## FINAL INDIVIDUAL PROJECT(CPSC 2221)- AWS

## SUBMITTED BY: SIMRAN

## 100377444

## INDEX 1-3

Course 1: Overview of Managed PostgreSQL with Amazon RDS and Aurora 4

AWS Databases and Analytics

Relational Databases

Amazon RDS

Amazon RDS Feature Highlights

Amazon RDS Performance Insights

Aurora

Aurora Feature Highlights

Course 2: Set Up and Configure PostgreSQL 5

Enter the RDS Console

Create a PostgreSQL DB Instance

Download a SQL Client

Connect to the PostgreSQL Database

Delete the DB Instance

Introduction to PostgreSQL psql

psql Client

Connect Using psql Client

Create a Test Table

Export Results to a File

SET Parameters

Watch Queries

Course 3: Introduction to PostgreSQL SQL 9

Enter the RDS Console

SQL Primer

SQL Functions

Using Nested Statements

Using Integrity Constraints

Manipulating Sequences

Combining Queries

Using Aggregates

Using Joins

Course 4: Moving Data in PostgreSQL 10

COPY Command

Foreign Data Wrappers

Amazon S3 Import

Pgloader Utility

Course 5: Managing PostgreSQL 12

Connecting to PostgreSQL

Creating PostgreSQL Databases

Managing Databases

Creating and Managing PostgreSQL Schemas

Parameter Groups

Roles

Users

Groups

Privileges

Ending Connections

Course 6: PostgreSQL Security 15

Security Management

Restricting Password Management

Connecting to PostgreSQL over SSL

Object Security Privileges

Row-Level Security

Object Security Auditing

Course 7: PostgreSQL Backups 17

Backups

Physical Backups

Logical Backups

Manual Snapshots

SQL Dump

Restoring Backups

Course 8: PostgreSQL Monitoring 19

Automated Monitoring Tools

PostgreSQL Native Monitoring

Process Monitoring

Wait Events

SSL Monitoring

Query Monitoring

Course 9: PostgreSQL System Catalog 20

PostgreSQL System Catalog

System Catalog Query

Object Identifiers

Structural Catalogs

Informational Catalogs

Performance Catalogs

Course 10: PostgreSQL Maintenance 22

Routine Database Maintenance Optimization

Bloat

Transaction ID Wraparound

Vacuum

Autovacuum

Data Statistics

pg\_repack

Routine Reindexing

Course 11: PostgreSQL Major Upgrades 23

Minor Version Upgrades

Major Version Upgrades

Specify a Version-Compatible Parameter Group

Check Version Compatibility

Check for Unsupported Usage

Handle Amazon RDS Read Replicas or Amazon Aurora Secondary Clusters

Perform a Backup

Update Extensions

Drop Extensions

Drop Unknown Data Types

Perform an Upgrade Dry Run

Resolve a Failed Upgrade

Upgrade the Production Instance

Post-Upgrade Activity

Course 12: PostgreSQL Scalability 25

Database Scaling

Vertical Scaling

Horizontal Scaling

Cross-Region Replicas

Logical Replication

Managing Database Connections

Course 13: PostgreSQL Migration 26

AWS DMS Console

Creating a Replication Instance

Specifying Source and Target Endpoints

Creating a Task

Monitoring Your Task

My Activities 27

Certificate of Completion 28

**Overview of Managed PostgreSQL with Amazon RDS and Aurora**

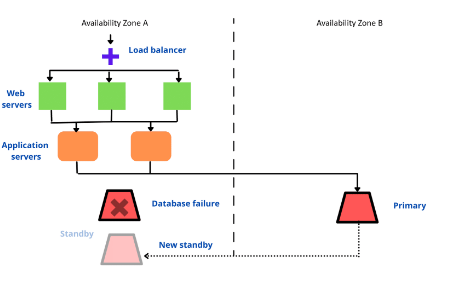
**AWS Databases and Analytics**

This tutorial describes about the portfolio of AWS(Amazon Web Services) which is specially built for builders. Some of the options offered by it are Business intelligence and machine learning, Relational and non-relational databases, Analytics, Data lake and Data movement. In this tutorial we will focus on the relational databases which consists Amazon Aurora and Amazon RDS.

A relational database is a type of database that stores and provides access to data points that are related to one another. Many organizations uses these because of their advantages like simple structuring, multiple tables relating to one another, quick to categorize and store data and filter the information and many more.

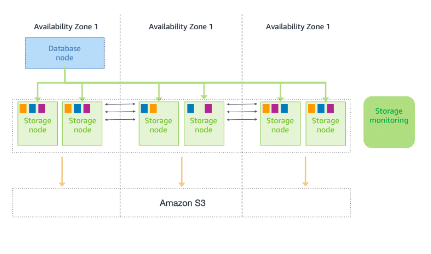
**Amazon RDS:** It simplifies the tasks by taking care of time consuming database tasks. Its goal is to automate tasks so that people can spend their time adding value to the business in more quantifiable way. Some of the major benefits are: Multi-Engine Support, Automated Tasks, Scalability to growth and Multi-AZ Deployment. There are many tasks that it completes which includes Automatic failover, Backup and recovery, Isolation and security, Industry compliance, Push-button scaling, Automated scaling, Advanced monitoring etc. Amazon RDS is secure and compliant, providing encryption in transit and at rest to help with government and industry compliance.

Multi-AZ deployments provide an enterprise-grade, fault-tolerant solution for production databases.



Benefits of performance insights are: Powerful and Automated.

