Dynamo DB - Programming

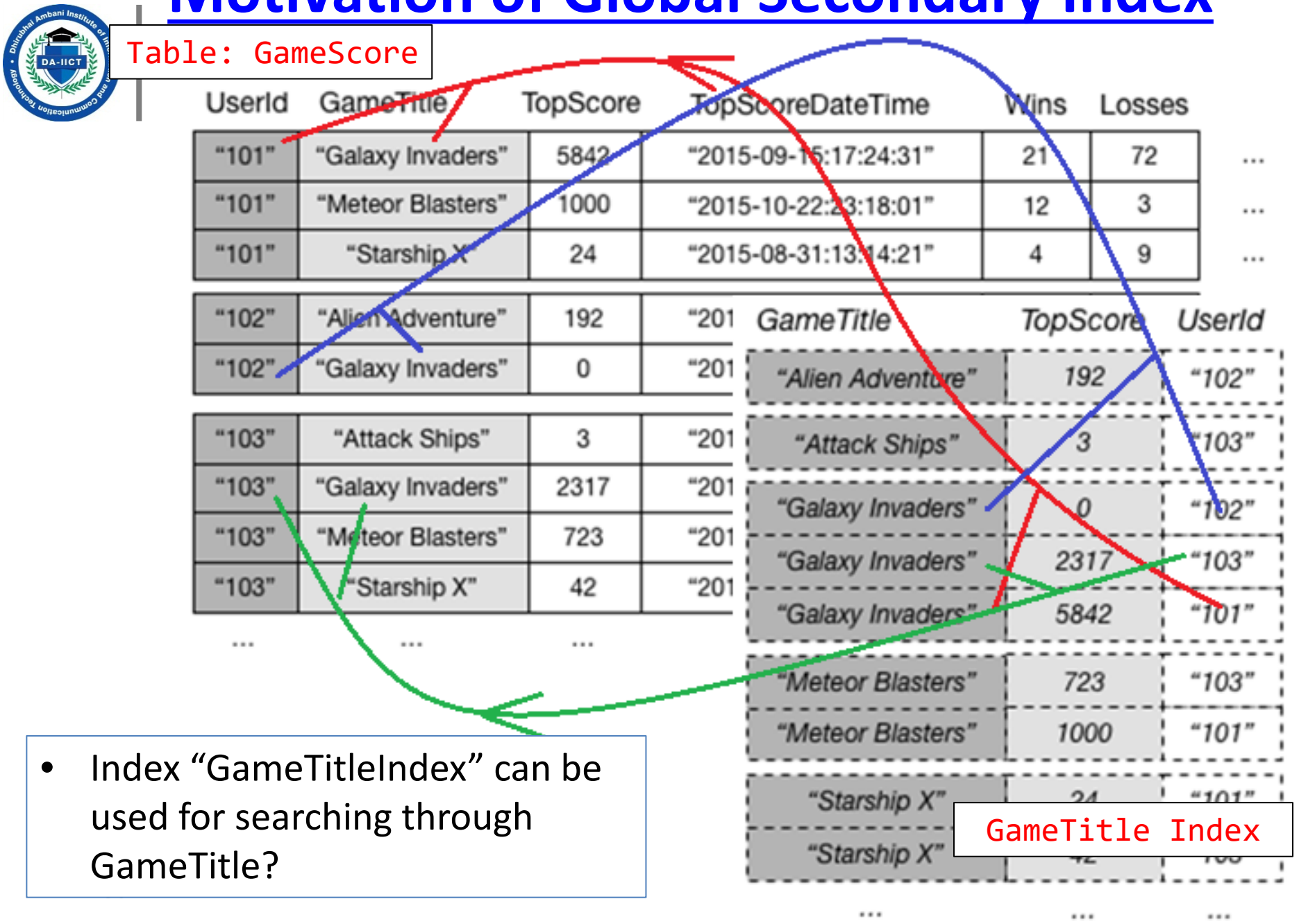
pm_jat @ daiict

Motivation of Global Secondary Index

UserId	GameTitle	TopScore	TopScoreDateTime	Wins	Losses	
"101"	"Galaxy Invaders"	5842	"2015-09-15:17:24:31"	21	72	...
"101"	"Meteor Blasters"	1000	"2015-10-22:23:18:01"	12	3	...
"101"	"Starship X"	24	"2015-08-31:13:14:21"	4	9	...
"102"	"Alien Adventure"	192	"2015-07-12:11:07:56"	32	192	...
"102"	"Galaxy Invaders"	0	"2015-09-18:07:33:42"	0	5	...
"103"	"Attack Ships"	3	"2015-10-19:01:13:24"	1	8	...
"103"	"Galaxy Invaders"	2317	"2015-09-11:06:53:00"	40	3	...
"103"	"Meteor Blasters"	723	"2015-10-19:01:13:24"	22	12	...

- By Having UserID (as Partition Key) and GameTitle (as sort key), we also have Index on UserID, GameTitle?
- In this scenario, Queries done on "both attributes" and "UserID" alone are efficient enough.
- But this "index" is not good when we require querying on "GameTitle" alone, or any other attribute. Why?

Motivation of Global Secondary Index



- Index "GameTitleIndex" can be used for searching through GameTitle?

Secondary Index - Summarized

- Indexes are mechanism of speeding up the execution of database operations.
- We require creating indexes on attributes that are often used as “search key” in queries (used in where clause)
- GSI in Dynamo have partition key, sort key, referencing attributes, and may have additional attributes.
- Following options we have in index for attributes in the index:
  - KEYS_ONLY, INCLUDE, ALL

https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Query.html

Exercises

