Developing with DynamoDB

Ivan Mushketyk

@mushketyk brewing.codes

Overview

How to access DynamoDB

Different APIs

Create shop items database

Optimistic locking

Transactions

How to implement full-text search

DynamoDB API

How to interact with DynamoDB

Low-level interface

Document interface

Object persistence interface

How to Interact with DynamoDB

RDBS - interact with SQL queries

DynamoDB – interact with HTTP interface

Has several HTTP methods

DynamoDB Methods

CreateTable
DeleteTable
ListTables
UpdateTable

BatchGetItem
GetItem
Query
Scan

BatchWriteItem
DeleteItem
PutItem
UpdateItem

DescribeTimeToLive
UpdateTimeToLive

TagResource
UntagResource
ListTagsOfResource

DescribeLimits

Request Example

```
POST / HTTP/1.1
Host: dynamodb.us-west1.amazonaws.com;
…
X-Amz-Date: 20160811T122000Z
X-Amz-Target: DynamoDB_20120810.GetItem
{
  “TableName”: ”ForumMessages”,
  “Key”: {
    ”UserId”: {”N”: “1”},
    ”Timestamp”: {”N”: “1498928631”}
  }
}
```

Reply Example

HTTP/1.1 200 OK
x-amzn-RequestId: <RequestId>
x-amz-crc32: <Checksum>
Content-Type: application/x-amz-json-1.0
Content-Length: <PayloadSizeBytes>

```
{
  "Item": {
    "UserId": {"N": "1"},
    "Timestamp": {"N": "1498928631"},
    "Message": {"S": "This forum is boring…"}
  }
}
```

DynamoDB APIs

Low-level

Directly maps DynamoDB HTTP methods

Document Interface

Higher-level interface to perform CRUD operations

Object Persistence

Create objects that represent tables and interact with them

Low-level Interface Example

```
HashMap<String, AttributeValue> key = new HashMap<>();
key.put(”UserId”, new AttributeValue()
        .withN(”1”));
key.put(”Timestamp”,new AttributeValue()        .withN(“1498928631”));

GetItemRequest request = new GetItemRequest()
.withTableName(”ForumMessages”)
.withKey(key);
```

Low-level Interface Example (cont.)

```
AmazonDynamoDB client =
AmazonDynamoDBClientBuilder.standard().build();

GetItemResult result = client.getItem(request);
AttributeValue year = result.getItem().get(”Message");
String message = attributeValue.getS();
```

Document Interface

```
AmazonDynamoDB client =
AmazonDynamoDBClientBuilder.standard().build();
DynamoDB docClient = new DynamoDB(client);

Table table = docClient.getTable(”ForumMessages”);
GetItemOutcome outcome = table.getItemOutcome(        ”UserId”, ”1”,
      ”Timestamp”, “1498928631”);

String message = outcome.getItem().getString(”Message“);
```

High-level DynamoDB API

Object persistence interface

How to define classes for it

How to use it

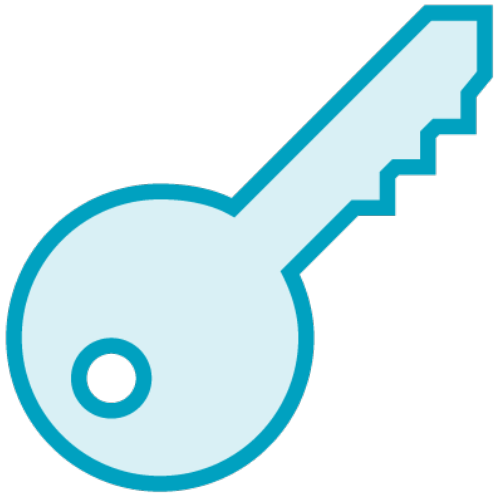
Define Class with Annotations

```
@DynamoDBTable(tableName="ForumUsers")
public class User {

    @DynamoDBHashKey(attributeName="UserId")
    public int getUserId() { return userId;}

    @DynamoDBAttribute(attributeName = "Name")
    public int getName() { return name;}
}
```

Get Item with Object Persistence Interface

```
AmazonDynamoDB client =
AmazonDynamoDBClientBuilder.standard().build();
DynamoDBMapper mapper = new DynamoDBMapper(client);
User key = new User();
key.setUserId(1);
User result = mapper.load(key);
```

Complex Queries



Composite key

Learn how to use LSI and GSI

Implement comments for shop items

What Queries do We Need?

