converted and imported from source to target.

Step 6: Validate the Schema Conversion

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

To validate the schema conversion 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 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
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      4
 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
```

3. Do the following steps to enable the auto_increment option on the EMPLOYEES table to emulate the sequence functionality of the source Oracle database.

   1. Verify that the mapping rules for data type conversion were executed properly for EMPLOYEES and its dependent tables by running the following query on the target Aurora database.

   ```
   SELECT
   kcu.constraint_name,
   kcu.column_name,
   col.data_type,
   kcu.table_schema,
   kcu.table_name,
   kcu.referenced_column_name
   FROM
   information_schema.key_column_usage kcu,
   information_schema.table_constraints tc,
   information_schema.columns col
   WHERE    kcu.referenced_table_schema = 'HR'
   AND      kcu.referenced_table_name = 'EMPLOYEES'
   AND      kcu.referenced_table_name=tc.table_name
   AND      kcu.referenced_table_schema=tc.table_schema
   AND     tc.constraint_type='PRIMARY KEY'
   AND col.column_name=kcu.column_name
   and col.table_name=kcu.table_name
   ORDER BY kcu.table_name,kcu.column_name;
   ```

   The results of the query should be the following:

   ```
   constraint_name column_name  data_type   table_schema table_name
    referenced_column_name
   DEPT_MGR_FK MANAGER_ID  Smallint   HR    DEPARTMENTS EMPLOYEE_ID
   EMP_MANAGER_FK    MANAGER_ID  Smallint   HR    EMPLOYEES EMPLOYEE_ID
   JHIST_EMP_FK     EMPLOYEE_ID  Smallint   HR    JOB_HISTORY EMPLOYEE_ID
   ```

   2. Disable foreign key checks for the EMPLOYEES table by running the following command. This step is required before you can alter the primary key column. You can ignore the warning messages.

```
SET FOREIGN_KEY_CHECKS=0;
```

3. Modify the primary key column to enable the auto_increment option by running the following command:

```
Alter table HR.EMPLOYEES modify column employee_id smallint
 auto_increment;
```

4. Verify the column details by running the following query:

```
SELECT column_name, column_type,column_key,extra
from information_schema.columns
where table_name = 'EMPLOYEES' AND COLUMN_NAME='EMPLOYEE_ID';
```

The results of the query should be the following:

