Chapter 7
Databases and AWS

THE AWS CERTIFIED SOLUTIONS ARCHITECT ASSOCIATE EXAM OBJECTIVES COVERED IN THIS CHAPTER MAY INCLUDE, BUT ARE NOT LIMITED TO, THE FOLLOWING:

Domain 1.0: Designing highly available, cost-efficient, fault-tolerant, and scalable systems

✓ **1.1 Identify and recognize cloud architecture considerations, such as fundamental components and effective designs.**

Content may include the following:

- Planning and design
- Architectural trade-off decisions (Amazon Relational Database Service [Amazon RDS] vs. installing on Amazon Elastic Compute Cloud [Amazon EC2])
- Best practices for AWS architecture
- Recovery Time Objective (RTO) and Recovery Point Objective (RPO) Disaster Recovery (DR) design
- Elasticity and scalability

Domain 3.0: Data Security

✓ **3.1 Recognize and implement secure practices for optimum cloud deployment and maintenance.**

Content may include the following:

- AWS administration and security services
- Design patterns

✓ **3.2 Recognize critical disaster recovery techniques and their implementation.**

This chapter will cover essential database concepts and introduce three of Amazon's managed database services: Amazon Relational Database Service (Amazon RDS), Amazon DynamoDB, and Amazon Redshift. These managed services simplify the setup and operation of relational databases, NoSQL databases, and data warehouses.

This chapter focuses on key topics you need to understand for the exam, including:

- The differences among a relational database, a NoSQL database, and a data warehouse
- The benefits and tradeoffs between running a database on Amazon EC2 or on Amazon RDS
- How to deploy database engines into the cloud
- How to back up and recover your database and meet your Recovery Point Objective (RPO) and Recovery Time Objective (RTO) requirements
- How to build highly available database architectures
- How to scale your database compute and storage vertically
- How to select the right type of storage volume
- How to use read replicas to scale horizontally
- How to design and scale an Amazon DynamoDB table
- How to read and write from an Amazon DynamoDB table
- How to use secondary indexes to speed queries
- How to design an Amazon Redshift table
- How to load and query an Amazon Redshift data warehouse
- How to secure your databases, tables, and clusters

Database Primer

Almost every application relies on a database to store important data and records for its users. A database engine allows your application to access, manage, and search large volumes of data records. In a well-architected application, the database will need to meet the performance demands, the availability needs, and the recoverability characteristics of the system.

Database systems and engines can be grouped into two broad categories: Relational Database Management Systems (RDBMS) and NoSQL (or non-relational) databases. It is not uncommon to build an application using a combination of RDBMS and NoSQL databases. A strong understanding of essential database concepts, Amazon RDS, and Amazon DynamoDB are required to pass this exam.

Relational Databases

The most common type of database in use today is the *relational database*. The relational database has roots going back to the 1970s when Edgar F. Codd, working for IBM, developed the concepts of the relational model. Today, relational databases power all types of applications from social media apps, e-commerce websites, and blogs to complex enterprise applications. Commonly used relational database software packages include MySQL, PostgreSQL, Microsoft SQL Server, and Oracle.

Relational databases provide a common interface that lets users read and write from the database using commands or queries written using *Structured Query Language (SQL)*. A relational database consists of one or more tables, and a table consists of columns and rows similar to a spreadsheet. A database column contains a specific attribute of the record, such as a person's name, address, and telephone number. Each attribute is assigned a data type such as text, number, or date, and the database engine will reject invalid inputs.

A database row comprises an individual record, such as the details about a student who attends a school. Consider the example in Table 7.1.

TABLE 7.1 Students Table

StudentID	FirstName	LastName	Gender	Age
1001	Joe	Dusty	M	29
1002	Andrea	Romanov	F	20
1003	Ben	Johnson	M	30
1004	Beth	Roberts	F	30

This is an example of a basic table that would sit in a relational database. There are five fields with different data types:

`StudentID` = Number or integer

`FirstName` = String

`LastName` = String

`Gender` = String (Character Length = 1)

`Age` = Integer

This sample table has four records, with each record representing an individual student. Each student has a `StudentID` field, which is usually a unique number per student. A unique number that identifies each student can be called a *primary key*.

One record in a table can relate to a record in another table by referencing the primary key of a record. This pointer or reference is called a foreign key. For example, the Grades table that records scores for each student would have its own primary key and an additional column known as a foreign key that refers to the primary key of the student record. By referencing the primary keys of other tables, relational databases minimize duplication of data in associated tables. With relational databases, it is important to note that the structure of the table (such as the number of columns and data type of each column) must be defined prior to data being added to the table.

A relational database can be categorized as either an *Online Transaction Processing (OLTP)* or *Online Analytical Processing (OLAP)* database system, depending on how the tables are organized and how the application uses the relational database. OLTP refers to transaction-oriented applications that are frequently writing and changing data (for example, data entry and e-commerce). OLAP is typically the domain of data warehouses and refers to reporting or analyzing large data sets. Large applications often have a mix of both OLTP and OLAP databases.

Amazon Relational Database Service (Amazon RDS) significantly simplifies the setup and maintenance of OLTP and OLAP databases. Amazon RDS provides support for six popular relational database engines: MySQL, Oracle, PostgreSQL, Microsoft SQL Server, MariaDB, and Amazon Aurora. You can also choose to run nearly any database engine using Windows or Linux Amazon Elastic Compute Cloud (Amazon EC2) instances and manage the installation and administration yourself.

Data Warehouses

A *data warehouse* is a central repository for data that can come from one or more sources. This data repository is often a specialized type of relational database that can be used for reporting and analysis via OLAP. Organizations typically use data warehouses to compile reports and search the database using highly complex queries.

Data warehouses are also typically updated on a batch schedule multiple times per day or per hour, compared to an OLTP relational database that can be updated thousands of times per second. Many organizations split their relational databases into two different databases: one database as their main production database for OLTP transactions, and the other database as their data warehouse for OLAP. OLTP transactions occur frequently and are relatively simple. OLAP transactions occur much less frequently but are much more complex.

Amazon RDS is often used for OLTP workloads, but it can also be used for OLAP. *Amazon Redshift* is a high-performance data warehouse designed specifically for OLAP use cases. It is also common to combine Amazon RDS with Amazon Redshift in the same application and periodically extract recent transactions and load them into a reporting database.

NoSQL Databases

NoSQL databases have gained significant popularity in recent years because they are often simpler to use, more flexible, and can achieve performance levels that are difficult or impossible with traditional relational databases. Traditional relational databases are difficult to scale beyond a single server without significant engineering and cost, but a NoSQL architecture allows for horizontal scalability on commodity hardware.

NoSQL databases are non-relational and do not have the same table and column semantics of a relational database. NoSQL databases are instead often key/value stores or document stores with flexible schemas that can evolve over time or vary. Contrast that to a relational database, which requires a very rigid schema.

Many of the concepts of NoSQL architectures trace their foundational concepts back to whitepapers published in 2006 and 2007 that described distributed systems like Dynamo at Amazon. Today, many application teams use Hbase, MongoDB, Cassandra, CouchDB, Riak, and *Amazon DynamoDB* to store large volumes of data with high transaction rates. Many of these database engines support clustering and scale horizontally across many machines for performance and fault tolerance. A common use case for NoSQL is managing user session state, user profiles, shopping cart data, or time-series data.

You can run any type of NoSQL database on AWS using Amazon EC2, or you can choose a managed service like Amazon DynamoDB to deal with the heavy lifting involved with building a distributed cluster spanning multiple data centers.

Amazon Relational Database Service (Amazon RDS)

Amazon RDS is a service that simplifies the setup, operations, and scaling of a relational database on AWS. With Amazon RDS, you can spend more time focusing on the application and the schema and let Amazon RDS offload common tasks like backups, patching, scaling, and replication.

Amazon RDS helps you to streamline the installation of the database software and also the provisioning of infrastructure capacity. Within a few minutes, Amazon RDS can launch one of many popular database engines that is ready to start taking SQL transactions. After the initial launch, Amazon RDS simplifies ongoing maintenance by automating common administrative tasks on a recurring basis.

With Amazon RDS, you can accelerate your development timelines and establish a consistent operating model for managing relational databases. For example, Amazon RDS makes it easy to replicate your data to increase availability, improve durability, or scale up or beyond a single database instance for read-heavy database workloads.

Amazon RDS exposes a database endpoint to which client software can connect and execute SQL. Amazon RDS does not provide shell access to Database (DB) Instances, and it restricts access to certain system procedures and tables that require advanced privileges. With Amazon RDS, you can typically use the same tools to query, analyze, modify, and administer the database. For example, current Extract, Transform, Load (ETL) tools and reporting tools can connect to Amazon RDS databases in the same way with the same drivers, and often all it takes to reconfigure is changing the hostname in the connection string.

Database (DB) Instances

The Amazon RDS service itself provides an Application Programming Interface (API) that lets you create and manage one or more *DB Instances*. A DB Instance is an isolated database environment deployed in your private network segments in the cloud. Each DB Instance runs and manages a popular commercial or open source database engine on your behalf. Amazon RDS currently supports the following database engines: MySQL, PostgreSQL, MariaDB, Oracle, SQL Server, and Amazon Aurora.

You can launch a new DB Instance by calling the `CreateDBInstance` API or by using the AWS Management Console. Existing DB Instances can be changed or resized using the `ModifyDBInstance` API. A DB Instance can contain multiple different databases, all of which you create and manage within the DB Instance itself by executing SQL commands with the Amazon RDS endpoint. The different databases can be created, accessed, and managed using the same SQL client tools and applications that you use today.

The compute and memory resources of a DB Instance are determined by its DB Instance class. You can select the DB Instance class that best meets your needs for compute and memory. The range of DB Instance classes extends from a db.t2.micro with 1 virtual CPU (vCPU) and 1 GB of memory, up to a db.r3.8xlarge with 32 vCPUs and 244 GB of memory. As your needs change over time, you can change the instance class and the balance of compute of memory, and Amazon RDS will migrate your data to a larger or smaller instance class. Independent from the DB Instance class that you select, you can also control the size and

performance characteristics of the storage used.

Amazon RDS supports a large variety of engines, versions, and feature combinations. Check the Amazon RDS documentation to determine support for specific features. Many features and common configuration settings are exposed and managed using *DB parameter groups* and *DB option groups*. A DB parameter group acts as a container for engine configuration values that can be applied to one or more DB Instances. You may change the DB parameter group for an existing instance, but a reboot is required. A DB option group acts as a container for engine features, which is empty by default. In order to enable specific features of a DB engine (for example, Oracle Statspack, Microsoft SQL Server Mirroring), you create a new DB option group and configure the settings accordingly.


Existing databases can be migrated to Amazon RDS using native tools and techniques that vary depending on the engine. For example with MySQL, you can export a backup using mysqldump and import the file into Amazon RDS MySQL. You can also use the AWS Database Migration Service, which gives you a graphical interface that simplifies the migration of both schema and data between databases. AWS Database Migration Service also helps convert databases from one database engine to another.

