Asia's Largest

Cloud & AI

Conference 2023

17 - 18, November 2023
IIT Madras Research Park, Chennai

Amazon DynamoDB Deep Dive

Architecting Data Models for High-Performance Applications

4 people walk into a NOSQL restaurant...
but immediately leaves why?

Because they couldn't find a table!

```
{
    name: "Bhuvana",
    role: "Technical Architect",
    company: "Accenture",
    skills: [ "NodeJS", "AWS", "SQL", "NoSQL"],
    volunteer: ["AWS Community Builder", "AZConf 2023"],
    sideHustles: ["Tech Speaker", "Runs a Children's Library"],
    favoriteMovies: ["Kungfu Panda", "Soul"],
    favoriteQuote: ”Just Keep Swimming!”
}
```

azconf

Agenda

What is DynamoDB?
Why DynamoDB?
How DynamoDB?

"Computers are useless. They can only give you answers."

– Pablo Picasso, 1968

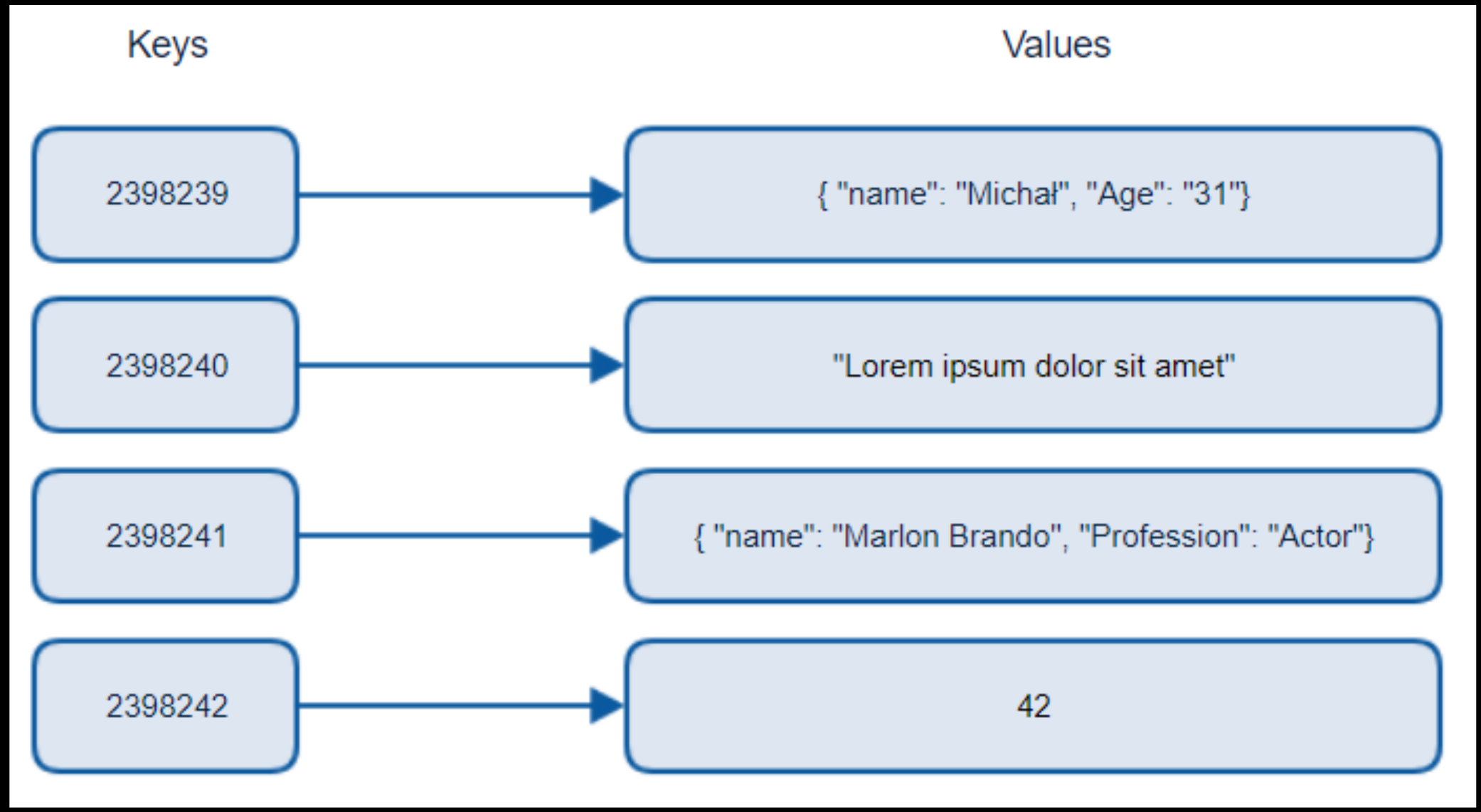
What is DynamoDB

- Amazon DynamoDB is a fully managed NoSQL database service that provides fast and predictable performance with seamless scalability. DynamoDB lets you offload the administrative burdens of operating and scaling a distributed database so that you don't have to worry about hardware provisioning, setup and configuration, replication, software patching, or cluster scaling.

- With DynamoDB, you can create database tables that can store and retrieve any amount of data and serve any level of request traffic. You can scale up or scale down your tables' throughput capacity without downtime or performance degradation. You can use the AWS Management Console to monitor resource utilization and performance metrics.

NoSQL Key-Value or Document Store

What?

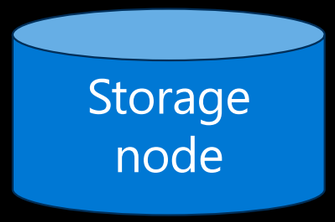

Keys
Values
2398239
{ "name": "Michał", "Age": "31"}
2398240
"Lorem ipsum dolor sit amet"
2398241
{ "name": "Marlon Brando", "Profession": "Actor"}
2398242
42

Predictable, Single digit millisecond response time at any scale

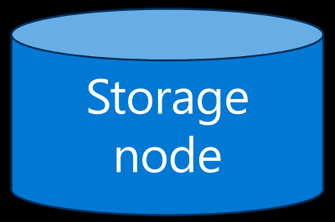
How?

Partition Key

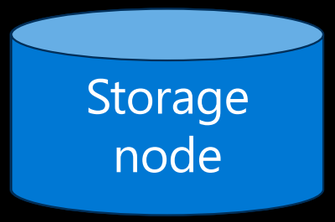
userId	created_ts	orderId	orderTotal
user1	01/07/23	order#1	2000
user2	03/07/23	order#2	600
user3	05/07/23	order#3	100
user4	01/07/23	order#4	1500
user5	05/07/23	order#5	1242