```
column_name column_type column_key extra
employee_id smallint(6) PRI      auto_increment
```

4. The following table shows the expected numbers of objects and whether they were migrated by AWS SCT.

Parameter	# on Oracle	# on Amazon Aurora	Migrated by AWS SCT	SCT Recommendation
INDEX	7	7	Yes	
PROCEDURE	2	2	Yes	
SEQUENCE	3	3	Yes, using mapping rules	Sequence functionality is implemented using the auto_increment feature of Aurora.
TABLE	7	7	Yes	
VIEW	1	1	Yes	
Primary key	10	10	Yes	
Foreign key	7	7	Yes	
Check constraints	2	4 (triggers)	Code conversion	Check constraints are not supported in Aurora. AWS SCT creates triggers before insert or update statements to mimic check constraints in the tables that had these constraints.

5. Validate the results as mentioned in the spreadsheet provided by AWS on this site or the text document provided by AWS on this site.

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

1. Sign in to the AWS Management Console, and 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 **Next** to start a database migration from the console's Welcome page.

3. On the **Create replication instance** page, specify your replication instance information as shown following.

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

4. 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.

To specify source or target database endpoints using the AWS console

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

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

The following table describes the target settings.

For This Parameter	Do This
Endpoint Identifier	Type `Aurtarget` (the Amazon Aurora endpoint).
Target Engine	Choose **aurora**.
Servername	Provide the Aurora DB instance name. This is the **Server name** you used for AWS SCT, such as "dmsdemo-auroracluster-1u1oyqny35jwv.cluster-cqiw4tcs0mg7.us-west-2.rds.amazonaws.com".
Port	Type `3306`.
SSL mode	Choose **None**.
Username	Type `auraadmin`.
Password	Provide the password for the Aurora 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				Target database connection details		
Endpoint identifier*	Orasource	ⓘ		Endpoint identifier*	Aurtarget	ⓘ
Source engine*	oracle ▼	ⓘ		Target engine*	aurora ▼	ⓘ
Server name*	rds.amazonaws.com			Server name*	rds.amazonaws.com	
Port*	1521 ⇅			Port*	3306 ⇅	
SSL mode*	none ▼	ⓘ		SSL mode*	none ▼	ⓘ
User name*	oraadmin			User name*	auradmin	
Password*	•••••••••			Password*	•••••••••	
SID*	ORCL			▸ Advanced		

▸ 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 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.
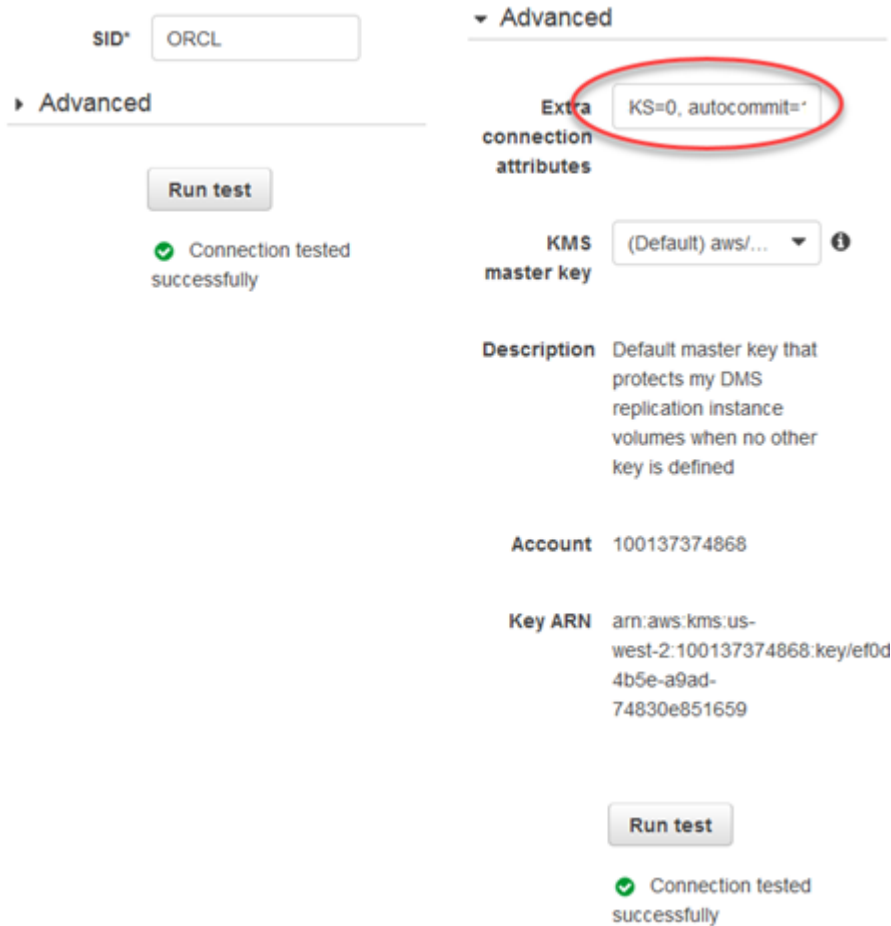
3. Choose **Next**.

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.

To create a migration task

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

For This Parameter	Do This
Task name	Type `migratehrschema`.
Task description	Type a description for the task.
Source endpoint	Shows `orasource` (the Amazon RDS Oracle endpoint).
Target endpoint	Shows `aurtarget` (the Amazon Aurora endpoint).

For This Parameter	Do This
Replication instance	Shows `DMSdemo-repserver` (the AWS DMS replication instance created in an earlier step).
Migration type	Choose the option **Migrate existing data**.
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*	migrateschema
Task description*	migrates HR schema
Source endpoint	orasource
Target endpoint	aurtarget
Replication instance	dmsdemo-repserver
Migration type*	Migrate existing data
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.

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.

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

To monitor a data migration task in progress

1.   On the navigation pane, choose **Tasks**.

2.   Choose your migration task (`migratehrschema`).

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.

To compare your migration task results with the expected results

1.   On the navigation pane, choose **Tasks**.

2.   Choose your migration task (`migratehrschema`).

3.   Choose the **Table statistics** tab, shown following.

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