Operational Benefits

Amazon RDS increases the operational reliability of your databases by applying a very consistent deployment and operational model. This level of consistency is achieved in part by limiting the types of changes that can be made to the underlying infrastructure and through the extensive use of automation. For example with Amazon RDS, you cannot use Secure Shell (SSH) to log in to the host instance and install a custom piece of software. You can, however, connect using SQL administrator tools or use DB option groups and DB parameter groups to change the behavior or feature configuration for a DB Instance. If you want full control of the Operating System (OS) or require elevated permissions to run, then consider installing your database on Amazon EC2 instead of Amazon RDS.

Amazon RDS is designed to simplify the common tasks required to operate a relational database in a reliable manner. It's useful to compare the responsibilities of an administrator when operating a relational database in your data center, on Amazon EC2, or with Amazon RDS (see Table 7.2).

TABLE 7.2 Comparison of Operational Responsibilities

Responsibility	Database On-Premise	Database on Amazon EC2	Database on Amazon RDS
App Optimization	You	You	You
Scaling	You	You	AWS
High Availability	You	You	AWS
Backups	You	You	AWS
DB Engine Patches	You	You	AWS
Software Installation	You	You	AWS
OS Patches	You	You	AWS
OS Installation	You	AWS	AWS
Server Maintenance	You	AWS	AWS
Rack and Stack	You	AWS	AWS
Power and Cooling	You	AWS	AWS

Database Engines

Amazon RDS supports six database engines: MySQL, PostgreSQL, MariaDB, Oracle, SQL Server, and Amazon Aurora. Features and capabilities vary slightly depending on the engine that you select.

MySQL

MySQL is one of the most popular open source databases in the world, and it is used to power a wide range of applications, from small personal blogs to some of the largest websites in the world. As of the time of this writing, Amazon RDS for MySQL currently supports MySQL 5.7, 5.6, 5.5, and 5.1. The engine is running the open source Community Edition with InnoDB as the default and recommended database storage engine. Amazon RDS MySQL allows you to connect using standard MySQL tools such as MySQL Workbench or SQL Workbench/J. Amazon RDS MySQL supports *Multi-AZ* deployments for high availability and *read replicas* for horizontal scaling.

PostgreSQL

PostgreSQL is a widely used open source database engine with a very rich set of features and advanced functionality. Amazon RDS supports DB Instances running several versions of PostgreSQL. As of the time of this writing, Amazon RDS supports multiple releases of PostgreSQL, including 9.5.x, 9.4.x, and 9.3.x. Amazon RDS PostgreSQL can be managed using standard tools like pgAdmin and supports standard JDBC/ODBC drivers. Amazon RDS PostgreSQL also supports Multi-AZ deployment for high availability and read replicas for

horizontal scaling.

MariaDB

Amazon RDS recently added support for DB Instances running MariaDB. MariaDB is a popular open source database engine built by the creators of MySQL and enhanced with enterprise tools and functionality. MariaDB adds features that enhance the performance, availability, and scalability of MySQL. As of the time of this writing, AWS supports MariaDB version 10.0.17. Amazon RDS fully supports the XtraDB storage engine for MariaDB DB Instances and, like Amazon RDS MySQL and PostgreSQL, has support for Multi-AZ deployment and read replicas.

Oracle

Oracle is one of the most popular relational databases used in the enterprise and is fully supported by Amazon RDS. As of the time of this writing, Amazon RDS supports DB Instances running several editions of Oracle 11g and Oracle 12c. Amazon RDS supports access to schemas on a DB Instance using any standard SQL client application, such as Oracle SQL Plus.

Amazon RDS Oracle supports three different editions of the popular database engine: Standard Edition One, Standard Edition, and Enterprise Edition. Table 7.3 outlines some of the major differences between editions:

TABLE 7.3 Amazon RDS Oracle Editions Compared

Edition	Performance	Multi-AZ	Encryption
Standard One	++++	Yes	KMS
Standard	++++++++	Yes	KMS
Enterprise	++++++++	Yes	KMS and TDE

Microsoft SQL Server

Microsoft SQL Server is another very popular relational database used in the enterprise. Amazon RDS allows Database Administrators (DBAs) to connect to their SQL Server DB Instance in the cloud using native tools like SQL Server Management Studio. As of the time of this writing, Amazon RDS provides support for several versions of Microsoft SQL Server, including SQL Server 2008 R2, SQL Server 2012, and SQL Server 2014.

Amazon RDS SQL Server also supports four different editions of SQL Server: Express Edition, Web Edition, Standard Edition, and Enterprise Edition. Table 7.4 highlights the relative performance, availability, and encryption differences among these editions.

TABLE 7.4 Amazon RDS SQL Server Editions Compared

Edition	Performance	Multi-AZ	Encryption
Express	+	No	KMS
Web	++++	No	KMS
Standard	++++	Yes	KMS
Enterprise	++++++++	Yes	KMS and TDE

Licensing

Amazon RDS Oracle and Microsoft SQL Server are commercial software products that require appropriate licenses to operate in the cloud. AWS offers two licensing models: *License Included* and *Bring Your Own License (BYOL)*.

License Included In the License Included model, the license is held by AWS and is included in the Amazon RDS instance price. For Oracle, License Included provides licensing for Standard Edition One. For SQL Server, License Included provides licensing for SQL Server Express Edition, Web Edition, and Standard Edition.

Bring Your Own License (BYOL) In the BYOL model, you provide your own license. For Oracle, you must have the appropriate Oracle Database license for the DB Instance class and Oracle Database edition you want to run. You can bring over Standard Edition One, Standard Edition, and Enterprise Edition.

For SQL Server, you provide your own license under the Microsoft License Mobility program. You can bring over Microsoft SQL Standard Edition and also Enterprise Edition. You are responsible for tracking and managing how licenses are allocated.

Amazon Aurora

Amazon Aurora offers enterprise-grade commercial database technology while offering the simplicity and cost effectiveness of an open source database. This is achieved by redesigning the internal components of MySQL to take a more service-oriented approach.

Like other Amazon RDS engines, Amazon Aurora is a fully managed service, is MySQL-compatible out of the box, and provides for increased reliability and performance over standard MySQL deployments. Amazon Aurora can deliver up to five times the performance of MySQL without requiring changes to most of your existing web applications. You can use the same code, tools, and applications that you use with your existing MySQL databases with Amazon Aurora.

When you first create an Amazon Aurora instance, you create a DB cluster. A DB cluster has one or more instances and includes a cluster volume that manages the data for those instances. An Amazon Aurora cluster volume is a virtual database storage volume that spans multiple Availability Zones, with each Availability Zone having a copy of the cluster data. An Amazon Aurora DB cluster consists of two different types of instances:

Primary Instance This is the main instance, which supports both read and write workloads. When you modify your data, you are modifying the primary instance. Each Amazon Aurora DB cluster has one primary instance.

Amazon Aurora Replica This is a secondary instance that supports only read operations. Each DB cluster can have up to 15 Amazon Aurora Replicas in addition to the primary instance. By using multiple Amazon Aurora Replicas, you can distribute the read workload among various instances, increasing performance. You can also locate your Amazon Aurora Replicas in multiple Availability Zones to increase your database availability.

Storage Options

Amazon RDS is built using Amazon Elastic Block Store (Amazon EBS) and allows you to select the right storage option based on your performance and cost requirements. Depending on the database engine and workload, you can scale up to 4 to 6TB in provisioned storage and up to 30,000 IOPS. Amazon RDS supports three storage types: Magnetic, General Purpose (Solid State Drive [SSD]), and Provisioned IOPS (SSD). Table 7.5 highlights the relative size, performance, and cost differences between types.

TABLE 7.5 Amazon RDS Storage Types

	Magnetic	General Purpose (SSD)	Provisioned IOPS (SSD)
Size	+++	+++++	+++++
Performance	+	+++	+++++
Cost	++	+++	+++++

Magnetic Magnetic storage, also called standard storage, offers cost-effective storage that is ideal for applications with light I/O requirements.

General Purpose (SSD) General purpose (SSD)-backed storage, also called gp2, can provide faster access than magnetic storage. This storage type can provide burst performance to meet spikes and is excellent for small- to medium-sized databases.

Provisioned IOPS (SSD) Provisioned IOPS (SSD) storage is designed to meet the needs of I/O-intensive workloads, particularly database workloads, that are sensitive to storage performance and consistency in random access I/O throughput.

For most applications, General Purpose (SSD) is the best option and provides a good mix of lower-cost and higher-performance characteristics.

Backup and Recovery

Amazon RDS provides a consistent operational model for backup and recovery procedures across the different database engines. Amazon RDS provides two mechanisms for backing up the database: automated backups and manual snapshots. By using a combination of both techniques, you can design a backup recovery model to protect your application data.

Each organization typically will define a *Recovery Point Objective (RPO)* and *Recovery Time Objective (RTO)* for important applications based on the criticality of the application and the expectations of the users. It's common for enterprise systems to have an RPO measured in minutes and an RTO measured in hours or even days, while some critical applications may have much lower tolerances.

RPO is defined as the maximum period of data loss that is acceptable in the event of a failure or incident. For example, many systems back up transaction logs every 15 minutes to allow them to minimize data loss in the event of an accidental deletion or hardware failure.

RTO is defined as the maximum amount of downtime that is permitted to recover from backup and to resume processing. For large databases in particular, it can take hours to restore from a full backup. In the event of a hardware failure, you can reduce your RTO to minutes by failing over to a secondary node. You should create a recovery plan that, at a minimum, lets you recover from a recent backup.

Automated Backups

An *automated backup* is an Amazon RDS feature that continuously tracks changes and backs up your database. Amazon RDS creates a storage volume snapshot of your DB Instance, backing up the entire DB Instance and not just individual databases. You can set the backup retention period when you create a DB Instance. One day of backups will be retained by default, but you can modify the retention period up to a maximum of 35 days. Keep in mind that when you delete a DB Instance, all automated backup snapshots are deleted and cannot be recovered. Manual snapshots, however, are not deleted.

Automated backups will occur daily during a configurable 30-minute maintenance window called the backup window. Automated backups are kept for a configurable number of days, called the *backup retention period*. You can restore your DB Instance to any specific time during this retention period, creating a new DB Instance.

Manual DB Snapshots

In addition to automated backups, you can perform manual *DB snapshots* at any time. A DB snapshot is initiated by you and can be created as frequently as you want. You can then restore the DB Instance to the specific state in the DB snapshot at any time. DB snapshots can be created with the Amazon RDS console or the `CreateDBSnapshot` action. Unlike automated snapshots that are deleted after the retention period, manual DB snapshots are kept until you explicitly delete them with the Amazon RDS console or the `DeleteDBSnapshot` action.

For busy databases, use Multi-AZ to minimize the performance impact of a snapshot. During the backup window, storage I/O may be suspended while your data is being backed up, and you may experience elevated latency. This I/O suspension typically lasts for the duration of the snapshot. This period of I/O suspension is shorter for Multi-AZ DB deployments because the backup is taken from the standby, but latency can occur during the backup process.

