Module 8: Database

Challenges of Relational Databases

Running a relational database on-premises comes with several administrative challenges. These include server maintenance, which involves regular upkeep of the server and managing its energy footprint. Additionally, software installation and patching, both for the database software and the operating system (OS), require constant attention. Ensuring high availability, which means making sure the database is always accessible, is another critical task. Scalability, or the ability to handle an increasing amount of work, also presents challenges. Data security is paramount, necessitating careful planning and implementation. All these tasks demand time, expertise, and resources, often diverting attention from other essential activities.

Amazon RDS: A Managed Database Service

Amazon RDS (Relational Database Service) is a managed service that automates the setup, operation, and scaling of relational databases in the cloud. This service addresses the challenges of managing standalone databases by handling time-consuming administrative tasks such as backups, patching, and scaling. With Amazon RDS, businesses can focus more on optimizing their applications rather than worrying about database management. This service offers cost-efficient and resizable capacity, ensuring that the database meets the performance, availability, security, and compatibility needs of the applications it supports.

Transitioning from On-Premises Databases to Amazon RDS

When managing an on-premises database, a database administrator is responsible for everything, from hardware setup to application optimization. However, transitioning to Amazon RDS or Amazon Aurora significantly reduces administrative responsibilities. While using Amazon EC2 (Elastic Compute Cloud) still requires handling software and backup operations, using Amazon RDS or Aurora automates most tasks, such as database backups, software patching, and high availability management. This transition allows businesses to focus on optimizing applications, with AWS taking care of most of the underlying infrastructure.

Responsibilities in Managed Services

In a managed service environment like Amazon RDS, the responsibilities are divided between the user and AWS. The user remains responsible for optimizing applications, while AWS manages the OS and database software installation and patching, automatic backups, high availability, scaling, and server maintenance. This arrangement reduces the operational workload and associated costs for maintaining a relational database, allowing businesses to focus on what matters most.

Amazon RDS DB Instances

The basic unit of Amazon RDS is the database instance, an isolated environment that can contain multiple user-created databases. The performance and cost of a database instance depend on the instance class (which includes CPU, memory, and network performance) and the type of storage (e.g., magnetic, SSD, or Provisioned IOPS). When creating a database instance, users must choose a database engine, such as MySQL, Amazon Aurora, Microsoft SQL Server, PostgreSQL, MariaDB, or Oracle. This flexibility allows businesses to tailor their database environment to meet specific performance and cost needs.

Amazon RDS in a Virtual Private Cloud (VPC)

Amazon RDS can be run within a Virtual Private Cloud (VPC), giving users control over their virtual networking environment. Within a VPC, users can select their IP address range, create subnets, and configure routing and access control lists (ACLs). Typically, the database instance is isolated in a private subnet, making it accessible only to specific application instances. By associating subnets with specific Availability Zones, users can also control the physical location of their database instances within the cloud infrastructure.

High Availability with Multi-AZ Deployment

Amazon RDS supports high availability through Multi-AZ (Availability Zone) deployment, where a standby copy of the database is created in another Availability Zone within the same VPC. This configuration ensures that transactions are synchronously replicated to the standby instance, minimizing data loss during failover events. If the primary database instance fails, Amazon RDS automatically promotes the standby instance to become the new primary instance. Since applications connect to the database through a DNS endpoint, no changes to the application code are needed for failover, ensuring seamless operation.

Example

Imagine a company using an on-premises MySQL database. The IT team spends a significant amount of time maintaining servers, installing patches, and performing backups. By switching to Amazon RDS, the company can offload these tasks to AWS, allowing the IT team to focus on optimizing the application. The database runs within a VPC, ensuring that it's securely isolated, and the company opts for a Multi-AZ deployment to ensure high availability. As a result, the company reduces its operational workload, improves database reliability, and can easily scale the database as needed.

▼ Definition of New and Difficult Terms

Definition of New and Difficult Terms

- **High Availability**: Ensuring that a system or component is continuously operational for a desirably long length of time.
- **Scalability**: The capability of a system to handle a growing amount of work or its potential to accommodate growth.
- **Managed Service**: A service where the provider handles the operation, maintenance, and administration, reducing the user's responsibilities.

- **Virtual Private Cloud (VPC)**: A private cloud environment within a public cloud, allowing users to control their virtual networking environment.
- **Subnets**: A segmented part of a network, allowing for control over traffic flow and security.
- **Availability Zone (AZ)**: A distinct location within a cloud provider's region, isolated from failures in other AZs to provide high availability.
- **Provisioned IOPS**: A storage option that provides a specific level of input/output operations per second (IOPS) for applications that require fast, consistent performance.
- **Multi-AZ Deployment**: A setup where the database is replicated across multiple availability zones to ensure high availability and failover protection.
- **Domain Name System (DNS) Endpoint**: The address used by applications to connect to a database, which remains constant even if the underlying database instance changes during failover.
- **On-premises** refers to software, hardware, or infrastructure that is hosted and operated within a physical location owned or controlled by the organization using it, rather than being hosted remotely in the cloud or by a third-party service provider.

  In the context of databases, an on-premises database means that the database servers, storage, and related infrastructure are physically located within the organization's data centers or offices. The organization is responsible for all aspects of managing and maintaining the infrastructure, including server maintenance, software updates, backups, and security. This contrasts with cloud-based services, where much of the management and infrastructure responsibilities are handled by a third-party provider, like AWS.

Amazon RDS Read Replicas

Features:

- **Asynchronous Replication:** Amazon RDS read replicas use asynchronous replication, meaning updates to the primary database are copied to the read replicas after they occur. This approach helps reduce latency and balance the load between multiple database instances.
- **Promotion to Primary:** Read replicas can be promoted to become the primary database instance if needed. However, due to the nature of asynchronous replication, this promotion must be done manually.

