Dynamo DB

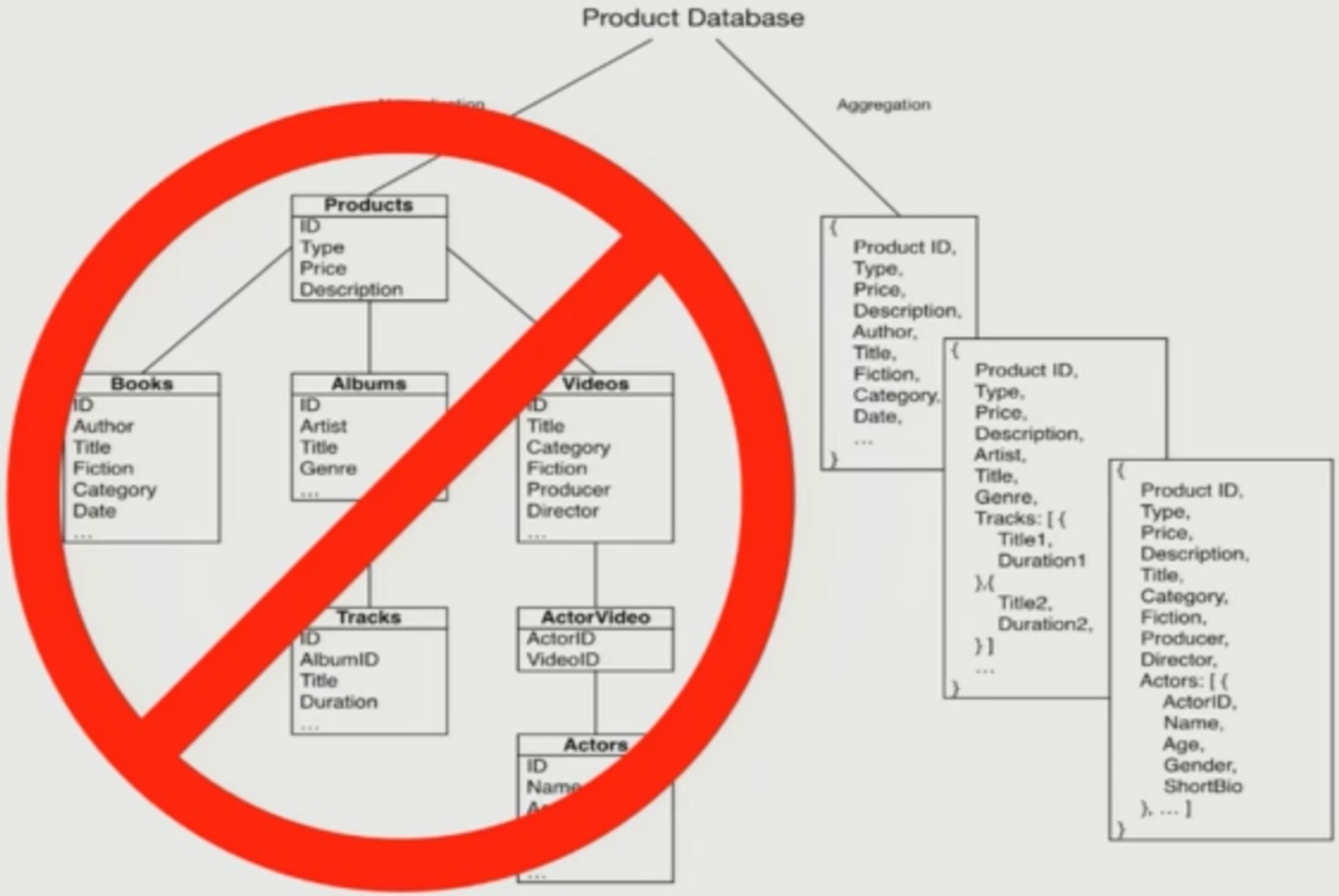
Fast and flexible nonrelational database service for any scale.

