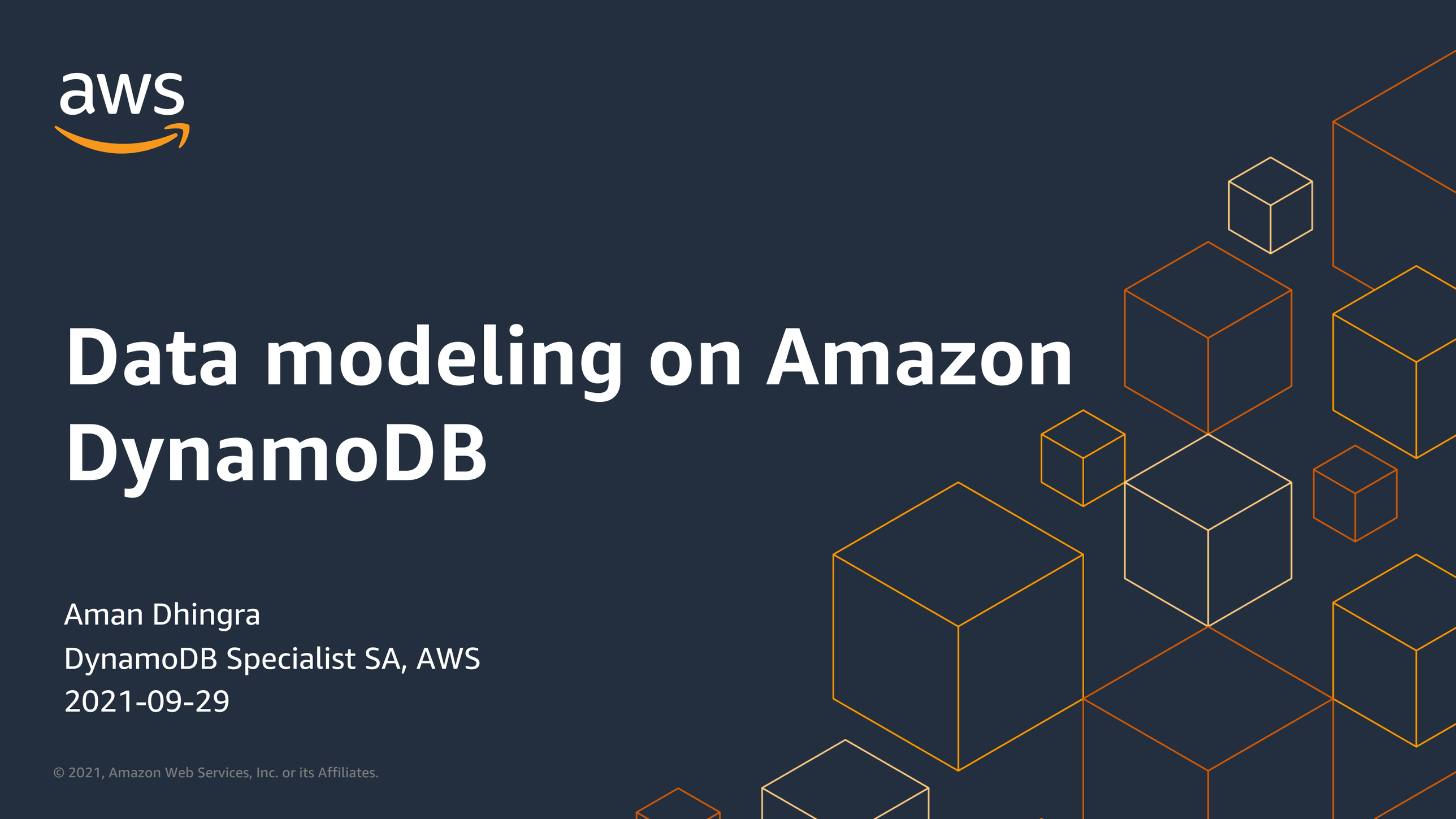


Data modeling on Amazon DynamoDB

Aman Dhingra
DynamoDB Specialist SA, AWS
2021-09-29

Agenda

Amazon DynamoDB key concepts

NoSQL Workbench Demo

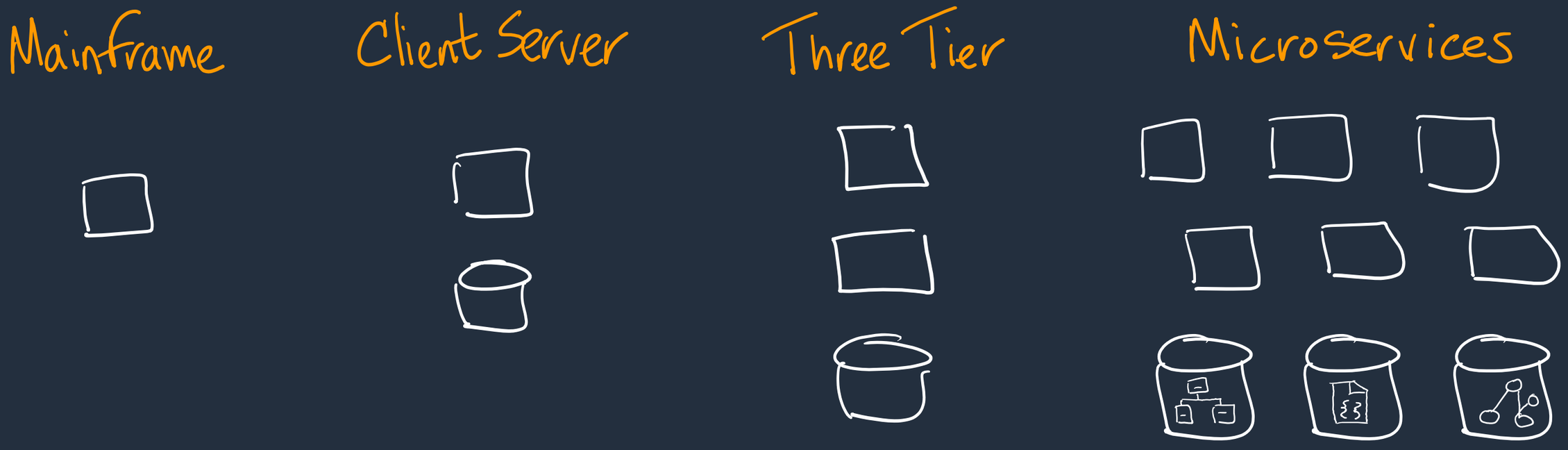
Hands-on lab

Quiz

Wrap-up

aws

Application architecture and patterns have evolved



aws

Characteristics of internet-scale apps

E-commerce | Media streaming | Social media | Online gaming | Shared economy

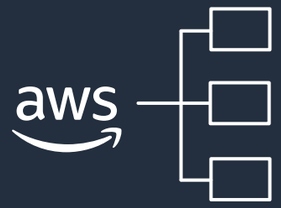
Users	1 million+
Data volume	TB, PB, EB
Locality	Global
Performance	Microsecond latency
Request rate	Millions per second
Access	Mobile, IoT, devices
Scale	Up and down
Economics	Pay as you go
Developer access	Instant API access

aws

DynamoDB

Performance at scale

- Handles millions of requests per second
- Delivers single-digit-millisecond latency
- Automated global replication
- New advanced streaming with Amazon Kinesis Data Streams for DynamoDB



No servers to manage

- Maintenance free
- Auto scaling
- On-demand capacity mode
- Change data capture for integration with AWS Lambda, Amazon Redshift, Amazon Elasticsearch Service
- AWS Glue Elastic Views for DynamoDB

Enterprise ready

- ACID transactions
- Encryption at rest
- Continuous backups (PITR), and on-demand backup and restore
- NoSQL Workbench
- Export table data to S3
- PartiQL (a SQL-compatible query language) support