**Aurora:** It is a MySQL and PostgreSQL- Compatible relational database built for the cloud and fully managed by Amazon RDS. It combines the performance and availability of traditional enterprise databases with the simplicity and cost effectiveness of open-source databases. Aurora uses a write quorum algorithm to guarantee acknowledgment from at least four of the six nodes before it commits the transaction as shown by:



Aurora Feature highlights: Fast recovery, low remote reader log, high throughput and fully managed feature in Aurora for fast deployment are some highlights of Aurora Global Database.

SUMMARY

In this we learnt about the Amazon RDS and Amazon Aurora. We described the features of the managed database, identify the main feature of the RDS and describe the architecture and key features and benefits of Aurora.

**Set Up and Configure PostgreSQL**

In this second course, we will learn about how to set up and configure the PostgreSQL and we will learn a lots of things in this.

**Enter the RDS Console:**

We need to open the AWS Management console and it will appear in a new tab or window and then go to Amazon RDS by clicking on Database and then RDS.

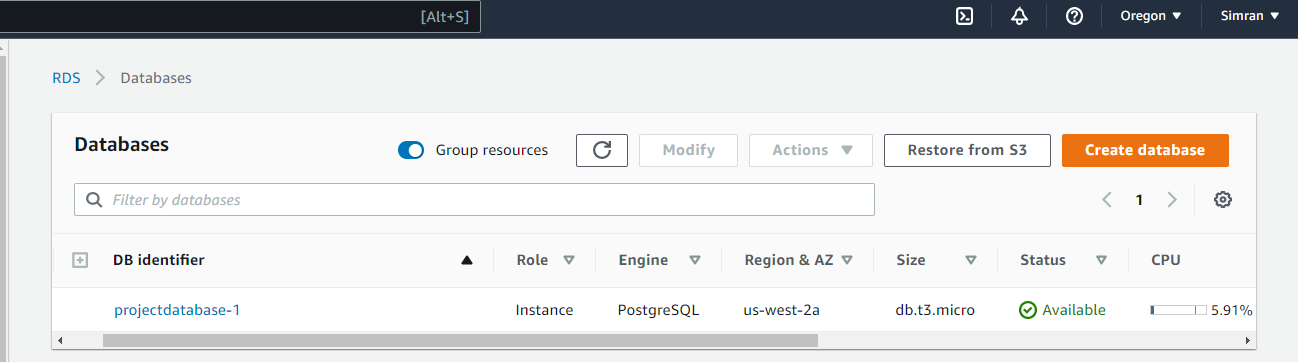
**Create a PostgreSQL DB instance:**

We will use free tier for this project and we need to follow the steps as told very carefully.

1. Select the AWS region for the DB instance.
2. Click on the create database option labelled with orange colour.
3. Select PostgreSQL for engine and templates.
4. Now, you need to be very carefull with your credentials and fill the username, DB instance identifier and password.
5. Keep the storage part and connectivity part default and enable the storage autoscaling and public access.
6. We will use port as 5432 which is default.
7. For advanced options:

* Give a database name and parameter group
* Backup period should be kept as 1 day
* Enable the enhanced monitoring and leave others default.

1. Now, click on create database. Wait for a while and you will see the windows as below:



**Download a SQL Client**

To download the SQL Client, go to SQL workbench and choose generic package for all systems including all optional libraries and then download the latest JDBC Driver.

**Connect to the PostgreSQL Database**

Now, we need to connect to database that we created. After installing SQL workbench, Find the JDBC URL on the RDS console and open database and locate the connectivity and security information.

Enter the username and password that we created for the RDS.

Click OK.

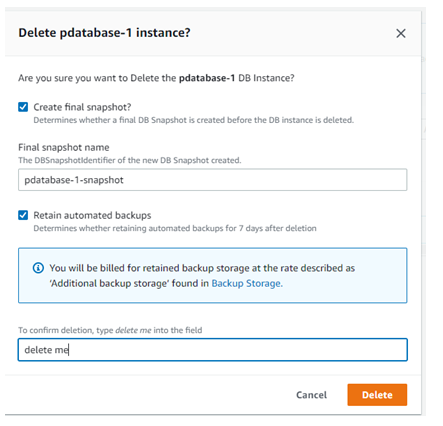
**Delete the DB Instance**

We can delete the PostgreSQL DB instance from the Amazon RDS console.

This can be done by selecting the database and clicking on actions and select delete.

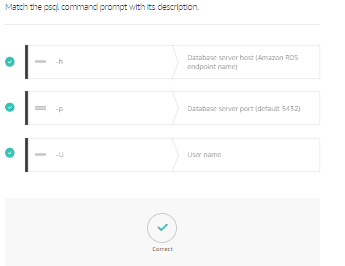
We will create a final snapshot before deleting.

Following window will appear and you need to do as mentioned:



**Introduction to PostgreSQL psql**

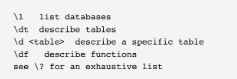
psql is a terminal-based front end to PostgreSQL. You can type in queries Interactively, issue them to PostgreSQL, and see the query results. There are different options for select and there are directions regarding how to download the Postgresql client tools for different systems. There are some short forms in this as shown in the figure:



**psql Client**

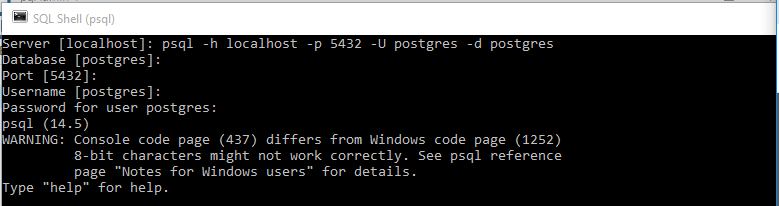
psql maintains a buffer of past and queries.