- Try Figuring out solution for following queries on Table “GameScores” and given global secondary index:
- Q#1: What is the score of user id “103” in Game “Galaxy Invaders”?
- Q#2: What are the top 3 games of user id “103”?
- Q#3: What is the top score ever recorded for the game “Meteor Blasters”?
- Q#4: Which user has the highest score for Galaxy Invaders?
- Q#5: What is the highest ratio of wins vs. losses?

https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Query.html

Exercises - solution

- For each query specify answer in terms of
  - Dynamo B Operation Name, and Parameters

Exercise - Solutions

- Q#1: What is the score of user id “103” in Game “Galaxy Invaders”?
  - Use Primary Index
  - Use “Get-Item” with inputs of UserID(Partition Key)=103, and GameTitle(Sort Key) = “Galaxy Invaders”
- Q#2: What are the top 3 games of user id “103”?
  - Use Primary Index
  - Use “Query” with inputs of User-ID 103, and Limit response to 3.
- Q#3: What is the top score ever recorded for the game “Meteor Blasters”?
  - Use Global Secondary Index “GameTitleInex”
  - Use Query with input GameTitle=“Meteor Blasters”, and Limit response to 1.
  - Project: Top Score

Exercise - Solutions

- Q#4: Which user has the highest score for Galaxy Invaders?
  - Use Global Secondary Index “GameTitleInex”
  - Use Query with input GameTitle=“Meteor Blasters”, and Limit response to 1.
  - Project: User ID
- Q#5: What is the highest ratio of wins vs. losses?
  - No index can be used here
  - Use SCAN, and get all items
  - Write code that iterates through returned items and compute highest ratio of wins/losses.