```
Show databases;
Use HR;
SELECT TABLE_NAME,TABLE_ROWS
    FROM INFORMATION_SCHEMA.TABLES
    WHERE TABLE_SCHEMA = 'HR' and TABLE_TYPE='BASE TABLE' order by 1;
```
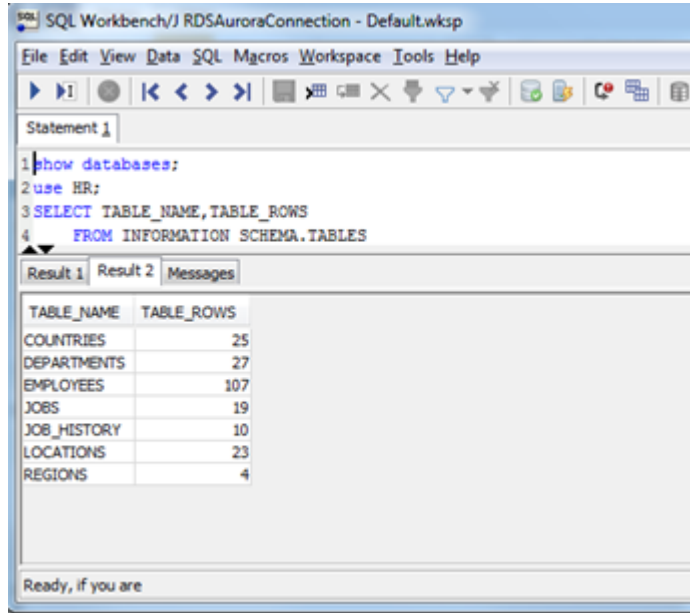
5. To verify whether the output for tables and number of rows from the preceding query matches what is expected for RDS Oracle, compare your results with those in the spreadsheet provided by AWS at this site.

6. 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.

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

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 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`.

To delete your AWS CloudFormation stack

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.

AWS CloudFormation Template, SQL Scripts, and Other Resources

You can find the AWS CloudFormation template, SQL scripts, and other resources used in this walkthrough on the AWS site as listed following:

*   Oracle schema SQL script
*   AWS CloudFormation template
*   SQL validation script, in spreadsheet format
*   SQL validation script, in text format
*   Architecture diagram, in .jpg format or Architecture diagram, in .vsd format
*   MySQL JDBC driver, in .jar file format
*   Oracle Database 12.1.0.2 JDBC driver, in .jar file format

References

The following documentation and sample schemas can be useful as references for this walkthrough:

- AWS DMS documentation
- AWS SCT documentation
- Oracle sample schemas

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. 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 DB cluster. For details, see Migrating Data from an Amazon RDS MySQL DB Instance to an Amazon Aurora DB Cluster (p. 78).
A MySQL database external to Amazon RDS	If your database supports the InnoDB or MyISAM tablespaces, you have these options for migrating your data to an Amazon Aurora DB cluster: • You can create a dump of your data using the `mysqldump` utility, and then import that data into an existing Amazon Aurora DB cluster. • You can copy the source files from your database to an Amazon S3 (S3) bucket, and then restore an Amazon Aurora DB cluster from those files. This option can be considerably faster than migrating data using `mysqldump`. For details, see Migrating MySQL to Amazon Aurora by Using mysqldump (p. 78).
A database that is not MySQL-compatible	You can also use AWS Database Migration Service (AWS DMS) to migrate data from a MySQL database. 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 DB instance as described in Migrating Data from an External MySQL Database to an Amazon Aurora Using Amazon S3 (p. 67). For more information on AWS DMS, see What Is AWS Database Migration Service?

AWS Database Migration Service
Step-by-Step Migration Guide
Migrating Data from an External MySQL Database
to an Amazon Aurora Using Amazon S3

Migrating a MySQL-Compatible Database to Amazon Aurora

If your database supports the InnoDB or MyISAM tablespaces, you have these options for migrating your data to an Amazon Aurora DB cluster:

- You can create a dump of your data using the `mysqldump` utility, and then import that data into an existing Amazon Aurora DB cluster. For more information, see Migrating MySQL to Amazon Aurora by Using mysqldump (p. 78).
- You can copy the source files from your database to an S3 bucket, and then restore an Amazon Aurora 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 Using Amazon S3 (p. 67).

Migrating Data from an External MySQL Database to an Amazon Aurora Using Amazon S3

You can copy the source files from your source MySQL version 5.5 or 5.6 database to an S3 bucket, and then restore an Amazon Aurora 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 DB cluster. By copying your source MySQL data files, Amazon Aurora can immediately use those files as the data for DB cluster.

> **Note**
> Restoring an Amazon Aurora DB cluster from backup files in an S3 bucket is not supported for the Asia Pacific (Mumbai) region.

Amazon Aurora 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 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 DB cluster.

Prerequisites

Before you can copy your data to an 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 to access your S3 bucket on your behalf.

Installing Percona XtraBackup

Amazon Aurora 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 DB cluster, several permissions are required:

- The user that is requesting that Amazon RDS create a new cluster from an 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 S3 bucket where you store the files used to create your Amazon Aurora 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.

To create an IAM role for Amazon RDS to access S3

1. Sign in to the Identity and Access Management (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 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 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 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 xbstream (.xbstream)

For example, the following command creates a backup of your MySQL database split into multiple Gzip files.

