CPSC 2221-002  
Individual Project Jay Lee  
 100 357 736

Overview of Managed PostgreSQL with Amazon RDS and Aurora

Information from AWS training and certification, Amazon Aurora PostgreSQL, and Amazon RDS PostgreSQL, summarizing key points of what I’ve learned

***Relational Databases:***

Relational database is a type of database that sores and provides access to data points that are related to one another. They are more intuitive, and straightforward by representing data in tables utilizing relational model.

Each row in a table is a record with a unique identifier called a *key*, and the columns of the table hold attributes of the data. Each record usually has a value for each attribute establishing the relationships among data points. Foreign keys are used to link tables to one another.

Relational database model provides a standard way of representing and querying data. Something that makes relational database stand out is in the use of tables and indexes to better structure information conveniently.

***Amazon RDS:***

Relational databases are widely used from personal projects to a large-scale application in corporate setting. However, relational databases can be hard to manage as you make updates or scale. Amazon Rational Database management, RDS, helps you and your team to manage the relational database of your choice in a handy manner.

Multi Engine Support: Amazon RDS gives you access to the capabilities of a familiar database. It supports MySQL, MariaDB, PostgreSQL, Oracle Server, and Microsoft SQL Server.

Automated Tasks: Amazon RDS manages the work involved in setting up a relational database, from provisioning the infrastructure capacity you request to installing the database software. After your database is set up, Amazon RDS automates common administrative tasks such as performing backups and patching the software that powers your database. Amazon RDS also automates scaling, replicas, and restore actions.

Scalability to handle growth: You benefit from the flexibility of being able to quickly scale the compute resources or storage capacity associated with your relational DB instance. Amazon RDS uses replication to enhance database availability, improve data durability, or scale beyond the capacity constraints of a single DB instance for read-heavy database workloads.

Multi-AZ Deployment: Amazon RDS Multi-AZ deployments provide enhanced availability and durability for RDS DB instances, making them a natural fit for production database workloads.

***Amazon RDS Feature Highlights:***

Amazon RDS Multi-AZ Deployments: Amazon RDS Multi-AZ deployments provide enhanced availability and durability for RDS DB instances. When you provision a Multi-AZ DB instance, amazon RDS automatically creates a primary DB, which then synchronously replicates the data to a standby instance in a different availability zone. Each availability zone runs on its own physically distinct independent infrastructure and is engineered to be highly reliable.

*Amazon RDS Performance Insights:*

Database administrators need to monitor and manage their databases, but the Amazon RDS performance insights feature can help you quickly assess any performance bottlenecks in your relational database workloads. Performance insights collects detailed database performance data and displays the data to drive a graphical interface.

***Aurora:***

Aurora is a cloud-based relational database management that is compatible with MySQL and PostgreSQL. It offers speed that is five times faster than standard MySQL and three times faster than PostgreSQL. Aurora offers fault-tolerant, self-healing storage which provides six copies of data across three Availability Zones and continuously generate backup to Amazon Simple Storage Service.

Aurora is highly secure and offers network isolation and encryption at rest and in transit. And furthermore, it has the same management benefits as Amazon RDS, meaning no hardware provisioning, software patching, setup, configuration, nor backups.

