첫 NoSQL 사용기

with 

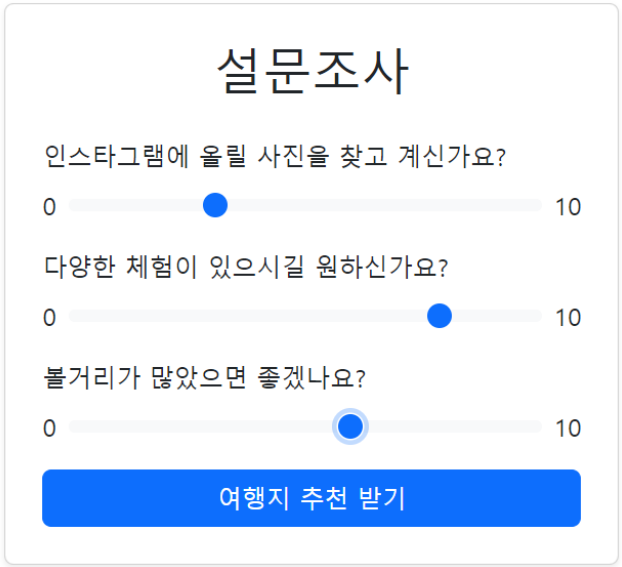
컴퓨터공학부 202017999 이주형

목차

1. 추천 서비스 소개
2. RDS 구현
3. DynamoDB 연동
4. DynamoDB 소개
5. AWS Translate

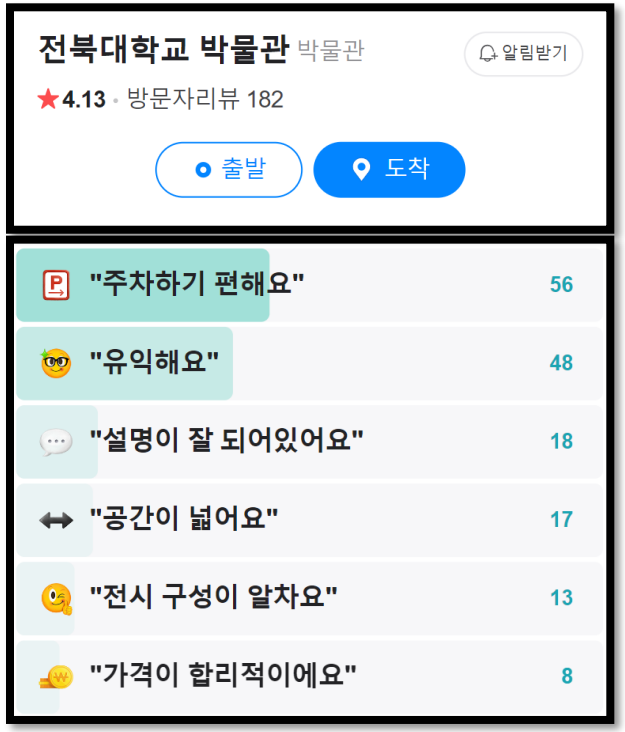
FollowMe 서비스 소개

- 주성분 분석을 활용한 여행지 추천 서비스
- 네이버 지도 리뷰 탭의 “이런 점이 좋았어요 "
- 관광지/특징 행렬의 PCA를 통해 상위 3개 latent vector 추출
- 각 Latent vector를 대표하는 질문



