Who Am I?

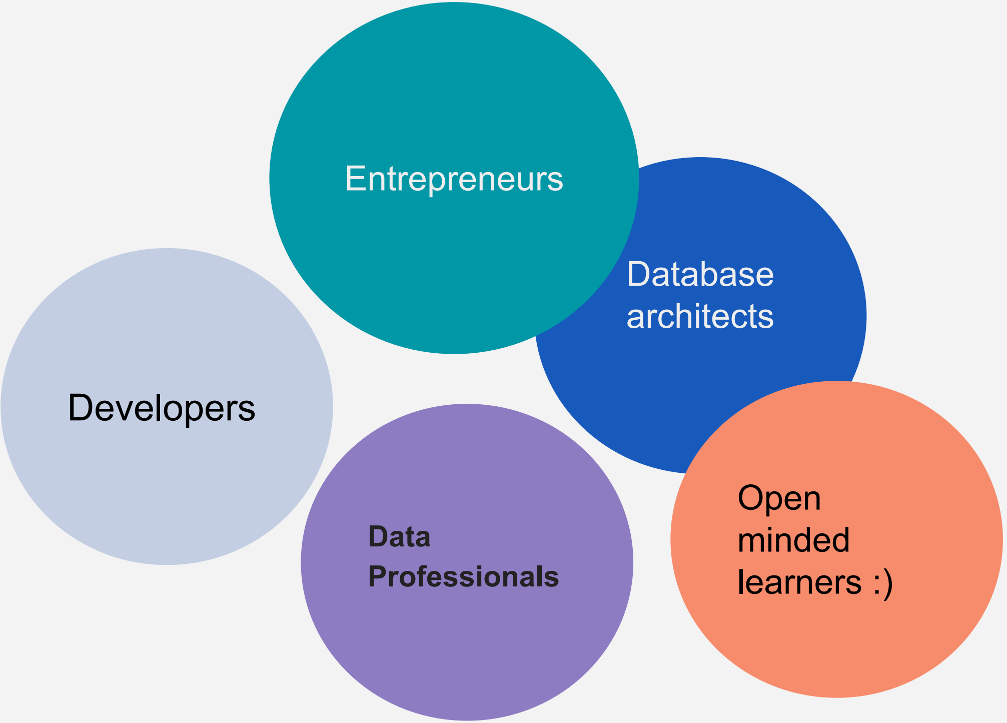
Paulo Dichone

Software, Cloud, AI Engineer
and Instructor

What Is This Course About?

- AWS DynamoDB
    - Fundamentals of DynamoDB
    - Create DynamoDB tables and modifying their contents using the AWS CLI and AWS SDK
    - Optimization and Best Practices

Who Is This Course For



Entrepreneurs

Database architects

Developers

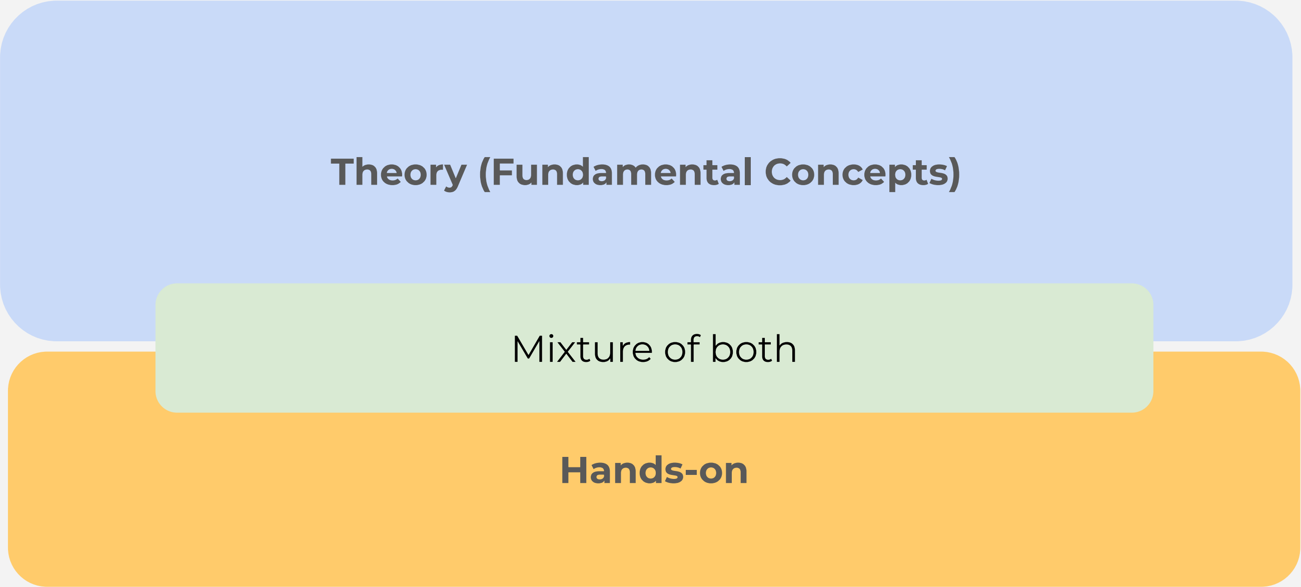
Data Professionals

Open minded learners :)

Course Prerequisites

1. Know programming (highly *preferred… at least the basics)*
    a. *There will be code*
2. Familiar with AWS Services
3. Know database fundamentals
4. *This is not a programming course*
5. Willingness to learn :)

Course Structure



Theory (Fundamental Concepts)

Mixture of both

Hands-on

Development Environment setup

- Python
- VS Code (or any other code editor)
- AWS Account

AWS Account & Services

*** **Please note** that you will need to have an AWS account, and there may be some costs associated with using DynamoDB.*

Dev Environment Setup

Python (Win, Mac, Linux)

https://kinsta.com/knowledgebase/install-python/

AWS Management Console

Hands-on - Create a simple DynamoDB table

Windows - line continuation ^

^ - ***caret*** - used as a line continuation character, allowing long commands to be broken into multiple lines for better readability

DynamoDB Deep Dive

- What is it?
- Why (motivation)?
- Advantages
- Key concepts
- Structure
- Components
- Use cases

Amazon DynamoDB

A **serverless**, **NoSQL fully managed** database with **single-digit millisecond** performance at any **scale**

FAST & Scalable

Amazon DynamoDB - Key Features

Fully Managed - AWS handles all the administrative tasks (hardware provisioning, set up, configuration...)

Scalability - scales automatically depending on load and traffic

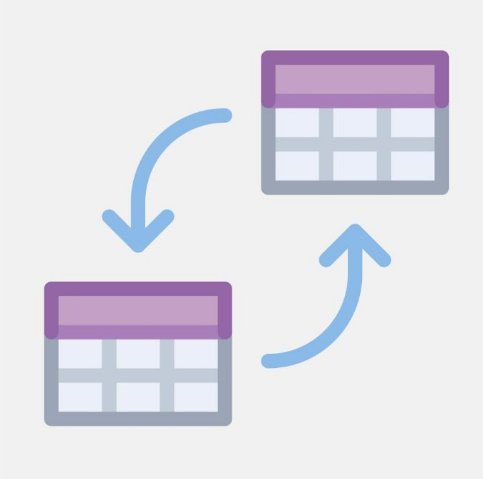
Performance - designed for high performance with single-digit milliseconds response time.

Flexible Data Model - you can store and retrieve any amount of data and serves any level of request traffic

Built-in Security - encryption, identity and access management

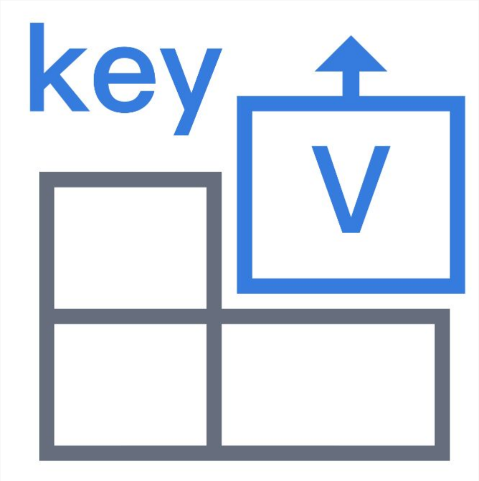
SQL vs NoSQL Database

SQL (Structured Query Language) database - uses a relational model to store data in a structured format, typically using tables. ***The relationships between the data are well-defined.***



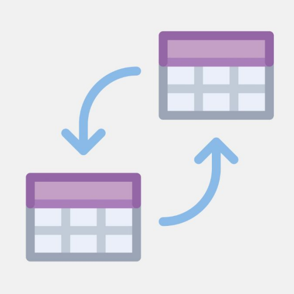
SQL vs NoSQL Database

NoSQL (Not Only SQL) databases - save data in a *non-relational* way. Uses a non-tabular format which makes them flexible, easy to develop and scale

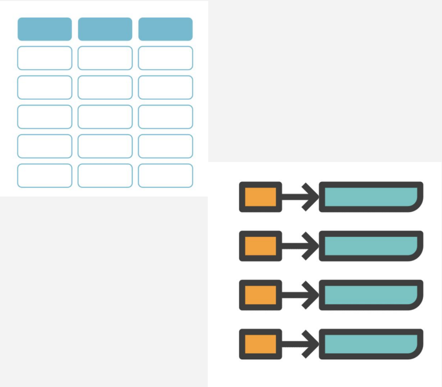


NoSQL vs SQL Database - Key Differences

1. **Data Model**



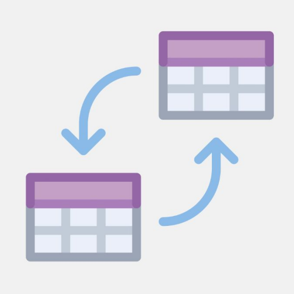
SQL databases - use a structured schema and relational tables - data is organized in rows and columns



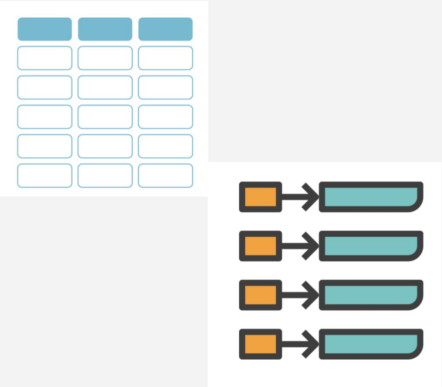
NoSQL databases - use various data models like documents, key-value, graph, column-family… flexible in how data is stored and accessed

NoSQL vs SQL Database - Key Differences

2. Scalability



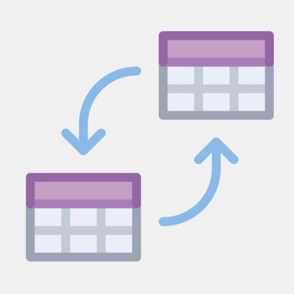
SQL databases - typically scale vertically by increasing the capacity of a single server



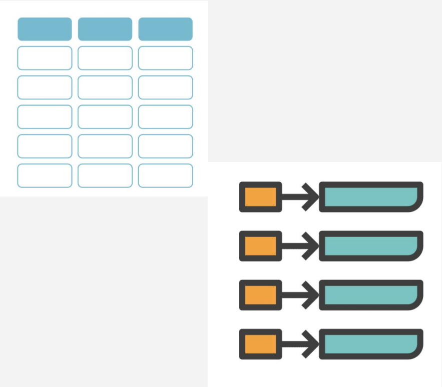
NoSQL databases - scale horizontally by adding more servers to distribute the load

NoSQL vs SQL Database - Key Differences

3. Schema



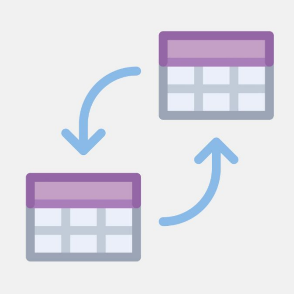
SQL databases - require a pre-defined schema, structure of the tables. Changing the schema is inflexible and time-consuming



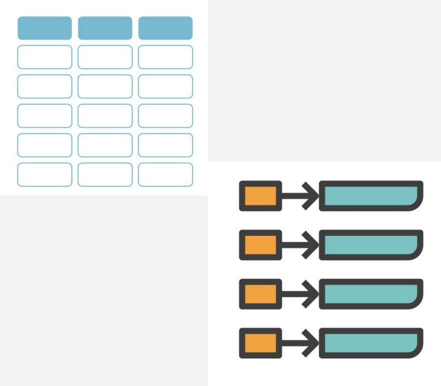
NoSQL databases - schemaless - or have a flexible schema: easy changes to the data structure