Functionality:

- **Read-Heavy Workloads:** Read replicas are particularly useful for applications with heavy read demands. By routing read queries to the replica, you can offload some of the workload from the primary database, ensuring it remains responsive.
- **Scalability:** Read replicas allow you to scale your database beyond the capacity of a single instance, enabling better performance for read-heavy workloads.
- **Cross-Region Replication:** Read replicas can be created in a different region than the primary database, which helps in disaster recovery and reduces latency for users located closer to the replica.

Use Cases for Amazon RDS

Application Type	**Benefits of Amazon RDS**
Web and Mobile Applications	High throughput, massive scalability, high availability, no licensing constraints.
Ecommerce Applications	Low-cost, secure, fully managed solution with high availability.
Mobile and Online Games	Automatic scaling, high throughput, and database monitoring.

Amazon RDS works well for web and mobile applications that need a database with high throughput, massive storage scalability, and high availability. Because Amazon RDS does not have any licensing constraints, it fits the variable usage pattern of these applications. For small and large ecommerce businesses, Amazon RDS provides a flexible, secure, and low-cost database solution for online sales and retailing. Mobile and online games require a database platform with high throughput and availability. Amazon RDS manages the database infrastructure, so game developers donot need to worry about provisioning, scaling, or monitoring database servers.

When to Use Amazon RDS

Use Amazon RDS When:

- **Complex Transactions or Queries:** Ideal for applications requiring complex database operations.
- **Medium to High Query or Write Rate:** Supports up to 30,000 IOPS, making it suitable for moderate to high database workloads.
- **Single Worker Node:** Best for applications that do not require sharding or massive data distribution.
- **High Durability:** Ensures data durability and high availability through automated backups and multi-AZ deployment.

Avoid Amazon RDS When:

- **Massive Read/Write Rates:** Not suitable for applications needing extremely high transaction rates, like 150,000 writes per second.
- **Sharding Needs:** If your application requires data sharding due to high data size or throughput demands, consider alternatives like DynamoDB.
- **Simple Queries:** For applications that can handle simple GET or PUT requests, a NoSQL database might be a better choice.

Amazon RDS: Cost Considerations

Clock-Hour Billing:

- **Resources Incurring Charges:** Charges are based on the time the database instance is running, from launch to termination.

Database Characteristics:

- **Physical Capacity:** Costs vary depending on the chosen database engine, size, and memory class.

Cost Factor	Details
Clock-Hour Billing	Charges are based on the time a database instance is active.
Database Characteristics	Costs depend on engine type, size, and memory class.

Amazon RDS: Purchase Types and Multiple DB Instances

DB Purchase Types:

- **On-Demand Instances:** Pay for compute capacity by the hour, with no minimum commitment.
- **Reserved Instances:** Make a low, upfront payment for a database instance reserved for 1 or 3 years, offering cost savings.

Number of DB Instances:

- **Provision Multiple Instances:** Amazon RDS allows provisioning of multiple instances to handle peak loads, ensuring high availability and performance.

Amazon RDS: Storage Considerations

Storage Type	Details
Provisioned Storage	Free backup storage up to 100% of provisioned storage for active instances.
Additional Storage	Charged per GB, per month for storage beyond the provisioned amount.

New terms Definitions

▼ Definitions of New and Difficult Terms

- **Asynchronous Replication:** A method of data replication where updates are not immediately copied to the replica but occur after the changes are made to the primary database, allowing for eventual consistency.
- **Clock-Hour Billing:** A billing method that charges based on the number of hours a service is running.
- **Provisioned Storage:** The amount of storage allocated to a database instance, which can be used to store data and backups.

- **On-Demand Instances:** A type of cloud service that allows users to pay for computing resources by the hour without any upfront commitment.
- **Reserved Instances:** A purchasing option where users can reserve a database instance for a long period (1-3 years) with an upfront payment, offering a lower overall cost.
- **Sharding:** A database partitioning technique that divides large datasets into smaller, more manageable pieces called shards, spread across multiple database instances.