Recovery

Amazon RDS allows you to recover your database quickly whether you are performing automated backups or manual DB snapshots. You cannot restore from a DB snapshot to an existing DB Instance; a new DB Instance is created when you restore. When you restore a DB Instance, only the default DB parameter and security groups are associated with the restored

instance. As soon as the restore is complete, you should associate any custom DB parameter or security groups used by the instance from which you restored. When using automated backups, Amazon RDS combines the daily backups performed during your predefined maintenance window in conjunction with transaction logs to enable you to restore your DB Instance to any point during your retention period, typically up to the last five minutes.

High Availability with Multi-AZ

One of the most powerful features of Amazon RDS is Multi-AZ deployments, which allows you to create a database cluster across multiple Availability Zones. Setting up a relational database to run in a highly available and fault-tolerant fashion is a challenging task. With Amazon RDS Multi-AZ, you can reduce the complexity involved with this common administrative task; with a single option, Amazon RDS can increase the availability of your database using replication. Multi-AZ lets you meet the most demanding RPO and RTO targets by using synchronous replication to minimize RPO and fast failover to minimize RTO to minutes.

Multi-AZ allows you to place a secondary copy of your database in another Availability Zone for disaster recovery purposes. Multi-AZ deployments are available for all types of Amazon RDS database engines. When you create a Multi-AZ DB Instance, a primary instance is created in one Availability Zone and a secondary instance is created in another Availability Zone. You are assigned a database instance endpoint such as the following:

```
my_app_db.ch6fe7ykq1zd.us-west-2.rds.amazonaws.com
```

This endpoint is a Domain Name System (DNS) name that AWS takes responsibility for resolving to a specific IP address. You use this DNS name when creating the connection to your database. Figure 7.1 illustrates a typical Multi-AZ deployment spanning two Availability Zones.

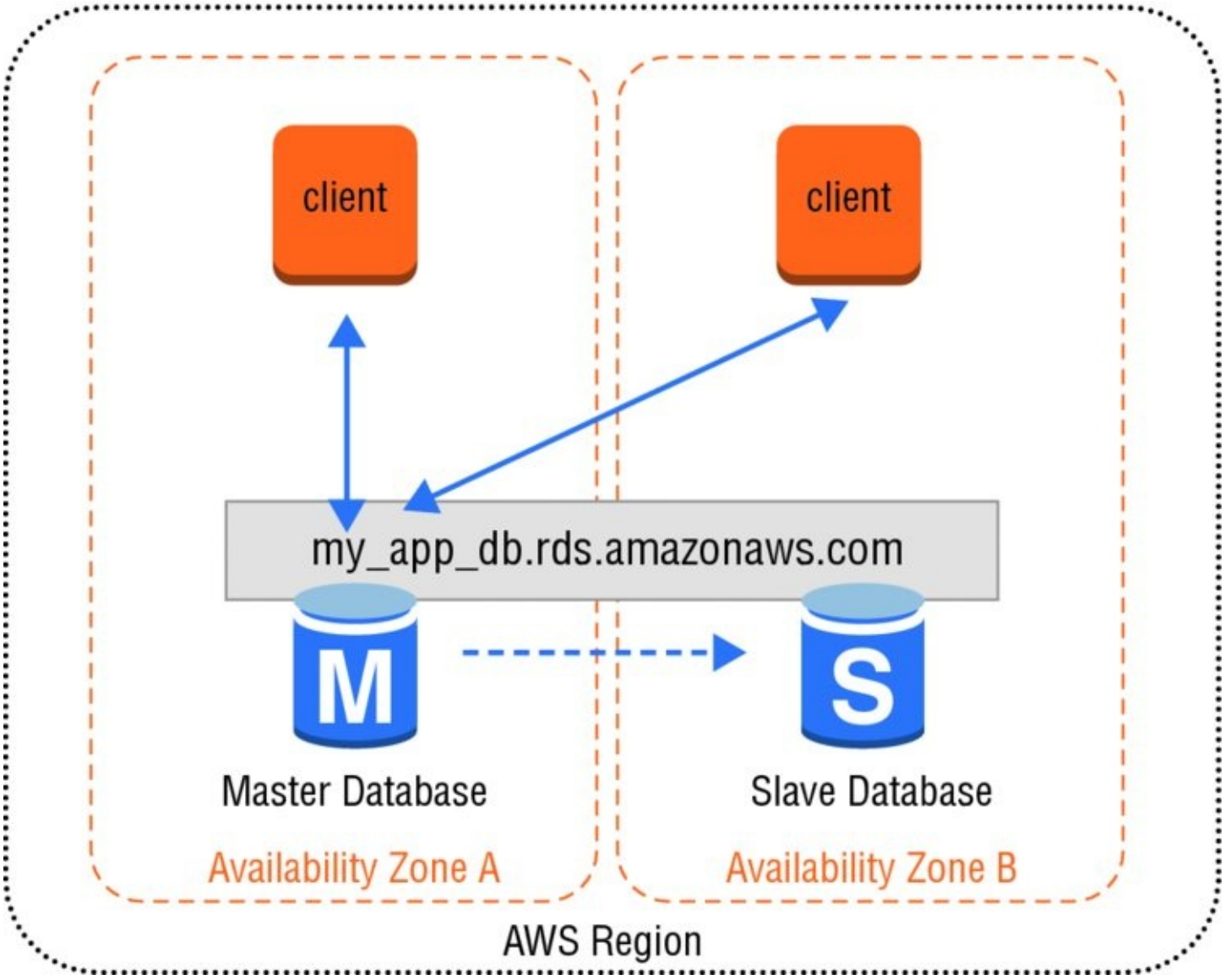

FIGURE 7.1 Multi-AZ Amazon RDS architecture

Amazon RDS automatically replicates the data from the master database or primary instance to the slave database or secondary instance using synchronous replication. Each Availability Zone runs on its own physically distinct, independent infrastructure and is engineered to be highly reliable. Amazon RDS detects and automatically recovers from the most common failure scenarios for Multi-AZ deployments so that you can resume database operations as quickly as possible without administrative intervention. Amazon RDS automatically performs a failover in the event of any of the following:

- Loss of availability in primary Availability Zone
- Loss of network connectivity to primary database
- Compute unit failure on primary database
- Storage failure on primary database

Amazon RDS will automatically fail over to the standby instance without user intervention. The DNS name remains the same, but the Amazon RDS service changes the CNAME to point to the standby. The primary DB Instance switches over automatically to the standby replica if there was an Availability Zone service disruption, if the primary DB Instance fails, or if the instance type is changed. You can also perform a manual failover of the DB Instance. Failover

between the primary and the secondary instance is fast, and the time automatic failover takes to complete is typically one to two minutes.

 It is important to remember that Multi-AZ deployments are for disaster recovery only; they are not meant to enhance database performance. The standby DB Instance is not available to offline queries from the primary master DB Instance. To improve database performance using multiple DB Instances, use read replicas or other DB caching technologies such as Amazon ElastiCache.

Scaling Up and Out

As the number of transactions increase to a relational database, scaling up, or vertically, by getting a larger machine allows you to process more reads and writes. Scaling out, or horizontally, is also possible, but it is often more difficult. Amazon RDS allows you to scale compute and storage vertically, and for some DB engines, you can scale horizontally.

Vertical Scalability

Adding additional compute, memory, or storage resources to your database allows you to process more transactions, run more queries, and store more data. Amazon RDS makes it easy to scale up or down your database tier to meet the demands of your application. Changes can be scheduled to occur during the next maintenance window or to begin immediately using the `ModifyDBInstance` action.

To change the amount of compute and memory, you can select a different DB Instance class of the database. After you select a larger or smaller DB Instance class, Amazon RDS automates the migration process to a new class with only a short disruption and minimal effort.

You can also increase the amount of storage, the storage class, and the storage performance for an Amazon RDS Instance. Each database instance can scale from 5GB up to 6TB in provisioned storage depending on the storage type and engine. Storage for Amazon RDS can be increased over time as needs grow with minimal impact to the running database. Storage expansion is supported for all of the database engines except for SQL Server.

Horizontal Scalability with Partitioning

A relational database can be scaled vertically only so much before you reach the maximum instance size. Partitioning a large relational database into multiple instances or shards is a common technique for handling more requests beyond the capabilities of a single instance.

Partitioning, or *sharding*, allows you to scale horizontally to handle more users and requests but requires additional logic in the application layer. The application needs to decide how to route database requests to the correct shard and becomes limited in the types of queries that can be performed across server boundaries. NoSQL databases like Amazon DynamoDB or Cassandra are designed to scale horizontally.

Horizontal Scalability with Read Replicas

Another important scaling technique is to use *read replicas* to offload read transactions from the primary database and increase the overall number of transactions. Amazon RDS supports read replicas that allow you to scale out elastically beyond the capacity constraints of a single DB Instance for read-heavy database workloads.

There are a variety of use cases where deploying one or more read replica DB Instances is helpful. Some common scenarios include:

- Scale beyond the capacity of a single DB Instance for read-heavy workloads.
- Handle read traffic while the source DB Instance is unavailable. For example, due to I/O suspension for backups or scheduled maintenance, you can direct read traffic to a replica.
- Offload reporting or data warehousing scenarios against a replica instead of the primary DB Instance.

For example, a blogging website may have very little write activity except for the occasional comment, and the vast majority of database activity will be read-only. By offloading some or all of the read activity to one or more read replicas, the primary database instance can focus on handling the writes and replicating the data out to the replicas.

Read replicas are currently supported in Amazon RDS for MySQL, PostgreSQL, MariaDB, and Amazon Aurora. Amazon RDS uses the MySQL, MariaDB, and PostgreSQL DB engines' built-in replication functionality to create a special type of DB Instance, called a read replica, from a source DB Instance. Updates made to the source DB Instance are asynchronously copied to the read replica. You can reduce the load on your source DB Instance by routing read queries from your applications to the read replica.

You can create one or more replicas of a database within a single AWS Region or across multiple AWS Regions. To enhance your disaster recovery capabilities or reduce global latencies, you can use cross-region read replicas to serve read traffic from a region closest to your global users or migrate your databases across AWS Regions.

Security

Securing your Amazon RDS DB Instances and relational databases requires a comprehensive plan that addresses the many layers commonly found in database-driven systems. This includes the infrastructure resources, the database, and the network.

Protect access to your infrastructure resources using AWS Identity and Access Management (IAM) policies that limit which actions AWS administrators can perform. For example, some key administrator actions that can be controlled in IAM include `CreateDBInstance` and `DeleteDBInstance`.

Another security best practice is to deploy your Amazon RDS DB Instances into a private subnet within an Amazon Virtual Private Cloud (Amazon VPC) that limits network access to the DB Instance. Before you can deploy into an Amazon VPC, you must first create a *DB subnet group* that predefines which subnets are available for Amazon RDS deployments. Further, restrict network access using network Access Control Lists (ACLs) and security groups to limit inbound traffic to a short list of source IP addresses.

At the database level, you will also need to create users and grant them permissions to read and write to your databases. Access to the database is controlled using the database engine-specific access control and user management mechanisms. Create users at the database level with strong passwords that you rotate frequently.