Get all comments for an item

Get all comments with rating higher than

Get all comments by a user

Data Model

Primary key

Sort key

GSI Primary key

LSI

ItemId	MsgId	Msg	Timestamp	UserId	Rating
124	1	Delivered on time.	300	101	5
124	2	Good stuff!	200	800	4
124	3	Not as described!	100	456	1
865	4	So-so...	454	2	3
317	5	Kittens photos here:	50	412	5

GSI Sort key

```
@DynamoDBRangeKey(
  attributeName=”MessageId”)
@DynamoDBIndexRangeKey(
 localSecondaryIndexName=“lsi”,
 attributeName=“Time”)
@DynamoDBIndexHashKey(
 globalSecondaryIndexName=“gsi”,
 attributeName=“UserId”)
@DynamoDBIndexRangeKey(
globalSecondaryIndexName=“gsi”,
 attributeName=“Time”)
```

- Specify sort key
- Local secondary index
- Global secondary index partition key
- Global secondary index sort key

Conditional Updates

Discuss conditional updates

Why are they important

How to use them

Write Conflict

Get item 1

Edit description

Write item

Get item 1

Edit title

Write item

Conditional Update

Perform an operation only if a condition met

Check attribute value

Check if attribute exists

Supports Boolean logic

Will implement optimistic locking

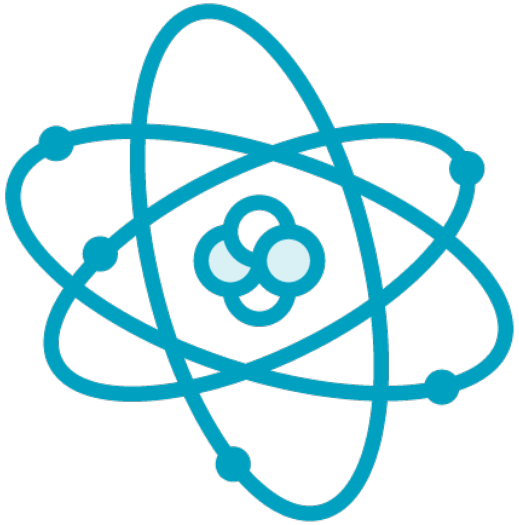
Add Version Attribute

```
@DynamoDBTable(tableName="ProductCatalog")
public class Item {
  …
  @DynamoDBVersionAttribute
  public Long getVersion() {
    return version;
  }
}
```

Override Optimistic Locking

```
mapper.save(item,
  new DynamoDBMapperConfig(
    DynamoDBMapperConfig.SaveBehavior.CLOBBER));
```

Transactions in DynamoDB



Why would we use transactions

How to use them

Limitations they have

Use-case to Implement

Add rating to item's comments

Rating of an item

- Average of all comments' ratings
- Updated when a comment added or updated

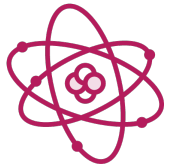
Will this Work?

```
Comment comment = …

Item item = getItem(comment.getItemId());

item.rating = newRating(comment);

mapper.save(comment);

mapper.save(item);
```

This is not Safe

```
Comment comment = …

Item item = getItem();

item.rating = newRating(comment);

mapper.save(comment);

// Process crashes here

mapper.save(item);
```

Transactions in DynamoDB



Atomic writes

Isolated reads

Removes stuck transactions

Easy to use

Transactions implemented via a client-side library in Java

How to Enable Transactions

```
AmazonDynamoDB client = new AmazonDynamoDBClient();
TransactionManager.verifyOrCreateTransactionTable(
client, “Transactions”,
10, 10, // RCU/WCU
10 * 60); // wait time in seconds
TransactionManager.verifyOrCreateTransactionImagesTable(
client, “TransactionImages”,
10, 10, // RCU/WCU
10 * 60); // wait time in seconds
```

How to Use TransactionManager

```
Transaction t = txManager.newTransaction();

Comment comment = …
Item item = getItem();
item.rating = newRating(comment);
t.save(comment);
t.update(item);

t.commit();
t.delete();
```

How Transactions Work

Transaction manager stores:

- List of updates in a transaction
- Items copies
- Lock attributes

Uses optimistic locking

Can read in more details:
https://github.com/awslabs/dynamodb-transactions/blob/master/DESIGN.md