user#1

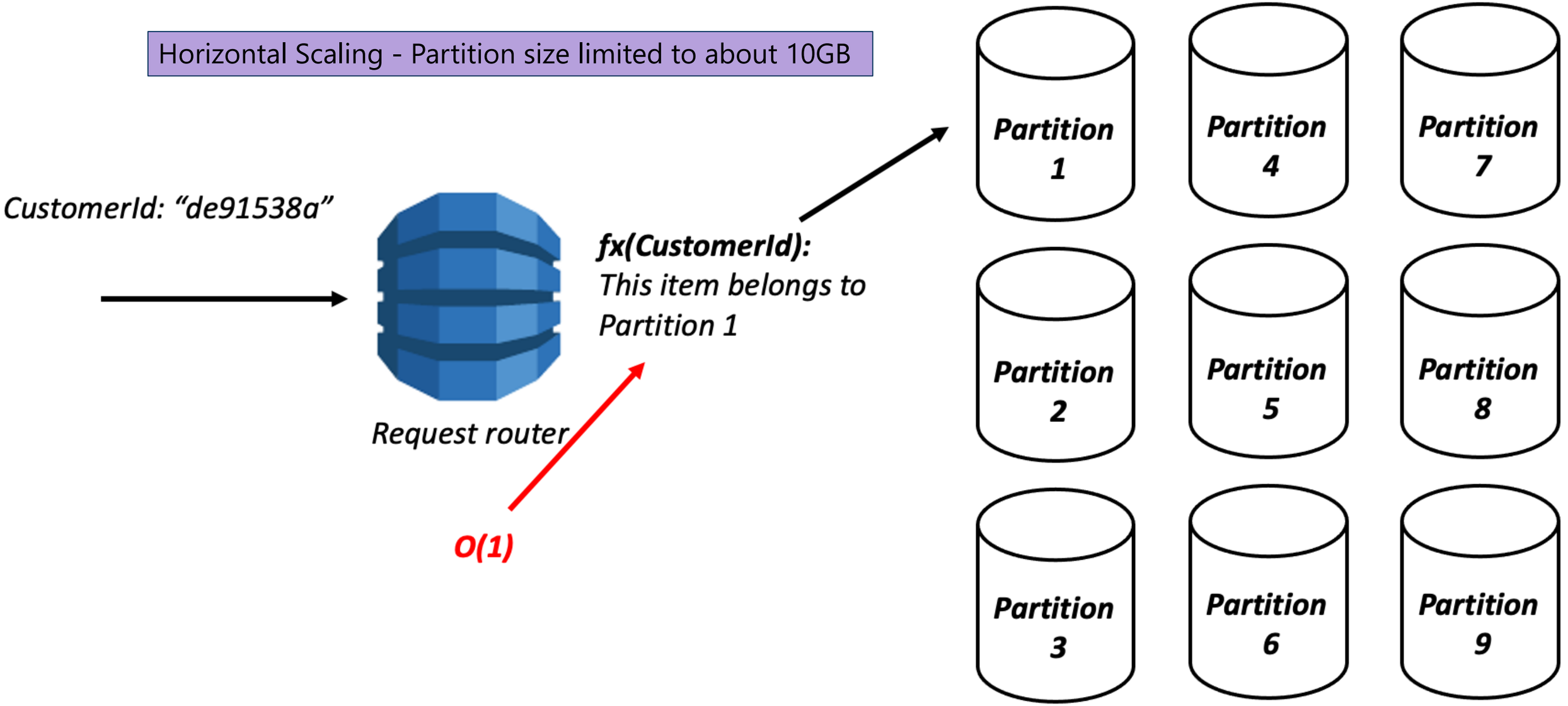


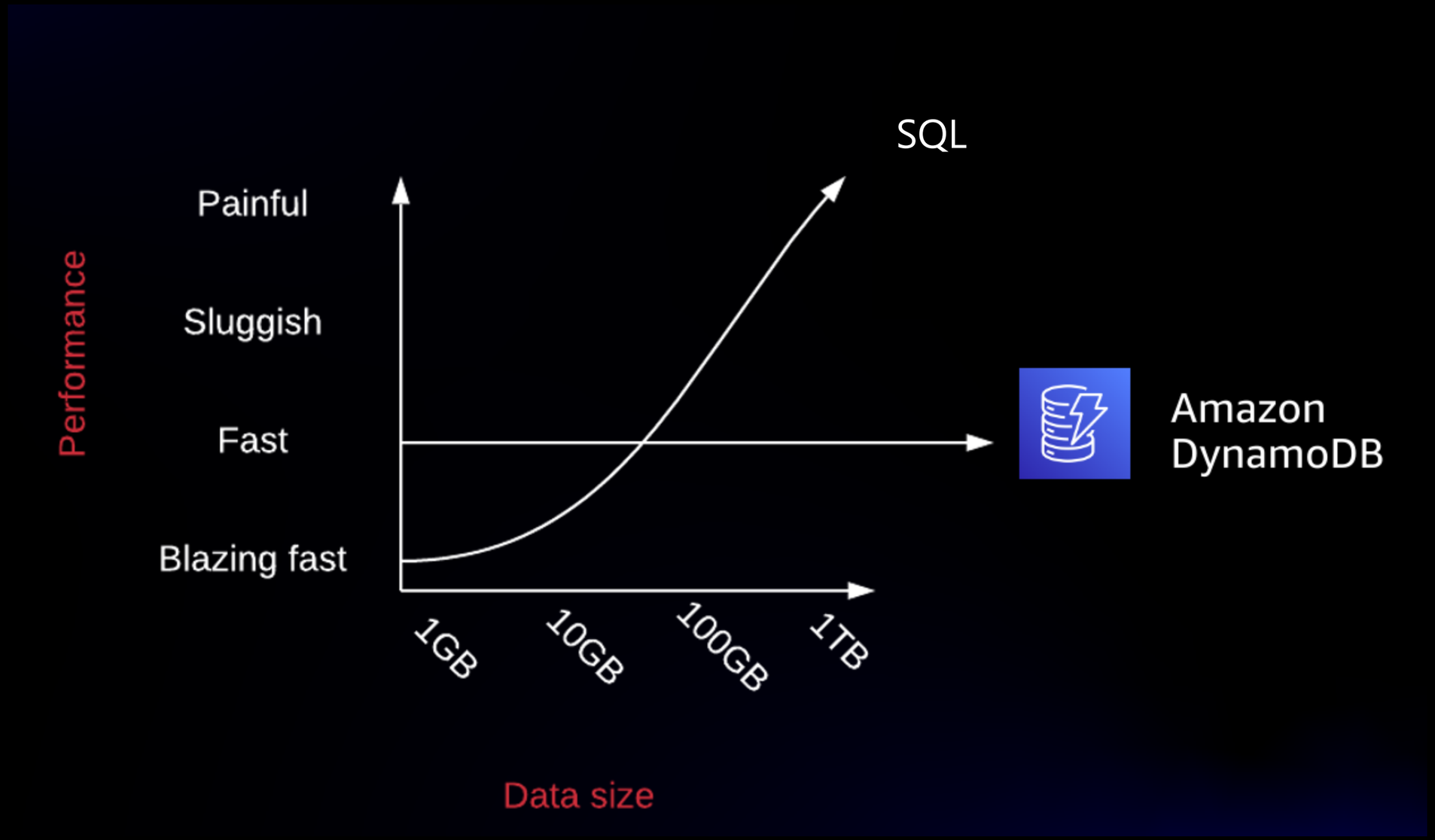
user#2
user#3



user#4
user#5

Horizontal Scaling - Partition size limited to about 10GB

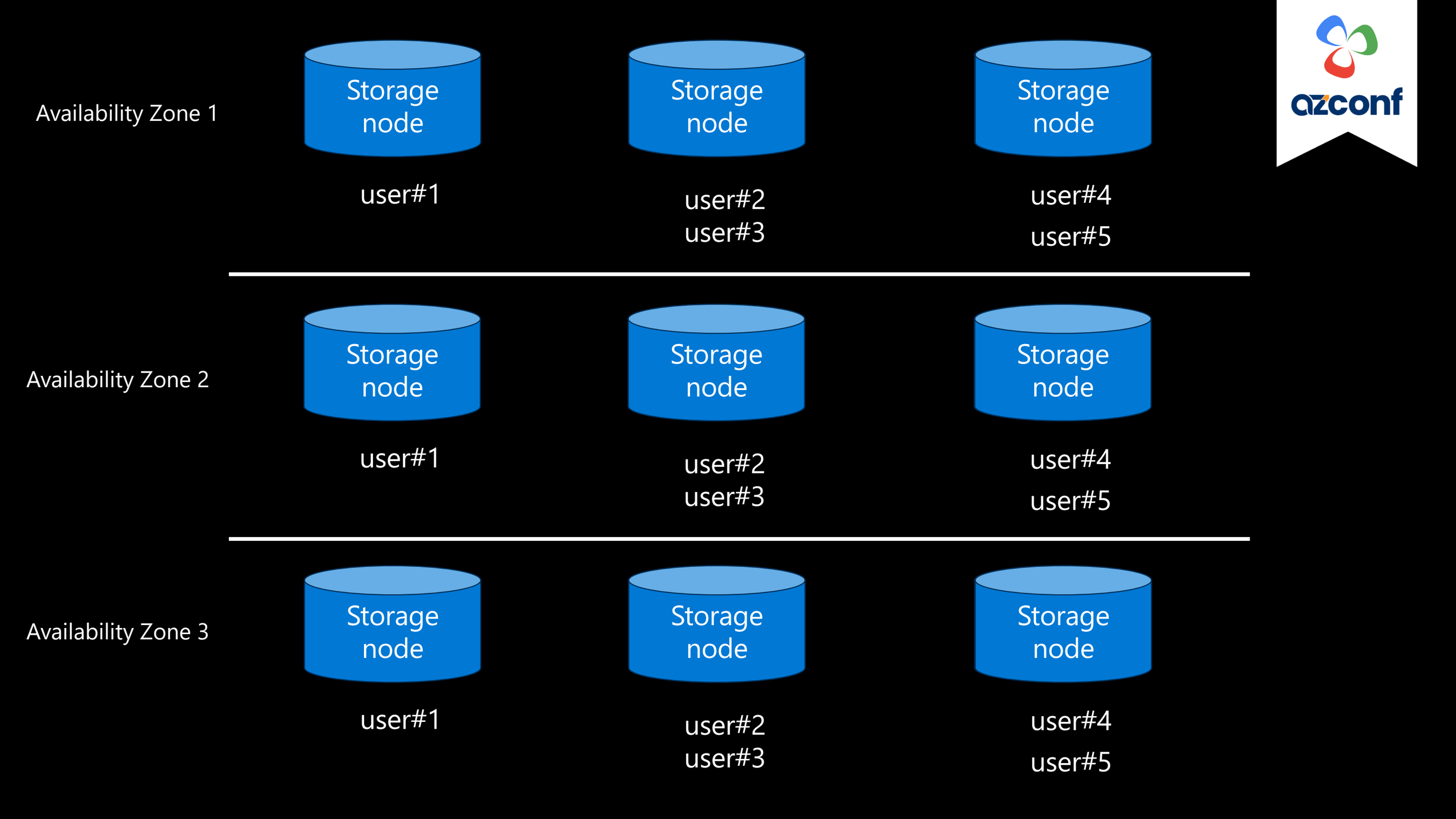




Source: Alex DeBrie blog

Highly Available - 99.999% Availability

How?


Availability Zone 1
Storage node
Storage node
Storage node
user#1
user#2
user#3
user#4
user#5
Availability Zone 2
Storage node
Storage node
Storage node
user#1
user#2
user#3
user#4
user#5
Availability Zone 3
Storage node
Storage node
Storage node
user#1
user#2
user#3
user#4
user#5
azconf

Amazon DynamoDB – DynamoDB powers multiple high-traffic Amazon properties and systems including Alexa, the Amazon.com sites, and all Amazon fulfillment centers. Over the course of Prime Day, these sources made trillions of calls to the DynamoDB API. DynamoDB maintained high availability while delivering single-digit millisecond responses and peaking at 126 million requests per second.

Source : Prime Day 2023 Powered by AWS – All the Numbers by Jeff Barr

Serverless

What?

Serverless

- A fully managed service –
  No worry about things like server health, storage, and network connectivity