What is Amazon DynamoDB?

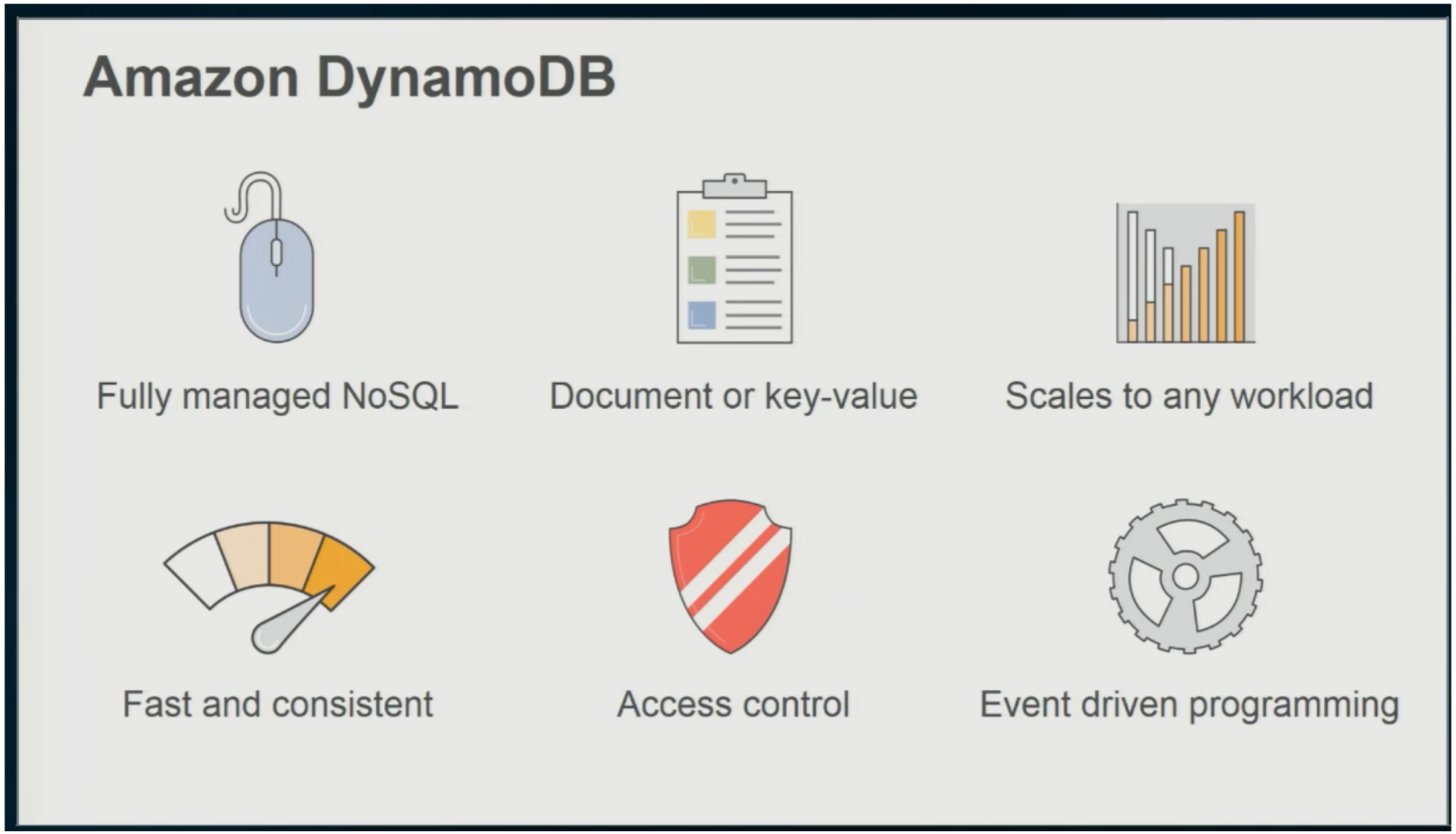
Amazon DynamoDB is a **fully managed NoSQL database** service that provides **fast** and predictable performance with seamless scalability.