There are some meta commands which are features of psql and help to perform powerful operations like



**Connect Using psql Client**

To connect to the psql client, we use its port and as for our database, we had port 5432 and the code is as follows:

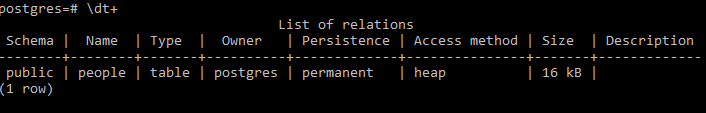


**Create a Test Table**

We will use an example of table for this and create table and insert the data.



After inserting the data, we can use \dt+ command to list the tables. The output of this command comes out as:



**Export Results to a File**

We can use the psql client to export the data to a file by using the \copy command and this will create a CSV file.

**SET Parameters**

We can .psqlrc file to set up common parameters while configuring our database.

**Watch Queries**

We can also run the queries in this same as we did in the pgadmin.

**Summary**

We learnt the most important parts of using the Amazon RDS Postgresql which were creating, connecting and deleting the database using the Amazon RDS Console.

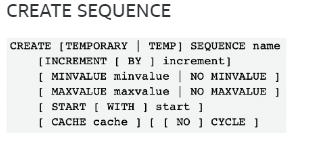
After this completing this course, we have configured and setup the postgreSQL in Amazon RDS. We knew the uses of native psql command line client and created, connected and deleted an Amazon RDS PostgreSQL database.

**Introduction to PostgreSQL SQL**

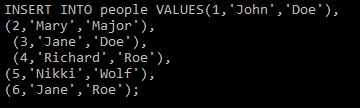
**SQL Primer**

In PostgreSQL, we can use a lot iof commands to create and modify the database.There are two languages:

DDL(Data Definition Language)used to create, modify and delete objects from database like CREATE TABLE, DROP TABLE, CREATE SEQUENCE etc.



DML(Data Manipulation Language) used to retrieve, store and delete data in the database like INSERT, SELECT, UPDATE, DELETE etc.



**SQL Functions**

SQL functions are databases objects that are commonly used for processing or manipulating data. We have different kinds of functions. Two common are string and format:

Format functions are like to char(int, text), to date(text, text), to number(text, text)etc.

**Using Nested Statements**

We can nect a query inside the statements such as SELECT, INSERT, UPDATE and DELETE.  Using nested statements is a way to tie together multiple statements Nested statements are also called subqueries.

**Using Integrity Constraints**

These helps to ensure that the values in one table make sense with related data in another table. Example: NOT NULL, CHECK, UNIQUE, PRIMARY KEY etc.

**Manipulating Sequences**

Sequence is the list of numbers for which the order is important. Common functions are NEXTVAL(), CURRVAL()

**Combining Queries**

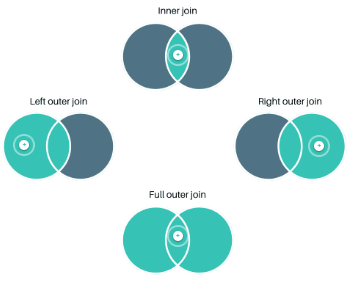
To combine the queries, we use union, intersect and except clause between the select statements. We use offset and limit clause as well in these queries.

**Using Aggregates**

These functions are used to compute a single result from multiple input rows like AVG(), MIN(), MAX(), SUM() etc.

**Using Joins**

To complete a meaningful set of data, we need to query data from multiple tables by using JOINS. There are different types of joins shown using the figure:



**Moving Data in PostgreSQL**

There are numerous ways to move data in and out of the PostgreSQL. We can use COPY, and common utilities like FDWs, AmazonS3 import and others which we will discuss in this course.

**COPY Command**

This is SQL command used in PostgreSQL. Copy command moves the data system files and PostgreSQL. But the use of copy command is not permitted in Amazon RDS to prevent it from being compromised.

The two commands: \copy from and \copy to import and export data to and from a database. We have some other commands as well like STDIN, STDOUT which direct the movement of the data and FORMAT (selects the data format to be read or written) and FREEZE command ( adds the rows to the copy).

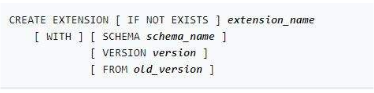
**Foreign Data Wrappers**

A PostgreSQL FDW is an installed extension which creates a link to another database and can move the data between the databases. We can use CREATE EXTENSION command to install the postgreSQL Foreign data Wrapper or FDW and CREATE USER MAPPING to set up the credentials for the remotw server as shown in the example below:



We will learn about creating and installing Postgres\_fdw. Go through the following steps:

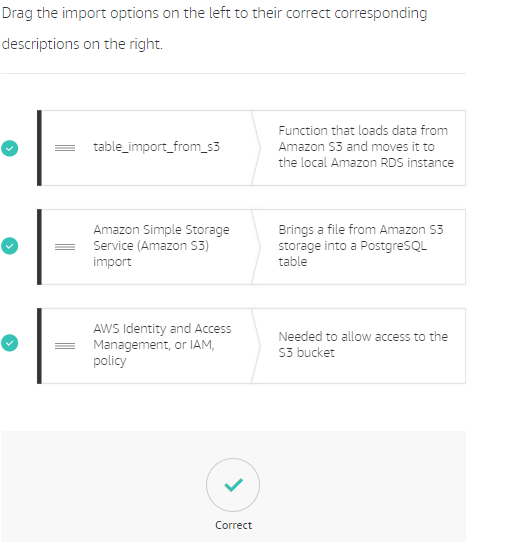
1. Install the postres\_fdw using following command:



1. Create a foreign server object using CREATE SERVER.
2. Create user mapping for each database you want to give access each foreign server.
3. Create a foreign table and use SELECT from a foreign table to access the stored data.

