2ND EDITION

Amazon Redshift Cookbook

Recipes for building modern data warehousing solutions

SHRUTI WORLIKAR | HARSHIDA PATEL
ANUSHA CHALLA

Foreword by Ippokratis Pandis, VP/Distinguished Engineer, AWS Analytics

Amazon Redshift Cookbook

Second Edition

Recipes for building modern data warehousing solutions

Shruti Worlikar
Harshida Patel
Anusha Challa

Amazon Redshift Cookbook
Second Edition

Portfolio Director: Rujuta Paradkar
Relationship Lead: Teny Thomas
Project Manager: Samuel Christa
Content Engineer: Divya Kartik Poliyath
Technical Editor: Aniket Shetty
Copy Editor: Safis Editing
Indexer: Manju Arasan
Production Designer: Prashant Ghare
Growth Lead: Kunal Raj

First published: June 2021
Second edition: April 2025

Production reference: 1040425

Published by Packt Publishing Ltd.
Grosvenor House
11 St Paul's Square
Birmingham
B3 1RB, UK.

ISBN 978-1-83620-691-0
www.packtpub.com

Foreword

I am pleased to present the second edition of this comprehensive guide to Amazon Redshift. As Vice President and Distinguished Engineer for Analytics at AWS, I have had the privilege of witnessing the remarkable evolution of Amazon Redshift, our fully managed, petabyte-scale data warehousing service. We have significantly transformed Amazon Redshift into a modern, cloud-native data analytics solution while maintaining industry-leading security, price/performance, and scalability. This book marks an important milestone in our ongoing commitment to empowering organizations with superior data analytics capabilities.

Since the release of the first edition of this book, Amazon Redshift has seen substantial advancements. Notable innovations include the introduction of Redshift Serverless and data sharing for both read and write workloads across Amazon Redshift compute environments within the same account, across different accounts, and even across Regions. Through SageMaker Lakehouse, Amazon Redshift databases can now be automatically visible to and accessible from multiple Amazon Redshift compute environments. We have also expanded our Zero-ETL, streaming ingestion, and auto-copy capabilities to make it very easy to move data into the analytics-optimized Amazon Redshift Managed Storage and incorporated cutting-edge AI/ML functionalities during query processing, all of which reflect our dedication to simplifying data management while enhancing analytical power.

This second edition, authored by Shruti, Harshida, and Anusha, provides an updated and thorough exploration of Amazon Redshift's architecture, administration, optimization strategies, and emerging trends. It covers everything from fundamental concepts to advanced implementations, making it an invaluable resource for data professionals at all stages of their careers. A key feature of this edition is its focus on practical, real-world implementations. The authors have distilled their extensive experience working with a diverse range of customers from various sectors into actionable insights and hands-on techniques. Whether you are migrating from legacy platforms, building new data warehouse workloads, or exploring AI/ML-driven analytics, this book provides the necessary guidance to unlock the full potential of Amazon Redshift. As data continues to grow in both volume and complexity, the role of efficient and scalable data warehousing becomes even more critical. This book equips you with the knowledge to leverage Amazon Redshift's power, enabling you to drive innovation and make data-driven decisions with confidence.

I commend the authors, Shruti, Harshida, and Anusha, for their dedication to producing this invaluable resource. Whether you are new to Amazon Redshift or seeking to deepen your expertise, this book will be an indispensable companion in your data analytics journey.

Ippokratis Pandis

VP/Distinguished Engineer

AWS Analytics

Contributors

About the authors

Shruti Worlikar is a cloud professional with technical expertise in data and analytics across cloud platforms. Her background has led her to become an expert in on-premises-to-cloud migrations and building cloud-based scalable analytics applications. Shruti earned her bachelor's degree in electronics and telecommunications from Mumbai University in 2009 and later earned her master's degree in telecommunications and network management from Syracuse University in 2011. Her work history includes work at JPMorgan Chase, MicroStrategy, and **Amazon Web Services (AWS)**. She is currently working in the role of Sr. Manager, Analytics Specialist SA, at AWS, helping customers to solve real-world analytics business challenges with cloud solutions and working with service teams to deliver real value. Shruti is the DC Chapter Director for the non-profit **Women in Big Data (WiBD)** and engages with chapter members to build technical and business skills to support their career advancements. Originally from Mumbai, India, Shruti currently resides in Northern Virginia, with her husband and two kids.

I would like to express my deepest gratitude to my husband and two children for their unwavering support and understanding during the many evenings and weekends I spent writing this book. Their patience and encouragement made this journey possible. I would like to thank the Packt team for their invaluable support throughout the writing and publishing journey. I am so proud to be part of the extended Redshift team where we are innovating to solve the most complex data warehousing challenges for our customers.

Harshida Patel is a principal analytics specialist solution architect at AWS, enabling customers to build scalable data lake and data warehousing applications using AWS analytical services. She has presented Amazon Redshift deep-dive sessions at re:Invent. Harshida has a bachelor's degree in electronics engineering and a master's in electrical and telecommunication engineering. She has over 15 years of experience in architecting and building end-to-end data pipelines in the data management space. In the past, Harshida has worked in the insurance and telecommunication industries. She enjoys traveling and spending quality time with friends and family, and she lives in Virginia with her husband and son.

I would like to sincerely thank Shruti for providing the opportunity to co-author this second edition. Sincere thanks to my son and husband for motivating me to complete the chapters on time.

Anusha Challa is a seasoned professional with over 15 years of expertise in architecting data and analytics solutions across on-premises and cloud environments. She has provided guidance to hundreds of Amazon Redshift customers, empowering them to design scalable and robust end-to-end data warehouse architectures. Anusha speaks at various AWS events, such as re:Invent and AWS Summits, where she shares best practices for using Amazon Redshift and other AWS analytics services. She has a bachelor's degree in computer science and a master's degree with a specialization in Machine Learning. Based in Chicago, Anusha enjoys reading books and traveling during her free time.

I would like to thank my parents, husband, and sister for their unwavering love, support, and patience—especially during my long disappearances into writing mode. A huge thanks to Shruti and Harshida for being fantastic co-authors and wonderful friends throughout this journey. My sincere appreciation to our amazing reviewers, Raghu, Nita, and Ritesh, for their invaluable feedback. It has been a pleasure working with the Packt team, whose support made this process so much smoother. My heartfelt thanks to the entire Amazon Redshift team for their guidance and encouragement. I couldn't have done this without all of you!

About the reviewers

Raghu Kuppala is a Redshift Specialist Solutions Architect at **Amazon Web Services (AWS)**. He has 15 years of experience in architecting Business Intelligence and Analytics applications. At AWS, he helps customers—from startups to Fortune 500 companies—design scalable data warehouse and analytics solutions that meet both predictable and unpredictable demands across various industries. Based in Houston, Texas, Raghu enjoys exploring local coffee shops.

Ritesh Kumar Sinha is a Senior Analytics Specialist Solution Architect at **Amazon Web Services (AWS)**. He holds a bachelor's degree in computer science and engineering and has over 19 years of experience in building data management solutions. At AWS, he specializes in helping Amazon Redshift customers design scalable data warehouse architectures. He is passionate about creating workshops, authoring blogs, and developing reusable solutions for customers. Based in San Francisco, Ritesh lives with his wife and two daughters. He deeply values the guidance of his parents and continues to learn from them. Outside of work, he finds joy in gardening, acquiring new skills, and cherishing quality time with friends and family.

Nita Shah is a Senior Analytics Specialist Solution Architect at AWS based out of New York. Nita has a bachelor's degree in electrical engineering and a master's in computer science. She has over 20 years of experience in architecting and building end-to-end data pipelines in the data management space. She loves to help customers design end-to-end analytics solutions on AWS. In the past, Nita has worked in the retail, insurance, and telecommunication industries. Outside of work, she enjoys traveling.

Table of Contents

Preface xxiii

Chapter 1: Getting Started with Amazon Redshift 1

Creating an Amazon Redshift Serverless data warehouse using the AWS Console **2**

Getting ready • 3

How to do it... • 3

How it works... • 6

Creating an Amazon Redshift provisioned cluster using the AWS Console **6**

Getting ready • 6

How to do it... • 6

Creating an Amazon Redshift Serverless cluster using AWS CloudFormation **8**

Getting ready • 8

How to do it... • 8

How it works... • 10

Creating an Amazon Redshift provisioned cluster using AWS CloudFormation **12**

Getting ready • 13

How to do it... • 13

Connecting to a data warehouse using Amazon Redshift query editor v2 ..... **17**

Getting ready • 17

How to do it... • 17

Connecting to Amazon Redshift using SQL Workbench/J client 20

Getting ready • 20

How to do it... • 20

Connecting to Amazon Redshift using Jupyter Notebook 23

Getting ready • 24

How to do it... • 24

Connecting to Amazon Redshift programmatically using Python and the Redshift Data API . 28

Getting ready • 28

How to do it... • 29

Connecting to Amazon Redshift using Command Line (psql) 30

Getting ready • 31

How to do it... • 31

Chapter 2: Data Management 33

Technical requirements 34

Managing a database in Amazon Redshift 34

Getting ready • 34

How to do it... • 34

Managing a schema in a database 36

Getting ready • 36

How to do it... • 36

Managing tables in a database 38

Getting ready • 39

How to do it... • 39

How it works... • 42

See also... • 42

Managing views in a database 43

Getting ready • 43

How to do it... • 43

Managing materialized views in a database 45

Getting ready • 45

How to do it... • 45

How it works... • 48

Managing stored procedures in a database 49

Getting ready • 49

How to do it... • 49

How it works... • 52

See also... • 52

Managing UDFs in a database 52

Getting ready • 53

How to do it... • 53

How it works... • 56

See also... • 56

Chapter 3: Loading and Unloading Data 57

Technical requirements 58

Loading data from Amazon S3 using COPY 58

Getting ready • 59

How to do it... • 59

How it works... • 64

See also... • 64

Loading data from Amazon DynamoDB 65

Getting ready • 65

How to do it... • 66

How it works... • 67

Updating and inserting data 68

Getting ready • 68

How to do it... • 68

Ingesting data from transactional sources using AWS DMS 74

Getting ready • 74

How to do it... • 75

How it works... • 83

See also... • 83

Cataloging and ingesting data using AWS Glue 84

Getting ready • 84

How to do it... • 84

How it works... • 89

Streaming data to Amazon Redshift via Amazon Data Firehose 89

Getting ready • 90

How to do it... • 90

How it works... • 95

Unloading data to Amazon S3 95

Getting ready • 95

How to do it... • 96

See also... • 97

Chapter 4: Zero-ETL Ingestions 99

Technical requirements 100

Ingesting data from Aurora MySQL/Aurora Postgres/RDS MySQL using zero-ETL integration 100

Getting ready • 101

How to do it... • 101

How it works... • 109

Ingesting data from Amazon DynamoDB using zero-ETL integration 110

Getting ready • 110

How to do it... • 110

How it works... • 117

Ingesting data from SaaS applications like Salesforce using zero-ETL integration 117

Getting ready • 118

How to do it... • 120

Ingesting streaming data from Amazon Kinesis Data Streams (KDS) 129

Getting ready • 129

How to do it... • 130

Ingesting streaming data from Amazon Managed Streaming for Apache Kafka (MSK) 133

Getting ready • 134

How to do it... • 134

How it works... • 135

Near-real-time ingestion of data from Amazon S3 using auto-copy 136

Getting ready • 137

How to do it... • 138

How it works... • 142

Chapter 5: Scalable Data Orchestration for Automation 143

Technical requirements 144

Scheduling queries using Amazon Redshift Query Editor V2 145

Getting ready • 145

How to do it... • 145

How it works... • 150

Event-driven applications using Amazon EventBridge on Amazon Redshift provisioned clusters 150

Getting ready • 151

How to do it... • 151

How it works... • 159

Event-driven applications using AWS Lambda on Amazon Redshift provisioned clusters 160

Getting ready • 160

How to do it... • 161

How it works... • 165

See also... • 165

Orchestration using AWS Step Functions on provisioned clusters 165

Getting ready • 166

How to do it... • 166

How it works... • 171

See also... • 172

Orchestration using Amazon Managed Workflows for Apache Airflow on provisioned clusters 172

Getting ready • 172

How to do it... • 173

How it works... • 179

Chapter 6: Platform Authorization and Security 181

Technical requirements 182

Managing infrastructure security 183

Getting ready • 183

How to do it... • 183

Data encryption at rest 188

Getting ready • 188

How to do it... • 188

Data encryption in transit 191

Getting ready • 191

How to do it... • 191

Managing superusers using an Amazon Redshift provisioned cluster 194

Getting ready • 194

How to do it... • 194

See also... • 195

Using IAM authentication to generate database user credentials for Amazon Redshift serverless clusters 195

Getting ready • 196

How to do it... • 196

Managing audit logs 197

Getting ready • 198

How to do it... • 198

How it works... • 201

Monitoring Amazon Redshift 202

Getting ready • 202

How to do it... • 202

How it works... • 206

Single sign-on using AWS IAM Identity Center 206

Getting ready • 206

How to do it... • 207

How it works... • 218

See also... • 219

Metadata security 219

Getting ready • 219

How to do it... • 220

How it works... • 222

Chapter 7: Data Authorization and Security 223

Technical requirements 223

Implementing RBAC 224

Getting ready • 224

How to do it... • 224

How it works... • 228

Implementing column-level security 229

Getting ready • 229

How to do it... • 229

How it works... • 230

Implementing row-level security 231

Getting ready • 231

How to do it... • 231

How it works... • 234

Implementing dynamic data masking 235

Getting ready • 235

How to do it... • 235

How it works... • 240

Chapter 8: Performance Optimization 241

Technical requirements 242

Configuring Amazon Redshift Advisor for provisioned clusters 243

Getting ready • 243

How to do it... • 244

How it works... • 245

Managing column compression 245

Getting ready • 245

How to do it... • 246

How it works... • 249

Managing data distribution 249

Getting ready • 250

How to do it... • 251

How it works... • 254

Managing the sort key 254

Getting ready • 255

How to do it... • 255

How it works... • 260

Analyzing and improving queries for provisioned clusters 261

Getting ready • 261

How to do it... • 261

How it works... • 264

Configuring Workload Management (WLM) for provisioned cluster 265

Getting ready • 265

How to do it... • 265

How it works... • 269

Utilizing concurrency scaling for provisioned clusters 270

Getting ready • 270

How to do it... • 271

How it works... • 273

Optimizing Spectrum queries for provisioned clusters 274

Getting ready • 274

How to do it... • 274

How it works... • 277

Chapter 9: Cost Optimization 279

Technical requirements 280

AWS Trusted Advisor 280

Getting ready • 280

How to do it... • 281

How it works... • 282

Amazon Redshift Reserved Instance pricing 283

Getting ready • 283

How to do it... • 284

See also... • 286

Scheduling pause and resume for Amazon Redshift provisioned cluster 286

Getting ready • 287

How to do it... • 287

How it works... • 289

Scheduling elastic resizing for an Amazon Redshift provisioned cluster 289

Getting ready • 290

How to do it... • 290

How it works... • 292

Using cost controls to set actions for Redshift Spectrum 293

Getting ready • 293

How to do it... • 293

See also... • 296

Using cost controls to set actions for concurrency scaling for an Amazon provisioned cluster 296

Getting ready • 296

How to do it... • 296

See also... • 298

Using cost controls for Redshift Serverless 298

Getting ready • 298

How to do it... • 298

How it works... • 302

Chapter 10: Lakehouse Architecture 305

Technical requirements 306

Building a data lake catalog using AWS Lake Formation 307

Getting ready • 308

How to do it... • 309

How it works... • 324

Carrying out a data lake export from Amazon Redshift 324

Getting ready • 324

How to do it... • 324

Extending a data warehouse using Amazon Redshift Spectrum 328

Getting ready • 328

How to do it... • 328

Querying an operational source using a federated query 331

Getting ready • 331

How to do it... • 332

Amazon SageMaker Lakehouse 337

Getting ready • 339

How to do it... • 339

How it works... • 346

Chapter 11: Data Sharing with Amazon Redshift 347

Technical requirements 349

Data sharing read access across multiple Amazon Redshift data warehouses 350

Getting ready • 350

How to do it... • 351

How it works... • 353

See also... • 353

Data sharing write access across multiple Amazon Redshift data warehouses 353

Getting ready • 353

How to do it... • 354

How it works... • 355

Data sharing using Amazon DataZone for cross-collaboration and self-service analytics 356

Getting ready • 356

How to do it... • 356

How it works... • 376

Data sharing using AWS Data Exchange for monetization and subscribing to third-party data 377

Getting ready • 378

How to do it... • 378

How it works... • 381

Chapter 12: Generative AI and ML with Amazon Redshift 383

Technical requirements 384

Building SQL queries automatically using Amazon Q generative SQL 384

Getting ready • 385

How to do it... • 385

How it works... • 387

Managing Amazon Redshift ML 387

Getting ready • 388

How to do it... • 389

How it works... • 391

Using LLMs in Amazon Bedrock using SQL statements 391

Getting ready • 392

How to do it... • 392

How it works... • 395

Using LLMs in Amazon SageMaker Jumpstart using SQL statements 395

Getting ready • 395

How to do it... • 396

How it works... • 401

Querying your data with natural language prompts using Amazon Bedrock knowledge bases for Amazon Redshift 401

Getting ready • 401

How to do it... • 402

How it works... • 408

Generative BI with Amazon Q with QuickSight querying an Amazon Redshift dataset ... 409

Getting ready • 409

How to do it... • 410

How it works... • 420

Appendix 421

Recipe 1: Creating an IAM user 421

Recipe 2: Storing database credentials using AWS Secrets Manager 422

Recipe 3: Creating an IAM role for an AWS service 422

Recipe 4: Attaching an IAM role to the Amazon Redshift cluster 423

Other Books You May Enjoy 426

Index 429

Preface

Amazon Redshift is a fully managed, petabyte-scale AWS cloud data warehousing service. It enables you to build new data warehouse workloads on AWS and migrate on-premises traditional data warehousing platforms to Redshift.

This book on Amazon Redshift starts by focusing on the Redshift architecture, showing you how to perform database administration tasks on Redshift. You'll then learn how to optimize your data warehouse to quickly execute complex analytic queries against very large datasets. Because of the massive amount of data involved in data warehousing, designing your database for analytical processing lets you take full advantage of Redshift's columnar architecture and managed services. As you advance, you'll discover how to deploy fully automated and highly scalable **extract, transform, and load** (**ETL**) processes, which help minimize the operational efforts that you have to invest in managing regular ETL pipelines and ensure the timely and accurate refreshing of your data warehouse. You'll gain a clear understanding of Redshift use cases, data ingestion, data management, security, and scaling so that you can build a scalable data warehouse platform. Finally, you'll learn emerging trends in utilizing Redshift data warehouses to enable AI/ML use cases.

By the end of this Redshift book, you'll be able to implement a Redshift-based data analytics solution and will have understood the best-practice solutions to commonly faced problems.

Who this book is for

This book is for anyone involved in architecting, implementing, and optimizing an Amazon Redshift data warehouse, such as data warehouse developers, data analysts, database administrators, data engineers, and data scientists. Basic knowledge of data warehousing, database systems, and cloud concepts and familiarity with Redshift would be beneficial.

What this book covers

Chapter 1, *Getting Started with Amazon Redshift*, discusses how Amazon Redshift is a fully managed, petabyte-scale data warehouse service in the cloud. An Amazon Redshift data warehouse comes in two deployment options: provisioned clusters (with one leader node and multiple compute nodes) and serverless (with automatic provisioning and scaling). Amazon Redshift integrates with lakehouse architecture for unified access to structured and semi-structured data. This chapter provides hands-on guidance for creating and connecting to Amazon Redshift resources through various methods.

Chapter 2, *Data Management*, discusses how a data warehouse system has very different design goals compared to a typical transaction-oriented relational database system for **online transaction processing (OLTP)**. Amazon Redshift is optimized for the very fast execution of complex analytic queries against very large datasets. Because of the massive amounts of data involved in data warehousing, designing your database for analytical processing lets you take full advantage of the columnar architecture and managed service. This chapter delves into the different data structure options to set up an analytical schema for the easy querying of your end users.

Chapter 3, *Loading and Unloading Data*, looks at how Amazon Redshift has in-built integrations with data lakes and other analytical services and how it is easy to move and analyze data across different services. This chapter discusses scalable options to move large datasets from a data lake based out of Amazon S3 storage, as well as AWS analytical services such as Amazon DynamoDB, ingesting from transactional sources using AWS DMS, cataloging with AWS Glue, and streaming via Amazon Kinesis Data Firehose.

Chapter 4, *Zero-ETL Ingestions*, introduces AWS zero-ETL as a revolutionary suite of fully managed integrations that streamline data analytics processes. This chapter explores how zero-ETL eliminates traditional ETL complexities by automatically replicating data from operational sources to analytical destinations, enabling real-time insights without the need for complex data pipeline management. It covers various zero-ETL integration methods, including native database integrations, ingestion from SaaS applications, streaming data ingestion, and near-real-time ingestion from Amazon S3 using auto-copy. These solutions significantly reduce time to insight, ensure data consistency, and allow organizations to scale their data operations efficiently while maintaining separation between transactional and analytical workloads, ultimately enabling faster, data-driven decision-making with reduced operational overhead and technical complexity.

Chapter 5, *Scalable Data Orchestration for Automation*, explores AWS's comprehensive suite of native services for workflow integration and automation. The chapter focuses on ETL process workflows for data warehouse refreshes, demonstrating how different tasks can be managed independently using purpose-built services. It covers various orchestration methods, including query scheduling, event-driven applications, workflow orchestration, and pipeline management. The chapter emphasizes how these tools enable the efficient management of complex data pipelines originating from various sources, supporting downstream applications such as machine learning pipelines, analytics dashboards, and business reports.

Chapter 6, *Platform Authorization and Security,* explores Amazon Redshift's comprehensive security features designed to meet the requirements of security-sensitive organizations within the AWS Shared Responsibility Model. The chapter covers essential security aspects, including infrastructure security, data encryption, authentication, and metadata security. The chapter emphasizes how these built-in features provide a robust security framework for protecting data while maintaining fine-grained access controls for underlying data structures.

Chapter 7, *Data Authorization and Security*, focuses on Amazon Redshift's granular data access control mechanisms for protecting sensitive information. The chapter explores key security features focusing on fine-grained access control. These features work together to create a comprehensive security framework that ensures users can only access and modify data according to their authorization level, providing precise control over data visibility and manipulation rights.

Chapter 8, *Performance Optimization*, examines how Amazon Redshift, being a fully managed service, provides great performance out of the box for most workloads. Amazon Redshift also provides you with levers that help you maximize the throughputs when data access patterns are already established. Performance tuning on Amazon Redshift helps you manage critical SLAs for workloads and easily scale up your data warehouse to meet/exceed business needs.

Chapter 9, *Cost Optimization*, discusses how Amazon Redshift is one of the best price-performant data warehouse platforms on the cloud. Amazon Redshift also provides you with scalability and different options to optimize the pricing, such as elastic resizing, pause and resume, Reserved Instances, and using cost controls. These options allow you to create the best price-performant data warehouse solution.

Chapter 10, *Lakehouse Architecture*, explores how Amazon Redshift serves as the foundation for the lakehouse architectural pattern, enabling seamless data access across various analytics solutions while preventing data silos. The chapter demonstrates how enterprises can query data across data lakes, operational databases, and multiple data warehouses without constant data movement. These patterns support unified data management, consistent security and governance, and flexible query engine usage, while maintaining compatibility with both AWS services and third-party tools through standard Iceberg APIs. The architecture enables organizations to efficiently combine data lakes, data warehouses, and purpose-built data stores under unified governance.

Chapter 11, *Data Sharing with Amazon Redshift*, explores Amazon Redshift's capability to securely share data across different Redshift data warehouses, AWS accounts, and Regions without physical data movement. This feature, enabled by Redshift's decoupled storage-compute architecture, provides live, transactionally consistent data views. Key benefits include workload isolation, clear cost chargeback, cross-collaboration, scalable read/write access, and potential data monetization. The chapter discusses common deployment patterns such as hub and spoke and data mesh, which facilitate multi-warehouse architectures. These capabilities enable organizations to implement flexible, efficient data-sharing strategies that support various business needs, from internal collaboration to external data services.

Chapter 12, *Generative AI and ML with Amazon Redshift*, explores the integration of machine learning and generative AI capabilities within Amazon Redshift's data warehouse environment. The chapter demonstrates how Redshift enables users to create, train, and deploy ML models directly within the warehouse, supporting both traditional supervised learning and advanced generative AI applications. These capabilities showcase how Amazon Redshift combines traditional data warehousing with cutting-edge AI capabilities to enhance data analytics and streamline processes like forecasting, sentiment analysis, and query authoring.

To get the most out of this book

You will need access to an AWS account to perform all the recipes in this book. You will need either administrator access to the AWS account or to work with an administrator who can help create the IAM user, roles, and policies as listed in the different chapters. All the data needed in the setup is provided as steps in the recipes, and the Amazon S3 bucket is hosted in the Europe (Ireland) (eu-west-1) AWS Region. It is preferable to use the Europe (Ireland) AWS Region to execute all the recipes. If you need to run the recipes in a different Region, you will need to copy the data from the source bucket (`s3://packt-redshift -cookbook/`) to an Amazon S3 bucket in the desired AWS Region, and use that in your recipes instead.

Download the example code files

The code bundle for the book is hosted on GitHub at `https://github.com/PacktPublishing/Amazon-Redshift-Cookbook-2E`. We also have other code bundles from our rich catalog of books and videos available at `https://github.com/PacktPublishing`. Check them out!

Download the color images

We also provide a PDF file that has color images of the screenshots/diagrams used in this book. You can download it here: `https://packt.link/gbp/9781836206910`.

Conventions used

There are a number of text conventions used throughout this book.

`CodeInText`: Indicates code words in text, database table names, folder names, filenames, file extensions, pathnames, dummy URLs, user input, and Twitter/X handles. For example: "Execute the `terraform graph` command."

A block of code is set as follows:

```
resource "azurerm_resource_group" "rg-app" {
  name     = "RG-APP-${terraform.workspace}"
  location = "westeurope"
```

When we wish to draw your attention to a particular part of a code block, the relevant lines or items are set in bold:

```
terraform {
  backend "azurerm" {
    resource_group_name  = "RG-TFBACKEND"
    storage_account_name = "storagetfbackend"
    container_name       = "tfstate"
    key                  = "myapp.tfstate"
    access_key           = xxxxxx-xxxxx-xxx-xxxxx
  }
}
```

Any command-line input or output is written as follows:

```
terraform init
```

Bold: Indicates a new term, an important word, or words that you see on the screen. For instance, words in menus or dialog boxes appear in the text like this. For example: "Select **System info** from the **Administration** panel."

Warnings or important notes appear like this..

Tips and tricks appear like this.

Get in touch

Feedback from our readers is always welcome.

General feedback: Email `feedback@packtpub.com` and mention the book's title in the subject of your message. If you have questions about any aspect of this book, please email us at `questions@packtpub.com`.

Errata: Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you have found a mistake in this book, we would be grateful if you reported this to us. Please visit `http://www.packtpub.com/submit-errata`, click **Submit Errata**, and fill in the form.

Piracy: If you come across any illegal copies of our works in any form on the internet, we would be grateful if you would provide us with the location address or website name. Please contact us at `copyright@packtpub.com` with a link to the material.

If you are interested in becoming an author: If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, please visit `http://authors.packtpub.com/`.

Share your thoughts

Once you've read *Amazon Redshift Cookbook*, we'd love to hear your thoughts! Please click here to go straight to the Amazon review page for this book and share your feedback.

Your review is important to us and the tech community and will help us make sure we're delivering excellent quality content.

Download a free PDF copy of this book

Thanks for purchasing this book!

Do you like to read on the go but are unable to carry your print books everywhere?

Is your eBook purchase not compatible with the device of your choice?

Don't worry, now with every Packt book you get a DRM-free PDF version of that book at no cost.

Read anywhere, any place, on any device. Search, copy, and paste code from your favorite technical books directly into your application.

The perks don't stop there, you can get exclusive access to discounts, newsletters, and great free content in your inbox daily.

Follow these simple steps to get the benefits:

1. Scan the QR code or visit the link below:

`https://packt.link/free-ebook/9781836206910`

2. Submit your proof of purchase.
3. That's it! We'll send your free PDF and other benefits to your email directly.

1

Getting Started with Amazon Redshift

Amazon Redshift is a fast, fully managed, petabyte-scale data warehouse service that makes it simple and cost-effective to efficiently analyze all your data using your existing business intelligence tools. It is optimized for datasets ranging from a few hundred gigabytes to a petabyte or more and costs less than $1,000 per terabyte per year, a tenth of the cost of most traditional data warehousing solutions. Amazon Redshift integrates into the data lake solution though the lakehouse architecture, allowing you to access all the structured and semi-structured data in one place. Each Amazon Redshift data warehouse is hosted as either a provisioned cluster or serverless. The Amazon Redshift provisioned data warehouse consists of one leader node and a collection of one or more compute nodes, which you can scale up or down as needed. The Amazon Redshift serverless data warehouse's resources are automatically provisioned, and data warehouse capacity is intelligently scaled based on workload patterns. This chapter walks you through the process of creating a sample Amazon Redshift resource and connecting to it from different clients.

The following recipes are discussed in this chapter:

- Creating an Amazon Redshift Serverless data warehouse using the AWS console
- Creating an Amazon Redshift provisioned cluster using the AWS console
- Creating an Amazon Redshift Serverless cluster using AWS CloudFormation
- Creating an Amazon Redshift provisioned cluster using AWS CloudFormation
- Connecting to a data warehouse using Amazon Redshift query editor v2
- Connecting to Amazon Redshift using the SQL Workbench/J client

- Connecting to Amazon Redshift using Jupyter Notebook
- Connecting to Amazon Redshift programmatically using Python and the Redshift API
- Connecting to Amazon Redshift using the command line (psql)

Technical requirements

Here is a list of the technical requirements for this chapter:

- An AWS account.
- The AWS administrator should create an IAM user by following *Recipe 1* in the *Appendix*. This IAM user will be used to execute all the recipes.
- The AWS administrator should deploy the AWS CloudFormation template to attach the IAM policy to the IAM user, which will give them access to Amazon Redshift, Amazon SageMaker, Amazon EC2, AWS CloudFormation, and AWS Secrets Manager. The template is available here: `https://github.com/PacktPublishing/Amazon-Redshift-Cookbook-2E/blob/main/Chapter01/chapter_1_CFN.yaml`.
- Client tools such as SQL Workbench/J, an IDE, and a command-line tool.
- Ensure your clients have network access to the VPC in which the Amazon Redshift data warehouse is deployed: `https://docs.aws.amazon.com/redshift/latest/mgmt/managing-clusters-vpc.html`.
- The code files for the chapter can be found here: `https://github.com/PacktPublishing/Amazon-Redshift-Cookbook-2E/tree/main/Chapter01`.

Creating an Amazon Redshift Serverless data warehouse using the AWS Console

The AWS Management Console allows you to interactively create an Amazon Redshift serverless data warehouse via a browser-based user interface. Once the data warehouse has been created, you can use the Console to monitor its health and diagnose query performance issues from a unified dashboard. In this recipe, you will learn how to use the unified dashboard to deploy a Redshift serverless.

Getting ready

To complete this recipe, you will need:

- An existing or new AWS Account. If new AWS accounts need to be created, go to `https://portal.aws.amazon.com/billing/signup`, enter the information, and follow the steps on the site.
- An IAM user with access to Amazon Redshift.

How to do it...

The following steps will enable you to create a cluster with minimal parameters:

1. Navigate to the AWS Management Console and select Amazon Redshift, `https://console.aws.amazon.com/redshiftv2/`.
2. Choose the AWS Region (`eu-west-1`) or the corresponding region at the top right of the screen and then click **Next**.
3. On the Amazon Redshift console, in the left navigation pane, choose **Serverless Dashboard**, and then click **Create workgroup**, as shown in *Figure 1.1*:

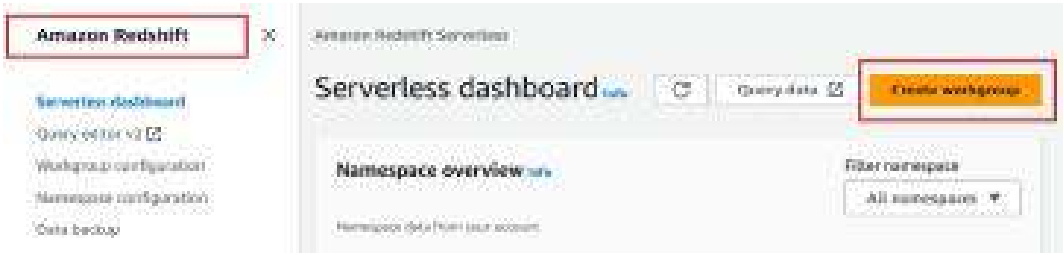


Figure 1.1 – Creating an Amazon Redshift Serverless workgroup

4. In the Workgroup section, type any meaningful **Workgroup Name** like `my-redshift-wg`.

5. In the Performance and cost controls section, you can choose the compute capacity for the workgroup. You have two options to choose from:

    - **Price-performance target** (recommended): This option allows Amazon Redshift Serverless to learn your workload patterns by analyzing factors such as query complexity and data volumes. It automatically adjusts compute capacity throughout the day based on your needs. You can set your price-performance target using a slider:

        - Left: Optimizes for cost
        - Middle: Balances cost and performance
        - Right: Optimizes for performance

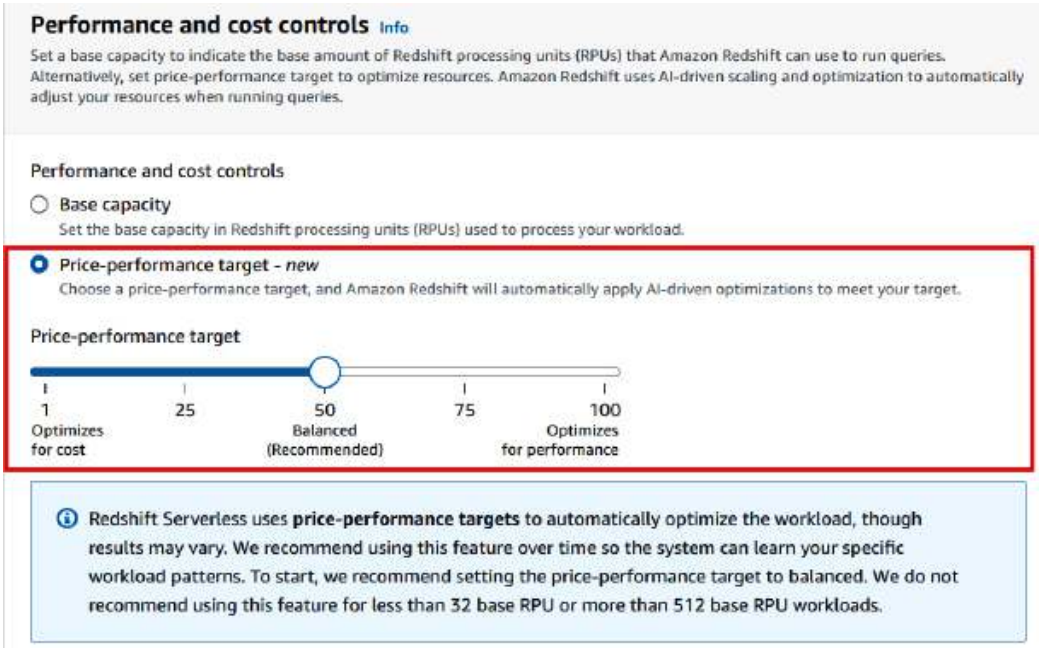


Figure 1.2 – Price performance target option

    - **Base capacity**: With this option, you will choose a static base compute capacity for the workgroup. Use this option only if you believe that you understand the workload characteristics well and want control of the compute capacity. Using the drop-down for **Base capacity**, you can choose a number for Redshift processing units (RPUs) between `8` and `1024`, as shown in the following screenshot. RPU is a measure of compute capacity.

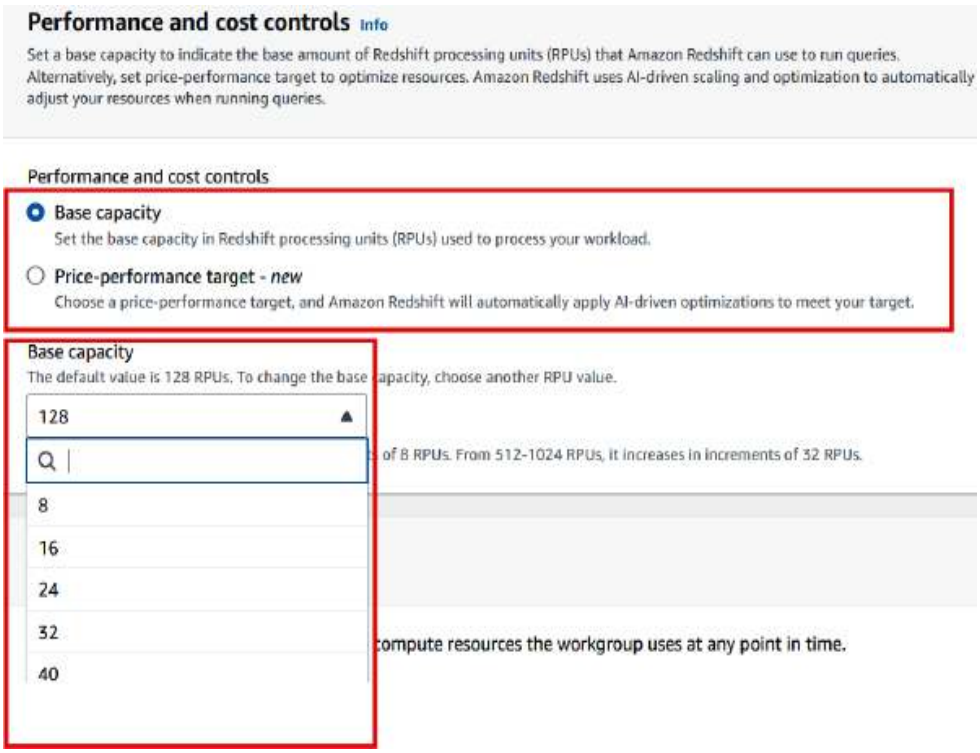


Figure 1.3 – Base capacity option

6. In the Network and security section, set **IP address type to IPv4**.
7. In the Network and security section, select the appropriate **Virtual private cloud (VPC)**, **VPC security groups**, and **Subnet**.
8. If your workload needs network traffic between your serverless database and data repositories routed through a VPC instead of the internet, then enable **Turn on enhanced VPC routing** by checking the box. For this book, we will leave it unchecked and then click **Next**.
9. In the Namespace section, select **Create a new Namespace** and type any meaningful name for **Namespace** like `my-redshift-ns`.
10. In the Database name and password section, leave the defaults as is, which will create a default database called `dev` and give the IAM credentials you are using as default admin user credentials.
11. In the **Permissions** section, leave all the settings as default.

12. In the Encryption and security section, leave all the settings at the defaults and then click **Next**.
13. In the Review and create section, validate that all the settings are correct and then click **Create**.

How it works...

The Amazon Redshift serverless data warehouse consists of a namespace, which is a collection of database objects and users, and a workgroup, which is a collection of compute resources. Namespaces and workgroups are scaled and billed independently. Amazon Redshift Serverless automatically provisions and scales the compute capacity based on the usage, when required. You only pay for a workgroup when the queries are run, there is no compute charge for idleness. Similarly, you only pay for the volume of data stored in the namespace.

Creating an Amazon Redshift provisioned cluster using the AWS Console

The AWS Management Console allows you to interactively create an Amazon Redshift provisioned cluster via a browser-based user interface. It also recommends the right cluster configuration based on the size of your workload. Once the cluster has been created, you can use the Console to monitor the health of the cluster and diagnose query performance issues from a unified dashboard.

Getting ready

To complete this recipe, you will need:

- An existing or new AWS account. If new AWS accounts need to be created, go to `https://portal.aws.amazon.com/billing/signup`, enter information, and follow the steps on the site.
- An IAM user with access to Amazon Redshift.

How to do it...

The following steps will enable you to create a cluster with minimal parameters:

1. Navigate to the AWS Management Console, select Amazon Redshift, `https://console.aws.amazon.com/redshiftv2/`, and browse to **Provisioned clusters dashboard**.
2. Choose the AWS Region (`eu-west-1`) or the corresponding region in the top right of the screen.
3. On the Amazon Redshift dashboard, select **CLUSTERS**, then click **Create cluster**.

4. In the Cluster configuration section, type any meaningful **Cluster identifier** like `myredshiftcluster`.
5. Choose either **Production** or **Free trial** depending on what you plan to use this cluster.
6. If you need help determining the right size for your compute cluster, select the **Help me choose** option. Alternatively, if you know the required size of your cluster (that is, the node type and number of nodes), select **I'll choose**. For example, you can choose **Node type**: **ra3.xlplus** with **Nodes**: **2**.

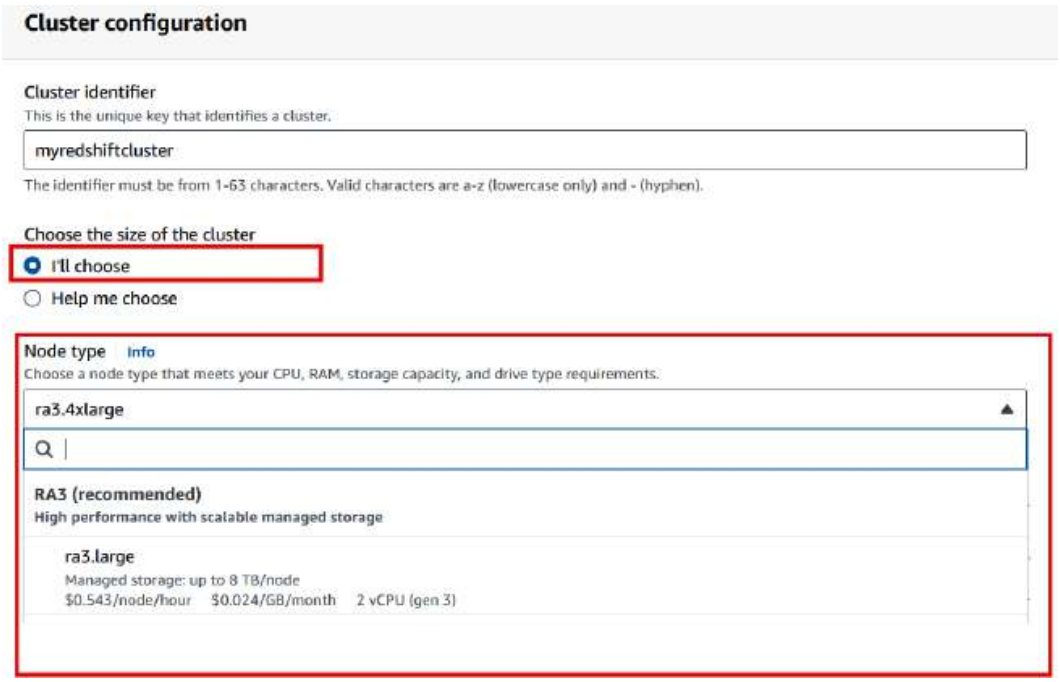


Figure 1.4 – Create Amazon Redshift provisioned cluster

7. In the Database configuration section, specify values for **Database name** (optional), **Database port** (optional), **Master user name**, and **Master user password**. For example:
    - **Database name** (optional): Enter `dev`.
    - **Database port** (optional): Enter `5439`.
    - **Master user name**: Enter `awsuser`.
    - **Master user password**: Enter a value for the password. Refer to PASSWORD parameter at `https://docs.aws.amazon.com/redshift/latest/dg/r_CREATE_USER.html#r_CREATE_USER-parameters` to understand password requirements.

8. Optionally, you can configure the **Cluster permissions** and **Additional configurations** section when you want to pick specific network and security configurations. The console defaults to the preset configuration otherwise.
9. Choose **Create cluster**.
10. The cluster creation takes a few minutes to complete. Navigate to the cluster, select **Query data**, and click on **Query in Query Editor v2** to connect to the cluster.

Creating an Amazon Redshift Serverless cluster using AWS CloudFormation

With an AWS `CloudFormation` template, you treat your infrastructure as code. This enables you to create the Amazon Redshift cluster using `json/yaml` file. The declarative code in the file contains the steps to create the AWS resources, and enables easy automation and distribution. This template allows you to standardize the Amazon Redshift creation to meet your organizational infrastructure and security standards. Further, you can distribute them to different teams within your organization using the AWS service catalog for an easy setup. In this recipe, you will learn how to use CloudFormation template to deploy an Amazon Redshift Serverless cluster and the different parameters associated with it.

Getting ready

To complete this recipe, you will need:

- An IAM user with access to AWS CloudFormation, Amazon EC2, and Amazon Redshift

How to do it...

We use a `CloudFormation` template to create the Amazon Redshift Serverless infrastructure as code using a JSON-based template. Follow these steps to create the Amazon Redshift using the Cloud Formation template:

1. Download the AWS CloudFormation template from here: `https://github.com/PacktPublishing/Amazon-Redshift-Cookbook-2E/blob/main/Chapter01/Create_Amazon_Redshift_Serverless.yaml`.
2. Navigate to the AWS Console, choose **CloudFormation**, and choose **Create stack**. Click on the **Template is ready** and **Upload a template file** options, choose the downloaded `Creating_Amazon_Redshift_Serverless.yaml` file from your local computer, and click **Next**.

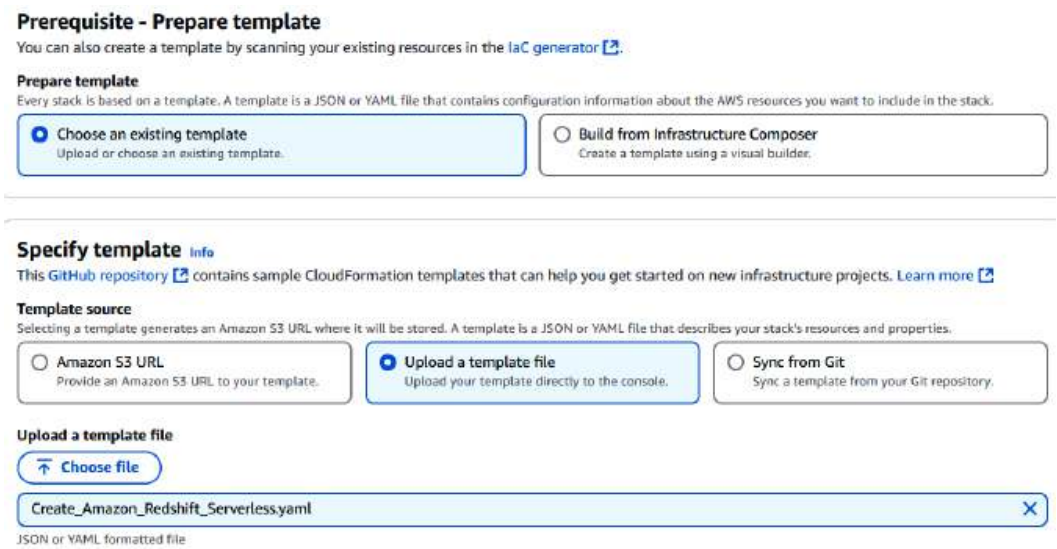


Figure 1.5 – Choose the CloudFormation template file

3. Choose the following input parameters:

    a. **Stack name**: Enter a name for the stack, for example, `myredshiftserverless`.

    b. **NamespaceName**: Enter a name for namespace, which is a collection of database objects and users.

    c. **WorkgroupName**: Enter a name for the workgroup, which is a collection of compute resources.

    d. **BaseRPU**: The base RPU for Redshift Serverless Workgroup ranges from 8 to 1024. The default is 8.

    e. **DatabaseName**: Enter a database name, for example, `dev`.

    f. **AdminUsername**: Enter an admin username, for example, `awsuser`.

    g. **AdminPassword**: Enter an admin user password. The password must be 8-64 characters long and must contain at least one uppercase letter, one lowercase letter, and one number. It can include any printable ASCII character except `/`, `""`, and `@`. The default is `Awsuser123`.

4. Click **Next** and **Create Stack**.

    AWS `CloudFormation` has deployed all the infrastructure and configuration listed in the template in completed and we'll wait till the status changes to **CREATE_COMPLETE**.

How it works...

Let's now see how this `CloudFormation` template works. The **CloudFormation** template is organized into three broad sections: input parameters, resources, and outputs. Let's discuss them one by one.

The parameters section is used to allow user input choices and also can be used to apply constraints against its value. To create the Amazon Redshift Serverless cluster, we collect parameters such as namespace name, workgroup name, base RPU, database name, and admin username/ password. The parameters will later be substituted when creating the resources. Here is the Parameters section from the template:

```
Parameters:
  NamespaceName:
    Description: The name for namespace, which is a collection of database
objects and users
    Type: String
  WorkgroupName:
    Description: The name for workgroup, which is a collection of compute
resources
    Type: String
  BaseRPU:
    Description: Base RPU for Redshift Serverless Workgroup.
    Type: Number
    MinValue: '8'
    MaxValue: '1024'
    Default: '8'
    AllowedValues:
      - 8
      - 512
      - 1024
  DatabaseName:
    Description: The name of the first database to be created in the
serverless data warehouse
    Type: String
    Default: dev
    AllowedPattern: ([a-z]|[0-9])+
  AdminUsername:
    Description: The user name that is associated with the admin user
```

```
account for the serverless data warehouse
    Type: String
    Default: awsuser
    AllowedPattern: ([a-z])([a-z]|[0-9])*
  AdminPassword:
    Description: The password that is associated with the admin user
account for the serverless data warehouse. Default is Awsuser123
    Type: String
    Default: Awsuser123
    NoEcho: 'true'
    MinLength: '8'
    MaxLength: '64'
    AllowedPattern: ^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)[^\x00-\x20\x22\x27\
x2f\x40\x5c\x7f-\uffff]+
```

In the above input section, `DatabaseName` is a string value that defaults to `dev` and also enforces an alphanumeric validation when specified using the condition check of `AllowedPattern: ([a-z]|[0-9])+`. Similarly, `BaseRPU` is defaulted to `8` and allows the valid `BaseRPU` from a list of values.

The Resources section contains a list of resource objects and the Amazon Serverless namespace is invoked using `AWS::RedshiftServerless::Namespace` along with references to input parameters such as `NamespaceName`, `DbName`, `AdminUsername`, and `AdminPassword`. The Amazon Serverless Workgroup is invoked using `AWS::RedshiftServerless::Workgroup` along with references to input parameters such as `NamespaceName`, `WorkgroupName`, `BaseCapacity`, and `PublicAccessible`:

```
Resources:
  Namespace:
    Type: AWS::RedshiftServerless::Namespace
    Properties:
      NamespaceName: !Ref NamespaceName
      AdminUsername: !Ref AdminUsername
      AdminUserPassword: !Ref AdminPassword
      DbName: !Ref DatabaseName
  Workgroup:
    Type: AWS::RedshiftServerless::Workgroup
    Properties:
      NamespaceName: !Ref NamespaceName
      WorkgroupName: !Ref WorkgroupName
```

```
      BaseCapacity: !Ref BaseRPU
      PubliclyAccessible: false
    DependsOn:
      - Namespace
```

The Resources section references the input section for values such as `NamespaceName`, `WorkgroupName`, `BaseRPU`, and `DatabaseName` that will be used when the resource is created.

The Outputs section is a handy way to capture the essential information about your resources or input parameters that you want to have available after the stack is created so you can easily identify the resource object names that are created. For example, you can capture output such as `RedshiftServerlessEndpoint` that will be used to connect into the cluster as follows:

```
Outputs:
RedshiftServerlessEndpoint:
    Description: Redshift Serverless endpoint
    Value:
      Fn::Join:
        - ':'
        - - Fn::GetAtt Workgroup.Endpoint.Address
          - "5439"
```

When authoring the template from scratch, you can take advantage of the `AWS Application Composer` – an integrated development environment for authoring and validating code. Once the template is ready, you can launch the resources by creating a stack (collection of resources), using the AWS CloudFormation console, API, or AWS CLI. You can also update or delete it afterward.

Creating an Amazon Redshift provisioned cluster using AWS CloudFormation

With an AWS `CloudFormation` template, you treat your infrastructure as code. This enables you to create an Amazon Redshift cluster using a JSON or YAML file. The declarative code in the file contains the steps to create the AWS resources and enables easy automation and distribution. This template allows you to standardize the Amazon Redshift provisioned cluster creation to meet your organizational infrastructure and security standards.

Further, you can distribute them to different teams within your organization using the AWS service catalog for an easy setup. In this recipe, you will learn how to use a CloudFormation template to deploy an Amazon Redshift provisioned cluster and the different parameters associated with it.

Getting ready

To complete this recipe, you will need:

- An IAM user with access to AWS CloudFormation, Amazon EC2, and Amazon Redshift

How to do it...

We use the `CloudFormation` template to author the Amazon Redshift cluster infrastructure as code using a JSON-based template. Follow these steps to create the Amazon Redshift provisioned cluster using the CloudFormation template:

1. Download the AWS CloudFormation template from `https://github.com/PacktPublishing/Amazon-Redshift-Cookbook-2E/blob/main/Chapter01/Creating_Amazon_Redshift_Cluster.json`.
2. Navigate to the AWS Console, choose **CloudFormation**, and choose **Create stack.**
3. Click on the **Template is ready** and **Upload a template file** options, choose the downloaded `Creating_Amazon_Redshift_Cluster.json` file from your local computer, and click **Next**.
4. Set the following input parameters:
    a. **Stack name**: Enter a name for the stack, for example, `myredshiftcluster`.
    b. **ClusterType**: Single-node or a multiple node cluster.
    c. **DatabaseName**: Enter a database name, for example, `dev`.
    d. **InboundTraffic**: Restrict the CIDR ranges of IPs that can access the cluster. 0.0.0.0/0 opens the cluster to be globally accessible, which would be a security risk.
    e. **MasterUserName**: Enter a database master user name, for example, `awsuser`.
    f. **MasterUserPassword**: Enter a master user password. The password must be `8-64` characters long and must contain at least one uppercase letter, one lowercase letter, and one number. It can contain any printable ASCII character except `/`, `""`, or `@`.
    g. **NodeType**: Enter the node type, for example, `ra3.xlplus`.
    h. **NumberofNodes**: Enter the number of compute nodes, for example, 2.
    i. **Redshift cluster port**: Choose any TCP/IP port, for example, `5439`.

5. Click **Next** and **Create Stack**.

   AWS `CloudFormation` has deployed all the infrastructure and configuration listed in the template in completed and we'll wait till the status changes to **CREATE_COMPLETE**.

6. You can now check the outputs section in the CloudFormation stack and look for the cluster endpoint, or navigate to the **Amazon Redshift | Clusters | myredshiftcluster | General information** section to find the JDBC/ODBC URL to connect to the Amazon Redshift cluster.

How it works...

Let's now see how this `CloudFormation` template works. The **CloudFormation** template is organized into three broad sections: input parameters, resources, and outputs. Let's discuss them one by one.

The **Parameters** section is used to allow user input choices and also can be used to apply constraints to the values. To create an Amazon Redshift resource, we collect parameters such as database name, master username/ password, and cluster type. The parameters will later be substituted when creating the resources. Here is an illustration of the Parameters section of the template:

```
"Parameters": {
        "DatabaseName": {
            "Description": "The name of the first database to be created
when the cluster is created",
            "Type": "String",
            "Default": "dev",
            "AllowedPattern": "([a-z]|[0-9])+"
        },
        "NodeType": {
            "Description": "The type of node to be provisioned",
            "Type": "String",
            "Default": "ra3.xlplus",
            "AllowedValues": [
                "ra3.16xlarge",
                "ra3.4xlarge",
                "ra3.xlplus",
            ]
        }
```

In the previous input section, `DatabaseName` is a string value that defaults to `dev` and also enforces an alphanumeric validation when specified using the condition check of `AllowedPattern: ([a-z]|[0-9])+`. Similarly, `NodeType` defaults to `ra3.xlplus` and allows the valid `NodeType` from a list of values.

The `Resources` section contains a list of resource objects, and the Amazon resource is invoked using `AWS::Redshift::Cluster` along with references to the input parameters, such as `DatabaseName`, `ClusterType`, `NumberOfNodes`, `NodeType`, `MasterUsername`, and `MasterUserPassword`:

```
"Resources": {
        "RedshiftCluster": {
            "Type": "AWS::Redshift::Cluster",
            "DependsOn": "AttachGateway",
            "Properties": {
                "ClusterType": {
                    "Ref": "ClusterType"
                },
                "NumberOfNodes": {
                   …
                },
                "NodeType": {
                    "Ref": "NodeType"
                },
                "DBName": {
                    "Ref": "DatabaseName"
                },
..
```

The **Resources** section references the input section for values such as `NumberOfNodes`, `NodeType`, `DatabaseName`, that will be used during the resource creation.

The `Outputs` section is a handy place to capture the essential information about your resources or input parameters that you want to have available after the stack has been created, so you can easily identify the resource object names that are created.

For example, you can capture output such as `ClusterEndpoint` that will be used to connect into the cluster as follows:

```
"Outputs": {
        "ClusterEndpoint": {
            "Description": "Cluster endpoint",
            "Value": {
                "Fn::Join": [
                    ":",
                    [
                        {
                            "Fn::GetAtt": [
                                "RedshiftCluster",
                                "Endpoint.Address"
                            ]
                        },
                        {
                            "Fn::GetAtt": [
                                "RedshiftCluster",
                                "Endpoint.Port"
                            ]
                        }
                    ]
                ]
            }
        }
```

When authoring the template from scratch, you can take advantage of the `AWS Application Composer` – an integrated development environment for authoring and validating code. Once the template is ready, you can launch the resources by creating a stack (collection of resources) or using the AWS CloudFormation console, API, or AWS CLI. You can also update or delete the template afterward.

Connecting to a data warehouse using Amazon Redshift query editor v2

The query editor v2 is a client browser-based interface available on the AWS Management Console for running SQL queries on Amazon Redshift Serverless or provisioned cluster directly. Once you have created the data warehouse, you can use query editor to jumpstart querying the cluster without needing to set up the JDBC/ODBC driver. This recipe explains how to use query editor to access a Redshift data warehouse.

Query editor V2 allows you to do the following:

- Explore schemas
- Run multiple DDL and DML SQL commands
- Run single/multiple select statements
- View query execution details
- Save a query
- Download a query result set up to 100 MB in CSV, text, or HTML

Getting ready

To complete this recipe, you will need:

- An Amazon Redshift data warehouse (serverless or provisioned cluster)
- IAM user with access to Amazon Redshift and AWS Secrets Manager
- Store the database credentials in Amazon Secrets Manager using *Recipe 2* in the *Appendix*

How to do it...

Here are the steps to query an Amazon Redshift data warehouse using the Amazon Redshift query editor v2.

1. Navigate to AWS Redshift Console `https://console.aws.amazon.com/redshiftv2` and select the query editor v2 from the navigation pane on the left.
2. Choose the AWS Region (`eu-west-1`) or corresponding region on the top right of the screen.

3. With query editor v2, all the Redshift data warehouses, both serverless and provisioned clusters, are listed on the console. Select the dots beside the cluster name and click **Create connection**:

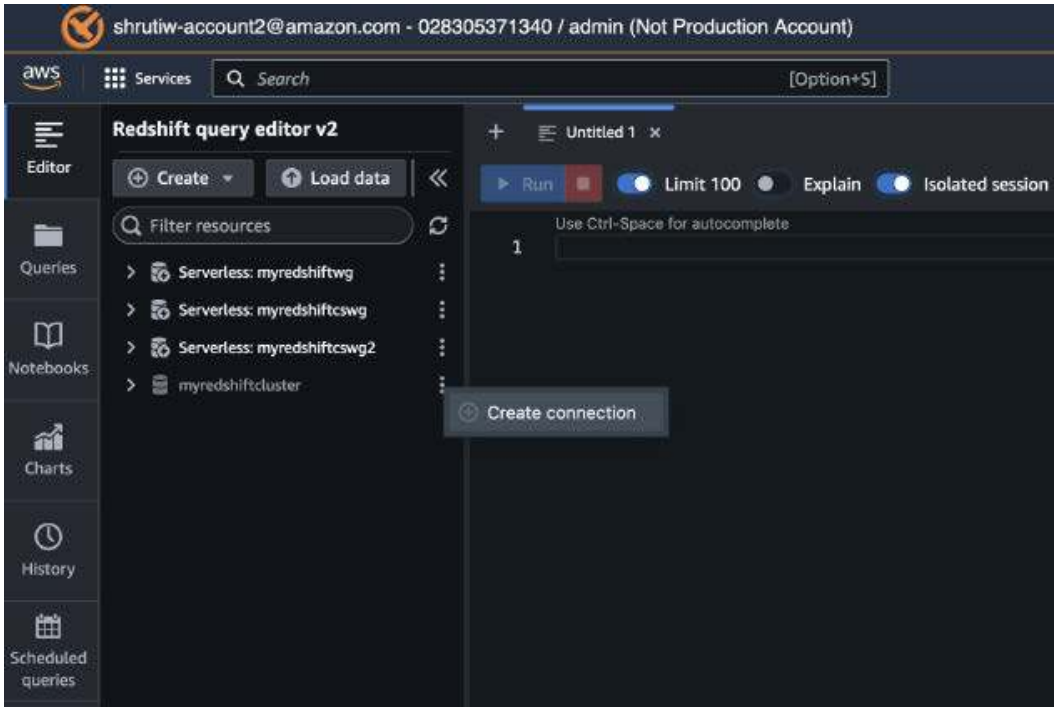


Figure 1.6 – Creating a connection in query editor v2

4. In the connection window, select **Other ways to connect**, and then select **AWS Secrets Manager**:

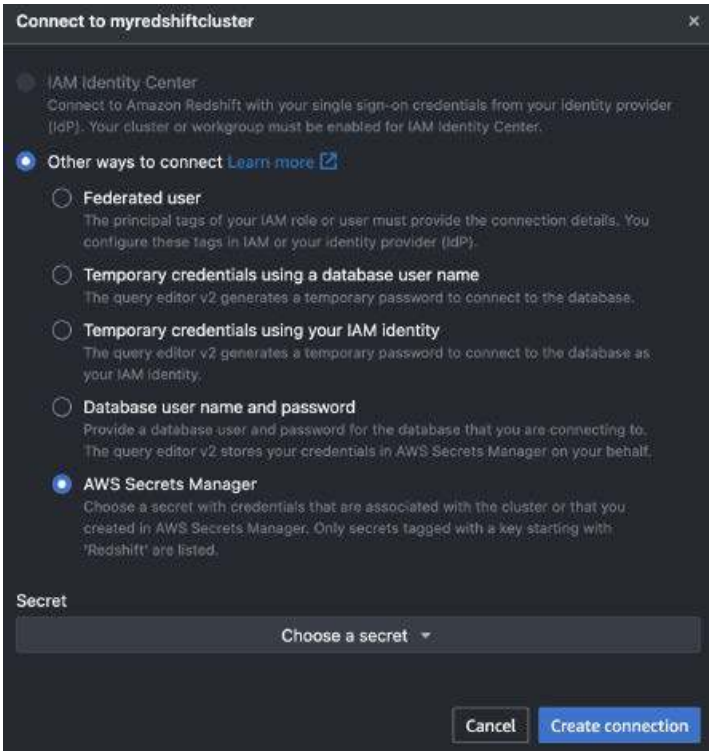


Figure 1.7 – Connection options in query editor v2

5. In the Secret section, click on **Choose a secret**, select the correct secret, and click on **Create connection**.
6. Now that you have successfully connected to the Redshift database, type the following query in the query editor:

```
SELECT current_user;
```

7. Then you can click on **Run** to execute the query.

The results of the query will appear in the Query Results section. You are now connected to the Amazon Redshift data warehouse and ready to execute more queries.

Connecting to Amazon Redshift using SQL Workbench/J client

There are multiple ways to connect to the Amazon Redshift data warehouse, but one of the most popular options is to connect using a UI based tool. SQL Workbench/J is a free cross-platform SQL query tool, which can be used to connect using your own local client.

Getting ready

To complete this recipe, you will need:

- An Amazon Redshift data warehouse (serverless or provisioned cluster) and its login credentials (username and password).
- Install SQL Workbench/J (`https://www.sql-workbench.eu/manual/install.html`).
- Download Amazon Redshift Driver. Visit `https://docs.aws.amazon.com/redshift/latest/mgmt/configuring-connections.html` to download the latest driver version.
- Modify the security group attached to the Amazon Redshift cluster to allow connection from a local client.
- For provisioned clusters, navigate to **Amazon Redshift | Provisioned clusters dashboard | myredshiftcluster | General information** to find the JDBC/ODBC URL to connect to an Amazon Redshift provisioned cluster.
- For Serverless, navigate to **Amazon Redshift | Redshift Serverless | myredshiftwg | General information** to find the JDBC/ODBC URL to connect to Amazon Redshift serverless clusters.

How to do it...

The following steps will enable you to connect using the SQL Workbench/J client tool from your computer:

1. Open SQL Workbench/J by double-clicking on the `SQLWorkbench.exe` (on Windows) or `SQLWorkbenchJ` application (on Mac).
2. In the SQL **Workbench**/J menu, select **File** and then select **Connect window**.
3. Select **Create a new connection profile**.
4. In the **New profile** box, choose any profile name, such as `examplecluster_jdbc`.

5. Select **Manage Drivers.** The **Manage Drivers** dialog will open; select **Amazon Redshift**:

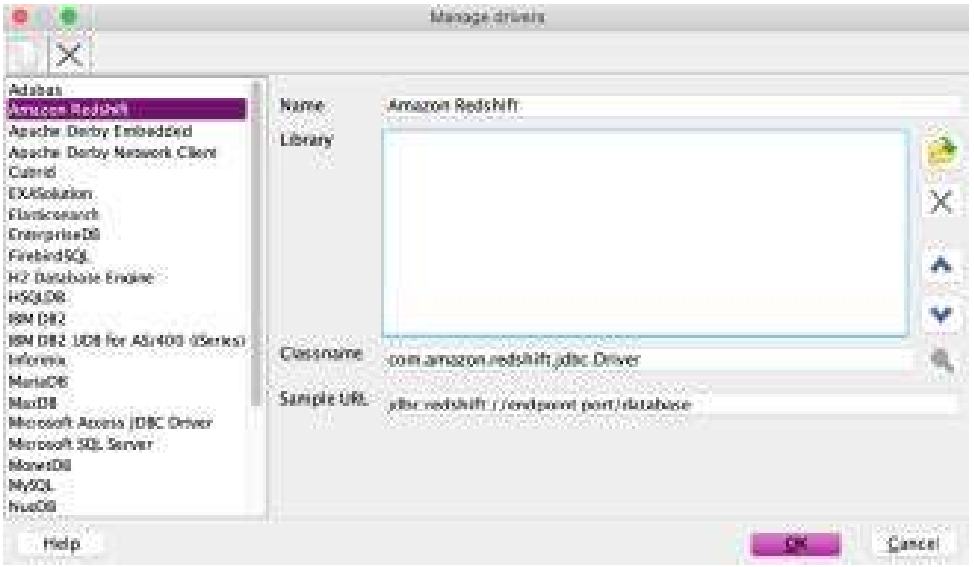


Figure 1.8 – SQL Workbench/J Manage drivers box

6. Select the folder icon adjacent to the Library box, browse and point it to the Amazon Redshift driver location, and then select **Choose**:

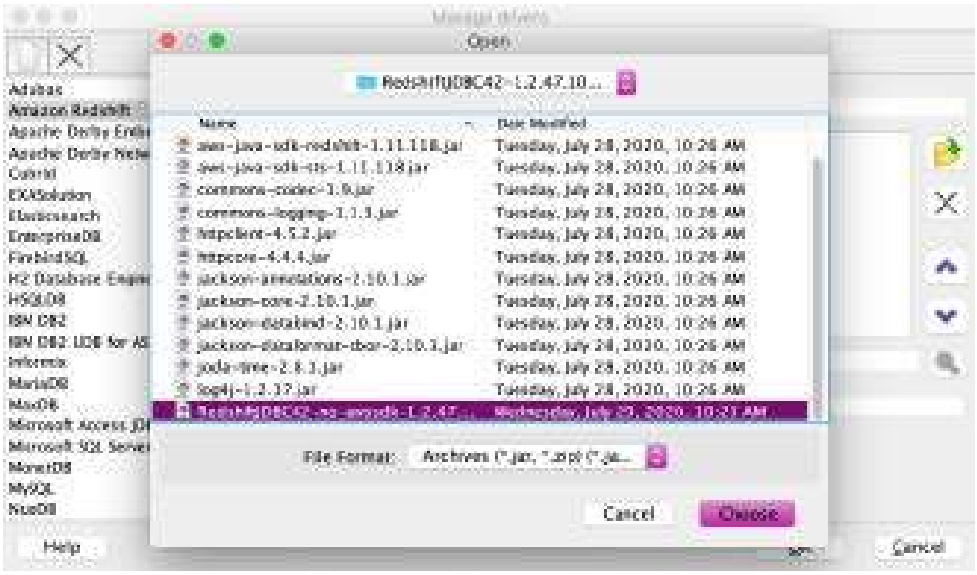


Figure 1.9 – SQL Workbench/J to select Amazon Redshift driver

7. To set up the profile for the Amazon Redshift connection, enter the following details:
    1. In the **Driver** box, select the Amazon Redshift drive.
    2. In **URL**, paste the Amazon Redshift cluster JDBC URL obtained previously.
    3. In **Username**, enter the username (or the master user name) associated with the cluster.
    4. In **Password**, provide the password associated with the username.
    5. Checkmark the **Autocommit** box.
    6. Select the Save profile list icon, as shown in the following screenshot, and select **OK**:

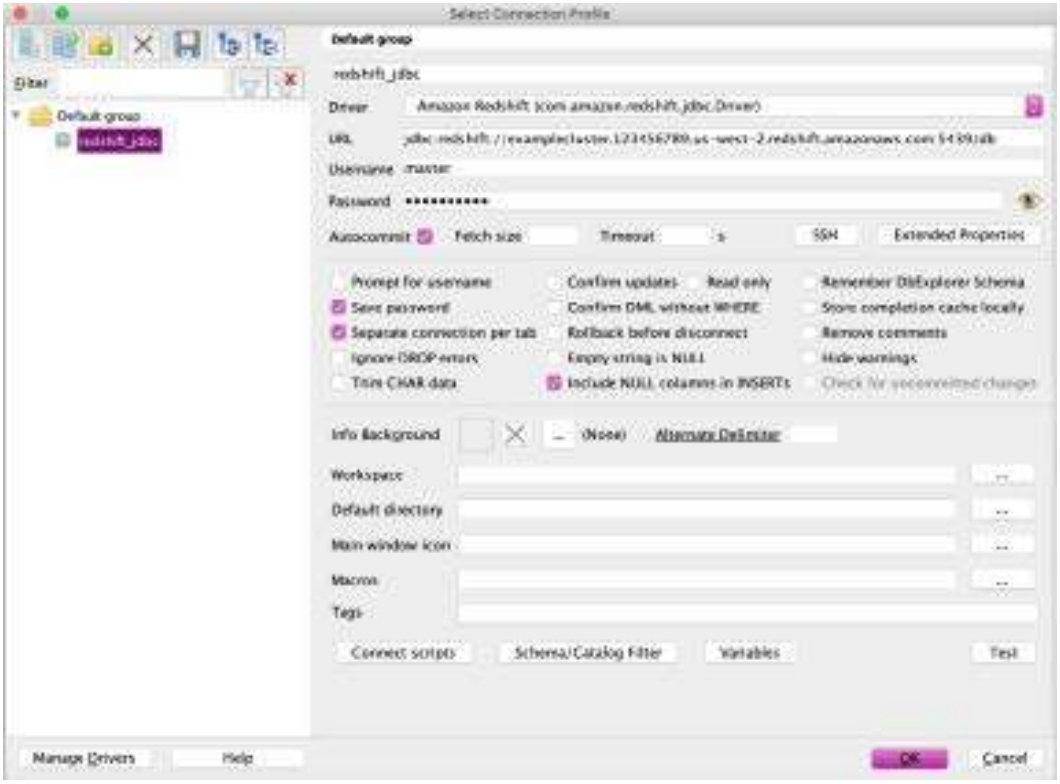


Figure 1.10 – Amazon Redshift Connection Profile

8. After setting up the JDBC connection, you can use the query to ensure you are connected to the Amazon Redshift cluster:

```
select * from information_schema.tables;
```

A list of records will appear in the **Results** tab if the connection was successful:

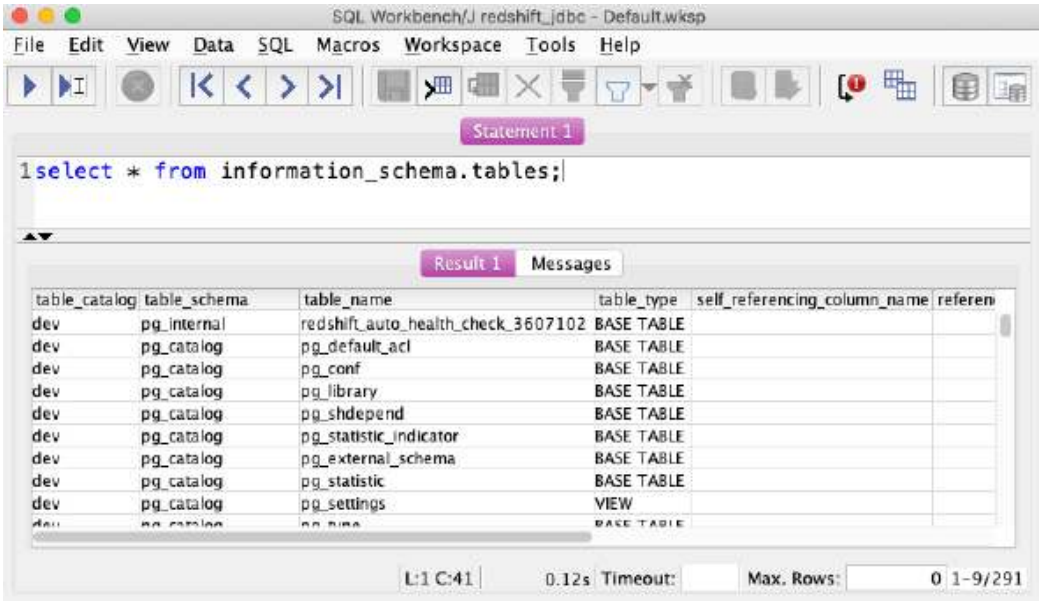


Figure 1.11 – Sample query output from SQL Workbench/J

Connecting to Amazon Redshift using Jupyter Notebook

The Jupyter Notebook is an interactive web application that enables you to analyze your data interactively. Jupyter Notebook is widely used by users such as business analysts and data scientists to perform data wrangling and exploration. Using Jupyter Notebook, you can access all the historical data available in an Amazon Redshift data warehouse (serverless or provisioned cluster) and combine that with data in many other sources, such as an Amazon S3 data lake. For example, you might want to build a forecasting model based on historical sales data in Amazon Redshift combined with clickstream data available in the data lake. Jupyter Notebook is the tool of choice due to the versatility it provides with exploration tasks and the strong support from the open source community. This recipe covers the steps to connect to an Amazon Redshift data warehouse using Jupyter Notebook.

Getting ready

To complete this recipe, you will need:

- An IAM user with access to Amazon Redshift, Amazon EC2, and Amazon Secrets Manager.
- An Amazon Redshift data warehouse (serverless or provisioned cluster) in a VPC. For more information, visit `https://docs.aws.amazon.com/redshift/latest/mgmt/getting-started-cluster-in-vpc.html`.
- A notebook instance (such as Amazon SageMaker) running the Jupyter Notebook in the same VPC as Amazon Redshift (`https://docs.aws.amazon.com/sagemaker/latest/dg/howitworks-create-ws.html`).
- Modify the security group attached to the Amazon Redshift cluster to allow connection from the Amazon SageMaker notebook instance.
- Store the database credentials in Amazon Secrets Manager using *Recipe 2* in *Appendix*.

How to do it...

The following steps will help you connect to an Amazon Redshift cluster using an Amazon SageMaker notebook:

1. Open the AWS Console and navigate to the Amazon SageMaker service.
2. Navigate to your notebook instance and open **JupyterLab**. When using the Amazon SageMaker notebook, find the notebook instance that was launched and click on the **Open JupyterLab** link, as shown in the following screenshot:

Figure 1.12 – Navigating to JupyterLab using the AWS Console

3. Now, let's install the Python driver libraries to connect to Amazon Redshift using the following code in the Jupyter Notebook. Set the kernel as `conda_python3`:

```
!pip install psycopg2-binary
### boto3 is optional, but recommended to leverage the AWS Secrets
Manager storing the credentials  Establishing a Redshift Connection
!pip install boto3
```

Important Note

You can connect to an Amazon Redshift cluster using Python libraries such as Psycopg (`https://pypi.org/project/psycopg2-binary/`) or pg (`https://www.postgresql.org/docs/7.3/pygresql.html`) to connect to the Notebook. Alternatively, you can also use a JDBC, but for ease of scripting with Python, the following recipes will use either of the preceding libraries.

4. Grant the Amazon SageMaker instance permission to use the stored secret. On the AWS Secrets Manager console, click on your secret and find the Secret ARN. Replace the ARN information in the resource section with the following JSON code:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "secretsmanager:GetResourcePolicy",
        "secretsmanager:GetSecretValue",
        "secretsmanager:DescribeSecret",
        "secretsmanager:ListSecretVersionIds"
      ],
      "Resource": [
        "arn:aws:secretsmanager:eu-west-
1:123456789012:secret:aes128-1a2b3c"
      ]
    }
  ]
}
```

5. Now, attach this policy as an inline policy to the execution role for your SageMaker notebook instance. To do this, follow these steps:

    a. Navigate to the Amazon SageMaker (`https://us-west-2.console.aws.amazon.com/sagemaker/`) console.

    b. Select Notebook Instances.

    c. Click on your notebook instance (the one running this notebook, most likely).

    d. Under **Permissions and Encryption**, click on the IAM role link.

    e. You should now be on an IAM console that allows you to **Add inline policy**. Click on the link.

    f. On the **Create Policy** page that opens, click **JSON** and replace the JSON lines that appear with the preceding code block.

    g. Click **Review Policy**.

    h. On the next page select a human-friendly name for the policy and click **Create policy**.

6. Finally, paste the ARN for your secret in the following code block of the Jupyter Notebook to connect to the Amazon Redshift cluster:

```
# Put the ARN of your AWS Secrets Manager secret for your redshift
cluster here:
secret_arn="arn:aws:secretsmanager:eu-west-
1:123456789012:secret:aes128-1a2b3c"
# This will get the secret from AWS Secrets Manager.
import boto3
import json
session = boto3.session.Session()
client = session.client(
    service_name='secretsmanager'
)
get_secret_value_response = client.get_secret_value(
    SecretId=secret_arn
)

if 'SecretString' in get_secret_value_response:
    connection_info = json.loads(get_secret_value_
response['SecretString'])
```

```
else:
    print("ERROR: no secret data found")
# Sanity check for credentials
expected_keys = set(['user', 'password', 'host', 'database',
'port'])
if not expected_keys.issubset(connection_info.keys()):
    print("Expected values for ",expected_keys)
    print("Received values for ",set(connection_info.keys()))
    print("Please adjust query or assignment as required!")

# jdbc:redshift://HOST:PORT/DBNAME
import time
import psycopg2
database = "dev"
con=psycopg2.connect(
    dbname   = database,
    host     = connection_info["host"],
    port     = connection_info["port"],
    user     = connection_info["username"],
    password = connection_info["password"]
)
```

7. Run basic queries against the database. These queries make use of the `cursor` class to execute a basic query in Amazon Redshift:

```
cur = con.cursor()
cur.execute("SELECT sysdate")
res = cur.fetchall()
print(res)
cur.close()
```

8. Optionally, you can use the code here to connect to Amazon Redshift using Amazon SageMaker notebook: `https://github.com/PacktPublishing/Amazon-Redshift-Cookbook-2E/blob/main/Chapter01/Connecting_to_AmazonRedshift_using_JupyterNotebook.ipynb`.

Connecting to Amazon Redshift programmatically using Python and the Redshift Data API

Python is widely used for data analytics due to its simplicity and ease of use. We will use Python to connect using the Amazon Redshift Data API.

The Data API allows you to access Amazon Redshift without using the JDBC or ODBC drivers. You can execute SQL commands on an Amazon Redshift data warehouse (serverless or provisioned cluster), invoking a secure API endpoint provided by the Data API. The Data API ensures the SQL queries to be submitted asynchronously. You can now monitor the status of the query and retrieve your results at a later time. The Data API is supported by the major programming languages, such as Python, Go, Java, Node.js, PHP, Ruby, and C++, along with the AWS SDK.

Getting ready

To complete this recipe, you will need:

- An IAM user with access to Amazon Redshift, Amazon Secrets Manager, and Amazon EC2.
- Store the database credentials in Amazon Secrets Manager using *Recipe 2* in *Appendix*.
- Linux machine terminal such as Amazon EC2, deployed in the same VPC as the Amazon Redshift cluster.
- Python 3.6 or higher version installed on the Linux instance where you can write and execute the code. If you have not installed Python, you can download it from `https://www.python.org/downloads/`.
- Install AWS SDK for Python (Boto3) on the Linux instance. You can see the getting started guide at `https://aws.amazon.com/sdk-for-python/`.
- Modify the security group attached to the Amazon Redshift cluster to allow connections from the Amazon EC2 Linux instance, which will allow it to execute the Python code.
- Create a VPC endpoint for Amazon Secrets Manager and allow the security group to allow the Linux instance to access the Secrets Manager VPC endpoint.

How to do it...

Follow these steps to use a Linux terminal to connect to Amazon Redshift using Python:

1. Open the Linux terminal and install the latest AWS SDK for Python (Boto3) using the following command:

```
pip install boto3
```

2. Next, we will write the Python code. Type `python` on the Linux terminal and start typing the following code. We will first import the `boto3` package and establish a session:

```
import boto3
import json

redshift_cluster_id = "myredshiftcluster"
redshift_database = "dev"
aws_region_name = "eu-west-1"
secret_arn="arn:aws:secretsmanager:eu-west-
1:123456789012:secret:aes128-1a2b3c"
def get_client(service, aws_region_name):
    import botocore.session as bc
    session = bc.get_session()
    s = boto3.Session(botocore_session=session, region_name=region)
    return s.client(service)
```

3. You can now create a client object from the `boto3.Session` object using `RedshiftData`:

```
rsd = get_client('redshift-data')
```

4. We will execute a SQL statement to get the current date by using the secrets ARN to retrieve credentials. You can execute DDL or DML statements. The query execution is asynchronous in nature. When the statement is executed, it returns `ExecuteStatementOutput`, which includes the statement ID:

```
resp = rsd.execute_statement(
    SecretArn= secret_arn
    ClusterIdentifier=redshift_cluster_id,
    Database= redshift_database,
    Sql="SELECT sysdate;"
)
```

```
queryId = resp['Id']
print(f"asynchronous query execution: query id {queryId}")
```

5. Check the status of the query using `describe_statement` and the number of records retrieved:

```
stmt = rsd.describe_statement(Id=queryId)
desc = None
while True:
        desc = rsd.describe_statement(Id=queryId)
        if desc["Status"] == "FINISHED":
            break
            print(desc["ResultRows"])
```

6. You can now retrieve the results of the above query using `get_statement_result`. `get_statement_result` returns a JSON-based metadata and result that can be verified using the below statement:

```
if desc and desc["ResultRows"]  > 0:
   result = rsd.get_statement_result(Id=queryId)
   print("results JSON" + "\n")
   print(json.dumps(result, indent = 3))
```

Note

The query results are available for retrieval only for 24 hours.

The complete script for the above Python code is also available at `https://github.com/PacktPublishing/Amazon-Redshift-Cookbook-2E/blob/main/Chapter01/Python_Connect_to_AmazonRedshift.py`. It can be executed as `python Python_Connect_to_AmazonRedshift.py`.

Connecting to Amazon Redshift using Command Line (psql)

PSQL is a command line front-end to PostgreSQL. It enables you to query the data in an Amazon Redshift data warehouse (serverless or provisioned cluster) interactively. In this recipe, we will see how to install psql and run interactive queries.

Getting ready

To complete this recipe, you will need:

- Install psql (comes with PostgreSQL). To learn more about using psql, you can refer to `https://www.postgresql.org/docs/8.4/static/app-psql.html`. Based on your operating system, you can download the corresponding PostgreSQL binary from `https://www.postgresql.org/download/`.
- If you are using Windows, set the `PGCLIENTENCODING` environment variable to `UTF-8` using the following command using the Windows command-line interface:

```
set PGCLIENTENCODING=UTF8
```

- Capture the Amazon Redshift login credentials.
- Modify the security group attached to the Amazon Redshift cluster to allow connection from the server or client running the psql application, which will allow access to execute the psql code.

How to do it...

The following steps will let you connect to Amazon Redshift through a command-line interface:

1. Open the command-line interface and type psql to make sure it is installed.
2. Provide the connection credentials as shown in the following command line to connect to Amazon Redshift:

```
C:\Program Files\PostgreSQL\10\bin> .\psql -h cookbookcluster-
2ee55abd.cvqfeilxsadl.eu-west-1.redshift.amazonaws.com -d dev -p
5439 -U dbuser
Password for user dbuser:
Type "help" for help.

dev=# help
You are using psql, the command-line interface to PostgreSQL.
Type:  \copyright for distribution terms
       \h for help with SQL commands
       \? for help with psql commands
       \g or terminate with semicolon to execute query
       \q to quit
```

To connect to Amazon Redshift using the psql command line, you will need the clusters endpoint, the database username, and the port. You can use the following command to connect to the Redshift data warehouse:

```
psql -h <clusterendpoint> -U <dbuser> -d <databasename> -p <port>
```

3. To check the database connection, you can use a sample query as specified in the following command:

```
dev=# select sysdate;
```

You are now successfully connected to the Amazon Redshift data warehouse and ready to run the SQL queries!

2

Data Management

Amazon Redshift is a data warehousing service optimized for **online analytical processing** (**OLAP**) applications. You can start with just a few hundred **gigabytes** (**GB**) of data and scale to a **petabyte** (**PB**) or more while maintaining high performance and cost-effectiveness. Its columnar storage architecture is specifically designed for large-scale analytics workloads, storing data by column rather than row. This approach is particularly efficient for OLAP operations as it allows Redshift to read only the specific columns needed for analysis, significantly improving query performance for operations like aggregations, trend analysis, and historical data comparisons. The columnar design also enables better data compression since similar data types are stored together, reducing storage costs and enhancing input/output (I/O) performance. By designing your database to take advantage of this columnar architecture, you can maximize the efficiency of your analytical processing workloads and achieve optimal performance for your data warehouse operations.

Amazon Redshift is best suited for dimensional data modeling, particularly star schemas, snowflake schemas, denormalized tables, etc., which are optimized for analytical querying and reporting. An analytical schema forms the foundation of your data model. This chapter explores how you can set up this schema, thus enabling optimal querying using standard **Structured Query Language** (**SQL**) and easy administration of access controls.

The following recipes are discussed in this chapter:

- Managing a database in Amazon Redshift
- Managing a schema in a database
- Managing tables in a database
- Managing views in a database

- Managing materialized views in a database
- Managing stored procedures in a database
- Managing UDFs in a database

Technical requirements

In order to complete all the recipes in this chapter, you will need an Amazon Redshift data warehouse (either serverless or provisioned cluster) deployed, and a SQL client of your choice to access Amazon Redshift (for example, Amazon Redshift Query Editor V2, DBeaver, SQL Workbench/J, DataGrip, etc.).

Managing a database in Amazon Redshift

Amazon Redshift consists of at least one database, and it is the highest level in the namespace hierarchy for the database objects. This recipe will guide you through the steps needed to create and manage a database in an Amazon Redshift data warehouse.

Getting ready

To complete this recipe, you will need everything mentioned in the *Technical requirements* section at the start of the chapter.

How to do it...

Let's now set up and configure a database on Amazon Redshift. Use the SQL client to connect to it and execute the following commands:

1. We will create a new database called `qa` in the Amazon Redshift data warehouse that has the owner `awsuser` and accepts a maximum of 50 connections. To do this, use the following code:

   ```
   CREATE DATABASE qa
   WITH
   OWNER awsuser
   CONNECTION LIMIT 50;
   ```

   `OWNER` and `CONNECTION LIMIT` are optional parameters. You can find the complete list of options available for database creation at this link: `https://docs.aws.amazon.com/redshift/latest/dg/r_CREATE_DATABASE.html`.

2. To view the details of the qa database, you can query `PG_DATABASE_INFO`, as shown in the following code snippet:

```
SELECT datname, datdba, datconnlimit
FROM pg_database_info
WHERE datname= 'qa';
```

This is the expected output. It has the database name, the user ID for the database admin, and the connection limit for the database:

```
datname datdba  datconnlimit
qa    100    50
```

This query will list the databases that exist in Redshift. If a database is successfully created, it will show up in the query result.

3. To make changes to the database—such as the database name, owner, or connection limit—use the following command, replacing `<qauser>` with the respective Amazon Redshift username:

```
/* Change database name */
ALTER DATABASE qa RENAME TO prod;

/* Change database owner */
ALTER DATABASE qa owner to <qauser>;

/* Change database connection limit */
ALTER DATABASE qa CONNECTION LIMIT 100;
```

4. To verify that the changes have been successfully completed, you can query `PG_DATABASE_INFO`, as shown in the following code snippet:

```
SELECT datname, datdba, datconnlimit
FROM pg_database_info
WHERE datname =  'qa';
```

This is the expected output:

```
datname datdba datconnlimit
prod    100    100
```

5. You can connect to the `prod` database using the connection endpoint, as follows:

```
<RedshiftHostname>:<Port>/prod
```

   Here, `prod` refers to the database you would like to connect to.

6. To drop the previously created database, execute the following query:

```
DROP DATABASE prod;
```

Managing a schema in a database

In Amazon Redshift, a schema is a namespace that groups database objects such as **tables**, **views**, **stored procedures**, and so on. Organizing database objects in a schema is good for security monitoring, and it also means objects are logically grouped. In this recipe, we will create a sample schema that will be used to hold all the database objects.

Getting ready

To complete this recipe, you will need the following:

- Access to any SQL interface, such as Query Editor V2 or a local SQL client
- An Amazon Redshift data warehouse (serverless or provisioned cluster) endpoint
- Log in as a superuser into an Amazon Redshift cluster

How to do it...

1. Users can create a schema using the `CREATE SCHEMA` (`https://docs.aws.amazon.com/redshift/latest/dg/r_CREATE_SCHEMA.html`) command. The following steps will enable you to set up a schema with the name `finance` and add the necessary access to the groups:
2. User groups are a collection of users. They are typically used to group users who would need the same permissions on database objects. It is common to see organization roles such as developers, analysts, and data scientists as groups. Another common grouping pattern is one group per organization, such as finance, audit, and so on.

   Create `finance_grp`, `audit_grp`, and `finance_admin_user` groups using the following SQL statements:

```
create group finance_grp;
create group audit_grp;
create user finance_admin_usr with password
'<PasswordOfYourChoice>';
```

3. Create a schema named `finance` with a space quota of `2 TB`, with a `finance_admin_usr` schema owner:

```
CREATE schema finance authorization finance_admin_usr QUOTA 2 TB;
```

You can also modify or drop existing schemas using the `ALTER SCHEMA` (`https://docs.aws.amazon.com/redshift/latest/dg/r_ALTER_SCHEMA.html`) and `DROP SCHEMA` (`https://docs.aws.amazon.com/redshift/latest/dg/r_DROP_SCHEMA.html`) statements, respectively.

4. For the `finance` schema, grant access privileges of `USAGE` and `ALL` to the `finance_grp` group. Further, grant read access to the tables in the schema using a `SELECT` privilege for the `audit_grp` group:

```
GRANT USAGE on SCHEMA finance TO GROUP finance_grp;
GRANT USAGE on SCHEMA finance TO GROUP audit_grp;
GRANT ALL ON schema finance to GROUP finance_grp;
GRANT SELECT ON ALL TABLES IN SCHEMA finance TO GROUP audit_grp;
```

`USAGE` grants permission on a specific schema, which makes objects in that schema accessible to users. Specific actions on these objects must be granted separately (for example, `SELECT` or `UPDATE` permission on tables). `ALL` grants all the possible privileges on the schema (i.e., `CREATE`, `USAGE`, `ALTER`, or `DROP`).

5. You can verify that the schema and owner group have been created by using the following code:

```
select nspname as schema, usename as owner
from pg_namespace, pg_user
where pg_namespace.nspowner = pg_user.usesysid
and pg_namespace.nspname ='finance';
```

6. Create a `foo` table within the schema by prefixing the schema name along with the table, as shown in the following command:

```
CREATE TABLE finance.foo (bar int);
```

7. Now, in order to select the `foo` table from the `finance` schema, you will have to prefix the schema name along with the table, as shown in the following command:

```
select * from finance.foo;
```

The preceding SQL code will not return any rows as we haven't loaded any data into `foo`.

8. Assign a search path to conveniently reference the database objects directly using single-part notation, like `select * from foo;`, instead of requiring the use of two-part notation, like `select * from finance.foo;`. The following command sets the search path to `'$user', finance, public`. When searching for objects not specified with a schema name, Amazon Redshift follows the search path. It looks through the schemas in the order listed in the search path until it finds the object.

   ```
   set search_path to '$user', finance, public;
   ```

   You can configure the search path by using the `set search_path` command at the current session level or the user level.

9. Now, executing the following `SELECT` query without the schema qualifier automatically locates the `foo` table in the `finance` schema:

   ```
   select * from foo;
   ```

   The preceding SQL code will not return any rows.

Now, the new `finance` schema is ready for use and you can keep creating new database objects in this schema.

Important note

A database is automatically created by default with a `PUBLIC` schema. Identical database object names can be used in different schemas of the database. For example, `finance.customer` and `marketing.customer` are valid table definitions that can be created without any conflict, where `finance` and `marketing` are schema names and `customer` is the table name. Schemas serve the key purpose of easy management through this logical grouping—for example, you can grant `SELECT` access to all the objects at a schema level instead of individual tables.

Managing tables in a database

In Amazon Redshift, you can create a collection of tables within a schema with related entities and attributes. Working backward from your business requirements, you can use different modeling techniques to create tables in Amazon Redshift. You can choose a **star** or **snowflake** schema by using **normalized**, **denormalized**, or **data vault** data modeling techniques.

In this recipe, we will create tables in the `finance` schema, insert data into those tables, and cover the key concepts to leverage the **massively parallel processing** (**MPP**) and columnar architecture.

Getting ready

To complete this recipe, you will need everything mentioned in the *Technical requirements* section at the start of the chapter.

How to do it...

Let's explore how to create tables in Amazon Redshift:

1. Let's create a customer table in the `finance` schema with the `customer_number`, `first_name`, `last_name`, and `date_of_birth` attributes:

```
CREATE TABLE finance.customer
(
  customer_number   INTEGER,
  first_name        VARCHAR(50),
  last_name         VARCHAR(50),
  date_of_birth     DATE
);
```

Note

Something key to creating a customer table is to define columns and their corresponding data types. Amazon Redshift supports data types such as numeric, character, date, datetime with time zone, Boolean, geometry, HyperLog, log sketch, super, and so on.

2. We will now insert 10 records into the customer table using a multi-value `insert` statement. Using multi-row insert statements is more efficient than executing separate `insert` statements for each row:

```
insert into finance.customer values
(1, 'foo', 'bar', '1980-01-01'),
(2, 'john', 'smith', '1990-12-01'),
 (3, 'spock', 'spock', '1970-12-01'),
 (4, 'scotty', 'scotty', '1975-02-01'),
 (5, 'seven', 'of nine', '1990-04-01'),
```

```
(6, 'kathryn', 'janeway', '1995-07-01'),
(7, 'tuvok', 'tuvok', '1960-06-10'),
(8, 'john', 'smith', '1965-12-01'),
(9, 'The Doctor', 'The Doctor', '1979-12-01'),
(10, 'B Elana', 'Torres', '2000-08-01');
```

3. You can now review the information about the customer table using the `svv_table_info` system view. Execute the following query:

```
select "schema", table_id, "table", encoded, diststyle, sortkey1,
cluster size, tbl_rows
from svv_Table_info
where "table" = 'customer'
and "schema" = 'finance';
```

This is the expected output:

```
schema table_id    table    encoded    diststyle    sortkey1    size
tbl_rows
finance    167482    customer    Y    AUTO(ALL)    AUTO(SORTKEY)
14    10
```

`table_id` is the object ID and the number of records in the table is 10. The encoded column indicates that the table is compressed. Amazon Redshift stores columns in 1 **megabyte (MB)** immutable blocks. The size of the table is 14 MB. When you created the customer table, notice that you didn't specify any keys. When no keys are specified, the tables will be created with a default distribution style of `AUTO` and a sort key of `AUTO`. `AUTO` indicates that Amazon Redshift will intelligently and automatically determine the right keys for you based on your workload pattern and update them as your workload evolves. Let's dive into the terminology and concepts of `diststyle` and `sortkey`:

- `diststyle` is a table property that dictates how that table's data is distributed across the data slices in your Amazon Redshift data warehouse. A data slice is a logical partition in which data is cached in the Amazon Redshift SSD.
- `KEY`: The value is hashed, and the same value goes to the same location (data slice) on the compute node.
- `ALL`: The full table data goes to the first slice of every compute node.
- `EVEN`: Uses a round-robin method to evenly spread table rows across all slices, ensuring balanced data distribution regardless of column value.

- AUTO: With AUTO distribution, Amazon Redshift assigns an optimal distribution style based on the size of the table data. For example, if the AUTO distribution style is specified, Amazon Redshift initially assigns the ALL distribution style to a small table. When the table grows larger, Amazon Redshift might change the distribution style to KEY, choosing the primary key (or a column of the composite primary key) as the distribution key. If the table grows larger and none of the columns are suitable to be the distribution key, Amazon Redshift changes the distribution style to EVEN. The change in distribution style occurs in the background with minimal impact on user queries.

4. Let's run a query against the customer table to list customers who were born before 1980:

```
select *
from finance.customer
where extract(year from date_of_birth) < 1980;
```

5. You can also create a copy of the permanent table using `create table as(CTAS)`. Let's execute the following query to create another table for a customer born in 1980:

```
create table finance.customer_dob_1980 as
select *
from finance.customer
where extract(year from date_of_birth) = 1980 ;
```

6. You can also create temporary tables—for example, to generate IDs in a data-loading operation. Temporary tables can only be queried during the current session and are automatically dropped when the session ends. They are created in a session-specific schema and are not visible to any other user. You can use a `create temporary/temp table` command to do this. Execute the following three queries in a single session:

```
create temporary table #customer(custid integer IDENTITY(1,1),
customer_number integer IDENTITY(1,1));
insert into #customer (customer_number) values(1);
select * from #customer;
```

This is the expected output:

```
custid  customer_number
1 1
```

7. Reconnect to Amazon Redshift using the SQL client. Reconnecting will create a new session. Now, try to execute the following query against the `#customer` temporary table:

```
select * from #customer;
```

You will get an **ERROR: 42P01: relation "#customer" does not exist** error message as the temporary tables are only visible to the current session.

How it works...

When you create a table in Amazon Redshift, it stores the data on disk, column by column, on 1 MB blocks. Amazon Redshift by default compresses the columns, which reduces the storage footprint and the **input/output** (**I/O**) when you execute a query against the table. Amazon Redshift provides different distribution styles to spread the data across all the compute nodes, to leverage the MPP architecture for your workload. The metadata and table summary information can be queried using the catalog table and summary view.

Amazon Redshift stores metadata about the customer table. You can query the `pg_table_def` catalog table to retrieve this information. You can execute the following query to view the table/column structure.

Important note

When data is inserted into a table, Amazon Redshift automatically builds, in memory, the metadata of the `min` and `max` values of each block. This metadata, known as a zone map, is accessed before a disk scan in order to identify which blocks are relevant to a query. Amazon Redshift does not have indexes; it does, however, have sort keys. Sort key columns govern how data is physically sorted for a table on disk and can be used as a lever to improve query performance. Sort keys will be covered in depth in *Chapter 8, Performance Optimization*.

See also...

Further information about distribution styles can be found at the following link:

`https://docs.aws.amazon.com/redshift/latest/dg/c_choosing_dist_sort.html`.

Managing views in a database

View database objects allow the result of a query to be stored. In Amazon Redshift, views run each time the view is mentioned in a query. The advantage of using a view instead of a table is that it can allow access to only a subset of data on a table, join more than one table into a single virtual table, and act as an aggregated table, and it takes up no space on the database since only the definition is saved, hence making it convenient to abstract complicated queries. In this recipe, we will create views to store queries for the underlying tables.

Getting ready

To complete this recipe, you will need everything mentioned in the *Technical requirements* section at the start of the chapter.

How to do it...

Let's create a view using the `CREATE VIEW` command. We will use the following steps to create a view:

1. Create a `finance.customer_vw` view based on the results of the query on `finance.customer`:

```
CREATE VIEW finance.customer_vw
AS
SELECT customer_number,
       first_name,
       last_name,
       EXTRACT(year FROM date_of_birth) AS year_of_birth
FROM finance.customer;
```

2. You can create late binding views using the `WITH NO SCHEMA BINDING` clause. The clause specifies that the view isn't bound to the underlying database objects, such as tables and user-defined functions. As a result, there is no dependency between the view and the objects it references. Because there is no dependency, you can drop or alter a referenced object without affecting the view.

Amazon Redshift doesn't check for dependencies until the view is queried:

```
CREATE VIEW finance.customer_lbvw
AS
SELECT customer_number,
       first_name,
       last_name,
       EXTRACT(year FROM date_of_birth) AS year_of_birth
FROM finance.customer
with no schema binding;
```

3. To verify that a view has been created, you can use the following command:

```
SELECT table_schema as schema_name,
       table_name as view_name,
       view_definition
FROM information_schema.views
WHERE table_schema not in ('information_schema', 'pg_catalog')
ORDER by schema_name,
         view_name;
```

Note

This script will provide an output of the views created under a particular schema and the SQL script for the view.

4. We can now select directly from the `finance.customer_vw` view, just like with any another database object, like so:

```
SELECT * from finance.customer_vw limit 5;
```

Note

Here, the `finance.customer_vw` view abstracts the `date_of_birth` **personally identifiable information** (**PII**) from the underlying table and provides the user with an abstracted view of only the essential data for that year to determine the age group.

This is the expected output:

```
outputcustomer_number,first_name,last_name,year_of_birth
1    foo    bar    1980
2    john    smith    1990
3    spock    spock    1970
4    scotty    scotty    1975
5    seven    of nine    1990
```

5. To delete the previously created view, you can use the following command:

```
DROP VIEW finance.customer_vw ;
```

Managing materialized views in a database

A materialized view is a database object that persists the results of a query to disk. In Amazon Redshift, materialized views allow frequently used complex queries to be stored as separate database objects, allowing you to access these database objects directly, and enabling faster query responses.

Employing materialized views is a common approach to powering repeatable queries in a **business intelligence** (**BI**) dashboard, and avoids expensive computation each time. Furthermore, materialized views allow an incremental refresh of the results, using the underlying table data. In this recipe, we will create a materialized view to query the tables and also to persist the results for a faster fetch.

Getting ready

To complete this recipe, you will need everything mentioned in the *Technical requirements* section at the start of the chapter.

How to do it...

Let's create a materialized view using the `CREATE MATERIALIZED VIEW` command. We will use the following steps to create a materialized view, in order to store the precomputed results of an analytical query and also see how to refresh it:

1. Create a `finance.customer_agg_mv` materialized view using the results of the query based on `finance.customer`. You can choose to add a distribution key or sort keys to materialized views like how you can on tables. If you don't specify it, the default distribution style is `EVEN`.

Amazon Redshift can refresh the materialized view automatically when data is updated in the underlying data objects. Use the `AUTO REFRESH` clause to control this behavior. The default for `AUTO REFESH` is no:

```
CREATE MATERIALIZED VIEW finance.customer_agg_mv
AUTO REFRESH YES
AS
SELECT
       EXTRACT(year FROM date_of_birth) AS year_of_birth,
       count(1) customer_cnt
FROM finance.customer
group by EXTRACT(year FROM date_of_birth);
```

2. We can now select directly from `finance.customer`, just like with any other database object, like so:

```
select * from finance.customer limit 5;
```

This is the expected output:

```
outputyear_of_birth,customer_cnt
1975    1
1979    1
1995    1
1970    1
1965    1
```

3. You can verify the state of a materialized view by using an `SVV_MV_INFO` system table (`https://docs.aws.amazon.com/redshift/latest/dg/r_SVV_MV_INFO.html`):

```
select * from SVV_MV_INFO where name='customer_agg_mv';
```

This is the expected output:

```
Output:
database_name,schema,name,is_stale,owner_user_
name,state,autorefresh, autorewrite
vdwpoc       finance customer_agg_mv    f    vdwadmin 1    f    t
```

Here, `stale='f'` indicates that the data is current, reflecting the `customer` underlying base table. This column can be used to refresh the materialized view when needed. Another key column in the `SVV_MV_INFO` table is the `state` column, which indicates if an incremental refresh is possible (`state=1`) or not (`state=0`). In the materialized view, we created a `state=1` state, which indicates that a faster incremental refresh is possible.

4. Now, let's load more data into the underlying `finance.customer` values using the following command, and check the `STV_MV_INFO` table:

```
insert into finance.customer values
(11, 'mark', 'bar', '1980-02-01'),
(12, 'pete', 'smith', '1990-2-01'),
 (13, 'woofy', 'spock', '1980-11-01'),
 (14, 'woofy jr', 'scotty', '1975-03-01'),
 (15, 'eleven', 'of nine', '1990-07-01');
```

5. Query the `SVV_MV_INFO` view again to check the status of the materialized view:

```
select name,is_stale,state from SVV_MV_INFO where name='customer_
agg_mv';
```

Output:

```
name,is_stale,state
customer_agg_mv    t    1
```

Note that `stale = 't'` indicates that the underlying data for the materialized view has changed, but it is possible to refresh it incrementally.

6. Refresh the materialized view using the `REFRESH MATERIALIZED VIEW` command and check the status again:

```
REFRESH MATERIALIZED VIEW finance.customer_agg_mv;
```

This is the expected output:

```
select name,is_stale, state from SVV_MV_INFO where name='customer_
agg_mv';
```

Output:

```
name,is_stale,state
customer_agg_mv  f 1
```

As we can see from the preceding code snippet, `customer_agg_mv` is now updated to reflect the underlying table data.

How it works...

A materialized view can be updated with the latest data from the underlying tables by using the `REFRESH MATERIALIZED VIEW` command. The owner of the materialized view can invoke a refresh and also have `SELECT` access on the tables' references in the materialized view. When the materialized view is being refreshed, it executes a separate transaction to update the dataset. Amazon Redshift also supports an autorefresh option to keep the materialized view up to date as soon as possible after base tables change.

Amazon Redshift offers **Automated Materialized Views** (**AutoMV**) feature to enhance query performance by automatically creating and managing materialized views based on workload monitoring and machine learning algorithms. The following are some of the key features of AutoMV:

- **Continuous monitoring** – Amazon Redshift continuously monitors the workload using machine learning techniques to identify opportunities for performance improvements through the creation of materialized views.
- **Automatic creation and deletion** – When the system detects that a materialized view would be beneficial, it automatically creates and maintains it. Conversely, if a previously created AutoMV is no longer providing performance benefits, the system will automatically drop it.
- **No impact on user workload** – The AutoMV feature only operates during periods of low user activity on the cluster. If you initiate a workload while an AutoMV operation is in progress, the AutoMV task will stop to release resources for the user workload. This ensures that your workloads take priority over the AutoMV operations.

Your users need not be aware of AutoMV. They can continue to use the base tables and Amazon Redshift will automatically rewrite those queries to use AutoMV to improve query performance. Developers don't need to revise queries to take advantage of AutoMV.

Managing stored procedures in a database

Stored procedures in Amazon Redshift are user-created objects using the **Procedural Language/PostgreSQL** (**PL/pgSQL**) procedural programming language. Stored procedures support both **data definition language** (**DDL**) and **data manipulation language** (**DML**). Stored procedures can take in input arguments but do not necessarily need to return results. **PL/pgSQL** also supports conditional logic, loops, and case statements. Stored procedures are commonly used to build reusable **extract, transform, load** (**ETL**) data pipelines and serve the **database administrator** (**DBA**) to automate routine administrative activities—for example, periodically dropping unused tables.

The `SECURITY` attribute controls who has privileges to access certain database objects.

Stored procedures can be created with security definer controls to allow the execution of a procedure without giving access to underlying tables—for example, they can drop a table created by another user and serve the DBA to automate administrative activities. This recipe covers managing stored procedures in a database.

The `NONATOMIC` attribute controls the atomicity of the stored procedure. If you don't specify `NONATOMIC`, the default is atomic—which means that all statements in the stored procedure must succeed, or the entire procedure rolls back. In non-atomic mode, each SQL statement within the stored procedure executes and commits independently. If one statement fails, the previous statements remain committed.

Getting ready

To complete this recipe, you will need everything mentioned in the *Technical requirements* section at the start of the chapter.

How to do it...

In this recipe, we will start with creating a scalar Python-based UDF that will be used to parse an **Extensible Markup Language** (**XML**) input:

1. Connect to Amazon Redshift using the SQL client, and copy and paste the following code to create an `sp_cookbook` stored procedure:

```
Create schema cookbook;
create or replace procedure sp_cookbook(indate in date, records_out
INOUT refcursor) as
$$
declare
```

```
  integer_var int;
begin
   RAISE INFO 'running first cookbook storedprocedure on date %',
indate;
   drop table if exists cookbook.cookbook_tbl;
  create table cookbook.cookbook_tbl
  (recipe_name varchar(50),
   recipe_date date
   );
   insert into cookbook.cookbook_tbl values('stored procedure',
indate);
  GET DIAGNOSTICS integer_var := ROW_COUNT;
   RAISE INFO 'rows inserted into cookbook_tbl = %', integer_var;
    OPEN records_out FOR SELECT * FROM cookbook.cookbook_tbl;
END;
$$ LANGUAGE plpgsql;
```

This stored procedure takes two **parameters**: `indate` is the input and `records_out` serves as both an input and output parameter. This stored procedure uses DDL and DML statements. The current user is the owner of the stored procedure and is also the owner of the `cookbook.cookbook_tbl` table.

Note

Some older versions of SQL client tools may error out with unterminated dollar-quoted string at or near "$$". Ensure that you have the latest version of the SQL client—for example, ensure you are using version 124 or higher for the SQL Workbench/J client.

2. Now, let's execute an `sp_cookbook` stored procedure using the `call` statement and retrieve the output from the resulting cursor using the `fetch` statement. Highlight both statements and execute:

```
call sp_cookbook(current_date, 'inputcursor');
fetch all from inputcursor;
```

This is the expected output:

```
Message
running first cookbook storedprocedure on date 2020-12-13
rows inserted into cookbook_tbl = 1
recipe_name    recipe_date
stored procedure      2020-12-13 00:00:00
```

3. To view a definition of the previously created stored procedure, you can run the following statement:

```
SHOW PROCEDURE sp_cookbook(indate in date, records_out INOUT
refcursor);
```

4. We will now create another stored procedure with a security definer privilege:

```
create or replace procedure public.sp_self_service(tblName in
varchar(60)) as
            $$
begin
    RAISE INFO 'running sp_self_service to drop table %',  tblName;
    execute 'drop table if exists cookbook.' || tblName;
    RAISE INFO 'table dropped %',  tblName;
END;
$$ LANGUAGE plpgsql
SECURITY DEFINER;
```

5. Let's create a user and check whether they have permission to drop the `cookbook.cookbook_tbl` table. The `user1` user does not have permission to drop the table:

```
create user user1 with password 'Cookbook1';
grant execute on procedure public.sp_self_service(tblName in
varchar(60)) to user1;

set SESSION authorization  user1;
select current_user;
drop table cookbook.cookbook_tbl;
```

This is the expected output:

```
ERROR: 42501: permission denied for schema cookbook
```

6. When `user1` executes the `sp_self_service` stored procedure, the procedure runs with the security of the owner of the procedure:

```
set SESSION authorization  user1;
select current_user;
call public.sp_self_service('cookbook_tbl');
```

This is the expected output:

```
running sp_self_service to drop table cookbook_tbl
table
```

This allows the user to drop the table without providing the full permissions for the tables in the cookbook schema.

How it works...

Amazon Redshift uses the PL/pgSQL procedural language to author the stored procedures. PL/pgSQL provides programmatic access that can be used to author control structures to the SQL language and allow complex computations. For example, you have a stored procedure that can create users and set up necessary access that meets your organizational needs—hence, rather than invoking several commands, this can now be done in a single step. The `SECURITY` access attribute of a stored procedure defines the privileges to access the underlying database objects used. By default, an `INVOKER` is used that uses the user privileges, and the `SECURITY DEFINER` allows the procedure user to use the privileges of the owner.

See also...

You can find the complete reference to the PL/pgSQL procedural language at `https://www.postgresql.org/docs/8.0/plpgsql.html#PLPGSQL-ADVANTAGES`. Ready-to-use stored useful procedures can be found at `https://github.com/awslabs/amazon-redshift-utils/tree/master/src/StoredProcedures`.

Managing UDFs in a database

Scalar UDF functions in Amazon Redshift are routines that are able to take parameters, perform calculations, and return the results. UDFs are handy when performing complex calculations that can be stored and reused in a SQL statement. Amazon Redshift supports UDFs that can be authored using either Python or SQL. In addition, Amazon Redshift also supports AWS Lambda UDFs, which opens up further possibilities to invoke other AWS services.

For example, let's say the latest customer address information is stored in AWS DynamoDB—you can invoke an AWS Lambda UDF to retrieve this using a SQL statement in Amazon Redshift.

Getting ready

To complete this recipe, you will need the following:

- Everything mentioned in the *Technical requirements* section
- Access to create an **Identity and Access Management (IAM)** role that can invoke AWS Lambda and attach it to Amazon Redshift
- Access to create an AWS Lambda function

How to do it...

In this recipe, we will start with a scalar Python-based UDF that will be used to parse an XML input:

1. Connect to Amazon Redshift using the SQL client, and copy and paste the following code to create a `f_parse_xml` function:

```
CREATE OR REPLACE FUNCTION f_parse_xml
(xml VARCHAR(MAX), input_rank int)
RETURNS varchar(max)
STABLE
AS $$
    import xml.etree.ElementTree as ET
    root = ET.fromstring(xml)
    res = ''
    for country in root.findall('country'):
        rank = country.find('rank').text
        if rank == input_rank:
           res =  name = country.get('name') + ':' + rank
           break
    return res
$$ LANGUAGE plpythonu;
```

Important note

The preceding Python-based UDF takes in the XML data and uses the `xml.etree.ElementTree` library to parse it to locate an element, using the input rank. See `https://docs.python.org/3/library/xml.etree.elementtree.html` for more options that are available with this XML library.

2. Now, let's validate the `f_parse_xml` function using the following statement, by locating the country name that has the rank 2:

```
select
f_parse_xml('<data>      <country name="Liechtenstein">
<rank>2</rank>         <year>2008</year>         <gdppc>141100</
gdppc>         <neighbor name="Austria" direction="E"/>
<neighbor name="Switzerland" direction="W"/> </country></data>',
'2') as col1
```

This is the expected output:

```
col1
Liechtenstein:2
```

3. We will now create another AWS Lambda-based UDF. Navigate to the AWS console and pick the Lambda service, then click on **Create function**, as shown in the following screenshot:

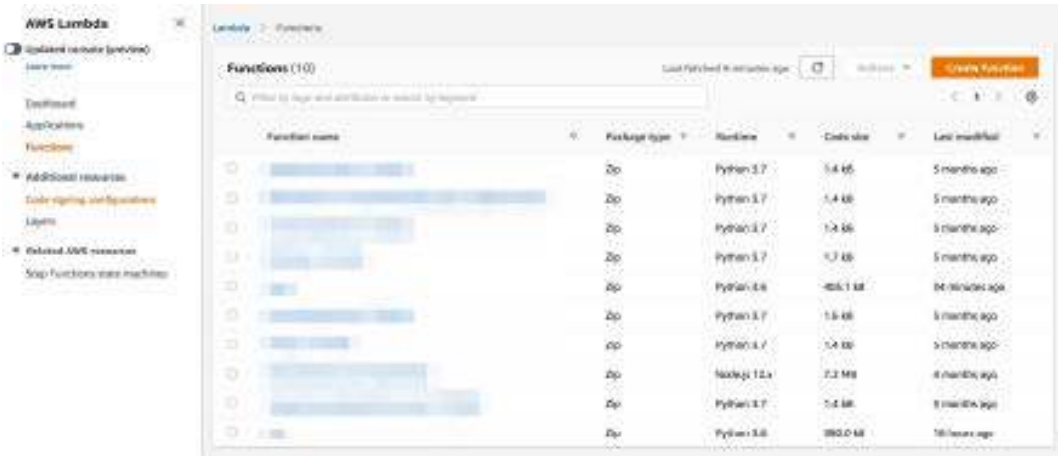


Figure 2.1 – Creating a Lambda function using the AWS console

4. On the **Create function** screen, enter `rs_lambda` under **Function name**, choose a **Python 3.6** runtime, and click on **Create function**.
5. Under **Function code**, copy and paste the following code and press the **Deploy** button:

```
import json
def lambda_handler(event, context):
    ret = dict()
    ret['success'] = True
    ret['results'] = ["bar"]
    ret['error_msg'] = "none"
    ret['num_records'] = 1
    return json.dumps(ret)
```

In the preceding Python-based Lambda function, a sample result is returned. This function can further be integrated to call any other AWS service—for example, you can invoke AWS **Key Management Service** (**KMS**) to encrypt input data.

6. Navigate to **IAM** and create a new role, `RSInvokeLambda`, using the following policy statement by replacing `[Your_AWS_Account_Number]`, `[Your_AWS_Region]` with your AWS account number/Region and attaching the role to Amazon Redshift:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "lambda:InvokeFunction",
            "Resource": "arn:aws:lambda:[Your_AWS_Region]: [Your_
AWS_Account_Number]:function:rs_lambda"
        }
    ]
}
```

7. Connect to Amazon Redshift using the SQL client, and copy and paste the following code to create a `f_redshift_lambda` function that links the AWS Lambda `rs_lambda` function:

```
CREATE OR REPLACE EXTERNAL FUNCTION f_redshift_lambda (bar varchar)
RETURNS varchar STABLE
LAMBDA 'rs_lambda'
```

```
IAM_ROLE 'arn:aws:iam::[Your_AWS_Account_Number]:role/
RSInvokeLambda';
```

You can validate the `f_redshift_lambda` function by using the following SQL statement:

```
select f_redshift_lambda ('input_str') as col1
--output
col1
bar
```

Amazon Redshift is now able to invoke the AWS Lambda function using a SQL statement.

