Advanced Design Patterns for Amazon DynamoDB

Sean Shriver, AWS DynamoDB SA
April, 2020

aws

1

What to Expect from the Session?

- DynamoDB 101
- Tenants of NoSQL Data Modeling
  - Normalized versus Denormalized schemas
- Common Design Patterns
  - Working Backwards: Building Queries
  - Indexes, Filters, and Hierarchies
  - Write Sharding, Distributed Transactions, & Complex Queries
- DynamoDB in the AWS Ecosystem

aws

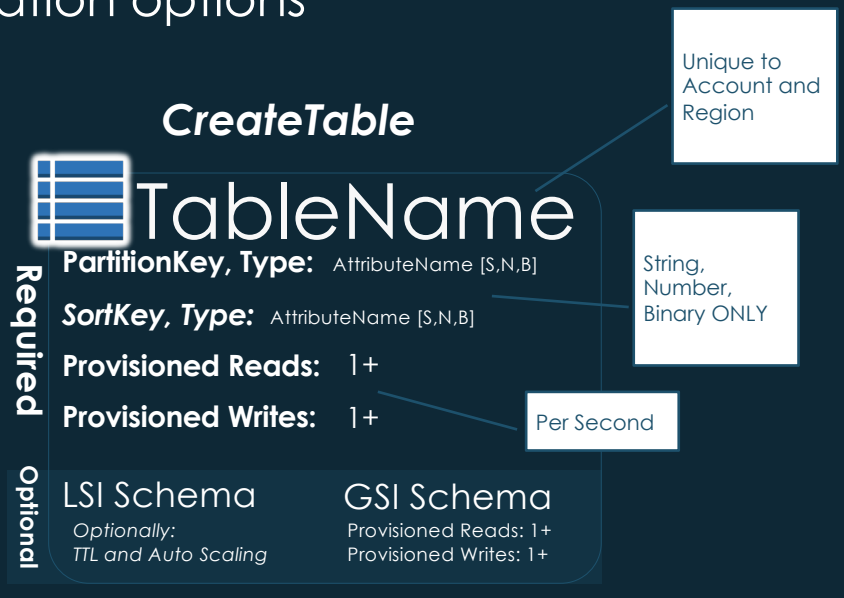
2

DynamoDB 101

aws

3

Table creation options



aws

4

Provisioned capacity

Provisioned capacity

Capacity is per second, rounded up to the next whole number

Read Capacity Unit (RCU)

1 RCU returns 4KB of data for strongly consistent reads, or double the data at the same cost for eventually consistent reads

Write Capacity Unit (WCU)