Amazon RDS: Deployment Type and Data Transfer

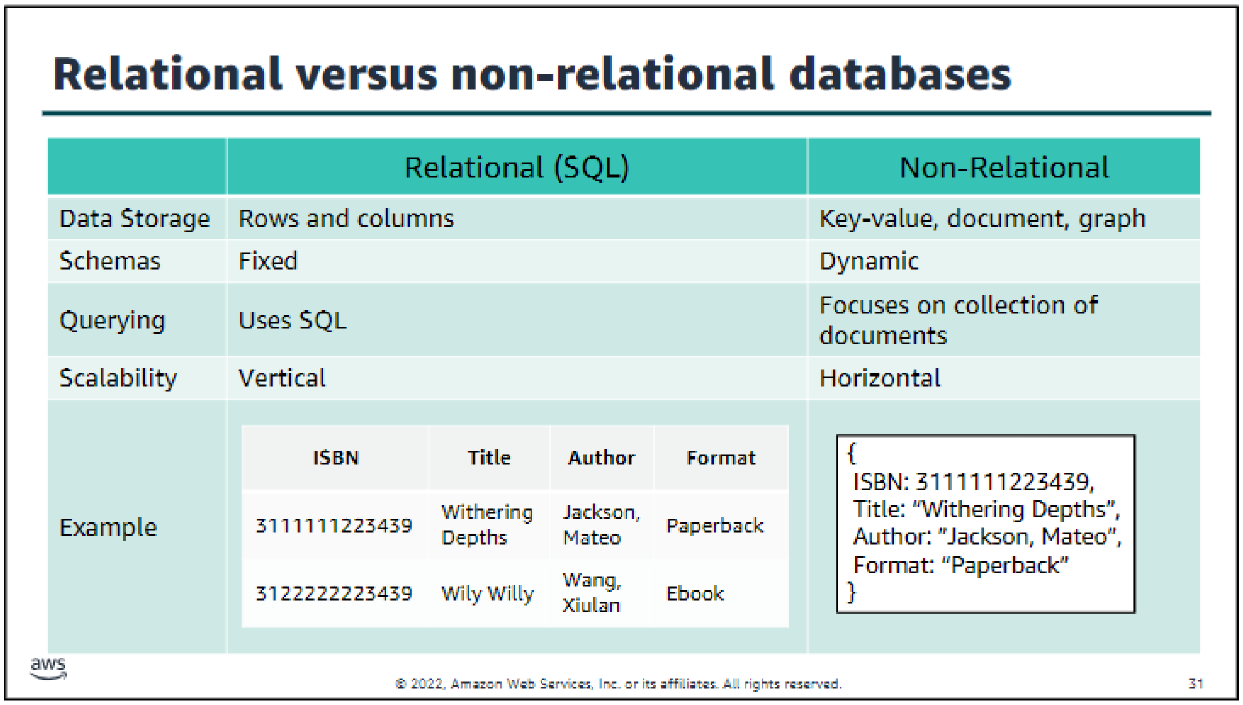
1. **Requests**:
    - **Input and Output (I/O) Requests**: These refer to the number of database operations, such as reads and writes, made to the Amazon RDS instance. The frequency and volume of these requests can influence the cost and performance of the database.
2. **Deployment Type**:
    - **Single Availability Zone (AZ)**: Deploying an RDS instance in a single Availability Zone is similar to running it in a standalone data center. This option typically incurs lower storage and I/O charges but offers less redundancy.
    - **Multiple Availability Zones (Multi-AZ)**: Deploying in multiple Availability Zones provides higher availability and durability by replicating data across zones, which is analogous to using a secondary data center. This deployment type is generally more expensive due to the added redundancy.
3. **Data Transfer**:
    - **Inbound Data Transfer**: There is no charge for inbound data transfer, meaning data sent to the RDS instance does not incur additional costs.
    - **Outbound Data Transfer**: Charges for outbound data transfer (data leaving the RDS instance) are tiered. The cost increases with the volume of data

transferred out.

4. **Cost Optimization**:
    - **Reserved Instances**: To optimize costs, especially for long-term projects, you can purchase Reserved Instances. This involves making a low, one-time upfront payment for each database instance you want to reserve for a set term (1 or 3 years). This payment provides a significant discount on the hourly usage charges for the reserved instance.

Amazon DynamoDB

Relational VS Non Relational databases



Relational versus non-relational databases

	Relational (SQL)	Non-Relational
Data Storage	Rows and columns	Key-value, document, graph
Schemas	Fixed	Dynamic
Querying	Uses SQL	Focuses on collection of documents
Scalability	Vertical	Horizontal
Example	(see below)	{ ISBN: 3111111223439, Title: "Withering Depths", Author: "Jackson, Mateo", Format: "Paperback" }

ISBN	Title	Author	Format
3111111223439	Withering Depths	Jackson, Mateo	Paperback
3122222223439	Wily Willy	Wang, Xiulan	Ebook

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved. 31

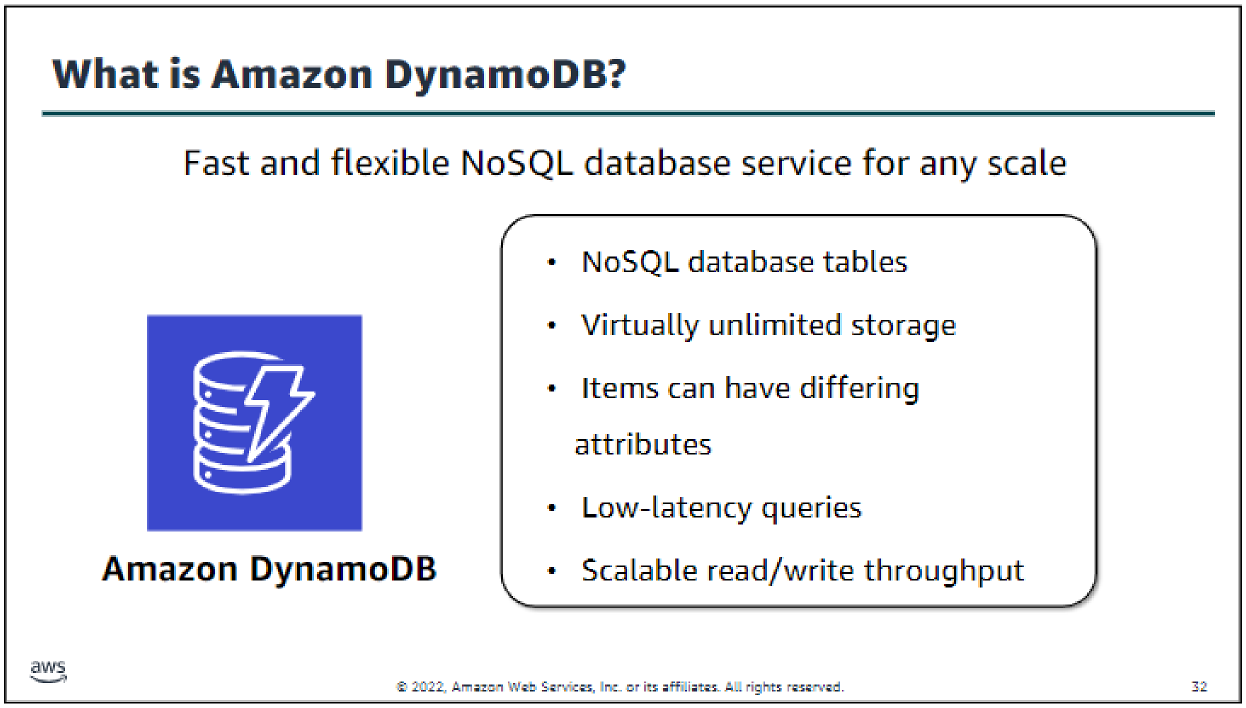
With DynamoDB, this module transitions from relational databases to non-relational databases. Here is a review of the differences between these two types of databases:

- A relational database (RDB) works with structured data that is organized by tables, records, and columns. RDBs establish a well-defined relationship between database tables. RDBs use structured query language (SQL), which is a standard user application that provides a programming interface for database interaction. Relational databases might have difficulties scaling out horizontally or working with semi-structured data, and might also require many joins for normalized data.