How it works...

Amazon Redshift allows you to create a scalar UDF using either a SQL `SELECT` clause or a Python program in addition to the AWS Lambda UDF illustrated in this recipe. The scalar UDFs are stored with Amazon Redshift and are available to any user when granted the required access.

See also...

You can find a collection of several ready-to-use UDFs that can be used to implement some of the complex reusable logic within a SQL statement at the following link: `https://github.com/aws-samples/amazon-redshift-udfs`.

3

Loading and Unloading Data

Data loading and unloading are crucial processes in managing an Amazon Redshift data warehouse. Loading refers to the ingestion of data from various sources into Redshift tables, while unloading is the process of exporting data from Redshift to external storage or applications. In a typical scenario, such as an ordering-system-based data warehouse, you might need to load the entire previous day's data rather than individual orders. While data can be loaded using standard `INSERT` statements, bulk loading methods are far more efficient given the large volumes of data warehouses typically handle. Similarly, unloading allows you to export data in bulk for use in other applications or analysis tools. This chapter will explore various methods of loading data into Amazon Redshift from different sources, as well as unloading data to external storage such as Amazon S3.

There are multiple ways of loading data into an Amazon Redshift data warehouse. The most common way is using the `COPY` command to load data from Amazon S3. This chapter will cover all the different ways you will be able to load data in Amazon Redshift data warehouse (serverless or provisioned cluster) from different sources.

The following recipes are discussed in this chapter:

- Loading data from Amazon S3 using COPY
- Loading data from Amazon DynamoDB
- Updating and inserting data
- Ingesting data from transactional sources using AWS DMS
- Cataloging and ingesting data using AWS Glue

- Streaming data to Amazon Redshift via Amazon Kinesis Data Firehose
- Unloading data to Amazon S3

Technical requirements

Here are the technical requirements to complete the recipes in this chapter:

- Access to AWS Console.
- The AWS administrator should create an IAM user and an IAM role by following *Recipe 1* and *Recipe 3* in *Appendix*. They will be used in some of the recipes in this chapter.
- The AWS administrator should deploy the AWS CloudFormation template (`https://github.com/PacktPublishing/Amazon-Redshift-Cookbook-2E/blob/main/Chapter03/chapter_3_CFN.yaml`) and create two IAM policies:
    - IAM policy attached to the IAM user that will give them access to Amazon Redshift, Amazon RDS, Amazon DynamoDB, Amazon S3, and Amazon EMR
    - IAM policy attached to the IAM role that will allow the Amazon Redshift data warehouse to access Amazon S3 and Amazon DynamoDB
- Attach an IAM role to the Amazon Redshift data warehouse by following *Recipe 4* in *Appendix*. Make a note of the IAM role name; we will use it in the recipes as `[Your-Redshift_Role]`.
- Amazon Redshift data warehouse deployed in AWS region eu-west-1.
- Amazon Redshift data warehouse admin user credentials.
- Access to any SQL interface, such as a SQL client or Amazon Redshift query editor V2
- Create an Amazon S3 bucket for staging and unloading the data in specific recipes. We will use it in recipes as `[Your-Amazon_S3_Bucket]`.
- AWS account number. We will use it in recipes as `[Your-AWS_Account_Id]`.

Loading data from Amazon S3 using COPY

Amazon Redshift supports a number of data model structures, including **dimensional**, **denormalized**, and **aggregate** (rollup) structures, which makes it optimal for analytics.

In this recipe, we will set up two separate sample datasets in Amazon Redshift that are publicly available:

- A dimensional model using a **star schema benchmark (SSB)** (`https://www.cs.umb.edu/~poneil/StarSchemaB.PDF`), a retail system-based dataset
- A denormalized model using an Amazon.com customer product reviews dataset

To load the datasets, we will use the `COPY` command, which allows data to be copied from Amazon S3 to an Amazon Redshift data warehouse (serverless or provisioned cluster), which is the recommended way to load large data.

Getting ready

To complete this recipe, you will need:

- An Amazon Redshift data warehouse deployed in AWS region eu-west-1
- Amazon Redshift data warehouse admin user credentials
- Access to any SQL interface, such as a SQL client or Amazon Redshift query editor v2
- An IAM role attached to Amazon Redshift data warehouse that can access Amazon S3

How to do it...

We will create and load the following dimensional model, which is based on the **star schema benchmark (SSB)** for an illustrative retail system. A star schema is a dimensional data model where a central fact table contains the main business metrics (facts) and is surrounded by dimension tables (like points of a star), hence the name.

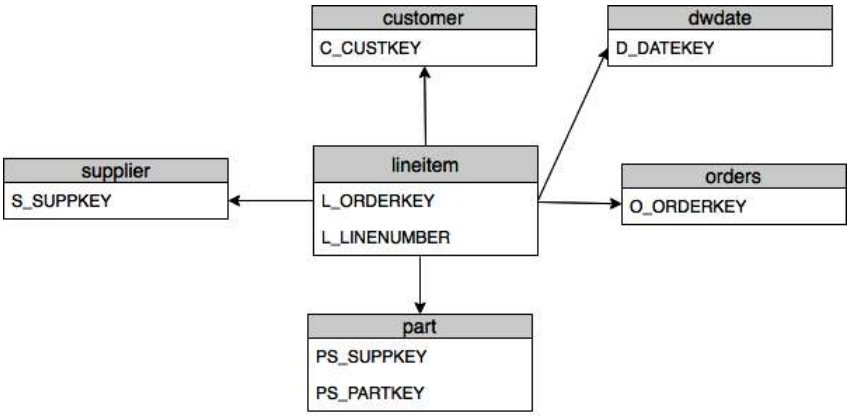


Figure 3.1 –SSB data model

Now, let's create the tables that mimic the previous data model and populate the data into the tables:

1. We will get started by setting up the data in your Amazon S3 bucket. Download `Ssb_Table_Ddl.sql` from the GitHub location `https://github.com/PacktPublishing/Amazon-Redshift-Cookbook-2E/blob/main/Chapter03/Ssb_Table_Ddl.sql`, copy and paste it into any SQL client tool, and execute it to create the dimensional model for the retail system dataset:

```
DROP TABLE IF EXISTS lineitem;
DROP TABLE IF EXISTS supplier;
DROP TABLE IF EXISTS part;
DROP TABLE IF EXISTS orders;
DROP TABLE IF EXISTS customer;
DROP TABLE IF EXISTS dwdate;
CREATE TABLE customer
(
  C_CUSTKEY      BIGINT NOT NULL,
  C_NAME         VARCHAR(25),
  C_ADDRESS      VARCHAR(40),
  C_NATIONKEY    BIGINT,
  C_PHONE        VARCHAR(15),
  C_ACCTBAL      DECIMAL(18,4),
  C_MKTSEGMENT   VARCHAR(10),
  C_COMMENT      VARCHAR(117)
…
CREATE TABLE dwdate
(
  d_datekey            INTEGER NOT NULL,
  d_date               VARCHAR(19) NOT NULL,
  d_dayofweek          VARCHAR(10) NOT NULL,
  d_month              VARCHAR(10) NOT NULL,
  d_year               INTEGER NOT NULL,
…
  d_lastdayinweekfl    VARCHAR(1) NOT NULL,
  d_lastdayinmonthfl   VARCHAR(1) NOT NULL,
  d_holidayfl          VARCHAR(1) NOT NULL,
  d_weekdayfl          VARCHAR(1) NOT NULL
);
```

2. We will now load data from the public S3 bucket to the previous tables. Use any SQL client tool and execute the following command by replacing the `[Your-AWS_Account_Id]` and `[Your-Redshift_Role]` values from the technical requirements in the following script. In each `COPY` command, notice that the file format and compression are specified, such as csv and gzip:

```
COPY customer
from 's3://packt-redshift-cookbook/customer/'
iam_role 'arn:aws:iam::[Your-AWS_Account_Id]:role/[Your-Redshift_
Role]'
CSV gzip;

COPY orders
from 's3://packt-redshift-cookbook/orders/'
iam_role 'arn:aws:iam::[Your-AWS_Account_Id]:role/[Your-Redshift_
Role]'
CSV gzip;

COPY part
from 's3://packt-redshift-cookbook/part/'
iam_role 'arn:aws:iam::[Your-AWS_Account_Id]:role/[Your-Redshift_
Role]'
CSV gzip;

COPY supplier
from 's3://packt-redshift-cookbook/supplier/'
iam_role 'arn:aws:iam::[Your-AWS_Account_Id]:role/[Your-Redshift_
Role]'
CSV gzip;

COPY lineitem
from 's3://packt-redshift-cookbook/lineitem/'
iam_role 'arn:aws:iam::[Your-AWS_Account_Id]:role/[Your-Redshift_
Role]'
CSV gzip;

COPY dwdate
from 's3://packt-redshift-cookbook/dwdate/'
```

```
iam_role 'arn:aws:iam::[Your-AWS_Account_Id]:role/[Your-Redshift_
Role]'
CSV gzip dateformat 'auto';
```

Note

The script will take ~10 minutes to complete. Each table load will output `Load into table *** completed, *** record(s) loaded successfully` to acknowledge a successful execution.

3. Verify that all the tables have been loaded with the correct number of rows using the following command and the output:

```
select count(1) from  lineitem; -- expected rows: 599037902
select count(1) from  supplier; -- expected rows: 1100000
select count(1) from  part; -- expected rows:20000000
select count(1) from  orders; -- expected rows: 76000000
select count(1) from  customer; -- expected rows: 15000000
select count(1) from  dwdate; -- expected rows: 2556
```

4. Now, the dimensional model is ready for querying. We can run an analytical query like the following to join the different tables of the dimensional model to retrieve order metrics for each market segment in 1992:

```
SELECT c_mktsegment,
       COUNT(o_orderkey) AS orders_count,
       SUM(l_quantity) AS quantity,
       SUM(l_extendedprice) AS extendedprice,
       COUNT(DISTINCT P_PARTKEY) AS parts_count,
       COUNT(DISTINCT L_SUPPKEY) AS supplier_count,
       COUNT(DISTINCT o_custkey) AS customer_count
FROM lineitem
  JOIN orders ON l_orderkey = o_orderkey
  JOIN customer c ON o_custkey = c_custkey
  JOIN dwdate
    ON d_date = l_commitdate
   AND d_year = 1992
  JOIN part ON P_PARTKEY = l_PARTKEY
```

```
  JOIN supplier ON L_SUPPKEY = S_SUPPKEY
GROUP BY c_mktsegment;
```

5. In addition to the dimensional model, let's also create a denormalized table using the Amazon product review data. Create the product review data table using:

```
CREATE TABLE product_reviews(
  marketplace varchar(2),
  customer_id varchar(32),
  review_id varchar(24),
  product_id varchar(24),
  product_parent varchar(32),
  product_title varchar(512),
  star_rating int,
  helpful_votes int,
  total_votes int,
  vine char(1),
  verified_purchase char(1),
  review_headline varchar(256),
  review_body varchar(max),
  review_date date,
  year int,
  product_category varchar(32),
  insert_ts datetime default current_timestamp)
DISTSTYLE KEY
DISTKEY (customer_id)
SORTKEY (
    marketplace,
    product_category,
    review_date);
```

6. Now, let's load the review data into the `product_reviews` table by executing the following command in the SQL client:

```
COPY product_reviews
FROM 's3://packt-redshift-cookbook/reviews_parquet/' iam_role
'arn:aws:iam::[Your-AWS_Account_Id]:role/[Your-Redshift_Role]'
PARQUET;
```

Note

The `packt-redshift-cookbook` S3 bucket should be in the eu-west-1 region. Your Amazon Redshift data warehouse must be in the same region in order to load Parquet or ORC files using the `COPY` command.

7. Now, the `product_reviews` table is ready for querying, and you can execute the following query to get the top 10 most-voted products in the `Apparel` product category:

```
SELECT product_title,
       SUM(total_votes)
FROM product_reviews
WHERE product_category = 'Apparel'
GROUP BY product_title
ORDER BY SUM(total_votes) DESC LIMIT 10;
```

Now, we have used Amazon S3 to move data into Amazon Redshift using the `COPY` command and set up a **dimensional** and **denormalized** dataset.

How it works...

The Amazon Redshift `COPY` command is used to load large datasets into Amazon Redshift from Amazon S3. This is the recommended approach as the `COPY` command takes advantage of the **massively parallel processing** (**MPP**) of the Amazon Redshift to ingest the data into the Amazon Redshift table efficiently. The `COPY` command also provides several ways to ingest incoming files. This includes support for multiple file formats (such as CSV, Parquet, and JSON) with error handling and the flexibility to ingest all kinds of structured data.

See also...

For more information on using the `COPY` command from Amazon S3, visit `https://docs.aws.amazon.com/redshift/latest/dg/copy-parameters-data-source-s3.html`.

Take a look at the best practices for the `COPY` command at `https://docs.aws.amazon.com/redshift/latest/dg/c_loading-data-best-practices.html`.

Loading data from Amazon DynamoDB

Amazon DynamoDB is a serverless, NoSQL, fully managed database with single-digit millisecond performance at any scale. DynamoDB is designed to be used as an operational database in OLTP use cases where you know access patterns and can design your data model for those access patterns. When you want to perform analytics, you can complement Amazon DynamoDB with Amazon Redshift's OLAP capabilities.

In this recipe, we will see how data from an Amazon DynamoDB table can be copied to an Amazon Redshift data warehouse (serverless or provisioned cluster) table using the `COPY` command. We will use the full table copy approach in this recipe.

You can also load data in near real-time from Amazon DynamoDB into Amazon Redshift by leveraging zero ETL integration. There is a recipe for this zero ETL integration in *Chapter 4*.

Getting ready

To complete this recipe, you will need:

- Access to the AWS Console.
- An Amazon Redshift data warehouse deployed in AWS region eu-west-1.
- Amazon Redshift data warehouse admin user credentials.
- Access to any SQL interface, such as a SQL client or Amazon Redshift query editor.
- An Amazon DynamoDB table deployed in AWS region eu-west-1. Refer to `https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/GettingStarted.Python.html` to set up the necessary AWS SDK for Python (Boto3) and use `https://github.com/PacktPublishing/Amazon-Redshift-Cookbook-2E/blob/main/Chapter03/CreateAndLoad_dynamodb.py` to set up the sample `part` table.
- An IAM role attached to the Amazon Redshift data warehouse that can access Amazon DynamoDB.
- Access to the AWS CLI to get the record count from the Amazon DynamoDB table

How to do it...

In this recipe, we will load data directly from Amazon DynamoDB to Amazon Redshift:

1. Let's start with making a `cli` call to a DynamoDB table part to verify the total number of item counts. Execute the following on the command line and you will see a count of `20000` in the part table:

```
aws dynamodb scan --table-name part --select "COUNT"
output:
  {
    "Count": 20000,
    "ScannedCount": 20000,
    "ConsumedCapacity": null }
```

2. Log in to Amazon Redshift data warehouse using a SQL client or query editor v2 and create the `part` table:

```
DROP TABLE IF EXISTS part;
CREATE TABLE part
(
  P_PARTKEY       BIGINT NOT NULL,
  P_NAME          VARCHAR(55),
  P_MFGR          VARCHAR(25),
  P_BRAND         VARCHAR(10),
  P_TYPE          VARCHAR(25),
  P_SIZE          INTEGER,
  P_CONTAINER     VARCHAR(10),
  P_RETAILPRICE   DECIMAL(18,4),
  P_COMMENT       VARCHAR(23)
)
diststyle ALL;
```

3. Frame the `COPY` command to load into the `part` table in Amazon Redshift from the `part` table in Amazon DynamoDB:

```
COPY part from 'dynamodb://part'
iam_role 'arn:aws:iam::[Your-AWS_Account_Id]:role/[Your-Redshift_
Role]'
readratio 50;
```

The `readratio` parameter specifies the percentage of the DynamoDB table's provisioned throughput to use for the data load. It is a mandatory parameter for COPY from DynamoDB.

4. Execute the above `COPY` command using the Amazon Redshift query editor.
5. Verify the record count of the data loaded into the part table. `20000` records have been loaded to the part table:

```
Select count(*) from part;

--expected sample output
count(*)
20000
```

6. Let's review the column values for the part table on Amazon Redshift:

```
Select p_partkey,p_name,p_mfgr from part limit 5;

--expected sample output
p_partkey p_name p_mfgr
800213    chartreuse steel indian burlywood    Manufacturer#2
1101041    red lemon khaki frosted blush    Manufacturer#1
2500838    tan cream cyan lemon olive    Manufacturer#2
12669574    bisque salmon honeydew violet steel    Manufacturer#2
12579584    pale linen thistle firebrick orange    Manufacturer#3
```

How it works...

In the `COPY` command that is used to load data from Amazon Dynamo DB, the column names in the Amazon Redshift table should match the attribute names in the `DynamoDB` `part` table. If a column name is not present in `DynamoDB`, it is loaded with empty or NULL based on the `COPY` command's `emptyasnull` option. If the attributes in `DynamodDB` are not present in the Amazon Redshift table, those attributes are discarded. Also notice that you can specify the Amazon DynamoDB readratio (in the preceding readratio of 50); this regulates the percentage of provisioned throughput that is consumed by the `COPY` command for `DynamoDB` table part.

Updating and inserting data

An **Extract Load Transform (ELT)** process is a common technique to refresh the data warehouse from the source system. The ELT process can be executed as a batch near real-time process that allows staging the data from the source system and performing bulk refresh into the Amazon Redshift data warehouse (serverless or provisioned cluster). Amazon Redshift being an RDBMS system allows data refresh in the form of `MERGE` operations, which combine the functionality of `INSERT`, `UPDATE`, and `DELETE` operations. In this recipe, we will delve into some of the common ELT strategies to refresh a dimensional model using the `MERGE` command.

Getting ready

To complete this recipe, you will need:

- Access to the AWS Console
- An Amazon Redshift data warehouse deployed in AWS region eu-west-1
- Amazon Redshift data warehouse admin user credentials
- Access to any SQL interface, such as a SQL client or the Amazon Redshift query editor
- A sample dimensional model setup

How to do it...

This recipe will illustrate refreshing the `part` dimension, followed by the `lineitem` fact table. The dimensional tables will be refreshed first and followed by the fact table to maintain the data's integrity. The complete script for this recipe is also available at `https://github.com/PacktPublishing/Amazon-Redshift-Cookbook-2E/blob/main/Chapter03/part.sql` and `https://github.com/PacktPublishing/Amazon-Redshift-Cookbook-2E/blob/main/Chapter03/Insert_Update_Lineitem.sql`. Let's start with the data refresh for the `part` dimension:

1. Open any SQL client tool and start the transaction for the `part` dimension table to refresh:

```
BEGIN TRANSACTION;
```

Tip

Using the transaction to update the data allows rollback if there is an error, and also, end users do not see the intermediate state of the data change.

2. Create the staging table and load the incoming incremental data from the source:

```
/* Create a staging table to hold the input data. Staging table is
created with BACKUP NO option for faster inserts and also since data
is temporary */
DROP TABLE IF EXISTS stg_part;
CREATE TABLE stg_part
(
  NAME          VARCHAR(55),
  MFGR          VARCHAR(25),
  BRAND         VARCHAR(10),
  TYPE          VARCHAR(25),
  SIZE          INTEGER,
  CONTAINER     VARCHAR(10),
  RETAILPRICE   DECIMAL(18,4),
  COMMENT       VARCHAR(23)
)
BACKUP NO
;
COPY stg_part
FROM 's3://packt-redshift-cookbook/etl/part/dt=2020-08-15/' iam_role
'arn:aws:iam::[Your-AWS_Account_Id]:role/[Your-Redshift_Role]'csv
gzip compupdate preset;
```

Tip

Notice that incremental data for `2020-08-15` is loaded into the `stg_part` table.

3. Data will be merged into the part dimension table using the `MERGE` command which will update existing matching records and insert new records. The update and insert operations are performed based on the natural key of the table:

```
MERGE INTO part AS tgt
USING stg_part AS src
ON tgt.p_name = src.name
WHEN MATCHED THEN
  UPDATE SET
```

```
    p_mfgr = src.mfgr,
    p_brand = src.brand,
    p_type = src.type,
    p_size = src.size,
    p_container = src.container,
    p_retailprice = src.retailprice,
    p_comment = src.comment
WHEN NOT MATCHED THEN
  INSERT (p_partkey, p_name, p_mfgr, p_brand, p_type, p_size, p_
container, p_retailprice, p_comment)
  VALUES (
    (SELECT MAX(p_partkey) + ROW_NUMBER() OVER (ORDER BY src.name)
FROM part),
    src.name,
    src.mfgr,
    src.brand,
    src.type,
    src.size,
    src.container,
    src.retailprice,
    src.comment
  );
```

4. The data refresh is now complete on the target `part` dimension. Commit the transaction using the following command:

```
--  commit and End transaction
END TRANSACTION;
```

Note

Similarly, you can repeat the preceding steps for other dimensional tables as well before starting the fact table.

5. Now, let's refresh the `lineitem` fact table using the following script. Start the transaction for the `lineitem` fact table:

```
-- Start a new transaction
BEGIN TRANSACTION;
```

6. Create the staging table to hold the incoming incremental data, as shown in the following block:

```
-- Drop stg_lineitem if exists
DROP TABLE IF EXISTS stg_lineitem;

-- Create a stg_lineitem staging table and COPY data from input S3
location with the refreshed incremental data
CREATE TABLE stg_lineitem(
  orderkey        BIGINT,
  LINENUMBER      INTEGER NOT NULL,
  QUANTITY        DECIMAL(18,4),
  EXTENDEDPRICE   DECIMAL(18,4),
  DISCOUNT        DECIMAL(18,4),
  TAX             DECIMAL(18,4),
  RETURNFLAG      VARCHAR(1),
  LINESTATUS      VARCHAR(1),
  SHIPDATE        DATE,
  COMMITDATE      DATE,
  RECEIPTDATE     DATE,
  SHIPINSTRUCT    VARCHAR(25),
  SHIPMODE        VARCHAR(10),
  COMMENT         VARCHAR(44),
  p_name          VARCHAR(55),
  s_name          VARCHAR(25)
)
BACKUP NO;

COPY stg_lineitem FROM 's3://packt-redshift-cookbook/etl/lineitem/
shipdate_dt=2020-08-15/' iam_role 'arn:aws:iam::[Your-AWS_Account_
Id]:role/[Your-Redshift_Role]' csv gzip;
```

Tip

Notice that the incremental data for `2020-08-15` is loaded into the `stg_lineitem` table.

7. Delete any existing data (if any) for `2020-08-15` and refresh it with the current data for this date:

```
-- Delete any rows from target store_sales for the input date for
idempotency
DELETE FROM lineitem WHERE l_shipdate = '2020-10-15';
```

8. Merge the new incoming data for `2020-18-15` into the `lineitem` fact table using the following `MERGE` statement:

```
WITH supplier_dim AS
  (SELECT DISTINCT s_name, s_suppkey FROM supplier),
part_dim AS
  (SELECT DISTINCT p_name, p_partkey FROM part)
MERGE INTO lineitem AS tgt
USING stg_lineitem AS src
  ON tgt.l_shipdate = '2020-10-15'
    AND tgt.l_partkey = (SELECT p_partkey FROM part_dim WHERE p_name
= src.p_name)
    AND tgt.l_suppkey = (SELECT s_suppkey FROM supplier_dim WHERE
s_name = src.s_name)
WHEN NOT MATCHED THEN
  INSERT (
    l_orderkey, l_partkey, l_suppkey, l_linenumber, l_quantity,
    l_extendedprice, l_discount, l_tax, l_returnflag, l_linestatus,
    l_shipdate, l_commitdate, l_receiptdate, l_shipinstruct, l_
shipmode, l_comment
  )
  VALUES (
    src.orderkey, (SELECT p_partkey FROM part_dim WHERE p_name =
src.p_name),
    (SELECT s_suppkey FROM supplier_dim WHERE s_name = src.s_name),
    src.linenumber, src.quantity, src.extendedprice, src.discount,
src.tax,
```

```
    src.returnflag, src.linestatus, src.shipdate, src.commitdate,
src.receiptdate,
    src.shipinstruct, src.shipmode, src.comment);
```

Important Note

Note that dimensional keys are derived from the dimensional table using the natural keys.

9. All the data refresh is now complete on the target `lineitem` fact table. Commit the transaction using the following code:

```
--  commit and End transaction
COMMIT;
```

Important Note

Notice that all data in the dimension and fact tables is handled in bulk to update/insert all the incoming data in one go. This is a best practice since the effort to perform **Data Manipulation Language** (**DML**) on a few rows or several rows is the almost the same.

10. Now you have a refreshed dimensional model with the latest data that can be verified by executing the following query:

```
SELECT c_mktsegment,
       COUNT(o_orderkey) AS orders_count,
       SUM(l_quantity) AS quantity,
       SUM(l_extendedprice) AS extendedprice,
       COUNT(DISTINCT P_PARTKEY) AS parts_count,
       COUNT(DISTINCT L_SUPPKEY) AS supplier_count,
       COUNT(DISTINCT o_custkey) AS customer_count
FROM lineitem
  JOIN orders ON l_orderkey = o_orderkey
  JOIN customer c ON o_custkey = c_custkey
  JOIN part ON P_PARTKEY = l_PARTKEY
  JOIN supplier ON L_SUPPKEY = S_SUPPKEY
```

```
WHERE l_shipdate = '2020-10-15'
GROUP BY c_mktsegment;
```

The previous ELT strategy can now be integrated with any workflow tool that will allow automatic refresh for the data warehouse.

Ingesting data from transactional sources using AWS DMS

When you have on-premises or AWS RDS transactional data sources that don't natively support zero-ETL integrations and you want to replicate or migrate that data to your Amazon Redshift data warehouse for consolidation or reporting purposes, you can utilize the AWS **Database Migration Service** (**DMS**). AWS DMS is a fully managed service that helps in performing full load as well as ongoing change data capture from supported transactional data sources to Amazon Redshift.

In this recipe, we will do a full replication of the `parts` table from Amazon RDS MySQL, servicing as a transactional source, to an Amazon Redshift data warehouse (serverless or provisioned cluster).

Getting ready

To complete this recipe, you will need:

- Amazon Redshift data warehouse deployed in the eu-west-1 AWS region
- Amazon Redshift data warehouse admin user credentials
- An IAM user with access to Amazon Redshift, Amazon RDS, and AWS DMS
- Amazon RDS MySQL Cluster deployed in AWS regions eu-west-1 in the same VPC as the Amazon Redshift data warehouse (reference: `https://aws.amazon.com/getting-started/hands-on/create-mysql-db/`)
- Capture the login credentials for the Amazon RDS MySQL cluster
- An AWS Database Migration Replication instance deployed in the eu-west-1 AWS region in the same VPC as Amazon Redshift data warehouse (reference: `https://docs.aws.amazon.com/dms/latest/sbs/CHAP_RDSOracle2Aurora.Steps.CreateReplicationInstance.html`)
- Command line to connect to Amazon RDS MySQL (reference: `https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/USER_ConnectToInstance.html`)
- Open connectivity between your local client, such as Amazon EC2 Linux, to the Amazon RDS MySQL database

- Open connectivity between Amazon RDS MySQL and AWS DMS instance
- Note the VPC ID where Amazon Redshift and Amazon RDS are deployed

How to do it...

This recipe illustrates full replication of the parts table from Amazon RDS MySQL to the Amazon Redshift data warehouse using AWS DMS as the replication engine:

1. Let's connect to the Amazon RDS MySQL database using the command line installed on the AWS EC2 instance. Enter the password, and it will connect you to the database:

```
mysql -h [yourMySQLRDSEndPoint] -u admin -p;
```

2. We will create an `ods` database on MySQL and create a `parts` table in the `ods` database:

```
create database ods;
CREATE TABLE ods.part
(
  P_PARTKEY       BIGINT NOT NULL,
  P_NAME          VARCHAR(55),
  P_MFGR          VARCHAR(25),
  P_BRAND         VARCHAR(10),
  P_TYPE          VARCHAR(25),
  P_SIZE          INTEGER,
  P_CONTAINER     VARCHAR(10),
  P_RETAILPRICE   DECIMAL(18,4),
  P_COMMENT       VARCHAR(23)
);
```

3. On your client server, download the `part.tbl` file from `https://github.com/PacktPublishing/Amazon-Redshift-Cookbook-2E/blob/main/Chapter03/part.sql`.
4. Now, we will load this file into the `ods.part` table in the MySQL database. This will load 20,000 records into the `parts` table:

```
LOAD DATA LOCAL INFILE 'part.tbl'
    INTO TABLE ods.part
    FIELDS TERMINATED BY '|'
    LINES TERMINATED BY '\n';
```

```
Let's verify the record count loaded into ods.part table.
MySQL [(none)]> select count(*) from ods.part;
+----------+
| count(*) |
+----------+
|    20000 |
+----------+
1 row in set (0.00 sec)
```

5. Turn on binary logging in the RDS MySQL database by executing the following command:

```
call mysql.rds_set_configuration('binlog retention hours', 24);
```

   In your MySQL database instance in parameter group, set the `binlog_format` parameter to `ROW`.

6. Now, we will go to the AWS Database Migration Services landing page to create a source and target for the replication instance (`https://console.aws.amazon.com/dms/v2/home?`).

7. First, we will create a **Source endpoint** for RDS MySql:

   1. Navigate to Endpoints and click on **Create Endpoint**.
   2. Select **Source Endpoint** and check **Select RDS DB instance**.
   3. From the dropdown, select your **RDS Instance**.

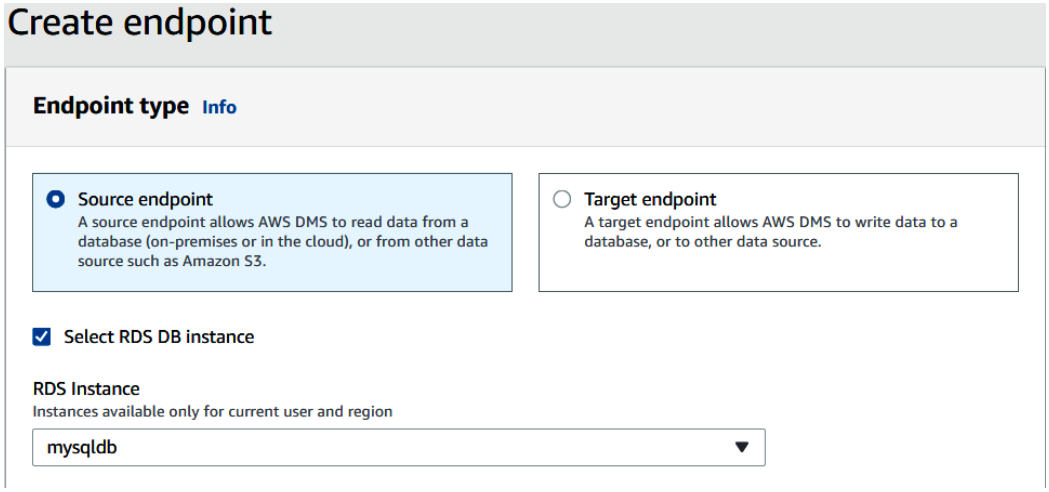


Figure 3.2 – Create an AWS DMS source endpoint for the MySQL database

4. Enter the password for your RDS MySQL database.

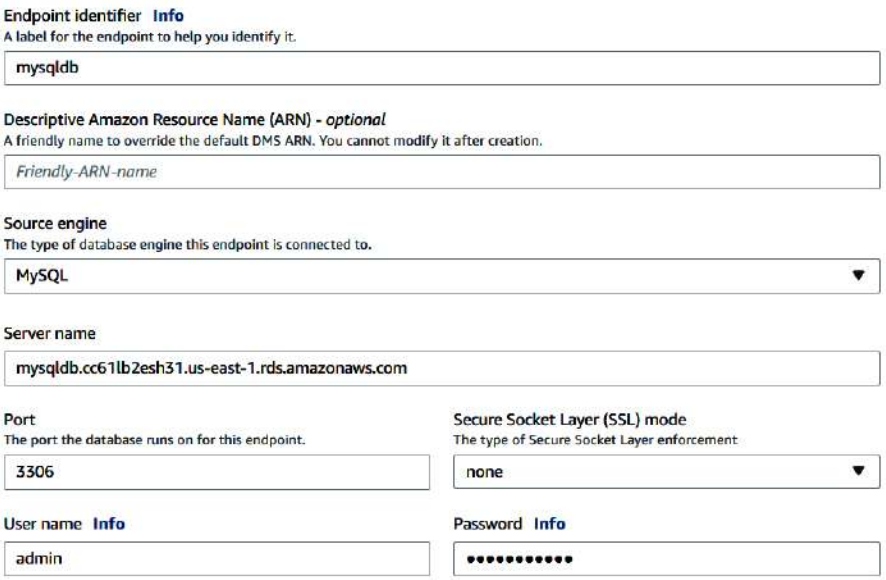


Figure 3.3 – AWS DMS source endpoint for the MySQL database

5. Test your endpoint connection from the AWS DMS replication you created earlier on. Select the **VPC** and replication instance and click **Run test**. You will receive a successful connection message on completion.

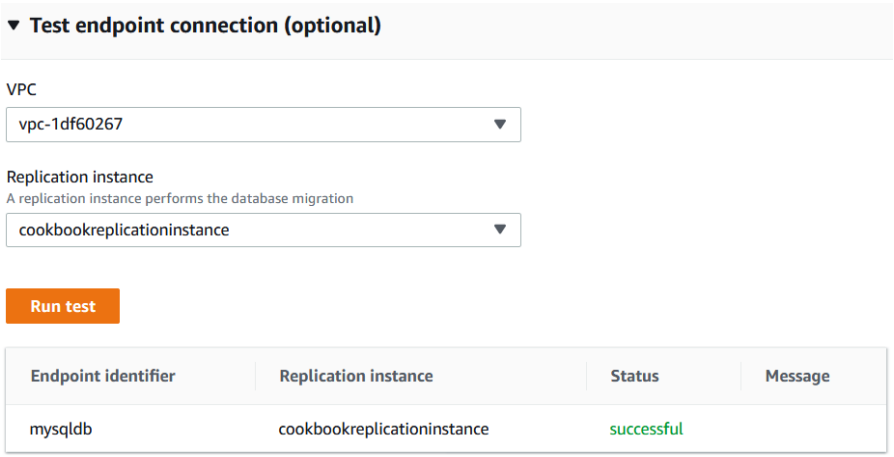


Figure 3.4 – AWS DMS source endpoint for MySQL database test connection

8. Secondly, we will create a target endpoint for the Amazon Redshift data warehouse. Click on Create endpoint and select the target endpoint. Populate the details of your Amazon Redshift data warehouse endpoint, userid, password, and database name. Test the connection using the pre-created database replication instance:

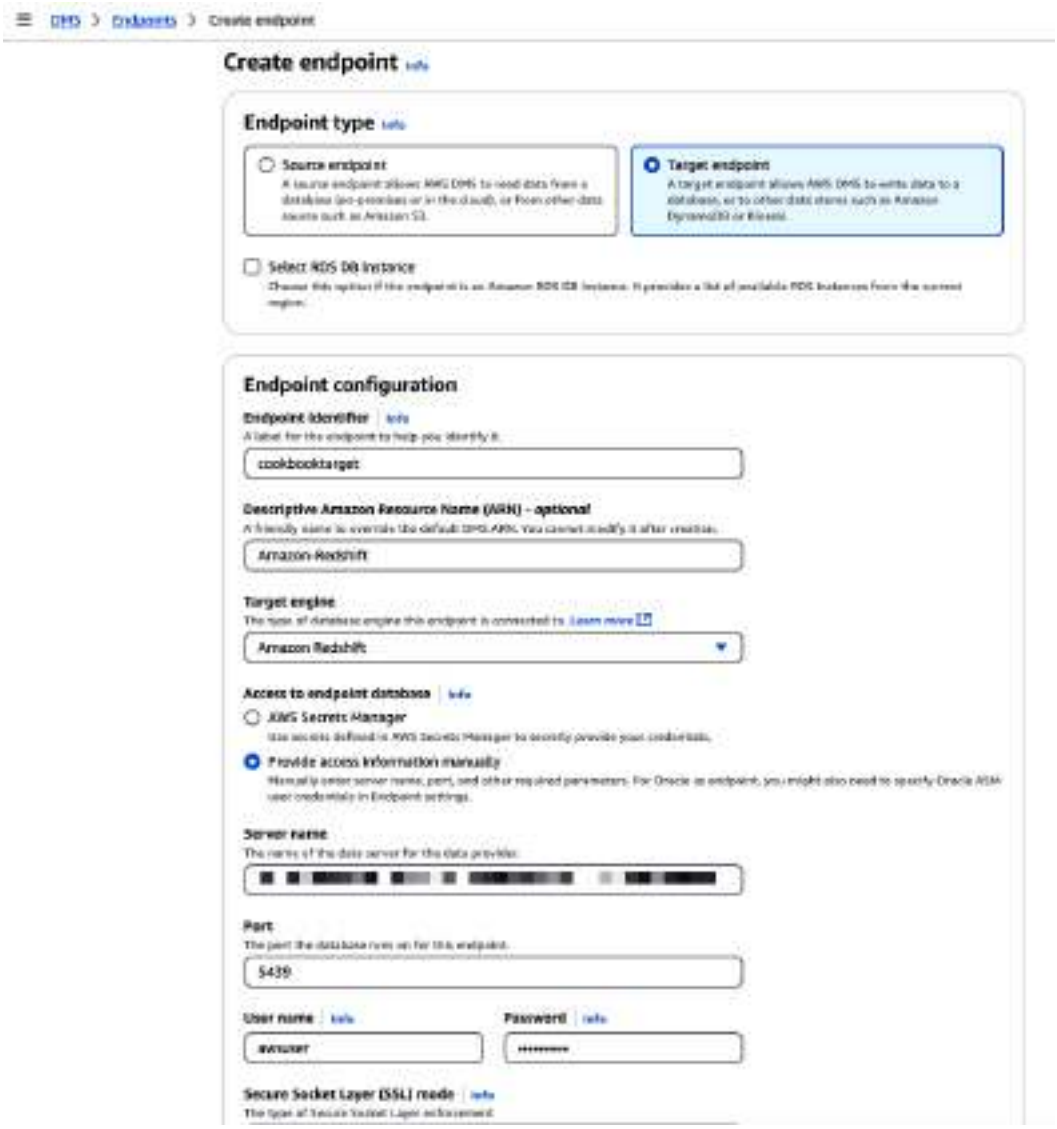


Figure 3.5 – AWS DMS target endpoint for Amazon Redshift

9. Now, we will create the database migration task:

    - Navigate to database migration tasks, and click on **Create task**.
    - Select the **Replication instance**.
    - For **source database endpoint**, select **mysqldb**, and for the target, select the Amazon Redshift endpoint, **cookbooktarget**.
    - For migration type, select **migrate existing and replicate ongoing changes**. This will do a full load followed by ongoing change data capture.

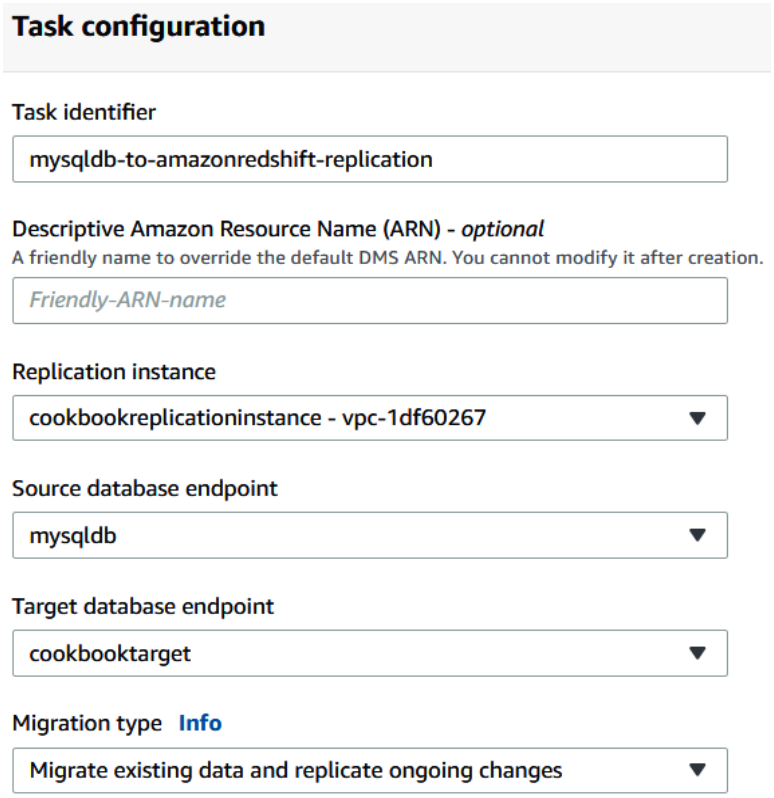


Figure 3.6 – AWS DMS migration task

10. For target table preparation mode, select **Do nothing**. AWS DMS assumes that the target tables have been pre-created in Amazon Redshift.

11. For table mappings, add the following rule. Enter the **ods** schema and set the table name to the **%** wildcard.

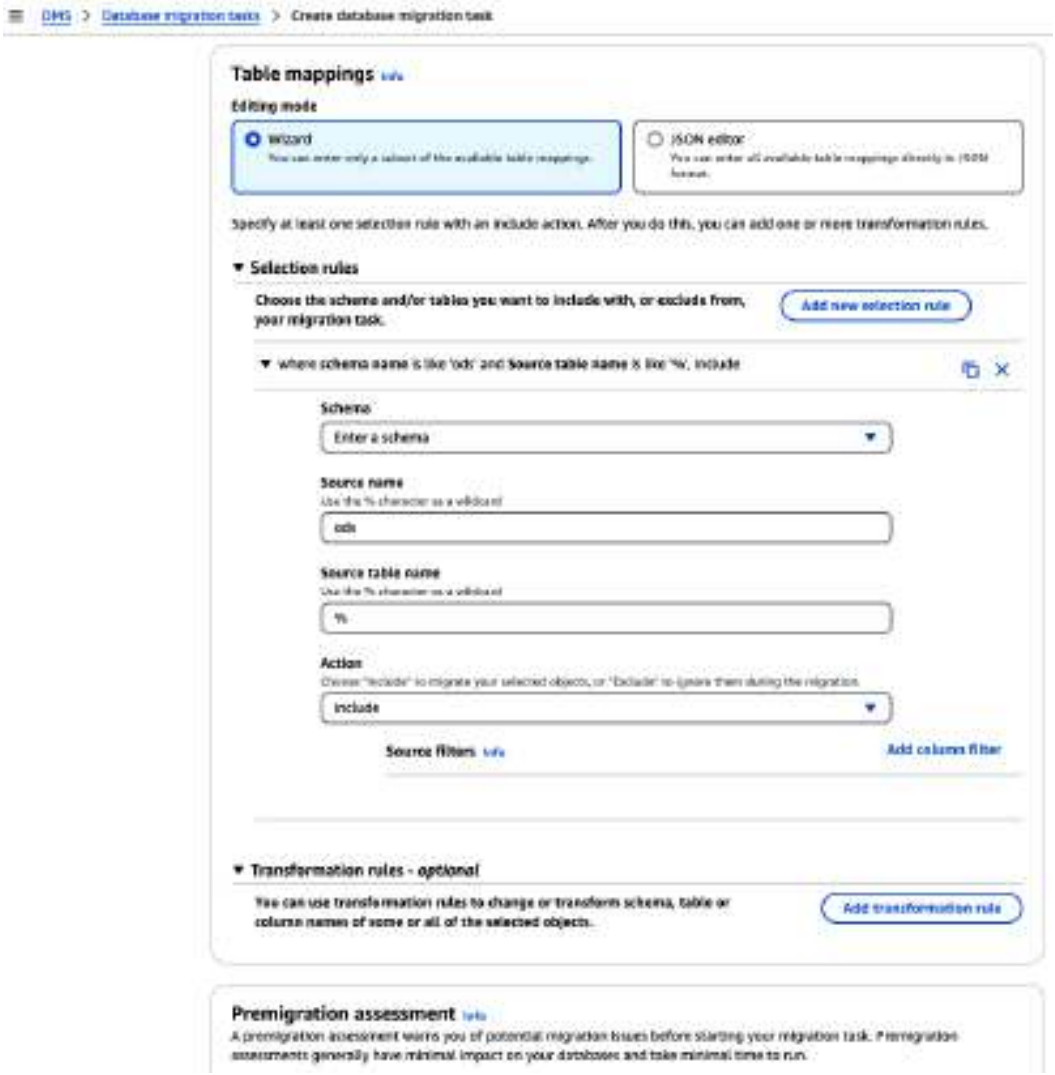


Figure 3.7 – AWS DMS migration task source table mapping rules

12. For transformation rules for the target, select the **ods** schema, set the Table name as a wildcard, and set Action to add the **stg_** prefix to the tablename on Amazon Redshift.

In the DMS task, you can apply some transformation rules, such as converting to lowercase or removing columns.

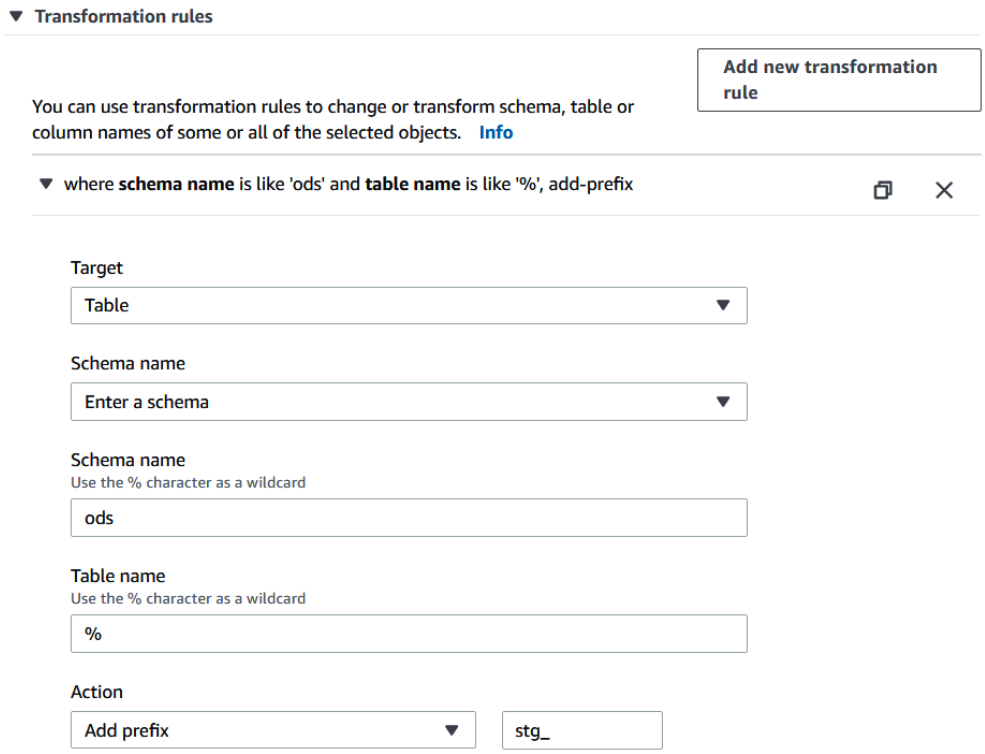


Figure 3.8 – AWS DMS migration task target transformation rule

13. In the Migration task startup configuration, select the **Manually later** option and click on **Create task**.
14. Once the task is ready, click on the task. Then, under action, select **restart and resume**. With this, the replication instance has been connected to the source and replicated data to Amazon Redshift.

15. To view the status of the replication, click on **Table statistics**. The load state on completion will display **Table completed**. The total number of rows is **20,000** in the target Amazon Redshift **ods.part** table.

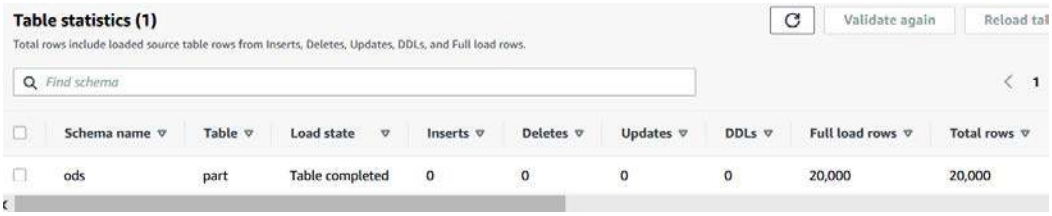


Figure 3.9 – AWS DMS migration task status and full mode replicated record count

16. Let's insert the following records in the source MySql database `part` table to see the change data capture scenario:

```
insert into ods.part values
(20001,'royal red metallic
dim','Manufacturer#2','Brand#25','STANDARD BURNISHED NICKEL',48,'SM
JAR',920.00,'sts-1');
insert into ods.part values
(20002,'royal red metallic
dim','Manufacturer#2','Brand#26','STANDARD BURNISHED NICKEL',48,'SM
JAR',921.00,'sts-2');
insert into ods.part values
(20003,'royal red metallic
dim','Manufacturer#2','Brand#27','STANDARD BURNISHED NICKEL',48,'SM
JAR',922.00,'sts-3');
insert into ods.part values
(20004,'royal red metallic
dim','Manufacturer#2','Brand#28','STANDARD BURNISHED NICKEL',48,'SM
JAR',923.00,'sts-4');
insert into ods.part values
(20005,'royal red metallic
dim','Manufacturer#2','Brand#29','STANDARD BURNISHED NICKEL',48,'SM
JAR',924.00,'sts-5');
```

17. Let's check the change data capture on the database migration task of the newly inserted five records. The **Inserts** column shows **5** and the **Total rows** on the target now has **20,005** records:

Figure 3.10 – AWS DMS migration task status and change data capture replicated record count

18. Let's confirm the record count on the `ods.stg_part` Amazon Redshift table. Execute the following query in the SQL client and the output will be 20,005 records:

```
select count(*) from ods.stg_part;
```

19. You can choose to stop the database migration task by navigating to **Database migration tasks** | **Actions** | **Stop**.

How it works...

AWS DMS provides the capability to do homogeneous (on the same database platform, such as from an on-premises MySQL to Amazon RDS MySQL) and heterogeneous (different database platform) replication. In this recipe, we saw the scenario of heterogeneous replication where the source is MySQL and the target is Amazon Redshift. Using an AWS DMS task, it first fully migrated the data to Amazon Redshift and the task captured changes from the source transactional logs, which were replicated to Amazon Redshift in near real time.

See also...

You can get more details on turning on binary logging on this link. This is to enable change data capture for AWS DMS: `https://docs.aws.amazon.com/dms/latest/userguide/CHAP_Source.MySQL.html#CHAP_Source.MySQL.AmazonManag`.

Cataloging and ingesting data using AWS Glue

Data that is staged in Amazon S3 can be cataloged using the AWS Glue service. Cataloging the data allows attaching the metadata and populating the AWS Glue Data Catalog. This process enriches the raw data that can queried as tables using many of the AWS analytical services, such as Amazon Redshift or Amazon EMR, for the analytical processing. It is easy to perform this data discovery using the AWS Glue crawlers that can create and update the metadata automatically.

In this recipe, we will enrich the data to catalog and enable ingestion into an Amazon Redshift data warehouse (serverless or provisioned cluster).

Getting ready

To complete this recipe, you will need:

- An Amazon Redshift data warehouse deployed in the eu-west-1 AWS region.
- Amazon Redshift data warehouse admin user credentials.
- An IAM user with access to Amazon Redshift, Amazon S3, and AWS Glue.
- An IAM role attached to Amazon Redshift data warehouse that can access Amazon S3, we will reference it in the recipes as `[Your-Redshift_Role]`.
- Access to any SQL interface, such as a SQL client or Amazon Redshift query editor.
- An Amazon S3 bucket for staging and unloading the data in specific recipes, we will reference it in recipes as `[Your-Amazon_S3_Bucket].` The Amazon S3 bucket must be in the same region as your Amazon Redshift data warehouse.
- An AWS account number, we will reference it in recipes as `[Your-AWS_Account_Id]`.

How to do it...

This recipe will catalog and ingest the Amazon.com customer product reviews dataset into Amazon Redshift:

1. Navigate to the AWS Console and pick **AWS Glue**. Verify that you are in the same AWS Region as the Amazon Redshift data warehouse. In the left navigation menu, inside AWS Glue, choose **Data Catalog** and then choose **Crawlers**.
2. Click the **Create Crawler** button and type in any crawler name, such as `product reviews dataset crawl` and click **Next**.

3. On the Choose data sources and classifiers page, follow the instructions as per the screenshots in *Figure 3.11*. In the Data source configuration section, for the question **Is your data already mapped to Glue tables?** select **Not yet**, and then choose **Add a data source**. In the dialog box that appears, enter the following information:

    a. For Data source, choose S3.

    b. For the location of S3 data, you can indicate whether the data is in the same account or a different account. For this recipe walkthrough, choose **In a different account**.

    c. For S3 path, copy and paste the path `s3://packt-redshift-cookbook/amazon-reviews-pds/parquet/`.

    d. For the **subsequent crawler runs** option, you can choose how the crawler behaves. For this walkthrough, choose **Crawl all sub-folders** and choose **Add an S3 data source**.

    e. Back on the **Choose data sources and classifiers** page, in the **Data Sources** section, you will see the S3 data source you added. Choose **Next** on this page.

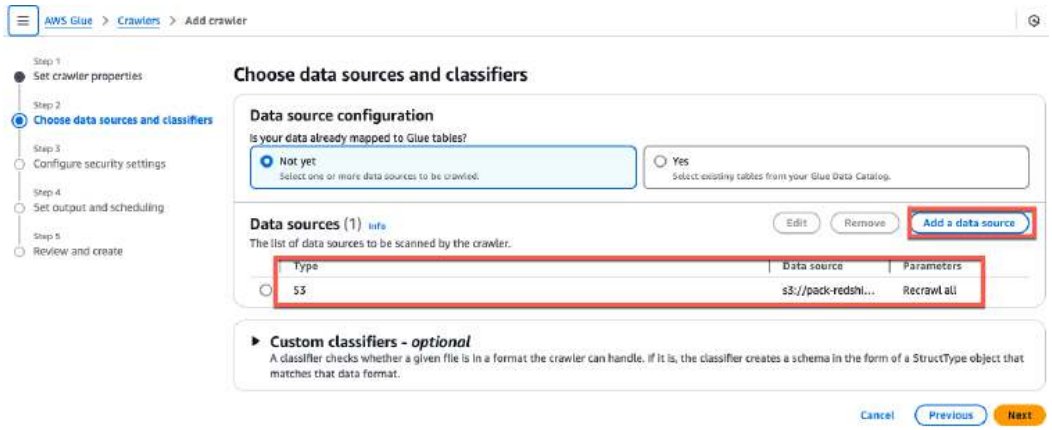


Figure 3.11 – Adding a crawler

4. On the **Configure security settings** page, for **IAM Role**, choose an IAM role to allow AWS Glue access to crawl and update the AWS Glue Catalog and click on the **Next** button.

5. On the **Set output and scheduling** page, in the **Output Configuration** section, for **Target database**, click on **Add database** and give the database a representative name, such as `reviews`, and click **Create Database**.

Go back to the **Set output and scheduling** page, choose the refresh button next to the **Target database** dropdown, choose **reviews**, and for **Table name prefix**, enter `product_reviews_src`. Finally, click **Next**.

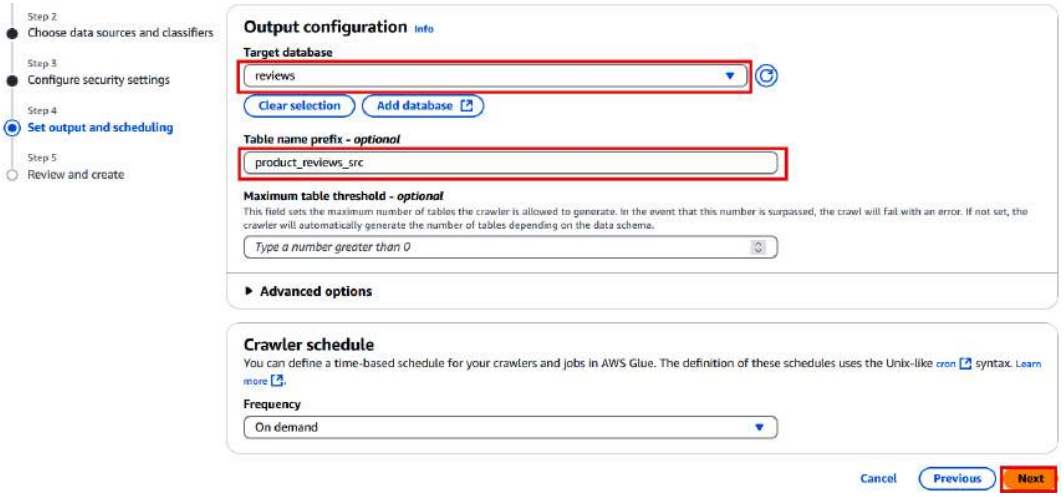


Figure 3.12 – Configure the crawler output

6. On **Review and create** page, click on **Create crawler** to create the crawler. On the crawler details page, choose **Run crawler** and wait until the status changes to **Success**:

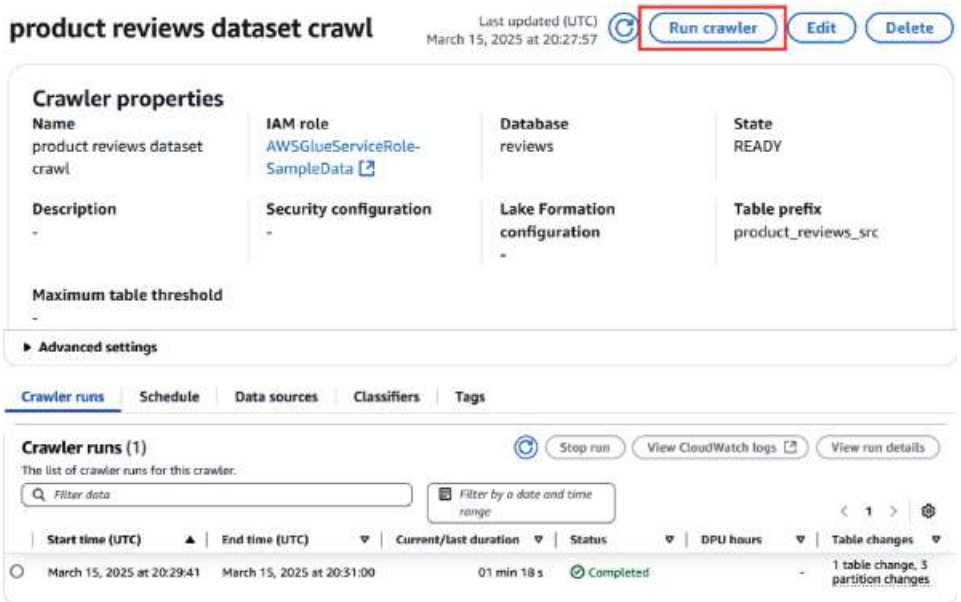


Figure 3.13 – Monitor the crawler status

7. Now, AWS Glue has crawled the product review dataset and discovered the table automatically. You can verify the table by navigating to the **Tables** option to view the `product_reviews_srcparquet` table in the list:

Figure 3.14 – View the table created by the crawler

8. Open any SQL client tool, connect to Amazon Redshift, and create a schema to point to the `reviews` AWS Glue catalog database using the following command by replacing the `[Your-AWS_Account_Id]` and `[Your-Redshift_Role]` values:

```
CREATE external SCHEMA review_ext_sch FROM data catalog DATABASE
'reviews' iam_role 'arn:aws:iam::[Your-AWS_Account_Id]:role/[Your-
Redshift-Role]' CREATE external DATABASE if not exists;
```

9. Create the `product_reviews_stage` table that will hold the incoming crawled data:

```
CREATE TABLE product_reviews_stage
(
  marketplace         VARCHAR(2),
  customer_id         VARCHAR(32),
  review_id           VARCHAR(24),
  product_id          VARCHAR(24),
  product_parent      VARCHAR(32),
  product_title       VARCHAR(512),
  star_rating         INT,
  helpful_votes       INT,
  total_votes         INT,
  vine                CHAR(1),
  verified_purchase   CHAR(1),
  review_headline     VARCHAR(256),
  review_body         VARCHAR(MAX),
  review_date         DATE,
```

```
  YEAR                INT
)
DISTSTYLE KEY DISTKEY (customer_id) SORTKEY (review_date);
```

10. Now, let's insert `Automotive` data from the crawled AWS Glue table into the Amazon Redshift table, `product_reviews_stage`:

```
INSERT INTO product_reviews_stage
(
  marketplace,
  customer_id,
  review_id,
  product_id,
  product_parent,
  product_title,
  star_rating,
  helpful_votes,
  total_votes,
  vine,
  verified_purchase,
  review_headline,
  review_body,
  review_date,
  year
)
SELECT marketplace,
       customer_id,
       review_id,
       product_id,
       product_parent,
       product_title,
       star_rating,
       helpful_votes,
       total_votes,
       vine,
       verified_purchase,
       review_headline,
       review_body,
```

```
        review_date,
        year
FROM review_ext_sch.reviewparquet
WHERE product_category = 'Automotive';
```

11. Now, the `public.product_reviews_stage` table is ready to hold the incoming Automotive dataset, which can be verified by using the following command:

```
SELECT *
FROM product_reviews_stage;
```

How it works...

AWS Glue provides a crawler that can automatically figure out the structure of data in Amazon S3. AWS Glue maintains the metadata catalog that can be accessed across other AWS analytical services, such as Amazon Redshift. Amazon Redshift can read/write data into Amazon S3 directly using the Amazon Redshift spectrum feature.

Streaming data to Amazon Redshift via Amazon Data Firehose

Streaming data is a continuous dataset that can originate from sources such as IoT devices, log files, gaming systems, and so on. Ingesting the streaming data into Amazon Redshift allows you to run near real-time analytics that can be combined with historical/operational data to produce actionable reporting. For example, in a manufacturing shop, analyzing data from several IoT sensors can help predict the failure of machinery so you can take preventive action.

In this recipe, we will simulate streaming data using the Amazon.com product review data to be ingested into Amazon Redshift using Amazon Kinesis Firehose. Amazon Kinesis Firehose provides out-of-the-box integration to capture the streaming datasets and place them into an Amazon Redshift table.

Getting ready

To complete this recipe, you will need:

- An Amazon Redshift data warehouse deployed in the eu-west-1 AWS region.
- Amazon Redshift data warehouse admin user credentials.
- An IAM user with access to Amazon Redshift, Amazon Kinesis, Amazon Cognito, and Amazon S3.
- Access to any SQL interface, such as a SQL client or Amazon Redshift query editor.
- An Amazon S3 bucket created in eu-west-1. We will reference it as [Your-`Amazon_S3_Bucket`].
- An IAM role attached to Amazon Redshift data warehouse that can access Amazon S3, we will reference it in the recipes as [`Your-Redshift_Role`].
- Access to Kinesis data generator. This is a UI that sends test data to Amazon Kinesis. Use this blog post to configure the open source Kinesis data generator (reference: `https://aws.amazon.com/blogs/big-data/test-your-streaming-data-solution-with-the-new-amazon-kinesis-data-generator/`).
- An AWS account number, we will reference it in recipes as [`Your-AWS_Account_Id`].

How to do it...

This recipe will stream the `Amazon.com` customer product reviews dataset and ingest it into Amazon Redshift using the Amazon Kinesis Firehose:

1. Navigate to the AWS Console and pick the Amazon Data Firehose service. In the left menu, choose **Firehose streams** and click on the **Create Firehose stream** button, as shown in the following screenshot:

Figure 3.15 – Creating a Kinesis Data Firehose stream

2. In the **Choose source and destination** section, for source, choose **Direct PUT**, and for the destination, choose **Amazon Redshift**.
3. Provide a **Firehose stream name** such as `product_reviews_stream` and click **Next** until you get to the Choose a destination option.
4. In the **Destination settings** section, provide the following parameters:
    - **Choose Amazon Redshift destination type**: Serverless workgroup or provisioned cluster based on the type of Amazon Redshift data warehouse you created
    - **Serverless workgroup**: If your Amazon Redshift data warehouse is serverless, choose your workgroup
    - **Provisioned cluster**: If your Amazon Redshift data warehouse is a provisioned cluster, choose your cluster
    - **Database**: Type your database name
    - **User name**: Type the username that you chose when you set up the Amazon Redshift data warehouse
    - **Password**: Type the password that you chose when you set up the Amazon Redshift data warehouse
    - **Database**: Type database name
    - **Table**: Type `product_reviews_stg`
    - **Columns - optional**: Leave this field empty.
    - **Intermediate S3 destination**: Choose an existing bucket or create an S3 bucket where data will be staged before it is copied into Amazon Redshift `[Your-Amazon_S3_Bucket]`
    - **Backup S3 bucket prefix - optional**: Type `/product_review_stg/`

5. In **COPY command options – optional**, type the following script:

```
COPY product_reviews_stg (marketplace,customer_id,review_id,product_
id,product_parent,product_title,star_rating,helpful_votes,total_
votes,vine,verified_purchase,review_headline,review_body,review_
date,year) FROM 's3://[Your-Amazon_S3_Bucket/product_review_stg/
manifest' CREDENTIALS 'aws_iam_role=arn:aws:iam::[Your-AWS_Account_
Id]:role/[Your-Redshift_Role]' MANIFEST  JSON 'auto';
```

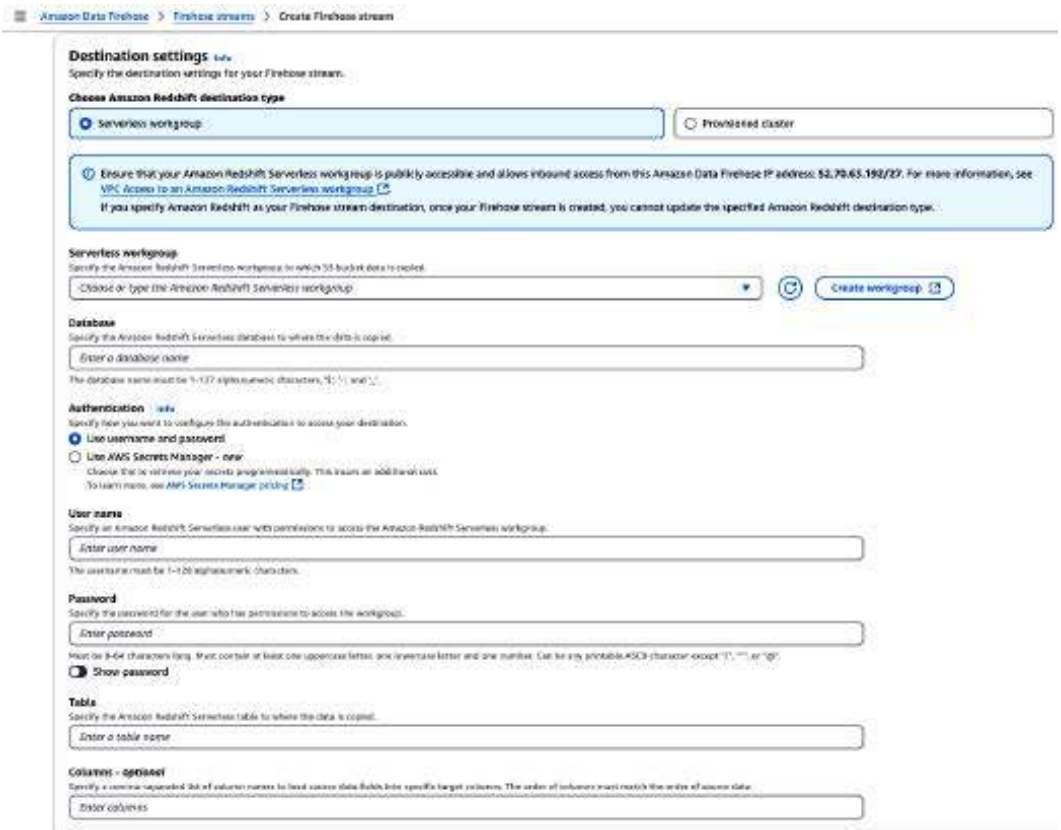


Figure 3.16 – Configure the Destination Amazon Redshift data warehouse

6. Navigate to the **Review** option and create the Amazon Kinesis Firehose stream.
7. Log into the Amazon Redshift data warehouse using the SQL client tool and create the `product_reviews_stg` table that will hold the incoming streaming data:

```
CREATE TABLE product_reviews_stg
(
  marketplace         VARCHAR(2),
  customer_id         VARCHAR(32),
  review_id           VARCHAR(24),
  product_id          VARCHAR(24),
  product_parent      VARCHAR(32),
  product_title       VARCHAR(512),
  star_rating         INT,
  helpful_votes       INT,
  total_votes         INT,
  vine                CHAR(1),
  verified_purchase   CHAR(1),
  review_headline     VARCHAR(256),
  review_body         VARCHAR(MAX),
  review_date         DATE,
  YEAR                INT
)
DISTSTYLE KEY DISTKEY (customer_id) SORTKEY (review_date);
```

8. Now, let's use the Amazon Kinesis Data Generator to produce streaming data and send it to the `product_reviews_stream` Kinesis Firehose Stream as follows:

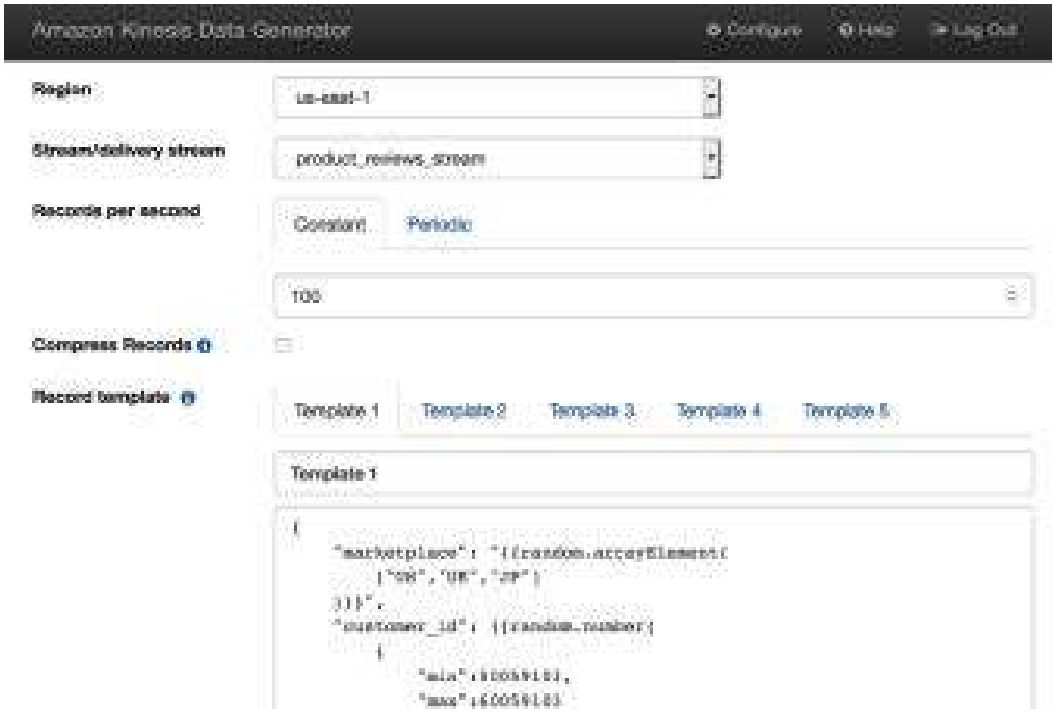


Figure 3.17 – Amazon Kinesis Data Generator

Here, you will use the stream/delivery stream as `product_review_stream` to send the streaming data and copy and paste the template from `https://github.com/PacktPublishing/Amazon-Redshift-Cookbook-2E/blob/master/Chapter04/kinesis_data_generator_template.json` to generate the product reviews data:

```
{
    "marketplace": "{{random.arrayElement(
        ["US","UK","JP"]
    )}}",
…
    "review_headline": "{{commerce.productAdjective}}",
    "review_body": "{{commerce.productAdjective}}",
    "review_date": "{{date.now("YYYY-MM-DD")}}",
    "year":{{date.now("YYYY")}}
}
```

9. After a while, the stream data should start landing into Amazon Redshift, that can be verified by using the following code:

```
SELECT *
FROM product_reviews_stage;
```

How it works...

Amazon Kinesis Data Firehose allows data to be sourced from and streamed into multiple destinations. It can capture, transform, and load streaming data into destinations of Amazon S3, Amazon Redshift, Amazon Elasticsearch Service, and Splunk. Kinesis Firehose, being a fully managed service, can automatically scale to meet the growth of the data.

Unloading data to Amazon S3

Amazon Redshift enables you to create a copy of your data stored in Amazon S3 using the `UNLOAD` command. This command leverages the power of Amazon Redshift's massive parallelism to unload data in parallel to Amazon S3, utilizing multiple concurrent splits.

This recipe will demonstrate how to utilize the `UNLOAD` command to export data from an Amazon Redshift data warehouse (serverless or provisioned cluster) to an Amazon S3 bucket.

Getting ready

To complete this recipe, you will need:

- Access to the AWS Console
- An Amazon Redshift data warehouse deployed in the AWS eu-west-1 region, and data loaded as referenced in the Loading data from Amazon S3 recipe
- Amazon Redshift data warehouse admin user credentials
- Access to any SQL interface such as a SQL client or the Amazon Redshift query editor
- Amazon S3 bucket created in eu-west-1; we will reference it as `[Your-Amazon_S3_Bucket]`
- IAM role attached to Amazon Redshift data warehouse that can access Amazon S3

How to do it...

To unload the data from Amazon Redshift to an Amazon S3 bucket, we will use the following steps:

1. Connect to the Redshift data warehouse using the SQL client of your choice.
2. Use the following command to unload the data from the Amazon Redshift data warehouse. Replace the values in [ ] with the corresponding values for your environment:

```
unload ('select * from orders')
to 's3://[Your-Amazon_S3_Bucket]/unload/orders_ '
iam_role 'arn:aws:iam::[Your-AWS_Account_Id]:role/[Your-Redshift_
Role]'
PARQUET;
```

   The `UNLOAD` command will write data in Parquet format to multiple files in parallel.

3. To validate the path for the unloaded data, you can use the following SQL statement, which queries the system table, `SYS_UNLOAD_DETAIL`:

```
select query_id, substring(file_name,0,100) as path
from SYS_UNLOAD_DETAIL
where query_id=pg_last_query_id()
order by path limit 10;

--expected sample output
query  path
21585117    s3://[ Your-Amazon_S3_Bucket]/unload/orders_000_
part_000.parquet
21585117    s3://[ Your-Amazon_S3_Bucket]/unload/orders_001_
part_000.parquet
21585117    s3://[ Your-Amazon_S3_Bucket]/unload/orders_002_
part_000.parquet
..
```

4. To confirm that the data is unloaded from Amazon S3, you can browse the Amazon S3 bucket and list the output parquet files. Note that when unloading data to columnar formats like parquet, the Amazon Redshift data warehouse and Amazon S3 bucket should be in the same region.

See also...

You can review the `https://docs.aws.amazon.com/redshift/latest/dg/r_UNLOAD.html` documentation for additional parameters that `UNLOAD` supports.

4

Zero-ETL Ingestions

AWS zero-ETL represents a transformative suite of fully managed integrations that revolutionize how organizations handle their data analytics needs. This comprehensive solution encompasses native database integrations, streaming capabilities, automated S3 ingestion, and federated querying—all working together to eliminate traditional ETL complexities. By automatically replicating data from operational sources to analytical destinations, zero-ETL enables near-real-time insights without building and maintaining complex data pipelines. This modern approach dramatically reduces time to insight, ensures data consistency, and allows organizations to scale their data operations efficiently while maintaining separation between transactional and analytical workloads.

The integration of native connectors for multiple applications, combined with automated data synchronization and transformation capabilities, enables businesses to make faster, data-driven decisions while significantly reducing operational overhead and technical complexity.

The following recipes are discussed in this chapter:

- Ingesting data from Aurora MySQL/Aurora Postgres/RDS MySQL using zero-ETL integration
- Ingesting data from Amazon DynamoDB using zero-ETL integration
- Ingesting data from SaaS applications like Salesforce using zero-ETL integration
- Ingesting streaming data from Amazon **Kinesis Data Streams (KDS)**
- Ingesting streaming data from Amazon **Managed Streaming for Apache Kafka (MSK)**
- Near-real-time ingestion of data from Amazon S3 using auto-copy

Technical requirements

Here are the technical requirements in order to complete the recipes in this chapter:

- Access to the AWS Management Console.
- An Amazon Redshift data warehouse deployed in the AWS Region eu-west-1.
- Amazon Redshift data warehouse admin user credentials.
- Attach an IAM role to the Amazon Redshift data warehouse by following *Recipe 4* in *Appendix*. Make note of the IAM role name; we will reference it in the recipes as `[Your-Redshift_Role]`.
- An AWS administrator should deploy the AWS CloudFormation template (`https://github.com/PacktPublishing/Amazon-Redshift-Cookbook-2E/blob/main/Chapter04/chapter_4_CFN.yaml`) to create two IAM policies:
    - An IAM policy attached to the IAM user that will give them access to Amazon Redshift, Amazon RDS, Amazon Kinesis, Amazon Kinesis Data Firehose, Amazon CloudWatch Logs, AWS CloudFormation, AWS Secret Manager, Amazon Cognito, Amazon S3, AWS DMS, and AWS Glue
    - An IAM policy attached to the IAM role that will allow an Amazon Redshift data warehouse to access Amazon S3
- Access to any SQL interface, such as a SQL client or Amazon Redshift Query Editor V2.
- An AWS account number; we will reference it in the recipes as `[Your-AWS_Account_Id]`.
- An Amazon S3 bucket created in eu-west-1; we will reference it as `[Your-Amazon_S3_Bucket]`.

The code files are provided in the GitHub repo: `https://github.com/PacktPublishing/Amazon-Redshift-Cookbook-2E/tree/main/Chapter04`.

Ingesting data from Aurora MySQL/Aurora Postgres/ RDS MySQL using zero-ETL integration

You can replicate data from Aurora MySQL, Aurora Postgres, or RDS MySQL databases into Amazon Redshift in near-real time using a zero-ETL integration. A zero-ETL integration is easy to use. You need not build complex ETL pipelines or use additional AWS services or third-party solutions for replications. The zero-ETL integration performs a one-time seeding of data and then keeps the data in sync with ongoing changes, providing data freshness latencies of just a few seconds. You can also include or exclude tables or databases that you don't need from replication.

If you don't want near-real-time ingestion and want to ingest at a specific interval, you can achieve that as well using this solution.

In this recipe, you will load sample data into your source Aurora MySQL/Postgres or RDS MySQL database, create a zero-ETL integration, and replicate it to Amazon Redshift. You will make some changes to the source data and see the changes reflected in the target Amazon Redshift data warehouse within seconds.

Getting ready

To complete this recipe, you will need:

- An Amazon Redshift data warehouse deployed in the AWS Region eu-west-1
- An Amazon Aurora MySQL, Aurora Postgres, or RDS MySQL cluster deployed in the AWS Region eu-west-1 in the same VPC as the Amazon Redshift data warehouse (`https://docs.aws.amazon.com/AmazonRDS/latest/AuroraUserGuide/zero-etl.setting-up.html`)
- For a list of supported Aurora MySQL versions, see `https://docs.aws.amazon.com/AmazonRDS/latest/AuroraUserGuide/Concepts.Aurora_Fea_Regions_DB-eng.Feature.Zero-ETL.html`
- For a list of supported Aurora PostgreSQL versions, see `https://docs.aws.amazon.com/AmazonRDS/latest/AuroraUserGuide/Concepts.Aurora_Fea_Regions_DB-eng.Feature.Zero-ETL.html#Concepts.Aurora_Fea_Regions_DB-eng.Feature.Zero-ETL-Postgres`
- For a list of supported RDS MySQL versions, see `https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/Concepts.RDS_Fea_Regions_DB-eng.Feature.ZeroETL.html`

How to do it...

This recipe will illustrate the replication of a parts table from your source database cluster (Amazon Aurora MySQL/Postgres or RDS MySQL) to Amazon Redshift:

1. Let's start by creating the necessary tables in an Aurora MySQL/Postgres or RDS MySQL database. Start by connecting to your database using the query editor (`https://docs.aws.amazon.com/AmazonRDS/latest/AuroraUserGuide/query-editor.html`) for Aurora MySQL/Postgres or using the command line for RDS MySQL (`https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/USER_ConnectToInstance.html`)
2. Create an `ods` database using the following SQL statement:

```
create database ods;
```

3. Create a `parts` table in the `ods` database using the following SQL statement. It is mandatory to have a primary key on your tables for zero-ETL integration to function. The syntax of this statement is the same for both MySQL and Postgres:

```
CREATE TABLE ods.part
(
  P_PARTKEY       BIGINT NOT NULL PRIMARY KEY,
  P_NAME          VARCHAR(55),
  P_MFGR          VARCHAR(25),
  P_BRAND         VARCHAR(10),
  P_TYPE          VARCHAR(25),
  P_SIZE          INTEGER,
  P_CONTAINER     VARCHAR(10),
  P_RETAILPRICE   DECIMAL(18,4),
  P_COMMENT       VARCHAR(23)
);
```

4. Load sample data into the `ods.part` table using the `insert` statement provided in the GitHub repository (`https://github.com/PacktPublishing/Amazon-Redshift-Cookbook-2E/blob/main/Chapter04/part_table_insert1.sql`). This `insert` statement inserts 50 rows into the table. You can use the same statement for MySQL as well as Postgres databases.
5. Verify the record count loaded into the `ods.part` table using the following SQL:

```
select count(*) from ods.part;
+----------+
| count(*) |
+----------+
|    50    |
+----------+
1 row in set (0.00 sec)
```

6. Now, we will create a zero-ETL integration in the Amazon RDS console. Navigate to the RDS console (`https://console.aws.amazon.com/rds/home`) and choose **Zero-ETL integrations** from the left navigation menu. Then click **Create zero-ETL integration**.

Figure 4.1 – Create zero-ETL integration

7. Provide a representative name for the integration, such as `product-redshift-integ`, and choose **Next**:

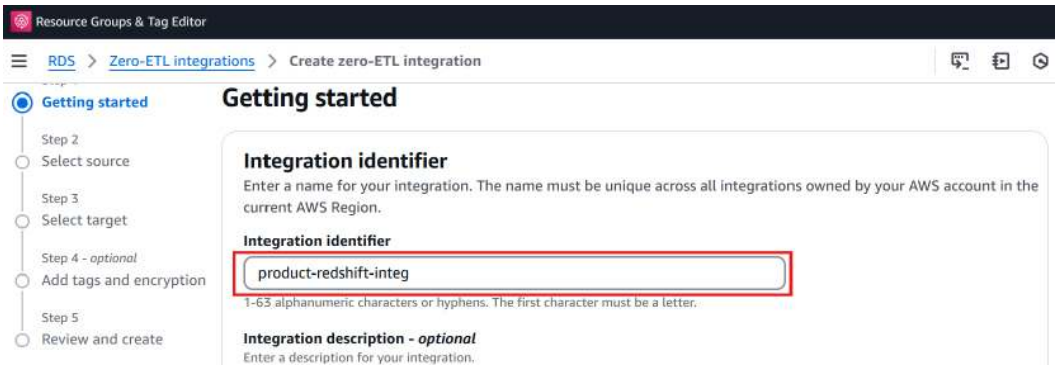


Figure 4.2 – Give the zero-ETL integration a name

8. On the **Select source** page, click the **Browse RDS databases** button and choose your source Aurora MySQL/Postgres or RDS MySQL database. If the database is missing any required parameters, you will see a **Fix it for me (requires reboot)** checkbox. This will make the necessary updates to parameter groups and reboot the cluster. Select **Fix it for me (requires reboot)**, as shown in the following screenshot, and click **Next**:

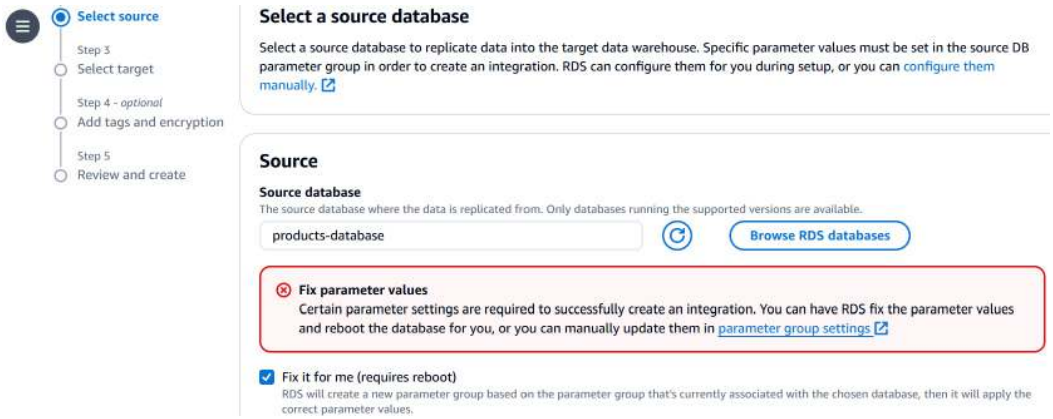


Figure 4.3 – Choose source database when creating zero-ETL integration

9. In the **Data filtering options - optional** section, you can choose the tables you would like to include or exclude. If you want to replicate the `part` table only, enter `ods.part`, as shown in the following screenshot. The filter pattern is in the format `database.table` for Aurora MySQL, or `database.schema.table` for Aurora PostgreSQL. You can specify literal names or define regular expressions. If you select an Aurora PostgreSQL source database cluster, you must specify at least one data filter pattern. At a minimum, the pattern must include a single database (`database-name.*.*`) for replication to Amazon Redshift. In this example, we want to include all tables in the `ods` database. So, the filter pattern is `ods.*.*` for Postgres.

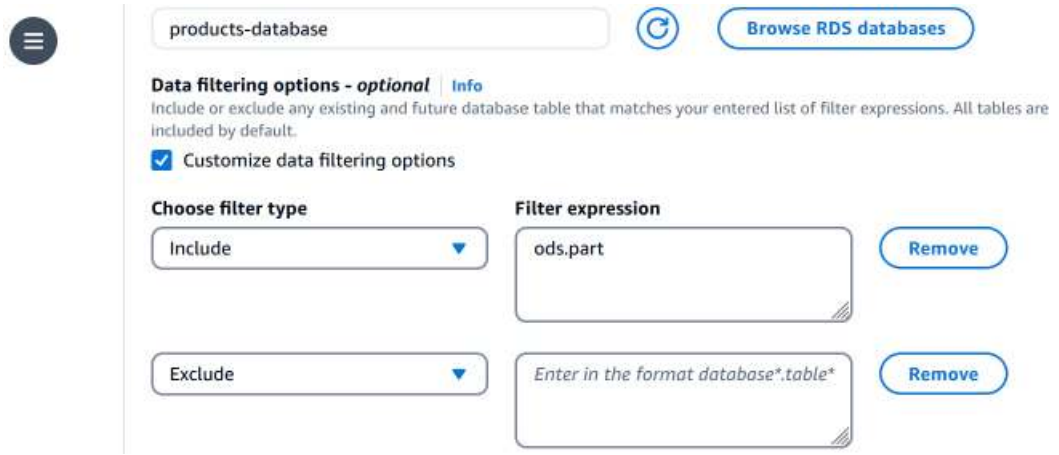


Figure 4.4 – Data filter options in zero-ETL integration

10. If you chose **Fix it for me (requires reboot)** in *step 8*, a confirmation page will appear for the changes that will be made to the source database. Provide the necessary confirmation and go to the next step.
11. On the **Select target** page, choose **Browse Redshift data warehouses** and then select your target Amazon Redshift data warehouse. If any required parameters (`https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/zero-etl.setting-up.html#zero-etl-setting-up.data-warehouse`) are missing from the target data warehouse, you will see a **Fix it for me** checkbox.

    Select it, as shown in the following screenshot, and click **Next**:

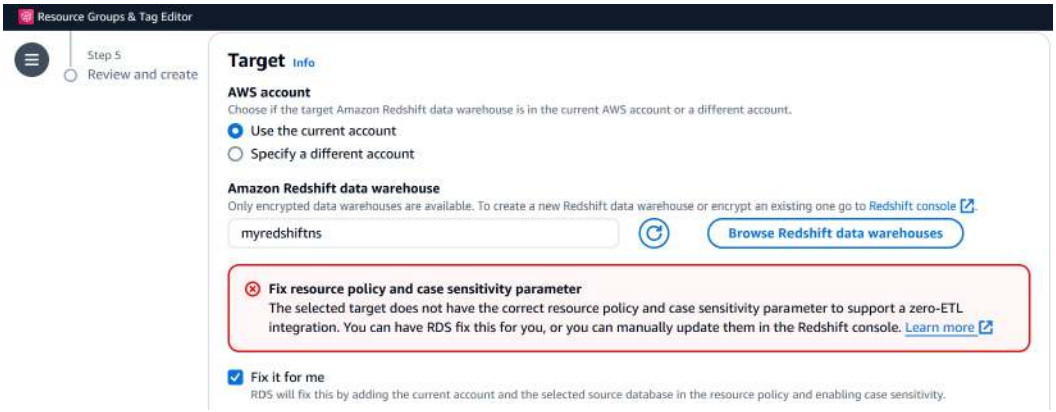


Figure 4.5 – Choose data target for zero-ETL integration

12. If you chose **Fix it for me** in the previous step, a **Review the changes** dialog box will appear. Verify the changes and click **Continue**. Click **Next** and then select **Create zero-ETL integration** to create the integration. It will take 15-20 minutes to create the integration depending on your data volume. The integration is active when its status changes to **Active**.
13. Next, navigate to the Amazon Redshift console and choose **Zero-ETL integrations** from the navigation pane on the left. You will see the integration you created listed, as shown in the following screenshot:

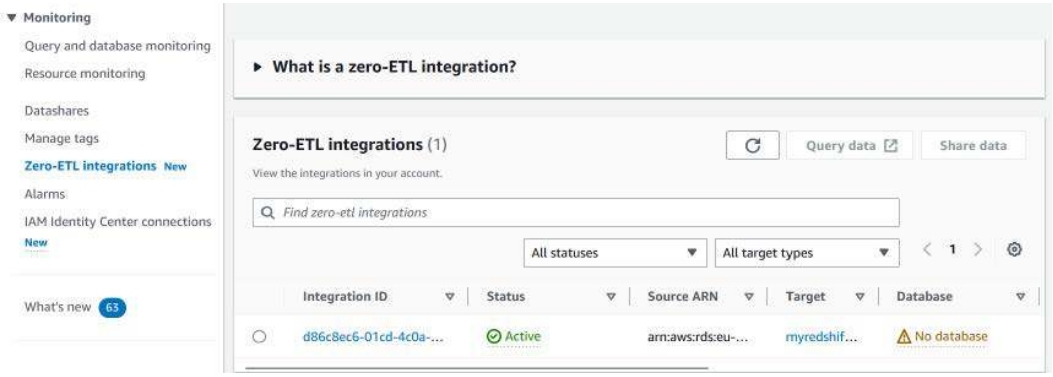


Figure 4.6 – Zero-ETL integration available in Amazon Redshift

14. Click the link listed under **Integration ID** and on the page that opens, you will get an alert saying that you need to create a database to access the data. Click **Create database from integration**:

Figure 4.7 – Create database for zero-ETL integration

15. On the page that opens, provide a name for the database that will be created, like `product_database_zetl`, and then select **Create database**.
16. You can now start querying data in Amazon Redshift. Connect to your Amazon Redshift data warehouse using Amazon Redshift Query Editor V2 (`https://console.aws.amazon.com/sqlworkbench/home`) to see the database you created listed there. Expand the database to see the replicated tables. You can change the database from the database selection dropdown and start querying the data using simple `SELECT` statements, as shown in the following screenshot:

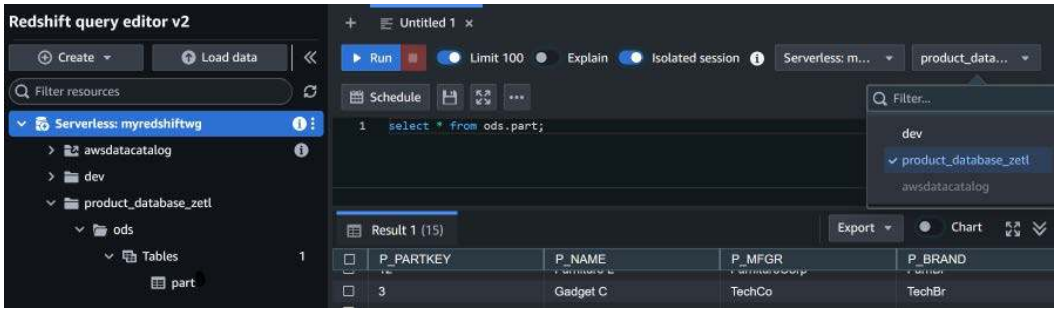


Figure 4.8 – Querying replicated tables in Amazon Redshift

17. Let's now add 50 new rows to the `parts` table using the `insert` statement in the GitHub repository (`https://github.com/PacktPublishing/Amazon-Redshift-Cookbook-2E/blob/main/Chapter04/part_table_insert2.sql`).
18. These 50 records will be replicated to the Amazon Redshift data warehouse within seconds. Let's confirm the record count on the Amazon Redshift table `ods.part` by executing the following count query.

    The output of this query will be `100`:

```
select count(*) from ods.part;
+----------+
| count(*) |
+----------+
|    100   |
+----------+
1 row in set (0.00 sec)
```

19. By default, the zero-ETL integration maintains a replica of the source table in the Amazon Redshift data warehouse. You can change this default behavior and track every version (including updates and deletes) of your records in source tables. This allows you to run advanced analytics on all your data, such as running a historical analysis, building look-back reports, performing trend analysis, and sending incremental updates to downstream applications built on top of Amazon Redshift. To manage history mode for a zero-ETL integration, open Amazon Redshift Query Editor V2 (`https://console.aws.amazon.com/sqlworkbench/home`).
20. From the left navigation pane, choose **Zero-ETL integrations** and select the zero-ETL integration that you want to manage. Then, select **Manage history mode**. The **Manage history mode** window is displayed:

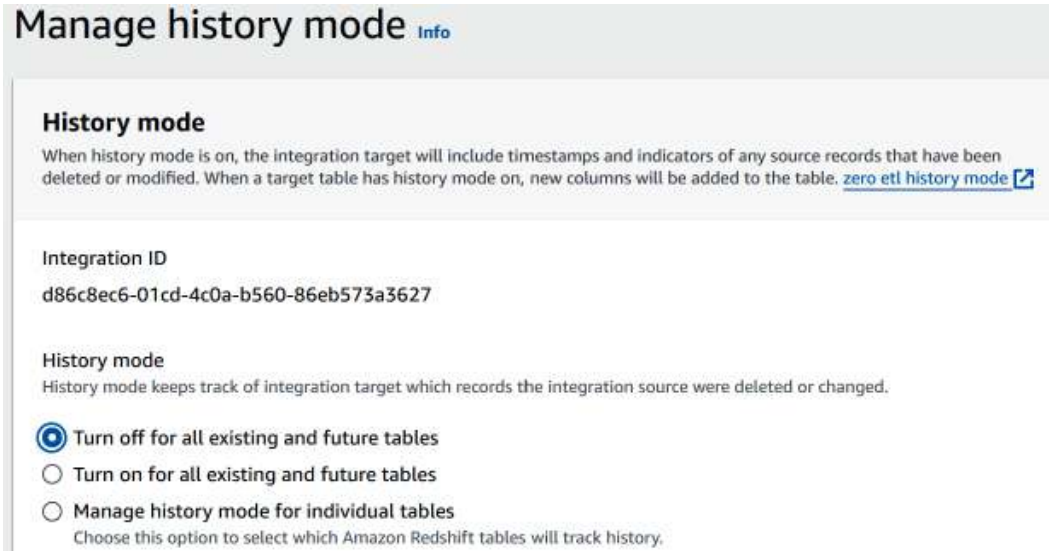


Figure 4.9 – History mode for zero-ETL integrations

21. You can either turn it on or off for all future and existing tables. You can also manage this at an individual table level. Three fields provided in the following table will be added to Amazon Redshift for the tables for which history mode is activated:

`_record_is_active`	Boolean	Indicates if a record in the target is currently active in the source. `True` indicates the record is active.
`_record_create_time`	Timestamp	Starting time (UTC) when the source record is active.
`_record_delete_time`	Timestamp	Ending time (UTC) when the source record is updated or deleted.

You can modify the sort keys of your tables replicated through the zero-ETL integration and achieve faster and more efficient querying of your replicated data in Amazon Redshift. Set the sort key to `AUTO` and allow Amazon Redshift to observe your workload and automatically set a sort key based on your evolving workload and data patterns.

How it works...

Zero-ETL integration automatically replicates data from supported sources (Aurora MySQL, Aurora PostgreSQL, or RDS for MySQL) to Amazon Redshift within seconds. The integration creates a destination database in Redshift for querying replicated data using SQL and automatically monitors and repairs the replication pipeline.

There is inbuilt monitoring and observability for zero-ETL integrations. Refer to `https://docs.aws.amazon.com/AmazonRDS/latest/AuroraUserGuide/zero-etl.describingmonitoring.html` for more information.

You can create zero-ETL integrations from one Aurora MySQL/RDS MySQL database to multiple Amazon Redshift target data warehouses. For an Aurora Postgres database, the association is one-to-one; that is, you can create only one zero-ETL integration for each Aurora Postgres source database.

If you don't need near-real-time replication and want to replicate every few minutes, you can alter the zero-ETL database and set a `REFRESH_INTERVAL`, for example:

```
ALTER DATABASE sample_integration_db INTEGRATION SET REFRESH_INTERVAL 600;
```

The interval can be set to 0–432,000 seconds (5 days) for zero-ETL integrations whose source type is Aurora MySQL, Aurora PostgreSQL, or RDS for MySQL.

If any tables fail replication, you will see an error in the zero-ETL monitoring section. Fix the error and refresh the tables as shown in the following line of SQL:

```
ALTER DATABASE sample_integration_db INTEGRATION REFRESH INERROR TABLES in
SCHEMA sample_schema;
```

Ingesting data from Amazon DynamoDB using zero-ETL integration

You can seamlessly replicate data from Amazon DynamoDB tables into Amazon Redshift using a zero-ETL integration. This native integration eliminates the need for complex ETL pipelines, additional AWS services, or third-party tools for data replication. The zero-ETL integration process begins with an initial bulk load of your DynamoDB data into Redshift, followed by continuous synchronization of any changes. You will create one zero-ETL integration per DynamoDB table. While the minimum latency is 15 minutes, you have the option to configure it to sync data at specific intervals based on your requirements.

In this recipe, you will create a sample DynamoDB table called `Music`, populate sample data into it, establish a zero-ETL integration with Amazon Redshift, and observe the replication in action.

Getting ready

To complete this recipe, you will need an Amazon Redshift data warehouse deployed in the AWS Region eu-west-1.

How to do it...

Create a new `Music` table using the DynamoDB console:

1. Sign in to the AWS Management Console and open the DynamoDB console (`https://console.aws.amazon.com/dynamodb/`).

2. In the left navigation pane, choose **Tables** and then select **Create table**. On the **Table details** page, for **Table name**, enter `Music`, for **Partition key**, enter `Artist`, and for **Sort key**, enter `SongTitle`. Then, select **Create table** to create the table:

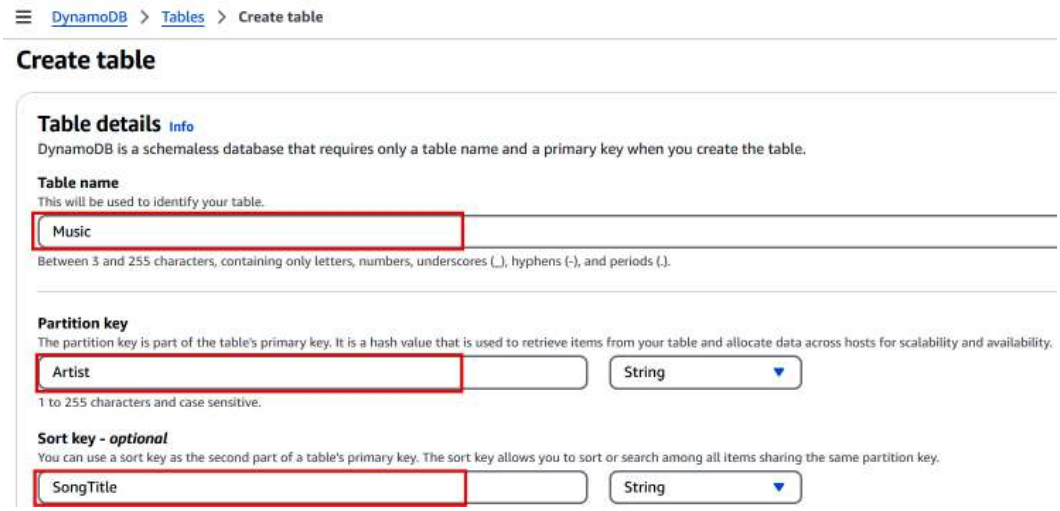


Figure 4.10 – Create a DynamoDB table

3. Next, create a zero-ETL integration for this table into Amazon Redshift, navigate to the DynamoDB console (`https://console.aws.amazon.com/dynamodb/`), and choose **Tables** in the left navigation pane. On the **Tables** page, select the **Music** table and then **Explore table items**. In the **Items returned** section, select **Create item**.
4. On the **Create item** page, select **Add new attribute**, and then select **Number**. For **Attribute name**, enter `Awards`. Repeat this process to create an attribute called `AlbumTitle` of type **String**.

Following this process, create three items with the following values:

Artist	SongTitle	AlbumTitle	Awards
No One You Know	Call Me Today	Somewhat Famous	1
Acme Band	Happy Day	Songs About Life	10
Acme Band	PartiQL Rocks	Another Album Title	8

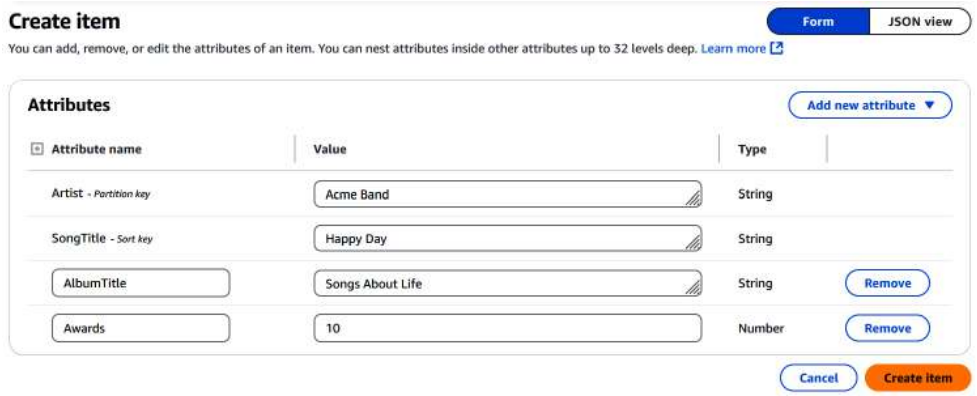


Figure 4.11 – Create items in a DynamoDB table

Next, to create the zero-ETL integration for the `Music` table, follow these steps:

1. Navigate to the Amazon Redshift **Zero-ETL integrations** page (`https://console.aws.amazon.com/redshiftv2/home#/zero-etl-integrations`) and choose **Create DynamoDB integration**, as shown in the following screenshot:

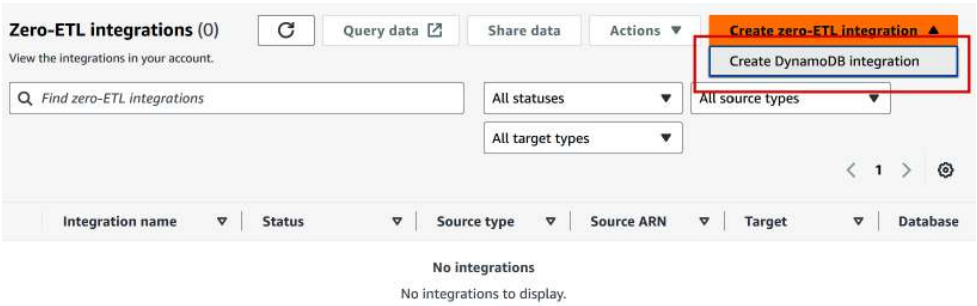


Figure 4.12 – Create DynamoDB integration

2. For **Integration Name**, enter a representative name, such as `music-integration-to-redshift`, and click **Next**.
3. On the **Select source** page, click **Browse DynamoDB tables** and choose the **Music** table. If there are any parameters needed for the zero-ETL integration that are missing, the setup wizard will provide a **Fix it for me** option, as shown in the following screenshot. Select that option and click **Next**.