NoSQL vs SQL Database - Key Differences

4. Transactions



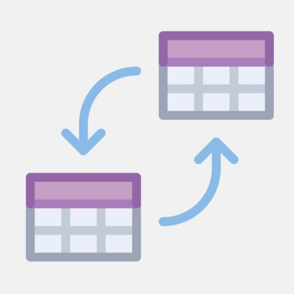
SQL databases - strong support for ACID (Atomicity, Consistency, Isolation, Durability) properties - give reliable transactions



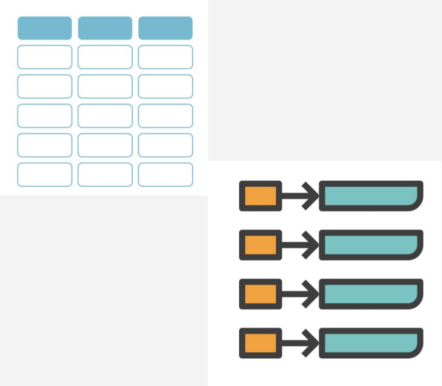
NoSQL databases - some have ACID, but most prioritize availability and partition tolerance over strict consistency

NoSQL vs SQL Database - Key Differences

5. Query Language



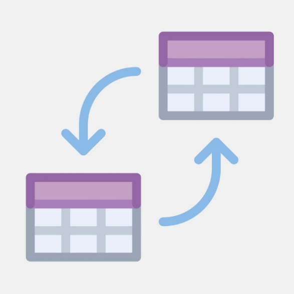
SQL databases - Use SQL as the query language for interacting with data



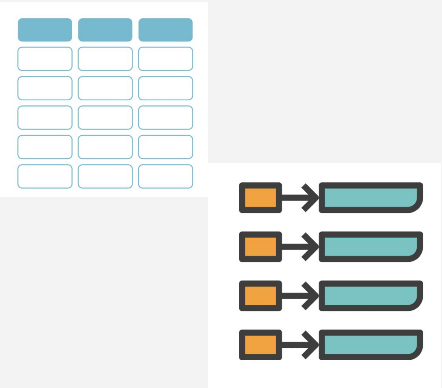
NoSQL databases - use various query languages specific to the data model used.

NoSQL vs SQL Database - Key Differences

6. Use cases



SQL databases - great for applications requiring complex queries, multi-row transactions and high level of data integrity



NoSQL databases - best for large scale applications, distributed data, and flexible data models.

NoSQL vs SQL Database - Key Differences

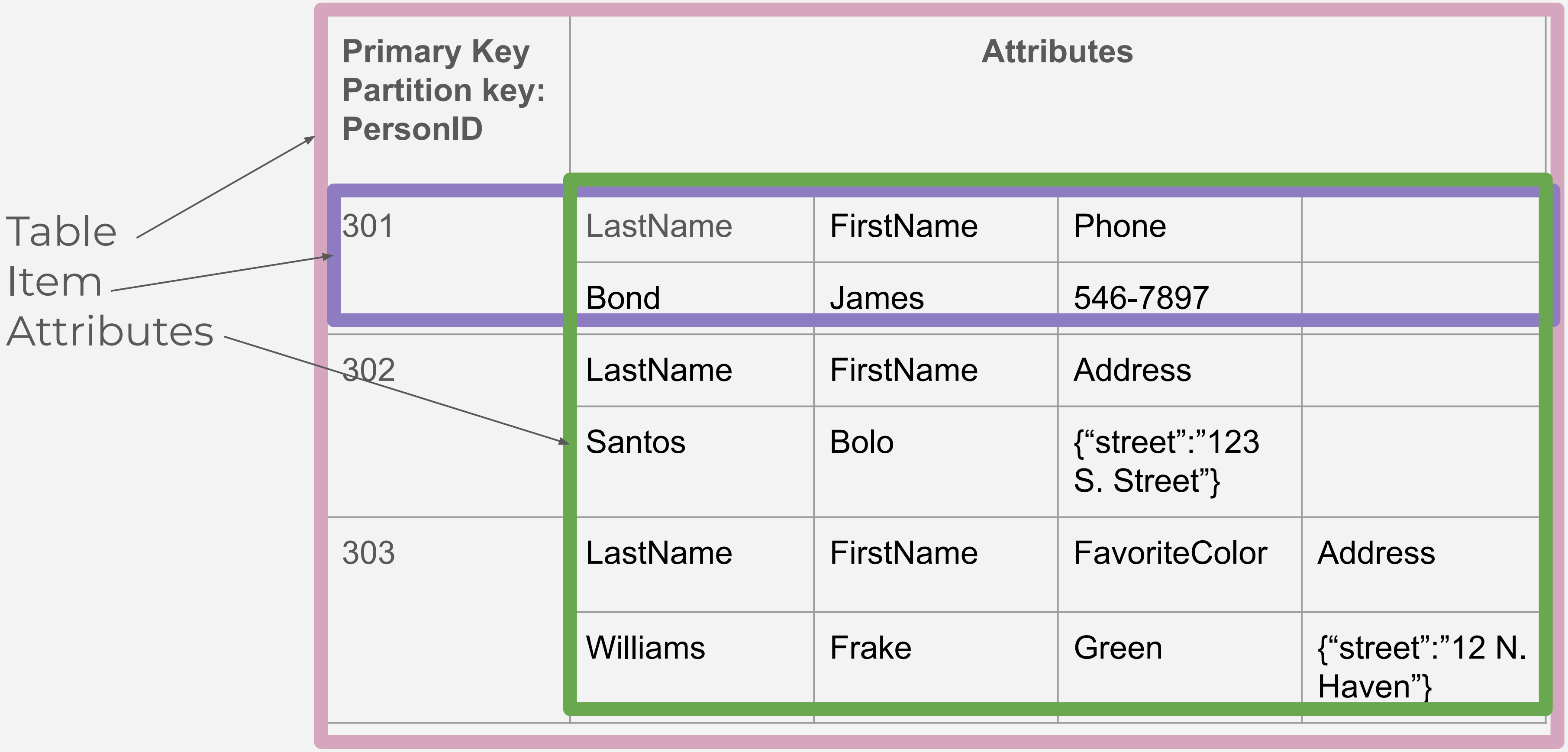
Summary

SQL databases - have structured schemas and prioritize ACID transactions

NoSQL databases - are flexible and offer horizontal scalability

DynamoDB - an AWS-managed NoSQL database service that excels in scalability, performance and flexibility

DynamoDB Key components



Table

Item

Attributes

Primary Key Partition key: PersonID	Attributes			
301	LastName	FirstName	Phone	
	Bond	James	546-7897	
302	LastName	FirstName	Address	
	Santos	Bolo	{“street”:”123 S. Street”}	
303	LastName	FirstName	FavoriteColor	Address
	Williams	Frake	Green	{“street”:”12 N. Haven”}

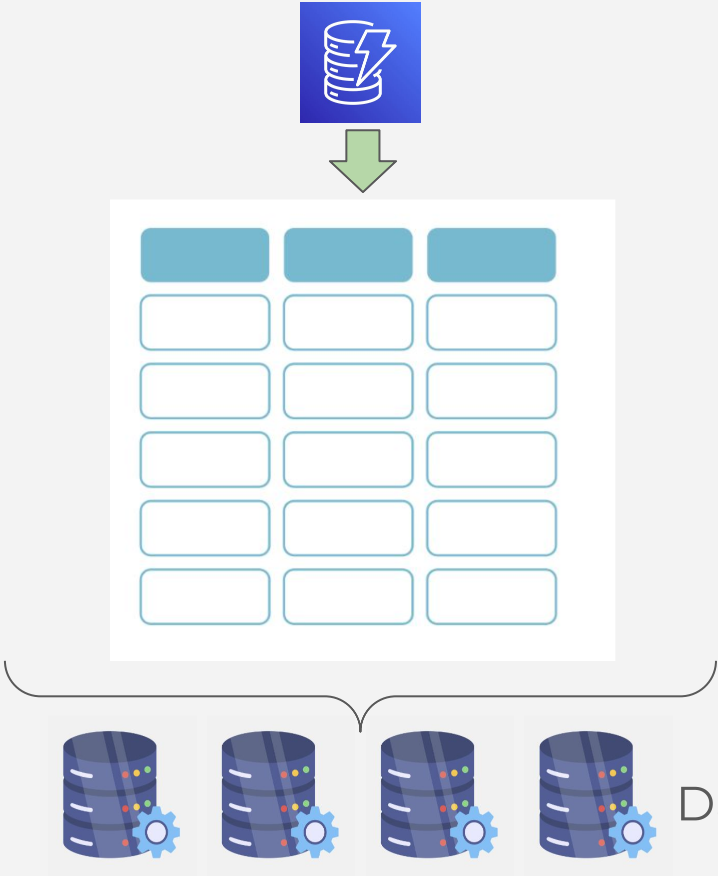
DynamoDB Key components

Primary Key

Unique identifier for each item in a table and it can be:

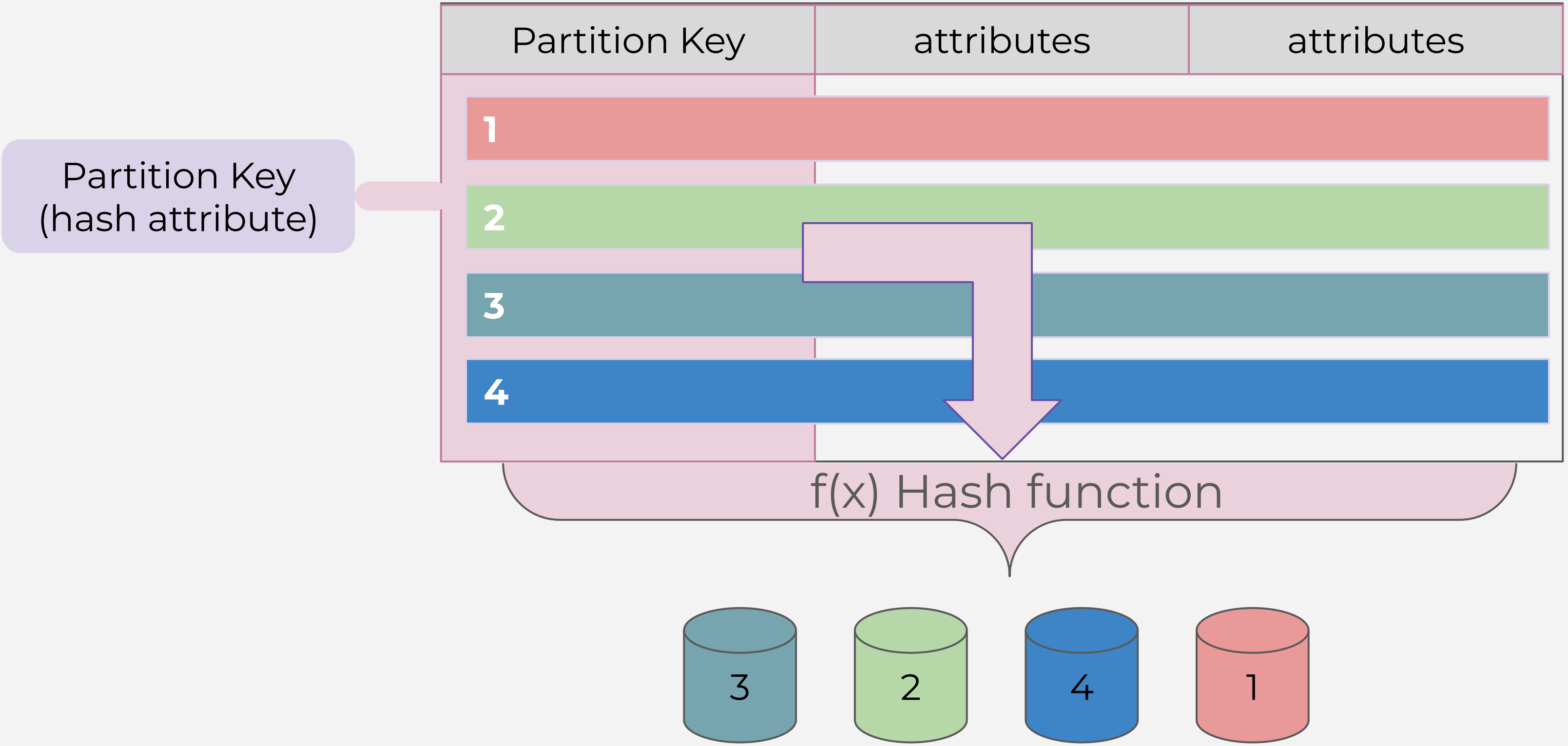
- **Partition Key (Hash Key)**: single attribute used to distribute data across partitions
- **Composite Key (Partition key + Sort Key)**: combination of two attributes that allow for more complex queries and sorting

DynamoDB Partitions



Data storage partitions

DynamoDB Partitions - How they work



Partition Key
(hash attribute)

Partition Key	attributes	attributes
1		
2		
3		
4		

f(x) Hash function

Data storage partitions

DynamoDB Pricing

Charges for **Reading**, **writing**, and **storing** data in your tables, plus other **features** you may choose.

