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.

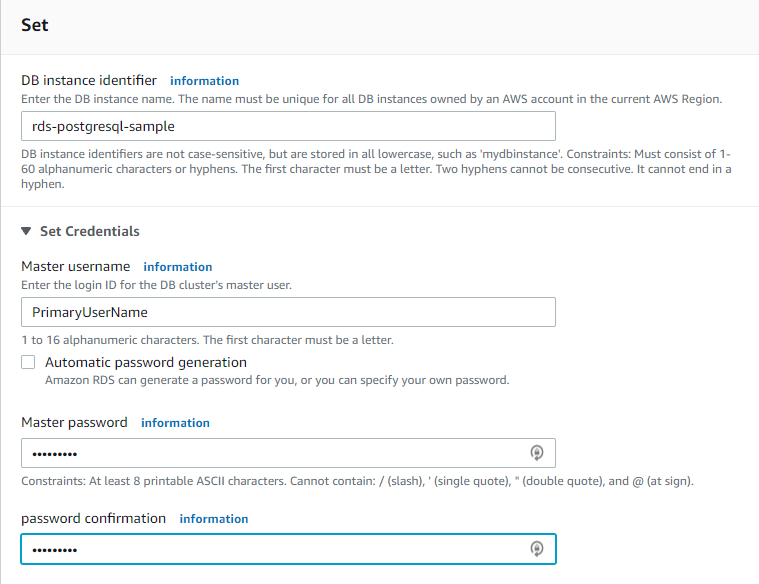
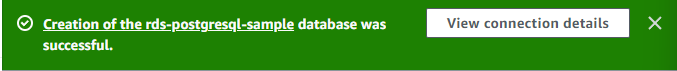
***Installing and initiating AWS RDS Postgres from scratch:***

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.

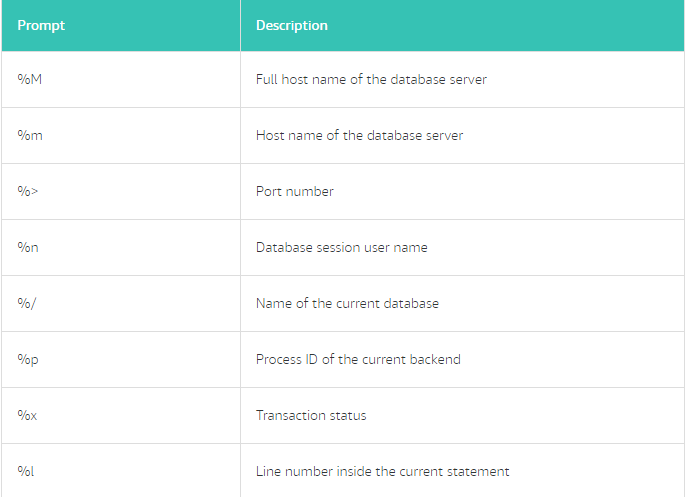
**Primary password:** Enter a password that contains 8–41 printable ASCII characters (excluding /,", and @).

**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.