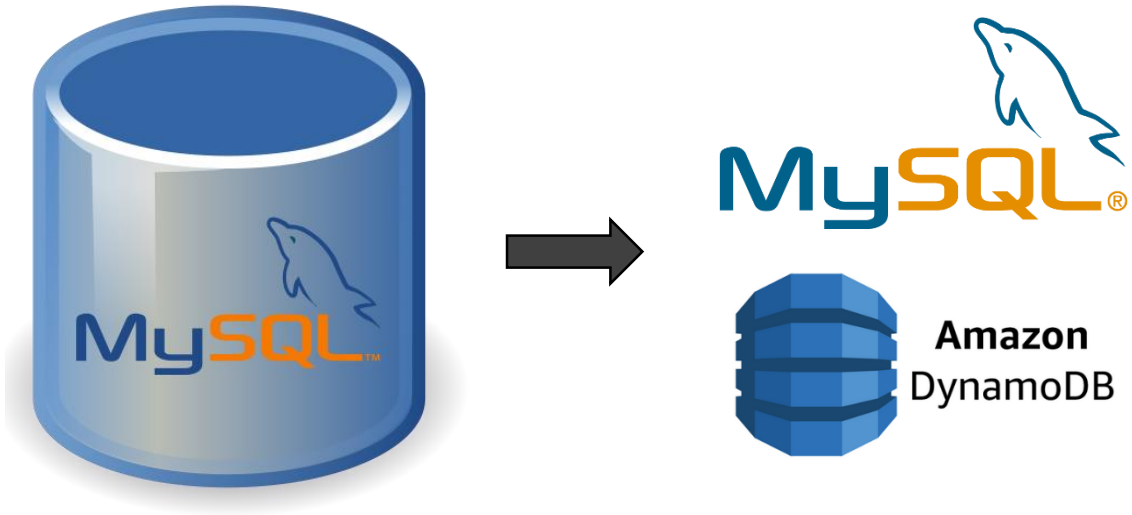
관광지

특징



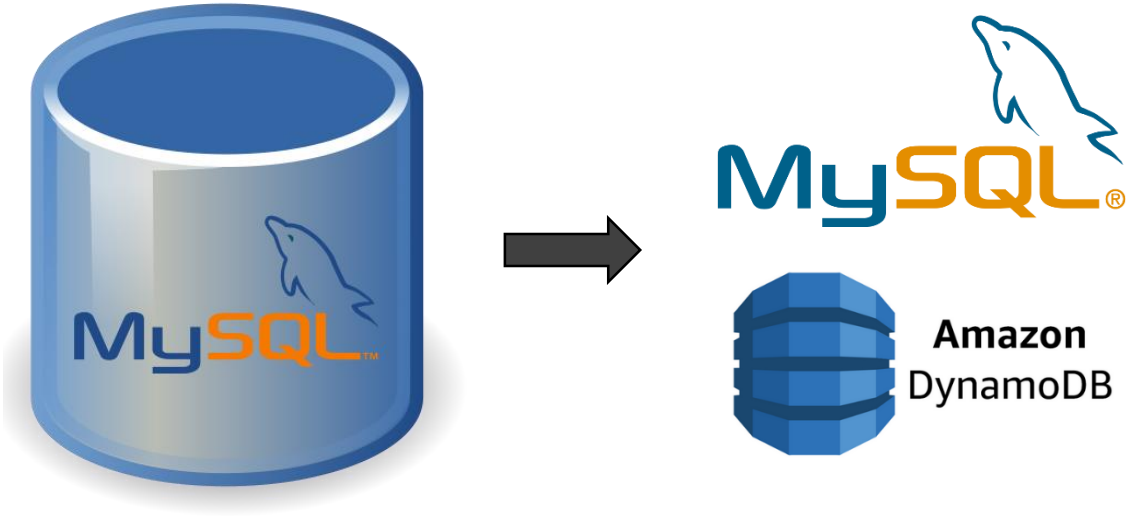
EC2 기반 아키텍처

- Linux, Tomcat, MySQL, JSP 스택 구성
- 추가적으로 RDS일부 테이블을 DynamoDB로 마이그레이션
  -> json이용