- Provides a Free Tier
  - 25 GB of storage
  - 25 provisioned Write and 25 provisioned Read Capacity Units - enough to handle 200 M requests per month
- For more information: https://aws.amazon.com/dynamodb/pricing/

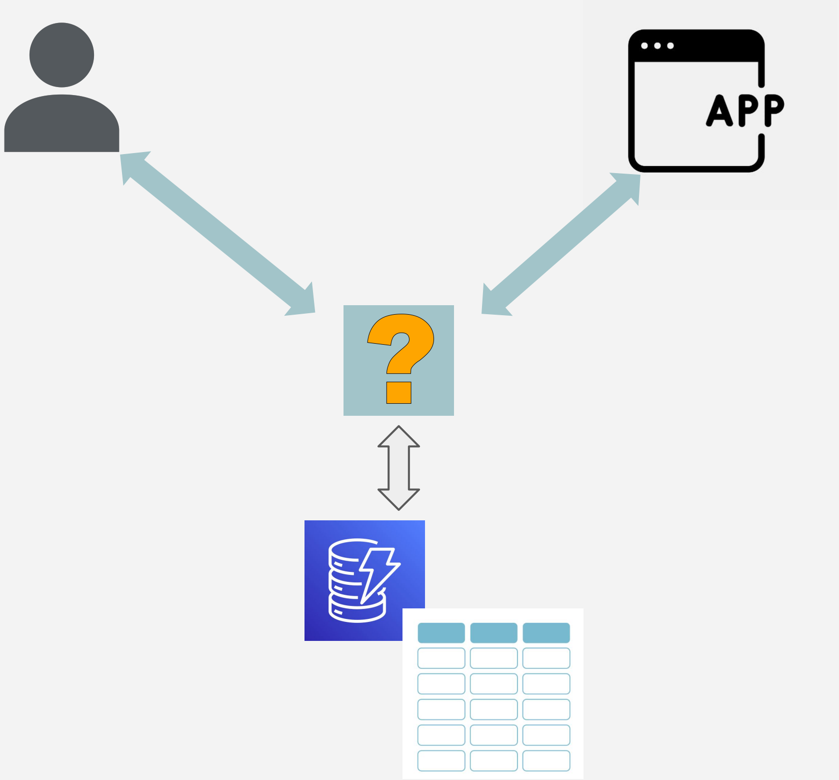
Summary

DynamoDB introduction

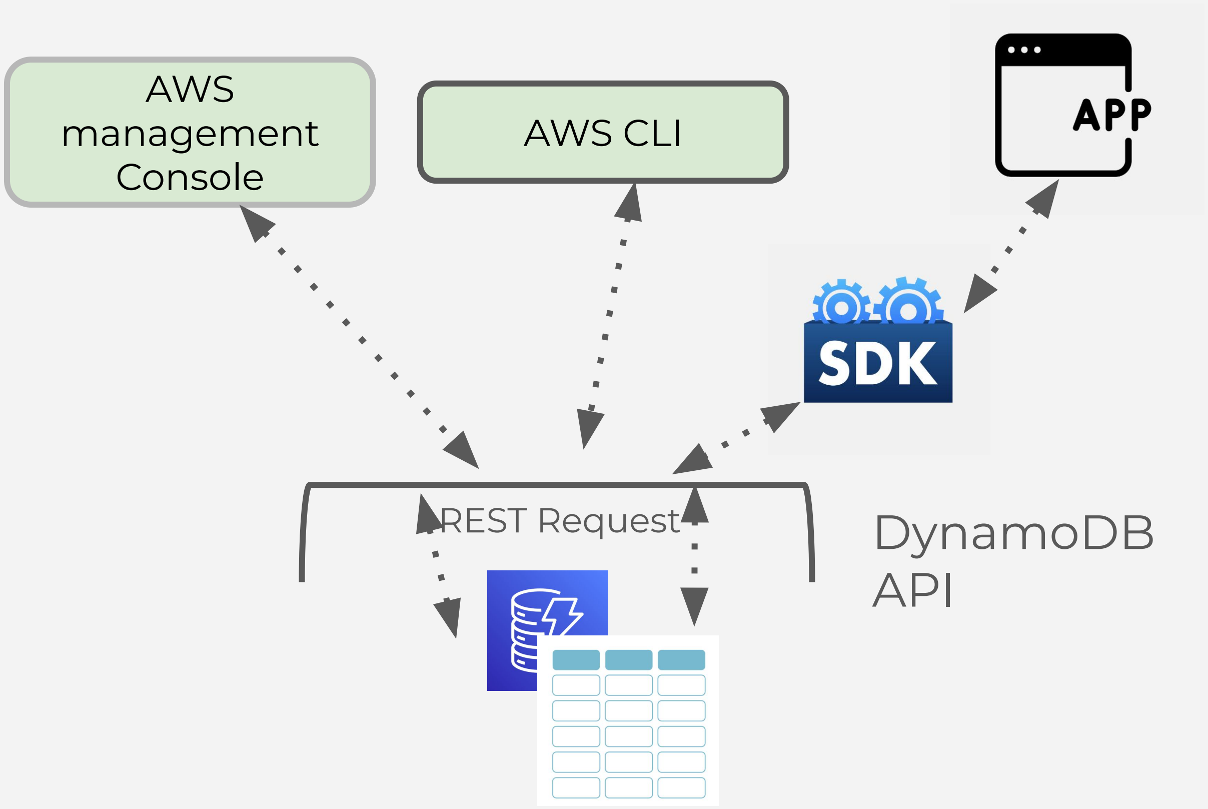
- Create table using the AWS Management Console
- DynamoDB fundamentals
    - Key components and concepts
    - SQL vs NoSQL Databases
    - DynamoDB key Differences
    - Partitions and how they work
    - Partition Keys

Hash Function (*Optional*)

Tools for interacting with DynamoDB



Tools for interacting with DynamoDB



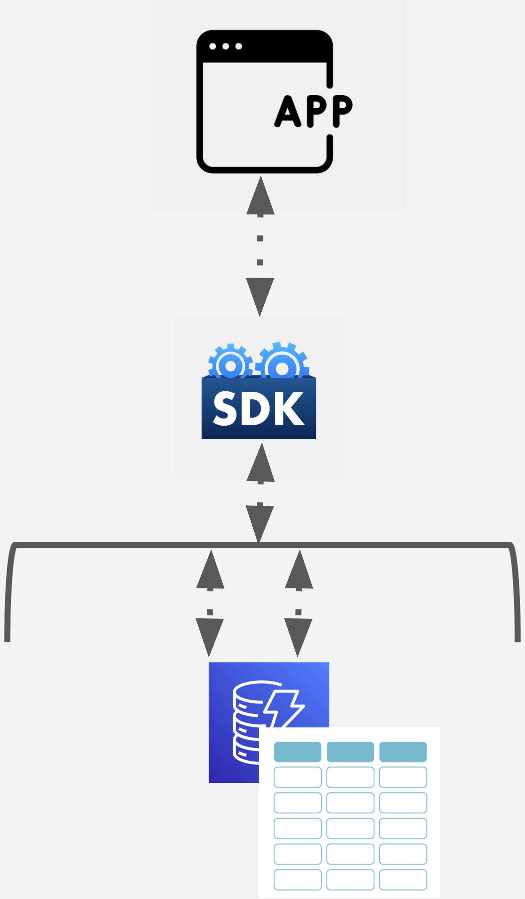
AWS management Console

AWS CLI

REST Request

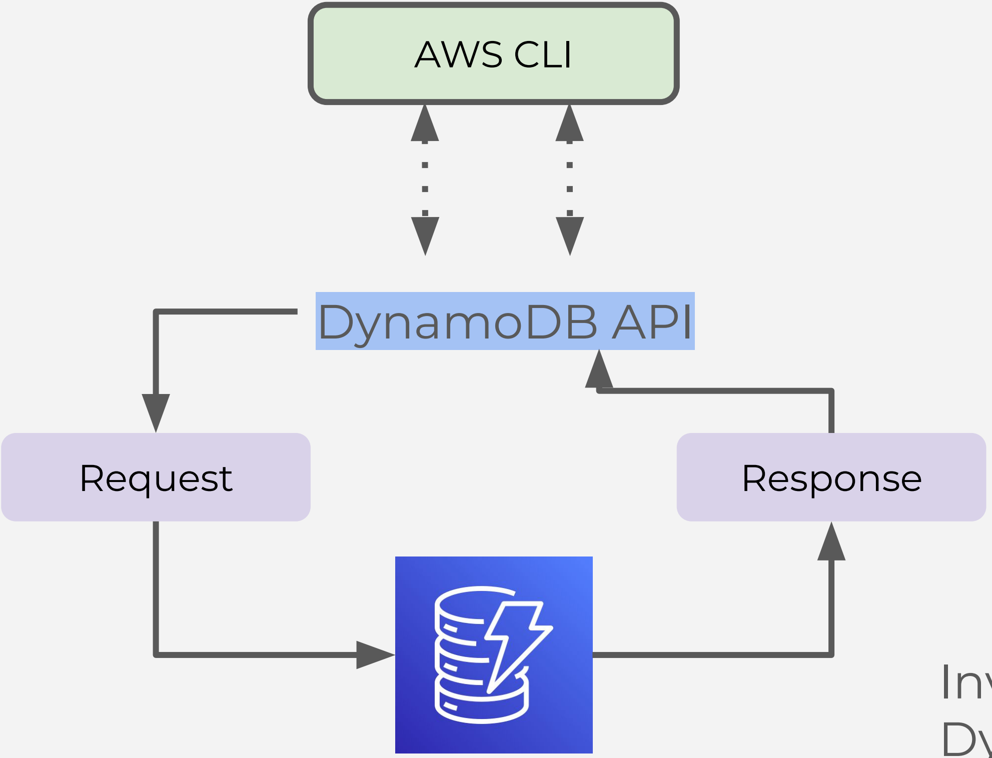
DynamoDB API

Tools for interacting with DynamoDB



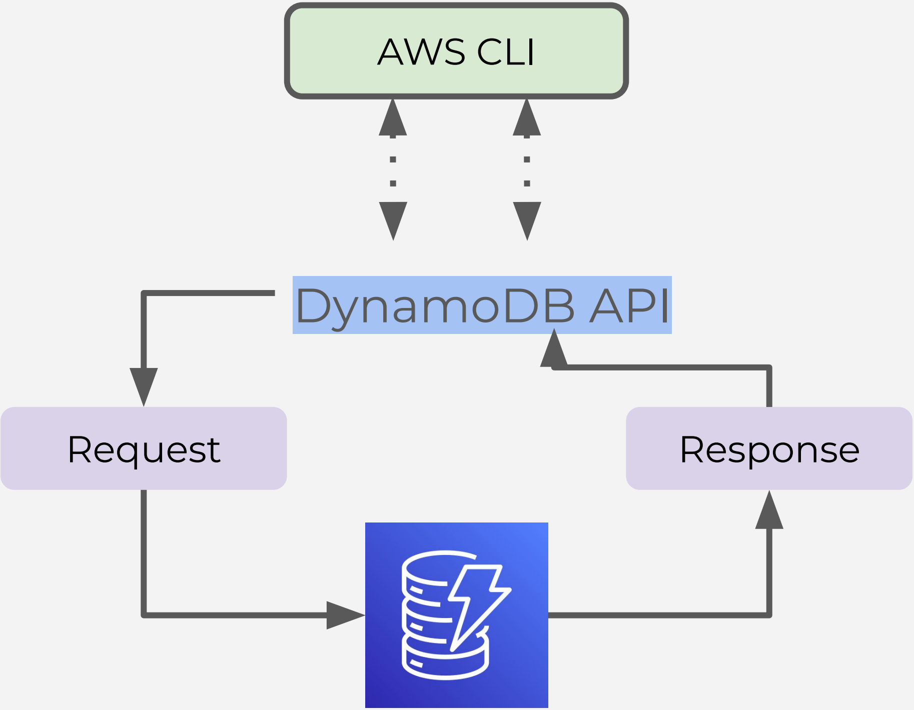
DynamoDB
API

AWS CLI



Invocation of DynamoDB API low-level API Actions

AWS CLI



Request payload

```
{
  "TableName": "MyTable",
  "Item": {
    "Id": {"S": "123"},
    "Name": {"S": "John Doe"},
    "Age": {"N": "30"}
  }
}
```