**Amazon S3 Import**

Amazon S3 is used by the users to move data from a file to a PostgreSQL table. We need to have the stored in an Amazon S3 file so that we it can conform to the syntax available to the Copy command. We have some commands and their functions as shown below:



**Pgloader Utility**

The pgloader is a tool which uses the COPY protocol to move data into the server. It manages the errors. It has many advantages over the COPY command like it supports different file formats, skips bad record when performing a copy of data, loading of more than one file at a time. Ther are main three command file steps to load the file.

**SUMMARY:**

There are many ways of moving in and out of PostgreSQL by using COPY command by using different utilities that we discussed in this course which were:

FDWs, Amazon S3 import and pgloader.

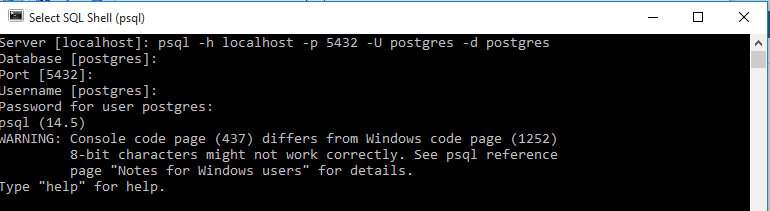
**Managing PostgreSQL**

Management of Amazon Relational Database Service(Amazon RDS) or Aurora instance which we will discuss in this course.

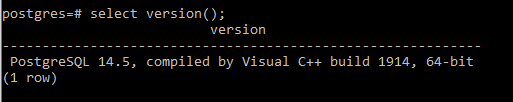
**Connecting to PostgreSQL**

We can connect to Amazon RDS PostgreSQL and Aurora PostgreSQL instance using command-line and GUI. We have some requirements to make the connection which are IP address, IP port, a valid database and valid user credentials.

Using command-line, we will have to enter some information like I did in the given example:



Then select version and make the connection.



Connecting with pgAdmin:

Launch pgadmin and create server, set master password and connect to the server.

**Creating PostgreSQL Databases**

For creating a postgres database, we firstly need to connect to the pgadmin and then click on create > database then write the name for that databse and select the options as per our demands.

**Managing Databases**

In this we can drop( DROP DATABASE database1; ), alter it( ALTER DATABASE database1 [ [WITH] OPTION [….] ] ) and rename a database(ALTER DATABASE database1 RENAME TO db1; )

**Creating and Managing PostgreSQL Schemas**

A schema includes tables, views, indexes, data type, functions, stored procedures and operations.

We use CREATE SCHEMA command to create it , ALTER SCHEMA to make changes like name of the schema and DROP SCHEMA to drop the schema.

We use search paths which are comma separated list of schema names to access a table or object.

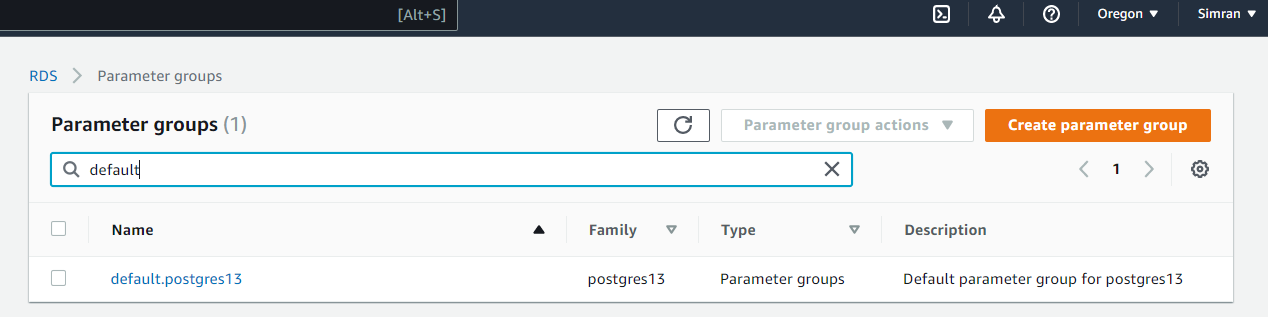
**Parameter Groups**

A parameter group is a container with the configuration values applied in the DB instance. Default parameter groups cannot be modified. These are of two types:

Static (needs to reboot the DB instance to apply changes)

Dynamic (update takes place immediately without a reboot of DB instance)

We can also review the parameter groups from RDS as shown below:



**Roles**

Users and groups are roles in postregSQL. They reside at same instance levels as shown in the figure below:

We can create roles:

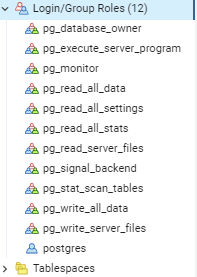
CREATE ROLE admin WITH CREATEDB CREATEROLE ROLE john, ana;

Alter roles:

Using ALTER ROLE command.

Drop role:

Using DROP ROLE command



**Users**

We use CREATE USER command to create a new user in SQL.

**Groups**

Groups are also a type of role but they can own database objects. They can be granted permissions like users. We use CREATE GROUP command to create new groups.