DB 구현

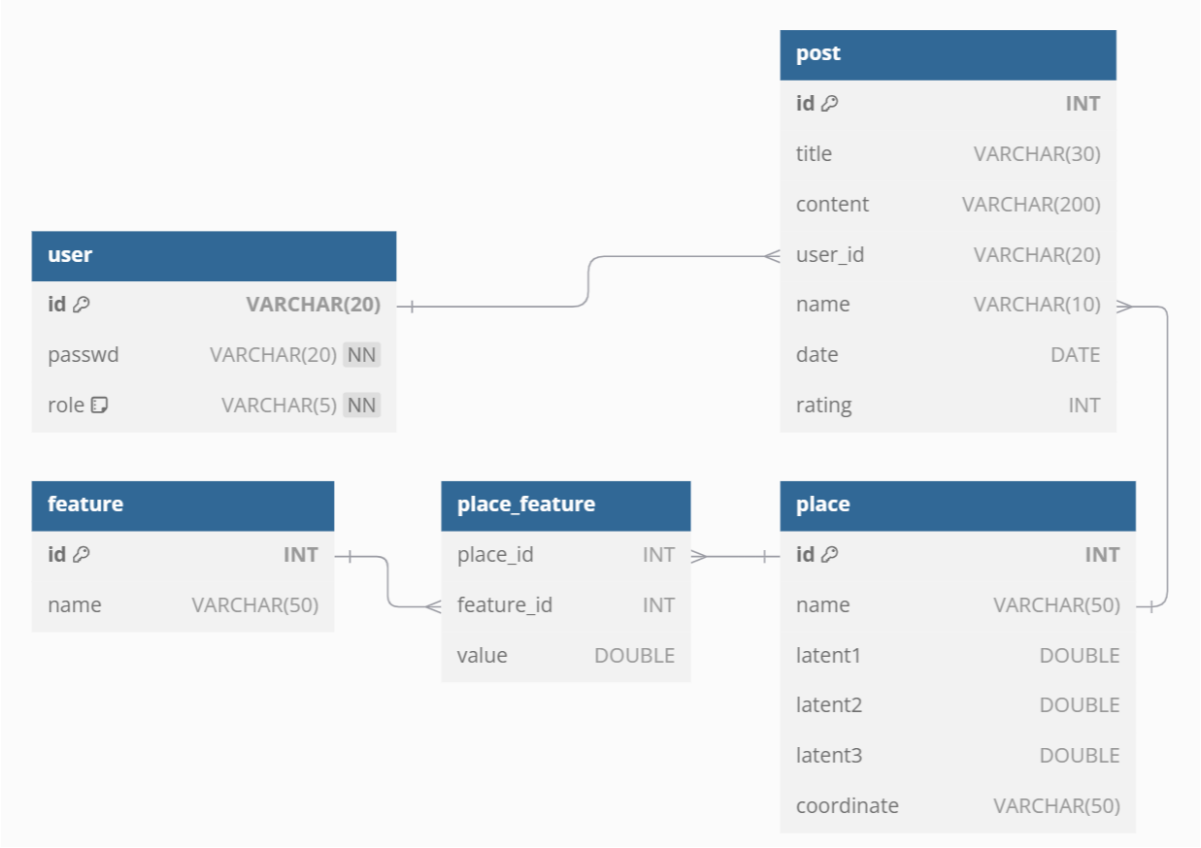
- 조인이 일어나는 추천 페이지
- MySQL 단일 구성
- MySQL + DynamoDB 구성



```
String sql =
    "SELECT bp.id, bp.name, bp.coordinate, bp.score, f.name AS feature_name " +
    "FROM ( " +
    "    SELECT p.id, p.name, p.coordinate, " +
    "           (p.latent1 * ? + p.latent2 * ? + p.latent3 * ?) AS score " +
    "    FROM place p " +
    "    ORDER BY score DESC " +
    "    LIMIT 1 " +
    ") bp " +
    "JOIN place_feature pf ON bp.id = pf.place_id " +
    "JOIN feature f ON pf.feature_id = f.id " +
    "ORDER BY pf.value DESC " +
    "LIMIT 3";
```