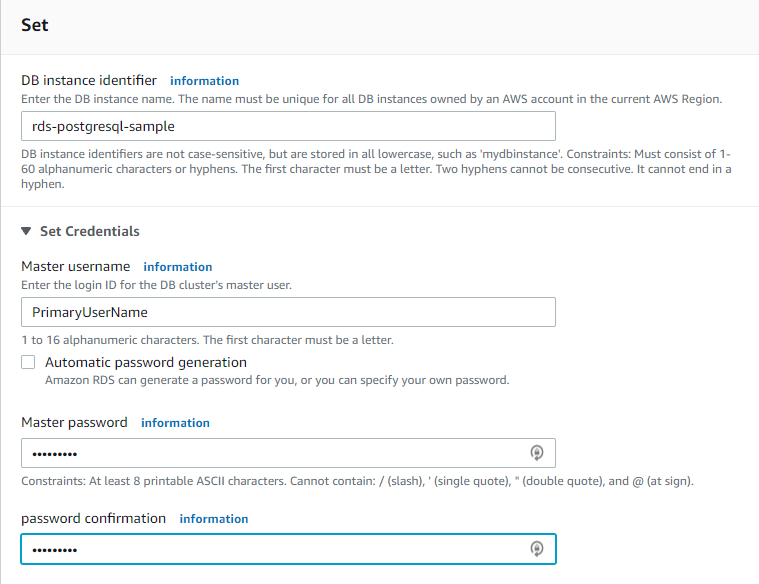
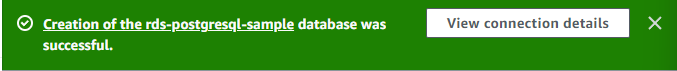
***Installing and initiating AWS RDS Postgres from scratch:***

1. First, user must have access to AWS and have a valid account (Can be free)
2. Navigate to Amazon RDS
3. Select the region for the DB instance
4. Click “Create Database” button
5. Choose your engine option as “PostgreSQL”
6. Choose the version of the engine
7. Choose “Free tier”
8. Use the following settings for your DB instance as an example:  
   **DB instance identifier:**Enter a name for the DB instance that is unique for your account in the Region you selected. For this example, we will name it rds-postgresql-sample.

**Primary username:**Enter a username that you will use to log in to your DB instance. We will use PrimaryUsername in this example.

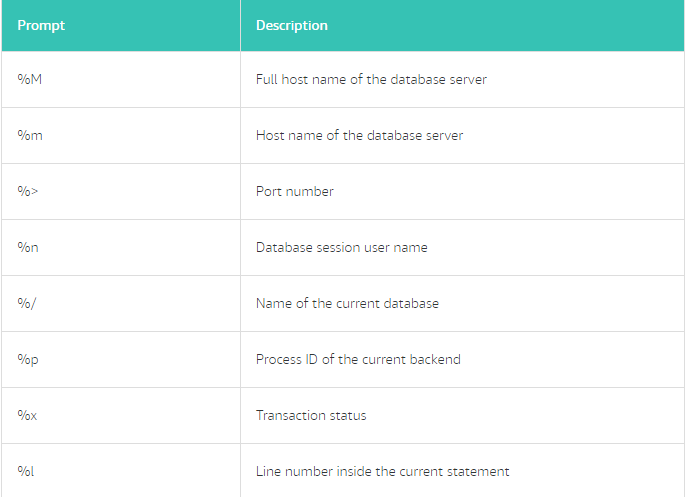
**Primary password:** Enter a password that contains 8–41 printable ASCII characters (excluding /,", and @).

**Confirm password:** Retype your password.

1. When you follow the above steps, you should be seeing the following screen:  
   
2. Set DB instance size to: db.t2.micro-1 vCPU, 1Gib RAM
3. Storage settings: Set your storage type to General Purpose (SSD), Allocated Storage to 20Gib, and enable storage auto scaling with multi-az deployment turned off.
4. Connectivity: Set your network and security settings to: VPC to default VPC, subnet group to default, public access to yes, VPC security group to create new, and availability zone to no preference.
5. Authentication: In this practice scenario, choose password authentication.
6. Lastly, under “Additional Configuration”, set your database port to 5432.
7. For the database options, enter your initial database name and db paramenter group
8. Set your backup settings to: Enable automatic backups, backup retention period to 1 day, and backup window to “No Preference”, and copy tags to selected
9. Click “Create Database”
10. And you should be able to create your own DB instance and redirected to the screen below:  
    

***Prompts – PostgreSQL:***

Prompt characters can be used to return types of information in psql, and used to short-handily process commands without typing the command in full.