```
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 xbstream files.

```
innobackupex --stream=xbstream  \
   /mydata/s3-restore/backup | split -d --bytes=512000 \
   - /mydata/s3-restore/backup/backup.xbstream
```

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 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 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, `.xbstream` for files stored using the Percona xbstream format.

Amazon Aurora 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 S3 bucket.

For information on creating and uploading a file to an S3 bucket, see Getting Started with Amazon Simple Storage Service in the *Amazon S3 Getting Started Guide*.

Step 3: Restoring an Aurora DB Cluster from an S3 Bucket

You can restore your backup files from your Amazon S3 bucket to a create new Amazon Aurora DB cluster by using the Amazon RDS console.

To restore an Amazon Aurora DB cluster from files on an S3 bucket

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 DB Cluster from S3**.

3. In the **Specify Source Backup Details**, specify the following:

For This Option	Do This
Source Engine	Amazon Aurora currently supports only restoring from backup files for the `mysql` database engine.
Source Engine Version	Specify the version of the MySQL database that the backup files were created from, for example `5.6.22`. MySQL version 5.5 and 5.6 are supported.
Select S3 Bucket	Select the S3 bucket where your backup files are stored.
S3 Bucket Prefix (Optional)	Specify a file path prefix for the files stored in your S3 bucket. The **S3 Bucket Prefix** is optional. If you don't specify a prefix, then Amazon Aurora 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 will create the DB cluster using the files in the S3 bucket where the full path for the file begins with the specified prefix. Amazon Aurora does not traverse subfolders in your S3 bucket looking for backup files. Only the files from the folder identified by the **S3 Bucket Prefix** 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. For example, if you store your backup files in a subfolder of your S3 bucket named `backups`, and you have multiple sets of backup files, each in its own directory (`gzip_backup1`, `gzip_backup2`, and so on), then you would specify a prefix of `backups/gzip_backup1` to restore from the files in the `gzip_backup1` folder.
IAM Role	Select the IAM role that you created to authorize Amazon Aurora to access S3 on your behalf. If you have not created an IAM role, you can choose **Create a New Role** to create one.

A typical **Specify Source Backup Details** page looks like the following.

4. Choose **Next Step**.

5. On the **Specify DB Details** page, specify your DB cluster information. The following table shows settings for a DB instance.

For This Option	Do This
DB Instance Class	Select a DB instance class that defines the processing and memory requirements for each instance in the DB cluster. Aurora supports the `db.r3.large`, `db.r3.xlarge`, `db.r3.2xlarge`, `db.r3.4xlarge`, and `db.r3.8xlarge` DB instance classes. For more information about DB instance class options, see .