**Privileges**

We have roles owning the database objects and assign privileges on those objects. There are two commands:

GRANT (Assigns specific privileges to an object)

REVOKE (Removes a previously granted privilege from a user or group)

**Ending Connections**

Our DBAs must be able to cancel a query or stop a transaction. To do this, two PostgreSQL functions are built-in:

pg\_cancel\_backend

pg\_terminate\_backend

**SUMMARY**

After completing course4, we are now able to connect to postgreSQL, indentify key aspects of creating and managing it, describing parameter groups, describing roles, users and groups and end a database connection.

**PostgreSQL Security**

The highest priority at AWS is the clould security which we will discuss in this course.

**Security Management**

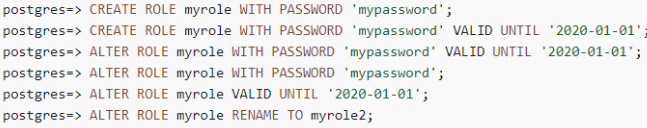
Amazon RDS and Aurora PostgreSQL Security can be managed at three different levels discussed as follows:

* AWS Identity and Access Management (IAM): In order to control who can perform management actions on Aurora database (DB) clusters or RDS DB instances, we can use IAM. Our AWS account must have IAM policies that grant the permissions required to perform Amazon RDS management operations.
* Amazon Virtual Private Cloud (Amazon VPC): Aurora PostgreSQL supports instance classes that use the default Amazon VPC only. You must use an Amazon VPC security group to control these connections.
* Standard PostgreSQL security management: To authenticate login and permissions for an Aurora DB cluster, you can take the same approach as with a standalone instance of PostgreSQL

**Restricting Password Management**

Database applications can restrict the password management to a special role by enabling the rds.restrict\_password\_commands in the parameter group.

The restricted SQL commands modify databse user passwords and the password expiration time. We can restirict new roles using the CREATE ROLE command as shown in following diagram:



**Connecting to PostgreSQL over SSL**

We can use SSL support to connect Amazon RDS and Aurora with PostgreSQL.

Use the following steps for this:

1. Download the certificate
2. Import the certificate into the operating system.
3. Connect to your database over SSL
4. Understand the SSL default
5. Tie it all together
6. Understand the error code (There might be errors in the code because of which it may not work correctly so we need to understand it properly.)

**Object Security Privileges**

Object security is driven by users and groups and each object have separate privileges that provides flexible control of the database.

There are many kinds of privileges that were discussed in the lesson which are:

RDS Super user

Grant and Revoke

DB privileges

Schema, Function, Table, Sequence privileges

Grant usage etc

**Row-Level Security**

Row-level Security is enforced in the tables. We can use CREATE POLICY to create a policy for row-level security. A template example is:

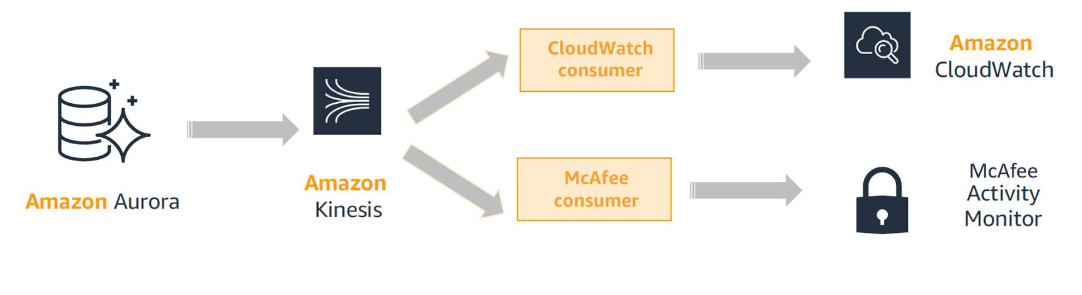


**Object Security Auditing**

Now, we are going to discuss about auditing which means checking or verifying that everything in the system is working as per our permissions without any other unauthorized accesses.

When the audit log is enabled, it will audit everything in the database.

Database activity streams: It provides more advanced functions than pg\_audit. There is an example of its working as shown below:



**Summary:**

We concluded from the lesson that Aurora and Amazon RDS uses a secured connection to help us ensure that our database information remains secure and confidential. Now, We know about the key elements of security management, user and role management, steps to secure RDS and PostgreSQL with SSL, how to use different levels of security to restrict access and using auditing.

**PostgreSQL Backups**

While creating and configuring a database, it is very important to have backup in case of any system failure or other circumstances that can leads to loss of data. In this course, we are going to learn about the backups.

**Backups**

There are two types of backups: physical and logical.

Each have its own pros and cons which we will discuss further in the course. PostgreSQL offers 2 main backup approaches and it depend upon user’s backup and recovery needs.

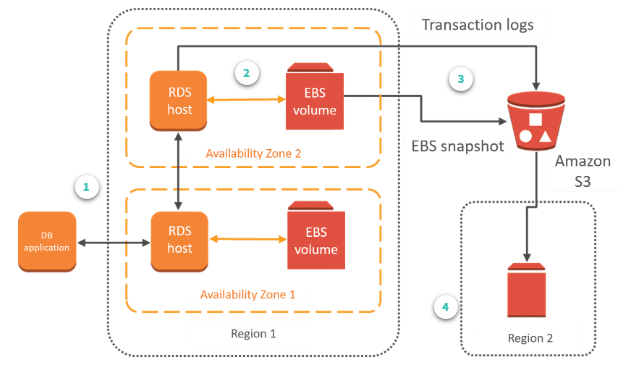
**Physical Backups**

Also called as hot backup, physical backups use Write Ahead Log to generate backup and it can be created while online. We can also have automated backups activated on the system.