***Watch Queries:***

Watch queries were used in need to view all the currently running queries and can be done by setting up /watch command. It can be set so that watch queries are ran in time intervals and can be included in a script

***Data Definition Language and Data Manipulation Language:***

You can use numerous commands to create and modify a database in PostgreSQL, and these commands are categorized under two different universal languages: Data Definition Language (DDL) and Data Manipulation Language (DML)

DDL: DDL is a set of commands to help you perform CRUD on database. Some of the commonly used DDL commands are: CREATE TABLE, DROP TABLE, CREATE SEQUENCE, DROP SEQUENCE

DML: DML is a set of commands that helps you retrieve, store, change an delete data in your database. Some common DML commands are: SELECT, INSERT, UPDATE, DELETE

***SQL Functions:***

SQL functions are database objects that are commonly used for processing or manipulating data. When a function is used as shown in this example, the user is formatting how the data is displayed. Think of any sort of database or even Microsoft Excel, in which built-in functions help you process or modify information.

***Nested Statements:***

PostgreSQL is very flexible in a sense user may insert nested statement almost everywhere. User may nest a PostgreSQL query inside statements such as SELECT, INSERT, UPDATE, and DELETE and tie multiple statements, joining data together across different tables. Nested statements are also called subqueries.

***Joins:***

In a relational database, data is distributed in multiple logical tables. To get a complete, meaningful set of data, you need to query data from these tables by using joins. Each join type specifies how the data from one table will be used to select rows in another table.

There are multiple types of joins including inner join, left outer join, right outer join, and full outer join which can be used to specify the data user may want depending on the situation.

Alias is very helpful when using joins to simplify the table names and column names in a complicated query. Aliases allows you to use a shorthand name within the query helping you to be a more efficient DMA.

As an example, table name insurance can be aliased as i, and subsequently it’s column name can be aliased as c. Which clears up a lot of mess when joining multiple tables by typing i.c instead of insurance.column\_name.

***Integrity Constraints:***

When designing tables, you might want to constrain data from individual columns and the tables themselves so that business rules can be enforced. This can be done using integrity constraints. In modern database best practice, constraints are not used in the database object. Note that PostgreSQL will allow the use of integrity constraints if the user chooses to use them.

Integrity constraints help ensure that values in one table make sense with related data in another table. Commonly used constraints include NOT NULL, CHECK, UNIQUE, PRIMARY KEY, and FOREIGN KEY.

***Combining Queries:***

There are times when you want to compare query results where best practice advises not to use a join statement. This is because data is being pulled from different result sets of those queries and combined into a single result.

When you want to combine queries, you can use UNION, INTERSECT, and EXCEPT. You can use these clauses to combine or exclude like rows from two or more tables. They are useful when you need to combine the results from separate queries into a single result. They differ from a join in that entire rows are matched. As a result, they are included or excluded from the combined result.

***Aggregates:***

In PostgreSQL, aggregate functions can be used to compute a single result from multiple input rows. This can be used to help the user to either target the needed information you need to display information or to eliminate certain information from display.

Some of the common aggregate functions include: AVG(), MIN(), MAX(), and SUM(). However, GROUP BY clause itself can be used to aggregate a set of rows to group the outcome of the query based on previously mentioned functions.

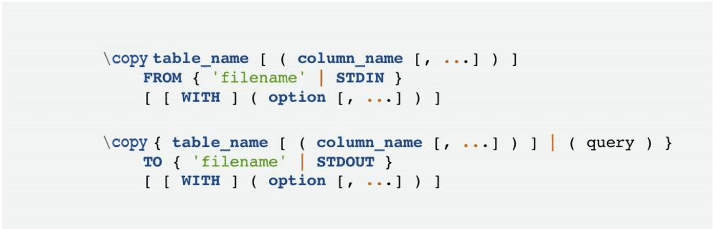
***Copy Command:***

The copy command is an interesting command as it is used to directly move data between tables and system files. You can specify whether data is coming to or going from the server by adding TO or FROM clause to the query and it is a server-side command. The command requires user to specify the viewpoint of the server, and the user ID logged in to the server must be able to access the file.

Copy, or \copy command runs as a SQL COPY command, therefore can be used interchangeably, however \copy command needs to be processed through psql; reading or writing files and routes of the data between the server and the local system.