- Anon-relational database is any database that does not follow the relational model that is provided by traditional relational database management systems (RDBMS). Non-relational databases have grown in popularity because they were designed to overcome the limitations of relational databases for handling the demands of variably structured data. Non-relational databases scale out horizontally, and they can work with unstructured and semi structured data.

Here is a look at what DynamoDB offers

What is Amazon DynamoDB ?



What is Amazon DynamoDB?

Fast and flexible NoSQL database service for any scale

Amazon DynamoDB

- NoSQL database tables
- Virtually unlimited storage
- Items can have differing attributes
- Low-latency queries
- Scalable read/write throughput

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved. 32

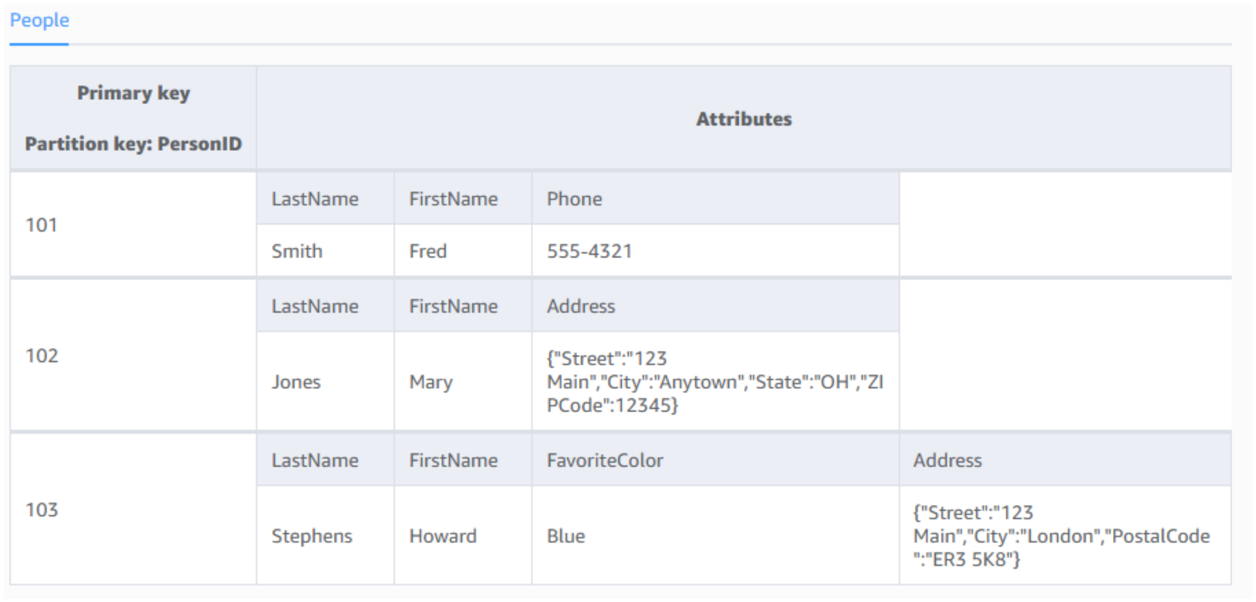
Here are the key points to remember about DynamoDB

1. **NoSQL Database**: DynamoDB is a fast and flexible NoSQL database service that offers consistent, single-digit-millisecond latency at any scale.

2. **Managed by Amazon**: Amazon manages the underlying data infrastructure, providing fault-tolerant architecture with redundant data storage across multiple facilities.
3. **Scalability**: There is no practical limit on the number of items you can store in a table. DynamoDB automatically partitions data and scales storage to meet workload requirements.
4. **Flexibility**: Items in the same table can have different attributes, allowing you to add or modify attributes as your application evolves without schema migrations.
5. **Performance**: Data is stored on SSDs, ensuring low-latency query performance. You can provision or automatically scale read/write throughput based on your application's needs.
6. **Additional Features**: DynamoDB supports global tables for automatic replication across AWS Regions, encryption at rest, and item Time-to-Live (TTL) for data expiration.

Core components of Amazon DynamoDB

In DynamoDB, tables, items, and attributes are the core components that you work with. A *table* is a collection of *items*, and each item is a collection of *attributes*. DynamoDB uses primary keys to uniquely identify each item in a table and secondary indexes to provide more querying flexibility.


People

Primary key Partition key: PersonID	Attributes			
101	LastName	FirstName	Phone	
	Smith	Fred	555-4321	
102	LastName	FirstName	Address	
	Jones	Mary	{"Street":"123 Main","City":"Anytown","State":"OH","ZIPCode":12345}	
103	LastName	FirstName	FavoriteColor	Address
	Stephens	Howard	Blue	{"Street":"123 Main","City":"London","PostalCode":"ER3 5K8"}

The following are the basic DynamoDB components:

- **Tables** – Similar to other database systems, DynamoDB stores data in tables. A *table* is a collection of data. For example, see the example table called *People* that you could use to store personal contact information about friends, family, or anyone else of interest. You could also have a *Cars* table to store information about vehicles that people drive.
- **Items** – Each table contains zero or more items. An *item* is a group of attributes that is uniquely identifiable among all of the other items. In a *People* table, each item represents a person. For a *Cars* table, each item represents one vehicle. Items in DynamoDB are similar in many ways to rows, records, or tuples in other database systems. In DynamoDB, there is no limit to the number of items you can store in a table.
- **Attributes** – Each item is composed of one or more attributes. An *attribute* is a fundamental data element, something that does not need to be broken down any further. For example, an item in a *People* table contains attributes called *PersonID*, *LastName*, *FirstName*, and so on. For a *Department* table, an item might have attributes such as *DepartmentID*, *Name*, *Manager*, and so on. Attributes in DynamoDB are similar in many ways to fields or columns in other database systems.