DynamoDB enables customers to **offload the administrative burdens** of operating and scaling distributed databases to AWS so that they don’t have to worry about hardware provisioning, setup and configuration, throughput capacity planning, replication, software patching, or cluster scaling.

SQL vs. NoSQL Design Pattern



DynamoDB Features



What does Amazon DynamoDB manage on my behalf?

- Amazon DynamoDB takes away one of the main stumbling blocks of scaling databases: the management of database software and the provisioning of the hardware needed to run it.

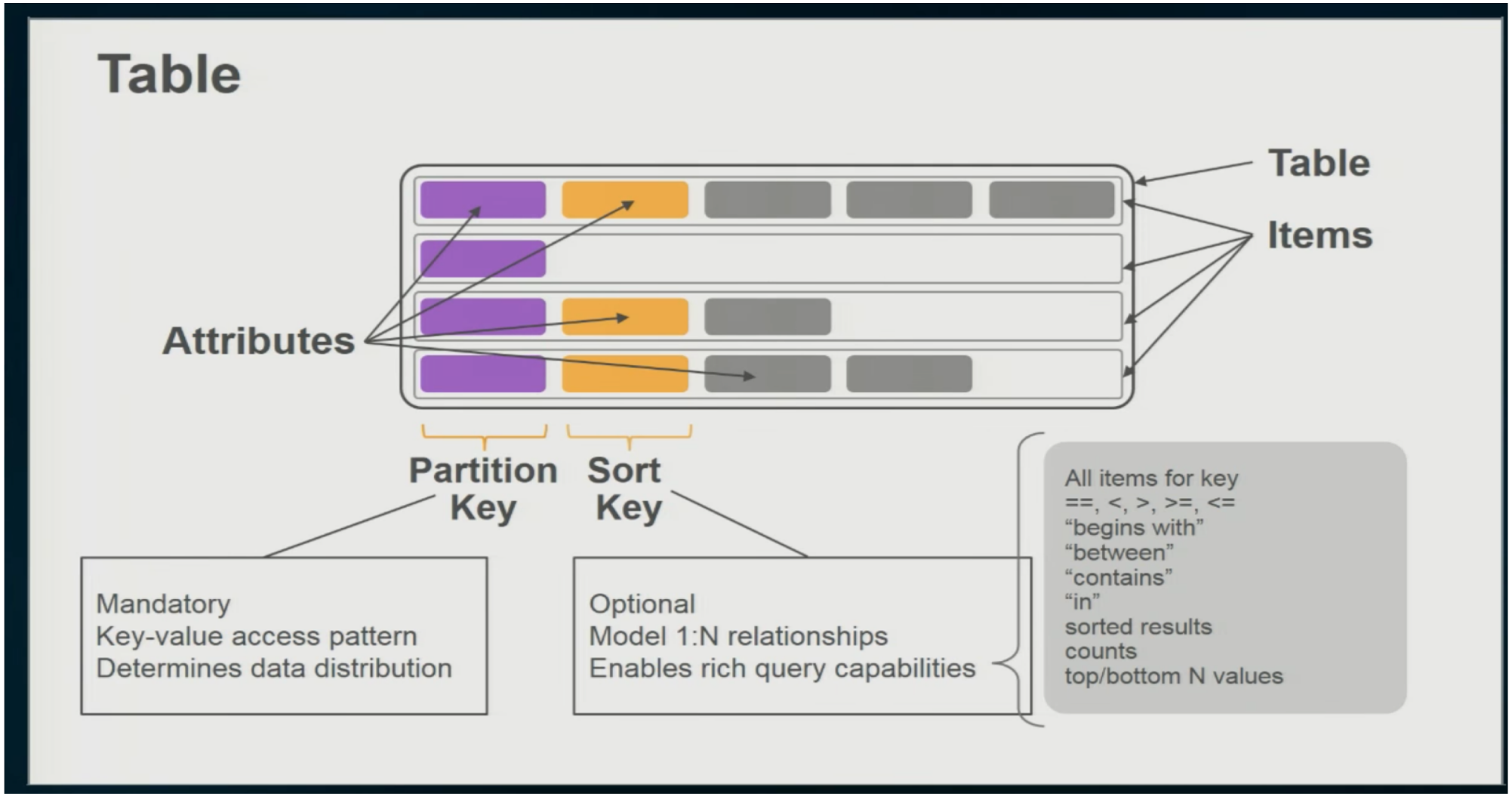
- You can deploy a nonrelational database in a matter of **minutes**. DynamoDB automatically scales throughput capacity to meet workload demands and partitions and repartitions your data as your table size grows. In addition, DynamoDB **synchronously replicates data across three facilities** in an AWS Region, giving you high availability and data durability.