For This Option	Do This
Multi-AZ Deployment	Determine if you want to create Aurora Replicas in other Availability Zones for failover support. For more information about multiple Availability Zones, see.
DB Instance Identifier	Type a name for the primary instance in your DB cluster. This identifier will be used in the endpoint address for the primary instance of your DB cluster. 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 region.
Master Username	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: `create, drop, references, event, alter, delete, index, insert, select, update, create temporary tables, lock tables, trigger, create view, show view, alter routine, create routine, execute, create user, process, show databases, grant option.`
Master Password	Type a password that contains from 8 to 41 printable ASCII characters (excluding /,", and @) for your master user password.

A typical **Specify DB Details** page looks like the following.

Specify DB Details

Instance Specifications

DB Engine	Aurora - compatible with MySQL 5.6.10
DB Instance Class	db.r3.large – 2 vCPU, 15 GiB RAM ▾
Multi-AZ Deployment	No ▾

Settings

DB Instance Identifier*	gs-db-instance1
Master Username*	myawsuser
Master Password*	••••••••
Confirm Password*	••••••••

* Required

Cancel   Previous   **Next Step**

6.   Confirm your master password, and then choose **Next**.

7.   On the **Configure Advanced Settings** page, you can customize additional settings for your Aurora DB cluster. The following table shows the advanced settings for a DB cluster.

For This Option	Do This
VPC	Select the VPC that will host the DB cluster. Select **Create a New VPC** to have Amazon RDS create a VPC for you. For more information, see earlier in this topic.
Subnet Group	Select the DB subnet group to use for the DB cluster. Select **Create a New DB Subnet Group** to have Amazon RDS create a DB subnet group for you. For more information, see earlier in this topic.
Publicly Accessible	Select **Yes** to give the DB cluster a public IP address; otherwise, select **No**. 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 .
Availability Zone	Determine if you want to specify a particular Availability Zone. For more information about Availability Zones, see.
VPC Security Group(s)	Select one or more VPC security groups to secure network access to the DB cluster. Select **Create a New**

For This Option	Do This
	VPC Security Group to have Amazon RDS create a VPC security group for you. For more information, see earlier in this topic.
DB Cluster Identifier	Type a name for your DB cluster that is unique for your account in the region you selected. This identifier will be used in the cluster endpoint address for your DB cluster. For information on the cluster endpoint, see . The DB cluster 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 clusters per AWS account, per region.
Database Name	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.
Database Port	Specify the port that applications and utilities will use to access the database. Aurora 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.
Parameter Group	Select a parameter group. Aurora has a default parameter group you can use, or you can create your own parameter group. For more information about parameter groups, see .
Option Group	Select an option group. Aurora has a default option group you can use, or you can create your own option group. For more information about option groups, see .
Enable Encryption	Select **Yes** to enable encryption at rest for this DB cluster. For more information, see .
Priority	Choose a failover priority for the instance. If you don't select a value, the default is **tier-1**. This priority determines the order in which Aurora Replicas are promoted when recovering from a primary instance failure. For more information, see .
Backup Retention Period	Select the length of time, from 1 to 35 days, that Aurora 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.
Enable Enhanced Monitoring	Choose **Yes** to enable gathering metrics in real time for the operating system that your DB cluster runs on. For more information, see .

For This Option	Do This
Granularity	This option is only available if **Enable Enhanced Monitoring** is set to **Yes**. Set the interval, in seconds, between times at which metrics are collected for your DB cluster.
Auto Minor Version Upgrade	Select **Yes** if you want to enable your Aurora 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 DB cluster. It doesn't apply to regular patches applied to maintain system stability.
Maintenance Window	Select the weekly time range during which system maintenance can occur.

A typical **Configure Advanced Settings** page looks like the following.

Configure Advanced Settings

Network & Security

VPC*	Default VPC (vpc-name)
Subnet Group	default
Publicly Accessible	Yes
Availability Zone	No Preference
VPC Security Group(s)	Create new Security Group / default (VPC)

Database Options

DB Cluster Identifier	
Database Name	
Database Port	3306
DB Parameter Group	default.aurora5.6
DB Cluster Parameter Group	default.aurora5.6
Option Group	default:aurora-5-6
Enable Encryption	No