MySQL 단일 구현

- 특징 관련 테이블은 중복되는 내용이 많았기 때문에 설계가 복잡함



인덱스 생성 전

```
mysql> EXPLAIN ANALYZE
    -> SELECT bp.id, bp.name, bp.coordinate, bp.score, f.name AS feature_name
    -> FROM (
    ->     SELECT p.id, p.name, p.coordinate,
    ->            (p.latent1 * 5 + p.latent2 * 5 + p.latent3 * 5) AS score
    ->     FROM place p
    ->     ORDER BY score DESC
    ->     LIMIT 1
    -> ) bp
    -> JOIN place_feature pf ON bp.id = pf.place_id
    -> JOIN feature f ON pf.feature_id = f.id
    -> ORDER BY pf.value DESC
    -> LIMIT 3;
+--------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------
| EXPLAIN


+--------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------
| -> Limit: 3 row(s)  (cost=68 rows=3) (actual time=0.218..0.222 rows=3 loops=1)
    -> Nested loop inner join  (cost=68 rows=301) (actual time=0.218..0.221 rows=3 loops=1)
        -> Sort: pf.`value` DESC  (cost=30.4 rows=301) (actual time=0.204..0.205 rows=3 loops=1)
            -> Filter: (pf.place_id = '21')  (cost=30.4 rows=301) (actual time=0.168..0.185 rows=17 loops=1)
                -> Table scan on pf  (cost=30.4 rows=301) (actual time=0.0118..0.171 rows=301 loops=1)
        -> Single-row index lookup on f using PRIMARY (id=pf.feature_id)  (cost=0.253 rows=1) (actual time=0.0052..0.00523 rows=1 loops=3)
```

인덱스 생성 후

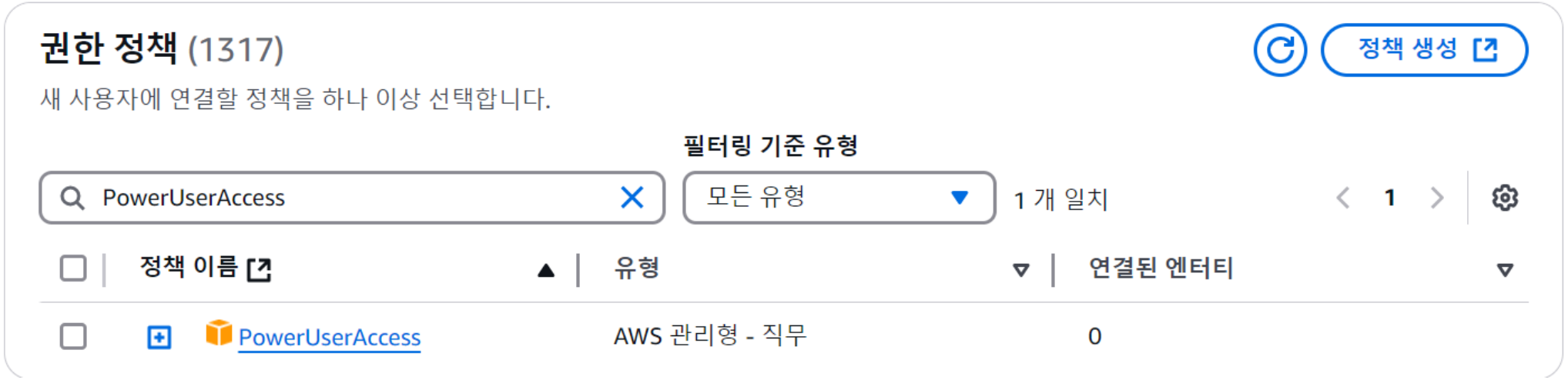
```
mysql> EXPLAIN ANALYZE
    -> SELECT /*+INDEX(pf idx_place_feature_placeid_value) INDEX(f PRIMARY) */
    ->        bp.id,
    ->        bp.name,
    ->        bp.coordinate,
    ->        bp.score,
    ->        f.name AS feature_name
    -> FROM (
    ->     SELECT p.id,
    ->            p.name,
    ->            p.coordinate,
    ->            (p.latent1 * 5 + p.latent2 * 5 + p.latent3 * 5) AS score
    ->     FROM place p
    ->     ORDER BY score DESC
    ->     LIMIT 1
    -> ) bp
    -> JOIN place_feature pf
    ->   ON bp.id = pf.place_id
    -> JOIN feature f
    ->   ON pf.feature_id = f.id
    -> ORDER BY pf.value DESC
    -> LIMIT 3;
+--------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------
| EXPLAIN

+--------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------
| -> Limit: 3 row(s)  (cost=8.4 rows=3) (actual time=0.0191..0.0269 rows=3 loops=1)
    -> Nested loop inner join  (cost=8.4 rows=17) (actual time=0.019..0.0266 rows=3 loops=1)
        -> Index lookup on pf using idx_place_feature_placeid_value (place_id='21') (reverse)  (cost=2.45 rows=17) (actual time=0.0116..0.0162 rows=3 loops=1)
        -> Single-row index lookup on f using PRIMARY (id=pf.feature_id)  (cost=0.256 rows=1) (actual time=0.00293..0.00293 rows=1 loops=3)
```

