



training and
certification

**Build Modern Applications with AWS NoSQL
Databases (KO)**

Student Guide

버전 1.1.7

200-DBNSQL-11-KO-SG

© 2025, Amazon Web Services, Inc. or its affiliates. All rights reserved.

본 내용은 Amazon Web Services, Inc.의 사전 서면 허가 없이 전체 또는 일부를 복제하거나 재배포할 수 없습니다. 상업적인 복제, 임대 또는 판매는 금지됩니다.

본 과정에 대한 수정 사항이나 피드백, 문의사항이 있으면
<https://support.aws.amazon.com/#/contacts/aws-training>
을 통해 연락해 주십시오.

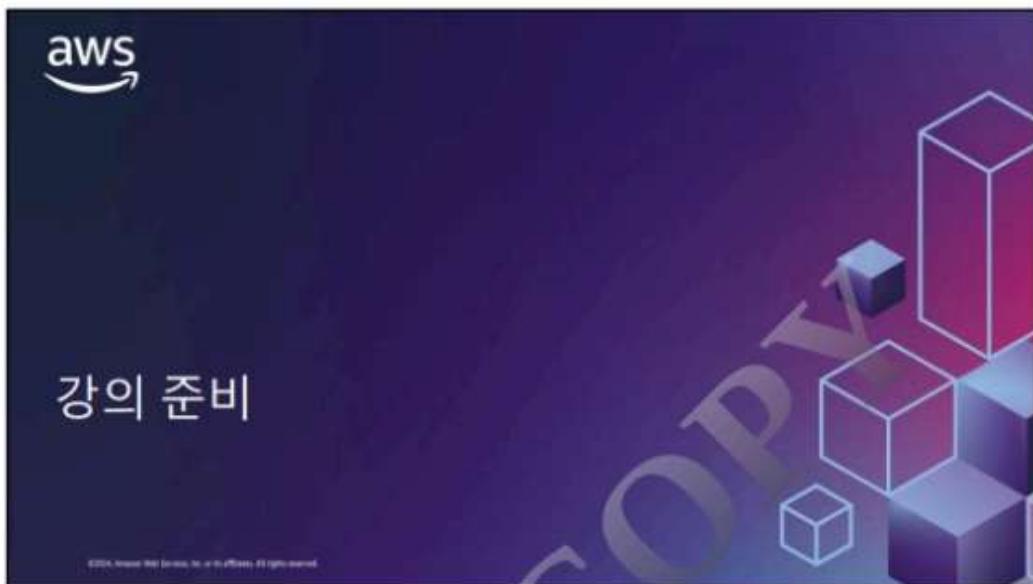
모든 상표는 해당 소유자의 자산입니다.

목차

모듈 0: 과정 소개	4
모듈 1: 과정 소개	15
모듈 2: 고급 Amazon DynamoDB 개념	32
모듈 3: 고급 Amazon DocumentDB 개념	77
모듈 4: 고급 ElastiCache for Redis 개념	120
모듈 5: 과정 요약	184

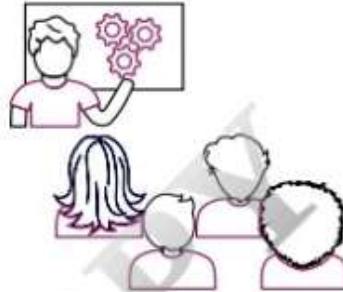


DO NOT COPY
Ssleeeee1@gmail.com



안내 사항

- 휴식 및 점심 시간
- 보안
- 휴대전화
- 가상 강의실 기능
 - 오디오
 - 채팅
 - 질문하기



aws

© 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

강사는 휴식 및 점심 시간, 수강생의 보안 및 휴대전화 정책에 대한 기대치를 검토합니다. 가상으로 교육에 참여하는 경우 담당 강사와 함께 가상 강의실 소프트웨어 기능을 검토하십시오. 담당 강사에게 이러한 기능을 사용하여 강의에 참여하는 방법에 대한 설명을 들을 수 있습니다.

선행 조건

- 클라우드 컴퓨팅 개념에 대한 지식
- Key-value 데이터베이스의 데이터 모델링을 비롯한 Amazon DynamoDB 디자인 실무 경험
- Amazon DocumentDB(MongoDB 호환) 실무 경험
- Amazon ElastiCache for Redis 실무 경험
- AWS Lambda 및 Amazon API Gateway 데이터베이스 서비스에 대한 지식
- Python 실무 지식
- 관계형 또는 NoSQL 데이터베이스의 데이터베이스 디자인 개념 또는 데이터 모델링 실무 지식



© 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

이 과정을 시작하기 전에 몇 가지 사전 지식을 갖추는 것이 좋습니다.

가이드와 실습 환경에 액세스하기 위한 등록

AWS Builder Labs에 등록해야 합니다.

- 환영 이메일의 등록 정보를 참조하십시오.



AWS

© 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

받은 편지함에서 강사가 보낸 환영 이메일을 확인하십시오. 환영 이메일에서 강의의 고유한 수강생 등록 URL을 찾을 수 있습니다. 해당 URL 링크를 사용하여 계정을 생성하거나 기존 AWS Builder Lab 계정에 로그인합니다. AWS Builder Lab에서 실습 환경, 실습 가이드 및 수강생 가이드에 액세스할 수 있습니다.

Labs

Lab will be made available by instructor.

Deploy Now! **Download Guide**

Amazon Elastic Kubernetes Service (Amazon EKS) for Developers - Lab 1: Deploying Kubernetes Pods

In this lab, you access an Amazon EKS cluster using kubectl. This command will allow you to implement Kubernetes application resources, get it directly on or a Kubernetes pod, and delete the Kubernetes application.

Amazon Elastic Kubernetes Service (Amazon EKS) for Developers - Lab 2: Deploying applications with Helm

This lab demonstrates the process of downloading and installing Helm v3, the creation of a private Helm chart repository in Amazon Simple Storage Service (Amazon S3), download package, and push the chart to Amazon S3.

Sign In

New User? [CREATE A NEW USER ACCOUNT](#)

Email

Password

Forgot Password?

Sign In

Powered by [VitalSource](#)

이제 AWS Builder Labs에 로그인되었을 것입니다. 여기서 실습 가이드 및 수강생 가이드에 액세스할 수 있습니다. 가이드는 eVantage Bookshelf(VitalSource)에 있습니다. 실습 가이드 및 수강생 가이드 버튼은 AWS Builder Labs 대시보드의 오른쪽 상단 모서리에 있습니다. 강의가 시작되기 전에는 실습과 버튼이 회색으로 비활성화되어 있습니다.

강의가 시작되면 두 버튼 중 하나를 선택하여 가이드에 액세스하십시오. 기존 eVantage Bookshelf(VitalSource) 계정에 로그인하거나 새 계정을 생성하라는 메시지가 표시됩니다. eVantage Bookshelf(VitalSource)에 로그인하면 강의의 수강생 및 실습 가이드에 액세스할 수 있습니다. 가이드는 온라인으로 액세스하거나 다운로드할 수 있습니다. 해당 가이드를 참조하여 과정을 진행하고 교육 후에는 가이드를 참조용으로 활용하십시오.

실습 요구 사항

- 다음 운영 체제를 실행하는 컴퓨터:
 - Windows
 - macOS
 - Linux: Ubuntu, SUSE 또는 Red Hat
- 권장 웹 브라우저:
 - Google Chrome
 - Mozilla Firefox
 - Microsoft Edge
- HTTPS를 사용하여 인터넷을 탐색할 수 있는 신뢰할 수 있는 인터넷 연결
- AWS Builder Labs에 등록:
 - 광고 및 스크립트 차단 기능 해제



© 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

과정을 진행하기 전에 실습 요구 사항을 확인하십시오.



DO NOT COPY
ssleeeee1@gmail.com

과정 목표

- AWS 목적별 NoSQL 데이터베이스를 사용하여 현대적 클라우드 중심 애플리케이션 빌드
- Key-value, 문서 및 메모리 내 데이터 범주를 처리하는 AWS 목적별 데이터베이스를 사용하는 솔루션 설명
- 비즈니스 사용 사례를 분석하고 Amazon DynamoDB, Amazon DocumentDB 및 Amazon ElastiCache의 고급 기능을 적용하여 확장 가능 솔루션 구현
- 변경 스트림 및 AWS Lambda를 사용하여 이벤트 중심 아키텍처 구현



© 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

어젠다 개요

모듈 또는 실습	제목
모듈 1	NoSQL 데이터베이스 사용 사례 분석
모듈 2	고급 Amazon DynamoDB 개념
실습 1	Amazon DynamoDB 테이블, 인덱스 및 변경 스트림을 사용하여 플랫 및 이동 데이터 관리 시스템 구현
점심 시간	
모듈 3	고급 Amazon DocumentDB 개념
실습 2	Amazon DocumentDB에서 사용자 프로파일 데이터 관리 워크로드 구현 및 최적화
모듈 4	고급 Amazon ElastiCache for Redis 개념
실습 3	Amazon ElastiCache for Redis를 사용하여 지리 공간 자전거 검색, 사용자 프로파일 캐싱 및 순위표 구현
모듈 5	과정 요약



© 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

강사는 과정 어젠다에 대한 기대치를 설정합니다. 모듈과 실습 사이에 휴식 시간이 주어집니다. 모듈과 실습 사이에 휴식 시간이 더 필요하면 강사에게 문의하십시오.







어젠다

aws

- AnyCompany BikeShare 비즈니스 개요
 - 비즈니스 개요, 목표 및 도전 과제
 - 주요 프로젝트 워크로드
- AWS NoSQL 데이터베이스 포트폴리오
- 디자인 결정

이 모듈에서 살펴보는 주제를 검토하십시오.



먼저 오늘 구축할 애플리케이션을 이해하기 위해 가상의 회사인 AnyCompany BikeShare 조직에 대해 알아봅니다.

비즈니스 컨텍스트

AnyCompany BikeShare

프로파일

AnyCompany BikeShare는 운송 회사입니다.

현재 상황

이 회사는 자전거와 스쿠터 공유를 위한 새로운 모바일 앱을 구축하고 있습니다. 사용자는 근처의 자산을 요청하여 도시를 이동할 수 있습니다.

목표

고성능 운송 모바일 앱을 대규모로 지원하는 워크로드를 개발하고 구축합니다.

도전 과제

AnyCompany BikeShare는 성장 지향적이며, 새로운 시장에서는 회사의 모바일 앱 사용을 예측할 수 없습니다. 이 회사는 속성이 다른 여러 운송 자산을 관리합니다. 거의 실시간으로 데이터 전달이 이루어져야 합니다.


© 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

AnyCompany BikeShare는 새롭게 성장 중인 도시 시장에 자전거 및 스쿠터 공유 서비스를 제공하려는 스타트업 운송 회사입니다. 이 회사는 인구 100,000명의 한 도시에서 사업을 시작할 예정입니다.

여러분은 데이터베이스 개발자로 채용되었으며, 모바일 앱 개발 팀 소속입니다. 여러분은 AWS 데이터베이스 서비스에 대한 기존 지식을 활용하여 강력하고 탄력적이면서도 성능이 뛰어나고 확장 가능한 데이터베이스를 구축해야 합니다.

AnyCompany BikeShare의 경영진은 여러분에게 앱 개발 단계에서 극복해야 하는 주요 도전 과제를 주었습니다.

- AnyCompany Bikeshare는 성장 지향적입니다.
- 새로운 시장에 진출할 때에는 앱 사용을 예측하기 어렵습니다.
- 이 회사는 속성이 다른 여러 자산을 관리할 계획입니다.
- 성능이 중요합니다. 이 회사는 비용을 최적화하면서도 프리미엄 경험을 제공하여 신규 고객의 신뢰를 얻고자 합니다.

워크로드 솔루션 개요



플릿 관리



이동 관리



사용자 프로파일



순위표

관계형 데이터베이스 서비스는 이러한 워크로드에 적합하지 않을 수 있습니다.


© 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

앱 개발 팀에는 플릿 관리, 이동 관리, 사용자 프로파일, 순위표의 네 가지 주요 비즈니스 워크로드가 제공되었습니다.

플릿 관리

AnyCompany BikeShare는 자전거 및 스쿠터 자산 정보를 저장할 프라이머리 데이터베이스가 필요합니다. 여기에는 수동 페달 자전거, 전기 자전거 또는 전기 스쿠터가 포함될 수 있습니다. 각 디바이스의 부품이 서로 다를 수 있습니다. 또한 각 자산의 서비스 수리 내역 및 사용을 관리해야 합니다. 위치를 검색하고 추적할 수 있도록 각 자산의 현재 위치를 저장해야 합니다.

이동 관리

각 사용자의 이동 기록을 중앙 데이터베이스에 저장해야 합니다. 나중에 고객 충성도 및 기타 지표를 분석할 수 있도록 이동이 추적됩니다.

사용자 프로파일

프로파일 데이터는 이름, 이메일, 결제 방법, 멤버십 등의 정보를 저장합니다.

순위표

AnyCompany BikeShare는 매달 최우수 고객을 추적합니다. 이러한 순위표는 나중에 마케팅 캠페인 대상을 찾거나 앱 사용 증가 및 수익 창출에 대한 인센티브를 제공하는 데 사용할 수 있습니다.

다음과 같은 워크로드를 개략적으로 생각해 보십시오. 물리적 속성이 서로 다른 자산(자전거 및 스쿠터)이 플릿에 포함되지만, 이러한 자산을 동일한 데이터베이스에 저장하려 합니다. 사용자 프로파일에서 속성이 서로 다른 여러 결제 방법이 있을 수 있습니다. 비관계형 데이터베이스는 이러한 사용 사례에 점점 더 적합해지고 있습니다.



다음으로, 이 모듈의 후반부에서 데이터베이스 옵션을 이해할 수 있도록 AWS NoSQL 솔루션 포트폴리오를 검토합니다.

AWS NoSQL 목적별 데이터베이스

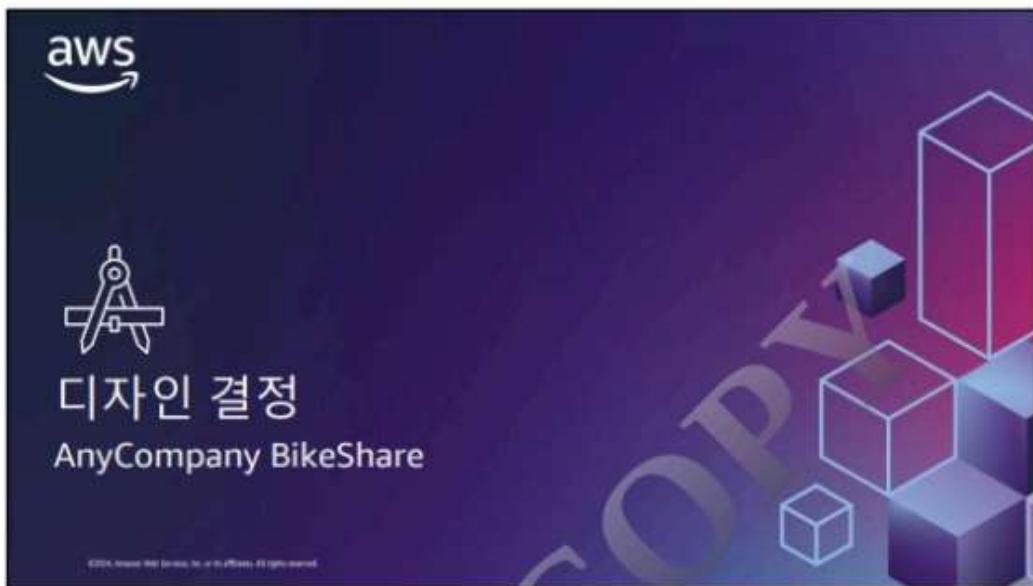
유형	예	AWS 서비스
Key-value	높은 트래픽의 웹 애플리케이션, 전자 상거래 시스템, 게이밍 애플리케이션	Amazon DynamoDB
문서	콘텐츠 관리, 카탈로그, 사용자 프로파일	Amazon DocumentDB
인 메모리	캐싱, 세션 관리, 게이밍 순위표, 지리 공간 애플리케이션	Amazon ElastiCache Amazon MemoryDB for Redis
와이드 컬럼	장비 관리 및 경로 최적화에 사용하는 대규모 산업용 앱	Amazon Keyspaces(Apache Cassandra™)
그래프	사기 행위 탐지, 소셜 네트워킹, 추천 엔진	Amazon Neptune
시계열	사물인터넷(IoT) 애플리케이션, DevOps, 산업용 원격 측정	Amazon Timestream
원장	레코드 시스템, 공급망, 등록, 은행 거래	Amazon QLDB



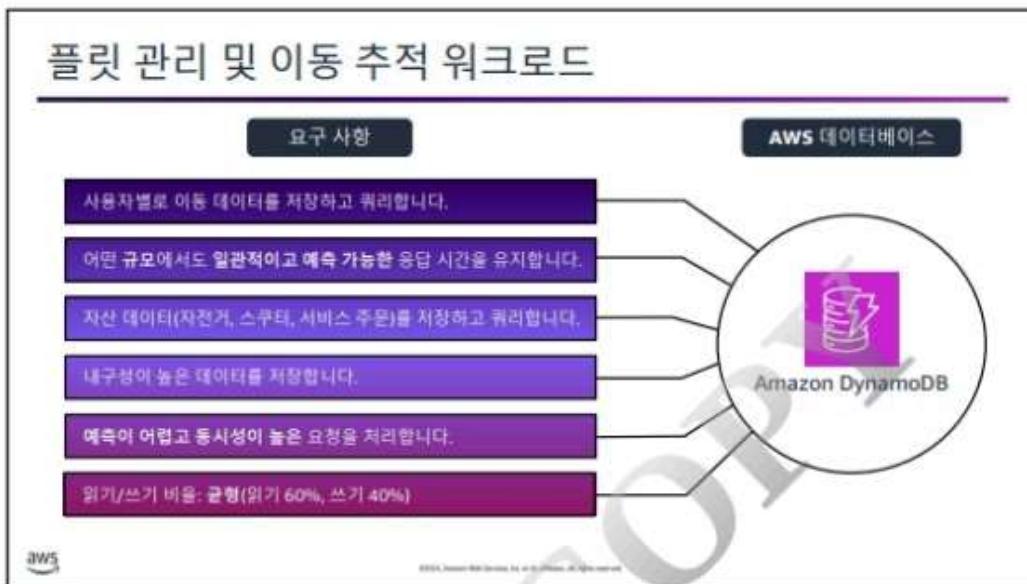
© 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

AWS는 여러분과 여러분의 고객에게 다양한 요구 사항에 적합한 데이터베이스를 제공합니다. 위의 표에서 각 데이터 형식, 해당 데이터 형식의 예시, 해당 사용 사례를 지원하는데 가장 적합한 AWS 서비스를 검토할 수 있습니다. 많은 솔루션이 해당 사용 사례에 사용할 수 있는 데이터베이스를 여러 개 제공한다는 것을 기억해야 합니다.

AWS 목적별 데이터베이스 포트폴리오에 대한 자세한 내용은 'AWS 클라우드 데이터베이스'(<https://aws.amazon.com/products/databases/>)를 참조하십시오.



AWS NoSQL 데이터베이스에 대해 알고 있는 지식을 활용하면 모바일 앱 빌드에 대한 AnyCompany BikeShare의 디자인 결정을 이해하는 데 도움이 됩니다.



모바일 앱의 플릿 관리 및 이동 추적 요구 사항을 검토하십시오.

이러한 데이터베이스의 핵심 목표는 플릿 및 이동 데이터를 지속적으로 저장하고 가져오는 것입니다. AWS Well-Architected Framework에서 정의하는 **내구성이**란 예상치 못한 이벤트가 발생하더라도 계속해서 책임을 이행할 수 있는 능력입니다. 내구성이 우수한 데이터베이스는 데이터를 손실 없이 안정적으로 저장합니다.

AWS 관리형 데이터베이스 서비스는 데이터 내구성을 높이는 기능을 제공합니다. 디자인 팀은 앱의 고유한 액세스 패턴을 이해하여 적절한 데이터베이스를 선택해야 합니다.

플릿 관리 및 이동 추적을 통해 디자인 팀은 다음 사항을 고려합니다.

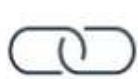
- 앱에서 급증하는 높은 동시성 트래픽을 처리해야 합니다.
- 데이터베이스가 어떤 규모에서도 일관된 응답 시간을 유지해야 합니다.
- 플릿 업그레이드 및 새로운 앱 기능 요청에는 유연한 데이터 스키마가 필요합니다.
- 읽기 트래픽은 60%, 쓰기 트래픽은 40%로 예상됩니다.

최종적으로 플릿 관리 및 이동 추적 워크로드에 적합한 프라이머리 데이터베이스로 Amazon DynamoDB가 선택되었습니다.

Amazon DynamoDB

어떤 규모에서도 빠르고 유연한 Key-value 데이터베이스

DynamoDB의 용도:



잘 정의된 검색을 통해
Key-value 액세스 패턴
최적화



높은 동시성을 처리할
때 예측 가능하고
일관된 응답 시간



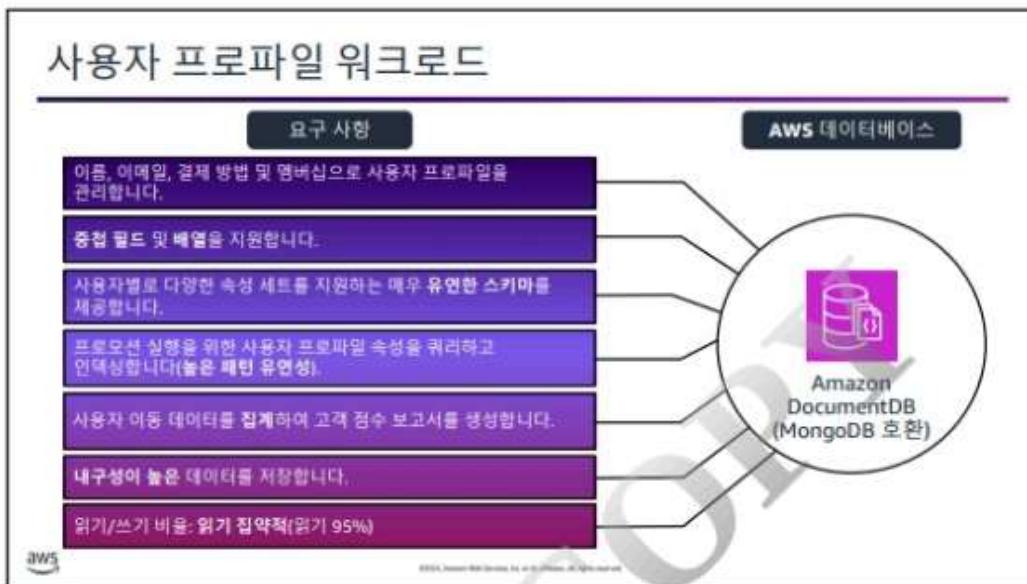
사실상 무제한의
처리량 및 스토리지



© 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Amazon DynamoDB는 완전관리형 NoSQL 데이터베이스 서비스로서 원활한 확장성과 함께 빠르고 예측 가능한 성능을 제공합니다.

이 과정에서는 DynamoDB를 사용하여 자산 및 이동의 Key-value 데이터를 저장합니다. DynamoDB는 높은 동시성 요구 사항을 충족합니다. 확장성이 뛰어난 데이터베이스이며 예측할 수 없는 데이터 액세스 패턴에 사용할 수 있습니다.



모바일 앱의 사용자 프로파일 워크로드 요구 사항을 검토하십시오.

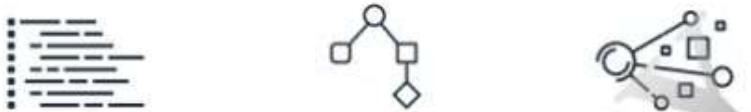
이 데이터베이스의 목적은 프로파일 데이터를 안정적으로 저장하는 것입니다. 여기에는 사용자 식별자(사용자 ID), 이름, 서비스 결제 방법 및 멤버십 데이터가 포함됩니다. 또한 사용자마다 고유할 수 있는 동적 프로파일 스키마를 처리해야 합니다. 신용 카드를 하나만 저장하는 사용자도 있고, 속성이 서로 다른 신용 카드와 은행 계좌를 모두 저장하는 사용자도 있습니다. 가장 중요한 것은 다양한 패턴을 사용하여 사용자 프로파일을 쿼리하고 인덱싱한다는 것입니다. 어떤 워크플로에서는 올해에 연간 멤버십에 가입한 특정 도시의 사용자에 대한 보고서를 앱에서 생성할 수 있고, 또 어떤 워크플로에서는 기본 결제 방법으로 사용자 수를 생성할 수도 있습니다.

이는 읽기 집약적 워크로드입니다. 데이터베이스 작업은 약 95%의 읽기와 5%의 쓰기로 구성됩니다. 유연한 쿼리 및 집계와 함께 동적 속성과 값을 관리하려면 매우 유연한 스키마가 필요합니다. 따라서 사용자 프로파일 프라이머리 데이터베이스로 Amazon DocumentDB가 선택되었습니다.

Amazon DocumentDB

빠르고 확장 가능한 완전관리형 JSON 문서 데이터베이스 서비스

Amazon DocumentDB의 용도:



동적 속성과 데이터
값을 갖는 심층 중첩
문서

スキ마 유연성이
필요한 문서

일회성 쿼리, 인덱싱
및 집계가 필요한
동적 데이터세트

AWS © 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Amazon DocumentDB는 빠르고 신뢰할 수 있는 완전관리형 JSON 도큐먼트 데이터베이스 서비스입니다. Amazon DocumentDB를 사용하면 클라우드에서 MongoDB 호환 데이터베이스를 설정, 운영 및 크기를 조정할 수 있습니다.

이 과정에서는 DocumentDB를 사용하여 사용자 프로파일 데이터를 저장합니다. DocumentDB는 쿼리, 인덱싱 및 집계 요구 사항을 충족합니다. DocumentDB를 사용하면 기본 키를 디자인할 필요가 없습니다. 데이터는 Key-value 페어 대신 JSON 문서에 저장됩니다. 유연한 데이터베이스 스키마는 주소, 결제 방법 등과 같은 데이터를 보관하는 데 가장 적합하며 즉시 사용할 수 있습니다.



모바일 앱의 순위표 워크로드 요구 사항을 검토하십시오.

앱이 거의 실시간으로 이동 활동을 기반으로 점수를 표시하고 사용자 순위를 매겨야 합니다. 이 액세스 패턴은 매우 읽기 집약적입니다. 또한 예측할 수 없는 트래픽 패턴을 처리하고 높은 동시성 읽기를 지원해야 합니다.

순위표 워크로드는 정렬되고 순위가 매겨진 데이터의 읽기 집약적 액세스 패턴을 처리할 수 있도록 밀리초 미만의 응답 시간이 필요합니다. 따라서 ElastiCache for Redis는 순위표 워크로드에 가장 적합합니다.

Amazon ElastiCache for Redis

관리형 Redis 호환 인 메모리 데이터 스토어

ElastiCache for Redis의 용도:



지연 시간이 1밀리초
미만인 인 메모리
캐싱



실시간 순위표



지리 공간 데이터



© 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Amazon ElastiCache for Redis는 인터넷 규모 실시간 애플리케이션에 필요한 1밀리초 미만의 지연 시간을 제공하는 고속 인 메모리 데이터 스토어입니다. 실시간 순위표는 Amazon ElastiCache for Redis에서 완벽하게 지원됩니다. 점수를 기준으로 정렬된 목록을 유지하면서 고유의 요소를 제공하는 Redis 정렬된 세트 데이터 구조를 사용할 수 있습니다.

이 과정에서 비즈니스 요구 사항과 액세스 패턴을 살펴보는 동안 다른 유용한 ElastiCache for Redis 기능을 발견할 수도 있습니다. 지리 공간 데이터에 액세스할 때에는 ElastiCache를 사용하는 것이 좋습니다. ElastiCache for Redis를 사용하여 이용 시작 시간, 이동 거리, 관심 지점 등의 위치 기반 기능을 애플리케이션에 추가할 수 있습니다.



플랫 및 이동 관리에는 DynamoDB를 사용합니다.

사용자 프로파일의 경우 DocumentDB를 살펴보겠습니다.

순위표 관리에는 ElastiCache for Redis를 사용합니다.

이 과정 전반에 걸쳐 새로운 요구 사항을 살펴보거나 서비스 간에 데이터를 스트리밍하여 워크로드를 최적화하는 방법을 활용합니다.

빌드할 준비가 되었습니까?



AnyCompany BikeShare
모바일 앱

AWS

© 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

AnyCompany BikeShare의 목표와 도전 과제를 검토하고 디자인을 결정했으므로, 이제 액세스 패턴과 엔터티 관계를 검토할 준비가 되었습니다. 이 정보를 바탕으로 고성능 고급 솔루션 구축을 시작할 수 있습니다.





어젠다: 고급 Amazon DynamoDB 개념

- 플릿 관리 및 이동 추적
 - 액세스 패턴 및 키 디자인
 - 데이터 모델: 테이블 디자인
- 자전거 및 스쿠터 검색
 - 성능 기반 디자인: 자산 위치 및 배터리 잔량 캐싱
 - Amazon DynamoDB Accelerator(DAX) 아니면 Amazon ElastiCache for Redis?
 - 이벤트 기반 디자인 패턴: ElastiCache for Redis에 데이터 채우기
 - Amazon DynamoDB Streams
- 실습 1: Amazon DynamoDB 테이블, 인덱스 및 변경 스트림을 사용하여 플릿 및 이동 데이터 관리 시스템 구현



© 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

이 모듈에서 살펴볼 주제를 검토하십시오.



다음은 모듈 1의 워크로드 요구 사항 중 일부입니다.

다음 섹션에서는 플릿 관리 및 이동 추적의 액세스 패턴을 검토합니다. 해당 정보를 근거로 데이터베이스 구축에 대한 합리적인 결정을 내리십시오.

플릿 관리 액세스 패턴

- 플릿 자산을 추가하고, 업데이트하고, 가져옵니다.
- 배터리 구동식 자산 중에서 배터리 잔량이 얼마 남지 않은 자산을 찾습니다.
- 자산별 서비스 기록을 가져옵니다.



aws

© 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

플릿 관리는 AnyCompany BikeShare의 핵심 사용 사례 중 하나입니다. 다음은 사용자 경험과 비즈니스에 직접적인 영향을 미치는 중요한 기능입니다.

- 사용자가 이용 가능한 자산을 찾을 수 있도록 자산 가용성 상태를 시스템에 정확하게 반영합니다.
- 자산이 사전 정의된 고정 간격으로 배터리 상태와 지리적 위치를 보고합니다.
- 플릿 사업자는 연중무휴 사업을 운영할 수 있도록 자산을 추가, 폐기 및 점검할 수 있어야 합니다.