Finally, protect the confidentiality of your data in transit and at rest with multiple encryption capabilities provided with Amazon RDS. Security features vary slightly from one engine to another, but all engines support some form of in-transit encryption and also at-rest encryption. You can securely connect a client to a running DB Instance using Secure Sockets Layer (SSL) to protect data in transit. Encryption at rest is possible for all engines using the Amazon Key Management Service (KMS) or *Transparent Data Encryption (TDE)*. All logs, backups, and snapshots are encrypted for an encrypted Amazon RDS instance.

Amazon Redshift

Amazon Redshift is a fast, powerful, fully managed, petabyte-scale data warehouse service in the cloud. Amazon Redshift is a relational database designed for OLAP scenarios and optimized for high-performance analysis and reporting of very large datasets. Traditional data warehouses are difficult and expensive to manage, especially for large datasets. Amazon Redshift not only significantly lowers the cost of a data warehouse, but it also makes it easy to analyze large amounts of data very quickly.

Amazon Redshift gives you fast querying capabilities over structured data using standard SQL commands to support interactive querying over large datasets. With connectivity via ODBC or JDBC, Amazon Redshift integrates well with various data loading, reporting, data mining, and analytics tools. Amazon Redshift is based on industry-standard PostgreSQL, so most existing SQL client applications will work with only minimal changes.

Amazon Redshift manages the work needed to set up, operate, and scale a data warehouse, from provisioning the infrastructure capacity to automating ongoing administrative tasks such as backups and patching. Amazon Redshift automatically monitors your nodes and drives to help you recover from failures.

Clusters and Nodes

The key component of an Amazon Redshift data warehouse is a *cluster*. A cluster is composed of a leader node and one or more compute nodes. The client application interacts directly only with the leader node, and the compute nodes are transparent to external applications.

Amazon Redshift currently has support for six different node types and each has a different mix of CPU, memory, and storage. The six node types are grouped into two categories: Dense Compute and Dense Storage. The Dense Compute node types support clusters up to 326TB using fast SSDs, while the Dense Storage nodes support clusters up to 2PB using large magnetic disks. Each cluster consists of one leader node and one or more compute nodes. Figure 7.2 shows the internal components of an Amazon Redshift data warehouse cluster.

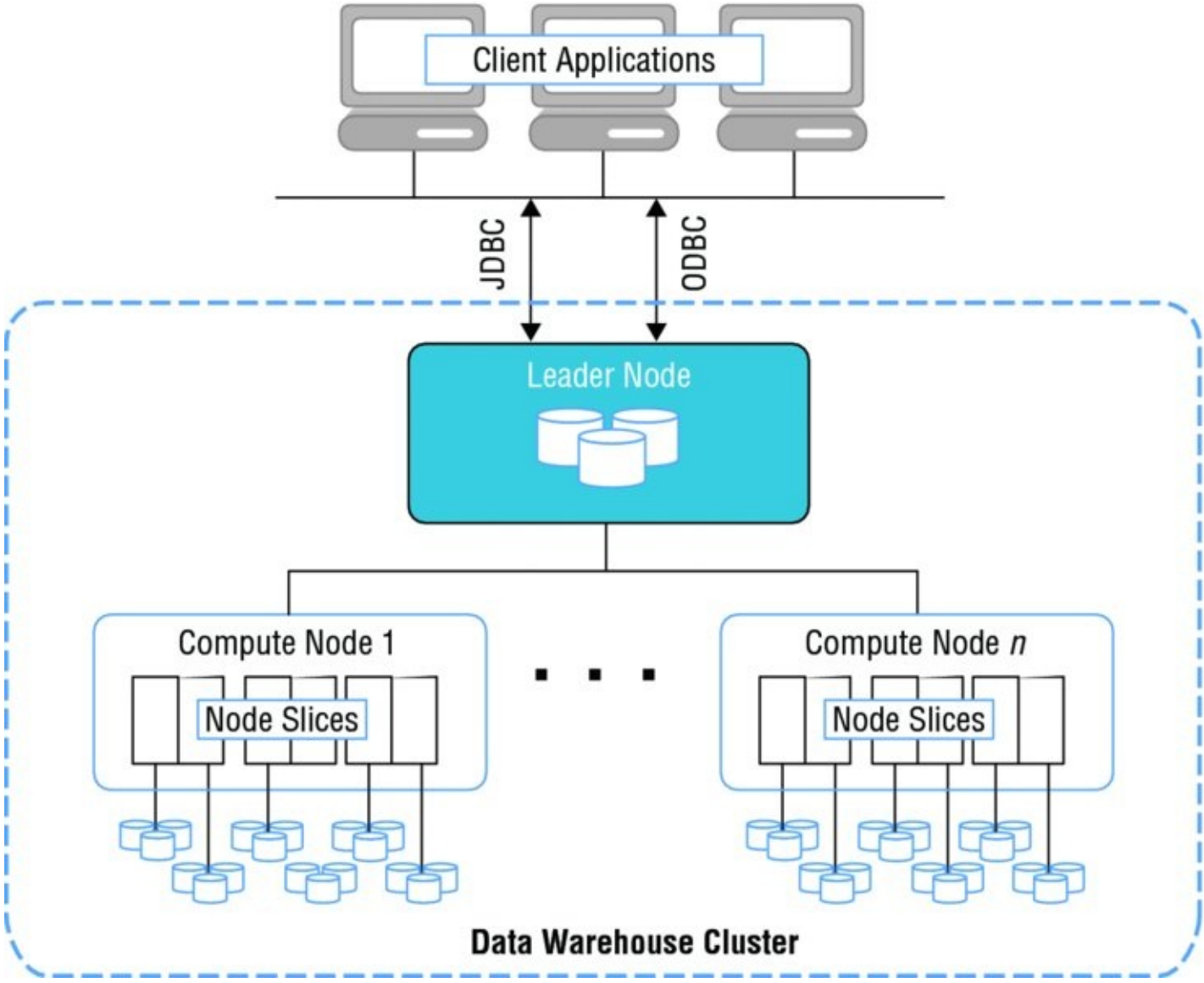

FIGURE 7.2 Amazon Redshift cluster architecture

Each cluster contains one or more databases. User data for each table is distributed across the compute nodes. Your application or SQL client communicates with Amazon Redshift using standard JDBC or ODBC connections with the leader node, which in turn coordinates query execution with the compute nodes. Your application does not interact directly with the compute nodes.

The disk storage for a compute node is divided into a number of slices. The number of slices per node depends on the node size of the cluster and typically varies between 2 and 16. The nodes all participate in parallel query execution, working on data that is distributed as evenly as possible across the slices.

You can increase query performance by adding multiple nodes to a cluster. When you submit a query, Amazon Redshift distributes and executes the query in parallel across all of a cluster's compute nodes. Amazon Redshift also spreads your table data across all compute nodes in a cluster based on a *distribution strategy* that you specify. This partitioning of data across multiple compute resources allows you to achieve high levels of performance.

Amazon Redshift allows you to resize a cluster to add storage and compute capacity over time as your needs evolve. You can also change the node type of a cluster and keep the overall size the same. Whenever you perform a resize operation, Amazon Redshift will create a new

cluster and migrate data from the old cluster to the new one. During a resize operation, the database will become read-only until the operation is finished.

Table Design

Each Amazon Redshift cluster can support one or more databases, and each database can contain many tables. Like most SQL-based databases, you can create a table using the `CREATE TABLE` command. This command specifies the name of the table, the columns, and their data types. In addition to columns and data types, the Amazon Redshift `CREATE TABLE` command also supports specifying compression encodings, distribution strategy, and sort keys.

Data Types

Amazon Redshift columns support a wide range of data types. This includes common numeric data types like INTEGER, DECIMAL, and DOUBLE, text data types like CHAR and VARCHAR, and date data types like DATE and TIMESTAMP. Additional columns can be added to a table using the `ALTER TABLE` command; however, existing columns cannot be modified.

Compression Encoding

One of the key performance optimizations used by Amazon Redshift is data compression. When loading data for the first time into an empty table, Amazon Redshift will automatically sample your data and select the best compression scheme for each column. Alternatively, you can specify compression encoding on a per-column basis as part of the `CREATE TABLE` command.

Distribution Strategy

One of the primary decisions when creating a table in Amazon Redshift is how to distribute the records across the nodes and slices in a cluster. You can configure the distribution style of a table to give Amazon Redshift hints as to how the data should be partitioned to best meet your query patterns. When you run a query, the optimizer shifts the rows to the compute nodes as needed to perform any joins and aggregates. The goal in selecting a table distribution style is to minimize the impact of the redistribution step by putting the data where it needs to be before the query is performed.

The data distribution style that you select for your database has a big impact on query performance, storage requirements, data loading, and maintenance. By choosing the best distribution strategy for each table, you can balance your data distribution and significantly improve overall system performance. When creating a table, you can choose between one of three distribution styles: EVEN, KEY, or ALL.

EVEN distribution This is the default option and results in the data being distributed across the slices in a uniform fashion regardless of the data.

KEY distribution With KEY distribution, the rows are distributed according to the values in one column. The leader node will store matching values close together and increase query performance for joins.

ALL distribution With ALL, a full copy of the entire table is distributed to every node. This is useful for lookup tables and other large tables that are not updated frequently.

Sort Keys

Another important decision to make during the creation of a table is whether to specify one or more columns as sort keys. Sorting enables efficient handling of range-restricted predicates. If a query uses a range-restricted predicate, the query processor can rapidly skip over large numbers of blocks during table scans.

The sort keys for a table can be either compound or interleaved. A compound sort key is more efficient when query predicates use a prefix, which is a subset of the sort key columns in order. An interleaved sort key gives equal weight to each column in the sort key, so query predicates can use any subset of the columns that make up the sort key, in any order.

Loading Data

Amazon Redshift supports standard SQL commands like `INSERT` and `UPDATE` to create and modify records in a table. For bulk operations, however, Amazon Redshift provides the `COPY` command as a much more efficient alternative than repeatedly calling `INSERT`.

A `COPY` command can load data into a table in the most efficient manner, and it supports multiple types of input data sources. The fastest way to load data into Amazon Redshift is doing bulk data loads from flat files stored in an Amazon Simple Storage Service (Amazon S3) bucket or from an Amazon DynamoDB table.

When loading data from Amazon S3, the `COPY` command can read from multiple files at the same time. Amazon Redshift can distribute the workload to the nodes and perform the load process in parallel. Instead of having one single large file with your data, you can enable parallel processing by having a cluster with multiple nodes and multiple input files.

After each bulk data load that modifies a significant amount of data, you will need to perform a `VACUUM` command to reorganize your data and reclaim space after deletes. It is also recommended to run an `ANALYZE` command to update table statistics.

Data can also be exported out of Amazon Redshift using the `UNLOAD` command. This command can be used to generate delimited text files and store them in Amazon S3.

Querying Data

Amazon Redshift allows you to write standard SQL commands to query your tables. By supporting commands like `SELECT` to query and join tables, analysts can quickly become productive using Amazon Redshift or integrate it easily. For complex queries, you can analyze the query plan to better optimize your access pattern. You can monitor the performance of the cluster and specific queries using Amazon CloudWatch and the Amazon Redshift web console.