The following diagram shows a table named *People* with some example items and attributes.

```
People

{
    "PersonID": 101,
    "LastName": "Smith",
    "FirstName": "Fred",
    "Phone": "555-4321"
}

{
    "PersonID": 102,
    "LastName": "Jones",
    "FirstName": "Mary",
    "Address": {
                "Street": "123 Main",
                "City": "Anytown",
                "State": "OH",
                "ZIPCode": 12345
    }
}

{
    "PersonID": 103,
    "LastName": "Stephens",
    "FirstName": "Howard",
    "Address": {
                "Street": "123 Main",
                "City": "London",
                "PostalCode": "ER3 5K8"
    },
    "FavoriteColor": "Blue"
}
```

Note the following about the *People* table:

- Each item in the table has a unique identifier, or primary key, that distinguishes the item from all of the others in the table. In the *People* table, the primary key consists of one attribute (*PersonID*).
- Other than the primary key, the *People* table is schemaless, which means that neither the attributes nor their data types need to be defined beforehand. Each item can have its own distinct attributes.
- Most of the attributes are *scalar*, which means that they can have only one value. Strings and numbers are common examples of scalars.
- Some of the items have a nested attribute (*Address*). DynamoDB supports nested attributes up to 32 levels deep.

💡 DynamoDB supports two different kinds of primary keys. The partition key is a simple primary key, which is composed of one attribute called the sort key. The partition key and sort key are also known as the composite primary key, which is composed of two attribute

Partitioning

- **Partitioning**: As the amount of data in a DynamoDB table grows, the data is divided into smaller parts, called partitions. This helps in managing large amounts of data efficiently.
- **Primary Key**: DynamoDB uses a primary key to organize and index the data. The primary key uniquely identifies each item in the table, making it easier and faster to find specific items.

Two Ways to Retrieve Data

1. **Query Operation (Using Primary Key)**
    - **How it Works**: The query operation uses the primary key to quickly find specific items. Since DynamoDB knows exactly where the data is located based on the key, it retrieves the data efficiently.
    - **Example**: Imagine you have a table of student records, and each record is identified by a student ID (the primary key). If you want to find the record

for a specific student, you would use their ID in a query, and DynamoDB would quickly find that record.

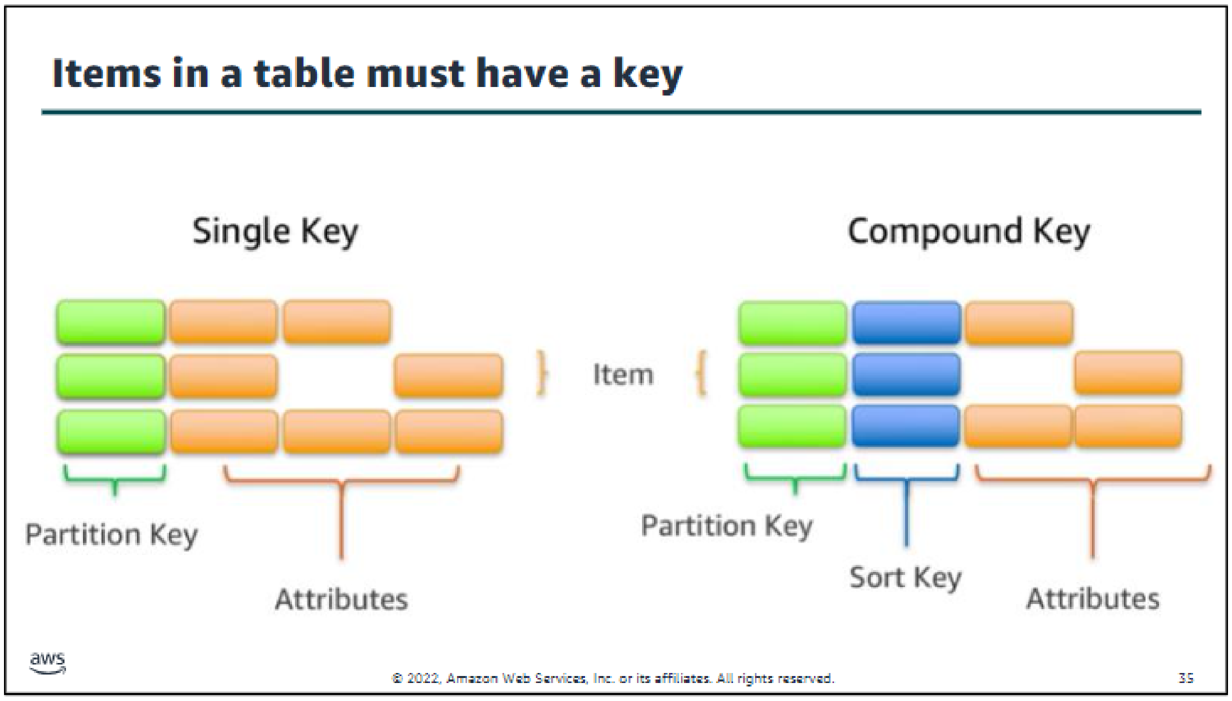
2. **Scan Operation (Using Non-Key Attributes)**
    - **How it Works**: The scan operation allows you to find items based on attributes other than the primary key. However, since the data isn't organized by these other attributes, DynamoDB has to check every item in the table to see if it matches your criteria.
    - **Example**: Continuing with the student records, let's say you want to find all students who scored above 90% in math. Math scores aren't part of the primary key, so you'd use a scan to search through all records to find those students. This process is slower because DynamoDB has to look at each record individually.