이 모든 기능은 이동 검색 및 사용자 경험에 영향을 미치기 때문에 매우 중요합니다.

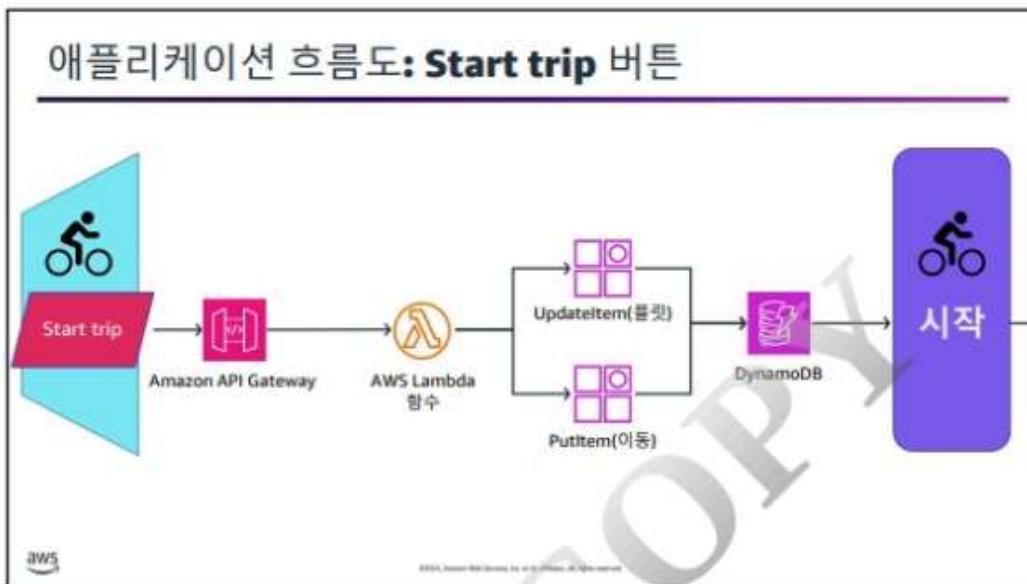
이동 추적 액세스 패턴

- 이동 데이터를 저장하고 액세스합니다.
- 지정된 수만큼 최근 사용자별 이동을 찾습니다.
- 모든 사용자의 월별 이동을 저장합니다.
- 무료 이용권을 제공할 수 있도록 월간 거리 상위 사용자 정보를 가져옵니다.



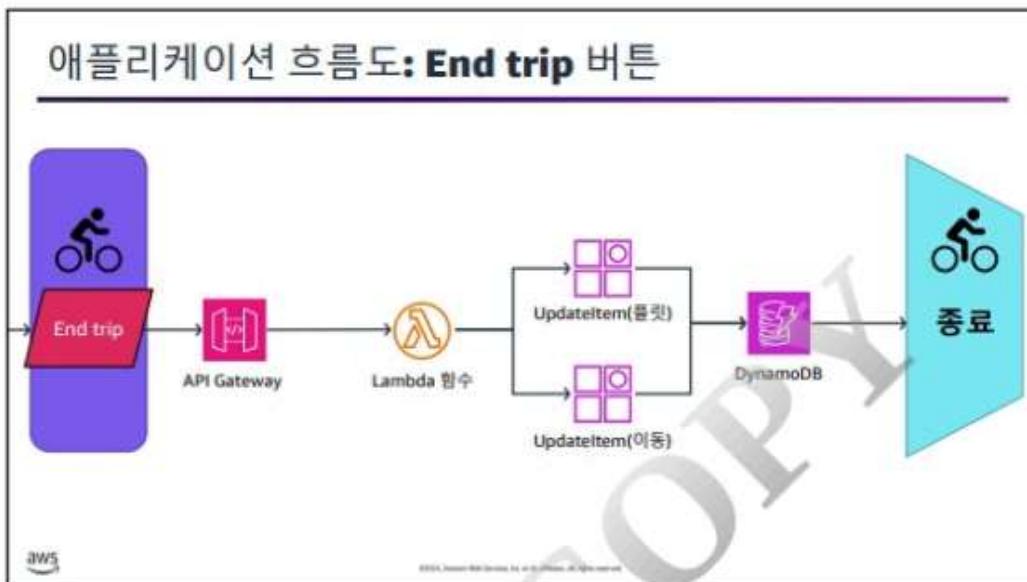
AWS

AnyCompany BikeShare는 앱에서 총 이동 거리를 수집할 수 있도록 사용자의 월별 이동을 저장하려 합니다. 이 회사는 이동 거리를 향후 앱 기능과 보상에 사용할 수 있습니다. 고객은 한 달 동안 이동한 횟수 또는 총 이동 거리에 따라 무료 사용 혜택을 받을 수 있습니다.



Start trip 버튼

1. 사용자는 근처에서 이용 가능한 자전거나 스쿠터를 찾은 후, 자전거의 QR 코드를 스캔하고 **Start trip** 버튼을 선택합니다. 이 버튼을 선택하면 HTTPS를 통해 Amazon API Gateway로 API 요청이 전송됩니다.
2. API Gateway는 요청을 처리하기 위해 AWS Lambda 함수를 시작합니다.
3. DynamoDB에 두 건의 API 호출이 이루어집니다.
4. 플릿 항목 상태 속성이 **in use**로 업데이트되어 플릿 자산을 사용할 수 없게 됩니다.
5. 새 이동 레코드를 생성하기 위해 DynamoDB 테이블에 항목이 생성(put)됩니다.
6. DynamoDB가 업데이트되면 사용자는 이동이 시작되었다는 확인 메시지를 받습니다.



1. 사용자가 목적지에 도착합니다. 사용자가 자전거 또는 스쿠터를 주차하고 모바일 앱에서 **End trip** 버튼을 선택합니다. 이 버튼을 선택하면 HTTPS를 통해 API Gateway로 새 API 요청이 전송됩니다.
2. 마찬가지로 Lambda가 API 요청을 처리합니다. Lambda는 DynamoDB에 두 건의 API 호출을 전송합니다.
3. 새로운 사용자가 자산을 사용할 수 있도록 플릿 항목이 업데이트됩니다. 이동 항목이 종료 타임스탬프와 총 이동 거리로 업데이트됩니다.
4. 데이터가 DynamoDB에 기록된 것이 확인되면 고객에게 영수증이 제공되고, 이동이 종료됩니다.



이동은 대부분 자산과 독립적 관계입니다. 각 이동에서 자산을 사용하지만, 자산을 관리하기 위해 대부분의 이동 정보를 사용할 필요는 없습니다.



일반적인 액세스 패턴을 이해했으므로, 이제 테이블의 기본 키 디자인을 선택하고 필요한 보조 인덱스에 대한 계획을 세워야 합니다.

디자인 선택: 별도의 테이블

두 항목(자산 및 이동)은 서로 관련이 없습니다.

별도의 테이블을 사용하면 운영 및 아키텍처가 더 유연해집니다.



DynamoDB fleet 테이블



DynamoDB trips 테이블



© 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

앱 개발 팀이 두 개의 별도 테이블을 구축하기로 결정합니다. 다음 사항을 고려하여 이렇게 결정한 것입니다.

- 두 워크로드의 액세스 패턴은 서로 독립적이며 독립적으로 모델링할 수 있습니다. 이렇게 하면 인덱스와 테이블을 더 간단하고 유연하게 디자인할 수 있습니다.
- 이 모듈의 뒷부분에서 살펴볼 Amazon DynamoDB Streams 기능은 스트림의 색드를 동시에 읽을 수 있는 숫자가 제한되어 있습니다. 별도의 테이블을 만들면 전용 테이블을 통해 각 워크로드에 대한 별도의 스트림을 구축하면서도 글로벌 테이블 구성에서 테이블 복제본을 허용할 수 있습니다. 글로벌 테이블은 동일한 DynamoDB Streams 기능을 사용하여 변경 내용을 복제본 테이블에 복제합니다.

단일 테이블 및 다중 테이블 디자인에 대한 자세한 내용은 [AWS Database 블로그](#)의 “Single-Table vs. Multi-Table Design in Amazon DynamoDB” (<https://aws.amazon.com/blogs/database/single-table-vs-multi-table-design-in-amazon-dynamodb/>)를 참조하십시오.

글로벌 테이블에 대한 자세한 내용은 “Amazon DynamoDB 글로벌 테이블” (<https://aws.amazon.com/dynamodb/global-tables/>)을 참조하십시오.



다음으로, 두 테이블의 디자인을 검토하십시오.

DynamoDB fleet 테이블

Primary Key		Attributes				
Partition Key (PK)	Sort Key (SK)	AssetType	Battery	Latitude	Longitude	Status
ASSETTYPE1	ASSET1SK1	Ebike	5	30.8571294	-70.95574231	LOW_BATTERY
	ASSET1SK2	ServiceCodeDate	ServiceDate	ServiceStatus		COMPLETED
ASSETTYPE2	ASSET2SK1	Ebike	48	30.8571294	-70.95574231	AVAILABLE
	ASSET2SK2	ServiceCodeDate	ServiceDate	ServiceStatus		IN_PROGRESS
	ASSET2SK3	ServiceCodeDate	ServiceDate	ServiceStatus		OPEN
ASSETTYPE3	ASSET3SK1	Ebike	47	30.8571294	-70.95574231	AVAILABLE
	ASSET3SK2	ServiceCodeDate	ServiceDate	ServiceStatus		COMPLETED
	ASSET3SK3	ServiceCodeDate	ServiceDate	ServiceStatus		OPEN

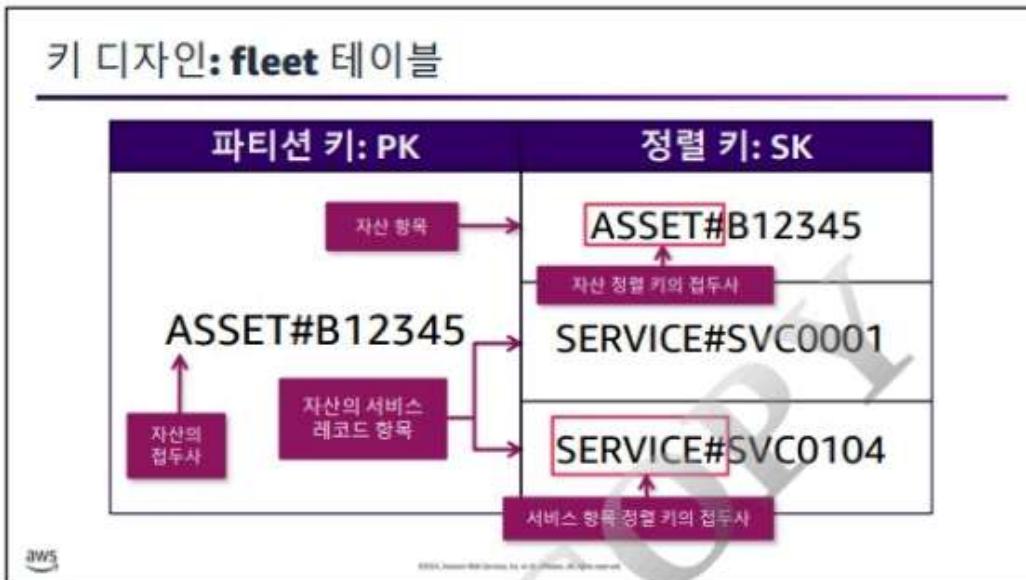
© 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

다음 테이블 및 항목 속성을 검토하십시오.

- 파티션 키(PK):** 테이블의 파티션 키이며 접두사를 사용합니다.
 - 정렬 키(SK):** 테이블의 정렬 키이며 접두사를 사용합니다. 자산 ID와 서비스 주문 ID 둘 다 정렬 키에 표시됩니다.
- AssetType:** 고객과 공유하는 자전거 또는 스쿠터 유형입니다. 이 예제의 유형은 EBIKE 및 ESCOOTER입니다.
- Battery:** 정수 값으로 표시되는 전동 자전거 및 스쿠터의 배터리 잔량 비율입니다.
 - Latitude:** 자산의 위도 지리적 위치 값입니다.
 - Longitude:** 자산의 경도 지리적 위치 값입니다.
 - Status:** 전동 자산의 상태가 LOW_BATTERY인지 추적하는 키입니다.
 - ServiceCreateDate:** 서비스 주문의 생성 날짜입니다.
 - ServiceDate:** 자산의 서비스 주문이 진행 중이거나 완료된 날짜입니다.
 - ServiceStatus:** 서비스 주문의 상태가 COMPLETED, IN_PROGRESS 또는 OPEN인지 추적하는 키입니다.

테이블은 파티션 키와 정렬 키를 사용하여 복합 기본 키의 이점을 얻습니다. 적절한 접두사를 사용하여 정렬 키를 디자인하면 상위 엔터티인 자산에 대한 다양한 항목 유형을 저장할 수 있습니다. 이 워크로드에서는 자산의 데이터와 서비스 코드를 동일한 논리 파티션에 저장할 수 있습니다. 정렬 키는 자산의 모든 항목(자산 데이터 및 서비스 코드) 또는 서비스 코드만 가져올 수 있는 쿼리 유연성을 제공합니다. 뿐만 아니라 점진적으로 증가하는 고유 식별자를 서비스 코드에 사용하면 서비스 코드가 정렬된 순서로 저장됩니다. 이러한 서비스 코드를 오름차순 또는 내림차순으로 쿼리할 수 있습니다.

자산 코드에 의미 있는 정렬 키가 없더라도 복합 기본 키에는 SK 값이 필요합니다. 따라서 자산의 코드를 가져오는 GetItem 호출을 실행하여 PK와 동일한 값(자산 ID)을 SK에 사용합니다.



정렬 키를 인코딩하는 목적은 다음과 같습니다.

- 다양한 항목 유형의 엔터티를 저장합니다.
- 특정 유형의 항목을 검색할 때 쿼리 유연성을 제공합니다.
- 다양한 항목 유형 간의 키 충돌을 방지합니다.
 - 예를 들어 자전거 식별자가 서비스 주문 식별자와 동일한 경우 접두사가 붙은 Key-value은 ASSET#<AssetID>, SERVICE#<ServiceOrderID> 형식이며, 따라서 고유하기 때문에 서로 덮어쓰는 것(충돌)을 방지할 수 있습니다.

보조 액세스 패턴

1. Status:LOW_BATTERY

배터리 잔량이 얼마 남지 않은 자전거와 스쿠터를 정렬하여 충전 목록에 추가해야 합니다. 배터리 잔량이 얼마 남지 않은 자전거를 고객에게 제공하면 안 됩니다.

2. ServiceStatus:OPEN

미해결 서비스 주문이 있는 자산을 나열하여 해당 자산을 현재 사용할 수 없다는 것을 나타낼 수 있어야 합니다.

소수의 항목만이 이러한 속성을 가질 것으로 예상됩니다.

aws

이 모듈의 앞부분에서 설명한 보조 액세스 패턴을 생각해 보십시오.

배터리 구동식 자산 중에서 배터리 잔량이 얼마 남지 않은 자산 찾기
 플릿에는 배터리 잔량이 얼마 남지 않은 것을 나타내는 LOW_BATTERY 속성을 비롯한 자산 데이터를 정기적으로 전송하는 사물 인터넷(IoT) 디바이스가 포함되어 있다고 가정합니다. 이 액세스 패턴의 경우 상태 속성을 사용하여 인덱싱하고 자산 ID를 제공하는 결과를 반환하려 합니다. 이렇게 하면 앱의 서비스에서 해당 자산을 제거하는 데 도움이 됩니다.

자전거는 자주 충전되며, 특정 시점에 LOW_BATTERY 상태인 자전거 플릿의 1%인 것으로 관찰되었습니다. 또한 LOW_BATTERY를 보고하는 자전거 빈도는 피크 활동 시간에 초당 4개 자산인 것으로 관찰되었습니다.

자산의 미해결 서비스 주문 찾기

FilterExpression(ServiceStatus:OPEN)을 사용하여 기본 테이블을 스캔할 수 있지만, 스캔을 권장하지는 않습니다. 서비스 주문 수가 무제한으로 비용이 많이 들 수 있습니다. 시간이 지나면 서비스 주문 수가 늘어나고, 따라서 테이블에서 스캔할 데이터의 양도 늘어납니다.

두 액세스 패턴 모두 작업 규모가 스케일업되더라도 반환되는 항목 수가 적을 것으로 예상됩니다.

GSI 디자인

Primary Key		Attributes			
Partition Key (PK)	Sort Key (SK)	AssetType	Battery	Latitude	Longitude
ASSETTYPE1	ASSETID1	Battery	5	30.8571294	-70.95574231
	ASSETID2	ServiceCodeDate	ServiceDate	ServiceStatus	ServiceTime
ASSETTYPE2	ASSETID3	Battery	48	30.8571294	-70.95574231
	ASSETID4	ServiceCodeDate	ServiceDate	ServiceStatus	ServiceTime
	ASSETID5	ServiceCodeDate	ServiceDate	ServiceStatus	ServiceTime
ASSETTYPE3	ASSETID6	Battery	47	30.8571294	-70.95574231
	ASSETID7	ServiceCodeDate	ServiceDate	ServiceStatus	ServiceTime
	ASSETID8	ServiceCodeDate	ServiceDate	ServiceStatus	ServiceTime

© 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

NoSQL Workbench 데이터 모델의 스크린샷입니다.

글로벌 보조 인덱스(GSI)를 사용하여 두 액세스 패턴을 모두 지원할 수도 있습니다. 속성에 1과 2라는 레이블이 지정되었습니다.

앞으로 2개 슬라이드에서 다음 개방형 질문에 대해 생각해 보십시오.

- 어떻게 하면 GSI를 사용하여 반환되는 결과 수를 줄일 수 있을까요?
- 두 인덱스를 결합하여 성능 효율성을 높일 수 있는 방법이 있을까요?

GSI sparse index : GSI1

GSI_PK 속성을 추가하고 LOW_BATTERY로 설정합니다.



Sparse index는 컬렉션의 일부만 인덱싱하는 데 사용할 수 있는 특별한 유형의 GSI입니다. 이렇게 하려면 일부 항목에만 있는 속성을 인덱싱하면 됩니다. 이 기술은 LOW_BATTERY 속성이 있는 항목을 찾으려는 경우처럼 특정 속성 값이 있는 항목을 빠르게 쿼리하는 데 유용합니다.

좋은 Sparse index를 생성하려면 해당 인덱스에 있어야 하는 항목에만 해당 인덱스의 정렬 Key-value가 있어야 합니다.

이 구현에서는 GSI1_PK 속성을 생성합니다. LOW_BATTERY 상태가 항목에 추가되면 GSI1_PK 속성에 LOW_BATTERY가 추가됩니다. GSI1_PK를 파티션 키로 사용하여 인덱싱하면 Sparse index가 생성됩니다.

LOW_BATTERY를 보고하는 자산의 빈도가 높아질 것으로 예상되면 쓰기 샤딩을 사용하여 기본 키를 다시 디자인해야 합니다. 그러면 핫 파티션과 GSI 배압이 방지됩니다.

GSI 오버로딩

여러 항목 유형의 공통 속성을 인덱싱

파티션 키: GSI1_PK	정렬 키: PK
LOW_BATTERY	ASSET#B12345
OPEN#B67890	SERVICE#SVC0001

AWS © 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

DynamoDB는 최대 20개의 글로벌 보조 인덱스와 5개의 로컬 보조 인덱스를 지원할 수 있습니다. 아직은 가급적이면 인덱스를 더 생성하지 말고 기존 인덱스를 사용해야 합니다. 이것을 **GSI 오버로딩**이라고 합니다.

GSI 오버로딩은 여러 항목 유형의 공통 속성을 인덱싱하는 것입니다. 공통 속성 값은 인덱스를 사용하여 쿼리할 다양한 항목 유형을 식별해야 합니다. 인덱스 수가 적을수록 기본 테이블의 오버헤드가 줄어듭니다. 유지 관리하고 운영해야 하는 인덱스가 하나 적습니다.

배터리 부족 액세스 패턴과 미해결 서비스 주문 액세스 패턴은 공통 특성을 공유합니다. 우리가 원하는 것은 작은 항목 하위 집합의 레코드만 반환하는 것입니다. 단일 인덱스는 용량 활용도를 최적화하고 인덱스 수를 줄일 수 있습니다.

GSI 오버로딩을 사용하면 단일 인덱스를 사용하여 여러 항목 유형(배터리 부족 및 미해결 서비스 주문)의 fleet 테이블에 대한 보조 액세스 패턴을 지원할 수 있습니다. 이렇게 하면 인덱스 수를 적게 유지하여 GSI 테이블당 할당량이 제한하는 것보다 더 많은 액세스 패턴을 지원할 수 있습니다.

GSI1_PK를 서비스 주문 속성 유형으로 추가하고 OPEN#<AssetID>로 채워서 미해결 서비스 주문을 이 파티션에 프로젝션합니다. 기본 테이블의 정렬 키를 GSI1의 정렬 키로 사용하면 자산의 가장 오래된 미해결 서비스 주문과 최근 미해결 서비스 주문을 검색할 수 있습니다(DynamoDB API 파라미터는 각각 ScanIndexForward:True 및 ScanIndexForward:False로 설정됨). 이렇게 하면 서비스 테크니션이 선입선출과 같은 전략을 사용하여 서비스 주문의 우선 순위를 정하는 데 도움이 될 수 있습니다.

DynamoDB fleet 테이블: GSI1

fleet 테이블에서 볼게				GSI1(GSI 오버로드 사용)			
Primary key Partition key: PK	Sort key: SK	Attributes	GSI1_PK	Primary key Partition key: GSI1_PK	Sort key: SK	Attributes	Status
ASSET#B74819	ASSET#B74819	Status LOW_BATTERY	GS11_PK LOW_BATTERY	OPEN#S73329	ASSET#B74819	ServiceStatus COMPLETED	LOW_BATTERY
	SERVICE#SVC12138857	ServiceStatus AVAILABLE	GS11_PK OPEN#S73329		OPEN#B65809	SERVICE#SVC6838317	OPEN
ASSET#B65809	ASSET#B65809	Status AVAILABLE	GS11_PK OPEN#B65809	SERVICE#SVC14296809	ASSET#B65809	ServiceStatus COMPLETED	LOW_BATTERY
	SERVICE#SVC56838318	ServiceStatus OPEN	GS11_PK OPEN#B65809		OPEN#B65809	SERVICE#SVC6838318	OPEN

AWS © 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

테이블에 있는 항목의 경우 인덱스 정렬 Key-value이 항목에 있는 경우에만 DynamoDB가 해당 인덱스 항목을 씁니다. 이 예제에서는 Sparse index와 GSI 오버로딩을 결합하면 작업에서 처리되는 항목 수를 줄일 수 있습니다.

DynamoDB trips 테이블

Primary key		Attributes			
Partition key: PK	Sort key: SK				
USER#1/1234567	TripID2023-06-19T20:14:53ZTripId123	StartTs TripId123	EndTs TripId123	2023-06-19T20:14:53Z 2023-06-19T20:14:53Z	Miles 4
	TripID2023-06-19T20:14:53ZTripId123	StartTs TripId123	EndTs TripId123	2023-06-19T20:14:53Z 2023-06-19T20:14:53Z	Miles 4
USER#2/1234568	TripID2023-06-19T20:14:53ZTripId126	StartTs TripId126	EndTs TripId126	2023-06-19T20:14:53Z 2023-06-19T20:14:53Z	Miles 5
	TripID2023-06-19T20:14:53ZTripId127	StartTs TripId127	EndTs TripId127	2023-06-19T20:14:53Z 2023-06-19T20:14:53Z	Miles 5
USER#3/1234569	TripID2023-06-19T20:14:53ZTripId128	StartTs TripId128	EndTs TripId128	2023-06-19T20:14:53Z 2023-06-19T20:14:53Z	Miles 7
	TripID2023-06-19T20:14:53ZTripId129	StartTs TripId129	EndTs TripId129	2023-06-19T20:14:53Z 2023-06-19T20:14:53Z	Miles 8

© 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

NoSQL Workbench 스크린샷과 다음 테이블 및 항목 속성을 검토하십시오.

- **PK:** trip 테이블의 파티션 키입니다. 스크린샷에서 PK는 USER#과 같은 접두사를 사용하여 사용자 ID를 나타냅니다.
- **SK:** 테이블의 정렬 키입니다. 스크린샷에는 타임스탬프(시작 시간 데이터)와 이동 ID가 포함된 복합 구조가 나와 있습니다.
- **TripID:** 임의의 문자열 텍스트 TripId와 일련 번호가 함께 표시되는 고유 이동 식별자입니다.
- **StartTs:** ISO8601 형식의 이동 시작 타임스탬프입니다.
- **EndTs:** ISO8601 형식의 이동 종료 타임스탬프입니다.
- **Miles:** 마일 단위로 측정된 총 이동 거리입니다.

핵심 액세스 패턴을 생각하십시오.

- 이동 데이터 저장 및 액세스: 사용자 ID, 이동 ID, 이동 시작, 이동 종료 및 이동 거리.
- 지정된 수만큼 최근 사용자별 이동을 찾습니다.
- 모든 사용자의 월별 이동을 저장합니다.
- 보상 프로그램을 제공할 월간 거리 상위 사용자 정보를 가져옵니다.

액세스 패턴에 따라 데이터는 사용자 참조와 함께 저장 및 검색됩니다. 사용자 ID를 기본 테이블의 파티션 키로 선택하면 강력한 일기 일관성이 제공됩니다. 그렇지 않고 이동 ID 속성을 기본 키로, 사용자 ID를 정렬 키로 사용하면 기본 테이블의 GSI 반전 인덱스가 있어야만 사용자별 데이터를 검색할 수 있습니다. 이는 효율성이 떨어지는 액세스 패턴입니다.

#(해시 또는 파운드 기호) 구분 기호를 사용하여 이동 시작 시간을 기준으로 trips 테이블의 정렬 키를 구성하십시오. 데모에서 사용자 이동을 저장하는 정렬 키는 TRIP#<ISO8601-Timestamp>#<TripID> 조합입니다. 타임스탬프를 통합하면 범위 기반 쿼리(타임스탬프 범위 사이)를 사용할 수 있습니다. 이렇게 하면 이동 데이터를 저장하고 사용자의 최근 이동 정보를 가져올 수 있으며, 액세스 패턴 1, 2, 3을 만족합니다.

기본 테이블 수준에서는 테이블의 집계된 이동을 정렬할 수 없습니다. 그렇게 하려면 보조 인덱스가 필요합니다. 다음 슬라이드에서는 이 테이블 디자인이 액세스 패턴 4를 어떻게 처리하는지 보여줍니다.



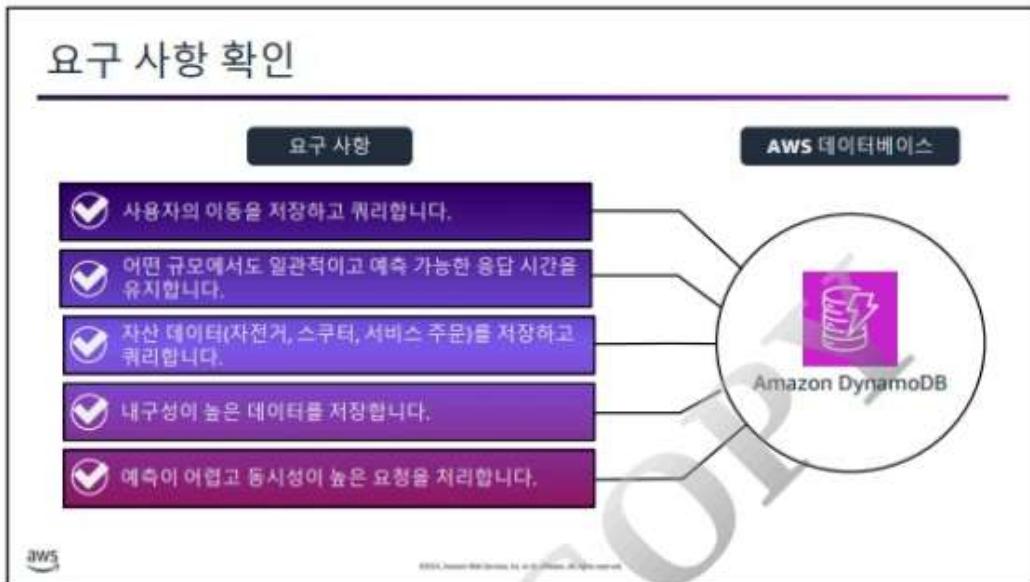
보상 프로그램을 제공할 월간 이동 거리 상위 사용자를 가져오는 액세스 패턴을 생각해 보십시오.

SK(trips 테이블 정렬 키)를 사용하여 매월 집계된 총 이동 거리를 저장하십시오. 집계된 사용자의 이동 거리를 저장하는 정렬 키는 복합 AGG#<YYYY-MM>으로 디자인되었습니다. 사용자의 집계된 거리 데이터는 기본 테이블에 저장되고 쿼리되므로, 액세스 패턴 4를 충족합니다.

파티션에서 유연한 쿼리 작업을 위한 정렬 키를 인코딩하십시오. 그러려면 PK가 AGG#<YYYY-MM>이어야 하며, 정렬 키는 각 사용자의 TotalMiles(총 거리 측정값)여야 합니다.

인덱스 수를 최소한으로 유지하십시오. 자주 쿼리하지 않는 속성에 대한 보조 인덱스를 생성하지 마십시오. 거의 사용되지 않는 인덱스는 애플리케이션 성능 향상에 도움이 되지 않으면서 스토리지 및 I/O 비용을 늘립니다.

보조 인덱스 모범 사례에 대한 자세한 내용은 [Amazon DynamoDB 개발자 가이드의 'DynamoDB의 보조 인덱스에 대한 일반 가이드'](https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/bp-indexes-general.html)(<https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/bp-indexes-general.html>)을 참조하십시오.



요구 사항을 확인하여 솔루션이 이해당사자의 요구 사항을 모두 충족하는지 확인하십시오.



제품 관리자가 기능 요청을 위해 여러분에게 연락합니다. 애플리케이션은 요청자의 현재 위치를 기반으로 한 자전거 검색을 지원해야 합니다. 그러면 고객이 가장 가까운 자전거를 찾을 수 있으므로 편리한 사용자 경험을 얻게 됩니다.

모바일 앱은 근처의 자전거 및 스쿠터를 검색하고 다음 요구 사항을 해결해야 합니다.