- Infinitely scalable

Pricing?

Pricing

- Pay-per-use model
  – you never pay for hardware or services you're not actually using
- Traditional DB – CPU, RAM, IOPS
- Pay for reads and writes – RCU and WCU

Why are we discussing architecture?

To model data that exploits this architecture

Why we need to understand architecture?

To get consistent, predictable and reliable performance from DynamoDB

Terminology

- Table
- Items – (SQL Rows)
- Primary Key
- Attributes – (SQL Columns)

Table - Employee

Primary key	Attributes			
Partition key: LoginAlias				
johns	FirstName	LastName	ManagerLoginAlias	Skills
	John	Stiles	NA	["executive management"]
marthar	FirstName	LastName	ManagerLoginAlias	Skills
	Martha	Rivera	johns	["software","management"]
mateoj	FirstName	LastName	ManagerLoginAlias	Skills
	Mateo	Jackson	marthar	["software"]
janed	FirstName	LastName	ManagerLoginAlias	Skills
	Jane	Doe	marthar	["software"]
diegor	FirstName	LastName	ManagerLoginAlias	Skills
	Diego	Ramirez	johns	["executive assistant"]

Primary Key

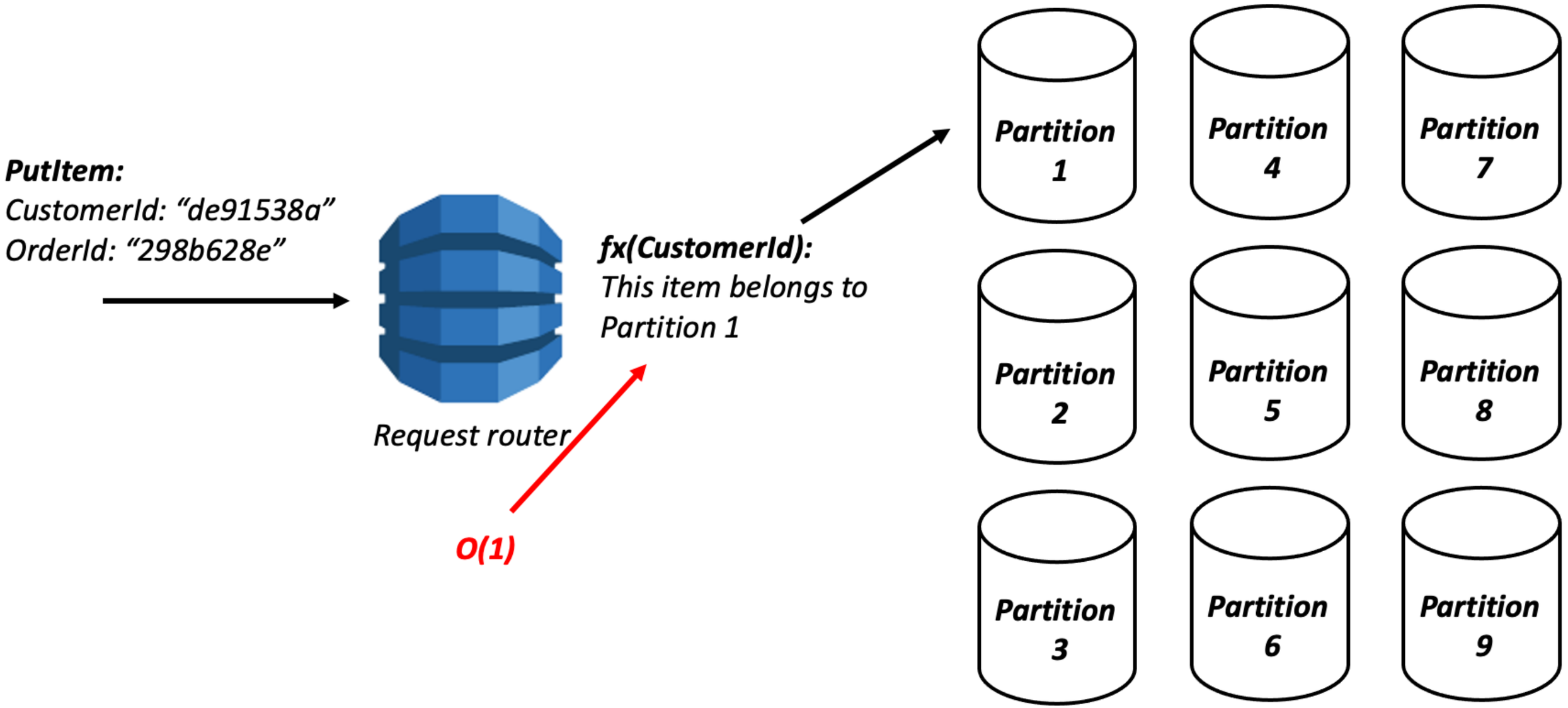
- Simple Primary Key (Partition Key)
- Composite Primary Key (Partition Key + Sort Key)