\copy from and \copy to commands import and export data to or from a database, better described as pull and pushing the data from a source to your database. These commands can be used concurrently with supplementary commands like REPLACE, APPEND, and RULE.

*STDIN and STDOUT formatting*

 *Copy Options List*

Table

Description automatically generated

***Foreign Data Wrappers***

A PostgreSQL FDW is an installed extension that creates a link to another PostgreSQL database. It can move data between databases. The postgres\_fdw module provides the FDW postgres\_fdw, which can access data stored in external PostgreSQL servers.

When you are linked to the server using the FDW with the qualified name postgres server, you can run other commands and group, or aggregate, them all together in a command string.

Graphical user interface, text, application, email

Description automatically generated

For users to successfully make full use of PostgreSQL FDW, user should:

1. user should first install the extension by using the CREATE EXTENSION command
2. Create a foreign server object by using CREATE SERVER, specify connection information and options
3. Create user mapping by using CREATE USER MAPPING, and this will allow access to each foreign server for each database user. In this command, you should specify remote username, password, and password options for the user mapping
4. Create foreign table is the next step, using CREATE FOREIGN TABLE or IMPORT FOREIGN SCHEMA for each remote table you want to access. The columns of the foreign table must match the referenced remote table, and there by specify correct remote names as options of the foreign table object

***Pgloader Utility***

The pgloader utility is a data loading tool that is based off on PostgreSQL COPY protocol to import the data into the server. It manages errors by filling a pair of reject.dat and reject.log files.

PGloader offers several advantages over using the COPY command which will be listed below:

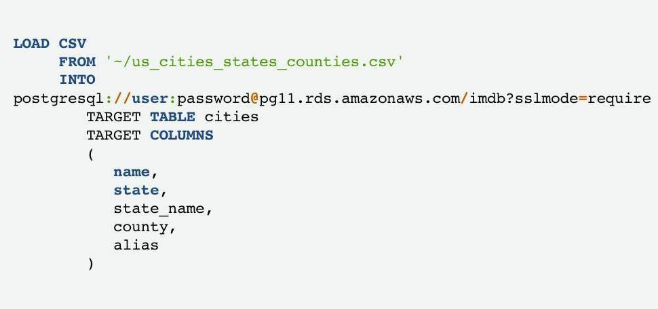
1. Supports more file formats, such as: CSV, Fixed-column formats, DB files, and IBM IXF files
2. Skips bad records when performing a copy of the data
3. Parallelism supported, which allows loading of more than one file at a time
4. Available in supported PostgreSQL repositories ex. yum, apt

In the context of pgloader, a command file (or load file) is a file that instructs pgloader how to perform a migration. Using a command file, you can list all pgloader commands you want to run and save them to a single file. You can then run that file and several commands together for greater efficiency. This gives you finer control over how your data is loaded into PostgreSQL and helps you to perform complex migrations.

The command copies information from a CSV format command file into specific columns of a table in PostgreSQL. pgloader uses a command file, or you can run it all from the command line.

***Using the pgloader from command file***

1. Load the command file from the CSV file. CSV file, user credentials, and SSL required
2. The ~/ UNIX command instructs pgloader where to find that CSV file
3. The connection information to the target database is provided.
4. Target table and columns need to be specified



1. Set field parameters
2. Set the standard\_conforming\_strings property to ‘on’
3. Allocate 12 megabytes of working memory per set

Graphical user interface, text

Description automatically generated

1. Check for existence of a table
2. Creates the table with the specified parameters, if it does not exist
3. Analyzes the table contents

Graphical user interface, text

Description automatically generated

***Security Management***

Security for RDS and Aurora PostgreSQL can be managed at three levels:

1. AWS Identity and Access Management (IAM)
2. Amazon Virtual Private Cloud (Amazon VPC)
3. Standard PostgreSQL security management

To get more into details for these three levels:

**IAM**: Controls who can perform management actions on Aurora DB clusters or RDS DB instances, user is recommended to use IAM. AWS account must have IAM policies that grand the permissions required to perform RDS management operations.