“I worked at companies like MongoDB I’ve worked with technologies like Cassandra DynamoDB and other no SQL databases and really the biggest thing that I can tell you as a customers at Dynamodb they benefit from the fact that they don’t need to manage the operational infrastructure of the database it's not easy to run a scaled out distributed no SQL database.”

- Rick Houlihan, AWS

“..(At)First , you may not notice this but
when you get to 50 100 150 200 instances
you will start to notice the amount of
money you're spending just managing
those servers so taking that away from
the customer is probably the biggest
value you can get out of an manage NO
SQL service like dynamo DB” - Rick Houlihan, AWS

Components of DynamoDB



Partition Key

What is a partition key?

DynamoDB supports two types of primary keys:

- **Partition key:** Also known as a hash key, the *partition key* is composed of a single attribute. Attributes in DynamoDB are similar in many ways to fields or columns in other database systems.
- **Partition key and sort key:** Referred to as a *composite primary key* or *hash-range key*, this type of key is composed of two attributes. The first attribute is the *partition key*, and the second attribute is the *sort key*. Here is an example:

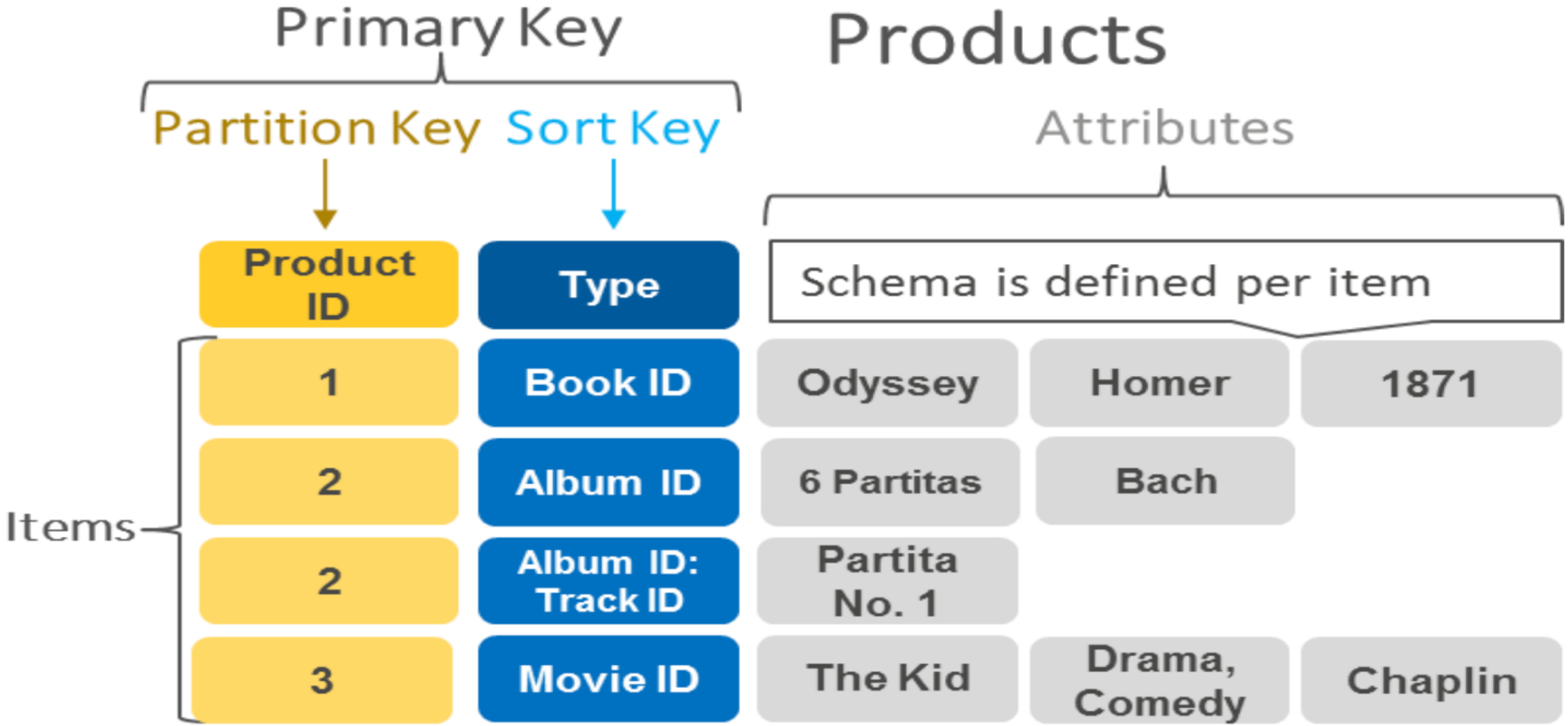


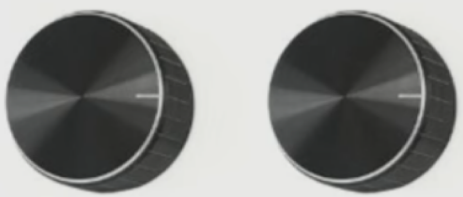
Figure 1 A DynamoDB table with a composite primary key

Throughput

Provisioned at the table level

- Write capacity units (WCUs) are measured in 1 KB per second
- Read capacity units (RCUs) are measured in 4 KB per second
  - RCUs measure strictly consistent reads
  - Eventually consistent reads cost 1/2 of consistent reads

