Aula 10 – Bancos de Dados Serverless: NoSQL com DynamoDB

No mundo da computação em nuvem, a agilidade e a escalabilidade são moedas de ouro. Imagine que você está construindo um aplicativo que precisa lidar com milhões de usuários simultaneamente, ou talvez um sistema de IoT que coleta dados de milhares de sensores a cada segundo. Em cenários como esses, a forma como você armazena e acessa seus dados se torna um fator crítico para o sucesso. É aqui que os bancos de dados serverless e, em particular, as soluções NoSQL, entram em cena, oferecendo uma nova perspectiva sobre gerenciamento de dados.

Esta aula foi cuidadosamente elaborada para desmistificar o universo dos bancos de dados NoSQL, com um foco especial no Amazon DynamoDB. Nosso objetivo é que, ao final deste módulo, você não apenas compreenda os conceitos fundamentais por trás dessas tecnologias, mas também seja capaz de discernir quando e como aplicá-las em seus projetos. Vamos explorar desde as bases do NoSQL até as nuances do DynamoDB, incluindo sua arquitetura, modelos de capacidade e as melhores práticas para modelagem de dados. Prepare-se para uma jornada que transformará sua visão sobre persistência de dados na era serverless.

Fundamentos

A Revolução dos Dados: Por Que Precisamos do NoSQL?

Por muito tempo, os bancos de dados relacionais (SQL) foram a espinha dorsal da maioria das aplicações. Com sua estrutura rígida de tabelas, linhas e colunas, e a garantia de consistência transacional (ACID), eles se mostraram robustos para muitos casos de uso. No entanto, o cenário digital mudou drasticamente. A explosão de dados não estruturados e semiestruturados, a necessidade de escalabilidade horizontal massiva e a demanda por baixa latência em aplicações globais começaram a expor as limitações dos modelos relacionais tradicionais.

> Imagine que você está organizando uma biblioteca. Um sistema SQL seria como um catálogo meticuloso, onde cada livro tem um lugar exato, com campos predefinidos para título, autor, ano, etc. É ótimo para encontrar um livro específico, mas e se você quiser adicionar um novo tipo de mídia, como um podcast, que não se encaixa perfeitamente nas categorias existentes?

É nesse contexto que os bancos de dados NoSQL (Not Only SQL) emergiram como uma alternativa poderosa. Eles foram projetados para atender às demandas de aplicações modernas, oferecendo flexibilidade de esquema, alta disponibilidade e escalabilidade massiva. Em vez de forçar os dados em um modelo relacional, o NoSQL abraça a diversidade, permitindo que os dados sejam armazenados em formatos mais adequados à sua natureza, seja como documentos, grafos, pares chave-valor ou colunas largas. Essa adaptabilidade é crucial para lidar com a velocidade e o volume de dados que caracterizam as aplicações de hoje.

Tipos de NoSQL

Desvendando os Tipos de Bancos de Dados NoSQL

A beleza do NoSQL reside em sua diversidade. Não existe um único "tipo" de banco de dados NoSQL; na verdade, é uma família de tecnologias, cada uma otimizada para diferentes padrões de acesso e tipos de dados. Compreender essas categorias é fundamental para escolher a ferramenta certa para o trabalho. Pense nisso como ter uma caixa de ferramentas: você não usaria um martelo para apertar um parafuso, certo? Da mesma forma, cada tipo de NoSQL tem seu propósito.

Chave-Valor

São os mais simples. Cada item é armazenado como um par chave-valor, como um dicionário gigante. A chave é única e usada para recuperar o valor associado. São extremamente rápidos para operações de leitura e escrita simples.

Exemplo: Carrinho de compras online, onde a chave é o ID do usuário e o valor é uma lista de itens.

Documentos

Armazenam dados em documentos semiestruturados, geralmente em formatos como JSON ou BSON. São flexíveis, pois os documentos podem ter estruturas diferentes.

Exemplo: Catálogos de produtos, perfis de usuários ou sistemas de gerenciamento de conteúdo.