**Amazon VPC**: Aurora DB clusters must be created using Amazon VPC, because devices and Amazon EC2 instances can open connections to the endpoint and port of the DB instance for Aurora DB clusters in VPC, you must use an AMAZON VPC security to control these connections.

**PostgreSQL:** To authenticate login and permissions ofr an Aurora DB cluster, you can take the same approach as with a standalone instance of PostgreSQL. Commands such as CREATE ROLE, ALTER ROLE, GRANT, and REVOKE works just as they do in on-premises databases.

***RDSadmin default privileges***

When each new object is created, the rdsadmin role is automatically created as a security measure. DBA is assigned the role of rdsadmin. DBAs have default privileges to manage the database and other users.

RDS DBA has the following default privileges when DB instance is created

* LOGIN
* RDS SUPERUSER
* INHERIT
* CREATEDB
* CREATEROLE
* VALID UNTIL ‘infinity’

Rdsadmin user is created upon db cluster creation to provide service management for Aurora DB cluster.

***Object Security Auditing***

Auditing is a method of verifying that everything is working as expected without any unauthorized access. Users can audit databases, roles, tables, or columns and can be done through pgaudit extension. This extension offers DBA the ability to take control of the object audit inside the database. Object security involves tracking who can access data in addition to restricting access.

First, DBA must create a specific rds\_pgaudit role. And then the DBA needs to modify the DB parameter group associated with the instance and set the role of pgaudit to the newly created rds\_pgaudit. Afterwards, all the users need to do is to reboot the instance.

The audit log can include the following classes: READ, WRITE, FUNCTION, ROLE, DDL, AND MISC.

Table

Description automatically generated

***Object Auditing***

When the audit log is enabled, it will audit everything in the database. But DBA might not need to audit anything. If data is to come from an application server that have passed through quality assurance and be vetted, audits mainly need to occur on anything coming from a user into the database.

This is conducted by setting different object-level audits on specific tables. As an example, DBA might audit if a user input or viewing information on a table. They are not required to audit information in a table that includes information that is irrelevant. In other words, DBA can create object-level audits on sensitive or confidential information but select not to audit public information.

***Reading the Audit Logs***

Audit records are added to the normal PostgreSQL logs, they often come in as a string and displayed in a CSV file. Below image includes the field seen in the log with a description of the types of information it contains.

Graphical user interface, table

Description automatically generated

***Backups***

PostgreSQL backups can be either physical or logical. Physical backups are best known as file-system-level snapshots while logical backups are best known as SQL dumps. Both backup types come in handy, however has it’s pros and cons.

Physical backup: An on-premises physical backup is done by copying individual directories and files, and then restoring them one at a time. For Amazon RDS and Aurora, the automated backup uses the physical backup mode but uses point-in-time restore (PITR) to back up and recover the entire database.

Logical backup: Logical backups use multiversion concurrency control (MVCC) to generate a consistent, logical backup of data from inside the database. Logical backup, or SQL dump, works by generating a text file with SQL commands to recreate the PostgreSQL cluster, a database, or a given table.

PostgreSQL offers two main backup approaches to address the different needs of its users.

***Automated Monitoring Tools – Scrapped from AWS skill builder***

AWS provides tools to monitor Amazon RDS and Aurora. DBAs can configure some of these tools to automate monitoring. Some other tools may require DBAs to manually intervene. What is recommended is to automate monitoring tasks as much as possible.

Amazon RDS events: Amazon RDS events subscribes users to receive notifications when changes occur with a DB instance, DB cluster, DB cluster snapshot, DB parameter group, or DB security group.

Database log files: Database log files enable you to view, download, or monitor database log files using the Amazon RDS console or Amazon RDS application programming interface (API) operations. You can also query some database log files that are loaded into database tables.

Amazon RDS Enhanced Monitoring: Amazon RDS Enhanced Monitoring provides metrics in real time for the operating system (OS). This is similar to how a user can use Secure Shell (SSH) to access information in a database server.