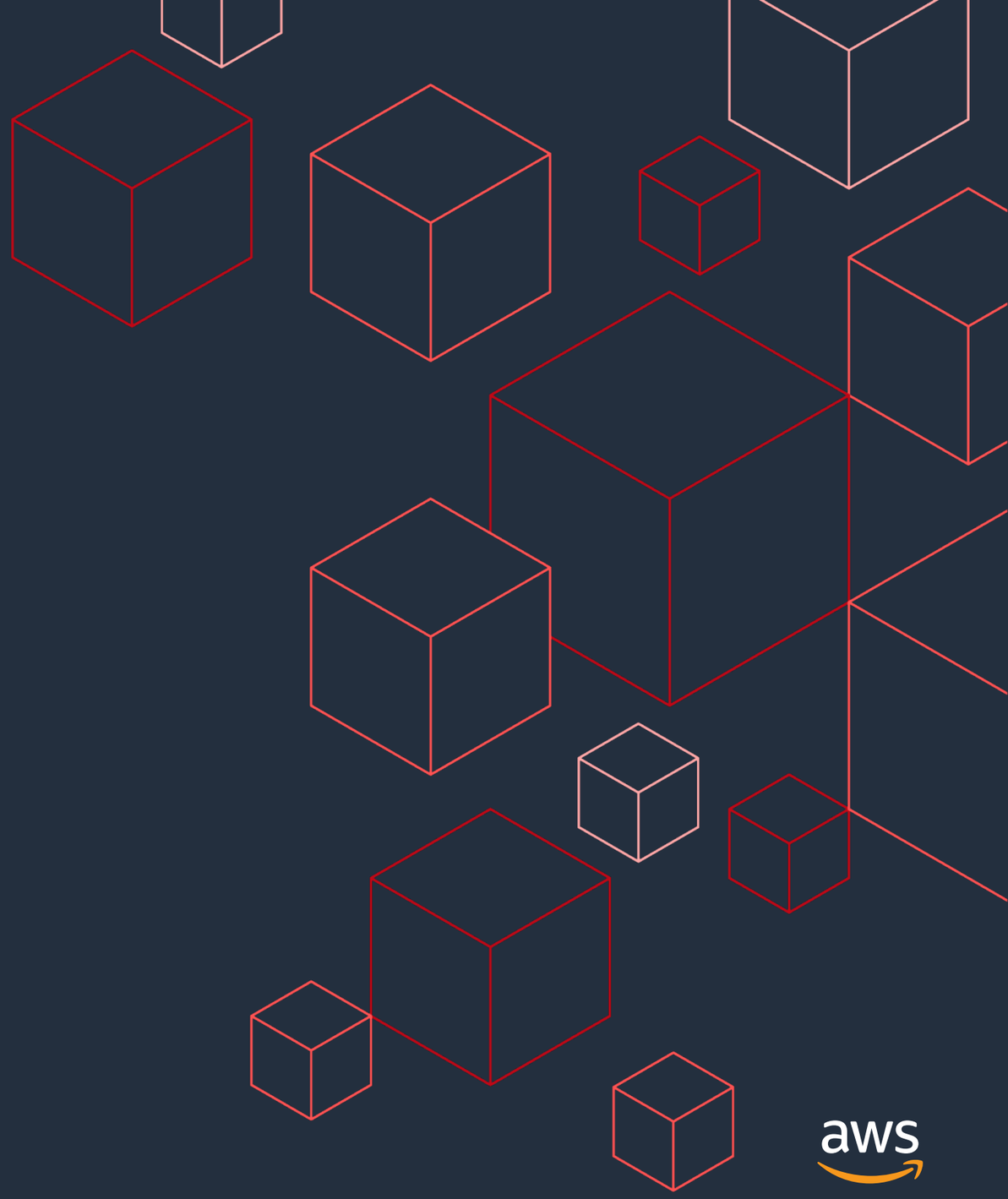
aws

More about DynamoDB features

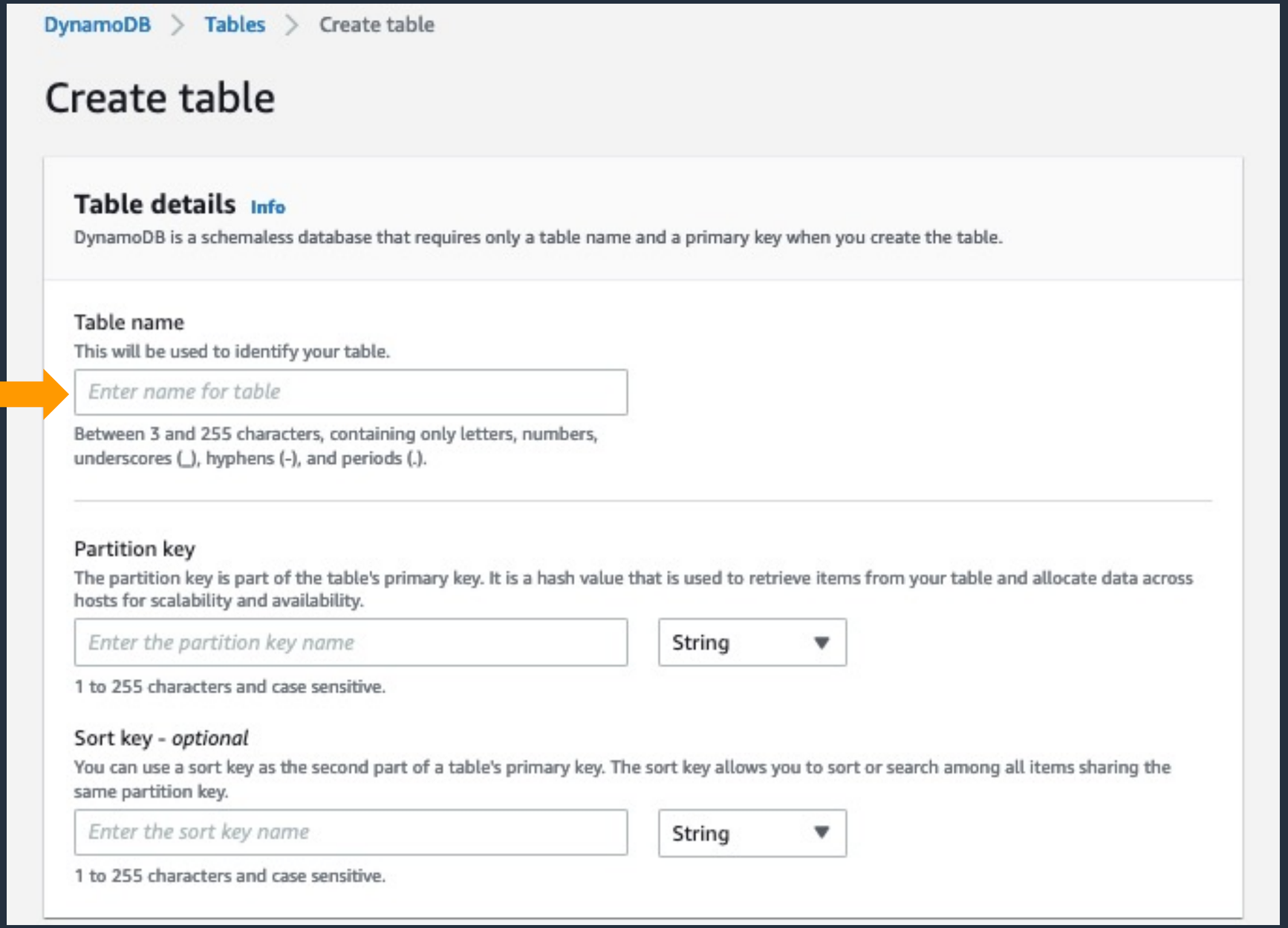
tinyurl.com/4dsa5jdn

aws

Key concepts



aws



Settings

- (●) **Default settings**
  The fastest way to create your table. You can modify these settings now or after your table has been created.
- ( ) **Customize settings**
  Use these advanced features to make DynamoDB work better for your needs.

Default settings

Read/write capacity **Info**

Using provisioned capacity mode. Read and write capacity are set to 5 units each with auto scaling enabled.

Secondary indexes **Info**

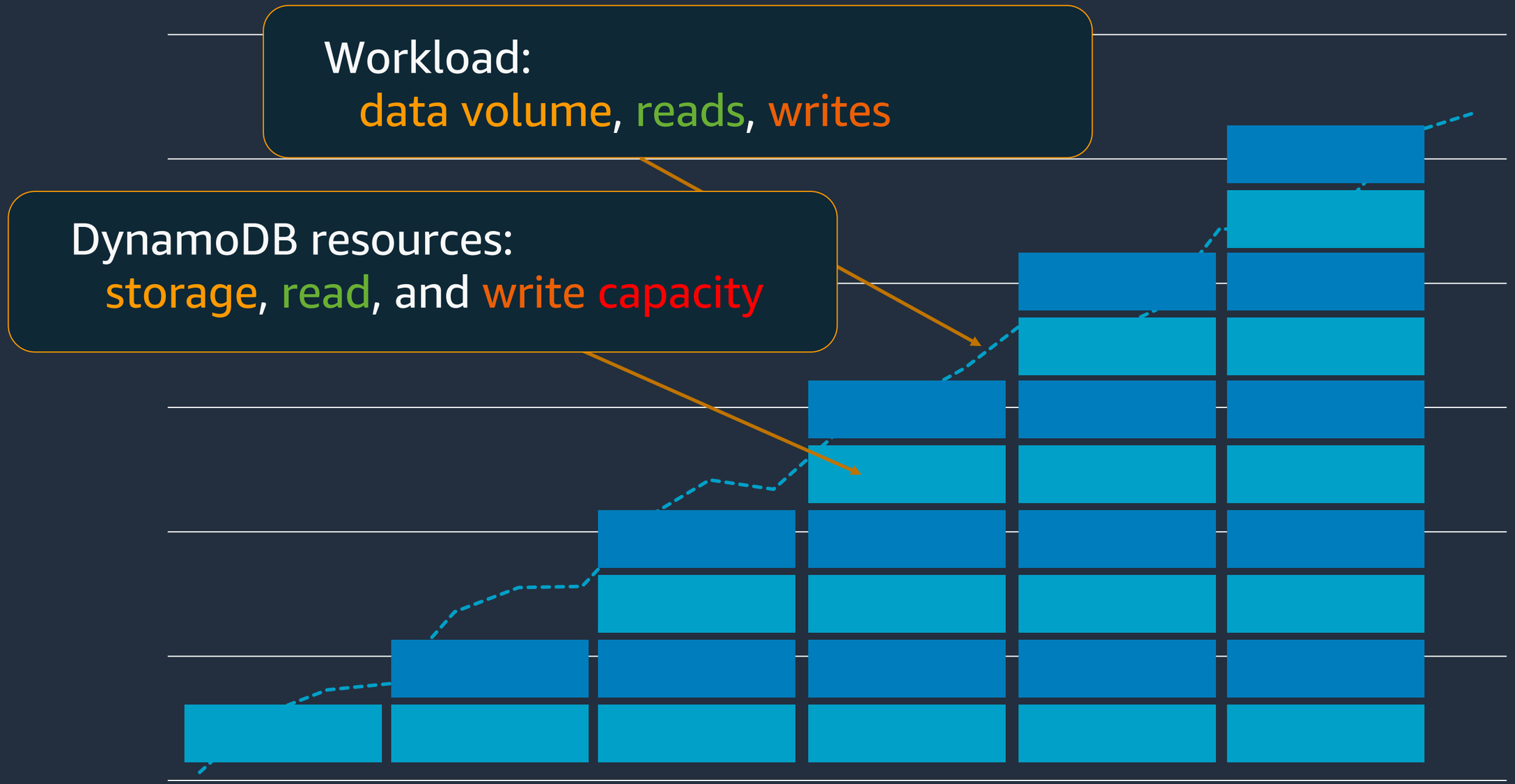
No secondary indexes have been created. Queries will be run by using the table's partition key and sort key only.

Key management for encryption at rest **Info**

Using the AWS owned customer master key. This key is managed by DynamoDB at no extra cost.