Colunas Largas

Organizados em famílias de colunas, onde cada linha pode ter um conjunto diferente de colunas. São otimizados para grandes volumes de dados e consultas analíticas.

Exemplo: Sistema de monitoramento de sensores, onde cada sensor pode reportar diferentes métricas.

Grafos

Projetados para armazenar e navegar por dados altamente conectados. Eles representam entidades (nós) e as relações entre elas (arestas).

Exemplo: Redes sociais, sistemas de recomendação ou detecção de fraudes.

Dica importante: A escolha do tipo de banco de dados NoSQL depende diretamente dos requisitos da sua aplicação. Se você precisa de alta velocidade para dados simples, chave-valor pode ser a resposta. Se a flexibilidade de esquema e a capacidade de armazenar dados complexos são prioritárias, um banco de documentos se encaixa melhor.

Amazon DynamoDB

Introdução ao Amazon DynamoDB: O Coração Serverless do NoSQL

Com a ascensão da arquitetura serverless, a necessidade de bancos de dados que se integrem perfeitamente a esse modelo se tornou evidente. É aqui que o Amazon DynamoDB se destaca como uma solução NoSQL serverless totalmente gerenciada. Lançado pela AWS em 2012, o DynamoDB foi projetado desde o início para oferecer escalabilidade massiva, alta performance e alta disponibilidade, sem que você precise se preocupar com a infraestrutura subjacente.

> **Pense no DynamoDB como um serviço de armazenamento de dados "plug-and-play" de alta performance.** Você não precisa provisionar servidores, gerenciar patches, configurar clusters ou se preocupar com backups. A AWS cuida de tudo isso para você.

O DynamoDB é um banco de dados NoSQL do tipo **chave-valor e documento**. Isso significa que ele combina a simplicidade e a velocidade do modelo chave-valor com a flexibilidade de armazenar documentos JSON. Essa combinação o torna incrivelmente versátil, adequado para uma vasta gama de aplicações, desde microsserviços e APIs em tempo real até jogos, IoT e sistemas de publicidade. Sua capacidade de lidar com picos de tráfego imprevisíveis e escalar para milhões de requisições por segundo com latência de milissegundos de um dígito é o que o diferencia no cenário de bancos de dados serverless.

Estrutura de Dados

Componentes Essenciais do DynamoDB: Tabelas, Itens e Atributos

Para começar a trabalhar com o DynamoDB, é crucial entender seus componentes fundamentais. Embora seja um banco de dados NoSQL, ele ainda utiliza uma terminologia que, à primeira vista, pode parecer familiar, mas com nuances importantes. A forma como você organiza seus dados dentro desses componentes é a chave para otimizar performance e custo.

01

Tabelas

No DynamoDB, uma tabela é uma coleção de itens. Diferente de um banco de dados relacional, onde as tabelas têm um esquema fixo, as tabelas do DynamoDB são "schemaless" (sem esquema fixo). Isso significa que cada item dentro da mesma tabela pode ter um conjunto diferente de atributos, exceto pela chave primária, que é obrigatória.

Pense em uma tabela do DynamoDB como uma grande caixa onde você guarda diferentes tipos de objetos, mas todos eles têm uma etiqueta de identificação única.

02

Itens

Um item é um grupo de atributos que é unicamente identificável entre todos os outros itens em uma tabela. Em termos mais simples, um item é como uma "linha" em um banco de dados relacional, mas com a flexibilidade de ter atributos variados. Cada item deve ter uma chave primária, que o identifica de forma exclusiva.

Exemplo: Em uma tabela de "Produtos", um item pode ser um "Laptop", com atributos como "nome", "preço", "fabricante", "cor", etc.

03

Atributos

Um atributo é um dado fundamental que compõe um item. É como uma "coluna" em um banco de dados relacional, mas sem a necessidade de ser preenchido para todos os itens. Os atributos podem ser de diversos tipos, como strings, números, booleanos, listas ou mapas (objetos aninhados).

A flexibilidade dos atributos permite que você armazene dados complexos e aninhados dentro de um único item, o que é uma das grandes vantagens do modelo de documento do DynamoDB.