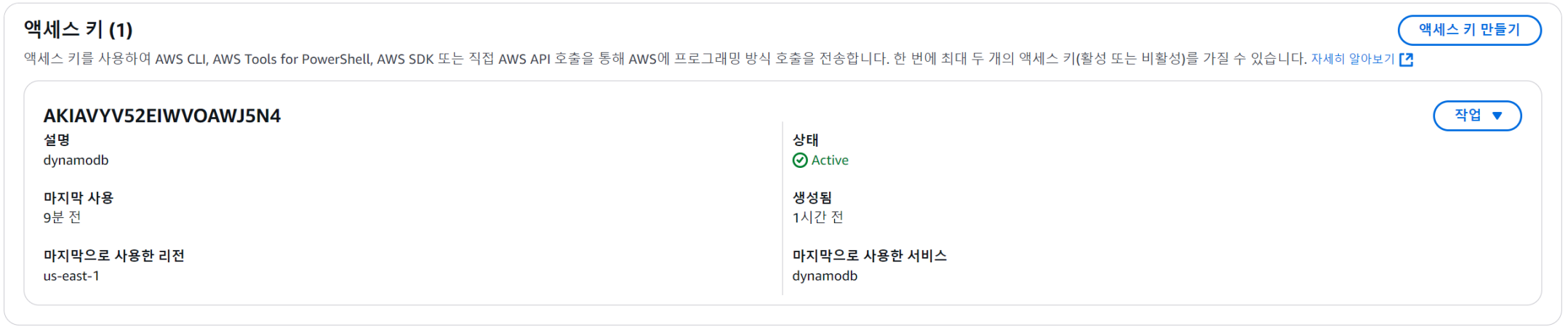
RDS의 복잡한 작업

- 스키마 설계
- 정규화
- 인덱스 생성

-> DynamoDB를 사용해보자

DynamoDB 로컬 사용 권한 부여





DynamoDB 테이블 생성

테이블 생성

테이블 세부 정보 정보

DynamoDB는 테이블을 생성할 때 테이블 이름과 기본 키만 필요한 스키마리스 데이터베이스입니다.

테이블 이름

테이블을 식별하는 데 사용됩니다.

feature

문자, 숫자, 밑줄(_), 하이픈(-) 및 마침표(.)만 포함하는 3~255자의 문자입니다.

파티션 키

파티션 키는 테이블 기본 키의 일부로, 테이블에서 항목을 검색하고 확장성과 가용성을 위해 호스트에 데이터를 할당하는 데 사용되는 해시 값입니다.

name | 문자열

1~255자이고 대소문자를 구분합니다.

정렬 키 - *선택 사항*

정렬 키를 테이블 기본 키의 두 번째 부분으로 사용할 수 있습니다. 정렬 키를 사용하면 동일한 파티션 키를 공유하는 모든 항목을 정렬하거나 검색할 수 있습니다.

정렬 키 이름 입력 | 문자열

1~255자이고 대소문자를 구분합니다.