DynamoDB Set up Options

We have two options when it comes to setting up DynamoDB:

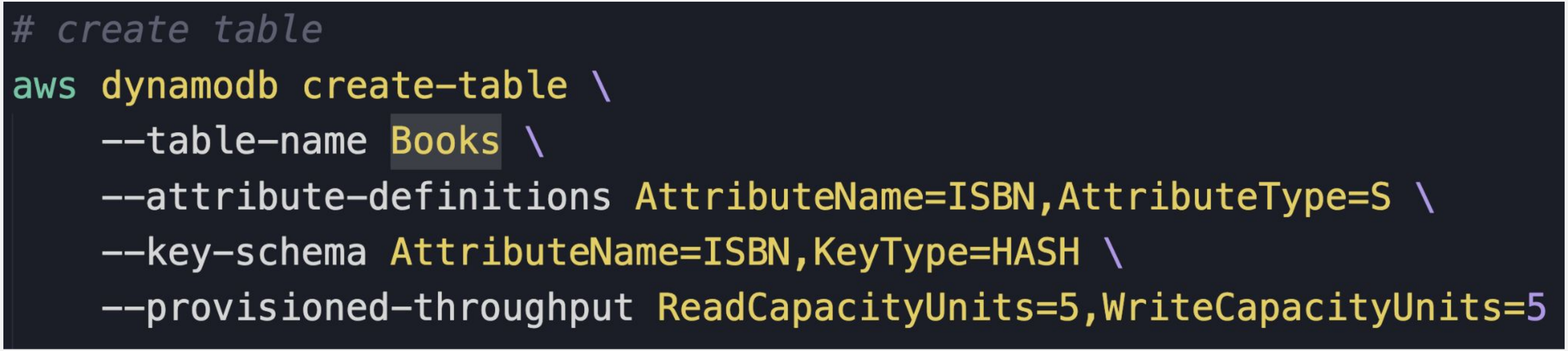
1. Free tier AWS account (you need to create an account)
2. Install DynamoDB locally on your computer
   a. Download locally
   b. Or as Docker image
   c. As an Apache Maven Dependency

Downloadable version is great for developing and testing your code.

AWS CLI - Hands on

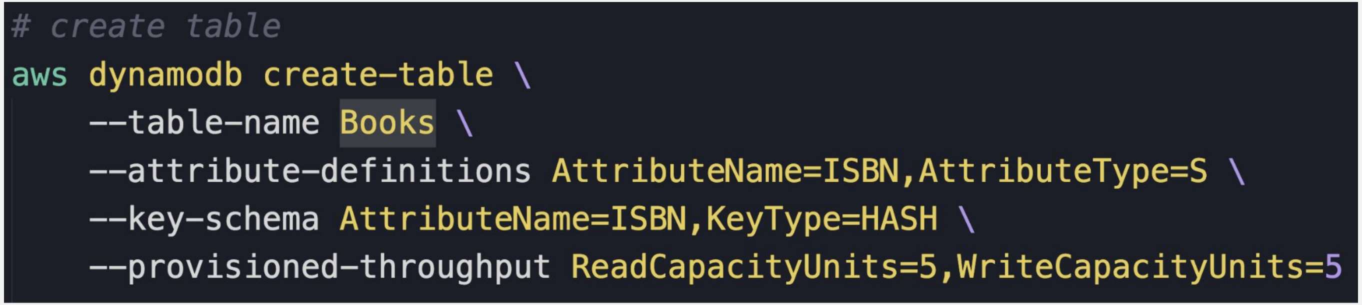
Interacting with DynamoDB API through
AWS CLI - CRUD

AWS CLI - Command breakdown



```
# create table
aws dynamodb create-table \
    --table-name Books \
    --attribute-definitions AttributeName=ISBN,AttributeType=S \
    --key-schema AttributeName=ISBN,KeyType=HASH \
    --provisioned-throughput ReadCapacityUnits=5,WriteCapacityUnits=5
```

DynamoDB Data Types



```
# create table
aws dynamodb create-table \
    --table-name Books \
    --attribute-definitions AttributeName=ISBN,AttributeType=S \
    --key-schema AttributeName=ISBN,KeyType=HASH \
    --provisioned-throughput ReadCapacityUnits=5,WriteCapacityUnits=5
```

Scalar

- Represent exactly one value (nums, string, binary, Boolean and null

Document Types

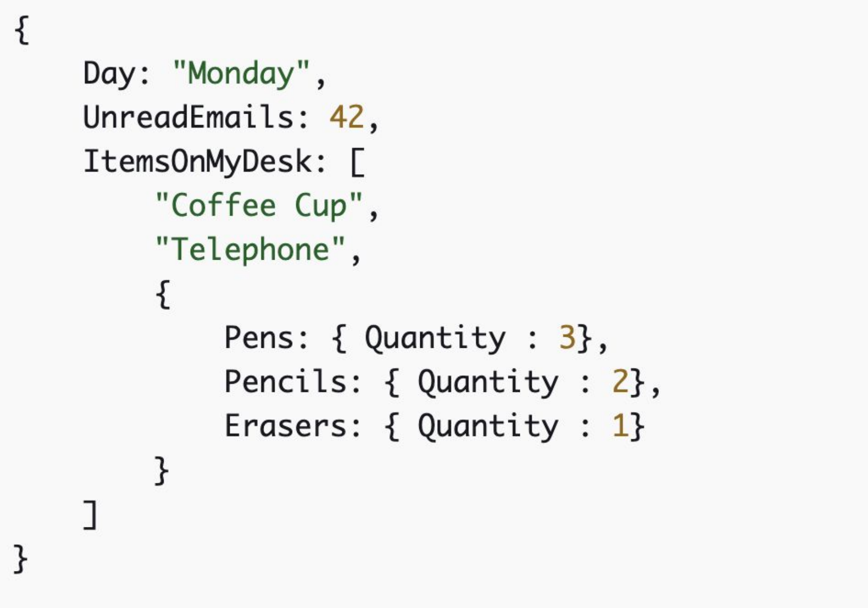
- Represent a complex structure - lists and maps (nested attributes - JSON docs...)

Set Types

- Represent multiple scalar values - string set, and binary set.

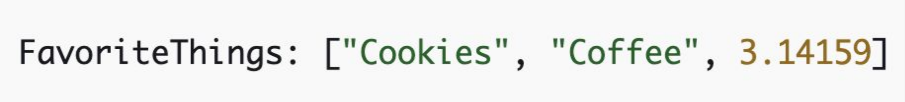
DynamoDB Data Types

Map



```
{
    Day: "Monday",
    UnreadEmails: 42,
    ItemsOnMyDesk: [
        "Coffee Cup",
        "Telephone",
        {
            Pens: { Quantity : 3},
            Pencils: { Quantity : 2},
            Erasers: { Quantity : 1}
        }
    ]
}
```

List



```
FavoriteThings: ["Cookies", "Coffee", 3.14159]
```

DynamoDB Data Types

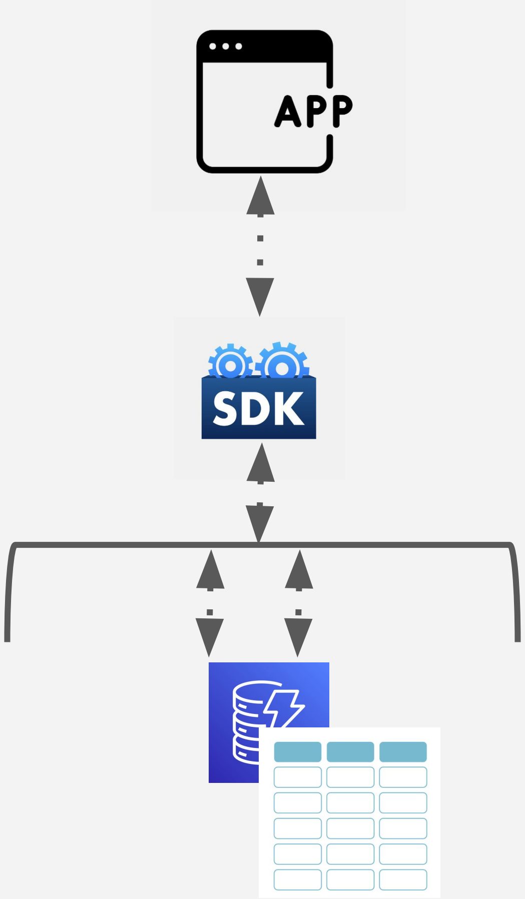
Low-level DynamoDB protocol uses *Data type descriptors* as tokens that signal DynamoDB how to interpret each attribute

- S - String
- N - Number
- B - Binary
- BOOL - Boolean
- NULL - Null
- M - Map
- L - List
- SS - String Set
- NS - Number Set
- BS - Binary Set