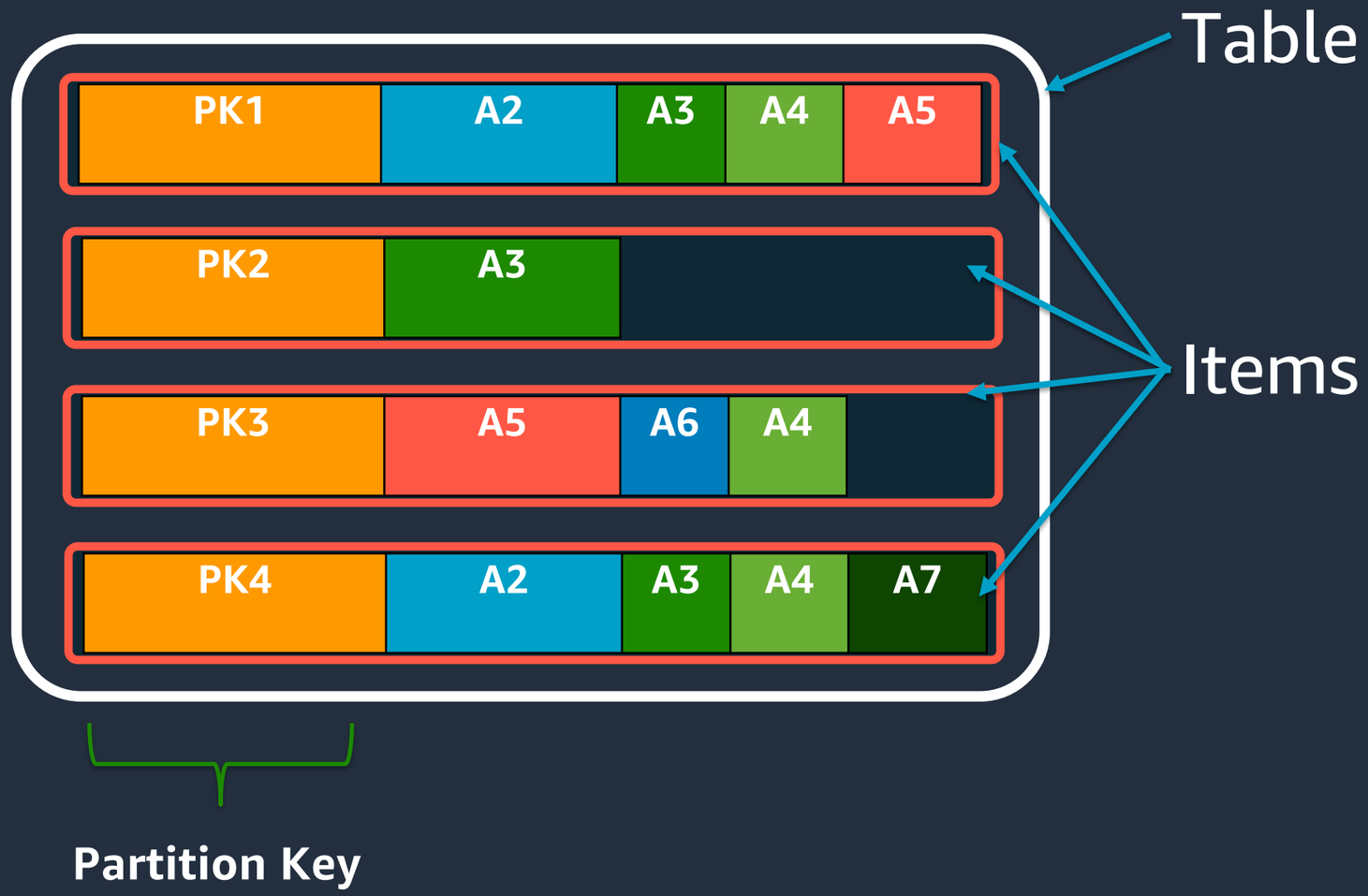
aws

Horizontal scaling with DynamoDB



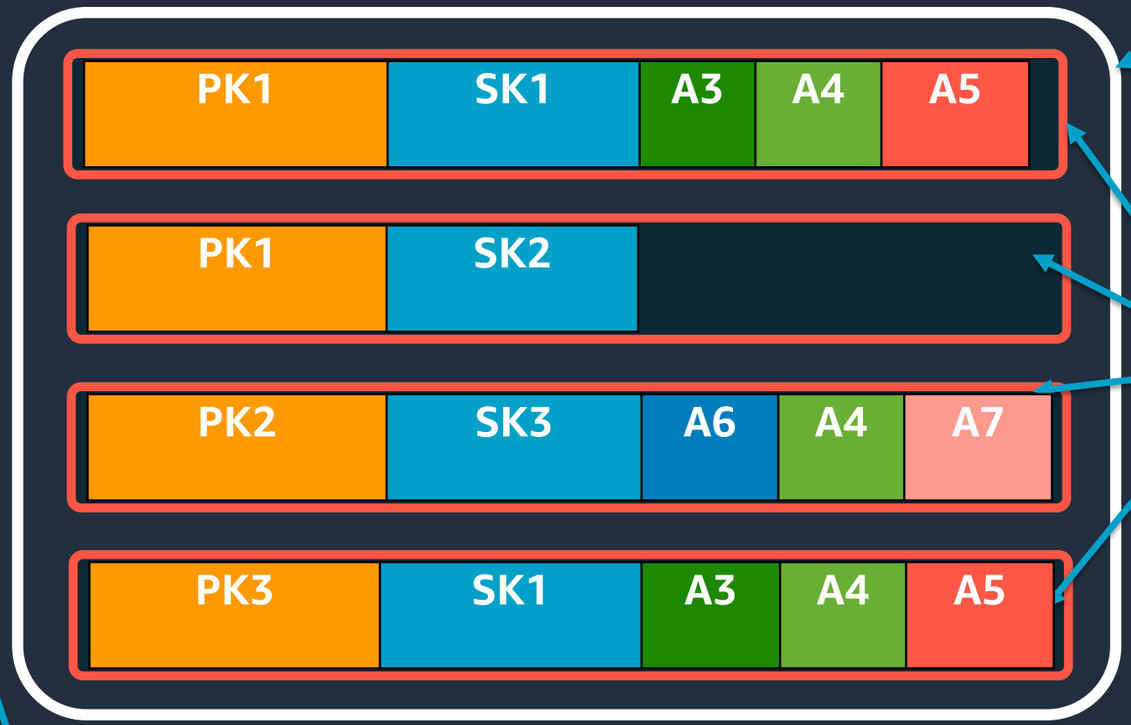
aws

DynamoDB Table



aws

DynamoDB Table



Mandatory
Key-value access pattern
Determines data distribution

All items for a partition key
==, <, >, >=, <=
“begins with”
“between”
sorted results
counts
top/bottom N values
paged responses

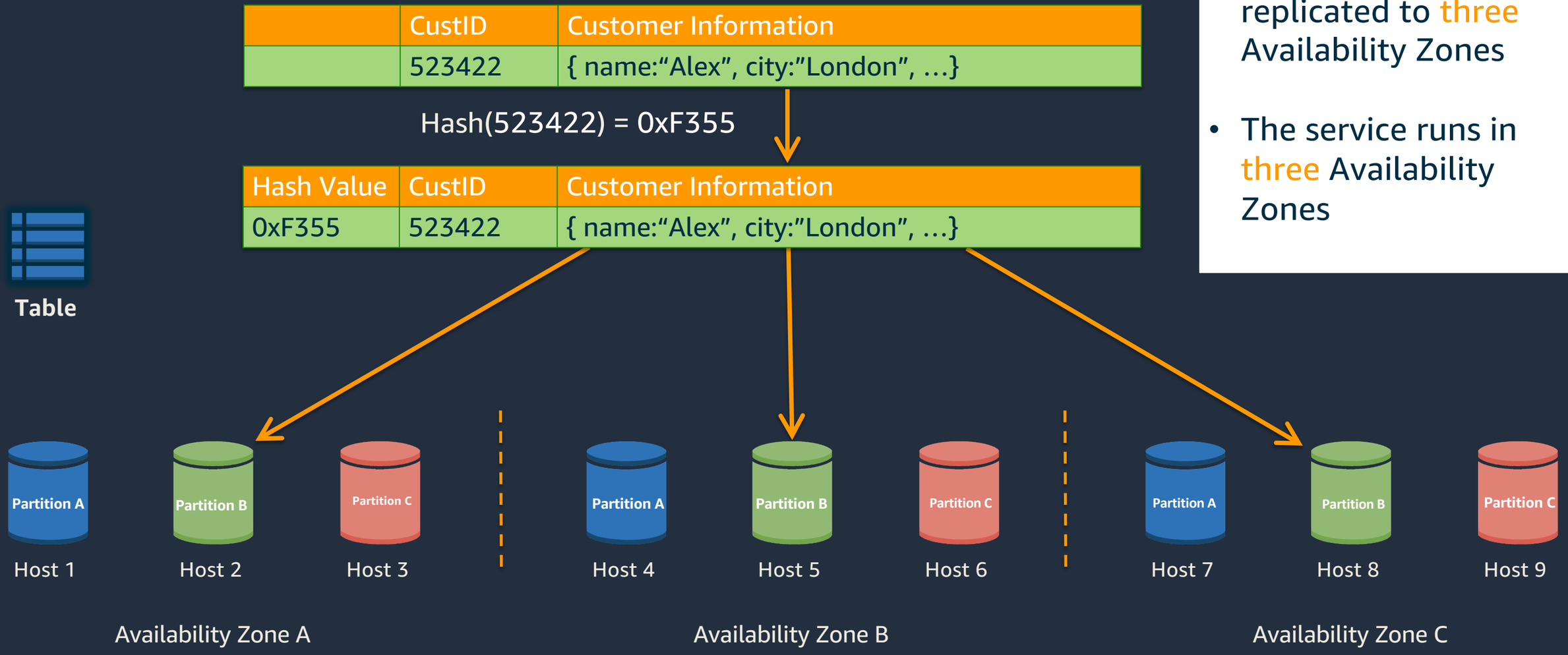
Optional
Model 1:N relationships
Enables rich query capabilities

aws

A view from “a different angle”

Three-way replication

- Data is always replicated to three Availability Zones
- The service runs in three Availability Zones