Summary

- **Querying by Primary Key**: Fast and efficient because DynamoDB knows where the data is based on the key.
- **Scanning by Non-Key Attributes**: More flexible but slower because DynamoDB has to look through all the data to find what you need.

In practice, you should use queries whenever possible to improve performance, and only use scans when you need to search based on non-key attributes.

Items in a table must have a key



Items in a table must have a key

35

Here's a simpler explanation of the key concepts:

Choosing a Key in DynamoDB

When setting up a DynamoDB table, you need to decide how to uniquely identify each item (or record) in the table. This is done using a **primary key**.

Two Types of Keys

1. **Simple Primary Key (Single Key)**
   - **What It Is**: This key is based on a single attribute (a piece of data) that uniquely identifies each item in the table.
   - **Example**: Suppose you have a table of products. You could use the product ID as the primary key because each product has a unique ID. This ID would be the single key used to locate any product in the table.
2. **Compound Key (Composite Key)**
   - **What It Is**: A compound key is made up of two parts: a **partition key** and a **secondary key**. The partition key is used to divide the data into partitions, and the secondary key is used to sort data within each partition.

  - **Example**: Imagine you have a table of books. You could use a combination of the author's name (partition key) and the book's title (secondary key) as the compound key. This means each book is uniquely identified by both the author and the title together. If you often search for books by a specific author, this method would make those searches faster and more efficient.

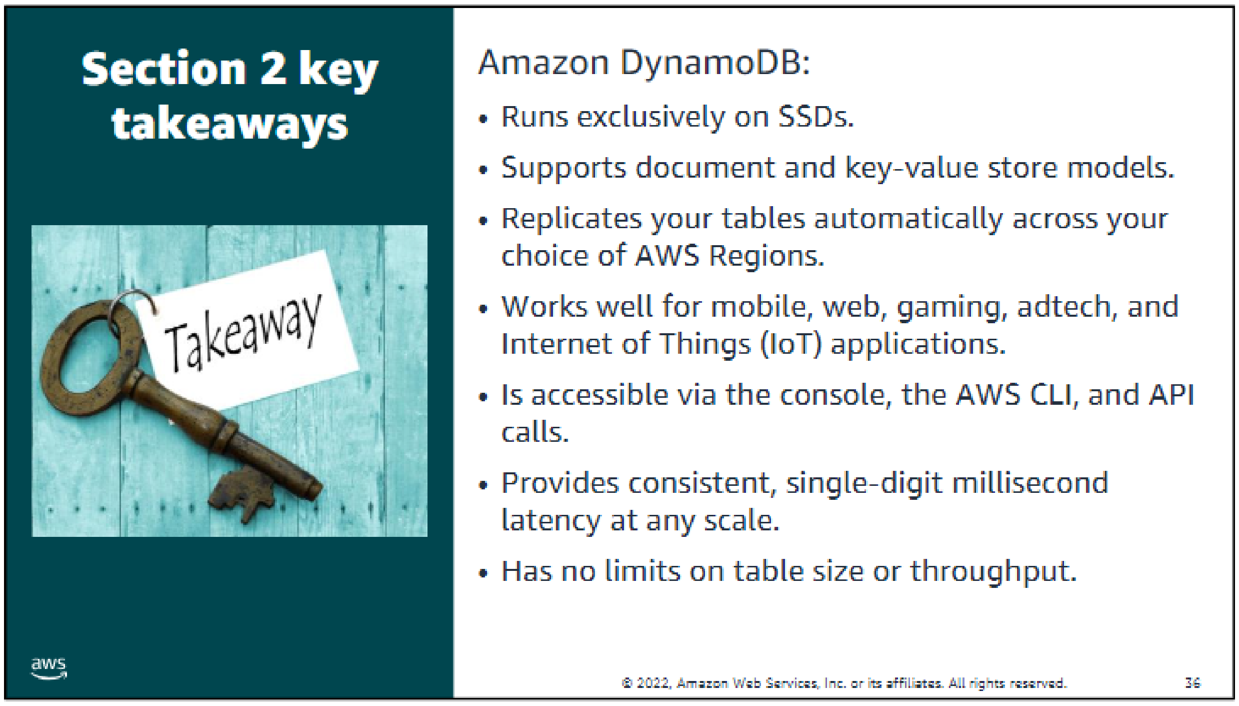
Why It Matters

- **Simple Key**: Easy to implement and works well when you have a single, unique identifier like a product ID.
- **Compound Key**: Useful when you need to search or sort items based on more than one attribute, like finding all books by a particular author.

Summary

- **Simple Key**: Use when each item has a unique attribute, like a product ID or GUID.
- **Compound Key**: Use when you want to combine two attributes to identify items, especially if you frequently search by one of those attributes, like books by the same author.

This choice affects how efficiently you can query your data, so it's important to plan your keys based on how you'll use your data.



Section 2 key takeaways

Amazon DynamoDB:

- Runs exclusively on SSDs.
- Supports document and key-value store models.
- Replicates your tables automatically across your choice of AWS Regions.
- Works well for mobile, web, gaming, adtech, and Internet of Things (IoT) applications.
- Is accessible via the console, the AWS CLI, and API calls.
- Provides consistent, single-digit millisecond latency at any scale.
- Has no limits on table size or throughput.