- 이동이 종료되는 시점과 지도에 사용 가능한 자산이 표시되는 시점 사이에 긴 시간 지연이 있어서는 안 됩니다.
- LOW_BATTERY 상태인 자산은 자전거 검색에서 제외해야 합니다.

자전거 및 스쿠터 검색 액세스 패턴

- 검색 반경 내의 결과
 - 선택 사항으로, 배터리 잔량 기준(LOW_BATTERY가 아닌 차선)
- 응답 시간이 1밀리초 미만인 빠른 검색 결과
- 준실시간 자전거 상태
- 높은 읽기 빈도 지원



aws

© 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

사용자가 근처에서 가장 좋은 자전거나 스쿠터를 정확하게 찾을 수 있어야 합니다. 다음은 사용자 경험과 비즈니스에 직접적인 영향을 미치는 중요한 기능입니다.

- LOW_BATTERY 상태인 자전거가 자전거 검색 결과에 정렬됩니다.
- 응답 시간 1밀리초 미만으로 검색 결과가 빠르게 제공됩니다.
- 결과가 거의 실시간 데이터로 정확합니다.
- 앱은 주요 이벤트가 있을 때 급증하는 자전거 검색 데이터 요청을 처리해야 합니다. 예를 들어 스포츠 행사나 콘서트가 끝난 후 많은 고객이 자전거나 스쿠터를 동시에 요청할 수 있습니다.



응답 시간이 1밀리초 미만이어야 한다는 요구 사항을 충족하려면 캐싱 솔루션 구축을 고려해야 합니다. 캐싱 계층을 사용하면 가장 빠른 읽기 시간을 제공할 수 있습니다.

읽어 들이는 데이터(플랫 자산 위치 및 배터리 상태)가 이미 DynamoDB 테이블에 있습니다. 기존 프라이머리 데이터베이스와 잘 작동하는 솔루션을 선택하십시오.

캐싱 솔루션 고민

어떤 솔루션을 구축하시겠습니까?



DAX



ElastiCache
for Redis

Amazon DynamoDB Accelerator(DAX)와 Amazon ElastiCache for Redis 둘 다 기존 워크플로와 잘 작동하며, 프로젝트 계획을 위해 이미 선택한 서비스와 통합할 수 있는 캐싱 옵션입니다.

어떤 솔루션을 구축하시겠습니까? 결정을 내리기 전에 다음 두 슬라이드의 서비스 요약 내용을 검토하십시오.

DAX



aws

DAX는 DynamoDB를 위한 완전관리형 고가용성 인 메모리 캐시입니다

- DAX는 인라인 읽기 및 동시 쓰기 워크플로를 지원합니다.
- 기존 DynamoDB API와 통합됩니다.
- DynamoDB 항목과 쿼리 결과를 캐시합니다.

DAX는 DynamoDB를 위한 네이티브 완전관리형 고가용성 인 메모리 캐시입니다
DAX는 초당 수백만 건의 요청이 발생하는 경우에도 최대 10배(밀리초~마이크로초)의 성능 향상을 제공합니다.

개발자가 캐시 무효화, 클러스터 관리 또는 데이터 집단을 관리할 필요 없이 DAX가 DynamoDB 테이블에 인 메모리 가속화를 추가하는 데 필요한 모든 작업을 수행합니다. 인라인 읽기 및 동시 쓰기 워크플로에 즉시 사용할 수 있으며 기존 DynamoDB API와 통합됩니다. DAX를 사용하면 애플리케이션 논리를 수정할 필요가 없습니다. DAX가 매력적인 캐싱 옵션인 이유입니다.

ElastiCache for Redis



ElastiCache for Redis



ElastiCache for Redis는 클라우드의 인 메모리 데이터 스토어 또는 캐시 환경입니다.

- 사이드 캐시로 사용하는 것이 가장 좋습니다.
- 구현하려면 코드를 변경해야 합니다.
- ElastiCache for Redis는 다음을 포함한 고급 기능을 지원합니다.
 - 정렬된 세트(순위표)
 - 지리 공간 데이터 형식

© 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

ElastiCache for Redis는 인터넷 규모 실시간 애플리케이션에 필요한 1밀리초 미만의 지연 시간을 제공하는 고속 인 메모리 데이터 스토어입니다. 오픈 소스 Redis를 기반으로 개발되었으며 Redis API와 호환되는 ElastiCache for Redis는 Redis 클라이언트와 함께 작동하며 개방형 Redis 데이터 형식을 사용하여 데이터를 저장합니다.

ElastiCache for Redis는 오픈 소스 Redis의 속도, 간편성 및 다양성과 Amazon의 관리 편의성, 보안 및 확장성을 결합하여 게임, 광고 기술, 전자 상거래, 의료 서비스, 금융 서비스 및 IoT 분야에서 가장 까다로운 실시간 애플리케이션을 지원합니다. 사이드 캐시 전략을 고려할 의향이 있다면 ElastiCache for Redis를 고려해 보십시오. 이를 위해서는 코드를 변경해야 하며 추가적인 개발 노력이 필요합니다.

다음과 같은 Redis 플랫폼의 이점을 누릴 수 있습니다. 다음 사용 사례에 잘 작동하는 고급 기능을 지원합니다.

- 인 메모리 데이터
- 게임 순위표(Redis 정렬된 세트)
- 지리 공간 데이터
- 메시지(Redis 게시/구독)
- 권장 사항 데이터(Redis 해시)

일반적인 사용 사례에 대한 자세한 내용은 **Amazon ElastiCache for Redis 사용 설명서**의 ‘일반적인 ElastiCache 사용 사례와 ElastiCache 활용 방법’(<https://docs.aws.amazon.com/AmazonElastiCache/latest/red-ug/elasticache-use-cases.html>)을 참조하십시오.



DAX와 ElastiCache for Redis의 핵심 기능을 검토했습니다.

여러분이 의사 결정권자라면 어떤 캐싱 솔루션을 선택하시겠습니까? 그 이유는 무엇입니까? 여러분의 목표는 자전거 유형, 위치 변경, 배터리 상태 등으로 필터링하여 각 자전거와 스쿠터의 위치 데이터를 정렬하는 것입니다.

캐싱 솔루션

DAX와 ElastiCache for Redis 둘 다 DynamoDB의 캐싱 비즈니스 요구 사항을 충족하는 구현 가능한 솔루션입니다.

결과를 계산하는 데 사용된 지리 공간 데이터에 따라 **ElastiCache for Redis**를 선택합니다.



ElastiCache for
Redis



© 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

지리 공간 데이터를 사용하여 결과를 계산한 끝에, 여러분과 앱 개발 팀은 ElastiCache for Redis가 최적의 솔루션이라고 결정합니다. 추가적인 개발 노력이 필요하지만, 데이터에 필요한 네이티브 기능 세트를 가장 많이 갖추고 있습니다.



ElastiCache for Redis를 선택했으므로, 사이드 캐시에 데이터를 로드하는 최적의 방법을 찾아야 합니다. 이벤트 기반 디자인 패턴은 서비스 인프라에 적합합니다. DynamoDB Streams를 사용하여 캐시를 채울 수 있습니다.

다음 슬라이드에서 DynamoDB Streams 서비스를 검토하십시오.

DynamoDB Streams

DynamoDB 스트림은 DynamoDB 테이블 항목 변경 사항에 대한 시간순 정보 흐름입니다.



DynamoDB Streams는 항목 수정 정보를 캡처합니다. 테이블 수준에서 사용 및 사용 중지할 수 있는 단일 스트림입니다.



DynamoDB Streams는 각 항목의 생성, 업데이트 또는 삭제 작업에 대한 스트림 레코드를 작성합니다.



스트림 레코드에는 수정된 항목의 기본 키 속성이 포함되어 있습니다. 이 데이터의 수명은 24시간입니다.



DynamoDB Streams는 스트림 레코드를 거의 실시간으로 쓰기 때문에 그 내용에 따라 조치를 취하는 애플리케이션을 빌드할 수 있습니다.



© 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

DynamoDB Streams를 사용하여 의미 있는 이벤트를 캡처하고 해당 데이터를 사용하여 캐시에 쓰십시오. DynamoDB Streams는 스트림 레코드를 거의 실시간으로 쓰기 때문에, 사용자가 이러한 스트림을 사용하는 애플리케이션을 만들고 콘텐츠에 따라 작업을 수행할 수 있습니다.

이 애플리케이션에서는 스트리밍된 이벤트를 처리하는 데 Lambda 함수를 사용합니다.

DynamoDB Streams에 대한 자세한 내용은 [Amazon DynamoDB 개발자 가이드의 'DynamoDB Streams를 위한 변경 데이터 캡처'](https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Streams.html) (<https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Streams.html>)를 참조하십시오.

스트림 보기 유형

항목이 수정될 때마다 스트림에 쓸 정보를 선택합니다.

1. KEYS_ONLY: 수정된 항목의 키 속성만
2. NEW_IMAGE: 전체 항목의 수정 후 보여지는 모습
3. OLD_IMAGE: 전체 항목의 수정 전 보여지는 모습
4. NEW_AND_OLD_IMAGES: 항목의 변경 전 및 변경 후 이미지 모두



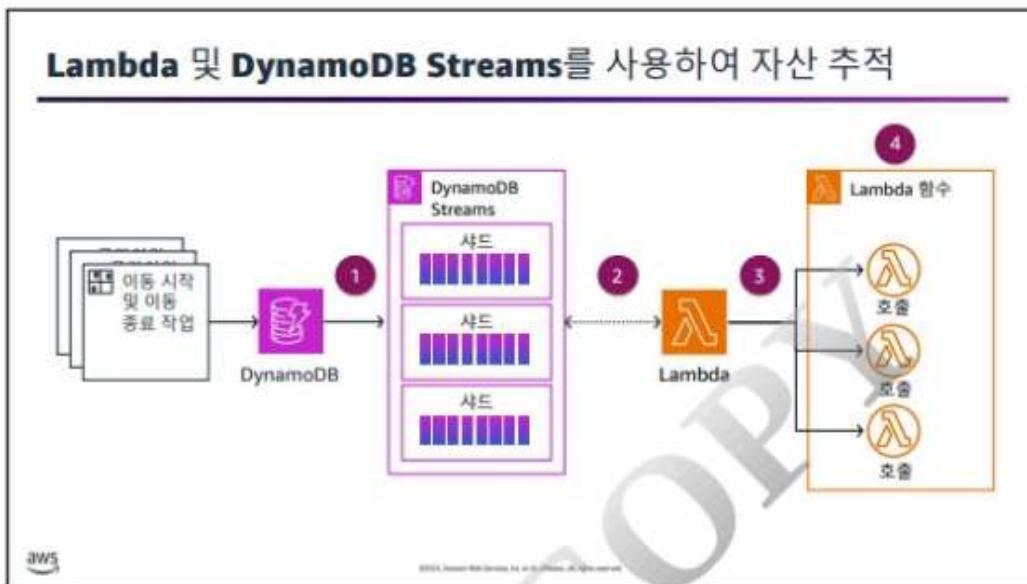
© 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

AWS Command Line Interface(AWS CLI) 또는 AWS SDK 중 하나를 사용하여 새 테이블을 생성할 때 해당 테이블에서 스트림을 사용할 수 있습니다. 기존 테이블에서 스트림을 사용 또는 사용 중지하거나 스트림 설정을 변경할 수도 있습니다. DynamoDB Streams는 비동기적으로 작동하므로 스트림을 사용해도 테이블 성능에 영향을 미치지 않습니다. DynamoDB Streams를 관리하는 가장 쉬운 방법은 AWS Management Console을 사용하는 것입니다.

이 슬라이드에 나열된 네 가지 스트림 보기 유형을 검토하십시오.

NEW_IMAGE에는 플랫 또는 이동 변경 데이터를 ElastiCache for Redis로 가져오는 데 필요한 모든 정보가 포함되어 있습니다. 아카이빙 사용 사례를 보고 있는 것은 아니지만 변경된 속성을 기록해야 합니다. 실습 1에서 NEW_IMAGE 스트림 보기 유형에 대해 자세히 알아보겠습니다.

스트림 사용에 대한 자세한 내용과 최신 가이드를 검토하려면 **Amazon DynamoDB 개발자 가이드의 '스트림 사용'**(<https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Streams.html#Streams.Enabling>)을 참조하십시오.



DynamoDB 스트림 이벤트에 자동으로 응답하는 코드를 작성하십시오.

스트림 레코드는 샤드라고 하는 그룹으로 구성됩니다. 각 샤드는 다중 스트림 레코드 컨테이너 역할을 하며 이러한 기록 액세스 및 반복 처리에 필요한 정보를 담고 있습니다.

Lambda는 테이블이 업데이트될 때마다 작업을 수행합니다. Lambda 서비스는 스트림의 샤드에서 레코드를 읽고 스트림 레코드가 포함된 이벤트를 통해 함수를 동기적으로 호출합니다. Lambda는 초당 4회의 기본 속도로 DynamoDB 스트림의 샤드를 폴링하여 레코드를 찾습니다. 처리가 성공하면 Lambda는 더 많은 레코드를 수신할 때까지 폴링을 재개합니다.

이 다이어그램은 Lambda가 DynamoDB와 상호 작용하는 방식을 보여줍니다.

1. DynamoDB 테이블에 대한 모든 변경 작업은 스트림에서 이벤트로 캡처됩니다.
2. Lambda는 초당 4회의 속도로 새 레코드에 대한 스트림을 폴링합니다.
3. 새 레코드를 사용할 수 있으면 구독한 Lambda 함수가 새 레코드로 호출됩니다.
4. Lambda는 레코드를 일괄적으로 읽고 함수를 호출하여 배치의 레코드를 처리합니다.

Lambda는 스트림 이벤트를 배치 처리합니다. 이벤트를 처리하려면 배치 처리 기간을 구성하십시오. 각 샤드의 여러 배치를 병렬로 처리하여 동시성을 높이십시오. 각 샤드의 배치를 동시에 10개까지 처리할 수 있습니다.

DynamoDB Streams를 사용하면 Lambda 소비자의 읽기 요청이 무료입니다.

Lambda 이외의 이벤트 소비자를 사용하는 경우 매달 최대 250만 건의 무료 읽기 요청이 허용되며, 그 이상의 읽기에 대해서는 요금이 부과됩니다.

스트림별로 함수를 제한하는 방안을 고려하십시오. 단일 리전 스트림에서는 샤크당 최대 2명이 동시에 읽을 수 있습니다. 글로벌 테이블에는 동일한 제한이 적용되지 않습니다. 글로벌 테이블은 변경 스트림 메커니즘을 복제에 사용하므로, 글로벌 테이블에 영향을 주지 않는 스트림 솔루션을 디자인해야 합니다. 글로벌 테이블의 경우 스트림당 Lambda 함수 하나로 제한됩니다.

배치 처리 동작에 대한 자세한 내용은 AWS Lambda 개발자 가이드의 ‘배치 처리 동작’(<https://docs.aws.amazon.com/lambda/latest/dg/invocation-eventsourcemapping.html#invocation-eventsourcemapping-batching>)을 참조하십시오.

Lambda 이벤트 소스 파라미터에 대한 자세한 내용은 AWS Lambda 개발자 가이드의 ‘Amazon DynamoDB Streams 구성 파라미터’(<https://docs.aws.amazon.com/lambda/latest/dg/with-ddb.html#services-ddb-params>)를 참조하십시오.

DynamoDB Streams 및 서비스 한도에 대한 자세한 내용은 AWS DynamoDB 개발자 가이드의 ‘DynamoDB Streams’(<https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/ServiceQuotas.html#limits-dynamodb-streams>)를 참조하십시오.

자산 상태 변화 이벤트 필터링

다음 조건을 모두 충족하는 경우에만 Lambda 함수를 호출합니다.

1. 키(**PK** 및 **SK**) 값이 자산 레코드를 나타내는 접두사 **ASSET#**으로 시작합니다.
2. **NewImage**의 **Status** 속성이 **AVAILABLE** 또는 **IN_USE**입니다.

```

1   {
      "dynamodb": {
        "Keys": {
          "PK": {
            "S": [
              {"prefix": "ASSET#"}
            ]
          },
          "SK": {
            "S": [
              {"prefix": "ASSET#"}
            ]
          }
        },
        "NewImage": {
          "Status": {
            "S": ["AVAILABLE", "IN_USE"]
          }
        }
      }
    }
  
```

이 예제에서는 이벤트 필터가 JSON 코드로 표시되었습니다. 이벤트 필터는 ASSET# 접두사가 붙은 파티션 키와 정렬 키가 있는 스트림 이벤트를 찾습니다. 서비스 레코드 ID도 SK 속성에 저장되므로 두 키 모두 ASSET# 접두사가 있어야 합니다.

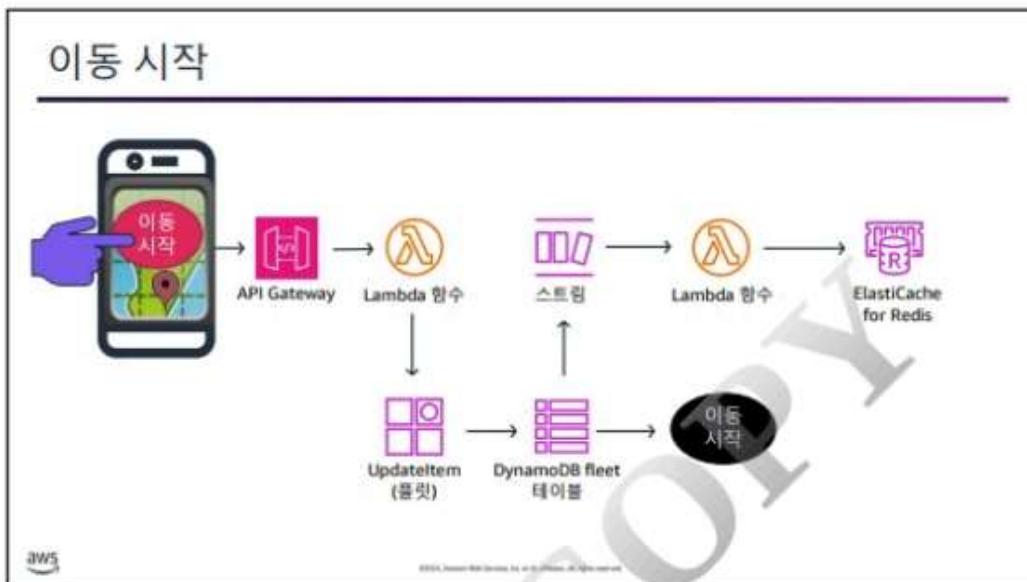
새 변경에서 이벤트 필터는 **AVAILABLE** 또는 **IN_USE**가 포함된 상태를 찾습니다. 그러면 **LOW_BATTERY** 상태로 변경되는 자산에 대한 이벤트가 제거됩니다.



사용자가 모바일 앱을 실행합니다.

모바일 앱이 지도를 탐색하여 사용자의 현재 위치와 가장 가까운 자전거와 스쿠터를 표시합니다.

모바일 앱이 검색 API를 사용하여 ElastiCache for Redis에서 최신 자전거 가용성 정보를 읽습니다.

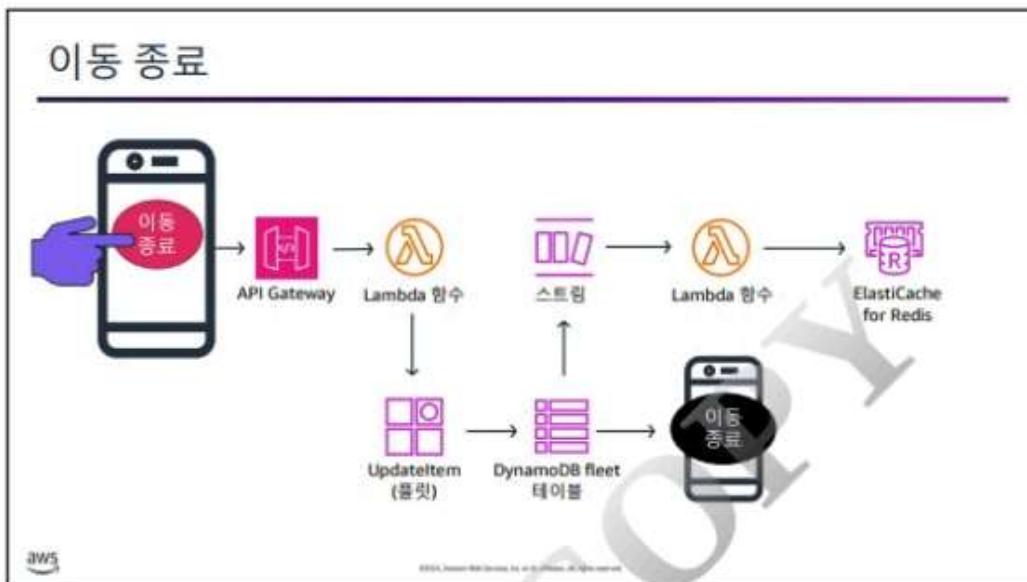


사용자가 앱에서 자전거를 선택한 후 **Start trip**을 선택합니다. 앱이 API Gateway에 요청을 보냅니다.

API 요청이 Lambda 함수를 시작합니다. 이 함수는 자산 ID를 기반으로 플랫 항목을 업데이트합니다.

DynamoDB Streams가 스트림 레코드의 업데이트를 캡처합니다.

스트림 레코드는 상태를 **In use**로 변경하여 지리 검색에서 제거하는 또 다른 Lambda 함수를 시작합니다.



사용자가 앱에서 **End trip**을 선택합니다. 앱이 API Gateway에 요청을 보냅니다.

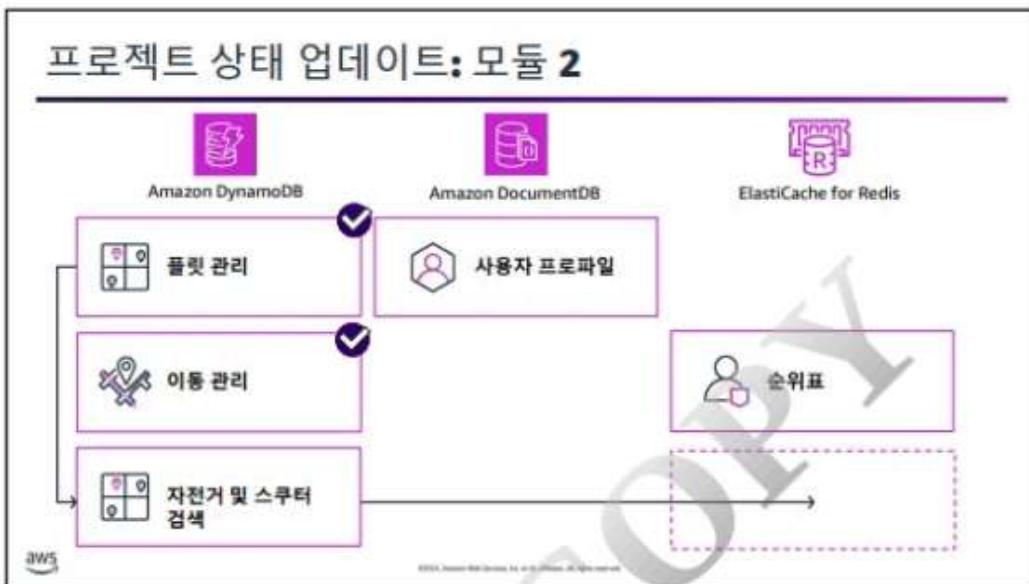
API 요청이 Lambda 함수를 시작합니다. 이 함수는 자산 ID를 기반으로 플랫 항목을 업데이트합니다.

DynamoDB Streams가 스트림 레코드의 업데이트를 캡처합니다.

스트림 레코드는 상태를 **Available**로 변경하여 지리 검색에서 제거하는 또 다른 Lambda 함수를 시작합니다.

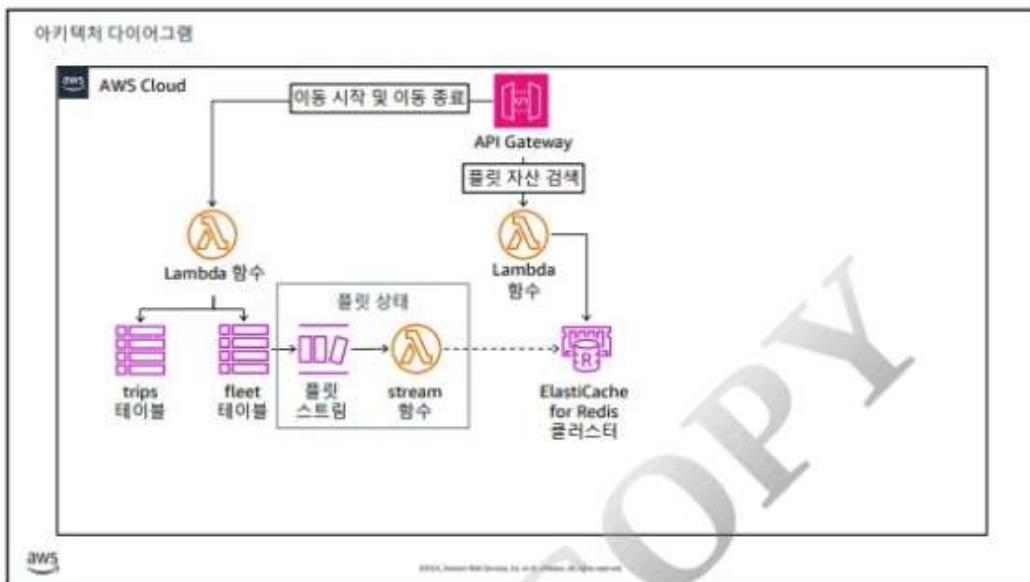
앱에서 이동이 종료되고 영수증이 표시됩니다.





플랫 및 이동 관리 기본 데이터 스토어 구축 준비를 마쳤습니다. 뿐만 아니라 ElastiCache for Redis에서 자산 위치와 배터리 상태를 추적할 계획입니다. DynamoDB Streams는 기본 테이블의 변경 내용을 추적하고 효율적으로 자전거를 검색하는 데 사용됩니다.

자전거 및 스쿠터 검색 워크로드는 모듈 4: 고급 Amazon ElastiCache for Redis 개념에서 다시 설명합니다.



지금까지 구축한 솔루션을 좀 더 기술적으로 요약한 것입니다.

이동 시작 및 종료 작업의 경우 API(API Gateway에 내장)는 항목 생성과 fleet 및 trips 테이블의 업데이트를 처리하는 단일 Lambda 함수를 시작합니다.

fleet 테이블에서 DynamoDB 스트림이 사용됩니다. 이벤트 필터는 레코드의 일부만 다른 Lambda 함수에 전달하여 ElastiCache for Redis 클러스터에 수집합니다. (ElastiCache for Redis에 대한 자세한 내용은 이 과정의 후반부에서 살펴보겠습니다.)

ElastiCache for Redis에 자산 위치와 배터리 잔량을 저장하면 다른 자전거 검색 API가 요청자의 지역에서 사용 가능한 자전거를 읽는 다른 함수를 시작할 수 있습니다.

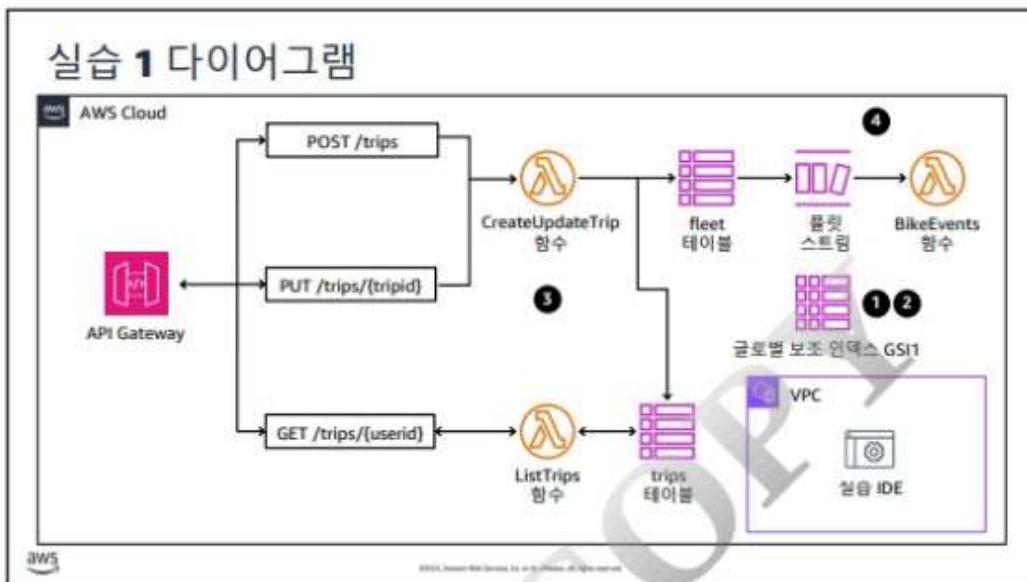
실습 1



**Amazon DynamoDB 테이블, 인덱스 및
변경 스트림을 사용하여 플릿 및 이동 데이터
관리 시스템 구현**

이 실습에서는 다음 태스크를 수행합니다.

1. 글로벌 보조 인덱스를 사용하여 배터리 잔량이 얼마 남지 않은 자산을 찾습니다.
2. GSI를 오버로드하고 자산별 미해결 서비스 주문을 찾습니다.
3. 이동 관리 Lambda 함수를 설정합니다.
4. 이동 시작 및 종료 이벤트를 감지하도록 DynamoDB Streams를 구성합니다.



이미지 설명: API Gateway는 POST /trips, PUT /trips/{tripid} 및 GET /trips/{userid}의 세 가지 API 작업을 관리합니다. POST 및 PUT 작업은 CreateUpdateTrip Lambda 함수를 시작합니다. 이 함수는 플릿 스트림을 사용하도록 설정된 trips 테이블과 fleet 테이블을 업데이트합니다. fleet 테이블에는 글로벌 보조 인덱스 GSI1이 있습니다. BikeEvents 함수는 플릿 스트림에서 필터링된 이벤트를 처리합니다. GET /trips/{userid} API 작업은 ListTrips 함수를 시작하여 trip 테이블에서 이동 레코드를 읽습니다. 실습 IDE를 관리하기 위해 Virtual Private Cloud(VPC)가 표시됩니다. 1단계와 2단계에는 글로벌 보조 인덱스 GSI1이 개입됩니다. 3단계에는 CreateUpdateTrip 함수와 ListTrips 함수가 개입됩니다. 4단계에는 DynamoDB 플릿 스트림과 BikeEvents 함수가 개입됩니다. 이것으로 설명을 마칩니다.