Vantagem chave: A ausência de um esquema fixo no nível da tabela oferece uma liberdade enorme para evoluir seus modelos de dados sem a necessidade de migrações complexas. Você pode adicionar novos atributos a itens existentes ou criar itens com estruturas completamente novas a qualquer momento.

Chaves Primárias

Chaves Primárias no DynamoDB: A Identidade dos Seus Dados

A chave primária é o elemento mais crítico na modelagem de dados do DynamoDB. Ela não apenas identifica unicamente cada item em uma tabela, mas também é fundamental para a forma como o DynamoDB distribui e acessa seus dados. Uma escolha inadequada da chave primária pode levar a problemas de performance, hotspots (partições sobrecarregadas) e custos elevados. É como o endereço de uma casa: se for mal definido, a correspondência pode não chegar ou demorar muito.

1. Chave de Partição

Também conhecida como "Hash Key". É um atributo simples que o DynamoDB usa para distribuir os dados entre suas partições de armazenamento. Quando você faz uma requisição para ler ou escrever um item, o DynamoDB usa a chave de partição para determinar em qual partição o item está armazenado.

Exemplo prático

Em uma tabela de "Pedidos", o **ID_do_Cliente** pode ser uma chave de partição. Todos os pedidos de um mesmo cliente estariam na mesma partição (ou próximas), o que é eficiente para buscar todos os pedidos de um cliente, mas pode sobrecarregar se um cliente tiver muitos pedidos e for acessado frequentemente.

Importante: Para garantir uma distribuição uniforme dos dados e evitar hotspots, é essencial que os valores da chave de partição sejam bem distribuídos e tenham alta cardinalidade (muitos valores únicos).

2. Chave Composta

Chave de Partição + Chave de Classificação. Também conhecida como "Composite Primary Key" ou "Hash-Range Key". Consiste em dois atributos: a chave de partição e uma chave de classificação. A chave de partição funciona como antes, distribuindo os dados. A chave de classificação, por sua vez, organiza os itens dentro de cada partição em ordem alfabética ou numérica.

Exemplo prático

Em uma tabela de "Pedidos", **ID_do_Cliente** pode ser a chave de partição e **Data_do_Pedido** a chave de classificação. Isso permite buscar todos os pedidos de um cliente em um determinado período, ou os pedidos mais recentes de um cliente.

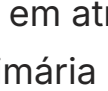
Benefício: Isso permite consultas mais flexíveis, como buscar todos os itens que compartilham a mesma chave de partição e que estão dentro de um determinado intervalo de valores da chave de classificação.

> **A escolha da chave primária é uma decisão de design crucial.** Ela deve refletir os padrões de acesso mais comuns da sua aplicação. Uma boa chave primária garante que as requisições sejam distribuídas uniformemente, evitando gargalos e maximizando a performance do seu banco de dados.

Índices Secundários

Índices Secundários: Flexibilizando Suas Consultas no DynamoDB

Embora a chave primária seja excelente para acessar itens diretamente ou por um intervalo específico, muitas vezes suas aplicações precisam consultar dados usando outros atributos que não fazem parte da chave primária. É aqui que os índices secundários entram em jogo, oferecendo flexibilidade adicional para suas consultas no DynamoDB. Eles são como índices em um livro: permitem que você encontre informações rapidamente, mesmo que não saiba o número da página exato.



Índice Secundário Global (GSI)

Um GSI é uma estrutura de dados completamente separada da tabela principal, com sua própria chave de partição e, opcionalmente, uma chave de classificação. Ele permite que você consulte a tabela usando atributos que não são a chave primária da tabela principal. Como é global, ele pode abranger todos os dados da tabela principal e ser consultado de forma independente.

Características principais:

- Pode ter uma chave de partição e chave de classificação diferentes da tabela principal
- Permite consultas em atributos que não são parte da chave primária da tabela
- É assíncrono: as atualizações na tabela principal são propagadas para o GSI com um pequeno atraso (consistência eventual)
- Possui suas próprias unidades de capacidade de leitura/escrita, que são cobradas separadamente

Exemplo: Em uma tabela de "Usuários" com ID_do_Usuário como chave primária, você pode criar um GSI com Email como chave de partição para buscar usuários pelo e-mail.