Interacting with DynamoDB using the AWS SDK

Tools for interacting with DynamoDB

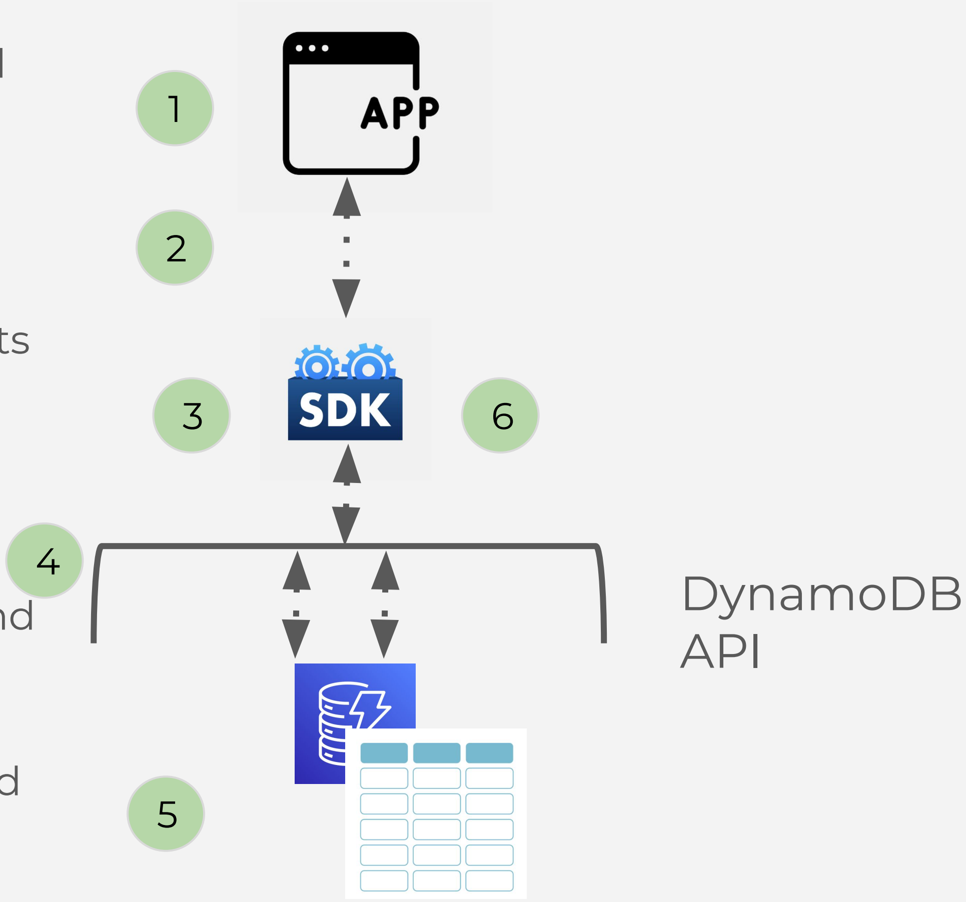
SDK - Software Development Kit



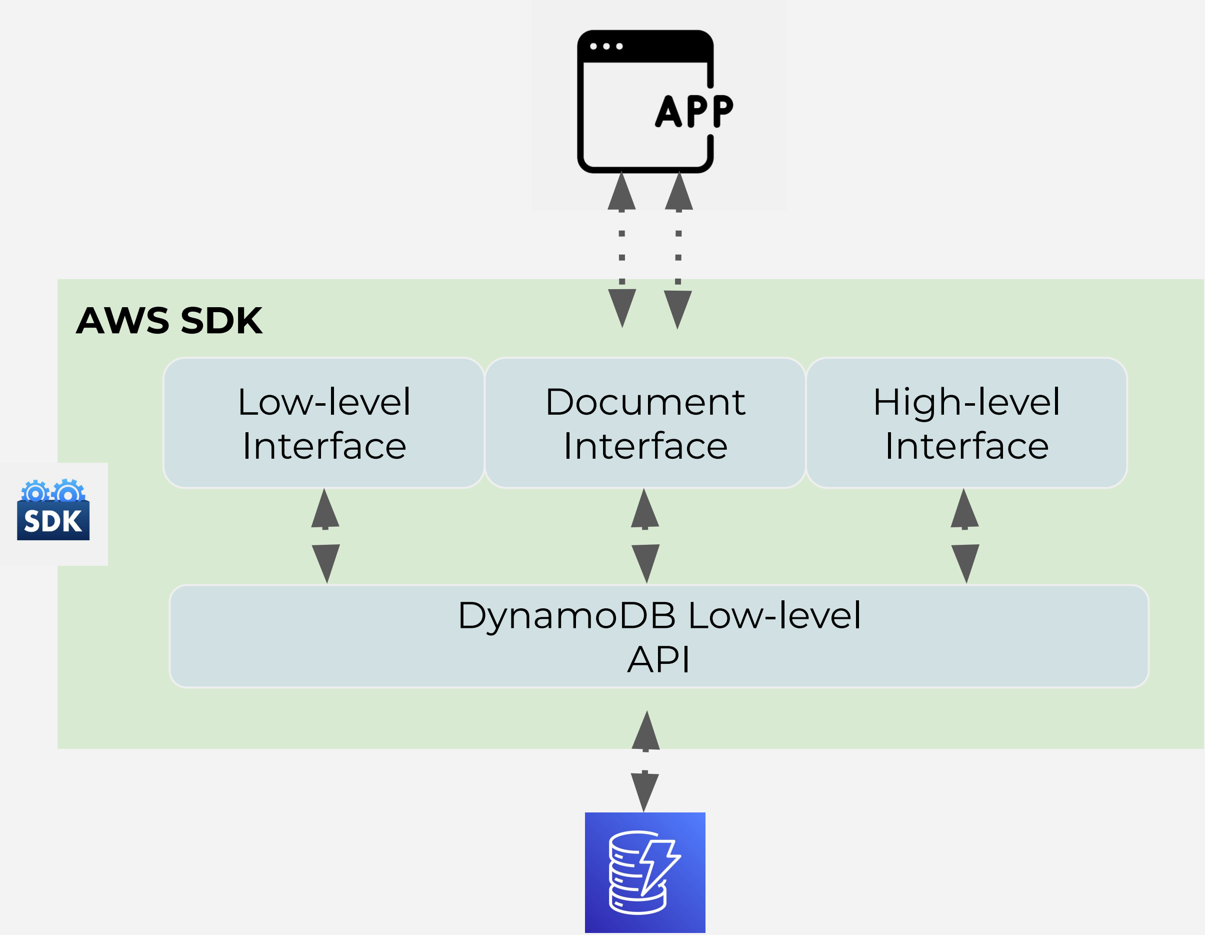
DynamoDB API

Tools for interacting with DynamoDB

1. Write app in AWS SDK provided programming language
2. SDK provides one/more programmatic interfaces for working with DynamoDB
3. SDK constructs HTTP(S) requests for use with the low-level DynamoDB API
4. SDK sends the request to the DynamoDB endpoint
5. DynamoDB runs the request and returns a response, including status code or error message
6. SDK processes the response and propagates it back to your application



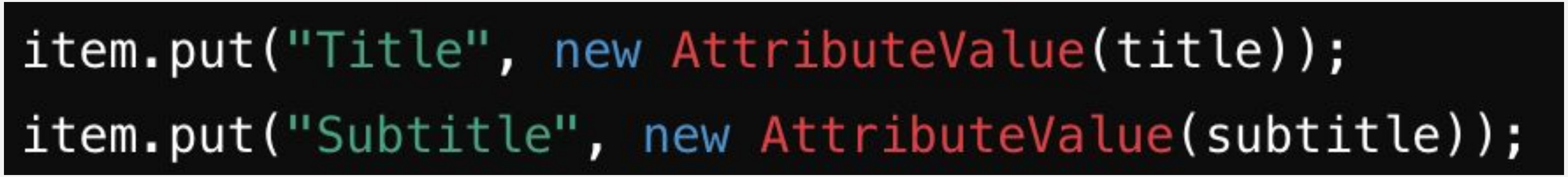
Programmatic Interfaces of the AWS SDK



Programmatic Interfaces of the AWS SDK

Low-level Interface

- Client-side classes and methods corresponding to low-level DynamoDB API
- Available in every language-specific AWS SDK
- Specify data types using the type descriptors (N, S, SS...)
- Write methods to read and write object data to database tables

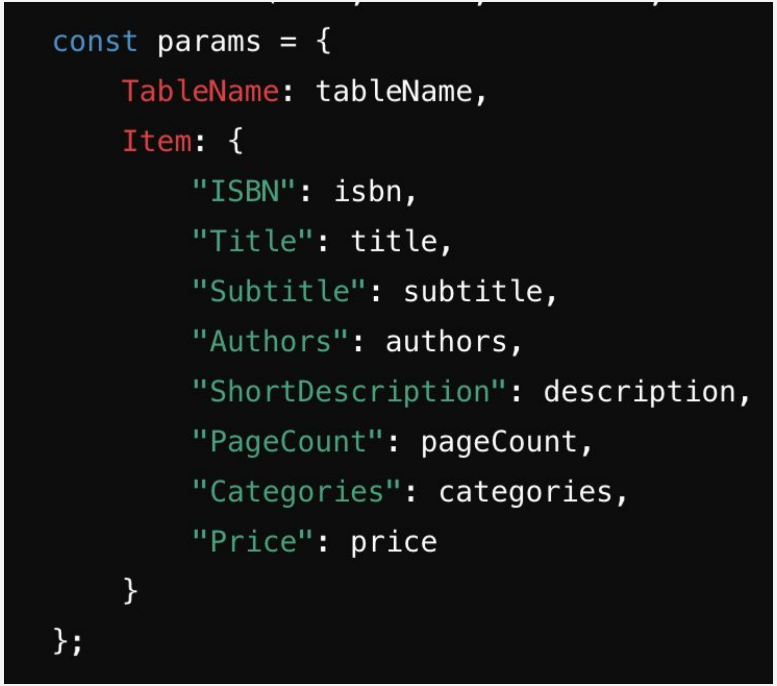


```
item.put("Title", new AttributeValue(title));
item.put("Subtitle", new AttributeValue(subtitle));
```

Programmatic Interfaces of the AWS SDK

Document Interface

- Acts as a wrapper around the low-level DynamoDB API
- Performs data plane operations (create, read, update, delete)
- No need to specify data type descriptors
- Converts JSON documents to and from DynamoDB data types



```
const params = {
    TableName: tableName,
    Item: {
        "ISBN": isbn,
        "Title": title,
        "Subtitle": subtitle,
        "Authors": authors,
        "ShortDescription": description,
        "PageCount": pageCount,
        "Categories": categories,
        "Price": price
    }
};
```

Programmatic Interfaces of the AWS SDK

High-level Interface

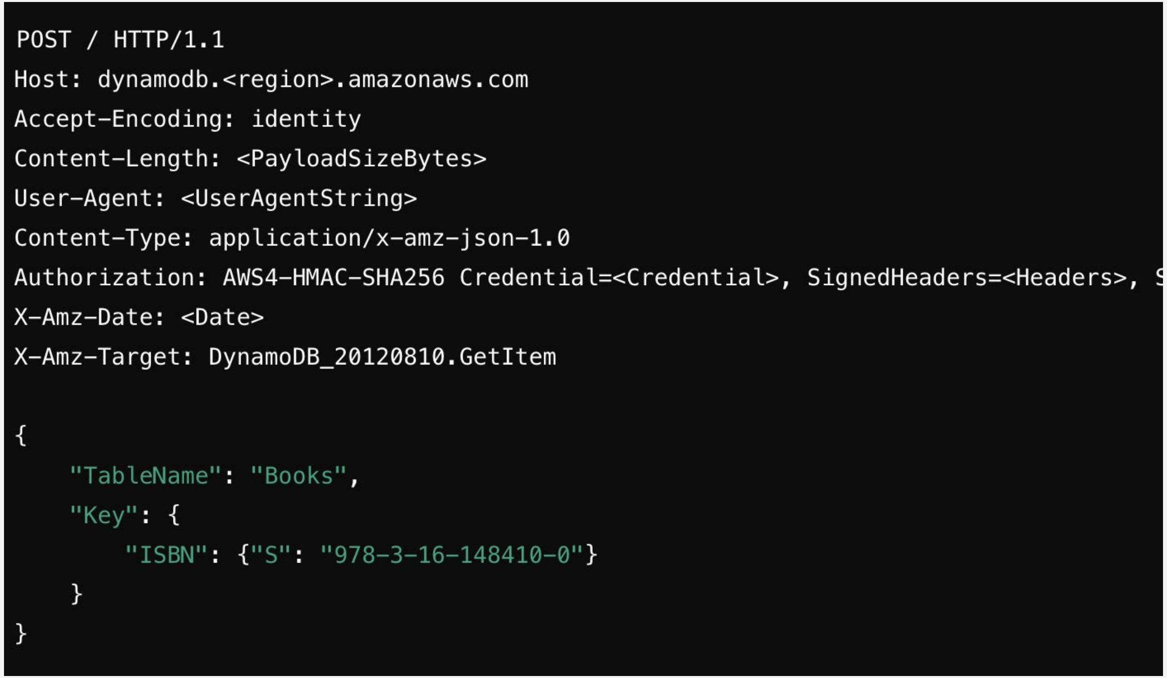
- Provide an object persistence interface
- No need to directly perform data plane operations
- Available for a subset of AWS SDKs (incl. Java and .NET)
- Higher level of abstraction that other interfaces
- Maps the relationships between your coding objects and tables

```
itemRetrieved.setBookRating(“8.9”);
```

Programmatic Interfaces of the AWS SDK

Low-level API (DynamoDB)

- Every HTTP(S) request must be correctly Formatted and with a digital signature
- It uses JSON as a wire protocol

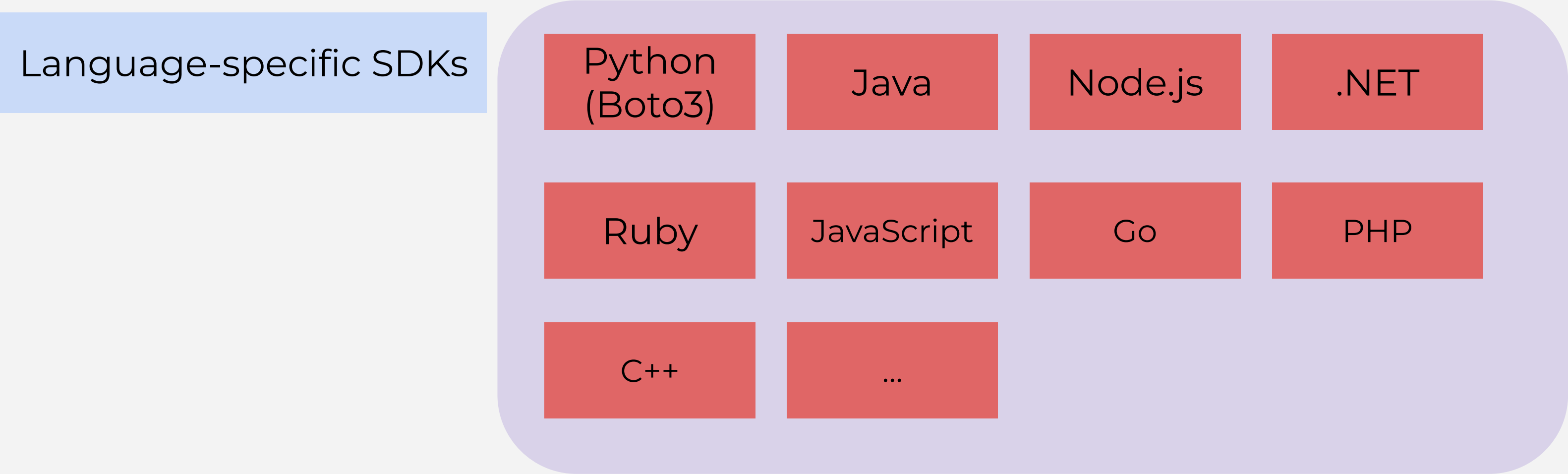


```
POST / HTTP/1.1
Host: dynamodb.<region>.amazonaws.com
Accept-Encoding: identity
Content-Length: <PayloadSizeBytes>
User-Agent: <UserAgentString>
Content-Type: application/x-amz-json-1.0
Authorization: AWS4-HMAC-SHA256 Credential=<Credential>, SignedHeaders=<Headers>, S
X-Amz-Date: <Date>
X-Amz-Target: DynamoDB_20120810.GetItem

{
    "TableName": "Books",
    "Key": {
        "ISBN": {"S": "978-3-16-148410-0"}
    }
}
```

AWS SDKs and tools

AWS provides SDKs to support many common languages



Language-specific SDKs

Python (Boto3)

Java

Node.js

.NET

Ruby

JavaScript

Go

PHP

C++

...