   The **Fix it for me** feature will automatically apply the necessary prerequisites to the source DynamoDB table (`https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/RedshiftforDynamoDB-zero-etl.html#RedshiftforDynamoDB-zero-etl-prereqs`).

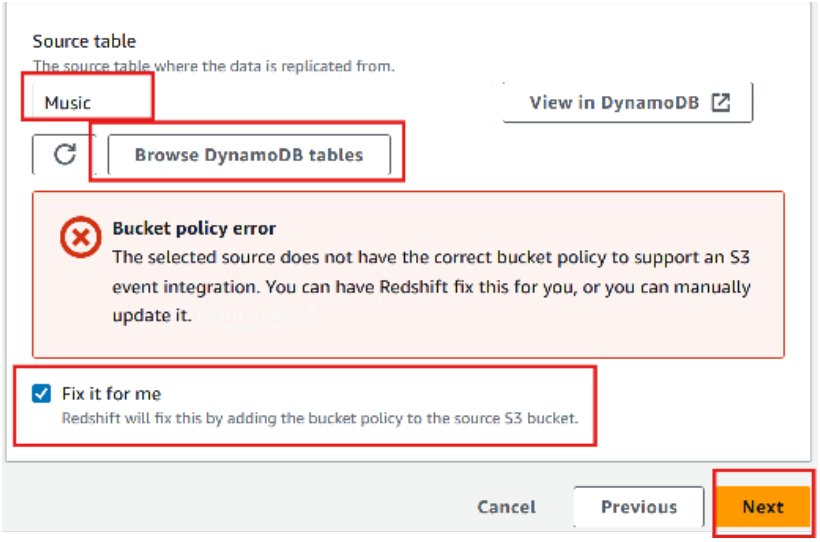


Figure 4.13 – Choose a DynamoDB table

4. A screen pops up asking you to review the changes that the setup wizard will make. Select **Continue**.

5. On the **Target** page, choose the target Amazon Redshift data warehouse you want to replicate the data to. If a **Fix it for me** option shows up, select it and click **Next**:

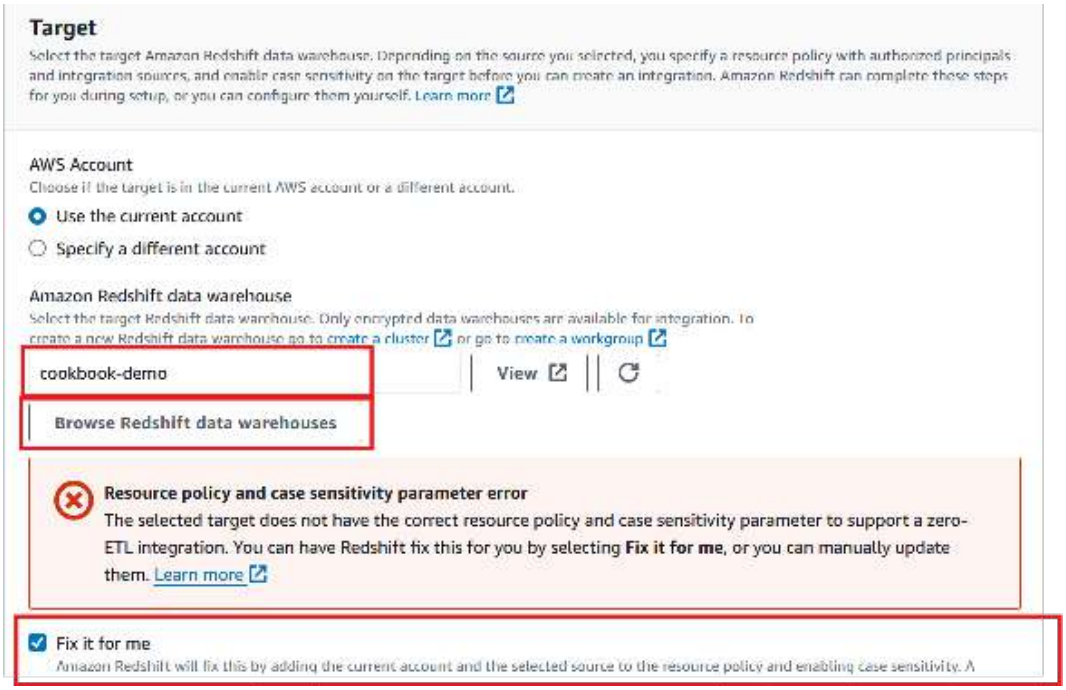


Figure 4.14 – Select the target Amazon Redshift data warehouse

6. If you chose **Fix it for me** option in the previous step, a **Review Changes** page will pop up. Review it and then click **Reboot and continue**.
7. On the **Add tags and Encryption** page, you can optionally add tags and configure encryption. Then, click **Next**.
8. On the **Review and Create** page, wait until the banner changes from **The target data warehouse update is in progress. If you exit this flow, unsaved changes will be discarded. However, the target data warehouse will still be updated** to a new banner saying **Target data warehouse successfully updated**. Then, select **Create DynamoDB integration.** Wait until the integration state is **Active**.

9. You will notice a **Create database from integration** button on the integration page. Select it:

Figure 4.15 – Create database from integration

10. On the **Create database from integration** page, for **Destination database name**, enter a descriptive name like `music` and then select **Create database**:

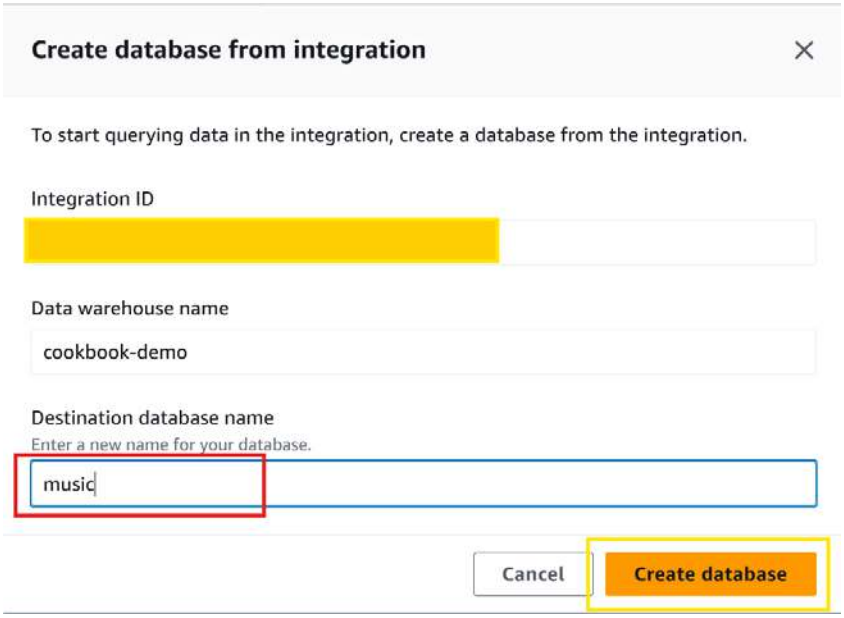


Figure 4.16 – Choose database name and create database

11. Navigate to Amazon Redshift Query Editor V2 (`https://console.aws.amazon.com/sqlworkbench/home#/client`) and connect to the target database. Under **native databases**, you will see the database you created, for example, `music`, and under it, under the `public` schema, you will see the replicated DynamoDB table, `Music` in this case. You can start analyzing that table by running `select` queries, as shown in the following screenshot:

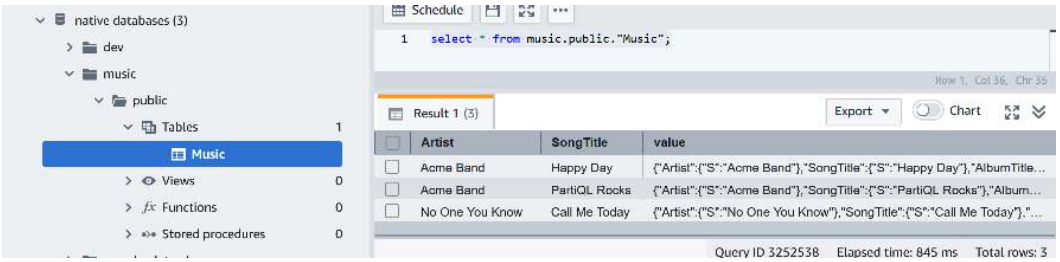


Figure 4.17 – Query the table in Amazon Redshift

12. In the `Music` table, notice that the partition key `Artist` and sort key `SongTitle` from the DynamoDB table are created as columns in the target Amazon Redshift table. They become the distribution key and sort key, respectively, on the Amazon Redshift table. The entire DynamoDB item is available in the `value` field in semi-structured format with the `SUPER` datatype. You can query it using PartiQL, as shown in the following SQL:

```
select "Artist", "SongTitle"
, value."AlbumTitle"."S" :: varchar(30) as "AlbumTitle"
, value."Awards"."N" :: int as "AlbumTitle"
from music.public."Music";
```

Figure 4.18 – Query the replicated DynamoDB data using Amazon Redshift Query Editor V2

How it works...

On activation, the integration exports the full DynamoDB table to populate the Amazon Redshift database. The time it takes for this initial process to complete depends on the DynamoDB table size. The zero-ETL integration then incrementally replicates updates from DynamoDB to Amazon Redshift every 15 minutes using DynamoDB incremental exports. This means the replicated DynamoDB data in Amazon Redshift is kept up to date automatically.

Once configured, users can analyze the DynamoDB data in Amazon Redshift using standard SQL clients and tools, without impacting DynamoDB table performance. You can increase the refresh interval and set it to 900–432,000 seconds (15 minutes–5 days) using the `CREATE DATABASE/ALTER DATABASE` command. Take the following example:

```
ALTER DATABASE sample_integration_db INTEGRATION SET REFRESH_INTERVAL
1500;
```

Ingesting data from SaaS applications like Salesforce using zero-ETL integration

Using zero-ETL integrations for Amazon Redshift, you can seamlessly ingest data from **software-as-a-service (SaaS)** applications, such as Facebook ads, Instagram ads, Salesforce, Salesforce Marketing Cloud Account Engagement, SAP OData, ServiceNow, Zendesk, and Zoho, into AWS analytics services such as Amazon Redshift and Amazon S3. This native integration eliminates the need for complex ETL pipelines, additional AWS services, or third-party tools for data replication. The zero-ETL integration process begins with an initial bulk load of your SaaS data into Redshift, followed by continuous synchronization of any changes. You can create one zero-ETL integration per SaaS application, and while the minimum latency is 1 hour, you have the option to configure it to sync data at specific intervals based on your requirements. By leveraging this solution, you can get fresher SaaS data for analytics, AI/ML, and reporting, leading to more accurate and timely insights for use cases like business dashboards, customer behavior analysis, and data quality monitoring.

In this recipe, we will create a zero-ETL integration between the SaaS application Salesforce to Amazon Redshift.

Getting ready

To complete this recipe, you will need:

- An Amazon Redshift data warehouse deployed in the AWS Region eu-west-1.
- Amazon Redshift data warehouse admin user credentials.
- Access to any SQL interface, such as a SQL client or Amazon Redshift Query Editor.
- Navigate to **Secret Manager** (`https://console.aws.amazon.com/secretsmanager/listsecrets`) and create a secret named `salesforce-secret`:
    - Select **Store a New Secret**.
    - For **Secret Type**, choose **Other type of secret**, and in the **Key/value pairs** section, for **Key**, enter `Consumer Secret`, and for **Value**, enter `USER_MANAGED_CLIENT_APPLICATION_CLIENT_SECRET`. Then, click **Next**:

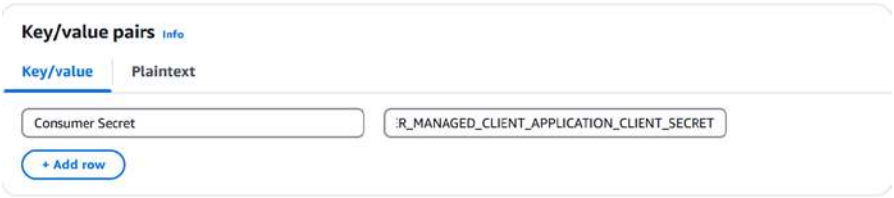


Figure 4.19 – Create a Salesforce secret

    - For the secret name, enter `salesforce-secret`. Click **Next** and then **Next** again on the next page, and then select **Store**.
- Create an IAM role named `sales-force-glue-role`:
    - Navigate to IAM (`https://console.aws.amazon.com/iam/home`).
    - In the left navigation pane, choose **Roles**, and on the page that opens up, select **Create Role**.
    - On the **Select Trusted Entity** page, in the **Trusted entity type** section, choose **AWS Service**. In the **Use case** section, for **Service or use case**, choose **Glue**.
    - Click **Next** and **Next** again. On the **Name, review and create page**, for **Role name**, enter `salesforce-glue-role`, and then select **Create Role**.

- On the roles page, for `salesforce-glue-role`, choose **Add permissions**, and within that, choose **Create inline policy**. For **Policy Editor**, choose **JSON** and paste the following JSON:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "secretsmanager:DescribeSecret",
                "secretsmanager:GetSecretValue",
                "secretsmanager:PutSecretValue",
                "ec2:CreateNetworkInterface",
                "ec2:DescribeNetworkInterface",
                "ec2:DeleteNetworkInterface"
            ],
            "Resource": "*"
        }
    ]
}
```

- Name the policy `secretmanager-ec2` and click **Save**.

- An IAM role attached to an Amazon Redshift data warehouse that can access Amazon S3; we will reference it in the recipe as `[Your-Redshift_Role]`.
- An Amazon S3 bucket created in eu-west-1; we will reference it as `[Your-Amazon_S3_Bucket]`.

- Create a free Salesforce developer account (`https://developer.salesforce.com/form/signup/freetrial.jsp`). Note down the sign-in information and the easy login URL from the confirmation email you received. We will refer to it as `[Your-salesforce_login_URL]`:

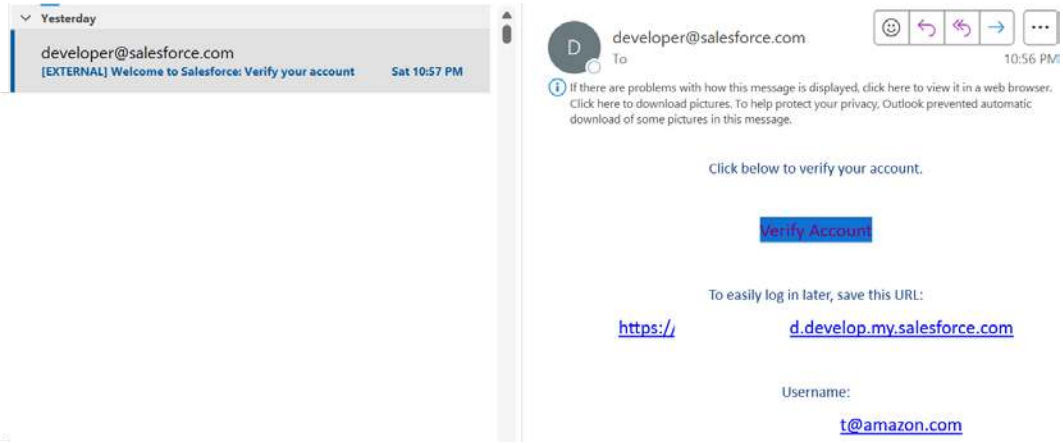


Figure 4.20 – Confirmation email after signing up for a free trial

How to do it...

Create a connection in AWS Glue for Salesforce:

1. Navigate to the AWS Glue console (`https://console.aws.amazon.com/glue/home`).
2. In the left navigation pane, choose **Data connections**, and in the **Connections** section, select **Create connection**:

Figure 4.21 – Create a new AWS Glue connection

3. On the **Choose data source** page, search for `salesforce`. Choose **Salesforce - new** and then click **Next**:

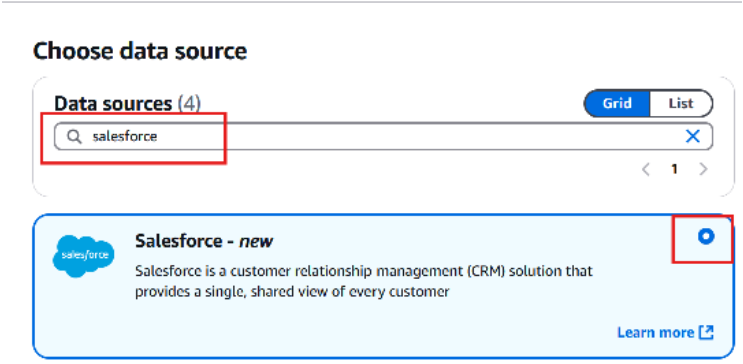


Figure 4.22 – Choose Salesforce as the source for connection

4. On the **Connection details** page, for **Instance URL**, enter `[your-salesforce_login_URL]`. For **Salesforce environment**, choose **Production**. For **IAM service role**, select **salesforce-glue-role**:

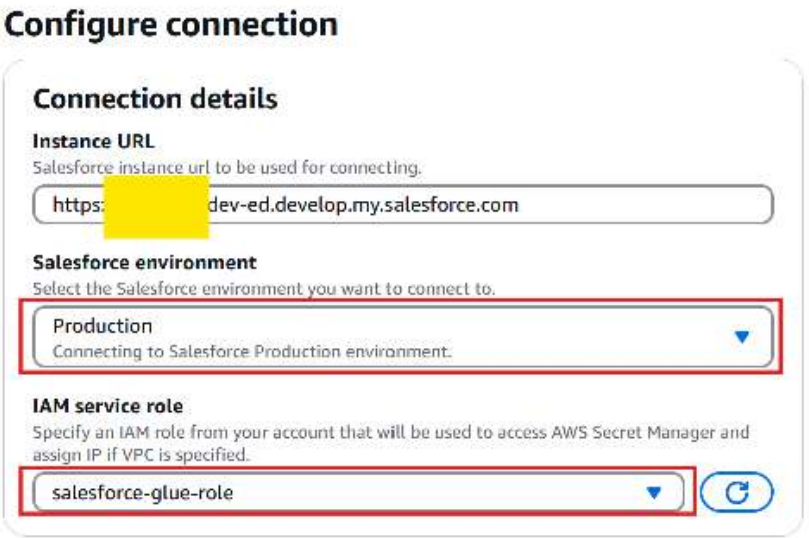


Figure 4.23 – Provide Salesforce connection details – part 1

5. In the **Authentication** section, for **OAuth grant type**, choose **Authorization Code**. For **AWS Secret**, choose **salesforce-secret**. Scroll down and choose **Test connection**:

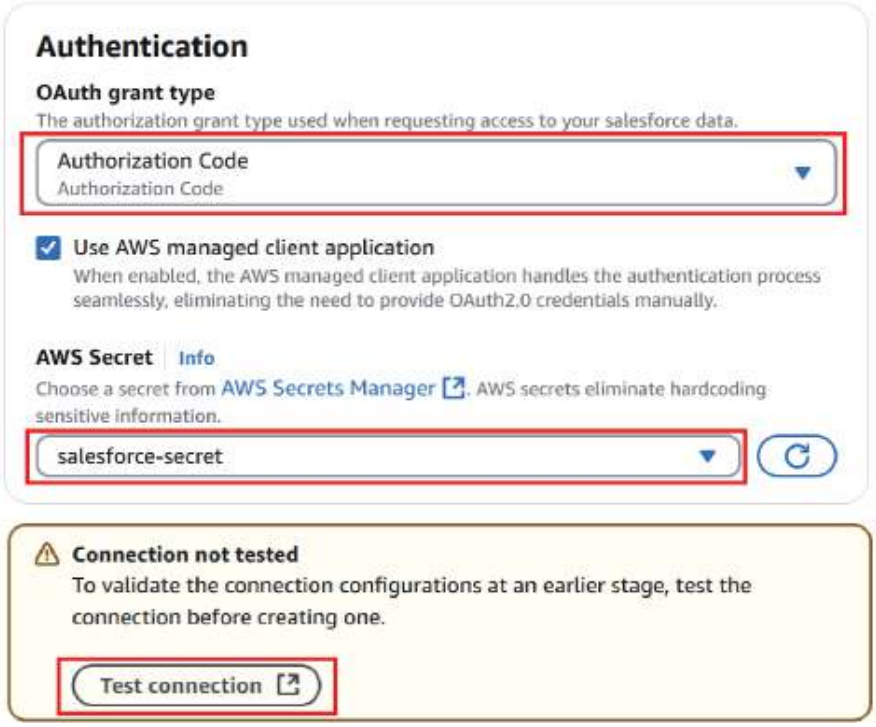


Figure 4.24 – Provide Salesforce connection details – part 2

6. You will be redirected to a Salesforce page to enter your login credentials and allow access. Once this is successful, you will see a **Connection test successful** message:

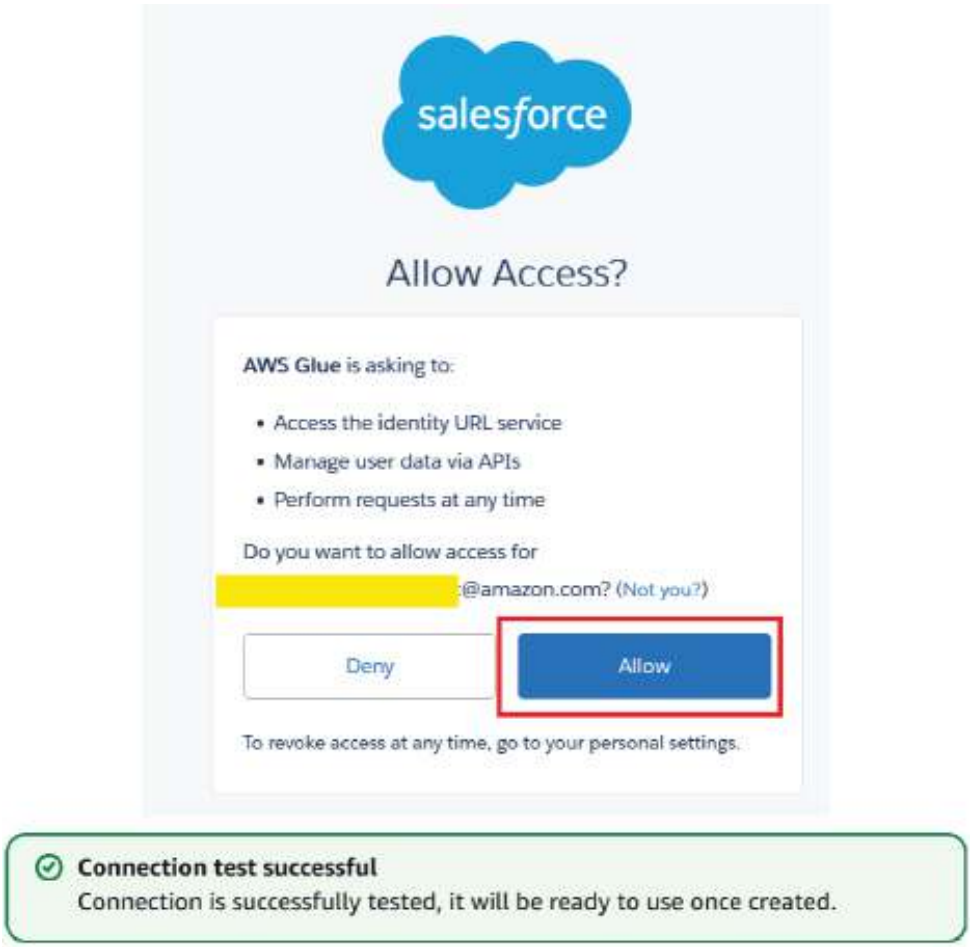


Figure 4.25 – Validate Salesforce connection

7. Once the connection is validated, click **Next**. In the **Connection Properties** section, for the name, choose `salesforce-connection` and then click **Next**. On the **Review and Create** page, review the details and select **Create Connection**.

Create a zero-ETL integration in the AWS Glue console between Salesforce and Amazon Redshift:

1. Navigate to the AWS Glue console (`https://console.aws.amazon.com/glue/home`) and select **Zero-ETL integrations** from the left navigation pane. Then, select **Create zero-ETL integration**:

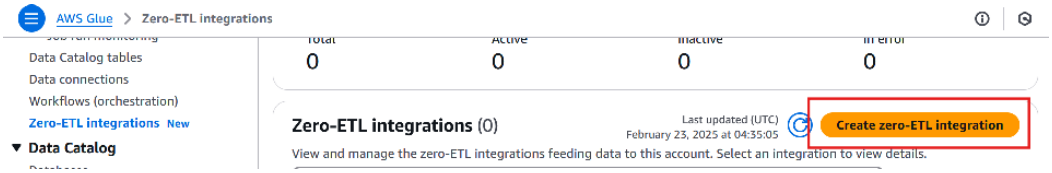


Figure 4.26 – Create Zero-ETL integration

2. On the **Select Source** page, you will see various SaaS applications, like Facebook Ads, Instagram Ads, Salesforce, Salesforce Marketing Cloud Account Engagement, SAP OData, ServiceNow, Zendesk, and Zoho. You can create zero-ETL integrations from any of the SaaS applications into Amazon Redshift. For this recipe, choose **Salesforce** and then click **Next**.
3. On the **Configure source and target** page, for **Salesforce connection**, choose **salesforce-connection**. For **Source IAM role**, choose **salesforce-glue-role**:

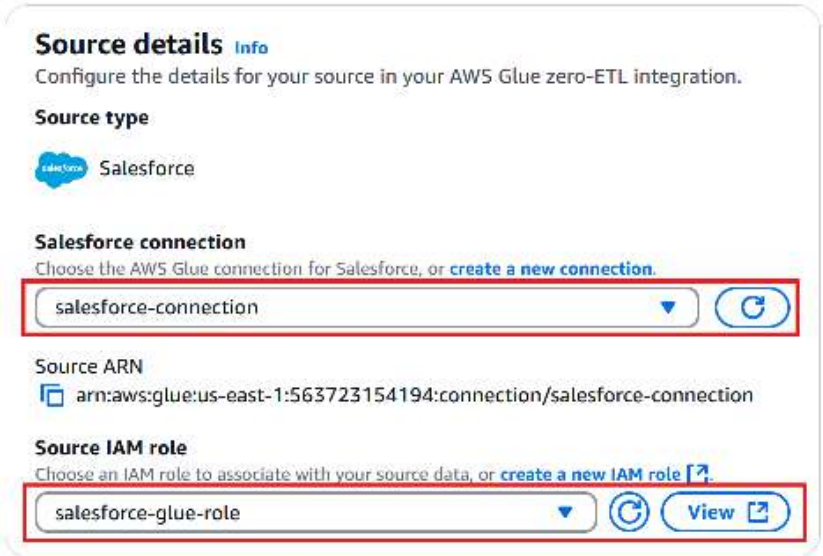


Figure 4.27 – Provide source details for the zero-ETL integration

4. In the **Select Source Data** section, you will see the Salesforce tables available. Choose the **Account**, **Opportunity**, and **Contact** tables for this recipe.

5. In the **Target details** section, choose **Use the current account** for **AWS account**. For **Data warehouse or catalog**, choose your target Amazon Redshift data warehouse. If your target data warehouse doesn't have the parameters required for this zero-ETL integration, a **Fix it for me** option will appear. Select it, review the tables in **Output settings**, and click **Next**:

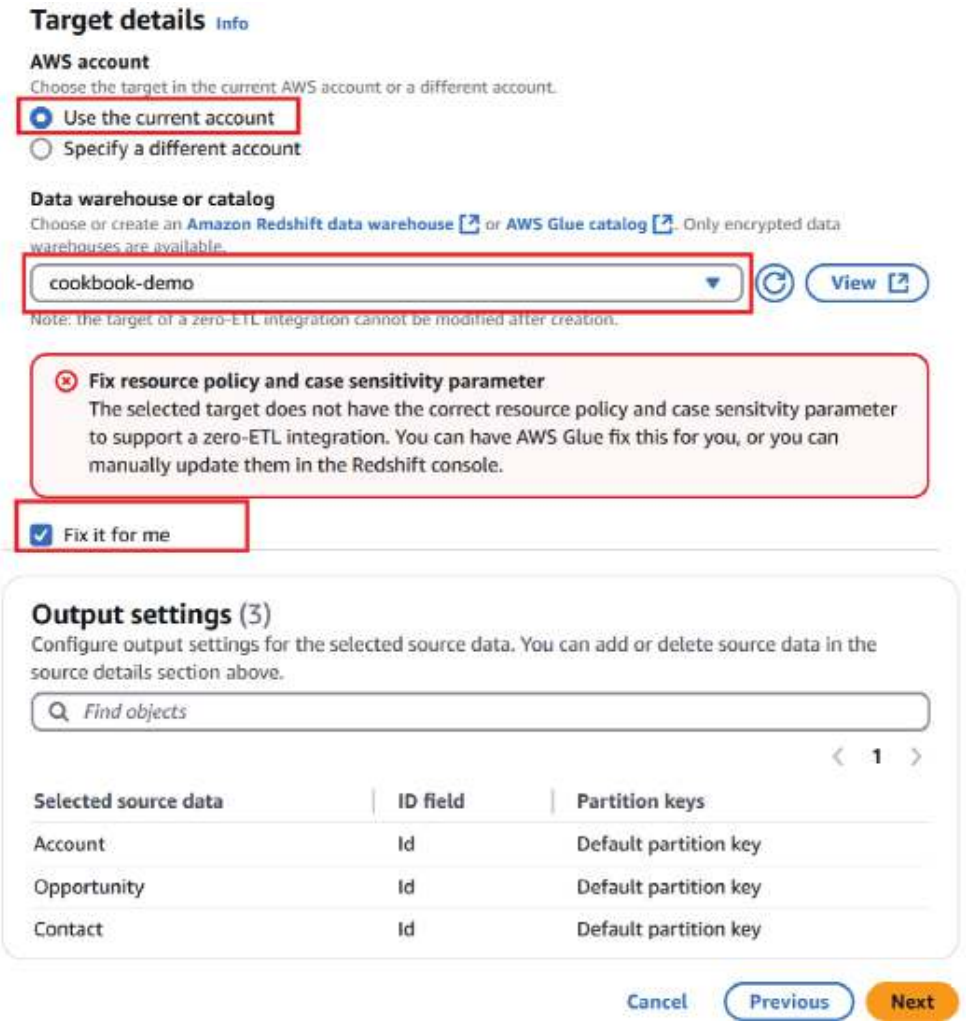


Figure 4.28 – Provide target details for the zero-ETL integration

6. If you chose **Fix it for me** in the previous step, a **Review changes** popup will appear. Review the changes that the setup wizard will apply on your target Amazon Redshift data warehouse and then select **Continue**.
7. On the **Configure Integration** page, notice that the refresh interval is 60 minutes. In the **Integration details** section, for **Name**, enter `salesforce-redshift-zetl-integ`. Click **Next**.
8. On the **Review and Create** page, review the integration details and click **Create and launch integration**. Wait until the integration status changes to **Active**. It takes around 10–15 minutes for the integration to be active.
9. Once the integration is active (refresh the page to get the latest status), you will notice a dialog box with the **Create database from integration** option. Select it:

Figure 4.29 – Create database from integration – part 1

10. On the **Create database from integration** page that opens, for **Destination database name**, enter `salesforce` and select **Create database**:

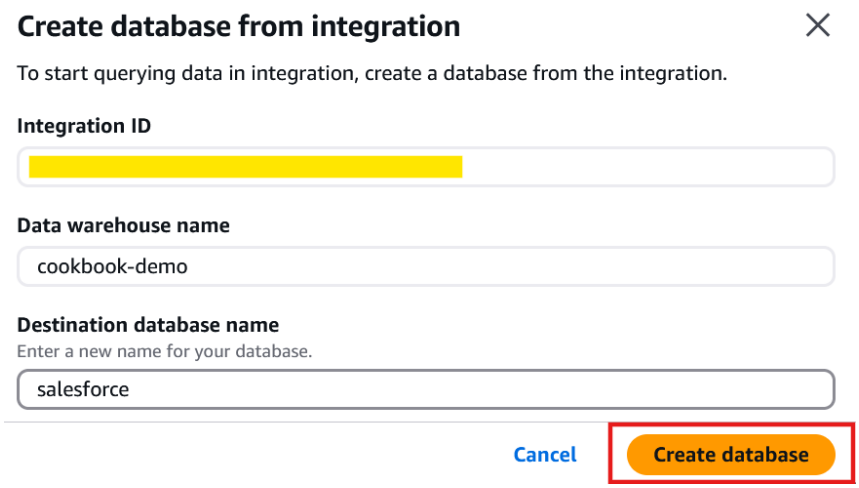


Figure 4.30 – Create database from integration – Part 2

11. Navigate to Amazon Redshift Query Editor V2 (`https://console.aws.amazon.com/sqlworkbench/home#/client`) and connect to the target database. Under **native databases**, you will see the database you created, for example, `salesforce`, and under it, under the `public` schema, you will see the tables `Account`, `Opportunity`, and `Contact`. You can start analyzing that table by running `select` queries, as shown in the following screenshot:

```
select count(0) from salesforce.public."Account";
```

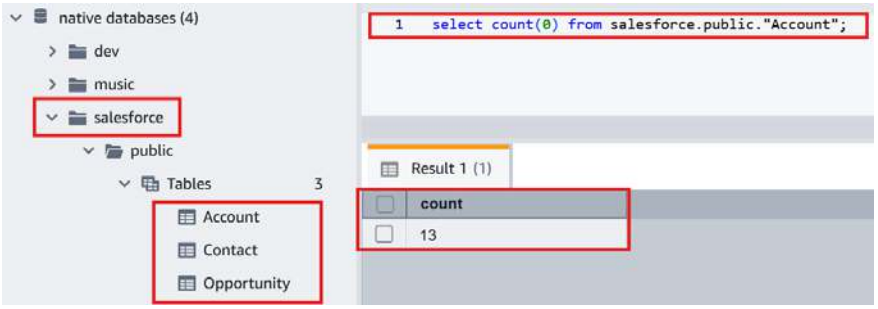


Figure 4.31 – Query Salesforce data in Amazon Redshift

12. Log in to your Salesforce developer account and select **Account** and then **+ New Account**. On the **New Account** page, for **Account Name**, enter `Test Account` and click **Save**:

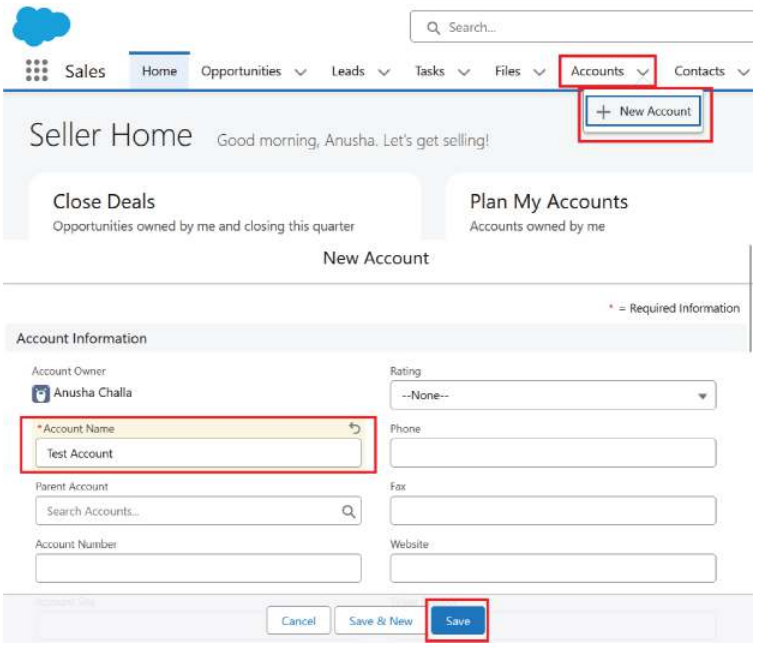


Figure 4.32 – Create a new account using a Salesforce developer account

13. Go back to Amazon Redshift Query Editor V2 after 60 minutes and query the `Account` table again. You will notice that the count of accounts has increased by 1 to 14. As of the time of writing, the minimum latency between applications like Salesforce and Amazon Redshift is 60 minutes:

```
select count(0) from salesforce.public."Account";
```

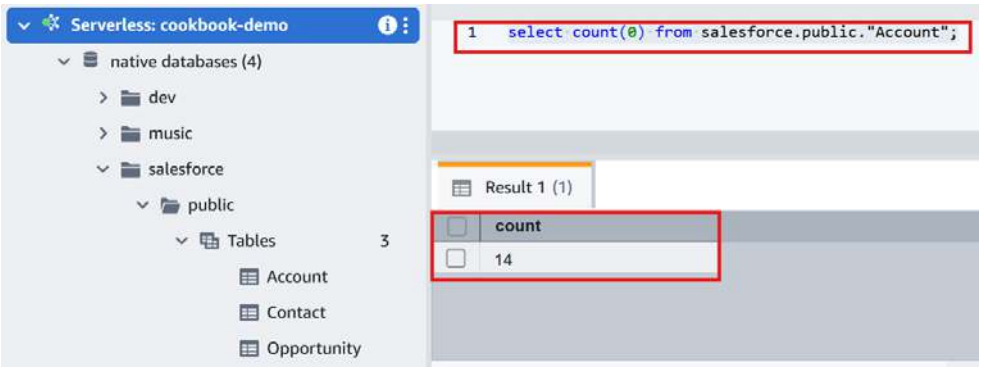


Figure 4.33 – Query Amazon Redshift for updated account count

14. Query Amazon Redshift for the account name `Test Account` using the following SQL statement:

```
select * from salesforce.public."Account"
where "Name" = 'Test Account';
```

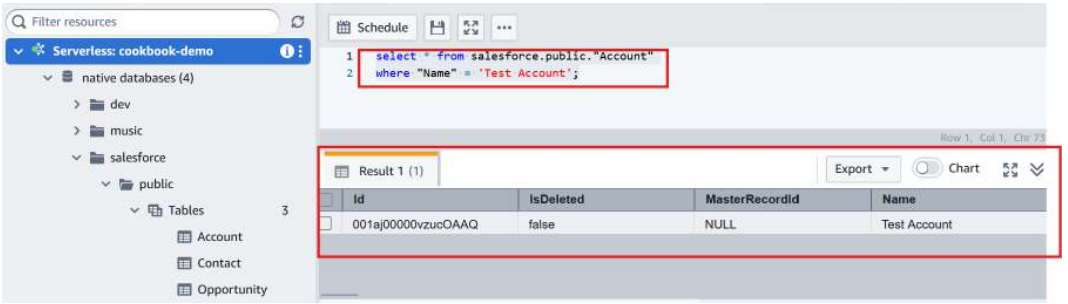


Figure 4.34 – Query Amazon Redshift for updated account details

Ingesting streaming data from Amazon Kinesis Data Streams (KDS)

Streaming data is a continuous flow of data originating from sources like IoT devices, log files, websites, and more. Ingesting this streaming data into Amazon Redshift allows running near-real-time analytics by combining it with historical and operational data to produce actionable insights. For example, analyzing sensor data streams from manufacturing equipment can help predict failures and enable preventive maintenance.

In this recipe, we will simulate a product review data stream to be ingested into an Amazon Redshift data warehouse (serverless or provisioned cluster) materialized view using Amazon KDS. KDS provides a seamless integration to capture, process, and load streaming datasets directly into Redshift materialized views configured for streaming ingestion. As the stream data arrives, it can be parsed and mapped to the materialized view's schema using SQL functions like `JSON_PARSE`. The materialized view then provides low-latency, continuous access to the latest streaming data for running analytics queries on the ingested data in near-real time.

Getting ready

To complete this recipe, you will need:

- An Amazon Redshift data warehouse deployed in the AWS Region eu-west-1.
- Amazon Redshift data warehouse admin user credentials.
- Access to any SQL interface, such as a SQL client or Amazon Redshift Query Editor.
- Access to Kinesis Data Generator. This is a UI that helps to send test data to Amazon Kinesis. Use this blog post to configure the open source Kinesis Data Generator (`https://aws.amazon.com/blogs/big-data/test-your-streaming-data-solution-with-the-new-amazon-kinesis-data-generator/`).
- The ARN of the IAM role attached to your Amazon Redshift data warehouse. We will refer to it as `[Your-Redshift-Role]`.

How to do it...

This recipe will illustrate the loading of product reviews data that is being streamed using KDS into Amazon Redshift:

1. Navigate to the AWS Management Console, search for Kinesis, and navigate to the Amazon Kinesis service home page. In the left navigation menu, choose **Data streams**, and then select **Create data stream**, as shown in the following screenshot:

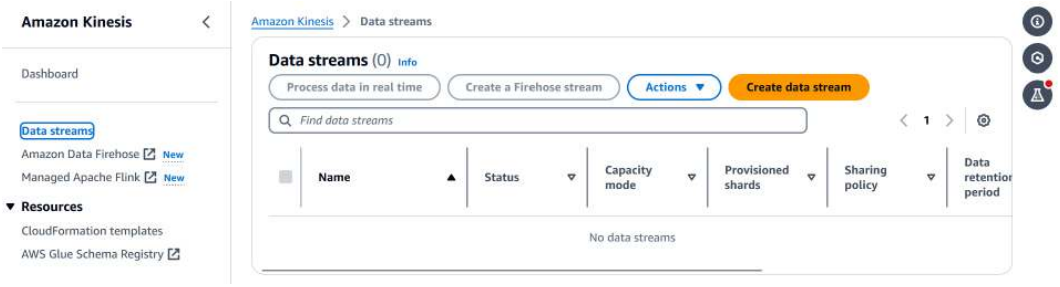


Figure 4.35 – Creating an Amazon KDS data stream

2. Give a name to the data stream, such as **product_reviews_stream**. For **Capacity mode**, choose **On-demand**. Leave the other options as the defaults and then select **Create data stream** to initiate creation. Wait until the status of the stream is **Active**, as shown in the following screenshot, before you go to the next step:

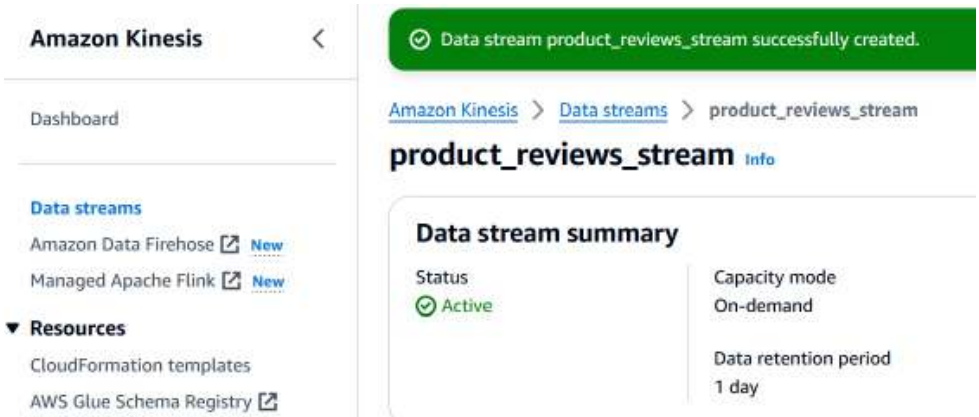


Figure 4.36 – AKS data stream is active

3. Now, let's use Amazon Kinesis Data Generator (`https://github.com/awslabs/amazon-kinesis-data-generator`) to produce streaming data and send it to the **product-reviews-stream** data stream as follows:

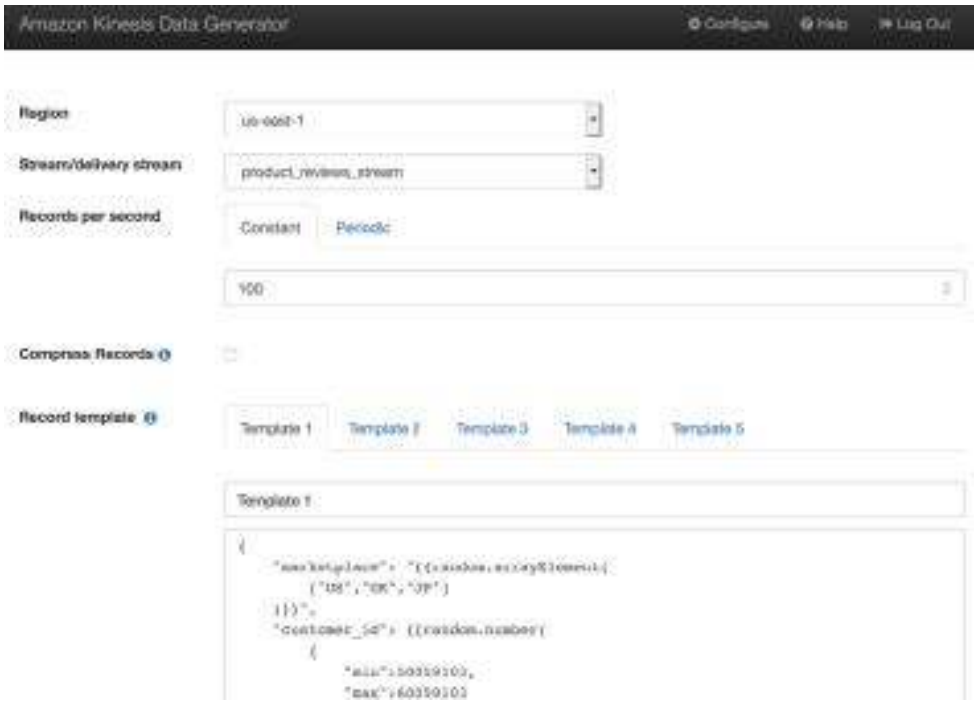


Figure 4.37 – Amazon Kinesis Data Generator

4. Here, you will use the stream/delivery stream as `product_review_stream` to send the streaming data and copy-paste the template from `https://github.com/PacktPublishing/Amazon-Redshift-Cookbook-2E/blob/main/Chapter04/kinesis_data_generator_template.json` to generate the product reviews data:

```
{
    "marketplace": "{{random.arrayElement(
        ["US","UK","JP"]
    )}}",
…
    "review_headline": "{{commerce.productAdjective}}",
    "review_body": "{{commerce.productAdjective}}",
    "review_date": "{{date.now("YYYY-MM-DD")}}",
    "year":{{date.now("YYYY")}}
}
```

5. In your Amazon Redshift data warehouse, you can now create an external schema pointing to Kinesis and a materialized view to load data from the Kinesis data stream you created. Navigate to Amazon Redshift Query Editor V2, connect to your Amazon Redshift data warehouse, and create an external schema and materialized view using the following SQL statements:

```
CREATE EXTERNAL SCHEMA kds
FROM KINESIS
IAM_ROLE '[Your-Redshift_Role]';

CREATE MATERIALIZED VIEW mv_product_reviews_from_kds
AUTO REFRESH YES AS
    SELECT approximate_arrival_timestamp,
    partition_key,
    shard_id,
    sequence_number,
    json_parse(kinesis_data) as payload
    FROM kds."product_reviews_stream"
```

6. The `payload` field in the materialized view contains the raw data. It has the `SUPER` datatype as it is in JSON format. You can use PartiQL to un-nest the JSON. In Query Editor V2, run the following SQL statement to view the data. The first statement enables a case-sensitive identifier as the semi-structured attribute names can be case-sensitive:

```
SET enable_case_sensitive_identifier TO true;
select
    payload."marketplace" :: varchar,
    payload."customer_id" :: varchar,
    payload."review_id" :: varchar,
    payload."product_id" :: varchar
from
    mv_product_reviews_from_kds;
```

```
26  select
27      payload."marketplace" :: varchar,
28      payload."customer_id" :: varchar,
29      payload."review_id" :: varchar,
30      payload."product_id" :: varchar
31  from
32      mv_product_reviews_from_kds;
```

Result 1 (100)

marketplace	customer_id	review_id	product_id
US	53312040	AKFi3RNymoilLa6	B0006H0PRK
UK	57042219	lE6sEDBVLA9HjgB	B0006H0PRK
UK	55860321	FillnHFfEL7ffVX	B0001FAX2E
US	54059227	DaHEcNe0cmljPHy	B000IZT7DI

Figure 4.38 – Query streaming materialized view

Ingesting streaming data from Amazon Managed Streaming for Apache Kafka (MSK)

In addition to KDS, Amazon Redshift also supports streaming ingestion directly from Amazon MSK, which is a fully managed Apache Kafka service. This allows ingesting streaming data from sources like database log files, IoT sensor data, clickstreams, and more that are already producing data to Kafka topics.

In this recipe, we will see how data from Amazon MSK can be loaded in near-real time to an Amazon Redshift data warehouse (serverless or provisioned cluster). The process works like Kinesis—an Amazon Redshift materialized view is configured to consume data from an Amazon MSK Kafka topic. As new records are published to the topic, they flow directly into the materialized view without intermediate storage. SQL functions like `JSON_PARSE` can parse and map the streaming data to the view's schema on arrival.

The materialized view provides low-latency querying access to this freshly ingested streaming data combined with existing data in Redshift tables. This enables running real-time analytics queries and dashboards on the continuous data stream. Streaming ingestion from MSK can allow the ingestion of hundreds of megabytes per second per refresh into the materialized view, which can optionally be set to auto-refresh for continuous ingestion.

Getting ready

To complete this recipe, you will need:

- An Amazon Redshift data warehouse deployed in the AWS Region eu-west-1.
- Amazon Redshift data warehouse admin user credentials.
- Access to any SQL interface, such as a SQL client or Amazon Redshift Query Editor.
- The ARN of the IAM role attached to your Amazon Redshift data warehouse. We will refer to it as `[Your-Redshift-Role]`.
- An MSK cluster. For instructions on how to create one, please refer to `https://docs.aws.amazon.com/msk/latest/developerguide/create-cluster.html`.
- Create a topic in your MSK cluster where your data producer can publish data (`https://docs.aws.amazon.com/msk/latest/developerguide/create-topic.html`).
- A data producer to write data to the topic in your MSK cluster. Refer to `msk-data-generator` to understand how to send sample data to an MSK cluster (`https://github.com/awslabs/amazon-msk-data-generator/tree/main`).

How to do it...

This recipe will illustrate the loading of product reviews data that is being streamed using Amazon MSK into Amazon Redshift:

1. In your Amazon Redshift data warehouse, create an external schema pointing to Amazon MSK and a materialized view to load data from the Kafka topic you created. Navigate to Amazon Redshift Query Editor V2, connect to your Amazon Redshift data warehouse, and create an external schema and materialized view using the following SQL statements:

```
CREATE EXTERNAL SCHEMA msk_schema
FROM KAFKA
IAM_ROLE  'iam-role-arn'
AUTHENTICATION { none | iam }
CLUSTER_ARN 'msk-cluster-arn';

CREATE MATERIALIZED VIEW mv_orders_stream
AUTO REFRESH YES AS
SELECT kafka_partition,
 kafka_offset,
 refresh_time,
```

```
 JSON_PARSE(kafka_value) as Data
FROM msk_schema."[Your-Topic-Name]"
WHERE CAN_JSON_PARSE(kafka_value);
```

Amazon Redshift can automatically refresh the streaming materialized view with new data. Auto-refresh needs to be turned on explicitly for a materialized view. To do this, specify `AUTO REFRESH` in the materialized view definition.

2. The column `kafka_value` has the raw data and its datatype is `SUPER`. You can use PartiQL to extract the fields of interest, as shown in the following SQL:

```
SELECT
    data."OrderID"::INT4 as OrderID
    ,data."ProductID"::VARCHAR(36) as ProductID
    ,data."ProductName"::VARCHAR(36) as ProductName
    ,data."CustomerID"::VARCHAR(36) as CustomerID
    ,data."CustomerName"::VARCHAR(36) as CustomerName
    ,data."Store_Name"::VARCHAR(36) as Store_Name
    ,data."OrderDate"::TIMESTAMPTZ as OrderDate
    ,data."Quantity"::INT4 as Quantity
    ,data."Price"::DOUBLE PRECISION as Price
    ,data."OrderStatus"::VARCHAR(36) as OrderStatus
    ,"kafka_partition"::BIGINT
    ,"kafka_offset"::BIGINT
FROM mv_orders_stream;
```

How it works...

Streaming ingestion allows low-latency, high-speed data ingestion directly from Amazon KDS or Amazon MSK into an Amazon Redshift materialized view, without using temporary storage. Amazon Redshift also supports streaming ingestion from the Confluent-managed cloud and self-managed Apache Kafka clusters on Amazon EC2 instances, expanding its capabilities beyond Amazon KDS and Amazon MSK. Amazon Redshift supports **mutual Transport Layer Security (mTLS)** as the authentication protocol for secure communication between Amazon Redshift and Kafka. The Redshift data warehouse or serverless workgroup acts as the consumer of the data stream. As data arrives, it can be parsed using SQL functions to map it to the materialized view's columns. When refreshed, Redshift ingests data from the allocated Kinesis shards or Kafka partitions until the view is fully updated with the latest stream data. Ingestion can process hundreds of megabytes per second per refresh.

The materialized view can use auto-refresh to automatically ingest new streaming data as it arrives by specifying the `AUTO REFRESH` option when creating or altering the view. With auto-refresh enabled, the view continuously updates without manual intervention, allowing low-latency access to the latest streaming data as part of the normal Redshift workload.

Figure 4.39 – How streaming ingestion works

You can remove records from a materialized view that's used for streaming ingestion, using `TRUNCATE` or `DELETE`:

```
truncate my_streaming_materialized_view;
delete from my_streaming_materialized_view;
```

Near-real-time ingestion of data from Amazon S3 using auto-copy

Amazon Redshift supports auto-copy support to simplify data loading from Amazon S3 into Amazon Redshift. You can now set up continuous file ingestion rules to track your Amazon S3 paths and automatically load new files without the need for additional tools or custom solutions. This also enables end users to have the latest data available in Amazon Redshift shortly after the source data is available.

In this recipe, you will learn how to build automatic file ingestion pipelines in Amazon Redshift when source files are located on Amazon S3 by using a simple SQL command. You will enable auto-copy using auto-copy jobs, learn how to monitor jobs, and review considerations and best practices for this feature.

Getting ready

To complete this recipe, you will need:

- An Amazon Redshift data warehouse deployed in the AWS Region eu-west-1.
- Amazon Redshift data warehouse admin user credentials.
- Access to any SQL interface, such as a SQL client or Amazon Redshift Query Editor.
- An Amazon S3 bucket. We will refer to it as `[Your-S3-Bucket-Name]`.
- Add the following to the Amazon S3 bucket policy:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Auto-Copy-Policy-01",
            "Effect": "Allow",
            "Principal": {
                "Service":"redshift.amazonaws.com"
            },
            "Action": [
                "s3:GetBucketNotification",
                "s3:PutBucketNotification",
                "s3:GetBucketLocation"
            ],
            "Resource": "arn:aws:s3:::<<your-s3-bucket-name>>",
            "Condition": {
                "StringLike": {
                    "aws:SourceArn": "arn:aws:redshift:<region-
name>:<aws-account-id>:integration:*",
                    "aws:SourceAccount": "<aws-account-id>"
                }
            }
        }
    ]
}
```

How to do it...

Set up Amazon S3 event integration:

1. Navigate to the Amazon Redshift console (`https://console.aws.amazon.com/redshift/`). Under the **Integrations** section, choose **S3 event integrations** and then select **Create S3 event integration**:

Figure 4.40 – Create S3 event integration

2. For **Integration name**, enter `orders-s3-redshift-integration` and then click **Next**.
3. On the **Select source** page, select **Browse S3 buckets** and then choose the S3 bucket you want to use for this recipe. Click **Next**. Note that when browsing S3 buckets, only buckets that are in the same AWS account and same AWS Region are shown.
4. On the **Select target** page, for **AWS Account**, choose **Use the current account**. Then, click the **Browse Redshift data warehouses** button and select your target Amazon Redshift data warehouse.
5. If your target Amazon Redshift data warehouse doesn't have the necessary parameters for auto-copy, you will see a **Fix it for me** option. Select it and click **Next**. If you chose **Fix it for me**, after clicking **Next**, you will see a popup for **Review changes** for you to review the changes that the zero-ETL integration setup wizard will apply.

Review them carefully and click **Continue**:

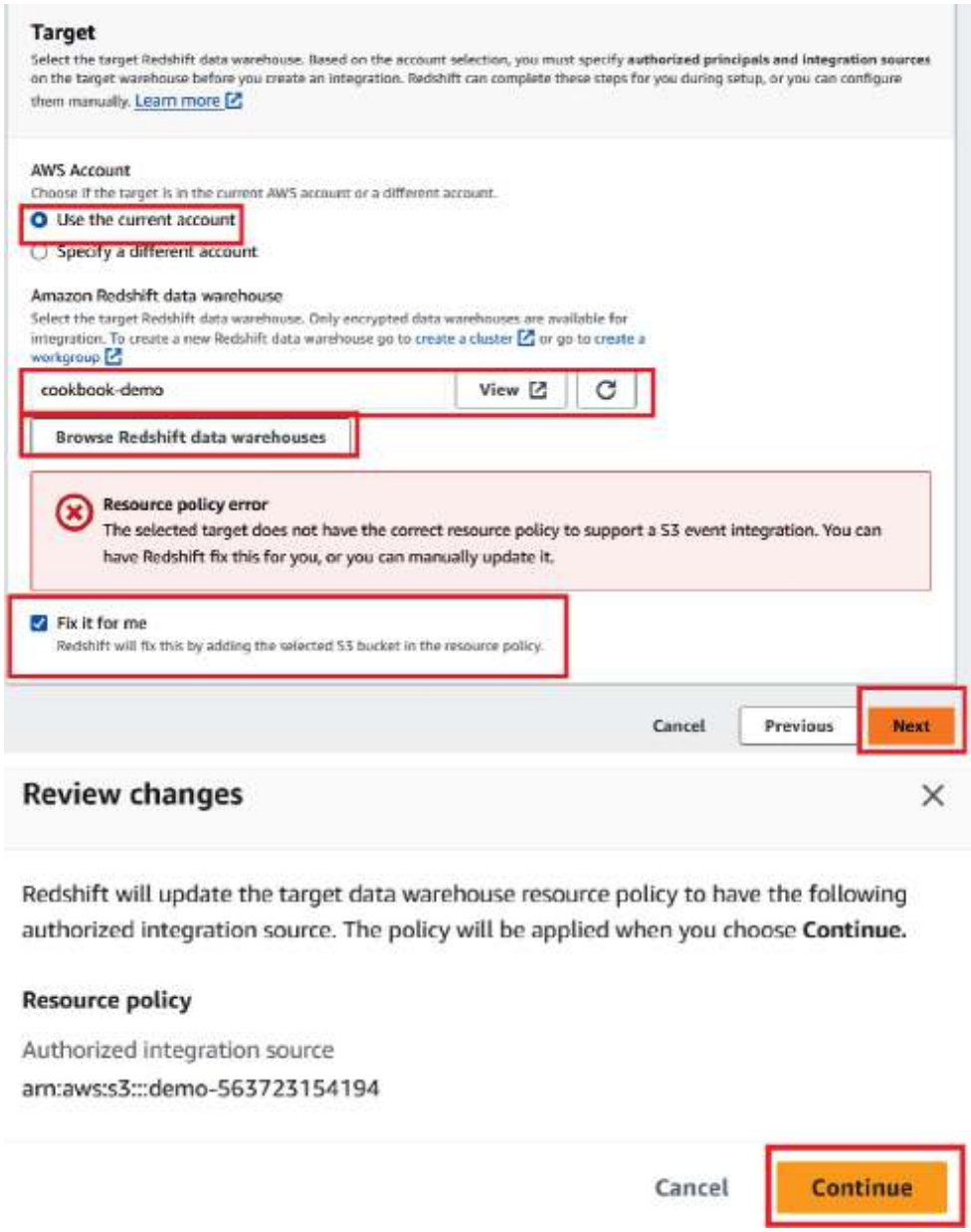


Figure 4.41 – Provide target details for S3 event integration

6. On the next page, click **Next**, and on the **Review and create** page, review the details of the integration and select **Create S3 event integration**.
7. Once the integration is created, on the integration details page, you will see an option to **Create autocopy job**. Click on it and you will be directed to Amazon Redshift Query Editor V2:

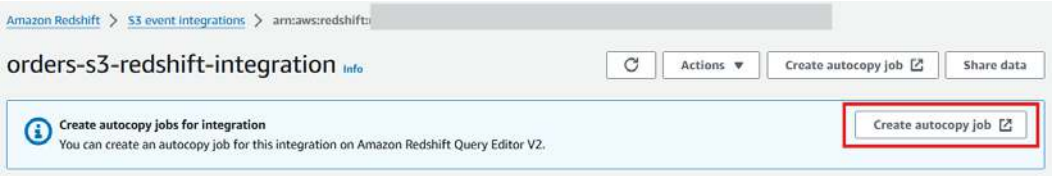


Figure 4.42 – Create autocopy job

8. In Amazon Redshift Query Editor V2, first create an `orders` table using the following DDL:

```
create table s3_orders (
  o_orderkey int8 not null,
  o_custkey int8 not null,
  o_orderstatus char(1) not null,
  o_totalprice numeric(12,2) not null,
  o_orderdate date not null,
  o_orderpriority char(15) not null,
  o_clerk char(15) not null,
  o_shippriority int4 not null,
  o_comment varchar(79) not null,
  Primary Key(O_ORDERKEY)
) distkey(o_orderkey) sortkey(o_orderdate, o_orderkey) ;
```

9. Next, create an auto-copy job that can load data from your S3 bucket into this table using the following SQL statement. Replace `[Your-S3-Bucket-Name]` and `[Your-Redshift-Role]` with your S3 bucket name and the Amazon Redshift data warehouse's IAM role:

```
COPY s3_orders
FROM 's3://[Your-S3-Bucket-Name]/orders'
IAM_ROLE [Your-Redshift-Role]
FORMAT CSV
gzip
JOB CREATE job_orders AUTO ON;
```

10. In your S3 bucket `[Your-S3-Bucket-Name]`, create a folder named `dt=2024-01-09` and upload the file (`https://github.com/PacktPublishing/Amazon-Redshift-Cookbook-2E/blob/main/Chapter04/2024-01-09-000.gz`) to that folder.
11. This file will automatically be copied into the `s3_orders` table within 30 seconds after you upload it. Verify by running a count of records on the `s3_orders` table using the following SQL:

```
Select count(0) from s3_orders;
```

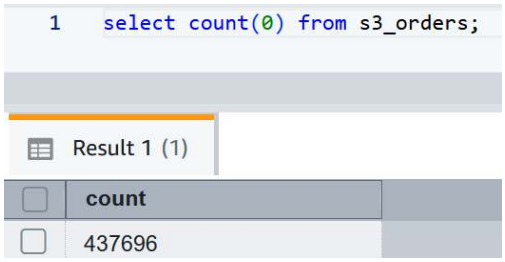


Figure 4.43 – Count from S3_orders after the first file upload

12. In your S3 bucket `[Your-S3-Bucket-Name]`, create another folder named `dt=2024-01-11` and upload the file (`https://github.com/PacktPublishing/Amazon-Redshift-Cookbook-2E/blob/main/Chapter04/2024-01-11-000.gz`) to that folder.
13. The data from this file will also automatically be copied into the `s3_orders` table within 30 seconds after you upload it. Verify by running a count of records on the `s3_orders` table using the following SQL:

```
Select count(0) from s3_orders;
```

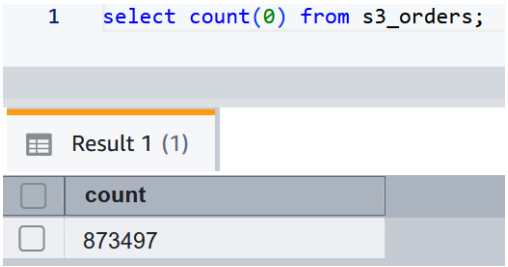


Figure 4.44 – Count from S3_orders after the second file is uploaded

How it works...

The auto-copy feature in Amazon Redshift leverages the S3 event integration to automatically load data into Amazon Redshift and simplifies automatic data loading from Amazon S3. You can enable Amazon Redshift auto-copy by creating auto-copy jobs. An auto-copy job is a database object that stores, automates, and reuses the `COPY` statement for newly created files that land in the S3 folder. The auto-copy job stores the list of files that are already loaded by the copy job into the target Amazon Redshift table. When it notices any new file with a file name different from what it already has, it automatically loads it into the table. Note that if you have updated the contents of an existing file and did not change the file name, it will not be loaded into the Amazon Redshift table.

You can create multiple auto-copy jobs from multiple Amazon S3 locations into the same Amazon Redshift table. The following diagram illustrates this process:

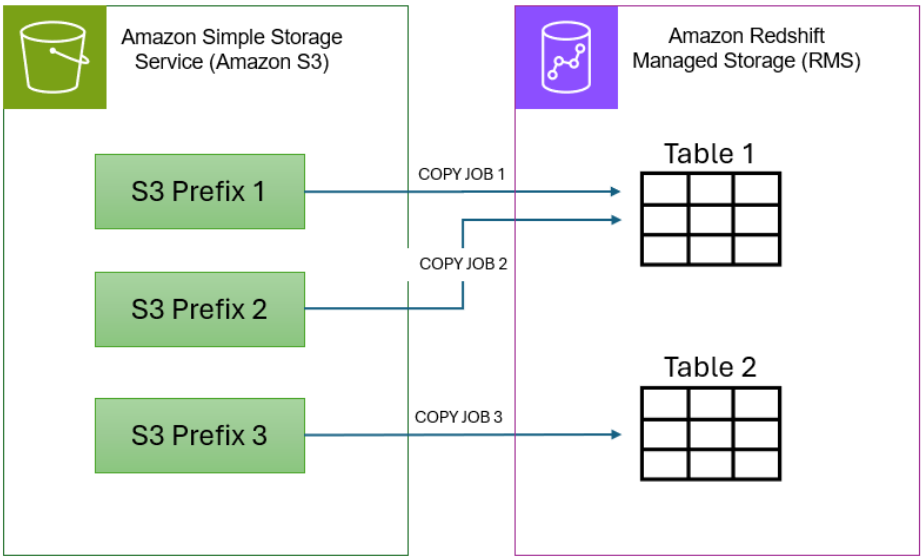


Figure 4.45 – Automatic data ingestion from Amazon S3 into Amazon Redshift

If necessary, users can manually invoke an auto-copy job as shown in the following SQL:

```
copy job RUN <auto-copy job Name>
```

You can alter existing auto-copy jobs to disable `AUTO ON` using the following command:

```
copy job ALTER <auto-copy job Name> AUTO OFF
```

5

Scalable Data Orchestration for Automation

AWS provides a rich set of native services to integrate a workflow. These workflows may involve multiple tasks that can be managed independently, thereby taking advantage of purpose-built services and decoupling them.

In this chapter, we will primarily focus on workflows such as an **Extract, Transform, and Load (ETL)** process that is used to refresh the data warehouse. We will illustrate different options that are available using the individual recipes, but they are interchangeable depending on your use case.

The following recipes are discussed in this chapter:

- Scheduling queries using Amazon Redshift Query Editor V2
- Event-driven applications using Amazon EventBridge on Amazon Redshift provisioned clusters
- Event-driven applications using AWS Lambda on Amazon Redshift provisioned clusters
- Orchestration using AWS Step Functions on provisioned clusters
- Orchestration using Amazon Managed Workflows for Apache Airflow on provisioned clusters

Technical requirements

Here are the technical requirements to complete the recipes in this chapter:

- You need access to the AWS console.
- The AWS administrator should create an IAM user by following *Recipe 1* in the *Appendix*. This IAM user will be used in some of the recipes in this chapter.
- The AWS administrator should create an IAM role by following *Recipe 3* in the *Appendix*. This IAM role will be used in some of the recipes in this chapter.
- The AWS administrator should deploy the AWS CloudFormation template (`https://github.com/PacktPublishing/Amazon-Redshift-Cookbook-2E/blob/main/Chapter05/chapter_5_CFN.yaml`) and create two IAM policies:
- An IAM policy attached to the IAM user, which will give them access to Amazon Redshift, Amazon EC2, AWS CloudFormation, Amazon S3, Amazon SNS, Amazon **Managed Workflows for Apache Airflow** (**MWAA**), Amazon EventBridge, Amazon CloudWatch, Amazon CloudWatch Logs, AWS Glue, AWS Lambda, and AWS Step Functions
- An IAM policy attached to the IAM role, which will allow Amazon Redshift data warehouse to access Amazon S3, AWS Lambda, and Amazon EventBridge
- Attach an IAM role to the Amazon Redshift data warehouse (serverless or provisioned cluster) endpoint by following *Recipe 4* in the *Appendix*. Take note of the IAM role name; we will reference it in the recipes as `[Your-Redshift_Role]`.
- You need an Amazon Redshift data warehouse deployed in the AWS Region, eu-west-1.
- You need Amazon Redshift data warehouse master user credentials.
- You need access to any SQL interface, such as a SQL client or Amazon Redshift Query Editor.
- You need an AWS account number; we will reference it in recipes as `[Your-AWS_Account_Id]`.
- You need an Amazon S3 bucket created in eu-west-1; we will refer to it as `[Your-Amazon_S3_Bucket]`.
- You need the code files referenced in the GitHub repository (`https://github.com/PacktPublishing/Amazon-Redshift-Cookbook-2E/tree/main/Chapter05`).

Scheduling queries using Amazon Redshift Query Editor V2

Amazon Redshift **Query Editor V2** (**QEV2**) allows users to schedule queries on the Redshift data warehouse. Users can schedule long-running or time-sensitive queries, refresh materialized views at regular intervals, and load or unload data.

In this recipe, we will automate the refresh of the `customer_agg_mv` materialized view, so that the data is up to date when the base tables change.

Getting ready

To complete this recipe, you will need the following:

- Amazon Redshift data warehouse (serverless or provisioned cluster) endpoint deployed in AWS Region eu-west-1
- IAM user with access to Amazon Redshift, Amazon Redshift QEV2, and Amazon EventBridge
- IAM role attached to Amazon Redshift data warehouse that can access Amazon EventBridge; we will reference it in the recipes as `[Your-Redshift_Role]`
- We will reuse the `customer_agg_mv` materialized view that was set up using the *Chapter 2* recipe titled *Managing materialized views in a database*

How to do it...

1. Open Amazon Redshift QEV2 and connect to your data warehouse where you have created `finance.customer_agg_mv`.
2. In the query editor, enter the following code:

```
REFRESH MATERIALIZED VIEW finance.customer_agg_mv
```

3. Choose **Schedule,** which opens the **Schedule query** window. In the **Schedule query** window, there are four sections: **Scheduler permissions**, **Query information**, **Scheduling options**, and **Monitoring**.

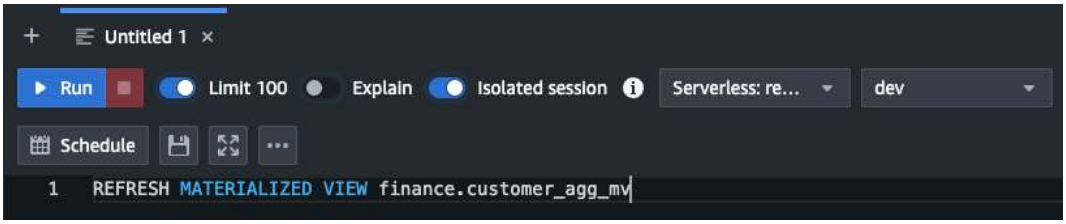


Figure 5.1: Schedule query using Amazon Redshift QEV2

4. In **Scheduler permissions**, enter the following details:

    - **IAM role**: Select the created role that has access to schedule queries – `[Your-Redshift_Role]`.
    - **Authentication**: There are two modes of authentication – **Temporary credentials** and **AWS Secret Manager**. By default, **Temporary credentials** is selected, which uses the `GetCredentials` IAM permission and the `db` user to generate the temporary credentials. You can also select **AWS Secret Manager**, where you can use secrets stored in AWS Secret Manager.
    - **Cluster or workgroup**: Select the Amazon Redshift data warehouse.

- **Database name:** Enter the database name.

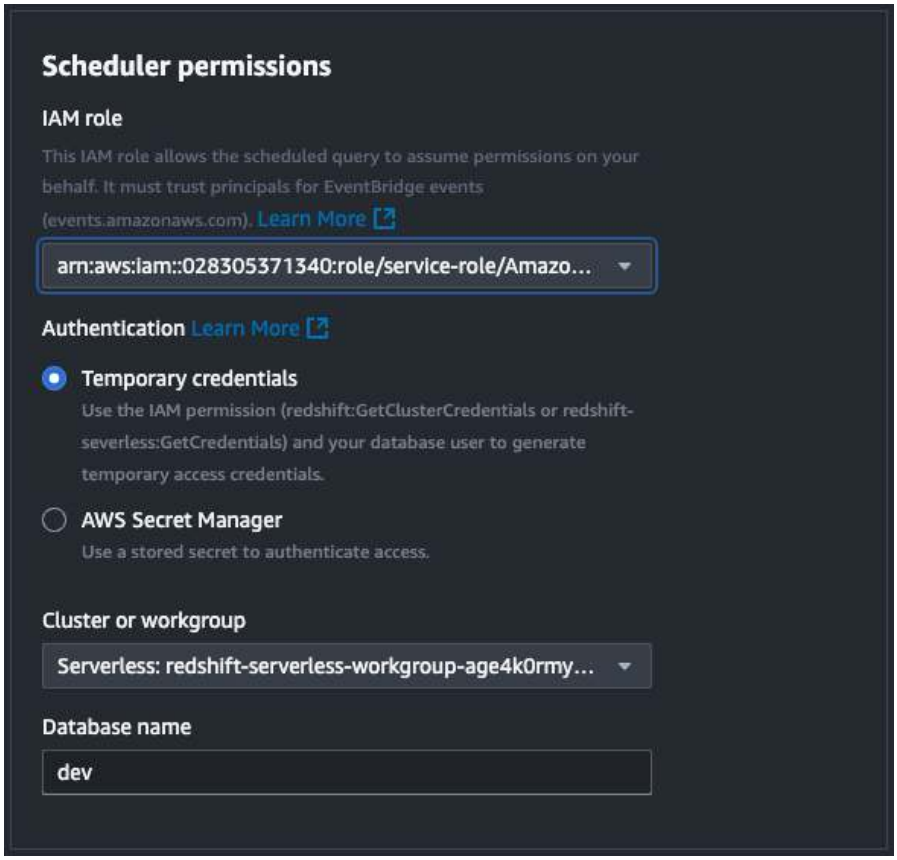


Figure 5.2: Setting up the schedule options for refresh

5. In **Query information**, enter the following details:

    - **Scheduled query name**: Enter the recognizable name of the query
    - **Query description:** You can type a query description

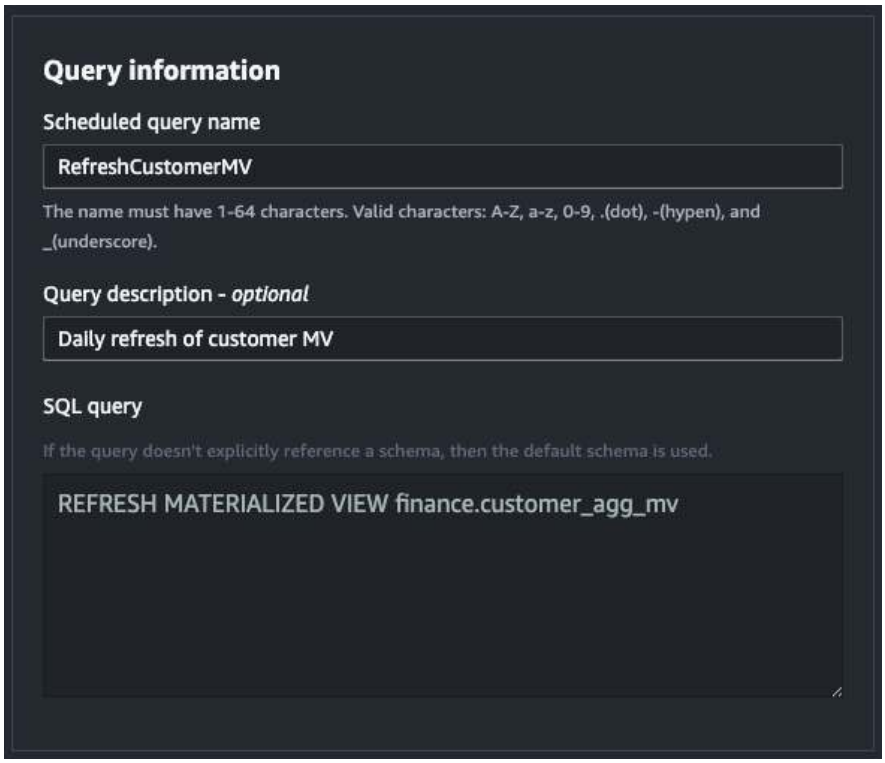


Figure 5.3: Setting up the schedule name and query

6. In **Scheduling options**, you can schedule queries by **Run frequency** or **Cron format**.

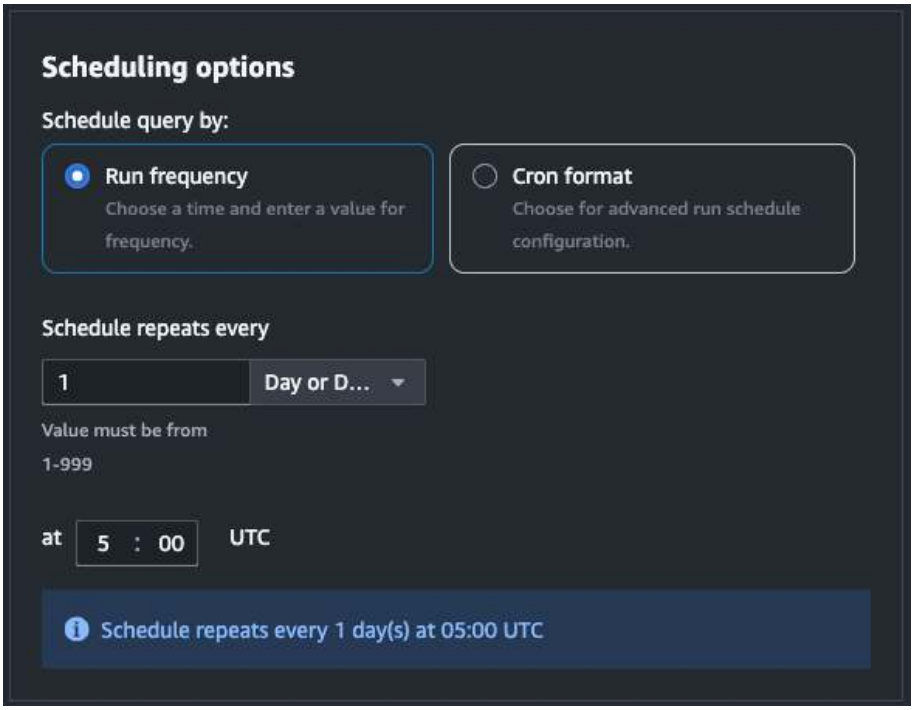


Figure 5.4: Setting up the schedule interval

7. In the **Monitoring** section, you can optionally configure SNS notifications.
8. Choose **Schedule query** to save the schedule.
9. Choose **Scheduled queries** from the left pane. This will show a list of scheduled queries and their respective state.

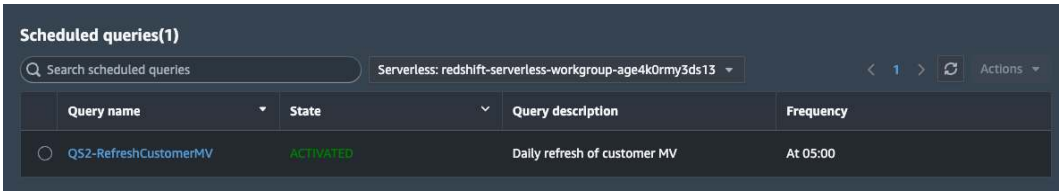


Figure 5.5: List of scheduled queries and their state

How it works...

The **Schedule** option in Amazon Redshift QEV2 is a convenient way to run a SQL statement. You can create a schedule to run your SQL statement at time intervals that match your business needs. When it's time for the scheduled query to run, Amazon EventBridge (`https://aws.amazon.com/eventbridge/`) invokes the query.

Event-driven applications using Amazon EventBridge on Amazon Redshift provisioned clusters

Event-driven data pipelines (where applications run in response to events) are increasingly used by organizations. Event-driven architectures are loosely coupled and distributed. This provides the benefit of decoupling producer and consumer processes, allowing greater flexibility in application design.

In an event-driven application, one action automatically triggers another. For example, when data arrives from a source system (the event), it automatically starts a chain of processing tasks in other connected systems. At the end of this workflow, another event gets initiated to notify end users about the completion of those transformations so that they can start analyzing the transformed dataset.

In this recipe, you will see the use of Amazon EventBridge serving as an event bus. Amazon EventBridge is a fully managed serverless event bus service that simplifies connecting with a variety of your sources. Think of EventBridge as a smart postal service for your digital world. It picks up messages and updates from everywhere – your own apps, the software services you use, and AWS tools. Like a skilled mail carrier who knows exactly where each package should go, EventBridge follows your instructions to deliver this information to the right destination at exactly the right time. This creates a smooth, automated flow of information that keeps your entire system running and responding in real time.

This recipe will use Amazon EventBridge to schedule the run of the Redshift data pipeline for the parts table. Lambda functions will use the Amazon Redshift Data API to make asynchronous calls. On the completion of the code execution, the pipeline will send an **Amazon Simple Notification Service** (**Amazon SNS**) notification.

Getting ready

To complete this recipe, you will need the following:

- An Amazon Redshift provisioned cluster deployed in the AWS Region eu-west-1. Note the data warehouse ID; we will refer to it as `[Your-Redshift_Cluster]`.
- Amazon Redshift provisioned cluster master user credentials. Note the username; we will refer to it as `[Your-Redshift_User]`.
- Access to any SQL interface, such as a SQL client or Amazon Redshift QEV2.
- An IAM user with access to Amazon SNS, Amazon EventBridge, and AWS Lambda.
- An IAM role with access to AWS Lambda; we will reference it in the recipes as `[Your-Redshift_Role]`.
- An AWS account number; we will reference it in the recipes as `[Your-AWS_Account_Id]`.

How to do it...

1. Create a product reviews table in the Amazon Redshift database using a SQL client or QEV2:

```
CREATE TABLE daily_product_reviews
(
  marketplace         VARCHAR(2),
  customer_id         VARCHAR(32),
  review_id           VARCHAR(24),
  product_id          VARCHAR(24),
  product_parent      VARCHAR(32),
  product_title       VARCHAR(512),
  star_rating         INT,
  helpful_votes       INT,
  total_votes         INT,
  vine                CHAR(1),
  verified_purchase   CHAR(1),
  review_headline     VARCHAR(256),
  review_body         VARCHAR(MAX),
  review_date         DATE,
  YEAR                INT
)
DISTSTYLE KEY DISTKEY (customer_id) SORTKEY (review_date);
```

2. Create a materialized view (`daily_product_review_fact_mv`) using the results of the query based on `daily_product_reviews`:

```
CREATE MATERIALIZED VIEW public.daily_product_review_fact_mv
AS
SELECT marketplace,
       product_id,
       COUNT(1) as count_rating,
       SUM(star_rating) as sum_rating,
       SUM(helpful_votes) AS total_helpful_votes,
       SUM(total_votes) AS total_votes,
       review_date
FROM public.daily_product_reviews
GROUP BY marketplace,
         product_id,
         review_date;
```

3. Create the stored procedure that will enable you to build the ETL pipeline:

```
            CREATE OR REPLACE PROCEDURE products_review_etl()
            AS $$
            BEGIN
            truncate public.product_reviews_daily;
              COPY public.product_reviews_daily FROM 's3://packt-
redshift-cookbook/amazon-reviews-pds/parquet/product_category=Home/'
                  iam_role 'arn:aws:iam: [Your-AWS_Account_
Id]:role/redshift-spectrum'
                  PARQUET ;
              REFRESH MATERIALIZED VIEW public.daily_product_review_
fact_mv;
            END;
            $$ LANGUAGE plpgsql;
```

4. Navigate to the AWS console and select **Amazon SNS**. From the menu on the left side, click on **Topics**, choose **Standard**, and name it `products-review-communication`. This SNS topic will be used for communication on the status of the data pipeline.

Also, note down the ARN value; let's call this `[Your-SNS_ARN]`, as follows:

Figure 5.6: products-review-communication

5. To subscribe to the `products-review-communication` topic, create a subscription. Select the ARN for the `products-review-communication` topic. Use the Protocol email and give it your email ID. Select **Create subscription**:

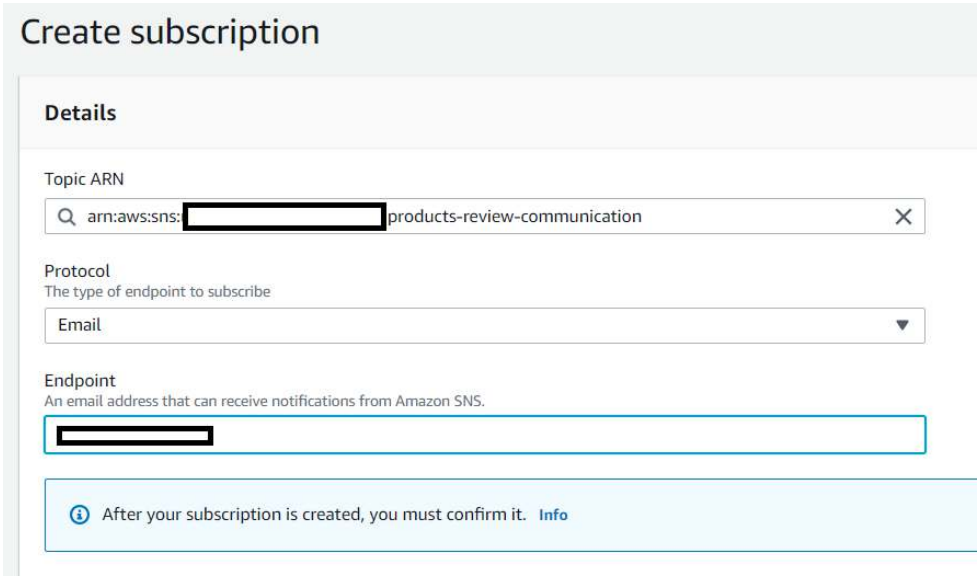


Figure 5.7: Creating the Amazon SNS subscription

6. You will receive an email to confirm the subscription for the `product-review-communication` topic. Select **Subscription confirmed**.

   Next in the pipeline, we will create a Lambda function that will execute the stored procedure using the Redshift Data API. This function will also check the status of the query execution and send notifications on the execution status.

7. Navigate to the AWS console, select **AWS Lambda**, choose **Functions** from the left-hand menu, and create the function as follows:

   - **Function name**: `product-reviews-etl-using-dataapi`
   - **Runtime**: **Python 3.13**
   - **Change default execution role**: Choose the Lambda role you created in the *Getting started* section

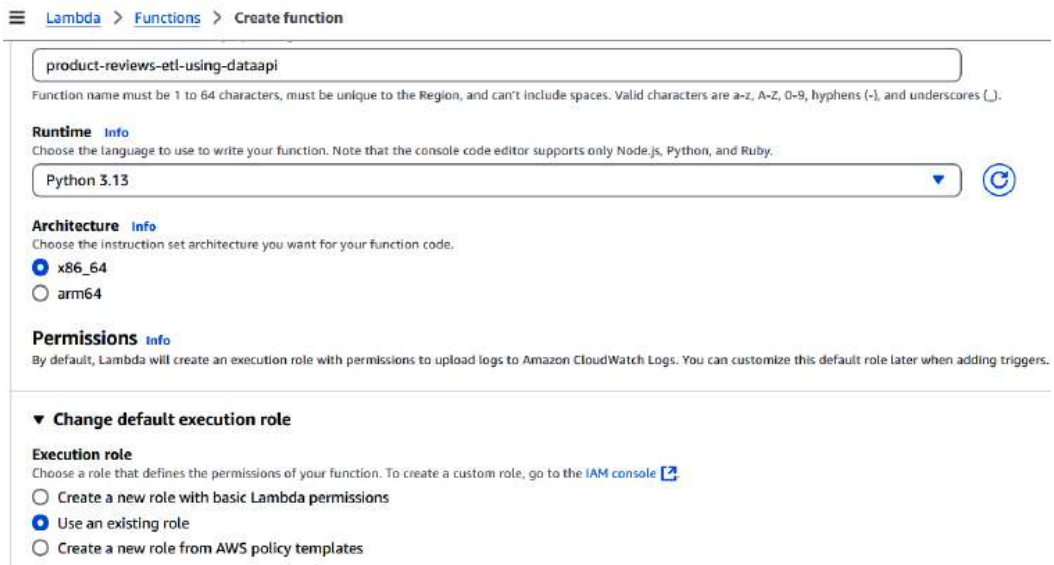


Figure 5.8: Creating the AWS Lambda function

   - Function code: Copy the code for the `product-reviews-etl-using-dataapi` function from `https://github.com/PacktPublishing/Amazon-Redshift-Cookbook-2E/blob/main/Chapter05/src/event-bridge-lambda-function.py`
   - Choose **Deploy**
   - **Change basic settings**: Set the Lambda timeout to **30 seconds**

8. Let's now create the scheduler event rule to trigger the `product-reviews-etl-using-dataapi` Lambda function. Navigate to the AWS console, select **Amazon EventBridge**, choose **Rules** from the left-hand menu, select **default** from the **Event bus** dropdown, and click on **Create rule**. Select the following options in the rules:

    - **Name**: `schedule-productsreview-etl-execution`
    - **Define pattern**: `Schedule`
    - Cron expression: `0 20 ? * MON-FRI *`

Note

This rule will trigger at 3 AM UTC from Monday to Friday.

9. In **Select targets,** choose **Lambda function** for the target and pick the `product-reviews-executesql` function from the dropdown, as follows:

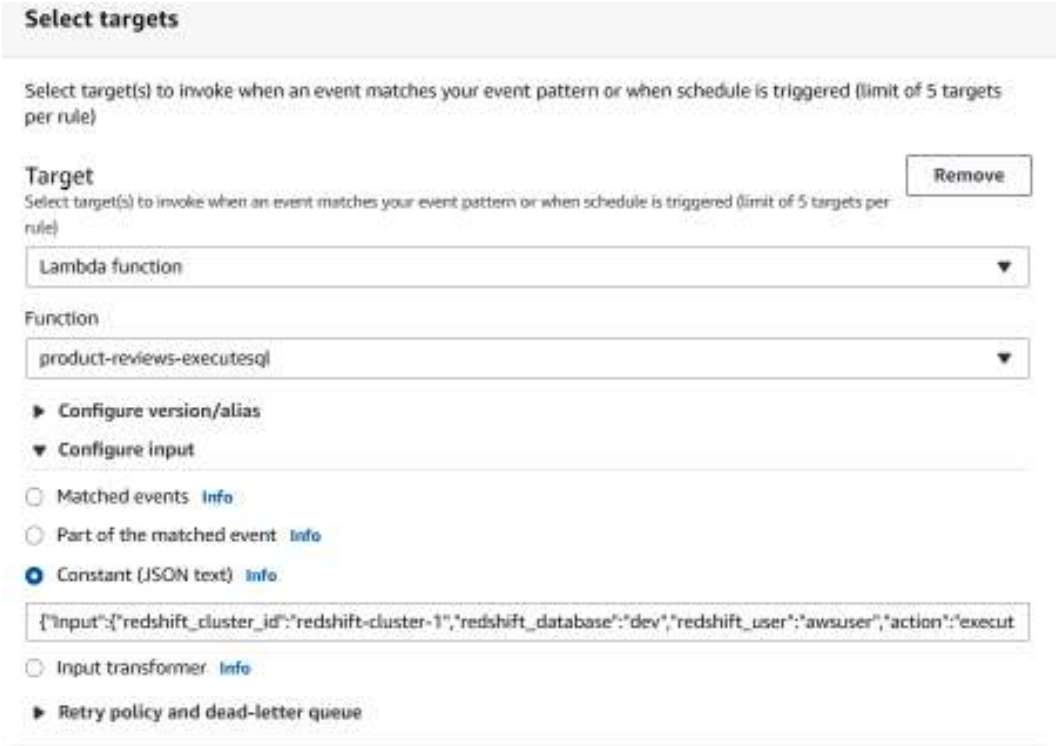


Figure 5.9: Selecting the targets for the Amazon EventBridge rules

10. Under **Configure input**, select **Constant (JSON text)** and provide the following, replacing `[Your-Redshift_Data warehouse]`, `[Your-Redshift_User]`, and `[Your-SNS_ARN]` with the respective values and click **Create:**

```
{
  "Input":{
    "redshift_data warehouse_id":"[Your-Redshift_Data warehouse]",
    "redshift_database":"dev",
    "redshift_user":"[Your-Redshift_User]",
    "action":"execute_sql",
    "sql_text":"call
    products_review_etl();",
    "sns_topic_arn":"[Your-SNS_ARN]"
  }
}
```

11. Let's create another rule to check the status of the stored procedure execution. Click on **Rules** from the left menu and select the options as follows:

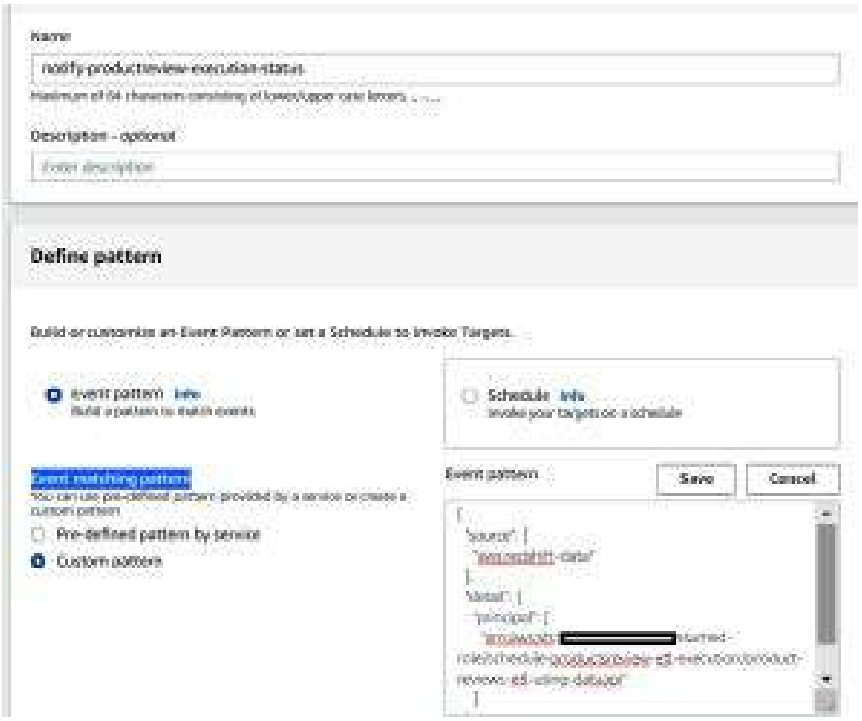


Figure 5.10: Creating a notify-productreview-execution-status rule

- **Name:** `notify-productreview-execution-status`
- **Define pattern: Event pattern**
- **Event matching pattern: Custom pattern**
- **Event pattern:** Provide the following, replacing `[Your-AWS_Account_Id]` and `[Your-Redshift_Role]` with their respective values. Then, select **Save**:

```
{
  "source": [
    "aws.redshift-data"
  ],
  "detail": {
    "principal": [
      "arn:aws:sts::[Your-AWS_Account_Id]:assumed-role/[Your-
Redshift_Role]/product-reviews-executesql"
    ]
  }
}
```

12. Under **Target**, select **Lambda function** and choose the `product-reviews-executesql` function, as follows:

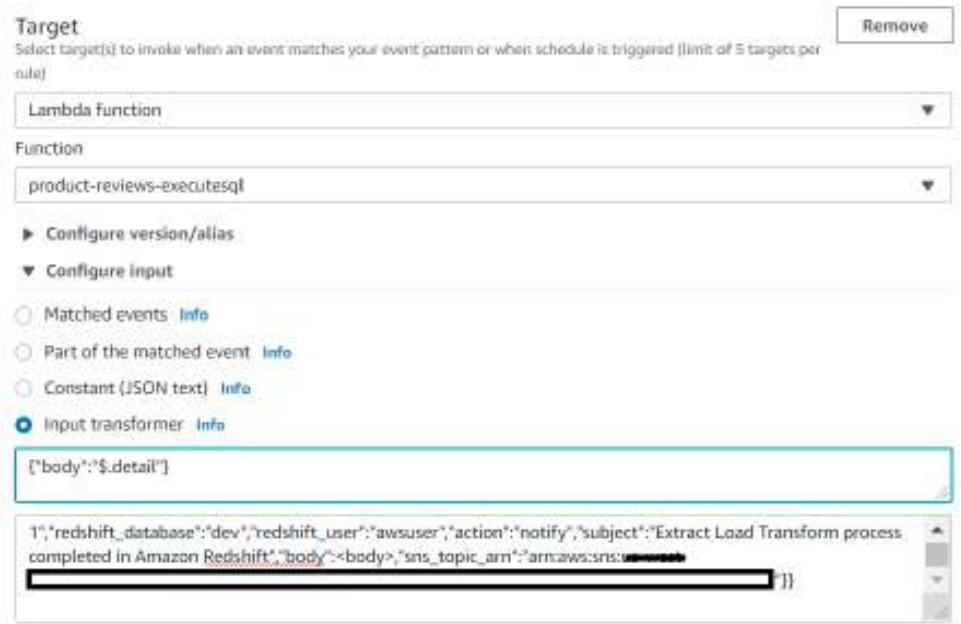


Figure 5.11: Configuring targets for the notify-productreview-execution-status rule

13. Choose **Input transformer** and enter `{"body":"$.detail"}` in the input path.
14. In the **Input template** textbox, enter the following, replacing `[Your-Redshift_Data warehouse]`, `[Your-Redshift_User]`, and `[Your-SNS_ARN]` with the respective values, and click on **Create**:

```
{"Input":{"redshift_data warehouse_id":"[Your-Redshift_Data
warehouse]","redshift_database":"dev","redshift_user":"[Your-
Redshift_User]","action":"notify","subject":"Extract Load Transform
process completed in Amazon Redshift","body":[body],"sns_topic_
arn":"[Your-SNS_ARN]"}}
```

15. When the set schedule is met, the Lambda function will trigger. To validate that the event pipeline is working correctly, navigate to the AWS console and select **CloudWatch**. From the left menu, choose **Log Groups** and filter for the `product-reviews-executesql` Lambda function.

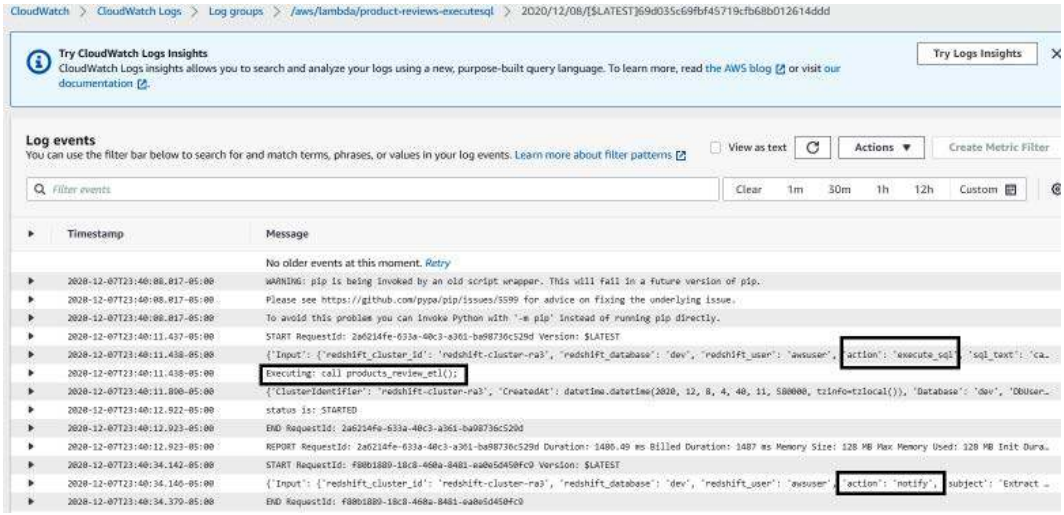


Figure 5.12: Verifying the Lambda function trigger using Cloudwatch

16. On completion of the query, you will receive an email notification on the completion status:

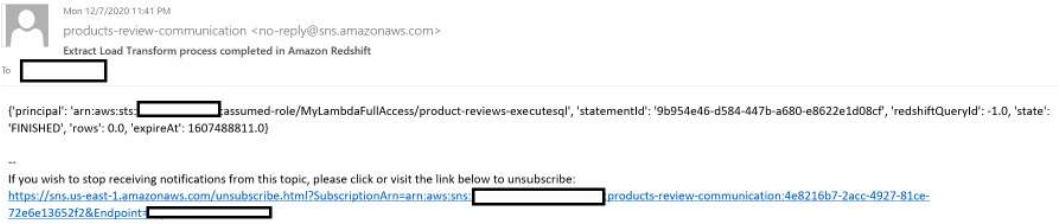


Figure 5.13: Email notification on completion of the event

17. Let's also validate the query execution on Amazon Redshift. In the AWS console, navigate to Amazon, click on **Query monitoring**, and notice the `product_review_etl` call in the list to confirm successful execution.

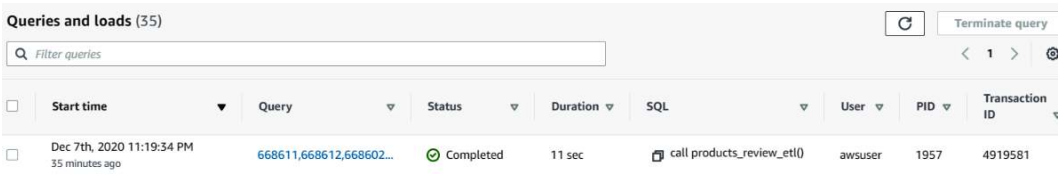


Figure 5.14: Verifying query execution using the Amazon Redshift console

How it works...

Amazon EventBridge is used to orchestrate the product reviews data pipeline. Here is the architecture of this setup:

Figure 5.15: Architecture of Amazon EventBridge setup

This workflow uses Amazon EventBridge to invoke the AWS Lambda function based on a schedule. AWS Lambda executes the data pipeline queries through the Amazon Redshift Data API. Amazon Redshift publishes custom notifications through Amazon SNS for the completion and notifies the users. You are able to integrate a serverless decoupled pipeline that is scalable.

EventBridge allows you to connect applications using events. An event is a trigger when the system state changes that can be used to drive a workflow such as ETL. This also allows you to integrate your own AWS applications with microservices, SaaS applications, and custom applications as event sources that publish events to an event bus.

Event-driven applications using AWS Lambda on Amazon Redshift provisioned clusters

AWS Lambda helps you build an event-driven microservice. This serverless process can be invoked using a variety of events such as when a file arrives, when a notification is received, and so on. This helps build a decoupled data workflow that can be invoked as soon as the upstream dependencies are met, instead of a schedule-based workflow. For example, let's say we have a website that is continuously sending clickstream logs every 15 minutes into Amazon S3.

Instead of accumulating all the log files and processing them at midnight in a typical ETL process, Amazon S3 can send an event to a Lambda function when an object is created and processed immediately. This provides several advantages, such as processing in smaller batch sizes to meet an SLA and to have the data current within the provisioned cluster.

In this recipe, you will learn how to use Python-based AWS Lambda to copy data into Amazon Redshift as soon as the file arrives at the Amazon S3 location.

There are several ways to invoke an AWS Lambda using an event that is detailed in

`https://docs.aws.amazon.com/lambda/latest/dg/lambda-invocation.html`

Getting ready

To complete this recipe, you will need the following:

- Amazon Redshift provisioned cluster deployed in the AWS Region eu-west-1. Note the data warehouse ID; we will refer to it as `[Your-Redshift_Cluster]`.
- Amazon Redshift data warehouse master user credentials. Note the username; we will refer to it as `[Your-Redshift_User]`.

- Access to any SQL interface, such as a SQL client or Amazon Redshift QEV2.
- IAM user with access to Amazon Redshift, Amazon S3, and AWS Lambda.
- Amazon S3 bucket created in eu-west-1; we will refer to it as `[Your-Amazon_S3_Bucket]`.
- AWS account number; we will reference it in recipes as `[Your-AWS_Account_Id]`.

How to do it...

In this recipe, we will use Python-based AWS Lambda to copy data into Amazon Redshift as soon as the file arrives at the Amazon S3 location. Follow these steps:

1. The AWS Lambda package is available at `https://github.com/PacktPublishing/Amazon-Redshift-Cookbook-2E/blob/main/Chapter05/src/my-lambda-deployment-package.zip`. Download this deployment package to your local folder.
2. Navigate to the AWS console, select the **Lambda** service, and click on **Create function**, as follows:

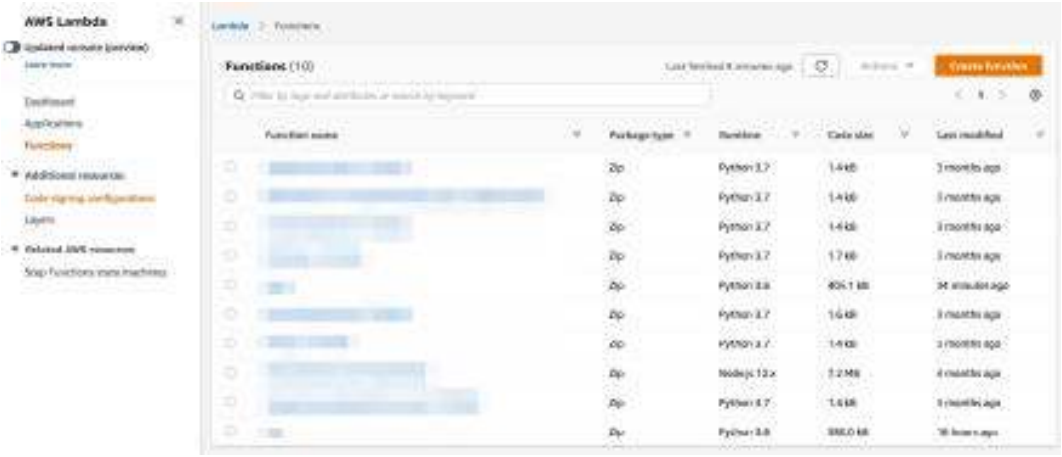


Figure 5.16: Creating an AWS Lambda function using the AWS console

3. In the **Create function** section, enter `lambda_function` under **Function name**, choose **Python 3.13** for **Runtime**, and click on **Create function**:

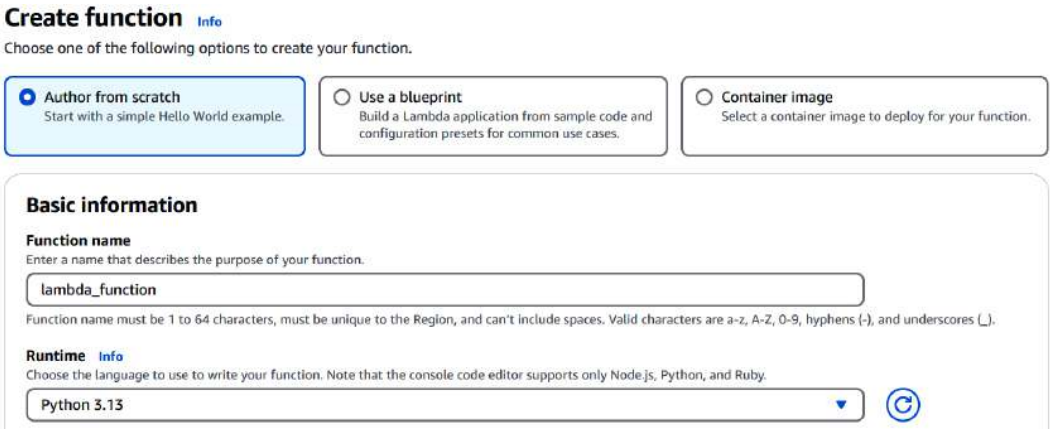


Figure 5.17: Creating the lambda_function function

4. In the **Code source** section, choose to upload from `.zip` file. Select the `my-lambda-deployment-package.zip` from your local folder and click on **Save**. Now, the Lambda code and Python package will be successfully imported.
5. Click on `lambda_function.py` and edit the values for the following parameters to point to your Amazon Redshift data warehouse:

```
  db_database = "[database]"
  db_user = "[user]"
  db_password = "[password]"
  db_port = "[port]"
  db_host = "[host]"
  iam_role = "'arn:aws:iam::[Your-AWS_Account_Id]:role/[Your-
Redshift-Role]"
```

6. Click on **Deploy** to save the changes.
7. You can now test `lambda_function` by clicking on the **Test** option. In the **Test** option, choose **Create new test event**. In the **Event** template, choose `hello-world`, and for the event name, enter `myevent`, and copy and paste the following test stub event value:

```
{
  "Records": [
    {
      "eventVersion": "2.1",
```

```
      "eventTime": "2030-12-06T18:43:42.795Z",
      "s3": {
        "s3SchemaVersion": "1.0",
        "configurationId": "test",
        "bucket": {
          "name": "packt-redshift-cookbook"
        },
        "object": {
          "key": "part/000.gz",
          "size": 540
        }
      }
    }
  ]
}
```

Note

This test event will output the bucket name and key. It will also perform a `COPY` operation in Amazon Redshift to create the `stg_part` table and ingest data from `s3://packt-redshift-cookbook/part/000.gz`.