DO NOT COPY
Ssleeeee1@gmail.com

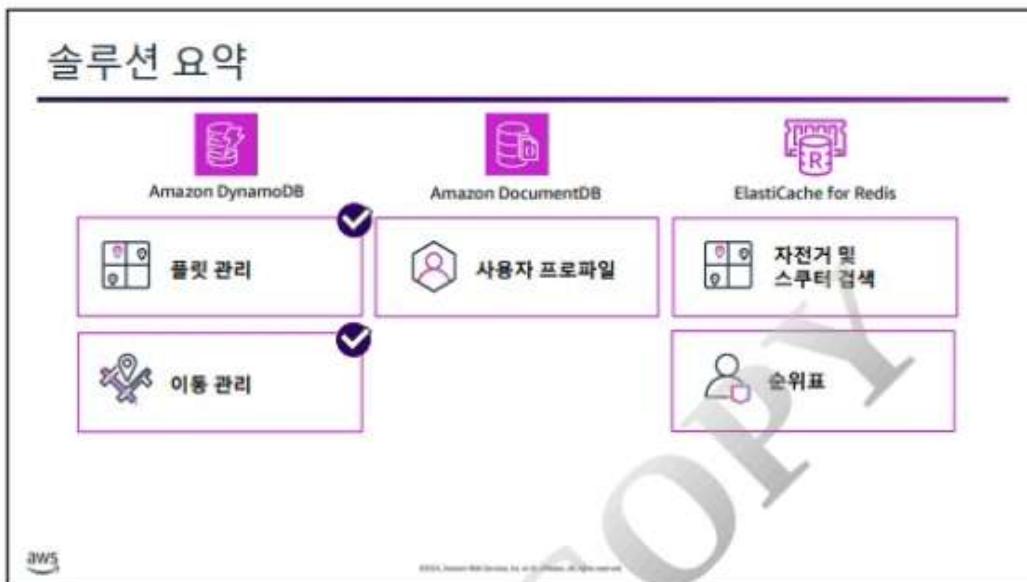
어젠다: 고급 Amazon DocumentDB 개념

- 사용자 프로파일 데이터 관리
 - 액세스 패턴
 - JSON 문서 구조
 - 데이터 모델
- 고객 점수 보고서
 - 액세스 패턴
 - 집계 프레임워크
 - 성능 기반 디자인
- 사용자 프로파일 액세스
 - 변경 스트림
- 실습 2: Amazon DocumentDB에서 사용자 프로파일 데이터 관리 워크로드 구현 및 최적화



© 2025, Amazon Web Services, Inc. or its affiliates. All rights reserved.

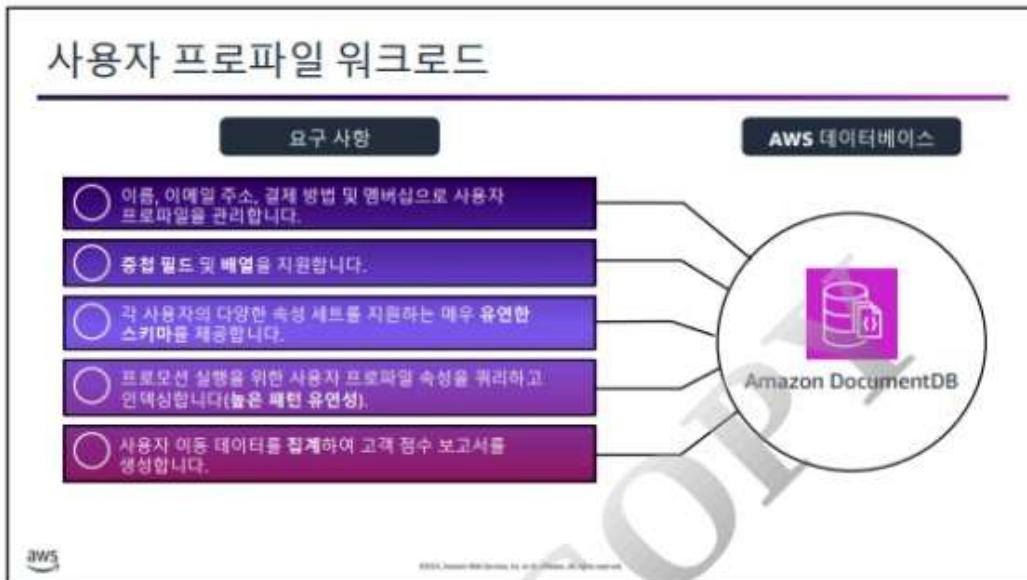
이 모듈에서 살펴볼 주제를 검토하십시오.



새로운 액세스 패턴을 검토하기 전에, 솔루션 요약 내용을 살펴보십시오.

플랫 및 이동 관리 기본 데이터 스토어 구축 준비를 마쳤습니다. 뿐만 아니라 ElastiCache for Redis에서 자산 위치와 배터리 상태를 추적할 계획입니다. DynamoDB Streams는 기본 테이블의 변경 내용을 추적하고 효율적으로 자전거를 검색하는 데 사용됩니다.

다음으로, 사용자 프로파일 워크로드 요구 사항을 확인합니다.



모바일 앱의 사용자 프로파일 워크로드 요구 사항을 검토하십시오. 이 데이터베이스의 목적은 프로파일 데이터를 안정적으로 저장하는 것입니다. 여기에는 사용자 식별자(사용자 ID), 이름, 서비스 결제 방법 및 멤버십 데이터가 포함됩니다. 또한 사용자마다 고유할 수 있는 동적 프로파일 스키마를 처리해야 합니다. 신용 카드를 하나만 저장하는 사용자도 있고, 속성이 서로 다른 신용 카드와 은행 계좌를 모두 저장하는 사용자도 있습니다.

가장 중요한 것은 다양한 패턴을 사용하여 사용자 프로파일을 쿼리하고 인덱싱한다는 것입니다. 어떤 워크플로에서는 올해에 연간 멤버십에 가입한 특정 도시의 사용자에 대한 보고서를 앱에서 생성할 수 있고, 또 어떤 워크플로에서는 기본 결제 방법으로 사용자 수를 생성할 수도 있습니다. 유연한 쿼리 및 집계와 함께 동적 속성과 값을 관리하려면 매우 유연한 스키마가 필요합니다.

사용자 프로파일 액세스 패턴

- 가입할 때 사용자 프로파일을 생성합니다.
- 로그인할 때 전체 프로파일을 가져옵니다.
- 결제 방법, 계정 기본 설정 및 멤버십을 추가하고 업데이트합니다.
- 사용자 프로파일을 쿼리하여 새 프로모션을 시작합니다.



aws

© 2025, Amazon Web Services, Inc. or its affiliates. All rights reserved.

사용자 프로파일 관리는 AnyCompany BikeShare 앱의 핵심 워크로드 중 하나입니다. 다음은 사용자 경험과 비즈니스에 직접적인 영향을 미치는 중요한 기능입니다.

- 사용자는 가입할 때 안정적으로 사용자 프로파일을 생성할 수 있어야 합니다.
- 프로파일을 신속하게 검색하여 사용자의 디바이스에 표시해야 합니다.
- 사용자는 결제 방법, 연락처 정보, 계정 기본 설정 및 멤버십을 업데이트할 수 있어야 합니다. 이러한 데이터는 여러 디바이스에서 액세스할 수 있고 쉽게 업데이트할 수 있어야 합니다.
- 마케팅 부서에서는 고객에게 가격 할인 및 프로모션을 제공하기 위해 사용자 프로파일을 쿼리합니다.

마케팅 캠페인은 같은 경우가 거의 없으므로 몇 년 전에 미리 예측할 수 없습니다. 향후 프로모션에 다음이 포함될 수 있습니다.

- 커피숍 25% 할인 혜택을 제공합니다.
- AnyCompany DigiPay 디지털 지갑 결제 방법을 기본 결제 유형으로 추가하는 사용자에게 5달러 기프트 카드를 제공합니다.

다음 마케팅 캠페인 예시를 검토하고, 프로파일 데이터를 사용하여 특정 대상을 타겟팅하는 방법을 고민해 보십시오.

- 마케팅 캠페인 예시:
 - 사용자가 AnyCompany DigiPay 디지털 지갑을 기본 결제 방법으로 설정하면 5달러 기프트 카드를 제공하여 AnyCompany DigiPay 디지털 지갑 파트너를 홍보합니다.
- 기준:
 - 멤버십 유형이 일일 이용권입니다.
 - 결제 방법이 AnyCompany DigiPay(anyDigiPay)입니다.
- 쿼리:
 - 일일 이용권을 보유하고 있으며 DigiPay를 기본 결제 방법으로 사용하는 사용자를 찾습니다.
- 업데이트:
 - 사용자 프로파일에 5달러 기프트 카드를 추가합니다.



이 섹션에서는 이 과정과 실습에서 사용되는 JSON 문서 구조에 대해 설명합니다.



기본 데이터 엔터티는 프로파일, 결제 방법 및 프로모션입니다.

프로파일에는 사용자 경험과 관련된 다양한 데이터 포인트가 포함됩니다. 이름, 사용자 ID, 청구 주소, 전화번호, 이메일 주소 등의 속성을 고려해야 합니다. 프로파일에는 중요한 멤버십 정보와 적격 마케팅 프로모션도 저장됩니다.

여러 결제 방법도 프로파일에 저장됩니다. 각 결제 방법에는 여러 속성 중 하나가 포함됩니다. 결제 방법이 신용 카드인 경우 카드에는 이름, 카드 번호 및 만료 날짜에 대한 값이 할당됩니다. AnyCompany DigiPay와 같은 디지털 결제 방법인 경우 고유 식별자(이메일 주소) 및 기타 기본 정보만 기록하면 될 수도 있습니다. 어떤 경우든 기본 결제 유형을 지정해야 합니다. 기본 속성을 사용하여 신용 카드인지 아니면 디지털 결제 유형인지 표시할 수 있습니다.

또한 프로파일에 여러 프로모션이 포함될 수 있습니다. 프로모션 속성은 향후 마케팅 활동에 따라 변경될 수 있습니다. 최소한 각 프로모션에는 제안 ID(문자열 데이터 형식)와 사용된 값(부울 데이터 형식)이 있어야 합니다.



DO NOT COPY
Ssleeeee1@gmail.com

간소화된 **userprofile** 문서

```
{
  "_id": "u100009",
  "name": "Martha Rivera",
  "email": "martha.rivera@example.com",
  "phone": "518-555-0130",
  "billingAddress": {
    "streetAddress1": "123 Any Street",
    "city": "Anytown",
    "state": "NY",
    "postal-code": "12345-6789",
    "country": "United States"
  },
  "membership": "DAYPASS"
}
```

간단한 설명을 위해 다른 예에서는 JSON에서 **billingAddress**가 제거됩니다.

userprofile 컬렉션의 단순화된 사용자 프로파일 예제를 검토하십시오.

다이어그램에서는 다음 속성을 간략하게 설명합니다.

- **_id** – 사용자 로그인을 위한 고유 식별자입니다. Amazon DocumentDB는 기본적으로 이 필드의 고유 인덱스를 유지합니다. **_id** 필드를 사용하여 사용자 ID를 저장하면 사용자 프로파일을 중복해서 생성할 수 없습니다.
- **name** – 사용자의 법적 이름입니다.
- **email** – 사용자의 연락처 이메일 주소입니다.
- **phone** – 사용자의 연락처 전화번호입니다.
- **billingAddress**(중첩 필드 포함) – 사용자의 청구 주소로, 결제 방법에 사용됩니다.
- **membership**(선택 사항) – 사용자가 첨부한 멤버십 플랜입니다.

디자인에서 팀은 주소 줄, 구/군/시, 시/도, 우편 번호 및 국가의 문자열 데이터를 별도의 필드에 저장하는 문서를 제공하기로 결정했습니다. 이러한 필드를 저장하면 데이터 주소를 세분화할 수 있습니다. 특정 국가 또는 도시의 사용자 찾기와 같은 쿼리 패턴을 사용할 수 있습니다. 또한 청구 주소와 우편 주소가 다른 경우 둘 다 저장할 수 있는 유연성을 제공합니다.

결제 방법 저장 요구 사항

1. 결제 방법을 5개까지 저장할 수 있습니다.
2. AnyCompany DigiPay(디지털 결제) 및 신용 카드를 지원합니다.
두 가지 결제 유형은 저장되는 속성이 다릅니다.



© 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

모바일 앱에서 사용자는 자신의 프로파일에 결제 방법을 5개까지 저장할 수 있습니다.

문서에서 AnyCompany DigiPay 모바일 지갑 서비스도 지원해야 합니다. 모바일 지갑의 속성이 신용 카드의 속성과 어떻게 다른지 생각해 보십시오. 두 결제 방법의 동일한 속성은 무엇일까요? 무엇이 다를까요?

도큐먼트 데이터베이스를 사용하면 여러 가지 결제 유형을 동일한 중첩 배열에 저장할 수 있는 유연성이 제공됩니다.

결제 방법 JSON 구조

신용 카드 결제 유형	디지털 지갑 결제 유형
<pre>{ "nickname": "anyCardA1234", "vendor": "anyCardA", "type": "card", "name": "Martha Rivera", "expiration": "04-2026", "cardtoken": "HX46YT794RG" }</pre>	<pre>{ "nickname": "anyDigiPay123", "vendor": "anyDigiPay", "type": "digital", "email": "martha.rivera@example.com", "default": 1 }</pre>

AWS, Amazon Web Services, Inc. or its affiliates. All rights reserved.

첫 번째 예제의 JSON은 카드 별칭, 유형, 카드 소지자 이름, 만료 날짜, 카드 토큰(카드 데이터를 안전하게 저장하기 위해 판매자가 생성한 토큰) 등 신용 카드 저장에 사용되는 속성을 보여줍니다.

두 번째 예제의 디지털 결제 속성은 이와 다릅니다. Amazon DocumentDB는 이러한 두 가지 결제 유형을 동일한 배열에 저장할 수 있는 스키마 유연성을 제공합니다. 여기서 사용자 프로파일은 기본 결제 유형이 AnyCompany DigiPay 유형인 것을 보여줍니다.

userprofile 컬렉션에 결제 방법 저장

```
{  
    "userid" : "u100009",  
    "name" : "Martha Rivera",  
    "email" : "martha.rivera@example.com",  
    "phone" : "518-555-0130",  
    "payment_methods" : [  
        {"nickname": "anyCardA1234", "vendor": "anyCardA", "name": "Martha  
Rivera", "expiration": "04-2026", "cardtoken": "HX46YT794BG"},  
        {"nickname": "anyDigiPay", "vendor": "anyDigiPay", "type": "digital",  
        "email": "martha.rivera@example.com", "default": 1}  
    ]  
}
```



© 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

여기서는 두 가지 결제 유형이 동일한 `Payment_methods` 배열에 표시되었습니다.

JSON의 멤버십 및 기본 설정

```
{  
    "membership": "DAYPASS",  
    "preferences" : {  
        "notifications" : {  
            "mobile" : {  
                "sms" : "true",  
                "push" : "false"  
            },  
            "email" : ""  
        }  
    }  
}
```

문서는 멤버십 유형 및 계정 통신 기본 설정과 같은 정보로 끝납니다. 사용자는 SMS 알림만 받고 푸시 또는 이메일 알림은 거부할 수 있습니다. 이 멤버의 계정에는 일일 이용권이 할당되었습니다.

샘플 **userprofile** 문서

```
{  
    "_id": ObjectId("6498beaa75040579f4e8c02f"),  
    "userid": "u100009",  
    "name": "Martha Rivera",  
    "email": "martha.rivera@example.com",  
    "phone": "518-555-0130",  
    "payment_methods": [  
        {"nickname": "anyCard41234", "vendor": "anyCardA", "name": "Martha Rivera",  
         "expiration": "04-2026", "cardtoken": "HX46YT794RG"},  
        {"nickname": "anyDigiPay", "vendor": "anyDigiPay", "type": "digital",  
         "email": "martha.rivera@example.com", "default": 1}  
    ],  
    "membership": "DAYPASS",  
    "preferences": {  
        "notifications": {  
            "mobile": {  
                "sms": "true",  
                "push": "false"  
            },  
            "email": ""  
        }  
    }  
}
```

이 최종 문서 예제에서는 이 모듈과 실습 2에서 사용된 전체 JSON 문서 구조를 검토할 수 있습니다.

인덱스 유연성 예제

중첩 필드에 Sparse index 를 생성합니다.

```
db.userprofile.createIndex({"payment_methods.vendor":1,  
"payment_methods.default": 1},{sparse: true, name:  
"sparse_payment_defaults"});  
  
db.userprofile.explain('executionStats').find(  
{"payment_methods.vendor": "anyDigiPay", "payment_methods.defau  
lt":{ $exists: true}}).count();
```

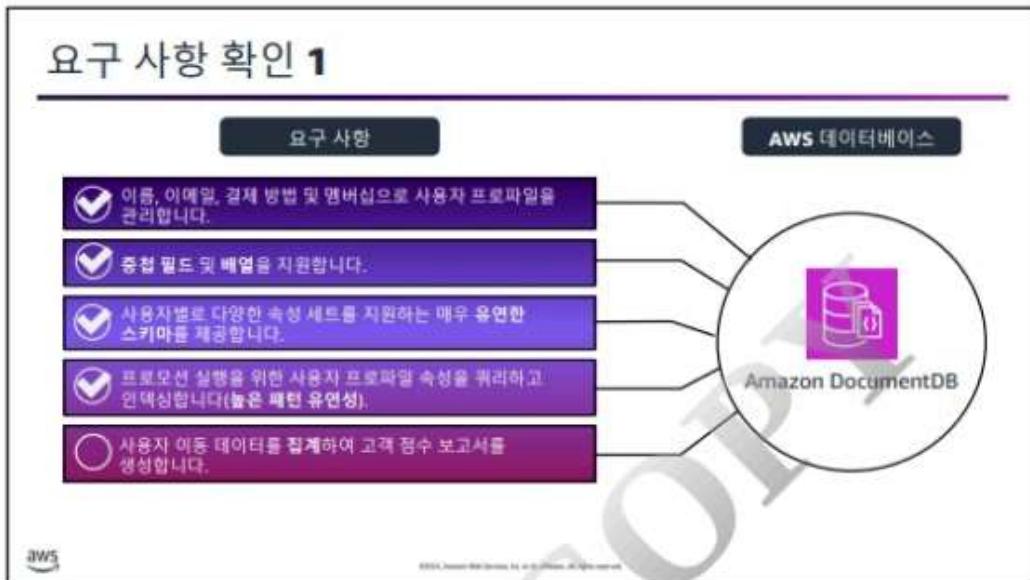


© 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

앱의 액세스 패턴에는 프로모션을 시작하기 위한 프로파일 데이터 쿼리가 포함됩니다. 앞에서 AnyCompany DigiPay를 기본 결제 방법으로 추가하는 사용자에게 기프트 카드를 제공하는 프로모션을 생각했습니다.

첫 번째 `createIndex` 명령을 사용하여 벤더 및 기본 결제 방법 유형에 따른 인덱스를 생성할 수 있습니다. 그러면 기본 결제 방법을 사용하는 사용자 프로파일 문서만 포함되는 Sparse index 가 생성됩니다.

기본 결제 유형이 `anyDigiPay` 문자열과 일치하는 계정을 찾는 쿼리를 실행할 수도 있습니다.



모바일 앱의 워크로드 요구 사항을 확인하십시오.

여러분은 사용자 이동 데이터를 집계하여 고객 점수 보고서를 생성하는 방법을 계속 연구하는 중입니다.



제품 관리자가 기능 요청을 위해 여러분에게 연락합니다. 고객 점수 보고서는 고객 관계 관리(CRM) 팀과 마케팅 팀이 고객 유지와 맞춤형 제안 및 할인을 제공하기 위해 사용하는 내부 보고서입니다.

마케팅 팀은 이동 거리나 총 지출과 같은 요소를 기반으로 고객 충성도를 파악할 수 있는 보고서를 운영해야 합니다. AnyCompany BikeShare는 이 데이터를 고객 유지, 맞춤형 제안 및 가격 할인에 사용하려 합니다.

고객 점수 보고서: 액세스 패턴

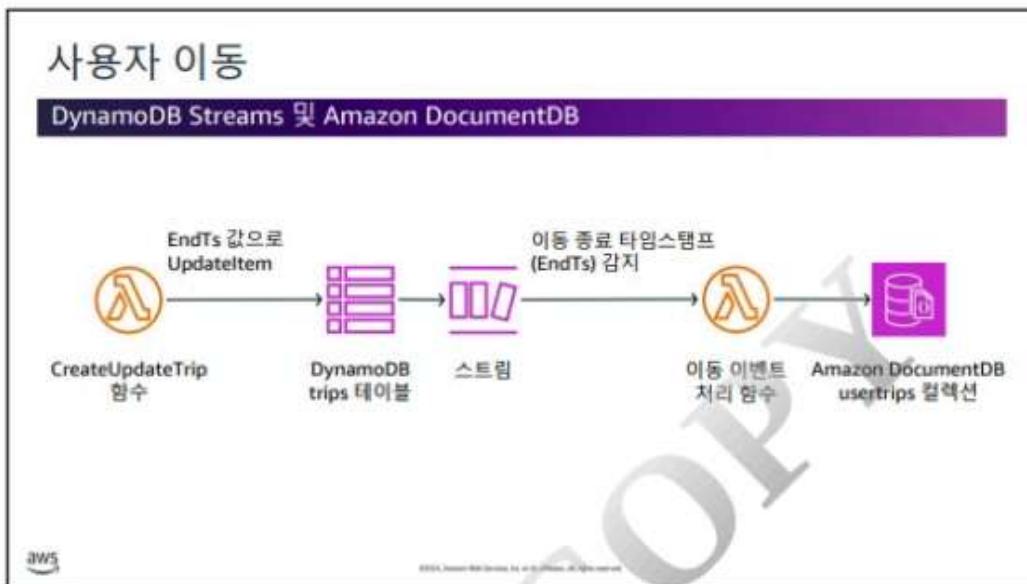
- 고객의 지출 데이터를 수집합니다.
- 고객 이동 보고서를 생성합니다.



aws

© 2025, Amazon Web Services, Inc. or its affiliates. All rights reserved.

마케팅 팀은 온디맨드 사용자를 추적하고 특정 기간 동안 이들의 총 이용 요금을 계산하려 합니다. 또한 마케팅 팀은 특별 제안이나 마케팅 캠페인을 통해 AnyCompany BikeShare 고객을 대상으로 하는 고객 점수 보고서를 생성하려 합니다. 예를 들어 자주 이용하는 사용자를 찾아서 멤버십 이용권을 제안할 수 있습니다. 뿐만 아니라 마케팅 팀은 이용 금액이 가장 큰 사용자를 대상으로 주변에 상품을 알리면 보너스를 제공하는 방식으로 신규 고객을 유치할 수 있습니다.



모듈 2에서는 fleet 테이블에서 DynamoDB Streams를 사용합니다. 기존 trips 테이블에서 DynamoDB Streams를 사용하여 집계 및 보고에 사용할 사용자 이동 데이터를 Amazon DocumentDB로 수집할 수 있습니다.

이동 스트림에서 애플리케이션은 특정 속성이 포함된 이벤트를 필터링할 수 있습니다. Amazon DocumentDB usertrips 컬렉션을 생성하려면 먼저 EndTs 속성의 이벤트 필터링부터 시작합니다. 이동 종료 타임스탬프를 추가한다는 것은 이동이 끝났다는 뜻이며 모든 관련 이동 데이터가 포함되어야 합니다.

이 예제의 이동 이벤트 처리 함수는 완료된 이동만 처리하며, 추가로 처리할 소량의 데이터만 수집합니다.

무제한 데이터: User trips

이동 데이터가
커질수록 JSON 문서
크기도 함께
커집니다.

```
"userTrips": [
    {
        "userid": "u264733",
        "tripid": "t#2022-08-18T00:10:13#esa958935134",
        "tripdate": ISODate("2022-08-18T14:19:13.128Z"),
        "miles": 1.5,
        "fare": 11
    },
    {
        "userid": "u264733",
        "tripid": "t#2022-08-18T00:10:13#esa958935134",
        "tripdate": ISODate("2022-08-19T19:10:23.128Z"),
        "miles": 0.4,
        "fare": 4
    },
    {
        "userid": "u264733",
        "tripid": "t#2022-08-18T00:10:13#esa958935134",
        "tripdate": ISODate("2022-08-20T11:47:44.128Z"),
        "miles": 0.25,
        "fare": 2
    }
]
```



이동 데이터를 어디에 저장해야 하는지 고민해 보십시오.

사용자 이동 중첩 배열을 `userprofile` 문서에 저장하는 것이 합리적인 생각입니다. 사용자는 이동하고, 이 데이터는 관련성이 높습니다. 하지만 사용자 이동 빈도에 따라 총 이동 횟수가 제한되지 않을 수 있습니다. 따라서 `userprofile` 문서 내의 배열에 저장하는 것은 권장하지 않습니다.

문서 크기를 작게 유지

- 변경 가능한 데이터를 변경 불가능한 데이터와 분리합니다(두 개의 문서 컬렉션으로 분할).
- 문서를 8킬로바이트(KB) 이하로 작성합니다.
- 성능 향상을 위해 I/O 작업을 줄입니다.

```
{
  "_id": "t#2022-08-18T00:10:13#esa958935134",
  "userid" : "U264733",
  "tripdate" : ISODate("2022-08-20T11:47:44.128Z"),
  "miles" : 0.25,
  "fare" : 2
}
```



© 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Amazon DocumentDB에서는 문서 크기가 16메가바이트(MB)로 제한됩니다. 최상의 성능을 얻으려면 문서 크기를 8KB 이하로 유지하십시오. 문서가 작을수록 성능이 향상됩니다.

이 시나리오에서는 이동 데이터를 자체 컬렉션에 저장하고 `userid` 속성으로 참조하면 `userprofile` 문서 크기가 작게 유지됩니다. `userprofile` 컬렉션과 `usertrips` 컬렉션을 `userid` 속성으로 조인하여 보고서를 생성할 수 있습니다.

아래의 포함된 속성과 해당 데이터 형식을 검토하십시오.

- `userid`(문자열)
- `tripid`(문자열)
- `tripdate`(날짜)
- `miles(double)`
- `fare(double)`

다음 슬라이드에서는 Amazon DocumentDB 집계 프레임워크의 구조를 살펴보십시오.

집계 파이프라인

db.<collection>.aggregate()

메서드를 사용합니다.

컬렉션의 문서는 정의된 스테이지 순서대로 한 스테이지에서 다음 스테이지로 전달됩니다.

```
db.collection.aggregate( [  
    { <stage1> },  
    { <stage2> },  
    { <stage3> },  
    ...  
] )
```



© 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

고객 보고서를 운영하려면 집계 파이프라인이 필요합니다. 집계 파이프라인은 하나 이상의 스테이지로 구성됩니다. 각 스테이지에서는 필터링, 그룹화, 계산 등의 문서 작업을 처리합니다. 완료되면 출력이 다음 집계 스테이지로 이동합니다.



집계 예제

```
db.usertrips.aggregate(  
  [  
    {  
      $match: {  
        "tripdate": {  
          $gte: ISODate('2023-01-01'),  
          $lt: ISODate('2023-04-01')  
        }  
      }  
    },  
    { $project: {  
      _id: 0, userid: 1, fare: 1  
    } },  
    { $group: { _id: "$userid", revenue: { $sum: "$fare" } } },  
    { $sort : { revenue: -1 } }  
  ]  
)
```



© 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

이 집계 예제에서는 먼저 이동 날짜로 문서를 필터링하여 2023년 1분기에 발생한 이동만 검색합니다.

그 후 `userid` 및 `fare` 값이 있는 결과만 프로젝션합니다. 그러면 해당 요금은 `userid`로 그룹화되고 해당 날짜 범위 동안 전체 사용자 이용 내역의 요금이 합산됩니다.

결과는 매출이 가장 높은 것부터 가장 낮은 순서로 정렬됩니다. 이용 금액이 가장 큰 사용자가 가장 먼저 나열됩니다.

집계에서 중요한 순서(1/2)

스테이지 1

스테이지 2

스테이지 n

\$match → \$project → \$sort

- 파이프라인을 통해 흐르는 문서의 양을 줄입니다.
- 필수 필드만 프로젝션합니다.
- 미리 정렬한다고 해서 출력의 정렬 순서가 보장되지는 않습니다.
- 이후 스테이지를 처리하는 데 필요한 메모리를 줄입니다.
- 가능하다면 끝까지 정렬합니다.



© 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

작업 순서는 \$match, \$project, \$sort입니다. 이 예제에서 스테이지 n은 집계 파이프라인의 마지막 스테이지를 나타냅니다.

스테이지 1(\$match)

인덱스를 사용하고 파이프라인을 통해 이동하는 문서의 양을 줄입니다. 가능한 한 파이프라인의 초기 스테이지에 배치합니다.

스테이지 2(\$project)

필수 필드만 프로젝션합니다. 이후 스테이지를 처리하는 데 필요한 메모리를 줄입니다. _id 속성을 0으로 설정합니다.

스테이지 n(\$sort)

파이프라인의 끝을 향해 데이터를 정렬합니다. 미리 정렬한다고 해서 출력의 정렬 순서가 보장되지는 않습니다. \$group, \$match, \$project와 같은 이전 스테이지에서 정렬할 데이터를 줄이면 소비되는 메모리를 줄일 수 있습니다.

집계에서 중요한 순서(2/2)

스테이지 1

`$project`

스테이지 2

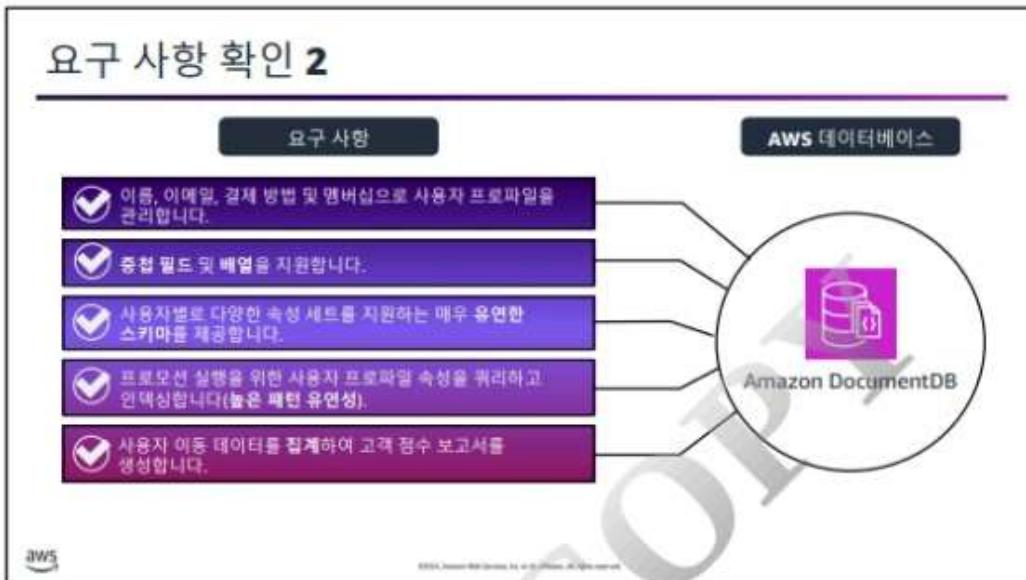
`$match`

- `$project`는 가용 인덱스를 사용할 수 없습니다.
- Amazon DocumentDB는 전체 컬렉션을 스캔하므로 캐시 오버런, 높은 I/O 비용, 성능 저하로 이어집니다.
- `$match` 역시 가용 인덱스를 사용할 수 없습니다.
- 파이프라인의 이전 스테이지에서 인덱스를 사용하지 않으면 모든 후속 스테이지가 인덱스를 사용하도록(해당하는 경우) 최적화되지 않습니다.