Advantages are: Can create copies quickly, benefits of fast recovery, can stop PostgreSQL from replaying WAL at a specific time.

Disadvantages are: Only restore data on same PostgreSQL version, must backup entire cluster of the database, can only copy the backup as it is.

The backup process for RDS and Aurora is shown using the following picture:



**Logical Backups**

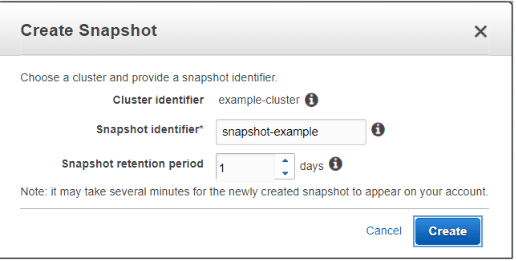
Logical backup is chosen when database is needed to move or archive a database, tables or schemas. Logical backups are taken with a PostgreSQL utility called pg\_dump.

Pros are: Maintain portability, modify data, support selective backup and restore.

Cons are: Requires more time, contain no incremental or differential options.

**Manual Snapshots**

A manual snapshot takes an image of the disks and creates a manual backup which are kept until we delete it.



**SQL Dump**

This is oldest method of having backup. It has three backup modes which are Plain text, compressed, directory mode.

**Restoring Backups**

Now, we are going to discuss about how to restore the backup. The storage of new instance is the same as original database instance but storage type can be changed.

Restore with Amazon RDS: We can restore a Database instance from snapshot using the AWS management console.

Restore with Aurora: We can create the data by creating a new cluster from the backup data having saved snapshots and then restoring a new copy of cluster created from backup data to any point in time during the backup retention period.

After completeing this course, we are now able to explain the types of backup and recovery, use of SQL dump and how to restore Amazon RDS and Aurora databases.

**PostgreSQL Monitoring**

Amazon RDS and Aurora provides enhanced monitoring tools which we will discuss in detail in this course.

**Automated Monitoring Tools**

Enhanced monitoring offers the ability to view and analyse the same metrics that are available in the OS but in a visually pleasing format. The automated monitoring tools used to monitor Aurora are:

Amazon RDS events

Database Log files

Amazon RDS enhanced monitoring, performance insights and recommendations

**PostgreSQL Native Monitoring**

There are many users who consider the native PostgreSQL command-line interface commands to be more powerful and efficient option because these enable users to monitor their PostgreSQL databases effectively.

Real time or runtime monitoring is accomplished using the postgreSQL statistics collector view.

**Process Monitoring**

It is a deeper study into pg\_stat\_activity. DBAs uses process monitoring to see number of active connections.

Managing connection activity:  pg\_stat\_activity view will have one row per server process, showing information related to the current activity of that process. Some column titles are:

Backend\_start, xact\_start, query\_start, state\_change , pid

**Wait Events**

The wait\_event\_type column indicates the type of event for which the backend is waiting. There are nine different types of wait event which are:

LWLock, Lock, BufferPin, Activity, Extension, Client, IPC, Timeout and IO.

Lock is a heavyweight lock which are used to protect SQL- visible objects like tables.

Lightweight lock is a quick event. LWLock is an example of lightweight lock.

**SSL Monitoring**

SSL is a standard protocol for establishing secure links among networked computers. DBAs ensure that the data is being secured as intended.

Pg\_stat\_ssl enables a user to see any connections that are using SSL, the ciphers that are being used and the SSl version that is being used.

**Query Monitoring**

Queries can be monitored using the dynamic statistics view pg\_stat\_activity. It displays one row per backend and shows the information about the current activity of the backend. Few important points to remember:

pg\_stat\_statements: Provides information about all the statements run in the database.

pg\_buffercache: Provides a list and priority to each table in the store.

pg\_stat\_tup: Provides information about rows in the database.

pg\_freespace\_map: Indicates free space on the individual pages on the database.

**SUMMARY:**

We learnt that monitoring is an important part of maintaining the reliability, availability and performance of the database and also talked about the different to monitor the database.

**PostgreSQL System Catalog**

PostgreSQL System Catalog is a schema with tables and views that contain metadata about all the object in the database. So, we will talk about System catalog, catalog queries, object identifiers and different types of catalogs.

**PostgreSQL System Catalog**

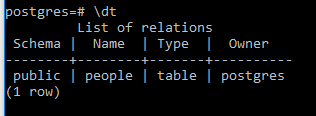
DBAs uses catalog to to discover when operations are occurring, how tables can be accessed and whether the database system is reading information from memory or disk. Pg\_catalog schema is contained in each database which consist of system tables and inbuilt data types, functions and operations or we can say it acts as a dictionary for PostgreSQL database. These can be categorized as: Structural, Informational and Performance. There are total 60 catalogs.

**System Catalog Query**

psql is the built-in command or default command-line interface that provides a view of objects such as tables and indexes.

\dt(command that describe the table) and \dv(command that describe the view) are the two examples of psql.

Psql \dt command: As we had only 1 relation or table so the output came out to be:



The primary focus of shared catalog tables is on roles, Tablespaces and Databases.

**Object Identifiers**

Object identifiers (OIDs) are the names for any type of object or element in a database. These are used by PostgreSQL as primary keys for system tables. It is used as a reference point for data in the table.

**Structural Catalogs**

It maintains the structure of the user data model and includes a list of table, a list of columns, a list of relationships between tables and function definitions.