8. Now, let's create an Amazon S3-triggered event so that files can be automatically copied into Amazon Redshift as they get put into your S3 location. Navigate to the Amazon S3 service in the AWS console and click on the `[Your-Amazon_S3_Bucket]` bucket. Select Properties, then click on **Event notifications**, as follows:

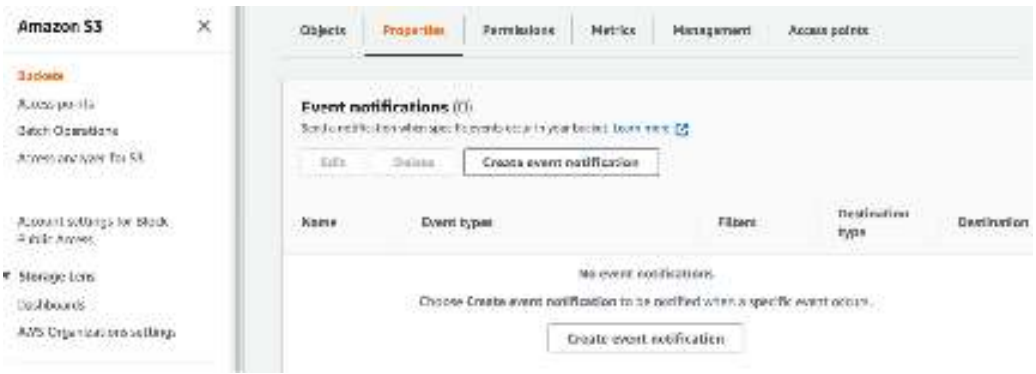


Figure 5.18: Creating event notifications from Amazon S3

9. For **Create event notification**, set up the event details as follows:

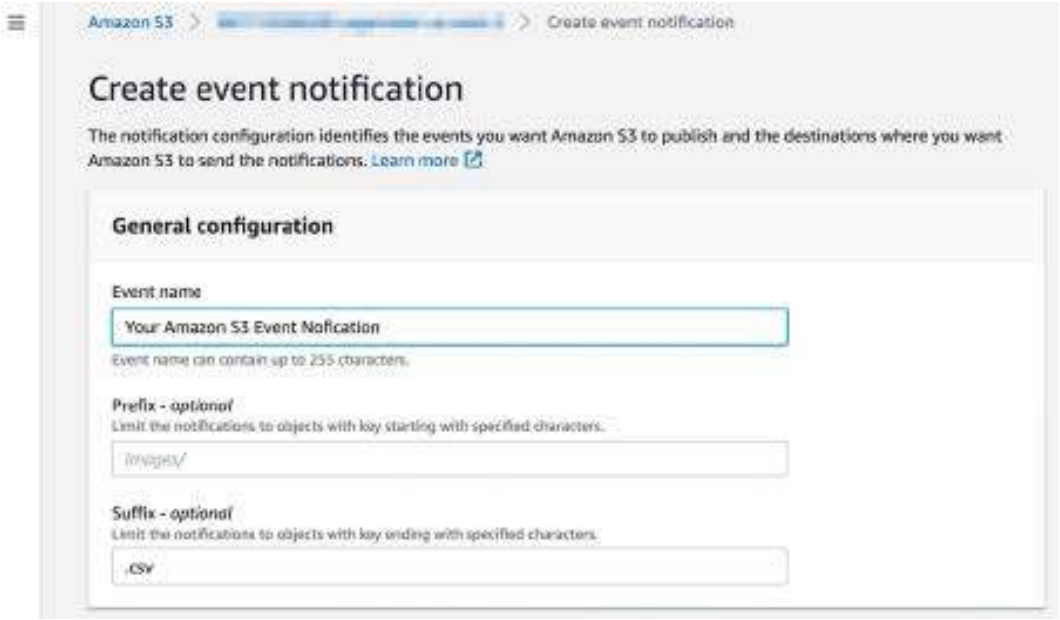


Figure 5.19: Configuring the event notification

   - **Event name**: Any event name of your choice
   - **Prefix**: Your S3 folder location where you plan to put the files to be copied, for example, `events/`
   - **Suffix**: `.csv`
   - **Event types**: Check **Put**
   - **Destination**: `Lambda Function`
   - **Specify Lambda function**: Choose `lambda_function` from the list

10. Now, click on **Save changes**.
11. Download the `s3://packt-redshift-cookbook/part/000.gz` and `s3://packt-redshift-cookbook/part/000.gz` public S3 files to your location folder.
12. Navigate to your Amazon S3 bucket, `[Your-Amazon_S3_Bucket]`, and upload `000.gz` from your local folder, followed by `001.gz`.
13. From the AWS console, navigate to **Lambda** and select the `lambda_function` function. Click on **Monitoring** and you will notice that there are two invocations of Lambda that copied the uploaded files automatically to Amazon Redshift.

To verify the execution of `lambda_function`, click on **View logs in CloudWatch**, which shows the execution logs.

How it works...

The AWS Lambda deployment package bundles the Python function code and the dependent `psycog2` library (`https://www.psycopg.org/`), which is used to connect to Amazon Redshift. You can include any other dependent packages that you may need to meet your organizational requirements when creating this deployment package.

Also, as a best practice, you can enhance the `lamda_function` code to retrieve the Amazon Redshift credentials using AWS Secret manager, as illustrated in `https://docs.aws.amazon.com/code-samples/latest/catalog/python-secretsmanager-secrets_manager.py.html`.

See also...

- There are several ways to invoke an AWS Lambda function using an event. These are detailed at `https://docs.aws.amazon.com/lambda/latest/dg/lambda-invocation.html`.
- You can build this deployment package from scratch using the instructions at `https://docs.aws.amazon.com/lambda/latest/dg/python-package.html` and `https://pypi.org/project/aws-psycopg2/`.

Orchestration using AWS Step Functions on provisioned clusters

AWS Step Functions allow you to author a workflow where each step is decoupled, but the application state can be maintained. AWS Step Functions is integrated with multiple AWS services that allow flexibility to call the specific service in each of the tasks.

AWS Step Functions supports the Amazon States Language, which allows the workflow to be authored and maintained like a JSON file. You can harness AWS Step Functions to execute any complex ETL workflow in Amazon Redshift. AWS Step Functions also integrates with Amazon Redshift serverless.

In this recipe, we will use AWS Step Functions to orchestrate a simple ETL workflow that will submit queries to Amazon Redshift asynchronously using the Amazon Redshift Data API. We will start with creating an AWS Lambda function that will be used to submit and status poll for the queries.

Getting ready

To complete this recipe, you will need the following:

- Amazon Redshift provisioned cluster deployed in the AWS Region eu-west-1. Note the data warehouse ID; we will refer to it as `[Your-Redshift_Cluster]`.
- Amazon Redshift data warehouse master user credentials. Note the username; we will refer to it as `[Your-Redshift_User]`.
- Access to any SQL interface such as a SQL client or Amazon Redshift QEV2.
- IAM user with access to Amazon Redshift and AWS Lambda.

How to do it...

1. Navigate to the AWS console, select the AWS Lambda service, and click on **Create function**, as follows:

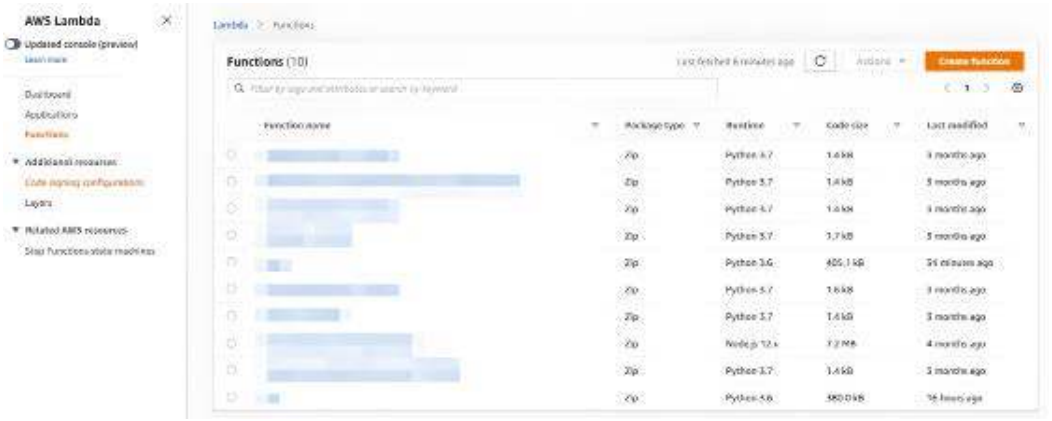


Figure 5.20: Creating an AWS Lambda function using the AWS console

2. In the **Create function** section, enter `submit_redshift_query` under **Function name**, choose **Python 3.6** for **Runtime**, and click on **Create function**.
3. In the function code for `lambda_function.py`, copy and paste the code from `https://github.com/PacktPublishing/Amazon-Redshift-Cookbook-2E/blob/main/Chapter05/src/stepfunction/lambda_submit_redshift_query.py` and click on **Deploy**, which will save the function.

4. In the **Permissions** tab of the AWS Lambda function, click on the auto-created **Role name** value (`submit_redshift_query-role-***`), as follows:

Figure 5.21: Configuring the permissions for the AWS Lambda

5. In **Identity and Access Management (IAM)**, which opens in a different tab, copy and paste the following policy by clicking on **Add inline policy**, which is available at `https://github.com/PacktPublishing/Amazon-Redshift-Cookbook-2E/blob/main/Chapter05/src/stepfunction/lambda_execute_policy.json`.
6. Click on **Test**, select **Configure events**, choose **Create new test event**, and set the event template as `hello-world`:

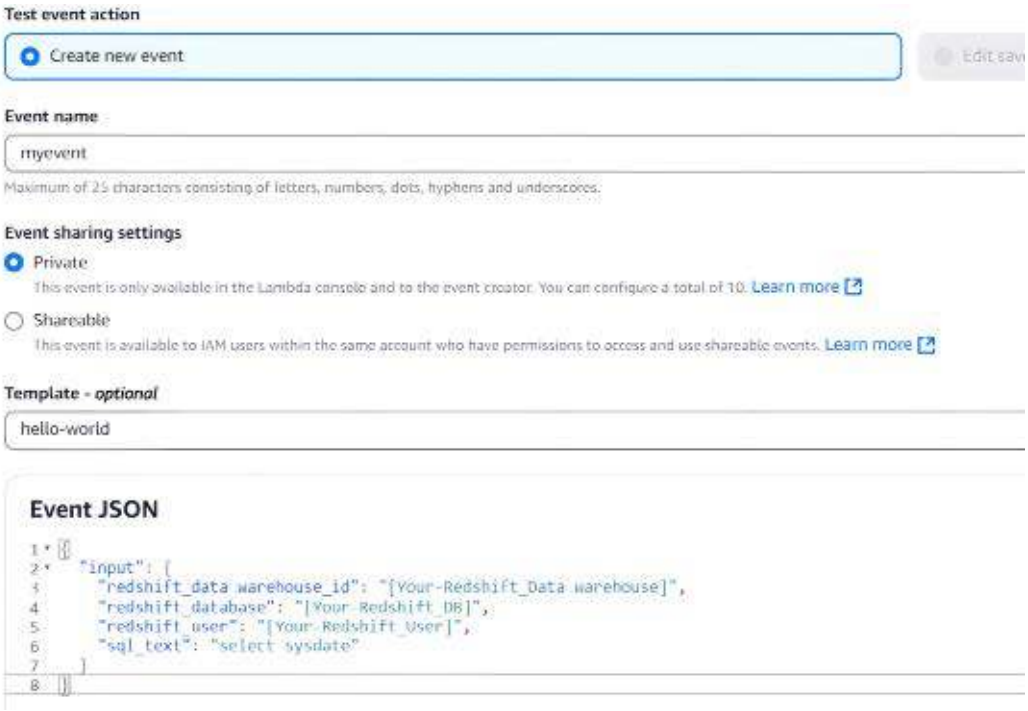


Figure 5.22: Setting up the test event for AWS Lambda

7. In the event textbox, copy the following sample input, replacing `[Your-Redshift_Cluster]`, `[Your-Redshift_DB]`, and `[Your-Redshift_User]` with your Amazon Redshift data warehouse, and press the **Create** button:

```
{
  "input": {
    "redshift_data warehouse_id": "[Your-Redshift_Data warehouse]",
    "redshift_database": "[Your-Redshift_DB]",
    "redshift_user": "[Your-Redshift_User]",
    "sql_text": "select sysdate"
  }
}
```

8. Press the **Test** button and you should be able to see that the sample query was submitted in the execution results, as follows, for successful submission:

```
START RequestId: 43df694d-3716-474f-b279-cd7b976ef05c Version:
$LATEST
{'input': {'redshift_data warehouse_id': 'demodata warehouse-
71f3476d',
       'redshift_database': 'dev',
       'redshift_user': 'demo',
       'sql_text': 'select sysdate'}}
{'Data warehouseIdentifier': 'demodata warehouse-71f3476d',
'CreatedAt': datetime.datetime(2020, 12, 9, 0, 47, 2, 353000,
tzinfo=tzlocal()),
'Database': 'dev',
'DbUser': 'demo',
'Id': '0ce38431-be55-4c4b-97c8-230624a01c76', 'ResponseMetadata':
{'RequestId': 'dbabb5dc-8de8-4f59-80f9-367319eeaecb',
       'HTTPStatusCode': 200,
```

```
        'HTTPHeaders': {'x-amzn-requestid': 'dbabb5dc-8de8-4f59-80f9-
367319eeaecb',
        'content-type': 'application/x-amz-json-1.1', 'content-
length': '150',
        'date': 'Wed, 09 Dec 2020 00:47:02 GMT'}, 'RetryAttempts':
0}}
END RequestId: 43df694d-3716-474f-b279-cd7b976ef05c
```

9. Repeat *steps 1–8* to create another AWS Lambda function named `poll_redshift_query` using the following code:

    - AWS Lambda code: `https://github.com/PacktPublishing/Amazon-Redshift-Cookbook-2E/blob/main/Chapter05/src/stepfunction/lambda_poll_redshift_query.py`
    - AWS Lambda test event: `https://github.com/PacktPublishing/Amazon-Redshift-Cookbook-2E/blob/main/Chapter05/src/stepfunction/lambda_poll_redshift_query_test.json`

10. Let's now start creating the AWS Step Functions function to orchestrate a simple workflow to submit and monitor the job using the AWS Lambda functions we have created. Navigate to the AWS console and select the **Step Functions** service. Click on **Create state machine**, as follows:

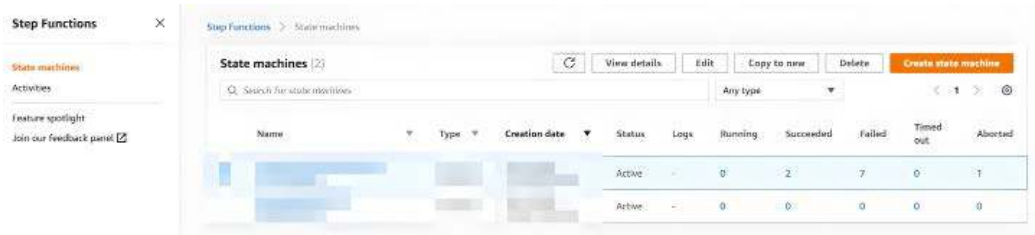


Figure 5.23: Creating a Step Functions state machine

11. Select **Generate code snippet** and **Standard** to copy and paste the following code (which is available at `https://github.com/PacktPublishing/Amazon-Redshift-Cookbook-2E/blob/main/Chapter05/src/stepfunction/stepfunction_job_redshift.json`) in the **Definition** section and click **Next**:

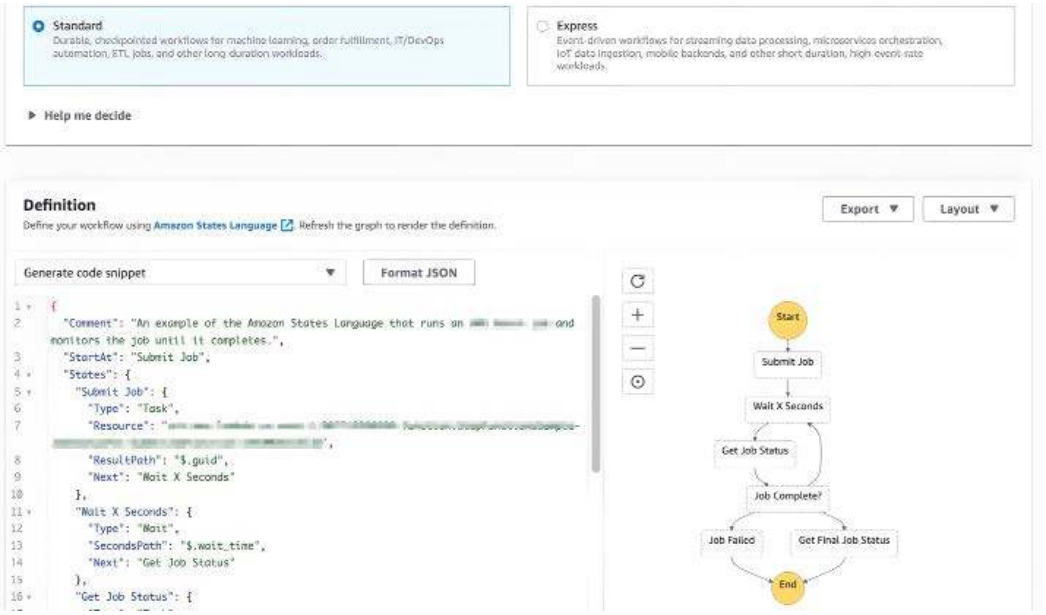


Figure 5.24: Setting up the Step Functions workflow definition

12. Under the **Permissions** tab, click on **Create new role** and click **Next** to create the AWS Step Functions state machine.
13. Click on **Start execution** and, under the input, provide the following details, which are also available at `https://github.com/PacktPublishing/Amazon-Redshift-Cookbook-2E/blob/main/Chapter05/src/stepfunction/stepfunction_job_redshift_test.json`:

```
{
  "input": {
    "redshift_data warehouse_id": "[Your-Redshift_Data warehouse]",
    "redshift_database": "[Your-Redshift_DB]",
    "redshift_user": "[Your-Redshift_User]",
    "sql_text": "select sysdate"
  },
  "wait_time": "3"
}
```

14. Now, you can monitor the execution of this workflow under the **Details** tab, as follows:

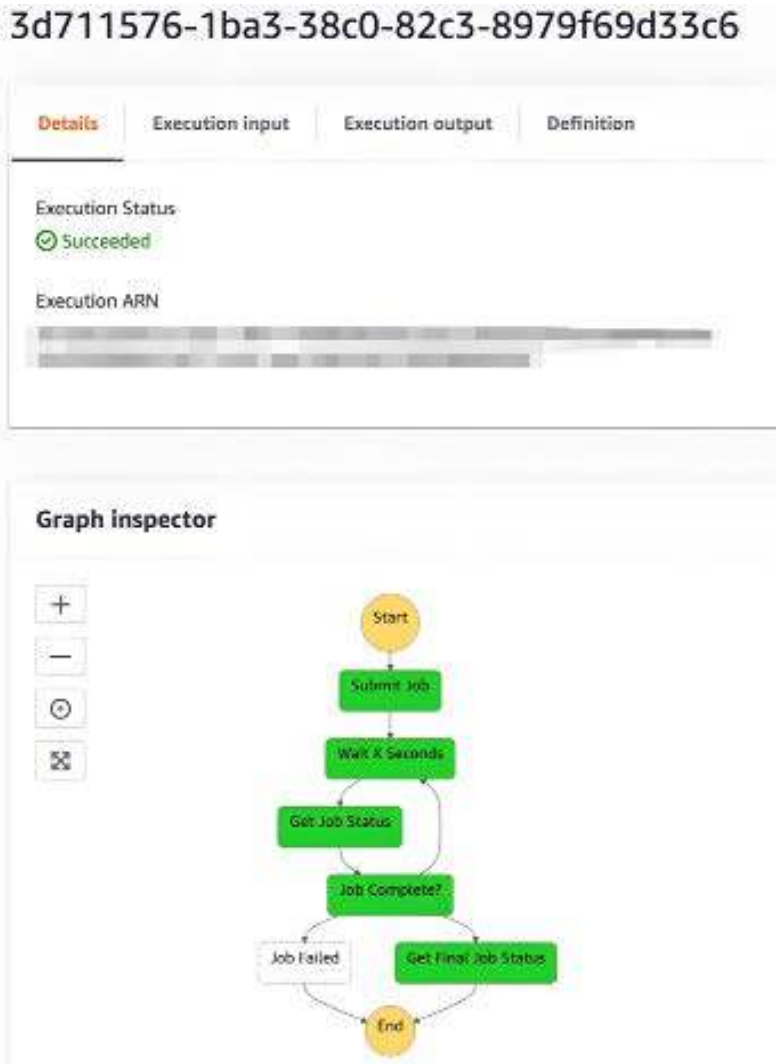


Figure 5.25: Monitoring the event function workflow

How it works...

AWS Step Functions uses the Amazon States Language, which is JSON-based. You can author most kinds of ETL process and drive a workflow that can wait for dependency between each task and also allow for parallelism when needed. The AWS state machine can be triggered either through an event or scheduled for automation.

See also...

- You can see the list of natively supported integrations here: `https://docs.aws.amazon.com/step-functions/latest/dg/concepts-service-integrations.html`.
- For more information on the Amazon States Language, go to `https://docs.aws.amazon.com/step-functions/latest/dg/concepts-amazon-states-language.html`.

Orchestration using Amazon Managed Workflows for Apache Airflow on provisioned clusters

Amazon **Managed Workflows for Apache Airflow** (**MWAA**) brings automation to life by managing complex data pipelines from beginning to end. At its core, it handles Apache Airflow, which creates, schedules, and monitors your workflows with precision. Each data pipeline breaks down into smaller, interconnected tasks that work together seamlessly in a coordinated flow.

Using Python, developers craft these workflows as **Directed Acyclic Graphs** (**DAGs**), defining the exact path and sequence for data processing. The system grows and adapts through powerful plugins, while a user interface provides clear visibility into every workflow's status and progress. Working hand in hand with Amazon Redshift serverless, it creates a complete ecosystem for data processing, all while Amazon manages the underlying infrastructure.

In this recipe, we will build the underlying infrastructure used for Apache Airflow using Amazon MWAA. After the infrastructure is built, we will build a data pipeline for the `part` table.

Getting ready

To complete this recipe, you will need the following:

- Amazon Redshift provisioned cluster deployed in the AWS Region eu-west-1. Note the cluster ID; we will refer to it as `[Your-Redshift_Cluster]`.
- Amazon Redshift data warehouse master user credentials. Note the username; we will refer to it as `[Your-Redshift_User]`.
- Access to any SQL interface, such as a SQL client or Amazon Redshift Query Editor.
- IAM user with access to Amazon Redshift and Amazon MWAA. Version 2.10.1
- Amazon S3 bucket created in eu-west--1; we will refer to it as `[Your-Amazon_S3_Bucket]`.

How to do it...

In this recipe, we will set up a data pipeline using Apache Airflow that will connect to Amazon Redshift to orchestrate a workflow:

1. Browse to the Amazon S3 console and select `[Your-Amazon_S3_Bucket]`. Create a folder called `airflow` within the bucket. We will use this folder to store the Airflow DAGs and requirements file providing the list of dependencies needed to run the Python DAG.
2. You can use the CLI or the S3 console to upload the files. Upload the requirements file (`https://github.com/PacktPublishing/Amazon-Redshift-Cookbook-2E/blob/main/Chapter05/src/requirements.txt`) to the bucket location at `s3://[Your-Amazon_S3_Bucket]/airflow`.
3. Download the DAG script from `https://github.com/PacktPublishing/Amazon-Redshift-Cookbook-2E/blob/main/Chapter05/src/redshift_parts_airflow_dag.py`. For `load_sql`, replace the name of the s3 bucket `<Your-Amazon_S3_Bucket >` and iam role `<your-Redshift-role>` in the script. Save it and upload the workflow Python script (DAG) to the newly created `dags` folder in your `airflow` bucket.

Figure 5.26: Setting up an Apache Airflow DAG

4. We are now ready to build the infrastructure and setup needed for Apache Airflow. Navigate to the AWS console in the AWS Region eu-west-1 and select **Managed Workflows for Apache Airflow** (**MWAA**). Choose **Create environment**.
5. Name the environment `MyAirflowEnvironment`.
6. Choose the latest Airflow version.
7. For **S3 Bucket**, specify the `s3://[Your-Amazon_S3_Bucket` bucket. The bucket needs to be in the same Region in which you are creating MWAA.
8. For **DAGs folder**, enter `s3://[Your-Amazon_S3_Bucket]/airflow/dags`

9. For **Requirements file**, enter `s3://[Your-Amazon_S3_Bucket]/airflow/requirements.txt`

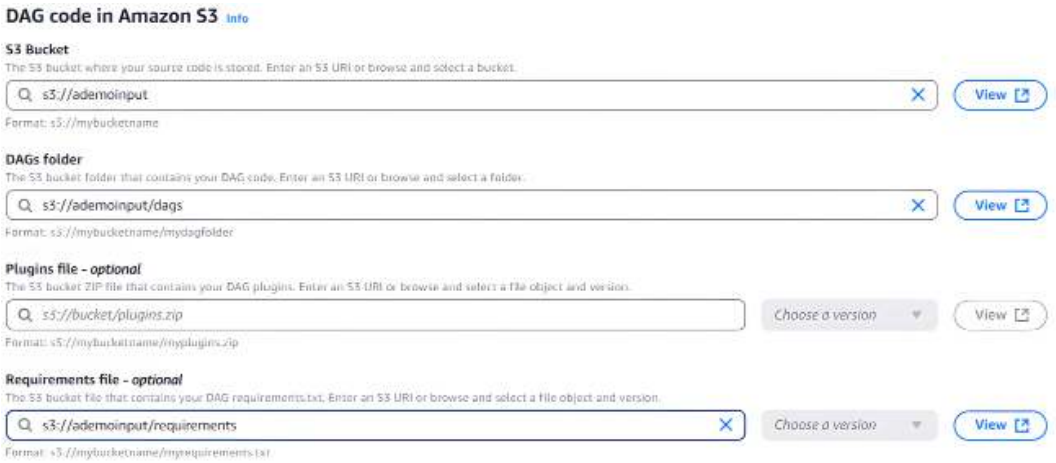


Figure 5.27: Configuring the source Amazon S3 bucket

10. Choose **Next**. If you have an existing VPC, choose it from the dropdown; if you do not have an existing VPC, choose **Create MWAA VPC**. This will launch a CloudFormation template, create the stack, and, on completion, navigate back to the MWAA setup step.
11. From the dropdown, select the VPC and the subnets. Set the web server access as **Public network**:

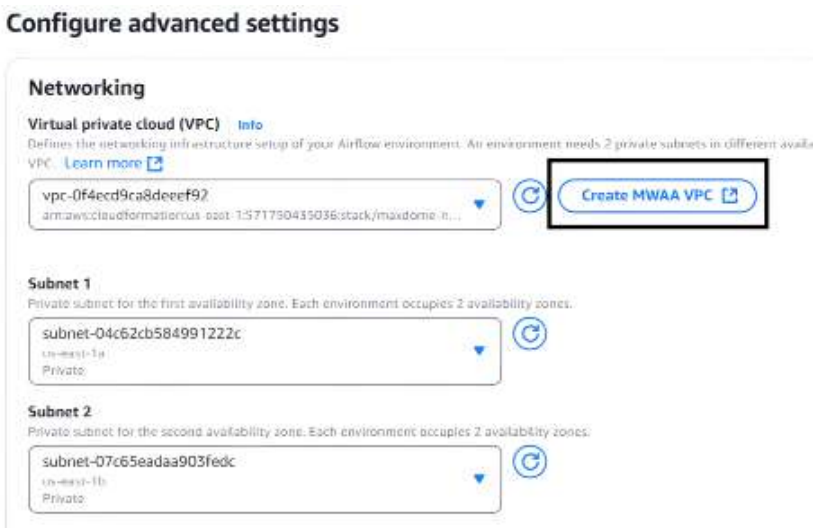


Figure 5.28: Setting up the network access to connect to Amazon Redshift

12. Select the **mw1.small** instance type. For the rest, keep the defaults for an IAM role.

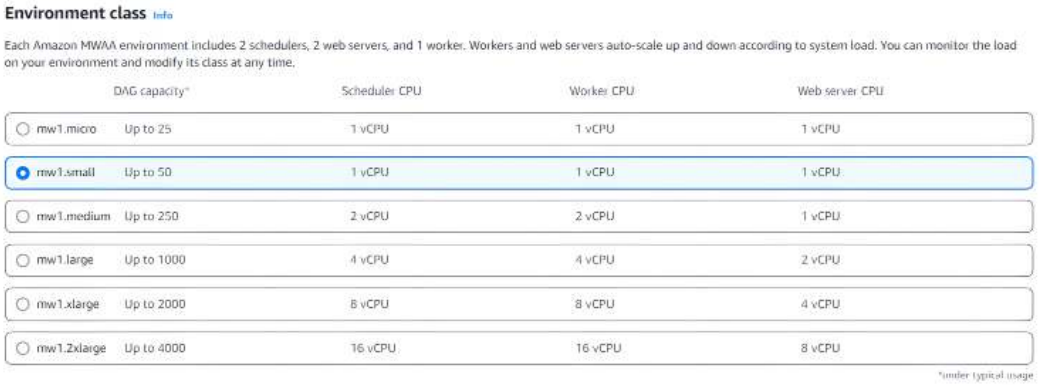


Figure 5.29: Configuring the Amazon EC2 instance for Airflow

13. Choose **Create environment**. On completion of the setup, it will make the environment available with Apache Airflow. We are now ready to execute the workflow.
14. Select **Open Airflow UI** from the environment.

Figure 5.30: Setting up the Airflow environment

15. From the UI, click on **Admin** and choose **Connections**. We will configure the connection for the Amazon Redshift data warehouse, which will be used in the workflow tasks.

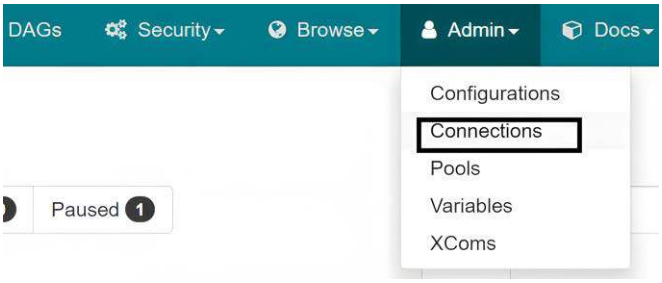


Figure 5.31: Setting up the Amazon Redshift connection

16. Navigate to the `conn_id` Postgres connection and choose **Edit**.
17. Specify your Redshift data warehouse endpoint, username, password, and port number. Click on **Save**.

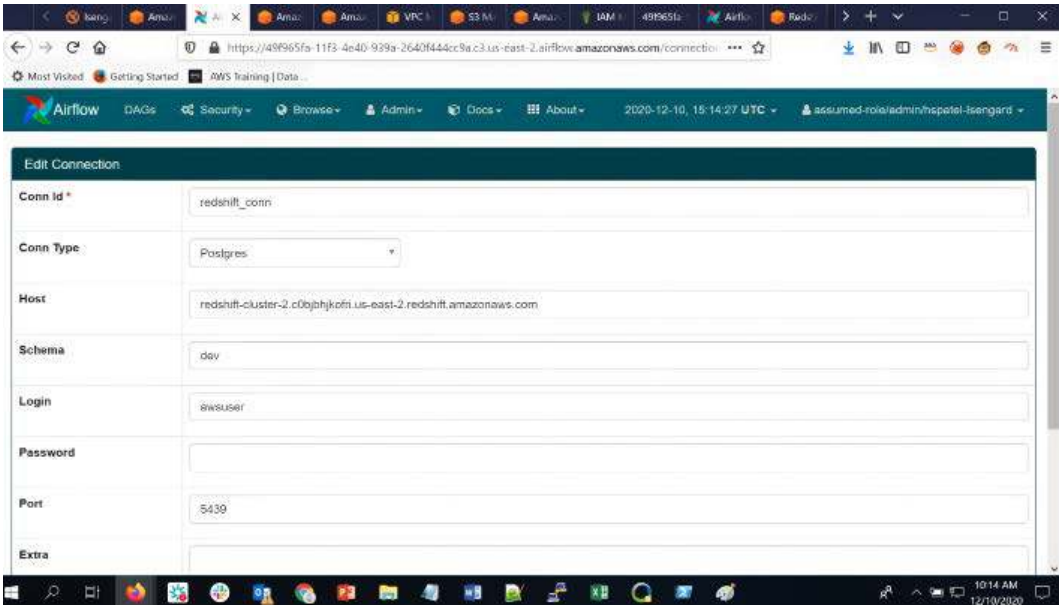


Figure 5.32: Configuring the Amazon Redshift connection properties

18. Now that the setup is complete, from the UI, click on **DAGs**. This will list the `parts-redshift-datapipeline-dag` DAG that you uploaded to the S3 bucket.

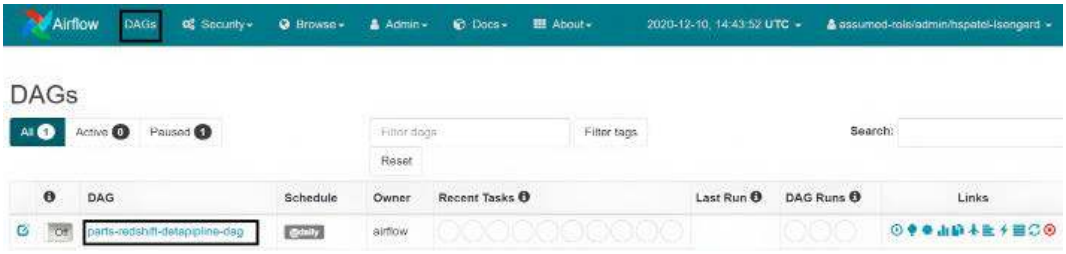


Figure 5.33: Configuring the Airflow DAG

19. Let's check the DAG. Click on the DAG name. This workflow has three tasks: the first will create the `part_stg` table using `PostgresOperator`. The second will use the `copy` command to load the parts sample data from S3. In the final step, it will check the record count in the `part_stg` table using `PythonOperator`.

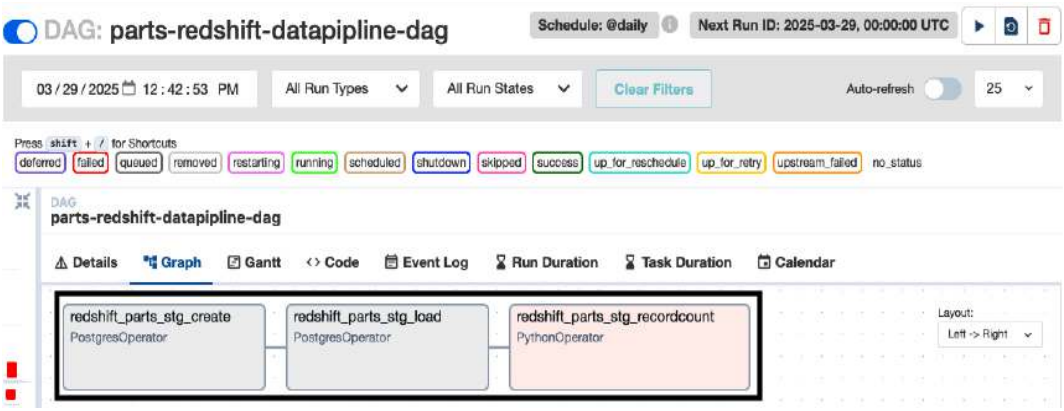


Figure 5.34: Verifying the DAG setup on Airflow

20. Click on **DAGs** in the UI and toggle the DAG to the **On** state – this will put the DAG in the schedule.

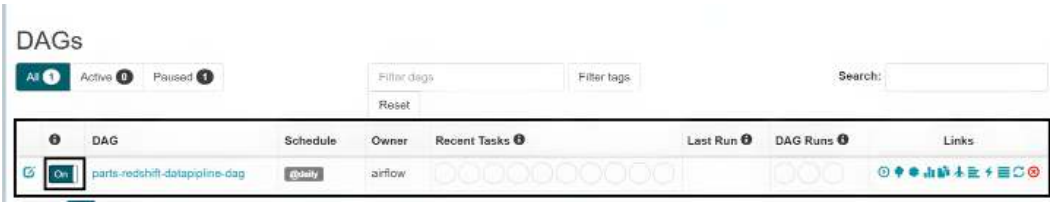


Figure 5.35: Scheduling the workflow execution

21. This will start the execution. Click on the green number under **DAG Runs**.
22. The workflow will execute as per the set dependency. It will run `redshift_parts_stg_create` first and, on it's completion, run the second task. When `redshift_parts_stg_load` has completed successfully, it will execute `redshift_parts_stg_recordcount`. This is the monitoring step.

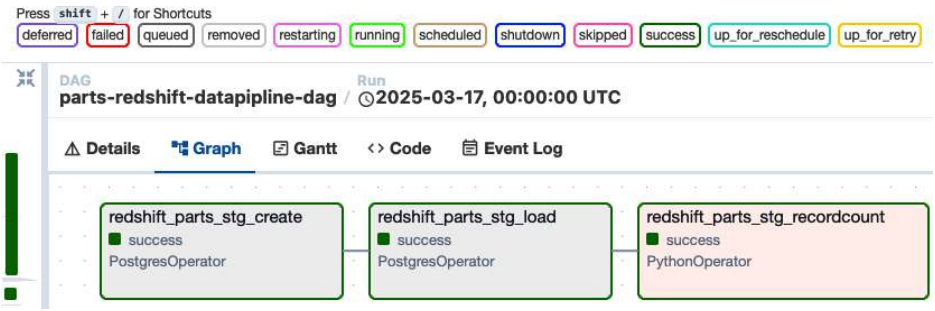


Figure 5.36: Verifying the execution of the workflow

23. Let's validate the logs for the copy and record count step. Click on **redshift_parts_stg_load**. Then, select **Logs**.

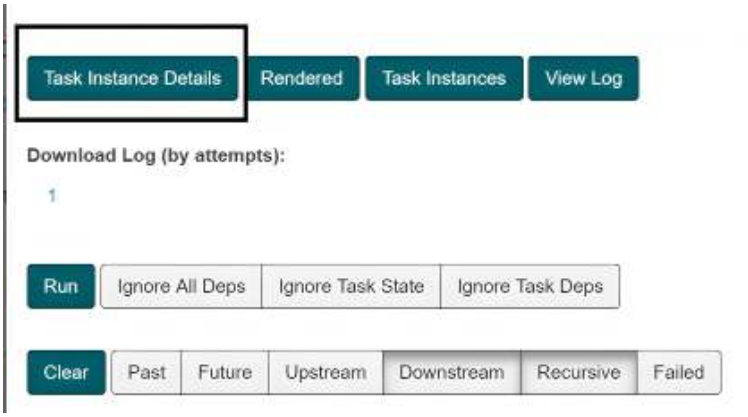


Figure 5.37: Viewing the task execution details

24. Capture the `log_url` value and open a new browser window and paste the URL. The copy task completed successfully; this is logged in the logs and you can verify how many records got loaded.

Figure 5.38: Verifying the task execution detailed logs

25. Similarly, capture the log for the final task and verify the log as a data quality check. This record count of the `part_stg` table is `20000000` records.

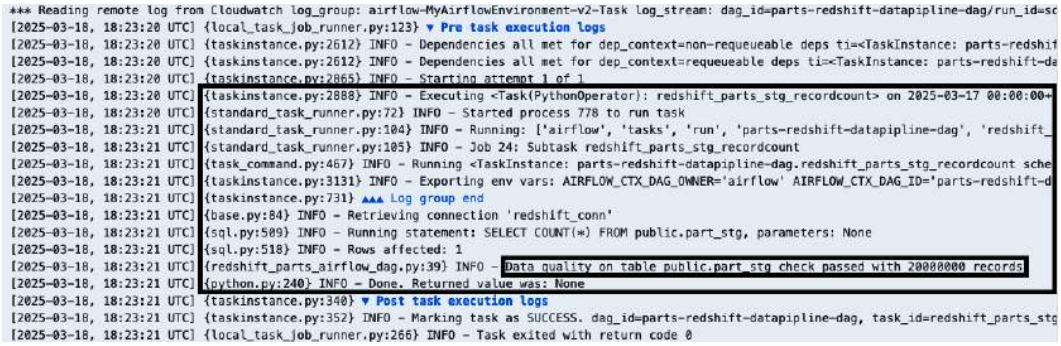


Figure 5.39: Verifying the task execution for the part_stg table

How it works...

Amazon MWAA simplifies the setup needed to build and orchestrate a data pipeline using Apache Airflow. Apache Airflow provides the means to build reusable data pipelines programmatically.

6

Platform Authorization and Security

Amazon Redshift provides out-of-the-box features that enable you to build the data warehouse to meet the requirements of the most security-sensitive organizations. In AWS, security is the highest priority and is a shared responsibility (`https://aws.amazon.com/compliance/shared-responsibility-model/`) between AWS and you. Using Amazon Redshift managed service, the data center and network architecture come out of the box to meet the security-sensitive organizations. You can now configure the data and cluster management controls to meet your organization's requirements. Data can be encrypted to keep your data secure in transit and at rest using industry-standard encryption techniques. Amazon Redshift resources are controlled in the four different levels of cluster management (creating and configuring the cluster), cluster connectivity, database access to objects, and temporary/single sign-on.

Specifically, the following topics are covered in this chapter:

- Managing infrastructure security
- Data encryption at rest
- Data encryption in transit
- Managing superusers using an Amazon Redshift provisioned cluster
- Using IAM authentication to generate database user credentials for an Amazon Redshift serverless cluster

- Managing audit logs
- Monitoring Amazon Redshift
- Single sign-on using AWS IAM Identity Center
- Metadata security

Technical requirements

Here are the technical requirements to complete the recipes in this chapter:

- Access to the AWS Console.
- An AWS administrator should create an IAM user by following Recipe 1 in the Appendix. This IAM user will be used in some of the recipes in this chapter.
- The AWS administrator should also create an IAM role by following Recipe 3 in the Appendix. This IAM role will be used in some of the recipes in this chapter.
- The AWS administrator should also deploy the AWS CloudFormation template (`https://github.com/PacktPublishing/Amazon-Redshift-Cookbook-2E/blob/main/Chapter06/chapter_6_CFN.yaml`) to create the following two IAM policies:
    - An IAM policy attached to the IAM user that will give them access to Amazon Redshift, Amazon S3, AWS Secrets Manager, Amazon CloudWatch, Amazon CloudWatch Logs, Amazon EC2, Amazon SNS, AWS IAM, AWS KMS, AWS Glue, and Amazon VPC.
    - An IAM policy attached to the IAM role that will allow Amazon Redshift data warehouse to access Amazon S3 and AWS IAM Identity Center.
- Attach an IAM role to the Amazon Redshift data warehouse (provisioned or serverless) by following Recipe 4 in the Appendix. Make a note of the IAM role name; we will use it in the recipes as `[Your-Redshift_Role]`.
- Amazon Redshift data warehouse deployed in the eu-west-1 AWS region.
- Amazon Redshift data warehouse master user credentials.
- Access to any SQL interface, such as a SQL client or the Amazon Redshift query editor.
- AWS account number; we will use it in the recipes as `[Your-AWS_Account_Id]`.
- Amazon S3 bucket created in eu-west-1; we will use it as `[Your-Amazon_S3_Bucket]`.
- Code files in the GitHub repo: `https://github.com/PacktPublishing/Amazon-Redshift-Cookbook-2E/tree/main/Chapter06`.

Managing infrastructure security

Amazon Virtual Private Cloud (**Amazon VPC**) allows you to launch an Amazon Redshift data warehouse in a logically isolated virtual network where you define the IP address range and subnets and configure the infrastructure security. When you provision an Amazon Redshift data warehouse, it is locked down by default, so nobody has access to it. To grant inbound access to an Amazon Redshift data warehouse, you associate the cluster with the security group. Making your Amazon Redshift data warehouse follow the least access security principle is a best practice.

Getting ready

To complete this recipe, you will need the following setup:

- An IAM user with access to Amazon VPC, Amazon EC2, and Amazon Redshift
- Access to any SQL interface, such as a SQL client or the Amazon Redshift query editor

How to do it...

In this recipe, you will launch an Amazon Redshift Provisioned cluster inside a custom VPC and subnet using the following steps:

1. Navigate to the AWS Console and select the **VPC** service. Click on **Launch VPC Wizard** and choose the default VPC with a Single Public Subnet option. Enter the following values and click on the **Create VPC** button:
    - **IPv6 CIDR block** - `Amazon provided IPv6 CIDR block`
    - **VPC name** – `vpc-redshift`
    - **Subnet name** - `subnet-redshift`
    - **Service endpoints** - `com.amazonaws.eu-west-1.s3`

Choosing the service endpoints from Amazon S3 allows the traffic to and from Amazon Redshift to be within the VPC rather than the default of via the internet.

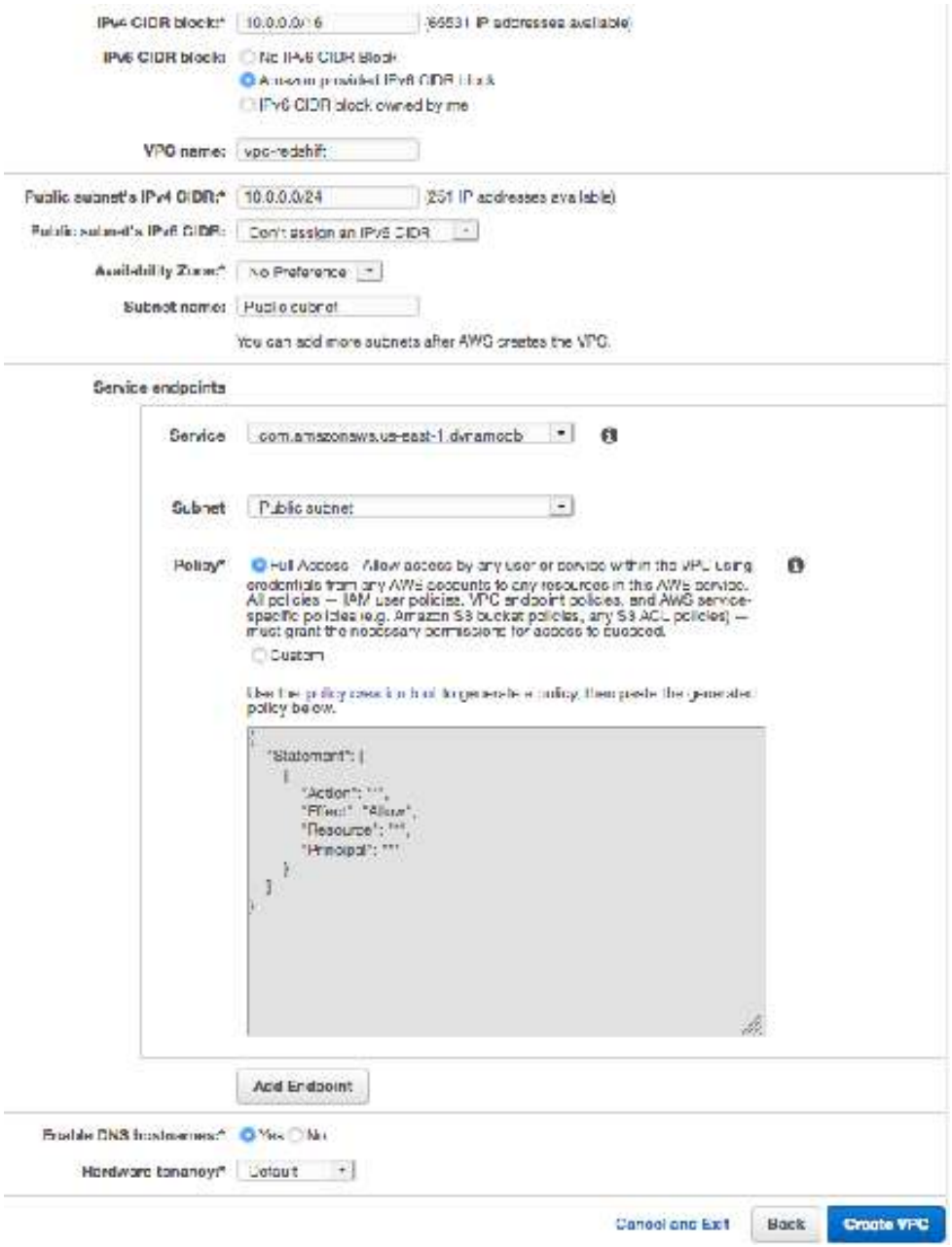


Figure 6.1 – Creating the VPC and subnet for Amazon Redshift

2. Navigate to **Your VPCs** in the left-hand menu and note the **VPC ID** associated with `vpc-redshift`. Click on **Security Group** in the left menu and click on the security group associated with the VPC ID. Click on **Inbound Rules**, remove the default rules selection, and choose **My IP**, as shown in the following screenshot:

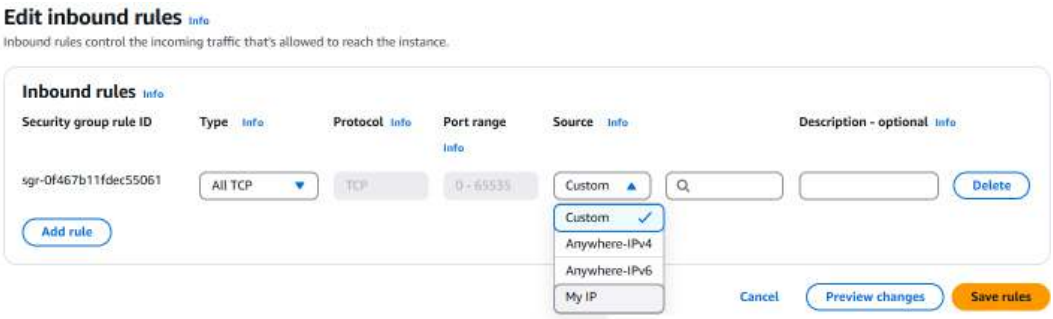


Figure 6.2 – Editing the inbound rules for the security group

In the list of **Inbound Rules**, instead of individual IP addresses, configuring the CIDR IP ranges allows flexibility for allowing connections within your organization.

Note

You can learn more about setting up VPC by using this guide: `https://docs.aws.amazon.com/vpc/latest/userguide/working-with-vpcs.html#add-ipv4-cidr`.

3. Navigate to the AWS Amazon Redshift console, click on the **Config** menu, and choose **subnet groups**. Click on **Create subnet group**, choose `vpc-redshift`, add all the subnets in this VPC, provide a friendly description, and click on **Create cluster subnet group**, as shown in the following screenshot:

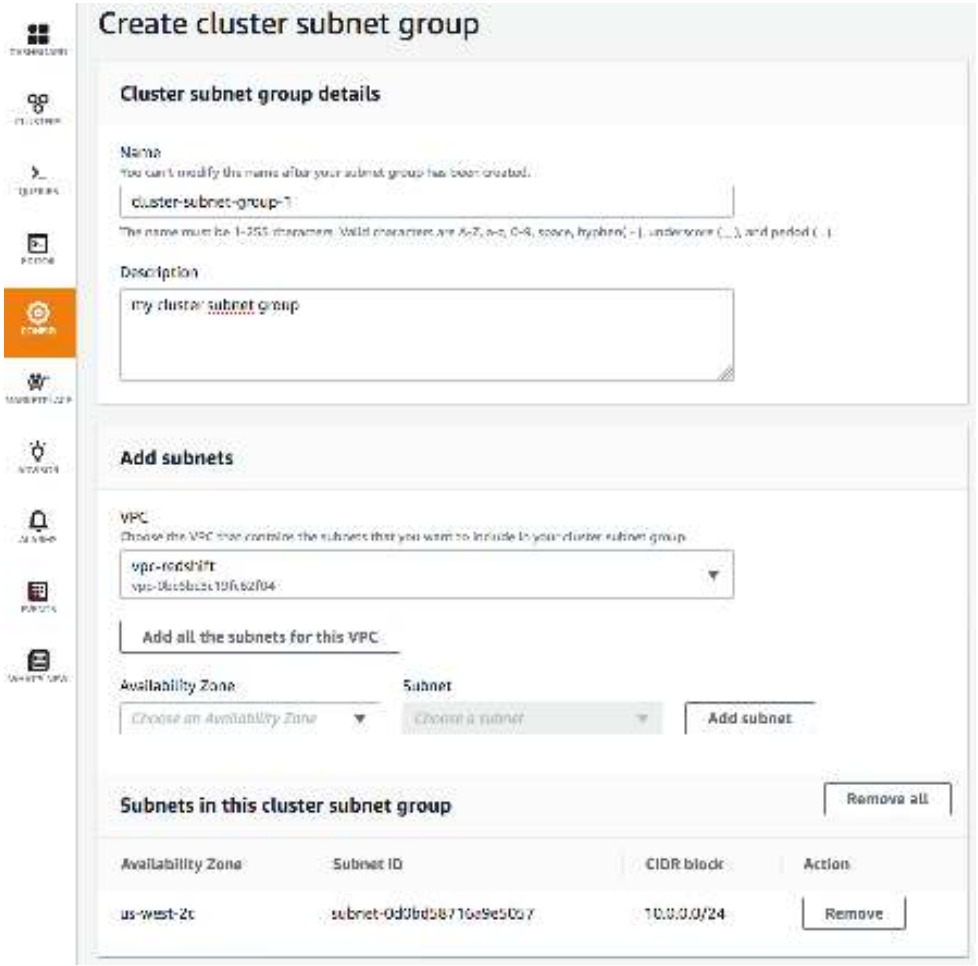


Figure 6.3 - Creating a subnet group for Amazon Redshift

4. Click on the **cluster** menu and navigate to **Amazon Redshift > Clusters > Create cluster**. Navigate to the **Additional Configurations** section and toggle off the **Use default** option. Choose `vpc-redshift` in the **VPC**, as shown in the following screenshot, and click on **Create cluster**:

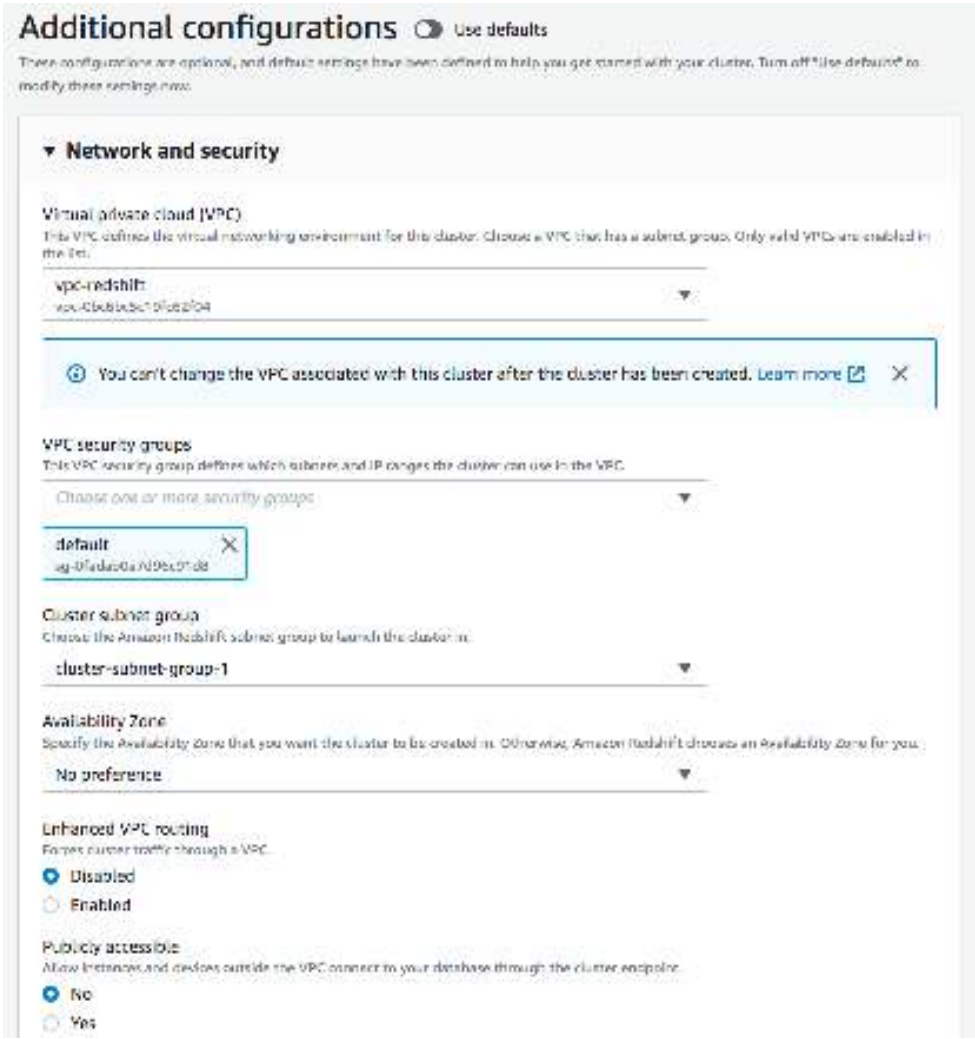


Figure 6.4 – Configuring the network and security when creating the Amazon Redshift provisioned cluster

5. Connect to the SQL client using the `masteruser` credentials to verify the connection. Refer to the *Connecting to Amazon Redshift using SQLWorkbench/J client* section in *Chapter 1*, *Getting Started with Amazon Redshift*, for step-by-step instructions.

Data encryption at rest

Amazon Redshift by default provides the option to encrypt the cluster at rest using the AES algorithm with a 256-bit key. Key management can be performed by AWS KMS or your hardware security module. When an Amazon Redshift data warehouse is encrypted at rest, it provides block-level encryption. When the data warehouse is encrypted the metadata, snapshots, and recovery points are also encrypted. This enables you to meet your security requirements to comply with PCI, SOX, HIPAA, or GDPR, depending on your needs. Amazon Redshift serverless clusters on creation require encryption using a default or customer-managed key.

Amazon Redshift uses envelope encryption using a robust four-tier hierarchy of encryption keys: master key, cluster encryption key, database encryption key, and data encryption key.

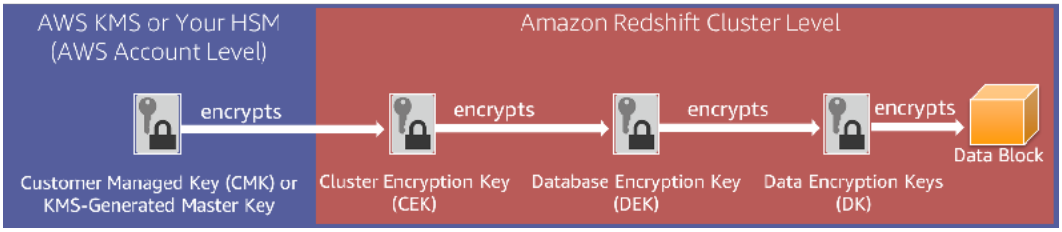


Figure 6.5 – Amazon Redshift encryption

Getting ready

To complete this recipe, you will need the following setup:

- An IAM user with access to Amazon KMS and Amazon Redshift
- Reference for encryption at rest in the AWS documentation: `https://docs.aws.amazon.com/redshift/latest/mgmt/working-with-db-encryption.html`
- Reference for the AWS CLI for Redshift: `https://docs.aws.amazon.com/cli/latest/reference/redshift/index.html`
- Reference for the Amazon Redshift API: `https://docs.aws.amazon.com/redshift/latest/APIReference/Welcome.html`

How to do it...

In this recipe, we will see how to encrypt a new and an existing Amazon Redshift Provisioned cluster:

Let's see how to turn on encryption while creating an Amazon Redshift provisioned cluster.

Navigate to the Amazon Redshift console and choose **Create cluster**. Scroll to **Additional configurations** and toggle the defaults. This will allow you to expand **Database configurations**. You have two options to choose from: an AWS managed key or a customer-managed key. When you choose an AWS managed key, you have the option to use the default Redshift key or use a key from an existing AWS account.

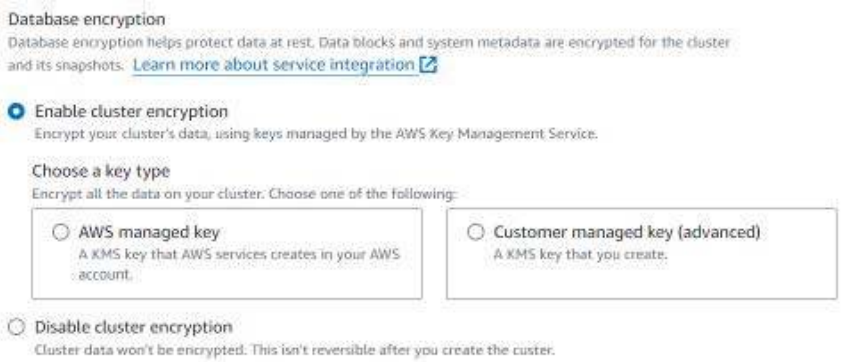


Figure 6.6 – Enabling AWS KMS encryption in Amazon Redshift

You can also create a cluster with encryption using the AWS CLI or an Amazon Redshift API call.

Let's see how to turn on encryption for an existing Amazon Redshift provisioned cluster:

1. Navigate to the Amazon Redshift console. Choose your provisioned cluster. Choose **properties** and select **Edit encryption**:

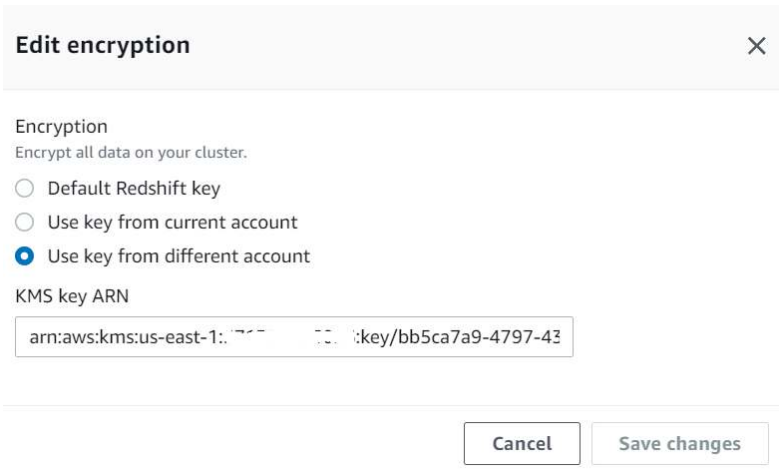


Figure 6.7 – Modifying encryption for an existing Amazon Redshift Provisioned cluster

The AWS CLI and the Amazon Redshift API support conversion to a KMS encrypted cluster.

2. Using the Amazon Redshift console, navigate to the existing Amazon Redshift cluster. Choose **Action** and select **rotate encryption**. You will see the following dialog box:

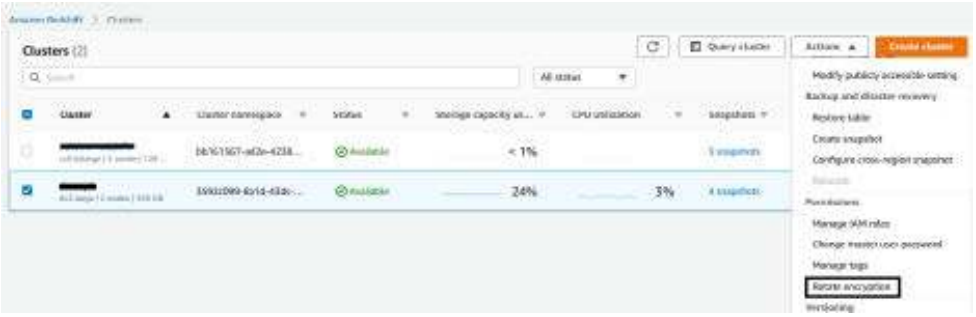


Figure 6.8 – Rotating encryption

You will see the following dialog box. Amazon Redshift will rotate the CEK for the cluster and the snapshot. The data encryption key for the cluster will change, but the **data encryption key** (**DEK**) cannot be changed for the snapshots that are on S3. During key rotation, the cluster is put in the `ROTATING_KEY` state until Amazon Redshift decrypts and re-encrypts the data. You can set the frequency of rotation to meet your organizational needs. You can balance the plan of rotating the keys with the availability considerations for your cluster.

Figure 6.9 – Amazon Redshift rotating the AWS KMS keys

You can rotate the encryption keys using the AWS CLI and the Amazon Redshift API.

Data encryption in transit

With Amazon Redshift, you can encrypt your data in transit. Enabling the **Secure Sockets Layer (SSL)** allows SQL clients to encrypt data in transit using certificates. In addition, the AWS CLI, SDK, or API client can communicate using HTTS endpoints. For communication between AWS services such as Amazon S3 and DynamoDB, Amazon Redshift uses hardware-accelerated SSL.

Getting ready

To complete this recipe, you will need:

- IAM user with access to Amazon Redshift.
- Download the JDBC driver from `https://docs.aws.amazon.com/redshift/latest/mgmt/configure-jdbc-connection.html`.
- A SQL client using a JDBC or ODBC connection. This recipe uses SQL workbench/j: `http://www.sql-workbench.net/`.
- Create a new parameter group for your Amazon Redshift Provisioned cluster: `https://docs.aws.amazon.com/redshift/latest/mgmt/managing-parameter-groups-console.html`.

How to do it...

In this recipe, we will enable SSL connectivity in Amazon Redshift and the SQL Workbench client:

1. To configure your Amazon Redshift provisioned cluster to require an SSL connection, navigate to the Amazon Redshift console. Choose your Amazon Redshift cluster and select the **Properties** tab. Scroll to database configuration and select the parameter group:

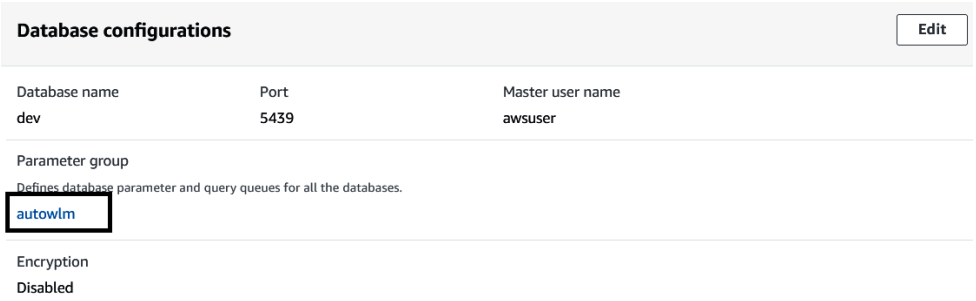


Figure 6.10 – Pick the parameter group associated with your Amazon Redshift cluster

2. Click on the parameter group, which will bring you to the workload management configuration page.
3. Set **require_ssl** to `true`. Choose **Save**. Navigate to the Redshift cluster. When the cluster is in the pending-reboot state, reboot the cluster by selecting **reboot under action**.

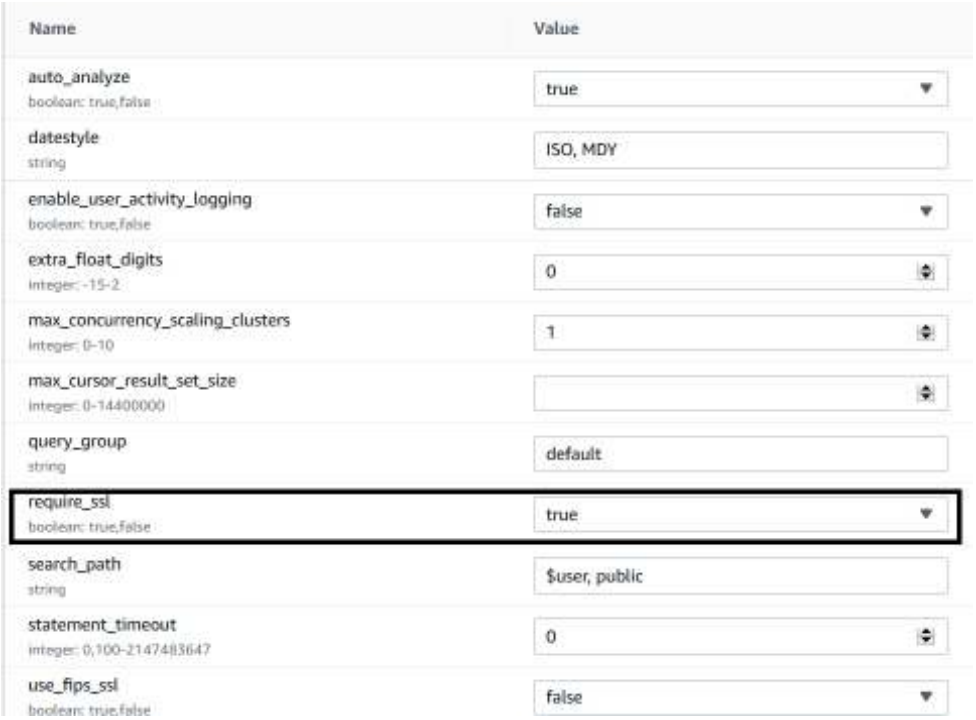


Figure 6.11 – Enabling the require_sql parameter in the parameter group

4. When **require_ssl** is set to `true`, Amazon Redshift accepts connections that are TLS encrypted. When **sslMode** is set to `verify-ca`, then the server is verified by checking the certificate chain up to the root certificate bundled with the Amazon Redshift JDBC/ODBC driver. When **sslMode** is set to `verify-full`, the server hostname provided in the connection will be checked against the name stored in the server certificate. If hostname matches the names stored in the server certificate, the connection is successful, otherwise it will be rejected.

5. Connect to Amazon Redshift Provisioned cluster using your SQLclient. This recipe is using **SQLWorkbench/j**. Get the cluster connection JDBC URL from the cluster's connection details on the properties tab. We are using **sslMode=verify-full**.

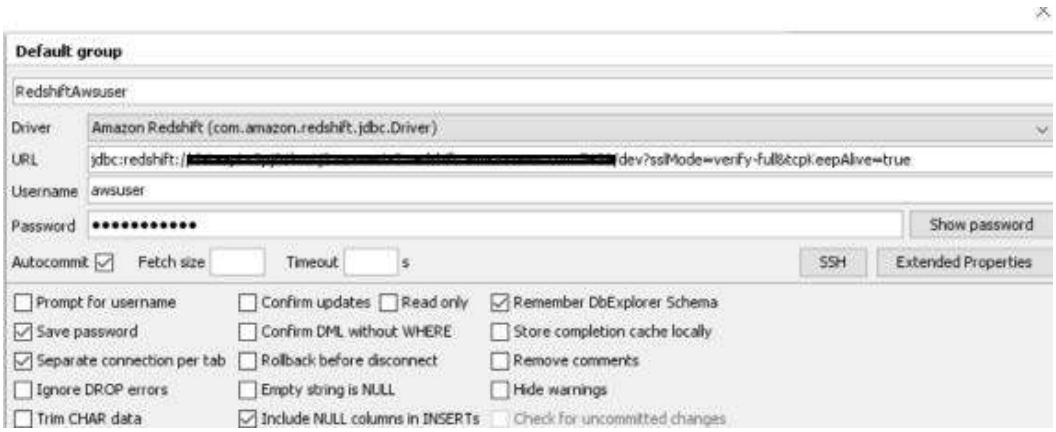


Figure 6.12 – Connecting to Amazon Redshift using SQL Workbench and SSL

6. Let's check whether the connection is using **sslMode**. Run the following code:

```
select * from stl_connection_log
order by recordtime desc
limit 2;
```

Here is the output of the preceding code:

Figure 6.13 – Verifying the SSL connection using the STL_CONNECTION_LOG

We have now successfully connected to Amazon Redshift using a TLS-encrypted connection.

Note

For an Amazon Redshift serverless cluster to configure SSL, you can modify your workgroup using `require_ssl`. This will restart your workgroup to take effect:

```
aws redshift-serverless update-workgroup --workgroup-name
yourWorkgroupName --config-parameters parameterKey=require_
ssl,parameterValue=true
```

Managing superusers using an Amazon Redshift provisioned cluster

A superuser allows you to get complete access to Amazon Redshift, independently of all permission checks; this is used for administrative tasks. For example, you can create other users, execute diagnostic queries on system tables, and take action as needed. Superuser access has to be granted sparingly; do not use this for day-to-day work.

`masteruser` is a special type of superuser that you set up when launching the cluster.

Getting ready

To complete this recipe, you will need the following setup:

- An IAM user with access to Amazon Redshift
- An Amazon Redshift provisioned cluster deployed in the eu-west-1 AWS region
- Amazon Redshift provisioned cluster master user credentials
- Access to any SQL interface, such as a SQL client or the Amazon Redshift query editor V2

How to do it...

This recipe will illustrate how to create a superuser, use it to list all the active SQL statements, and terminate a particular statement:

1. Connect to Amazon Redshift using the SQL client using the `masteruser` credentials and execute the following statement to create another superuser, replacing `[masteruser_password]` with the password of your choice:

```
create user myadmin createuser password '[masteruser_password]';
```

2. If you have forgotten the `masteruser` credentials, you can navigate to the Amazon Redshift AWS Console, click on your `cluster-id` (**Amazon Redshift** -> **Clusters** -> **YOUR_CLUSTER**), click on **Actions,** and click on **Change master user password** to reset it.
3. Now, use the `myadmin` superuser to reconnect to Amazon Redshift using the SQL Workbench/J client. Execute the following statement to see a list of all the **Running** SQL statements:

```
SELECT pid,
       TRIM(user_name),
       starttime,
       duration,
```

```
       SUBSTRING(query,1,50) AS stmt
FROM stv_recents
WHERE status = 'Running';
```

Here is the expected output:

```
pid    btrim    starttime         duration             stmt
18764    user_a    2021-03-28 18:39:49    3000  select part_id,
seller_id
18790    user_b    2021-03-28 18:39:49    60    Insert into parts(
```

Queries from `user_a` are taking over 3,000 seconds to execute and probably consume resources (this can be confirmed using the AWS Console), and we assume you would like to terminate this query.

4. Execute the following statement to terminate the query with `pid = 18764`:

```
set query_group to 'superuser';
cancel 18764;
```

5. Using the optional `query_group` to `'superuser'` allows access to a special superuser queue, which makes the query execute immediately.

See also...

To learn more about WLM queue assignment rules please refer to (`https://docs.aws.amazon.com/redshift/latest/dg/cm-c-wlm-queue-assignment-rules.html`).

Using IAM authentication to generate database user credentials for Amazon Redshift serverless clusters

Amazon Redshift allows us to programmatically generate temporary database user credentials that can be used for automated scripts to connect to the cluster. Using the `get-credentials` command in the AWS CLI and `GetCredentials` in the API, you can generate temporary credentials that can then be used in JDBC and ODBC.

Getting ready

To complete this recipe, you will need the following setup:

- An IAM user with access to Amazon Redshift and AWS IAM.
- An Amazon Redshift serverless cluster deployed in the eu-west-1 AWS region. We will use the cluster ID as `[Your-Redshift_Cluster]`.
- Amazon Redshift serverless cluster master user credentials.
- Access to any SQL interface, such as a SQL client or the Amazon Redshift query editor V2.
- AWS CLI on your local client.

How to do it...

In this recipe, we will generate temporary credentials to connect to an Amazon Redshift provisioned cluster:

1. Open the Command line interface (CLI) tool on your local client where AWS CLI is configured. Type the following command to verify that the AWS CLI has been installed. This should show the help manual:

```
aws help
```

2. Execute the following command to generate temporary credentials for your Amazon Redshift serverless cluster. Remember to replace `[Your-Redshift_Cluster]` and `[Your-Redshift_DB]` with the respective values:

```
aws redshift-serverless get-credentials --workgroup cookbook  --db-
name dev
```

3. The preceding CLI command returns `dbuser` and `dbpassword`. This can be used to log in to Redshift. Credentials generated using the CLI are temporary:

```
{
    "dbPassword": "ENnq6ge8KlZsGSWMTIM8Ko7/Z13IrJvLJQC5t1jX7xNxxLvtm
j+dEiLzcQcurIjzPqFPjk6h/e6SbwVJ+2y9wDswyqzEgLrMdkS/EHX3kY+hruUPvzn6L
0hqn3Xf8vLVdAoJ0AMlsaRJ+j9fqQOvvgzmFsWxqN77uHs2wtfYE+9Xan9J+KLMtfDAOv
63wTAMeGZiqcfM09bovSyWKFrsB2DFzbuhN/EWKL3+6njlq6LPZILKAxvKGhMUVktNX
jyPBETUUMUp604K/UTC9yvpb0J8nccdk3d4Tk4pOPafDFkkVcNqdoD90cv+9s46utk4
TQVhJ8JASsIkfLUlnNjjW7iAtzxGGHXJpJ1V7ENysTjD9grc7WX83KOPjiglrCyh+vit
9FwV9rpGUr7bNuqPYbWKVCPUdE1+nCqFveIqrfTcWsoxsGQWMKI407E1Q4JcA6TSsTxt
EUs8nShtpi947EmQuie+1swuUNroU7A4PCEDIry+831Brte3zHf/p8WCngWZDrxphjF
1N10VWp6V9yoQFhmidaXu5Rzdy1ka8OKvsMIaQ27S9csA3kntItwKZL7R8R/LEjcSFl
```

```
YQ4dBl4w0tjgxHpBkSYfK+QMKQ1//
y6fUlm0i5e504aVH0F/y0Wl2DvRKzXc4PE0zRTOiZHr2p/
FXCufWen3LFC0w0RFY4byWsgZXQwdD7nTefRowukr8qrmOo8oQYaG
xd9/0lf5vkf242x1vo1hAjymvAC2mAdpactPPSQqnwjWjt1Y49O0qPOzAu3jYe561tJw/
ZacOGcJQqsmICesK5jWYtMMqzIsnfChO3zAxBYA7V5r/21H3x2rycihX3woj9XK4LjKwP
Fhzc8LbHsuxz0qtBhsK9HaB4hOOA2wDz+logLJNGM/UFqBv2XMHdWHLKozqmWkrv4
PCjsuKA8vpvQ5DJkiHRw6LV1irYP1tvHuzZYI3CKRtlTWgWOa3yR7BvjkXaSRrfd9S/
1TBkRO++rvnoAWOg/8ZBlpRokkqBjMxgSMZhfmIeVTKY+o1kgT/
I50jRw5o0u0rs9hCbvj5jVD74UcX5EA0gRdU5nsmiNNr3LJzbwBP8sQJWKSLcR0g0bI
cnpS78Z6y84hkWOVQCclr/C1i7Pvowjygzlx9aq9B4Oz7DBJERx3/
NVY0vDauEo4bP/yAfSWwPjUQ/002WgebwZL/
X1HbZT35EAmNewCmPTs7M1WXF65M806WNqjJDTJFAzan7i8
vKSF2apO9+C7NKo2/gIkiS+fmlXEjrox9wWqsKhSsWayz/
moosrPc46O3H09QByf4NApV
vQ0kN7LxTdRDXlCa5i4O3BjaL1OL5Lc0Tw4oOXIqT8QpxP0O8B70VbuvSbGBq607Zza2s
3rhkfJSLgGQOAxKsIRz68J6EV+YJnZrz3OmmXQc7W79dwIqME8puStQi+6VefzGlptHTZR
xzikYzKNYLq+KFdmOejevThhmTNUOKeMaxzTSz/MK/SnsjbSg9DNDR7HvLbDs/
F4Md5tT/
sXCzpEuUd6E+Rm8YMGajp+dUMHn/
eGzGARW5LCI0ZRs6CtZ96TSMnuY2UGkmnArJe936ER
rt7o4oMwcW69nSYKJOnoKEVqWY9kjMcGQmp3wAewpXYB2M0lzxRTQ80jh/
YR8c6ibliiCfZtD6cjfg=",
    "dbUser": "IAMR:admin",
    "expiration": "2024-09-30T23:04:17.932000+00:00",
    "nextRefreshTime": "2024-10-01T00:49:17.932000+00:00"
}
```

Managing audit logs

Amazon Redshift allows you to log user activity, connections, and database operations by using audit logs. Audit logs are published asynchronously into Amazon S3 or to AWS CloudWatch. These logs are a way to monitor the requests to your clusters, which can be used for implementing security requirements and for troubleshooting purposes. For example, let's say that you want to find the user who might have truncated a particular table on a particular day in the past. Audit logs can uncover this information. Amazon Redshift provisioned clusters can be configured to send audit logs to either Amazon S3 or AWS CloudWatch (`https://docs.aws.amazon.com/redshift/latest/mgmt/db-auditing.html#db-auditing-cloudwatch-provisioned`). For Amazon Redshift serverless endpoints, audit logs can be sent to Amazon CloudWatch. (`https://docs.aws.amazon.com/redshift/latest/mgmt/serverless-audit-logging.html`).

Getting ready

To complete this recipe, you will need the following setup:

- An IAM user with access to Amazon Redshift and AWS Glue.
- An Amazon Redshift provisioned cluster deployed in the eu-west-1 AWS region. We will use the cluster ID as `[Your-Redshift_Cluster]`.
- Amazon Redshift provisioned cluster master user credentials.
- Access to any SQL interface, such as a SQL client or the Amazon Redshift query editor V2.
- An IAM role that can access Amazon S3. We will use it in the recipes as `[Your-Redshift_Role]`.
- An AWS account number. We will use it in recipes as `[Your-AWS_Account_Id]`.

How to do it...

In this recipe, we will illustrate how to turn on audit logging into Amazon S3 (it is turned off by default) and easily query it:

1. Connect to the AWS Amazon Redshift console and navigate to **Amazon Redshift | Clusters | [YOUR_CLUSTER]**. Click on the **Maintenance and monitoring** tab and scroll down to the **Audit Logging** option, as shown in the following screenshot:

Figure 6.14 – Enabling Amazon Redshift audit logging

2. Click on the **Edit** button **in Audit logging**, set **Enable audit logging** to `Yes`, and select (or create) an **Amazon S3 bucket**, as shown in the following screenshot:

Figure 6.15 – Configuring the target S3 buckets for logging

The previous option turns on connection logging, which will start capturing connection information such as client host IP and username, as detailed in `https://docs.aws.amazon.com/redshift/latest/mgmt/db-auditing.html#db-auditing-logs`. Logs will be delivered asynchronously hourly into the S3 prefix location.

3. Once the user connections are made in the Amazon Redshift cluster, connection logs are delivered to the previously specified target Amazon S3 location, which can be verified using the AWS Console for Amazon S3 or the AWS CLI using the `aws s3 ls [AWS S3 Target bucket]` command.

   The logs files are organized as `<AWS Account #>/redshift/<region>/<Year>/<Month>/<Day>/<Hour>`.

4. Create a new crawler named `audit_crawl` with a database called `audit_logs_db` and a table called `auditawslogs` by using the Amazon S3 location configured in the preceding step. Choose **Add crawler** under **Tutorials**. For step-by-step instructions on configuring the AWS Glue crawler, refer to *Chapter 10*, *Lakehouse Architecture*

5. Run the `audit_crawl` crawler. Once the crawler has finished, you will find a new table named `auditawslogs` under **Data Catalog | Databases | Tables**, as shown in the following screenshot:

Figure 6.16 – auditawslogs

6. Connect to the SQL client using the superuser credentials and create `audit_logs` schema pointing to the AWS Glue `audit_logs_db` database created previously:

```
create external schema audit_logs
from data catalog
database 'audit_logs_db'
iam_role 'arn:aws:iam::[Your-AWS_Account_Id]:role/[Your-Redshift_
Role]'
create external database if not exists;
```

7. Use the following query to retrieve audit information:

```
SELECT col0 AS event,
       col1 AS recordtime,
       col2 AS remotehost,
       col3 AS remoteport,
       col4 AS pid,
       col5 AS dbname,
       col6 AS username,
       col7 AS authmethod,
```

```
        col8 AS duration,
        col9 AS sslversion,
        col10 AS sslcipher,
        col11 AS mtu,
        col12 AS sslcompression,
        col13 AS sslexpansion,
        col14 AS iamauthguid,
        col15 AS application_name,
        col16 AS driver_version,
        col17 AS os_version,
        col18 AS plugin_name
FROM audit_logs.auditawslogs
WHERE partition_5 = 25
AND   partition_4 = 12
AND   partition_3 = 2020 LIMIT 10;
```

Here is the output of the preceding code:

```
event,recordtime,remotehost,remoteport,pid,dbname,username,authmethod,
duration,sslversion,sslcipher,mtu,sslcompression,sslexpansion,
iamauthguid,application_name,driver_version,os_version,plugin_name
authenticated     Fri, 25 Dec 2020 09:02:04:228    [local]
49050    dev     rdsdb     Ident     0              0
initiating session     Fri, 25 Dec 2020 09:02:04:228    [local]
49050    dev     rdsdb     Ident     0              0
disconnecting session     Fri, 25 Dec 2020 09:02:04:346    [local]
49050    dev     rdsdb     Ident     118856              0
authenticated     Fri, 25 Dec 2020 09:02:40:156    [local]
49238    dev     rdsdb     Ident     0              0
```

As observed in the preceding output, all the session activity is logged as part of audit logging and can be easily queried using SQL statements.

How it works...

Audit logs are also available in system log tables, `SYS_USERLOG` (`https://docs.aws.amazon.com/redshift/latest/dg/r_STL_USERLOG.html`) and `SYS_CONNECTION_LOG` (`https://docs.aws.amazon.com/redshift/latest/dg/r_STL_CONNECTION_LOG.html`), but retention is limited in system tables.

For longer retention and convenient sharing of the audit information, Amazon Redshift logs can be enabled, which asynchronously send logs to Amazon S3. The user activity log can be enabled by setting the `enable_user_activity_logging` parameter to true in the database parameter group in addition to the connection logs.

Monitoring Amazon Redshift

Monitoring the cluster performance metrics allows you to ensure the cluster is healthy. Amazon Redshift publishes metrics such as CPU, disk utilization, and query workloads continuously, which can be automatically monitored for anomalies to trigger notification events. Amazon Redshift publishes cluster performance metrics to AWS CloudWatch (`https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/WhatIsCloudWatch.html`) as well, which allows you to monitor all your AWS services in a centralized location.

Getting ready

To complete this recipe, you will need the following setup:

- An IAM user with access to Amazon Redshift and Amazon SNS
- An Amazon Redshift provisioned or serverless endpoint deployed in the eu-west-1 AWS region
- An Amazon SNS topic (called `AmazonRedshiftHealthNotification`) to receive the alarm notifications using `https://docs.aws.amazon.com/sns/latest/dg/sns-create-topic.html`

How to do it...

In this recipe, we will illustrate how to watch the cluster and query monitoring metrics and also set up a health check alarm:

1. Connect to the AWS Amazon Redshift console and navigate to **Amazon Redshift | Clusters | [YOUR_CLUSTER]**. Click on **Cluster performance** to view metrics such as CPU and disk utilization, as shown in the following screenshot:

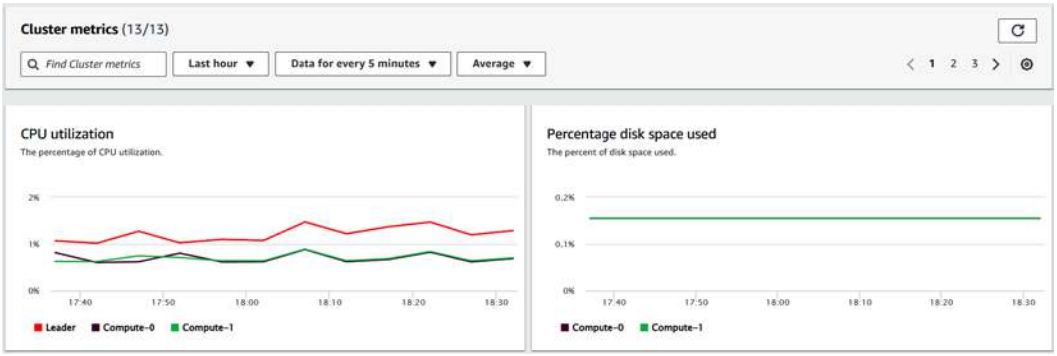


Figure 6.17 – Monitoring cluster performance

2. Click on the **Query Monitoring** tab, which displays the data warehouses's performance along with query history, a list of queries that are running/completed, along with the status of the queries:

Figure 6.18 – Data warehouse performance

Performance metric monitoring also allows us to get insights into the overall workload in the cluster. You can drill down into individual queries or look at users with the highest workload.

3. Choose Query History. Search by `query_id` or `sql` text. Select `query_id`, this will bring you to the query details. This view provides detailed insights into each query.

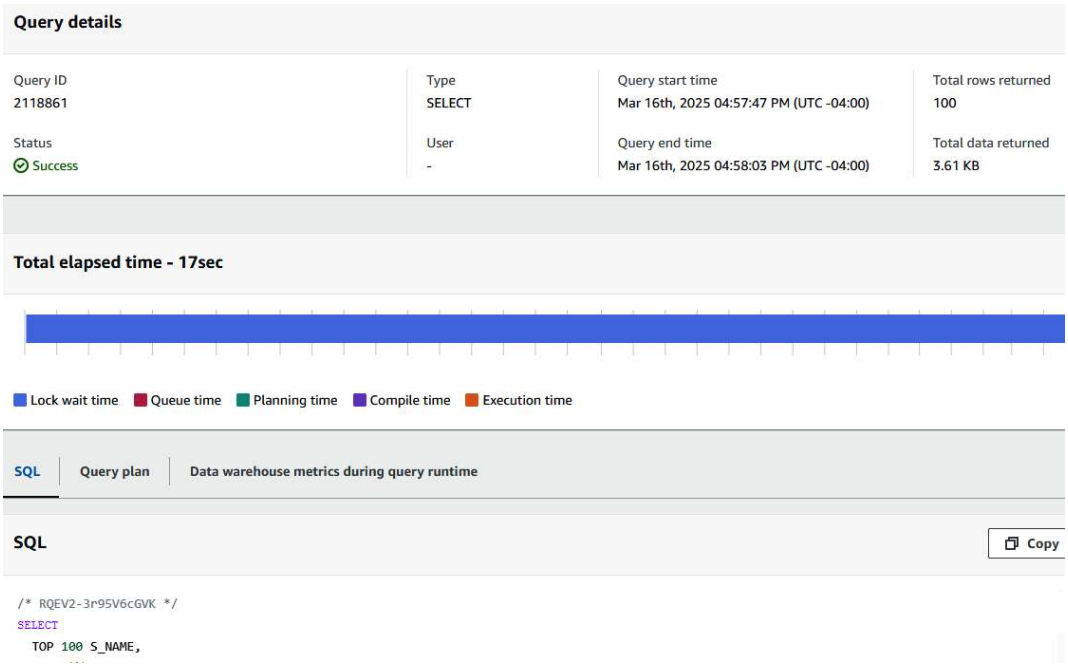


Figure 6.19 – Query details

4. Choose **Query plan. Select child query.** This will bring you to a detailed view of the query plan. This view gives insights into query performance and what streams in the query can be optimized. In this example, the query used a nested loop join, which is cross-joining two tables. This could be the result of overlooking the join condition in the query.

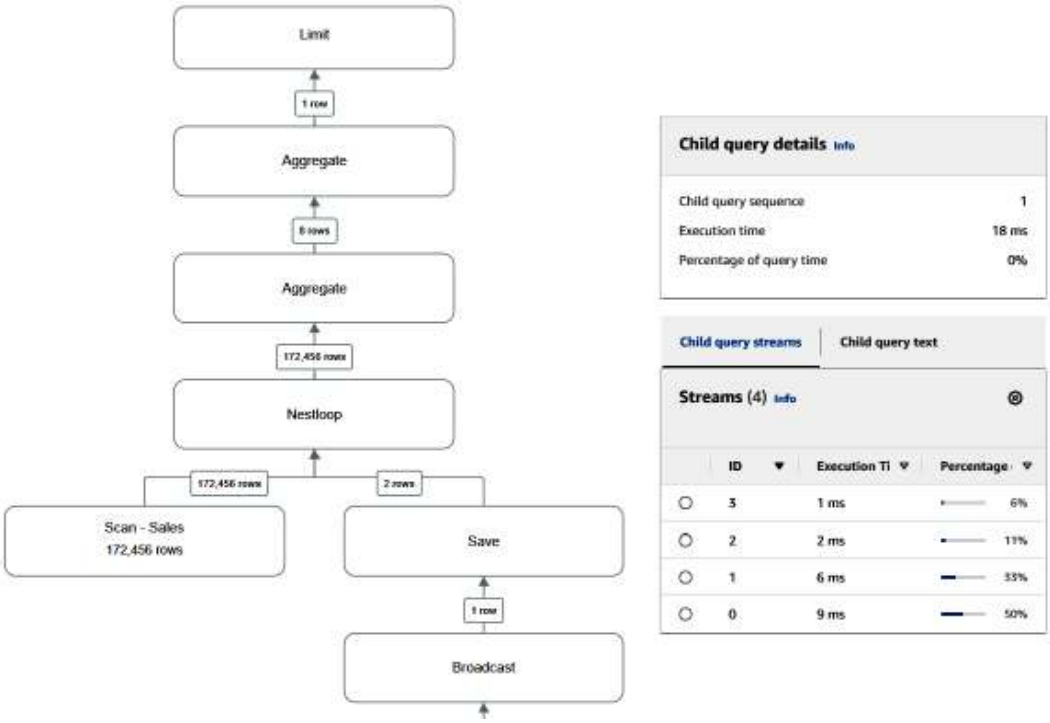


Figure 6.20 – Query plan performance breakdown

5. Click on **Amazon Redshift | Alarms | Create alarm** and choose the following options to set up a health check alarm for the cluster:
    a. **Cluster identifier**: Choose the Amazon Redshift cluster that you want to set up the alarm with
    b. **Alarm for metric**: Choose the maximum for all nodes
    c. **When metric value is**: Less than (<) `1`
    d. **If the alarm state is maintained for**: `10` consecutive periods of `5` minutes
6. In the alarm details, choose the following options:
    a. **Alarm name**: Any meaningful name for the health alarm
    b. **Notification**: Enabled
    c. **Notify SNS topic**: Select `AmazonRedshiftHealthNotification`
7. Click on **Create alarm** to complete the setup of the health check alarm.

How it works...

The health check alarm is a binary value, where `1` indicates a healthy cluster node and `0` indicates an unhealthy node. The health check alarm is monitoring for any value that is less than `1` for `10` consecutive instances for a duration of 5 minutes, at which point it will notify the SNS topic. Similarly, other performance metrics and be configured and notified when the thresholds are breached.

Single sign-on using AWS IAM Identity Center

AWS IAM Identity Center allows you to manage single sign-on access to your AWS accounts and applications from a single location. Amazon Redshift now integrates with AWS IAM Identity Center, supporting trusted identity propagation (`https://docs.aws.amazon.com/singlesignon/latest/userguide/trustedidentitypropagation-overview.html`) and the use of third-party identity providers for the authentication and authorization of Redshift users. A list of supported third-party identity providers can be found at `https://docs.aws.amazon.com/singlesignon/latest/userguide/tutorials.html`. IAM Identity Center allows automatic provisioning of users and groups from an identity provider (IdP) into IAM Identity Center using the SCIM v2.0 protocol. This integration ensures accurate and up-to-date user and group data is in AWS IAM Identity Center. This integration simplifies access to the Redshift data warehouse and enables the use of database role-based access control for enhanced security.

Getting ready

To complete this recipe, you will need the following setup:

- An IAM user with access to the Amazon Redshift console and AWS IAM.
- An Amazon Redshift serverless endpoint deployed in the eu-west-1 AWS region.
- Amazon Redshift serverless endpoint master user credentials.
- Access to the Amazon Redshift query editor V2.
- Enable IAM Identity Center. For more information, follow the steps at `https://docs.aws.amazon.com/singlesignon/latest/userguide/get-set-up-for-idc.html`.
- An Okta account that has an active subscription. You need an admin role to set up the application on Okta. If you're new to Okta, you can sign up for a free trial (`https://www.okta.com/free-trial`) or sign up for a developer account (`https://developer.okta.com/`). Create a user called `ethan.joe@mail.com` and a group called `awssso-idp-sales`.

- Connect IAM Identity Center with your preferred IdP and sync users and groups. For this recipe, we will use Okta. Follow the steps in `https://docs.aws.amazon.com/singlesignon/latest/userguide/gs-okta.html`.
- An AWS account number. We will reference it in the recipes as `[Your-AWS_Account_Id]`.

How to do it...

In this recipe, we will demonstrate how to use Okta identity provider with AWS IAM Identity Center to enable single sign-on access to Amazon Redshift in a single account setup. We will use the query editor v2:

1. Enable Amazon Redshift as an AWS-managed application with IAM Identity Center. In the Redshift console, on the left navigation pane, choose IAM Identity Center connections (`https://us-west-2.console.aws.amazon.com/redshiftv2/home?region=us-west-2#/serverless-iam-idc-integration`).

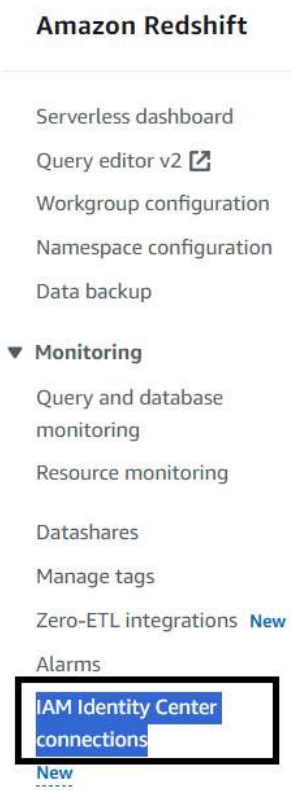


Figure 6.21 – Redshift console IAM Identity Center connections

2. Choose **Create Application**, then select **Next:**

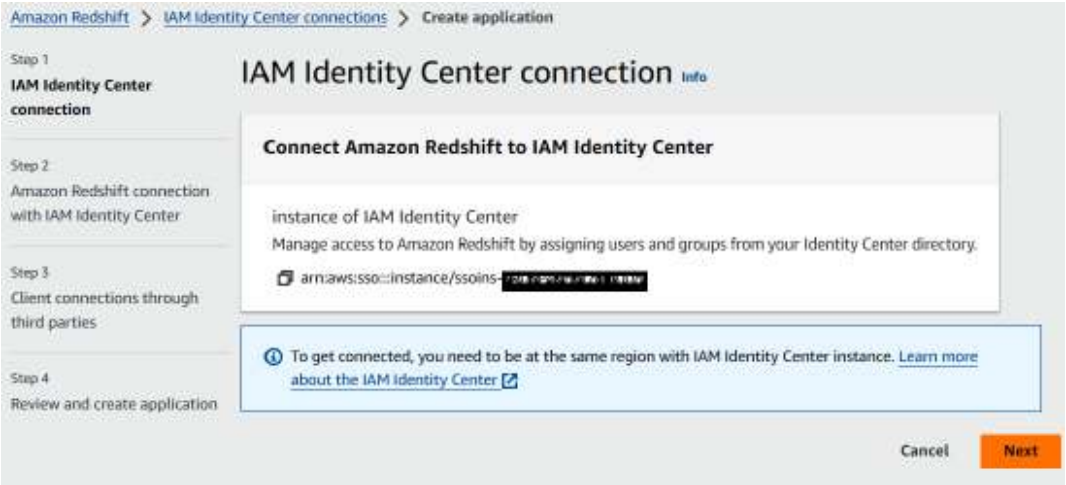


Figure 6.22 – Redshift console IAM Identity Center connection

3. The IAM Identity Center Redshift connection setup has three sections:

    a. Connection properties: For the IAM Identity Center display name, use **redshift-idc-app**. For the Managed application name, use the default name.

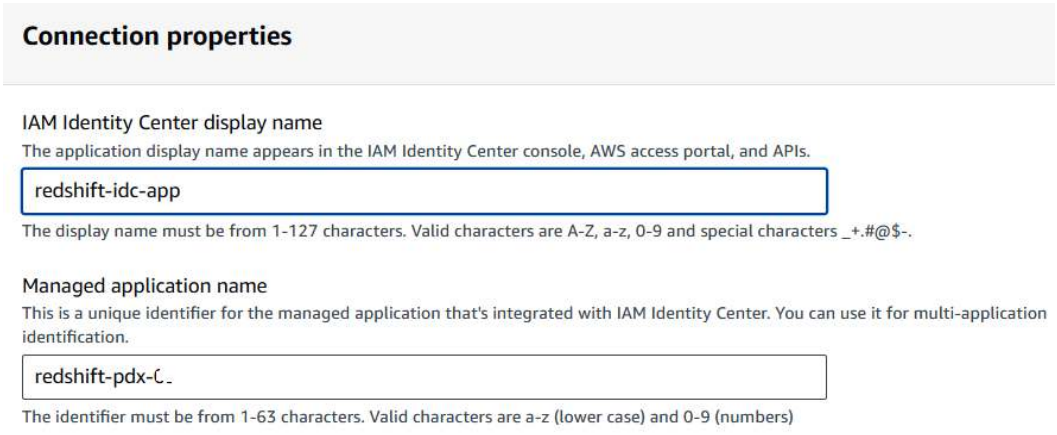


Figure 6.23 – Redshift connection with IAM Identity Center

    b. For Connection with third party identity providers, use **awsidc.**

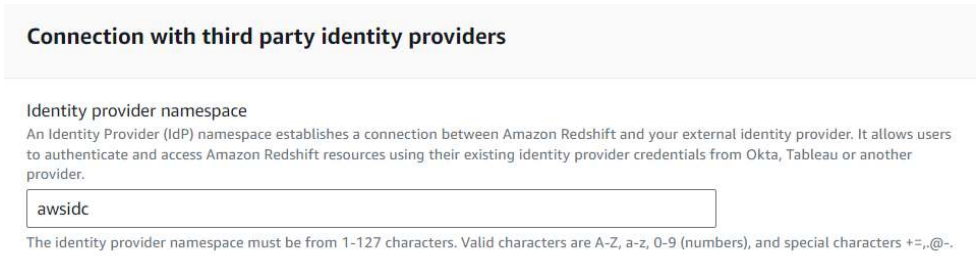


Figure 6.24 – Connection with third party identity providers

c. For IAM role for IAM Identity Center access, select the Redshift role.

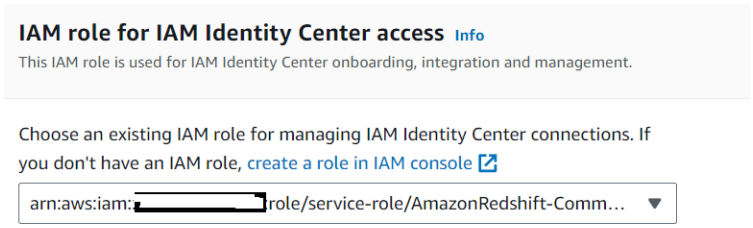


Figure 6.25 – IAM role for IAM identity Center access

4. Select **Enable the query editor V2** application to allow IAM Identity Center users and groups to connect to Redshift and run queries based on the provided permissions. **Select AWS Lake Formation access** grants for trusted identity propagation.

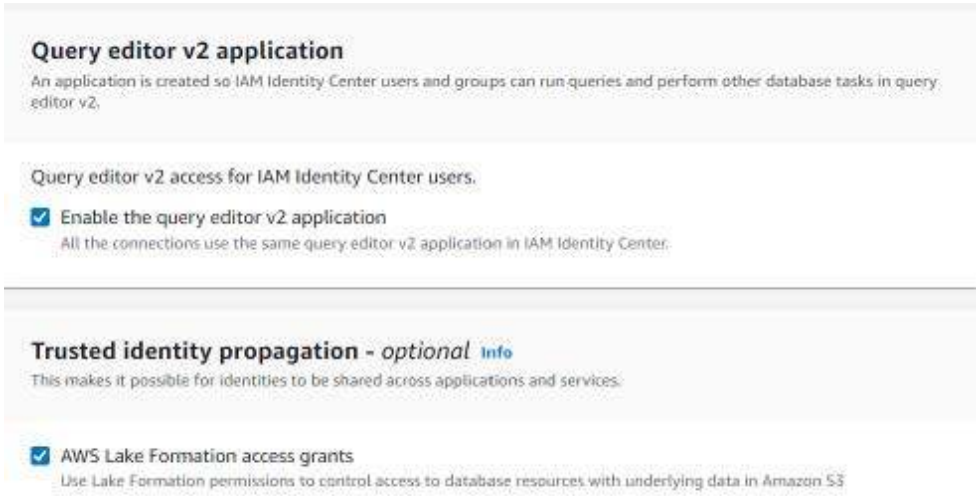


Figure 6.26 – Enable Redshift query editor V2 and trusted identity propagation

5. Choose **Next**
6. As for this recipe we will be using query editor V2 choose **No**

Figure 6.27 – Client connections through third-party IdPs

7. Choose **Next.**
8. Validate the configuration and choose **Create application**.
9. With a successful setup, you will see that the **status** is now connected for the application.

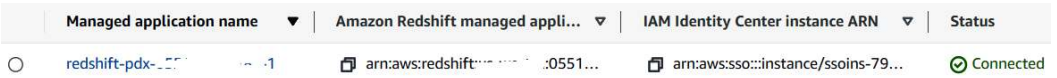


Figure 6.28 – Successful setup of the Redshift managed application

10. Choose redshift-integration-app. Save the IAM Identity Center managed application ARN. This will be referred to as `[IDC-APPLICATION-ARN]`.

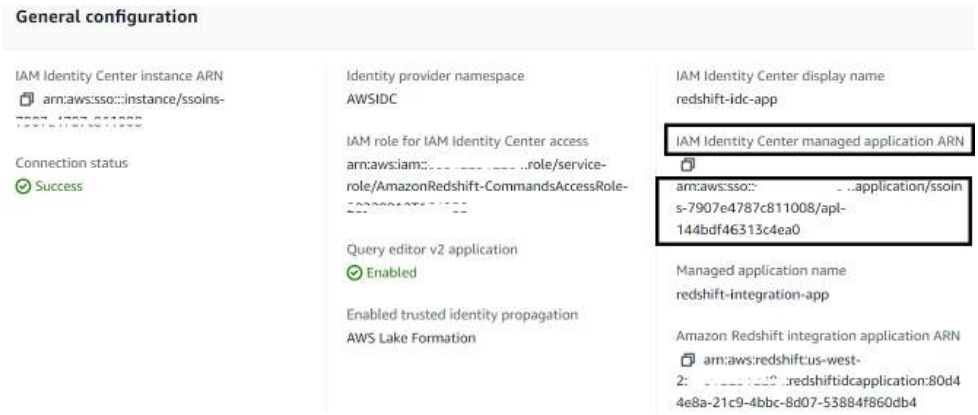


Figure 6.29 – Managed application configuration

11. Assign users or groups from **IAM Identity Center** (**IDC**) to the Redshift application. On the Redshift console, choose the managed application that was created in the previous step. This configuration allows you to add all or selected groups from IAM Identity Center to the Redshift application. This allows you to control who should get access to connect to the Redshift data warehouse. Choose **Groups**.

Figure 6.30 – Select groups to add to Redshift managed application

12. Choose Assign. Enter the name of the group, `awssso-idp-sales`, and select the group.

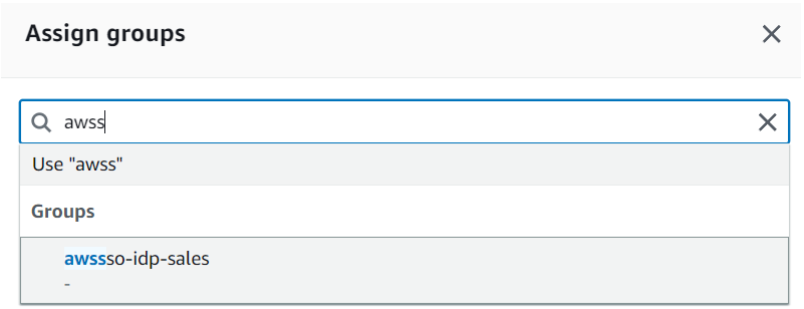


Figure 6.31– Add groups to the Redshift managed application

13. Choose **Assign.**

Figure 6.32 – Sales group addition to Redshift managed application

Let's now associate an IAM Identity Center application with a Redshift serverless endpoint:

1. Connect to the Redshift serverless endpoint using the SQL client or query editor v2. Connect using admin credentials.

2. Enter the following code to create the integration:

```
CREATE IDENTITY PROVIDER "redshift-idc-app" TYPE AWSIDC
NAMESPACE 'awsidc'
APPLICATION_ARN 'IDC-APPLICATION-ARN]'
IAM_ROLE 'arn:aws:iam::[Your-AWS_Account_Id]:role/[Your-Redshift_
Role]';
```

3. To check that the identity provider has been created, you can run the following code:

```
select * from svv_identity_providers;
```

4. We will now create Redshift roles corresponding to the names of the IAM Identity Center groups that were synched from Okta. Use the following code to create a role in Redshift. Here, the awsidc namespace is the prefix for the group name in IAM Identity Center:

```
create role "awsidc:awssso-idp-sales";
```

5. Set up the schema and tables for this recipe. Use the following code to create the `store_sales` schema and table:

```
create schema sales_schema;
CREATE TABLE IF NOT EXISTS sales_schema.store_sales
(
  id INTEGER   ENCODE az64,
  produce_name varchar(20),
  purchase_amount INTEGER   ENCODE az64
)
DISTSTYLE AUTO
;
Insert into sales_schema.store_sales values (1,'product1',1000);
Insert into sales_schema.store_sales values (2,'product2',2000);
Insert into sales_schema.store_sales values (3,'product3',3000);
Insert into sales_schema.store_sales values (4,'product4',4000);
```

6. Grant access to the `store_sales` schema and table to the `"awsidc:awssso-idp-sales"` Redshift role:

```
grant usage on schema sales_schema to role "awsidc:awssso-idp-
sales";
grant select on all tables in schema sales_schema to role
"awsidc:awssso-idp-sales";
```

We will now create a permission set to grant console access to the Amazon Redshift query editor V2 application:

1. **Navigate** to the IAM Identity Center console. From the left navigation pane, choose **Permission sets**.
2. Choose **create permission**. Then choose **Custom permission set**. Choose **next**.

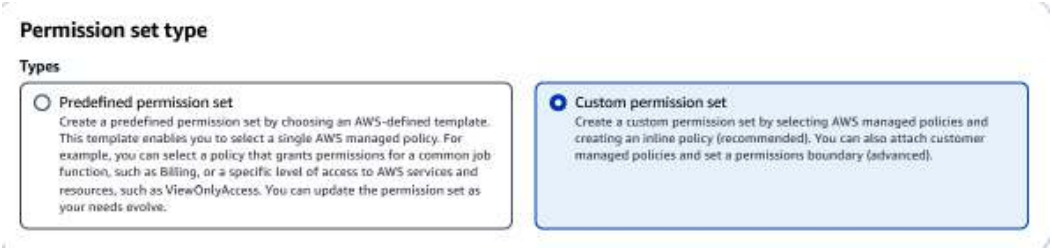


Figure 6.33 – Custom permission setup

3. Expand the custom managed policy and enter the name **Chapter6RedshiftPolicy**, which is the policy created using the CloudFormation template in *Chapter 6*.