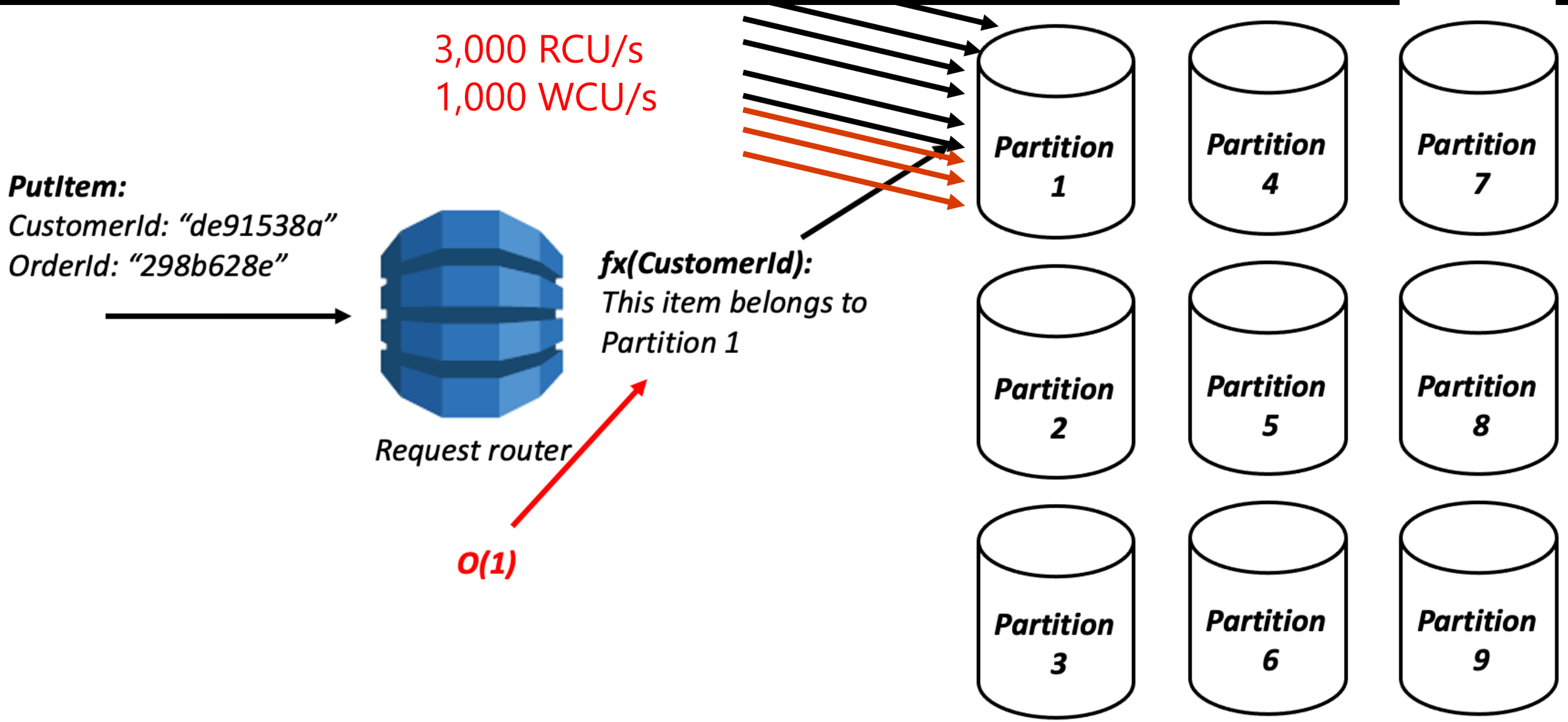
How to choose Primary Key?

- Partition key is for grouping
- Sort key is for ordering

Partition Key

- High cardinality to distribute requests


PutItem:
CustomerId: “de91538a”
OrderId: “298b628e”
Request router
fx(CustomerId):
This item belongs to
Partition 1
O(1)
Partition 1
Partition 4
Partition 7
Partition 2
Partition 5
Partition 8
Partition 3
Partition 6
Partition 9


3,000 RCU/s
1,000 WCU/s
PutItem:
CustomerId: “de91538a”
OrderId: “298b628e”
Request router
fx(CustomerId):
This item belongs to
Partition 1
O(1)
Partition 1
Partition 4
Partition 7
Partition 2
Partition 5
Partition 8
Partition 3
Partition 6
Partition 9

Partition Key

- High cardinality to distribute requests
- Unique and meaningful
- Cannot update the primary key – field nor value

Sort Key

- For Ordering
- Alphabetical
- ISO Timestamps
- ULID - Universally Unique Lexicographically Sortable Identifier

Sort Key Operations

- a = b
- a < b
- a <= b
- a > b
- a >= b

- a BETWEEN b AND c

- begins_with (a, substr)

Choosing Primary Key is one of the most important decisions in the DynamoDB data modeling.

DynamoDB Read and Write Operations

Read Operations

Query	Scan
Retrieves items based on specified key criteria	Scans the entire table or a subset based on filters
Highly efficient for retrieving specific items	Less efficient for large tables or full-table scans
Can query based on partition key and sort key	Limited to filtering based on non-key attributes
Consumes fewer read capacity units (RCUs)	Consumes more RCUs due to scanning the entire table
Lower cost as it retrieves specific items	Higher cost due to scanning the entire or large subset
Ideal for retrieving specific items by key	Suitable for ad-hoc queries or infrequent use cases

Write Operations

- PutItem
- UpdateItem
- DeleteItem
- BatchWriteItem

What if we want to query based on attributes that are not primary key?

Indexing

Local Secondary Index

TABLE	Id (Partition)	Name (Sort Key)	Address	Phone	Email		
LSI	Id (Partition)	Address (Sort Key)	Name (Key)			KEYS ONLY	Projectsions
	Id (Partition)	Phone (Sort Key)	Name (Key)	Email		INCLUDE "EMAIL"	
	Id (Partition)	Email (Sort Key)	Name (Key)	Address	Phone	ALL	

Global secondary index

TABLE	Id (Partition)	Name	Address	Phone	Email		
GSI	Name (Partition)	Id (Key)				KEYS ONLY	Projectsions
	Phone (Partition)	Address (Sort Key)	Id (Key)	Name		INCLUDE "NAME"	
	Email (Partition)	Phone (Sort Key)	Id (Key)	Name	Address	ALL	

Demo


Words are useless! Gobble, gobble, gobble, gobble!
Too much of it, darling! Too much!
ABORTIA CLINIQUE

Data Model Blog App

CLOUD INFORMATION

Lorem ipsum dolor sit amet, aliqua.

Dec 02,2019.

Lorem Ipsum   Follow 64

Lorem ipsum dolor sit amet,aliqua.

Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur.

Lorem Ipsum   Follow 64

Lorem ipsum dolor sit amet,aliqua.

Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur.

Lorem Ipsum   Follow 64

Lorem ipsum dolor sit amet,aliqua.

Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur.