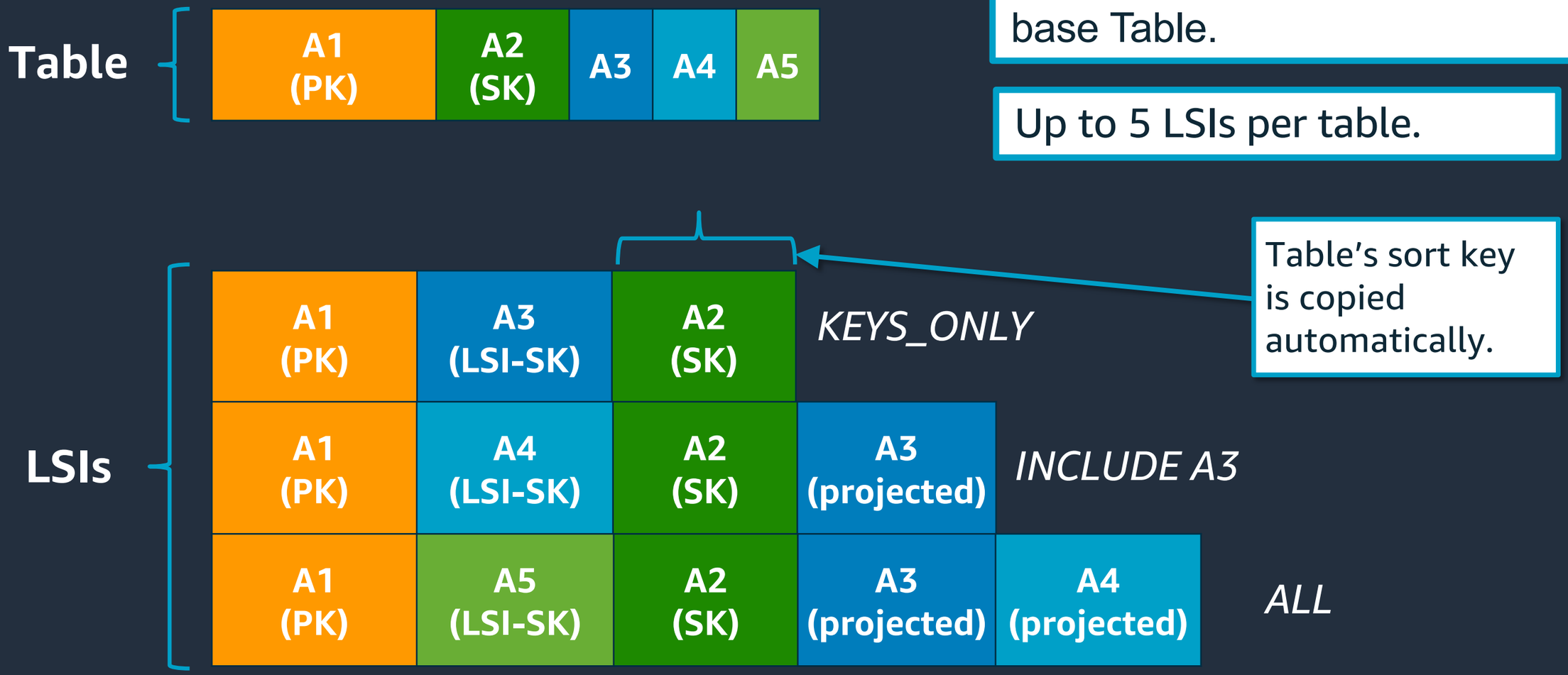


Local secondary index (LSI)

Added at table creation time.

RCUs/WCUs are consumed from base Table.

Up to 5 LSIs per table.

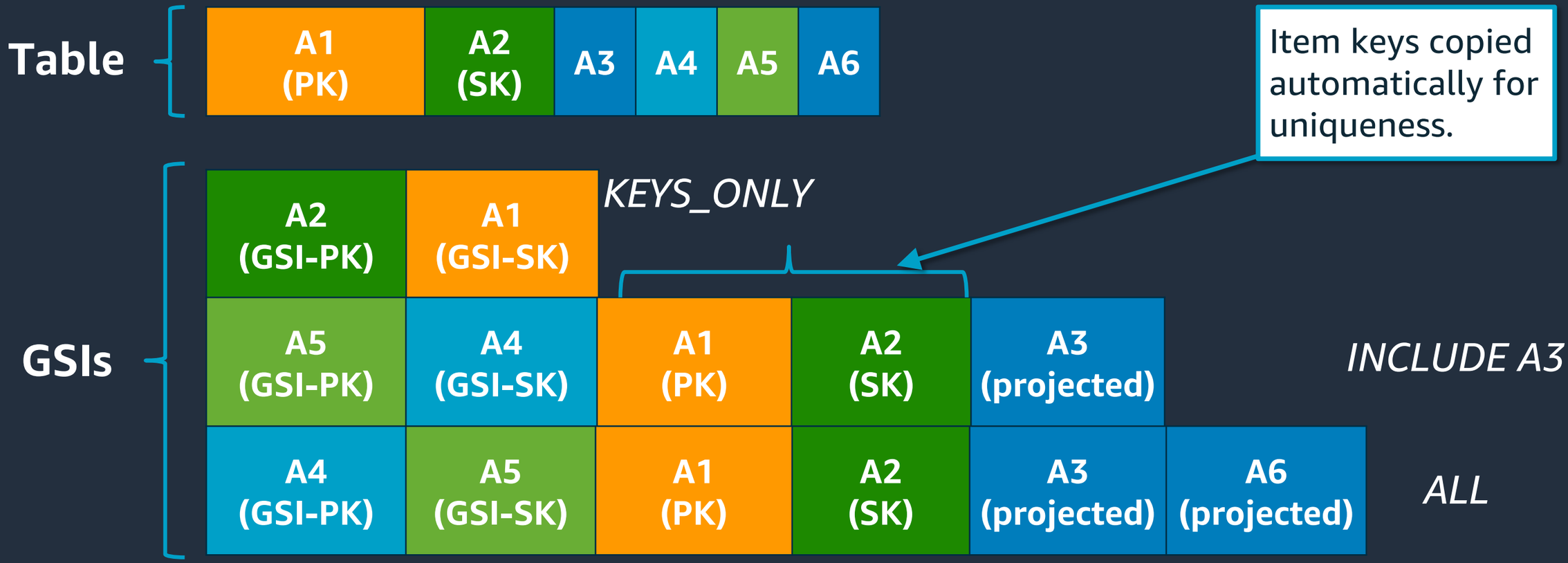


aws

Global secondary index (GSI)

Up to 20 GSIs per table.

RCUs/WCUs provisioned separately for GSIs.



aws

DynamoDB Types			Data Types
Scalars	Number	N	Integer Float Timestamp
	String	S	Timestamp Date (ISO 8601)
	Binary	B	Blob
	Boolean	BOOL	Boolean
	Null	NULL	Null
Document	List	L	List
	Map	M	Map
Set	String Set	SS	Set
	Number Set	NS	
	Binary Set	BS	

aws

DynamoDB Operations		Expressions
Working with specific Items	GetItem	Projection Expression
	BatchGetItem	Projection Expression
	TransactReadItems	Projection Expression
	PutItem	Condition Expression
	BatchWriteItem	
	UpdateItem	Condition Expression
		Update Expression
	DeleteItem	Condition Expression
	TransactWriteItems	Condition Expression

aws

DynamoDB Operations		Expressions
Working with an Item collection	Query	Key Condition Expression
		Filter Expression
		Projection Expression
Working with the entire table	Scan	Filter Expression
		Projection Expression

aws

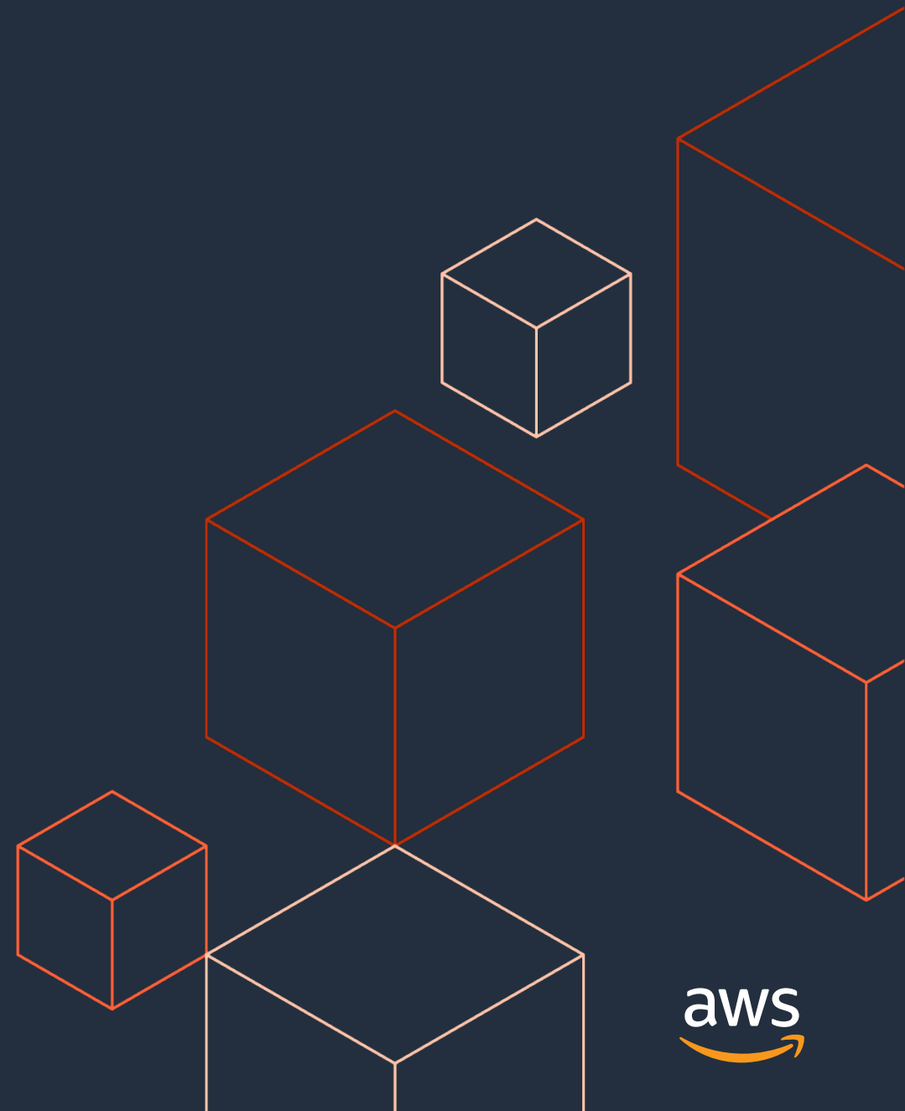
PartiQL APIs

Operations for PartiQL:

- ***ExecuteStatement***
    - Support for single or multiple item SELECT
    - Support for single item INSERT, UPDATE or DELETE
- ***BatchExecuteStatement***
    - Support for a batch of single item SELECT OR batch of single item INSERT/DELETE of up to 25 items
- ***ExecuteTransaction***
    - Support for a batch of single item SELECT OR batch of single item INSERT/UPDATE/DELETE/CONDITION CHECK of up to 25 items

aws

NoSQL data modeling



aws

All data is relational

IT monitoring

Social graph

Documents

Auth services

aws

Tenets of DynamoDB data modeling

- Understand the use case
- Identify the access patterns
  - Read/write workloads
  - Query dimensions and aggregations
- Data modeling
  - Using NoSQL design patterns
- Review -> Repeat -> Review

- Nature of the data
  - OLTP / OLAP / full-text search
- Relationships between the entities
- What does concurrent access look like?
- Time series data
- Archiving needs, etc.

aws

Tenets of DynamoDB data modeling

- Understand the use case
- Identify the access patterns
  - Read/write workloads
  - Query dimensions and aggregations
- Data modeling
  - Using NoSQL design patterns
- Review -> Repeat -> Review

- Source data analysis (write workload)
- Reading one item versus multiple items (read workload)
- Query aggregations and KPIs

aws

Tenets of DynamoDB data modeling

- Understand the use case
- Identify the access patterns
  - Read/write workloads
  - Query dimensions and aggregations
- Data modeling
  - Using NoSQL design patterns
- Review -> Repeat -> Review

- 1:1, 1:n, m:n relationships
- **1 application = 1 table**
  - Avoid unnecessary fetches
  - Simplify access patterns
- Identify primary key
  - Partition key and Sort key
- Query dimensions using LSIs and GSIs

aws

Tenets of DynamoDB data modeling

- Understand the use case
- Identify the access patterns
  - Read/write workloads
  - Query dimensions and aggregations
- Data modeling
  - Using NoSQL design patterns
- Review -> Repeat -> Review

aws

Use case Discovery Phase

Online Shop

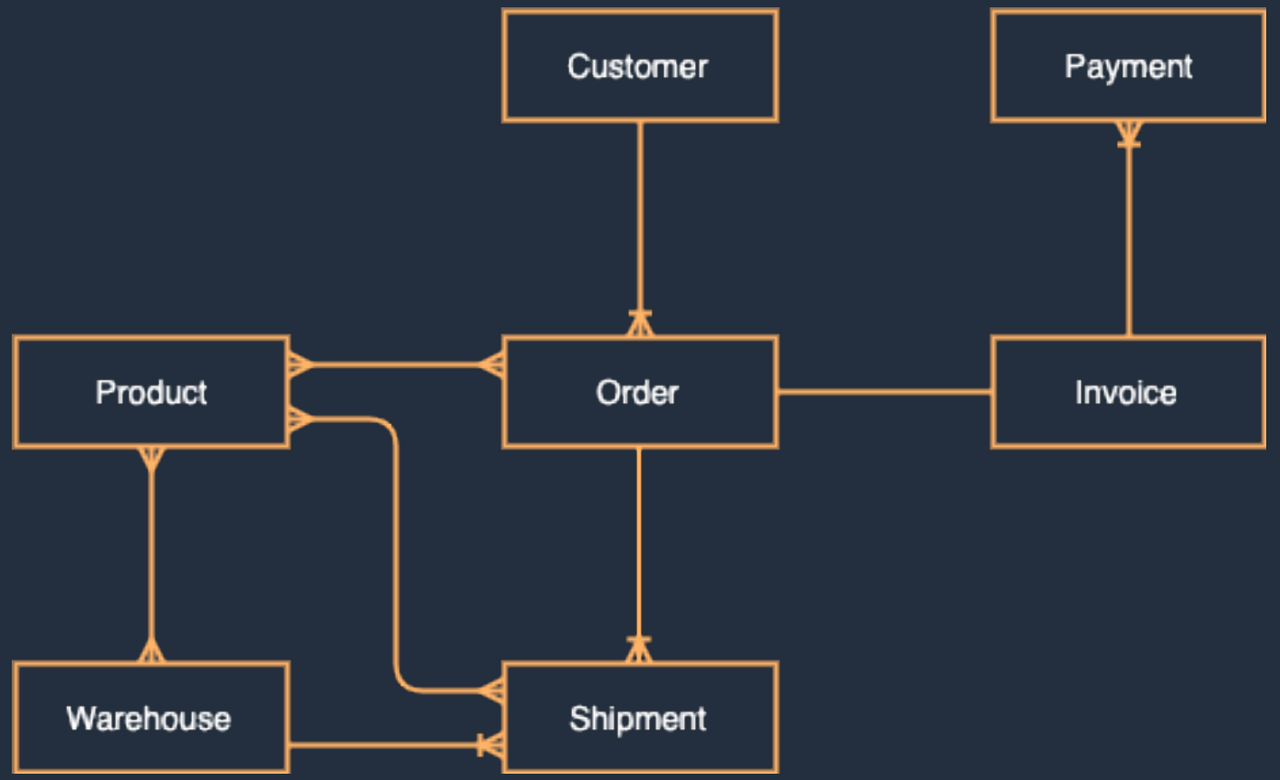
…

A customer visits an online shop, browses through different products and places an order for some of the products. Based on the invoice, customer can pay using discount code or gift card and pay for the remaining amount by credit card. Purchased products will be picked from one or several warehouses and will be shipped to the provided address.

…

aws

Entity Relation Diagram



aws

Identified Access Patterns
Get customer for a given customerId
Get product for a given productId
Get warehouse for a given warehouseId
Get a product inventory for all warehouses by a productId
Get order for a given orderId
Get all products for a given orderId
Get invoice for a given orderId
Get all shipments for a given orderId
Get all orders for a given productId for a given date range
Get invoice for a given invoiceId
Get all payments for a given invoiceId
Get shipment detail for a given shipmentId
Get all shipments for a given warehouseId
Get inventory of all products for a given warehouseId
Get all invoices for a given customerId for a given date range
Get all products ordered by a given customerId for a given date range

aws

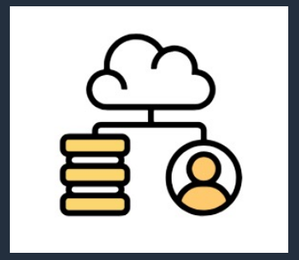
Access Patterns	Table/GSI/LSI	Key Condition	Filter Expression
Get customer for a given customerId			
Get product for a given productId			
Get warehouse for a given warehouseId			
Get a product inventory for all warehouses by a productId			
Get order for a given orderId			
Get all products for a given orderId			
Get invoice for a given orderId			
Get all shipments for a given orderId			
Get all orders for a given productId for a given date range			
Get invoice for a given invoiceId			
Get all payments for a given invoiceId			
Get shipment detail for a given shipmentId			
Get all shipments for a given warehouseId			
Get inventory of all products for a given warehouseId			
Get all invoices for a given customerId for a given date range			
Get all products ordered by a given customerId for a given date range			

aws

NoSQL Workbench

Data modeler

Visualizer



Operation builder

A **client-side application** that helps you build scalable, high-performance data models

Simplifies query development and testing

A rich GUI-based tool that **helps you visualize data models** and **perform DynamoDB operations**

Available for **Windows, macOS, and Linux**

aws

NoSQL Workbench

To download NoSQL Workbench:

Step-by-step data modeling examples:

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

https://github.com/aws-samples/amazon-dynamodb-design-patterns

aws

1:M Order to Order Item

Primary key		Attributes				
Partition key: PK	Sort key: SK					
o#12345	c#12345	EntityType	Date			
		order	2021-04-20T09:10:00Z			
	p#12345	EntityType	GSI1-PK	GSI1-SK	Price	Quantity
		orderItem	p#12345	2021-04-20T09:18:00Z	100	2
	p#99887	EntityType	GSI1-PK	GSI1-SK	Price	Quantity
		orderItem	p#99887	2021-04-20T09:20:00Z	40	5
o#54321	c#12345	EntityType	Date			
		order	2021-01-19T09:18:00Z			
	p#12345	EntityType	GSI1-PK	GSI1-SK	Pric	
		orderItem	p#12345	2021-01-19T09:18:00Z	100	
	p#34567	EntityType	GSI1-PK	GSI1-SK	Pric	
		orderItem	p#34567	2021-01-19T09:18:00Z	50	

1:M OrderItem to Order

Primary key		Attributes				
Partition key: GSI1-PK	Sort key: GSI1-SK					
p#12345	2021-01-19T09:18:00Z	PK	SK	EntityType	Price	Quantity
		o#54321	p#12345	orderItem	100	2
	2021-04-20T09:18:00Z	PK	SK	EntityType	Price	Quantity
		o#12345	p#12345	orderItem	100	2
p#99887	2021-04-20T09:20:00Z	PK	SK	EntityType	Price	Quantity
		o#12345	p#99887	orderItem	40	5
p#34567	2021-01-19T09:18:00Z	PK	SK	EntityType	Price	Quantity
		o#54321	p#34567	orderItem	50	4