AWS SDKs and tools

AWS provides SDKs to support many common languages

Mobile & IoT

Mobile SDKs:

- Android
- iOS
- React native
- Mobile Web
- Unity
- Xamarin

Mobile Tools:

- AWS Amplify
- AWS AppSync
- AWS Device Farm

IoT Device SDKs:

- Embedded C
- JavaScript, Arduino Yun, Java, Python, C++

AWS SDKs and tools

Run, debug, deploy applications on AWS using AWS Cloud9 IdE or language-specific IDE toolkits:

AWS Toolkits

- Eclipse
- IntelliJ
- PyCharm
- VS Code
- Visual Studio
- Azure DevOps
- Rider...

AWS SDK - Hands on

Interacting with DynamoDB API through AWS SDK (Python)- CRUD

AWS CLI - Hands on

Write multiple items into a DynamoDB table with

batch_writer()

- Stores up to 25 items (or 16 MB) into a DynamoDB table with one single command
- Processes the writes in **parallel** and **reduces** round trips vs. PutItem requests

AWS CLI - Hands on

Write multiple items into a DynamoDB table with

batch_writer()

- Stores up to 25 items (or 16 MB) into a DynamoDB table with one single command
- Processes the writes in **parallel** and **reduces** round trips vs. PutItem requests

Summary

We covered:

- Explain the interfaces that the AWS SDKs provide
- Create a table using the AWS SDKs
- Read Write from the table using the AWS SDKs

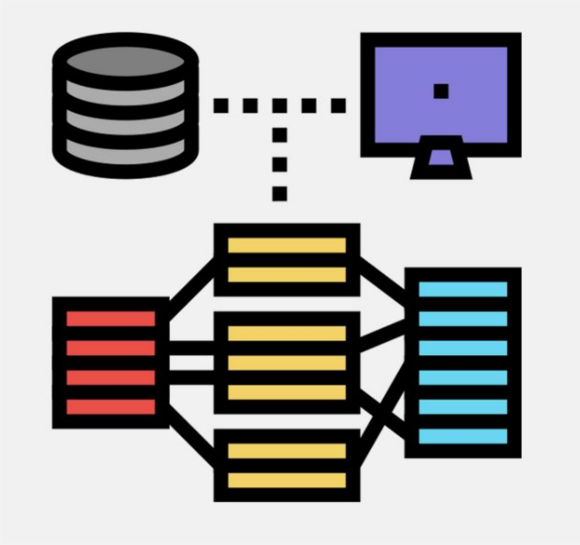
Working with Indexes in DynamoDB

We will cover:

- What are indexes and how to implement them
- Benefits of indexes
- How to structure the indexes for efficiency
- Use cases for queries, scans and indexes

Indexes in databases

Indexes - a tool used in the background to speed up querying.



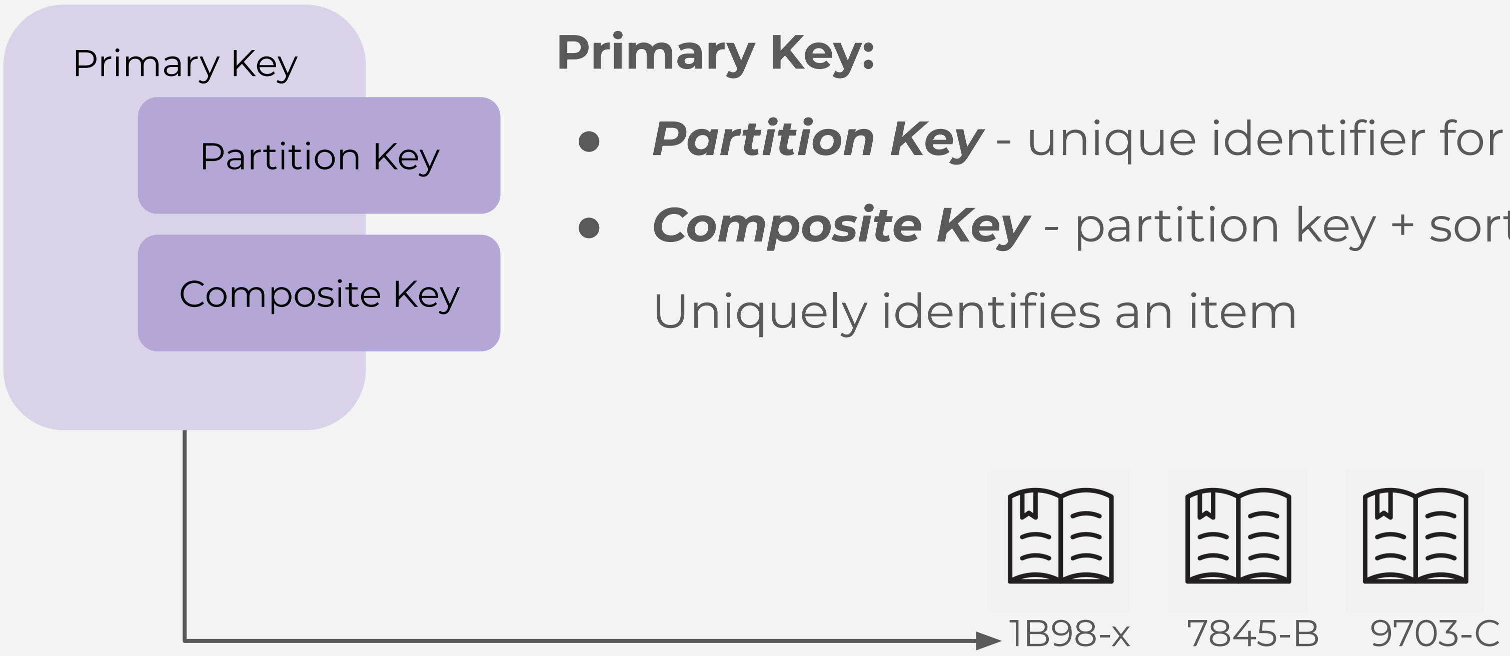
Primary Index (Primary Key)

In DynamoDB

Primary Key:

- ***Partition Key*** - unique identifier for an item
- ***Composite Key*** - partition key + sort key that

  Uniquely identifies an item



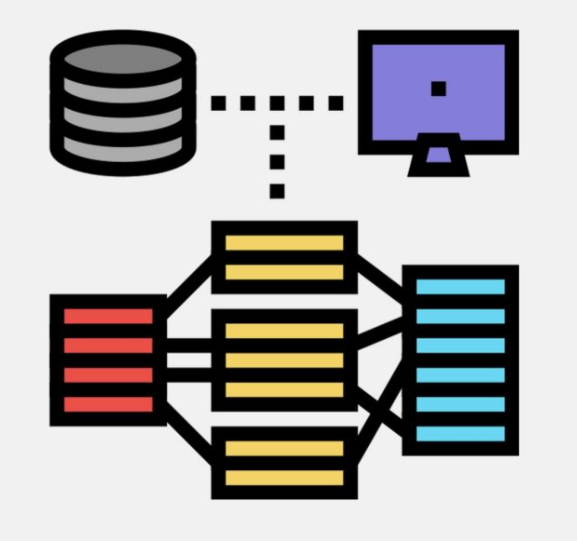
Indexes Motivation

DynamoDB offers different access patterns:

- Scan
- Filter

Problem:

- Scan and filter are expensive - they require examining every item in the table (slow, and resource-intensive and costly)

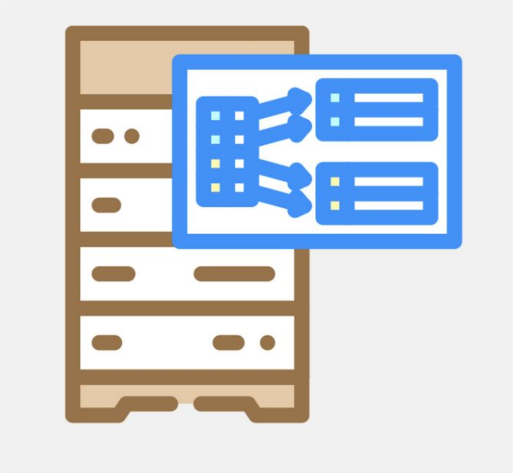


Solution: Indexes

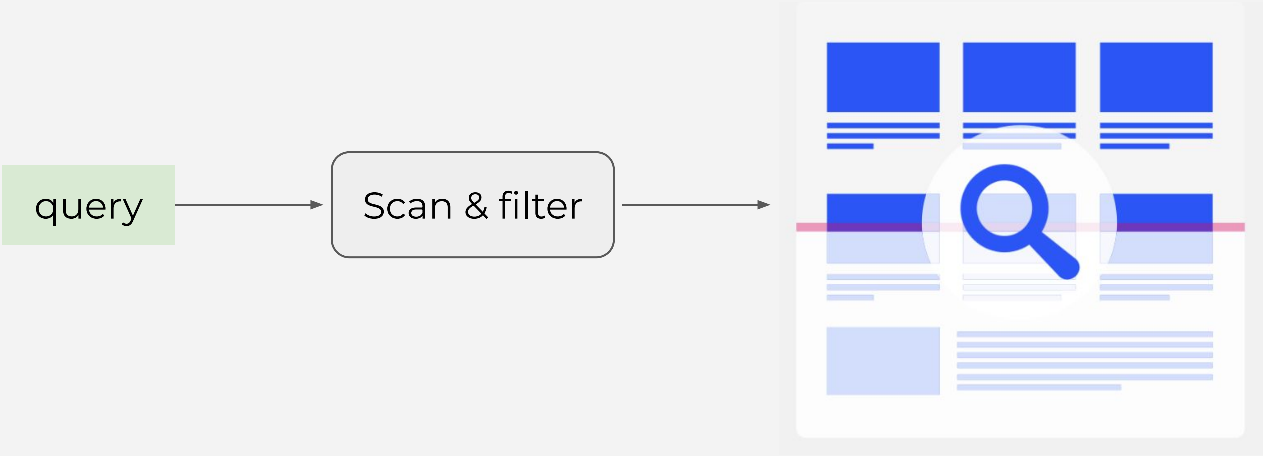
Indexes solve this issue!

Benefits:

- Enable fast and efficient querying directly accessing the data required - no more scanning the whole table
- Reduce cost (reduces read capacity units consumed - cost savings)
- Scalability - maintain performance as table size increases
- Targeted queries - precise queries based on specific attributes



Use case example - E-commerce site



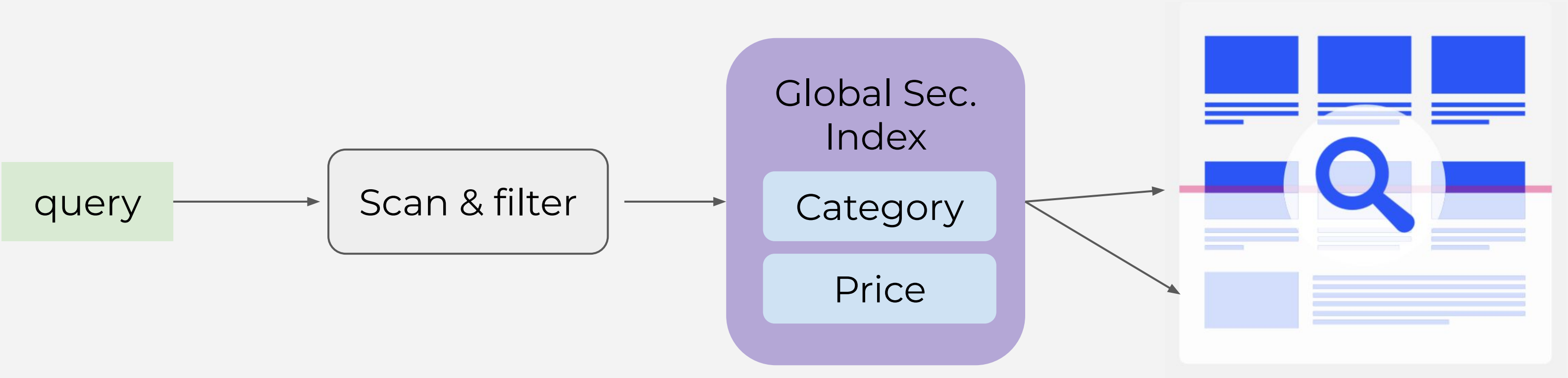
query

Scan & filter

Problem:

- Scan the entire “Product” table (slow and costly)

Use case example - E-commerce site



query

Scan & filter

Global Sec. Index

Category

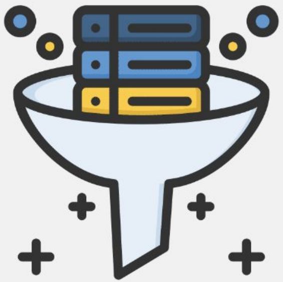
Price

Benefit:

- Query only scans the relevant index (faster and cost-effective)

Finding and fetching data with scan and filter

Once results are scanned, you can refine the results through filter expression which determines which items within the scan results should be returned. The rest is discarded.