DynamoDB 테이블 조회

- 파티션 키로 조회
- 특징을 모두 요청한 뒤 서버에서 정렬
- Top-N Query 이용 불가로
  약간의 I/O(unit) 수가 증가하지만
  접근 패턴이 하나밖에 없기에
  효율적 키 디자인 패턴 사용가능

```java
//DynamoDB feature 테이블에서 상위 3개의 특성 조회
String tableName = "feature";
String primaryKey = "name";


Map<String, AttributeValue> keyToGet = new HashMap<>();
keyToGet.put(primaryKey, AttributeValue.builder().s(bestPlaceName).build());

GetItemRequest req = GetItemRequest.builder()
    .tableName(tableName)
    .key(keyToGet)
    .build();

GetItemResponse res = dynamoDbClient.getItem(req);
if (res.hasItem()) {
    Map<String, AttributeValue> item = res.item();


    List<Map.Entry<String, AttributeValue>> sortedEntries = new ArrayList<>(item.entrySet());
    sortedEntries.removeIf(entry -> entry.getKey().equals(primaryKey)); // name 필드 제외

    sortedEntries.sort((e1, e2) -> {
        String v1 = e1.getValue().s() != null ? e1.getValue().s() : e1.getValue().n();
        String v2 = e2.getValue().s() != null ? e2.getValue().s() : e2.getValue().n();
        return v2.compareTo(v1);
    });

    // 상위 3개의 특징을 배열에 저장
    for (int i = 0; i < Math.min(3, sortedEntries.size()); i++) {
        topFeatures[i] = sortedEntries.get(i).getKey();
    }
} else {
    for (int i = 0; i < topFeatures.length; i++) {
        topFeatures[i] = "데이터 없음";
    }
}
```

NoSQL vs RDS

NoSQL 장점

- 높은 수평적 확장성
- 파티션 키 별로 분산 저장
- 파티션 키 = 해싱
- 작은 서버들의 Scale Out

RDS 단점

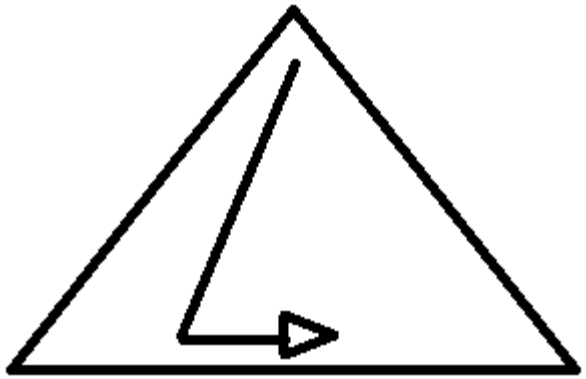
- 중앙 집중 트랜잭션
  - ACID를 통한 무결성 보장
  - 분산 처리에 유연하지 않음
- B-트리 인덱스
  - 동기화 성능 저하
- RDS주안점
  - 정규화
  - Scale Up

NoSQL 단점

- 해싱
  - 등호 조건만 검색 가능(=)
  - >,< 등의 범위 검색 불가능

- 접근 패턴 유연성 낮음
  - GSI를 통해 해결(만능은 아님)

- RDS
  - Ad-hoc쿼리



Lesson Learned

- 단순한 조회 패턴 덕분에 DynamoDB를 효율적으로 활용했음
- NoSQL의 간편함
- 추후 여행지가 많아지더라도 효과적으로 처리 가능

발표 영상 외 추가 구현 사항

- AWS translate 활용
- 간단하게 번역 지원 국가 추가 가능
- 앞 선 슬라이드와 마찬가지로
  PowerUserAccess 권한 부여 후
  발급 받은 엑세스 키를 이용

여행지 추천 결과

추천 여행지: **전주드림랜드**

여행자들이 뽑은 상위 3개 특징!

1. 볼거리가 많아요

2. 아이와 가기 좋아요

3. 가격이 합리적이에요

GPT가 설명하는 전주드림랜드

전주드림랜드는 한국의 대표적인 놀이동산으로, 다양한 스릴 있는 놀이기구와 테마존으로 구성되어 있습니다.
놀이기구를 타고 즐기는 동안 전통 한옥 마을을 감상할 수 있는 곳으로, 전주의 역사와 문화를 느낄 수 있는 특별한 경험이 가능합니다.

번역할 언어를 선택하세요:

영어

번역 요청하기

Jeonju Dreamland is Korea's representative amusement park, which consists of a variety of thrilling rides and themed zones.
It is a place where you can enjoy a traditional hanok village while enjoying the rides, and you can have a special experience where you can feel the history and culture of Jeonju.

감사합니다