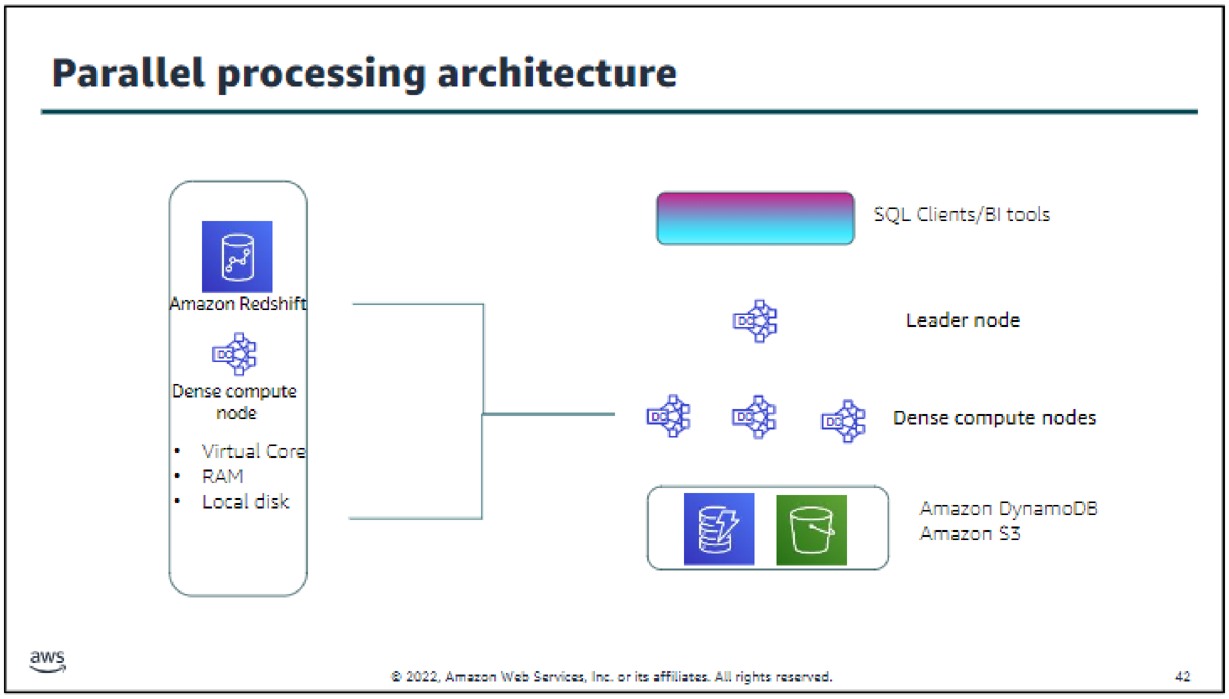
© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved. 36

Amazon Redshift

Amazon Redshift is a fast, fully managed data warehouse that makes it simple and cost-effective to analyze all your data by using standard SQL and your existing business intelligence (BI) tools. Here is a look at Amazon Redshift and how you can use it for analytic applications.

Analytics is important for businesses today, but building a data warehouse is complex and expensive. Data warehouses cantake months and significant financial resources to set up.Amazon Redshift is a fast and powerful, fullymanaged data warehouse that is simple and cost-effective to set up, use, and scale. It enables you to run complex analytic queries against petabytes of structured data by using sophisticated query optimization, columnar storage on high-performance local disks, and massively parallel data processing. Most results come back in seconds.

Parallel Processing Architecture


Parallel processing architecture

42

1. **Leader Node**:
    - **Role**: The leader node is the central brain of an Amazon Redshift cluster. It handles all communication with client programs (like the ones you use to query the database) and manages communication with the compute nodes (the workers).
    - **Tasks**: When you run a complex query, the leader node:
        - **Parses** the query to understand what needs to be done.
        - **Develops a plan** that breaks down the query into smaller steps.
        - **Compiles code** for each step and assigns this code to the compute nodes to execute.
2. **Compute Nodes**:
    - **Role**: These are the worker nodes that do the heavy lifting. The leader node gives them specific tasks (like running parts of the query), and they process the data.
    - **Process**: After completing their tasks, the compute nodes send the intermediate results back to the leader node.

    - **Final Aggregation**: The leader node takes all the intermediate results from the compute nodes, combines them, and produces the final output that is sent back to the client.

Cost and Pricing

- **Pay-as-You-Go**: Like other AWS services, you only pay for the resources you use, starting as low as 25 cents per hour.
- **At Scale**: If you're using Amazon Redshift at a larger scale, it can cost around $1,000 per terabyte per year (with specific pricing plans like the 3-Year Partial Upfront Reserved Instance).

Redshift Spectrum

- **What It Does**: Redshift Spectrum allows you to run queries on huge amounts of data stored directly in Amazon S3, without needing to load that data into Redshift. This is useful for analyzing large datasets that don't fit into your main Redshift cluster.

Automation and Scaling

Many routine administrative tasks, such as managing, monitoring, and scaling your Amazon Redshift cluster, can be automated. This means you don’t have to spend time manually handling these tasks, allowing you to focus on analyzing your data and running your business.

Amazon Redshift is designed to easily scale according to your needs. This scalability is built into the system. You can increase or decrease the size of your Redshift cluster as your data and processing needs change. This can be done quickly with just a few clicks in the AWS Management Console.

AWS places a strong emphasis on security. In Amazon Redshift, security features are built-in to ensure that your data is protected. Your data is encrypted both when it is stored (at rest) and when it is being transferred (in transit), providing strong protection against unauthorized access.

Compatibility

Amazon Redshift is compatible with the tools that you already know and use. Amazon Redshift supports standard SQL. Italso provides high-performance Java

Database Connectivity (JDBC) and Open Database Connectivity (ODBC) connectors, which enable you to use the SQL clients and BI tools of your choice

Amazon Redshift Use cases

1. This use case describes how many businesses, both large and small, benefit from using Amazon Redshift as their data warehouse solution. Large companies with massive amounts of data often struggle to manage their existing systems, while smaller companies may lack the resources to set up and maintain a complex data warehouse. Amazon Redshift solves these problems by allowing businesses to start small, scale as needed, and avoid complicated IT processes. Since Redshift is a managed service, it takes care of the setup and maintenance, allowing companies to focus on analyzing their data rather than dealing with the technical aspects of running a data warehouse.
2. This use case highlights how Software as a Service (SaaS) providers benefit from using Amazon Redshift. SaaS customers can easily scale their data warehouse capacity as demand grows, adding analytical features to their applications without worrying about managing complex infrastructure. Some providers even create a separate Redshift cluster for each customer, using tags to simplify service level agreements (SLAs) and billing. Overall, Amazon Redshift helps SaaS providers reduce the costs associated with hardware and software, while offering powerful data analytics capabilities.