© 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

`$match` 스테이지를 `$project` 스테이지 뒤에 배치하면 어떻게 될지 생각해 보십시오. 이전 스테이지(`$project`)에서 인덱스를 사용하지 않았으므로 Amazon DocumentDB는 인덱스를 사용하여 효율성을 최적화할 수 없습니다.

연산자에 대한 자세한 내용은 **Amazon DocumentDB 개발자 가이드**의 '지원되는 MongoDB API, 작업 및 데이터 형식' (<https://docs.aws.amazon.com/documentdb/latest/developerguide/mongo-apis.html>)을 참조하십시오.



사용자 프로파일 워크로드에 대한 핵심 요구 사항을 모두 충족했습니다. 이제 빌드를 더욱 효율적으로 만들려면 어떻게 해야 하는지 고민해 보십시오.

마케팅 팀이 찾아와서 특정 날짜 범위의 가장 긴 이동부터 가장 짧은 이동 순으로 검색하는 기능을 구현해 달라고 요청한다고 상상해 보십시오. 인덱스를 성능 중심으로 구축하려면 어떻게 해야 할까요?



다음으로, 복합 쿼리 예제를 검토하고 Amazon DocumentDB 복합 인덱스를 최적화하겠습니다.

쿼리 예제

사용자의 2023년 1월 1일 이후 이동을 찾아서 가장 긴 이동부터 가장 짧은 이동 순으로 표시하십시오.

```
db.usertrips.find( {  
    _id: "U123235",  
    tripdate: {  
        $gte: ISODate('2023-01-01')  
    } }.sort( { miles: -1 } )  
})
```



© 2025, Amazon Web Services, Inc. or its affiliates. All rights reserved.

가장 자주 사용하는 쿼리를 최적화하십시오. 최적화되지 않은 실행 계획이 포함된 쿼리는 성능이 저하되고 I/O 비용이 높아집니다. 최적화되지 않은 쿼리를 자주 실행할수록 클러스터에서 워크로드를 실행하는 데 드는 비용과 성능에 악영향을 미치게 됩니다.

그래픽의 예제를 검토하십시오. 이 Amazon DocumentDB 예제에서는 usertrips 컬렉션을 쿼리하여 사용자의 가장 긴 이동 거리(마일)를 찾는 방법을 보여줍니다. 모든 사용자 이동은 동일한 컬렉션에 저장됩니다. 따라서 스캔한 데이터 집합은 결과 세트에 비해 상당히 높습니다.

이 쿼리는 인덱스를 사용하면 좋습니다. 다음으로, 인덱스를 생성하여 쿼리를 최적화하는 방법을 살펴보겠습니다.

ESR 규칙을 사용하여 복합 인덱스 디자인

```
db.usertrips.find( {
  _id: "u123235",
  tripdate: {
    $gte: ISODate('2023-01-01')
  }
}).sort( { miles: -1 })
})
```

ESR(Equality sort range) 규칙

Equality = _id
Sort = miles
Range = tripdate

```
db.usertrips.CreateIndex({ _id: 1, miles: -1, tripdate: 1 })
```


© 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

쿼리를 최적화하려면 ESR(Equality sort range) 규칙을 사용하십시오. 복합 인덱스에서는 열 순서가 중요합니다. 데이터베이스 실행 엔진이 작동하는 방식 때문입니다. ESR 규칙을 기반으로 쿼리를 작성하면 옵티マイ저는 여러분이 생성한 인덱스를 선택하여 쿼리를 실행하게 됩니다.

ESR 규칙을 사용하여 인덱스를 생성하려면 다음을 선택하십시오.

- **Equality = _id**
 - 정확한 값과 일치하는 필드를 첫 번째 인덱스 필드로 배치합니다.
- **Sort = miles**
 - 다음으로, 정렬 기준이 될 필드를 배치합니다.
- **Range: tripdate**
 - 다음으로, 값 범위에서 검색된 필드를 배치합니다.

복합 인덱스는 선택성을 높여 필드가 쿼리에 함께 나타날 때 잘 작동합니다.

Amazon DocumentDB는 다음을 비롯한 풍부한 인덱싱 기능을 제공합니다.

- 단일 필드 인덱스
- 복합 인덱스
- 다중 키 인덱스
- 복합 다중 키 인덱스
- Sparse index
- Time to Live(TTL) 인덱스

인덱스 유형에 대한 자세한 내용은 AWS Database 블로그의 'Amazon DocumentDB(MongoDB 호환)에서 인덱싱하는 방법'(<https://aws.amazon.com/blogs/database/how-to-index-on-amazon-documentdb-with-mongodb-compatibility/>)을 참조하십시오.



사용자 경험 관점에서 프로파일 액세스 패턴을 검토하십시오. 고성능 중심으로 새 애플리케이션을 최적화하는 것이 중요합니다.



고객이 자신의 사용자 프로파일에 어떻게 액세스합니까?

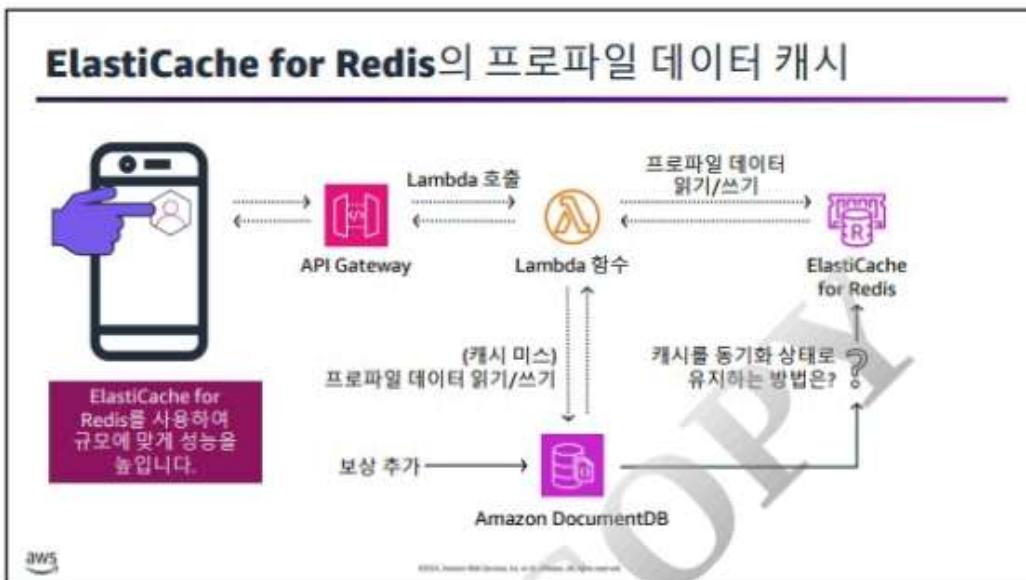
사용자가 모바일 앱을 열고 로그인하거나 프로파일 버튼을 누른다고 상상해 보십시오. 예제 애플리케이션의 모바일 앱은 데이터베이스에서 사용자의 프로파일을 가져오기 위해 API를 호출합니다. API Gateway는 데이터베이스에서 프로파일 데이터를 쿼리하는 Lambda 함수를 호출합니다. 사용자 프로파일 데이터는 항상 애플리케이션 로그인 시에 가져옵니다. 사용자 프로파일 데이터에는 이동을 시작할 때의 요금을 결정하는 회원 정보가 담겨있습니다.

또한 앱은 사용자가 사용할 수 있는 현재 제안과 보상을 표시합니다. 사용자는 요금을 할인해 주는 적격 보상을 받을 수 있으며, 사용자의 프로파일에서도 같은 내용을 확인할 수 있습니다.

모든 규모에서 일관된 성능을 제공해야 하는 데이터베이스 요구 사항을 고려하십시오.

- 사용자가 신속하게 로그인하여 이동을 시작할 수 있도록 사용자 프로파일 조회 속도가 1밀리초 미만으로 빨라야 합니다.
- 프로파일 조회 지연 시간은 트래픽에 상관없이 일관되어야 합니다.

이러한 두 가지 요구 사항을 충족하면 사용자에게 최적의 사용자 경험을 제공할 수 있습니다.



소속 팀에서 Amazon DocumentDB를 사용하여 유연한 스키마와 동적 쿼리 기능을 지원하고 있습니다. 여러분은 캐싱 계층을 도입하여 성능을 향상하려 합니다. 데이터베이스 캐시 계층을 추가하면 이러한 요구 사항이 충족됩니다. 프로파일 데이터는 전통적으로 정적이며 쓰기보다 읽기가 더 많은 트래픽 패턴이라는 것을 기억하십시오. 캐시 계층이 추가된 애플리케이션은 고성능을 제공하고 확장 가능합니다.

API 호출이 사용자 프로필을 업데이트하면 Lambda 함수가 캐시와 데이터베이스 모두에 기록하여 동기화 상태를 유지할 수 있습니다. 사용자는 읽기보다 쓰기에 더 관대한데, 요청에서 쓰기가 차지하는 비율은 극히 일부입니다.

마케팅 팀이 새 프로모션을 추가하거나 보상이 프라이머리 데이터베이스에서 직접 업데이트되면 어떻게 될까요? 사용자가 자신의 프로파일을 확인할 때, 업데이트된 데이터가 캐시에 없을 수 있습니다. 데이터가 더 이상 동기화 상태가 아닌 것입니다.

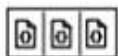
ElastiCache for Redis의 데이터를 최신 상태로 유지하려면 어떻게 해야 할까요? 다시 쓰기 전략이 필요한 경우 Amazon DocumentDB의 프로파일 데이터를 어떻게 ElastiCache for Redis와 동기화하셔겠습니까?

변경 스트림

변경 스트림은 클러스터의 컬렉션 내에서 발생하는 변경 이벤트의 시간 순서를 제공합니다.



변경 스트림을 사용하여 개별 컬렉션의 데이터 변경 내용을 구독할 수 있습니다.



변경 스트림 이벤트는 클러스터에서 발생하는 순서대로 정렬됩니다.



이벤트의 기본 보존 기간은 3시간이며, 최대 7일까지 변경할 수 있습니다.



클러스터에서 변경 스트림을 사용하면 읽기 및 쓰기 I/O와 스토리지 비용이 추가로 발생합니다.


© 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

변경 스트림은 클러스터의 컬렉션 및 데이터베이스 내에서 발생하는 업데이트 이벤트의 시간 순서를 제공합니다. 변경 스트림은 Amazon DocumentDB 클러스터의 프라이머리 인스턴스에 대한 연결에서만 열 수 있습니다. 복제본 인스턴스에서 변경 스트림 읽기는 현재 지원되지 않습니다.

변경 스트림에 기록된 이벤트는 기본적으로 3시간 동안 저장됩니다. 새로운 변경 사항이 발생하지 않더라도 로그 보존 기간이 경과하면 변경 스트림 데이터가 삭제됩니다.

클러스터에서 변경 스트림을 사용하면 추가 IOPS 및 스토리지 비용이 발생합니다. 애플리케이션은 변경 스트림을 사용하여 데이터베이스 또는 개별 컬렉션의 모든 데이터 변경 사항을 구독할 수 있습니다.

자세한 내용은 **Amazon DocumentDB 개발자 가이드**의 'Amazon DocumentDB에서 변경 스트림 사용'

(https://docs.aws.amazon.com/documentdb/latest/developerguide/change_streams.html)을 참조하십시오.



프로젝트 아키텍처에서 Amazon DocumentDB 변경 스트림은 Amazon DocumentDB 컬렉션에 이벤트를 기록합니다. 사용자 프로파일에 새 프로모션이 추가되면 캐시 업데이트 Lambda 함수가 호출되어 ElastiCache for Redis의 프로파일을 업데이트합니다.

다음 중 하나라도 해당되면 컬렉션에 변경 스트림이 사용됩니다.

- 데이터베이스와 컬렉션이 모두 명시적으로 사용됩니다.
- 컬렉션을 포함하는 데이터베이스가 사용됩니다.
- 모든 데이터베이스가 사용됩니다.

상위 데이터베이스에도 변경 스트림이 사용되거나 클러스터의 모든 데이터베이스가 사용되는 경우 데이터베이스의 컬렉션을 삭제해도 해당 컬렉션에 사용되는 변경 스트림이 사용 중지되지 않습니다. 삭제된 컬렉션과 동일한 이름으로 새 컬렉션이 생성되면 해당 컬렉션에 변경 스트림이 사용됩니다.

`$listChangeStreams` 집계 파이프라인 스테이지를 사용하면 클러스터에 사용되는 모든 변경 스트림을 나열할 수 있습니다. 파이프라인에서 Amazon DocumentDB가 지원하는 모든 집계 스테이지를 추가 처리에 사용할 수 있습니다. 이전에 사용하도록 설정된 컬렉션이 사용 중지된 경우 `$listChangeStreams` 출력에 표시되지 않습니다.

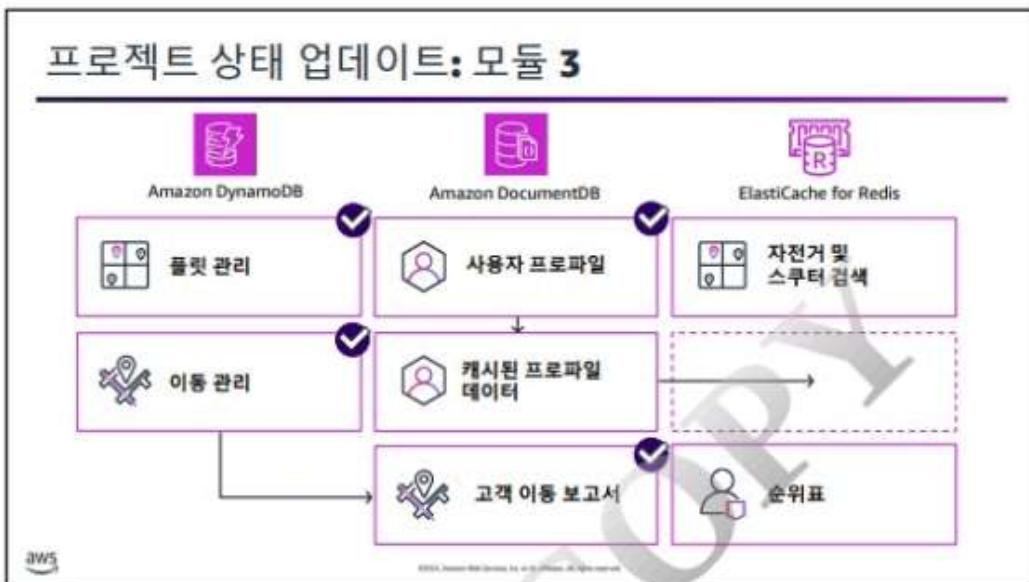


Amazon DocumentDB의 변경 스트림 기능은 클러스터의 컬렉션 내에서 발생하는 변경 이벤트의 시간 순서를 제공합니다. 워크로드에서 이러한 이벤트를 Lambda에 사용할 수 있습니다. Amazon DocumentDB 변경 스트림을 Lambda의 이벤트 소스로 사용하면 데이터베이스 변경에 거의 실시간으로 반응하는 이벤트 중심 애플리케이션을 만들 수 있습니다.

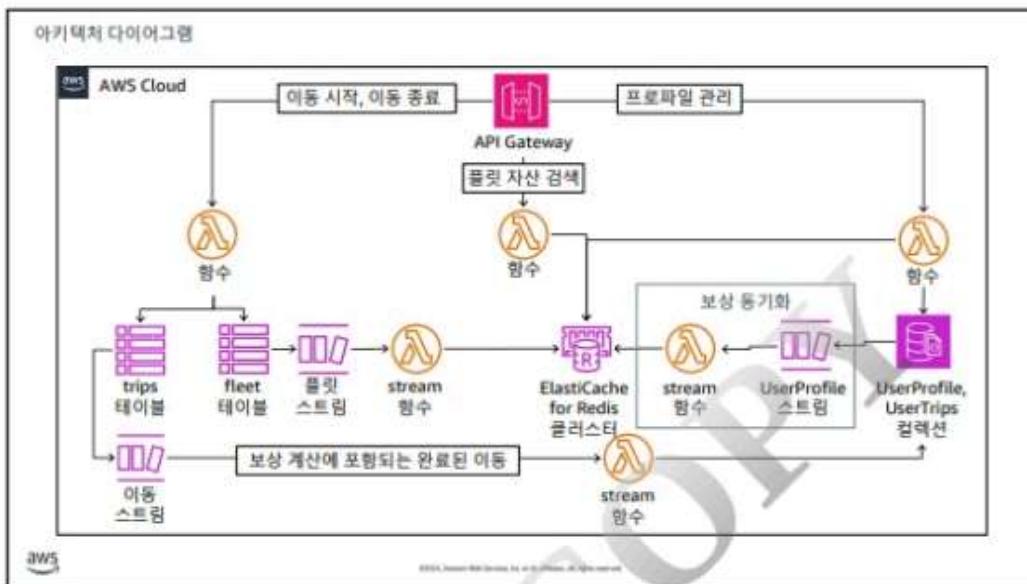
다이어그램에서 Lambda가 Amazon DocumentDB의 다음 이벤트 배치를 요청합니다. Amazon DocumentDB 서비스는 변경 스트림에서 이벤트를 가져와 해당 이벤트 배치를 Lambda에 전달합니다. Lambda는 각 JSON 페이로드를 해당 Lambda 함수로 보내서 호출합니다.

Lambda 이벤트 소스 매핑 구성에 대한 자세한 내용은 [AWS Lambda 개발자 가이드의 'Amazon DocumentDB에서 Lambda 사용'](#) (<https://docs.aws.amazon.com/lambda/latest/dg/with-documentdb.html>)을 참조하십시오.





사용자 프로파일 워크로드 구축 준비를 마쳤습니다. 사용자 프로파일 데이터는 Amazon DocumentDB 컬렉션의 JSON 문서에 저장됩니다. 완료된 이동 데이터는 DynamoDB에서 가져온 Amazon DocumentDB의 다른 컬렉션에도 저장됩니다. Amazon DocumentDB 프로파일 데이터는 ElastiCache for Redis에 캐시됩니다.



지금까지 구축한 솔루션을 좀 더 기술적으로 요약한 것입니다. 모듈 2에서는 이동 시작, 이동 종료 및 플릿 자산 검색 액세스 패턴을 구현했습니다.

프로파일 관리 API는 UserProfile 컬렉션과 ElastiCache for Redis 클러스터 모두에 쓸 수 있는 새 Lambda 함수를 트리거합니다. UserProfile 변경 내용은 Amazon DocumentDB 변경 스트림에서 추적되고, 스트림 함수에 의해 사용되며, ElastiCache for Redis에 백엔드 변경 내용을 적용합니다.

DynamoDB 스트림이 trips 테이블에 추가됩니다(**trip stream**이라는 레이블이 붙음). Lambda는 필터링되고 완료된 이동을 보상 계산에 사용합니다. 스트림 함수는 UserTrips 컬렉션을 변경합니다.

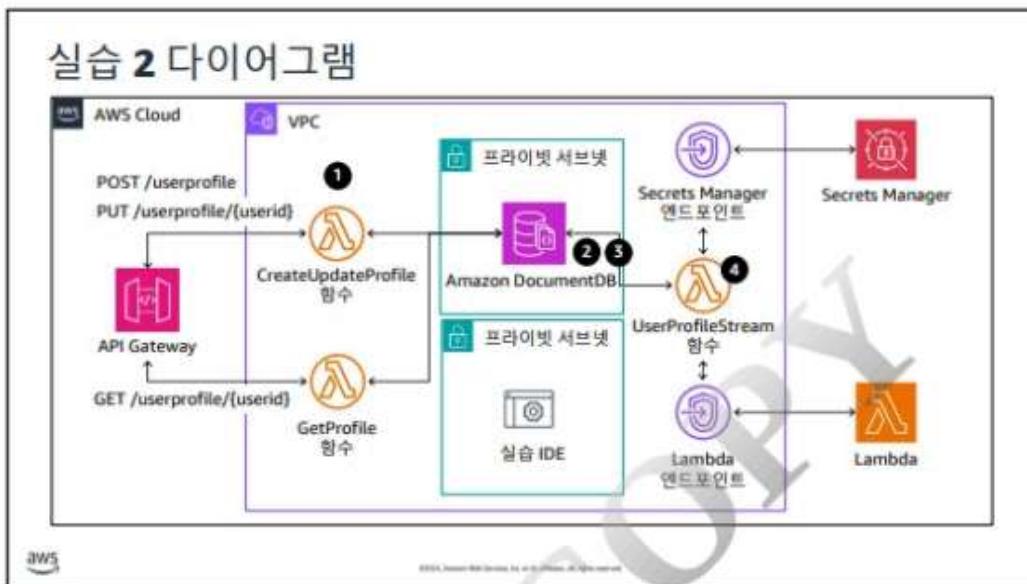
실습 2



Amazon DocumentDB에서 사용자 프로파일 데이터 관리 워크로드 구현 및 최적화

이 실습에서는 다음 태스크를 수행합니다.

1. 사용자 프로파일 관리 API를 설정합니다.
2. 집계 프레임워크를 사용하여 고객 점수 보고서를 생성합니다.
3. 인덱스 및 Amazon DocumentDB 프로파일러를 사용하여 고객 점수 쿼리를 최적화합니다.
4. 사용자 프로파일 업데이트를 위한 Amazon DocumentDB 변경 스트림을 설정합니다.



이미지 설명: 이 다이어그램은 2개의 프라이빗 서브넷이 있는 AWS 클라우드의 Virtual Private Cloud(VPC)를 보여줍니다. API Gateway는 VPC 외부에 POST userprofile, PUT userprofile 및 GET userprofile API를 갖고 있습니다. PUT 및 POST API는 한 프라이빗 서브넷에서 Amazon DocumentDB에 대한 읽기 및 쓰기 작업을 수행하는 CreateUpdateProfile 함수에 대한 직접 작업을 호출합니다. GET userprofile API는 Amazon DocumentDB에서 단일 프로파일을 검색하는 GetProfile 함수를 호출합니다. 표시되지 않은 Amazon DocumentDB 변경 스트림은 VPC 내부의 VPC 앤드포인트를 통해 Lambda 및 AWS Secrets Manager와 상호 작용하는 UserProfileStream Lambda 함수에 이벤트를 제공합니다. 숫자 4개는 실습에서 수행되는 네 단계와 해당 작업이 수행되는 위치를 하이라이트합니다.

1. 사용자 프로파일 관리 API를 설정합니다.
2. 단계 프레임워크를 사용하여 고객 점수 보고서를 생성합니다.
3. 인덱스 및 Amazon DocumentDB 프로파일리를 사용하여 고객 점수 쿼리를 최적화합니다.
4. 사용자 프로파일 업데이트를 위한 Amazon DocumentDB 변경 스트림을 설정합니다. 이것으로 설명을 마칩니다.





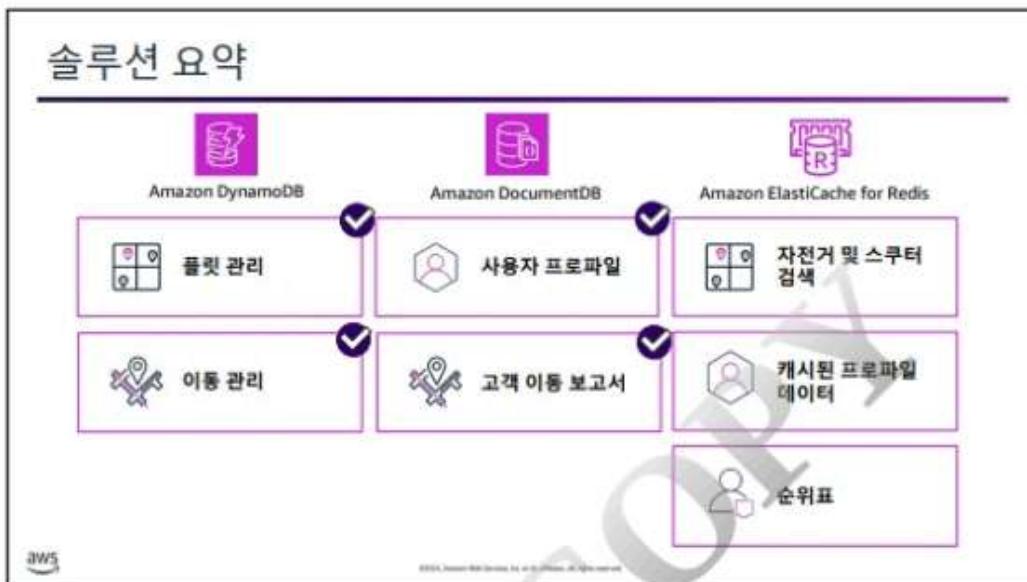
어젠다: 고급 ElastiCache for Redis 개념

- 자전거 및 스쿠터 검색
 - 데이터 구조 및 키 체계
 - 자전거 검색 알고리즘
 - 구현 예시
- 캐시된 프로파일 데이터
 - 데이터 구조 및 키 체계
 - 구현 예시
- 순위표
 - 준실시간 순위표 구현
 - 데이터 구조 및 키 체계
 - 아키텍처
 - 구현 예시
- 실습 3: Amazon ElastiCache for Redis를 사용하여 Geospatial 자전거 검색, 사용자 프로파일 캐싱 및 순위표 구현



© 2025, Amazon Web Services, Inc. or its affiliates. All rights reserved.

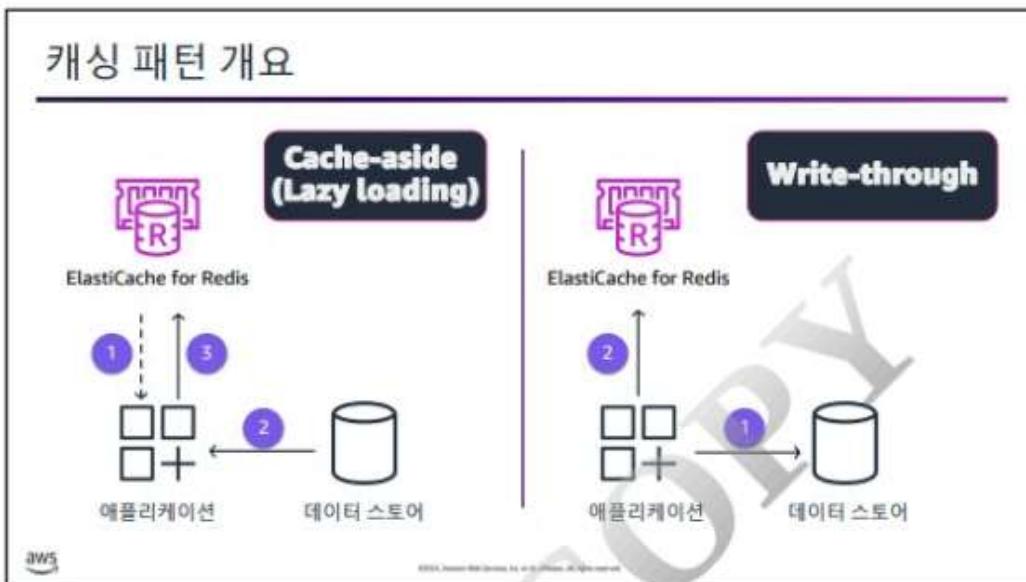
이 모듈에서 살펴볼 주제를 검토하십시오.



모듈 2에서는 플랫 및 이동 관리를 주로 Amazon DynamoDB에서 추적하고 저장하는 방법을 배웠습니다.

모듈 3에서는 Amazon DocumentDB(MongoDB 호환)를 사용하여 사용자 프로파일 데이터를 저장 및 관리하는 방법과 부분 이동 데이터를 저장하여 고객 이동 보고서를 실행하는 방법을 알아보았습니다.

이 모듈에서는 ElastiCache for Redis와 관련된 자전거 및 스쿠터 검색 워크로드를 다시 살펴봅니다. 관련 있는 활성 프로파일 데이터를 캐시하여 성능을 극대화하는 방법도 검토합니다. 마지막으로, 최종 워크로드 요구 사항을 충족하기 위해 네이티브 Redis 순위표를 집계된 이동 거리 데이터에 사용하는 방법을 알아봅니다.



이 모듈에서는 두 가지 캐싱 패턴을 살펴봅니다. 정의를 검토한 후 계속 진행하십시오.

Cache-aside(Lazy Loading)

Cache-aside 캐시는 사용 가능한 가장 일반적인 캐싱 전략입니다. 핵심 데이터 검색 논리는 다음과 같이 요약할 수 있습니다.

1. 애플리케이션은 데이터베이스에서 데이터를 읽어야 하는 경우 먼저 캐시부터 검사하여 데이터를 사용할 수 있는지 확인합니다.
2. 데이터를 사용할 수 있으면(캐시 히트), 캐시된 데이터가 반환되고 응답이 호출자에게 전달됩니다.
3. 데이터를 사용할 수 없으면(캐시 미스), 데이터베이스의 데이터를 쿼리합니다. 그 후 데이터베이스에서 검색된 데이터로 캐시가 채워지고, 데이터가 호출자에게 반환됩니다.

Write Through

Write Through 캐시는 캐시를 채우는 순서를 반대로 바꿉니다. 캐시 미스 후 캐시에 데이터를 지연 로드하는 대신 프라이머리 데이터베이스 업데이트 직후에 캐시가 선제적으로 업데이트됩니다. 핵심 데이터 검색 논리는 다음과 같이 요약할 수 있습니다.