Lorem Ipsum   Follow 64

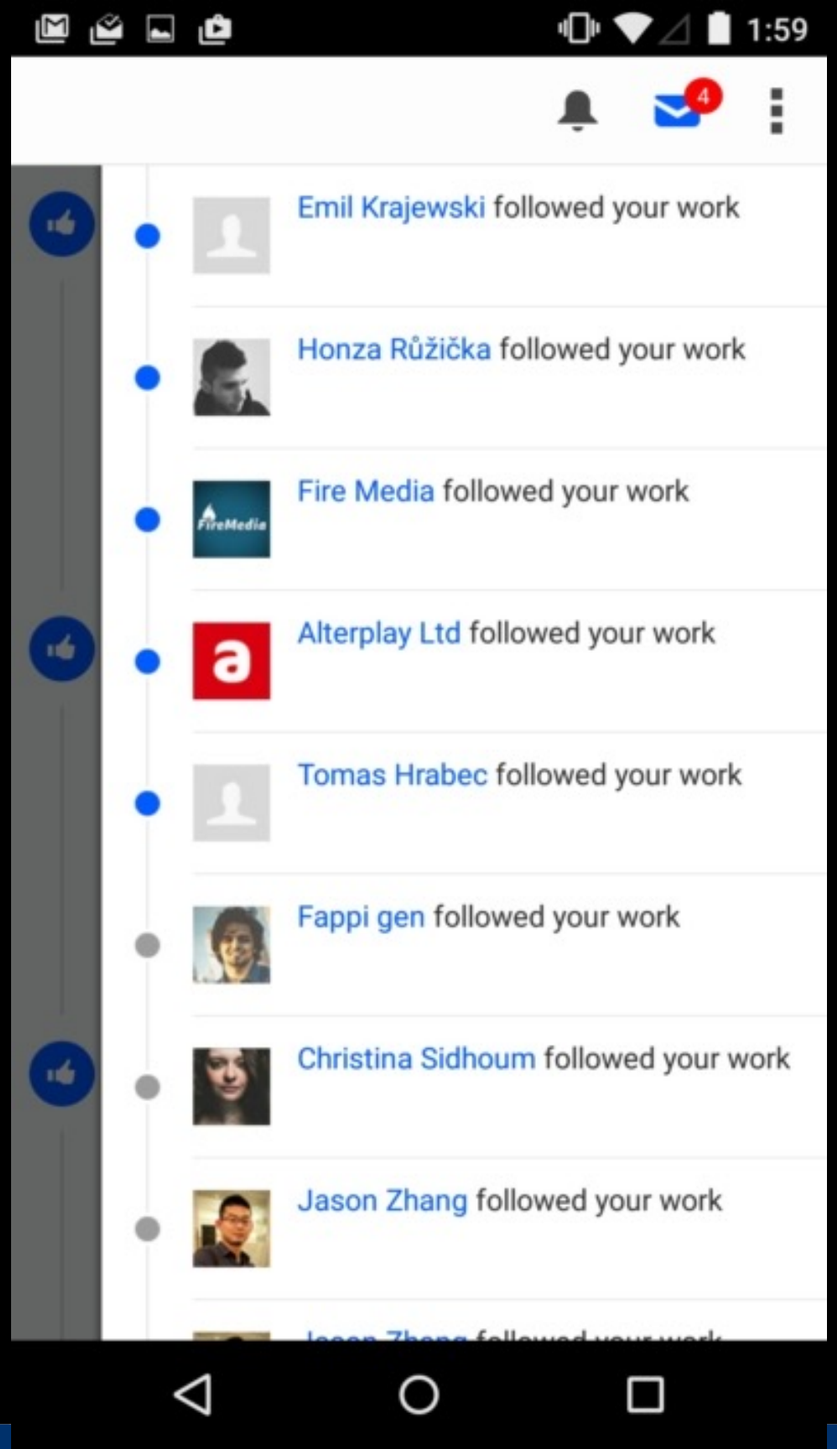
Lorem ipsum dolor sit amet,aliqua.

Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur.

Lorem Ipsum   Follow 64

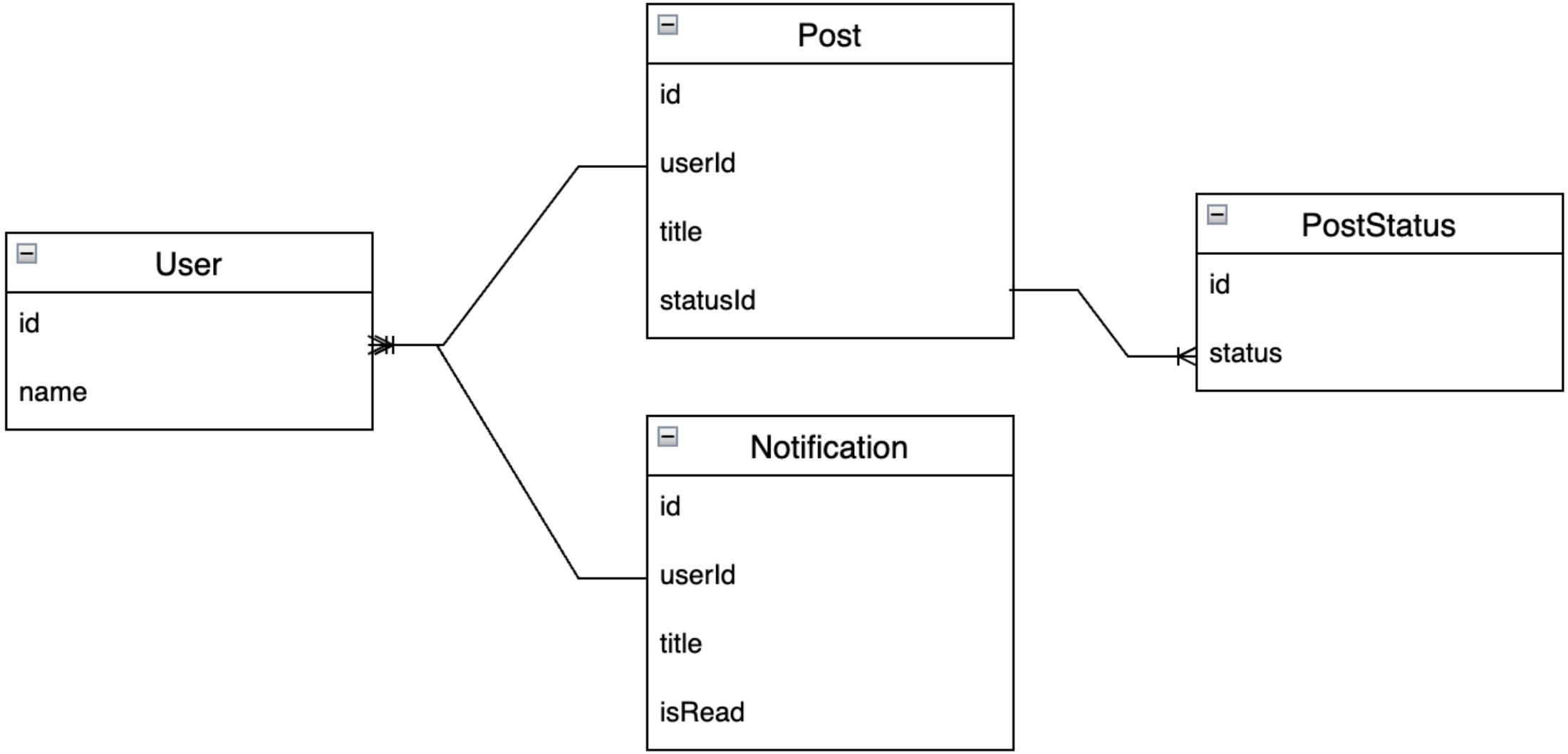
Lorem ipsum dolor sit amet,aliqua.

Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur.