For large Amazon Redshift clusters supporting many users, you can configure Workload Management (WLM) to queue and prioritize queries. WLM allows you define multiple queues and set the concurrency level for each queue. For example, you might want to have one queue set up for long-running queries and limit the concurrency and another queue for short-running queries and allow higher levels of concurrency.

Snapshots

Similar to Amazon RDS, you can create point-in-time snapshots of your Amazon Redshift cluster. A snapshot can then be used to restore a copy or create a clone of your original Amazon Redshift cluster. Snapshots are durably stored internally in Amazon S3 by Amazon Redshift.

Amazon Redshift supports both automated snapshots and manual snapshots. With automated snapshots, Amazon Redshift will periodically take snapshots of your cluster and keep a copy for a configurable retention period. You can also perform manual snapshots and share them across regions or even with other AWS accounts. Manual snapshots are retained until you explicitly delete them.

Security

Securing your Amazon Redshift cluster is similar to securing other databases running in the cloud. Your security plan should include controls to protect the infrastructure resources, the database schema, the records in the table, and network access. By addressing security at every level, you can securely operate an Amazon Redshift data warehouse in the cloud.

The first layer of security comes at the infrastructure level using IAM policies that limit the actions AWS administrators can perform. With IAM, you can create policies that grant other AWS users the permission to create and manage the lifecycle of a cluster, including scaling, backup, and recovery operations.

At the network level, Amazon Redshift clusters can be deployed within the private IP address space of your Amazon VPC to restrict overall network connectivity. Fine-grained network access can be further restricted using security groups and network ACLs at the subnet level.

In addition to controlling infrastructure access at the infrastructure level, you must protect access at the database level. When you initially create an Amazon Redshift cluster, you will create a master user account and password. The master account can be used to log in to the Amazon Redshift database and to create more users and groups. Each database user can be granted permission to schemas, tables, and other database objects. These permissions are independent from the IAM policies used to control access to the infrastructure resources and the Amazon Redshift cluster configuration.

Protecting the data stored in Amazon Redshift is another important aspect of your security design. Amazon Redshift supports encryption of data in transit using SSL-encrypted connections, and also encryption of data at rest using multiple techniques. To encrypt data at rest, Amazon Redshift integrates with KMS and AWS CloudHSM for encryption key management services. Encryption at rest and in transit assists in meeting compliance requirements, such as for the Health Insurance Portability and Accountability Act (HIPAA) or the Payment Card Industry Data Security Standard (PCI DSS), and provides additional protections for your data.

Amazon DynamoDB

Amazon DynamoDB is a fully managed NoSQL database service that provides fast and low-latency performance that scales with ease. Amazon DynamoDB lets you offload the administrative burdens of operating a distributed NoSQL database and focus on the application. Amazon DynamoDB significantly simplifies the hardware provisioning, setup and configuration, replication, software patching, and cluster scaling of NoSQL databases.

Amazon DynamoDB is designed to simplify database and cluster management, provide consistently high levels of performance, simplify scalability tasks, and improve reliability with automatic replication. Developers can create a table in Amazon DynamoDB and write an unlimited number of items with consistent latency.

Amazon DynamoDB can provide consistent performance levels by automatically distributing the data and traffic for a table over multiple partitions. After you configure a certain read or write capacity, Amazon DynamoDB will automatically add enough infrastructure capacity to support the requested throughput levels. As your demand changes over time, you can adjust the read or write capacity after a table has been created, and Amazon DynamoDB will add or remove infrastructure and adjust the internal partitioning accordingly.

To help maintain consistent, fast performance levels, all table data is stored on high-performance SSD disk drives. Performance metrics, including transactions rates, can be monitored using Amazon CloudWatch. In addition to providing high-performance levels, Amazon DynamoDB also provides automatic high-availability and durability protections by replicating data across multiple Availability Zones within an AWS Region.

Data Model

The basic components of the Amazon DynamoDB data model include *tables*, items, and attributes. As depicted in Figure 7.3, a table is a collection of items and each item is a collection of one or more attributes. Each item also has a primary key that uniquely identifies the item.

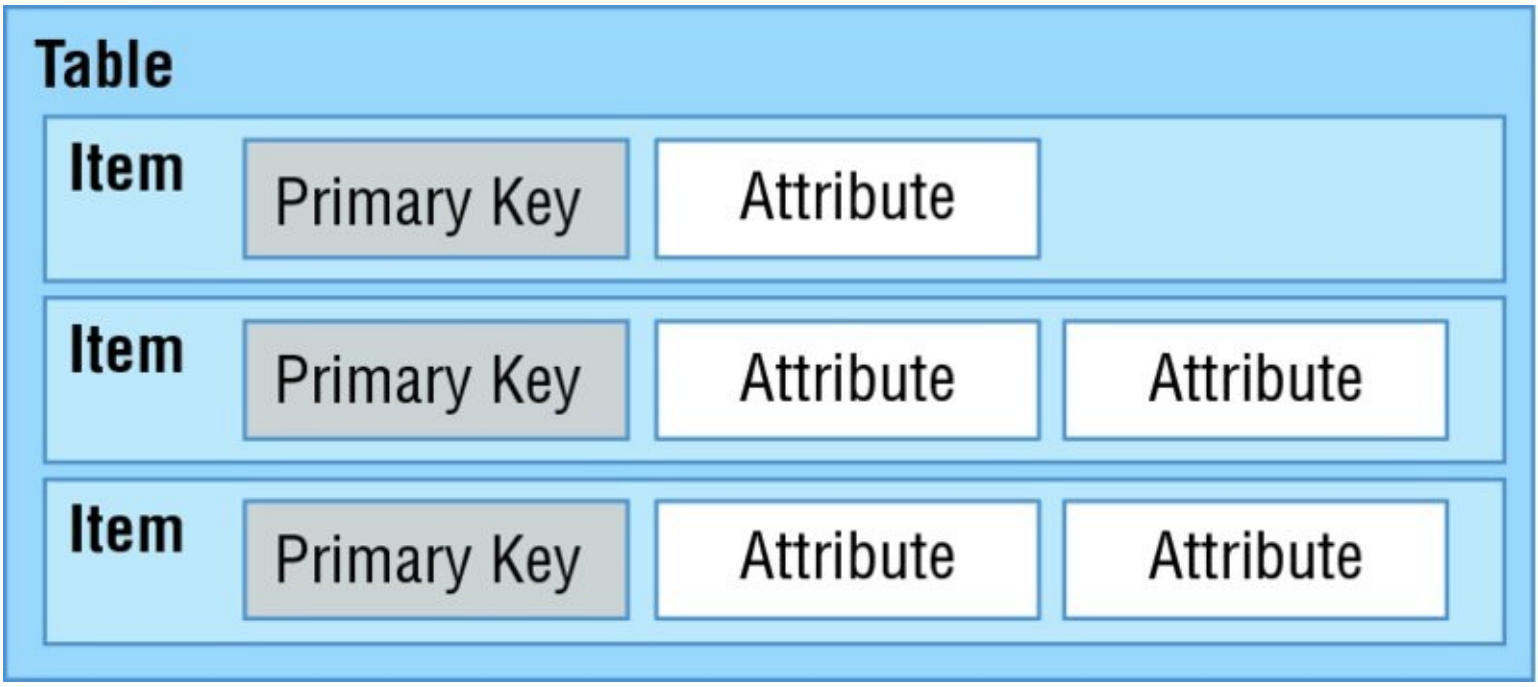

FIGURE 7.3 Table, items, attributes relationship

In a relational database, a table has a predefined schema such as the table name, primary key, list of its column names, and their data types. All records stored in the table must have the same set of columns. In contrast, Amazon DynamoDB only requires that a table have a primary key, but it does not require you to define all of the attribute names and data types in advance. Individual items in an Amazon DynamoDB table can have any number of attributes, although there is a limit of 400KB on the item size.

Each attribute in an item is a name/value pair. An attribute can be a single-valued or multi-valued set. For example, a book item can have title and authors attributes. Each book has one title but can have many authors. The multi-valued attribute is a set; duplicate values are not allowed. Data is stored in Amazon DynamoDB in key/value pairs such as the following:

```
{
   Id = 101
   ProductName = "Book 101 Title"
   ISBN = "123-1234567890"
   Authors = [ "Author 1", "Author 2" ]
   Price = 2.88
   Dimensions = "8.5 x 11.0 x 0.5"
   PageCount = 500
   InPublication = 1
   ProductCategory = "Book"
}
```

Applications can connect to the Amazon DynamoDB service endpoint and submit requests over HTTP/S to read and write items to a table or even to create and delete tables. DynamoDB provides a web service API that accepts requests in JSON format. While you could program directly against the web service API endpoints, most developers choose to use the AWS Software Development Kit (SDK) to interact with their items and tables. The AWS SDK is available in many different languages and provides a simplified, high-level programming interface.

Data Types

Amazon DynamoDB gives you a lot of flexibility with your database schema. Unlike a traditional relational database that requires you to define your column types ahead of time, DynamoDB only requires a primary key attribute. Each item that is added to the table can then add additional attributes. This gives you flexibility over time to expand your schema without having to rebuild the entire table and deal with record version differences with application logic.

When you create a table or a secondary index, you must specify the names and data types of each primary key attribute (partition key and sort key). Amazon DynamoDB supports a wide range of data types for attributes. Data types fall into three major categories: Scalar, Set, or Document.

Scalar Data Types A scalar type represents exactly one value. Amazon DynamoDB supports the following five scalar types:

String Text and variable length characters up to 400KB. Supports Unicode with UTF8 encoding

Number Positive or negative number with up to 38 digits of precision

Binary Binary data, images, compressed objects up to 400KB in size

Boolean Binary flag representing a true or false value

Null Represents a blank, empty, or unknown state. String, Number, Binary, Boolean cannot be empty.

Set Data Types Sets are useful to represent a unique list of one or more scalar values. Each value in a set needs to be unique and must be the same data type. Sets do not guarantee order. Amazon DynamoDB supports three set types: String Set, Number Set, and Binary Set.

String Set Unique list of String attributes

Number Set Unique list of Number attributes

Binary Set Unique list of Binary attributes

Document Data Types Document type is useful to represent multiple nested attributes, similar to the structure of a JSON file. Amazon DynamoDB supports two document types: List and Map. Multiple Lists and Maps can be combined and nested to create complex structures.

List Each List can be used to store an ordered list of attributes of different data types.

Map Each Map can be used to store an unordered list of key/value pairs. Maps can be used to represent the structure of any JSON object.

Primary Key

When you create a table, you must specify the primary key of the table in addition to the table name. Like a relational database, the primary key uniquely identifies each item in the table. A primary key will point to exactly one item. Amazon DynamoDB supports two types of primary keys, and this configuration cannot be changed after a table has been created:

Partition Key The primary key is made of one attribute, a partition (or hash) key. Amazon DynamoDB builds an unordered hash index on this primary key attribute.

Partition and Sort Key The primary key is made of two attributes. The first attribute is the partition key and the second one is the sort (or range) key. Each item in the table is uniquely identified by the combination of its partition and sort key values. It is possible for two items to have the same partition key value, but those two items must have different sort key values.