The pg\_class catalog is the most important catalog that contains all the tables, indexes and sequences. PostgreSQL stores everything like table or relation in the pg\_class catalog table.

pg\_attribute: catalog pg\_attribute stores information about the table columns.

**Informational Catalogs**

Informational Catalogs provides data about relations in the database to the users. Two most important info catalogs are ps\_stat\_activity and pg\_locks.

ps\_stat\_activity shows one row per server process.

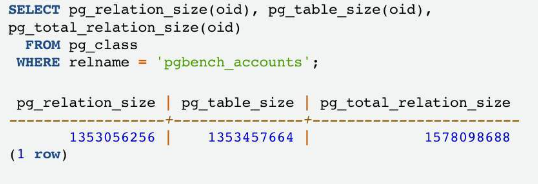
pg\_locks contains one row per active lockable object, requested lock mode and relevant process.

**Performance Catalogs**

Performance catalogs are used by users to monitor runtime performance and usage metrics in the database. There are main three groups of monitoring stats tables which are: Database Specific, Table Specific, and Query Specific

The catalog pg\_proc stores information about functions, procedures, aggregate functions, and window functions.

A query comparing pg\_relation\_size, pg\_table\_size, and pg\_total\_relation\_size.is shown below:



**SUMMARY:**

The system catalog of postgreSQL is a schema with tables and views that contain metadata about the objects in the database. So, we learnt about the categories and functions of postgreSQL system catalog, identify key tables and functions and explain how these key tables and functions are used.

**PostgreSQL Maintenance**

As maintenance of database is a vital step for accuracy and efficient working. So, we will discuss about the tasks that can be performed to automate some tasks to time and efforts.

**Routine Database Maintenance Optimization**

We have 4 major tasks to do in this which are:

Create backup copies: A successful recovery after catastrophic events requires the creation of backups of data on regular schedule.

Vacuum the database: Database requires the periodic vacuuming.

Manage the log files: We should save the server's log output and Set a retention period for system logs by using the rds.log\_retention\_period parameter.

Update planner statistics: The planner that relies on statistical info about the tables generates good plans for queries.

**Bloat**

Bloat is the side effect of PostgreSQL MVCC. When we run UPDATE or DELETE statements, dead spaces are left behind. An unavailable row is left behind when user deletes any row.

**Transaction ID Wraparound**

A transaction ID (XID) is assigned for every transaction that occurs in a database. The PostgreSQL MVCC transaction isolation-level semantics compares XID numbers. The best way to address XID wraparound exhaustion with minimal workload interference is to proactively locate troublesome tables. This can handle these tables during planned maintenance periods.

**Vacuum**

**Vacuum can manually as well as it can be automated:**

Manual vacuum mode: There are 2 types of VACUUM: standard VACUUM and VACUUM FULL.

The standard default VACUUM command vacuums the entire table and all associated indexes.

VACUUM FULL compacts tables by writing a completely new version of the table so that it is left with no dead space.

**Autovacuum**

Autovacuum automates the launch of VACUUM and ANALYZE commands (to gather statistics). Autovacuum checks for bloated tables in the database and reclaims the space for reuse. The autovacuum daemon is activated by default in Aurora and Amazon RDS PostgreSQL.

**Data Statistics**

A query plan is a sequence of steps used to access data. PostgreSQL devises a query plan for each query it receives.

PostgreSQL uses a cost-based optimizer to generate query plans. The cost optimizer compares all query methods and selects the lowest cost option. The meaning of age is the interval at which statistics run and depends on the application.

**pg\_repack**

This is used to remove the bloat from the tables and indexes. It works online and requires server-side extension and client-side utility.

**Routine Reindexing**

It works by using REINDEX command.REINDEX rebuilds an index using the data stored in the index's table, replacing the old copy of the index.

**SUMMARY:**

So, in this course we learnt about the different methods for maintenance, its key features and situations that can be prevented through this, proper use of REINDEX commnd and functionality of vacuum feature.

**PostgreSQL Major Upgrade**

Whenever RDS supports new version of database engine, we can upgrade it to that version. There are two kinds of upgrades for postgreSQL DB instances which we will discuss further.

**Minor Version Upgrades**

Minor version upgrades are commonly referred to as bug-fix releases and include only changes that are backward-compatible with existing applications.

**Major Version Upgrades**

Major version upgrades frequently contain DB changes that are not backward-compatible with previous versions of the DB.  These are not applied automatically. Its benefits are:

Community support, Performance, Security and compliance , and features and extensions.

**Specify a Version-Compatible Parameter Group**

This is the first step and we have two options here which are:

* Specify a default parameter group for the new DB engine version.
* create your own custom parameter group for the new DB engine version

**Check Version Compatibility**

In this 2nd step, we need to check that DB’s instance class is compatible with PostgreSQL version to which we are upgrading.

**Check for Unsupported Usage**

We need to check for unsupported usage in this step. To do this, access prepared transactions. We can use

SELECT count(\*) FROM pg\_catalog.pg\_prepared\_xacts;

to verify that there are no open prepared transactions on our instance.

**Handle Amazon RDS Read Replicas or Amazon Aurora Secondary Clusters**

To prevent a replica upgrade, promote the replica to a standalone instance or delete it before starting the upgrade process. In case of deletion, we should recreate it after upgrading of the instance. And upgrade the Aurora.

**Perform a Backup**

In Amazon RDS, if our backup retention period is greater than 0, the upgrade process creates DB snapshots of your DB instance before and after upgrading.

In Aurora, the upgrade process creates a DB cluster snapshot before upgrading.