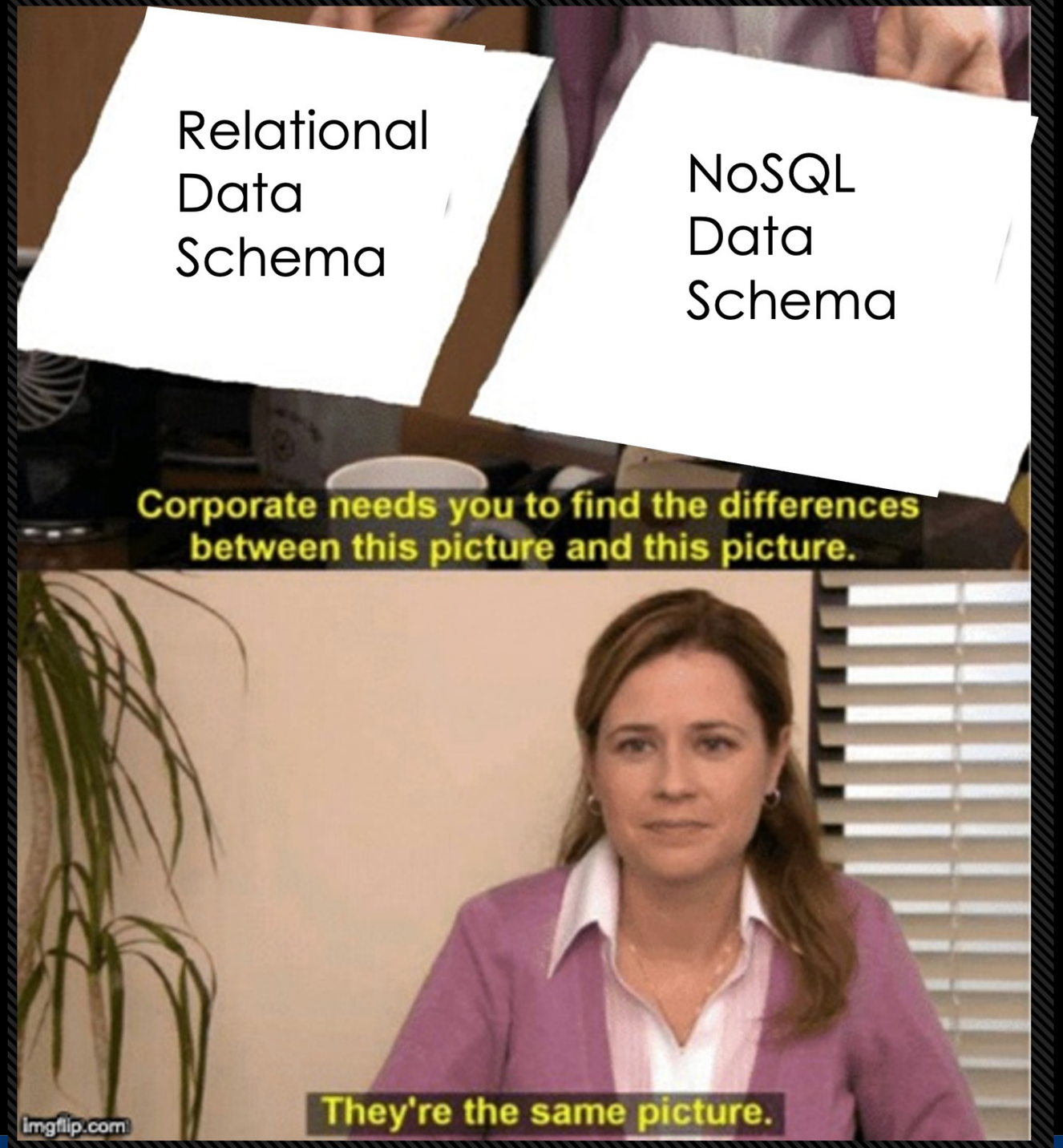

1:59
4
Emil Krajewski followed your work
Honza Růžička followed your work
Fire Media followed your work
FireMedia
Alterplay Ltd followed your work
Tomas Hrabec followed your work
Fappi gen followed your work
Christina Sidhoum followed your work
Jason Zhang followed your work

azconf

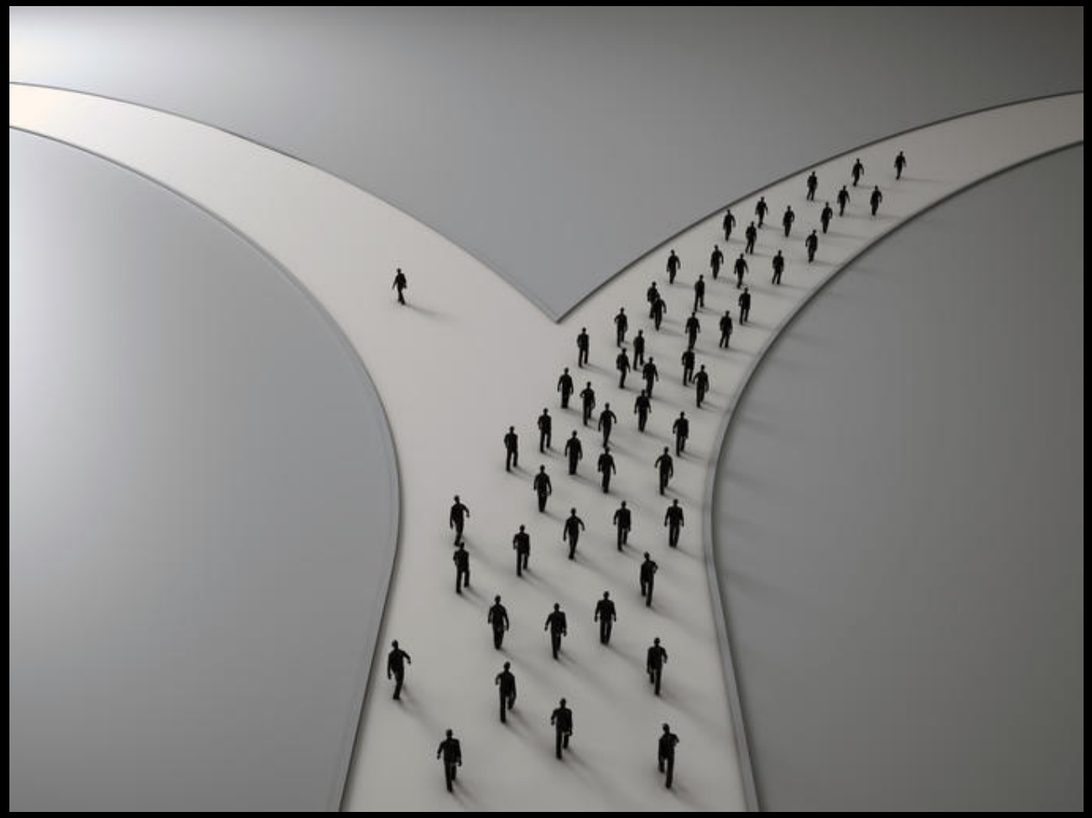
Entity Relationship Diagram


Post
id
userId
title
statusId
PostStatus
id
status
User
id
name
Notification
id
userId
title
isRead

Schema for DynamoDB


Relational
Data
Schema
NoSQL
Data
Schema
Corporate needs you to find the differences between this picture and this picture.
They're the same picture.
imgflip.com

Different Mindset



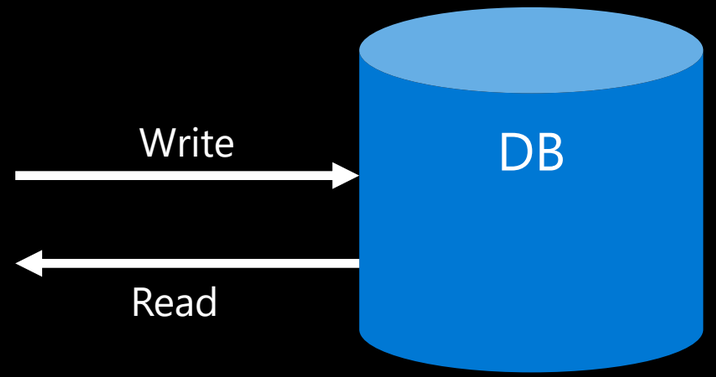
azconf

How NoSQL design is different

- By contrast, you shouldn't start designing your schema for DynamoDB until you know the questions it will need to answer. Understanding the business problems and the application use cases up front is essential.
- You should maintain as few tables as possible in a DynamoDB application. Having fewer tables keeps things more scalable, requires less permissions management, and reduces overhead for your DynamoDB application. It can also help keep backup costs lower overall.