Índice Secundário Local (LSI)

Um LSI é uma estrutura de dados que tem a mesma chave de partição da tabela principal, mas uma chave de classificação diferente. Ele é "local" porque seu escopo é limitado a uma única partição da tabela principal.

Características principais:

- Deve ter a mesma chave de partição da tabela principal
- Permite uma chave de classificação diferente da tabela principal
- É síncrono: as atualizações na tabela principal são consistentes com o LSI (consistência forte)
- Não possui unidades de capacidade separadas; usa as da tabela principal
- Você pode criar até 5 LSIs por tabela

Exemplo: Em uma tabela de "Pedidos" com ID_do_Cliente (partição) e Data_do_Pedido (classificação), você pode criar um LSI com ID_do_Cliente (partição) e Status_do_Pedido (classificação) para buscar pedidos de um cliente por status.

A escolha entre GSI e LSI depende dos seus requisitos de consulta e consistência. GSIs são mais flexíveis para consultas globais e podem ter chaves completamente diferentes, mas oferecem consistência eventual. LSIs são para consultas dentro de uma mesma partição e oferecem consistência forte, mas são mais restritivos em termos de chaves. O uso inteligente de índices secundários é crucial para otimizar a performance das suas aplicações no DynamoDB.

Modelos de Capacidade

Modelo de Capacidade de Leitura/Escrita: Provisionado vs. Sob Demanda

Um dos aspectos mais importantes do DynamoDB, especialmente para gerenciar custos e performance, é o seu modelo de capacidade de leitura/escrita. Ele define como você paga pelo throughput (vazão) e como o DynamoDB aloca recursos para sua tabela. Entender a diferença entre os modos provisionado e sob demanda é fundamental para otimizar sua aplicação.

> **Imagine que você está alugando um carro.** Você tem duas opções: Aluguel por período fixo (Provisionado) - você paga um valor fixo por dia, independentemente de quanto você dirige. Ou Aluguel por quilometragem (Sob Demanda) - você paga apenas pelos quilômetros que realmente dirige.

Modo Provisionado

Neste modo, você especifica antecipadamente a quantidade de throughput que sua aplicação precisa em termos de Unidades de Capacidade de Leitura (RCUs) e Unidades de Capacidade de Escrita (WCUs).

- **1 RCU** = uma leitura fortemente consistente de 4 KB por segundo ou duas leituras eventualmente consistentes de 4 KB por segundo
- **1 WCU** = uma escrita de 1 KB por segundo
- Você paga por essas unidades provisionadas, independentemente de usá-las ou não

---

✔ Vantagens

Mais econômico para cargas de trabalho previsíveis e constantes. Permite otimizar custos se você souber exatamente o que precisa.

✖ Desvantagens

Requer monitoramento e ajuste manual (ou com Auto Scaling) para lidar com picos de tráfego, evitando throttling (limitação de requisições) ou desperdício de recursos.

Modo Sob Demanda

Neste modo, você não precisa especificar RCUs ou WCUs. O DynamoDB escala automaticamente sua capacidade para atender às suas necessidades de throughput.

- Você paga apenas pelas requisições de leitura e escrita que sua aplicação realmente executa
- Escalabilidade automática sem configuração
- Ideal para tráfego imprevisível

✔ Vantagens

Ideal para cargas de trabalho imprevisíveis, com picos de tráfego intensos ou para novas aplicações onde o padrão de uso ainda não é conhecido. Elimina a necessidade de gerenciar a capacidade.

✖ Desvantagens

Pode ser mais caro que o modo provisionado para cargas de trabalho constantes e previsíveis, pois o custo por unidade de requisição é geralmente mais alto.

Decisão estratégica: A escolha do modo de capacidade depende do perfil de uso da sua aplicação. Para cargas de trabalho estáveis e previsíveis, o modo provisionado com Auto Scaling pode ser a opção mais econômica. Para cargas de trabalho voláteis, com picos imprevisíveis ou para ambientes de desenvolvimento/teste, o modo sob demanda oferece simplicidade e flexibilidade.