A filter expression is applied after a scan finishes, but before the results are Returned; therefore, a scan consumes the same amount of read capacity, regardless of whether a filter expression is present.

Capacity considerations

Read/write capacity mode

DynamoDB has 2 read/write capacity modes for processing reads and writes on tables:

- On-demand
- Provisioned (default, eligible for the AWS Free Tier)

The read/write capacity mode controls how you are charged for read and write throughput and how you manage capacity - read/write capacity can be set upon table creation or later.

Capacity considerations

Read capacity units (RCUs)

One RCU = one consistent read per second (or two eventually consistent reads per second, for an item up to 4 KB in size

Capacity considerations

Write capacity units (WCUs)

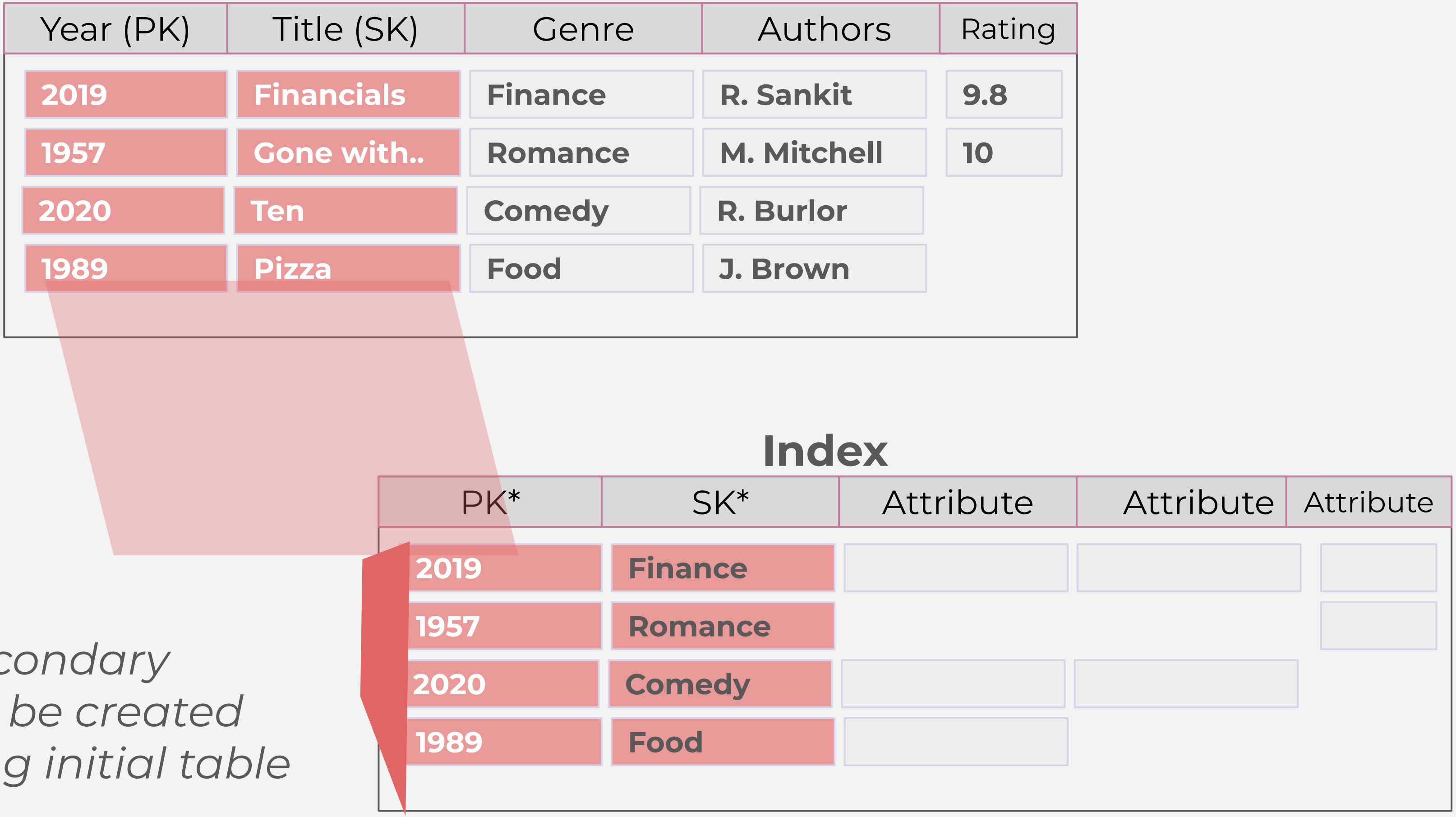
One WCU = one write per second for an item up to 1 KB in size.

If you need to write an item that is larger than 1KB, DynamoDB must consume additional write request units.

Global Secondary Indexes (alternate keys)

Improving querying and costs

Local Secondary Index (LSI)



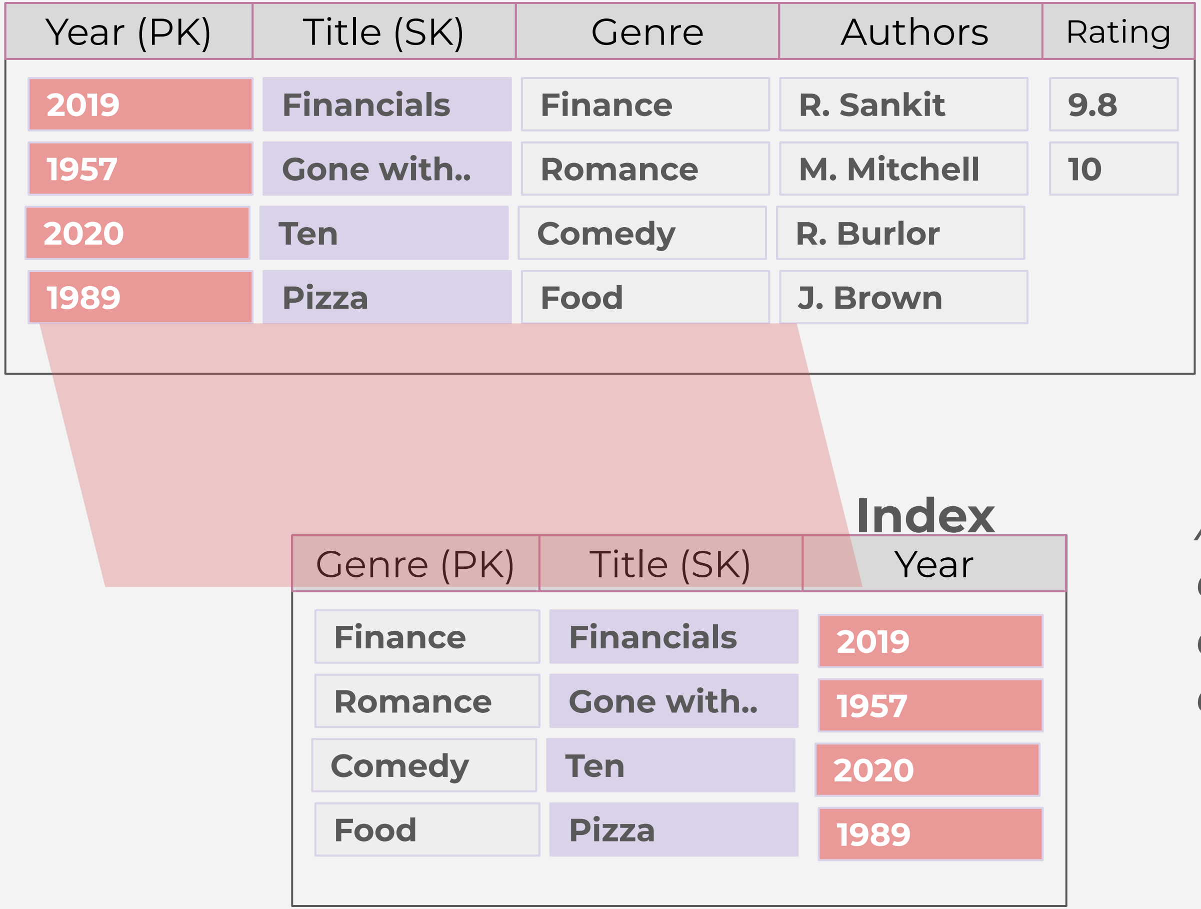
Year (PK)	Title (SK)	Genre	Authors	Rating
2019	Financials	Finance	R. Sankit	9.8
1957	Gone with..	Romance	M. Mitchell	10
2020	Ten	Comedy	R. Burlor	
1989	Pizza	Food	J. Brown	

Index

PK*	SK*	Attribute	Attribute	Attribute
2019	Finance			
1957	Romance			
2020	Comedy			
1989	Food			

A local secondary index can be created only during initial table creation

Global Secondary Index (GSI)



Year (PK)	Title (SK)	Genre	Authors	Rating
2019	Financials	Finance	R. Sankit	9.8
1957	Gone with..	Romance	M. Mitchell	10
2020	Ten	Comedy	R. Burlor	
1989	Pizza	Food	J. Brown	

Index

Genre (PK)	Title (SK)	Year
Finance	Financials	2019
Romance	Gone with..	1957
Comedy	Ten	2020
Food	Pizza	1989

A local secondary index can be created only during initial table creation

Creating global secondary index

The approximate creation time for global secondary index is 5 minutes… however, it depends on several factors

1. Size of the table
2. Number of items in the table that qualify for inclusion in the index
3. Number of attributes projected into the index
4. Provisioned write capacity of the index
5. Write activity on the main table during index builds

DescribeTable - details about the table

```
Aws dynamodb describe-table \
--table-name Books
```

Sparse Indexes in DynamoDB

Movies

Year (PK)	Title (SK)	Genre	Authors	Rating	Award
2019	Financials	Finance	R. Sankit	9.8	award
1957	Gone with..	Romance	M. Mitchell	10	
2020	Ten	Comedy	R. Burlor		
1989	Pizza	Food	J. Brown		award

GSI are sparse by default.

Sparse index

Year (PK)	Title	Genre	Authors	Award
2019	Financials	Finance	R. Sankit	award
1989	Pizza	Food	J. Brown	award

Summary

1. Scanning and filter operations are expensive and inefficient
2. Creating secondary indexes for efficiency and cost reduction
3. Use secondary indexes to avoid elevated charges!

DescribeTable

Deep understanding of the structure of a dynamoDB table

Optimizing Indexes for Efficiency

Design considerations for future table and index creations.