Furthermore, each primary key attribute must be defined as type string, number, or binary. Amazon DynamoDB uses the partition key to distribute the request to the right partition.

 If you are performing many reads or writes per second on the same primary key, you will not be able to fully use the compute capacity of the Amazon DynamoDB cluster. A best practice is to maximize your throughput by distributing requests across the full range of partition keys.

Provisioned Capacity

When you create an Amazon DynamoDB table, you are required to provision a certain amount of read and write capacity to handle your expected workloads. Based on your configuration settings, DynamoDB will then provision the right amount of infrastructure capacity to meet your requirements with sustained, low-latency response times. Overall capacity is measured in read and write capacity units. These values can later be scaled up or down by using an `UpdateTable` action.

Each operation against an Amazon DynamoDB table will consume some of the provisioned capacity units. The specific amount of capacity units consumed depends largely on the size of the item, but also on other factors. For read operations, the amount of capacity consumed also depends on the read consistency selected in the request. Read more about eventual and strong consistency later in this chapter.

For example, given a table without a local secondary index, you will consume 1 capacity unit if you read an item that is 4KB or smaller. Similarly, for write operations you will consume 1 capacity unit if you write an item that is 1KB or smaller. This means that if you read an item that is 110KB, you will consume 28 capacity units, or 110 / 4 = 27.5 rounded up to 28. For read operations that are strongly consistent, they will use twice the number of capacity units, or 56 in this example.

You can use Amazon CloudWatch to monitor your Amazon DynamoDB capacity and make scaling decisions. There is a rich set of metrics, including `ConsumedReadCapacityUnits` and `ConsumedWriteCapacityUnits`. If you do exceed your provisioned capacity for a period of time, requests will be throttled and can be retried later. You can monitor and alert on the `ThrottledRequests` metric using Amazon CloudWatch to notify you of changing usage patterns.

Secondary Indexes

When you create a table with a partition and sort key (formerly known as a hash and range key), you can optionally define one or more secondary indexes on that table. A secondary index lets you query the data in the table using an alternate key, in addition to queries against the primary key. Amazon DynamoDB supports two different kinds of indexes:

Global Secondary Index The *global secondary index* is an index with a partition and sort key that can be different from those on the table. You can create or delete a global secondary index on a table at any time.

Local Secondary Index The *local secondary index* is an index that has the same partition key attribute as the primary key of the table, but a different sort key. You can only create a local secondary index when you create a table.

Secondary indexes allow you to search a large table efficiently and avoid an expensive scan operation to find items with specific attributes. These indexes allow you to support different query access patterns and use cases beyond what is possible with only a primary key. While a table can only have one local secondary index, you can have multiple global secondary indexes.

Amazon DynamoDB updates each secondary index when an item is modified. These updates consume write capacity units. For a local secondary index, item updates will consume write capacity units from the main table, while global secondary indexes maintain their own provisioned throughput settings separate from the table.

Writing and Reading Data

After you create a table with a primary key and indexes, you can begin writing and reading items to the table. Amazon DynamoDB provides multiple operations that let you create, update, and delete individual items. Amazon DynamoDB also provides multiple querying options that let you search a table or an index or retrieve back a specific item or a batch of items.

Writing Items

Amazon DynamoDB provides three primary API actions to create, update, and delete items: `PutItem`, `UpdateItem`, and `DeleteItem`. Using the `PutItem` action, you can create a new item with one or more attributes. Calls to `PutItem` will update an existing item if the primary key already exists. `PutItem` only requires a table name and a primary key; any additional attributes are optional.

The `UpdateItem` action will find existing items based on the primary key and replace the attributes. This operation can be useful to only update a single attribute and leave the other attributes unchanged. `UpdateItem` can also be used to create items if they don't already exist. Finally, you can remove an item from a table by using `DeleteItem` and specifying a specific primary key.

The `UpdateItem` action also provides support for atomic counters. *Atomic counters* allow you to increment and decrement a value and are guaranteed to be consistent across multiple concurrent requests. For example, a counter attribute used to track the overall score of a mobile game can be updated by many clients at the same time.

These three actions also support conditional expressions that allow you to perform validation before an action is applied. For example, you can apply a conditional expression on `PutItem` that checks that certain conditions are met before the item is created. This can be useful to prevent accidental overwrites or to enforce some type of business logic checks.

Reading Items

After an item has been created, it can be retrieved through a direct lookup by calling the `GetItem` action or through a search using the Query or Scan action. `GetItem` allows you to retrieve an item based on its primary key. All of the item's attributes are returned by default, and you have the option to select individual attributes to filter down the results.

If a primary key is composed of a partition key, the entire partition key needs to be specified to retrieve the item. If the primary key is a composite of a partition key and a sort key, `GetItem` will require both the partition and sort key as well. Each call to `GetItem` consumes read capacity units based on the size of the item and the consistency option selected.

By default, a `GetItem` operation performs an eventually consistent read. You can optionally request a strongly consistent read instead; this will consume additional read capacity units, but it will return the most up-to-date version of the item.

Eventual Consistency

When reading items from Amazon DynamoDB, the operation can be either eventually consistent or strongly consistent. Amazon DynamoDB is a distributed system that stores multiple copies of an item across an AWS Region to provide high availability and increased durability. When an item is updated in Amazon DynamoDB, it starts replicating across multiple servers. Because Amazon DynamoDB is a distributed system, the replication can take some time to complete. Because of this we refer to the data as being eventually consistent, meaning that a read request immediately after a write operation might not show the latest change. In some cases, the application needs to guarantee that the data is the latest and Amazon DynamoDB offers an option for strongly consistent reads.

Eventually Consistent Reads When you read data, the response might not reflect the results of a recently completed write operation. The response might include some stale data. Consistency across all copies of the data is usually reached within a second; if you repeat your read request after a short time, the response returns the latest data.

Strongly Consistent Reads When you issue a strongly consistent read request, Amazon DynamoDB returns a response with the most up-to-date data that reflects updates by all prior related write operations to which Amazon DynamoDB returned a successful response. A strongly consistent read might be less available in the case of a network delay or outage. You can request a strongly consistent read result by specifying optional parameters in your request.

Batch Operations

Amazon DynamoDB also provides several operations designed for working with large batches of items, including `BatchGetItem` and `BatchWriteItem`. Using the `BatchWriteItem` action, you can perform up to 25 item creates or updates with a single operation. This allows you to minimize the overhead of each individual call when processing large numbers of items.

Searching Items

Amazon DynamoDB also gives you two operations, Query and Scan, that can be used to search a table or an index. A Query operation is the primary search operation you can use to find items in a table or a secondary index using only primary key attribute values. Each Query requires a partition key attribute name and a distinct value to search. You can optionally

provide a sort key value and use a comparison operator to refine the search results. Results are automatically sorted by the primary key and are limited to 1MB.

In contrast to a Query, a Scan operation will read every item in a table or a secondary index. By default, a Scan operation returns all of the data attributes for every item in the table or index. Each request can return up to 1MB of data. Items can be filtered out using expressions, but this can be a resource-intensive operation. If the result set for a Query or a Scan exceeds 1MB, you can page through the results in 1MB increments.


For most operations, performing a Query operation instead of a Scan operation will be the most efficient option. Performing a Scan operation will result in a full scan of the entire table or secondary index, then it filters out values to provide the desired result. Use a Query operation when possible and avoid a Scan on a large table or index for only a small number of items.

Scaling and Partitioning

Amazon DynamoDB is a fully managed service that abstracts away most of the complexity involved in building and scaling a NoSQL cluster. You can create tables that can scale up to hold a virtually unlimited number of items with consistent low-latency performance. An Amazon DynamoDB table can scale horizontally through the use of partitions to meet the storage and performance requirements of your application. Each individual partition represents a unit of compute and storage capacity. A well-designed application will take the partition structure of a table into account to distribute read and write transactions evenly and achieve high transaction rates at low latencies.

Amazon DynamoDB stores items for a single table across multiple partitions, as represented in Figure 7.4. Amazon DynamoDB decides which partition to store the item in based on the partition key. The partition key is used to distribute the new item among all of the available partitions, and items with the same partition key will be stored on the same partition.

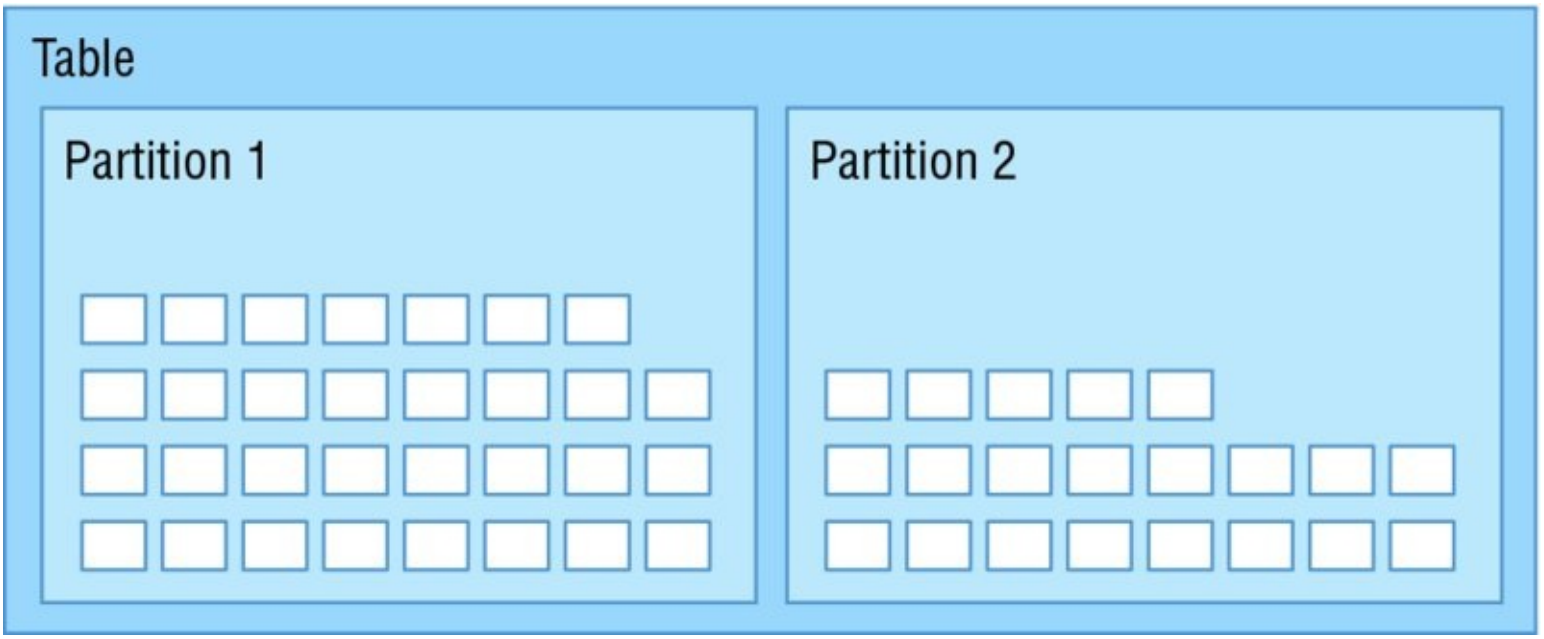

FIGURE 7.4 Table partitioning

As the number of items in a table grows, additional partitions can be added by splitting an existing partition. The provisioned throughput configured for a table is also divided evenly among the partitions. Provisioned throughput allocated to a partition is entirely dedicated to that partition, and there is no sharing of provisioned throughput across partitions.