Tendências

Tendências e o Futuro dos Bancos de Dados Serverless

O cenário da computação serverless está em constante evolução, e os bancos de dados serverless acompanham essa jornada, tornando-se cada vez mais robustos e integrados. As tendências atuais apontam para uma maior flexibilidade, capacidade de lidar com estados complexos e uma integração mais profunda com o ecossistema de desenvolvimento.

Function-as-a-Service (FaaS)

Inicialmente, funções serverless eram ideais para tarefas curtas e sem estado. No entanto, a demanda por tempos de execução mais longos e a necessidade de gerenciar estados de forma mais eficiente levaram a aprimoramentos significativos. Agora, é comum ver funções FaaS orquestrando fluxos de trabalho complexos e interagindo com bancos de dados serverless para persistir estados entre invocações.

Serverless Containers

Tecnologias como AWS Fargate e Google Cloud Run preenchem a lacuna entre a simplicidade do serverless e a flexibilidade dos contêineres. Elas permitem que você execute aplicações conteinerizadas sem gerenciar servidores subjacentes, combinando a portabilidade dos contêineres com a escalabilidade e o modelo de pagamento por uso do serverless.

Infraestrutura como Código (IaC)

Ferramentas como Serverless Framework e AWS SAM (Serverless Application Model) se tornaram padrões de mercado para automatizar o deploy e o gerenciamento de aplicações serverless, incluindo a criação e configuração de tabelas DynamoDB e seus índices. A automação do deploy de bancos de dados serverless, juntamente com o código da aplicação, é essencial para o ciclo de vida de desenvolvimento moderno.

Essas tendências reforçam a importância de dominar tecnologias como o DynamoDB. À medida que o serverless amadurece, a capacidade de construir e gerenciar sistemas de dados escaláveis e eficientes se torna um diferencial ainda maior para profissionais da área.

Comparativo

Quadro Comparativo: Modos de Capacidade do DynamoDB

Para consolidar o entendimento sobre os modos de capacidade do DynamoDB, veja um comparativo rápido:

Característica	Modo Provisionado	Modo Sob Demanda
Definição	Você especifica RCUs e WCUs antecipadamente.	O DynamoDB escala automaticamente.
Custo	Baseado nas unidades provisionadas (usadas ou não).	Baseado nas requisições reais de leitura/escrita.
Ideal para	Cargas de trabalho previsíveis e constantes.	Cargas de trabalho imprevisíveis, picos intensos.
Gerenciamento	Requer monitoramento e ajuste (manual ou Auto Scaling).	Não requer gerenciamento de capacidade.
Previsibilidade	Alta previsibilidade de custos.	Menor previsibilidade de custos (varia com uso).
Throttling	Pode ocorrer se a capacidade provisionada for excedida.	Raramente ocorre, a menos que haja um pico extremo.

Comparativo

Quadro Comparativo: Índices Secundários do DynamoDB

Característica	Índice Secundário Global (GSI)	Índice Secundário Local (LSI)
Chave de Partição	Pode ser diferente da tabela principal.	Deve ser a mesma da tabela principal.
Chave de Classificação	Pode ser diferente da tabela principal.	Pode ser diferente da tabela principal.
Consistência	Eventual (pequeno atraso na propagação).	Forte (síncrono com a tabela principal).
Capacidade	Possui suas próprias RCUs/WCUs (custo separado).	Usa as RCUs/WCUs da tabela principal.
Limite	Até 20 GSIs por tabela.	Até 5 LSIs por tabela.
Uso	Consultas em atributos não-chave primária, globais.	Consultas em atributos não-chave primária, dentro da mesma partição.

Exemplos Práticos

Chaves Primárias: Exemplos Práticos

Para ilustrar a importância da chave primária, vamos considerar alguns exemplos:

1

Tabela de Usuários

Requisito: Acessar usuários rapidamente pelo seu ID único.

Chave Primária: ID_Usuario (Chave de Partição).