Dynamo Operations

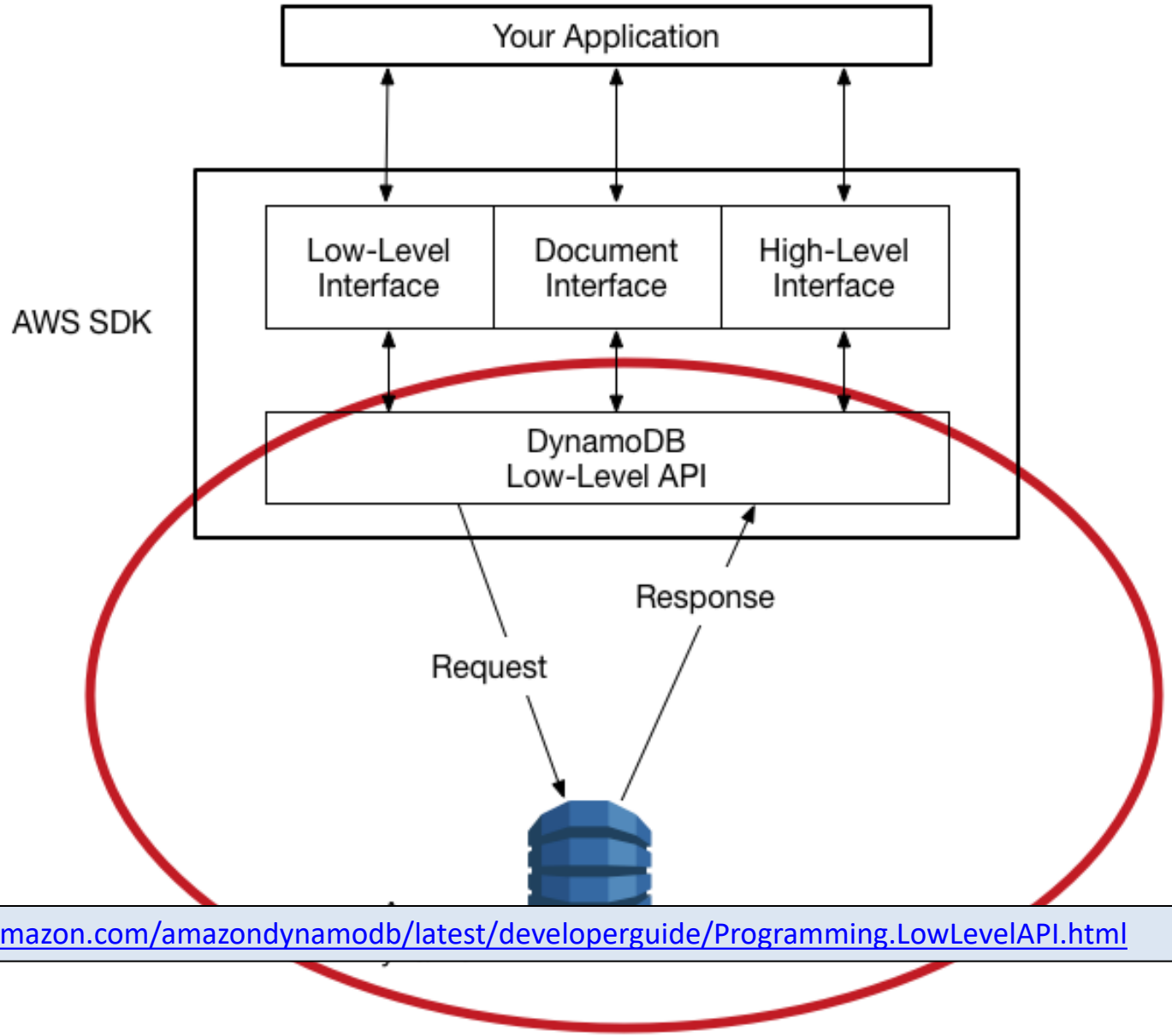
- PUT operations (Primary Key based)
  - Parameters: <Key, Value> pair or List of <Key, Value> pairs if bulk load
  - `put-item(K,V), batch-write-item (LIST(<K-V>, Update-Item (K, update-instructions)`
- GET operations (Primary Key based)
  - Parameters: Key (values for Partition Key and Sort Key) or Set of Keys.
  - Returns: Data Item object, or List of Data Items
  - `get-item (KEY)`, `batch-get-item (KEY-LIST)`

Dynamo Operations

- QUERY (Partition Key based)
  - Parameters: Partition Key
  - Returns: Data Item object, or List of Data Items matching the specified Partition Key value.
  - Result can be further Filtered
- SCAN
  - Scans full table.
  - Returns: List of all Data Items.
  - Result can be further Filtered.