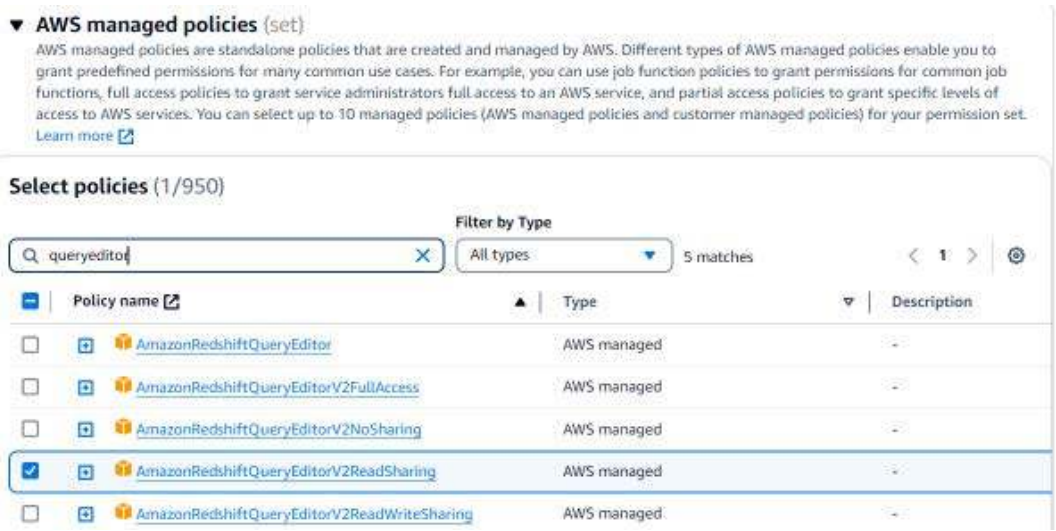


Figure 6.34 – AWS Managed Policy Redshift query editor V2 read sharing

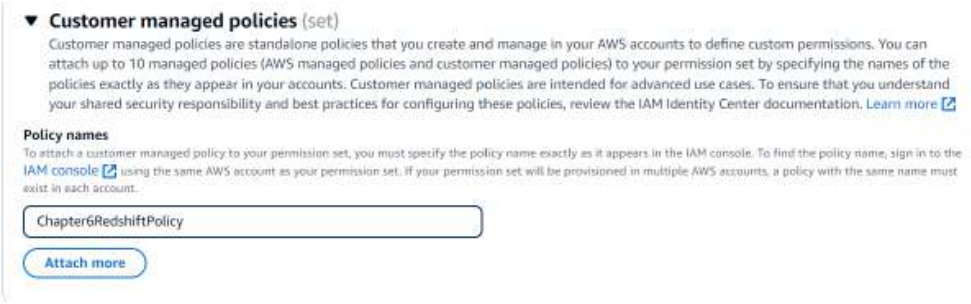


Figure 6.35 – Customer managed policy

4. Choose Next.
5. Enter the permission set name redshift-permission-set and put `https://eu-west-2.console.aws.amazon.com/sqlworkbench/home` in the relay state field. The following screenshot shows the values:

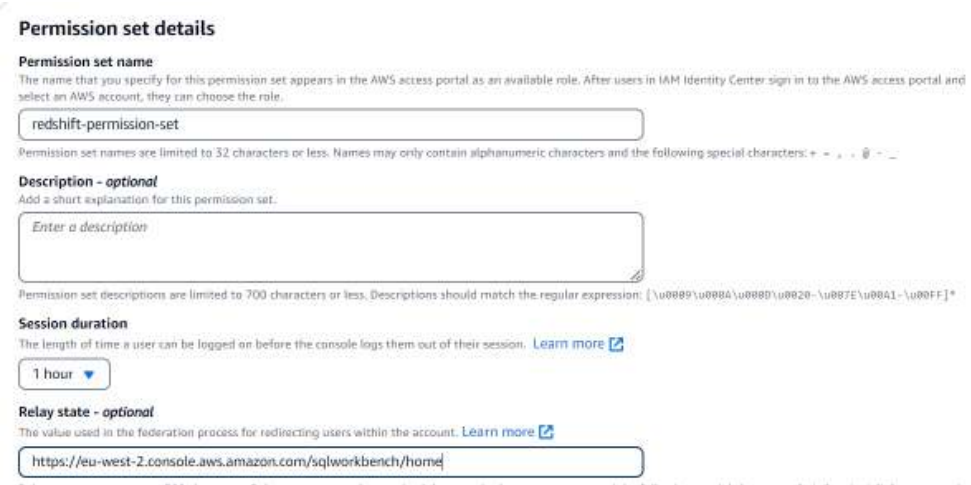


Figure 6.36 – Permission set

6. Choose Next. Review the permission set configuration and choose Create.

Let's assign IAM Identity Center groups to accounts:

1. From the left navigation pane of the IAM Identity Center, under **Multi-account permissions**, choose **AWS accounts**.
2. Select the account you want to assign single sign-on to. Then, choose **Assign users or groups**.

3. Choose `awssso-idp-group`. Choose **Next**.

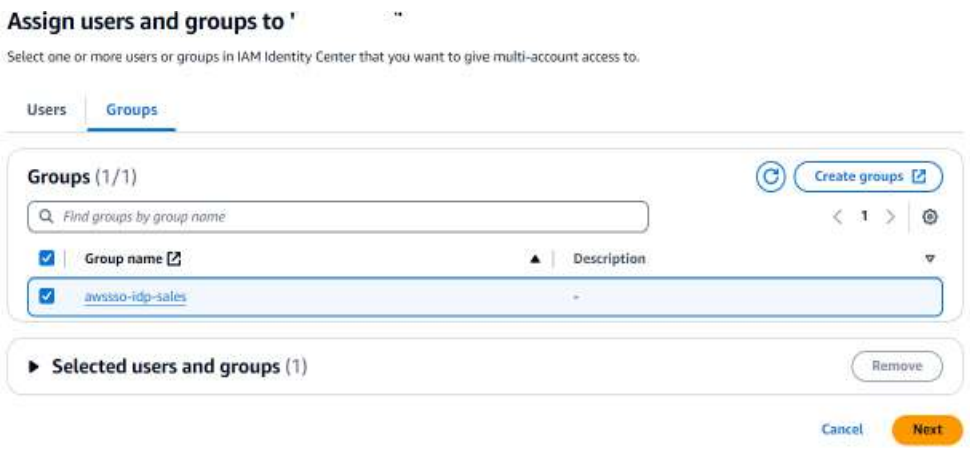


Figure 6.37 – Assigning users and groups to an AWS account

4. Choose the redshift-permission-set permission set. Choose **Next**.
5. Choose submit. This step completes the setup.

Federate access to Amazon Redshift query editor V2:

1. Navigate to the IAM Identity Center console. Choose dashboard.
2. Choose the **AWS access portal URL** from **Settings summary** on the right pane.

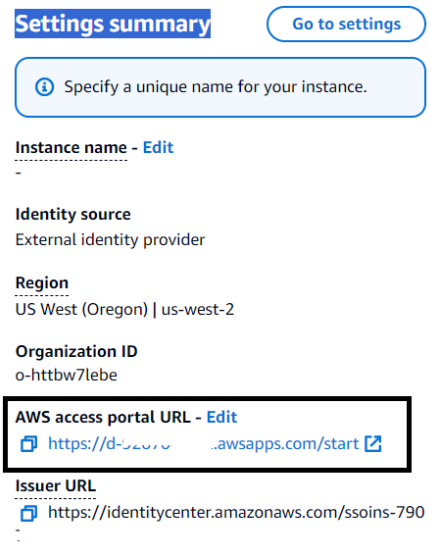


Figure 6.38 – Access portal URL from IAM Identity Center

3. This will open a browser window. Type in the username `ethan.joe@mail.com`, enter the credentials, and choose **Sign in**.

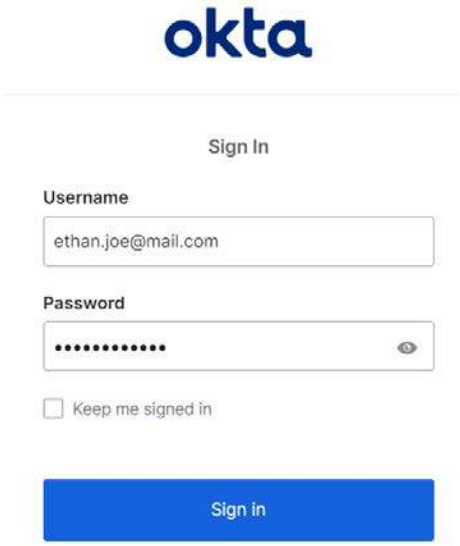


Figure 6.39 – Authentication using Okta sign in

4. Expand the account and choose **redshift-permission-set**. This will take you to Redshift query editor v2.
5. Choose your Redshift serverless endpoint and edit the connection.
6. Choose **IAM Identity Center**.

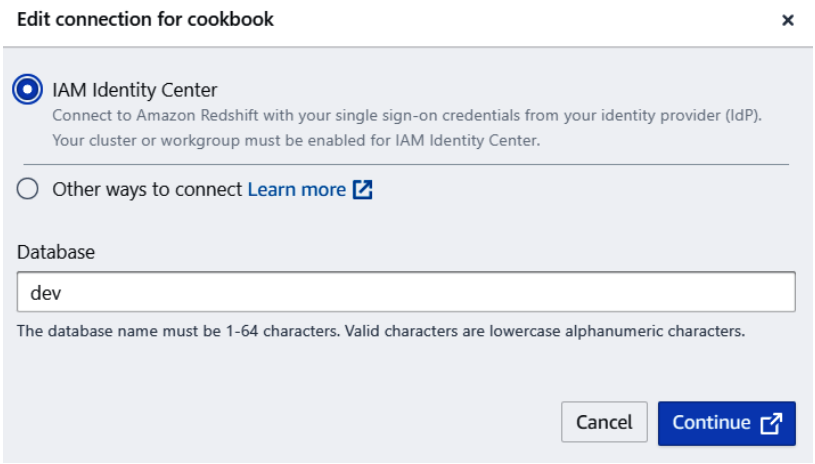


Figure 6.40 – Redshift query editor v2 connection

7. Run the following code to check the current user:

```
select current_user;
```

current_user
awsidc:ethan.joe@mail.com

Figure 6.41 – Current user output

8. Run the following code to select the `sales_schema.store_sales` table:

```
select * sales_schema.store_sales;
```

id	produce_name	purchase_amount
1	product1	1000
2	product2	2000
3	product3	3000
4	product4	4000

Figure 6.42 – Query output

9. Let's try to delete a record. This will produce a permission denied error as the user does not have delete access:

```
delete  from sales_schema.store_sales where id = 1;
Output:
ERROR: permission denied for relation store_sales
```

How it works...

By turning on the metadata security flag for a database, you can restrict the listing of metadata objects to only the objects that the user or role has access to. This enables you to hide metadata from one tenant when the schema names and table names include tenant names in order to comply with privacy controls.

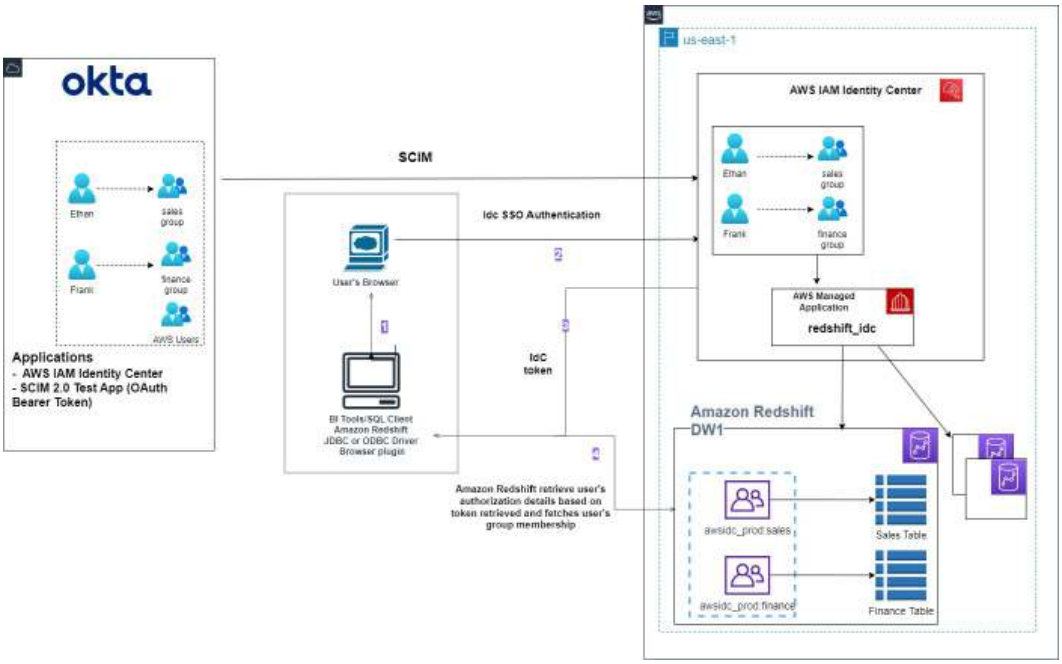


Figure 6.43 – Redshift query editor V2 workflow with IAM Identity center and an external identity provider

Access the AWS IAM Identity Center dashboard in the Management Account to get the URL for the AWS access portal. Use the URL to go to the AWS access portal. A browser popup will appear, prompting you to enter your IdP credentials. After successful authentication, you'll be logged into the AWS Console as a federated user. Choose your AWS account and choose the Amazon Redshift query editor v2 application. In the query editor v2, select the IAM Identity Center authentication method.

Query editor v2 invokes the browser flow, where you re-authenticate using your IdP credentials. You have already entered your IdP credentials, which are cached in the browser. At this step, the federation flow with IAM Identity Center initiates, and at the end of this flow, the session token and access token are available to the query editor v2 console in the browser as cookies.

Redshift retrieves your authorization details based on the session token retrieved and fetches the user's group membership from IAM Identity Center.

You are redirected back to query editor v2 upon successful authentication, logged in as an IAM Identity Center authenticated user.

See also...

If you have a multi-account setup with management and member accounts, take a look at this blog: `https://aws.amazon.com/blogs/big-data/integrate-identity-provider-idp-with-amazon-redshift-query-editor-v2-and-sql-client-using-aws-iam-identity-center-for-seamless-single-sign-on/`.

Metadata security

Metadata security (`https://docs.aws.amazon.com/redshift/latest/dg/t_metadata_security.html`) in Amazon Redshift allows more granular control over who can view metadata about data objects such as tables, views, and functions. When enabled, users can only see metadata for objects they have access to. This metadata security feature can be beneficial in scenarios where a single data warehouse holds data for multiple tenants, and tenant-specific schema or table names are used. By restricting metadata visibility between tenants, you can prevent tenants from viewing the metadata of other tenants. This helps maintain the privacy and separation between different tenants occupying the same data warehouse infrastructure. Metadata security supports all data object types and can be enabled or disabled using the ALTER SYSTEM command. Granting the ACCESS CATALOG permission to a role is one way to control who can view the data.

Getting ready

To complete this recipe, you will need:

- An Amazon Redshift data warehouse deployed in the eu-west-1 AWS region
- Amazon Redshift data warehouse master user credentials
- Access to any SQL interface, such as a SQL client or the Amazon Redshift query editor v2

How to do it...

In this recipe, we will use the multi-tenant scenario to permit users to only see the metadata of objects the user has access to:

1. Connect to the Amazon Redshift data warehouse using the SQL client or Redshift query editor v2. Create a database called `metadata_security` using the following code:

```
Create database metadata_security.
```

2. Connect to the `metadata_security` database and create `schema and tables for tenant1 and tenant2` using the following code:

```
Create database metadata_security.
create schema tenant1_schema;
create table tenant1_schema.tenant1
(tenant1_col1 varchar(10)
);
create schema tenant2_schema;
create table tenant2_schema.tenant2
(tenant2_col1 varchar(10)
);
```

3. Create users and roles using the following code:

```
create user tenant1_user with password 'Test1243!';
create user tenant2_user with password 'Test1234!';

create role tenant1_ro;
create role tenant2_ro;

grant role tenant1_ro to tenant1_user;
grant role tenant2_ro to tenant2_user;
```

4. Grant tenant1_ro access to tenant1_schema using the following code:

```
grant select on all tables in schema tenant1_schema to role tenant1_
ro;
grant usage on schema tenant1_schema to role tenant1_ro;
```

5. Grant tenant2_ro access to tenant2_schema using following code.

```
grant select on all tables in schema tenant2_schema to role tenant2_
ro;
grant usage on schema tenant2_schema to role tenant2_ro;
```

6. Turn metadata security on:

```
ALTER SYSTEM SET metadata_security = TRUE;
```

7. Connect to the `metadata_security` database as the `tenant1_user` user. Expand the metadata of the database. `tenant1_user` is only able to see the `tenant1_schema` metadata.

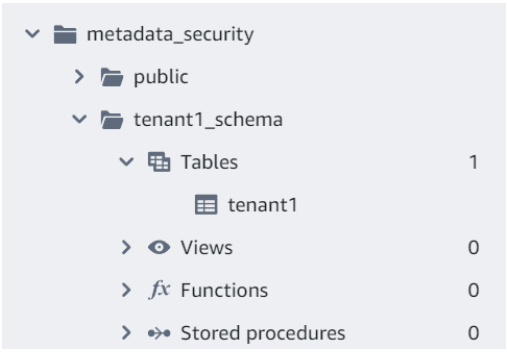


Figure 6.44 – Metadata view for tenant1_user

8. Connect to the `metadata_security` database as the `tenant2_user` user. Expand the metadata of the database. `tenant2_user` is only able to see `tenant2_schema` metadata.

Figure 6.45 – Metadata view for tenant2_user

How it works...

By turning on the metadata security flag for a database, you can restrict the listing of metadata objects to only the objects that the user or role has access to. This enables you to hide metadata from one tenant when the schema names and table names include tenant names in order to comply with privacy controls.

7

Data Authorization and Security

Amazon Redshift provides robust security features to help you protect sensitive data through multiple layers of access control. We will cover how to implement fine-grained access controls in Amazon Redshift using **role-based access control** (**RBAC**), including data masking, row-level, and column-level security. These security mechanisms allow you to precisely manage who can access specific data elements, ensuring that users can only view and modify the data they are authorized to handle.

Specifically, the following topics are covered in this chapter:

- Implementing RBAC
- Implementing column-level security
- Implementing row-level security
- Implementing dynamic data masking

Technical requirements

Here are the technical requirements to complete the recipes in this chapter:

- Access to the AWS Console.
- An AWS administrator should create an IAM user by following Recipe 1 in Appendix. This IAM user will be used in some of the recipes in this chapter.
- Amazon Redshift data warehouse deployed in the eu-west-1 AWS region.
- Amazon Redshift data warehouse master user credentials.
- Access to any SQL interface, such as a SQL client or the Amazon Redshift query editor.

Implementing RBAC

It is best practice to design security by giving users the minimum privileges they need to do their work. Amazon Redshift applies this principle through RBAC (`https://docs.aws.amazon.com/redshift/latest/dg/t_Roles.html`), which grants privileges based on a user's specific role. Privileges are assigned at the role level, without needing to grant permissions individually for each user. Redshift provides four system-defined roles (`https://docs.aws.amazon.com/redshift/latest/dg/r_roles-default.html`) to start with, and you can create additional, more specific roles with targeted privileges. RBAC allows you to limit access to certain commands and assign roles to authorized users, as well as set object-level and system-level privileges for those roles. Roles can be nested using role hierarchy. RBAC enables you to apply fine-grained access control such as column-level security, row-level security, and dynamic data masking.

Getting ready

To complete this recipe, you will need:

- An Amazon Redshift data warehouse deployed in the eu-west-1 AWS region
- Amazon Redshift data warehouse master user credentials
- Access to any SQL interface, such as a SQL client or the Amazon Redshift query editor v2

How to do it...

In this recipe, we will create three database roles: read-only, read-write, and admin. Using the recipe table as an example. The analyst user with the read-only role will be able to run select queries. The `dataengineer` user with the read-write role will be able to insert, delete, and update rows in the customer table. The `dbaadmin` user with the admin role will be assigned the system-defined `sys:dba` role. Let's look at the steps:

1. Connect to Amazon Redshift data warehouse using the SQL client or Redshift query editor v2. Create schema `data_analytics` using the following code;

```
CREATE SCHEMA if not exists data_analytics;
```

2. Create the `recipe` table using the following code:

```
CREATE TABLE data_analytics.recipe (
  recipe_id INT PRIMARY KEY,
  recipe_name VARCHAR(100) NOT NULL,
  ingredients VARCHAR(500) NOT NULL,
  instructions VARCHAR(1000) NOT NULL,
```

```
  cuisine VARCHAR(50) NOT NULL,
  dietary_tags VARCHAR(50),
  cook_time_minutes INT NOT NULL,
  prep_time_minutes INT NOT NULL,
  rating INT NOT NULL,
  num_reviews INT NOT NULL
  );
```

3. Insert data into the `recipe` table using multi-value insert:

```
INSERT INTO data_analytics.recipe
  (recipe_id,recipe_name, ingredients, instructions, cuisine,
dietary_tags, cook_time_minutes, prep_time_minutes, rating, num_
reviews)
VALUES
  (1,'Vegetable Stir-Fry',
   'Broccoli, Bell Peppers, Carrots, Mushrooms, Tofu, Soy Sauce,
Garlic, Ginger',
   'Sauté vegetables in oil, add tofu and soy sauce, serve over
rice.',
   'Asian', 'Vegetarian, Vegan', 20, 15, 4.7, 112),
  (2,'Lentil Curry',
   'Lentils, Coconut Milk, Spinach, Onion, Garlic, Ginger, Curry
Powder, Cumin',
   'Sauté onions and spices, add lentils and coconut milk, simmer
until lentils are tender, add spinach.',
   'Indian', 'Vegetarian, Vegan', 45, 20, 4.3, 89),
  (3,'Vegetable Lasagna',
   'Lasagna Noodles, Ricotta Cheese, Mozzarella Cheese, Spinach,
Zucchini, Eggplant, Tomato Sauce',
   'Layer lasagna noodles, ricotta, vegetables, and tomato sauce,
bake until heated through.',
   'Italian', 'Vegetarian', 60, 45, 4.6, 134);
```

4. Create three users using the following code:

```
create user analyst password disable;
create user dataengineer password disable;
create user dbaadmin password disable;
```

5. Create three database roles, `read_ro`, `readwrite_ro`, and `admin_ro` using the following code:

```
create role read_ro;
create role readwrite_ro;
create role admin_ro;
```

6. Grant read-only access for the `recipe` table to the `read_ro` role. Grant the `read-ro` role to the analyst user:

```
grant usage on schema data_analytics to role read_ro;
grant select on table data_analytics.recipe to role read_ro;
grant role read_ro to analyst;
```

7. Let's run select on the recipe table as the analyst user to view the data using the following code:

```
set session authorization 'analyst';
select * from data_analytics.recipe;
```

recipe_id	recipe_name	ingredients	instructions	cuisine	dietary_tags	cook_time_minutes	prep_time_minutes
1	Vegetable Stir-Fry	Broccoli, Bell Peppers, C...	Sauté vegetables in oil, a...	Asian	Vegetarian, Vegan	20	15
2	Lentil Curry	Lentils, Coconut Milk, Spi...	Sauté onions and spices, ...	Indian	Vegetarian, Vegan	45	20
3	Vegetable Lasagna	Lasagna Noodles, Ricotta...	Layer lasagna noodles, ri...	Italian	Vegetarian	60	45

Figure 7.1 – List of records in the recipe table

8. Now, try to run an update as the analyst user on the table recipe using the following code. You will get a permission denied error:

```
set session authorization 'analyst';
update data_analytics.recipe
set cook_time_minutes = 25
where recipe_name = 'Vegetable Stir-Fry'
--output
ERROR: permission denied for relation recipe [ErrorId: 1-66e5fe89-
060afafb3c5f700022ce6f7b]
```

9. Inherit the `read_ro` role to the `readwrite_ro` role to allow inheritance of permissions. Grant write access on the recipe table to the `readwrite_ro` role:

```
reset session authorization;
grant role read_ro to role readwrite_ro;
grant update, insert, delete on table data_analytics.recipe to role
readwrite_ro;
```

10. Grant `readwrite_ro` to the `dataengineer` user:

```
grant role readwrite_ro to dataengineer;
```

11. As the `dataengineer` user, let's update the table recipe using the following code. The diagram below shows the updated value for the `cook_time_minutes` column for the row with the Vegetable Stir-Fry recipe:

```
set session authorization 'dataengineer';
update data_analytics.recipe
set cook_time_minutes = 25
where recipe_name = 'Vegetable Stir-Fry';
select * from data_analytics.recipe;
reset session authorization;
```

recipe_id ↑	recipe_name	ingredients	instructions	cuisine	dietary_tags	cook_time_minutes	prep_time_min…
1	Vegetable Stir-Fry	Broccoli, Bell Peppers, C…	Sauté vegetables in oil, a…	Asian	Vegetarian, Vegan	25	15
2	Lentil Curry	Lentils, Coconut Milk, Spi…	Sauté onions and spices, …	Indian	Vegetarian, Vegan	45	20
3	Vegetable Lasagna	Lasagna Noodles, Ricotta…	Layer lasagna noodles, ri…	Italian	Vegetarian	60	45

Figure 7.2 – Updated cook_time_minutes to 25

12. Grant the system-defined `sys:dba` role to `admin_ro` using the following code:

```
grant role sys:dba to role admin_ro;
```

13. Grant `admin_ro` to the `dbaadmin` user using the following code:

```
grant role admin_ro to dbaadmin;
```

14. Drop the `data_analytics` schema as the `admin_ro` user using the following code. This drops the entire `data_analytics` schema:

```
set session authorization 'dbaadmin';
drop schema data_analytics cascade;
reset session authorization;
--output:
Output: drop cascades to table data_analytics.recipe
```

15. Let's review the roles assigned to users using the `svv_user_grants` system view. Run the following code:

```
select * from svv_user_grants;
```

user_id	user_name	role_id	role_name	admin_option
102	analyst	348939	read_ro	false
103	dataengineer	348940	readwrite_ro	false
104	dbaadmin	348941	admin_ro	false

Figure 7.3 – Users to roles relationships

How it works...

RBAC allows you to do role inheritance. This simplifies the management of permissions. Roles can be used to grant and revoke permissions at the system level and at the object level. It provides you with security controls to create custom roles with the added benefit of out-of-the-box system-defined roles. System views (`https://docs.aws.amazon.com/redshift/latest/dg/svv_views.html`) provide insights into the list of roles, role inheritance, and which role has access to which database objects. This allows you to more easily understand and manage the access and permissions in your Redshift database. This diagram summarizes RBAC.

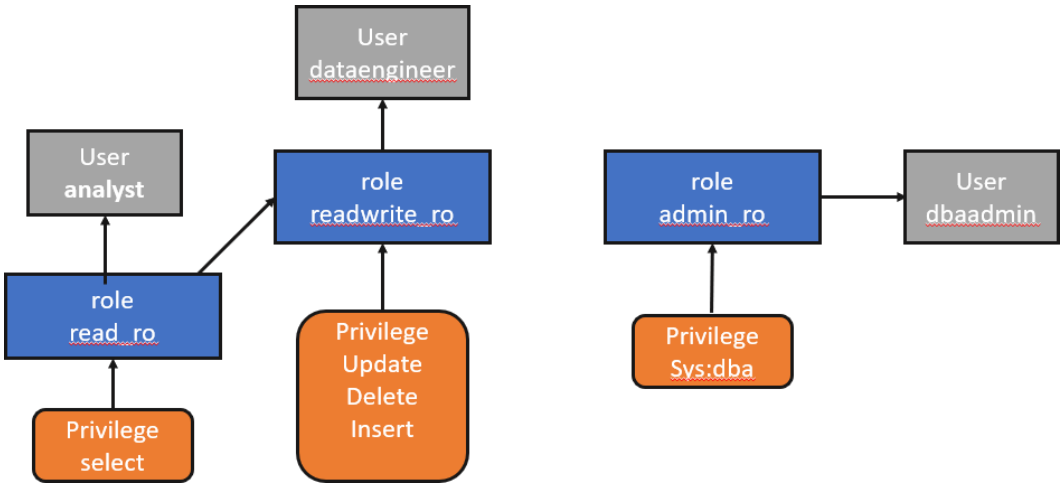


Figure 7.4 – Role inheritance with privileges

Implementing column-level security

Amazon Redshift supports fine-grained data security with column-level controls. Column-level security can be applied to local tables, views, and materialized views. Applying column-level security allows you to restrict access to **personally identifiable information** (**PII**) or **payment card information** (**PCI**) to selective personas. For instance, the finance or human resources team can be granted access to sensitive information but you can restrict access from the sales and marketing team.

Getting ready

To complete this recipe, you will need:

- An Amazon Redshift data warehouse deployed in the eu-west-1 AWS region
- Amazon Redshift data warehouse master user credentials
- Access to any SQL interface, such as a SQL client or the Amazon Redshift query editor v2

How to do it...

1. In this recipe, we will use the customer table. Using column-level access control, the sales user will be restricted from accessing the phone number column.
2. Connect to the Amazon Redshift cluster using the SQL client or query editor. Create the `customer` table using the following code:

```
create table customer
(C_CUSTKEY integer,
 C_NAME varchar(15),
 C_NATIONKEY integer,
 c_PHONE varchar(15),
 C_ACCTBAL decimal(6,2),
 C_MKTSEGMENT varchar(15),
 C_COMMENT varchar(25));
```

3. Insert the following records into the `customer` table:

```
Insert into customer values
(1, 'customer-0001', 1, '123-123-1234', 111.11, 'MACHINERY', 'FIRST
ORDER'),
(2, 'customer-0002', 2, '122-122-1234', 222.11, 'HOUSEHOLD', 'SECOND
ORDER');
```

Let's create the sales user:

```
CREATE user sales with password 'Sales1234';
```

4. Grant access to the `sales` users on all the columns in the `customer` table except the `c_phone` column:

```
GRANT SELECT (C_CUSTKEY, C_NAME, C_NATIONKEY, C_ACCTBAL, C_
MKTSEGMENT, C_COMMENT) ON customer TO sales;
```

5. Let's verify the column-level access for sales users. Run the following code. You will receive a permission denied error message, as the sales user does not have access to the `c_phone` column:

```
SET SESSION AUTHORIZATION 'sales';
SELECT CURRENT_USER;
SELECT * FROM customer;
--output
ERROR: 42501: permission denied for relation customer
```

6. Let's select the columns in the select statement that the `sales` users have access to:

```
SET SESSION AUTHORIZATION 'sales';
SELECT CURRENT_USER;
SELECT C_CUSTKEY, C_NAME, C_NATIONKEY, C_ACCTBAL, C_MKTSEGMENT, C_
COMMENT FROM customer;
```

Here is the output of the preceding code:

c_custkey	c_name	c_nationkey	c_acctbal	c_mktsegment	c_comment
1	customer-0001	1	111.1100	MACHINERY	FIRST ORDER
2	customer-0002	2	222.1100	HOUSEHOLD	SECOND ORDER

Figure 7.5 – Verify the successful selection of the PII columns

How it works...

Using the GRANT and REVOKE statements, you can manage column-level access control in Amazon Redshift by enabling or disabling permissions for users, roles, and groups on tables, views, and materialized views. You can refer to the GRANT and REVOKE syntax to get fine-grained access control using `https://docs.aws.amazon.com/redshift/latest/dg/r_GRANT.html` and `https://docs.aws.amazon.com/redshift/latest/dg/r_REVOKE.html`.

Implementing row-level security

Amazon Redshift using RBAC supports **row-level security** (**RLS**) (`https://docs.aws.amazon.com/redshift/latest/dg/t_rls.html`) for granular access control over sensitive data, letting you specify which users or roles can access specific records. RLS policies defined at the table level restrict access to particular rows, complementing column-level security. Enforcing RLS policies on tables limits the result sets returned to users based on the policy expressions. As a user with the necessary permissions, such as a superuser and the `sys:secadmin` role, you can create, modify, or manage all row-level security policies for tables. Multiple RLS policies can be attached to objects, roles, or users. Depending on the RLS CONJUNCTION TYPE setting for the table, Amazon Redshift applies all the policies defined for a user using either AND or OR syntax.

Getting ready

To complete this recipe, you will need:

- An Amazon Redshift data warehouse deployed in the eu-west-1 AWS region
- Amazon Redshift data warehouse master user credentials
- Access to any SQL interface, such as a SQL client or the Amazon Redshift query editor v2

How to do it...

1. In this recipe, we will use row-level security on the customer table so users can only see their own data. The `Customer` table contains multi-tenant data, data across internal and external customers.
2. Connect to the Amazon Redshift data warehouse using the SQL client or the Redshift query editor v2. Create `schema data_analytics` using the following code:

```
CREATE SCHEMA if not exists data_analytics;
```

3. Create the `customer` table using the following code:

```
Drop table if exists data_analytics.customer;
CREATE TABLE data_analytics.customer
(
  C_CUSTKEY      BIGINT NOT NULL,
  C_NAME         VARCHAR(25),
  C_NATIONKEY    BIGINT,
  C_PHONE        VARCHAR(15),
  C_ACCTBAL      DECIMAL(18,4),
```

```
  C_MKTSEGMENT    VARCHAR(10),
  C_COMMENT       VARCHAR(117)
);
```

4. Insert data into the `customer` table using the following code:

```
Insert into data_analytics.customer values
(1, 'customer-0001', 1, '123-123-1234', 111.11, 'MACHINERY', 'FIRST
ORDER'),
(2, 'customer-0002', 2, '122-122-1234', 222.11, 'HOUSEHOLD', 'SECOND
ORDER'),
(3, 'customer-0003', 3, '122-122-1234', 333.11, 'RETAIL', 'Third
Order');
```

5. Let's create users and roles, and grant roles to users:

```
CREATE ROLE internal;
CREATE ROLE external;

create user "customer-0001" PASSWORD DISABLE;
create user "customer-00010" PASSWORD DISABLE;
create user "external-user" password DISABLE;

GRANT ROLE internal TO "customer-0001";
GRANT ROLE internal TO "customer-00010";
GRANT ROLE external TO "external-user";
```

6. Create RLS policies for users who are logging directly into the data warehouse:

```
CREATE RLS POLICY see_only_own_user_rows
WITH ( c_name varchar(25) )
USING ( c_name = current_user);Attach RLs policy to table
```

7. Attach the RLS policy to tables and roles:

```
ATTACH RLS POLICY see_only_own_user_rows
ON data_analytics.customer
TO ROLE internal;
```

8. Grant the internal select access role on the customer table and the `data_analytics` schema:

```
GRANT USAGE on SCHEMA data_analytics to role internal;
GRANT SELECT ON data_analytics.customer to role internal;
```

9. Turn on RLS at the table level:

```
ALTER TABLE data_analytics.customer ROW LEVEL SECURITY on;
```

10. Let's verify that the `'customer-0001'` user only has one record:

```
SET SESSION AUTHORIZATION 'customer-0001';
select * from data_analytics.customer;
reset session authorization;
```

c_custkey	c_name	c_nationkey	c_phone	c_acctbal	c_mktsegment	c_comment
1	customer-0001	1	123-123-1234	111.11	MACHINERY	FIRST ORDER

Figure 7.6 – Output for customer-0001 with RLS

11. Let's verify the number of users for `'customer-00010'`. The output will be zero records:

```
SET SESSION AUTHORIZATION 'customer-00010';
select * from data_analytics.customer;
--output zero records
reset session authorization;
```

12. Now let's verify the number of users for `externa-user`:

```
SET SESSION AUTHORIZATION 'external-user';
select * from data_analytics.customer;
--output zero records
reset session authorization;
```

13. If you have an external-facing application for external users, you can set a context variable for that session. In this example, it will be `customer_name`. Let's create an RLS policy for external users:

```
CREATE RLS POLICY see_only_own_customer_rows_external
WITH ( c_name varchar(25) )
USING ( c_name = current_setting('app.customer_name', FALSE));
```

14. Attach RLS to the table:

```
ATTACH RLS POLICY see_only_own_customer_rows_external ON data_
analytics.customer TO ROLE EXTERNAL;
```

15. Grant the `external select` role on the customer table:

```
grant usage on schema data_analytics to role EXTERNAL;
GRANT select ON TABLE data_analytics.customer TO ROLE EXTERNAL;
```

16. Apply RLS with the OR conjunction type:

```
ALTER TABLE data_analytics.customer ROW LEVEL SECURITY ON
CONJUNCTION TYPE or;
```

17. Let's verify RLS for an external user using a context variable for the session:

```
SET SESSION AUTHORIZATION 'external-user';
select set_config('app.customer_name', 'customer-0003', FALSE);
select * from data_analytics.customer;
reset session authorization;
```

c_custkey	c_name	c_nationkey	c_phone	c_acctbal	c_mktsegment	c_comment
3	customer-0003	3	122-122-1234	333.11	RETAIL	Third Order

Figure 7.7 – Output for an external user with RLS

How it works...

Row-level security is built on the foundation of RBAC. A superuser or user with the `sys:secadmin` role creates policies to apply fine-grained access control. These policies are attached to the table and apply to `SELECT`, `UPDATE`, and `DELETE` operations. When the user queries tables with RLS, based on the policy, it only returns records the user is authorized to see. Multiple policies can be attached to a table, and a single policy can be attached to multiple tables. All attached policies on the table can use the or or and conjunction types. System views can be used to audit and monitor RLS policies (`https://docs.aws.amazon.com/redshift/latest/dg/t_rls_ownership.html`).

Implementing dynamic data masking

Dynamic data masking (DDM) (`https://docs.aws.amazon.com/redshift/latest/dg/t_ddm.html`) lets you hide sensitive data. It is often used to meet regulations or privacy standards. DDM allows you to control how sensitive data is displayed, based on a user's permissions. This is done at the time the data is accessed, not when it is stored. DDM is an alternative to permanently obscuring the data during the data loading process. With DDM, you don't need to modify your data pipelines. You set up masking policies that determine who can see what data. These policies are attached to tables and columns. The policies apply to individual users, roles, or everyone. DDM makes it easier to adapt to changing privacy requirements, without needing to change your data transformations, underlying data, or application queries.

Using DDM and Amazon Redshift Lambda User-Defined Functions (`https://docs.aws.amazon.com/redshift/latest/dg/udf-creating-a-lambda-sql-udf.html`), you can apply data tokenization with your own policies or through integration with third-party partners. For reference, see the Protegrity blog (`https://aws.amazon.com/blogs/apn/data-tokenization-with-amazon-redshift-dynamic-data-masking-and-protegrity/`).

Getting ready

To complete this recipe, you will need:

- An Amazon Redshift data warehouse deployed in the eu-west-1 AWS region
- Amazon Redshift data warehouse master user credentials
- Access to any SQL interface, such as a SQL client or the Amazon Redshift query editor v2

How to do it...

1. In this recipe, we will use a DDM security policy on the `card_number` column in the `pci_data` table to define how sensitive data is returned to users at query time.
2. Connect to Amazon Redshift data warehouse using the SQL client or the Redshift query editor v2. Create the `data_analytics` schema using the following code:

```
CREATE SCHEMA if not exists data_analytics;
```

3. Create the `pci_data` table using the following code:

```
Drop table if exists data_analytics.pci_data;
CREATE TABLE data_analytics.pci_data (
    card_number VARCHAR(16) NOT NULL,
    expiration_date VARCHAR(7) NOT NULL,
    customer_id VARCHAR(50) NOT NULL
);
```

4. Insert data into the `pci_data` table using the following code:

```
INSERT INTO data_analytics.pci_data (card_number, expiration_date,
customer_id)
VALUES
    ('4111111111111111', '03/25',  'John Doe'),
    ('5555555555554444', '08/26',  'Jane Smith'),
    ('6011111111111117', '12/24',  'customer-111'),
    ('3456789012345678', '05/27',  'Sarah Williams'),
    ('4012888888881881', '11/23', 'Michael Brown');
```

5. Let's create users and roles, and grant roles to users. Grant select to the public role for the `pci_data` table:

```
       create user "agent" PASSWORD DISABLE;
       create user "jane" PASSWORD DISABLE;
       create user "cust_ro" password DISABLE;
CREATE ROLE agent_ro;
CREATE ROLE nopci_ro;
CREATE ROLE cust_ro

-- Grant Roles to Users
GRANT ROLE agent_ro to agent;
GRANT ROLE nopci_ro  to jane;
GRANT ROLE cust_ro to "customer-111";

GRANT SELECT ON data_analytics.pci_data TO PUBLIC;
GRANT usage on schema data_analytics to public;
```

6. Create masking policies for the `card_number` column. The first policy, `"Mask_CC_Full"`, completely masks `card_number`; the second policy, `Mask_CC_Partial`, is redacting the `credit_card` number; and `Mask_CC_Raw` uses an expression to return the raw `card_number` for the current user matching `customer_id`:

```
        CREATE MASKING POLICY Mask_CC_Full
        WITH (credit_card VARCHAR(256))
        USING ('XXXXXXXXXXXXXXXX'::text);

CREATE FUNCTION REDACT_CREDIT_CARD (text)
  returns text
immutable
as $$
  select left($1,6)||'XXXXXX'||right($1,4)
$$ language sql;

CREATE MASKING POLICY Mask_CC_Partial
WITH (card_number VARCHAR(16))
USING (REDACT_CREDIT_CARD(card_number));

CREATE MASKING POLICY Mask_CC_Raw
WITH (card_number varchar(16), customer_id VARCHAR(50))
USING (
CASE customer_id WHEN current_user THEN card_number ELSE NULL END
);
```

7. Attach a masking policy to fully mask access to `PUBLIC`. `PUBLIC` is the default role:

```
ATTACH MASKING POLICY Mask_CC_Full
ON data_analytics.pci_data(card_number)
TO PUBLIC;
```

8. Attach the `Mask_cc_partial` and `Mask_cc_Raw` masking policies. Both of these masking policies are attached to the `pci_data` table for the `card_number` column. Both masking policies have been assigned priority to remove potential conflicts:

```
ATTACH MASKING POLICY Mask_CC_Partial
ON data_analytics.pci_data(card_number)
TO ROLE agent_ro
```

```
priority 2;

ATTACH MASKING POLICY Mask_CC_Raw
ON data_analytics.pci_data(card_number)
USING (card_number, customer_id)
TO ROLE cust_ro
priority 1;
```

9. To check masking policies have been created, use the following code:

```
SELECT * FROM svv_masking_policy;
```

policy_database	policy_name	input_columns	policy_expression	policy_modified_by	policy_modified_time
dev	mask_cc_full	[{"colname":"card_numbe...	[{"expr":"CAST('XXXXXX...	awsuser	2024-09-22 20:26:41
dev	mask_cc_partial	[{"colname":"card_numbe...	[{"expr":"\"public\".\"redact...	awsuser	2024-09-22 20:58:08
dev	mask_cc_raw	[{"colname":"card_numbe...	[{"expr":"CASE WHEN (\"...	awsuser	2024-09-22 20:58:13

Figure 7.8 – Output for svv_masking_policy

10. To that confirm masking policies have been attached to tables, use the following code:

```
SELECT * FROM svv_attached_masking_policy;
```

policy_name	schema_name	table_name	table_type	grantor	grantee	grantee_type	priority
mask_cc_full	data_analytics	pci_data	table	awsuser	public	public	0
mask_cc_partial	data_analytics	pci_data	table	awsuser	agent_ro	role	2
mask_cc_raw	data_analytics	pci_data	table	awsuser	cust_ro	role	1

Figure 7.9 – Output for svv_attached_masking_policy

11. Let's run the following code to verify masking for the `agent` user. The agent user is able to see the redacted `card_number`:

```
set session authorization 'agent';
select * from data_analytics.pci_data;

reset session authorization;
```

card_number	expiration_date	customer_id
411111XXXXXX1111	03/25	John Doe
555555XXXXXX4444	08/26	Jane Smith
601111XXXXXX1117	12/24	customer-111
345678XXXXXX5678	05/27	Sarah Williams
401288XXXXXX1881	11/23	Michael Brown

Figure 7.10 – Output for redacted card_number

12. Let's verify the masking policy for the `Jane` user using the following code:

```
set session authorization 'jane';
select * from data_analytics.pci_data;

reset session authorization;
```

card_number	expiration_date	customer_id
XXXXXXXXXXXXXXXX	03/25	John Doe
XXXXXXXXXXXXXXXX	08/26	Jane Smith
XXXXXXXXXXXXXXXX	12/24	customer-111
XXXXXXXXXXXXXXXX	05/27	Sarah Williams
XXXXXXXXXXXXXXXX	11/23	Michael Brown

Figure 7.11 – Output for fully masked card_number

13. Let's verify the masking policy for the `'customer-111'` user using the following code:

```
set session authorization 'customer-111';
select * from data_analytics.pci_data;
reset session authorization;
```

card_number	expiration_date	customer_id
NULL	03/25	John Doe
NULL	08/26	Jane Smith
6011111111111117	12/24	customer-111
NULL	05/27	Sarah Williams
NULL	11/23	Michael Brown

Figure 7.12 – Output for conditional masked card_number

How it works...

DDM allows you to define configuration-driven, consistent, format-preserving, and irreversible masked data values. This capability enables you to control your data masking approach using familiar SQL. You can use RBAC to implement different levels of data masking. This involves creating a masking policy to identify which columns need to be masked, and you have the flexibility to choose how the masked data will be displayed. For example, you can completely hide all the information in the data, replace partial real values with wildcard characters, or define your own masking method using SQL expressions, Python, or Lambda **User-Defined Functions (UDFs)**. Additionally, you can apply conditional masking based on the values in other columns, which selectively protects the data in a table. You can use row-level masking, DDM, and column-level security together on the table, materialized views, and late binding views to achieve fine-grained access control.

8

Performance Optimization

Amazon Redshift provides out-of-the-box performance for most workloads. It defaults table design choices such as compression, sort, and distribution key to `AUTO` and is able to learn from user workloads to automatically set up the right structure. For more information, see the *Working with automatic table optimization* page in the docs (`https://docs.aws.amazon.com/redshift/latest/dg/t_Creating_tables.html`). Amazon Redshift provides you with flexible controls that allow you to optimize performance and make alternative configuration choices whenever necessary. The sort, distribution key, and table encoding choices influence the performance of queries. In this chapter, we will discuss the optimization techniques to improve throughput. Also, we will take a deep dive into analyzing queries to understand the rationale behind the tuning exercise.

In this chapter, we will look into the following recipes:

- Configuring Amazon Redshift Advisor for provisioned clusters
- Managing column compression
- Managing data distribution
- Managing the sort key
- Analyzing and improving queries for provisioned clusters
- Configuring Workload Management (WLM) for provisioned clusters
- Utilizing concurrency scaling for provisioned clusters
- Optimizing Spectrum queries for provisioned clusters

Technical requirements

Here are the technical requirements in order to complete the recipes in this chapter:

- Access to the AWS Management Console.
- AWS administrator permission to create an IAM user by following *Recipe 1* in *Appendix*. This IAM user will be used in some of the recipes in this chapter.
- AWS administrator permission to create an IAM role by following *Recipe 3* in *Appendix*. This IAM role will be used in some of the recipes in this chapter.
- AWS administrator permission to deploy the AWS CloudFormation template (`https://github.com/PacktPublishing/Amazon-Redshift-Cookbook-2E/blob/main/Chapter08/chapter_8_CFN.yaml`) and create two IAM policies:
    - An IAM policy attached to the IAM user that will give them access to Amazon Redshift, Amazon EC2, AWS Secrets Manager, AWS IAM, AWS CloudFormation, AWS KMS, AWS Glue, and Amazon S3.
    - An IAM policy attached to the IAM role that will allow the Amazon Redshift data warehouse to access Amazon S3.
- Attach an IAM role to the Amazon Redshift data warehouse (provisioned or serverless) by following *Recipe 4* in *Appendix*. Take note of the IAM role name; we will refer to it in the recipes with `[Your-Redshift_Role]`.
- An Amazon Redshift data warehouse deployed in the AWS Region eu-west-1.
- Amazon Redshift data warehouse master user credentials.
- Access to any SQL interface, such as a SQL client or Amazon Redshift Query Editor V2.
- An AWS account number; we will refer to it in the recipes with `[Your-AWS_Account_Id]`.
- The code files available in the GitHub repo (`https://github.com/PacktPublishing/Amazon-Redshift-Cookbook-2E/tree/main/Chapter08`).
- Access to AWS CloudShell.

Configuring Amazon Redshift Advisor for provisioned clusters

Amazon Redshift Advisor was launched in mid-2018. It runs daily and continuously observes the workload's operational statistics on the data warehouse with the lens of best practices. Amazon Redshift Advisor uses sophisticated algorithms to provide tailored best practice recommendations, which enable you to get the best possible performance and cost savings. The recommendations are prioritized based on their severity level and potential impact on the system, which eases administration. Some of the recommendations include:

- Optimization of the copy command for optimal data ingestion
- Optimization of physical table design
- Optimization of manual workload management
- Cost optimization with a recommendation to delete clusters after taking a snapshot, if the cluster is not utilized

Along with the Advisor recommendations, the automatic table optimization feature allows applying these recommendations automatically without requiring administrator intervention, creating a fully self-tuning system.

In this recipe, you will see where to find Amazon Redshift Advisor to view the recommendations.

Getting ready

To complete this recipe, you will need the following setup:

- An IAM user with access to an Amazon Redshift provisioned cluster
- An Amazon Redshift provisioned cluster deployed in the AWS Region eu-west-1
- AWS CloudShell with the latest AWS CLI (`https://docs.aws.amazon.com/cli/latest/userguide/getting-started-install.html#getting-started-install-instructions`)

How to do it...

In this recipe, we will use the Amazon Redshift console and the AWS CLI to access the Advisor recommendations for our cluster:

1. Navigate to the AWS Management Console and select **Amazon Redshift**.
2. On the left-hand side, you will see **ADVISOR**. Click on it.
3. If you have multiple clusters in a Region, you will be able to view the recommendations for all the data warehouses. You can group the recommendations by data warehouse or by category – cost, performance, security, or other.

Figure 8.1 – Access Amazon Redshift Advisor

4. You can distribute the recommendations by exporting them from the console to a file. To export the recommendations from the Advisor page, select **Export**.

Figure 8.2 – Amazon Redshift Advisor recommendations

5. Using AWS CloudShell, run the following code using the AWS CLI to list the output of Advisor's recommendations for all clusters in the account:

```
aws redshift list-recommendations
```

How it works...

Amazon Redshift builds recommendations by continuously analyzing the operational data of your data warehouse. Advisor provides recommendations that will have a significant impact on the performance of your data warehouse. Advisor, alongside automatic table optimization, collects the query access patterns and analyzes them using a **machine learning** (**ML**) service to predict recommendations about the sort and distribution keys. These recommendations are then applied automatically to the target tables in the data warehouse. Advisor and automatic table optimization run when the workload is low so that there is no impact on user queries.

Managing column compression

Amazon Redshift columnar architecture stores data column by column on the disk. Analytical queries select a subset of the column and perform aggregation on millions to billions of records. The columnar architecture reduces the I/O by selecting a subset of the columns and hence improves query performance. When data is ingested into an Amazon Redshift table, it provides three to four times compression. This further reduces the storage footprint, which in turn reduces I/O and hence improves query performance. Reducing the storage footprint also saves you costs. Amazon Redshift Advisor provides recommendations for compressing uncompressed tables.

In this recipe, you will see how Amazon Redshift automatically applies compression on new and existing tables. You will also see how column-level compression can be modified for existing columns.

Getting ready

To complete this recipe, you will need:

- An IAM user with access to Amazon Redshift
- An Amazon Redshift data warehouse deployed in the AWS Region eu-west-1
- Amazon Redshift data warehouse master user credentials
- Access to any SQL interface, such as a SQL client or Amazon Redshift Query Editor V2
- An IAM role attached to an Amazon Redshift data warehouse that can access Amazon S3; we will refer to it in the recipes with `[Your-Redshift_Role]`
- An AWS account number; we will refer to it in the recipes with `[Your-AWS_Account_Id]`

How to do it...

In this recipe, we will be analyzing the automatic table-level compression applied by Amazon Redshift:

1. Connect to the Amazon Redshift data warehouse using a SQL client or Query Editor V2 and create the `customer` table using the following command:

```
drop table if exists customer;
CREATE TABLE customer
(
  C_CUSTKEY      BIGINT NOT NULL,
  C_NAME         VARCHAR(25),
  C_ADDRESS      VARCHAR(40),
  C_NATIONKEY    BIGINT,
  C_PHONE        VARCHAR(15),
  C_ACCTBAL      DECIMAL(18,4),
  C_MKTSEGMENT   VARCHAR(10),
  C_COMMENT      VARCHAR(117)
)
diststyle AUTO;
```

2. Let's now analyze the compression types applied to the columns. Run the following command:

```
SELECT "column", type, encoding FROM pg_table_def
WHERE tablename = 'customer';
```

Here is the expected output:

```
    column    |          type          | encoding
--------------+------------------------+----------
 c_custkey    | bigint                 | az64
 c_name       | character varying(25)  | lzo
 c_address    | character varying(40)  | lzo
 c_nationkey  | bigint                 | az64
 c_phone      | character varying(15)  | lzo
 c_acctbal    | numeric(18,4)          | az64
 c_mktsegment | character varying(10)  | lzo
 c_comment    | character varying(117) | lzo
```

When no encoding type is specified for columns, the table is created with `ENCODE AUTO`. Listing the encoding type of columns shows the default encoding, which is preserved. With `ENCODE AUTO`, when Redshift determines a better encoding type based on the operational metrics of your workload, it will automatically apply encoding. Amazon Redshift automatically applies a compression type of `az64` or `AZ64` for `INT`, `SMALLINT`, `BIGINT`, `TIMESTAMP`, `TIMESTAMPTZ`, `DATE`, and `NUMERIC` column types. `az64` is Amazon's proprietary compression encoding algorithm designed to achieve a high compression ratio and improved query processing. The default encoding of `lzo` is applied to varchar and `character` columns.

Refer to this page in the docs to learn more about different encoding types in Amazon Redshift:

`https://docs.aws.amazon.com/redshift/latest/dg/c_Compression_encodings.html`

3. Now, let's recreate the `customer` table with `C_CUSTKEY` encoded as raw using the following SQL:

```
drop table if exists customer ;
CREATE TABLE customer
(
  C_CUSTKEY      BIGINT NOT NULL encode raw,
  C_NAME         VARCHAR(25),
  C_ADDRESS      VARCHAR(40),
  C_NATIONKEY    BIGINT,
  C_PHONE        VARCHAR(15),
  C_ACCTBAL      DECIMAL(18,4),
  C_MKTSEGMENT   VARCHAR(10),
  C_COMMENT      VARCHAR(117)
)
diststyle AUTO;
SELECT "column", type, encoding FROM pg_table_def
WHERE tablename = 'customer';
```

Here is the expected output:

column	type	encoding
c_custkey	bigint	none
c_name	character varying(25)	lzo
c_address	character varying(40)	lzo
c_nationkey	bigint	az64
c_phone	character varying(15)	lzo
c_acctbal	numeric(18,4)	az64
c_mktsegment	character varying(10)	lzo
c_comment	character varying(117)	lzo

Figure 8.3 – Output of the preceding query

Notice that the `c_custkey` column is encoded with raw encoding (none).

4. Let's now use `COPY` to load data from Amazon S3 using the following command, replacing `[Your-AWS_Account_Id]` and `[Your-Redshift_Role]` with the respective values:

```
COPY customer from 's3://packt-redshift-cookbook/RetailSampleData/
customer/' iam_role 'arn:aws:iam::[Your-AWS_Account_Id]:role/[Your-
Redshift_Role]'  CSV gzip COMPUPDATE PRESET;
SELECT "column", type, encoding FROM pg_table_def
WHERE tablename = 'customer';
```

Here is the expected output:

column	type	encoding
c_custkey	bigint	az64
c_name	character varying(25)	lzo
c_address	character varying(40)	lzo
c_nationkey	bigint	az64
c_phone	character varying(15)	lzo
c_acctbal	numeric(18,4)	az64
c_mktsegment	character varying(10)	lzo
c_comment	character varying(117)	lzo

Figure 8.4 – Output of the preceding query

Note

An Amazon Redshift command that contains `COMPUPDATE` determines the encoding for the columns for an empty table even for columns set to raw encoding with no compression. We first create the table with the column `c_custkey` set to encode raw. Then, run the `COPY` command with the `compupdate` preset option, which determines the encoding of the columns for an empty table. Then, we verify the encodings of the columns and that the column `c_custkey` has the encoding type of `az64`.

How it works...

Amazon Redshift by default applies compression, which helps to reduce the storage footprint and hence improve query performance due to a decrease in I/O. Each column can have different encoding types and columns that can grow and shrink independently. When no encoding is specified, Redshift uses `AUTO` encoding and automatically switches to the optimal encoding type based on workload performance. For an existing table, you can use the `ANALYZE COMPRESSION` command to determine the encoding type that results in storage savings.

It is a built-in command that will find the optimal compression for each column. You can then apply the recommended compression to the table using an `alter` statement or by creating a new table with the new encoding types and copying the data from the old to the new table.

Managing data distribution

Distribution style is a table property that dictates how that table's data is distributed throughout the compute nodes. The goal of data distribution is to leverage the massively parallel processing of Amazon Redshift and reduce the I/O during query processing to improve performance. Amazon Redshift Advisor provides actionable recommendations on distribution style for the table with the `alter` statement. Using automatic table optimization enables you to self-manage the table distribution style based on workload patterns:

- **KEY**: The value is hashed; the same value goes to the same location (slice)
- **ALL**: All table data goes to the first slice of every compute node
- **EVEN**: Round-robin data distribution across the compute nodes and slices

- **AUTO:** Applies EVEN, ALL, and KEY distribution based on the scenario

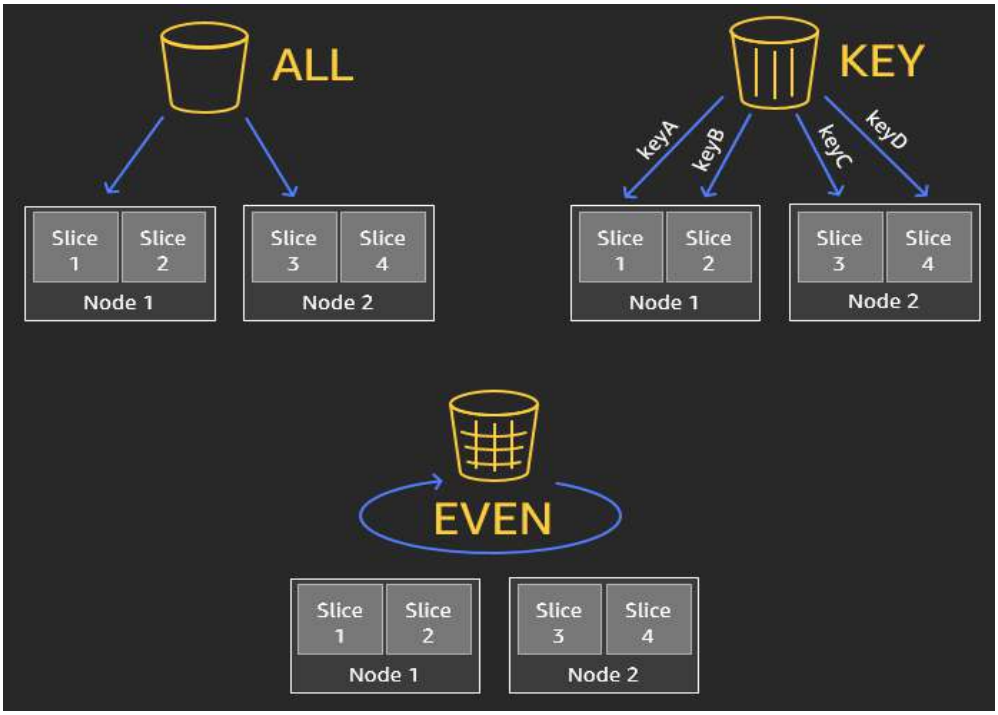


Figure 8.5 – Data distribution styles

In this recipe, you will see how Amazon Redshift's automatic table style works and the benefits of different distribution styles.

Getting ready

To complete this recipe, you will need:

- An IAM user with access to Amazon Redshift
- An Amazon Redshift data warehouse deployed in the AWS Region eu-west-1
- Amazon Redshift data warehouse master user credentials
- Access to any SQL interface, such as a SQL client or Amazon Redshift Query Editor V2
- An IAM role attached to an Amazon Redshift data warehouse that can access Amazon S3; we will refer to it in the recipes with `[Your-Redshift_Role]`
- An AWS account number; we will refer to it in the recipes with `[Your-AWS_Account_Id]`

How to do it...

In this recipe, we will create the `customer` table with different distribution keys and analyze the join effectiveness and data distribution:

1. Connect to the Amazon Redshift data warehouse using a SQL client or Query Editor V2.
2. Let's create the `dwdate` table with the default auto-distribution style. Then, run the `COPY` command, replacing `[Your-AWS_Account_Id]` and `[Your-Redshift_Role]` with the respective values:

```
DROP TABLE IF EXISTS dwdate;
CREATE TABLE dwdate
(
  d_datekey            INTEGER NOT NULL,
  d_date               VARCHAR(19) NOT NULL,
  d_dayofweek          VARCHAR(10) NOT NULL,
  d_month              VARCHAR(10) NOT NULL,
  d_year               INTEGER NOT NULL,
  d_yearmonthnum       INTEGER NOT NULL,
  d_yearmonth          VARCHAR(8) NOT NULL,
  d_daynuminweek       INTEGER NOT NULL,
  d_daynuminmonth      INTEGER NOT NULL,
  d_daynuminyear       INTEGER NOT NULL,
  d_monthnuminyear     INTEGER NOT NULL,
  d_weeknuminyear      INTEGER NOT NULL,
  d_sellingseason      VARCHAR(13) NOT NULL,
  d_lastdayinweekfl    VARCHAR(1) NOT NULL,
  d_lastdayinmonthfl   VARCHAR(1) NOT NULL,
  d_holidayfl          VARCHAR(1) NOT NULL,
  d_weekdayfl          VARCHAR(1) NOT NULL
);
COPY public.dwdate from 's3://packt-redshift-cookbook/dwdate/' iam_
role 'arn:aws:iam::[Your-AWS_Account_Id]:role/[Your-Redshift_Role]'
CSV gzip COMPUPDATE PRESET dateformat 'auto';
```

Here is the expected output:

schema	table	diststyle	skew_rows
public	dwdate	AUTO(ALL)	NULL

Figure 8.6 – Output of the preceding query

Amazon Redshift by default sets the distribution style to `AUTO(ALL)`. Amazon Redshift automatically manages the distribution style for the table; for small tables, it sets `ALL` as the distribution style. With the distribution style `ALL`, the data for this table is stored on every compute node slice `0`. The distribution style of `ALL` is well suited for small dimension tables, which are not updated frequently.

3. Let's create the `customer` table with the default `AUTO` distribution style using the following, replacing `[Your-AWS_Account_Id]` and `[Your-Redshift_Role]` with the respective values:

```
DROP TABLE IF EXISTS customer;
CREATE TABLE public.customer
(
  C_CUSTKEY      BIGINT NOT NULL encode raw,
  C_NAME         VARCHAR(25),
  C_ADDRESS      VARCHAR(40),
  C_NATIONKEY    BIGINT,
  C_PHONE        VARCHAR(15),
  C_ACCTBAL      DECIMAL(18,4),
  C_MKTSEGMENT   VARCHAR(10),
  C_COMMENT      VARCHAR(117)
);

COPY customer from 's3://packtcookbook-hsp/ssb/customer/'
iam_role 'arn:aws:iam::[Your-AWS_Account_Id]:role/[Your-Redshift_
Role]' CSV gzip  compupdate preset region 'eu-west-1';
```

4. To verify the distribution style of the `customer` table, execute the following command:

```
select "schema", "table", "diststyle", skew_rows
from svv_table_info
where "table" = 'customer';
```

Here is the expected output:

schema	table	diststyle	skew_rows
public	customer	AUTO(EVEN)	NULL

Figure 8.7 – Output of the preceding query

Here, Amazon Redshift sets the distribution style to `EVEN`, based on the table size. The distribution style of `AUTO` converts the table from `ALL` to `EVEN` based on the threshold of records in the table.

5. Let's now modify the distribution style of the `customer` table using the `c_nationkey` column, by executing the following query:

```
alter table customer alter distkey C_NATIONKEY;
```

6. Now, let's verify the distribution style of the `customer` table. Run the following query:

```
select "schema", "table", "diststyle", skew_rows
from svv_table_info
where "table" = 'customer';
```

Here is the expected output:

schema	table	diststyle	skew_rows
public	customer	KEY(c_nationkey)	100

Figure 8.8 – Output of the preceding query

`c_nationkey` causes the skewness in the distribution, as shown by the `skew_row` column, since it has less distinct values (low cardinality). Ideally, the `skew_row` ratio should be less than 5. When data is skewed, some compute nodes will do more work compared to others. The performance of the query is affected by the compute node that has more data.

7. Let's now alter the distribution key for the `customer` table using a high-cardinality column, `c_custkey`. Run the following query and verify the table skew:

```
alter table customer alter distkey c_custkey;
select "schema", "table", "diststyle", skew_rows
from svv_table_info
where "table" = 'customer';
```

Here is the expected output:

schema	table	diststyle	skew_rows
public	customer	KEY(c_custkey)	1.02

Figure 8.9 – Output of the preceding query

Now, the customer table has low `skew_rows` that will ensure all the compute nodes can perform equal work when processing the query.

How it works...

Amazon data distribution is a physical table property. It determines how the data is distributed across the compute nodes. The purpose of data distribution is to have every compute node work in parallel to run the workload and reduce the I/O during join operations, to optimize performance. Amazon Redshift's automatic table optimizations enable you to achieve this. You also have the option to select your distribution style to fine-tune your most demanding workloads to achieve significant performance. Creating a Redshift table with automatic table optimization will automatically change the distribution style based on the workload pattern. You can review the alter table recommendations in the `svv_alter_table_recommendations` view and the actions applied by automatic table optimization in the `svl_auto_worker_action` view.

Managing the sort key

Data sorting in Amazon Redshift refers to how data is physically sorted on the disk. Data sorting is determined by the sort key defined on the table. Amazon Redshift automatically creates in-memory metadata, called zone maps. Zone maps contain the minimum and maximum values for each block. Zone maps automatically enable eliminating I/O by not scanning blocks that do not contain data for queries. Sort keys make zone maps more efficient.

A sort key can be defined on one or more columns. The columns defined in the sort keys are based on your query pattern. Most frequently, filtered columns are good candidates for the sort key. The order of sort key columns is defined from low to high cardinality. Sort keys enable range-restricted scans to prune blocks, eliminating I/O and hence optimizing query performance. Redshift Advisor provides recommendations on optimal sort keys and automatic table optimization handles the sortkey changes based on our query pattern.

In this recipe, you will see how the Amazon Redshift compound sort key works.

Getting ready

To complete this recipe, you will need:

- An IAM user with access to Amazon Redshift
- An Amazon Redshift data warehouse deployed in the AWS Region eu-west-1
- Amazon Redshift data warehouse master user credentials
- Access to any SQL interface, such as a SQL client or Amazon Redshift Query Editor V2
- An IAM role attached to an Amazon Redshift data warehouse that can access Amazon S3; we will refer to it in the recipes with `[Your-Redshift_Role]`
- An AWS account number; we will refer to it in the recipes with `[Your-AWS_Account_Id]`

How to do it...

In this recipe, we will use the `lineitem` table with sort keys and analyze the performance queries:

1. Connect to the Amazon Redshift data warehouse using a SQL client or Query Editor V2.
2. Let's create the `lineitem` table with the default `AUTO` sort key using the following, replacing `[Your-AWS_Account_Id]` and `[Your-Redshift_Role]` with the respective values:

```
drop table if exists lineitem;
CREATE TABLE lineitem
(
  L_ORDERKEY        BIGINT NOT NULL,
  L_PARTKEY         BIGINT,
  L_SUPPKEY         BIGINT,
  L_LINENUMBER      INTEGER NOT NULL,
  L_QUANTITY        DECIMAL(18,4),
  L_EXTENDEDPRICE   DECIMAL(18,4),
  L_DISCOUNT        DECIMAL(18,4),
  L_TAX             DECIMAL(18,4),
  L_RETURNFLAG      VARCHAR(1),
  L_LINESTATUS      VARCHAR(1),
  L_SHIPDATE        DATE,
  L_COMMITDATE      DATE,
  L_RECEIPTDATE     DATE,
  L_SHIPINSTRUCT    VARCHAR(25),
  L_SHIPMODE        VARCHAR(10),
  L_COMMENT         VARCHAR(44)
```

```
)
distkey (L_ORDERKEY);
COPY lineitem from 's3://packt-redshift-cookbook/lineitem/' iam_role
'arn:aws:iam::[Your-AWS_Account_Id]:role/[Your- Redshift_Role]'  CSV
gzip COMPUPDATE PRESET;
```

Note

Depending on the size of the data warehouse, the `COPY` command will take around 15 minutes due to the size of the data.

3. Let's verify the sort key of the `lineitem` table with the default auto sort key using the following query:

```
select "schema", "table", "diststyle", skew_rows, sortkey1, unsorted
from svv_table_info
where "table" = 'lineitem';
```

Here is the expected output:

schema	table	diststyle	skew_rows	sortkey1	unsorted
public	lineitem	KEY(l_orderkey)	1.01	AUTO(SORTKEY)	NULL

Figure 8.10 – Output of the preceding query

As shown in the preceding output, the `lineitem` table is set with a sort key of `AUTO(sortkey)`. Amazon Redshift Advisor, based on your workload pattern, will make a recommendation and the automatic table optimization will alter the table with the optimal sort key.

4. To see the effectiveness of block pruning using the sort key, run the following query and note down the `query_id`:

```
SELECT
    l_returnflag,
    l_linestatus,
    sum(l_quantity) as sum_qty,
    sum(l_extendedprice) as sum_base_price,
    sum(l_extendedprice * (1 - l_discount)) as sum_disc_price,
    count(*) as count_order
FROM
    lineitem
```

```
WHERE
      l_shipdate = '1992-01-10'
GROUP BY
    l_returnflag,
    l_linestatus
ORDER BY
    l_returnflag,
    l_linestatus;
select last_user_query_id() as query_id;
Here is the expected output:
query_id
5409
```

Note

Amazon Redshift captures operational statistics of each query step in system tables. Details of `SVL_QUERY_SUMMARY` can be found at this link: `https://docs.aws.amazon.com/redshift/latest/dg/r_SVL_QUERY_SUMMARY.html`.

5. Run the following query to measure the effectiveness of the sort key for the above query, replacing `[query_id]`, with the output from the preceding step:

```
SELECT query_id, step_id, table_name, is_rrscan, input_rows, output_
rows
from sys_query_detail where query_id in (5409)
and table_name like '%lineitem%'
order by query_id,step_id;
```

Here is the expected output:

query_id	step_id	table_name	is_rrscan	input_rows	output_rows
5409	0	dev.public.lineitem	t	173634575	18385

Figure 8.11 – Output of the preceding query

`input_rows` indicates that Amazon Redshift was effectively able to use the sort key to pre-filter records. `is_rrscan` is true for these range scans. Amazon Redshift automatically leverages zone maps to prune out the blocks that do not match the filter criteria of the query.

6. Let's alter the `lineitem` table to add the column `l_shipdate` as the sort key. Most of the queries we will run will use `l_shipdate` as the filter. `l_shipdate` is a low-cardinality column:

```
alter table lineitem alter sortkey (L_SHIPDATE);
```

Note

Depending on the size of the data warehouse, the `ALTER` statement will take around 15 minutes to complete due to the size of the data.

7. To see the effectiveness of the sort key, run the following query and capture the query ID:

```
SELECT
    l_returnflag,
    l_linestatus,
    sum(l_quantity) as sum_qty,
    sum(l_extendedprice) as sum_base_price,
    sum(l_extendedprice * (1 - l_discount)) as sum_disc_price,
    count(*) as count_order
FROM
    lineitem
WHERE
      l_shipdate = '1992-01-10'
GROUP BY
    l_returnflag,
    l_linestatus
ORDER BY
    l_returnflag,
    l_linestatus;
select last_user_query_id() as query_id_1;
Here is the expected output:
```

```
query_id_1
5596
```

8. Run the following query to measure the effectiveness of the sort key for the above query, replacing [query_id_1] with the output from the preceding step:

```
SELECT query_id, step_id, table_name, is_rrscan, input_rows, output_
rows
from sys_query_detail where query_id in ([query_id_1])
and table_name like '%lineitem%'
order by query_id,step_id;
```

Here is the expected output:

query_id	step_id	table_name	is_rrscan	input_rows	output_rows
5596	0	dev.public.lineitem	t	2033144	18385

Figure 8.12 – Output of the preceding query

With the sort key `l_shipdate` applied to the table, block pruning is very effective, reducing the pre-filtered records to 2,033,144.

9. Now, let's modify the query to cast the `l_shipdate` column to the varchar data type and then apply the filter. Run the following modified query and capture the `query_id_2` output:

```
set enable_result_cache_for_session = off;

SELECT
    l_returnflag,
    l_linestatus,
    sum(l_quantity) as sum_qty,
    sum(l_extendedprice) as sum_base_price,
    sum(l_extendedprice * (1 - l_discount)) as sum_disc_price,
    count(*) as count_order
FROM
    lineitem
WHERE
      cast(l_shipdate as varchar(10) ) = '1992-01-10'
GROUP BY
    l_returnflag,
    l_linestatus
```

```
ORDER BY
    l_returnflag,
    l_linestatus;

select last_user_query_id() as query_id_2;
---expected sample output---
query_id_2
5759
```

10. Now, let's run the following query to analyze the effectiveness of the sort key columns, replacing `[query_id_1]` and `[query_id_2]` with the query IDs from the preceding steps:

```
SELECT query, step, label, is_rrscan, rows, rows_pre_filter, is_
diskbased
from svl_query_summary where query in ([query_id_1],[ query_id_2])
and label like '%lineitem%'
order by query,step;
```

Here is the expected output:

query_id	step_id	table_name	is_rrscan	input_rows	output_rows
5409	0	dev.public.lineitem	t	173634575	18385
5759	0	dev.public.lineitem	t	599037902	18385

Figure 8.13 – Output of the preceding query

`[query_id_1]`, which used the `l_shipdate` filter, has an `input_rows` value of 173634575, compared to `[query_id_2]`, which used the `pre-filtered` and has an input_rows value of 599037902, which means a full table scan was carried out. To make sort keys effective, it is best practice to avoid applying functions or casting to sort key columns.

How it works...

Using the sort keys when creating tables allows efficient range-restricted scans of the data, when the sort key is referenced in the `where` conditions. Amazon Redshift automatically leverages the in-memory metadata to prune out the blocks. Sort keys make the zone maps more pristine. Applying sort keys on the most commonly used columns as filters in a query can significantly reduce the I/O and hence optimize query performance for workloads of any scale. You can learn more about sort keys at `https://docs.aws.amazon.com/redshift/latest/dg/t_Sorting_data.html`.

Analyzing and improving queries for provisioned clusters

The default table encoding, sort key, and distribution key in Amazon Redshift is `AUTO`. Amazon Redshift can learn from the workloads and automatically set the right encoding and sort and distribution style, which are the biggest contributors to table design and optimization. Amazon Redshift also provides insights into the query plan, which helps to optimize the queries when authoring them. It lists the detailed steps it takes to fetch the data.

Getting ready

To complete this recipe, you will need:

- An IAM user with access to Amazon Redshift
- An Amazon Redshift provisioned cluster deployed in the AWS Region eu-west-1
- Amazon Redshift provisioned cluster master user credentials
- Access to any SQL interface, such as a SQL client or Amazon Redshift Query Editor V2
- An IAM role attached to an Amazon Redshift provisioned cluster that can access Amazon S3; we will refer to it in the recipes with `[Your-Redshift_Role]`
- An AWS account number; we will refer to it in the recipes with `[Your-AWS_Account_Id]`

How to do it...

In this recipe, we will use the retail system dataset from *Chapter 3*, *Loading and Unloading Data*, to perform analytical queries and explore opportunities to optimize them:

1. Connect to the Amazon Redshift provisioned cluster using any SQL interface, such as a SQL client, and run `EXPLAIN` on a query:

```
explain
SELECT o_orderstatus,
       COUNT(o_orderkey) AS orders_count,
       SUM(l_quantity) AS quantity,
       MAX(l_extendedprice) AS extendedprice
FROM lineitem
  JOIN orders ON l_orderkey = o_orderkey
WHERE
  L_SHIPDATE = '1992-01-29'
GROUP BY o_orderstatus;
```

Here is the expected output:

```
QUERY PLAN
------------------------------------------------------------------------
---
 XN HashAggregate  (cost=97529596065.20..97529596065.22 rows=3
width=36)
   ->  XN Hash Join DS_BCAST_INNER  (cost=3657.20..97529594861.20
rows=120400 width=36)
         Hash Cond: ("outer".o_orderkey = "inner".l_orderkey)
         ->  XN Seq Scan on orders  (cost=0.00..760000.00
rows=76000000 width=13)
         ->  XN Hash  (cost=3047.67..3047.67 rows=243814 width=31)
               ->  XN Seq Scan on lineitem  (cost=0.00..3047.67
rows=243814 width=31)
                     Filter: (l_shipdate = '1992-01-29'::date)
```

As noted in the above output, the `explain` command provides insights into the steps taken by the query. As you can see above, the `lineitem` and `orders` table are joined using a hash join. Each step also provides the relative cost to review the expensive steps in the query for optimization.

Note

Please also see `https://docs.aws.amazon.com/redshift/latest/dg/c-query-planning.html` for a step-by-step illustration of the query planning and execution steps.

2. Now, run the analytical query using the following command to capture the `query_id` for analysis:

```
SELECT o_orderstatus,
       COUNT(o_orderkey) AS orders_count,
       SUM(l_quantity) AS quantity,
       MAX(l_extendedprice) AS extendedprice
FROM lineitem
  JOIN orders ON l_orderkey = o_orderkey
WHERE L_SHIPDATE = '1992-01-29'
```

```
GROUP BY o_orderstatus;
select last_user_query_id() as query_id;
```

Here is the expected output:

```
query_id
24580051
```

Make note of the above `query_id`, which will be used in later steps to analyze the query.

3. Run the following command to analyze the effectiveness of the sort key column on the `lineitem` table by replacing [`query_id`] with the query ID from the previous step:

```
SELECT step, label, is_rrscan, rows, rows_pre_filter, is_diskbased
from svl_query_summary where query = [query_id]
order by step;
```

Here is the expected output:

```
step |                  label                  | is_rrscan |  rows
| rows_pre_filter | is_diskbased
------+-----------------------------------------+-----------+-----
---+-----------------+-------------
    0 | scan   tbl=1450056 name=lineitem         | t         |
57856 |       599037902 | f
    0 | scan   tbl=361382 name=Internal Worktable | f         |
1 |              0 | f
    0 | scan   tbl=1449979 name=orders           | t         |
79119 |        76000000 | f
    0 | scan   tbl=361380 name=Internal Worktable | f         |
173568 |               0 | f
    0 | scan   tbl=361381 name=Internal Worktable | f         |
32 |              0 | f
```

As can be noticed from the above output, the query optimizer is able to effectively make use of the range-restricted scan (`is_rrscan`) on the `l_shipdate` column in the `lineitem` table, to reduce the number of rows from `599037902` to `57856`. Also, none of the steps spill to disk, as indicated by `is_diskbased = f`.

4. Amazon Redshift provides consolidated alerts from the query execution to prioritize the analysis effort. You can run the following query to view the alerts from the query execution:

```
select event, solution
from stl_alert_event_log
where query in (24580051);
```

Here is the expected output:

```
Very selective query filter:ratio=rows(2470)/rows_pre_user_
filter(2375000)=0.001040   Review the choice of sort key to enable
range restricted scans, or run the VACUUM command to ensure the
table is sorted
```

In the above query output, since we already confirmed that sort keys are being used effectively, performing `VACUUM` will ensure data is sorted and range-restricted scans can be more effective.

5. Another alert that you can see in the **stl_alert_event_log** is **Statistics for the tables in the query are missing or out of date**. To fix this issue, you can run the `analyze` query as follows:

```
analyze lineitem;
```

Here is the expected output:

```
ANALYZE run successfully
```

Now, `lineitem` is updated with the current statistics that will enable the optimizer to pick an optimal plan.

How it works...

Amazon Redshift automates performance tuning as part of the managed service. This includes automatic vacuum delete, automatic table sort, automatic analyze, and Amazon Redshift Advisor for actionable insights into optimizing cost and performance. These capabilities are enabled through an ML model that can learn from your workloads to generate and apply precise high-value optimizations. You can read more about automatic table optimization here: `https://aws.amazon.com/blogs/big-data/automate-your-amazon-redshift-performance-tuning-with-automatic-table-optimization/`.

Configuring Workload Management (WLM) for provisioned cluster

Amazon Redshift **workload management (WLM)** enables you to set up query priorities in a data warehouse. WLM helps you create query queues that can be defined based on different parameters, such as memory allotment, priority, user groups, query groups, and query monitoring rules. Users generally use WLM to set priorities for different query types, such as long-running versus short-running, ETL versus reporting, and so on. In this recipe, we will demonstrate how to configure WLM within a Redshift data warehouse. Hence, you can manage multiple workloads running on the same data warehouse and each of them can be assigned different priorities based on business needs.

Note

Amazon Redshift serverless automatically manages WLM. You do not need to configure WLM on a serverless endpoint.

Getting ready

To complete this recipe, you will need the following setup:

- An IAM user with access to Amazon Redshift
- An Amazon Redshift provisioned cluster deployed in the AWS Region eu-west-1

How to do it...

In this recipe, we will configure the WLM for your data warehouse using the AWS Management Console:

1. Open the Amazon Redshift console: `https://console.aws.amazon.com/redshiftv2/home`.

2. In the left-hand toolbar, browse to **CONFIG** and select **Workload management**.

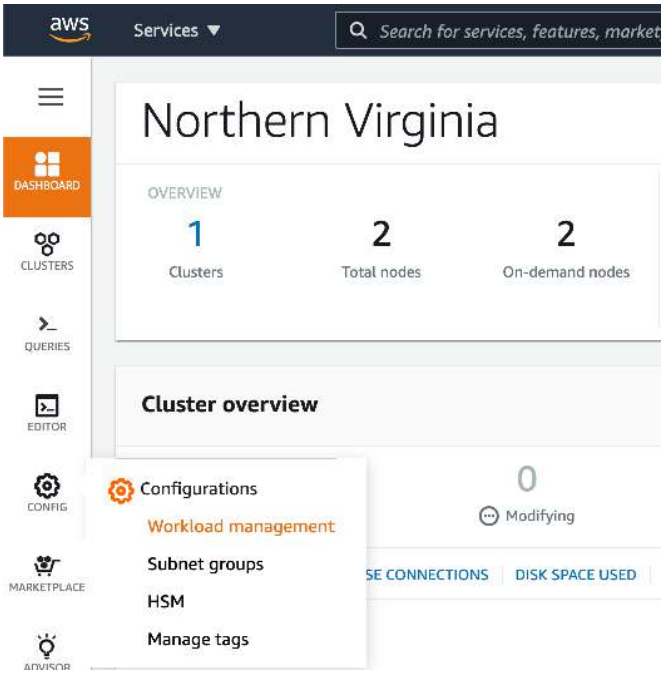


Figure 8.14 – Navigate to Workload management in the AWS Redshift console

3. On the **Workload management** page, we will need to create a new parameter group by clicking the **Create** button.

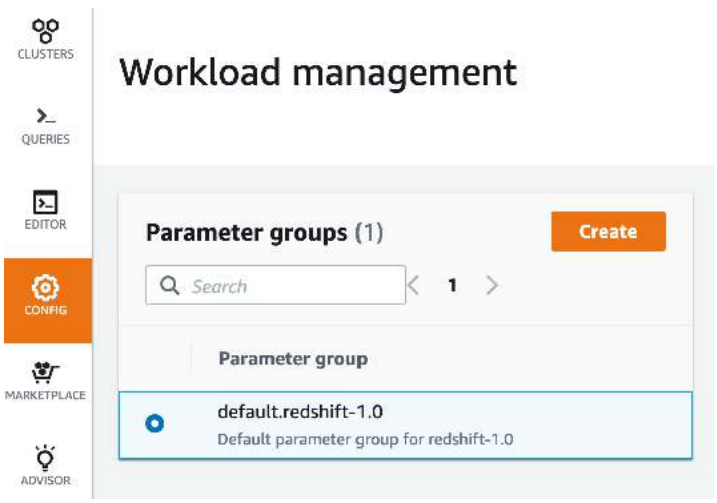


Figure 8.15 – Configure a new parameter group

4. A **Create parameter group** popup will open. Enter a **Parameter group name** and **Description**. Click on **Create** to finish creating a new parameter group.

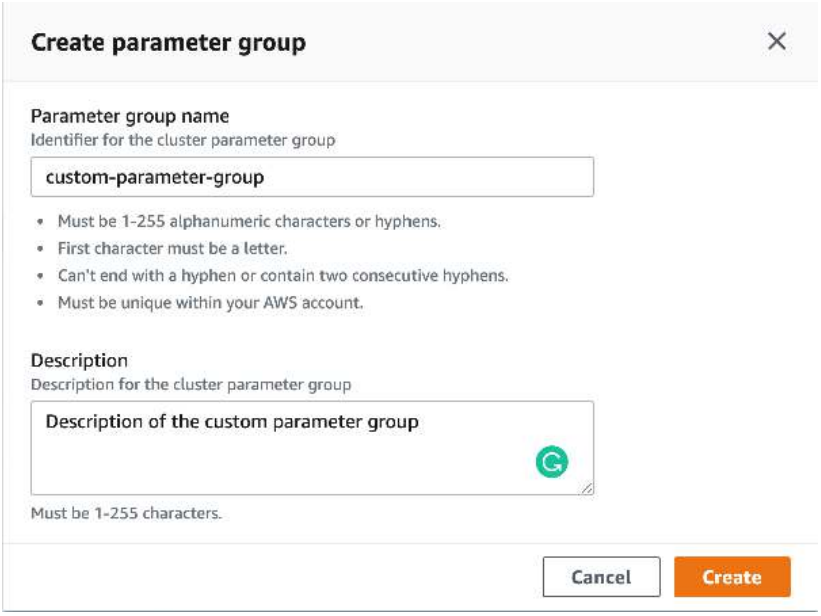


Figure 8.16 – Create a new parameter group called custom-parameter-group

5. By default, **Automatic WLM** is configured under **Workload Management**. **Automatic WLM** is recommended, and it calculates the optimal memory and concurrency for query queues.
6. To create a new queue, click on **Edit workload queues** under the **Workload queues** section. On the **Modify workload queues: custom-parameter-group** page, click on **Add queue**.
7. You can configure the queue name by replacing the `Queue 1` string and configuring other settings, such as **Concurrency scaling mode** to **auto** or **off** and **Query priority** to one of five levels ranging from lowest to highest. Additionally, you can include the user groups, user roles, or query groups that need to be routed to this specific queue.

For example, we created an `ETL` queue, with concurrency scaling disabled, a query priority of **Normal**, and a user role of `data_engineers`. The `load` and `transform` query groups will be routed to this queue.

Figure 8.17 – Configure the ETL queue on the parameter group

8. You can repeat steps 6 and 7 to create a total of eight queues.
9. You can create query monitoring rules by selecting either **Add rule from template** or **Add custom rule**. This allows you to perform a log, abort, or change query priority action based on the predicates for the given query monitoring metrics.

   For example, we created a rule to abort the query if more than 100 million rows are returned.

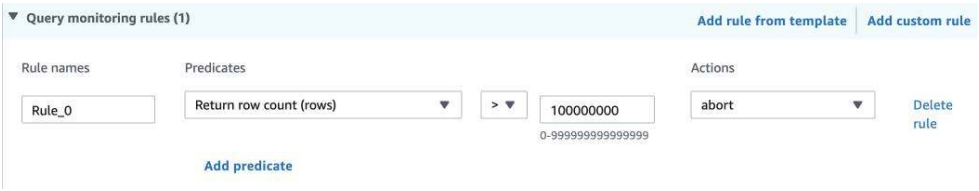


Figure 8.18 – Configure a query monitoring rule

10. To finish the configuration of the WLM settings, browse to the bottom of the page and click **Save**.

11. To apply the new WLM settings to the cluster, browse to **CLUSTERS** and choose the checkbox beside the Amazon Redshift data warehouse to which you want to apply the new WLM settings. Go to **Actions** and select **Modify**.

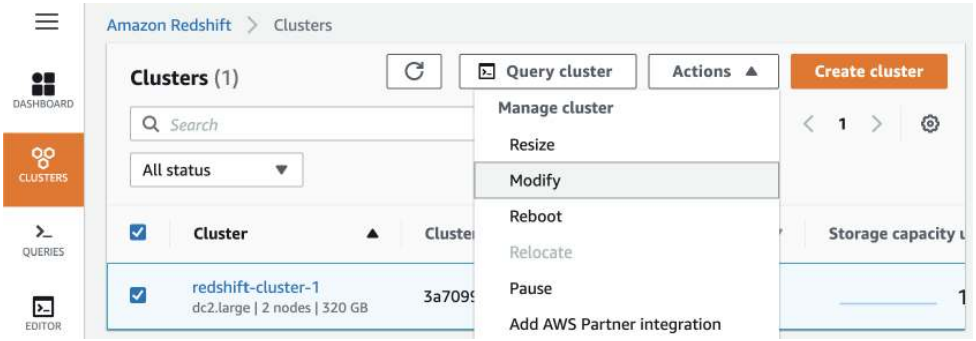


Figure 8.19 – Apply custom-parameter-group to your data warehouse

12. Under the **Modify** page, browse to the second set of **Database configurations**. Drop down **Parameter groups** and select the newly created parameter group.
13. Go to the bottom of the page and select **Modify Cluster**. The changes are in the queue and applied after the data warehouse is rebooted.
14. To reboot the data warehouse at an appropriate time that suits the business, click the checkbox beside the Amazon Redshift data warehouse, go to **Actions**, and select **Reboot**. A popup will appear to confirm the reboot. Select **Reboot Cluster**.

How it works...

Amazon Redshift's WLM settings allow you to set the workload priorities and concurrency of different types of workloads that run on an Amazon Redshift data warehouse. In addition, you can apply Auto WLM (recommended), which manages the short query acceleration, memory allotment, and concurrency automatically. Using manual WLM, you can configure the memory and concurrency values for your workloads if needed (not recommended).

Utilizing concurrency scaling for provisioned clusters

The concurrency scaling feature in Amazon Redshift allows you to support concurrent users and queries for steady query performance. Amazon Redshift utilizes resources that are available in a data warehouse to maximize throughput for an analytical query. Amazon Redshift uses WLM to optimize query execution by running a limited number of queries concurrently, prioritizing those that can complete quickly and managing the remaining queries to ensure efficient resource utilization and overall performance.

With the concurrency scaling feature turned on, Amazon Redshift is able to instantly bring up additional redundant data warehouses to run the queued-up queries and support burst traffic into the data warehouse. The redundant data warehouses are automatically shut down once the queries complete/there are no more queries waiting in the queue.

Note

In Amazon Redshift, serverless concurrency scaling is enabled by default. You do not need to configure concurrency scaling on a serverless endpoint.

Getting ready

To complete this recipe, you will need:

- An Amazon Redshift provisioned cluster deployed in the AWS Region eu-west-1 with the retail system dataset from *Chapter 3*, using the *Loading data from Amazon S3 using COPY* recipe
- Amazon Redshift provisioned cluster master user credentials
- Access to any SQL interface, such as a SQL client or Amazon Redshift Query Editor V2
- Install the client tool `par_psql` (`https://github.com/gbb/par_psql`) and `psql` (`https://docs.aws.amazon.com/redshift/latest/mgmt/connecting-from-psql.html`) on a Linux machine that can connect to an Amazon Redshift data warehouse

How to do it...

In this recipe, we will use the `par_psql` (`https://github.com/gbb/par_psql`) tool to run parallel queries on Amazon Redshift to simulate a concurrent workload:

1. Navigate to the Amazon Redshift console and then to **Amazon Redshift** > **CLUSTERS** > your Amazon Redshift data warehouse. Click on the **Properties** tab and scroll down to **Database configurations**, as shown in the following image:

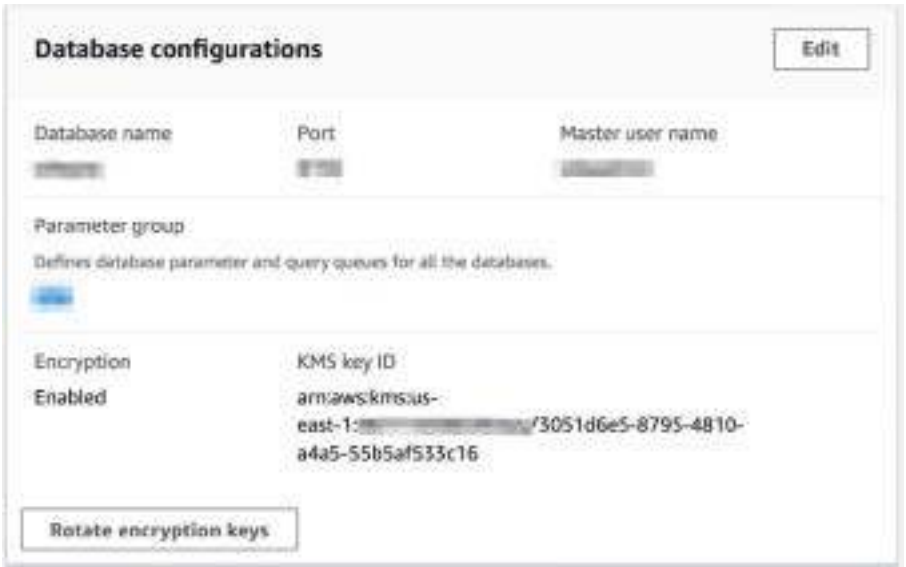


Figure 8.20 – Database configurations

2. Select the parameter group associated with the Amazon Redshift data warehouse.

3. Verify that **max_concurrency_scaling_data warehouses** is set to a value > =1 and **Workload queues** has **Concurrency scaling mode** set to **auto**, as shown below:

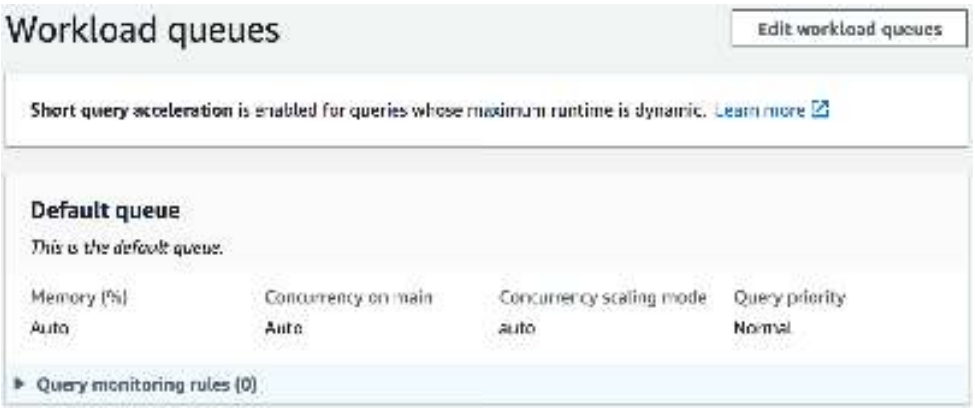


Figure 8.21 – Workload queues

For a step-by-step guide to setting concurrency scaling, refer to the *Configuring Workload Management (WLM) for provisioned cluster* recipe of this chapter.

4. Download the `par_psql` script from the following GitHub location: `https://github.com/PacktPublishing/Amazon-Redshift-Cookbook-2E/blob/main/Chapter08/conc_scaling.sql`. Copy it into the same location that `par_psql` is installed. This script uses the retail system dataset, mentioned in the *Getting started* section.
5. Run the following command using the SQL client to capture the `starttime` of the test:

```
select sysdate as starttime
```

Here is the expected output:

```
starttime
2020-12-04 16:10:43
```

6. Run the following command on the Linux box to simulate 100 concurrent query runs:

```
export PGPASSWORD=[PASSWORD]
./par_psql --file=conc_scaling.sql -h [YOUR AMAZON REDSHIFT HOST] -p
[PORT] -d [DATABASE_NAME] -U [USER_NAME]
```

7. Wait until all the queries have completed. Run the following query to analyze the query execution, replacing `[starttime]` with the value corresponding to the date and time at the start of the script execution:

```
SELECT w.service_class AS queue
     , case when q.concurrency_scaling_status = 1 then 'Y' else 'N'
end as conc_scaled
     , COUNT( * ) AS queries
     , SUM( q.aborted )  AS aborted
     , SUM( ROUND( total_queue_time::NUMERIC / 1000000,2 ) ) AS
queue_secs
     , SUM( ROUND( total_exec_time::NUMERIC / 1000000,2 ) )  AS
exec_secs
FROM stl_query q
     JOIN stl_wlm_query w
          USING (userid,query)
WHERE q.userid > 1
  AND q.starttime > '[starttime]'
GROUP BY 1,2
ORDER BY 1,2;
```

Here is the expected output:

```
queue | conc_scaled | queries | aborted | queue_secs | exec_secs
-------+-------------+---------+---------+------------+-----------
     9 | N           |      75 |       0 |    3569.83 |     31.24
     9 | Y           |      25 |       0 |    0.0     |     10.97
```

As can be noticed from the above output, Amazon Redshift was able to take advantage of the concurrency scaling feature to run 25% of the queries on the burst data warehouse.

How it works...

Concurrency scaling in Amazon Redshift automatically and elastically scales query processing power to handle peak workloads, ensuring fast performance and preventing query delays by routing eligible queries to concurrency scaling clusters when a WLM queue's concurrency exceeds defined limits. You can find more details on the queries that are eligible for concurrency scaling here: `https://docs.aws.amazon.com/redshift/latest/dg/concurrency-scaling.html`.

Optimizing Spectrum queries for provisioned clusters

Amazon Redshift Spectrum allows you to extend your Amazon Redshift data warehouse to use SQL queries on data that is stored in Amazon S3. Optimizing Amazon Redshift Spectrum queries allows optimal throughputs for the SQL queries, as well as saving on the costs associated with them. In this recipe, we will demonstrate techniques to get insights into the performance of Spectrum-based queries and optimize them.

Getting ready

To complete this recipe, you will need:

- An IAM user with access to Amazon Redshift and Amazon S3
- An Amazon Redshift data warehouse deployed in the AWS Region eu-west-1
- Amazon Redshift data warehouse master user credentials
- Access to any SQL interface, such as a SQL client or Amazon Redshift Query Editor V2
- An IAM role attached to an Amazon Redshift data warehouse that can access Amazon S3; we will refer to it in the recipes with `[Your-Redshift_Role]`
- An AWS account number; we will refer to it in the recipes with `[Your-AWS_Account_Id]`

How to do it...

In this recipe, we will use the Amazon.com customer product reviews dataset (refer to *Chapter 3*, *Loading data from Amazon S3 using COPY* recipe) to demonstrate getting insights into Spectrum SQL performance and tuning it:

1. Open any SQL client tool or Redshift Query Editor V2 and connect to Amazon Redshift. Create a schema to point to the `reviews` dataset using the following command by replacing the `[Your-AWS_Account_Id]` and `[Your-Redshift_Role]` values:

```
CREATE external SCHEMA reviews_ext_schema
FROM data catalog DATABASE 'reviews_ext_schema'
iam_role 'arn:aws:iam::[Your-AWS_Account_Id]:role/[Your-Redshift_
Role]'
CREATE external DATABASE if not exists;
```

2. Using the `reviews` dataset, create the Parquet version of the external tables using the command below:

```
CREATE external TABLE reviews_ext_schema.amazon_product_reviews_
parquet(
  marketplace varchar(2),
  customer_id varchar(32),
  review_id varchar(24),
  product_id varchar(24),
  product_parent varchar(32),
  product_title varchar(512),
  star_rating int,
  helpful_votes int,
  total_votes int,
  vine char(1),
  verified_purchase char(1),
  review_headline varchar(256),
  review_body varchar(max),
  review_date date,
  year int)
stored as parquet
location 's3://packt-redshift-cookbook/reviews_parquet/';
```

3. Using the `reviews` dataset, create the plain-text file (tab-delimited) version of the external tables using the command below:

```
CREATE external TABLE reviews_ext_schema.amazon_product_reviews_tsv(
  marketplace varchar(2),
  customer_id varchar(32),
  review_id varchar(24),
  product_id varchar(24),
  product_parent varchar(32),
  product_title varchar(512),
  star_rating int,
  helpful_votes int,
  total_votes int,
  vine char(1),
  verified_purchase char(1),
  review_headline varchar(256),
```

```
  review_body varchar(max),
  review_date date,
  year int)
row format delimited
fields terminated by '\t'
stored as textfile
location 's3://packt-redshift-cookbook/reviews_tsv/';
```

4. Run the following analytical queries to calibrate the throughputs and note down the `parquet_query_id` and `tsv_query_id` outputs:

```
SELECT verified_purchase,
       SUM(total_votes) total_votes,
       avg(helpful_votes) avg_helpful_votes,
       count(customer_id) total_customers
FROM reviews_ext_schema.amazon_product_reviews_parquet
WHERE review_headline = 'Y'
GROUP BY verified_purchase;

select PG_LAST_QUERY_ID() as parquet_query_id;

SELECT verified_purchase,
       SUM(total_votes) total_votes,
       avg(helpful_votes) avg_helpful_votes,
       count(customer_id) total_customers
FROM reviews_ext_schema.amazon_product_reviews_tsv
WHERE review_headline = 'Y'
GROUP BY verified_purchase;

select PG_LAST_QUERY_ID() as tsv_query_id;
```

5. Analyze the performance of both of these queries using the following command by substituting `[parquet_query_id]` and `[tsv_query_id]` with the values from the previous step:

```
select query, segment, elapsed as elapsed_ms, s3_scanned_rows, s3_
scanned_bytes, s3query_returned_rows, s3query_returned_bytes, files
from svl_s3query_summary
where query in ([parquet_query_id], [tsv_query_id])
order by query,segment ;
```

Here is the expected output:

```
query,elapsed_ms,s3_scanned_rows,s3_scanned_bytes,s3query_returned_
rows,s3query_returned_bytes,files
parquet_query_id        3000554    5906460    142428017    4    1917
10
tsv_query_id        9182604    5906460    2001945218    4    5222
10
```

As you can notice above, the TSV version of the datasets took 9 seconds, compared to the 3 seconds it took in Parquet, since the TSV version had to scan 2 GB of data while the Parquet format scanned 0.14 MB of data, despite the content of the files being the same.

Having the data in a columnar format such as Parquet improves the query throughput and reduces the costs incurred with the query as the scan being carried out on the dataset is most optimal.

How it works...

Optimizing Amazon Redshift Spectrum queries works on the principle of reducing the Amazon S3 scan and pushing down operations as much as possible into the scalable Spectrum engine.

This can be achieved by using the following techniques:

- Amazon Redshift Spectrum supports structured and semi-structured data formats such as Avro, Parquet, ORC, file, JSON, etc., and using a columnar file format like Parquet or ORC can reduce I/O by reading only the needed columns.
- Compress the row format file, e.g., textfile, with compression such as gzip, Snappy, or bzip to save costs and allow faster performance.
- Use the optimal file size:
    - Avoid excessive small files (<1 MB)
    - Avoid large files (1 GB) if the file format is not splittable, e.g., gzip/Snappy compressed text file
- Organize the files as partitions. Take advantage of partition pruning to save costs when running the query.

You can read more about optimization techniques here: `https://aws.amazon.com/blogs/big-data/10-best-practices-for-amazon-redshift-spectrum/`.

9

Cost Optimization

Amazon Redshift allows you to operate your data warehouse from a few gigabytes to a petabyte in a way that is simple to manage and cost effective. The cost is predictable even with unpredictable workloads and provides up to 7x better price performance than any other data warehouse, at just $1,000 per terabyte per year.

Amazon Redshift provides flexible pricing options, both on demand and reserved for provisioned clusters. With Reserved Instance pricing, you save up to 63% by committing to a 1-year or 3-year term. There are multiple cost controls you can choose from to manage your Redshift serverless spend. There are several best practices you can follow to ensure you're getting the best value with Amazon Redshift. This chapter also discusses some of the common cost optimization methods to get the best cost performance.

The following recipes will be covered in this chapter:

- AWS Trusted Advisor
- Amazon Redshift Reserved Instance pricing
- Scheduling pause and resume for an Amazon Redshift provisioned cluster
- Scheduling elastic resizing for an Amazon Redshift provisioned cluster
- Using cost controls to set actions for Spectrum
- Using cost controls to set actions for concurrency scaling for an Amazon provisioned cluster
- Using cost controls for Redshift serverless

Technical requirements

Here are the technical requirements to complete the recipes in this chapter:

- Access to the AWS Console.
- An AWS administrator should create an IAM user by following *Recipe 1* in the *Appendix*. This IAM user will be used in some of the recipes in this chapter.
- The AWS administrator should deploy the AWS CloudFormation template at `https://github.com/PacktPublishing/Amazon-Redshift-Cookbook-2E/blob/main/Chapter09/chapter_9_CFN.yaml` and create one IAM policy and one IAM role:
    - An IAM policy attached to the IAM user that will give them access to Amazon Redshift, AWS Secrets Manager, Amazon CloudWatch, Amazon CloudWatch Logs, AWS KMS, AWS Glue, Amazon EC2, AWS Trusted Advisor, AWS Billing, AWS Cost Explorer, and Amazon S3.
    - An IAM role with the ability to schedule pause and resume, and elastic resizing for a Redshift cluster. We will refer to this as `Chapter9RedshiftSchedulerRole`.
- An Amazon Redshift cluster deployed in the eu-west-1 AWS region.

AWS Trusted Advisor

AWS Trusted Advisor provides you with a summarized dashboard and detailed real-time guidance to help you provision your resources following AWS best practices. Its checks help you to optimize your AWS infrastructure, reduce your overall costs, and increase security and performance, and it also monitors your service limit.

AWS Trusted Advisor provides cost optimization checks for unutilized Amazon Redshift clusters. It also provides cost optimization checks for the on-demand Amazon Redshift clusters that can benefit from Reserved Instance pricing, providing you with significant savings.

Getting ready

To complete this recipe, you will need:

- An IAM user with access to Amazon Redshift and AWS Trusted Advisor
- An Amazon Redshift cluster deployed in the eu-west-1 AWS region

How to do it...

In this recipe, we will use AWS Trusted Advisor to identify opportunities for potential savings:

1. Navigate to the AWS Management Console and select **AWS Trusted Advisor**. On the dashboard, you will see a summary of the checks for cost optimization with potential monthly savings:

Figure 9.1 – AWS Trusted Advisor Dashboard

2. To further drill down into the details of cost optimization, select **Cost Optimization** from the left pane. If the Amazon Redshift clusters are underutilized, it will list the clusters with the corresponding cost. You can choose to pause the clusters or delete the clusters for savings on on-demand clusters:

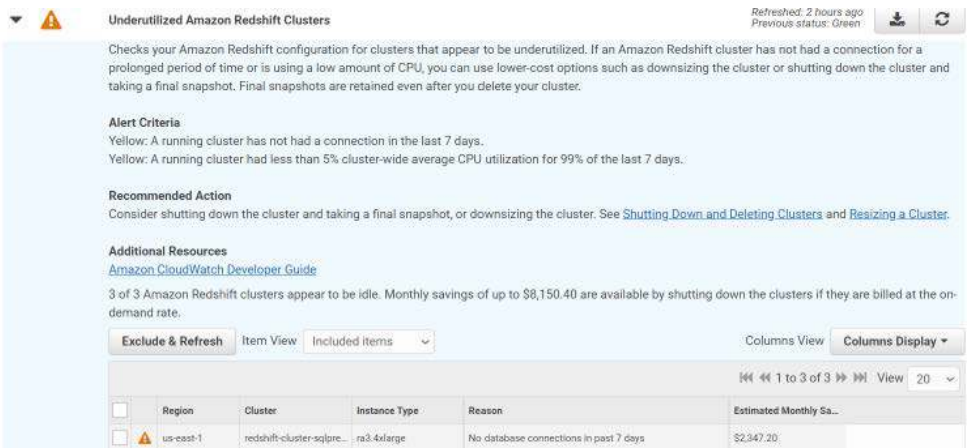


Figure 9.2 – Cost optimization recommendations

3. Cost optimization recommendation results in potential savings with Reserved Instances for the on-demand cluster. This is based on the usage for the past 30 days:

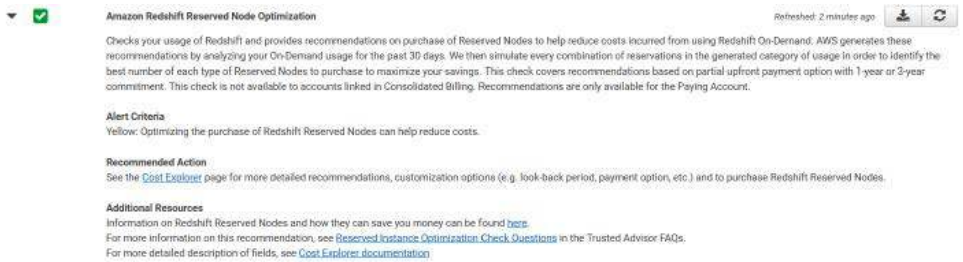


Figure 9.3 – Amazon Redshift cost optimization opportunities

4. To view the potential cost savings, navigate to Cost Explorer from the Management Console. Choose **Recommendations** under **Reservations**. The recommendations are to use Reserved Instances instead of on-demand, which would result in potential savings of **34%**:

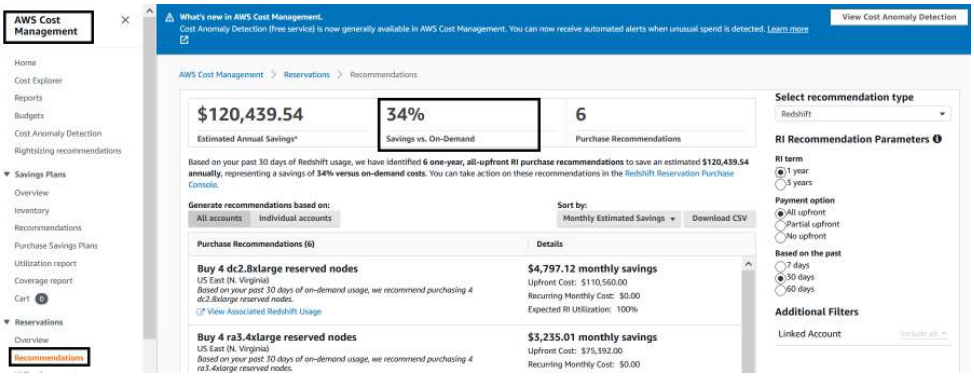


Figure 9.4 – Amazon Redshift cost optimization recommendation

We will dive deeper into the potential savings with Reserved Instance pricing in the next recipe.

How it works...

AWS Trusted Advisor is an application that infers best practices based on operational data derived from thousands of AWS customers. These checks fall into categories such as cost optimization, security, fault tolerance, performance, and service limits. For a full list of checks, visit `https://aws.amazon.com/premiumsupport/technology/trusted-advisor/best-practice-checklist/`.

Amazon Redshift Reserved Instance pricing

Amazon Redshift Reserved Instance pricing is a billing construct that results in significant savings for on-demand provisioned clusters that are utilized 24x7. To get deep discounts on the cluster for your data warehouse workload, you can reserve your instances. Once you have determined the size and number of clusters for your workload, you can purchase **Reserved Instances** with discounts from 30% to 63% compared to on-demand pricing.

Reserved Instances can be purchased using full upfront, partial upfront, or sometimes a no upfront payment plan. Reserved Instances can be purchased for one or three years. Reserved Instances are not tied to a particular cluster; they can be pooled across clusters in your account. The following table shows the significant cost optimization you can get by using Reserved Instance pricing with one year and three years for different instances for the upfront payment option:

Instance type	**Managed storage limit per node**	**Memory (GB)**	**vCPUs**	**Reserved Instance discount***	
				1-Year	3-Year
ra3.16xlarge	128 TB	384	48	34%	63%
ra3.4xlarge	128 TB	96	12	34%	63%
ra3.xlplus	32 TB	32	4	34%	62%
ra3.large	8 TB	16	2	34%	63%

Table 9.1 – Representative Reserved Instance savings

Please see `https://aws.amazon.com/redshift/pricing/` for the latest pricing and savings.

In this recipe, we will use Cost Explorer to see the significant cost savings using Reserved Instances for an existing on-demand cluster. Then, using the Amazon Redshift console, we will dive into how to purchase the reserved nodes.

Getting ready

To complete this recipe, you will need:

- An IAM user with access to Amazon Redshift, AWS Billing, and AWS Cost Explorer
- An Amazon Redshift provisioned cluster deployed in the eu-west-1 AWS region

How to do it...