Explicação: Cada usuário tem um ID único. Usar ID_Usuario como chave de partição garante que as requisições sejam distribuídas e que a busca por um usuário específico seja muito eficiente.

2

Tabela de Pedidos de E-commerce

Requisito: Buscar todos os pedidos de um cliente e também os pedidos de um cliente em um determinado período.

Chave Primária: ID_Cliente (Chave de Partição) e Data_Pedido (Chave de Classificação).

Explicação: ID_Cliente agrupa todos os pedidos de um cliente na mesma partição. Data_Pedido permite ordenar e filtrar os pedidos por data, facilitando buscas por intervalos de tempo (ex: "todos os pedidos do cliente X em 2024").

3

Tabela de Dados de Sensores IoT

Requisito: Armazenar dados de milhares de sensores e buscar leituras de um sensor específico em um intervalo de tempo.

Chave Primária: ID_Sensor (Chave de Partição) e Timestamp (Chave de Classificação).

Explicação: ID_Sensor garante que os dados de cada sensor sejam agrupados. Timestamp permite buscar as leituras de um sensor em um período específico (ex: "todas as leituras do sensor Y na última hora").

Conclusão: A escolha da chave primária é a decisão mais impactante na performance e escalabilidade da sua tabela DynamoDB. Dedique tempo para analisar seus padrões de acesso e projetar chaves que os atendam de forma eficiente.

Exemplos Práticos

Índices Secundários: Exemplos Práticos

Para exemplificar o uso de índices secundários, vamos continuar com os cenários anteriores:

Cenário 1: Tabela de Usuários (com GSI)

Tabela Principal: ID_Usuario (PK).

Requisito Adicional: Buscar usuários pelo Email ou Nome_Usuario.

Solução: Criar um **GSI** com Email como Chave de Partição (ou Nome_Usuario como PK).

Explicação: Como Email não é a chave primária da tabela principal, um GSI permite essa consulta. O GSI terá sua própria capacidade e consistência eventual, mas resolve o problema de acesso secundário.

Cenário 2: Tabela de Pedidos (com LSI)

Tabela Principal: ID_Cliente (PK), Data_Pedido (SK).

Requisito Adicional: Para um dado cliente, buscar pedidos por Status_Pedido (ex: "pendente", "entregue").

Solução: Criar um **LSI** com ID_Cliente (PK - igual à tabela principal) e Status_Pedido (SK).

Explicação: Este LSI permite que, dentro da partição de um cliente, os pedidos sejam ordenados e filtrados por status, sem precisar escanear todos os pedidos do cliente. Ele compartilha a capacidade da tabela principal e oferece consistência forte.

Cenário 3: Sensores IoT (com GSI)

Tabela Principal: ID_Sensor (PK), Timestamp (SK).

Requisito Adicional: Buscar todas as leituras que excederam um Limite_Alerta em um determinado período, independentemente do sensor.

Solução: Criar um **GSI** com Tipo_Alerta (PK) e Timestamp (SK).

Explicação: O Tipo_Alerta (ex: "Temperatura_Alta", "Umidade_Baixa") pode ser um atributo que aparece em alguns itens. Um GSI permite agrupar e consultar esses alertas globalmente, sem ter que saber o ID_Sensor de antemão.

O uso estratégico de GSIs e LSIs é crucial para suportar os diversos padrões de acesso que uma aplicação moderna pode exigir, garantindo que as consultas sejam eficientes e escaláveis.

Casos de Uso

Em Prática: Onde o DynamoDB Brilha

O Amazon DynamoDB é uma escolha excelente para uma variedade de casos de uso que exigem alta performance, escalabilidade e disponibilidade, sem a complexidade de gerenciar servidores. Ele brilha em cenários onde a latência é crítica e o volume de dados pode ser massivo e imprevisível.

Microsserviços e APIs

Sua capacidade de fornecer latência de milissegundos de um dígito o torna ideal para backends de APIs e microsserviços que precisam responder rapidamente.

Aplicações Web e Mobile

Para perfis de usuário, carrinhos de compras, histórico de pedidos e outras funcionalidades que exigem acesso rápido a dados de sessão ou de usuário.