Dynamo DB Programming Interfaces

Dynamo DB Programming Interfaces



https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Programming.LowLevelAPI.html

Dynamo DB Programming Interfaces

- Dynamo DB provides three Programming Interfaces:
  - Low-Level Interfaces
    - Exposes same set of operation discussed at console interface: create-table, put-item, get-item, query, scan, etc.
  - Document Interfaces
    - Table is exposed as a “document”
  - Object Persistence Interface
    - Higher level persistence abstraction for dynamo DB (seems somewhat inspired from JPA)

Low-Level Interfaces (Java)

- Package: `com.amazonaws.services.dynamodbv2.model`
- Indicative list of related java classes:
  - `AttributeDefinition, AttributeValue`
  - `CreateTableRequest, DescribeTableRequest, TableDescription`
  - `KeySchemaElement, KeyType`
  - `PutItemRequest, PutItemResult`
  - `GetItemRequest, GetItemResult`
  - `ScanRequest, ScanResult`

Low-Level Interfaces (Java)

Sample Code:

```
HashMap<String, AttributeValue> key = new HashMap<String, AttributeValue>();
key.put("Artist", new AttributeValue().withS("No One You Know"));
key.put("SongTitle", new AttributeValue().withS("Call Me Today"));

GetItemRequest request = new GetItemRequest()
    .withTableName("Music")
    .withKey(key);

try {
    GetItemResult result = client.getItem(request);
    if (result && result.getItem() != null) {
        AttributeValue year = result.getItem().get("Year");
        System.out.println("The song was released in " + year.getN());
    } else {
        System.out.println("No matching song was found");
    }
```

Document Interfaces

- Document interface allows us performing CRUD operations on tables.
- With a document interface, require less work around “types” and “conversions”. Makes dynamo db accessing some what simpler.
- Document interfaces are available in for Java, .NET, and Node.js
- Here is an indicative example code in Java

https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Programming.LowLevelAPI.html

Document Interfaces (Java)

Related Package: `com.amazonaws.services.dynamodbv2.document`, and main class here is `DynamoDB`.

All operations are performed through this object.

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

Table table = docClient.getTable("Music");
GetItemOutcome outcome = table.getItemOutcome(
        "Artist", "No One You Know",
        "SongTitle", "Call Me Today");

int year = outcome.getItem().getInt("Year");
System.out.println("The song was released in " + year);
```

https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Programming.LowLevelAPI.html

Document Interfaces (Java)

- Get-Item

```
Table table = dynamoDB.getTable(tableName);
Item item = table.getItem("Id", 120, "Id, ISBN, Title, Authors", null);

System.out.println("Printing item after retrieving it....");
System.out.println(item.toJSONPretty());
```

https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/JavaDocumentAPICRUDExample.html

Example: Document Interfaces

- Put-Item

```
Table table = dynamoDB.getTable(tableName);
Item item = new Item().withPrimaryKey("Id", 120).withString("Title", "Book 120 Title")
    .withString("ISBN", "120-1111111111")
    .withStringSet("Authors", new HashSet<String>(Arrays.asList("Author12", "Author22")))
    .withNumber("Price", 20).withString("Dimensions", "8.5x11.0x.75").withNumber("PageCount", 500)
    .withBoolean("InPublication", false).withString("ProductCategory", "Book");
table.putItem(item);
```

https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/JavaDocumentAPICRUDExample.html

Example: Document Interfaces

- List Tables

```
TableCollection<ListTablesResult> tables = dynamoDB.listTables();
Iterator<Table> iterator = tables.iterator();

System.out.println("Listing table names");

while (iterator.hasNext()) {
    Table table = iterator.next();
    System.out.println(table.getTableName());
}
```

https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/JavaDocumentAPITablesExample.html

Object Persistence Interface

- Inspired from technologies like Java Persistence API* (and ADO.Net)
- Here we do not directly perform database manipulation operations, instead we use the concept of “Persistent Objects”
- Persistent objects specify “object” to “database” mappings (similar to Object-Relational Mapping)
- Application programs only interact with and manipulate these “persistent objects”, while underlying system perform all database operations.
- AWS SDKs for Java and .NET only provides this interface. Resource: https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/HigherLevelInterfaces.html

Hibernate is popular JPA implementation

Dynamo Persistent Object (Java)

- Here we create a “persistent object” by annotating its class as Dynamo Table, where
- We provide mappings for
  - Class Name to Table Name
  - “primary key information” about the table and its correspondences in the persistent object.
  - getter/setter methods for reading data from object and populating object data from table.
  - Getter methods are used to read from object while writing object into table, where as
  - Setter methods are used to populate object by data read from the table.

Dynamo Persistent Object (Java)

- Other attribute name in table and their corresponding object data are also specified in the mapping
- All mappings are done using “Java Annotations”
- Java’s main object: https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/DynamoDBMapper.html

Example Persistent Object (Java)