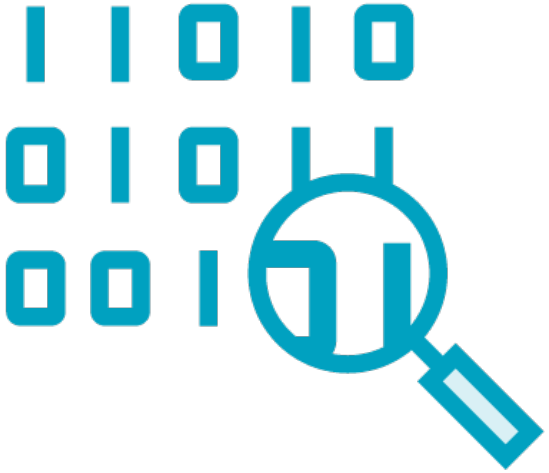
Limitations

Will not scale to many update commands

Does not lock ranges

Every transaction requires 7N+4 writes

Full Text Search with DynamoDB

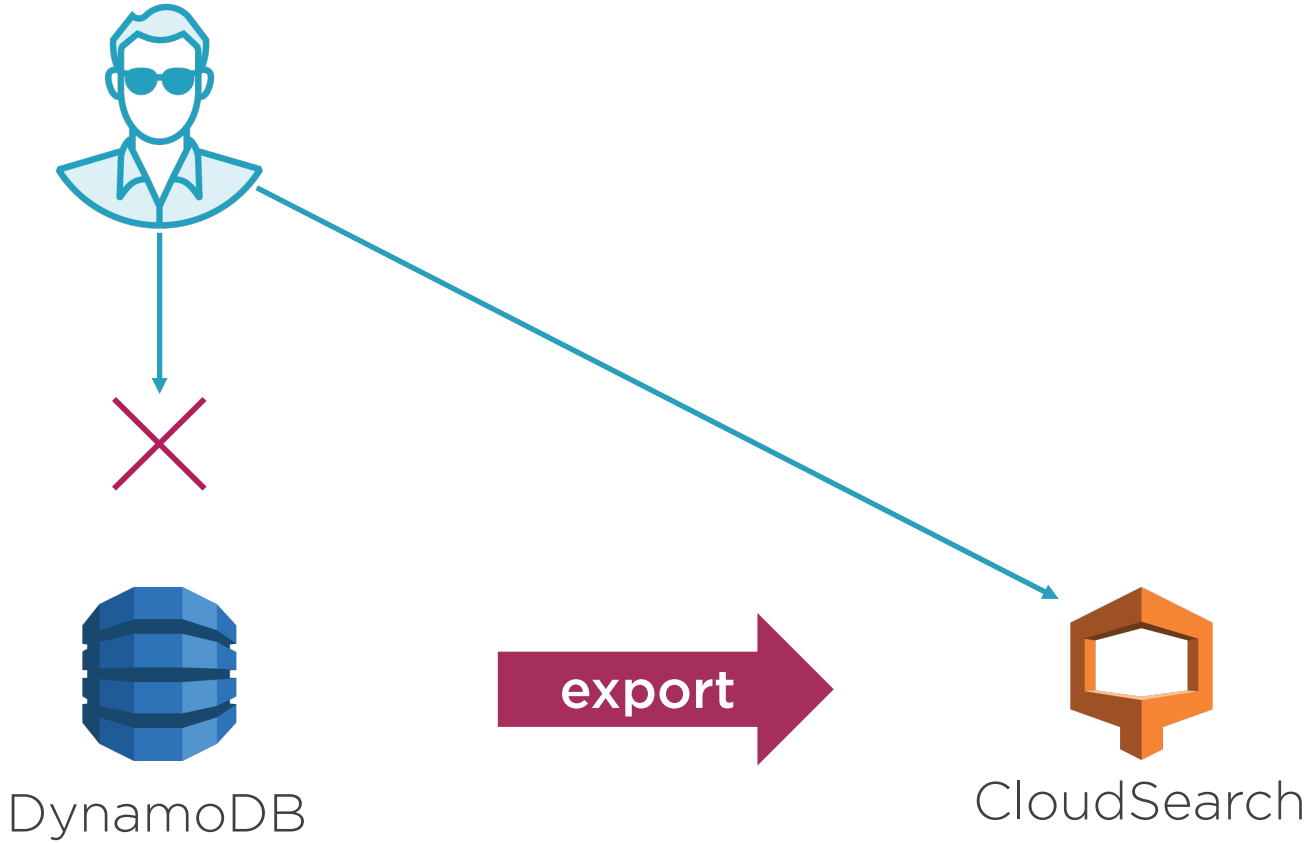


Implement text search

Overview of CloudSearch service

How to use CloudSearch with DynamoDB

Text Search in DynamoDB



What is CloudSearch

- Managed search service
- Built on top of Apache Solr
- Allows to import data from different sources
- Simple text searches
- Boolean combination of fields
- Implements ranking

Search Domain

Document service

Search service

Configuration service

How to Use CloudSearch

Create a search domain

Upload data to the search domain

Submit search requests

Import Data to CloudSearch

Cloud Search console

- From filesystem/DynamoDB/S3
- Up to 5MB

cs-import-documents

Summary

- Low-level and high-level APIs
- How to use keys, LSIs, and GSIs
- Conditional updates
- DynamoDB transactions
- Full-text search with CloudSearch