1. Navigate to the AWS Management Console and select **Cost Explorer**.
2. On the left side, choose **Recommendations** under **Reservation**. By selecting a **1 year** Reserved Instance with **All upfront**, you get **34%** savings compared to on-demand:

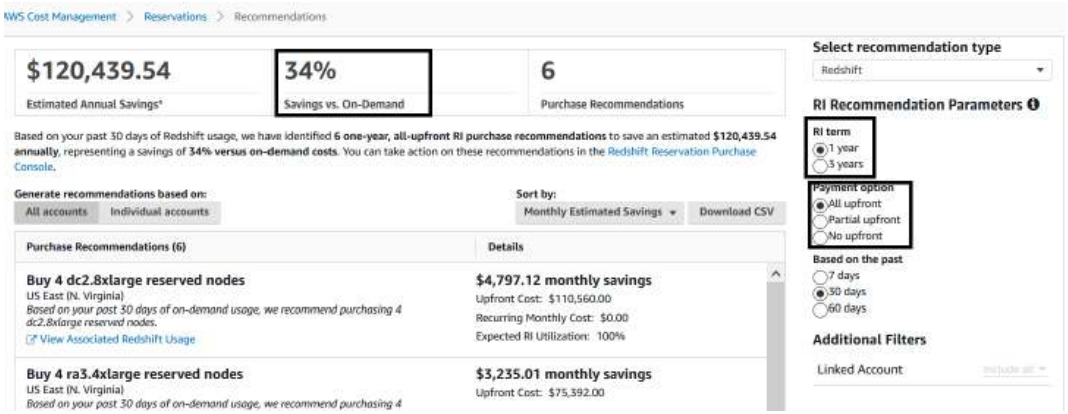


Figure 9.5 – AWS cost optimization recommendations

3. Now, let's see the benefits of cost savings with three Reserved Instances. If we have three years all upfront, we get a significant saving of **65%** compared to on-demand pricing:

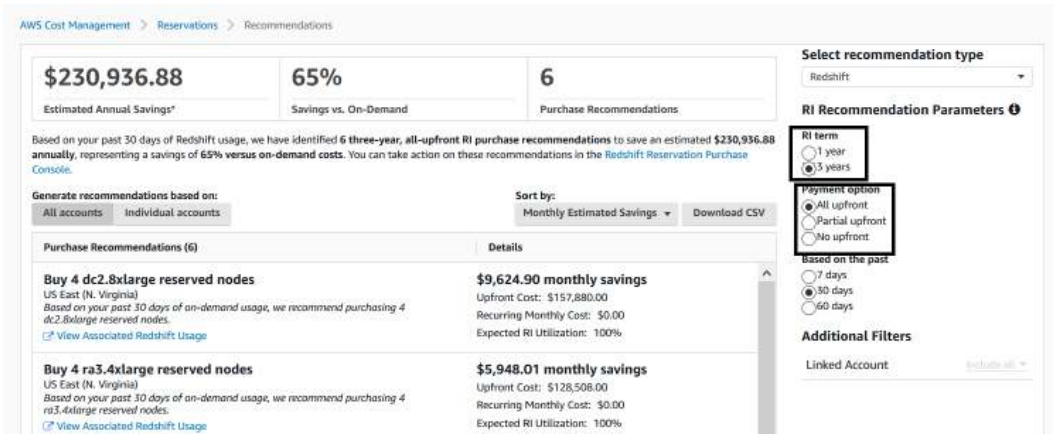


Figure 9.6 – AWS cost optimization benefits

4. To purchase the reserved nodes, navigate to the Amazon Redshift console. Choose **Clusters** and then select **Reserved nodes**:

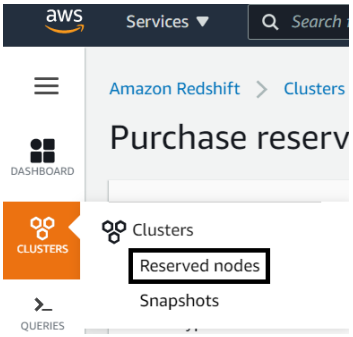


Figure 9.7 – Purchasing Reserved Instances

5. Choose the instance types and the Reserved Instance term, **1 year** or **3 years**:

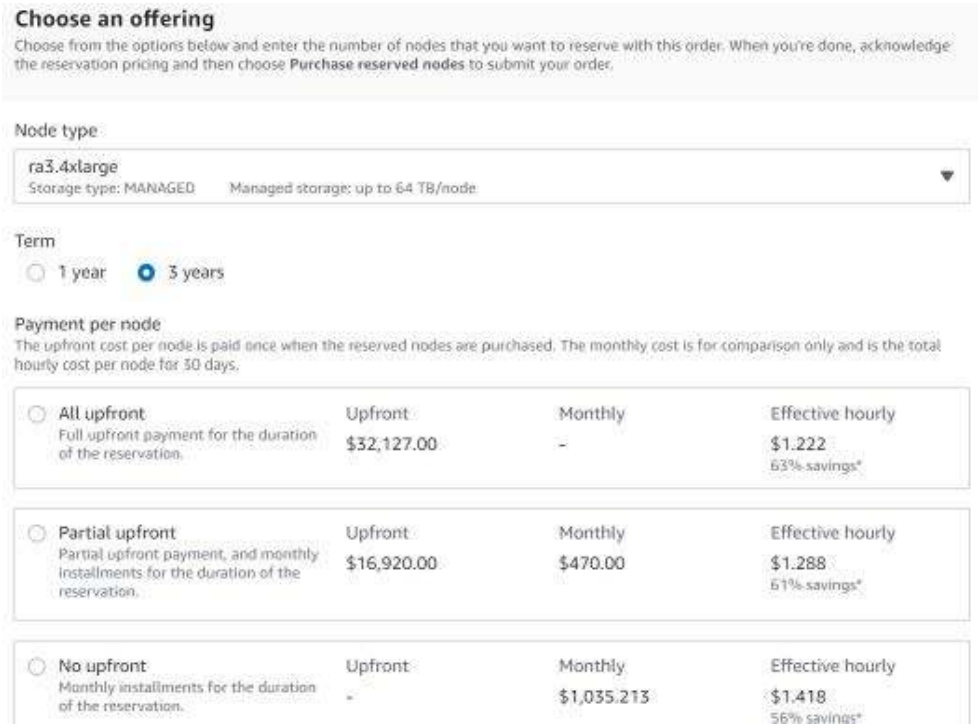


Figure 9.8 – Reserved Instance plans and savings

6. Enter the number of nodes you need, check the acknowledgment checkbox, and select Purchase reserved nodes. Once you have purchased the reserved nodes, your billing will reflect the savings:

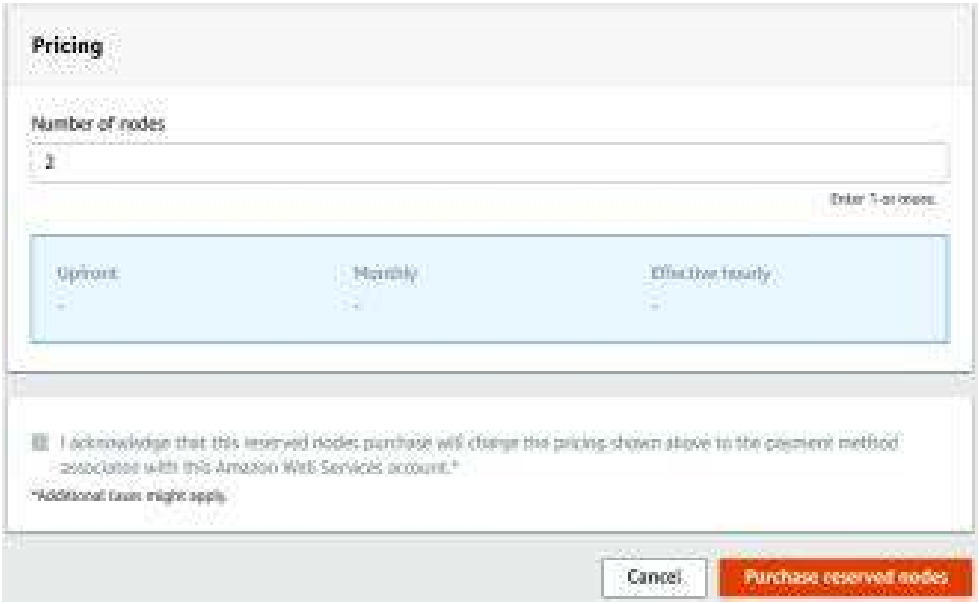


Figure 9.9 – Purchasing the Reserved Instance

See also...

Find out more about Reserved Instance pricing for Amazon Redshift here:

- `https://docs.aws.amazon.com/redshift/latest/mgmt/purchase-reserved-node-instance.html`
- `https://aws.amazon.com/redshift/pricing/`

Scheduling pause and resume for Amazon Redshift provisioned cluster

The customers generally have a set of development, test, and production workloads. While production workloads need to be up and running 24/7, the same can't be said for development and test workloads. To make cost-conscious decisions, customers can use the pause and resume feature within Amazon Redshift to only resume for the development and test clusters when they are in use and pause when not in use. The customers can perform this action on-demand or even schedule on a specific interval.

In the recipe, we will learn how to pause and resume the Amazon Redshift cluster on a schedule.

Getting ready

To complete this recipe, you will need:

- An IAM user with access to Amazon Redshift
- An IAM role called `Chapter8RedshiftSchedulerRole` with the ability to schedule pause and resume for Redshift clusters
- An Amazon Redshift provisioned cluster deployed in the eu-west-1 AWS region

How to do it...

1. Open the Amazon Redshift console: `https://console.aws.amazon.com/redshiftv2/home`.
2. Select the cluster that you would like to pause, click on **Actions**, and select **Pause**, as shown in the following screenshot:

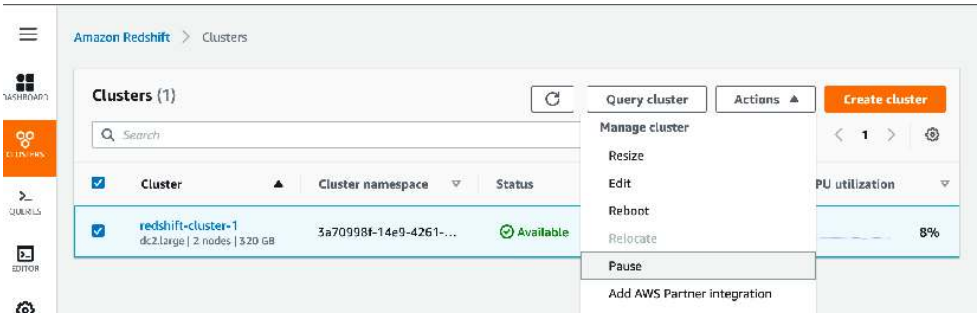


Figure 9.10 – Select your cluster from the Amazon Redshift Console

3. In the **Pause** cluster window, you have multiple options:
    - **Resume now**: This option allows you to perform resume operations on demand.
    - **Resume later**: This option allows you to perform a resume operation at a particular date and time.
    - **Resume and pause on schedule**: This option allows you to perform pause and resume operations on a given schedule.

4. We will review resuming the cluster by scheduling a pause and resume operation here. We will select **Pause and resume on schedule**. Provide **Schedule name** and **Description**:

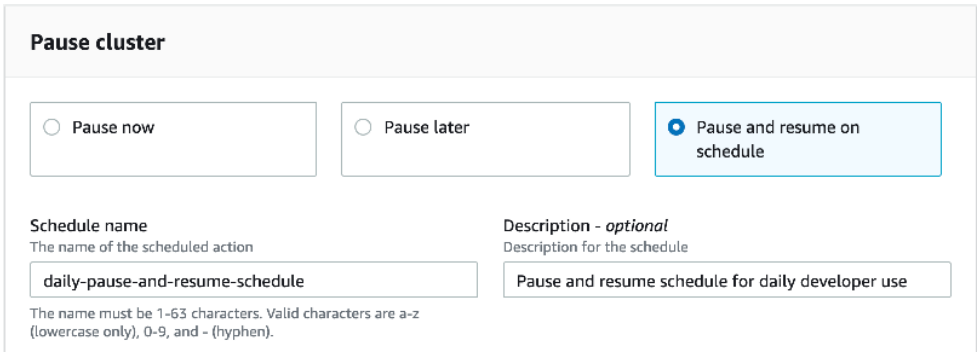


Figure 9.11 – Create a schedule for pause and resume

5. For the schedule, select the **Starts on** and **Ends on** dates that should be applied. In the **Editor**, you can choose Week, Day, or Month for the pause and resume schedule:

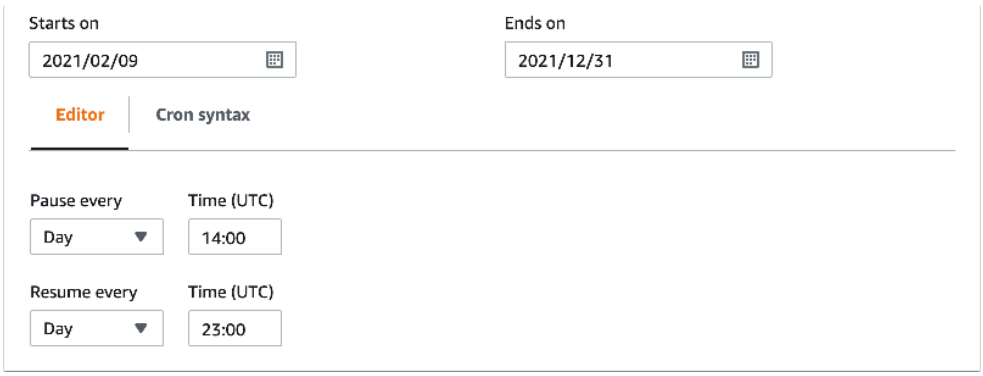


Figure 9.12 – Pick the times to pause and resume

6. In the **Scheduler permissions** section, you will need to select the pre-created **IAM role** from the dropdown that can perform the modify operation on the Redshift cluster and can call the Redshift scheduler.

Finally, click on the **Schedule recurring pause and resume** button to schedule the operation:

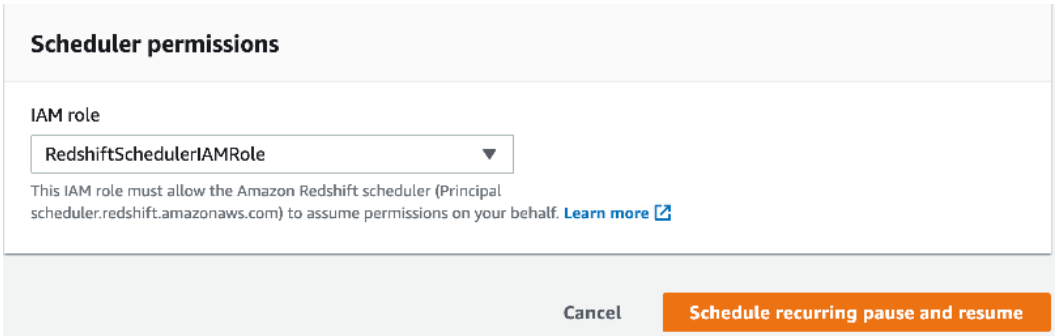


Figure 9.13 – Associated permissions to perform pause and resume

How it works...

When you pause a cluster, a snapshot is created, queries are terminated, and the cluster enters the paused state. From a pricing perspective, on-demand billing is suspended for that cluster, and only storage incurs charges. When you resume the cluster, it creates a cluster from the snapshot that was taken during the pause operation.

Note

Pause and resume operations can be can also be performed using the Redshift API or SDK (`https://docs.aws.amazon.com/redshift/latest/APIReference/API_Operations.html`). This allows you to automate your operational tasks easily. For example, you can pause your development/ test cluster when it's not in use during non-business hours.

Scheduling elastic resizing for an Amazon Redshift provisioned cluster

The analytics workload requirements for enterprises change over time, and resizing makes it easy to scale the workload up or down and even change to newer instance classes with few clicks. Elastic resize is a mechanism to add nodes, remove nodes, and change node types for an existing Amazon Redshift cluster. In this recipe, we will cover how to schedule a resize operation based on business requirements. For example, you might want to upsize your cluster before the start of your scheduled ETL process to satisfy the SLA needs.

In this recipe, you will learn how conduct elastic resizing on a Redshift cluster using a schedule:

Getting ready

To complete this recipe, you will need:

- An IAM user with access to Amazon Redshift
- An IAM role called `Chapter9RedshiftSchedulerRole` with the ability to schedule elastic resize for a Redshift cluster
- An Amazon Redshift provisioned cluster deployed in the eu-west-1 AWS region

How to do it...

1. Open the Amazon Redshift console: `https://console.aws.amazon.com/redshiftv2/home`.
2. Select the cluster, click **Actions**, and select **Resize**:

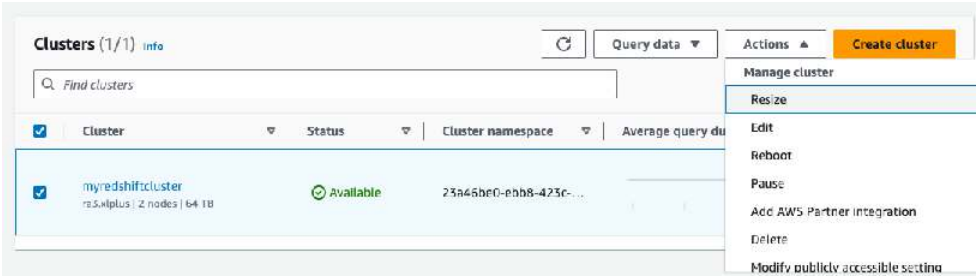


Figure 9.14 – Cluster management for resizing

3. Under **Resize Cluster**, keep the default selection for Elastic resize (recommended).
4. Under **Schedule resize**, the options are **Resize the cluster now**, **Schedule resize at a later time**, and **Schedule recurring resize events**. For a resize based on a recurring event, we will select **Schedule recurring resize events** to repeat upsize/downsize operations based on a schedule:

Figure 9.15 – Creating a recurring resize event

5. Under **Scheduling options**, enter the name for the schedule under **Schedule name** and enter the dates when this schedule needs to start and stop in the **Starts on** and **Ends on** fields. You can now select when and how the cluster configuration needs to change by selecting **Node type**, **Number of nodes**, and **Increase size every**.

   For instance, we want to scale the workload up to 4 nodes on every last Monday of the month to manage the end-of-month reporting workload and scale it back down to 2 nodes at the start of the month.

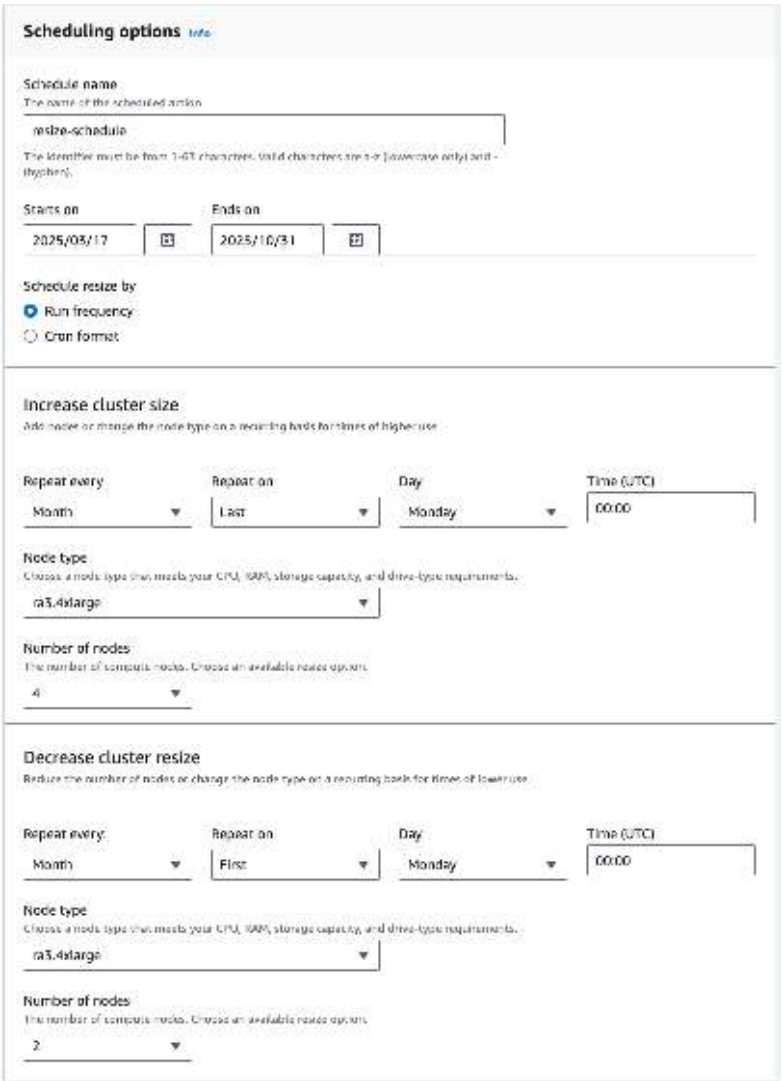


Figure 9.16 – Creating an elastic resize (upsize and downsize) schedule

6. In the **Scheduler permissions** section, select the pre-created IAM role from the dropdown that can perform the resize operation on the Redshift cluster and can call the Redshift scheduler. Finally, click on the **Schedule resize** button to schedule the elastic resize operation.

   For instance, we are selecting the IAM role from the dropdown called **Chapter9RedshiftSchedulerRole**, which was pre-created with correct access:

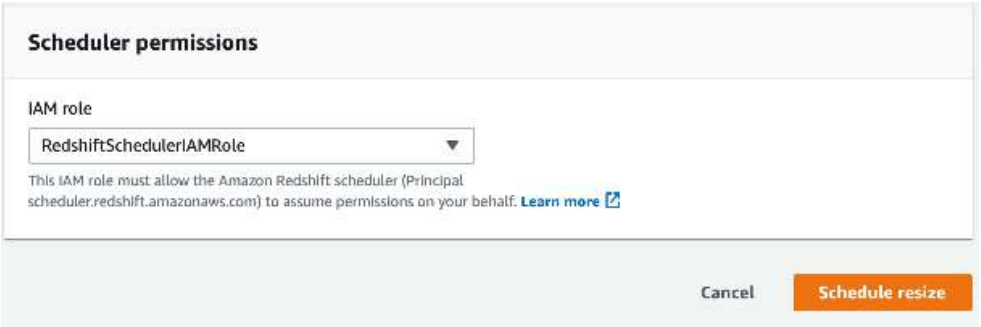


Figure 9.17 – Selecting the IAM role for scheduling the elastic resize

7. Validate that the resize operation has been created, click on your cluster from the main **CLUSTER** option, and select the **Schedule** tab. In the **Resize schedule** section, you will have the resize operations listed.

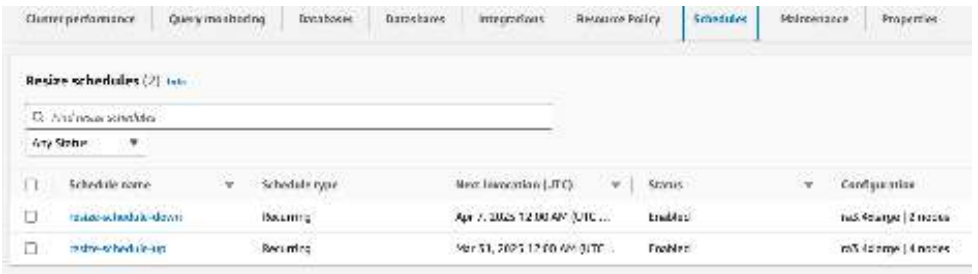


Figure 9.18 – Validate the elastic resize schedule

How it works...

An elastic resize takes around 10-15 mins to complete, and during this time the cluster is in read-only mode. When changing just the node count but keeping the node type the same, the data is redistributed at the backend, queries are temporarily paused, and connections are held open. When changing the node type, the operation creates a new cluster from a snapshot, and open connections will be terminated.

Using cost controls to set actions for Redshift Spectrum

Amazon Redshift allows you to extend your data warehouse to the data lake by performing SQL queries directly on data on Amazon S3. You will be charged based on the number of bytes scanned by Redshift Spectrum, rounded up to the next MB, with a 10 MB minimum per query (`https://aws.amazon.com/redshift/pricing/#Redshift_Spectrum_pricing`). There are no charges for **Data Definition Language (DDL)** statements like `CREATE/ALTER/DROP TABLE` statements for managing partitions and failed queries.

In the recipe, we will set up cost controls on the Amazon Redshift spectrum usage to prevent any accidental scan by a monstrous query.

Getting ready

To complete this recipe, you will need:

- An IAM user with access to Amazon Redshift
- An Amazon Redshift cluster deployed in the eu-west-1 AWS region

How to do it...

1. Navigate to the AWS Amazon Redshift console and navigate to **Amazon Redshift | Clusters** and click on your Amazon Redshift cluster. Click on the **Properties** tab and scroll down to the **Database configurations**, as shown in the following screenshot:

Figure 9.19 – Selecting the parameter group associated with the Amazon Redshift cluster

2. Click on the Parameter group associated with the cluster. Click on the edit workload queues and click on **Add custom rule**, as shown in the following screenshot:

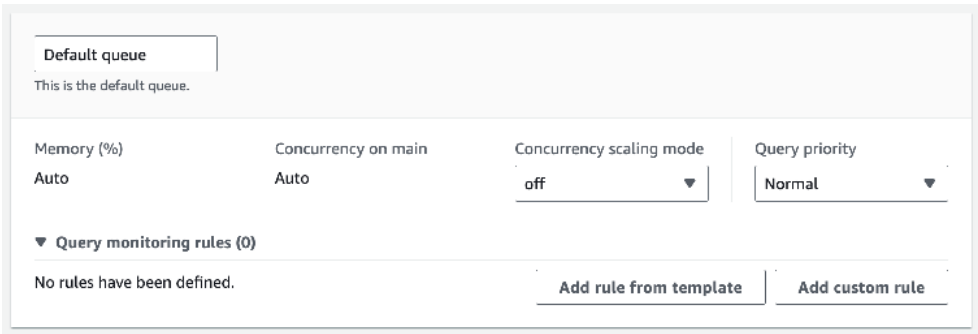


Figure 9.20 – Modifying workload queues

Note

You cannot edit the default parameter groups and will have to create a custom parameter group to edit the queues and monitoring rules associated with your cluster.

3. Choose a rule name (any user-friendly name) and click on the Predicates dropdown to select **Spectrum scan (MB)**. Choose values **> 100,000,000** and set Actions to **abort**, as shown in the following screenshot, and click on **Save**:

Figure 9.21 – Adding a custom query monitoring rule for Spectrum

Amazon Redshift will now abort any query that scans data over 100 TB, and you will not be charged for any queries that were aborted. This prevents any user from accidentally scanning a large amount of data for your data warehouse.

4. You will now create cost controls at the Amazon Redshift cluster level. Navigate to **Amazon Redshift | Clusters | Actions | Manage usage limit**.
5. Click on **Add limit** and set a limit corresponding to **Redshift Spectrum usage limit**, as shown in the following screenshot:

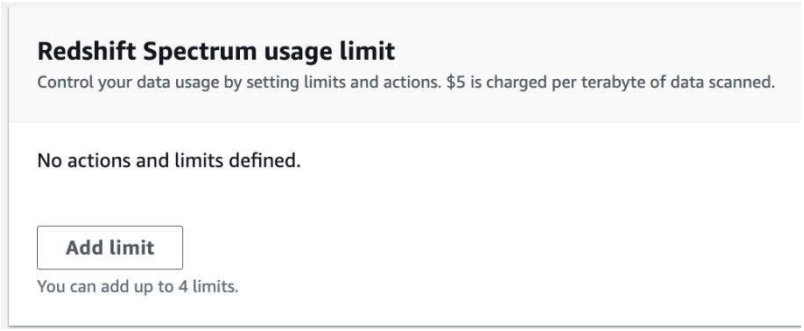


Figure 9.22 – Configuring limits and action for Spectrum

6. For Time period, select **Monthly** and set **Usage Limit (TB)** to **1000** and click on **Save changes**:

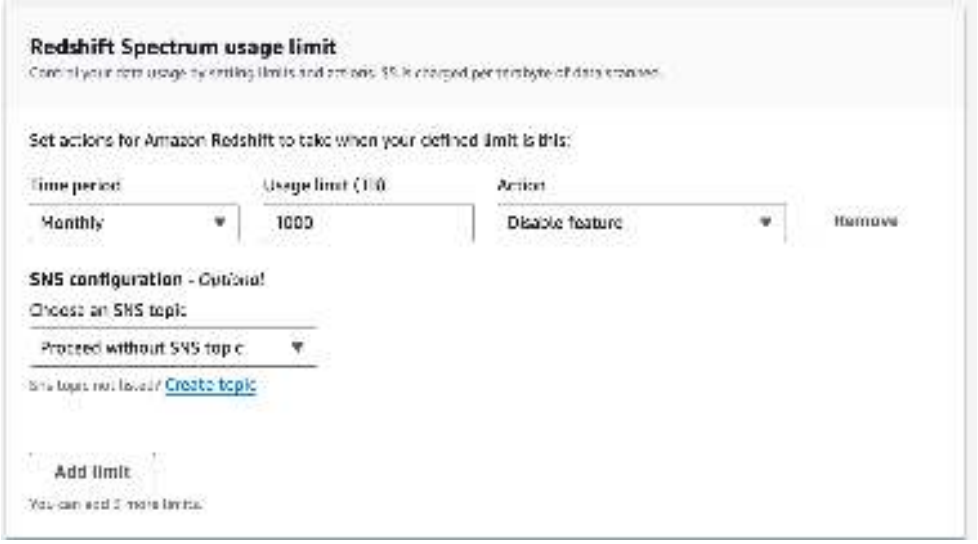


Figure 9.23 – Setting up monthly limits for Spectrum usage

The Amazon Redshift Spectrum feature is disabled when the monthly limit of 1,000 TB of data scanned is exceeded.

See also...

For a step-by-step guide to setting up the workload management, refer to the *Configuring Workload Management (WLM) for provisioned cluster* recipe in *Chapter 8*.

Using cost controls to set actions for concurrency scaling for an Amazon provisioned cluster

Amazon Redshift offers a feature called concurrency scaling that automatically adds temporary clusters when your system needs to handle multiple user queries at once. Every day that your main Redshift cluster is running, you earn one hour of free credits to use these temporary clusters, though these credits expire at the end of each month. If you need more capacity beyond your free credits, you'll only be charged when the temporary clusters are actively processing queries. The billing is calculated per second at the on-demand rate, with a minimum charge of one minute each time a temporary cluster is activated. This way, you only pay for the extra processing power when you actually need it, making it a cost-effective solution for handling periodic spikes in query volume. In the recipe, we will set up controls for concurrency scaling usage on your Amazon Redshift cluster.

Getting ready

To complete this recipe, you will need:

- An IAM user with access to Amazon Redshift
- An Amazon Redshift cluster deployed in the eu-west-1 AWS region

How to do it...

1. Navigate to **Amazon Redshift | Clusters | Actions | Manage usage limit**.
2. Click on **Add limit** and set a limit corresponding to **Concurrent scaling usage limit**, as shown in the following screenshot:

Figure 9.24 – Configuring limits and actions for Spectrum

3. For Time period, select **Monthly**, for and **Usage Limit (hh:mm)**, set **30** and click on **Save changes**:

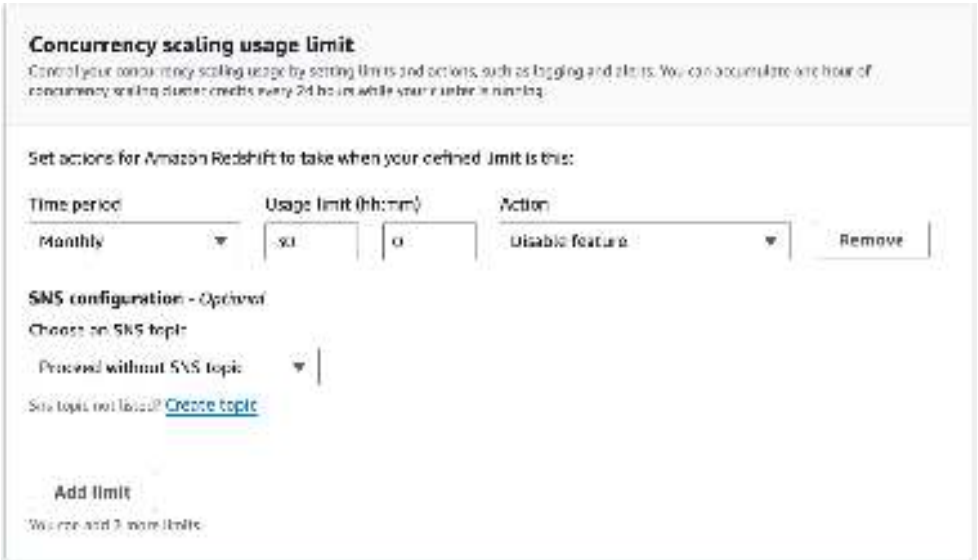


Figure 9.25 – Setting up monthly limits for concurrency scaling usage

Now, the Amazon Redshift concurrency scaling feature is disabled when the monthly limit exceeds 30 hours.

In addition to disabling concurrency scaling when limits are exceeded in your cluster, you can also limit the number of concurrent clusters that are spun up using the `max_concurrency_scaling_clusters` parameter we saw in *Chapter 8*.

See also...

Concurrency scaling pricing: `https://aws.amazon.com/redshift/pricing/#Concurrency_Scaling_pricing`

Using cost controls for Redshift Serverless

Redshift Serverless automatically adjusts capacity based on the workload demand when there is a query running against it and it shuts down when it's not in use. The workload is charged on a per-second basis (with a 60 seconds minimum charge). There are three crucial settings, **Base Capacity**, **Max RPU-hours**, and **Max RPU**, to fine-tune your Redshift serverless cost efficiency while maintaining performance.

In this recipe, you will learn how to have cost controls when using the Amazon Redshift Serverless to prevent any surprises by adjusting Base Capacity, Max RPU-hours, and MaxRPU.

Getting ready

To complete this recipe, you will need:

- An IAM user with access to Amazon Redshift
- Amazon Redshift Serverless cluster deployed in the eu-west-1 AWS region

How to do it...

1. Navigate to **Amazon Redshift | Serverless dashboard** and click on your workgroup.
2. Go to the **Performance** tab and select **Edit** for **Performance and cost controls**.

3. Under **Performance and cost controls**, you can manually select the **Base capacity** from the dropdown ranging from 8 to 512. Click on **Save** changes to confirm the selection:

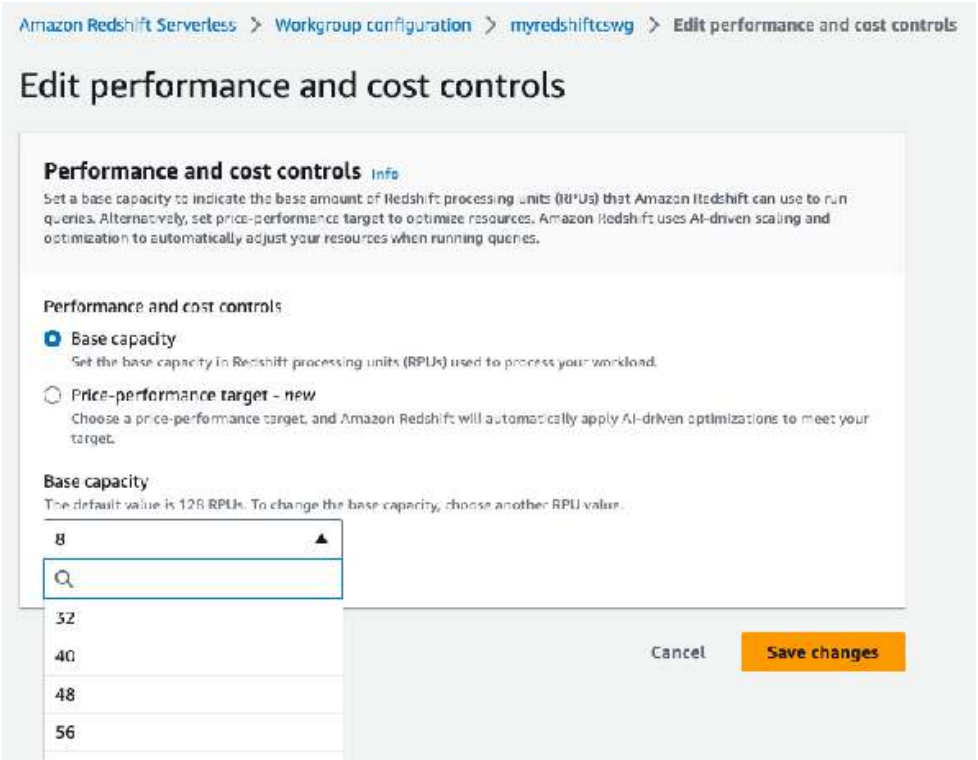


Figure 9.26 – Setting up base capacity

Now, the base capacity for the Amazon Redshift Serverless cluster is configured at 8 RPUs, which means when the workgroup starts it defaults at 8 RPUs before scaling up for additional demand as needed.

4. Optionally, you can select **Price-performance target**, where Redshift automatically applies AI-driven optimization to meet your target, ranging from 1 for cost optimization to 100 for performance optimization. This feature works best when the system has learned the specific workload pattern and when the base RPU is between 32 and 512. Click on **Save** changes to confirm the selection:

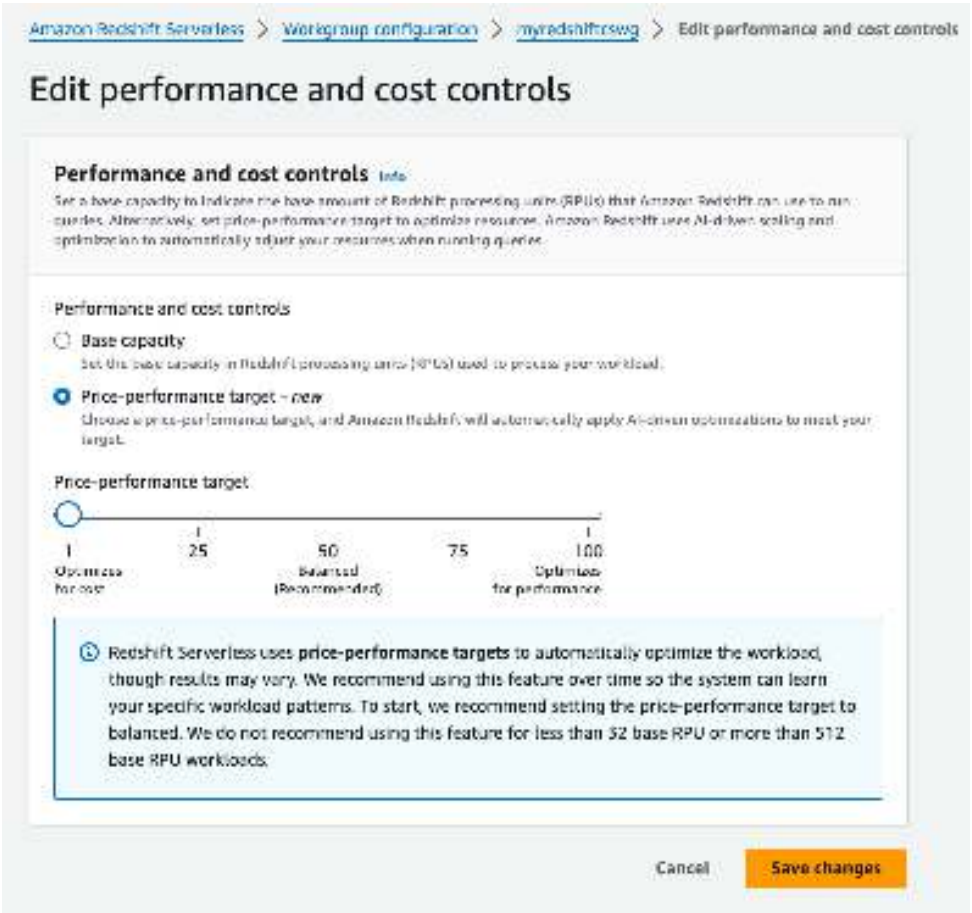


Figure 9.27 – Setting up the price-performance target

The price-performance target value of 1 indicates that Amazon Redshift's AI-driven scaling is configured to prioritize cost optimization. At this setting, the system will make scaling decisions that emphasize cost efficiency over maximum performance, helping to balance workload requirements with cost management

5. To set the MaxRPU (which will cap the maximum RPU Serverless will scale up to), browse to **Limits** tab and select **Edit** for **Max Capacity**.
6. In the **Edit max capacity** section, set the **Max capacity** ranging from 8 to 5632 in increments of 8. Click on **Save** changes to confirm the selection:

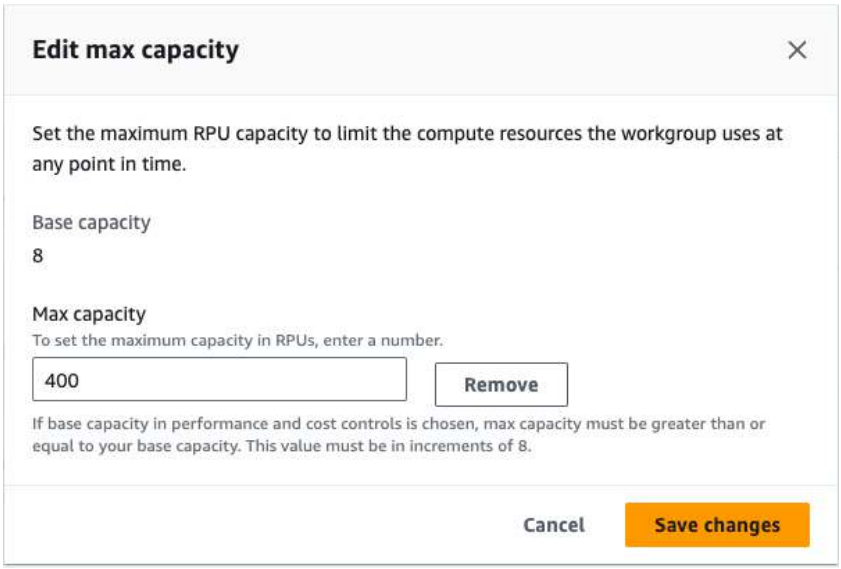


Figure 9.28 – Setting up max capacity

7. Now, the max capacity for the Amazon Redshift Serverless cluster is configured at 400, which means when the workgroup will not scale beyond 400 RPUs. To set the Max RPU hours (the budget cap for potential spending), browse to the **Limits** tab and select **Manage usage limits**.
8. In the **Manage usage limits** section, under **Maximum Redshift processing units (RPU)**, select **Add limit**.
9. In the **Compute usage limits** section, you have the option to set the frequency to daily, weekly, or hourly. You can also select the **Usage limit (RPU hours)**.

10. Under **Action**, you can select from different options, from tracking by logging to system tables to turning off user queries if you have hard budget limits. You have the option to configure up to 4 limits.

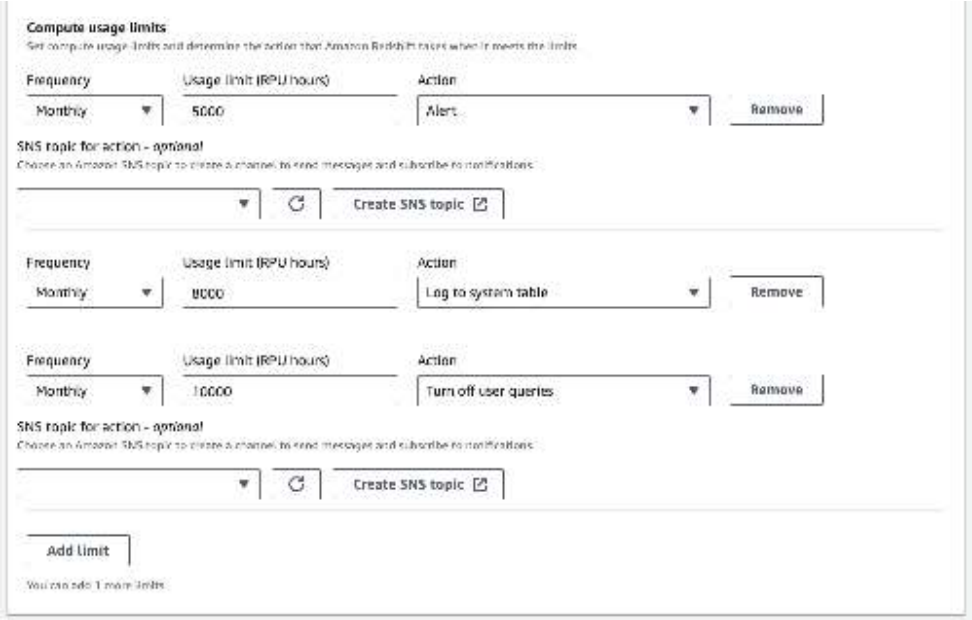


Figure 9.29 – Setting up usage limits

Now, the maximum number of hours the RPUs will operate is 1000 beyond which no user queries will run on the workgroup.

How it works...

The following settings allow us to build cost controls for the Redshift Serverless environment:

- **Base Capacity**: This is your cost foundation. By setting the Base RPUs, you're essentially choosing your minimum ongoing cost. While a higher base can improve performance for data-intensive tasks, it also means a higher fixed cost. The key here is to find the sweet spot where you're not overpaying for unused capacity during quiet periods but still have enough power for your regular workload.

- **Max RPU-hours**: This is your cost ceiling. By specifying Max RPU-hours over daily, weekly, or monthly periods, you're putting a hard limit on your potential spending. It's like setting a budget cap – Amazon Redshift will take automatic actions to ensure you don't exceed this limit. This feature is crucial for maintaining predictable costs, especially if your workload can be variable or if you're working within a strict budget.
- **MaxRPU (Max Capacity)**: This acts as your cost safeguard against unexpected spikes. While automatic scaling can help handle sudden increases in demand, unrestricted scaling could lead to unexpectedly high costs. The MaxRPU setting prevents this by capping the maximum resources your warehouse can scale up to, even during peak periods.

10

Lakehouse Architecture

Lakehouse is an architectural pattern that makes data easily accessible across a customer's analytics solutions, thereby preventing data silos. Amazon Redshift is the backbone of the lakehouse architecture. It allows enterprise customers to query data across the data lake, operational database, and multiple data warehouses to build an analytics solution without having to move data in and out of these different systems. The key benefits of a lakehouse include unified data management (no need to maintain separate copies of data), consistent security and governance across all data, and the ability to use multiple query engines and tools to access the same data. AWS's implementation specifically allows customers to use different storage options (S3 buckets, S3 tables, or **Redshift managed storage (RMS)**) while providing access through standard Iceberg APIs, making the data accessible to both AWS services and third-party tools without requiring data migration or copies.

In this chapter, you will learn how you can leverage the lakehouse architecture to extend a data warehouse to services outside Amazon Redshift to build your solution, while taking advantage of the built-in integration.

The following recipes are discussed in this chapter:

- Building a data lake catalog using AWS Lake Formation
- Carrying out a data lake export from Amazon Redshift
- Extending a data warehouse using Amazon Redshift Spectrum
- Querying an operational source using a federated query
- Amazon SageMaker Lakehouse

Technical requirements

Here are the technical requirements to complete the recipes in this chapter:

- Access to the AWS Management Console.
- AWS administrator permission to create an IAM user by following *Recipe 1* in *Appendix*. This IAM user will be used for some of the recipes in this chapter.
- AWS administrator permission to create an IAM role by following *Recipe 3* in *Appendix*. This IAM role will be used for some of the recipes in this chapter.
- AWS administrator permission to deploy an AWS CloudFormation template (`https://github.com/PacktPublishing/Amazon-Redshift-Cookbook-2E/blob/main/Chapter10/chapter_10_CFN.yaml`) to create two IAM policies:
    - An IAM policy attached to the IAM user that will give them access to Amazon Redshift, Amazon EC2, Amazon S3, Amazon SNS, Amazon CloudWatch, Amazon CloudWatch Logs, AWS KMS, AWS IAM, AWS CloudFormation, AWS CloudTrail, Amazon RDS, AWS Lake Formation, AWS Secrets Manager, and AWS Glue
    - An IAM policy attached to the IAM role that will allow an Amazon Redshift data warehouse to access Amazon S3, Amazon RDS, and AWS Glue
- Attach an IAM role to the Amazon Redshift data warehouse by following *Recipe 4* in *Appendix*. Make note of the IAM role name; we will refer to it in the recipes with `[Your-Redshift_Role]`.
- AWS administrator permission to run the CLI commands using AWS Cloud Shell (`https://github.com/PacktPublishing/Amazon-Redshift-Cookbook-2E/blob/main/Chapter10/Chapter10_DataTransferRoleAndLakeFormation`). This will create `DataTransferRole` and configure AWS LakeFormation settings.
- An Amazon Redshift data warehouse deployed in the AWS Region eu-west-1.
- Amazon Redshift data warehouse admin user credentials.
- Access to any SQL interface such as a SQL client or Amazon Redshift Query Editor.
- An AWS account number; we will refer to it in the recipes with `[Your-AWS_Account_Id]`.
- An Amazon S3 bucket created in eu-west-1; we will refer to it with `[Your-Amazon_S3_Bucket]`.
- The code files can be found in the Git repo: `https://github.com/PacktPublishing/Amazon-Redshift-Cookbook-2E/tree/main/Chapter10`.

Building a data lake catalog using AWS Lake Formation

The data lake design pattern has been widely adopted by the industry. Data lakes help to break data silos, by allowing you to store all of your data in a single, unified place. You can collect the data from different sources. Data can arrive at different frequencies, for example, clickstream data. The data format can be structured, unstructured, or semi-structured. Analyzing a unified view of data allows you to derive more value and insight from the data to drive business value.

Your data lake should be secure and meet your compliance requirements. It should include a comprehensive, searchable index of all the data stored in the lake. This catalog makes it easy for users to locate and access the specific data they need. One of the advantages of data lakes is that you can run a variety of analytic tools against it. It also allows you to carry out new types of analysis on your data. For example, you may want to move from answering questions about what happened in the past to focusing on real-time insights and using statistical models and forecasting techniques to understand and answer what could happen in the future. To do this, you need to incorporate **machine learning (ML)**, big data processing, and real-time analytics. The pattern that allows you to integrate your analytics with a data lake is the lakehouse architecture. Amazon S3 object stores are used for centralized data lakes due to their scalability, high availability, and durability.

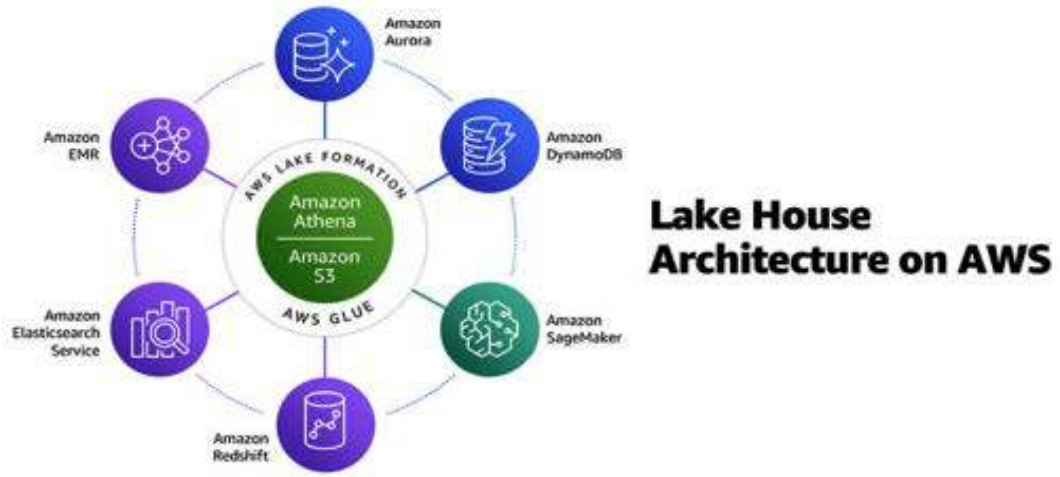


Figure 10.1 – Lakehouse architecture

Typical challenges and steps involved in building a data lake include the following:

- Identifying sources and defining the frequency with which the data lake needs to be hydrated
- Cleaning and cataloging the data
- Centralizing the configuration and application of security policies
- Integrating the data lake with analytical services that adhere to the centralized security policies

The following is a representation of the lakehouse workflow moving data from raw format to ready for analytics:

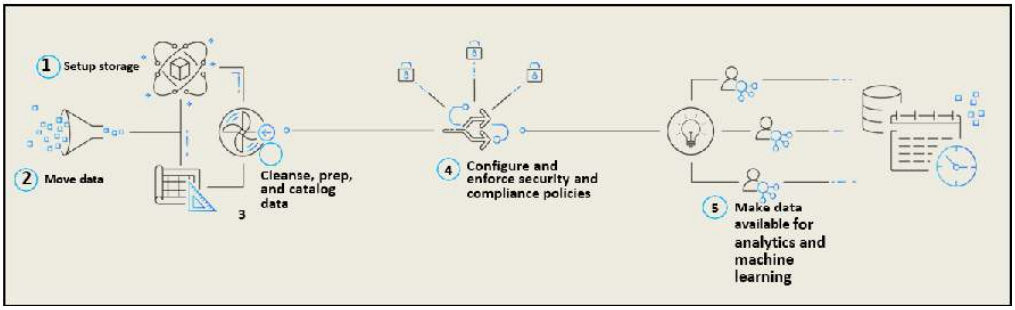


Figure 10.2 – Data workflow using the lakehouse architecture

The AWS Lake Formation service allows you to simplify the build, centralize the management, and configure security policies. AWS Lake Formation leverages AWS Glue for cataloging, data ingestion, and data transformation.

In this recipe, you will learn how to use Lake Formation to hydrate a data lake from a relational database, catalog the data, and apply security policies.

Getting ready

To complete this recipe, you will need the following setup:

- An IAM user with access to Amazon RDS, Amazon S3, and AWS Lake Formation.
- An Amazon RDS MySQL database; create an RDS MySQL cluster: `https://aws.amazon.com/getting-started/hands-on/create-mysql-db/`.

  In this recipe, the version of **MySQL engine** is **5.7.44**.

- A command line to connect to RDS MySQL: `https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/USER_ConnectToInstance.html`.

- This recipe uses the **AWS EC2 Linux** instance with the **MySQL command line**. Open the security group for the **RDS MySQL** database to allow connectivity from your client.

How to do it...

In this recipe, we will learn how to set up a data flow with a MySQL-based transactional database to be cataloged using a Lake Formation catalog and query it easily using an Amazon Redshift data warehouse (serverless or provisioned cluster):

1. Let's connect to a `RDS MySQL` database using the following command line. Enter the password and it will connect you to the database:

```
mysql -h [yourMySQLRDSEndPoint] -u admin -p
```

2. We will create an `ods` database in MySQL and create a `parts` table in the `ods` database:

```
create database ods;
CREATE TABLE ods.part
(
  P_PARTKEY       BIGINT NOT NULL,
  P_NAME          VARCHAR(55),
  P_MFGR          VARCHAR(25),
  P_BRAND         VARCHAR(10),
  P_TYPE          VARCHAR(25),
  P_SIZE          INTEGER,
  P_CONTAINER     VARCHAR(10),
  P_RETAILPRICE   DECIMAL(18,4),
  P_COMMENT       VARCHAR(23)
);
```

3. On your client server, download the file `part.tbl` from `https://github.com/PacktPublishing/Amazon-Redshift-Cookbook-2E/blob/main/Chapter10/part.tbl` to your local disk.
4. Now we will load this file into the `ods.part` table on a MySQL database. This will load 100 records into the `parts` table:

```
LOAD DATA LOCAL INFILE 'part.tbl'
    INTO TABLE ods.part
    FIELDS TERMINATED BY '|'
    LINES TERMINATED BY '\n';
```

5. Let's verify the count of records loaded into the `ods.part` table:

```
MySQL [(none)]> select count(*) from ods.part;
+----------+
| count(*) |
+----------+
|    100 |
+----------+
1 row in set (0.00 sec)
```

6. Navigate to **AWS Lake Formation** and select **Get started**.

Figure 10.3 – Navigate to Lake Formation

7. Now, let's set up the data lake location. Select **Register location**.

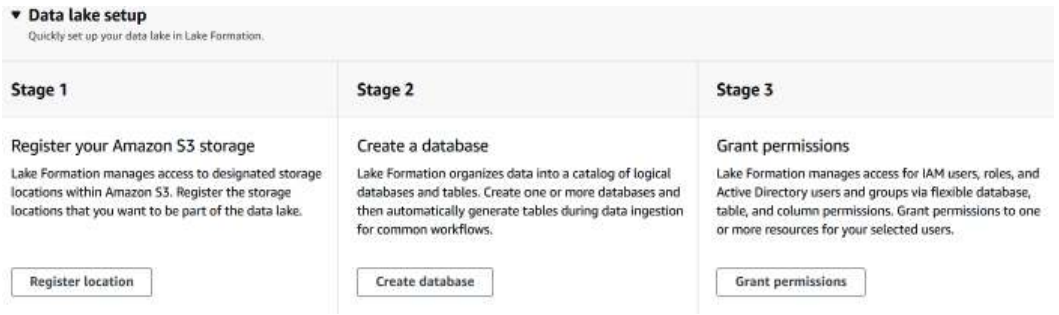


Figure 10.4 – Data lake setup

8. Enter the location of an S3 bucket or folder in your account. If you do not have one, create a bucket in S3 in your account. Keep the default IAM role and click on **Register location**. With this, Lake Formation will manage the data lake location.

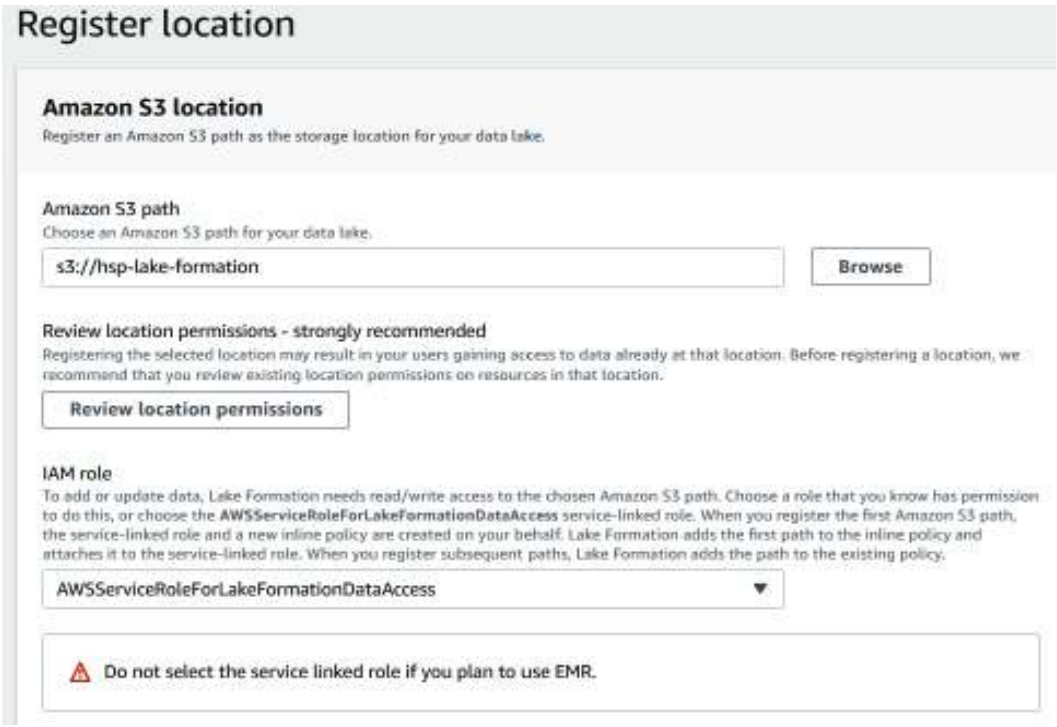


Figure 10.5 – Register the Amazon S3 location in the data lake

9. Next, we will create a database, which will serve as the catalog for the data in the data lake. Click on **Create database**, as shown:

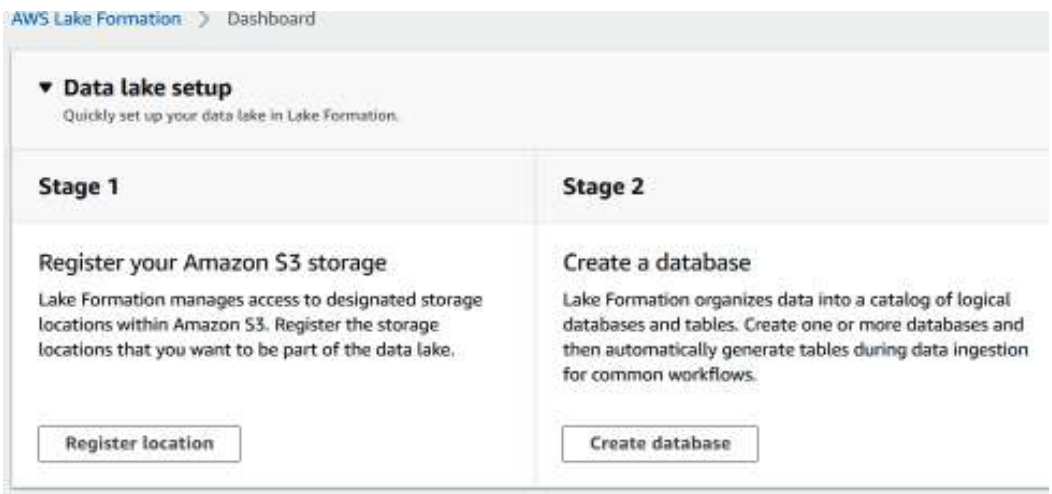


Figure 10.6 – Create a database in Lake Formation

10. Use `cookbook-data-lake` as the database name. For **Catalog**, select the default catalog, which is named with your account number. Select the `s3` path that you registered in AWS Lake Formation. Select the checkbox **Use only IAM access control for new tables in this database**. Click on **Create database**.

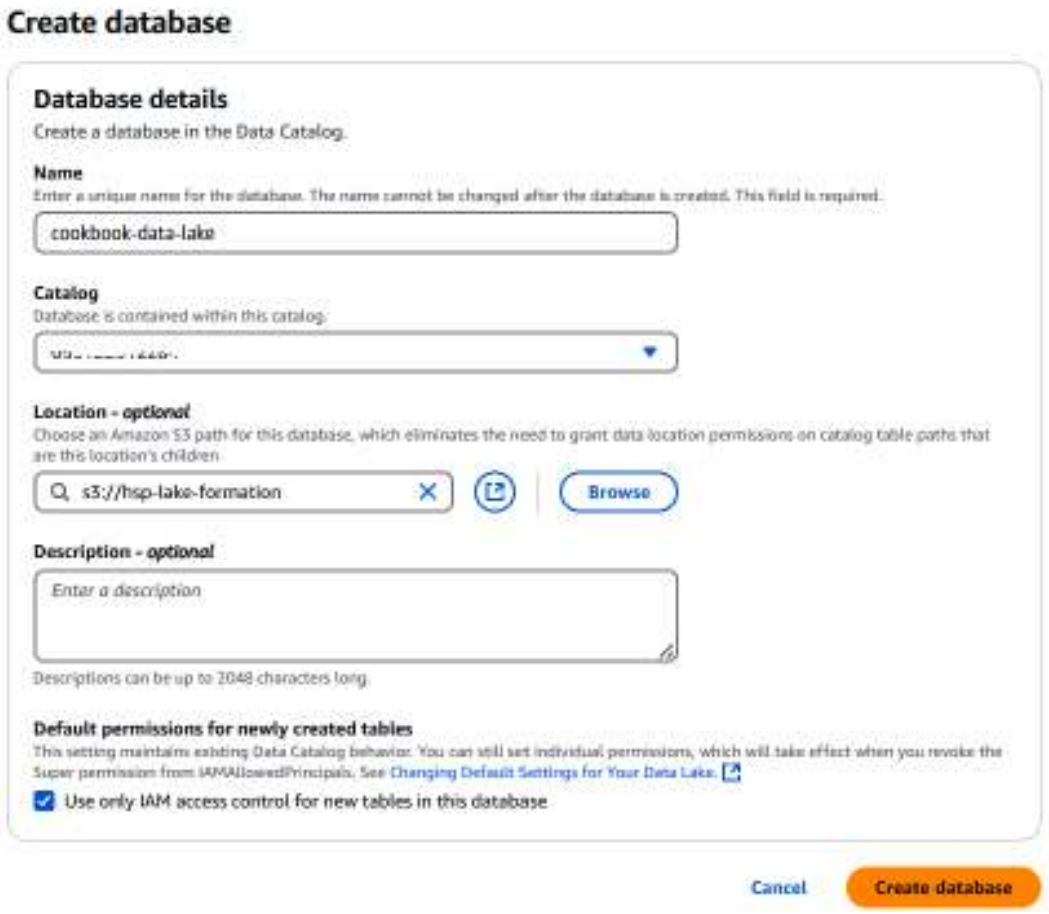


Figure 10.7 – Configure the Lake Formation database

11. Now we will hydrate the data lake using MySQL as the source. From the left menu, select **Blueprint**, then click on **Use blueprint**.
12. Select **Database snapshot** and then click on **Create a connection in AWS Glue**. It will open in a new tab:

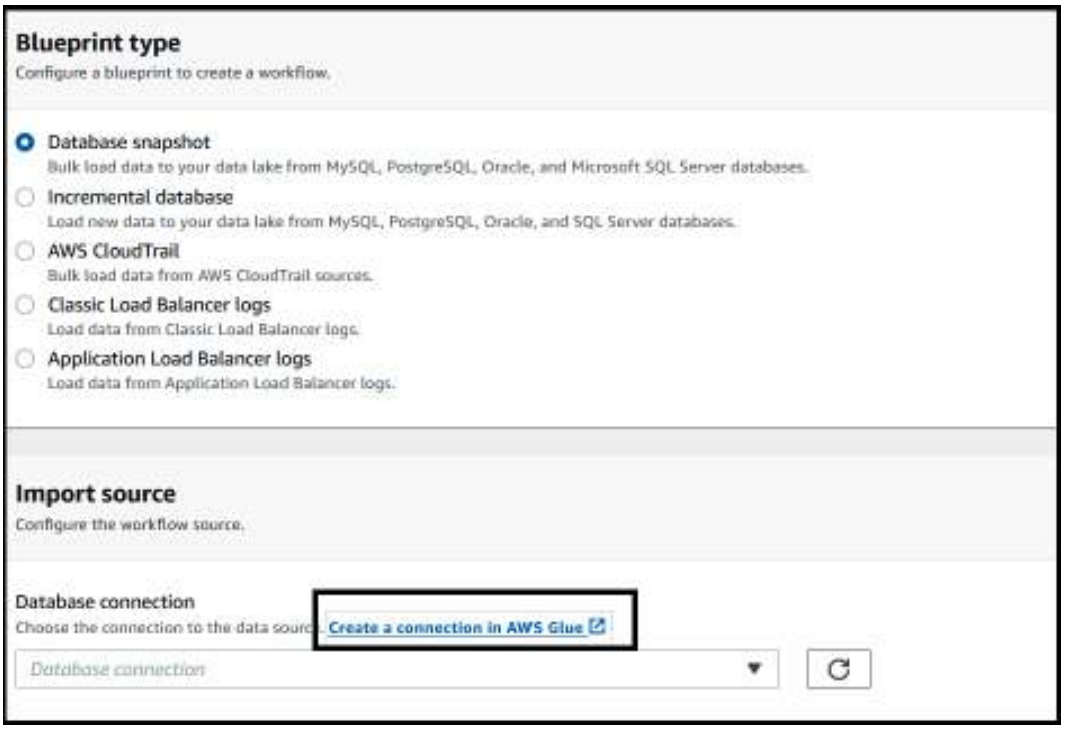


Figure 10.8 – Use a blueprint to create a database snapshot-based workflow

13. Choose **JDBC** and click **Next**. Set the following properties, as shown in the following screenshot:

    - **JDBC URL**: `jdbc:mysql://<your-mysql-instance-endpoint>:3306/ods`.
    - Your username and password.
    - For **Network options**, specify the VPC, subnet, and security group from your RDS MySQL instance. You can find this information on the RDS console for your instance under the **Connectivity and Security** tab.
    - Select **Next**.
    - For the name of the connection, enter `datalake-mysql`.

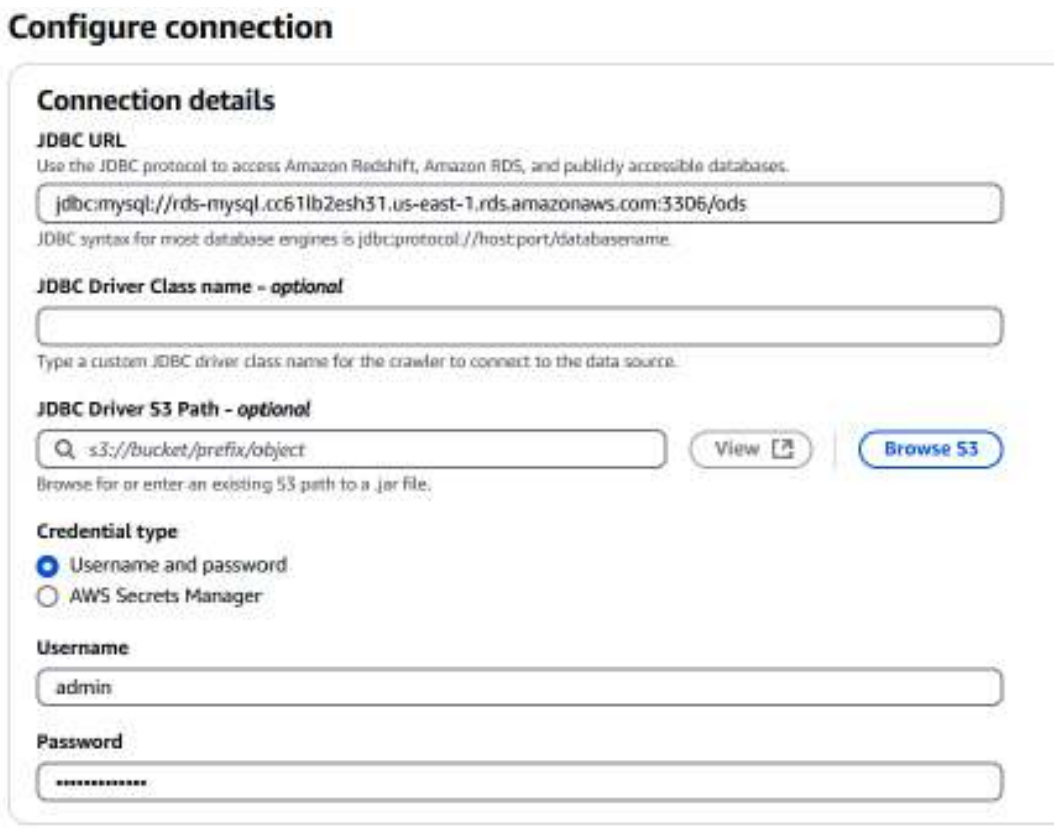


Figure 10.9 – Configure Amazon RDS connection properties

14. Select the connection `datalake-mysql`. Choose **Action** and select **TestConnection**. For the IAM role, use `AWSGlueServiceRole-cookbook`. Select **TestConnection**. It will take a few minutes for the test to run. When it is successful, it will show the **connected successfully to your instance** message. If you run into issues with the connection setup, you can refer to the following URL:

    `https://aws.amazon.com/premiumsupport/knowledge-center/glue-test-connection-failed/`

The successful connection message looks as follows:

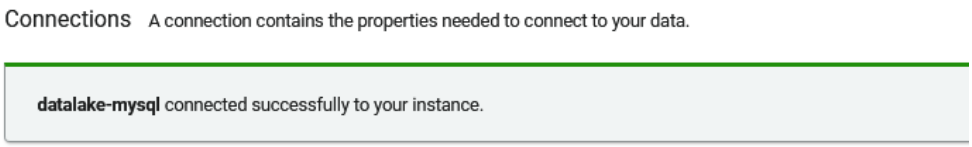


Figure 10.10 – Verify successful connection to the MySQL database

15. In AWS Lake Formation, set the following properties under **Import source**:

    - For **Database connection**, from the dropdown, select **datalake-mysql**
    - For **Source data path**, enter `ods/part`

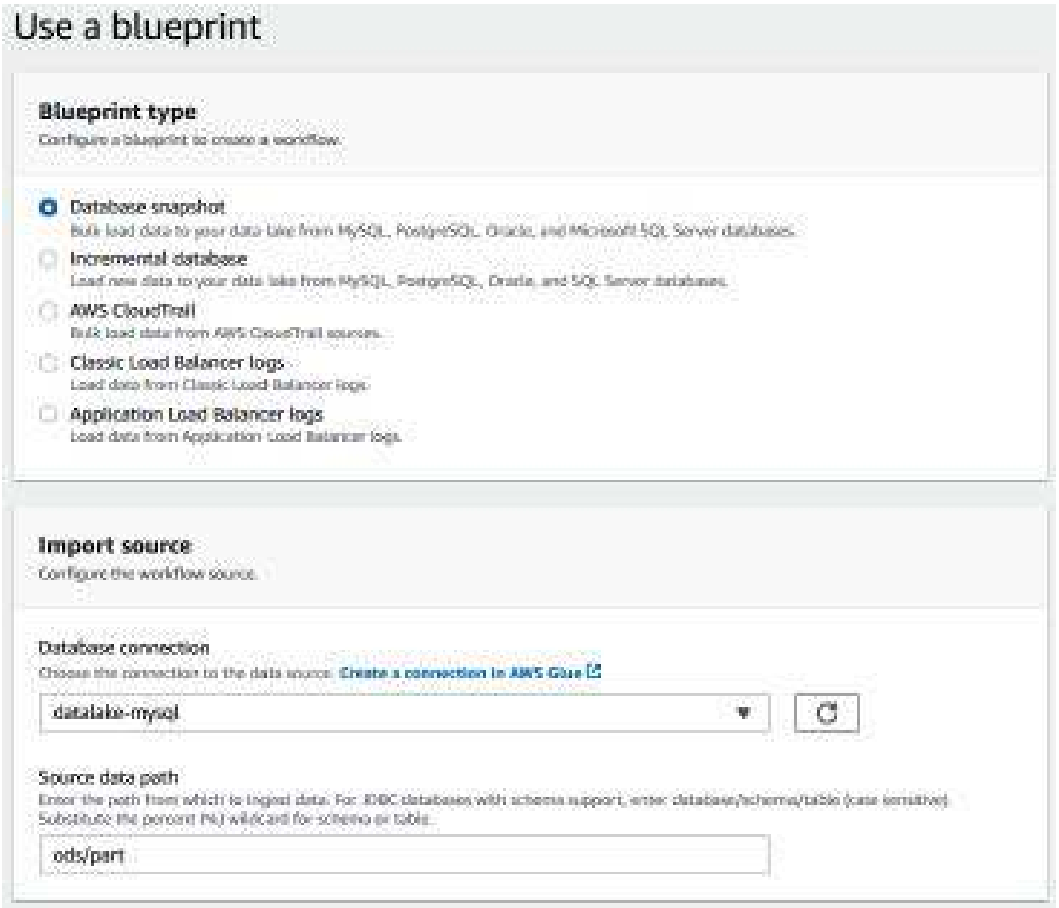


Figure 10.11 – Use a blueprint to create a database snapshot-based workflow

16. Under **Import target**, for **Target database**, select **cookbook-data-lake**. For **Target storage location**, specify your bucket path with `mysql` as the folder. We will unload the data from `mysql` in Parquet format.

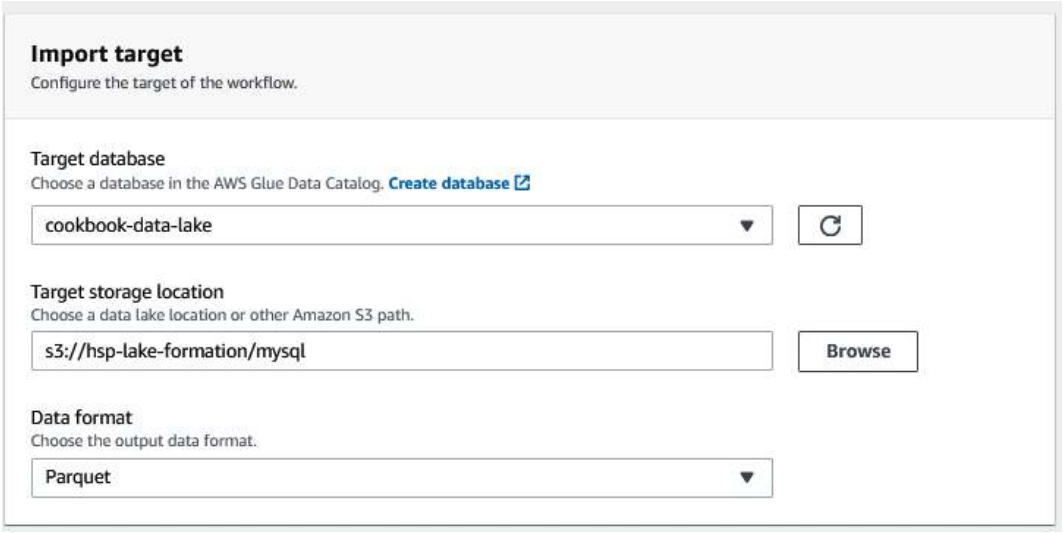


Figure 10.12 – Set up the target for the data workflow

17. Under **Import frequency**, select **Run on demand**.

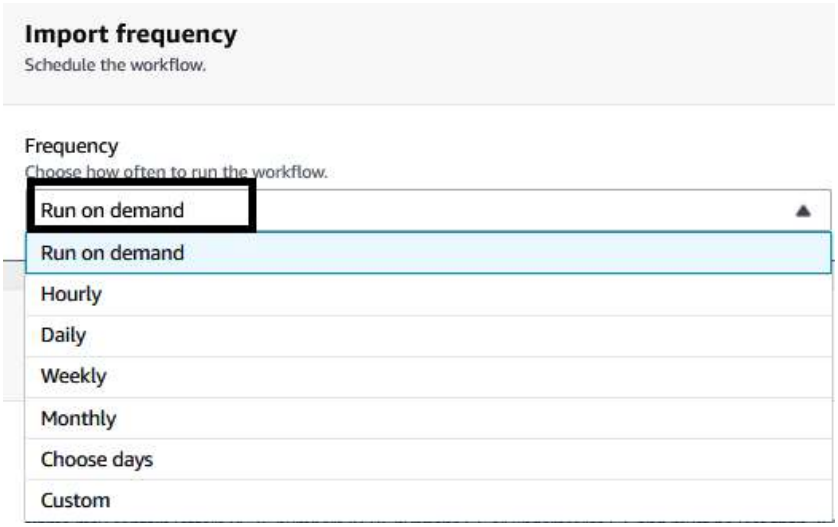


Figure 10.13 – Configure the import frequency for the workflow

18. Under **Import options**, specify the name of the workflow as `hydrate-mysql`. Under **IAM role**, use **AWSGlueServiceRole-FooGlue**. For **Table prefix**, use `mysql`. Select **Create**.

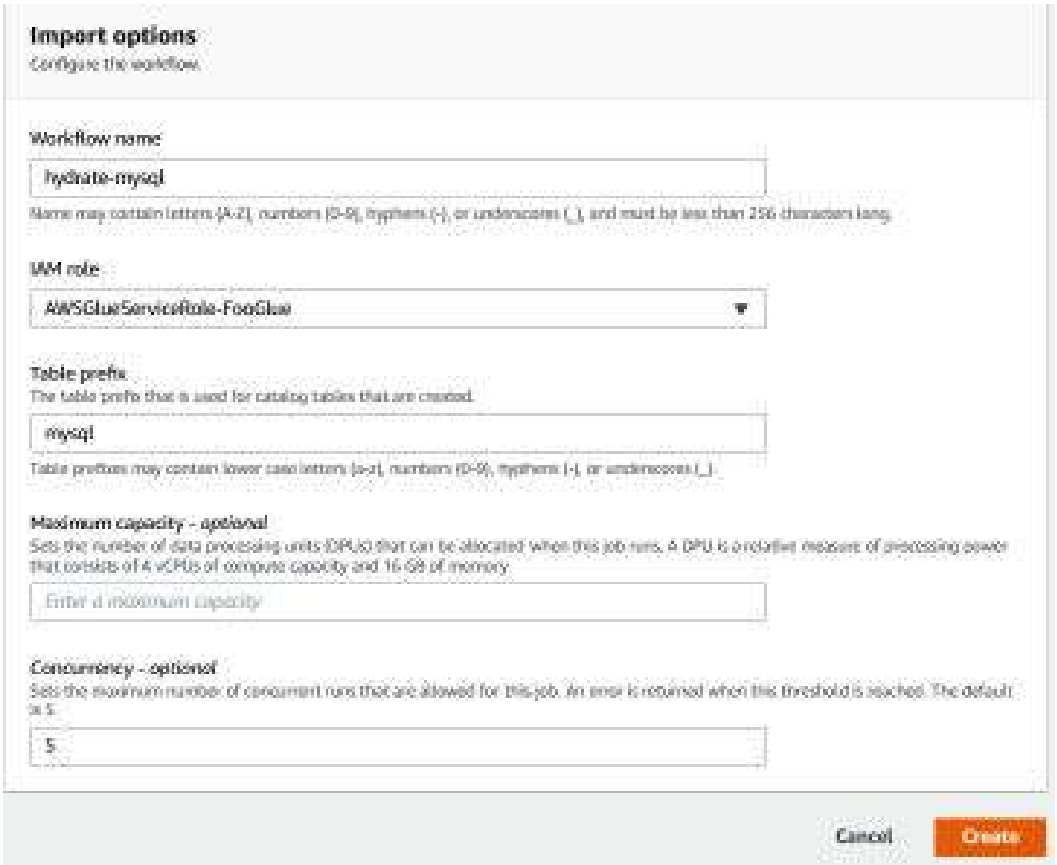


Figure 10.14 – Configure import options for the workflow

19. On creation of the workflow, select **Workflow**. Select **Action** and start the workflow:

    a. The workflow will crawl the `mysql` table metadata, which will catalog it in the `cookbook-data-lake` database.

    b. It will then unload the data from the `mysql ods.part` table in Parquet format into the S3 location you provided.

c. Finally, it will crawl the Parquet data in S3 and create a table in the `cookbook-data-lake` database.

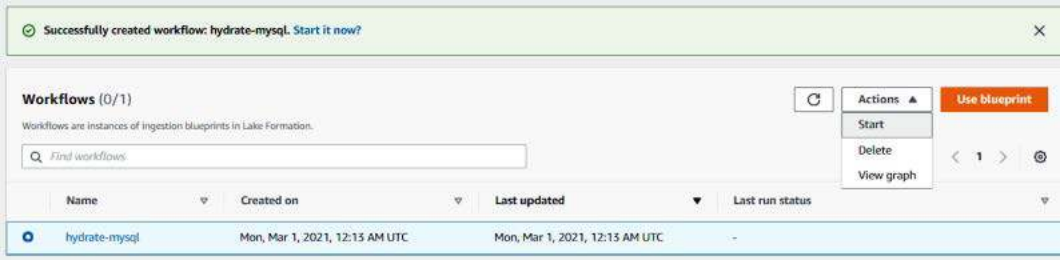


Figure 10.15 – Crawl the target S3 Parquet bucket

20. To view the status of the workflow, click on **runid**. Then, select **View graph**.

Figure 10.16 – Visualize the data workflow

21. You can view the workflow steps and the corresponding status of the steps.

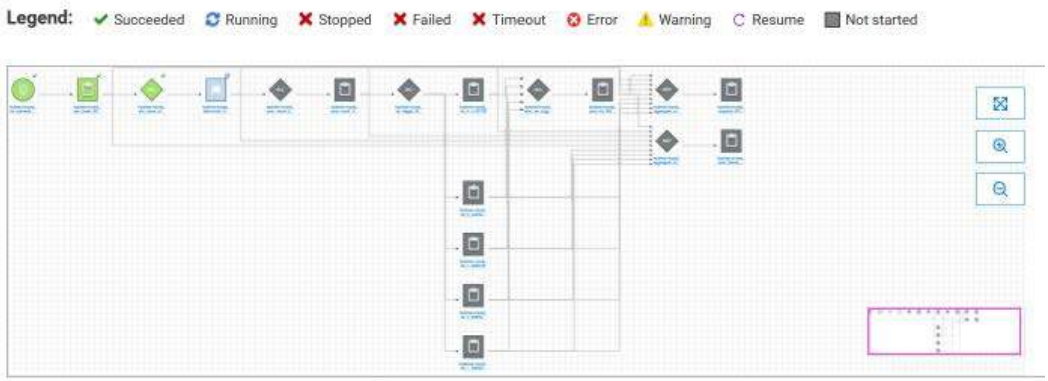


Figure 10.17 – Data workflow steps

22. On successful completion of the workflow, **Last run status** will be marked as **COMPLETED**.

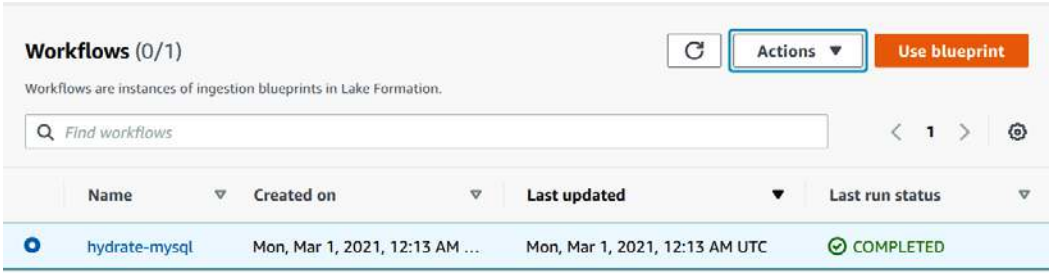


Figure 10.18 – Data workflow execution status

23. Let's now view the details of your first data lake. To view the tables created in your catalog, in the AWS Lake Formation console, from the left, select **Databases**. Select **cookbook-data-lake**.
24. Select **View tables**.

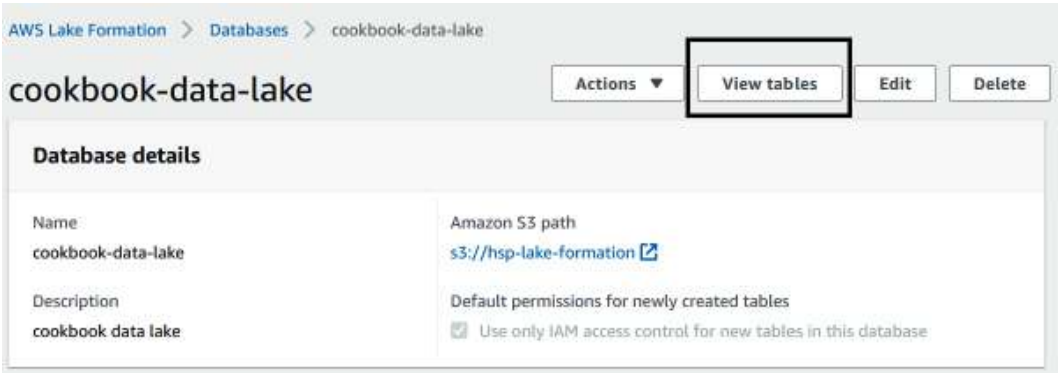


Figure 10.19 – View tables created for the target

25. Note that the blueprint cataloged the tables.

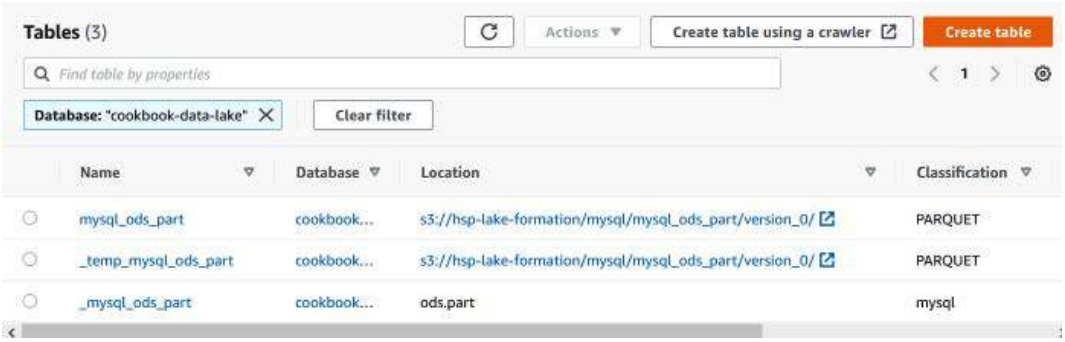


Figure 10.20 – Verify the target dataset

26. To view the metadata of the unloaded Parquet data, select the **mysql_ods_part** table. This table contains the metadata of the data. The crawler identified the column names and the corresponding data types.

Column #	Name	Data type
1	p_container	`string`
2	p_mfgr	`string`
3	p_comment	`string`
4	p_size	`int`
5	p_partkey	`bigint`
6	p_retailprice	`decimal(18,4)`
7	p_name	`string`
8	p_type	`string`
9	p_brand	`string`

Figure 10.21 – View metadata for the target

27. The classification is **PARQUET** and the table points to the location in S3 where the data resides.

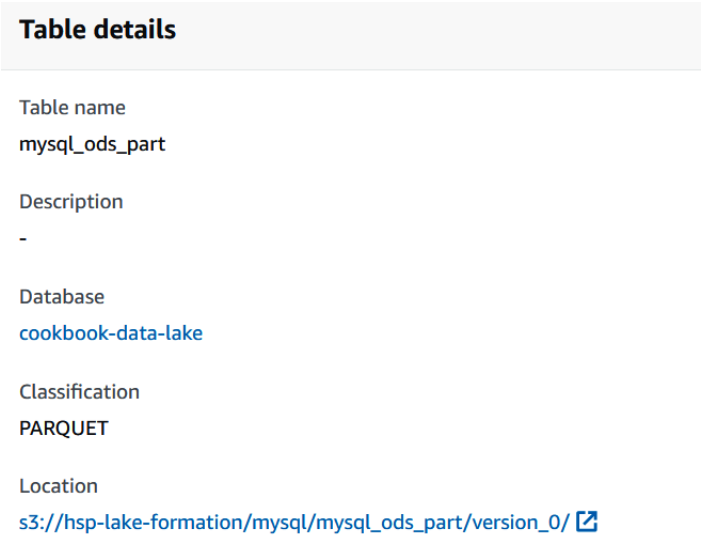


Figure 10.22 – Verify the target table format

28. To view the unloaded files on S3, navigate to your S3 location.

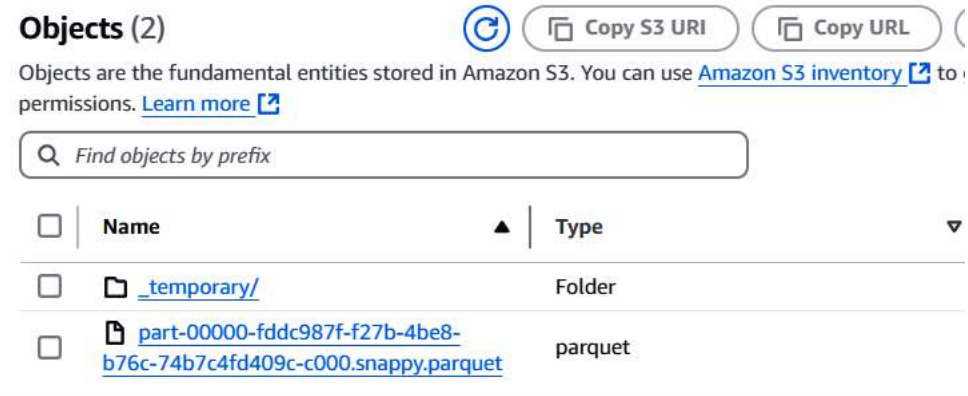


Figure 10.23 – Verify the underlying Parquet files in Amazon S3

29. Going back to AWS Lake Formation, let's see how the permissions can be managed. In this step, we will use the `mysql_ods_part` table. Select the **mysql_ods_part** table, select **Action**, and select **Grant**:

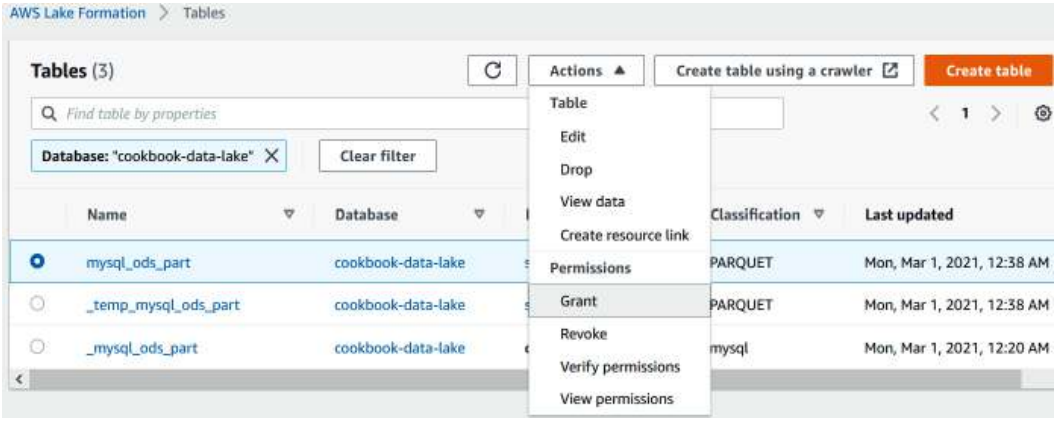


Figure 10.24 – Set up permission for the target dataset

30. AWS Lake Formation enables you to centralize the process of configuring access permission to the IAM roles. Table-level and fine-grained access at the column level can be granted and controlled in a centralized place.

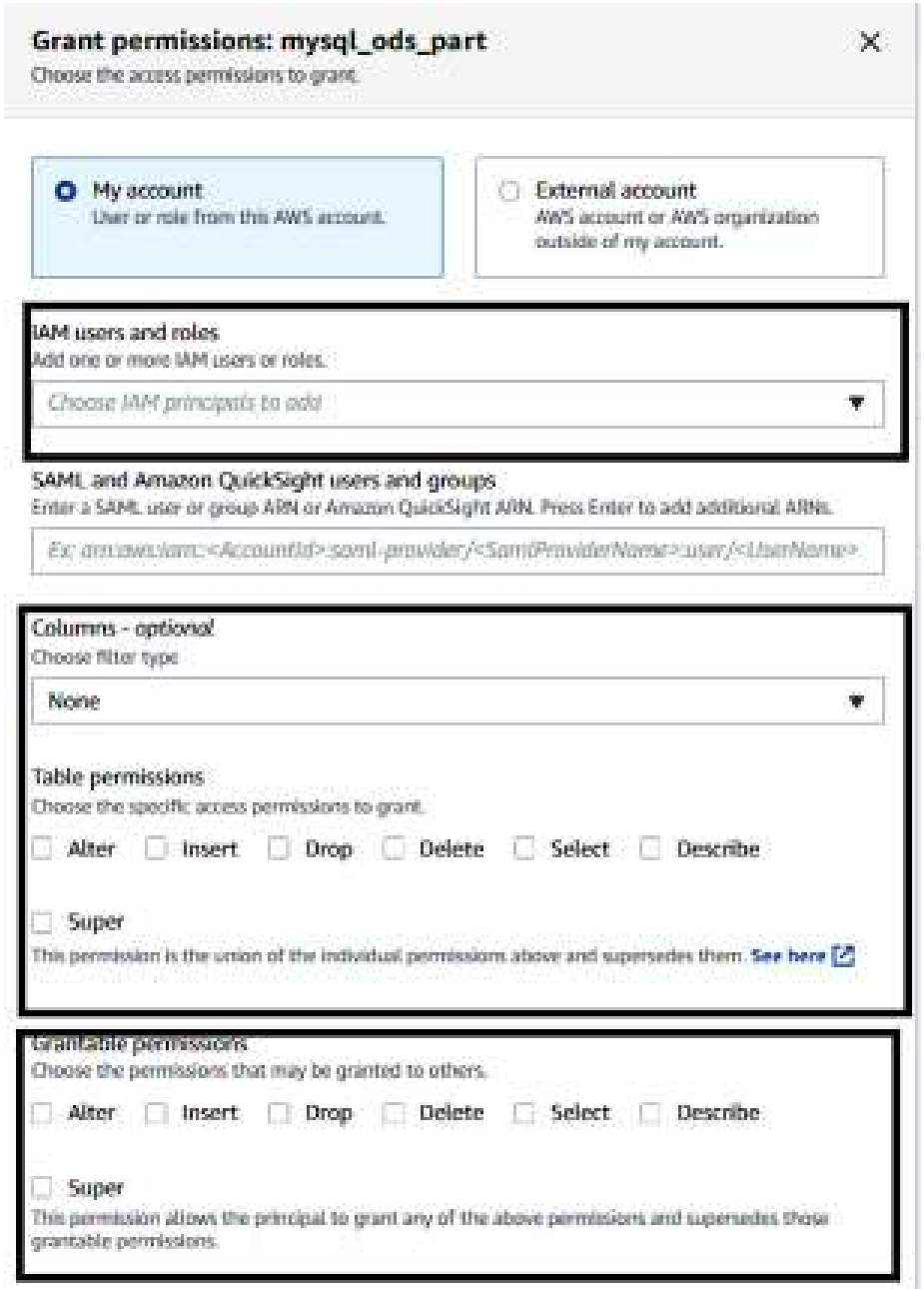


Figure 10.25 – Administer the Lake Formation catalog

Later in the chapter, in the *Extending a data warehouse using Redshift Spectrum* recipe, you will learn how to query this data using Amazon Redshift.

How it works...

AWS Lake Formation simplifies the management and configuration of a data lake by providing a centralized place for doing so. AWS Glue's extract, transform, load functionality, leveraging Python and the Spark shell, as well as ML transform, enables you to customize workflows to meet your needs. The AWS Glue/Lake Formation catalog integrates with Amazon Redshift for your data warehousing, Amazon Athena for ad hoc analysis, Amazon SageMaker for predictive analysis, and Amazon EMR for big data processing.

Carrying out a data lake export from Amazon Redshift

Amazon Redshift allows the use of the lakehouse architecture, enabling you to query data within a data warehouse or a data lake using Amazon Redshift Spectrum and also export your data back to the data lake on Amazon S3, to be used by other analytical and ML services. You can store data in open file formats in your Amazon S3 data lake when performing the data lake export to integrate with your existing data lake formats.

Getting ready

To complete this recipe, you will need the following setup:

- An IAM user with access to Amazon Redshift
- An Amazon Redshift data warehouse deployed in the AWS Region eu-west-1 with the retail dataset created in *Chapter 3*, using the recipe *Loading data from Amazon S3 using COPY*
- Amazon Redshift data warehouse admin user credentials
- Access to any SQL interface, such as a SQL client or Amazon Redshift Query Editor
- An AWS account number; we will refer to it in the recipes with `[Your-AWS_Account_Id]`
- An Amazon S3 bucket created in eu-west-1; we will refer to it with `[Your-Amazon_S3_Bucket]`
- An IAM role attached to an Amazon Redshift data warehouse that can access Amazon S3; we will refer to it in the recipes with `[Your-Redshift_Role]`

How to do it...

In this recipe, we will use the sample dataset created in *Chapter 3*, using the recipe *Loading data from Amazon S3 using COPY* to write the data back to an Amazon S3 data lake:

1. Connect to the Amazon Redshift data warehouse using a client tool such as SQL Workbench.

2. Execute the following analytical query to verify the sample dataset:

```
SELECT c_mktsegment,
       COUNT(o_orderkey) AS orders_count,
       SUM(l_quantity) AS quantity,
       COUNT(DISTINCT P_PARTKEY) AS parts_count,
       COUNT(DISTINCT L_SUPPKEY) AS supplier_count,
       COUNT(DISTINCT o_custkey) AS customer_count
FROM lineitem
  JOIN orders ON l_orderkey = o_orderkey
  JOIN customer c ON o_custkey = c_custkey
  JOIN dwdate
    ON d_date = l_commitdate
   AND d_year = 1992
  JOIN part ON P_PARTKEY = l_PARTKEY
  JOIN supplier ON L_SUPPKEY = S_SUPPKEY
GROUP BY c_mktsegment limit 5;
```

Here's the expected sample output:

```
c_mktsegment | orders_count |   quantity   | parts_count | supplier_
count | customer_count
--------------+--------------+--------------+-------------+---------
-------+----------------
 MACHINERY    |        82647 | 2107972.0000 |       75046 |
72439 |          67404
 AUTOMOBILE   |        82692 | 2109248.0000 |       75039 |
72345 |          67306
 HOUSEHOLD    |        82521 | 2112594.0000 |       74879 |
72322 |          67035
 BUILDING     |        83140 | 2115677.0000 |       75357 |
72740 |          67411
 FURNITURE    |        83405 | 2129150.0000 |       75759 |
73048 |          67876
```

3. Create a schema to point to the data lake using the following command, replacing `[Your-AWS_Account_Id]` and `[Your-Redshift_Role]` with the relevant values:

```
CREATE external SCHEMA datalake_ext_schema
FROM data catalog DATABASE 'datalake_ext_schema'
```

```
iam_role 'arn:aws:iam::[Your-AWS_Account_Id]:role/[Your-Redshift_
Role] '
CREATE external DATABASE if not exists;
```

4. Create the external table that will be used to export the dataset:

```
CREATE external TABLE datalake_ext_schema.order_summary
  (c_mktsegment VARCHAR(10),
   orders_count BIGINT,
   quantity numeric(38,4),
   parts_count BIGINT,
   supplier_count BIGINT,
   customer_count BIGINT
   )
STORED
AS
PARQUET LOCATION
's3://[Your-Amazon_S3_Bucket]/order_summary/';
```

Note

You can specify the output data format as PARQUET. You can use any of the supported data formats, listed here: `https://docs.aws.amazon.com/redshift/latest/dg/c-spectrum-data-files.html`.

5. Use the results of the above analytical query to export the data into the external table, which will be stored in Parquet format in Amazon S3. Use the following command:

```
INSERT INTO datalake_ext_schema.order_summary
SELECT c_mktsegment,
       COUNT(o_orderkey) AS orders_count,
       SUM(l_quantity) AS quantity,
       COUNT(DISTINCT P_PARTKEY) AS parts_count,
       COUNT(DISTINCT L_SUPPKEY) AS supplier_count,
       COUNT(DISTINCT o_custkey) AS customer_count
FROM lineitem
  JOIN orders ON l_orderkey = o_orderkey
  JOIN customer c ON o_custkey = c_custkey
  JOIN dwdate
```

```
    ON d_date = l_commitdate
   AND d_year = 1992
  JOIN part ON P_PARTKEY = l_PARTKEY
  JOIN supplier ON L_SUPPKEY = S_SUPPKEY
GROUP BY c_mktsegment;
```

6. You can now verify the results of the export using the following command:

```
select * from datalake_ext_schema.order_summary limit 5;
```

Here's the expected sample output:

```
c_mktsegment | orders_count |   quantity    | parts_count | supplier_
count | customer_count
--------------+--------------+--------------+-------------+---------
-------+----------------
 HOUSEHOLD    |        82521 | 2112594.0000 |       74879 |
72322 |          67035
 MACHINERY    |        82647 | 2107972.0000 |       75046 |
72439 |          67404
 FURNITURE    |        83405 | 2129150.0000 |       75759 |
73048 |          67876
 BUILDING     |        83140 | 2115677.0000 |       75357 |
72740 |          67411
 AUTOMOBILE   |        82692 | 2109248.0000 |       75039 |
72345 |          67306
```

7. Inspect the Amazon S3 location `s3://[Your-Amazon_S3_Bucket]/order_summary/` for the presence of Parquet files, as shown below:

```
$ aws s3 ls s3://[Your-Amazon_S3_Bucket]/order_summary/

-- expected sample output--
2021-03-02 00:00:11       1588 20210302_000002_331241_25860550_00
02_part_00.parquet
2021-03-02 00:00:11       1628 20210302_000002_331241_25860550_00
13_part_00.parquet
2021-03-02 00:00:11       1581 20210302_000002_331241_25860550_00
16_part_00.parquet
2021-03-02 00:00:11       1581 20210302_000002_331241_25860550_00
20_part_00.parquet
```

The previous sample output shows a list of all the Parquet files in the external table.

Extending a data warehouse using Amazon Redshift Spectrum

Amazon Redshift Spectrum empowers Amazon Redshift customers to directly query data from Amazon S3. This capability enables the seamless integration of data warehouse data with a data lake, leveraging open-source file formats such as Parquet, CSV, Sequence, Avro, and more. Furthermore, it allows querying data in open table formats like Apache Iceberg and Hudi. As a serverless solution, Amazon Redshift Spectrum relieves customers of the burden of provisioning or managing infrastructure. It enables unified analytics on data residing in both Amazon Redshift data warehouses and Amazon S3 data lakes, facilitating the effortless creation of insights from disparate datasets.

Getting ready

To complete this recipe, you will need the following setup:

- An IAM user with access to Amazon Redshift
- An Amazon Redshift data warehouse deployed in the AWS Region eu-west-1 with the retail dataset created from *Chapter 3*, using the recipe *Loading data from Amazon S3 using COPY*
- Amazon Redshift data warehouse admin user credentials
- Access to any SQL interface, such as a SQL client or Amazon Redshift Query Editor
- An AWS account number; we will refer to it in the recipes with `[Your-AWS_Account_Id]`
- An Amazon S3 bucket created in eu-west-1; we will refer to it with `[Your-Amazon_S3_Bucket]`
- An IAM role attached to an Amazon Redshift data warehouse that can access Amazon S3 and AWS Glue; we will refer to it in the recipes with `[Your-Redshift_Role]`

How to do it...

In this recipe, we will create an external table in an external schema and query data directly from Amazon S3 using Amazon Redshift:

1. Connect to the Amazon Redshift data warehouse using a client tool such as SQL Workbench.

2. Execute the following query to create an external schema, replacing `[Your-AWS_Account_Id]` and `[Your-Redshift_Role]` with the respective values:

```
create external schema packt_spectrum
from data catalog
database 'packtspectrumdb'
iam_role 'arn:aws:iam::[Your-AWS_Account_Id]:role/[Your-Redshift_
Role]'
create external database if not exists;
```

3. Execute the following command to copy data from the Packt S3 bucket (`s3://packt-redshift-cookbook/`) to your S3 bucket by replacing `[Your-Amazon_S3_Bucket]` with the respective value in the following command:

```
aws cp s3://packt-redshift-cookbook/spectrum/sales s3://[Your-
Amazon_S3_Bucket]/spectrum/sales --recursive
```

4. Execute the following query to create an external table, replacing `[Your-Amazon_S3_Bucket]` with the respective value:

```
create external table packt_spectrum.sales(
salesid integer,
listid integer,
sellerid integer,
buyerid integer,
eventid integer,
dateid smallint,
qtysold smallint,
pricepaid decimal(8,2),
commission decimal(8,2),
saletime timestamp)
row format delimited
fields terminated by '\t'
stored as textfile
location 's3://[Your-Amazon_S3_Bucket]/spectrum/sales/'
table properties ('numRows'='172000');
```

5. Execute the following command to query data in S3 directly from Amazon Redshift:

```
select count(*) from packt_spectrum.sales;
```

```
--
expected sample output -
count
------
172462
```

6. Execute the following command to create a table locally in Amazon Redshift:

```
create table packt_event(
eventid integer not null distkey,
venueid smallint not null,
catid smallint not null,
dateid smallint not null sortkey,
eventname varchar(200),
starttime timestamp);
```

7. Execute the following command to load data in the event table, replacing `[Your-AWS_Account_Id]` and `[Your-Redshift_Role]` with the respective values:

```
copy packt_event from 's3://packt-redshift-cookbook/spectrum/event/
allevents_pipe.txt'
iam_role 'arn:aws:iam::[Your-AWS_Account_Id]:role/[Your-Redshift_
Role]
delimiter '|' timeformat 'YYYY-MM-DD HH:MI:SS' region 'us-east-1';
```

8. Execute the following query to join the data across a Redshift local table and a Spectrum table:

```
SELECT top 10 packt_spectrum.sales.eventid,
       SUM(packt_spectrum.sales.pricepaid)
FROM packt_spectrum.sales,
     packt_event
WHERE packt_spectrum.sales.eventid = packt_event.eventid
AND   packt_spectrum.sales.pricepaid > 30
GROUP BY packt_spectrum.sales.eventid
ORDER BY 2 DESC;


--
expected sample output--
eventid | sum
```

```
--------+---------
    289 | 51846.00
   7895 | 51049.00
   1602 | 50301.00
    851 | 49956.00
   7315 | 49823.00
   6471 | 47997.00
   2118 | 47863.00
    984 | 46780.00
   7851 | 46661.00
   5638 | 46280.00
```

Now, Amazon Redshift is able to join the external and the local tables to produce the desired results.

Querying an operational source using a federated query

Amazon Redshift federated queries enable unified analytics across databases, data warehouses, and data lakes. With the federated query feature in Amazon Redshift, you can query live data across Amazon RDS and Aurora PostgreSQL databases. For example, you might have up-to-date customer address data that you want to join with the historical orders data to enrich your reports. This can be easily done using the federated query feature.

Getting ready

To complete this recipe, you will need:

- An IAM user with access to Amazon Redshift, AWS Secrets Manager, and Amazon RDS.
- An Amazon Redshift data warehouse deployed in the AWS Region eu-west-1 with the retail sample data from *Chapter 3*.
- An Amazon Aurora serverless PostgreSQL database. Create an RDS PostgreSQL cluster using this guide: `https://aws.amazon.com/getting-started/hands-on/building-serverless-applications-with-amazon-aurora-serverless/`. Launch this in the same VPC as your Amazon Redshift data warehouse.
- Access to any SQL interface, such as a SQL client or Amazon Redshift Query Editor
- An IAM role attached to an Amazon Redshift data warehouse that can access Amazon RDS; we will refer to it in the recipes with `[Your-Redshift_Role]`.
- An AWS account number; we will refer to it in the recipes with `[Your-AWS_Account_Id]`.

How to do it...

In this recipe, we will use an Amazon Aurora serverless PostgreSQL database as the operational data store to federate with Amazon Redshift:

1. Let's connect to an Aurora PostgreSQL database using Query Editor. Navigate to the Amazon RDS landing page and select **Query Editor**.
2. Choose the RDS instance from the dropdown. Enter the username and password. For the database name, enter `postgres` and then select **Connect to database**.