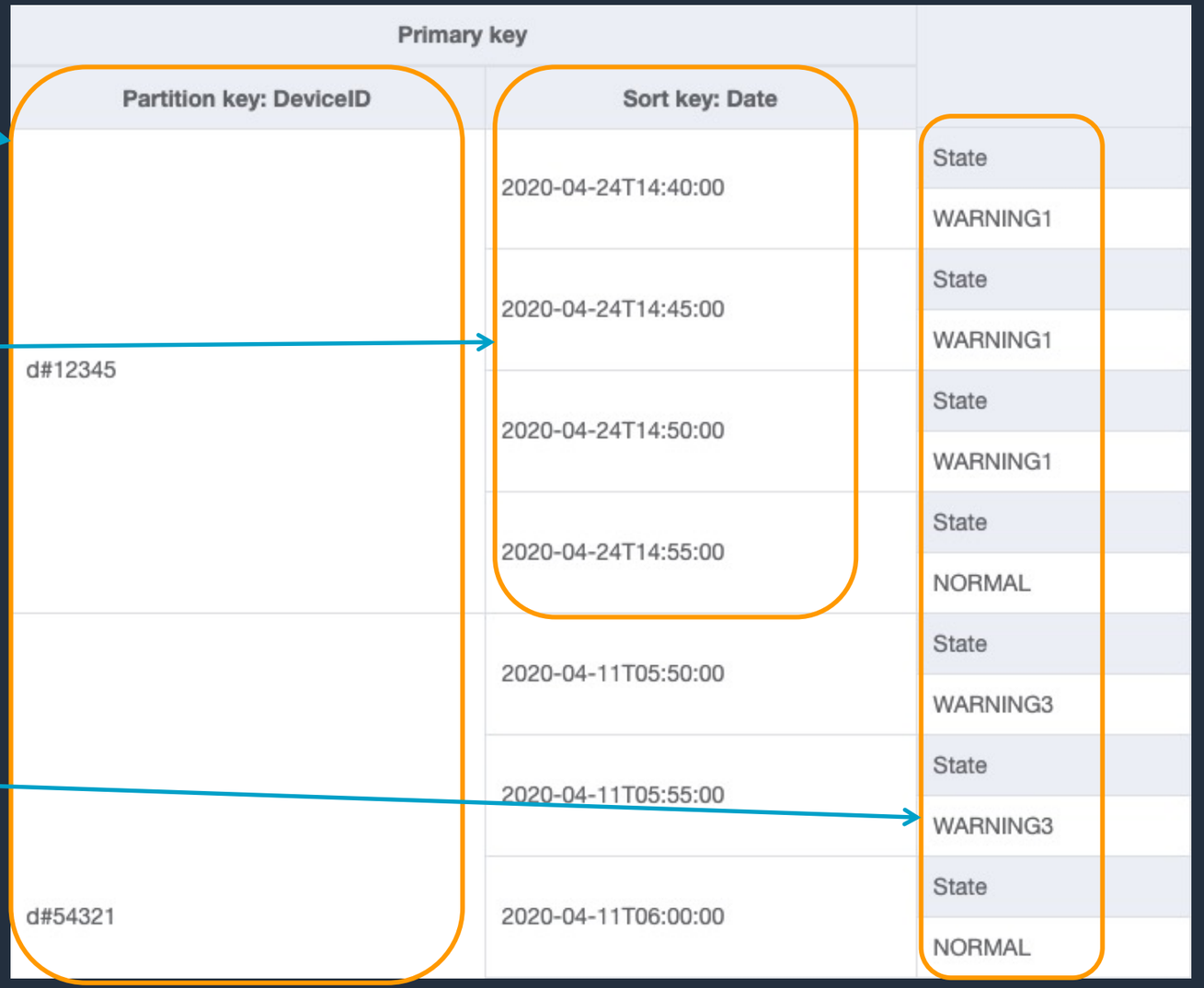
aws

Example – Device State Log

High cardinality

Sorted by date time

Filter by Status



Primary key		
Partition key: DeviceID	Sort key: Date	
d#12345	2020-04-24T14:40:00	State
		WARNING1
	2020-04-24T14:45:00	State
		WARNING1
	2020-04-24T14:50:00	State
		WARNING1
	2020-04-24T14:55:00	State
		NORMAL
d#54321	2020-04-11T05:50:00	State
		WARNING3
	2020-04-11T05:55:00	State
		WARNING3
	2020-04-11T06:00:00	State
		NORMAL

aws

Access Pattern: Get all logs for a specific device state showing the most recent logs first

```
SELECT * FROM DeviceLog
WHERE DeviceID = ‘d#12345'
ORDER BY Date DESC
FILTER ON State=‘WARNING1'
```

Returned →

Filtered →

Primary key		
Partition key: DeviceID	**Sort key: Date**	
d#12345	2020-04-24T14:40:00	State
		WARNING1
	2020-04-24T14:45:00	State
		WARNING1
	2020-04-24T14:50:00	State
		WARNING1
	2020-04-24T14:55:00	State
		NORMAL
d#54321	2020-04-11T05:50:00	State
		WARNING3
	2020-04-11T05:55:00	State
		WARNING3
	2020-04-11T06:00:00	State
		NORMAL

```
aws dynamodb query
    --table-name DeviceStateLog
    --key-condition-expression "#dID = :dID"
    --no-scan-index-forward
    --filter-expression "#s = :s"
    --return-consumed-capacity TOTAL
    --expression-attribute-names '{"#dID":"DeviceID","#s":"State"}’
    --expression-attribute-values '{":dID":{"S":"d#12345"},":s":{"S":"WARNING1"}}’
```


aws

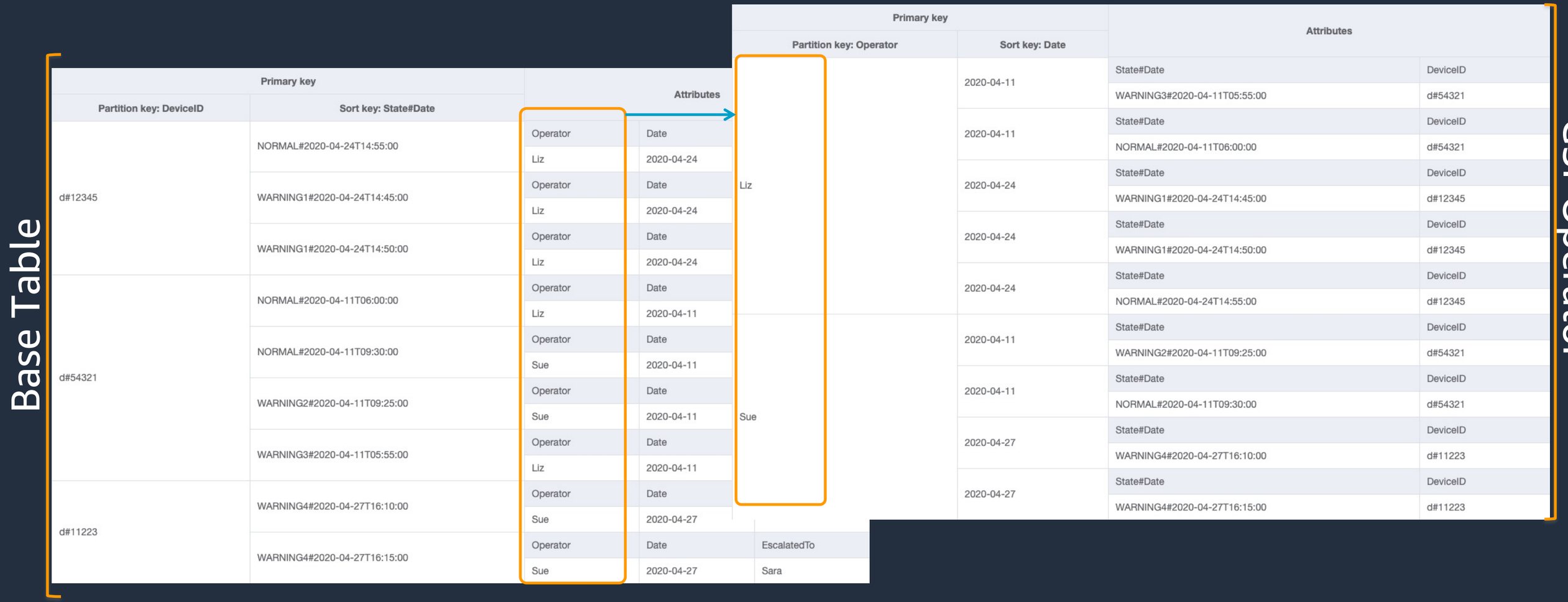
Use Composite Sort Key Instead

Primary key	
Partition key: DeviceID	Sort key: State#Date
d#12345	NORMAL#2020-04-24T14:55:00
	WARNING1#2020-04-24T14:40:00
	WARNING1#2020-04-24T14:45:00
	WARNING1#2020-04-24T14:50:00
d#54321	NORMAL#2020-04-11T06:00:00
	NORMAL#2020-04-11T09:30:00
	WARNING2#2020-04-11T09:25:00
	WARNING3#2020-04-11T05:50:00
	WARNING3#2020-04-11T05:55:00

```
aws dynamodb query
    --table-name DeviceStateLog
    --no-scan-index-forward
    --key-condition-expression "#dID = :dID AND begins_with(#s, :sd)”
    --expression-attribute-names '{"#dID":"DeviceID","#s":"State#Date"}’
    --expression-attribute-values '{":dID":{"S":"d#12345"},":sd":{"S":"WARNING1#"}}’
    --return-consumed-capacity TOTAL
```


aws

Access Pattern: Get all device logs for a given operator between two dates



Access Pattern: Get all device logs for a given operator between two dates

Primary key		Attributes	
Partition key: Operator	Sort key: Date		
	2020-04-11	State#Date	DeviceID
		WARNING3#2020-04-11T05:55:00	d#54321
	2020-04-11	State#Date	DeviceID
		NORMAL#2020-04-11T06:00:00	d#54321
Liz	2020-04-24	State#Date	DeviceID
		WARNING1#2020-04-24T14:45:00	d#12345
	2020-04-24	State#Date	DeviceID
		WARNING1#2020-04-24T14:50:00	d#12345
	2020-04-24	State#Date	DeviceID
		NORMAL#2020-04-24T14:55:00	d#12345
	2020-04-11	State#Date	DeviceID
		WARNING2#2020-04-11T09:25:00	d#54321
	2020-04-11	State#Date	DeviceID
Sue		NORMAL#2020-04-11T09:30:00	d#54321
	2020-04-27	State#Date	DeviceID
		WARNING4#2020-04-27T16:10:00	d#11223
	2020-04-27	State#Date	DeviceID
		WARNING4#2020-04-27T16:15:00	d#11223

GSI – Operator

```
aws dynamodb query
    --table-name DeviceStateLog
    --index-name GSI1
    --key-condition-expression "#op = :op AND #d between :d1 AND :d2"
    --expression-attribute-names '{"#op": "Operator" , "#d": "Date"}'
    --expression-attribute-values '{":op": {"S":"Liz"} , ":d1": {"S":"2020-04-20"}, ":d2":{"S":"2020-04-25"}}'
    --return-consumed-capacity TOTAL
```


aws

Access Pattern: Get all escalated logs for a given supervisor

Base Table

Primary key		Attributes	
Partition key: DeviceID	Sort key: State#Date		
d#12345	NORMAL#2020-04-24T14:55:00	Operator	Date
		Liz	2020-04-24
	WARNING1#2020-04-24T14:45:00	Operator	Date
		Liz	2020-04-24
	WARNING1#2020-04-24T14:50:00	Operator	Date
		Liz	2020-04-24
d#54321	NORMAL#2020-04-11T06:00:00	Operator	Date
		Liz	2020-04-11
	NORMAL#2020-04-11T09:30:00	Operator	Date
		Sue	2020-04-11
	WARNING2#2020-04-11T09:25:00	Operator	Date
		Sue	2020-04-11
	WARNING3#2020-04-11T05:55:00	Operator	Date
		Liz	2020-04-11
d#11223	WARNING4#2020-04-27T16:10:00	Operator	Date
		Sue	2020-04-27
	WARNING4#2020-04-27T16:15:00	Operator	Date
		Sue	2020-04-27

GSI-Supervisor

Primary key		Attributes	
Partition key: EscalatedTo	Sort key: State#Date		
Sara	WARNING4#2020-04-27T16:15:00	DeviceID	Operator
		d#11223	Sue

EscalatedTo
Sara

Sparse GSI: Only items that match the GSI index are projected.