Failover

Priority	tier-0

Backup

Backup Retention Period	1 days

Monitoring

Enable Enhanced Monitoring	No

Maintenance

|	
Auto Minor Version Upgrade	Yes
Maintenance Window	No Preference

* Required   Cancel   Previous   Launch DB Instance

8. Choose **Launch DB Instance** to launch your Aurora 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 .
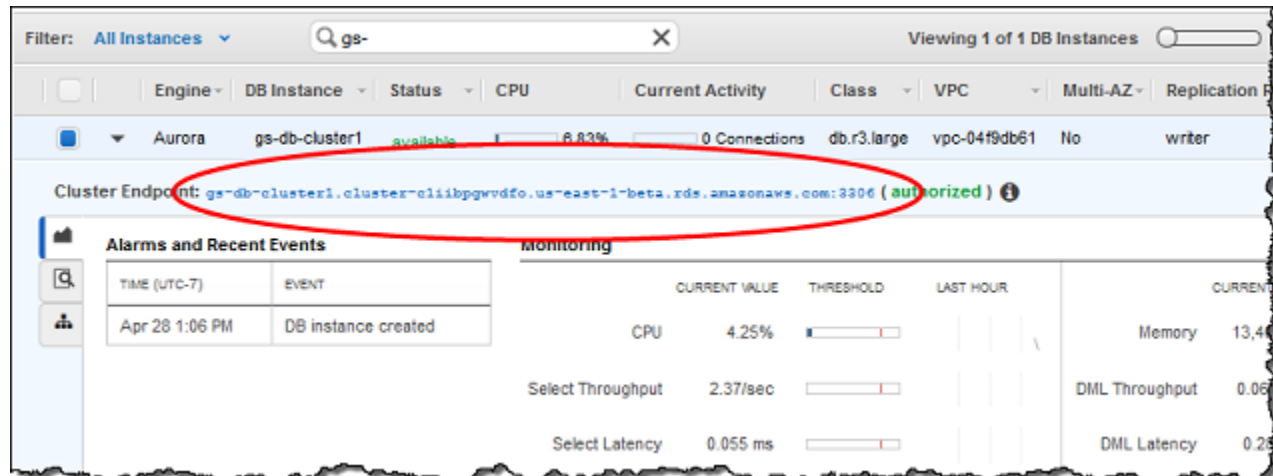
Note the port and the endpoint of the cluster. Use the endpoint and port of the cluster in your JDBC and ODBC connection strings for any application that performs write or read operations.

Migrating MySQL to Amazon Aurora 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 DB cluster. For more information, see reduced downtime...... .

Because Amazon Aurora 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 DB cluster. For a discussion of how to do so with MySQL databases that are very large, see For MySQL databases that have smaller amounts of data, see .

Migrating Data from an Amazon RDS MySQL DB Instance to an Amazon Aurora DB Cluster

You can migrate (copy) data to an Amazon Aurora DB cluster from an Amazon RDS snapshot, as described following.

Note
Because Amazon Aurora is compatible with MySQL, you can migrate data from your MySQL database by setting up replication between your MySQL database, and an Amazon Aurora DB

cluster. We recommend that your MySQL database run MySQL version 5.5 or later. For more information, see

Migrating an RDS MySQL Snapshot to Aurora

You can migrate a DB snapshot of an Amazon RDS MySQL DB instance to create an Aurora 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 Replicas.

The general steps you must take are as follows:

1. Determine the amount of space to provision for your Amazon Aurora DB cluster. For more information, see
2. Use the console to create the snapshot in the region where the Amazon RDS MySQL 5.6 instance is located. For information about creating a DB snapshot, see.
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 .
4. Use the console to migrate the DB snapshot and create an Amazon Aurora DB cluster with the same databases as the original DB instance of MySQL 5.6.

> **Caution**
> 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 DB cluster, Aurora 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 supports storage up to 64 TB in size, the process of migrating a snapshot into an Aurora 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 must use additional space in the volume to convert the tables to be compatible with Aurora. If you have compressed tables, then Aurora must use additional space in the volume to expand these tables before storing them on the Aurora 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

You might want to modify your database schema prior to migrating it into Amazon Aurora. 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.

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 DB cluster to avoid any service interruptions on your production database.