When a table is created, Amazon DynamoDB configures the table's partitions based on the desired read and write capacity. One single partition can hold about 10GB of data and supports a maximum of 3,000 read capacity units or 1,000 write capacity units. For partitions that are not fully using their provisioned capacity, Amazon DynamoDB provides some burst capacity to handle spikes in traffic. A portion of your unused capacity will be reserved to handle bursts for short periods.

As storage or capacity requirements change, Amazon DynamoDB can split a partition to accommodate more data or higher provisioned request rates. After a partition is split, however, it cannot be merged back together. Keep this in mind when planning to increase provisioned capacity temporarily and then lower it again. With each additional partition added, its share of the provisioned capacity is reduced.

To achieve the full amount of request throughput provisioned for a table, keep your workload spread evenly across the partition key values. Distributing requests across partition key values distributes the requests across partitions. For example, if a table has 10,000 read capacity units configured but all of the traffic is hitting one partition key, you will not be able to get more than the 3,000 maximum read capacity units that one partition can support.

To maximize Amazon DynamoDB throughput, create tables with a partition key that has a large number of distinct values and ensure that the values are requested fairly uniformly. Adding a random element that can be calculated or hashed is one common technique to improve partition distribution.

Security

Amazon DynamoDB gives you granular control over the access rights and permissions for users and administrators. Amazon DynamoDB integrates with the IAM service to provide strong control over permissions using policies. You can create one or more policies that allow or deny specific operations on specific tables. You can also use conditions to restrict access to individual items or attributes.

All operations must first be authenticated as a valid user or user session. Applications that need to read and write from Amazon DynamoDB need to obtain a set of temporary or permanent access control keys. While these keys could be stored in a configuration file, a best practice is for applications running on AWS to use IAM Amazon EC2 instance profiles to manage credentials. IAM Amazon EC2 instance profiles or roles allow you to avoid storing sensitive keys in configuration files that must then be secured.

 For mobile applications, a best practice is to use a combination of web identity federation with the AWS Security Token Service (AWS STS) to issue temporary keys that expire after a short period.

Amazon DynamoDB also provides support for fine-grained access control that can restrict access to specific items within a table or even specific attributes within an item. For example, you may want to limit a user to only access his or her items within a table and prevent access to items associated with a different user. Using conditions in an IAM policy allows you to restrict which actions a user can perform, on which tables, and to which attributes a user can read or write.

Amazon DynamoDB Streams

A common requirement for many applications is to keep track of recent changes and then perform some kind of processing on the changed records. Amazon DynamoDB Streams makes it easy to get a list of item modifications for the last 24-hour period. For example, you might need to calculate metrics on a rolling basis and update a dashboard, or maybe synchronize two tables or log activity and changes to an audit trail. With Amazon DynamoDB Streams, these types of applications become easier to build.

Amazon DynamoDB Streams allows you to extend application functionality without modifying the original application. By reading the log of activity changes from the stream, you can build new integrations or support new reporting requirements that weren't part of the original design.

Each item change is buffered in a time-ordered sequence or stream that can be read by other applications. Changes are logged to the stream in near real-time and allow you to respond quickly or chain together a sequence of events based on a modification.

Streams can be enabled or disabled for an Amazon DynamoDB table using the AWS Management Console, Command Line Interface (CLI), or SDK. A stream consists of stream records. Each stream record represents a single data modification in the Amazon DynamoDB table to which the stream belongs. Each stream record is assigned a sequence number, reflecting the order in which the record was published to the stream.

Stream records are organized into groups, also referred to as *shards*. Each shard acts as a container for multiple stream records and contains information on accessing and iterating through the records. Shards live for a maximum of 24 hours and, with fluctuating load levels, could be split one or more times before they are eventually closed.

 To build an application that reads from a shard, it is recommended to use the Amazon DynamoDB Streams Kinesis Adapter. The Kinesis Client Library (KCL) simplifies the application logic required to process reading records from streams and shards.

Summary

In this chapter, you learned the basic concepts of relational databases, data warehouses, and NoSQL databases. You also learned about the benefits and features of AWS managed database services Amazon RDS, Amazon Redshift, and Amazon DynamoDB.

Amazon RDS manages the heavy lifting involved in administering a database infrastructure and software and lets you focus on building the relational schemas that best fit your use case and the performance tuning to optimize your queries.

Amazon RDS supports popular open-source and commercial database engines and provides a consistent operational model for common administrative tasks. Increase your availability by running a master-slave configuration across Availability Zones using Multi-AZ deployment. Scale your application and increase your database read performance using read replicas.

Amazon Redshift allows you to deploy a data warehouse cluster that is optimized for analytics and reporting workloads within minutes. Amazon Redshift distributes your records using columnar storage and parallelizes your query execution across multiple compute nodes to deliver fast query performance. Amazon Redshift clusters can be scaled up or down to support large, petabyte-scale databases using SSD or magnetic disk storage.

Connect to Amazon Redshift clusters using standard SQL clients with JDBC/ODBC drivers and execute SQL queries using many of the same analytics and ETL tools that you use today. Load data into your Amazon Redshift clusters using the `COPY` command to bulk import flat files stored in Amazon S3, then run standard `SELECT` commands to search and query the table.

Back up both your Amazon RDS databases and Amazon Redshift clusters using automated and manual snapshots to allow for point-in-time recovery. Secure your Amazon RDS and Amazon Redshift databases using a combination of IAM, database-level access control, network-level access control, and data encryption techniques.

Amazon DynamoDB simplifies the administration and operations of a NoSQL database in the cloud. Amazon DynamoDB allows you to create tables quickly that can scale to an unlimited number of items and configure very high levels of provisioned read and write capacity.

Amazon DynamoDB tables provide a flexible data storage mechanism that only requires a primary key and allows for one or more attributes. Amazon DynamoDB supports both simple scalar data types like String and Number, and also more complex structures using List and Map. Secure your Amazon DynamoDB tables using IAM and restrict access to items and attributes using fine-grained access control.

Amazon DynamoDB will handle the difficult task of cluster and partition management and provide you with a highly available database table that replicates data across Availability Zones for increased durability. Track and process recent changes by tapping into Amazon DynamoDB Streams.

Exam Essentials

Know what a relational database is. A relational database consists of one or more tables. Communication to and from relational databases usually involves simple SQL queries, such as "Add a new record," or "What is the cost of product *x*?" These simple queries are often referred to as OLTP.

Understand which databases are supported by Amazon RDS. Amazon RDS currently supports six relational database engines:

- Microsoft SQL Server
- MySQL Server
- Oracle
- PostgreSQL
- MariaDB
- Amazon Aurora

Understand the operational benefits of using Amazon RDS. Amazon RDS is a managed service provided by AWS. AWS is responsible for patching, antivirus, and management of the underlying guest OS for Amazon RDS. Amazon RDS greatly simplifies the process of setting a secondary slave with replication for failover and setting up read replicas to offload queries.

Remember that you cannot access the underlying OS for Amazon RDS DB instances. You cannot use Remote Desktop Protocol (RDP) or SSH to connect to the underlying OS. If you need to access the OS, install custom software or agents, or want to use a database engine not supported by Amazon RDS, consider running your database on Amazon EC2 instead.

Know that you can increase availability using Amazon RDS Multi-AZ deployment. Add fault tolerance to your Amazon RDS database using Multi-AZ deployment. You can quickly set up a secondary DB Instance in another Availability Zone with Multi-AZ for rapid failover.

Understand the importance of RPO and RTO. Each application should set RPO and RTO targets to define the amount of acceptable data loss and also the amount of time required to recover from an incident. Amazon RDS can be used to meet a wide range of RPO and RTO requirements.

Understand that Amazon RDS handles Multi-AZ failover for you. If your primary Amazon RDS Instance becomes unavailable, AWS fails over to your secondary instance in another Availability Zone automatically. This failover is done by pointing your existing database endpoint to a new IP address. You do not have to change the connection string manually; AWS handles the DNS change automatically.

Remember that Amazon RDS read replicas are used for scaling out and increased performance. This replication feature makes it easy to scale out your read-intensive databases. Read replicas are currently supported in Amazon RDS for MySQL, PostgreSQL,

and Amazon Aurora. You can create one or more replicas of a database within a single AWS Region or across multiple AWS Regions. Amazon RDS uses native replication to propagate changes made to a source DB Instance to any associated read replicas. Amazon RDS also supports cross-region read replicas to replicate changes asynchronously to another geography or AWS Region.

Know what a NoSQL database is. NoSQL databases are non-relational databases, meaning that you do not have to have an existing table created in which to store your data. NoSQL databases come in the following formats:

- Document databases
- Graph stores
- Key/value stores
- Wide-column stores

Remember that Amazon DynamoDB is AWS NoSQL service. You should remember that for NoSQL databases, AWS provides a fully managed service called Amazon DynamoDB. Amazon DynamoDB is an extremely fast NoSQL database with predictable performance and high scalability. You can use Amazon DynamoDB to create a table that can store and retrieve any amount of data and serve any level of request traffic. Amazon DynamoDB automatically spreads the data and traffic for the table over a sufficient number of partitions to handle the request capacity specified by the customer and the amount of data stored, while maintaining consistent and fast performance.

Know what a data warehouse is. A data warehouse is a central repository for data that can come from one or more sources. This data repository would be used for query and analysis using OLAP. An organization's management typically uses a data warehouse to compile reports on specific data. Data warehouses are usually queried with highly complex queries.

Remember that Amazon Redshift is AWS data warehouse service. You should remember that Amazon Redshift is Amazon's data warehouse service. Amazon Redshift organizes the data by column instead of storing data as a series of rows. Because only the columns involved in the queries are processed and columnar data is stored sequentially on the storage media, column-based systems require far fewer I/Os, which greatly improves query performance. Another advantage of columnar data storage is the increased compression, which can further reduce overall I/O.

Exercises

In order to pass the exam, you should practice deploying databases and creating tables using Amazon RDS, Amazon DynamoDB, and Amazon Redshift. Remember to delete any resources you provision to minimize any charges.

EXERCISE 7.1

Create a MySQL Amazon RDS Instance

1. Log in to the AWS Management Console, and navigate to the Amazon RDS Console.
2. Launch a new Amazon RDS DB Instance, and select MySQL Community Edition instance as the database engine.
3. Configure the DB Instance to use Multi-AZ and General Purpose (SSD) storage.

   Warning: This is not eligible for AWS Free Tier; you will incur a small charge by provisioning this instance.
4. Set the DB Instance identifier and database name to **`MySQL123`**, and configure the master username and password.
5. Validate the configuration settings, and launch the DB Instance.
6. Return to the list of the Amazon RDS instances. You will see the status of your Amazon RDS database as `Creating`. It may take up to 20 minutes to create your new Amazon RDS instance.