```java
@DynamoDBTable(tableName="Music")
public class MusicItem {
    private String artist;
    private String songTitle;
    private String albumTitle;
    private int year;

    @DynamoDBHashKey(attributeName="Artist")
    public String getArtist() { return artist;}
    public void setArtist(String artist) {this.artist = artist;}

    @DynamoDBRangeKey(attributeName="SongTitle")
    public String getSongTitle() { return songTitle;}
    public void setSongTitle(String songTitle) {this.songTitle = songTitle;}
```

Object to Dynamo table mapping

- Table Name “Music” class name MusicItem
- HASH Key attribute: “Artist”
- RANGE Key attribute: “SongTitle”
- getter/setter methods for object read/write

Example Persistent Object (Java)

```
@DynamoDBHashKey(attributeName="Arti
public String getArtist() { return a
public void setArtist(String artist)

@DynamoDBRangeKey(attributeName="SongTitle")
public String getSongTitle() { return songTitle;}
public void setSongTitle(String songTitle) {this.songTitle = songTitle;}

@DynamoDBAttribute(attributeName = "AlbumTitle")
public String getAlbumTitle() { return albumTitle;}
public void setAlbumTitle(String albumTitle) {this.albumTitle = albumTitle;}

@DynamoDBAttribute(attributeName = "Year")
public int getYear() { return year; }
public void setYear(int year) { this.year = year; }
```

Object to Dynamo table mapping

- More table attributes and corresponding getter/setter methods for object read/write

Dynamo Persistent Object (Java)

- Related Package: `com.amazonaws.services.dynamodbv2.datamodeling`, and following are common classes:
  - `DynamoDBMapper`
  - `DynamoDBTable`
  - `DynamoDBAttribute`
  - `DynamoDBHashKey`
  - `DynamoDBRangeKey`
- DynamoDBMapper is main class. We perform database READ/WRITE operations through mapper

Example Persistent Object (Java)

- Sample Code: PUT Item (save object)

```
DynamoDBMapper mapper = new DynamoDBMapper(client);

//Create PersistentObject, and populate with data
MusicDynamoPO music = new MusicDynamoPO();
music.setArtist("Artist2");
music.setSongTitle("TitleC");
music.setYear(1971);
//Save the object in the repository
//Note: We do not do anything like PutItem
mapper.save(music);
```

Example Persistent Object (Java)

- Sample Code: GET Item (read object)

```
//Create Key, of same object type, and
//    put search key values into this
MusicDynamoPO key = new MusicDynamoPO();
key.setArtist("Artist2");
key.setSongTitle("TitleC");

try {
    //perform GET-ITEM by load method of mapper
    MusicDynamoPO result = mapper.load( key );
    if (result != null) {
        System.out.println("The song was released in "+ result.getYear());
    } else {
        System.out.println("No matching song was found");
```

Example Persistent Object (Java)

- Sample Code: execute a query

```
//List all titles from Metallica
MusicDynamoPO partitionKey = new MusicDynamoPO();
partitionKey.setArtist("Metallica");
DynamoDBQueryExpression<MusicDynamoPO> queryExpression
    = new DynamoDBQueryExpression<MusicDynamoPO>()
    .withHashKeyValues(partitionKey);

List<MusicDynamoPO> itemList
    = mapper.query(MusicDynamoPO.class, queryExpression);

System.out.println("Titles from Metallica:");
for (int i = 0; i < itemList.size(); i++)
    System.out.println(itemList.get(i).getSongTitle());
```