Table Type	Limitation or Guideline
MyISAM tables	Amazon Aurora supports InnoDB tables only. If you have MyISAM tables in your database, then those tables must be converted before being migrated into Amazon Aurora. The conversion process requires additional space for the MyISAM to InnoDB conversion during the migration procedure. 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 for that table. To convert a MyISAM table to InnoDB, run the following command: `alter table <schema>.<table_name> engine=innodb, algorithm=copy;`
Compressed tables	Amazon Aurora does not support compressed tables (that is, tables created with `ROW_FORMAT=COMPRESSED`). 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 https://dev.mysql.com/doc/refman/5.6/en/innodb-row-format.html.

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 Amazon's Aurora 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'
        )
    )
  )
  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 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 .

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 .

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 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.

To migrate a MySQL 5.6 DB snapshot by using the console

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 DB cluster.

4. Choose **Migrate Database**.
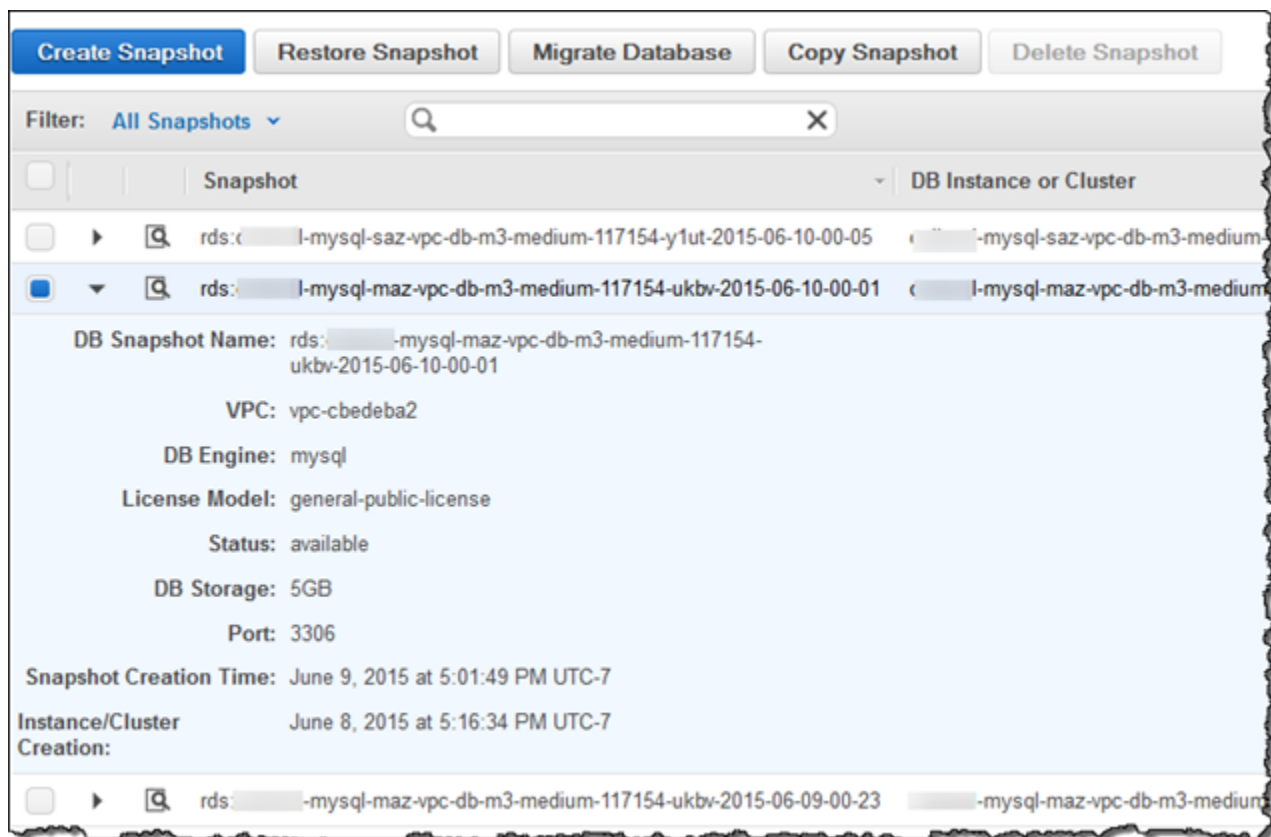
5. Set the following values on the **Migrate Database** page:

   - **DB Instance Class:** Select a DB instance class that has the required storage and capacity for your database, for example `db.r3.large`. Aurora 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. For more information, see .