**Update Extensions**

* For Amazon RDS, Extensions are: address\_standardizer , address\_standardizer\_data\_us , postGIS , postgis\_tiger\_geocoder, postgis\_topology
* FOR Aurora, extensions are pgRouting and postGIS.

**Drop Extensions**

As we know that the engine upgrade doesn’t upgrade any PostgreSQL extensions, so we must drop certain extensions before completing a major version upgrade.

pgRouting *extension is not supported when you skip a major version and upgrade from version 9.4 to 11.x.*

**Drop Unknown Data Types**

Because of the reason that PostgreSQL version 10 does not support the unknown data type, we must drop any unknown data types.

**Perform an Upgrade Dry Run**

It is recommended to test major version upgrades on the duplicate DB first. There are three ways that you can create a duplicate test instance:

* Restore your DB from a recent snapshot.
* Do a point-in-time restore of your DB to its latest restorable time.
* Clone your DB, for Aurora only.

**Resolve a Failed Upgrade**

Precheck: The precheck procedure checks all potential incompatible conditions across all DBs in the instance.

Issue encountered: If the precheck encounters an issue, it creates a log event indicating that the upgrade precheck failed.

**Upgrade the Production Instance**

Amazon RDS creates two logs that the pg\_upgrade utility produces, so that users can view information about an upgrade in progress. These are pg\_upgrade\_internal.log and pg\_upgrade\_server.log.

**Post-Upgrade Activity**

We need to complete following five steps after the upgrade:

* Review the same logs that pg\_upgrade utility produces to ensure that the upgrade completed successfully.
* Run the ANALYZE operation to refresh the pg\_statistic table.
* Confirm in the release notes which extensions we may need to upgrade.
* Run the following command: ALTER EXTENSION PostgreSQL-extension UPDATE TO 'new-version'.
* We can also upload the upgrade logs to Amazon CloudWatch Logs.

**SUMMARY:**

So, we have learnt minor version upgrades, locating info about completing minor version upgrades, major version upgrades and the steps of a major version upgrades.

**PostgreSQL Scalability**

Scaling a relational database is critical to supporting the increased application needs which we will learn in this course.

**Database Scaling**

There are three ways to improve scalability of the database:

Optimizing

Vertical scaling

Horizontal Scaling

**Vertical Scaling**

One of the simplest ways to increase the capacity of your database is to scale vertically, or use a bigger instance, to get more CPU, memory, and disk I/O.

**Horizontal Scaling**

We can use Amazon RDS read replicas to quickly and seamlessly scale horizontally beyond the capacity constraints of a single DB instance.

A centralized SSD virtualized storage layer in Amazon Aurora replicas eliminates the need to copy data to the read replicas, provides a single point to write to the storage from the primary source and makes horizontal scaling of Amazon Aurora replicas extremely fast.

**Cross-Region Replicas**

Cross-Region read replicas in Amazon RDS:

* Improve your disaster recovery capabilities.
* Scale read operations into an AWS Region closer to your users.
* Help migrate from a data center in one AWS Region to a data center in another AWS Region.
* Promote remote readers to a primary for faster recovery in the event of a disaster.

**Logical Replication**

The two most suitable applications for logical replication in postgresql are Replicating changes from PostgreSQL instances running on premises andUpgrading between PostgreSQL versions

**Managing Database Connections**

The creation of new database connections in PostgreSQL is expensive and uses valuable resources. PostgreSQL has built-in functionality for the reuse of connections.

**SUMMARY:**

We discussed about the importance of database scaling functionality, key differences between vertical and horizontal scaling, process of logical replication and many more.

**PostgreSQL Migration**

AWS Database Migration Service (AWS DMS) helps you securely migrate databases to Amazon Web Services (AWS). We will learn about it in detail.

**AWS DMS Console**

We can begin our database migration process in two ways:

1. We can choose the AWS DMS console, and perform each step there.
2. We can use the AWS Command Line Interface (AWS CLI). For more information about using the AWS CLI with AWS DMS.

**Creating a Replication Instance**

The first task in migrating a database is to create a replication instance.

We can navigate to **Replication instances** to begin:

1. In the navigation pane, choose **Replication instances**.
2. Choose **Create replication instance** and follow the instructions.

**Specifying Source and Target Endpoint**

From the side panel, choose **Endpoints**. Then, choose the orange **Create endpoint** button. Then SELECT RDS DB instance check box and fill the application info and choose create endpoint button.

**Creating a Task**

Go to create database migration task tab and fill in the task configuration and some others steps needed to completed and then you need to choose create task and the task will be created.

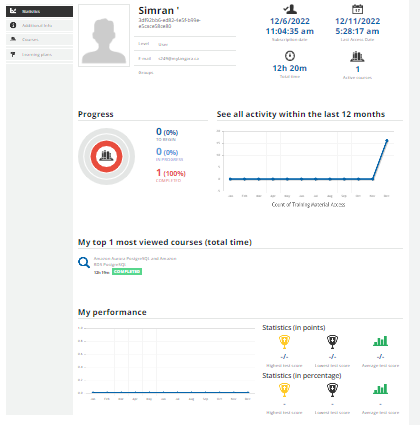
**Monitoring Your Task**

AWS provides us with the five tools for monitoring which are: AWS DMS events and notifications, Task status, Amazon CloudWatch alarms and logs, AWS CloudTrail logs, and Database logs**.**

**SUMMARY:**

In this course, we studied the migration options, how to migrate a database using AWS DMS console, ways to monitor AWS DMS tasks.

**MY ACTIVITIES:**

****

**CERTIFICATE OF COMPLETION:**

****