Access Patterns

How you access your data – Business Usecases



Write Operations

1. Create User
2. Create Post
3. Update Post
4. Like a Post
5. Create Notification
6. Mark Notification As Read

Read Operations

1. Get User profile
2. Get All Posts for a user
3. Get All Published Posts for a user
4. Get Draft Posts for a user
5. Sort Published Posts By Likes and return top 3 for featured section
6. Get No. of Published Posts for a user
7. Get All Notifications for a user
8. Get Unread Notifications for a user
9. Get No. of Unread Notifications for a user


WHY DO NOSQL DEVELOPERS EAT LUNCH ALONE?
I DON'T KNOW, WHY?
THEY DON'T KNOW HOW TO JOIN TABLES
imgflip.com

azconf

Single Table Design

WHAT!!

1. Get User profile

Primary key		Attributes
Partition key: userId	Sort key: sk	
user_1	profile	Name
		Rose Luettgen
user_2	profile	Name
		Ida Hegmann Sr.
user_3	profile	Name
		Ralph Walker

2. Get All Posts for a user

{userId: “user1”, sk: {BEGINS_WITH : “post#” }

Primary key		Attributes		
Partition key: userId	Sort key: sk			
user_1	post#1	Title	Status	NoOfLikes
		Title of Post 1	PUBLISHED	10
	post#2	Title	Status	
		Title of Post 2	DRAFT	
	post#3	Title	Status	NoOfLikes
		Title of Post 3	PUBLISHED	7
	post#4	Title	Status	NoOfLikes
		Title of Post 4	PUBLISHED	12
	profile	Name		
		Rose Luettgen		

3. Get All Published Posts for a user

Filter: { Status: PUBLISHED }

Primary key		Attributes		
Partition key: userId	Sort key: sk			
user_1	post#1	Title	Status	NoOfLikes
		Title of Post 1	PUBLISHED	10
	post#2	Title	Status	
		Title of Post 2	DRAFT	
	post#3	Title	Status	NoOfLikes
		Title of Post 3	PUBLISHED	7
	post#4	Title	Status	NoOfLikes
		Title of Post 4	PUBLISHED	12
	profile	Name		
		Rose Luettgen		

4. Get All Draft Posts for a user

Filter: { Status: DRAFT }

Primary key		Attributes		
Partition key: userId	Sort key: sk			
user_1	post#1	Title	Status	NoOfLikes
		Title of Post 1	PUBLISHED	10
	post#2	Title	Status	
		Title of Post 2	DRAFT	
	post#3	Title	Status	NoOfLikes
		Title of Post 3	PUBLISHED	7
	post#4	Title	Status	NoOfLikes
		Title of Post 4	PUBLISHED	12
	profile	Name		
		Rose Luettgen		

Index - { userId , status }

5. Sort Published Posts By Likes and return top 3 for featured section

Index - { userId , noOfLikes }

Solution

1. Add NoOfLikes = -1 for Draft Posts

2. Create Index on { userId, NoOfLikes }

Primary key		Attributes		
Partition key: userId	Sort key: sk			
user_1	post#1	Title	Status	NoOfLikes
		Title of Post 1	PUBLISHED	10
	post#2	Title	Status	NoOfLikes
		Title of Post 2	DRAFT	-1
	post#3	Title	Status	NoOfLikes
		Title of Post 3	PUBLISHED	7
	post#4	Title	Status	NoOfLikes
		Title of Post 4	PUBLISHED	12
	profile	Name	NoOfPublishedPosts	
		Rose Luettgen	3	
user_2	post#10	Title	Status	NoOfLikes
		Title of Post 10	PUBLISHED	0
	post#11	Title	Status	NoOfLikes
		Title of Post 11	DRAFT	-1
	profile	Name	NoOfPublishedPosts	
		Ida Hegmann Sr.	1	

noOfLikesIndex

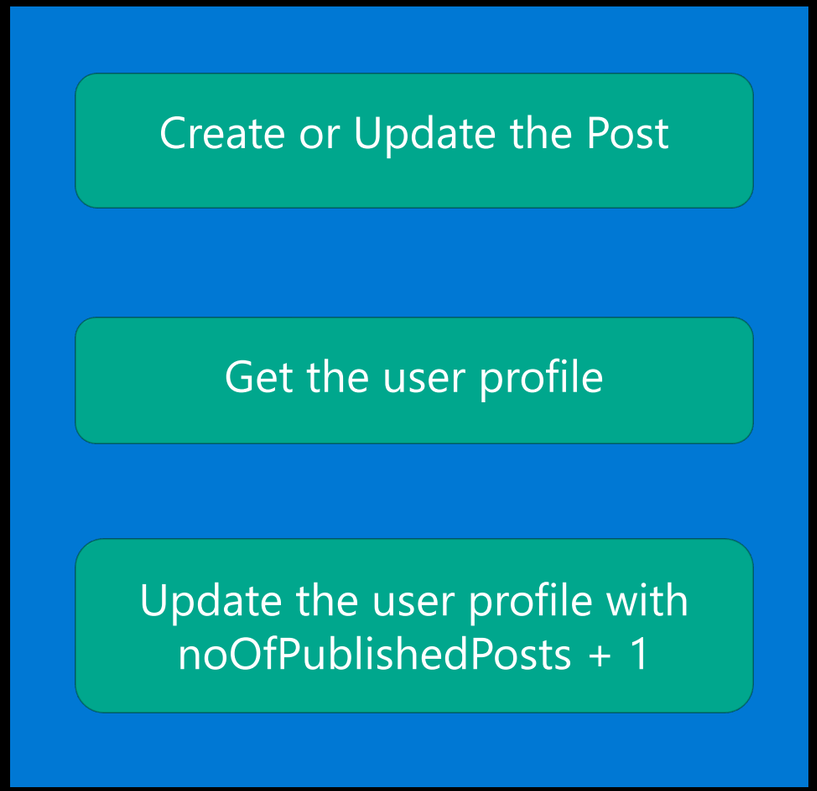
Primary key		Attributes		
Partition key: userId	Sort key: NoOfLikes			
user_1	-1	sk	Title	Status
		post#2	Title of Post 2	DRAFT
	10	sk	Title	Status
		post#1	Title of Post 1	PUBLISHED
	12	sk	Title	Status
		post#4	Title of Post 4	PUBLISHED
	7	sk	Title	Status
		post#3	Title of Post 3	PUBLISHED
user_2	-1	sk	Title	Status
		post#11	Title of Post 11	DRAFT
	0	sk	Title	Status
		post#10	Title of Post 10	PUBLISHED

6. Get No. of Published Posts for a user

Primary key		Attributes		
Partition key: userId	Sort key: sk			
user_1	post#1	Title	Status	NoOfLikes
		Title of Post 1	PUBLISHED	10
	post#2	Title	Status	
		Title of Post 2	DRAFT	
	post#3	Title	Status	NoOfLikes
		Title of Post 3	PUBLISHED	7
	post#4	Title	Status	NoOfLikes
		Title of Post 4	PUBLISHED	12
	profile	Name	NoOfPublishedPosts	
		Rose Luettgen	3	
user_2	post#10	Title	Status	NoOfLikes
		Title of Post 10	PUBLISHED	0
	post#11	Title	Status	
		Title of Post 11	DRAFT	
	profile	Name	NoOfPublishedPosts	
		Ida Hegmann Sr.	1	