Read and write throughput limits are independent



RCU WCU

Throughput provisioning in DynamoDB

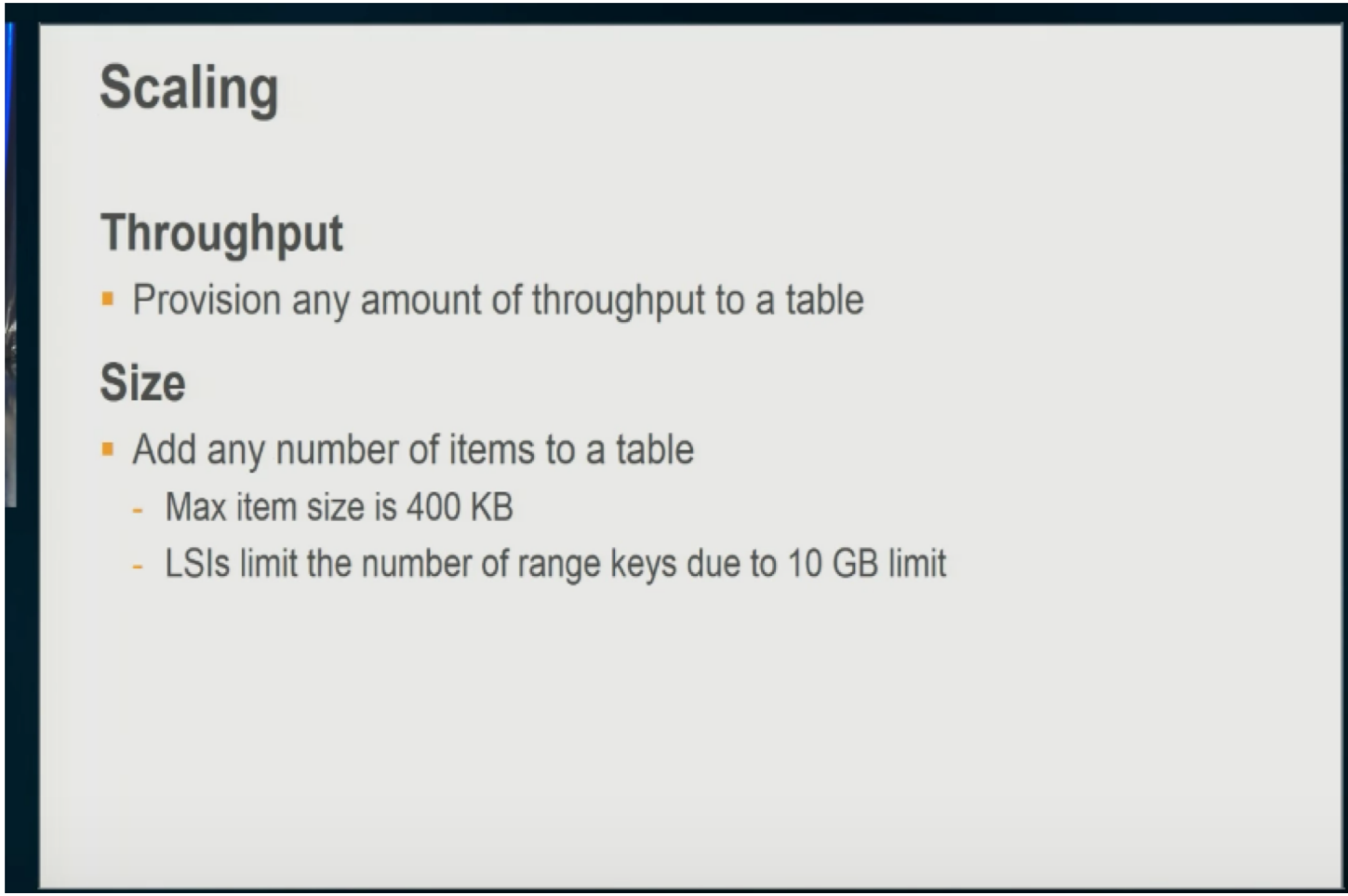
- each table you provision in dynamo DB will have throughput provisioned independently.
- Throughput is provisioned in write capacity and read capacity
- These two knobs are completely independent of each other

Scaling a DynamoDB table

Tables scales on throughput and size

Throughput: you tell AWS how much throughput you need on the table in write capacity and read capacity. AWS uses those numbers to split the table across a number of partitions

Size: you start inserting items on the table, every 10 gigabytes of items AWS starts to to split to another partition



What does read consistency mean?

- Amazon DynamoDB stores three geographically distributed replicas of each table to enable high availability and data durability. Read consistency represents the manner and timing in which the successful write or update of a data item is reflected in a subsequent read operation of that same item.

- DynamoDB exposes logic that enables you to specify the consistency characteristics you desire for each read request within your application.(There is a cost difference depending on what you choose)

What is the consistency model of Amazon DynamoDB?

When reading data from Amazon DynamoDB, users can specify whether they want the read to be eventually consistent or strongly consistent:

1. Eventually consistent reads (Default)
2. Strongly consistent reads

Eventually consistent reads (Default)

- The eventual consistency option maximizes your read throughput. However, an eventually consistent read might not reflect the results of a recently completed write. Consistency across all copies of data is usually reached **within a second**. Repeating a read after a short time should return the updated data.