You have provisioned your first Amazon RDS instance using Multi-AZ.

EXERCISE 7.2

Simulate a Failover from One AZ to Another

In this exercise, you will use Multi-AZ failover to simulate a failover from one Availability Zone to another.

1. In the Amazon RDS Console, view the list of DB Instances.
2. Find your DB Instance called MySQL123, and check its status. When its status is `Available`, proceed to the next step.
3. Select the instance, and issue a Reboot command from the actions menu.
4. Confirm the reboot.

You have now simulated a failover from one Availability Zone to another using Multi-AZ failover. The failover should take approximately two or three minutes.

EXERCISE 7.3

Create a Read Replica

In this exercise, you will create a read replica of your existing MySQL123 DB server.

1. In the Amazon RDS Console, view the list of DB Instances.
2. Find your DB Instance called `MySQL123`, and check its status. When its status is `Available`, proceed to the next step.
3. Select the instance, and issue a Create Read Replica command from the list of actions.
4. Configure the name of the read replica and any other settings. Create the replica.
5. Wait for the replica to be created, which can typically take several minutes. When it is complete, delete both the MySQL123 and MySQLReadReplica databases by clicking the checkboxes next to them, clicking the Instance Actions drop-down box, and then clicking Delete.

In the preceding exercises, you created a new Amazon RDS MySQL instance with Multi-AZ enabled. You then simulated a failover from one Availability Zone to another by rebooting the primary instance. After that, you scaled your Amazon RDS instance out by creating a read replica of the primary database. Delete the DB Instance.

EXERCISE 7.4

Read and Write from a DynamoDB Table

In this exercise, you will create an Amazon DynamoDB table and then read and write to it using the AWS Management Console.

1. Log in to the AWS Management Console, and view the Amazon DynamoDB console.
2. Create a new table named **`UserProfile`** with a partition key of userID of type String.
3. After the table has been created, view the list of items in the table.
4. Using the Amazon DynamoDB console, create and save a new item in the table. Set the userID to U01, and append another String attribute called **name** with a value of **`Joe`**.
5. Perform a scan on the table to retrieve the new item.

You have now created a simple Amazon DynamoDB table, put a new item, and retrieved it using Scan. Delete the DynamoDB table.

EXERCISE 7.5

Launch a Redshift Cluster

In this exercise, you will create a data warehouse using Amazon Redshift and then read and write to it using the AWS Management Console.

1. Log in to the AWS Management Console, and view the Amazon Redshift Console.
2. Create a new cluster, configuring the database name, username, and password.
3. Configure the cluster to be single node using one SSD-backed storage node.
4. Launch the cluster into an Amazon VPC using the appropriate security group.
5. Install and configure SQL Workbench on your local computer, and connect to the new cluster.
6. Create a new table and load data using the `COPY` command.

You have now created an Amazon Redshift cluster and connected to it using a standard SQL client. Delete the cluster when you have completed the exercise.

Review Questions

1. Which AWS database service is best suited for traditional Online Transaction Processing (OLTP)?
   A. Amazon Redshift
   B. Amazon Relational Database Service (Amazon RDS)
   C. Amazon Glacier
   D. Elastic Database
2. Which AWS database service is best suited for non-relational databases?
   A. Amazon Redshift
   B. Amazon Relational Database Service (Amazon RDS)
   C. Amazon Glacier
   D. Amazon DynamoDB
3. You are a solutions architect working for a media company that hosts its website on AWS. Currently, there is a single Amazon Elastic Compute Cloud (Amazon EC2) Instance on AWS with MySQL installed locally to that Amazon EC2 Instance. You have been asked to make the company's production environment more resilient and to increase performance. You suggest that the company split out the MySQL database onto an Amazon RDS Instance with Multi-AZ enabled. This addresses the company's increased resiliency requirements. Now you need to suggest how you can increase performance. Ninety-nine percent of the company's end users are magazine subscribers who will be reading additional articles on the website, so only one percent of end users will need to write data to the site. What should you suggest to increase performance?
   A. Alter the connection string so that if a user is going to write data, it is written to the secondary copy of the Multi-AZ database.
   B. Alter the connection string so that if a user is going to write data, it is written to the primary copy of the Multi-AZ database.
   C. Recommend that the company use read replicas, and distribute the traffic across multiple read replicas.
   D. Migrate the MySQL database to Amazon Redshift to take advantage of columnar storage and maximize performance.
4. Which AWS Cloud service is best suited for Online Analytics Processing (OLAP)?
   A. Amazon Redshift
   B. Amazon Relational Database Service (Amazon RDS)
   C. Amazon Glacier
   D. Amazon DynamoDB
5. You have been using Amazon Relational Database Service (Amazon RDS) for the last

year to run an important application with automated backups enabled. One of your team members is performing routine maintenance and accidentally drops an important table, causing an outage. How can you recover the missing data while minimizing the duration of the outage?

A. Perform an undo operation and recover the table.

B. Restore the database from a recent automated DB snapshot.

C. Restore only the dropped table from the DB snapshot.

D. The data cannot be recovered.

6. Which Amazon Relational Database Service (Amazon RDS) database engines support Multi-AZ?

A. All of them

B. Microsoft SQL Server, MySQL, and Oracle

C. Oracle, Amazon Aurora, and PostgreSQL

D. MySQL

7. Which Amazon Relational Database Service (Amazon RDS) database engines support read replicas?

A. Microsoft SQL Server and Oracle

B. MySQL, MariaDB, PostgreSQL, and Aurora

C. Aurora, Microsoft SQL Server, and Oracle

D. MySQL and PostgreSQL

8. Your team is building an order processing system that will span multiple Availability Zones. During testing, the team wanted to test how the application will react to a database failover. How can you enable this type of test?

A. Force a Multi-AZ failover from one Availability Zone to another by rebooting the primary instance using the Amazon RDS console.

B. Terminate the DB instance, and create a new one. Update the connection string.

C. Create a support case asking for a failover.

D. It is not possible to test a failover.

9. You are a system administrator whose company has moved its production database to AWS. Your company monitors its estate using Amazon CloudWatch, which sends alarms using Amazon Simple Notification Service (Amazon SNS) to your mobile phone. One night, you get an alert that your primary Amazon Relational Database Service (Amazon RDS) Instance has gone down. You have Multi-AZ enabled on this instance. What should you do to ensure the failover happens quickly?

A. Update your Domain Name System (DNS) to point to the secondary instance's new IP address, forcing your application to fail over to the secondary instance.

B. Connect to your server using Secure Shell (SSH) and update your connection strings

so that your application can communicate to the secondary instance instead of the failed primary instance.

C. Take a snapshot of the secondary instance and create a new instance using this snapshot, then update your connection string to point to the new instance.

D. No action is necessary. Your connection string points to the database endpoint, and AWS automatically updates this endpoint to point to your secondary instance.

10. You are working for a small organization without a dedicated database administrator on staff. You need to install Microsoft SQL Server Enterprise edition quickly to support an accounting back office application on Amazon Relational Database Service (Amazon RDS). What should you do?

A. Launch an Amazon RDS DB Instance, and select Microsoft SQL Server Enterprise Edition under the Bring Your Own License (BYOL) model.

B. Provision SQL Server Enterprise Edition using the License Included option from the Amazon RDS Console.

C. SQL Server Enterprise edition is only available via the Command Line Interface (CLI). Install the command-line tools on your laptop, and then provision your new Amazon RDS Instance using the CLI.

D. You cannot use SQL Server Enterprise edition on Amazon RDS. You should install this on to a dedicated Amazon Elastic Compute Cloud (Amazon EC2) Instance.

11. You are building the database tier for an enterprise application that gets occasional activity throughout the day. Which storage type should you select as your default option?

A. Magnetic storage

B. General Purpose Solid State Drive (SSD)

C. Provisioned IOPS (SSD)

D. Storage Area Network (SAN)-attached

12. You are designing an e-commerce web application that will scale to potentially hundreds of thousands of concurrent users. Which database technology is best suited to hold the session state for large numbers of concurrent users?

A. Relational database using Amazon Relational Database Service (Amazon RDS)

B. NoSQL database table using Amazon DynamoDB

C. Data warehouse using Amazon Redshift

D. Amazon Simple Storage Service (Amazon S3)

13. Which of the following techniques can you use to help you meet Recovery Point Objective (RPO) and Recovery Time Objective (RTO) requirements? (Choose 3 answers)

A. DB snapshots

B. DB option groups

C. Read replica

D. Multi-AZ deployment

14. When using Amazon Relational Database Service (Amazon RDS) Multi-AZ, how can you offload read requests from the primary? (Choose 2 answers)
    A. Configure the connection string of the clients to connect to the secondary node and perform reads while the primary is used for writes.
    B. Amazon RDS automatically sends writes to the primary and sends reads to the secondary.
    C. Add a read replica DB instance, and configure the client's application logic to use a read-replica.
    D. Create a caching environment using ElastiCache to cache frequently used data. Update the application logic to read/write from the cache.
15. You are building a large order processing system and are responsible for securing the database. Which actions will you take to protect the data? (Choose 3 answers)
    A. Adjust AWS Identity and Access Management (IAM) permissions for administrators.
    B. Configure security groups and network Access Control Lists (ACLs) to limit network access.
    C. Configure database users, and grant permissions to database objects.
    D. Install anti-virus software on the Amazon RDS DB Instance.
16. Your team manages a popular website running Amazon Relational Database Service (Amazon RDS) MySQL back end. The Marketing department has just informed you about an upcoming television commercial that will drive thousands of new visitors to the website. How can you prepare your database to handle the load? (Choose 3 answers)
    A. Vertically scale the DB Instance by selecting a more powerful instance class.
    B. Create read replicas to offload read requests and update your application.
    C. Upgrade the storage from Magnetic volumes to General Purpose Solid State Drive (SSD) volumes.
    D. Upgrade to Amazon Redshift for faster columnar storage.
17. You are building a photo management application that maintains metadata on millions of images in an Amazon DynamoDB table. When a photo is retrieved, you want to display the metadata next to the image. Which Amazon DynamoDB operation will you use to retrieve the metadata attributes from the table?
    A. Scan operation
    B. Search operation
    C. Query operation
    D. Find operation
18. You are creating an Amazon DynamoDB table that will contain messages for a social chat application. This table will have the following attributes: Username (String), Timestamp (Number), Message (String). Which attribute should you use as the partition key? The

sort key?

A. Username, Timestamp
B. Username, Message
C. Timestamp, Message
D. Message, Timestamp

19. Which of the following statements about Amazon DynamoDB tables are true? (Choose 2 answers)

A. Global secondary indexes can only be created when the table is being created.
B. Local secondary indexes can only be created when the table is being created.
C. You can only have one global secondary index.
D. You can only have one local secondary index.

20. Which of the following workloads are a good fit for running on Amazon Redshift? (Choose 2 answers)

A. Transactional database supporting a busy e-commerce order processing website
B. Reporting database supporting back-office analytics
C. Data warehouse used to aggregate multiple disparate data sources
D. Manage session state and user profile data for thousands of concurrent users