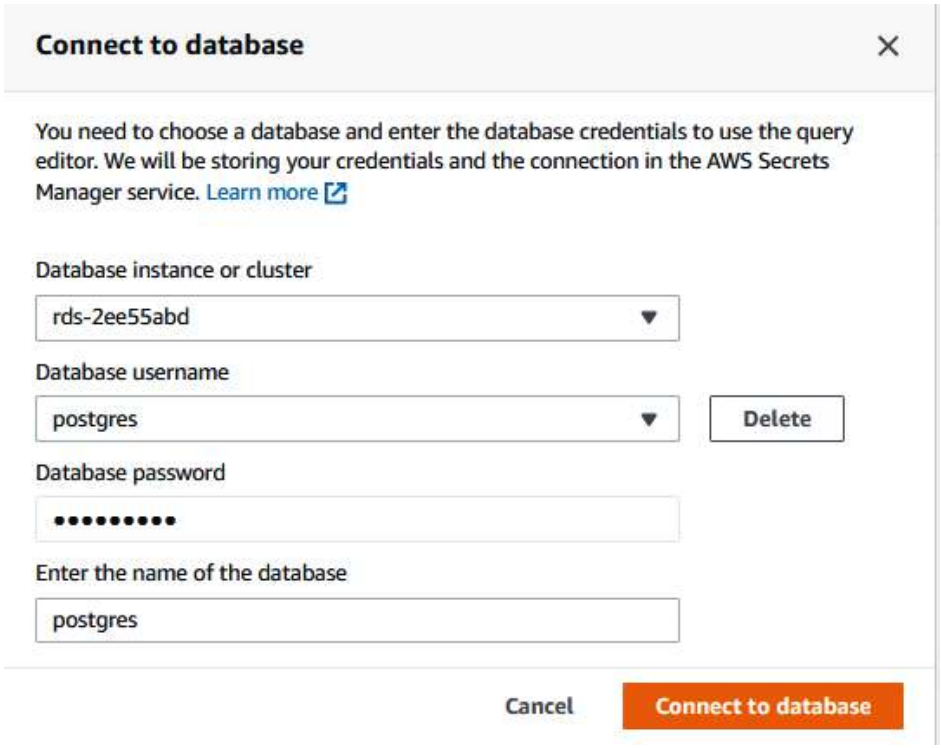


Figure 10.26 – Configure an Amazon Aurora PostgreSQL database

3. Copy and paste the SQL script available at `https://github.com/PacktPublishing/Amazon-Redshift-Cookbook-2E/blob/main/Chapter10/aurora_postgresql_orders_insert.sql` into the editor. Select **Run**.

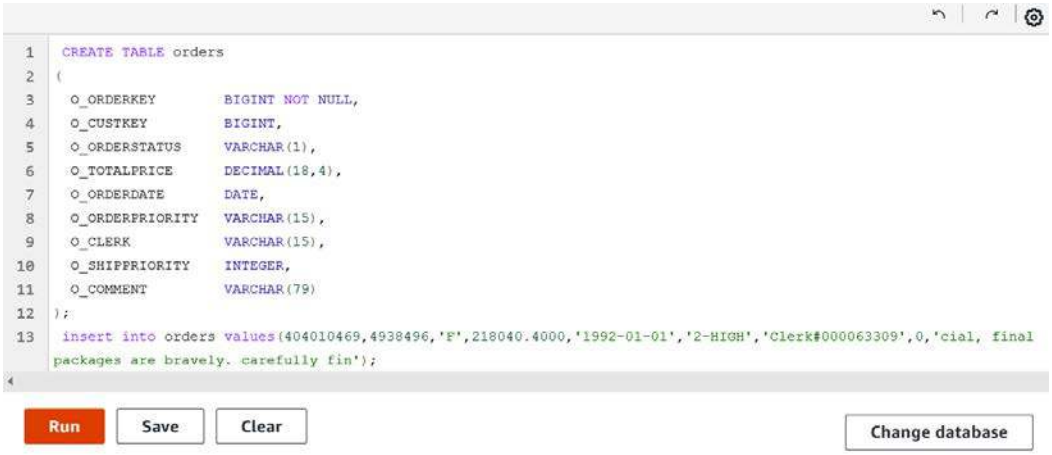


Figure 10.27 – Create the orders tables

4. We will now create an Aurora PostgreSQL database secret using AWS Secrets Manager to store the user ID and password.
5. Navigate to the AWS Secrets Manager console. Select **Store a new secret**.
6. Select **Credentials for RDS database**, then enter the username and password. Select your database instance and click **Next**.

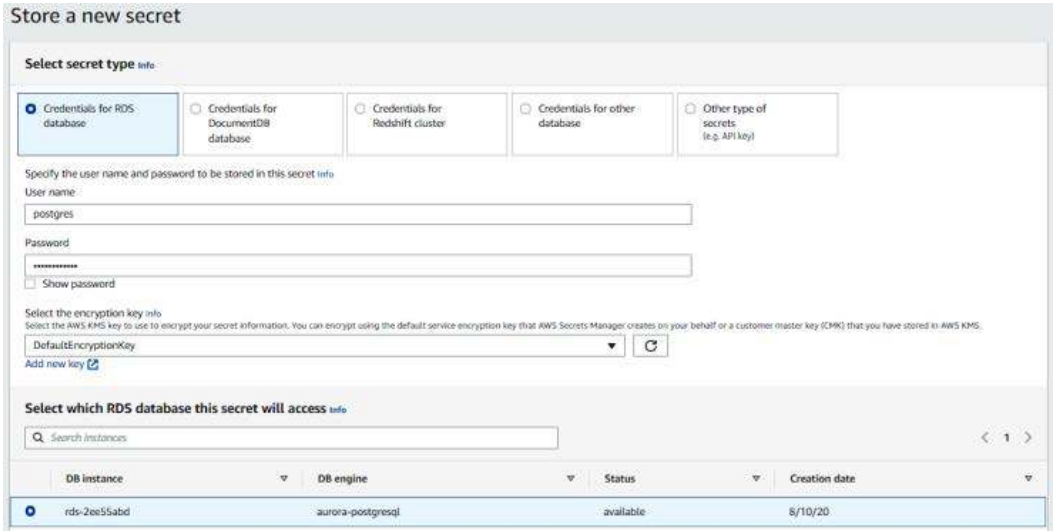


Figure 10.28 – Set up credentials for RDS

7. Enter the name `aurora-pg/RedshiftCookbook` for the secret. Select **Next**.

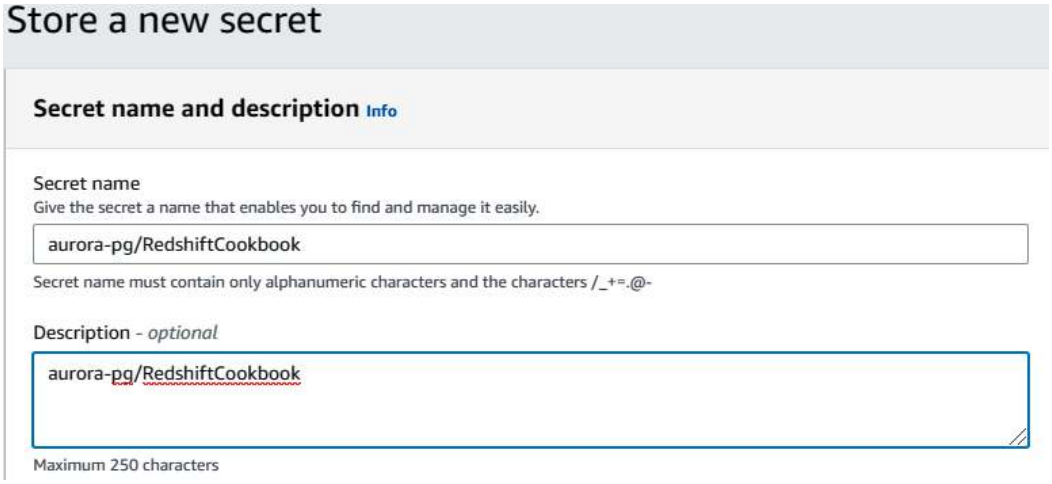


Figure 10.29 – Create the Aurora PostgreSQL secret

8. Select **Next**, keep the defaults, and click **Store**.
9. Select the newly created secret and copy the ARN of the secret.

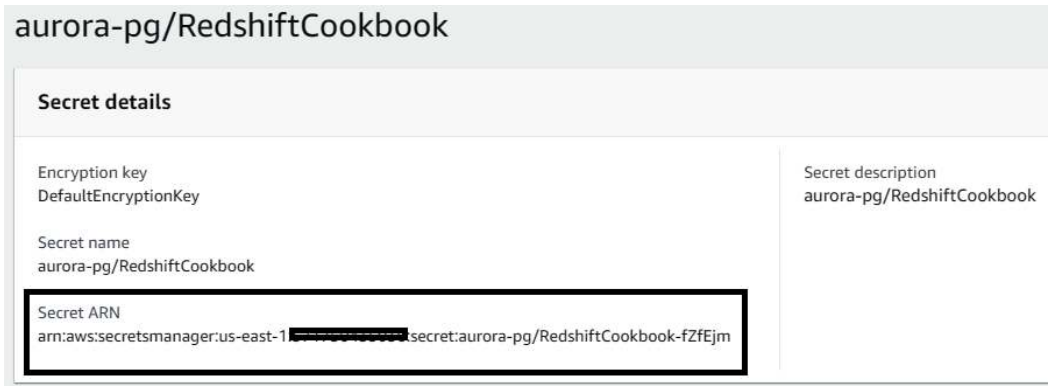


Figure 10.30 – Copy the ARN for the secret

10. To configure Amazon Redshift to federate with the Aurora PostgreSQL database, we need to attach an inline policy to the IAM role attached to our Amazon Redshift data warehouse to provide access to the secret created in the preceding steps. To do this, navigate to the IAM console and select **Roles**.

11. Search for the role. Add the following inline policy. Replace `[Your-AWS_Account_Id]` with your AWS account number:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AccessSecret",
            "Effect": "Allow",
            "Action": [
                "secretsmanager:GetResourcePolicy",
                "secretsmanager:GetSecretValue",
                "secretsmanager:DescribeSecret",
                "secretsmanager:ListSecretVersionIds"
            ],
            "Resource": "arn:aws:secretsmanager:us-east-1:[Your-AWS_
Account_Id]:secret:aurora-pg/RedshiftCookbook"
        },
        {
            "Sid": "VisualEditor1",
            "Effect": "Allow",
            "Action": [
                "secretsmanager:GetRandomPassword",
                "secretsmanager:ListSecrets"
            ],
            "Resource": "*"
        }
    ]
}
```

12. Let's set up Amazon Redshift to federate to an Aurora PostgreSQL database to query the orders operational data. For this, connect to your Amazon Redshift data warehouse using a SQL client or Query Editor from the Amazon Redshift console.

13. Create the external schema `ext_postgres` on Amazon Redshift. Replace `[AuroraClusterEndpoint]` with the endpoint of the instance from your account for the Aurora PostgreSQL database. Replace `[Your-AWS_Account_Id]` and `[Your-Redshift-Role]` with the respective values. Also, replace `[AuroraPostgreSQLSecretsManagerARN]` with the value of the secret ARN from *Step 9*:

```
DROP SCHEMA IF EXISTS ext_postgres;
CREATE EXTERNAL SCHEMA ext_postgres
FROM POSTGRES
DATABASE 'postgres'
URI '[AuroraClusterEndpoint]'
IAM_ROLE 'arn:aws:iam::[Your-AWS_Account_Id]:role/[Your-Redshift-
Role]'
SECRET_ARN '[AuroraPostgreSQLSecretsManagerARN]';
```

14. To list the external schemas, execute the following query:

```
select *
from svv_external_schemas;
```

15. To list the external schema tables, execute the following query

```
select *
from svv_external_tables
where schemaname = 'ext_postgres';
```

16. To validate the configuration and the setup of the federated query from Amazon Redshift, let's execute the `count` query for the `orders` table in the Aurora PostgreSQL database:

```
select count(*) from ext_postgres.orders;
```

Here's the expected output:

```
1000
```

17. With a federated query, you can join an external table with an Amazon Redshift local table:

```
SELECT O_ORDERSTATUS,
       COUNT(o_orderkey) AS orders_count
FROM ext_postgres.orders
    JOIN dwdate
```

```
    ON d_date = O_ORDERDATE
   AND d_year = 1992
GROUP BY O_ORDERSTATUS;
```

Here's the expected output:

```
o_orderstatus    orders_count
F                1000
```

18. You can also create a materialized view using a federated query. The results of a materialized view query, typically stored virtually in a view, will be stored as a physical table on Amazon Redshift. You can refresh the materialized view to get more fresh data from your ODS:

```
create materialized view public.live_orders as
SELECT O_ORDERSTATUS,
       COUNT(o_orderkey) AS orders_count
FROM ext_postgres.orders
    JOIN dwdate
    ON d_date = O_ORDERDATE
   AND d_year = 1992
GROUP BY O_ORDERSTATUS;
```

As observed, the materialized view can federate between Aurora PostgreSQL and Amazon Redshift.

Amazon SageMaker Lakehouse

Amazon SageMaker Lakehouse unifies all your data across Amazon S3 data lakes and Amazon Redshift data warehouses, helping you build powerful analytics and AI/ML applications on a single copy of data. SageMaker Lakehouse gives you the flexibility to access and query your data in place with all Apache Iceberg-compatible tools and engines. You can secure your data in the lakehouse by defining fine-grained permissions that are enforced across all analytics and ML tools and engines. You can bring data from operational databases and applications into your lakehouse in near-real time through zero-ETL integrations.

Additionally, you can access and query data in place with federated query capabilities across third-party data sources.

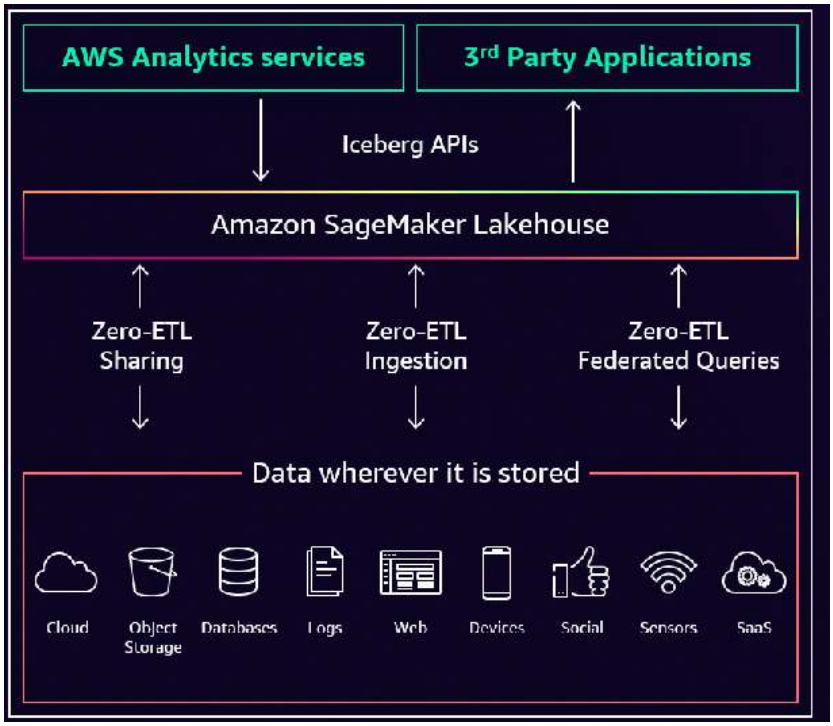


Figure 10.31 – Amazon SageMaker Lakehouse

Amazon SageMaker Lakehouse is accessible from the AWS Management Console via the Unified Studio, using APIs, the AWS CLI, or AWS SDKs. Data from different sources is organized into logical containers called catalogs in SageMaker Lakehouse. Each catalog represents data either from existing data sources, such as Amazon Redshift data warehouses, data lakes, or databases, or new catalogs, which can be directly created in the lakehouse to store data in Amazon S3 or Amazon RMS. Data in SageMaker Lakehouse can be accessed from Apache Iceberg-compatible engines such as Apache Spark, Athena, or Amazon EMR. Additionally, these catalogs can be discovered as databases in Amazon Redshift data warehouses, allowing you to use your SQL tools to analyze your lakehouse data.

In this recipe, we will register an Amazon Redshift data warehouse as a federated source and query data using Amazon Redshift and Amazon Athena.

Getting ready

To complete this recipe, you will need the following:

- An IAM user with access to Amazon Redshift and Lake Formation with admin access.
- Two separate data warehouses, one with a two-node Amazon Redshift `ra3.xlplus` provisioned cluster and one with Amazon Redshift Serverless deployed in the AWS Region eu-west-1 in `account1`.
- Access to any SQL interface, such as a SQL client or Amazon Redshift Query Editor V2.
- Add `AWSServiceRoleForRedshift` as a read-only data lake admin role in AWS Lake Formation.

How to do it...

In the recipe, we will use an Amazon Redshift RA3 provisioned cluster:

1. Connect to the Amazon Redshift provisioned cluster using a client tool such as SQL Workbench or Query Editor V2.
2. Run the following SQL to create sales and marketing schemas and tables:

```
create schema sales;
create schema marketing;
-- Create Marketing Campaigns Table
CREATE TABLE marketing.marketing_campaigns (
    campaign_id VARCHAR(20) PRIMARY KEY,
    campaign_name VARCHAR(100),
    campaign_type VARCHAR(50),
    channel VARCHAR(50),
    start_date DATE,
    end_date DATE,
    budget DECIMAL(12,2),
    target_audience VARCHAR(100),
    status VARCHAR(20)
);
-- Create Sales Transactions Table
CREATE TABLE sales.sales_transactions (
    transaction_id VARCHAR(20) PRIMARY KEY,
    campaign_id VARCHAR(20),
    customer_id VARCHAR(20),
    transaction_date TIMESTAMP,
```

```
    product_id VARCHAR(20),
    quantity INTEGER,
    unit_price DECIMAL(10,2),
    total_amount DECIMAL(12,2),
    region VARCHAR(50),
    source VARCHAR(50)
);
-- Insert sample data for Marketing Campaigns
INSERT INTO marketing.marketing_campaigns VALUES
('CAM001', 'Summer Sale 2024', 'Seasonal', 'Email', '2024-06-01',
'2024-06-30', 50000.00, 'All Customers', 'Active'),
('CAM002', 'Social Media Push', 'Digital', 'Social Media', '2024-05-
15', '2024-07-15', 75000.00, 'Young Adults', 'Active'),
('CAM003', 'Holiday Special', 'Seasonal', 'Multi-channel', '2024-12-
01', '2024-12-31', 100000.00, 'All Customers', 'Planned');
-- Insert sample data for Sales Transactions
INSERT INTO sales.sales_transactions VALUES
('T001', 'CAM001', 'CUST001', '2024-06-01 10:30:00', 'PROD001', 2,
99.99, 199.98, 'North', 'Email'),
('T002', 'CAM002', 'CUST002', '2024-06-01 11:45:00', 'PROD002', 1,
149.99, 149.99, 'South', 'Email'),
('T003', 'CAM003', 'CUST003', '2024-05-15 14:20:00', 'PROD003', 3,
79.99, 239.97, 'West', 'Instagram');
```

3. Navigate to the Redshift console and choose **Provisioned clusters dashboard** from the left-hand navigation panel.
4. Choose the provisioned cluster. Select **Actions**. Select **Register with AWS Glue Data Catalog**. Click **Register**. This will register the metadata of databases and schemas with AWS Glue Data Catalog as the federated catalog.

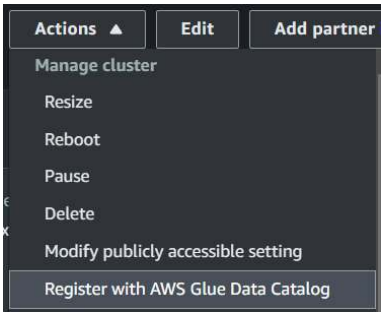


Figure 10.32 – Register an Amazon Redshift data warehouse

5. Navigate to the AWS Lake Formation console using the LF Admin role. You will see an invitation.

Figure 10.33 – AWS Lake Formation catalog invitation

6. Select the invitation. Click **Approve and create catalog**.
7. Under **Catalog details**, use `sales_marketing` as the name.

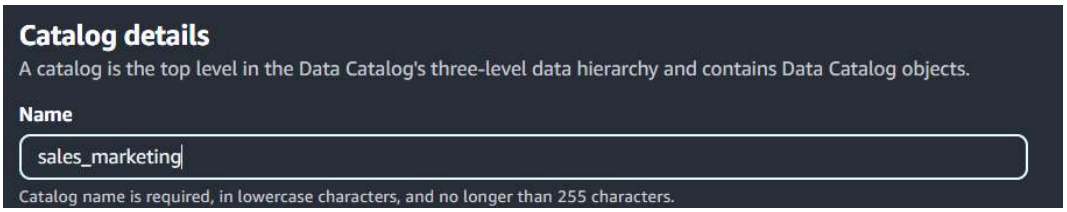


Figure 10.34 – Catalog details specifying the name

8. Enable the checkbox **Access this catalog from Apache Iceberg compatible engines**.
9. Under **IAM role**, choose **DataTransferRole**. Click **Next**.

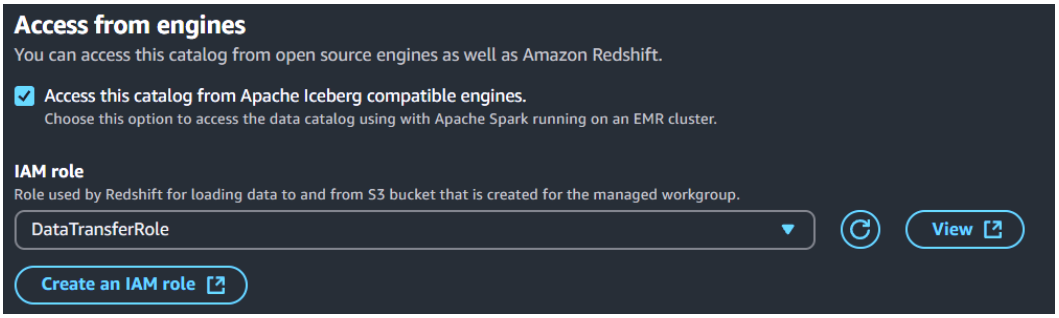


Figure 10.35 – Catalog configuration to allow other Apache Iceberg engines access

10. Click **Add permissions**. Choose **IAM users and roles**. Select the **chapter_10** role. Under **Catalog permissions**, select the **Super user** permission:

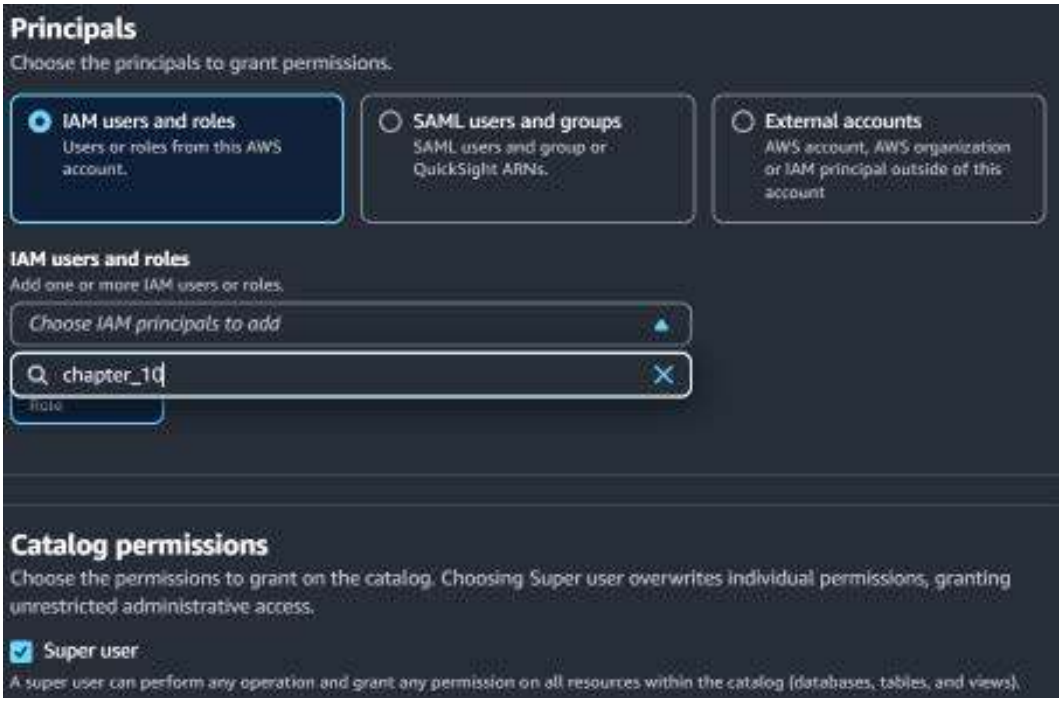


Figure 10.36 – Catalog IAM principal access settings

11. Click **Next**. Select **Create catalog**.
12. Under **Catalog**, select **dev**. You will see databases listing Redshift schemas. AWS Glue Data Catalog supports a hierarchy of registered sources.

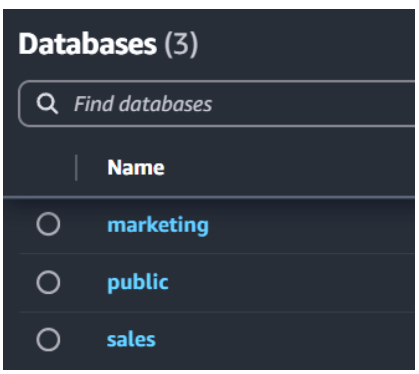


Figure 10.37 – Catalog listing databases

13. Choose **Marketing**. Select **View** and choose **Tables**.

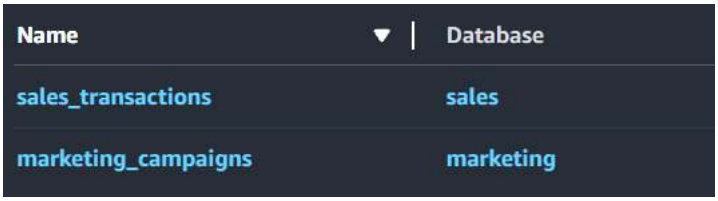


Figure 10.38 – Catalog listing tables

14. Once your Redshift data warehouse is registered, all data warehouses in that account and Region will see Glue Data Catalogs auto-mounted as data shares.
15. Navigate to Query Editor V2 using the `chapter_10` role and log on to Redshift Serverless using a federated role.
16. Browse the external databases and you will see datashare **dev@sales_marketing**.

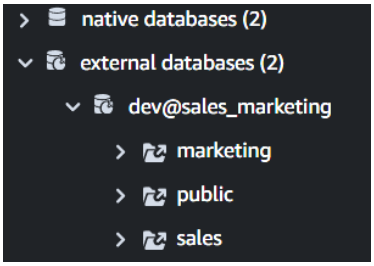


Figure 10.39 – Amazon Redshift consumer Query Editor V2 listing data shared schemas

17. Select the database **dev@sales_marketing** in Query Editor. Run the following analytical query.

```
SELECT
    mc.campaign_name,
    mc.channel,
    COUNT(st.transaction_id) as total_transactions,
    SUM(st.total_amount) as total_revenue,
    SUM(st.total_amount)/COUNT(st.transaction_id) as avg_
transaction_value
FROM
    marketing.marketing_campaigns mc
    LEFT JOIN sales.sales_transactions st ON mc.campaign_id =
st.campaign_id
GROUP BY
```

```
    mc.campaign_name,
    mc.channel
ORDER BY
    total_revenue DESC;
```

Result 1 (3)

campaign_name	channel	total_transactions	total_revenue	avg_transaction_value
Holiday Special	Multi-channel	1	239.97	239.97
Summer Sale 2024	Email	1	199.98	199.98
Social Media Push	Social Me-dia	1	149.99	149.99

Figure 10.40 – Amazon Redshift Query Editor V2 query results

18. The Redshift data warehouse is registered to AWS Glue Data Catalog as an Apache Iceberg-compatible table. Glue Data Catalog supports the Apache Iceberg REST catalog API. This allows a registered Redshift federated catalog to be queried by engines that support Apache Iceberg libraries.
19. Navigate to the Amazon Athena console using the `chapter_10` role.
20. Select **Editor**. Select **sales_marketing/dev** as the catalog. You will see the **sales** and **marketing** schemas. Choose **marketing**.

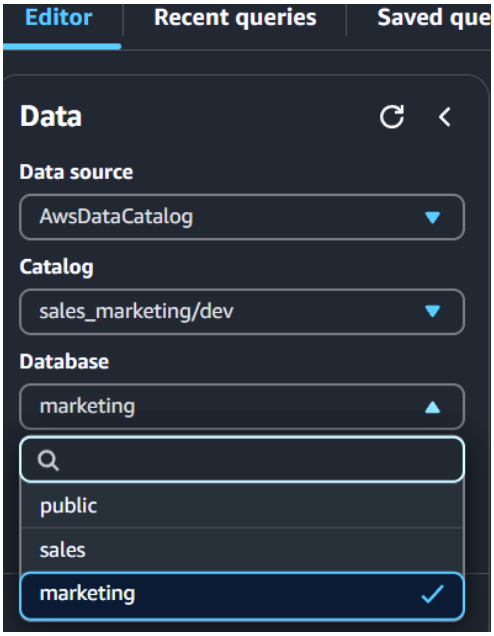


Figure 10.41 – Amazon Athena editor listing Redshift federated catalog databases

21. Let's run the following analytical query:

```
SELECT
    mc.campaign_name,
    mc.channel,
    COUNT(st.transaction_id) as total_transactions,
    SUM(st.total_amount) as total_revenue,
    SUM(st.total_amount)/COUNT(st.transaction_id) as avg_
transaction_value
FROM
    marketing.marketing_campaigns mc
    LEFT JOIN sales.sales_transactions st ON mc.campaign_id =
st.campaign_id
GROUP BY
    mc.campaign_name,
    mc.channel
ORDER BY
    total_revenue DESC;
```

Figure 10.42 – Analytical query

How it works...

Amazon SageMaker Lakehouse uses AWS Glue Data Catalog as its unified technical catalog. AWS Lake Formation manages security and access control for this catalog. Both federated and managed data tables are registered using the Apache Iceberg table format for compatibility.

AWS Glue Data Catalog supports Apache Iceberg's REST catalog API, enabling interoperability with AWS analytics services, open-source Spark, and third-party engines that support Apache Iceberg libraries.

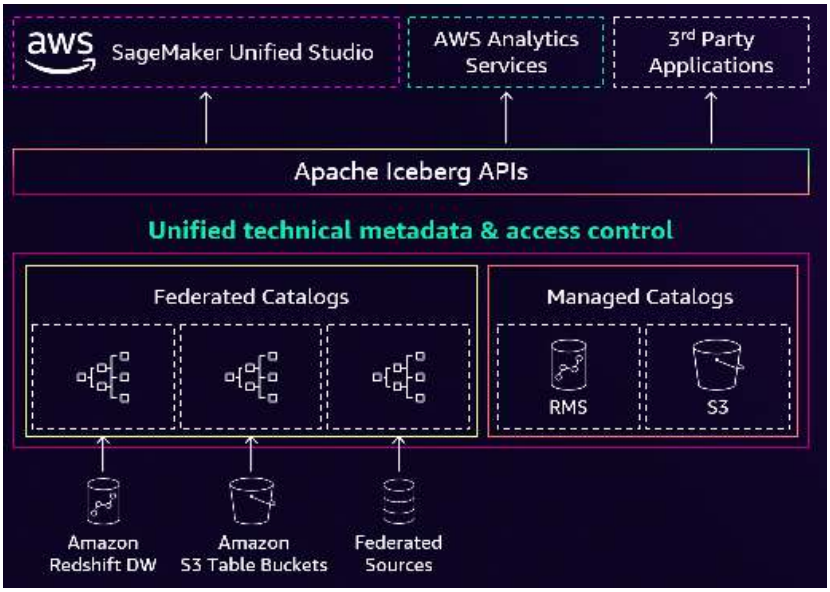


Figure 10.43 – Amazon SageMaker Lakehouse

11

Data Sharing with Amazon Redshift

Amazon Redshift's decoupled architecture to separate storage from compute provides the capability to share data. Data sharing allows Redshift data warehouses to securely share data with other Redshift data warehouses, within the same AWS account, across different AWS accounts, and across different AWS Regions. This data-sharing capability provides a live, transactionally consistent view of the data without the need to move or copy the data physically.

Here are the key benefits of Redshift data sharing:

1. **Workload isolation**: Redshift data sharing allows you to isolate workloads across different teams or business units, ensuring that one team's queries or data processing doesn't impact the performance of another team's workload.
2. **Clear chargeback**: With data sharing, you can clearly identify and charge back the costs associated with data access and usage to the appropriate teams or business units.
3. **Cross-collaboration**: Data sharing enables cross-team and cross-organizational collaboration by making datasets easily accessible to authorized users, fostering data-driven decision-making across the enterprise.
4. **Scalable read and write access**: Redshift data sharing allows you to scale read and write access to data, enabling multiple teams to access and work with the same datasets concurrently without the need to create duplicate copies.
5. **Monetize data as a service**: Organizations can leverage Redshift data sharing to monetize their data by offering it as a service to external customers or partners, generating additional revenue streams.

Common data-sharing deployment patterns are hub and spoke and data mesh, which allow you to implement multi-warehouse architecture:

- **Hub and spoke**: Using the hub-and-spoke architecture, you bring your data into one or more central warehouses that are generally used to write all the data. You then share your data with consumer warehouses that can read and write data. This pattern is also used for data as a service to share data with external customers.
- **Data mesh**: Using a data mesh architecture, different business groups can seamlessly share and collaborate on data. A common use case is enabling teams to collaborate on live datasets quickly and easily. For example, in this diagram, sales, marketing, and finance are all collaborating on a common customer 360 model.

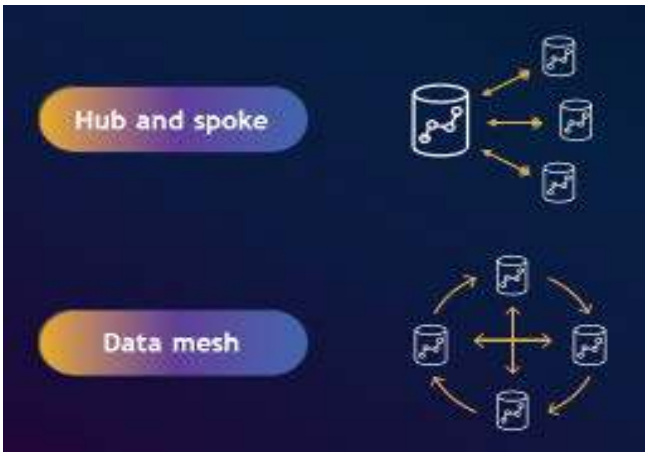


Figure 11.1 – Data-sharing patterns

The following recipes are covered in this chapter:

- Data sharing read access across multiple Amazon Redshift data warehouses
- Data sharing write access across multiple Amazon Redshift data warehouses
- Data sharing using Amazon DataZone for cross-collaboration and self-service analytics
- Data sharing using AWS Data Exchange for monetization and subscribing to third-party data

Technical requirements

Here are the technical requirements in order to complete the recipes in this chapter:

- Access to the AWS Management Console.
- AWS administrator permission to create an IAM user by following *Recipe 1* in *Appendix*. This IAM user will be used in some of the recipes in this chapter.
- AWS administrator permission to create an IAM role by following *Recipe 3* in *Appendix*. This IAM role will be used in some of the recipes in this chapter.
- AWS administrator permission to deploy the AWS CloudFormation template at `https://github.com/PacktPublishing/Amazon-Redshift-Cookbook-2E/blob/main/Chapter11/chapter_11_CFN.yaml`. This creates:
    a. An IAM policy attached to the IAM user that will give them access to Amazon Redshift, Amazon S3, AWS KMS, AWS IAM, AWS CloudFormation, AWS CloudTrail, AWS Lake Formation, AWS Secrets Manager, AWS SageMaker, and AWS Glue
    b. An IAM policy attached to the IAM role that will allow an Amazon Redshift cluster to access Amazon S3, AWS Lake Formation, and AWS Glue
- An Amazon Redshift data warehouse (serverless or provisioned) deployed in the AWS Region eu-west-1 in an AWS account.
- Amazon Redshift data warehouse master user credentials.
- Attach an IAM role to the Amazon Redshift cluster by following *Recipe 4* in *Appendix*. Make a note of the IAM role name; we will refer to it in the recipes with `[Your-Redshift_Role]`.
- Access to any SQL interface such as a SQL client or Amazon Redshift Query Editor V2.
- An AWS Region; we will refer to it in the recipes with `[Your-AWS_Region]`.
- An AWS account number; we will refer to it in the recipes with `[Your-AWS_Account]`.
- An Amazon S3 bucket created in eu-west-1; we will refer to it with `[Your-Amazon_S3_Bucket]`.
- The code files can be found in the Git repo: `https://github.com/PacktPublishing/Amazon-Redshift-Cookbook-2E/tree/master/Chapter11`.

Data sharing read access across multiple Amazon Redshift data warehouses

Amazon Redshift RA3 and Amazon Redshift serverless architecture decouple storage and compute. Decoupled architecture provides the ability to scale compute independently to meet workload SLAs and only pay for storage that is consumed. The decoupled storage allows data to be read and written by different consumer data warehouses. This allows for workload isolation. The data producer data warehouse controls access to the data that is shared. This feature opens up the possibility of setting up a flexible multi-tenant architecture; for example, within an organization, data produced by a business unit can be shared with any of the different teams, such as marketing, finance, data science, etc., that can be independently consumed using their own Amazon Redshift clusters.

In this recipe, we will use a producer Amazon Redshift RA3 provisioned cluster, with a sample dataset that will be shared with the consumer serverless endpoint.

Getting ready

To complete this recipe, you will need the following:

- An IAM user with access to Amazon Redshift.
- Two separate data warehouses, one with a two-node Amazon Redshift `ra3.xlplus` provisioned cluster and one that is Amazon Redshift serverless deployed in the AWS Region eu-west-1 in `account1`.

  The first provisioned cluster should be deployed with the retail sample data from `Chapter03` (`https://github.com/PacktPublishing/Amazon-Redshift-Cookbook-2E/blob/main/Chapter03/Ssb_Table_Ddl.sql`). This cluster will be called the **producer Amazon Redshift provisioned cluster** where data will be shared from (outbound). Note down the namespace of this cluster, which can be found by running the command `SELECT current_namespace`. Let's say this cluster namespace value is `[Your_Redshift_Producer_Namespace]`.

  The second serverless endpoint can be an empty data warehouse. This data warehouse will be called the **consumer Amazon Redshift data warehouse** from where data will be consumed (inbound). Note down the namespace of this data warehouse, which can be found by running the command `SELECT current_namespace`. Let's say this data warehouse namespace value is `[Your_Redshift_Consumer_Namespace]`.

- Access to any SQL interface, such as a SQL client or Amazon Redshift Query Editor V2.

How to do it...

1. Connect to the Amazon Redshift producer provisioned cluster using a client tool such as SQL Workbench or Query Editor V2.
2. Execute the following analytical query to verify the sample dataset:

```
SELECT DATE_TRUNC('month',l_shipdate),
       SUM(l_quantity) AS quantity
FROM lineitem
WHERE l_shipdate BETWEEN '1992-01-01' AND '1992-06-30'
GROUP BY DATE_TRUNC('month',l_shipdate);

--Sample output dataset
     date_trunc      |    quantity
---------------------+----------------
 1992-05-01 00:00:00 | 196639390.0000
 1992-06-01 00:00:00 | 190360957.0000
 1992-03-01 00:00:00 | 122122161.0000
 1992-02-01 00:00:00 |  68482319.0000
 1992-04-01 00:00:00 | 166017166.0000
 1992-01-01 00:00:00 |  24426745.0000
```

3. Create a data share and add the `lineitem` table so that it can be shared with the consumer serverless endpoint using the following command, replacing `[Your_Redshift_Consumer_Namespace]` with `consumer cluster namespace`:

```
CREATE DATASHARE SSBDataShare;
ALTER DATASHARE SSBDataShare ADD TABLE lineitem;
GRANT USAGE ON DATASHARE SSBDataShare TO NAMESPACE ' [Your_Redshift_
Consumer_Namespace]';
```

4. Execute the following command to verify that data sharing is available:

```
SHOW DATASHARES;
```

Here's the expected output:

```
owner_account,owner_namespace,sharename,shareowner,share_
type,createdate,publicaccess
123456789012,redshift-cluster-data-share-1,ssbdatashare,100,outbou
nd,2021-02-26 19:03:16.0,false
```

5. Connect to the Amazon Redshift consumer serverless endpoint using a client tool such as SQL Workbench or Amazon Redshift Query Editor V2. Run the following command:

```
DESC DATASHARE ssbdatashare OF NAMESPACE [Your_Redshift_Producer_
Namespace];
```

Here's the expected output:

```
producer_account  |          producer_namespace          | share_
type | share_name | object_type |           object_name
------------------+--------------------------------------+---------
---+------------+-------------+--------------------------------
 123456789012      | [Your_Redshift_Producer_Namespace]| INBOUND
| ssbdatashare | table       | public.lineitem
```

6. Create local databases that reference the data shares using the following command:

```
CREATE DATABASE ssb_db FROM DATASHARE ssbdatashare OF NAMESPACE
[Your_Redshift_Producer_Namespace];
```

7. Create an external schema that references the data shares database, `ssb_db`, by executing the following:

```
CREATE EXTERNAL SCHEMA ssb_schema FROM REDSHIFT DATABASE 'ssb_db'
SCHEMA 'public';
```

8. Verify the data share access for the `linetime` table using full qualification, as follows:

```
SELECT DATE_TRUNC('month',l_shipdate),
       SUM(l_quantity) AS quantity
FROM ssb_db.public.lineitem
WHERE l_shipdate BETWEEN '1992-01-01' AND '1992-06-30'
GROUP BY DATE_TRUNC('month',l_shipdate);
```

Here's the sample dataset:

```
     date_trunc      |    quantity
---------------------+----------------
 1992-05-01 00:00:00 | 196639390.0000
 1992-06-01 00:00:00 | 190360957.0000
 1992-03-01 00:00:00 | 122122161.0000
 1992-02-01 00:00:00 |  68482319.0000
```

```
1992-04-01 00:00:00 | 166017166.0000
1992-01-01 00:00:00 |  24426745.0000
```

As you will notice, in the preceding step, the data that is shared by the producer provisioned cluster is available for querying in the consumer serverless endpoint.

How it works...

With Amazon Redshift, you can share data at different levels. These levels include databases, schemas, tables, views (including regular, late-binding, and materialized views), data lake tables, and SQL **user-defined functions (UDFs)**. You can create multiple data shares for a given database. A data share can contain objects from multiple schemas in the database on which sharing is created.

By having this flexibility in sharing data, you get fine-grained access control. You can tailor this control for different users and businesses that need access to Amazon Redshift data. Amazon Redshift provides transactional consistency on all producer and consumer clusters and shares up-to-date and consistent views of the data with all consumers. You can also use `SVV_DATASHARES`, `SVV_DATASHARE_CONSUMERS`, and `SVV_DATASHARE_OBJECTS` to view the data shares, the objects within the data share, and the data share consumers.

See also...

For more details on cross-account data sharing across AWS accounts, refer to `https://docs.aws.amazon.com/redshift/latest/dg/across-account.html`.

For more details on data sharing across Regions, refer to `https://docs.aws.amazon.com/redshift/latest/dg/across-region.html`.

Data sharing write access across multiple Amazon Redshift data warehouses

In this recipe, we will use a producer Amazon Redshift RA3 provisioned cluster, with the sample dataset to be shared with the consumes serverless endpoint. The serverless endpoint will write to the data share. We will also cover granular access control on a data share.

Getting ready

To complete this recipe, you will need the following:

- An IAM user with access to Amazon Redshift.

- Two separate data warehouses, one with a two-node Amazon Redshift `ra3.xlplus` provisioned cluster and Amazon Redshift serverless deployed in the AWS Region eu-west-1 in `account1`.

  The first provisioned cluster will be called the **producer Amazon Redshift provisioned cluster**, where data will be shared from (outbound). Note down the namespace of this cluster, which can be found by running the command `SELECT current_namespace`. Let's say this cluster namespace value is `[Your_Redshift_Producer_Namespace]`.

  The second serverless endpoint can be an empty data warehouse. This data warehouse will be called the **consumer Amazon Redshift data warehouse**, where data will be consumed from (inbound). Note down the namespace of this data warehouse, which can be found by running the command `SELECT current_namespace`. Let's say this data warehouse namespace value is `[Your_Redshift_Consumer_Namespace]`.

- Access to any SQL interface, such as a SQL client or Amazon Redshift Query Editor V2.

How to do it...

1. Connect to the Amazon Redshift producer provisioned cluster using a client tool such as SQL Workbench or Query Editor V2.
2. Run the following command to create a table:

```
CREATE TABLE credit_risk_score (
    customer_id VARCHAR(20) NOT NULL,
    risk_score INTEGER NOT NULL,
    assessment_date DATE NOT NULL DEFAULT GETDATE(),
    risk_category VARCHAR(10) NOT NULL
);
```

3. Create a data share and add the `credit_risk_score` table along with permission to select and insert into table:

```
CREATE DATASHARE riskscore_ds;

GRANT USAGE ON SCHEMA public TO DATASHARE riskscore_ds;

GRANT SELECT ON TABLE public.credit_risk_score  TO DATASHARE
riskscore_ds;
```

```
GRANT INSERT ON TABLE public.credit_risk_score TO DATASHARE
riskscore_ds;

-- You can optionally choose to include new objects to be
automatically shared
ALTER DATASHARE riskscore_ds SET INCLUDENEW = TRUE FOR SCHEMA
public;
```

4. To grant usage on the data share to the serverless consumer namespace, run the following command:

```
GRANT USAGE ON DATASHARE riskscore_ds TO namespace '[Your_Redshift_
Consumer_Namespace]';
```

5. Log on to the serverless namespace and create a database from the data share using `WITH` permissions:

```
CREATE DATABASE riskscore_db WITH PERMISSIONS FROM DATASHARE
riskscore_ds OF NAMESPACE '[Your_Redshift_Producer_Namespace]';
```

6. Let's query and write to the data share using the `use` command:

```
use riskscore_db;
--The first select will return zero records
select * from public.credit_risk_score;
INSERT INTO public.credit_risk_score (customer_id, risk_score,
assessment_date, risk_category)
VALUES ('CUST001', 720, sysdate,'Low');
select * from public.credit_risk_score;
```

How it works...

With Amazon Redshift, you can write through data shares. You can grant usage, create, select, insert, and update access to data share objects. On the consumer side, you can grant permissions to data share objects for consumer users and roles. Data sharing enables you to scale your read and write workloads. The producer is the owner of the objects. When the consumer writes through, the data share commit is handled by the producer.

Data sharing using Amazon DataZone for cross-collaboration and self-service analytics

Amazon DataZone can be effectively utilized to enhance governance for Redshift producers and consumers. It provides a centralized platform for data discovery, access management, and collaboration. For producers, DataZone allows you to catalog your Redshift datasets, add rich business metadata using generative AI, and define access policies, ensuring that data is properly documented and secured. Consumers can easily search for and request access to relevant Redshift data through DataZone's user-friendly interface. The platform's built-in approval workflows and fine-grained access controls help maintain compliance and data security. Additionally, DataZone's data lineage features allow both producers and consumers to track data origins and transformations within Redshift, promoting transparency and trust. By implementing DataZone, organizations can streamline their Redshift data governance processes, fostering a culture of responsible data use while maximizing the value of their data assets.

Getting ready

To complete this recipe, you will need the following:

- An IAM user with access to Amazon Redshift.
- Two Amazon Redshift serverless endpoints, named `sales-wg` and `marketing-wg`.
- Create a sales tables and load data on your Amazon Redshift serverless `sales-wg` using Query Editor V2 with the following script: `https://github.com/PacktPublishing/Amazon-Redshift-Cookbook-2E/blob/main/Chapter11/sales_data.sql`.
- `AWSDataExchangeSubscriberFullAccess` IAM permission associated with your IAM user/role.
- AWS administrator permission to deploy the AWS CloudFormation template to create an Amazon DataZone domain and two projects, `SalesPublisher` and `MarketingConsumer`. Name the stack `cookbook-chapter11-datazone` (`https://github.com/PacktPublishing/Amazon-Redshift-Cookbook-2E/blob/main/Chapter11/datazone_domain_CFN.yaml`).

How to do it...

In this recipe, we will use Amazon Data Zone as the business catalog for sales data in a Redshift serverless and subscribe to this data with an approval workflow:

1. Navigate to the AWS CloudFormation landing page. Select the completed stack **cookbook-chapter11-datazone**.

2. Make note of all key-value pairs. We will reference them in this recipe.

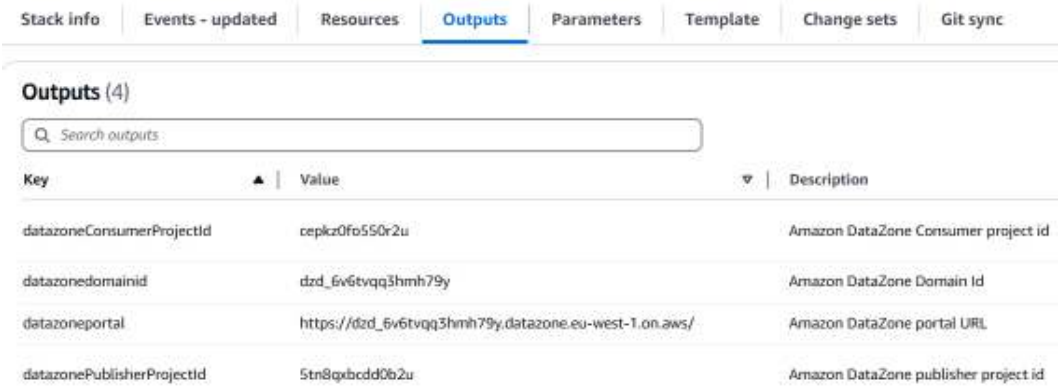


Figure 11.2 – DataZone CloudFormation outputs

3. Next, we will set up AWS Secrets Manager for Amazon Redshift serverless `sales-wg`. Navigate to the AWS Secrets Manager console.
4. Choose store a new secret. Select **Credentials for Amazon Redshift data warehouse**. Enter **User name** and **Password**. Select your serverless `sales-wg` and choose **Next**.

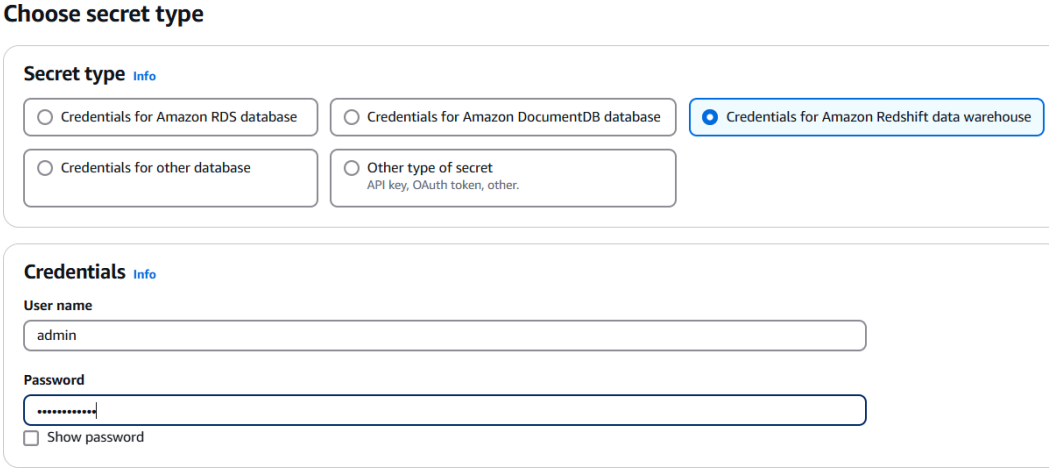


Figure 11.3 – AWS Secrets Manager for sales-wg

5. Enter a name for the secret, such as `AmazonDataZone-saleswgsecrets`. Add the tags `AmazonDataZoneDomain` and `AmazonDataZoneProject` with the values you captured for the keys `datazonedomainid` and `datazonePublisherProjectId` from the CloudFormation stack output in *Step 1*.

Click **Next** and choose store:

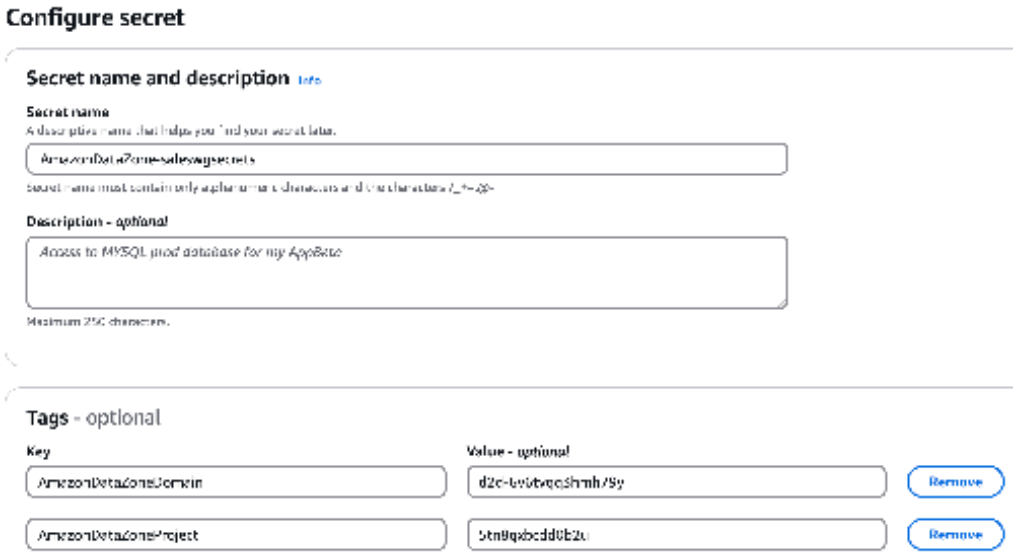


Figure 11.4 – AWS Secrets Manager for sales-wg with tags

6. Repeat *Step 3* and *Step 4* to create a secret for the `marketing-wg` serverless endpoint. Name this secret `AmazonDataZone-marketingwgsecrets`. Add the tags `AmazonDataZoneDomain` and `AmazonDataZoneProject` with the values you captured for the keys `datazonedomainid` and `datazoneConsumerProjectId` from the CloudFormation stack output in *Step 1*.

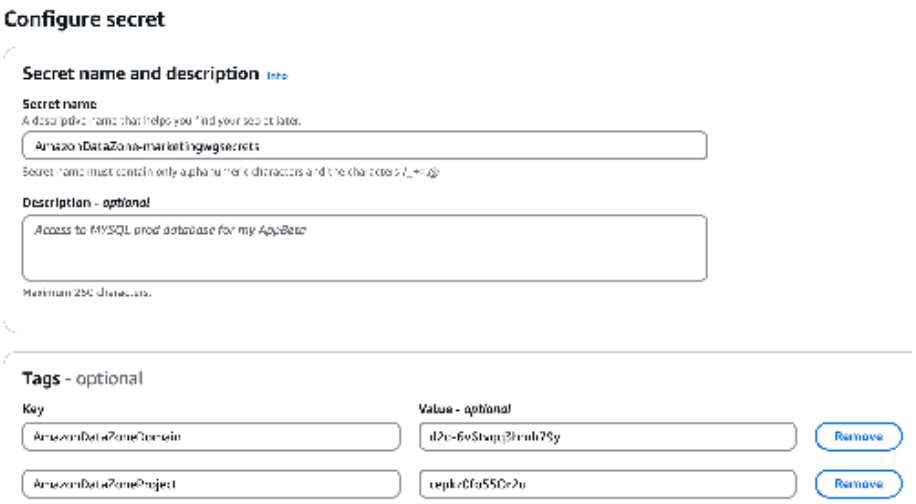


Figure 11.5 – AWS Secrets Manager for marketing-wg with tags

7. Let's now navigate to an Amazon SageMaker domain. Select the domain `cookbookChapter11`.
8. We will now configure the blueprint. A blueprint allows us to configure the resources you want to bring to projects. Select the Blueprint tab. Choose **Default Data Warehouse**.

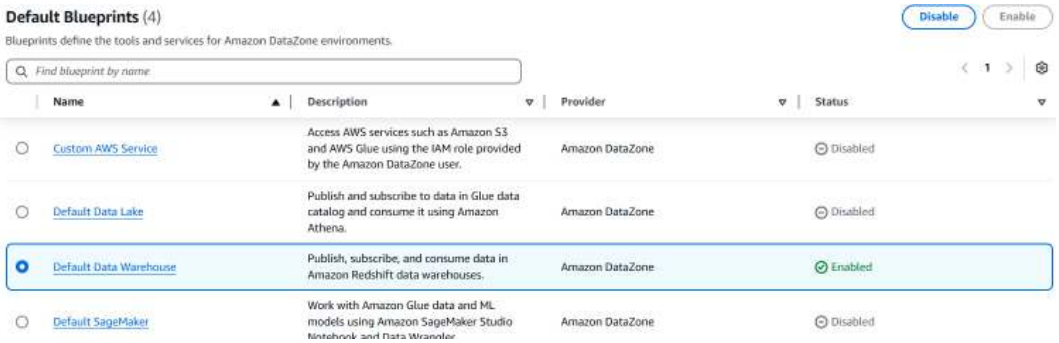


Figure 11.6 – DataZone domain blueprints

9. Let's create a parameter set to allow Amazon DataZone to connect to the `sales-wg` serverless workgroup. Choose **Parameter Set** and select **Create parameter set.** Enter the name `sales-redshift-serverless`:

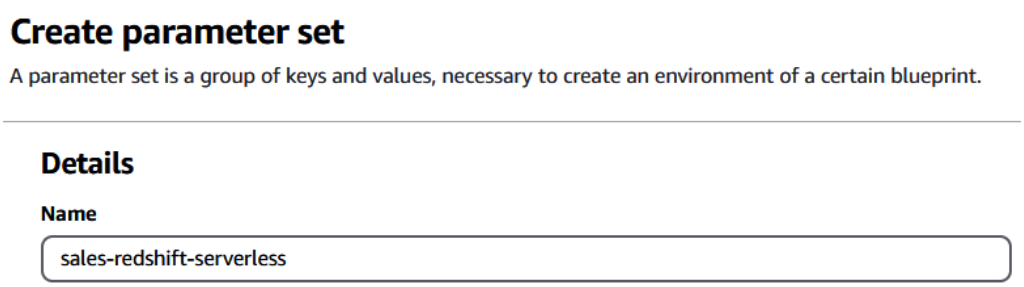


Figure 11.7 – Create parameter set

10. Select the Region your serverless endpoint is in. Choose **Amazon Redshift Serverless**. For **AWS Secret**, it will present you with a dropdown. Select your secret. It will populate with an ARN. Next, select your **Redshift Serverless workgroup** name. Enter the name of the database you loaded sales data in. Select **Create parameter set**.

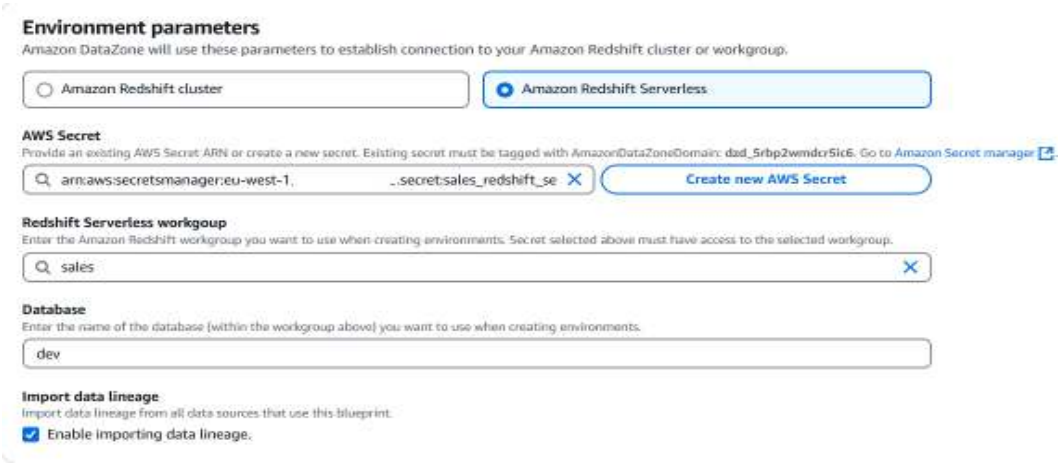


Figure 11.8 – Parameter set for sales-wg

11. Repeat *steps 8* and *9* to create a parameter set for `marketing-wg`. Name the parameter set `marketing-redshift-serverless`.
12. Let's now set up environment profiles. Predefined templates called environment profiles encapsulate technical details such as AWS account, Region, and required resources and tools needed to create an environment for projects. Navigate to the domain and select the portal URL.

Figure 11.9 – DataZone domain portal

13. It will ask you whether you want to create projects. You can skip this step, as we already have projects created using a CloudFormation template.

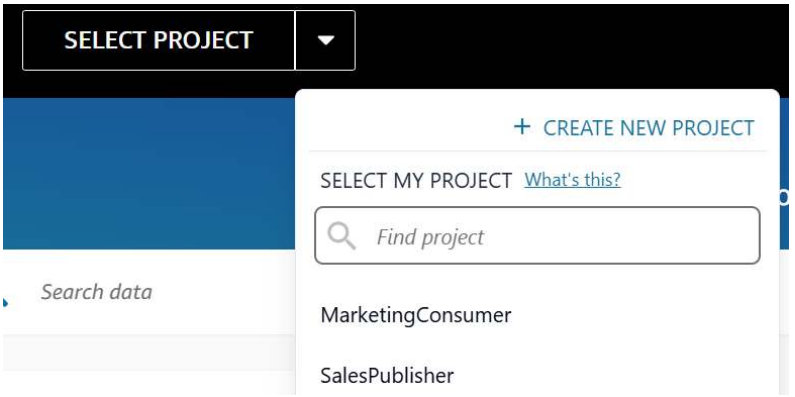


Figure 11.10 – DataZone domain projects navigation

14. Choose the **SalesPublisher** project. In the following steps, we will create an environment profile, an environment, and a data source.
15. On the **Environments** page, select **CREATE ENVIRONMENT PROFILE**:

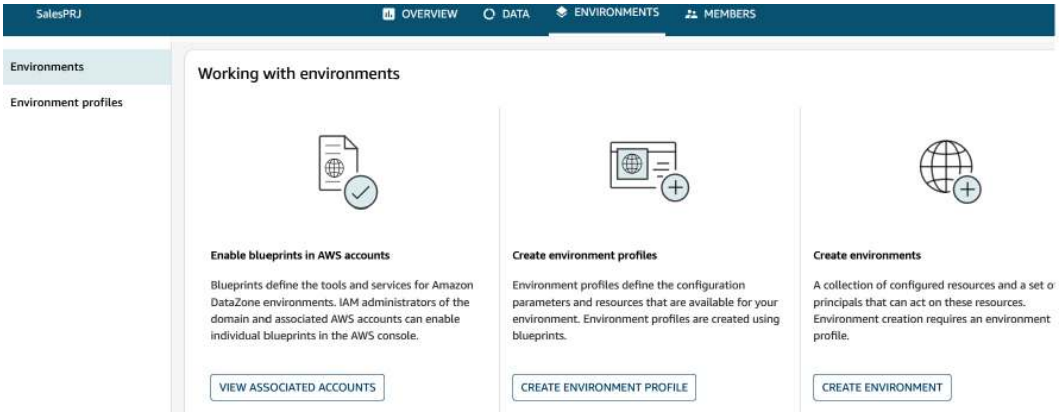


Figure 11.11 – Create an environment profile for the SalesPublisher project

16. Enter the name `SalesEnvironmentProfile`.

17. Set the owner as **SalesPublisher**.

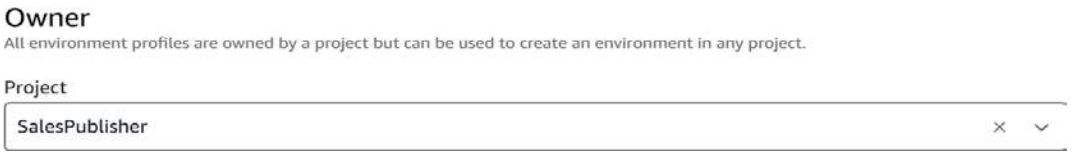


Figure 11.12 – Create the environment profile owner SalesPublisher

18. For **Blueprint**, select **Default Data warehouse**.
19. Then, select **Choose a parameter set**. Select your account AWS account number. Choose the **sales-redshift-serverless** parameter set.

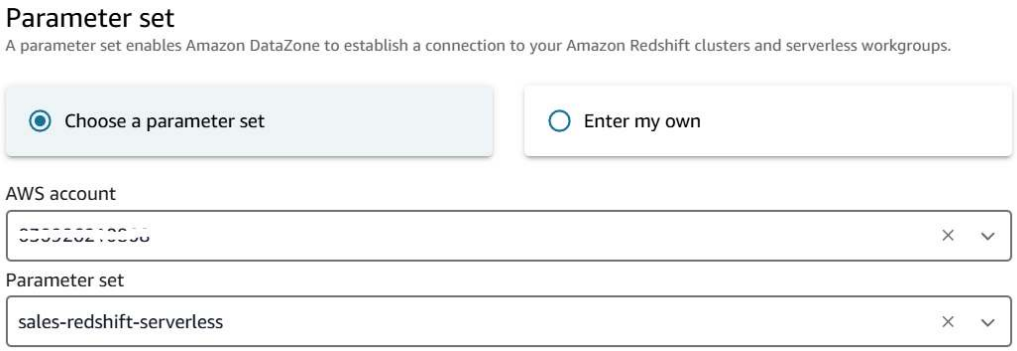


Figure 11.13 – Create the environment profile parameter set for SalesPublisher

20. Select **SalesPublisher** under **Authorized projects**. Choose publish from any schema. Choose create environment profile:

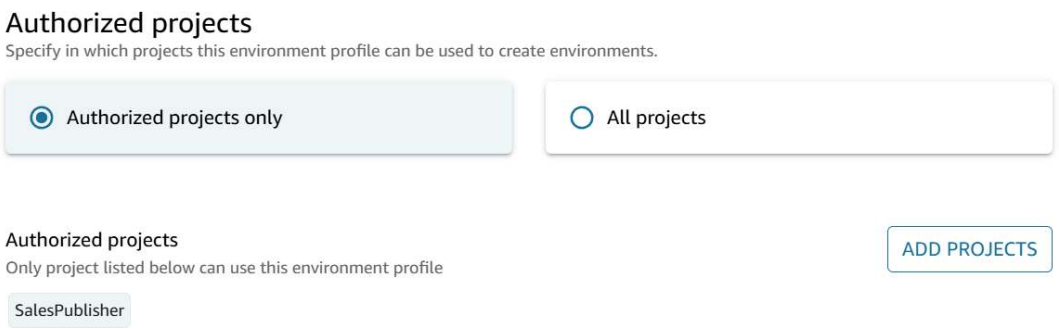


Figure 11.14 – Create environment profile – Authorized projects set to SalesPublisher

21. Let's create an environment from a profile before publishing the dataset. Choose **Create Environment** on the profile page.
22. For **Name**, enter `SalesEnvironment`. Select the profile **SalesEnvironmentProfile**. Select **Create environment**. This will take a few minutes to complete:

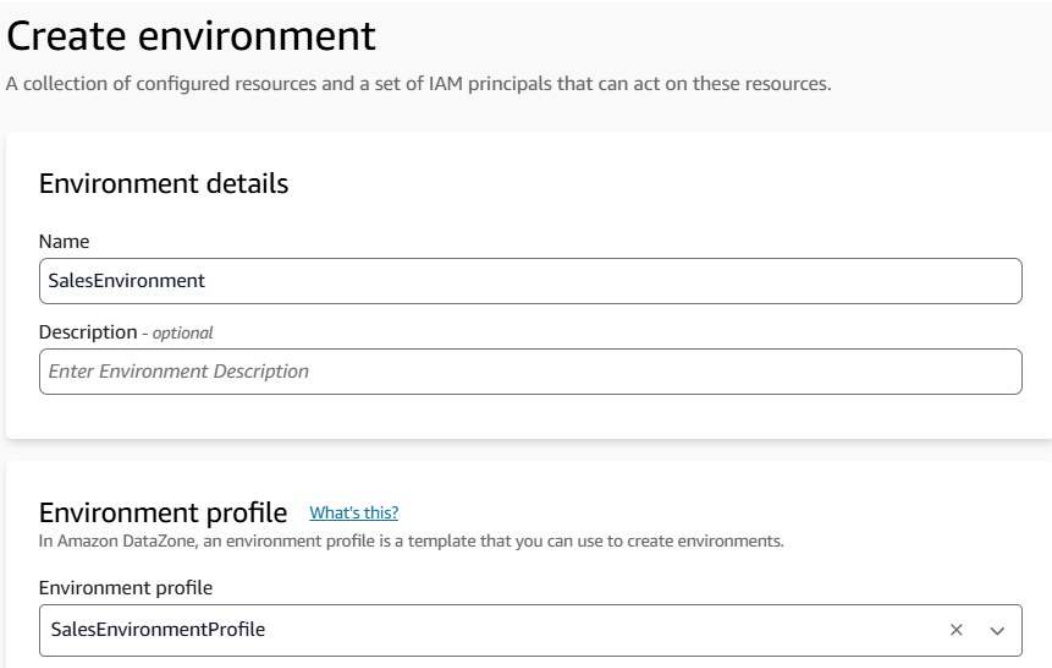


Figure 11.15 – Create the environment SalesEnvironment

The following are the automated environment creation steps:

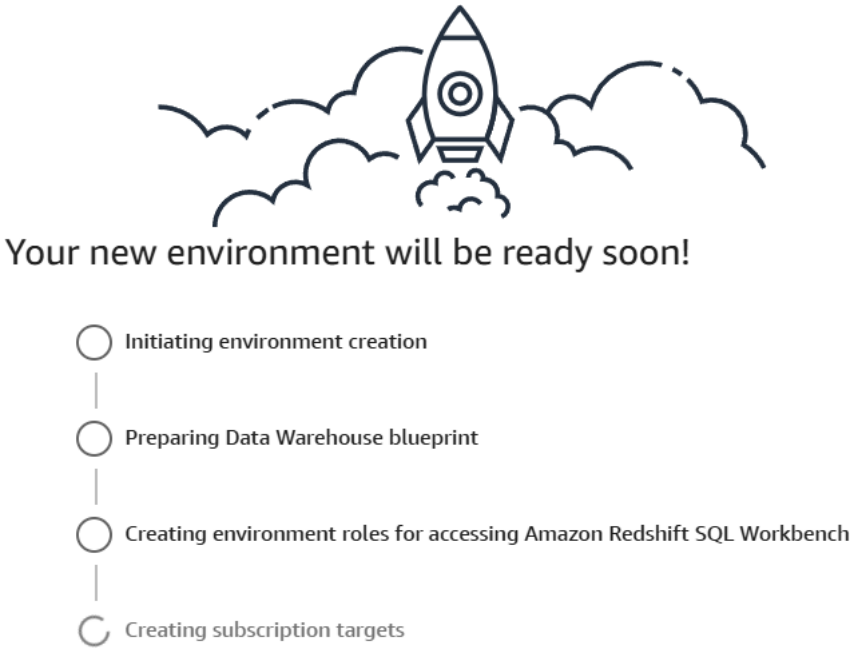


Figure 11.16 – SalesEnvironment creation progress steps

23. Let's create a data source using this environment for `SalesPublisher`. On the **Data** page, select **Create data source**.
24. For **Name**, enter `SalesData`. For **Data source**, select **Amazon Redshift**:

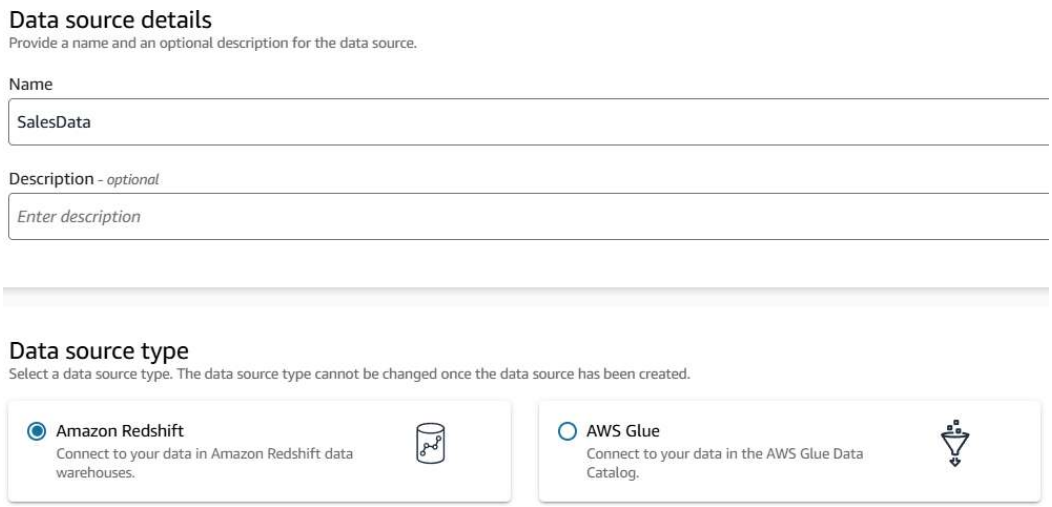


Figure 11.17 – Data source creation – SalesData

25. For the environment, select **SalesEnvironment**. For credentials, click **Use environment's credentials**. Under **Data Selection**, enter the schema name where your data is located, in this case `public`, and then specify a table selection criterion, *.

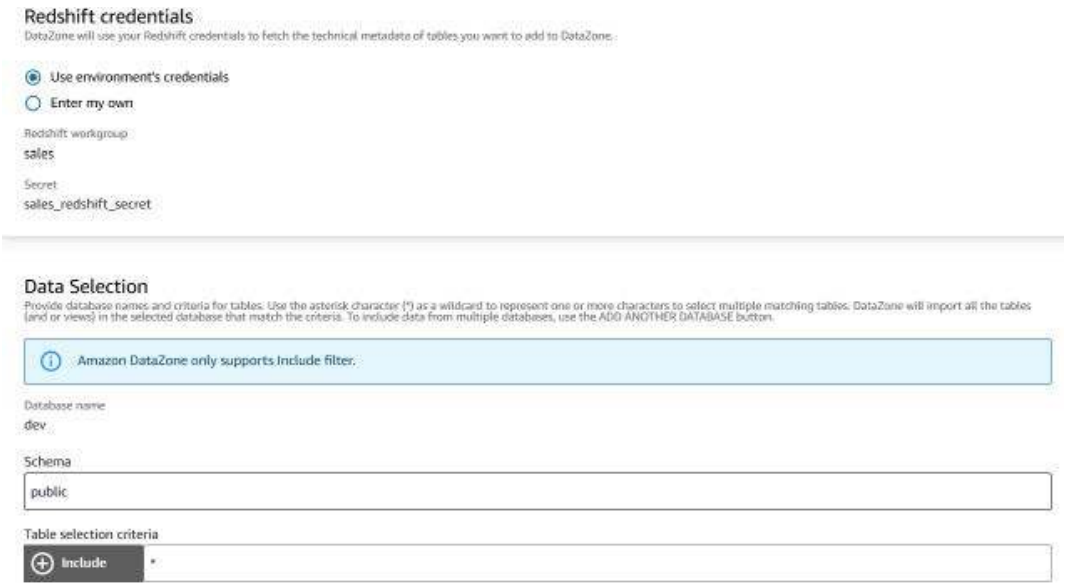


Figure 11.18 – Data source creation setup

26. Click **Next**.
27. Select **Yes** under **Publish assets to the catalog**. Select **Automated business name generation**. This means that Amazon DataZone will automatically generate the business names of the table and columns for that asset using generative AI.

Figure 11.19 – Data source creation configuration for publishing assets

28. Click **Next**. Select **Run on demand**. Click **Next**:

Figure 11.20 – Data source run preference

29. Review the configuration for the data source. Click **Select**.
30. Once the data source is created, click **Run**.

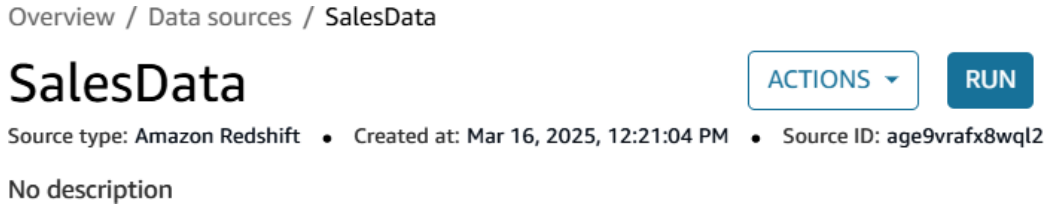


Figure 11.21 – Data source run to refresh assets

31. On completion, you will see a list of assets.

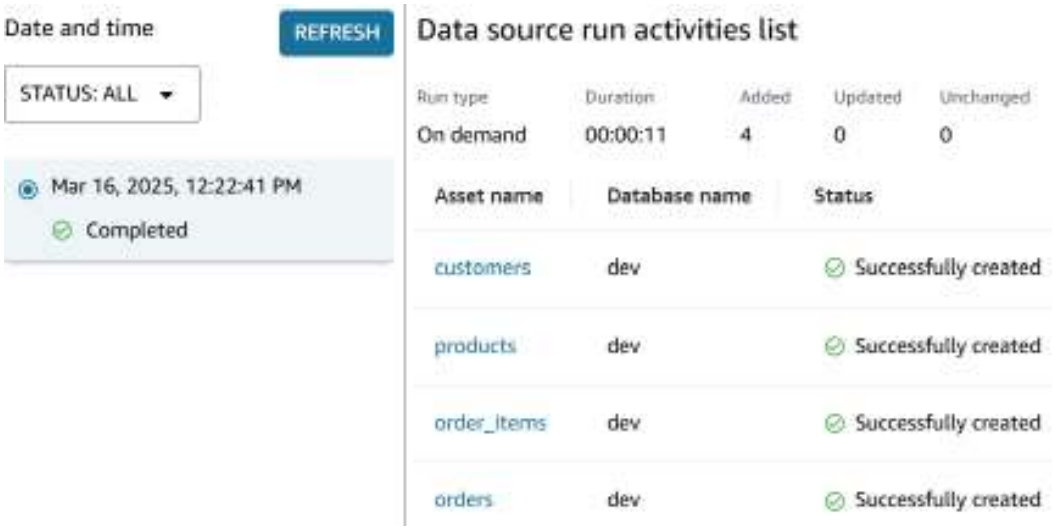


Figure 11.22 – Data source assets list

32. Before publishing, let's generate a business description and terms for these assets using DataZone's generative AI. Choose **Customers** and select **GENERATE DESCRIPTIONS**:

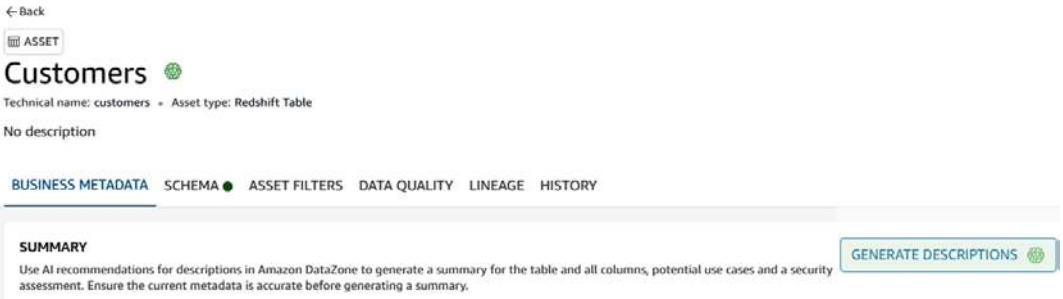


Figure 11.23 – Generate descriptions for the Customers asset

33. Once business metadata has been generated, click **Accept**. You can choose to modify metadata. You can view lineage using an OpenLineage-compatible data lineage visualization (`https://aws.amazon.com/blogs/big-data/amazon-datazone-introduces-openlineage-compatible-data-lineage-visualization-in-preview/`). You can also import data quality scores for assets (`https://docs.aws.amazon.com/datazone/latest/userguide/datazone-data-quality.html`).

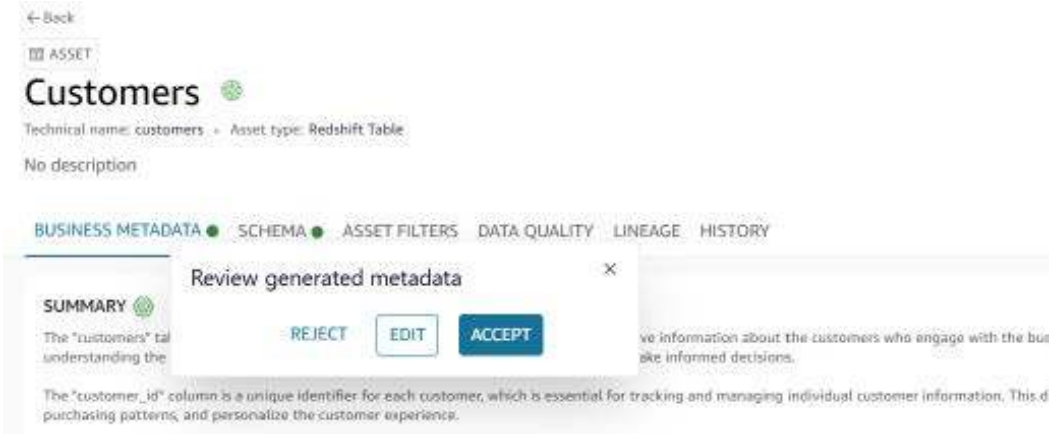


Figure 11.24 – Accept generated business metadata for the Customers asset

34. Navigate to the data quality and lineage tabs.
35. Now publish the **Customers** asset.

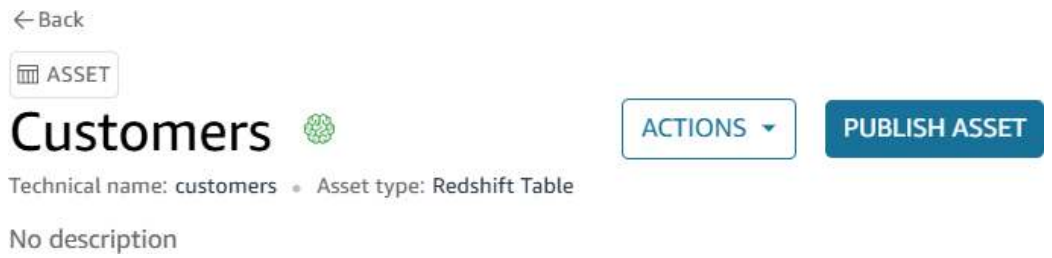


Figure 11.25 – Publish the Customers asset

36. Repeat *steps 31* to *33* for the `products`, `order_items`, and `orders` assets.
37. Navigate to the **SalesData** data source details and validate that **PUBLISHING SETTINGS** is set to **Yes**.

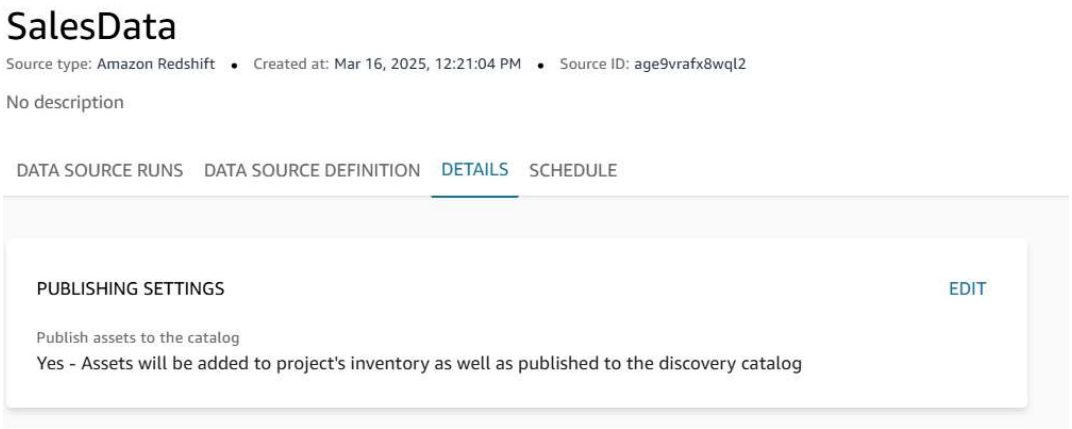


Figure 11.26 – Data source publishing settings

38. Navigate to **Data**, select **Published data**, and then choose **Assets**.

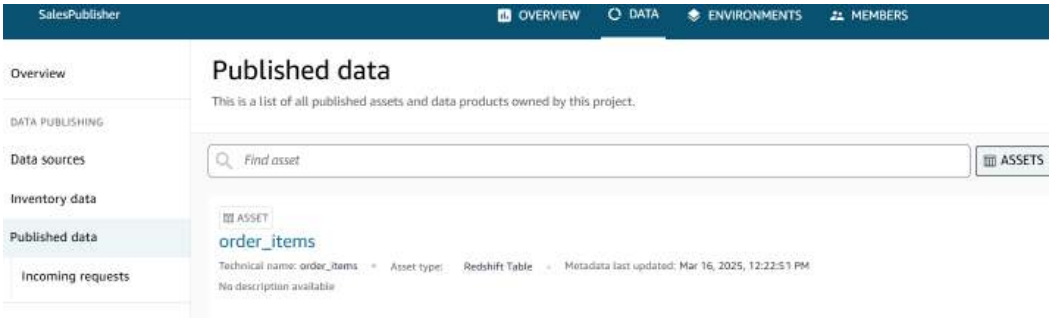


Figure 11.27– Published data view

We will now select **MarketingConsumer** and subscribe to published assets.

39. Let's set up an environment profile and environment for the **MarketingConsumer** project. Choose the **MarketingConsumer** project.
40. Navigate to **Create environment profile**. For **Name**, enter `MarketingEnvironmentProfile`.

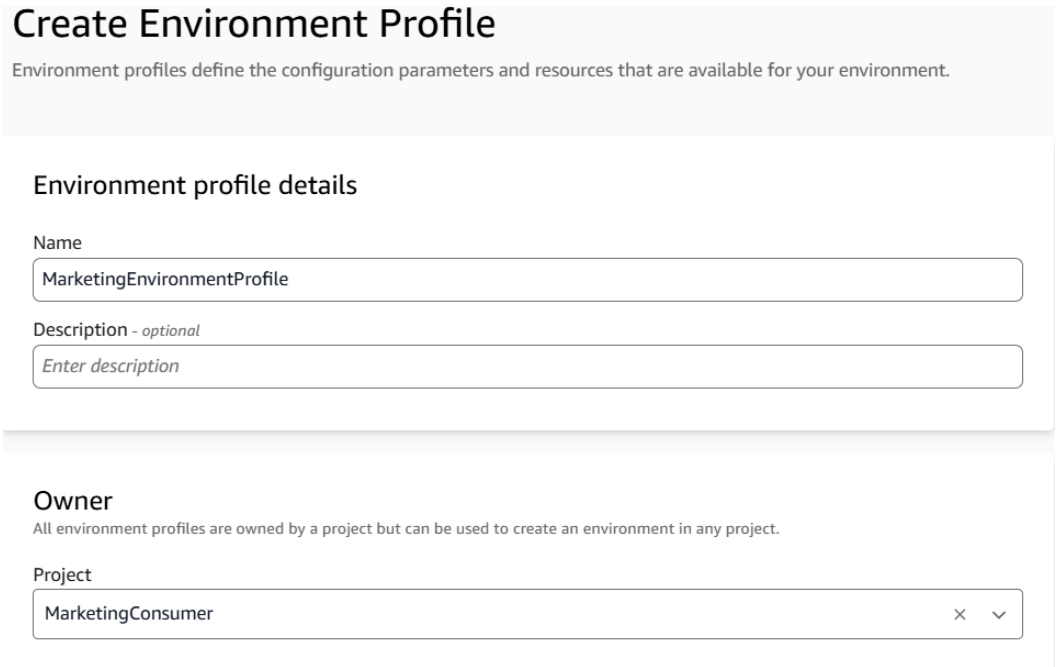


Figure 11.28 – Create a marketing environment profile

41. For blueprint, choose **Default Data warehouse**. For **Parameter set**, select your AWS account and choose the **marketing-redshift-serverless** parameter set.

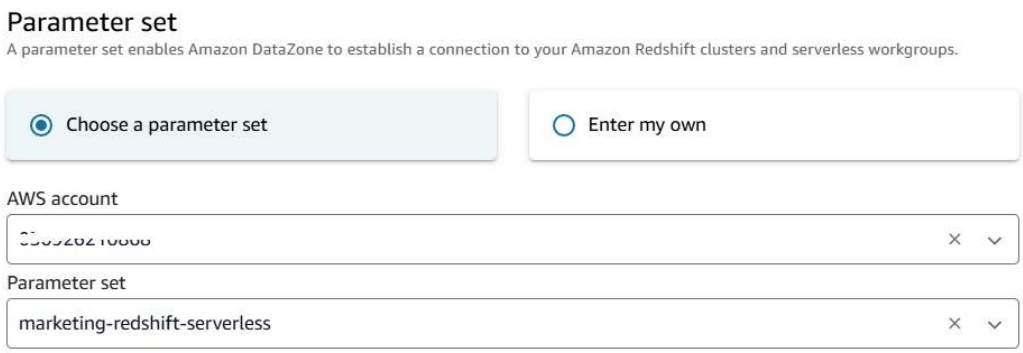


Figure 11.29 – Create a marketing environment profile parameter set

42. Choose **Authorized project only** and select the **MarketingConsumer** project. For **Publishing**, select **Don't allow publishing**. Choose **Create environment profile**.

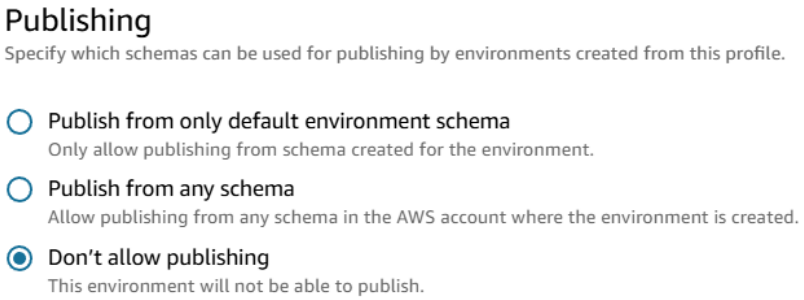


Figure 11.30 – Create the marketing environment profile – Don't allow publishing setting

43. Let's create an environment for **MarketingEnvironmentProfile**. For **Name**, enter `MarketingEnvironment`. For the profile, choose **MarketingEnvironmentProfile**. Click **Create environment**.

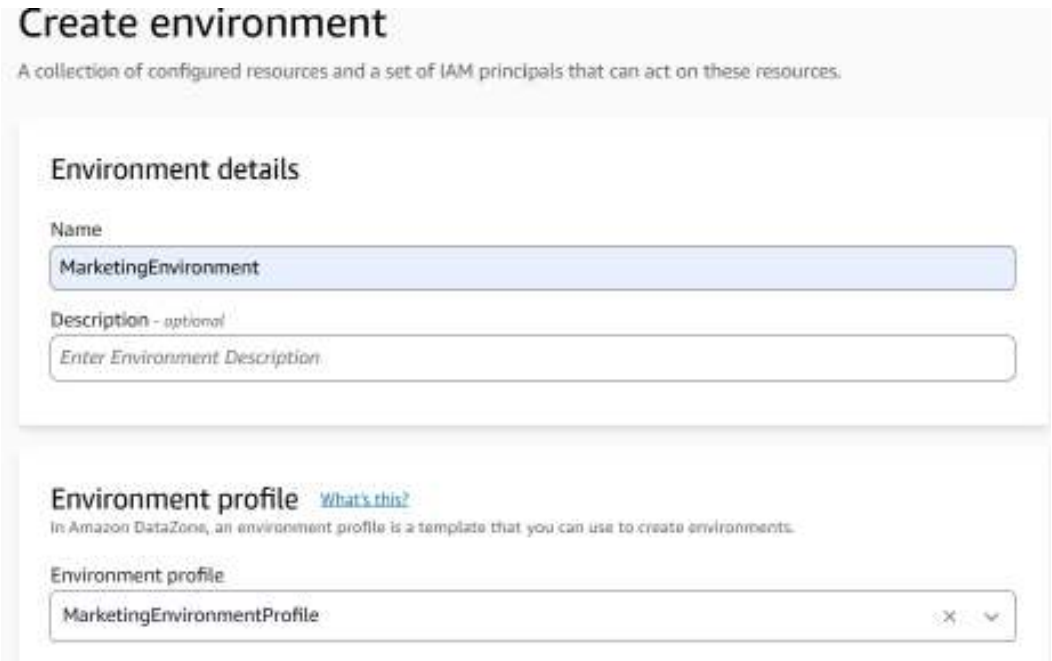


Figure 11.31 – Create a marketing environment

44. Let's submit a request to subscribe to the `Customers` asset from the **SalesPublishing** project. In the search bar, type in `customers`. From the list, select **customers**.

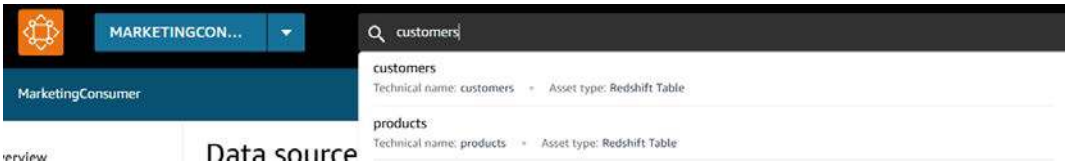


Figure 11.32 – Search for customers data

45. Click **Subscribe**. Provide a reason for the subscription. Enter `For promotional events` as the reason for the request. Click **Request**.

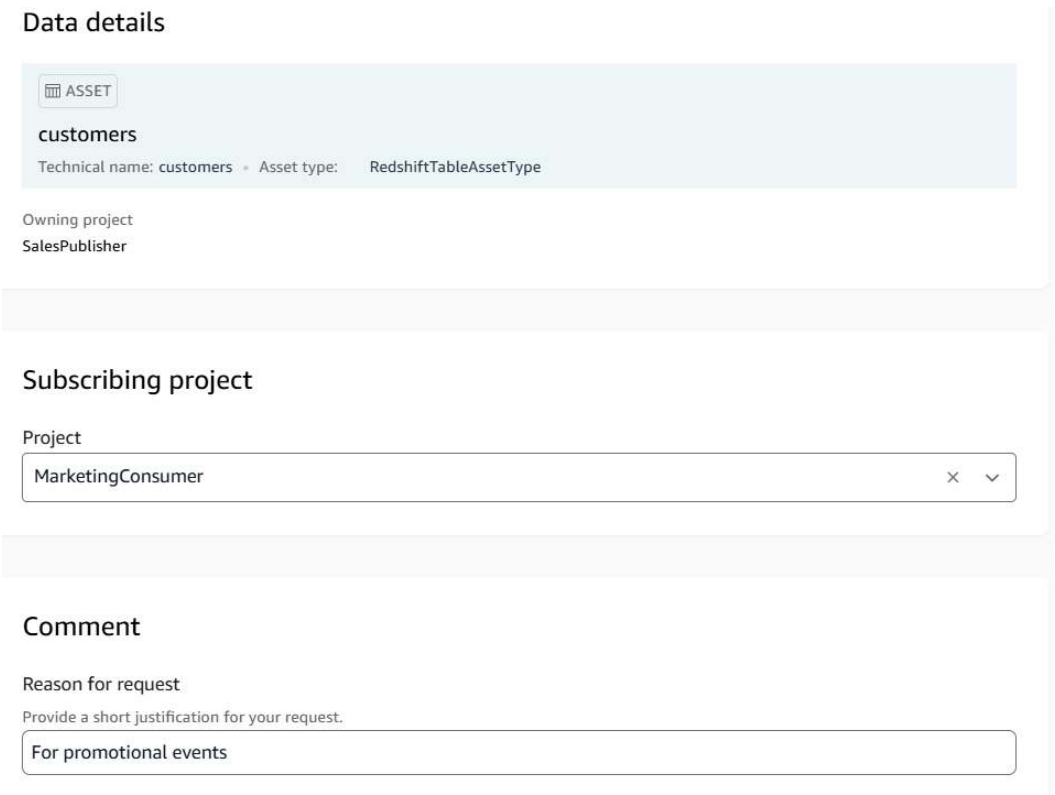


Figure 11.33 – Subscribe to a request for customers data

46. Choose the **SalesPublisher** project. Navigate to **Published data**, then **Incoming requests**. You will see the subscription request for the `Customers` asset.

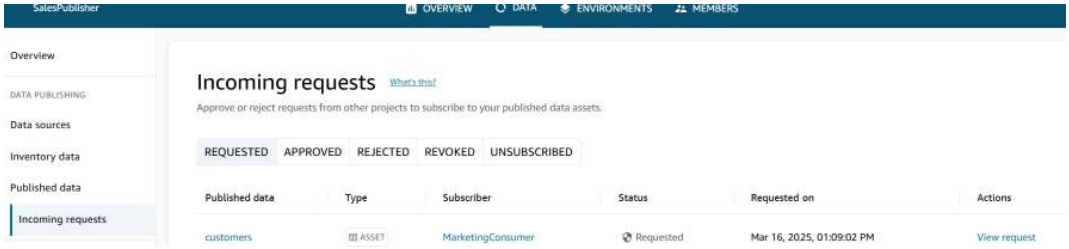


Figure 11.34 – Subscribe request for customers data for approval in the SalesPublisher project

47. Let's view the request and approve it. Select **Full access**. For **Decision comment**, enter `approved`. Choose **Approve**.

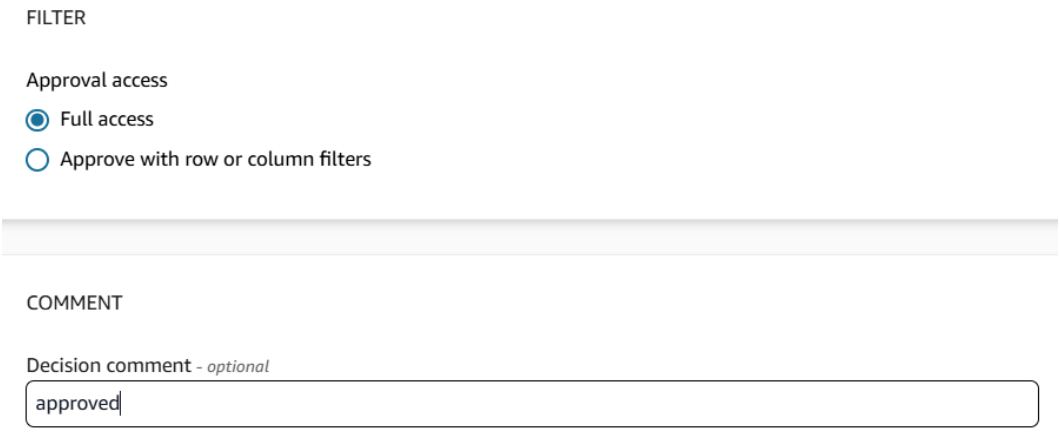


Figure 11.35 – Subscribe request for customers asset approved

48. Navigate to the **MarketingConsumer** project and select **Subscription and review subscribed data**. It will take a few minutes to add the customers assets to the environment.

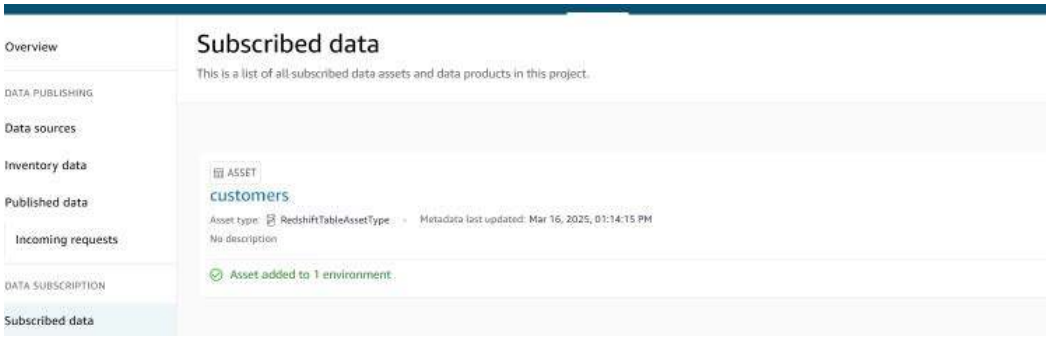


Figure 11.36 – MarketingConsumer project asset added to the environment

49. Let's query the customers assets. Navigate to the environment and choose **MarketingEnvironment**. Click **Query data with Amazon Redshift**. On subscription approval, DataZone creates a data share in the Redshift serverless `marketing-wg`, creates a federated user, and creates a view to access the assets. This abstraction simplifies your collaboration of assets between projects for analytics.

Figure 11.37 – Query customers data using Amazon Redshift

50. Select **OPEN AMAZON REDSHIFT.**

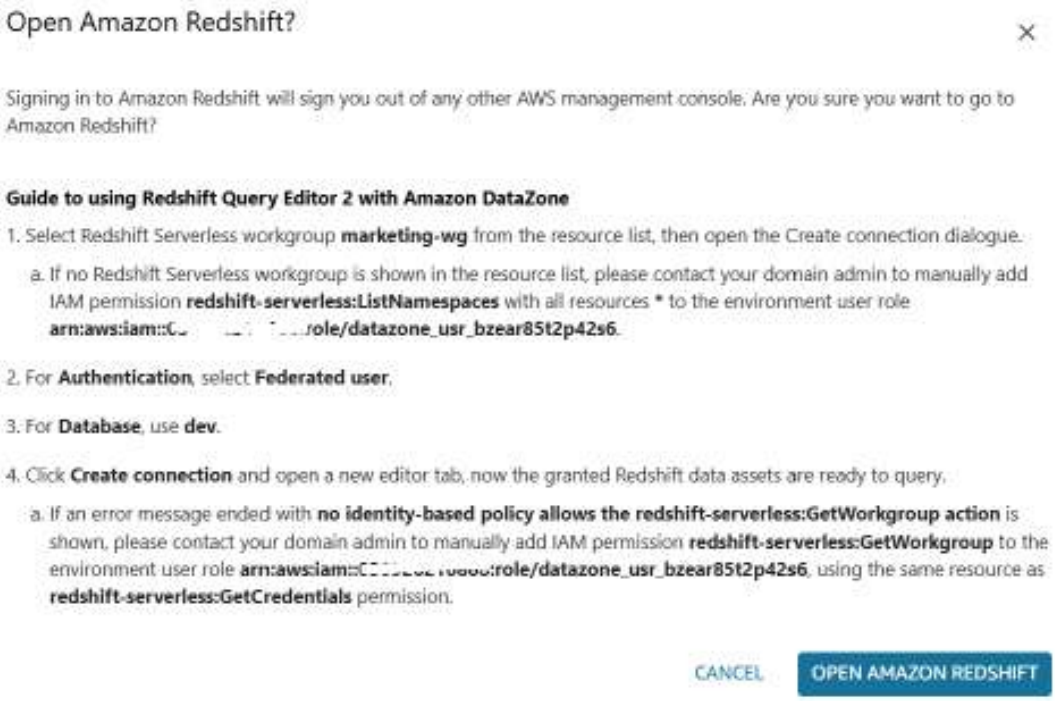


Figure 11.38 – Open Amazon Redshift approval

51. This will bring you to Query Editor V2. Log in to the `marketing-wg` serverless endpoint using the federated user. Expand the data explorer. You will see the **datazone_env_marketingenvironment** schema as well as **customers**.

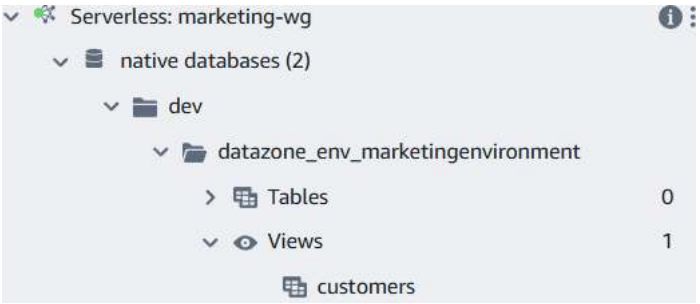


Figure 11.39 – Query Editor V2 data object explorer for marketing-wg

52. In Query Editor V2, run the following query:

```
select * from "datazone_env_marketingenvironment"."customers";
```

customer_id	first_name	last_name	email	phone	address
1	John	Doe	john.doe@email.com	555-0101	123 Main St
2	Jane	Smith	jane.smith@email.com	555-0102	456 Oak Ave
3	Bob	Johnson	bob.j@email.com	555-0103	789 Pine Rd

Figure 11.40 – Query Editor V2 query results for the customers asset

How it works...

In the producer account, data assets stored across Amazon S3, Redshift, RDS, and third-party sources are registered in the Amazon DataZone catalog within the central governance account. Amazon DataZone's business catalog serves as a central governance hub in a data mesh architecture, enabling domain-oriented teams to autonomously manage and discover data assets. The catalog leverages generative AI capabilities to automatically generate rich metadata, business context, and use case descriptions for data assets, making them easily discoverable across organizational boundaries. This central account serves as the control center, hosting the DataZone domain and data portal, while connecting producer and consumer AWS accounts through DataZone domain associations. End users access the DataZone portal using IAM credentials or SSO integration through IAM Identity Center, where they can explore asset information including data quality metrics and both business and technical metadata. The architecture implements a self-service model where users request access through DataZone's subscription feature, subject to asset owner approval. Once granted access, consumers can leverage the data in their accounts for various purposes: developing AI/ML models in SageMaker, performing analytics with Athena, conducting data warehousing operations in Redshift, or creating visualizations using QuickSight.

Figure 11.41 – Amazon DataZone data mesh pattern

Data sharing using AWS Data Exchange for monetization and subscribing to third-party data

AWS Data Exchange enables secure data sharing of Amazon Redshift data through its integration with Redshift data shares. Organizations can publish their Redshift data as licensed data products, which subscribers can directly query from their own Redshift clusters without copying or moving the underlying data. To share data, providers first create a data share in their Redshift cluster using the `CREATE DATASHARE` command, add specific schemas, tables, or views, and then publish it as a data product on AWS Data Exchange. Publishers can choose to publicly or privately share products. Subscribers can discover these data products through the AWS Data Exchange catalog, and once subscribed, the data appears as a database in their Redshift cluster through the use of a managed AWS Data Exchange data share. This streamlines data monetization and consumption while maintaining security.

Getting ready

To complete this recipe, you will need the following:

- An IAM user with access to Amazon Redshift
- Amazon Redshift serverless in the us-east-1 Region
- The `AWSDataExchangeSubscriberFullAccess` IAM permission associated with your IAM user/role

How to do it...

In this recipe, we will subscribe to the AWS Data Exchange data product **Worldwide Event Attendance**. Then, we will use Amazon Redshift serverless to access **Worldwide Event Attendance**:

1. Navigate to the AWS Data Exchange console.
2. From the left navigation pane, choose **Product catalog** under **Subscribed with AWS Marketplace**.
3. Search for **Worldwide Event**.
4. Choose **Worldwide Event Attendance (Test Product)**. Select **Continue to subscribe**. This product costs zero dollars.

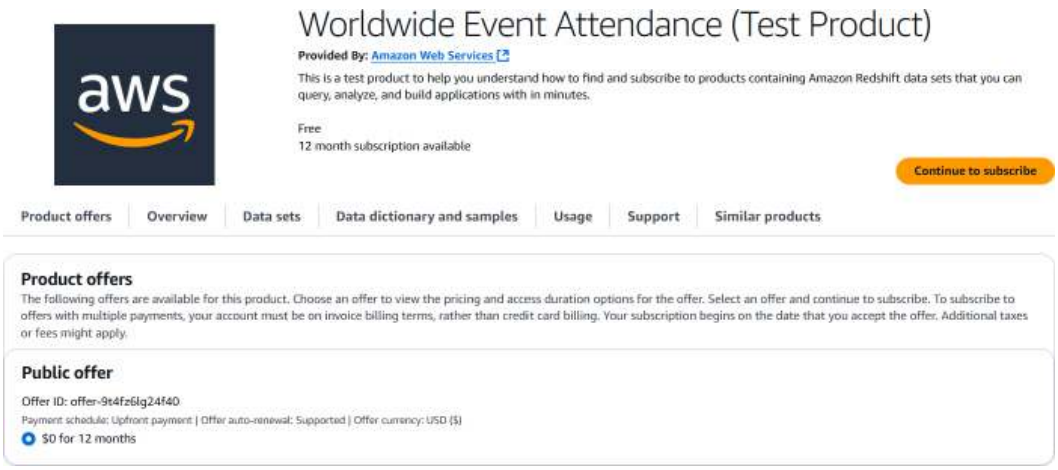


Figure 11.42 – Amazon Data Exchange test product subscription

5. Review the terms and conditions. Click **Subscribe**. You can choose **No** for **Offer auto-renewal**.

Figure 11.43 – Amazon Data Exchange test product subscribe

6. Once the subscription is successful, you will be able to see the product under **Subscribed with AWS Marketplace | Active subscriptions**.
7. Navigate to the Redshift serverless namespace on the console.
8. Click the **Datashares** tab and create a connection for your Redshift user to the dev database.

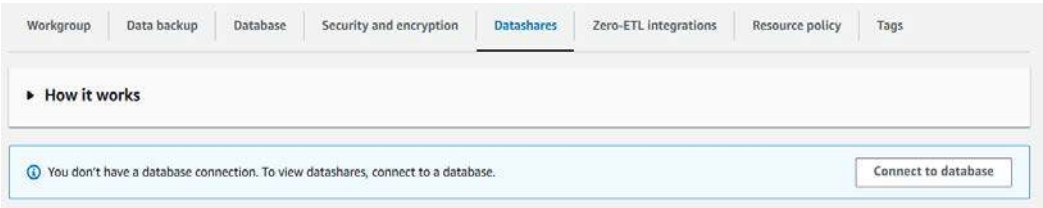


Figure 11.44 – Amazon Redshift console | Datashares | Connect to database

9. Navigate to **Subscriptions to AWS Data Exchange datashares**.

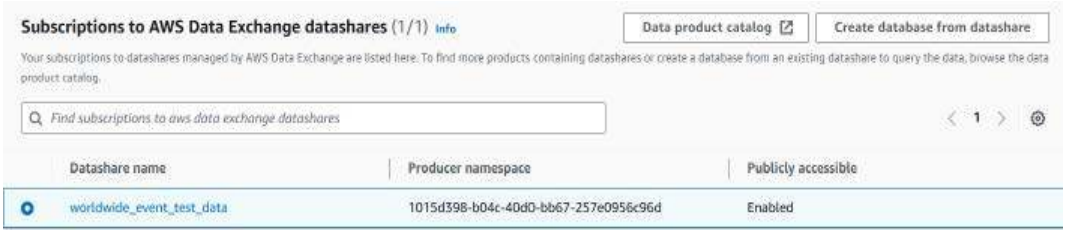


Figure 11.45 – Amazon Redshift data shares from subscriptions

10. Choose **worldwide_event_test_data** and select **Create database from datashare**. Enter the `worldwide_event_test_db` database name. Select **Without Permissions**. Click **Create**.

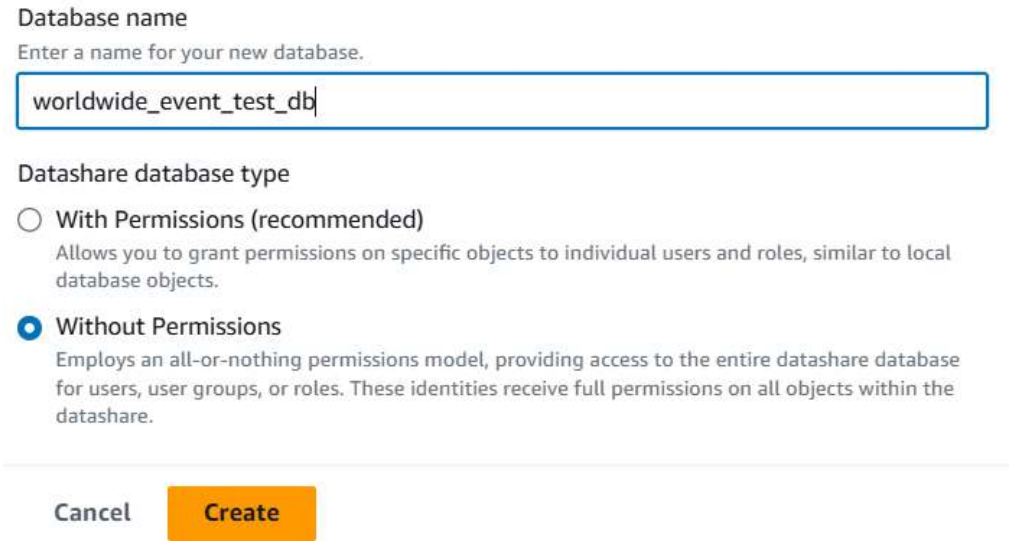


Figure 11.46 – Amazon Redshift create database from the subscriptions data share

11. Log on to Query Editor V2 and connect to your serverless workgroup.
12. Expand the data object explorer on the left. Expand **external_databases**. Further expand **worldwide_event_test_db**.

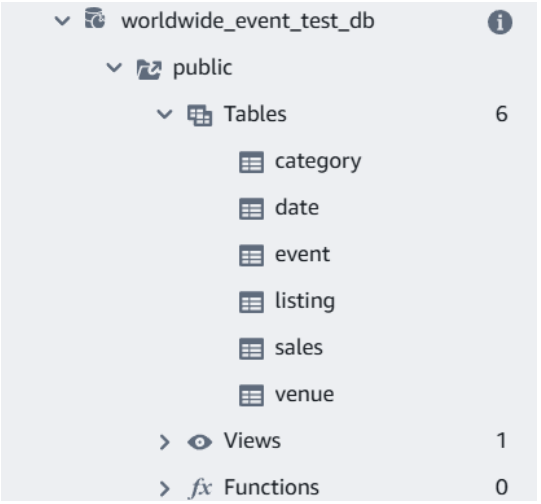


Figure 11.47 – Amazon Redshift Query Editor V2 data explorer view listing data share objects

13. Let's analyze the data shared with the third-party dataset using the following SQL. Run this query:

```
SELECT event.eventname, event.starttime, venue.venuename, venue.
venuecity, venue.venuestate,venueseats
FROM "worldwide_event_test_db"."public"."event" as event
JOIN "worldwide_event_test_db"."public"."venue" as venue
on event.venueid=venue.venueid
WHERE venue.venueseats > 0;
```

The following is the output of the preceding query:

eventname	starttime	venuename	venuecity	venuestate	venueseats
Joe Cocker	2008-01-12 14:00:00	McAfee Coliseum	Oakland	CA	63026
David Byrne	2008-01-13 19:00:00	Safeco Field	Seattle	WA	47116
Jamboree in the Hills	2008-01-27 15:00:00	Reliant Stadium	Houston	TX	72000
Ben Harper	2008-01-28 15:00:00	Arrowhead Stadium	Kansas City	MO	79451
Daughtry	2008-02-01 20:00:00	Miller Park	Milwaukee	WI	42200
Missy Higgins	2008-02-02 15:00:00	Great American Ball Park	Cincinnati	OH	42059
Squeeze	2008-02-04 14:00:00	M&T Bank Stadium	Baltimore	MD	70107
Wyclef Jean	2008-02-07 19:30:00	LP Field	Nashville	TN	68804

Figure 11.48 – Amazon Redshift Query Editor V2 data share query output

How it works...

The process consists of two main workflows – the provider and subscriber paths. On the provider side, the data analyst first creates an AWS Data Exchange managed data share and loads data into their Redshift database. They then create a dataset from this data share, add it to a product, and can create private or public offers for specific customers. The provider can also set up pipelines to refresh new data to their Redshift cluster automatically. On the subscriber side, once they subscribe to the product through AWS Data Exchange, a database is automatically created from the data share in their environment. The subscriber's data analyst can then launch a serverless workspace and run queries directly against the shared data through their own Redshift instance.

This serverless architecture enables seamless data sharing between providers and subscribers while maintaining data security and governance through AWS's managed services. This pattern works for both provisioned and serverless Amazon Redshift.

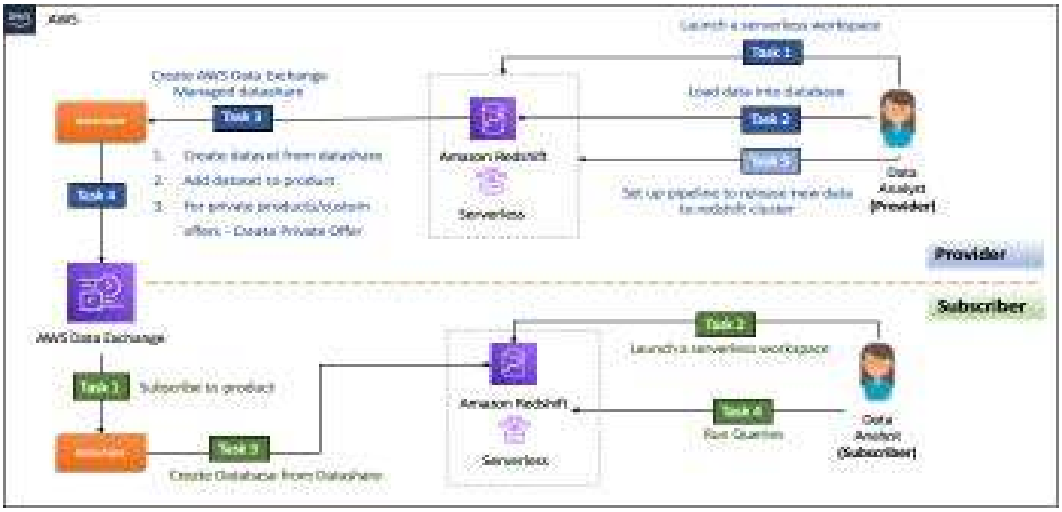


Figure 11.49 – AWS Data Exchange provider and subscriber flow

12

Generative AI and ML with Amazon Redshift

In this chapter, we explore how Amazon Redshift is transforming data analytics with integrated **machine learning** (**ML**) capabilities and generative AI. Redshift ML enables users to create, train, and deploy ML models directly within a Redshift data warehouse, supporting a range of use cases from supervised learning to advanced generative AI models like **large language models** (**LLMs**). You'll discover how Redshift ML can streamline processes such as forecasting and sentiment analysis, and how Amazon Q enhances these capabilities with AI-driven assistants for query authoring and **business intelligence** (**BI**).

Finally, we'll delve into the future potential of generative AI within the Amazon Redshift data warehouse (serverless or provisioned cluster) and its broader impact on data analytics.

The following recipes are covered in this chapter:

- Building SQL queries automatically using Amazon Q generative SQL
- Managing Amazon Redshift ML
- Using LLMs in Amazon Bedrock using SQL statements
- Using LLMs in Amazon SageMaker to jumpstart using SQL statements
- Querying your data with natural language prompts using Amazon Bedrock knowledge bases for Amazon Redshift
- Generative BI with Amazon Q with QuickSight querying an Amazon Redshift dataset

Technical requirements

Here are the technical requirements in order to complete the recipes in this chapter:

- Access to the AWS Management Console.
- An Amazon Redshift data warehouse deployed in the AWS Region eu-west-1.
- Amazon Redshift data warehouse admin user credentials.
- AWS administrator permission to create an IAM role by following *Recipe 3* in *Appendix*. This IAM role will be used for some of the recipes in this chapter.
- Attach an IAM role to the Amazon Redshift data warehouse by following *Recipe 4* in *Appendix*. Make note of the IAM role name; we will refer to it in the recipes with `[Your-Redshift_Role]`.
- AWS administrator permission to deploy the AWS CloudFormation template (`https://github.com/PacktPublishing/Amazon-Redshift-Cookbook-2E/blob/main/Chapter12/chapter_12_CFN.yaml`), which creates two IAM policies and the `quicksight-role` role:
    - An IAM policy attached to the IAM user that will give them access to Amazon Redshift, Amazon RDS, Amazon Kinesis, Amazon Kinesis Data Firehose, Amazon CloudWatch Logs, AWS CloudFormation, AWS Secret Manager, Amazon Cognito, Amazon S3, AWS DMS and AWS Glue
    - An IAM policy attached to the IAM role that will allow the Amazon Redshift data warehouse to access Amazon S3
    - The IAM role `quicksight-role` gives access to QuickSight to create a VPC connection to Amazon Redshift
- Access to any SQL interface, such as a SQL client or Amazon Redshift Query Editor.
- An AWS account number; we will refer to it in the recipes with `[Your-AWS_Account_Id]`.
- An Amazon S3 bucket created in eu-west-1; we will refer to it with `[Your-Amazon_S3_Bucket]`.
- The code files can be found in the GitHub repo: `https://github.com/PacktPublishing/Amazon-Redshift-Cookbook-2E/tree/master/Chapter11`.

Building SQL queries automatically using Amazon Q generative SQL

In this recipe, you'll learn how to leverage Amazon Q, a generative AI-powered SQL assistant in Amazon Redshift Query Editor. Amazon Q simplifies query authoring by transforming natural language prompts into SQL queries, reducing the complexity of writing SQL manually.

Whether you're a data engineer, analyst, or non-technical user, Amazon Q generative SQL enables you to interact with your data more intuitively, speeding up analysis and improving efficiency. By the end of this recipe, you'll understand how to use natural language commands to generate SQL queries that can be run directly using Amazon Redshift for insights.

Getting ready

To complete this recipe, you will need:

- An Amazon Redshift data warehouse deployed in the AWS Region eu-west-1, with the retail dataset from *Chapter 3*, in the *Loading and unloading data* recipe
- Amazon Redshift data warehouse master user credentials
- Access to Amazon Redshift Query Editor V2

How to do it...

1. Navigate to the Amazon Redshift console and then to Query Editor V2 (`https://console.aws.amazon.com/sqlworkbench/home`) and click on the gear icon, and select **Q generative SQL** settings, as shown in the following screenshot:

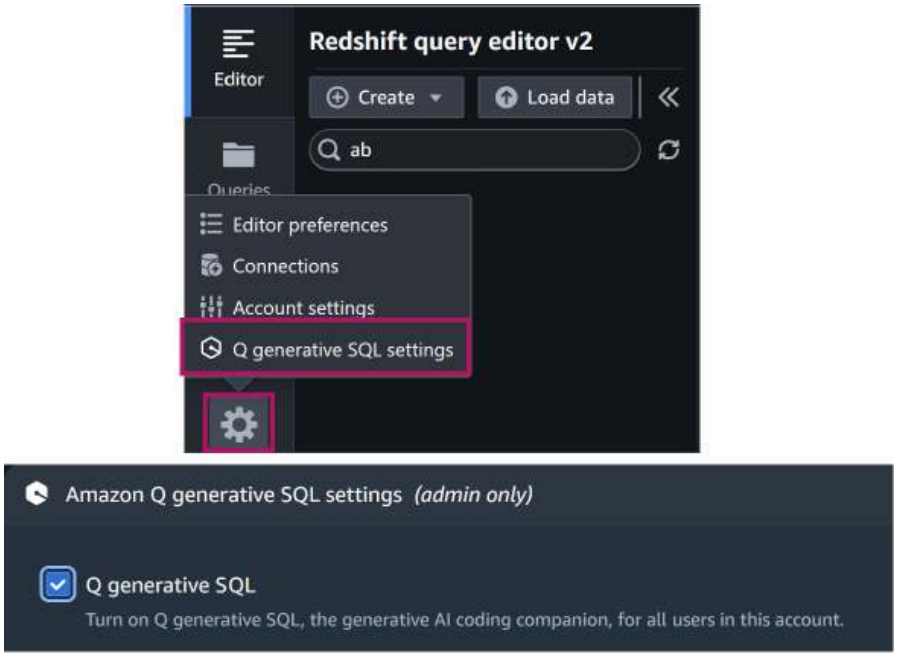


Figure 12.1 – Q generative SQL settings in Query Editor V2

2. Optionally, you can scroll down to the **Custom Context** section. In this section, you can add additional details about the schema in JSON format that generative SQL can leverage to generate more accurate SQL. Click on **Add custom context** and paste the sample JSON from GitHub (`https://github.com/PacktPublishing/Amazon-Redshift-Cookbook-2E/blob/main/Chapter11/SampleCustomContext.json`) in the editor, then click **Save**.
3. Amazon Q generative SQL is available in the notebooks feature of Query Editor V2. Click on the + icon in Query Editor and choose **Notebook**. Then, click on the Amazon Q generative SQL icon, as shown in the following screenshot, to open the Amazon Q generative SQL window, where you can ask questions about the dataset in natural language to generate SQL statements.

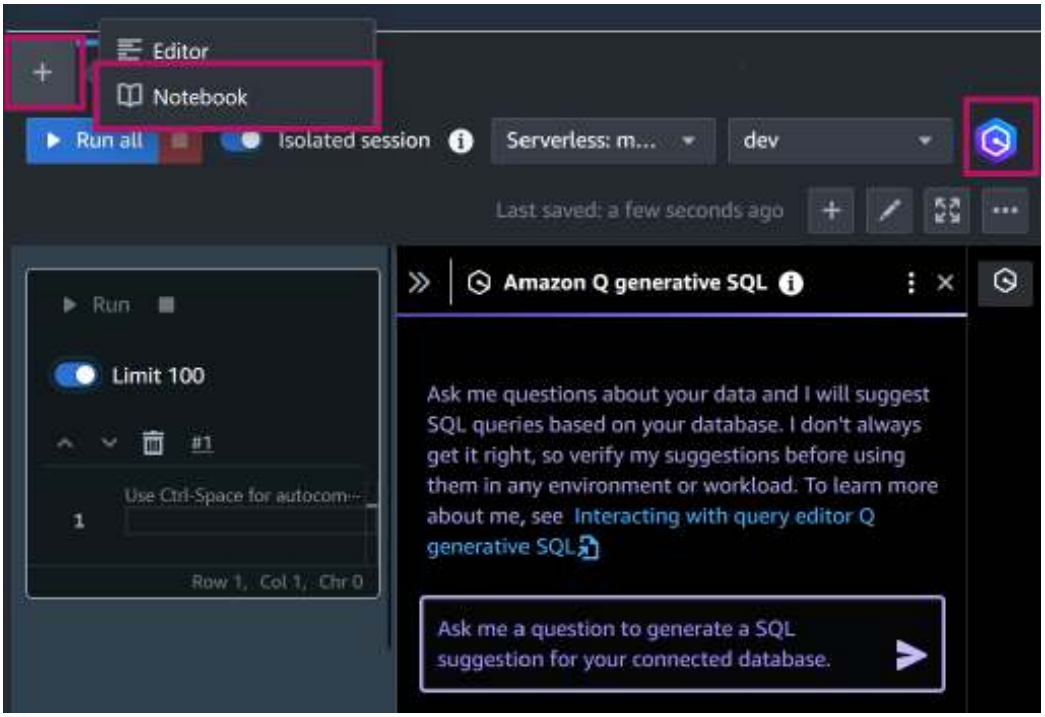


Figure 12.2 – Opening the Amazon Q generative SQL chat window

4. Type a question about the data, for example, `Who are the top 10 high value customers?`, and submit it to Amazon Q. You will receive a SQL statement as output, along with an **Add to notebook** button. When you click on it, the query will be added to the notebook and you can run it to see the results.

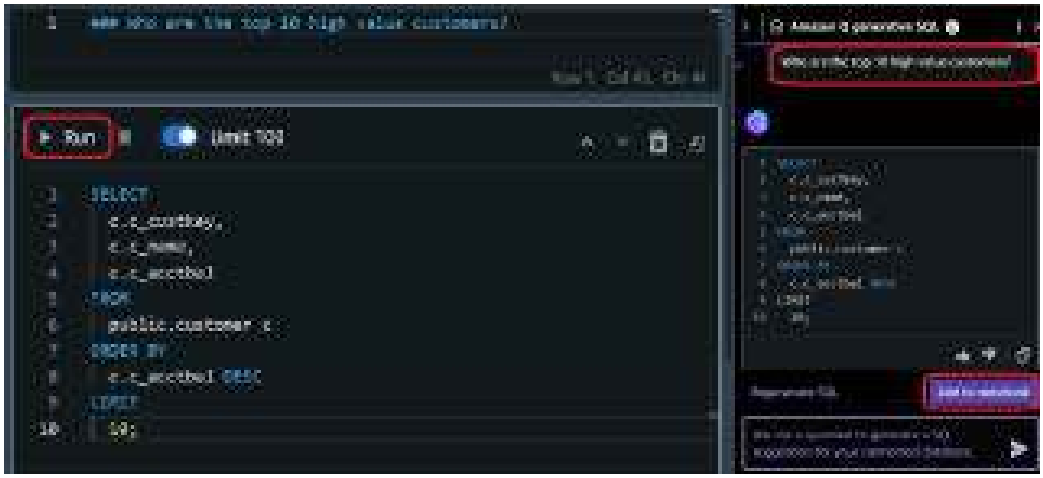


Figure 12.3 – Sample interaction with Q generative SQL

How it works...

Amazon Q generative SQL uses the schema of objects in your database or the custom content you provided as context to the generative AI model. In addition to the schema, it also uses the user query history as context. The generative AI model then generates a SQL statement that can answer your natural language question and presents it to you. You can then add it to a notebook, execute it, and share the results.

Managing Amazon Redshift ML

Amazon Redshift ML enables Amazon Redshift users to create, deploy, and execute ML models using familiar SQL commands. Amazon Redshift has built-in integration with Amazon SageMaker Autopilot, which chooses the best ML algorithm based on your data using automatic algorithm selection. It enables users to run ML algorithms without the need for expert knowledge in ML. On the other hand, ML experts such as data scientists have the flexibility to select algorithms such as XGBoost and specify the hyperparameters and preprocessors. Once the ML model is deployed in Amazon Redshift, you can run the prediction using SQL at scale. This integration completely simplifies the pipeline required to create, train, and deploy the model for prediction.

Amazon Redshift ML allows you to create, deploy, and predict using the data in the data warehouse as follows:

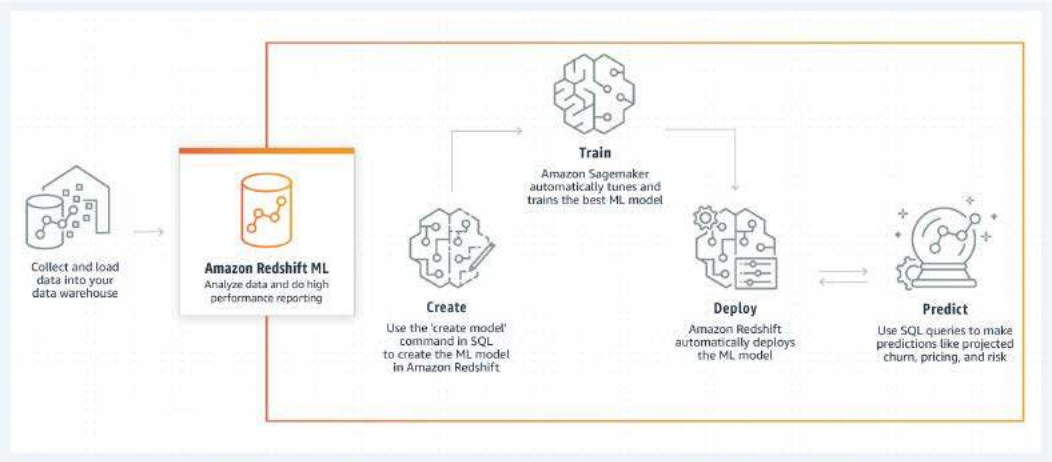


Figure 12.4 – Amazon Redshift ML capabilities

Getting ready

To complete this recipe, you will need:

- An Amazon Redshift data warehouse deployed in the eu-west-1 Region.
- Sample data loaded using the *Chapter 3* recipe *Loading and unloading data*.
- Amazon Redshift data warehouse admin user credentials.
- Access to any SQL interface, such as a SQL client or Amazon Redshift Query Editor.
- An IAM role attached to an Amazon Redshift data warehouse that can access Amazon S3 and Amazon SageMaker. We will refer to it in the recipes with `[Your-Redshift_Role]`.
- An Amazon S3 bucket created in eu-west-1. We will refer to it with `[Your-Amazon_S3_Bucket]`.

How to do it...

In this recipe, we will use the product reviews data that was set up in *Chapter 3*, in the *Loading and unloading data* recipe. We will build the model to predict the `star_rating` values of the `products` table:

1. Open any SQL client tool and execute the following query to create the training data to train the model by using `50000` records for the product category home:

```
create schema product_reviews;

create table product_reviews.amazon_reviews_train
as SELECT * FROM product_reviews where product_category = 'Home'
limit 50000;
```

2. To create the model, execute the following query. This will use Autopilot to determine the problem type, with a max runtime of 900 seconds. This model will predict the `star_rating`. The `CREATE MODEL` SQL will run asynchronously. With this step, Amazon Redshift will unload the data to the S3 bucket and Autopilot will use that dataset to train the model. After the model is trained, the code will be compiled using Amazon SageMaker Neo and will be deployed to the Amazon Redshift data warehouse. The model can then be accessed using the user-defined function `func_product_rating`:

```
CREATE MODEL product_rating
FROM (
SELECT marketplace, customer_id
       , review_id, product_id
       , product_parent, product_title
       , product_category, star_rating
       , helpful_votes, total_votes
       , vine, verified_purchase
       , review_headline, review_body
 FROM product_reviews.amazon_reviews_train
    ) TARGET star_rating
FUNCTION func_product_rating
IAM_ROLE '[Your-Redshift_Role]'
SETTINGS(S3_BUCKET '[Your-Amazon_S3_Bucket]', MAX_RUNTIME 1800, S3_
GARBAGE_COLLECT OFF);
```

3. To check the status of the model creation, execute the following query. Check if the model state is **Ready**. When the model state is **Ready**, it shows the problem type of **MulticlassClassification** and an accuracy of **0.62940** for this model:

```
show model product_rating;
```

The preceding query will return output similar to the following:

Key	Value
Schema Name	public
Owner	awuser
Creation Time	Tue, 30.03.2021 11:21:47
Model State	READY
Estimated Cost	0.964440
validation:accuracy	0.629420
Query	SELECT MARKETPLAC CUSTOMER_ID, REVIEV FROM DEV.PRODUCT_ REVIEWS.AMAZON_RE WHERE PRODUCT CA- TEGORY = 'HOME'
Target Column	STAR_RATING
PARAMETERS:	MulticlassClassification

Figure 12.5 – Output of the preceding query

4. To predict the `star_ratings`, execute the following query to validate the accuracy of the ML model. The user-defined function `func_product_rating` predicts the `star_rating` and we compare it to the actual value to determine the accuracy of the model:

```
WITH infer_data
AS (
    SELECT star_rating AS actual
       ,func_product_rating(marketplace
       , customer_id, review_id, product_id
       , product_parent, product_title
       , product_category, helpful_votes
       , total_votes, vine, verified_purchase
       , review_headline, review_body) AS predicted
       , CASE WHEN star_rating = predicted
                THEN 1::INT ELSE 0::INT
                END AS correct
```

```
    FROM product_reviews.amazon_reviews
    where product_category = 'Home'
    )
    ,aggr_data AS (
    SELECT SUM(correct) AS num_correct
        ,COUNT(*) AS total
    FROM infer_data
    )
SELECT (num_correct::FLOAT / total::FLOAT) AS accuracy
FROM aggr_data;
```

The preceding query will return the following output:

```
accuracy
0.627847778989157
```

How it works...

Amazon Redshift simplifies the pipeline to create the models and use the model for prediction using SQL. With Amazon Redshift, you can build models for different use cases, such as customer churn prediction, predicting whether a sales lead will close, fraud detection, etc. You can use simple SQL statements to create ML workflows. Use the `CREATE MODEL` SQL command to specify training data as either a table or a `SELECT` statement. Redshift ML then compiles and imports the trained model inside the Redshift data warehouse and prepares a SQL inference function that can be immediately used in SQL queries. Redshift ML automatically handles all the steps needed to train and deploy a model.

Using LLMs in Amazon Bedrock using SQL statements

In this recipe, you'll learn how to leverage LLMs for language translation directly within Amazon Redshift through its native integration with Amazon Bedrock. Amazon Redshift ML now extends its SQL-based ML capabilities to include foundation models available through Bedrock, enabling you to perform sophisticated natural language processing tasks using familiar SQL commands. We'll demonstrate this capability by translating order comments from Spanish to English using Anthropic Claude 3 Haiku, one of the foundation models available through Amazon Bedrock. This integration showcases how you can combine Amazon Redshift's data warehousing capabilities with advanced language models to perform translations directly within your database environment, eliminating the need for external translation services or data movement.

Getting ready

To complete this recipe, you will need:

- An Amazon Redshift data warehouse deployed in the AWS Region eu-west-1
- Amazon Redshift data warehouse admin user credentials
- Access to any SQL interface, such as a SQL client or Amazon Redshift Query Editor
- An IAM role attached to an Amazon Redshift data warehouse with the managed policy `AmazonBedrockFullAccess` attached to it

How to do it...

We will start by configuring access to the Anthropic Claude 3 Haiku model through Amazon Bedrock and load sample order data into Amazon Redshift. Then, we'll create a model using Amazon Redshift ML that connects to the Amazon Bedrock foundation model. Finally, we'll write SQL queries to invoke this model and translate the Spanish comments to English, demonstrating the seamless integration between these AWS services:

1. Navigate to the Amazon Bedrock console (`https://console.aws.amazon.com/bedrock/`).
2. In the navigation pane, choose **Model Access**, as shown in the following screenshot:

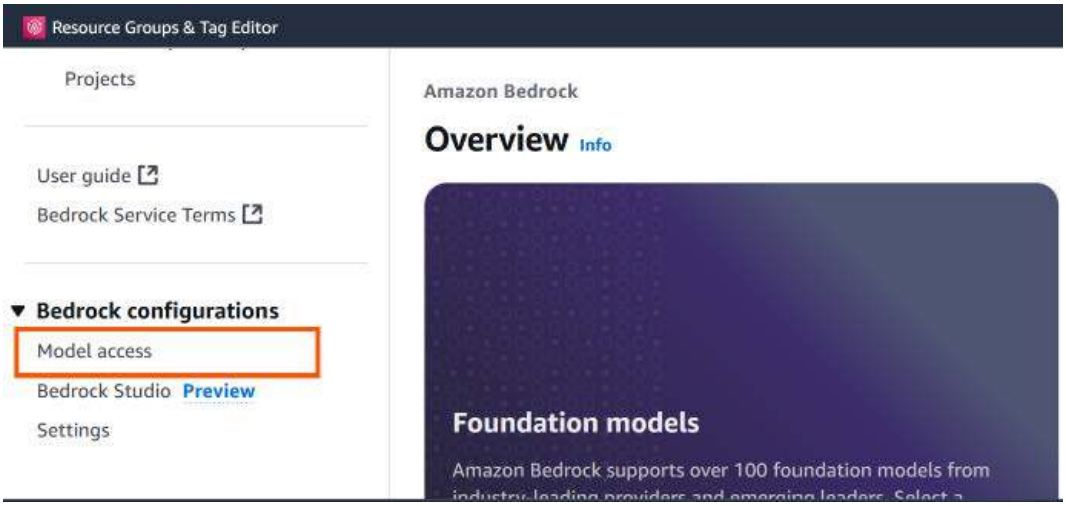


Figure 12.6 – Amazon Bedrock model access

3. If you are accessing Amazon Bedrock for the first time, you will see the **Enable specific models** button. Select it. Otherwise, you will see the **Modify model access** button. Select it.

4. In the search box, search for the `Claude 3 Haiku` model. Select it and click **Next**, as shown in the following screenshot:

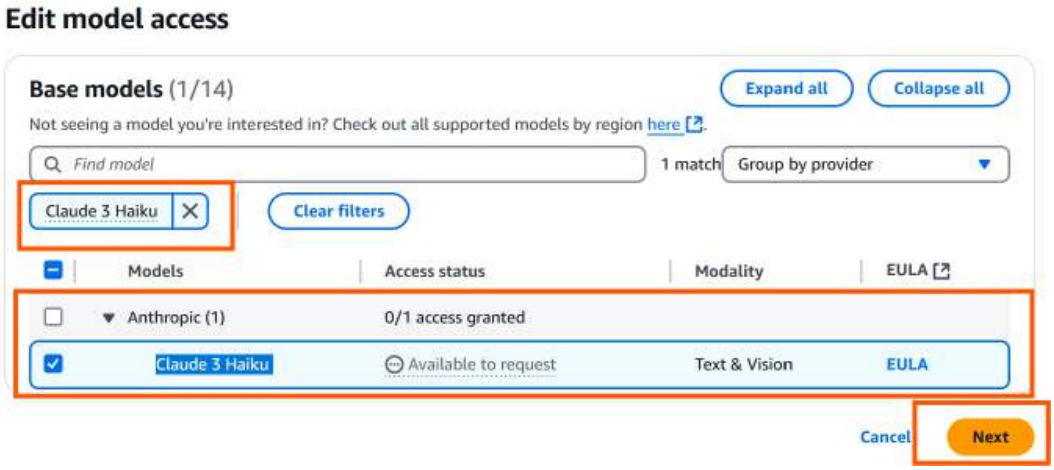


Figure 12.7 – Amazon Bedrock model selection

5. Read the terms and click the **Submit** button.
6. Next, connect to your Amazon Redshift data warehouse using Query Editor V2 (`https://console.aws.amazon.com/sqlworkbench/home`).
7. Create an `order_comments` table and load some sample data into it using the SQL statements provided below. The `insert` statement loads some orders with Spanish comments into the table:

```
create table order_comments (order_id int, comment varchar(100));

insert into order_comments values (1, 'Dirección de envío
actualizada'),
(2,'Por favor manipular con cuidado frágil'),(3,'Llamar antes de
entregar');
```

8. Create an external model using the SQL statement below. This will create a SQL function named `'translate_to_english'` that can be invoked in SQL statements. For `MODEL_ID`, Anthropic Claude 3 Haiku's ID is provided. Refer to this documentation page for more information about Amazon Bedrock model IDs (`https://docs.aws.amazon.com/bedrock/latest/userguide/model-ids.html`). The model's behavior is configured using two key parameters: `'Prompt'` and `'Suffix'`. The `'Prompt'` parameter provides the initial instruction to the LLM describing the translation task.

Your input data from Amazon Redshift is then automatically appended between the prompt and suffix. The `'Suffix'` parameter adds any final instructions needed to complete the prompt before it's sent to the LLM:

```
CREATE EXTERNAL MODEL claude_model_translate_to_english
FUNCTION translate_to_english
IAM_ROLE DEFAULT
MODEL_TYPE BEDROCK
SETTINGS (
   MODEL_ID 'anthropic.claude-3-haiku-20240307-v1:0'
   ,PROMPT 'Translate this statement from Spanish to English'
   , SUFFIX 'Return only the translated sentence in lowercase and
nothing else'
   , REQUEST_TYPE UNIFIED
   , RESPONSE_TYPE VARCHAR
);
```

9. Run the following SQL to translate the Spanish comments to English:

```
SELECT comment as comment_in_spanish,
translate_to_english(comment) as comment_in_english
from order_comments
limit 5;
```

Figure 12.8 – Output of Amazon Redshift and Amazon Bedrock integration

How it works...

Amazon Redshift's integration with Amazon Bedrock enables direct access to foundation models through SQL statements. It has the following features:

- **Native integration**: Amazon Redshift connects directly with Amazon Bedrock through built-in integration, allowing you to perform generative AI tasks using SQL commands across both provisioned and serverless environments.
- **External model creation**: The `CREATE EXTERNAL MODEL` statement establishes a connection with the specified Bedrock foundation model and automatically generates a SQL function for model interaction.
- **Prompt engineering**: Redshift constructs a complete prompt by combining your template, input data, and suffix instructions, then passes these to the Converse API, ensuring consistent interaction with foundation models.
- **Execution**: The translation function handles communication between Redshift and Bedrock automatically, with the foundation model processing requests and returning results in real time for use in SQL operations.

Using LLMs in Amazon SageMaker Jumpstart using SQL statements

In this recipe, you'll explore how to use LLMs for tasks like sentiment analysis directly within Amazon Redshift by leveraging the integration with Amazon SageMaker Jumpstart. Amazon Redshift ML allows you to create ML models using SQL commands, and now, with LLM support, you can tap into powerful pre-trained models for text processing.

By following this recipe, you'll learn how to deploy an LLM through SageMaker Jumpstart, connect it to Redshift ML, and perform advanced natural language processing tasks like sentiment classification on your data—all without the need to manage complex ML pipelines. This integration simplifies the use of generative AI for extracting insights from unstructured data in Redshift.

Getting ready

To complete this recipe, you will need:

- An Amazon Redshift data warehouse deployed in the AWS Region eu-west-1
- Amazon Redshift data warehouse admin user credentials
- Access to any SQL interface, such as a SQL client or Amazon Redshift Query Editor

- An IAM role attached to an Amazon Redshift data warehouse that can invoke Amazon SageMaker Jumpstart endpoints. Ensure that the role includes `sagemaker:InvokeEndpoint` permissions.

How to do it...

We will start by deploying a pre-trained LLM using Amazon SageMaker Jumpstart. This will involve navigating to SageMaker in the AWS Management Console, selecting a foundation model, and setting up an endpoint that will be used by Amazon Redshift ML to perform sentiment analysis:

1. Navigate to **Foundation models** (`https://console.aws.amazon.com/sagemaker/home#/foundation-models`) in Amazon SageMaker Jumpstart.
2. Search for the foundation model by typing `Falcon 7B Instruct BF16` in the search box and click **View model**:

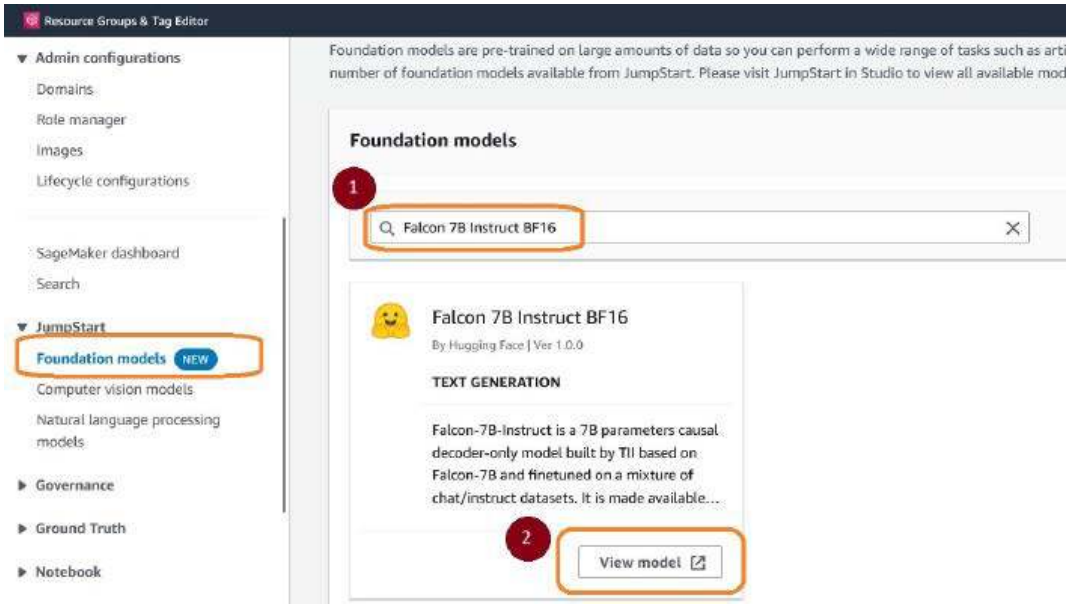


Figure 12.9 – Selecting an LLM in Amazon SageMaker Jumpstart

3. On the **Model Details** page, choose **Open model in Studio**. When the **Select domain and user profile** dialog box opens up, choose the profile you'd like from the dropdown and click **Open Studio**. (If this is the first time you're doing this, you may see the **Create a SageMaker domain** button. Click it and select **Setup**.) When the model is open, select **Deploy**, as shown in the following screenshot:

Figure 12.10 – Deploy the LLM

4. On the **Deploy model to endpoint** page, select `ml.g5.2xlarge` or any other instance type recommended in the notebook, and then click **Deploy**. Wait until the status of the model changes from **Creating** to **In Service**:

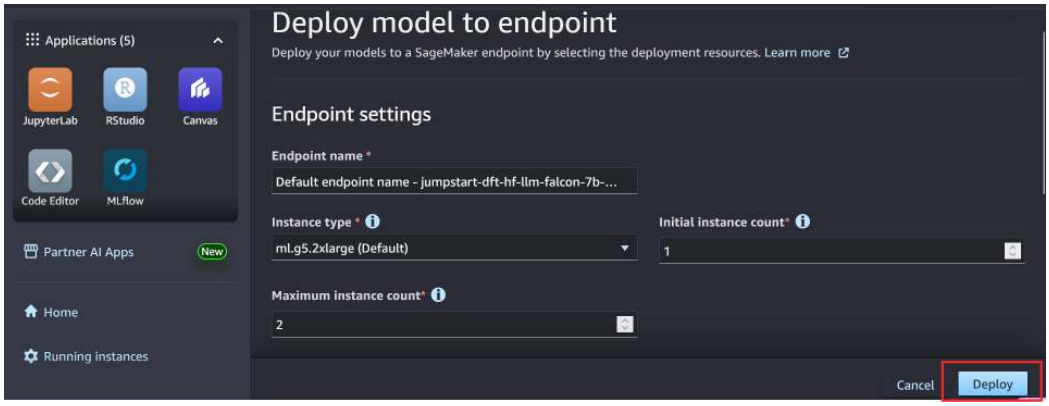


Figure 12.11 – Setting up the SageMaker endpoint environment

Figure 12.12 – Model in service

5. Log in to the Amazon Redshift endpoint. You can use Query Editor V2 to log in.
6. Ensure you have the below IAM policy added to your IAM role. Replace `<endpointname>` with the SageMaker Jumpstart endpoint name captured earlier:

```
{
  "Statement": [
      {
          "Action": "sagemaker:InvokeEndpoint",
          "Effect": "Allow",
          "Resource":
"arn:aws:sagemaker:<region>:<AccountNumber>:endpoint/<endpointname>",
          "Principal": "*"
      }
  ]
}
```

7. Create a model in Amazon Redshift using the `CREATE MODEL` statement given below. Replace `<endpointname>` with the endpoint name captured earlier. The input and output data type for the model needs to be `SUPER`:

```
CREATE MODEL falcon_7b_instruct_llm_model
FUNCTION falcon_7b_instruct_llm_model(super)
RETURNS super
SAGEMAKER '<endpointname>'
IAM_ROLE default;
SETTINGS (
   MAX_BATCH_ROWS 1 );
```

8. Create the `sample_reviews` table using the following SQL statement. This table will store the sample reviews dataset:

```
CREATE TABLE sample_reviews(review varchar(4000));
```

9. Download the sample file (`https://aws-blogs-artifacts-public.s3.amazonaws.com/BDB-3745/sample_reviews.csv`), upload it into your S3 bucket, and load data into the `sample_reviews` table using the following `COPY` command:

```
COPY sample_reviews
FROM 's3://<<your_s3_bucket>>/sample_reviews.csv'
IAM_ROLE DEFAULT
```

```
csv
DELIMITER ','
IGNOREHEADER 1;
```

10. Create a UDF that engineers the prompt for sentiment analysis. The input to the LLM consists of two main parts – the prompt and the parameters. The **prompt** is the guidance or set of instructions you want to give to the LLM. The prompt should be clear to provide proper context and direction for the LLM. The parameters (`https://huggingface.co/blog/sagemaker-huggingface-llm#4-run-inference-and-chat-with-our-model`) allow configuring and fine-tuning the model's output. This includes settings like maximum length, randomness levels, stopping criteria, and more. Parameters give control over the properties and style of the generated text and are model-specific.

    The UDF given below has both the prompt and a parameter:

    - Prompt: `Classify the sentiment of this sentence as Positive, Negative, Neutral. Return only the sentiment nothing else`. This instructs the model to classify the review into three sentiment categories.
    - Parameter: `"max_new_tokens":1000`. This allows the model to return up to 1,000 tokens.

    ```
    CREATE FUNCTION udf_prompt_eng_sentiment_analysis (varchar)
    returns super
    stable
    as $$
      select json_parse(
      '{"inputs":"Classify the sentiment of this sentence as
    Positive, Negative, Neutral. Return only the sentiment
    nothing else.' || $1 || '","parameters":{"max_new_
    tokens":1000}}')
    $$ language sql;
    ```

    - Make a remote inference to the LLM to generate sentiment analysis for the input dataset.

      The output of this step is stored in a newly created table called `sentiment_analysis_for_reviews`. Run the below SQL statement to create a table with output from the LLM:

      ```
      CREATE table sentiment_analysis_for_reviews
      as
      ```

```
(
    SELECT review,
        falcon_7b_instruct_llm_model
        (udf_prompt_eng_sentiment_analysis(review)) as
sentiment
    FROM sample_reviews
);
```

11. Analyze the output. The output of the LLM is of the data type `SUPER`. For the Falcon model, the output is available in the attribute named `generated_text`. Each LLM has its own output payload format. Please refer to the documentation for the LLM you would like to use for its output format. Run the below query to extract the sentiment from the output of the LLM. For each review, you can see its sentiment analysis:

```
SELECT review, sentiment[0]."generated_text" :: varchar as sentiment
FROM sentiment_analysis_for_reviews;
```

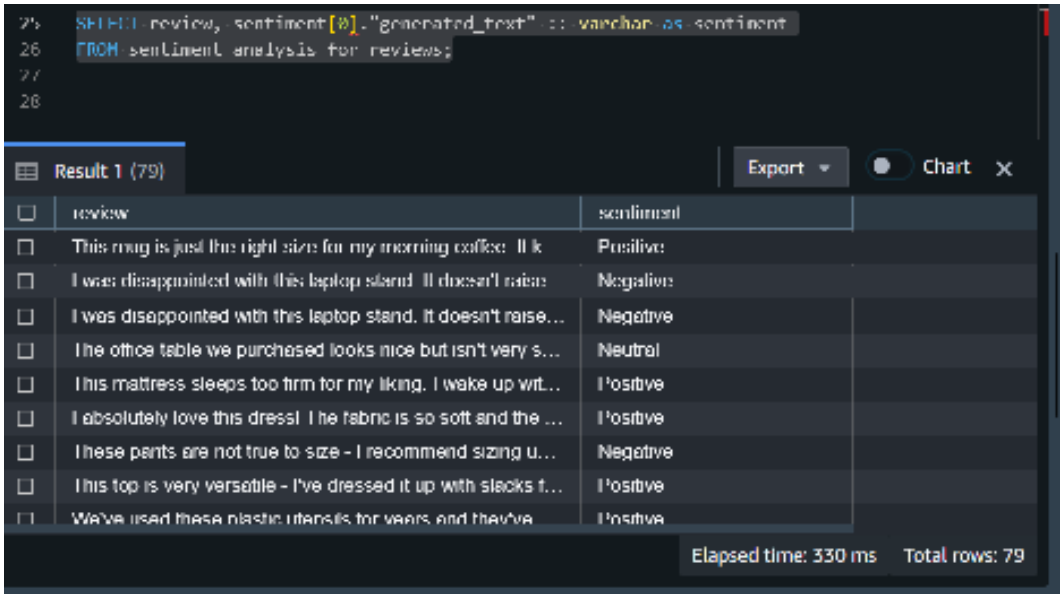


Figure 12.13 – Output of sentiment analysis in Amazon Redshift using LLM integration

How it works...

Amazon Redshift ML enables direct integration with LLMs from Amazon SageMaker Jumpstart using SQL. It curates the prompt and sends it to the model endpoint deployed using SageMaker Jumpstart. The request is sent in JSON format and the response from the LLM is also received in JSON format, which is then parsed and shown to the end user.

This streamlined workflow enables data teams to leverage advanced NLP capabilities directly within their SQL environment, eliminating the need for complex ML pipelines.

Querying your data with natural language prompts using Amazon Bedrock knowledge bases for Amazon Redshift

In this recipe, you'll learn how to implement natural language querying capabilities for your Amazon Redshift data warehouse using Amazon Bedrock knowledge bases. This powerful integration allows business users to interact with their data using conversational language, rather than writing complex SQL queries. Amazon Bedrock knowledge bases create an intelligent layer between your structured data in Redshift and LLMs, enabling users to ask questions in plain English and receive data-driven responses. We'll demonstrate how to set up a knowledge base with your Redshift data warehouse as the data source, configure the necessary permissions, and start querying your data using natural language prompts with models like Anthropic Claude 3 Haiku. This solution bridges the gap between complex data structures and business users, democratizing data access while maintaining security and governance through proper IAM roles and permissions.

Getting ready

To complete this recipe, you will need:

- An Amazon Redshift data warehouse deployed in the AWS Region eu-west-1
- Amazon Redshift data warehouse admin user credentials
- Load the sample data in the Amazon Redshift data warehouse using the *Loading data from Amazon S3 using COPY* recipe in *Chapter 3* into the dev database

How to do it...

We will create a knowledge base with a structured data store with your Amazon Redshift data warehouse as a data source. We will then grant access to the data in your Amazon Redshift data warehouse to the role associated with the knowledge base. We will then test the knowledge base by asking natural language questions about the data stored:

1. Navigate to the Amazon Bedrock console (`https://console.aws.amazon.com/bedrock/`).
2. In the navigation pane, choose **Knowledge Bases**. In the page that opens up, click **Create** and then **Knowledge Base with structured data store**, as shown in the following screenshot:

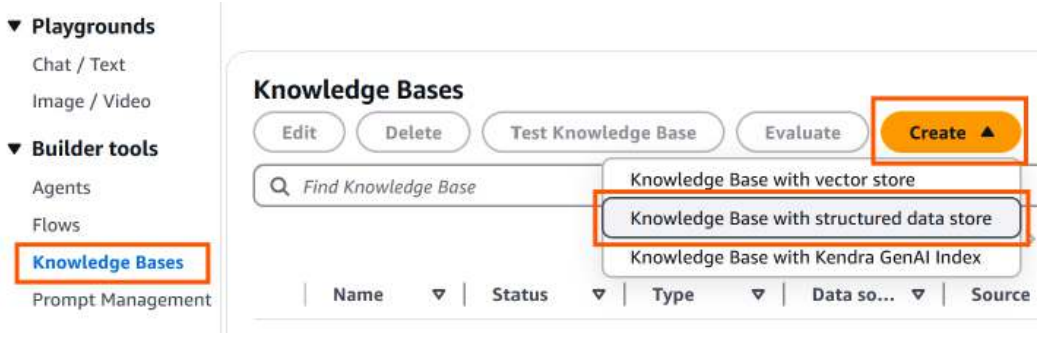


Figure 12.14 – Creating a knowledge base with a structured data store

3. For **Knowledge Base name**, choose a name of your choice, for example, `knowledge base for redshift sales warehouse`. Enter an optional description. In **Data source details**, choose **Amazon Redshift** as the query engine. In the **IAM Permissions** section, choose the **Create and use a new service role** option and click **Next**.

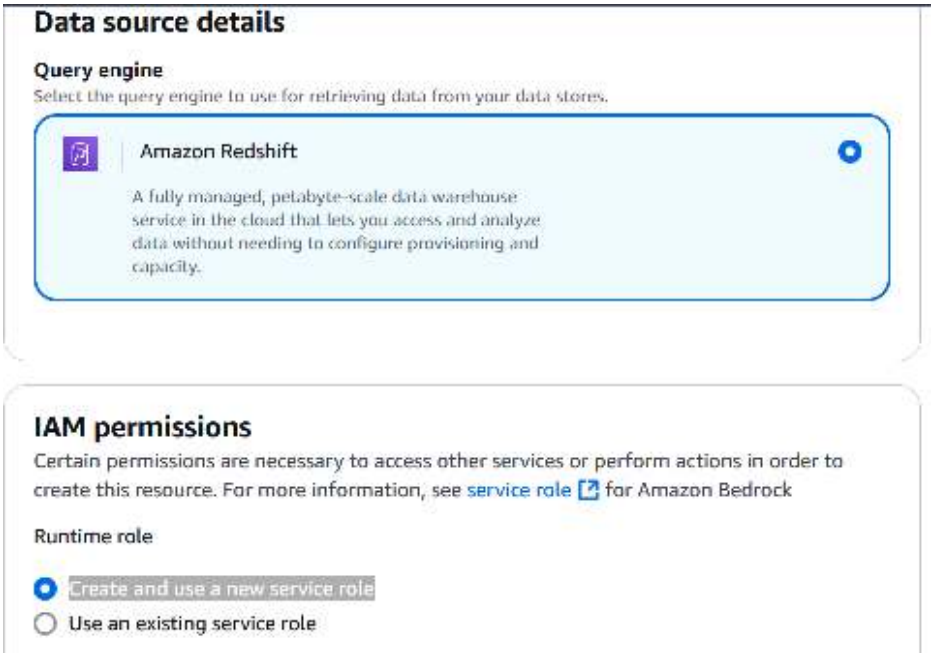


Figure 12.15 – Choose Amazon Redshift as data source, and create and use new service role for IAM permissions

4. On the page that opens, in the **Query engine details** section, choose **Redshift Serverless** or **Redshift Provisioned** based on the compute type for your Amazon Redshift data warehouse.

5. In the **Default storage metadata** section, under **Options**, choose **Amazon Redshift databases** and **Redshift database list**. If you are using data sharing using Lake Formation, you can choose the **AWS Default Glue Data Catalog** option here. In the same section, for **Database**, choose the database you want to connect to, for example, **dev**:

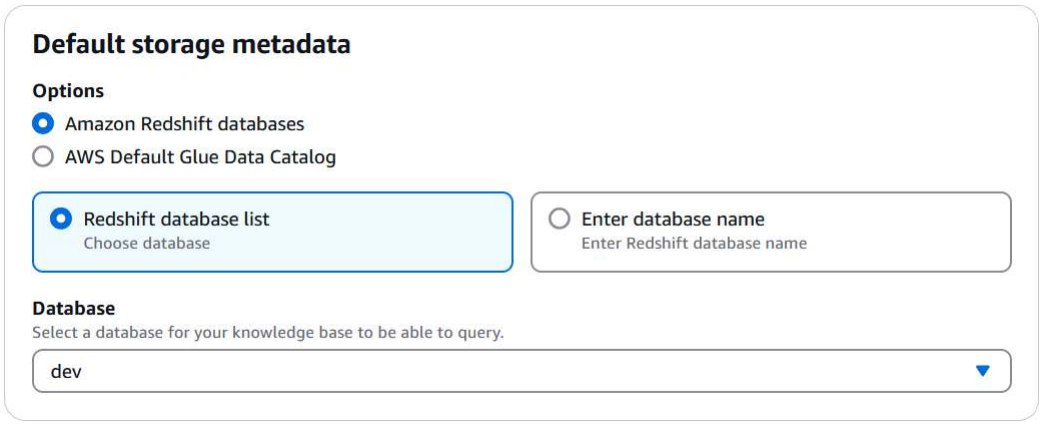


Figure 12.16 – Choose default storage metadata

6. You can use the optional **Query configurations** section to provide more information to the LLM regarding the table structure. You can add descriptions of tables and columns, as shown in *Figure 12.17*.

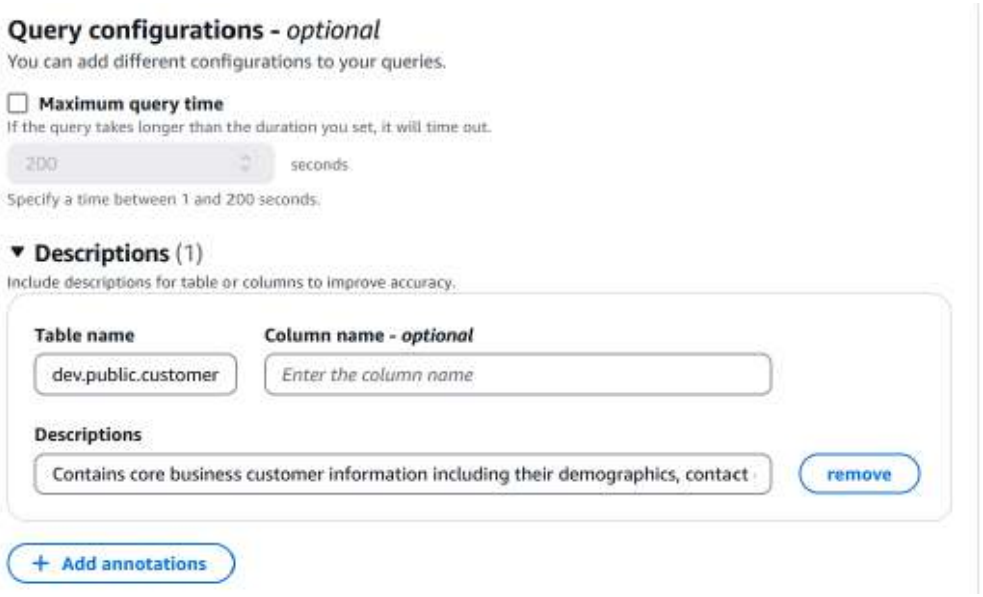


Figure 12.17 – Sample description for the customer table

7. You can choose to include/exclude columns that the LLM should consider. You can provide curated queries that the LLM can use as a reference to generate new queries, as shown in *Figure 12.18*:

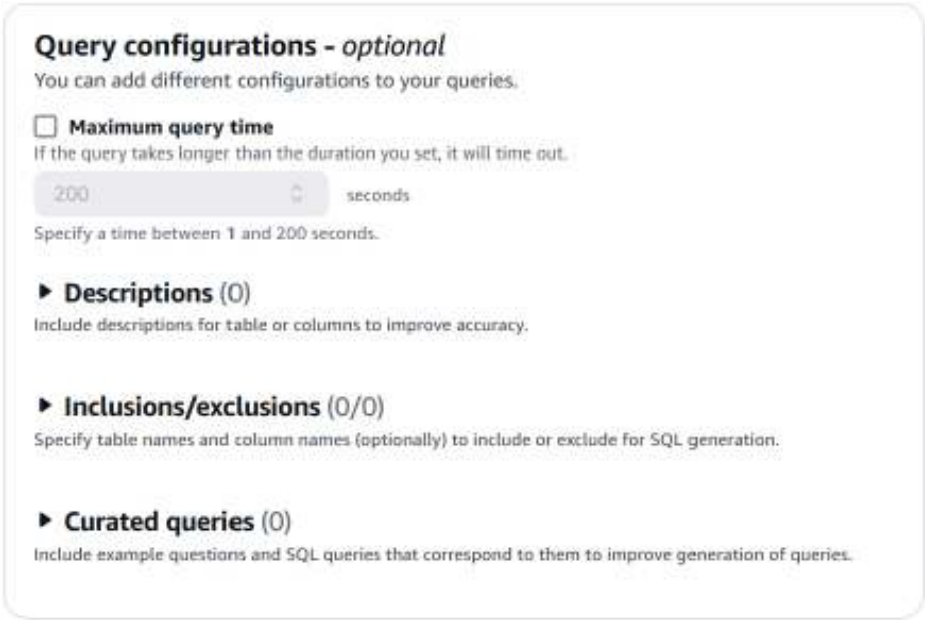


Figure 12.18 – Optional configurations for a knowledge base for structured data

If you don't configure this section, the LLM will use the table and column names to understand the table schema. You can also choose the maximum time a query issued by Amazon Bedrock is allowed to run on Amazon Redshift using the **Maximum query time** attribute.

8. Click **Next**, and on the next page, review the configuration and select **Create Knowledge Base**. Once the knowledge base is available, copy the name of the service role:

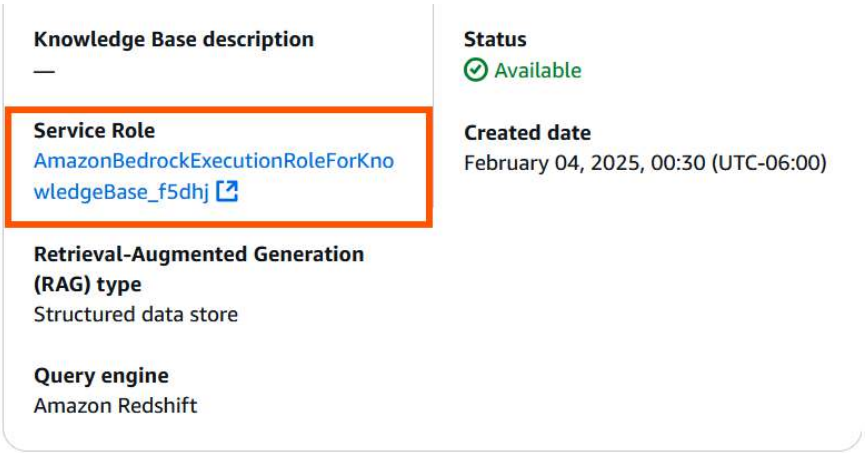


Figure 12.19 – Copy the service role's ARN by clicking on the link as shown

9. Connect to your Amazon Redshift warehouse using Query Editor V2 and run the below SQL statements to grant `SELECT` permission on your tables to the service role:

```
CREATE USER "IAMR:service-role-name" WITH PASSWORD DISABLE;
GRANT SELECT ON customer TO "IAMR:service-role-name";
GRANT SELECT ON orders TO "IAMR:service-role-name";
GRANT SELECT ON lineitem TO "IAMR:service-role-name";
GRANT SELECT ON supplier TO "IAMR:service-role-name";
GRANT SELECT ON dwdate TO "IAMR:service-role-name";
GRANT SELECT ON part TO "IAMR:service-role-name";
```

10. Go back to the knowledge base, select the **dev** database in the **Query engine** section, and click **Sync**:

Figure 12.20 – Sync the query engine

11. When the sync completes, click the **Test** button to start testing the knowledge base:

Figure 12.21 – Start testing the knowledge base

12. In the test window, you can select any model you'd like, for example, Anthropic Claude 3 Haiku, and start asking it questions in natural language. For example, you can ask `Name the customer who placed the highest number of orders`, as shown in the following screenshot, and receive a natural language answer based on your data. You can ask complex, open-ended analysis questions to perform generative BI using this solution.

For example, you can ask `Who is the most effective supplier` and get the following response:

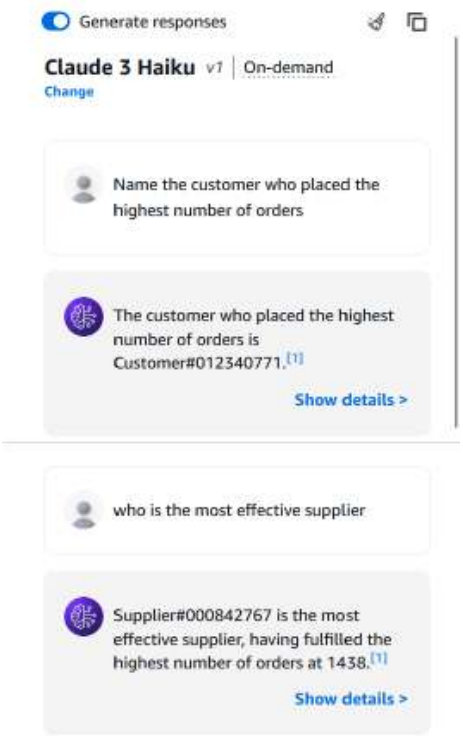


Figure 12.22 – Natural language query response

How it works...

Here's how Amazon Bedrock knowledge bases work with structured data:

- **Schema understanding**: Amazon Bedrock fetches your schema either by directly accessing Amazon Redshift metadata or by reading schema information from AWS Glue Data Catalog. It then understands the business context of the schema using either the names or the query configurations you provided.
- **Natural language processing and SQL generation**: When you ask a question in plain English, Bedrock interprets your intent and understands the context and requirements of your query. It then converts your natural language question into one or more SQL queries designed to extract exactly the information needed from your database.

- **Data retrieval and presentation**: The generated SQL query or queries execute against your Amazon Redshift database. Bedrock fetches the results and transforms them into natural language, providing you with a clear, conversational response that answers your original question.

Generative BI with Amazon Q with QuickSight querying an Amazon Redshift dataset

In this recipe, you'll learn how to leverage the generative AI capabilities of Amazon Q in Amazon QuickSight to enhance your BI workloads on top of your Amazon Redshift data warehouse. Amazon Q allows users to interact with their data through natural language queries, interpreting their natural language prompts and generating relevant visualizations. By combining the powerful data processing of Amazon Redshift with the AI-driven interface of Amazon Q, this solution empowers both technical and non-technical users to uncover actionable insights more efficiently. You'll discover how to leverage Amazon Q to build dashboards, create topics for enhanced Q&A experiences, and generate automated data stories – all without the need for complex SQL queries or extensive data modeling or reporting knowledge. This recipe showcases how generative AI can transform traditional BI workflows and democratize data access across your organization.

Getting ready

To complete this recipe, you will need:

- An Amazon Redshift data warehouse deployed in the AWS Region eu-west-1, with the retail dataset from *Chapter 3*
- Amazon Redshift data warehouse master user credentials
- Access to Amazon Redshift Query Editor V2
- To get started with Amazon Q in QuickSight's generative BI capabilities, upgrade your account's users to Admin Pro, Author Pro, or Reader Pro roles
- Ensure that you deploy the CloudFormation template provided on GitHub in order to create the `quicksight-role` role needed for this recipe (`https://github.com/PacktPublishing/Amazon-Redshift-Cookbook-2E/blob/main/Chapter11/chapter_11_CFN.yaml`)

How to do it...

1. Navigate to the Amazon Redshift console, open Query Editor V2, and create a view called `customer_orders` using the following SQL statement. In the next steps, we will see how you can visualize this view using generative BI capabilities in Amazon QuickSight powered by Amazon Q:

```
create view customer_orders
AS
(
    select c_name, c_address, c_acctbal,c_mktsegment,o_orderstatus,
o_totalprice,o_orderdate,o_orderpriority,o_clerk,o_shippriority
    from customer c
    join orders o on c.c_custkey = o.o_custkey
);
```

2. Navigate to Amazon QuickSight (`https://quicksight.aws.amazon.com/sn/start`), click the profile icon in the top-right corner, and select **Manage QuickSight**.

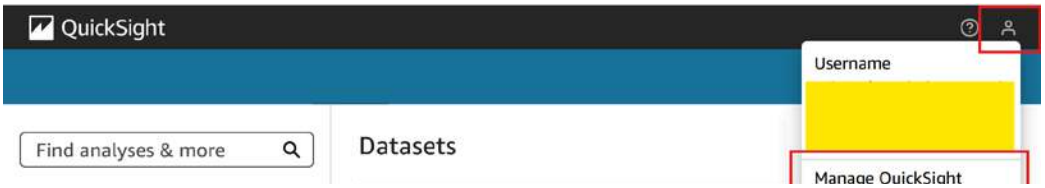


Figure 12.23 – Select Manage QuickSight to change the settings

3. Choose **Manage users** from the left navigation menu and ensure that the role for the user you are logged in as is **Admin Pro**. If it is not Admin Pro, change the role and click **CONFIRM** on the **Change user role confirmation** page that pops up. Then, sign out and sign back in:

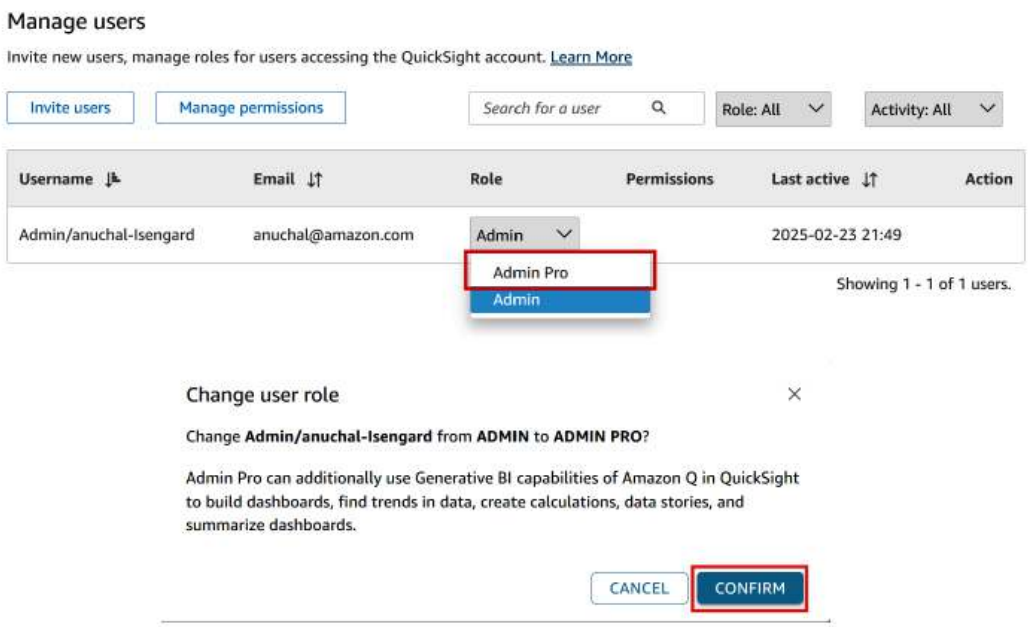


Figure 12.24 – Change the user role to Admin Pro and confirm the change

4. Navigate again to Amazon QuickSight (`https://quicksight.aws.amazon.com/sn/start`), click the profile icon in the top-right corner, and click **Manage QuickSight**. Then, choose **Manage VPC connections** from the left navigation pane and click **ADD VPC CONNECTION**.

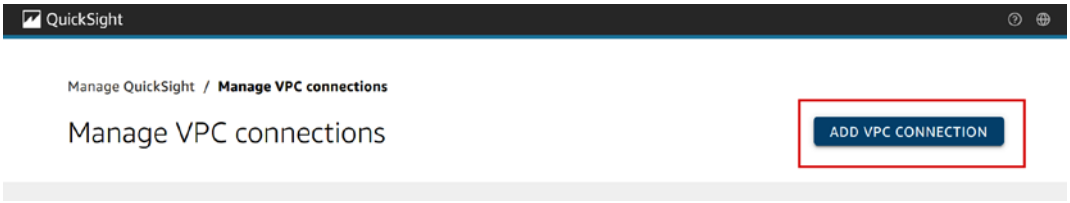


Figure 12.25 – Click ADD VPC CONNECTION

5. On the **VPC connection creation** page, enter the following details:
    - For **VPC Connection name**, enter a representative name like `redshift-vpc-connection`
    - For **VPC ID**, choose the VPC that your Amazon Redshift data warehouse is deployed in
    - For **Execution role**, choose `quicksight-role`

- For **Subnets**, choose the subnet for the **Availability Zone** (**AZ**) that is used for Amazon Redshift
- For **Security group**, choose the security group associated with your Amazon Redshift data warehouse

6. Once done, click **ADD** and wait until the status of the VPC connection changes from **UNAVAILABLE** to **AVAILABLE** (refresh the page to get the most recent status).

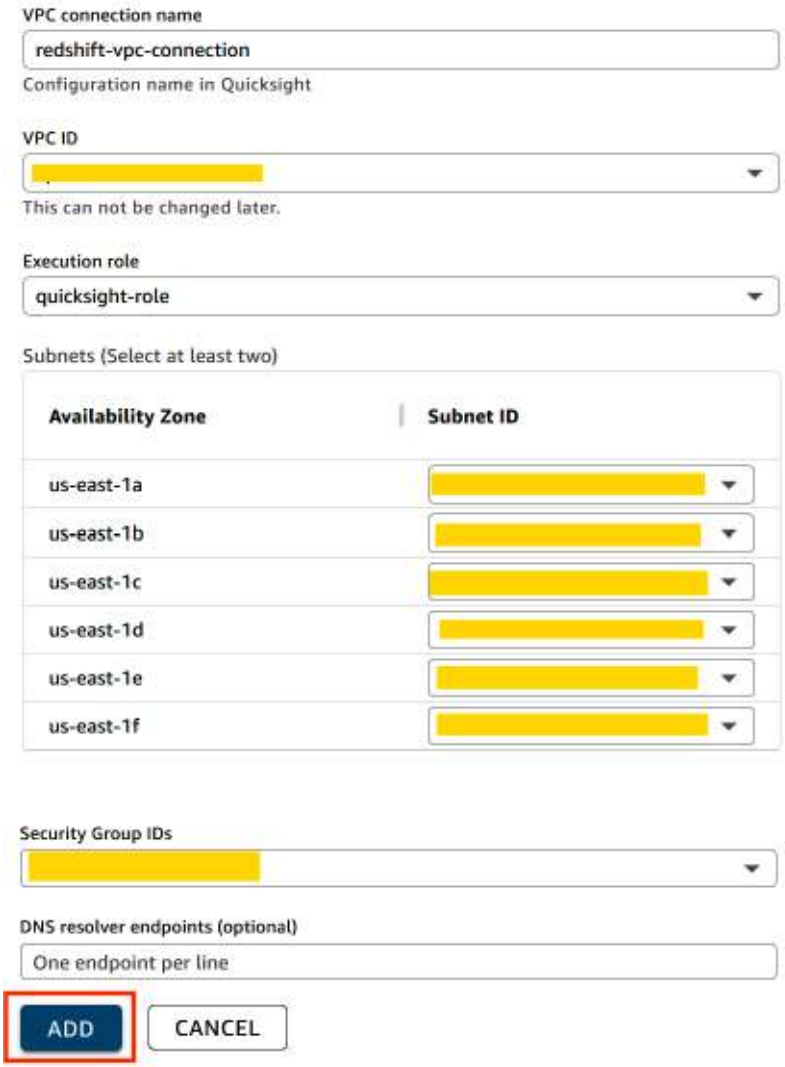


Figure 12.26 – VPC connection details and creation

7. Navigate to Amazon QuickSight (`https://quicksight.aws.amazon.com/sn/start`), and from the left navigation menu, choose **Datasets** and then select **New dataset**.

Figure 12.27 – Create a new QuickSight dataset

8. You have two ways of connecting to an Amazon Redshift data warehouse: **Redshift Auto-discovered** and **Redshift Manual connect**. If you are using an Amazon Redshift provisioned cluster, you can choose either of the options. If you are using an Amazon Redshift Serverless data warehouse, choose **Redshift Manual connect**. In this recipe, we are using the **Redshift Manual connect** option.

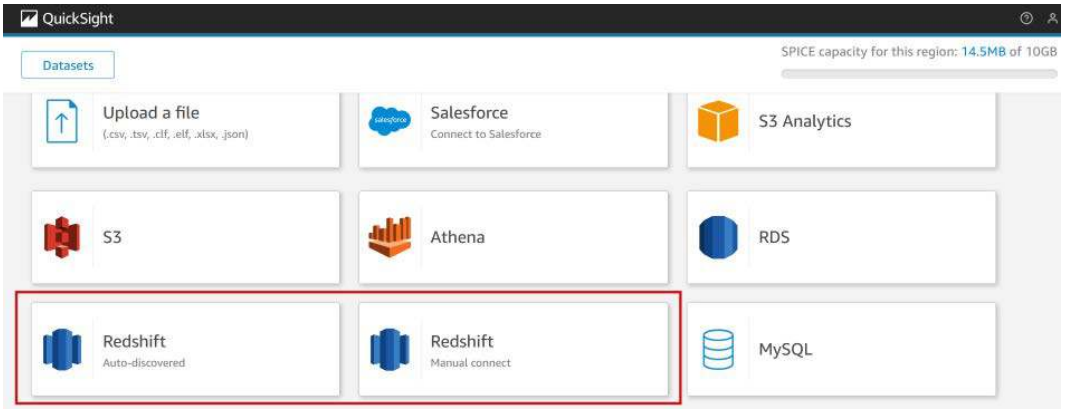


Figure 12.28 – Choose Amazon Redshift as a source for the dataset

9. On the **New Redshift data source** page that pops up:

    - For **Data source name**, provide a representative name like `customer-orders-datasource`.
    - For **Connection type**, choose `redshift-vpc-connection`.
    - For **Database server**, enter your Amazon Redshift data warehouse endpoint without the port and database name. For example, `cookbook-demo.1234567890.us-east-1.redshift-serverless.amazonaws.com`.

- For **Port**, enter the port number associated with your Amazon Redshift data warehouse. For example, 5439. For **Database name**, enter your Amazon Redshift database name, for example, dev.
- For **Username**, enter the admin username, and for **Password**, enter the admin user's password. Click **Validate connection** and wait until the button changes to **Validated**. Then, select **Create data source**.

Figure 12.29 – Enter connection details for Amazon Redshift

10. On the **Choose your table** page, choose the **customer_orders** view and click **Select**:

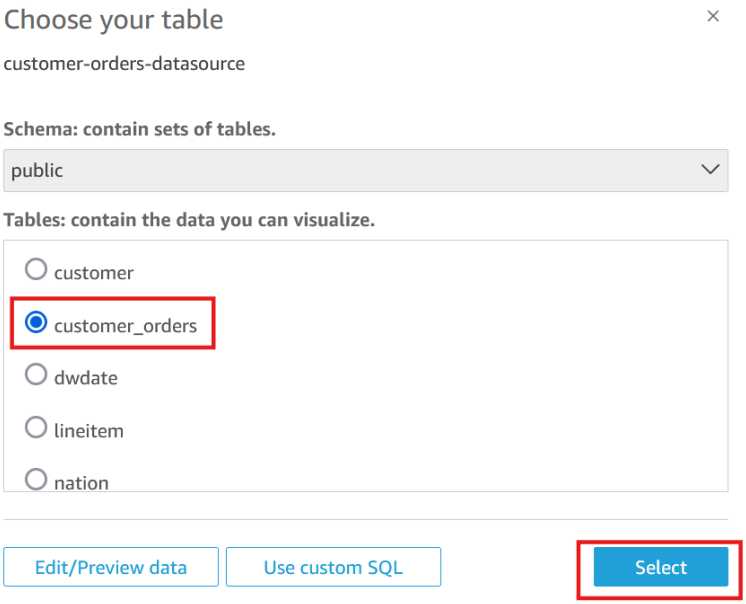


Figure 12.30 – Select tables/views for visualization

11. On the **Finish dataset creation** page, you can choose one of the two options: **Import to SPICE for quicker analytics** or **Directly query your data**. For this recipe, we will choose the **Directly query your data** option. Click **Visualize**.

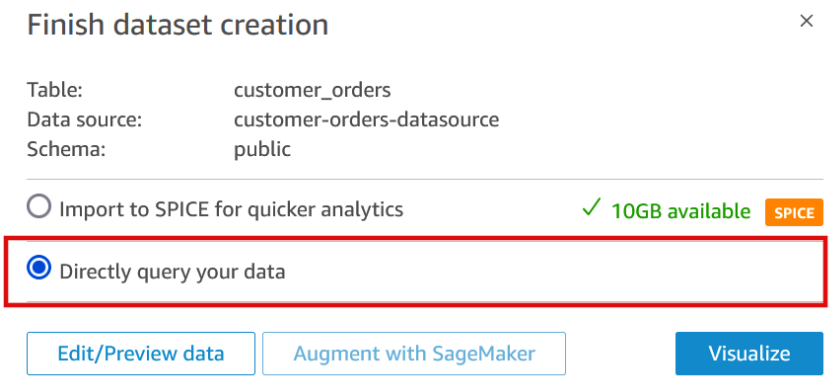


Figure 12.31 – Choose dataset type

12. When the **New sheet** page opens, click **CREATE**.

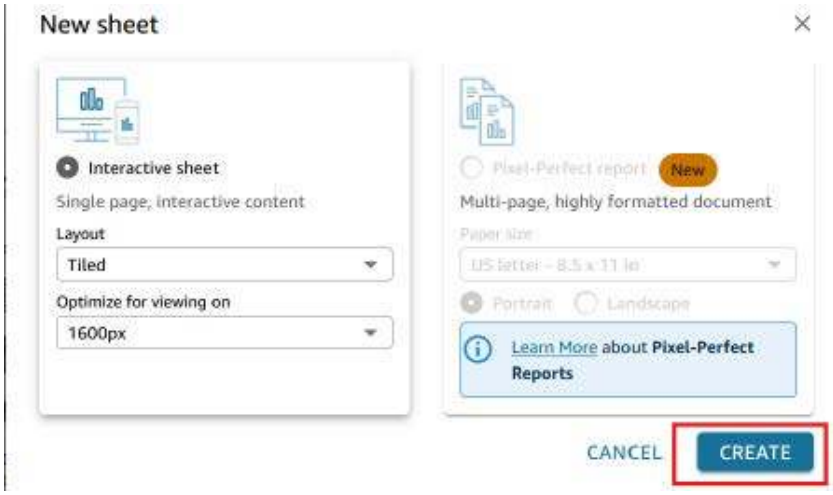


Figure 12.32 – Create a new QuickSight analysis sheet

13. `customer_orders` analysis will be created. To this analysis, you can now add visuals using QuickSight Q. You can enter natural language prompts describing what you want to visualize and QuickSight Q will automatically generate those visuals for you.

14. Click the **BUILD** option in the visual to open QuickSight Q in the right-side pane. You can enter a prompt for the visual you want, for example, `Stacked bar chart for total revenue by year and market segment`. Then, click **BUILD**. QuickSight Q will generate a visual. Click the **ADD TO ANALYSIS** button to add the visual to the analysis.

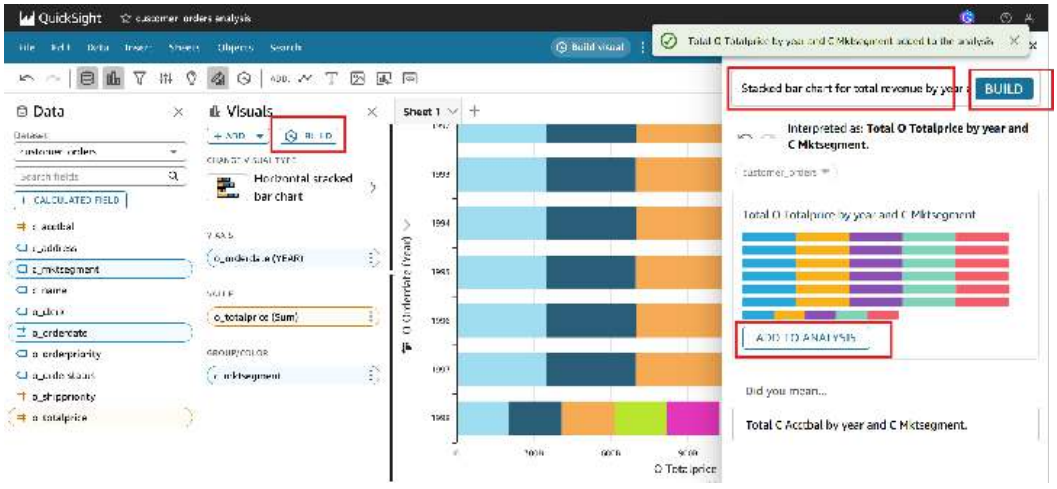


Figure 12.33 – Create visual for revenue by year and segment

15. Try one more prompt, `Pie chart for order amt by market segment`. QuickSight Q will generate a pie chart; add it to the visual by clicking the **ADD TO ANALYSIS** button.

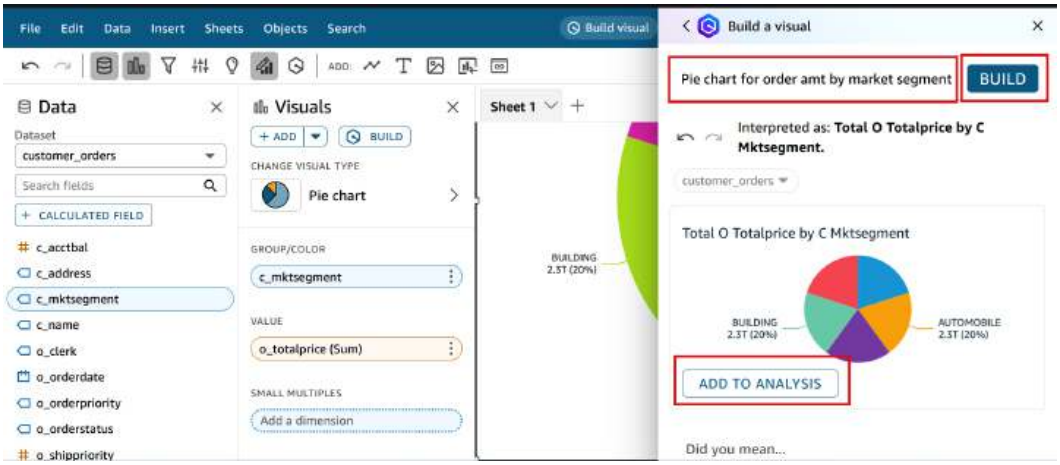


Figure 12.34 – Create a pie chart for total order amount by market segment

16. The analysis will look like this after you have added both of the visuals. Click **Publish** in the top-right corner to publish the visual to a dashboard.

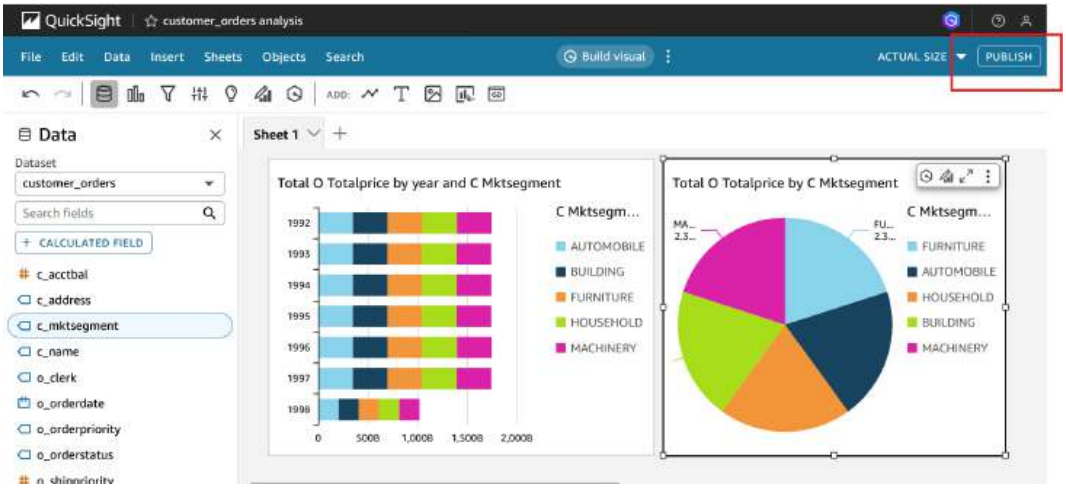


Figure 12.35 – Analysis with QuickSight Q-generated visuals

17. On the **Publish a dashboard** page, provide a representative name for the dashboard, for example, `market segment dashboard`. In the **Generative capabilities** section, select both the **Allow executive summary** and **Allow data Q&A** options and click **Publish dashboard**.

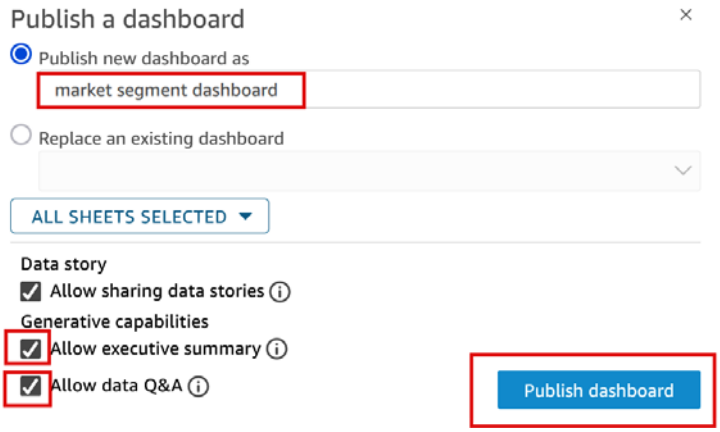


Figure 12.36 – Publish analysis to dashboard

18. Now let's see how you can generate data stories to create a narrative and share it across the organization. Navigate again to Amazon QuickSight (`https://quicksight.aws.amazon.com/sn/start`) and choose **Data stories** from the left navigation pane. Then, click **New data story**.

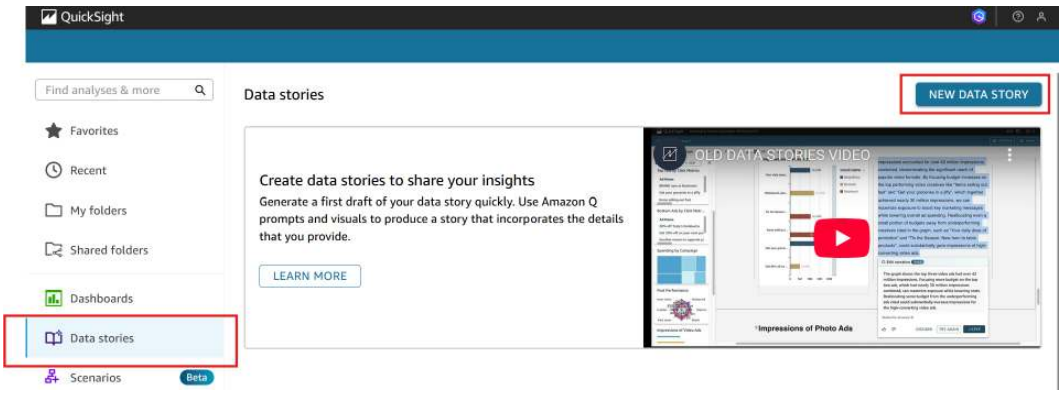


Figure 12.37 – Build a new data story

19. In the **Build story** wizard of QuickSight Q, enter a prompt, `Build a story describing how market segments are performing`, and uncheck the **Use insights from Amazon Q Business** option.

In the **Select visuals** section, click **+Add**. Choose the visuals you created earlier and then click **BUILD** and wait for QuickSight Q to automatically generate the story for you.

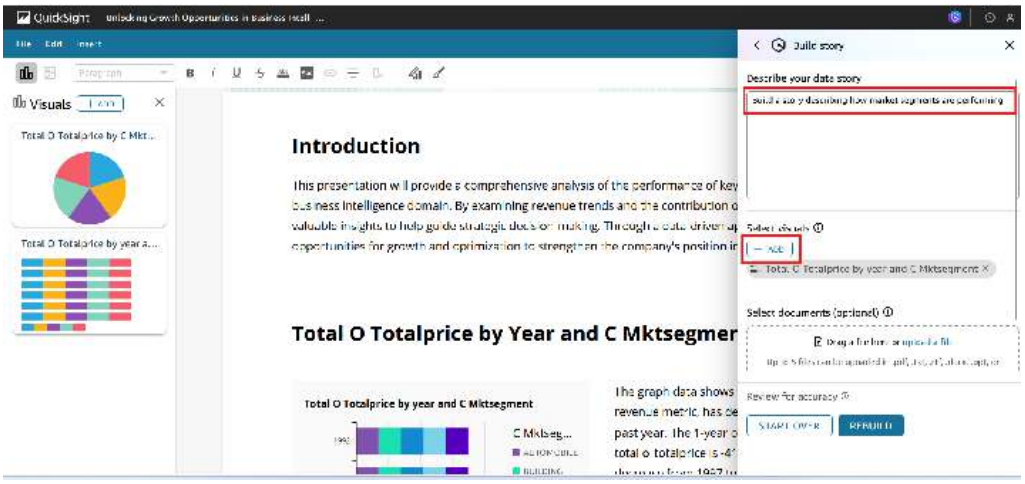


Figure 12.38 – Build a new data story using QuickSight Q

20. QuickSight Q will generate an editable story, as shown in the following screenshot, which you can share across your organization.

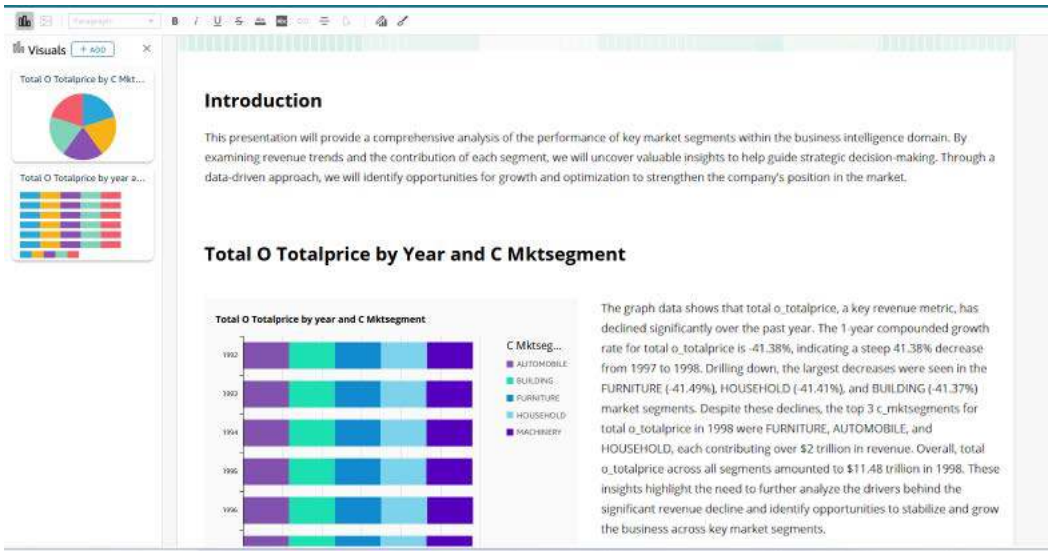


Figure 12.39 – QuickSight Q generated data story

21. Using QuickSight Q, you can also ask natural language questions about the dashboard and get answers in the form of visuals, tables, and natural language. Navigate to Amazon QuickSight (`https://quicksight.aws.amazon.com/sn/start`), and from the left navigation menu, choose **Dashboards** and select the dashboard you previously published – the market segment dashboard.
22. Click the **Ask a question about this dashboard** option at the top and ask `What is the total revenue for machinery segment`. QuickSight Q will answer you as shown in the following screenshot:

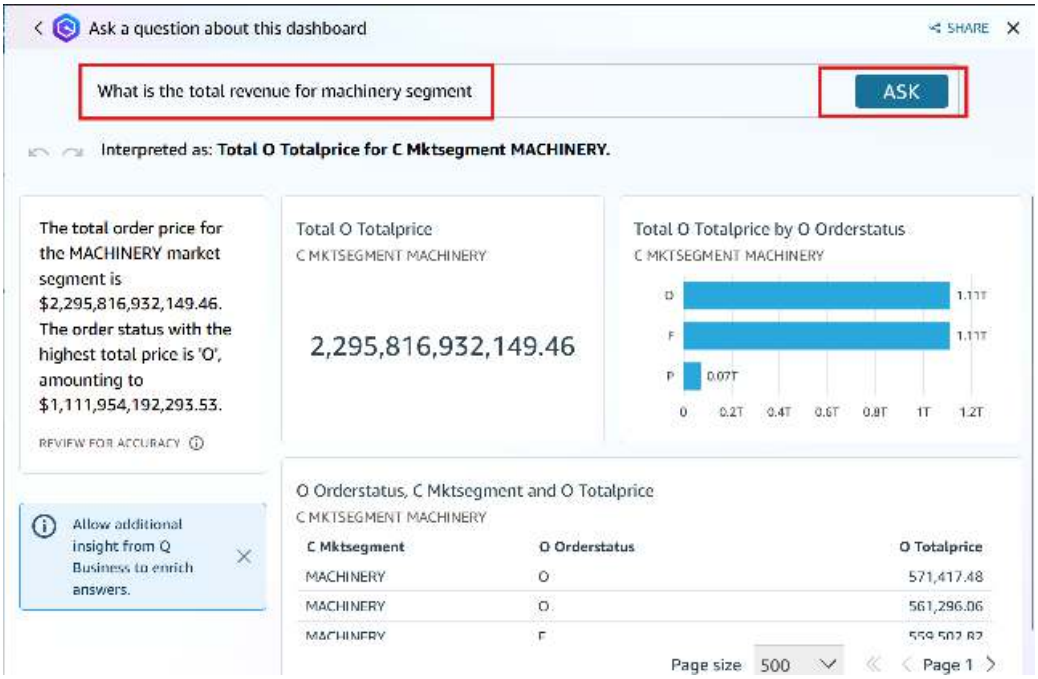


Figure 12.40 – QuickSight Q generated data answer

How it works...

Amazon Q integrates with Amazon QuickSight to give QuickSight users access to a suite of new generative BI capabilities. This allows QuickSight users to utilize the generative BI authoring experience, create executive summaries of their data, ask and answer questions of data, and generate data stories.

Appendix

Recipe 1: Creating an IAM user

You can use the following steps to create an IAM user:

1. Navigate to the **IAM** console.
2. Select **Users** and then click **Add user**.
3. Type a username for the new user. IAM usernames need to be unique in a single AWS account. This username will be used by the user to sign in to the AWS console.
4. For access type, select both **Programmatic access** and **AWS Management Console access**:
    - **Programmatic access** grants users access through the API, AWS CLI, or AWS Tools for PowerShell. An access key and a secret key are created for a user, and they are available to download on the final page.
    - **AWS Management Console access** grants users access through the AWS Management Console. A password is created for the user, and it is available to download on the final page.
5. For **Console password**, choose one of the following:
    - **Autogenerated password**: This will randomly generate a password for the user that complies with the account password policy.
    - **Custom password**: You can type a password that complies with the account password policy.
    - (Optional) You can select **Require password reset** to ensure that users are forced to change their password when they log in for the first time.

6. Select **Next: Permissions**.
7. Skip the **Set permissions** page and select **Next: Tags**.
8. Select **Next:Review**, and then select **Create user**.
9. This will generate the user's access keys (access key IDs and secret access keys) and password. Download the generated credentials by selecting the **Download .csv** and then save the file to a safe location.

   Share the credentials with users who need to access AWS services. This is an empty IAM user with no access to any AWS services. An AWS administrator will need to execute the CloudFormation template in some chapters to allow the appropriate access.

Recipe 2: Storing database credentials using AWS Secrets Manager

You can use the following steps to store database credentials using AWS Secrets Manager:

1. To create the secrets, navigate to the AWS Secrets Manager dashboard at `https://console.aws.amazon.com/secretsmanager/`.
2. Select **Store a new secret**.
3. Then, select **Credentials for Redshift Cluster**.
4. Specify the username and password.
5. Set `DefaultEncryptionKey` as the encryption key.
6. Select the Redshift cluster from the list that this secret will access, and click **Next**.
7. Specify the name for the secrets, keep the defaults, and click **Next**.
8. Keep the defaults for the configure automatic rotation, and click **Next**.
9. Review and click **Store**.
10. Capture the secret store ARN.

Recipe 3: Creating an IAM role for an AWS service

You can use the following steps to create an IAM role:

1. Navigate to the **IAM** console.
2. Select **Roles**, and then click **Create role**.
3. For **Select type of trusted entity**, choose **AWS service**.
4. For **Choose a use case**, select **Redshift**.

5. For **Select your use case**, choose **Redshift – Customizable** (allows the Redshift cluster to call AWS services on your behalf). Click **Next: Permissions**.
6. Skip **Create Policy**, click **Next: Tags**, and click **Next: Review**.
7. Provide a role name and click **Create role**. Note the role name to attach it to the Amazon Redshift cluster.

Recipe 4: Attaching an IAM role to the Amazon Redshift cluster

You can use the following steps to attach the IAM role to the Amazon Redshift cluster:

1. Navigate to the **Redshift** console.
2. Select **CLUSTERS** in the left navigation window.
3. Select the checkbox beside the Amazon Redshift cluster, and select **Actions**. From the dropdown, select **Manage IAM roles** under **Permissions**.

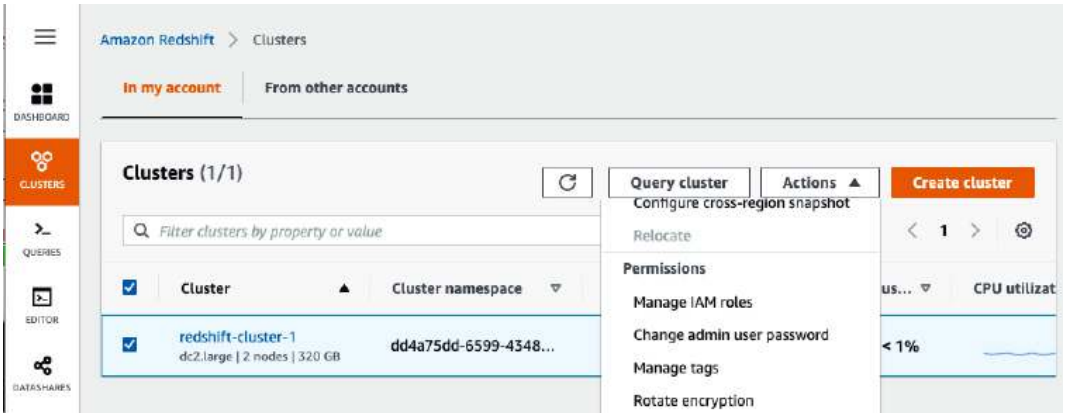


Figure 1: Managing an IAM role for an Amazon Redshift cluster

4. In the **Manage IAM roles** section, select the correct IAM role from the dropdown and click on **Associate IAM role**. Click on **Save changes**.

packtpub.com

Subscribe to our online digital library for full access to over 7,000 books and videos, as well as industry leading tools to help you plan your personal development and advance your career. For more information, please visit our website.

Why subscribe?

- Spend less time learning and more time coding with practical eBooks and Videos from over 4,000 industry professionals
- Improve your learning with Skill Plans built especially for you
- Get a free eBook or video every month
- Fully searchable for easy access to vital information
- Copy and paste, print, and bookmark content

At www.packt.com, you can also read a collection of free technical articles, sign up for a range of free newsletters, and receive exclusive discounts and offers on Packt books and eBooks.

Other Books You May Enjoy

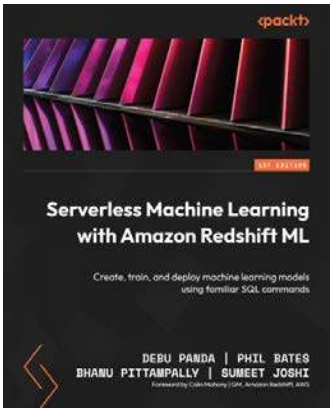
If you enjoyed this book, you may be interested in these other books by Packt:

Managing Data as a Product

Andrea Gioia

ISBN: 978-1-83546-853-1

- Overcome the challenges in scaling monolithic data platforms, including cognitive load, tech debt, and maintenance costs
- Discover the benefits of adopting a data-as-a-product approach for scalability and sustainability
- Navigate the complete data product lifecycle, from inception to decommissioning
- Automate data product lifecycle management using a self-serve platform
- Implement an incremental, value-driven strategy for transitioning to data-product-centric architectures
- Optimize data modeling in distributed environments to enhance GenAI-based use cases



Serverless Machine Learning with Amazon Redshift ML

Debu Panda, Phil Bates, Bhanu Pittampally, Sumeet Joshi

ISBN: 978-1-80461-928-5

- Utilize Redshift Serverless for data ingestion, data analysis, and machine learning
- Create supervised and unsupervised models and learn how to supply your own custom parameters
- Discover how to use time series forecasting in your data warehouse
- Create a SageMaker endpoint and use that to build a Redshift ML model for remote inference
- Find out how to operationalize machine learning in your data warehouse
- Use model explainability and calculate probabilities with Amazon Redshift ML

Packt is searching for authors like you

If you're interested in becoming an author for Packt, please visit `authors.packtpub.com` and apply today. We have worked with thousands of developers and tech professionals, just like you, to help them share their insight with the global tech community. You can make a general application, apply for a specific hot topic that we are recruiting an author for, or submit your own idea.

Share your thoughts

Now you've finished *Amazon Redshift Cookbook*, we'd love to hear your thoughts! If you purchased the book from Amazon, please click here to go straight to the Amazon review page for this book and share your feedback or leave a review on the site that you purchased it from.

Your review is important to us and the tech community and will help us make sure we're delivering excellent quality content.

Index

A

Amazon Bedrock
LLMs, using with SQL statements 391-401

Amazon Bedrock knowledge bases
used, for querying data with natural language prompts for Amazon Redshift 401-409

Amazon Data Firehose
used, for streaming data to Amazon Redshift 89-95

Amazon DataZone
used, for data sharing for cross-collaboration and self-service analytics 356-376

Amazon DynamoDB
data, loading from 65-67
zero-ETL integration, used for ingesting data from 110-117

Amazon EventBridge
used, for event-driven applications on Amazon Redshift provisioned clusters 150-160

Amazon Kinesis Data Streams (KDS)
streaming data, ingesting from 129-132

Amazon Managed Streaming
streaming data, ingesting for Apache Kafka (MSK) 133-136

Amazon Managed Workflows
used, for orchestration for Apache Airflow on provisioned clusters 172-179

Amazon provisioned cluster
cost controls, using to set actions for concurrency scaling 296-298

Amazon Q generative SQL
used, for building SQL queries automatically 384-387

Amazon RDS MySQL database
reference link 308

Amazon Redshift
Command Line (psql), used for connecting to 30-32
connecting, programmatically with Python 28-30
connecting, programmatically with Redshift Data API 28-30
database, managing 34-36
data, streaming via Amazon Data Firehose 89-95
data with natural language prompts, querying with Amazon Bedrock knowledge bases 401-409
Jupyter Notebook, used for connecting to 23-27
monitoring 202-206

SQL Workbench/J client, used for connecting to 20-23
used, for creating data lake export 324-328

Amazon Redshift Advisor
configuring, for provisioned clusters 243-245

Amazon Redshift dataset
querying 409-420

Amazon Redshift data warehouse
data sharing read access 350-353
data sharing write access 353, 354
registering, as federated source 338, 339

Amazon Redshift ML
managing 387-391

Amazon Redshift provisioned clusters
cost controls, using to set actions 293-296
creating, with AWS CloudFormation 12-16
creating, with AWS Console 6-8
elastic resizing, scheduling 289-292
event-driven applications, with Amazon EventBridge on 150-160
event-driven applications, with AWS Lambda on 160-165
orchestration, using AWS Step Functions on 165-171
pause and resume, scheduling 286-289
used, for managing superusers 194, 195

Amazon Redshift Query Editor V2 (QEV2)
used, for connecting to data warehouse 17-19
used, for scheduling queries 145-150

Amazon Redshift RA3
using 339-344

Amazon Redshift Reserved Instance
pricing 283-286

Amazon Redshift Serverless clusters
creating, with AWS CloudFormation 8-12
IAM authentication, using to generate database user credentials 195, 196

Amazon Redshift Serverless data warehouse
creating, with AWS Console 2-6

Amazon Redshift Spectrum
used, for extending data warehouse 328-330

Amazon S3
auto-copy, used for ingesting data from 136-142
COPY command, used for loading data from 58-64
data, unloading to 95, 96

Amazon SageMaker Lakehouse 337, 338
AWS Glue Data Catalog, using 346

Amazon Simple Notification Service (Amazon SNS) 150

Amazon Virtual Private Cloud (Amazon VPC) 183

Apache Kafka (MSK)
streaming data, ingesting from Amazon Managed Streaming 133-136

audit logs
managing 197-202

Aurora MySQL
zero-ETL integration, used for ingesting data from 100-110

Aurora Postgres
zero-ETL integration, used for ingesting data from 100-110

auto-copy
used, for ingesting data from Amazon S3 136-142

Automated Materialized Views (AutoMV)
features 48
AWS CloudFormation
used, for creating Amazon Redshift provisioned cluster 12-16
used, for creating Amazon Redshift Serverless cluster 8-12
AWS Console
used, for creating Amazon Redshift provisioned cluster 6-8
used, for creating Amazon Redshift Serverless data warehouse 2-6
AWS Data Exchange
used, for data sharing for monetization 377-381
used, for data sharing for subscribing to third-party data 377-381
AWS DMS
used, for ingesting data from transactional sources 74-83
AWS EC2 Linux 309
AWS Glue
used, for cataloging and ingesting data 84-89
AWS Glue Data Catalog
using, in Amazon SageMaker Lakehouse 346
AWS Lake Formation
used, for building data lake catalog 307-324
AWS Lambda
used, for event-driven applications on Amazon Redshift provisioned clusters 160-165
AWS Step Functions
used, for orchestration on provisioned clusters 165-171
AWS Trusted Advisor 280-282

B

Base Capacity 302
business intelligence (BI) 45, 383

C

column compression
managing 245-249
column-level security
customer table, using 229, 230
GRANT statements, using 230
implementing 229
REVOKE statements, using 230
Command Line (psql)
used, for connecting to Amazon Redshift 30-32
concurrency scaling
utilizing, for provisioned clusters 270-273
consumer Amazon Redshift data warehouse 350, 354
COPY command
used, for loading data from Amazon S3 58-64
cost controls
using, for Redshift Serverless 298-302
using, to set actions for concurrency scaling for Amazon provisioned cluster 296-298
using, to set actions for Redshift Spectrum 293-296

D

data
cataloging and ingesting, with AWS Glue 84-89
data, loading from Amazon S3 with COPY command 58-64

ingesting, from Amazon DynamoDB with zero-ETL integration 110-117
ingesting, from Amazon S3 with auto-copy 136-142
ingesting, from Aurora MySQL with zero-ETL integration 100-110
ingesting, from Aurora Postgres with zero-ETL integration 100-110
ingesting, from RDS MySQL with zero-ETL integration 100
ingesting, from SaaS application Salesforce with zero-ETL integration 117-128
ingesting, from transactional sources with AWS DMS 74-83
loading, from Amazon DynamoDB 65-67
streaming, to Amazon Redshift via Amazon Data Firehose 89-95
unloading, to Amazon S3 95, 96
updating and inserting 68-74

database
managing, in Amazon Redshift 34-36
materialized views, managing 45-48
schema, managing 36-38
stored procedures, managing 49-52
tables, managing 38-42
UDFs, managing 52-56
views, managing 43-45

database administrator (DBA) 49

Database Migration Service (DMS) 74

database user credentials
IAM authentication, using to generate for Amazon Redshift serverless clusters 195, 196

Data Definition Language (DDL) 49, 293

data distribution
managing 249-254

data encryption 188-190
in transit 191-193

data lake catalog
building, with AWS Lake Formation 307-324

data lake export
creating, from Amazon Redshift 324-328

data manipulation language (DML) 49

data sharing read access
across multiple Amazon Redshift data warehouses 350-353

data sharing, with Amazon DataZone
for cross-collaboration and self-service analytics 356-376

data sharing, with AWS Data Exchange
for monetization 377-381

data sharing write access
across multiple Amazon Redshift data warehouses 353-355

data warehouse
Amazon Redshift query editor v2, used for connecting to 17-19
extending, with Amazon Redshift Spectrum 328-330

data, with natural language prompts
querying, with Amazon Bedrock knowledge bases 409
querying, with Amazon Bedrock knowledge bases for Amazon Redshift 401-408

Directed Acyclic Graphs (DAGs) 172

dynamic data masking (DDM) 235
implementing 235-240
reference link 235

E

environment profiles 360

event-driven applications
with Amazon EventBridge, on Amazon Redshift provisioned clusters 150-165

Extensible Markup Language (XML) 49

Extract Load Transform (ELT) 68

Extract, Transform, and Load (ETL) 49, 143

F

federated query
used, for querying operational source 331-337

G

gigabytes (GB) 33

GRANT syntax
reference link 230

I

IAM authentication
using, to generate database user credentials for Amazon Redshift serverless clusters 195, 196

IAM Identity Center
used, for single sign-on 206-218

Identity and Access Management (IAM) 167

infrastructure security
managing 183-187

input/output (I/O) 42

J

Jupyter Notebook
used, for connecting to Amazon Redshift 23-27

K

Key Management Service (KMS) 55

L

large language models (LLMs) 383
using, in Amazon Bedrock with SQL statements 391-395
using, in Amazon SageMaker Jumpstart with SQL statements 395-401

M

machine learning (ML) 245, 307, 383

Managed Workflows for Apache Airflow (MWAA) 172, 173

massively parallel processing (MPP) 39, 64

Max RPU-hours 303

MaxRPU (Max Capacity) 303

metadata security 219-222

mutual Transport Layer Security (mTLS) 135

MySQL command line 309

MySQL engine 308

O

online analytical processing (OLAP) 33

operational source
querying, with federated query 331-337

orchestration
with Amazon Managed Workflows for Apache Airflow on provisioned clusters 172-179
with AWS Step Functions, on provisioned clusters 165-171

P

payment card information (PCI) 229

personally identifiable information (PII) 44, 229

petabyte (PB) 33

Procedural Language/PostgreSQL (PL/pgSQL) 49

producer Amazon Redshift provisioned cluster 350-354

provisioned clusters
Amazon Redshift Advisor, configuring 243-245
concurrency scaling, utilizing 270-273
queries, analyzing and improving 261-264
Spectrum queries, optimizing 274-277
Workload Management (WLM), configuring 265-269

psycog2 library
URL 165

Python
used, for connecting to Amazon Redshift programmatically 28-30

Q

queries
scheduling, with Amazon Redshift Query Editor V2 (QEV2) 145-150

query data
registering, with Amazon Athena 338, 339
registering, with Amazon Redshift 338, 339

R

RDS MySQL 309
zero-ETL integration, used for ingesting data from 100-110

RDS PostgreSQL cluster
reference link 331

Redshift Data API
used, for connecting to Amazon Redshift programmatically 28-30

Redshift managed storage (RMS) 305

Redshift Serverless
cost controls, using 298-302

REVOKE syntax
reference link 230

role-based access control (RBAC) 223
admin database role, creating 224-228
implementing 224
read-only database role, creating 224-227
read-write database role, creating 224-227
role inheritance 228

row-level security (RLS)
implementing 231-234

S

software-as-a-service (SaaS) 117

sort key
managing 254-260

Spectrum queries
optimizing, for provisioned clusters 274-277

SQL queries
building, automatically with Amazon Q generative SQL 384-387

SQL statements
features 395
used, for using LLMs in Amazon Bedrock 391-395
used, for using LLMs in Amazon SageMaker Jumpstart 395-401

SQL Workbench/J client
used, for connecting to Amazon Redshift 20-23

star schema benchmark (SSB) 59

streaming data
ingesting, from Amazon Kinesis Data Streams (KDS) 129-132
ingesting, from Amazon Managed Streaming for Apache Kafka (MSK) 133-136

Structured Query Language (SQL) 33

superusers
managing, with Amazon Redshift provisioned cluster 194, 195

T

transactional sources
AWS DMS, used for ingesting data from 74-83

U

User-Defined Functions (UDFs) 240, 353
managing, in database 52-56

W

Workload Management (WLM)
configuring, for provisioned clusters 265-269

Z

zero-ETL integration
used, for ingesting data from Amazon DynamoDB 110-117
used, for ingesting data from Aurora MySQL 100-110
used, for ingesting data from Aurora Postgres 100-110
used, for ingesting data from RDS MySQL 100-110
used, for ingesting data from SaaS application Salesforce 117-128

Download a free PDF copy of this book

Thanks for purchasing this book!

Do you like to read on the go but are unable to carry your print books everywhere?

Is your eBook purchase not compatible with the device of your choice?

Don’t worry, now with every Packt book you get a DRM-free PDF version of that book at no cost.

Read anywhere, any place, on any device. Search, copy, and paste code from your favorite technical books directly into your application.

The perks don’t stop there, you can get exclusive access to discounts, newsletters, and great free content in your inbox daily.

Follow these simple steps to get the benefits:

1. Scan the QR code or visit the link below:

`https://packt.link/free-ebook/9781836206910`

2. Submit your proof of purchase.
3. That’s it! We’ll send your free PDF and other benefits to your email directly.