Good for:

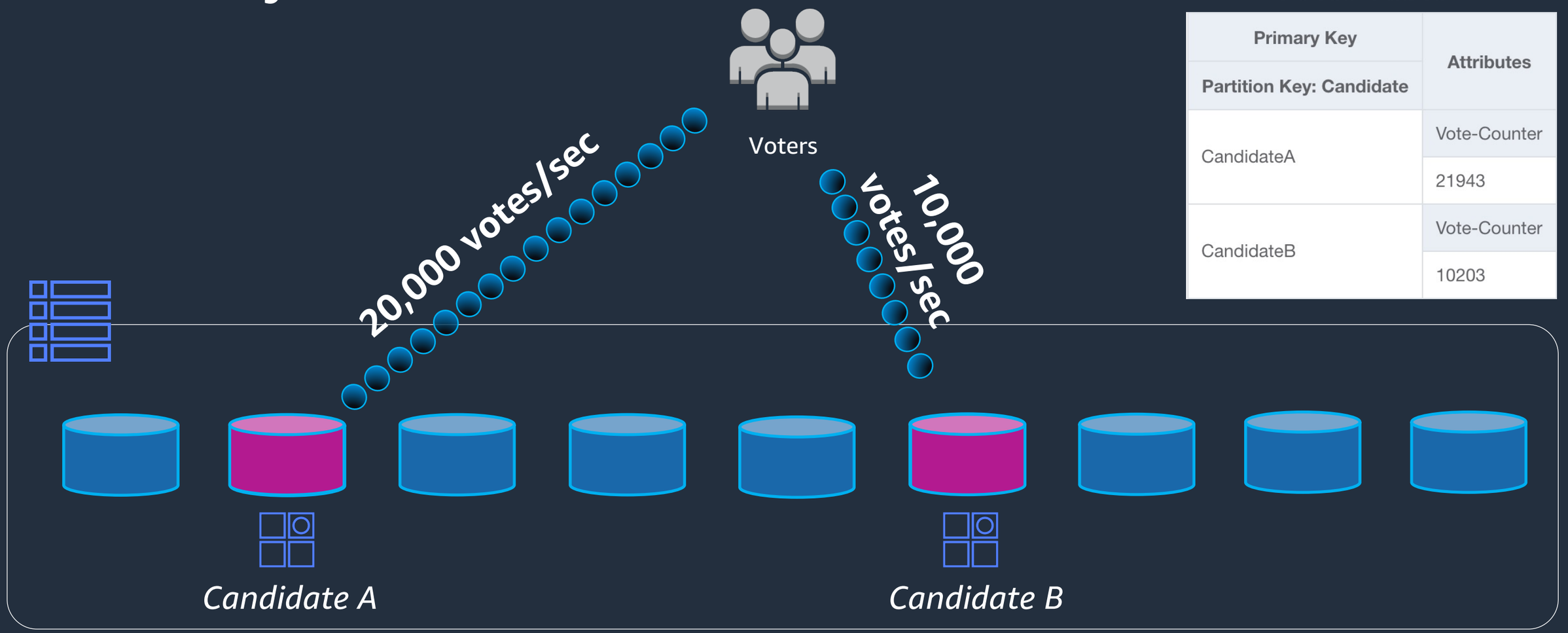
- ‘Needle in the haystack’
- Cost effective ‘scans’
- Item management

aws

Example – Voting: Scaling high write throughput & low cardinality…

Votes Table

Primary Key Partition Key: Candidate	Attributes
CandidateA	Vote-Counter
	21943
CandidateB	Vote-Counter
	10203



Single Item limit: 1000 WCUs

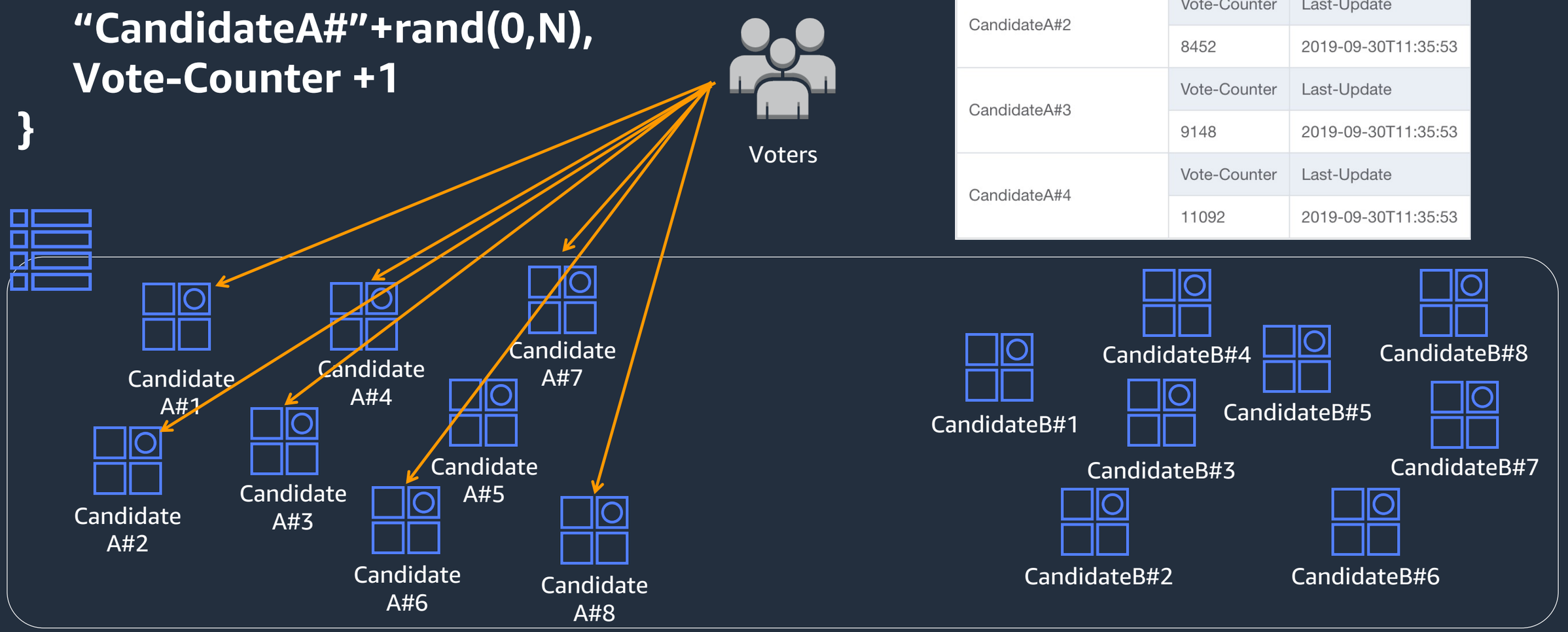
aws

Write shard the partition key…

UPDATE Item :

```
{
    “CandidateA#”+rand(0,N),
    Vote-Counter +1
}
```

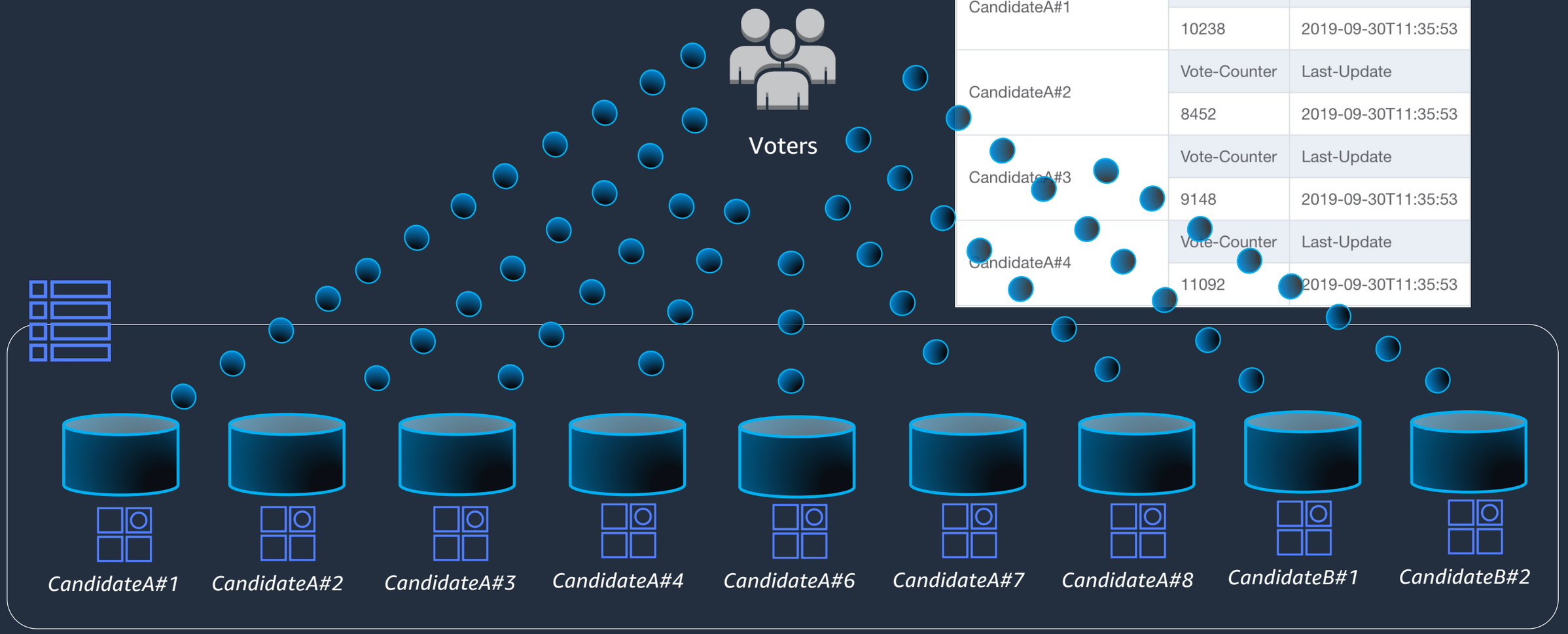
Primary Key	Attributes	
Partition Key: Candidate		
CandidateA#1	Vote-Counter	Last-Update
	10238	2019-09-30T11:35:53
CandidateA#2	Vote-Counter	Last-Update
	8452	2019-09-30T11:35:53
CandidateA#3	Vote-Counter	Last-Update
	9148	2019-09-30T11:35:53
CandidateA#4	Vote-Counter	Last-Update
	11092	2019-09-30T11:35:53