Jogos Online

Gerenciamento de placares de líderes, perfis de jogadores, inventários e dados de sessão em tempo real.

IoT (Internet das Coisas)

Armazenamento de dados de sensores em alta velocidade e volume, permitindo análises em tempo real e monitoramento.

Ad Tech

Armazenamento de perfis de usuários para segmentação de anúncios, dados de cliques e impressões.

Sistemas de Cache

Embora não seja um cache primário, pode ser usado para armazenar dados frequentemente acessados para reduzir a carga em outros bancos de dados.

A flexibilidade do DynamoDB em lidar com diferentes tipos de dados e a facilidade de escalar o tornam uma ferramenta poderosa no arsenal de qualquer desenvolvedor serverless.

Avaliação

Autoavaliação

1

Questão 1

Qual das seguintes características é uma vantagem principal dos bancos de dados NoSQL em comparação com os relacionais para aplicações modernas?

1. Esquema rígido e garantia ACID completa.
2. Escalabilidade horizontal massiva e flexibilidade de esquema.
3. Linguagem de consulta SQL padronizada.
4. Melhor para transações complexas e joins.

2

Questão 2

No Amazon DynamoDB, qual é a principal função da Chave de Partição?

1. Definir a ordem de classificação dos itens dentro de uma partição.
2. Distribuir os dados entre as partições de armazenamento para escalabilidade.
3. Garantir a consistência forte em todas as leituras.
4. Armazenar atributos complexos e aninhados.

3

Questão 3

Você está desenvolvendo uma aplicação que precisa lidar com picos de tráfego imprevisíveis e não quer se preocupar em provisionar capacidade. Qual modo de capacidade do DynamoDB seria mais adequado?

1. Modo de Capacidade Provisionada.
2. Modo de Capacidade Sob Demanda.
3. Modo de Capacidade Elástica.
4. Modo de Capacidade Fixa.

4

Questão 4

Qual tipo de índice secundário no DynamoDB deve ter a mesma Chave de Partição da tabela principal e oferece consistência forte?

1. Índice Secundário Global (GSI).
2. Índice Secundário Local (LSI).
3. Índice Primário Composto.
4. Índice de Coluna Larga.

Gabarito: 1. b) | 2. b) | 3. b) | 4. b)

Questão Discursiva

> Explique a importância da desnormalização e do "Single-Table Design" na modelagem de dados para o Amazon DynamoDB, contrastando com a abordagem de normalização em bancos de dados relacionais.

Aplicação Prática

Em Prática

Desafio Prático

Para aplicar o conhecimento desta aula, comece identificando um pequeno projeto ou funcionalidade em que você possa usar o DynamoDB. Pense em um cenário onde a escalabilidade e a performance são cruciais, como:

- Um contador de visitas para um blog
- Um sistema de gerenciamento de tarefas simples
- Um registro de eventos de uma aplicação

Passos Recomendados

1. Defina seus padrões de acesso primários e secundários
2. Projete sua chave primária
3. Determine se precisa de índices secundários
4. Experimente os modos de capacidade provisionado e sob demanda
5. Analise o impacto no custo e na performance

Próximos Passos

Próxima Aula

Aula 11 – Armazenamento de Objetos com AWS S3

Na próxima aula, exploraremos outra peça fundamental da arquitetura serverless: o armazenamento de objetos. Aprenderemos como o Amazon S3 oferece uma solução de armazenamento altamente durável, escalável e de baixo custo para uma vasta gama de dados, desde backups até hospedagem de sites estáticos.

Recursos Adicionais

- **Documentação Oficial do Amazon DynamoDB**

  Para detalhes técnicos e guias de melhores práticas.

- **AWS Well-Architected Framework (Serverless Lens)**

  Para entender como projetar aplicações serverless robustas e eficientes.

- **Artigos e Tutoriais sobre Modelagem de Dados no DynamoDB**

  Para aprofundar-se em padrões avançados de design.

NOTA IMPORTANTE: As informações regulatórias/legais/técnicas desta aula estão atualizadas até 2025. Consulte sempre fontes oficiais para verificar alterações.