- it's a very cheap way to double your read capacity

Strongly consistent reads

- In addition to eventual consistency, Amazon DynamoDB also gives you the flexibility and control to request a strongly consistent read if your application, or an element of your application, requires it. A strongly consistent read returns a result that reflects all writes that received a successful response prior to the read.

Partitioning Math

Number of Partitions	
By Capacity	(Total RCU / 3000) + (Total WCU / 1000)
By Size	Total Size / 10 GB
Total Partitions	CEILING(MAX (Capacity, Size))

In the future, these details might change…

What causes throttling?

If **sustained** throughput goes beyond provisioned throughput per partition

Non-uniform workloads

- Hot keys/hot partitions
- Very large items

Mixing hot data with cold data

- Use a table per time period

From the example before:

- Table created with 5000 RCUs, 500 WCUs
- RCUs per partition = 1666.67
- WCUs per partition = 166.67
- If sustained throughput > (1666 RCUs or 166 WCUs) per key or partition, DynamoDB may throttle requests
  - Solution: Increase provisioned throughput

Best Practice

Getting the most out of DynamoDB throughput

"To get the most out of DynamoDB throughput, create tables where the partition key element has a large number of distinct values, and values are requested fairly uniformly, as randomly as possible."

—DynamoDB Developer Guide

Best Practices

Cache the popular items when there is a high volume of read traffic. The cache acts as a low-pass filter, preventing reads of unusually popular items from swamping partitions. For example, consider a table that has deals information for products. Some deals are expected to be more popular than others during major sale events like Black Friday or Cyber Monday.

Add random numbers/digits from a predetermined range for write-heavy use cases. If you expect a large volume of writes for a partition key, use an additional prefix or suffix (a fixed number from predeternmined range, say 1-10) and add it to the partition key. For example, consider a table of invoice transactions. A single invoice can contain thousands of transactions per client. How do we enforce uniqueness and ability to query/update the invoice details for high-volumetric clients?

Here is the recommended table layout for this scenario:

- Partition key: Add a random suffix (1-10 or 1-100) with the InvoiceNumber, depending on the number of items per InvoiceNumber.
- Sort key: ClientTransactionid.

Partition Key	Sort Key	Attribute1
InvoiceNumber+Randomsuffix	ClientTransactionid	Invoice_Date
121212-1	Client1_trans1	2016-05-17 01.36.45
121212-1	Client1-trans2	2016-05-18 01.36.30
121212-2	Client2_trans1	2016-06-15 01.36.20
121212-2	Client2_trans2	2016-07-1 01.36.15

- This combination will give us a good spread through the partitions. The sort key can be used to filter for a specific client (for example, where InvoiceNumber=121212-1 and ClientTransactionid begins with Client1).
- Because we have a random number appended to our partition key (1-10), we need to query the table 10 times for a given InvoiceNumber. Our partition key could be 121212-[1-10], so we need to query where partition key is 121212-1 and ClientTransactionid begins with Client1. We need to repeat this for 121212-2, on up to 121212-10 and then merge the results.

Best Practice for avoiding hotkey/throttling

- https://aws.amazon.com/blogs/database/choosing-the-right-dynamodb-partition-key/

Batch Read and Write

- **BatchGetItem**A single BatchGetItem operation can retrieve a maximum of 100 items. The total size of all the items retrieved cannot exceed 16 MB.
- **BatchWriteItem**A single BatchWriteItem operation can contain up to 25 PutItem or DeleteItem requests. The total size of all the items written cannot exceed 16 MB.

Limits

- The maximum item size in DynamoDB is 400 KB
- You can define a maximum of 5 local secondary indexes and 5 global secondary indexes per table.

Referenced from

AWS re:Invent 2016: Deep Dive on Amazon DynamoDB (DAT304):
https://www.youtube.com/watch?v=bCW3lhsJKfw&t=1791s