   - **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 DB cluster by selecting your VPC identifier, for example `vpc-a464d1c1`. For information on using an existing VPC, see .

  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 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 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 DB cluster.

- **Enable Encryption:** Choose **Yes** for your new Aurora DB cluster to be encrypted "at rest." If you choose **Yes**, you will be required to choose an AWS KMS encryption key as the **Master Key** value.

- **Auto Minor Version Upgrade:** Select **Yes** if you want to enable your Aurora 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 DB cluster. It doesn't apply to regular patches applied to maintain system stability.

Instance Specifications

Migrate to DB Engine   aurora

DB Instance Class   - Select One -

Settings

DB Snapshot ID   rds⬛⬛⬛⬛⬛⬛⬛⬛-2016-02-22-07-42

DB Instance Identifier*   migrated-instance-name

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*   Default VPC (vpc-⬛⬛⬛⬛⬛⬛⬛⬛)

Subnet Group   default

Publicly Accessible   Yes

Availability Zone   No Preference

Database Options

Database Port   3306

Enable Encryption   No

Maintenance

Auto Minor Version Upgrade   Yes

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 DB cluster, see .
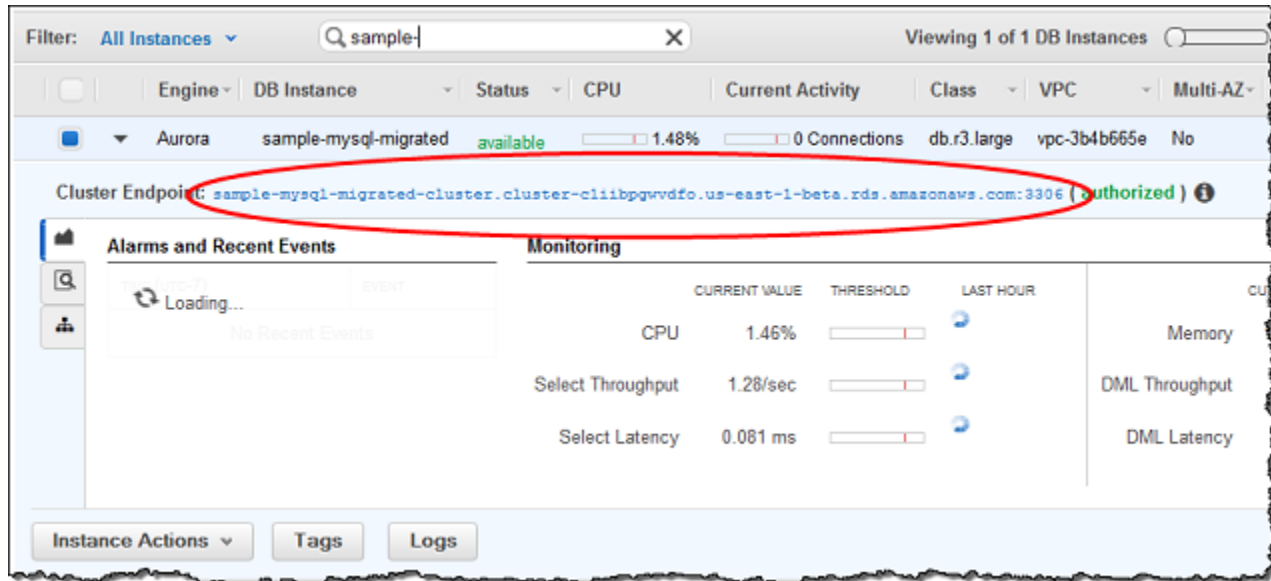
Document History

The following table describes the important changes to the documentation since the last release of AWS Database Migration Service Step-by-Step Migration Guide.

- **API version:** 20160101
- **Latest documentation update:** November 17, 2016

Change	Description	Date
On-premises Oracle to Amazon Aurora migration guide added	Added On-premises Oracle to Amazon Aurora database migration guide.	November 17, 2016