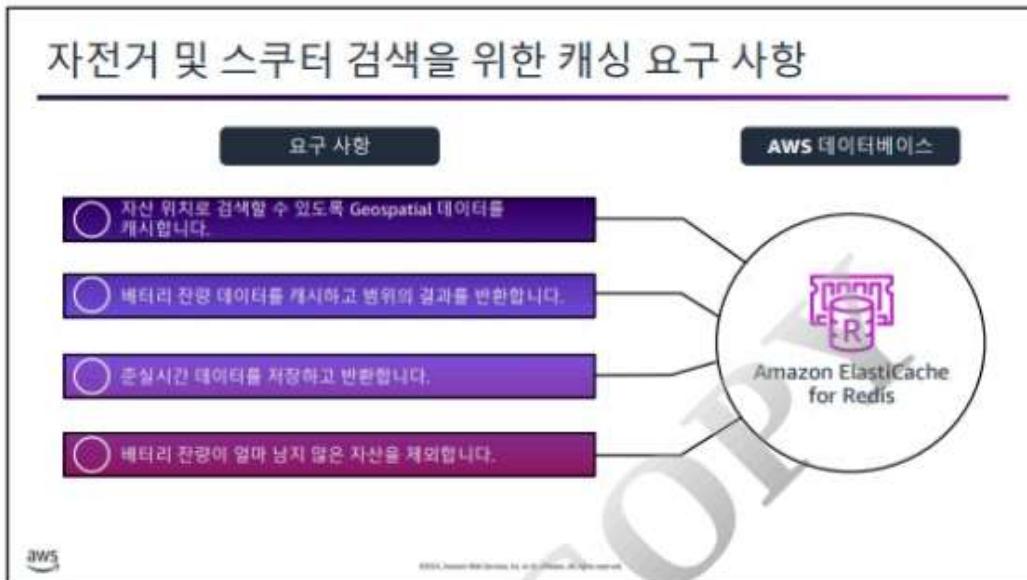
1. 애플리케이션, 배치 또는 백엔드 프로세스가 프라이머리 데이터베이스를 업데이트합니다.
2. 직후에 데이터도 캐시에서 업데이트됩니다.

Write Through 패턴은 거의 항상 Lazy Loading과 함께 구현됩니다. 데이터가 없거나 만료되어 애플리케이션에서 캐시 미스가 발생하면 캐시를 업데이트하기 위해 Lazy Loading 패턴이 수행됩니다.

Write Through 방식에는 몇 가지 장점이 있습니다. 캐시가 항상 프라이머리 데이터베이스로 업데이트되므로 캐시에서 데이터를 찾을 가능성이 훨씬 더 높습니다. 따라서 전반적인 애플리케이션 성능과 사용자 경험이 향상됩니다. 데이터베이스 읽기 횟수가 줄어들기 때문에 데이터베이스 성능이 향상됩니다.

Write Through 방식의 단점은 자주 요청되지 않는 데이터도 캐시에 쓰기 때문에 캐시가 더 커지고 비용이 더 많이 든다는 점입니다.

적절한 캐싱 전략이 되려면 데이터의 Write Through 및 Lazy Loading을 효과적으로 사용해야 합니다. 또한 데이터를 관련성이 높으면서도 간결하게 유지할 수 있도록 데이터의 만료를 적절하게 설정해야 합니다.



자전거 및 스쿠터 검색을 위한 캐싱 요구 사항을 검토합니다.

1. **자산 위치로 검색할 수 있도록 Geospatial 데이터를 캐시합니다.** Redis Geospatial 데이터 형식을 사용하여 위치 데이터를 관리합니다.
2. **배터리 잔량 데이터를 캐시하고 범위의 결과를 반환합니다.** 캐시에 배터리 잔량 데이터를 저장합니다. 이 데이터를 사용하여 선택할 수 있는 자전거 및 스쿠터 자산을 결정합니다.
3. **준실시간 데이터를 저장하고 반환합니다.** Amazon DynamoDB Streams는 일괄 처리되며, 최대 처리 대기 시간이 약간 있습니다(거의 실시간). 여기서는 실시간 데이터를 즉시 또는 직접 처리하는 대신 데이터가 거의 실시간으로 전달되기를 기대합니다.
4. **배터리 잔량이 얼마 남지 않은 자산을 제외합니다.** 배터리 잔량 데이터에 액세스하여 사용자가 배터리 잔량이 얼마 남지 않은 자전거나 스쿠터를 요청할 수 없도록 합니다. 그러면 좋은 고객 경험을 제공할 수 있습니다.

자전거 및 스쿠터 검색 액세스 패턴

- 위치 반경 내의 자산 결과를 가져옵니다.
- 배터리 잔량이 얼마 남지 않은 자산을 검색에서 제외합니다.
- 요청된 배터리 잔량을 기준으로 결과를 필터링합니다.



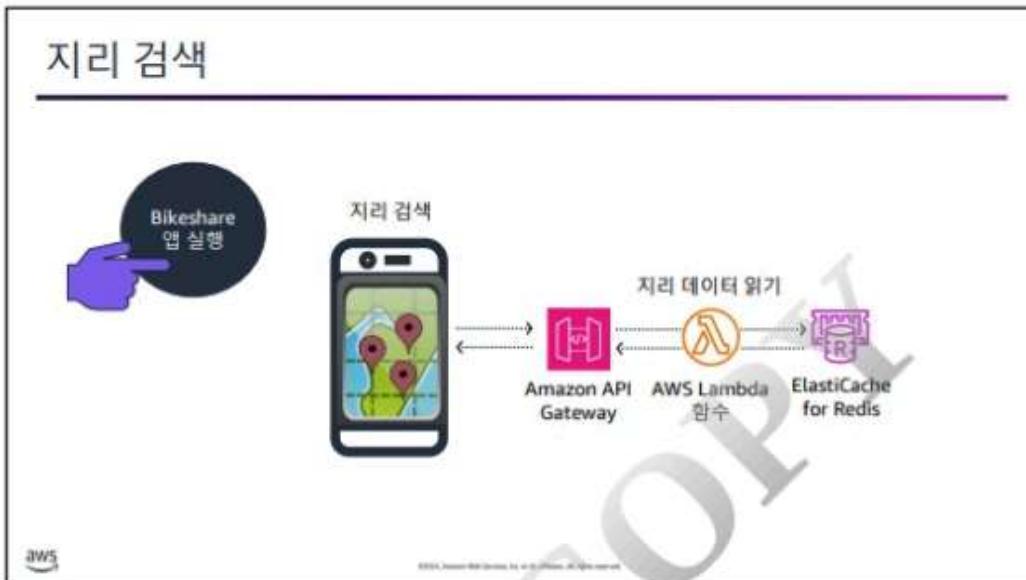
aws

© 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

모듈 2에서 살펴본 자전거 및 스쿠터 검색 워크로드의 액세스 패턴을 떠올려 보십시오. 모바일 앱은 사용자가 요청한 반경 내에 있는 자전거 및 스쿠터 자산을 가져와야 합니다. 그러면 사용자가 가까이 있는 자전거를 찾을 수 있습니다.

Amazon DynamoDB 자산 테이블에 **low battery**로 표시된 자산은 이러한 검색에서 제외해야 합니다. 자산이 배터리 부족을 보고하면 검색 알고리즘에서 해당 자산을 제외하도록 캐시를 업데이트해야 한다는 뜻입니다.

검색 기능의 일부로 사용자는 최소 배터리 비율을 요청할 수 있어야 합니다. 예를 들어 사용자가 스쿠터를 장거리 이동에 사용하려 할 수 있습니다. 디바이스에서 배터리가 40% 이상 남아 있는 스쿠터로 검색한 후 그 중에서 선택하기를 원할 수 있습니다. 이동 중에 배터리가 완전히 고갈되기를 원하는 사람은 아무도 없습니다.



자전거 및 스쿠터 검색 워크플로에서는 Amazon API Gateway를 사용하여 AWS Lambda 함수를 호출하고 ElastiCache for Redis에서 현재 자전거 및 스쿠터 위치 데이터를 읽어야 합니다.

모듈 2에서 자전거 및 스쿠터 위치 데이터를 ElastiCache for Redis로 스트리밍하는 전략을 수립했습니다.



Write Through 캐시는 캐시를 채우는 순서를 반대로 바꿉니다. 캐시 미스 후 캐시에 데이터를 자연 로드하는 대신 프라이머리 데이터베이스 업데이트 직후에 캐시가 선제적으로 업데이트됩니다.

핵심 데이터 검색 논리는 다음과 같이 요약할 수 있습니다.

1. 애플리케이션, 배치 또는 백엔드 프로세스가 프라이머리 데이터베이스를 업데이트합니다.
2. 직후에 데이터도 캐시에서 업데이트됩니다.

이 예에서는 자전거 검색 캐싱 전략을 보여줍니다. CreateUpdateTrip Lambda 함수는 이동이 종료되면 DynamoDB fleet 테이블의 자전거 위치를 업데이트합니다. DynamoDB 스트림은 캐시의 동일한 위도 및 경도 데이터를 업데이트하는 BikeEvents 함수를 호출합니다.

위치 데이터는 내구성을 위해 DynamoDB에 저장되지만, 지리 검색 작업에 사용할 수 있도록 캐시되어 ElastiCache for Redis에 사용됩니다.

캐싱 전략에 대한 자세한 내용은 [Redis를 사용하는 데이터베이스 캐싱 전략](#) 백서에서 '캐싱 패턴'의 Write Through

섹션(<https://docs.aws.amazon.com/whitepapers/latest/database-caching-strategies-using-redis/caching-patterns.html#write-through>)을 참조하십시오.



DO NOT COPY
ssleeeee1@gmail.com

Redis 데이터 구조 예시

String	"Hi there! My name is Amazon ElastiCache for Redis!"
Bitmap (String)	0011011100010001001101101
Bitfield (String)	{112345569}{2334}{2334}{655567}
Hash	{ A:"Amazon", W:"Web", S:"Services" }
List	[A -> B -> C -> D -> E]
Set	{ A, C, E, D, B }
Sorted set	{ A:1, B:2, C:3, D:4, E:5 }
Geospatial	{ A:[32.1,34.7], B:[51.5,0.12] }
JSON	{ {'Name':'Jane Doe', Email:'jane_doe@example.com'} }
HyperLogLog	01101110 00100010 01101101
Stream	... 1:msg1 ... 2:msg2 ... 3:msg3

다음은 Redis 데이터 구조의 예시입니다.

- **문자열:** 일반 텍스트, JSON, 사진, 비디오 등 모든 것을 포함할 수 있는 이진 안전 값입니다. 문자열은 이진 안전 값이므로 비트맵 및 비트필드로도 사용할 수 있습니다.
- **해시:** Row in Table 또는 Python Dictionary와 유사한 키/값 페어 컬렉션입니다.
- **목록:** 순서를 유지하는 이중 연결 문자열 목록입니다.
- **세트:** 순서가 없는 고유한 값 컬렉션입니다.
- **Sorted set:** 점수가 연결된 고유한 값 컬렉션입니다.
- **Geospatial:** 위도와 경도를 사용하여 지도에서 항목을 찾는 위치 인덱스 컬렉션입니다.
- **JSON:** Redis 데이터베이스에서 JSON 값을 저장, 업데이트 및 검색할 수 있습니다.
- **HyperLogLog:** 메모리를 최소한으로 사용하면서 항목 컬렉션의 카디널리티를 유지하는 확률적 데이터 구조입니다.
- **스트림:** 추가 누적만 가능한 로그 데이터 구조로, 시간 기반 데이터 컬렉션으로 사용할 수 있습니다



다음으로, 강사는 각 데이터 구조와 키 체계에 대해 설명합니다.

요약 1: 데이터 구조 및 키 체계

워크로드	데이터 형식	키 체계	설명
자전거 검색	Sorted set	assetbattery:<city> battery asset_id	배터리 잔량을 기준으로 정렬된 자산 저장
	Geospatial	assetgeo:<city> long lat asset_id	각 자산의 경도와 위도 저장
프로파일 캐시			
순위표			



© 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

자전거 검색 워크로드에 사용되는 Redis 데이터 형식의 요약 내용을 검토합니다.

자전거 검색

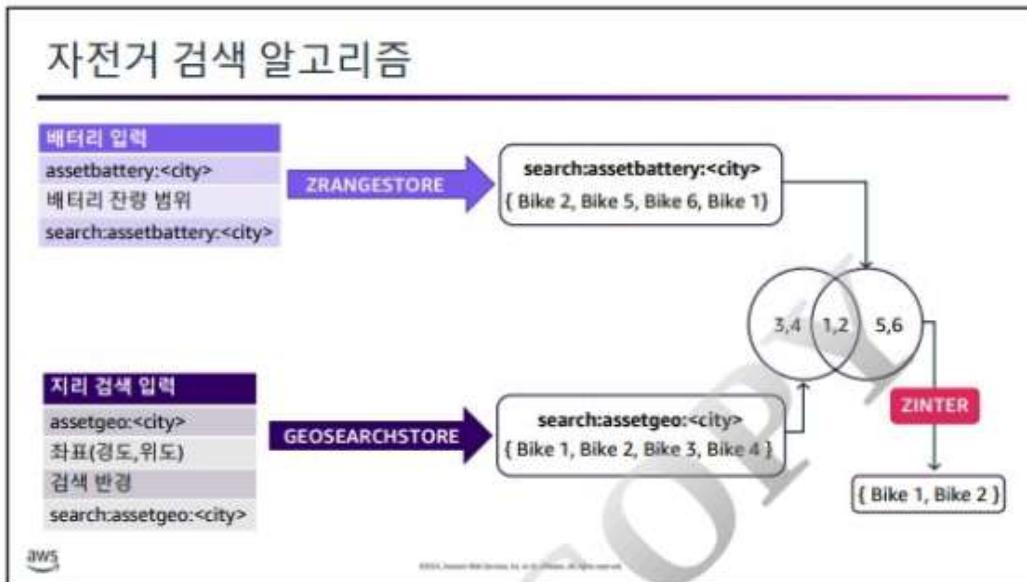
- Sorted set 데이터 형식은 배터리 잔량을 기준으로 정렬된 자산의 ID를 저장합니다.
- Sorted set 키 체계는 assetbattery:<city> battery asset_id입니다.
- Geospatial 데이터 형식은 각 자산의 경도(long)와 위도(lat)를 저장하고 자산 ID를 레이블로 지정합니다.
- Geospatial 키 체계는 assetgeo:<city> long lat asset_id입니다.

프로파일 캐시

이 모듈의 후반부에서 자세히 알아봅니다.

순위표

이 모듈의 후반부에서 자세히 알아봅니다.



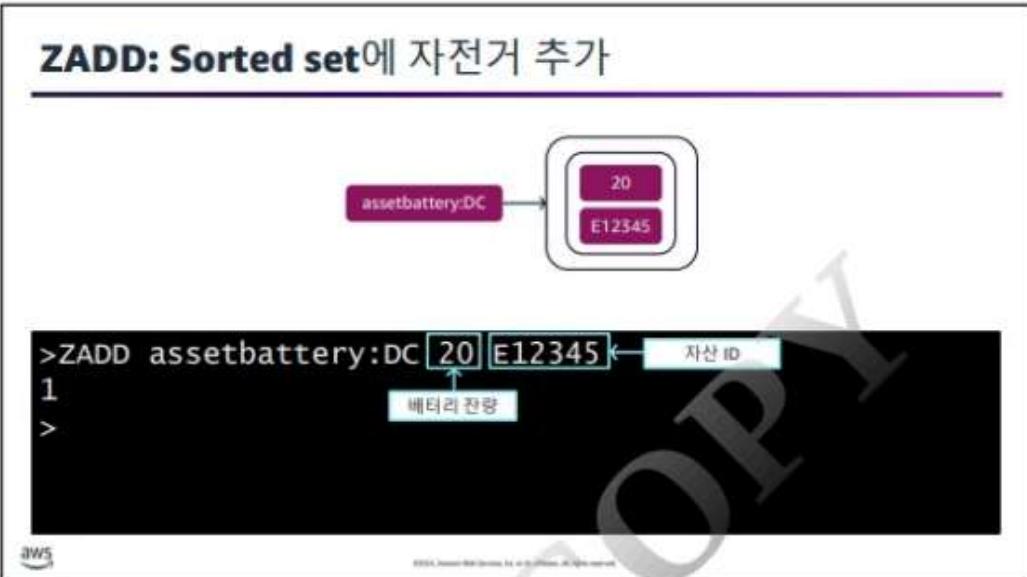
강사가 실습을 시작하기 전에, 잠시 시간을 내어 ElastiCache for Redis에서 사용되는 자전거 검색 알고리즘에 대해 알아보십시오.

ZRANGESTORE 명령은 Sorted set의 지정된 요소 범위를 반환하고 결과를 지정된 대상에 저장합니다. 여기서는 배터리 잔량이 (기본값으로 또는 검색 입력을 통해) 지정된 범위 내에 있는 자전거를 반환하는 데 사용됩니다. 단순화된 예시에서 ZRANGESTORE는 자전거 1, 2, 5, 6을 반환합니다.

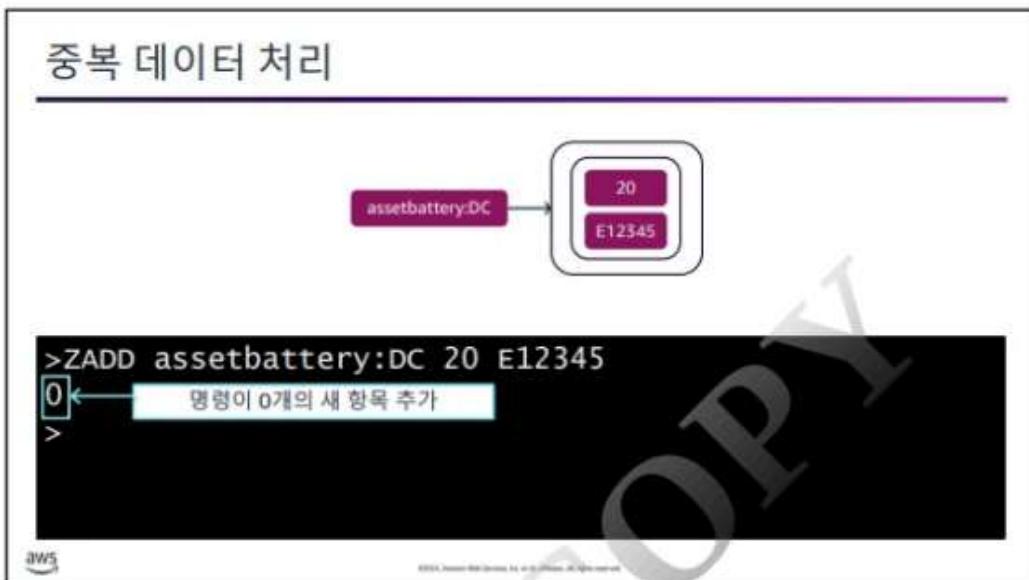
GEOSEARCHSTORE 명령은 **Geospatial** 정보로 채워진 Sorted set의 멤버를 반환하고 결과를 대상 키와 함께 저장합니다. 이 구현에서 GEOSEARCHSTORE 명령은 4개 입력을 받습니다. assetgeo:<city> 키는 지리 검색을 실행할 캐시 위치를 제공합니다. 이 위치에 개별 자전거 및 스쿠터 위치 데이터가 캐시됩니다. 두 번째 입력은 모바일 앱에서 제공하는 사용자의 현재 위치입니다. 검색 반경이 지정됩니다(기본적으로 또는 지정된 사용자 입력 파라미터를 통해). 마지막으로, 결과를 캐시하기 위한 키가 제공됩니다. 여기서 지리 검색 결과는 search:assetgeo:<city> 대상 키에 캐시됩니다. 예제의 GEOSEARCHSTORE는 자전거 1, 2, 3, 4를 반환합니다.

이처럼 저장된 두 세트에서, 알고리즘은 ZINTER 명령을 사용하여 두 세트 간의 공통 요소를 찾습니다. 예제의 원형 벤다이어그램에서 두 세트의 교차점은 자전거 1 및 자전거 2입니다. 이 반환된 ZINTER 값은 모바일 앱에서 사용자에게 관련 자전거 및 스쿠터 세트를 표시하는 데 사용됩니다.





강사는 ZADD 명령을 사용하여 자전거를 Sorted set에 추가하는 방법을 시연합니다. 요소에는 배터리 잔량과 자산 ID가 포함됩니다. 이 예제에서 워싱턴 DC의 E12345 자산은 배터리가 20% 남아 있습니다.



동일한 자산을 동일한 Sorted set에 두 번 추가해도 데이터가 중복되지 않습니다.
동일한 명령의 출력은 0을 출력합니다. Sorted set에 새 요소가 추가되지
않습니다.

ZADD: Sorted set에 여러 자산 추가

```
>ZADD assetbattery:DC 30 S54987 40 S29654 60 S70777  
70 E76890 90 E49167  
5  
>
```

하나의 명령을 사용하여 여러 자산을 Sorted set에 추가할 수 있습니다. 여기서는 Sorted set에 다른 자산 배터리 잔량 5개를 삽입합니다.

배터리 잔량으로 정렬

배터리가 50% 넘게 남은 자전거 및 스쿠터 찾기

오름차순

```
>ZRANGE assetbattery:DC 50
+inf BYSCORE WITHSCORES
1) "S70777"
2) "60"
3) "E76890"
4) "70"
5) "E49167"
6) "90"
>
```

내림차순

```
>ZRANGE assetbattery:DC +inf
50 BYSCORE REV WITHSCORES
1) "E49167"
2) "90"
3) "E76890"
4) "70"
5) "S70777"
6) "60"
>
```

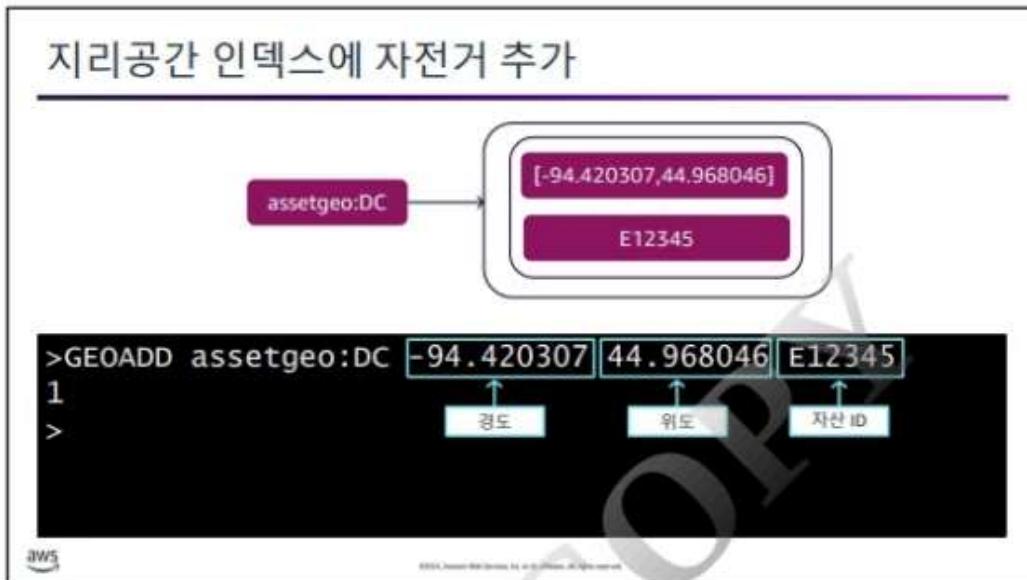


© 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Sorted set를 사용하면 오름차순 또는 내림차순으로 정렬할 수 있습니다. 또한 결과를 값으로 필터링할 수 있습니다. 여기서는 배터리가 50% 넘게 남은 자전거와 스쿠터만 결과에 포함됩니다.



DO NOT COPY
ssleeeee1@gmail.com



한 자전거의 지리적 위치가 `assetgeo:DC`라는 Geospatial 인덱스에 추가되는 예제입니다. Geospatial 데이터는 단일 값으로 저장됩니다. 간단한 설명을 위해 디아어그램에서 경도와 위도 좌표로 표시했습니다. 실제 Geospatial 인덱스에서는 데이터가 경도 및 위도 값으로 추가되며, 해당 값이 함께 해시되어 Geospatial 데이터가 Sorted set에 삽입됩니다.

명령 하나를 사용하여 여러 자산 추가

The diagram illustrates a sorted set named 'assetgeo:DC'. It contains four members, each represented by a rounded rectangle divided into two horizontal sections. The top section shows a coordinate pair, and the bottom section shows a unique identifier. The members are:

- [-94.420307,44.968046] E12345
- [-94.44786,44.92057] S70777
- [-94.43,44.43328] S54987
- [-94.43,44.53328] S29654

A command line interface shows the execution of the GEOADD command:

```
>GEOADD assetgeo:DC -94.44786 44.92057 E70777 -94.43  
44.43328 S54987 -94.43 44.53328 S29654  
3  
>
```

Geospatial 데이터는 Sorted set 데이터와 비슷하게 처리됩니다. 여기서 GEOADD 명령은 ZADD 명령과 마찬가지로 여러 자산을 한꺼번에 추가합니다.

GEOSEARCH: 반경별로 주변 자산 찾기

```
>GEOSEARCH assetgeo:DC FROMLONLAT -94.42 44.968 BYRADIUS 50 mi WITHDIST
WITHCOORD ASC
1) 1) "E12345" ← 자산 ID
   2) "0.0152" ← -94.42 44.968 지점과 떨어진 거리(마일)
3) 1) "-94.420307" ← 경도
   2) "44.968046" ← 위도
2) 1) "S70777"
   2) "3.5500"
3) 1) "-94.44786"
   2) "44.92057"
3) 1) "S29654"
   2) "30.0488"
3) 1) "-94.43"
   2) "44.53328"
...
>
```

AWS

이 예제에서 GEOSEARCH 명령은 좌표(-94.42,44.968)로부터 50마일 이내에 있는 세트의 Geospatial 데이터를 수집합니다. BYRADIUS 파라미터는 결과를 반경으로 제한합니다.

결과에는 각 일치 항목이 거리 오름차순으로 나열됩니다. 제공되는 데이터는 다음과 같습니다.

1. 자산 ID
2. 입력한 좌표로부터 거리(마일)
3. 일치하는 자산의 경도 및 위도(순서대로)

이 예제에서 GEOSEARCH 명령은 E12345, S70777, S29654의 3개 자산을 반환합니다.

GEOSEARCH: 영역별로 주변 자산 찾기

```
>GEOSEARCH assetgeo:DC FROMLONLAT -94.42 44.968 BYBOX 40 50 mi WITHDIST  
WITHCOORD ASC  
1) 1) "E12345"  
2) "0.0152"  
3) 1) "-94.420307"  
2) "44.968046"  
2) 1) "S70777"  
2) "3.5500"  
3) 1) "-94.44786"  
2) "44.92057"  
>
```

© 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

BYBOX 파라미터를 사용하여 원형 반경이 아닌 직사각형 경계 상자로 정의된 영역 내에서 지리적 위치를 검색할 수도 있습니다.

반경을 매칭하고 결과 저장

```
>GEOSEARCHSTORE search:assetgeo:-94.4244.968 assetgeo:DC
FROMMLONLAT -94.42 44.968 BYRADIUS 50 mi STOREDIST
3
>
> ZRANGE search:assetgeo:-94.4244.968 -inf +inf BYSCORE
WITHSCORES
1) "E12345"
2) "0.015226359949341149"
3) "S70777"
4) "3.5500481525586021"
5) "S29654"
6) "30.048788052024847"
>
```

© 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

GEOSEARCHSTORE 명령을 통해 지정된 키(여기서는 search:assetgeo:-94.4244.968 사용)를 사용하여 지리 검색 결과를 저장합니다. 터미널에서 ZRANGE 명령은 저장된 출력을 표시하고 거리 결과를 오름차순으로 정렬합니다.

배터리 잔량을 매칭하고 결과 저장

```
>ZRANGESTORE search:assetbattery:DC:50 assetbattery:DC 50
+inf BYSCORE
3
>
>ZRANGE search:assetbattery:DC:50 -inf +inf BYSCORE
WITHSCORES
1) "S70777"
2) "60"
3) "E76890"
4) "70"
5) "E49167"
6) "95"
>
```



© 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

ZRANGESTORE 명령은 search:assetbattery:DC:50 키를 사용하여 배터리 값이 50%보다 큰 자산을 저장합니다.

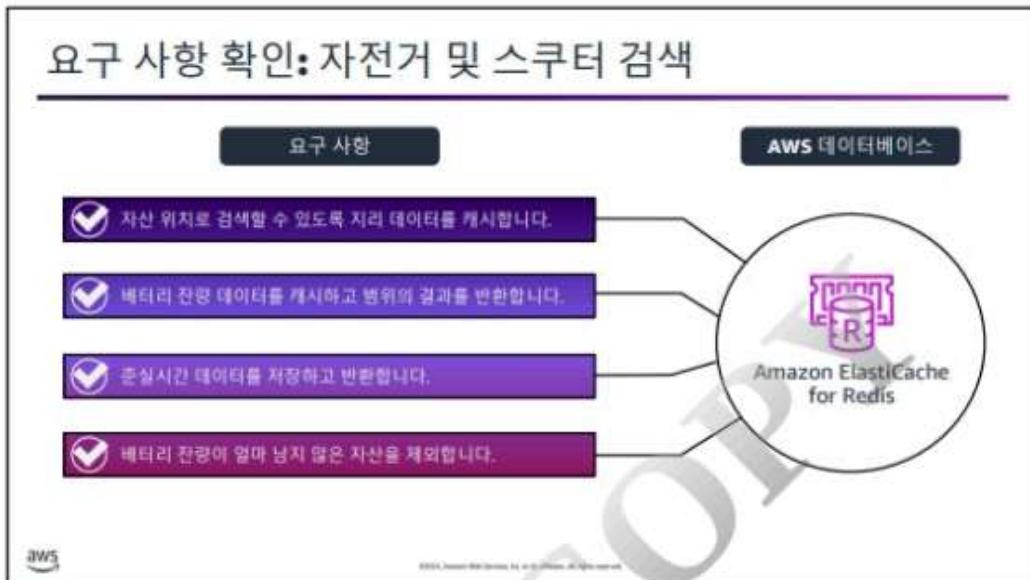
ZINTER: 교차하는 결과 가져오기

```
>ZINTER 2 search:assetgeo:-94.4244.968
search:assetbattery:DC:50 WEIGHTS 1 0 WITHSCORES
1
>
>ZRANGE assetbattery:DC +inf 50 BYSCORE REV WITHSCORES
1) "s70777"
2) "3.5500481525586021"
>
```



© 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

자전거 검색 알고리즘을 기억해 보십시오. ZINTER 명령을 사용하여 최종 결과를 검색합니다. 그러면 저장된 데이터 세트 2개가 교차하고 결과가 출력됩니다. 이 예제에서는 s70777 자산만 남습니다.