Transactions API

- Synchronous PUT, UPDATE, DELETE
- Atomic
- Automated Rollback



7. Get All Notifications for a user

userId,
BEGINS_WITH :
“notification#”

Primary key		Attributes		
Partition key: userId	Sort key: sk			
user_1	notification#1	NotificationMessage	NotificationIsRead	
		Notification 1	false	
	notification#2	NotificationMessage	NotificationIsRead	
		Notification 2	true	
	post#1	Title	Status	NoOfLikes
		Title of Post 1	PUBLISHED	10
	post#2	Title	Status	NoOfLikes
		Title of Post 2	DRAFT	-1
	post#3	Title	Status	NoOfLikes
		Title of Post 3	PUBLISHED	7
	post#4	Title	Status	NoOfLikes
		Title of Post 4	PUBLISHED	12
	profile	Name	NoOfPublishedPosts	
		Rose Luettgen	3	

8. Get Unread Notifications for a user

Index - {userId, NotificationIsRead}

Indexes both unread and read

Primary key		Attributes	
Partition key: userId	Sort key: NotificationIsRead		
user_1	false	sk	NotificationMessage
		notification#4	Notification 4
	true	sk	NotificationMessage
		notification#1	Notification 1
	true	sk	NotificationMessage
		notification#2	Notification 2
	true	sk	NotificationMessage
		notification#3	Notification 3

Delete NotificationIsRead value for read notification

Primary key		Attributes		
Partition key: userId	Sort key: sk			
user_1	notification#1	NotificationMessage		
		Notification 1		
	notification#2	NotificationMessage		
		Notification 2		
	notification#3	NotificationMessage		
		Notification 3		
	notification#4	NotificationMessage	NotificationIsRead	
		Notification 4	false	
	post#1	Title	Status	NoOfL
		Title of Post 1	PUBLISHED	10

Index - {userId, NotificationIsRead}

Primary key		Attributes	
Partition key: userId	Sort key: NotificationIsRead		
user_1	false	sk	NotificationMessage
		notification#4	Notification 4

9. Get No. of Unread Notifications for a user

Count

Single Table Design

[Table] userBlogApp

Total 3 | 100/page | 1 | Actions

userBlogApp | GSI: NoOfLikesIndex | GSI: UnreadNotificationsIndex

Primary key		Attributes		
Partition key: userId	Sort key: sk			
user_1	notification#1	NotificationMessage		
		Notification 1		
	notification#2	NotificationMessage		
		Notification 2		
	notification#3	NotificationMessage		
		Notification 3		
	notification#4	NotificationMessage	NotificationIsRead	
		Notification 4	false	
	post#1	Title	Status	NoOfLikes
		Title of Post 1	PUBLISHED	10
	post#2	Title	Status	NoOfLikes
		Title of Post 2	DRAFT	-1
	post#3	Title	Status	NoOfLikes
		Title of Post 3	PUBLISHED	7
	post#4	Title	Status	NoOfLikes
		Title of Post 4	PUBLISHED	12
	profile	Name	NoOfPublishedPosts	
		Rose Luettgen	3	

Best Practices

- Identify your application's access patterns
- Understand the single-table design
- Avoid hot keys
- Avoid full-table scans
- Keep the sizes of indexes as small as possible
- Keep the number of indexes to a minimum

Data that is accessed together should be stored together.


Relational
NoSQL

SQL

NoSQL

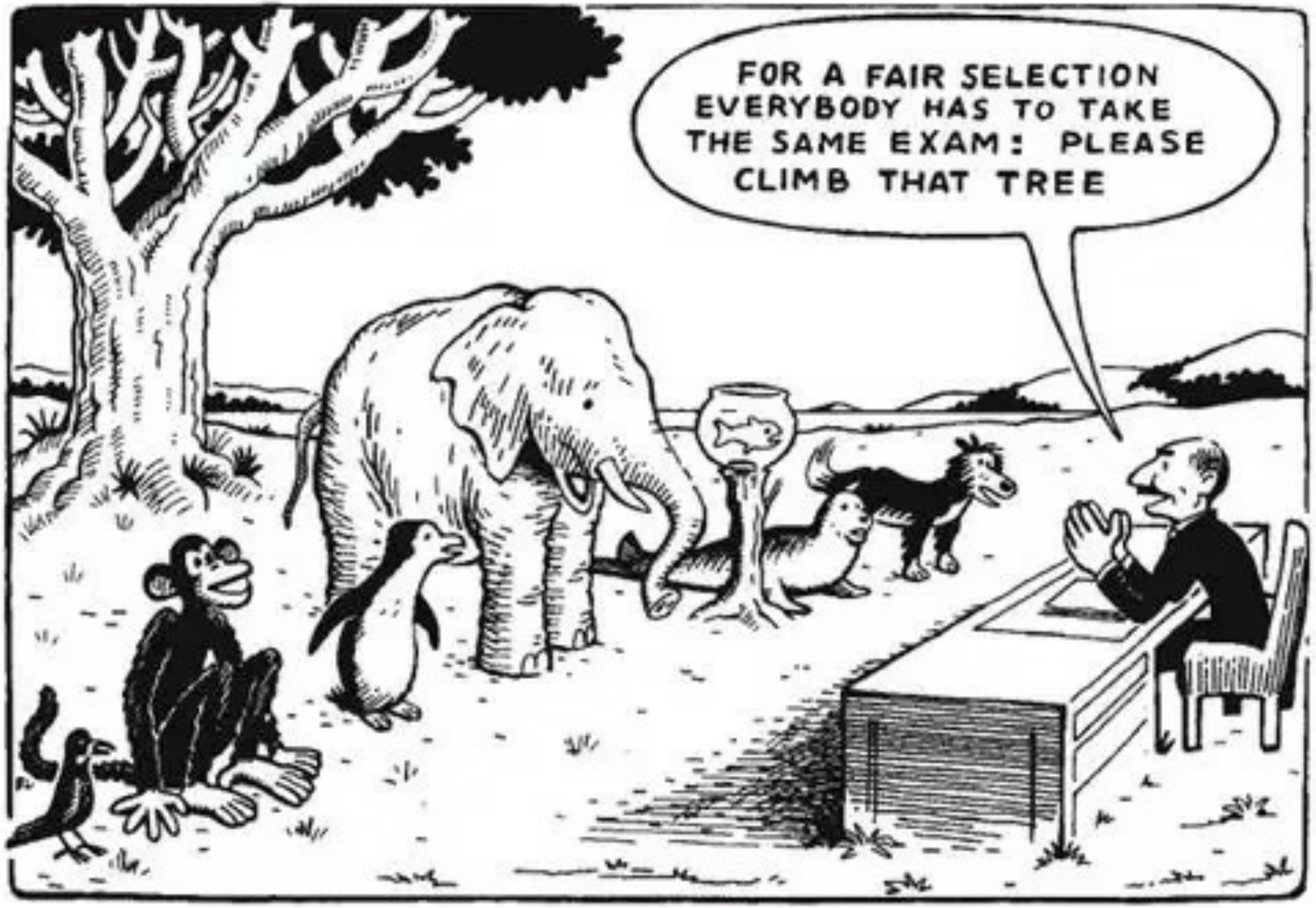
azconf


cheese straws
CRACKERS
SQL


No-SQL

Dem

Conten


FOR A FAIR SELECTION EVERYBODY HAS TO TAKE THE SAME EXAM: PLEASE CLIMB THAT TREE

Thank you! Happy No-SQLing!

Bhuvana

Technical Architect, Accenture

Bhuvana R

AWS Community Builder | 2x AWS Certified | Architect | NodeJS

azconf

Gold Partner

PRESIDIO®

Silver Partner

ManageEngine Site24x7 Crafted at ZOHO Corp.

SAGENT

Securra Health

kyn Where we belong

Syncfusion®

PRO