aws

Even Write Distribution

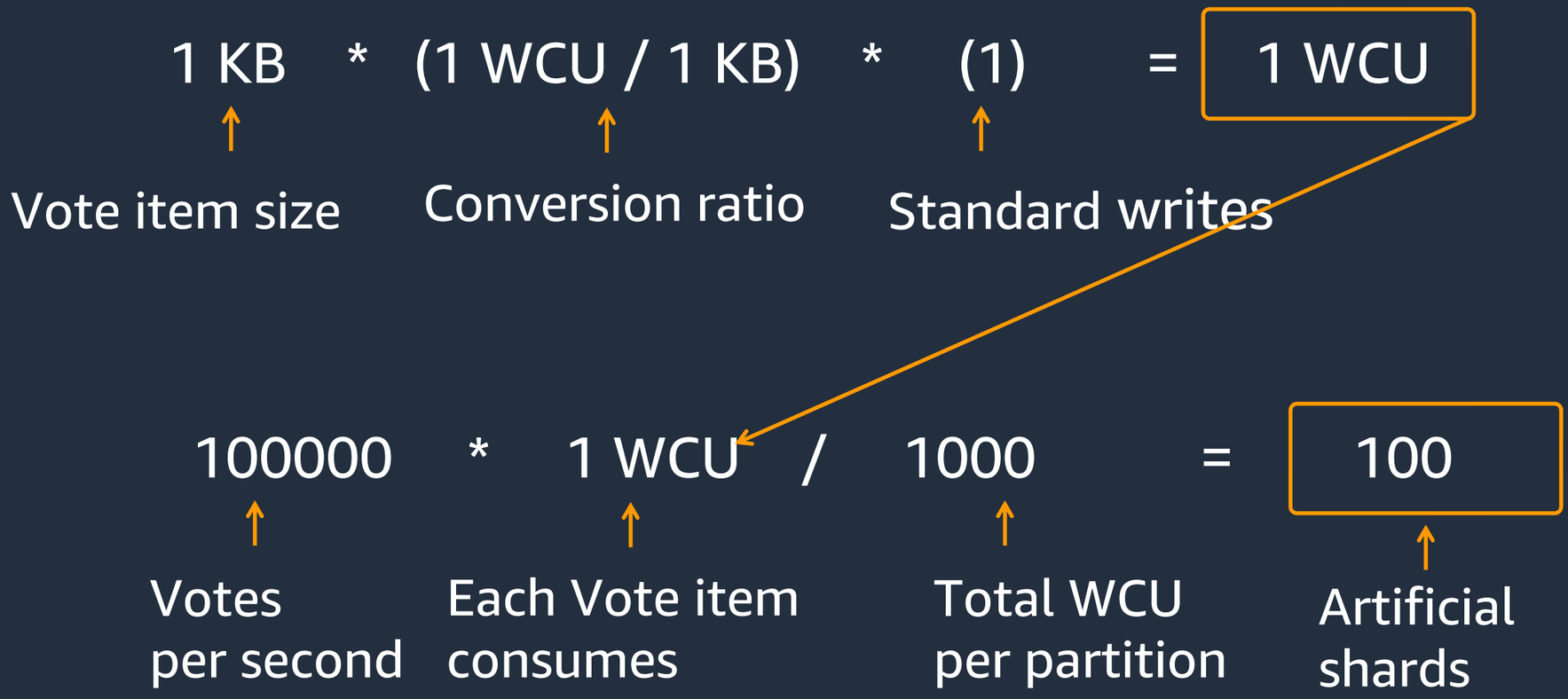
Primary Key	Attributes	
Partition Key: Candidate		
CandidateA#1	Vote-Counter	Last-Update
	10238	2019-09-30T11:35:53
CandidateA#2	Vote-Counter	Last-Update
	8452	2019-09-30T11:35:53
CandidateA#3	Vote-Counter	Last-Update
	9148	2019-09-30T11:35:53
CandidateA#4	Vote-Counter	Last-Update
	11092	2019-09-30T11:35:53



Item Throughput = N * 1000 WCUs

aws

How many artificial shards? *(writes)*



aws

Retrieve result: *(in parallel)*

1. Collect

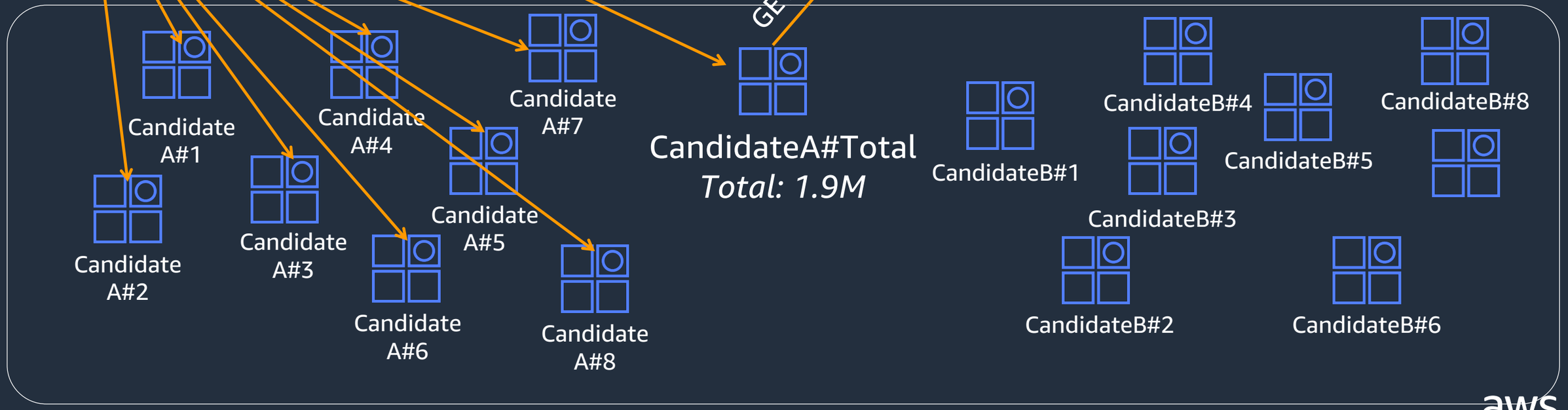
2. Store

UPDATE: **CandidateA#Total + *N***

3. Retrieve

GET: **CandidateA#Total**

Primary Key Partition Key: Candidate	Attributes		
CandidateA#Total	Vote-Counter	Last-Update	Total
	28692	2019-09-30T11:35:53	28692
CandidateA#2	Vote-Counter	Last-Update	
	8452	2019-09-30T11:35:53	
CandidateA#3	Vote-Counter	Last-Update	
	9148	2019-09-30T11:35:53	
CandidateA#4	Vote-Counter	Last-Update	
	11092	2019-09-30T11:35:53	



aws

Example – Shopping Cart : Document Indexing

```
{
        ”UserProfile" : {
                "FirstName": "Paul",
                "LastName": "Atreides",
                "DateJoined":"1965-08-01"},
        ”Store" : {
                ”store_id": ”STOREUID",
                 "city": "Los Angeles",
                 "zip_code":"90029”}
        "ShoppingCart" : [
                {"Spice":
                        { "SKU": "SpiceSKU",
                          "CategoryID":"FictionalSpice",
                          "DateAddded " :"2019-06-11"}},
                {"EspressoBeans":
                        { "SKU": "CaffeineSKU",
                          "CategoryID":"FOODANDDRINK",
                          "DateAddded " :"2019-06-10"}}],
        "ShippingAddress" : {
                 "street_address": "1234 Arrakis Dr",
                 "city": "Los Angeles",
                 "zip_code":"90029",
                 "status": "default"}
        ”OrderHistory#OrderUID" : {
                ”ProductA": ”SKU_A",
                ”ProductB": ”SKU_B",
                "DateOrdered":”2018-09-28"}
}
```


aws

Vertical partitioning

Primary key		Attributes	
Partition key: PK	Sort key: SK		
UserID	Address#USA#CA#LA#90029	data	GSI-SK
		"Street Address"	Default
	Cart#ACTIVE#Coffee	data	GSI-SK
		CoffeeSKU	2019-11-27T103324
	Cart#ACTIVE#Spice	data	GSI-SK
		SpiceSKU	2019-11-28T091245
	Cart#SAVED#Cocoa	data	GSI-SK
		CocoaSKU	2019-11-28T125642
	OrderHistory#OrderUID	data	GSI-SK
		{Order:DataMap}	2019-10-08T132612
	ProfileName	data	
		"Paul Atreides"	
	Store#StoreUID	data	GSI-SK
		Los Angeles	Active

Selectively query ‘nested’ attributes

Fetch items for a specific user that are active in the shopping cart.

BEGINS_WITH ‘Cart#ACTIVE'

- Optimize object size
- Selective Queries
- Reduce capacity and cost
- Improve App performance
- Overload GSI on SK…

aws

Takeaways

- Build targeted Queries using composite keys
- Start with Sort Key conditions
- Supplement with Filter Expressions
- Use Global Secondary Index to create new item collections

Find these examples at:

https://github.com/aws-samples/amazon-dynamodb-design-patterns

aws

Please complete the survey!

https://eventbox.dev/survey/XHCETBL

We value your feedback.

Thank you!

Visit DynamoDB.com

Follow @DynamoDB on