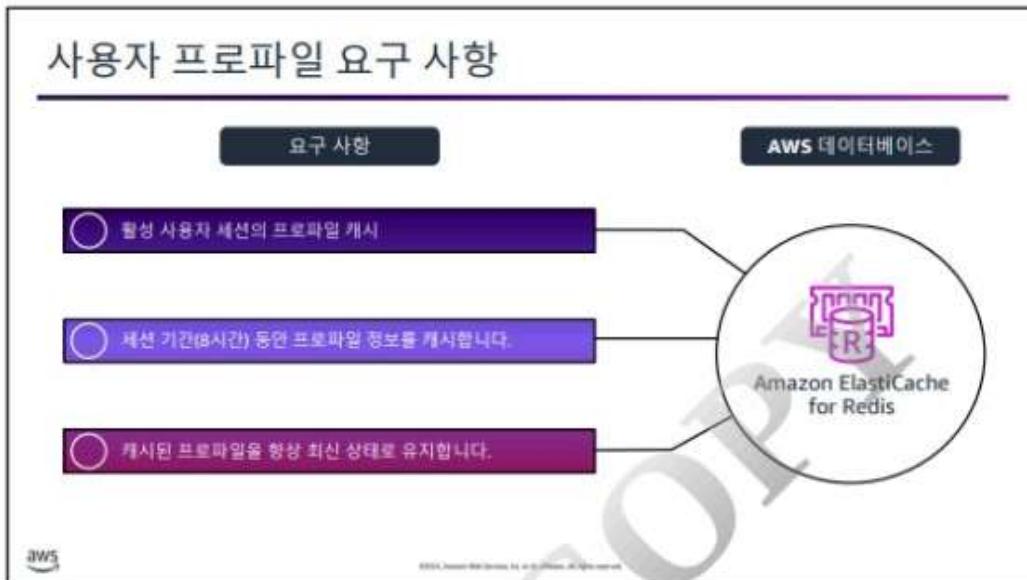


ElastiCache for Redis를 사용하여 이 섹션 시작 부분의 요구 사항을 충족하는 방법을 배웠습니다.



다음으로, 사용자 프로파일 스토리지에 대해 배운 내용을 다시 생각해 보십시오. 사용자 프로파일 데이터는 Amazon DocumentDB에 저장되고, 활성 프로파일은 ElastiCache for Redis에 캐시됩니다.

다음 4개의 슬라이드는 핵심 요구 사항과 액세스 패턴을 기억하는 데 도움이 됩니다.

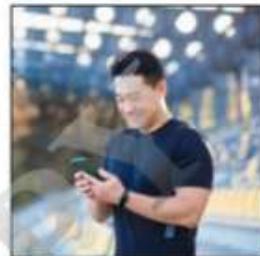


제품 관리자가 모바일 앱에 대한 세 가지 캐싱 요구 사항을 전달합니다.

- 활성 사용자 세션의 프로파일 캐시 오래된 비활성 프로파일을 캐시에 추가하지 않습니다. 그러면 캐시를 사용할 때 가성비가 향상됩니다.
- 세션 기간 동안 프로파일 정보를 캐시합니다. 캐시된 프로파일 데이터는 모바일 디바이스에 8시간 세션 동안 저장됩니다. 이 기간이 지나면 앱이 캐시에서 프로파일 데이터를 다시 검색해야 합니다. 현재 구현에서는 이 작업이 컬렉션 수준이 아닌 키 수준에서 처리됩니다.
- 캐시된 프로파일을 최신 상태로 유지합니다. 추가 프로모션이나 쿠폰과 같은 백엔드 업데이트가 발생하면 해당 변경 내용이 거의 실시간으로 캐시에 반영되어야 합니다.

사용자 프로파일 액세스 패턴

- 사용자 프로파일을 가져옵니다.
- 활성 프로파일을 캐시합니다(아직 캐시되지 않은 경우).
- 프로파일 내용을 업데이트합니다.

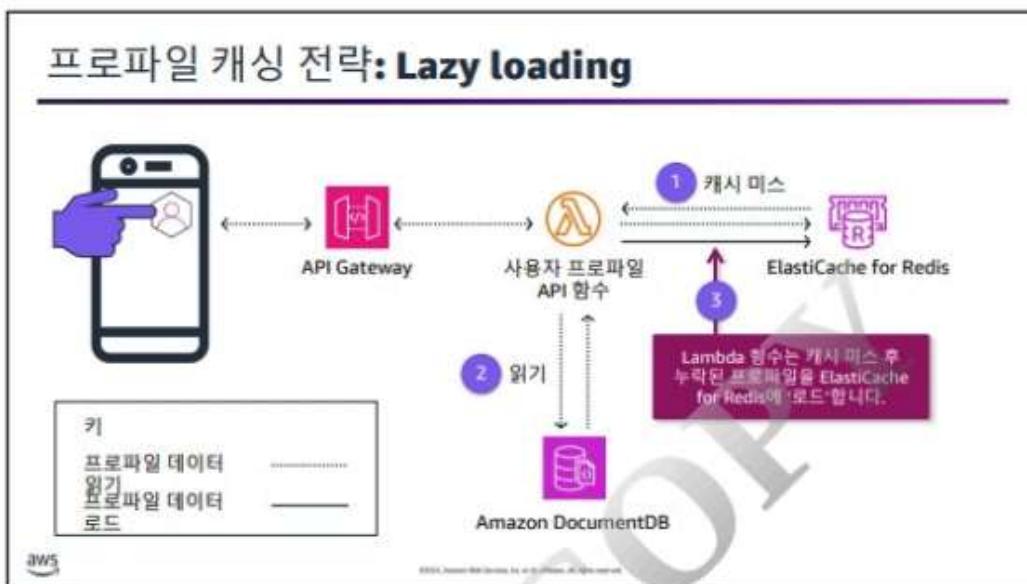


AWS

© 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

사용자 프로파일 관리는 AnyCompany BikeShare 앱의 핵심 워크로드 중 하나입니다. 다음은 사용자 경험과 비즈니스에 직접적인 영향을 미치는 중요한 기능입니다.

- 프로파일을 신속하게 검색하여 사용자의 디바이스에 표시해야 합니다.
- 사용자는 결제 방법, 연락처 정보, 계정 기본 설정 및 멤버십을 업데이트해야 합니다. 이 데이터는 여러 디바이스에서 액세스할 수 있어야 하며 변경될 때마다 캐시와 프라이머리 데이터베이스에서 업데이트되어야 합니다.

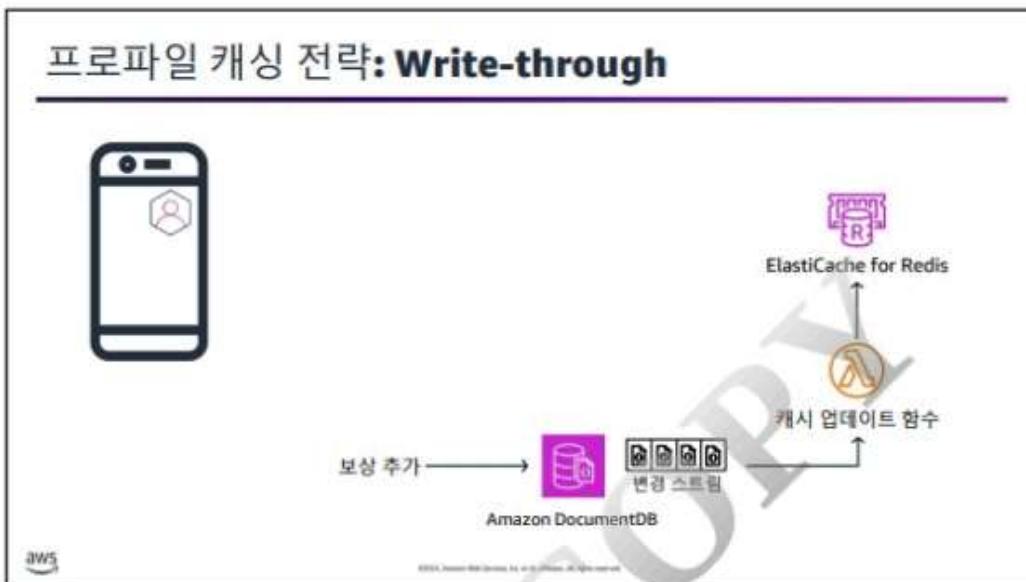


Lazy Loading 전략은 프로파일 읽기에 사용됩니다.

사용자가 앱에 로그인하면 최신 프로파일 데이터를 캐시에서 디바이스로 다시 가져오려는 API 요청이 수행됩니다. 프로파일이 캐시에 있으면 Lambda 함수는 ElastiCache for Redis에서 프로파일 데이터를 읽습니다. 이것을 **캐시 히트**이라고 합니다. 반대 작업인 **캐시 미스**는 다음과 같이 처리됩니다.

1. ElastiCache for Redis가 null 응답을 반환합니다.
2. Lambda 함수가 DocumentDB에서 프로파일을 읽고 해당 데이터를 캐시에 쓰는 동시에 프로파일을 다시 모바일 앱으로 보냅니다. 이것을 **캐시 미스**라고 합니다. Lambda 함수는 DocumentDB 컬렉션에서 읽습니다.
3. Lambda 함수는 반환된 JSON 프로파일 데이터를 ElastiCache for Redis에 쓰거나 로드합니다.

프로파일이 새 청구 주소와 같은 새 데이터로 업데이트되면 해당 데이터를 DocumentDB와 ElastiCache for Redis에 모두 써야 합니다.



Write Through 전략은 프로모션이나 쿠폰 추가와 같은 백엔드 프로파일 업데이트에 사용됩니다. 이러한 작업이 발생하면 DocumentDB의 프로파일이 업데이트된 다음, 변경 스트림을 사용하여 동일한 업데이트가 캐시에 적용됩니다. 프로파일이 이미 캐시에 로드된 경우 캐시의 프로파일 데이터는 항상 최신이어야 합니다.

프로모션 또는 보상 업데이트와 같은 백엔드 작업이 발생하면 해당 변경 내용이 DocumentDB 컬렉션에 적용됩니다. 변경 스트림은 ElastiCache for Redis의 프로파일을 업데이트하는 캐시 업데이트 Lambda 함수를 호출하여 변경 내용을 주적하도록 활성화됩니다.



강사가 사용자 프로파일 캐싱 워크로드에 사용되는 데이터 구조와 키 체계를 검토합니다.

JSON 데이터 형식의 이점

- 기본적으로 저장된 JSON 문서를 사용하며, 사용자 프로파일 데이터를 프런트엔드 및 프라이머리 데이터베이스에서 관리합니다.
- 사용자 지정 코드 없이 JSON을 저장하고 업데이트하여 데이터를 직렬화 및 역직렬화합니다.
- 전체 객체를 조작할 필요 없이 JSON 문서의 특정 부분을 효율적으로 검색하고 업데이트합니다.



© 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

원래 DocumentDB 사용자 프로파일 컬렉션의 JSON 데이터 형식을 유지합니다. 네이티브 DocumentDB 및 변경 스트림 데이터는 캐시를 업데이트할 수 있습니다. 애플리케이션은 이미 JSON 데이터를 작업하는 방법을 알고 있습니다. ElastiCache for Redis에서 데이터를 작업할 때 데이터를 직렬화 및 역직렬화할 필요가 없습니다. 중첩 필드와 같은 기능을 사용하면 전체 개체를 작업할 필요 없이 특정 데이터를 효율적으로 업데이트하고 검색할 수 있습니다.

요약 2: 데이터 구조 및 키 체계

워크로드	데이터 형식	키 체계	설명
자전거 검색	Sorted set	assetbattery:<city> battery asset_id	배터리 잔량을 기준으로 정렬된 자산 저장
	Geospatial	assetgeo:<city> lat long asset_id	각 자산의 위도와 경도 저장
프로파일 캐시	JSON	userprofile:<userid> '{...}'	사용자 프로파일 데이터 전체를 JSON으로 저장
순위표			



© 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

프로파일 캐시 워크로드에 사용되는 Redis 데이터 형식의 요약 내용을 검토합니다.

자전거 검색

- Sorted set 데이터 형식은 배터리 잔량을 기준으로 정렬된 자산의 ID를 저장합니다.
- Sorted set 키 체계는 assetbattery:<city> battery asset_id입니다.
- Geospatial 데이터 형식은 각 자산의 경도(long)와 위도(lat)를 저장하고 자산 ID를 레이블로 지정합니다.
- Geospatial 키 체계는 assetgeo:<city> long lat asset_id입니다.

프로파일 캐시

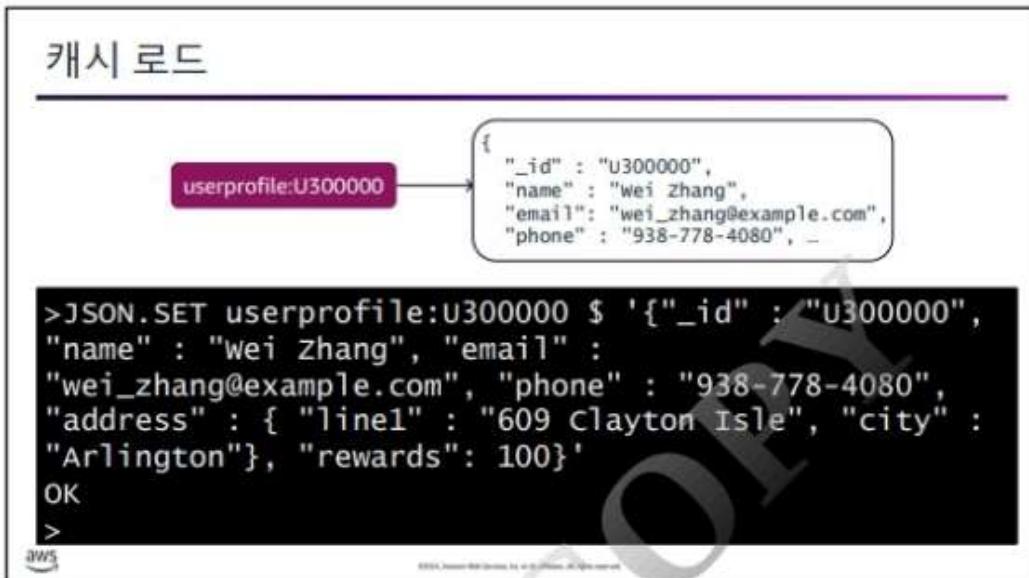
- JSON 데이터 형식은 프로파일 데이터를 변환하지 않고 저장합니다. 여기서 JSON 파라미터는 '{...}'로 표시되며, 줄임표(...)는 포함된 JSON 데이터를 나타냅니다.
- JSON 키 체계는 userprofile:<userid> '{...}'입니다.

순위표

이 모듈의 후반부에서 자세히 알아봅니다.



JSON 명령 및 JSON 경로에 대한 자세한 내용은 [Amazon ElastiCache for Redis 사용 설명서의 'Redis JSON 데이터 형식 개요'](https://docs.aws.amazon.com/AmazonElastiCache/latest/red-ug/json-document-overview.html)(<https://docs.aws.amazon.com/AmazonElastiCache/latest/red-ug/json-document-overview.html>)를 참조하십시오.



먼저 JSON.SET Redis 명령을 사용하여 캐시를 로드합니다. 여기서는 사용자 ID U300000의 단일 사용자 프로파일을 로드합니다.

프로파일 기본 설정 업데이트

```
>JSON.SET userprofile:u300000 $.preferences '{  
  "notifications" : { "frequency" : "WEEKLY", "mobile" : {  
    "sms" : "false", "app" : "false" }, "email" : "true" } }'  
OK  
>
```

© 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

사용자의 기본 설정을 업데이트하려면 캐시와 Amazon DocumentDB userprofile 컬렉션 모두에서 기본 설정을 업데이트해야 합니다. 여기서는 JSON 문서의 일부만 업데이트하는 데 사용되는 Redis 명령 JSON.SET의 예를 볼 수 있습니다. 이 예에서는 기존 프로파일에 새 기본 설정을 삽입합니다.

프로파일 데이터 검색

```
>JSON.GET userprofile:U300000
'$.payment_methods[?(@.nickname=="anyDigi23513")]'  
[{"nickname":"anyDigi23513","vendor":"anyDigi2","e  
mail":"digi.Fg93tJ32sT87119example.com","type":  
"digital"}]  
>
```

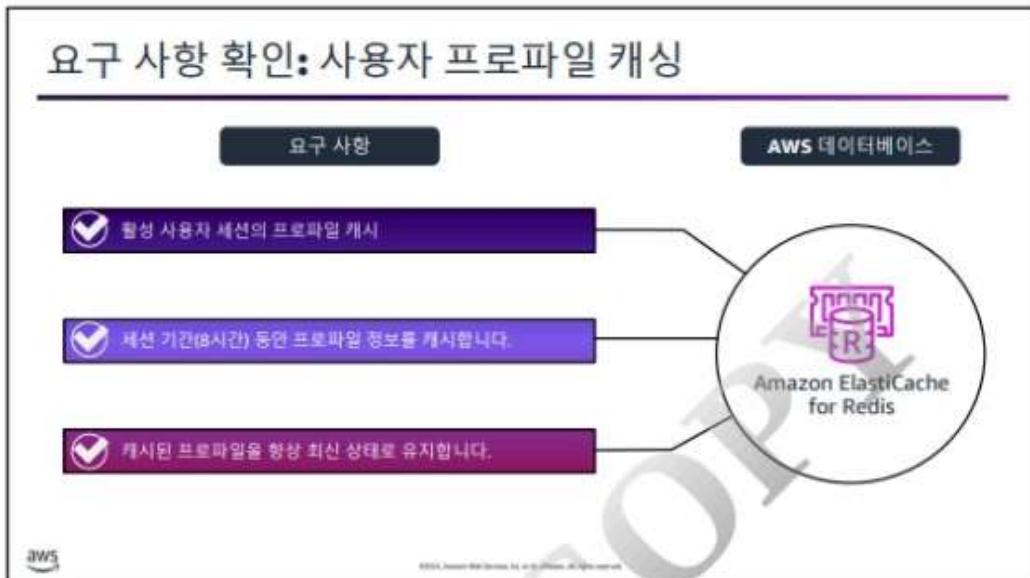


© 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

JSON.GET 명령을 사용하여 캐시의 프로파일 데이터를 검색합니다. 여기서는 JSON.GET을 사용하여 사용자 프로파일 U300000에서 결제 방법 정보를 검색합니다.



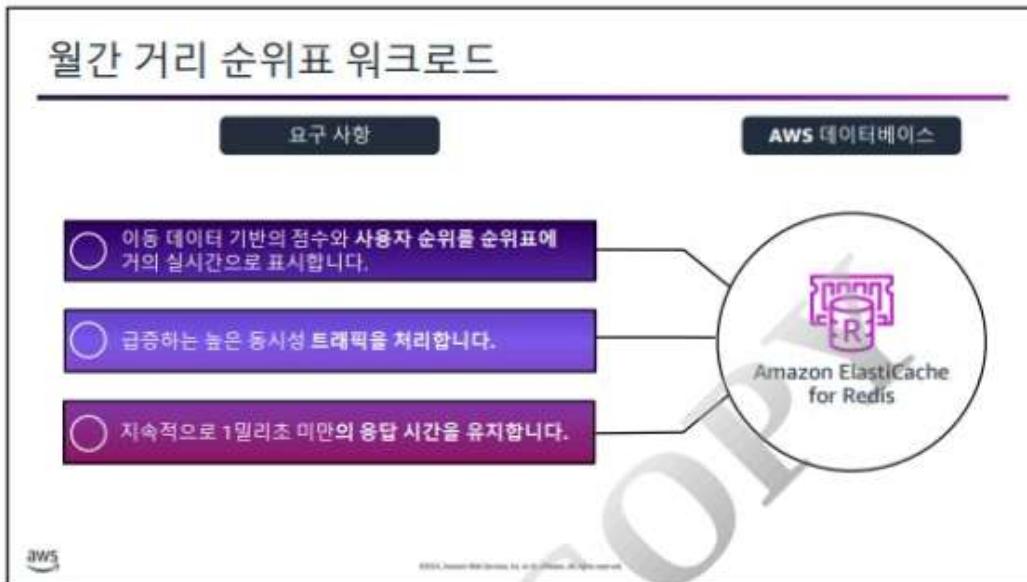
EXPIRE Redis 명령을 사용하여 세션 시간 제한 요구 사항을 충족합니다. 사용자 프로파일 데이터의 만료 시간을 초 단위로 설정합니다. 데이터가 만료되면 캐시는 Amazon DocumentDB에서 데이터를 다시 로드합니다.



이제 여러분은 ElastiCache for Redis를 사용하여 이 섹션을 시작할 때 언급한 요구 사항을 충족하는 방법을 알고 있습니다.



다음으로, 모듈 1에서 배운 순위표 워크로드를 구현합니다. 월간 거리 순위표의 요구 사항 및 액세스 패턴을 검토하십시오.



AnyCompany BikeShare는 사용자의 참여와 관심을 유도하려는 큰 계획을 갖고 있습니다. 순위표의 순위에 따라 사용자에게 보상을 제공할 예정입니다.

매력적이고 몰입도 높은 경험을 제공하기 위해 AnyCompany BikeShare는 각 사용자의 월간 누적 거리를 추적하는 준실시간 순위표를 원합니다. 다음 요구 사항을 고려하십시오.

1. **이동 데이터 기반의 점수와 사용자 순위를 순위표에 거의 실시간으로 표시합니다.** 월간 거리 순위표에는 거리 상위 사용자가 표시되고 사용자 순위가 제공됩니다.
2. **급증하는 높은 동시성 트래픽을 처리합니다.** 트래픽 패턴을 어떻게 측정해야 하는지 확실하지 않습니다. 솔루션은 급증하는 워크로드를 지원해야 합니다.
3. **지속적으로 1밀리초 미만의 응답 시간을 유지합니다.** AnyCompany BikeShare는 성능에 중점을 둡니다. 개발 팀은 순위표를 읽을 때 지연이 발생하는 일이 없기를 원합니다.

순위표 액세스 패턴

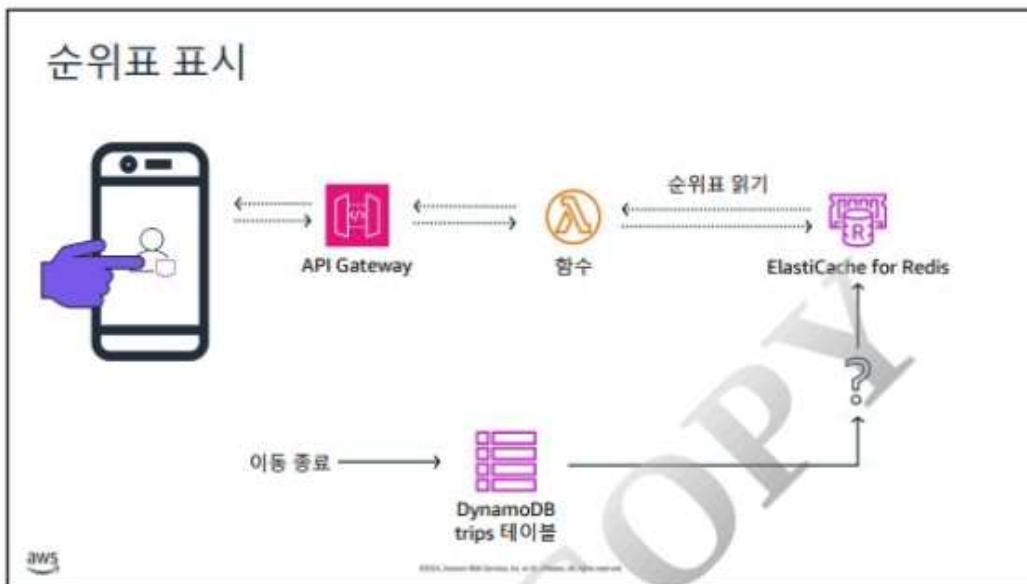
- 사용자의 이동 거리를 추가합니다.
- 이동 거리 점수 상위 5명을 가져옵니다.
- 사용자의 순위를 가져옵니다.



aws

© 2025, Amazon Web Services, Inc. or its affiliates. All rights reserved.

모바일 앱에서 사용자는 해당 도시의 상위 5명과 점수를 볼 수 있습니다. 또한 모든 사용자는 특정 시점에 순위표에서 자신의 순위를 알 수 있어야 합니다.



사용자가 월간 순위표를 확인하면 API 요청이 API Gateway로 전송되어 Lambda 함수를 호출합니다. 이 함수의 역할은 ElastiCache for Redis에 저장된 순위표 데이터를 읽고 그 결과를 모바일 앱에 반환하는 것입니다.

캐시를 이동 데이터로 업데이트하는 방법을 고민하십시오. 이동 데이터의 기본 스토리지는 Amazon DynamoDB trips 테이블에 있습니다.

거리 데이터



- 각 이동은 해당 이동에서 움직인 거리를 마일 단위로 기록합니다.
- 거의 실시간으로 업데이트되는 순위표를 구축해야 합니다.
- 이동 데이터를 지속적으로 저장해야 합니다.

aws

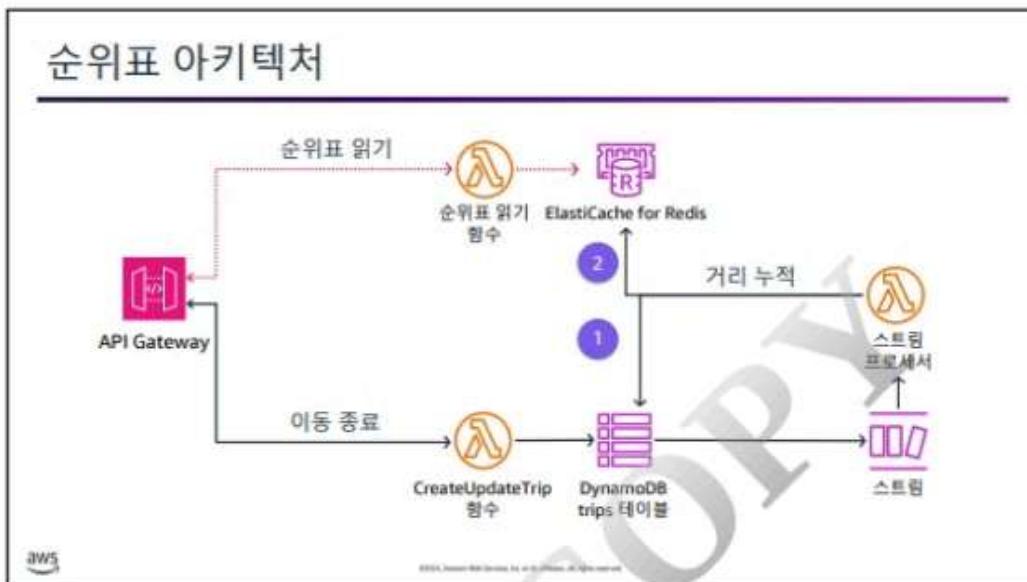
© 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

각 사용자의 월간 누적 이동 거리를 업데이트하는 전략을 선택해야 합니다.

ElastiCache for Redis 클러스터에서 이 데이터를 업데이트해야 할 뿐 아니라
동일한 데이터를 안정적으로 저장해야 합니다.



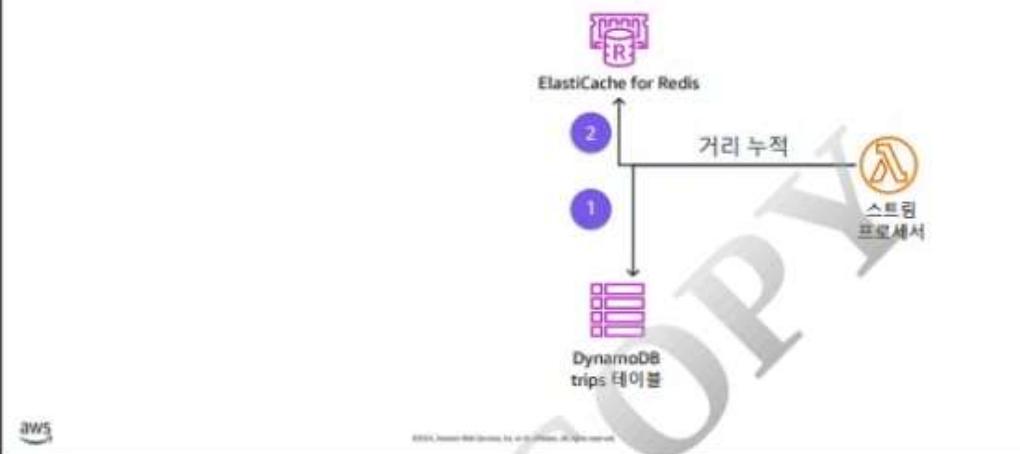
다음으로, 순위표를 구현하는 방법과 워크로드 요구 사항을 달성하기 위해 사용할 캐싱 전략을 살펴보겠습니다.



다음은 순위표 아키텍처를 요약한 내용입니다. 모든 순위표 읽기는 Amazon API Gateway를 거쳐 순위표 읽기 Lambda 함수를 호출합니다. 순위표 상태는 ElastiCache for Redis에서 검색되어 요청자의 모바일 앱으로 반환됩니다.

이동 종료 작업에서 Lambda 함수는 이동 거리를 포함한 최신 사용 데이터로 trips 테이블을 업데이트합니다. DynamoDB 스트림은 해당 사용자의 새 거리 값을 감지하고, 사용자 및 월별로 저장되는 trips 테이블의 총 이동 거리를 업데이트합니다. Lambda 함수는 ElastiCache for Redis에서 동일한 값을 업데이트합니다. 요구 사항 중 하나는 거의 실시간으로 점수를 표시하는 것입니다. 이 아키텍처에서는 추가 이동 정보를 trips 테이블과 ElastiCache for Redis 클러스터 모두에 반영합니다.

순위표 캐싱 전략: Write-through



이 다이어그램은 Write Through 전략을 구현하는 순위표 아키텍처의 일부를 보여줍니다. 먼저, 월간 이동 거리가 DynamoDB trips 테이블에 업데이트됩니다. 그 후 총 거리가 해당 사용자의 ElastiCache for Redis에 기록되고, 거의 실시간으로 순위표가 업데이트됩니다.



강사가 이 사용 사례에 가장 적합한 데이터 구조에 대해 설명합니다. 다음 예에서 어떤 키 체계를 사용하는지 강사와 함께 알아보십시오.