Amazon RDS Performance Insights: Performance Insights assesses the load on your database and determines when and where to act.

Amazon RDS recommendations: Amazon RDS recommendations look at automated recommendations for database resources, such as DB instances, DB clusters, and DB cluster parameter groups.

Enhanced Monitoring using Amazon RDS and Aurora monitors two things

1. OS metrics including the central processing unit (CPU), memory, and I/O.
2. Database Performance

***Monitoring Logs***

DBAs are given options to automatically generate error logs through configuring Amazon RDS. The logs will load via Amazon CloudWatch. Once error logs are placed in CloudWatch, user may set alerts based on specific errors.

Text

Description automatically generated

***Wait Events***

The wait\_event\_type column indicates the type of events are a great indicator for the types of events the backend is waiting on. NULL indicates that there is no event specified. There always are wait event types and wait event names, for example, there might be an LWLock and wait events may have 1 to up to 65 wait event names.

The wait event name is a great indicator for what exactly is being waited on. Often times user can get hints from the name of the wait event. For example, ShmemIndexLock is a wait event for backend waiting to find or allocate space in shared memory.

Two types of wait events as an example: One instance of Lock and seven of LWLockNamed. The wait\_event named types are WALWriteLock and transactionid.

Table

Description automatically generated

***Wait Event Types***

There are nine different wait event types, and following table below shows the type name and description of the event types.

Table

Description automatically generated

***PostgreSQL System Catalog***

The PostgreSQL system catalog can be described as a schema with tables and views and contain metadata about all objects in the database. DBAs can utilize this in many ways including specifying operations that are occurring, access records of tables and records, and database functionalities such as reading information from memory or disk.

In addition to public and custom schemas, all databases should contain pg\_catalog schema. This schema tend to contain the system tables and all of the built in data types, functions, and operators. Pg\_catalog acts just like a dictionary for data for PostgreSQL ran databases.

There are a total of 60 catalogs, which can be distinguished in 3 categories:

**Structural**: Used to manage the relational management system (RDBMS)  
**Informational**: Used to view table size and monitor activity  
**Performance**: Used to monitor performance statistics about queries, tables, and indexes

***Vacuum Command***

PostgreSQL databases require periodic vacuuming. This prevents XID wraparound from causing problems and potential exhaustion. User will be given a choice to run vacuuming manually or automatically and details of the vacuuming will be discussed below.

Manual Vacuum mode:

There are two variants of VACUUM: standard VACUUM and VACUUM FULL. We will cover the standard VACUUM first. The standard default VACUUM command vacuums the entire table and all associated indexes. It looks for free space and marks it in the Free Space Map (FSM).

The FSM keeps track of pages that have free space available for use. You can add parameters to the VACUUM command to fine-tune results and accomplish additional tasks.

Options for manual vacuum mode:

VACUUM [table]: This is the basic command used to vacuum a table

VACUUM ANALYZE [table]: This command vacuums and performs a statistical analysis

VACUUM ANALYZE FREEZE [table]: This command vacuums, performs a statistical analysis, initiates the FREEZE operation, and resets xmin (the identity of the inserting transaction for this row version) to 2.

VACUUM ANALYZE VERBOSE [table]: This command vacuums, performs a statistical analysis, and prints the statistics

***Automatic Vacuum mode:***

PostgreSQL has an optional but highly recommended feature called autovacuum. Autovacuum is a daemon that 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.

In the default configuration, autovacuuming is activated and the related configuration parameters are appropriately set. VACUUM can run manually. However, PosgreSQL recommends that you run the autovacuum daemon.

The autovacuum daemon consists of multiple processes. There is a persistent daemon process, called the autovacuum launcher, that oversees starting autovacuum worker processes for all databases.

There is a maximum of autovacuum\_max\_workers worker processes that can run at the same time. If there are more processes than the maximum, the next database will process as soon as the first worker finishes.

Autovacuum follows these steps on execution:

1. Wake up
2. Look for a table that has hit a certain threshold
3. Vacuum the table
4. Sleep