What we’ll cover

- How to structure the base table data for indexes
- How to structure the indexes for efficiency
- How to analyze use cases for queries, scans and indexes

Key Strategies and considerations for index optimization

- Understand access patterns
- Choose the right index type
- Design efficient partition keys
- Optimize sort keys
- Project only necessary attributes
- Use Sparse Indexes
- Monitor and adjust

Use case: Access Patterns for a Books Table

Use case 1: Retrieve Books by Author

Year (PK)	Title (SK)	Genre	Authors	Rating
2019	Financials	Finance	R. Sankit	9.8
1957	Gone with..	Romance	M. Mitchell	10
2020	Ten	Comedy	R. Burlor	
1989	Pizza	Food	J. Brown	

Query requirement: *Retrieve all books written by a specific author, sorted by rating*

Attributes needed: `ISBN`, `Title`, `Authors`, `Year`, `Info`, `Rating`

Query Frequency: High

Use case: Access Patterns for a Books Table

Use case 2: Retrieve Books by Year

Year (PK)	Title (SK)	Genre	Authors	Rating
2019	Financials	Finance	R. Sankit	9.8
1957	Gone with..	Romance	M. Mitchell	10
2020	Ten	Comedy	R. Burlor	
1989	Pizza	Food	J. Brown	

Query requirement: *Retrieve all books written by a specific year, sorted by rating*

Attributes needed: \`ISBN\`, \`Title\`, \`Authors\`, \`Year\`, \`Info\`, \`Rating\`

Query Frequency: Medium

Use case: Access Patterns for a Books Table

Use case 3: Retrieve Award-Winning Books

Year (PK)	Title (SK)	Genre	Authors	Rating	Award
2019	Financials	Finance	R. Sankit	9.8	award
1957	Gone with..	Romance	M. Mitchell	10	
2020	Ten	Comedy	R. Burlor		
1989	Pizza	Food	J. Brown		award

Query requirement: *Retrieve books that have won awards, sorted by authors.*

Attributes needed: `ISBN`, `Title`, `Award`, `Info`, `Authors`

Query Frequency: Medium

Steps to optimize based on access patterns

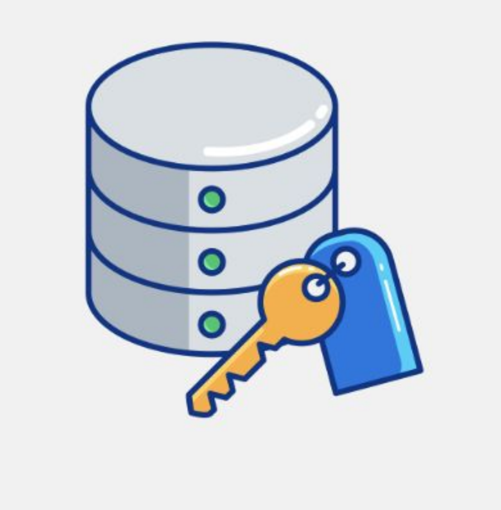
1. **Partition key and sort key design:**
    1. **Primary Key**: ISBN (unique identifier for each book).
    2. **Composite Primary Key**: ISBN (partition key) and Year (sort key) to support querying by year and sorting.
2. Secondary Indexes:
    1. GSI for Author:
        1. Partition Key: Author
        2. Sort Key: Rating
        3. Supports querying books by author and sorting them by rating.
3. LSI for Year and Price:
    1. Partition Key: ISBN
    2. Sort Key: Rating
    3. Supports querying books by year and sorting them by price within the same partition.

Steps to optimize based on access patterns

continuation...

1. Sparse GSI for Award:
   a. Partition Key: Award
   b. Sort Key: Rating
   c. Supports querying award-winning books and sorting them by price.

Partition key design



Designing an effective partition key is crucial for **optimizing** the **performance** and **scalability** of your DynamoDB tables.

Governing principles for Partition Key Design:

- **High Cardinality**
  - Having a large number of unique values for the pk
  - Ensures data is evenly distributed across partitions - no hotspots

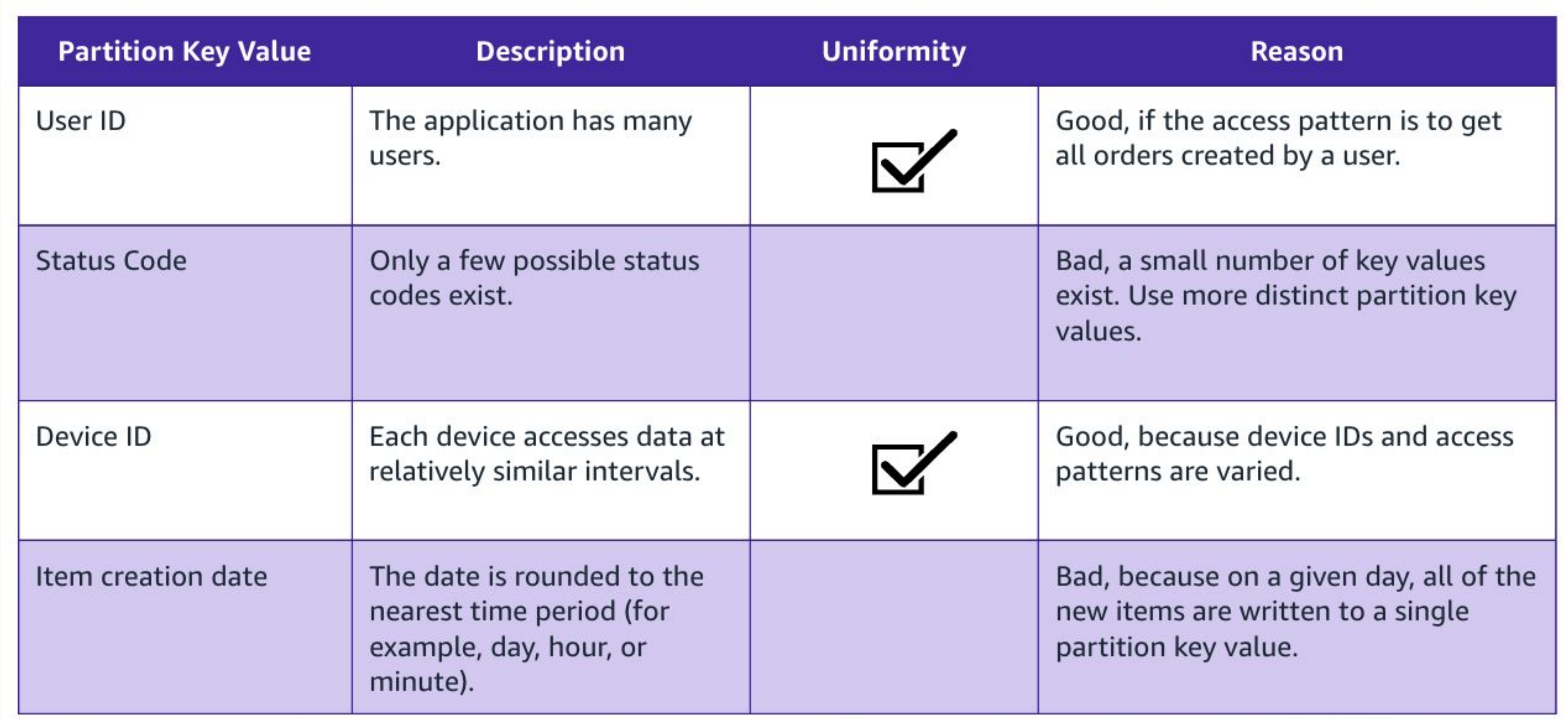
Partition key design

Governing principles for Partition Key Design:

- **Uniform Access Patterns:**
    - Ensure reads and writes are evenly distributed across all partitions
    - Prevents some partitions from over-utilization
- **Avoid Hotspots:**
    - Occur when when a disproportionate number of requests are directed to a single partition
    - These can lead to throttling and degraded performance
        - Avoid using attributes with low cardinality or highly skewed access patterns as partition (e.g., current date)

Partition key design

Example of partition key schema designs considering the business needs and necessary access pattern.



Partition Key Value	Description	Uniformity	Reason
User ID	The application has many users.	☑	Good, if the access pattern is to get all orders created by a user.
Status Code	Only a few possible status codes exist.		Bad, a small number of key values exist. Use more distinct partition key values.
Device ID	Each device accesses data at relatively similar intervals.	☑	Good, because device IDs and access patterns are varied.
Item creation date	The date is rounded to the nearest time period (for example, day, hour, or minute).		Bad, because on a given day, all of the new items are written to a single partition key value.

Image source: aws documentation

Optimizing sort keys

Key considerations:

- Support Range Queries:
  - Should support efficient range queries
  - Example: A *timestamp* sort key allows to query items within a specific time range
- Enable Sorting:
  - Inherently support sorting - retrieve items in asc/desc order
- Composite attributes:
  - Combine multiple attributes into a single sort key to support complex query patterns
  - Example: joining “year” and “month” (YearMonth) for efficient querying by year and month

Secondary Indexes *Structuring*

Structuring secondary indexes for efficiency

Structuring secondary indexes for efficiency

Year (PK)	Title (SK)	Genre	Authors	Rating
2019	Financials	Finance	R. Sankit	9.8
1957	Gone with..	Romance	M. Mitchell	10
2020	Ten	Comedy	R. Burlor	
1989	Pizza	Food	J. Brown	

BookTable Access Pattern	Design considerations
Identify book titles released during a specific year.	The base table design of a partition key and a sort key can be used to identify this information
Identify specific book genres that have been released during a certain year	A global secondary index (GSI) can be created with a partition key of year and sort key of genre
Identify specific book genres that have received high ratings	A GSI can be created with a partition key of genre and sort key of title, with ratings attribute projected

Choose projections carefully

Projections in DynamoDB define the set of attributes are copied from the base table to a secondary index.

Types of projections:

DynamoDB supports three types of projections:

- KEYS_ONLY
  - Only pk and sk are projected into the index
  - Use this when you only need pk attributes in your queries
- INCLUDE:
  - Includes pk attributes plus any specified non-key attributes
  - Good for when you need a subset of attributes from the base table in your index queries
- ALL:
  - All attributes from base table are projected into index
  - Use this when you need all attributes available in your index queries

Choose projections carefully

Consider projecting **fewer attributes** to minimize the size of items written to the index

As long as th index items are small, you can project more attributes at no extra cost.

Avoid projecting attributes that you know will rarely be needed in queries

- Everytime you update an attribute that is projected in an index, you **incur the extra cost** of updating the index as well.

Specify ALL for projected attributes only if you want your queries to return the entire table

  - Projecting ALL attributes eliminate the need for table fetches, **but** in most cases, it **doubles your costs** for storage and write activity

Analyze use cases for...

Queries

The query operation returns all of the items from the table or index with a specific partition key value.

Scans

Return one or more items and item attributes by accessing every item in a table or secondary index.

Secondary indexes

Allow the grouping of alternative attributes that differ from the base table partition key and sort key.

Analyze use cases for...

Year (PK)	Title (SK)	Genre	Authors	Rating
2019	Financials	Finance	R. Sankit	9.8
1957	Gone with..	Romance	M. Mitchell	10
2020	Ten	Comedy	R. Burlor	
1989	Pizza	Food	J. Brown	

Index

Genre (PK)	Title (SK)	Year
Finance	Financials	2019
Romance	Gone with..	1957
Comedy	Ten	2020
Food	Pizza	1989

It's important to always analyze access patterns and business needs to determine the cost vs benefit for any associated secondary indexes.

Use case to be analyzed - Design and access considerations

Scenario:

A developer is using a *ProductCatalog* table. The *ProductCatalog* table includes the following attributes: product ID, product name, category, price, stock quantity, and description. The *ProductCatalog* table uses the partition key product ID.

The developer needs to create a product listing page that features products by category, sorted by price. Should the developer use a query, a scan operation, create a secondary index, or use a combination to meet this business need?

Design considerations

In this use case, the developer should create a Global Secondary Index (GSI) to meet the business need. Since the *ProductCatalog* table uses product ID as the partition key, it does not naturally support queries by category and sorting by price. By creating a GSI with category as the partition key and price as the sort key, the developer can efficiently query products by category and sort them by price.

Use case to be analyzed - Design and access considerations

Scenario:

A developer is using a *MovieCollection* table. The *MovieCollection* table includes the following attributes: movie title, director name, release year, duration, genre, and rating. The *MovieCollection* table uses the partition key release year and the sort key movie title.

The developer needs to create a list of movies released in 1995. Should the developer use a query, a scan operation, create a secondary index, or use a combination to meet this business need?

Design considerations

In this use case, the developer can use the established partition key release year and the sort key movie title to meet the business need. Because the collection of data is associated with the partition key, a Query command can be used to identify all movies released in 1995.

Summary

- How to structure the base table data for indexes
- How to structure the indexes for efficiency
- How to analyze use cases for queries, scans and indexes

Clean-up

- Make sure to delete all AWS resources we created to avoid incurring costs!

Congratulations!

You made it to the end!

- Next steps…

Course Summary

- AWS DynamoDB Deep Dive
  - Fundamentals of DynamoDB
    - Create and interact with DynamoDB tables
  - AWS CLI and AWS SDK for interacting with DynamoDB
  - NoSQL vs SQL databases
  - DynamoDB key concepts and components
    - Partitions, Primary Keys, Sort Keys, Composite keys
    - Global Secondary Indexes
    - Sparse Index
    - Index optimization
    - Partition key design
    - Optimizing sort keys
    - Structuring Secondary Indexes for efficiency
    - Design and access considerations (use cases)
    - ...

Wrap up - Where to Go From Here?

- Keep learning
    - Extend the projects we worked on in this course
    - Design and implement your own DynamoDB tables and optimize them!
- Read more AWS DynamoDB- https://docs.aws.amazon.com/dynamodb/

Thank you!