요약 3: 데이터 구조 및 키 체계

워크로드	데이터 형식	키 체계	설명
자전거 검색	Sorted set	assetbattery:<city> battery asset_id	배터리 잔량을 기준으로 정렬된 자산 저장
	Geospatial	assetgeo:<city> lat long asset_id	각 자산의 위도와 경도 저장
프로파일 캐시	JSON	userprofile:<userid> '{...}'	사용자 프로파일 데이터 전체를 JSON으로 저장
순위표	Sorted set	lb:dl:<city>:<month> <distance> <userid>	거의 실시간으로 월간 거리 상위 사용자 저장



© 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

프로파일 캐시 워크로드에 사용되는 Redis 데이터 형식의 요약 내용을 검토합니다.

자전거 검색

- Sorted set 데이터 형식은 배터리 잔량을 기준으로 정렬된 자산의 ID를 저장합니다.
- Sorted set 키 체계는 assetbattery:<city> battery asset_id입니다.
- Geospatial 데이터 형식은 각 자산의 경도(long)와 위도(lat)를 저장하고 자산 ID를 레이블로 지정합니다.
- Geospatial 키 체계는 assetgeo:<city> long lat asset_id입니다.

프로파일 캐시

- JSON 데이터 형식은 프로파일 데이터를 변환하지 않고 저장합니다. 여기서 JSON 파라미터는 '{...}',로 표시되며, 줄임표(...)는 포함된 JSON 데이터를 나타냅니다.
- JSON 키 체계는 userprofile:<userid> '{...}'입니다.

순위표

- Sorted set 데이터 형식은 순위표 워크로드에도 사용됩니다. Sorted set 데이터 형식은 지정된 키 체계를 사용하여 근처의 월간 거리 상위 사용자를 거의 실시간으로 저장합니다.
- 순위표 Sorted set 키 체계는 lb:<city>:<month> <distance> <userid>입니다.



DO NOT COPY
Ssleeeee1@gmail.com

ZADD: 월간 거리 순위표에 사용자 추가



```
>ZADD ZADD lb:d1:DC:August 120 U1000 90 U2310 210  
U3369 150 U5345  
4  
>
```

순위표는 Sorted set 데이터 형식을 사용합니다. 여기서는 순위표 세트에 여러 거리 수를 추가합니다.

이 데이터는 Amazon DynamoDB의 trips 테이블에 안정적으로 저장됩니다. 캐시가 중단되거나 손실될 경우 DynamoDB에서 이 데이터를 수집할 수 있습니다.

완료된 이동 거리만큼 총 거리 늘리기

Ib:d1:DC:August →

90	125	150	210
U2310	U1000	U5345	U3369

```
>ZINCRBY 1b:d1:DC:August 5 u1000
125
>
```

AWS

순위표 아키텍처는 사용자별 총 이동 거리가 먼저 DynamoDB에서 증가한 다음, ElastiCache for Redis에서 증가하는 것을 보여줍니다. 누적 이동 거리를 늘리려면 ZINCRBY Redis 명령을 사용합니다. 이 예에서는 사용자 u1000의 총 이동 거리가 5 증가하여 총 125마일이 되었습니다.

순위표에 사용자 순위 표시

```

>ZREVRANK 1b:d1:DC:August u1000
2
>

```

© 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

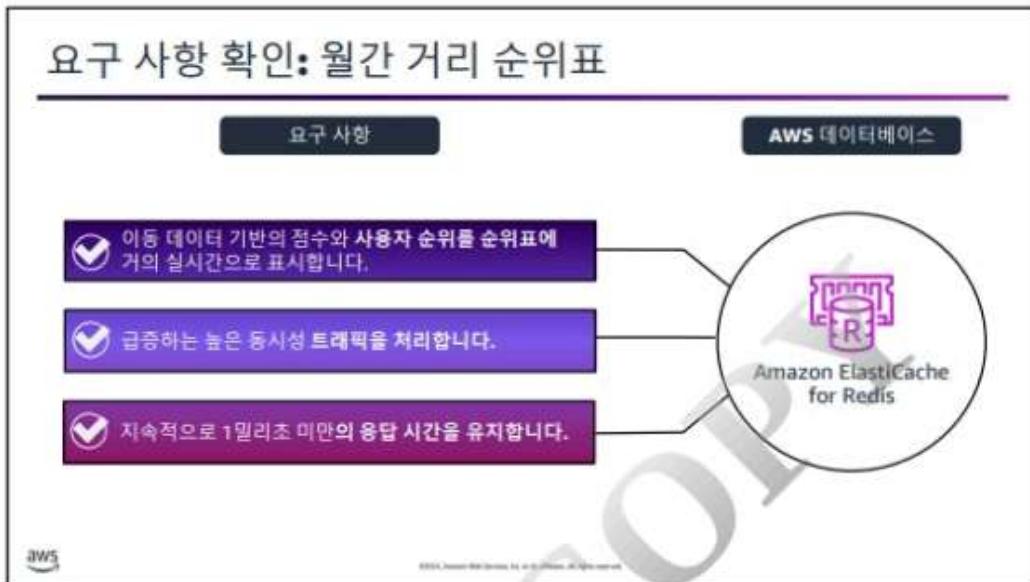
기본적으로 Redis는 Sorted set을 가장 낮은 값부터 가장 높은 값까지 오름차순으로 나열합니다. 따라서 ZREVRANK 명령을 사용하여 배터리 잔량이 가장 많은 것이 먼저 나열되고 가장 적은 것이 마지막에 나열되는 정렬된 순위를 검색합니다. Sorted set은 0부터 시작하는 순위 시스템을 사용합니다. 이동 거리가 가장 긴 사용자가 순위 0으로 표시됩니다.

이 예에서는 순위표에서 사용자 U1000의 순위를 요청합니다. 값 2가 반환됩니다. 사용자 U1000의 순위가 순위표에서 세 번째로 높다는 뜻입니다.

결과 제한(상위 3명만 표시)

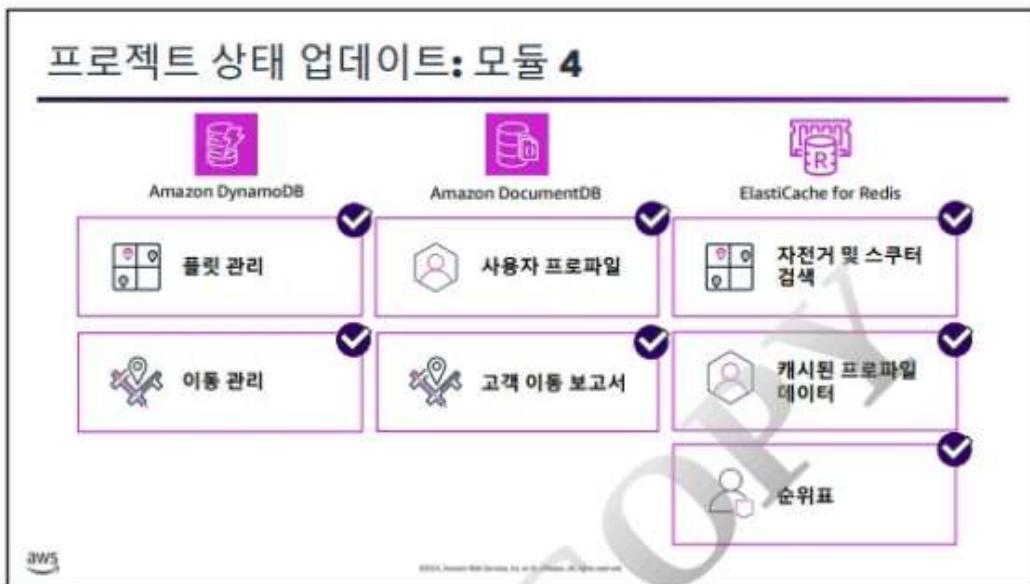
```
>ZRANGE lb:dt:DC:August +inf 0 BYSCORE REV WITHSCORES LIMIT 0 3
1) u3369
2) 210
3) u5345
4) 150
5) u1000
6) 120
>
```

모바일 앱에서 전체 사용자 순위표를 표시하면 안 됩니다. 사용자 기반이 커져도 순위표는 스케일업되지 않기 때문입니다. 여기서 ZRANGE 명령에는 워싱턴 DC의 8월 순위표를 역순으로 나열하고 출력을 멤버 3명으로 제한하는 파라미터가 있습니다.

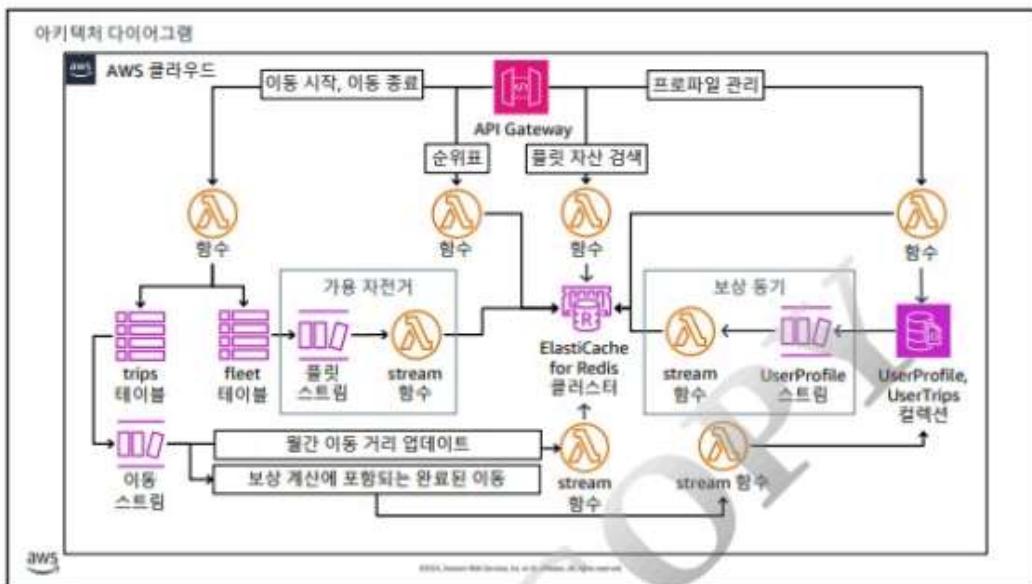


ElastiCache for Redis를 사용하여 이 섹션 시작 부분의 요구 사항을 충족하는 방법을 배웠습니다.





Amazon ElastiCache for Redis에서 자전거 검색, 프로파일 캐싱 및 순위표 워크로드를 계획하고 구축했습니다. 제품 팀에서 제시한 초기 프로젝트 요구 사항을 모두 충족했습니다. 다음 슬라이드에서 최종 아키텍처 디어그램을 살펴보십시오.



지금까지 구축한 솔루션을 좀 더 기술적으로 요약한 것입니다. Amazon API Gateway에 순위표 읽기 및 플랫 자산 검색 API를 추가했습니다. AWS Lambda 함수는 Amazon ElastiCache for Redis 클러스터에 대한 작업을 처리합니다.

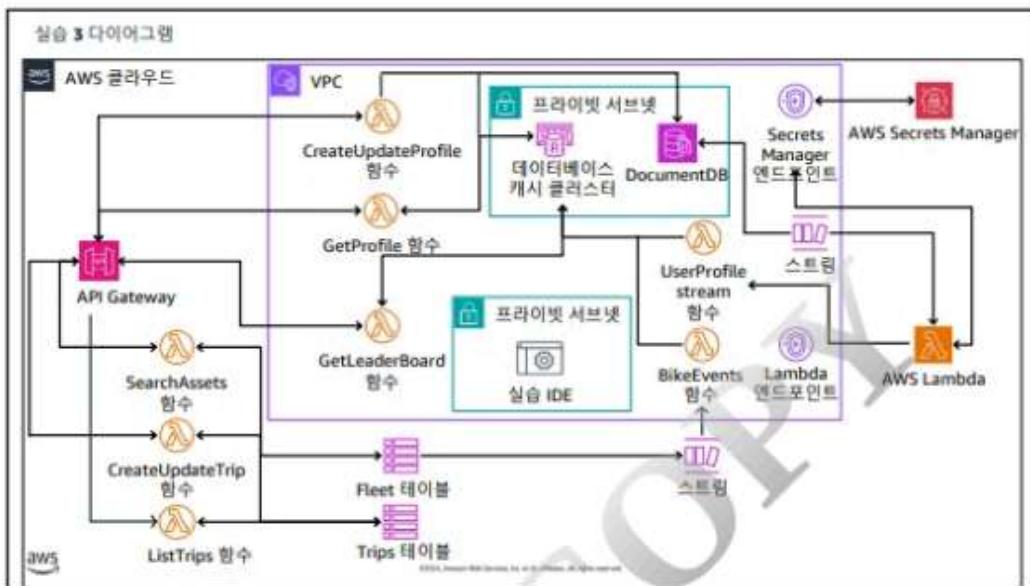
실습 3



Amazon ElastiCache for Redis를 사용하여 **Geospatial** 자전거 검색, 사용자 프로파일 캐싱 및 순위표 구현

이 실습에서는 다음 태스크를 수행합니다.

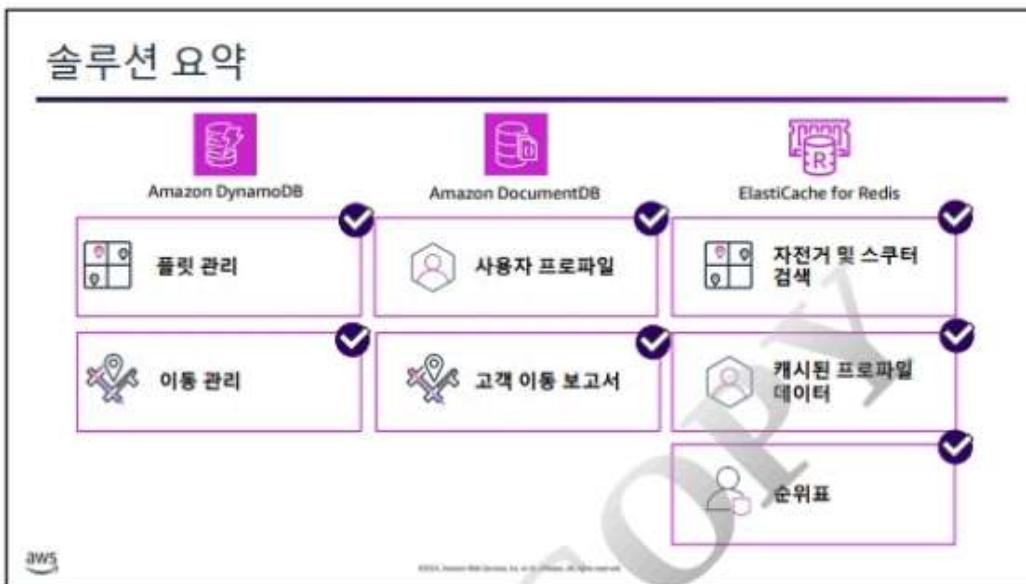
1. 자전거 검색 액세스 패턴을 구현합니다.
2. 사용자 프로파일 데이터의 프로파일 데이터 캐싱 솔루션을 구현합니다.
3. 거리 기준 상위 N명과 특정 사용자의 순위 및 점수를 보여주는 준실시간 월간 순위표를 구현합니다.



Amazon API Gateway에 순위표 및 플릿 자산 검색 API가 추가된 최종 실습 아키텍처. AWS Lambda 함수는 Amazon ElastiCache for Redis 클러스터에 대한 작업을 처리합니다.







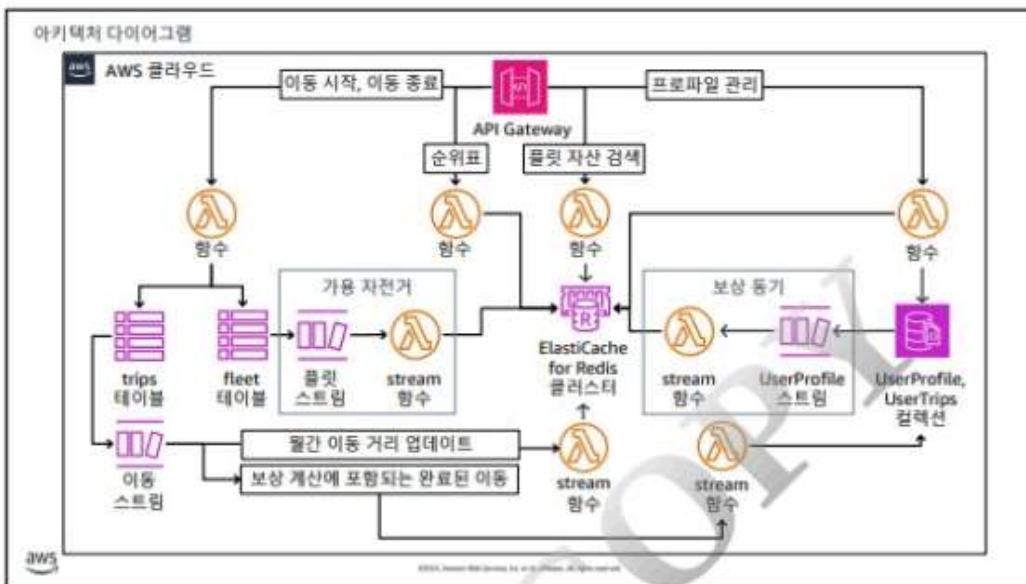
다음은 오늘 과정에서 완료한 내용을 요약한 것입니다.

모듈 1에서는 AnyCompany BikeShare에 대한 정보와 AWS NoSQL 데이터베이스를 사용하여 자전거 및 스쿠터 공유를 관리하는 모바일 앱을 구축하려는 AnyCompany BikeShare의 목표에 대해 알아보았습니다. AWS NoSQL 포트폴리오를 검토하고, 애플리케이션을 빌드하는 데 도움이 되는 세 가지 데이터베이스 유형을 선택했습니다.

모듈 2에서는 플릿 및 이동 관리를 주로 Amazon DynamoDB에서 추적하고 저장하는 방법을 배웠습니다.

모듈 3에서는 Amazon DocumentDB를 사용하여 사용자 프로파일 데이터를 저장 및 관리하는 방법과 부분 이동 데이터를 저장하여 고객 이동 보고서를 실행하는 방법을 알아보았습니다.

모듈 4에서는 Amazon ElastiCache for Redis와 관련된 자전거 및 스쿠터 검색 워크로드를 다시 살펴보았습니다. 관련 있는 활성 프로파일 데이터를 캐시하여 성능을 극대화하는 방법을 검토했습니다. 마지막으로, 순위표 요구 사항을 충족하기 위해 Redis 네이티브 순위표를 집계된 이동 거리 데이터에 사용했습니다.



이 과정에서는 Amazon API Gateway, AWS Lambda 및 여러 AWS 데이터베이스와 같은 서비스를 사용하여 이벤트 중심 아키텍처를 구축하기 위한 요구 사항과 액세스 패턴을 살펴보았습니다.

앱 사용자는 자산의 캐시된 위치를 기반으로 근처의 사용 가능한 자전거와 스쿠터를 요청합니다. 사용자가 이동을 시작하면 앱은 선택된 자산을 다른 사용자가 사용할 수 없게 만듭니다. 사용자가 이동을 마치면 앱은 DynamoDB *trips* 테이블과 Amazon ElastiCache for Redis 클러스터의 거리 점수를 업데이트합니다. 앱의 순위표 기능은 사용자의 진행 상황과 성과를 지속적으로 업데이트합니다. 앱이 자산을 해제하면 다시 사용할 수 있게 된 자전거나 스쿠터를 다른 사용자가 다시 요청할 수 있습니다.

Amazon DocumentDB는 프로파일 데이터를 관리하고, ElastiCache for Redis는 개인의 프로파일을 캐시하여 성능을 향상합니다. Amazon DocumentDB 변경 스트림 및 Lambda 함수는 ElastiCache for Redis 클러스터의 보상 및 프로모션에 대한 백엔드 변경을 수행합니다.

과정 목표

- AWS 목적별 NoSQL 데이터베이스를 사용하여 현대적 클라우드 중심 애플리케이션 빌드
- Key-value, 문서 및 메모리 내 데이터 범주를 처리하는 AWS 목적별 데이터베이스를 사용하는 솔루션 설명
- 비즈니스 사용 사례를 분석하고 Amazon DynamoDB, Amazon DocumentDB 및 Amazon ElastiCache의 고급 기능을 적용하여 확장 가능 솔루션 구현
- 변경 스트림 및 AWS Lambda를 사용하여 이벤트 중심 아키텍처 구현



© 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

이 과정을 시작할 때 과정 목표를 소개했습니다. 이 과정에서는 모듈 콘텐츠, 클래스 챗, 가이드 투어 및 실습을 통해 AWS 서비스에 대해 자세히 알아보면서 이러한 목표를 달성했습니다. 배운 새 기술을 실제 경험에 적용하고 교육 여정을 이어가십시오.





AWS Certification은 업계에서 인정하는 보안 인증으로 클라우드 전문 지식을 검증함으로써 신뢰와 확신을 쌓는 데 도움이 됩니다. 이 자격증은 조직에서 Amazon Web Services(AWS)를 사용하여 클라우드 이니셔티브를 이끌 수 있는 숙련된 전문가를 파악하는 데 도움이 됩니다.

슬라이드에는 현재 사용 가능한 AWS 자격증이 나와 있습니다. AWS 자격증을 취득하려면 감독관이 감독하는 시험에서 합격 점수를 받아야 합니다. 역할 기반 자격증은 다음과 같이 각 자격증 수준에 맞는 AWS 클라우드 서비스에 대한 권장되는 경험 수준을 제시합니다.

- **Professional:** 2년의 AWS 클라우드를 사용한 솔루션 설계, 운영, 문제 해결과 관련된 경험
- **Associate:** 1년의 AWS 클라우드를 통한 문제 해결 및 솔루션 구현 경험
- **Foundational:** 6개월의 기본 AWS 클라우드 경험 및 업계 지식 습득 경험

특수 전문 분야 자격증은 특정 기술 분야에 중점이 맞춰져 있습니다. 특수 전문 분야 시험 응시에 권장되는 경력은 시험 가이드에 지정된 분야에서의 기술 경력을 말합니다.

AWS는 자격증 시험에서 다루는 서비스 또는 기능의 전체 목록을 게시하지 않습니다. 하지만 각 시험의 시험 안내서에 시험에서 다루는 최신 주제 영역과 목표가 나와 있습니다. 자세한 내용과 시험 가이드 및 기타 준비 자료 리뷰는 **AWS Certification 시험 준비**

페이지(<https://aws.amazon.com/certification/certification-prep/>)에서 확인할 수 있습니다.

이 정보는 2024년 10월을 기준으로 유효합니다. 그러나 시험은 자주 업데이트되며 제공되는 시험과 각 시험의 테스트 항목에 관한 세부 정보는 변경될 수 있습니다. 최신 AWS Certification 시험 정보에 대한 자세한 내용은 **AWS Certification** 페이지(<https://aws.amazon.com/certification/>)를 참조하십시오.

3년마다 자격증을 갱신(또는 재인증)해야 합니다. 자세한 내용은 **AWS 자격증 갱신** 페이지(<https://aws.amazon.com/certification/recertification/>)를 참조하십시오.

자신 있게 자격증 시험 준비하기

1 단계 시험 및 시험 스타일 문제에 대해 알아보기

2 단계 AWS 지식과 실력을 복습하기

3 단계 시험 복습 및 연습하기

4 단계 시험 준비 상태 평가하기

AWS Skill Builder의 시험 준비 섹션 알아보기

© 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

이 과정에는 AWS Certification 시험과 관련이 있을 수 있는 콘텐츠가 포함됩니다. 시험 준비를 계속하려면 다음 핵심 4단계를 따르십시오.

각 시험에 대한 4단계 계획의 세부 사항을 확인하려면 QR 코드를 스캔하여 AWS Skill Builder의 준비 섹션을 탐색하거나 <https://skillbuilder.aws/#prepare-for-exam>를 방문하세요.

1 단계: 시험 스타일 문제로 시험에 대해 알아보기

시험의 중점 영역과 문제 유형에 익숙해지기



- 1 시험 가이드를 검토합니다.
- 2 AWS 온라인 학습 센터인 AWS Skill Builder에 액세스하기 위해 가입합니다.
- 3 등록하고 AWS Certification 공식 연습 문제 세트를 진행합니다.

1단계는 시험 스타일 문제로 시험에 대해 알아보기입니다.

시험 가이드 검토하시는 것을 추천하며, 4단계 계획은 이전 슬라이드에서 확인하십시오.

공식 연습 문제 세트는 인증 시험 문제 스타일을 보여주기 위해 AWS에서 개발한 20개의 문제로 구성됩니다. 시험 스타일의 문제를 이해하기 위해 이 문제 세트를 풀어보세요. 연습 문제에는 자세한 피드백과 시험 준비에 도움이 되는 추천 자료가 포함되어 있습니다.

2 단계: AWS 지식과 실력을 복습하기

AWS Skill Builder 학습 사용하기

The screenshot shows the AWS Skill Builder website with a purple header. Below it, a large banner for 'AWS Builder Labs' is displayed, featuring the text 'Learn cloud skills in a live-AWS environment' and a 'Start Learning' button. To the right of the banner is a section titled 'About AWS Builder Labs' with a small icon of a flask.

1 디지털 학습 과정을 수강합니다.

2 AWS Cloud Quest를 플레이합니다.

3 AWS Builder Labs에 액세스하여 실습을 진행하고 AWS Console에서 능력을 발휘합니다.

2단계는 시험 주제를 복습하는 것입니다. 각 시험을 위한 4단계 계획에는 구체적인 추천 사항이 포함되어 있습니다.

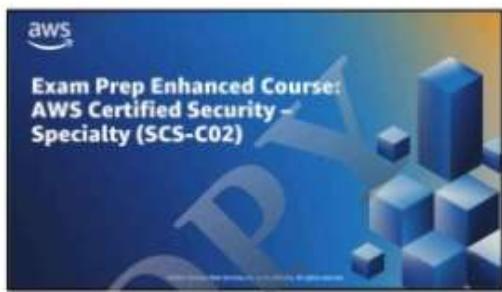
AWS 자격증을 취득하기 위해 필수 교육 과정은 없지만, 디지털 과정, Builder labs, AWS Cloud Quest, AWS Jams에 대한 추천을 제공합니다. 이러한 모든 교육 기회를 활용하려면 AWS Skill Builder를 구독하세요.

3 단계: 시험 복습 및 연습하기

시험 준비 과정 수강하기



- 1 각 시험 영역을 설명하는 비디오를 시청합니다.
- 2 Labs를 완료합니다.
- 3 시험 스타일 문제와 플래시카드로 연습합니다.

© 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

3단계는 복습과 연습입니다.

시험 준비를 위해 시험의 주제 영역을 살펴보고 이러한 영역이 AWS 개발 및 특정 학습 분야와 어떻게 연관되는지 확인하세요. 각 작업 설명에 따라 주제가 그룹화된 개념과 이해도를 평가하세요. 실습 실습과 시험 스타일 질문 설명을 통해 지식을 강화하고 학습 격차를 파악하세요. 강사가 시험 스타일 질문을 검토하는 것을 따라가고, 잘못된 응답을 식별하는 시험 응시 전략을 배우세요.

Note: 일부 자료는 AWS Skill Builder 구독이 필요합니다.

4 단계: 시험 준비 상태 평가하기

공식 모의 시험 또는 사전 테스트를 응시하기



시험 스타일로 채점되는 AWS Certification 공식 연습 시험을 실시합니다.

4단계는 시험 준비 상태 평가입니다.

시험에 따라 시험 준비 상태를 평가하기 위해 사전 테스트 또는 모의 시험을 응시하세요. 사전 테스트와 모의 시험의 문제는 실제 시험 질문과 동일한 스타일, 깊이, 높은 기준을 가지고 있습니다. 모두 자세한 피드백과 성적 보고서를 포함합니다. 유일한 차이점은 모의 시험이 시험 스타일 점수를 제공하는 반면, 사전 테스트는 표준 백분율 점수를 제공한다는 것입니다.



AWS는 시험에 응시하는 유연하고 편리한 옵션을 제공합니다. 시험 예약 페이지에서 본인에게 가장 적합한 시험 옵션을 선택합니다. 자세한 내용은 시험 일정 예약: 자신에게 가장 알맞은 테스트 옵션 찾기 (<https://aws.amazon.com/certification/certification-prep/testing/>)를 참조하십시오.

AWS Skill Builder 온라인 학습 센터



게임형 학습



자습형 실습(SPL)



사용 사례 과제



시험 준비

AWS의 전문가들이 개발한 600개가 넘는 강좌와 대화형 교육을 원하는 대로 활용하여 필요한 기술을 계속 발전시키십시오.



시작하기

© 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

온라인 학습 센터인 AWS Skill Builder로 학습을 계속하십시오.

원하는 속도로 목표를 달성할 준비가 되었습니까? AWS Skill Builder의 무료 디지털 교육은 필요한 기술을 원하는 방식으로 학습할 수 있도록 600개가 넘는 온디맨드 강좌와 학습 플랜을 제공합니다.

대화형 학습 환경에서 클라우드 문제 해결 스킬을 배우고 싶습니까? Skill Builder를 구독하면 자습형 실습(SPL), 연습 시험, 역할 기반 게임 및 실제 과제를 통해 빠르게 배울 수 있습니다.

더 많이 배우고 시작하는 방법에 관한 자세한 내용은 AWS Skill Builder(<https://aws.amazon.com/training/digital>)를 참조하십시오.

추가 학습 기회를 놓치지 마세요.



무료 디지털 교육 AWS Fundamentals에서 제공하는 수백 개의 무료 자습형 디지털 과정을 통해 배우십시오.	강의실 교육 기술 스킬을 발전시키고 개인 AWS 강사에게 배우십시오.	AWS Certification 업계에서 인정받는 자격증으로 전문 지식을 검증하십시오.
--	--	--

AWS Fundamentals은 Amazon Web Services, Inc.의 저작물입니다. © 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

AWS 교육 및 자격증은 AWS에 대한 지식을 확대 및 심화하고 AWS 서비스 사용의 확산을 촉진하는 데 전념하는 조직입니다. AWS 프로그램은 고객, AWS 파트너 및 AWS 직원을 위해 설계되었습니다. 지난 몇 개월 동안 우리는 고객과 파트너에게 몇 가지 새로운 과정, 교육 실습 및 자격증을 출시했습니다.

AWS 클라우드 기술을 확장하십시오. 자세한 내용은 다음 리소스를 참조하십시오.

- Digital training – <https://explore.skillbuilder.aws/>
- Classroom training – <https://aws.amazon.com/training>
- AWS Certification – <https://aws.amazon.com/certification>
- AWS Workshops – <https://workshops.aws/>
- Tech Talks – <https://aws.amazon.com/events/online-tech-talks/on-demand/>
- AWS Ramp-Up Guides – <https://aws.amazon.com/training/ramp-up-guides/>