Summary

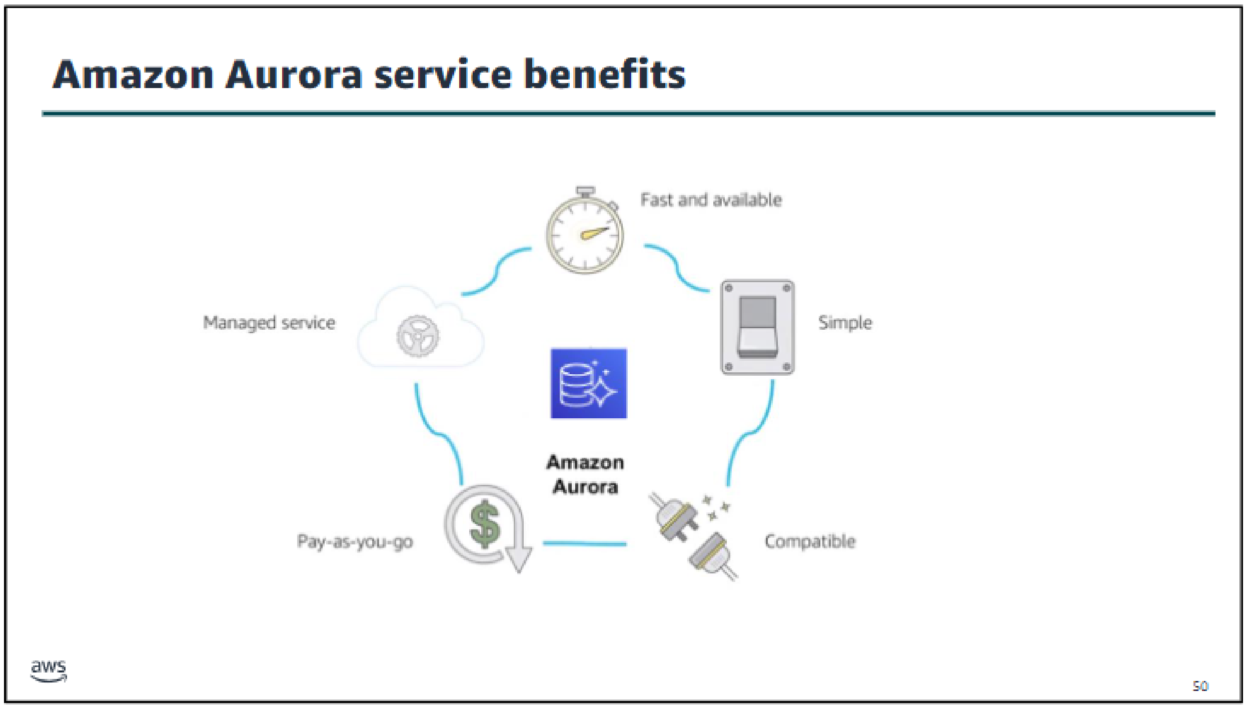
Amazon Redshift is a powerful, fully managed data warehouse service that grows with your business. As your needs increase, you can easily scale by adding more nodes (computing resources) without any downtime. Redshift automatically integrates these nodes into your system and redistributes your data to maintain high performance.

Redshift is designed to perform well consistently, thanks to its use of **columnar storage** (storing data in columns rather than rows) and **massively parallel processing** (splitting tasks across multiple nodes). It also automatically monitors your system, backs up your data, and offers built-in encryption for security, which you can enable as needed

Amazon Aurora

Amazon Aurora is a MySQL- and PostgreSQL-compatible relational database built for the cloud. It combines the performance and availability of high-end commercial databases with the simplicity and cost-effectiveness of open-source databases. Using Amazon Aurora can reduce your database costs while improving the reliability and availability of the database. As a fully managed service, Aurora automates time-consuming tasks such as provisioning, patching, backup, recovery, failure detection, and repair.

Amazon Aurora Service Benefits



Amazon Aurora service benefits

- Amazon Aurora is highly available with a fast, distributed storage subsystem.
- It is straightforward to set up and uses SQL queries.
- It is designed to be drop-in compatible with MySQL and PostgreSQL, allowing use of existing database tools with minimal changes.
- Aurora operates on a pay-as-you-go basis, so you only pay for the services and features you use.

- It integrates with AWS Database Migration Service (AWS DMS) and the AWS Schema Conversion Tool to help migrate datasets to Aurora.

High Availability

Why might you use Amazon Aurora over other options, like SQL with Amazon RDS? Most of that decision involves the high availability and resilient design that Amazon Aurora offers. Amazon Aurora is designed to be highly available: it stores multiple copies of your data across multiple Availability Zones with continuous backups to Amazon S3. Amazon Aurora can use up to 15 read replicas can be used to reduce the possibility of losing your data. Additionally, Amazon Aurora is designed for instant crash recovery if your primary database becomes unhealthy.

Resilient Design

After a database crash, Amazon Aurora does not need to replay the redo log from the last database checkpoint. Instead, it performs this on every read operation. This reduces the restart time after a database crash to less than 60 seconds in most cases. With Amazon Aurora, the buffer cache is moved out of the database process, which makes it available immediately at restart. This reduces the need for you to throttle access until the cache is repopulated to avoid brownouts.

Thank you

2024/8/24

(Bhadra 8, 2081)

Asmita Ojha