1 WCU writes 1KB of data, and *each item consumes 1 WCU minimum*

aws

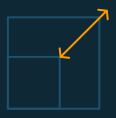
5

On demand capacity

Read/write capacity on demand

No capacity planning

No need to specify how much read/write throughput you expect to use



Ideal for unpredictable workloads

Ramp from zero to tens of thousands of requests per second on demand

Pay only for what you use

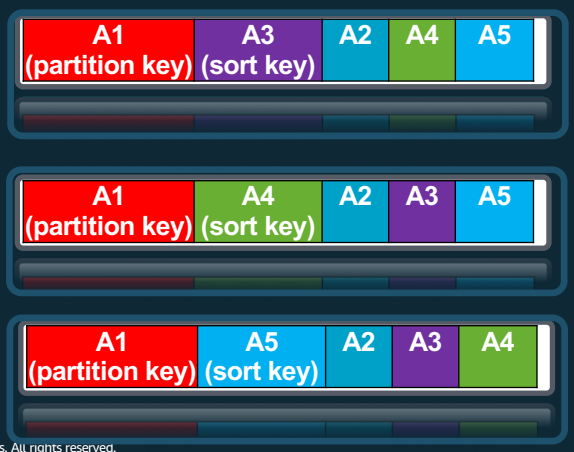
Pay-per-request pricing

aws

6

Local Secondary Indexes

- Alternate sort key attribute
- Index is local to a partition key



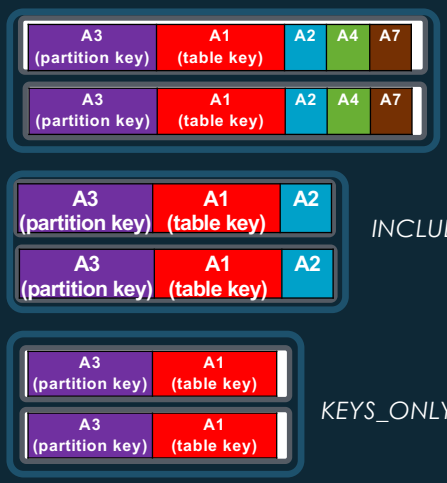
10 GB max per partition key, i.e. LSIs limit the # of sort keys!

aws

7

Global Secondary Indexes

- Alternate partition (+sort) key
- Index is across all table partition keys
- Can be added or removed anytime

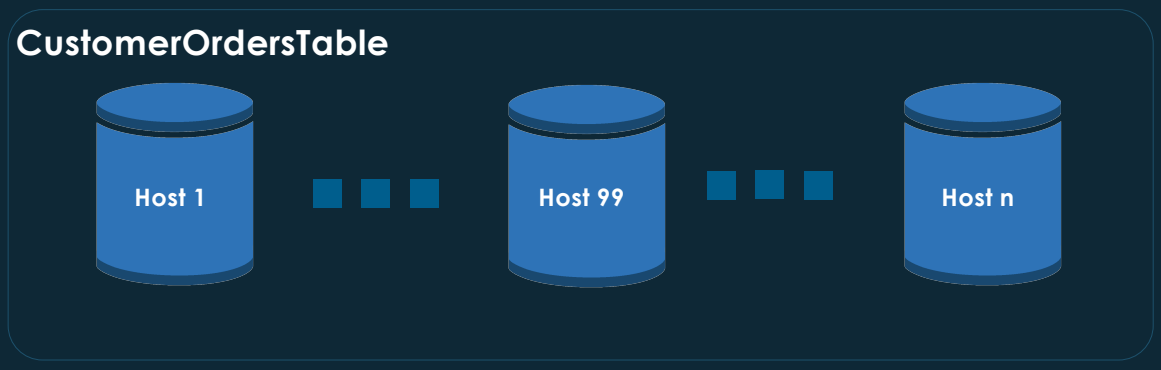


RCUs/WCUs provisioned separately for GSIs

aws

8

Horizontal Sharding

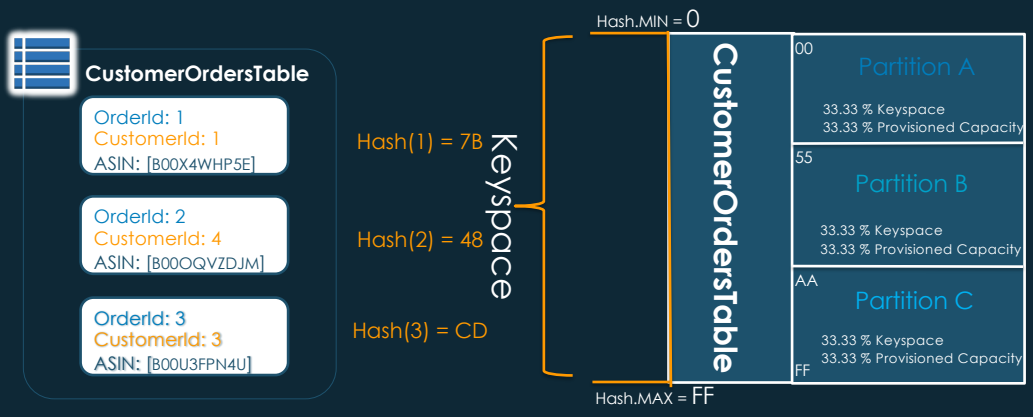


~Each new host brings compute, storage and network bandwidth~

aws

9

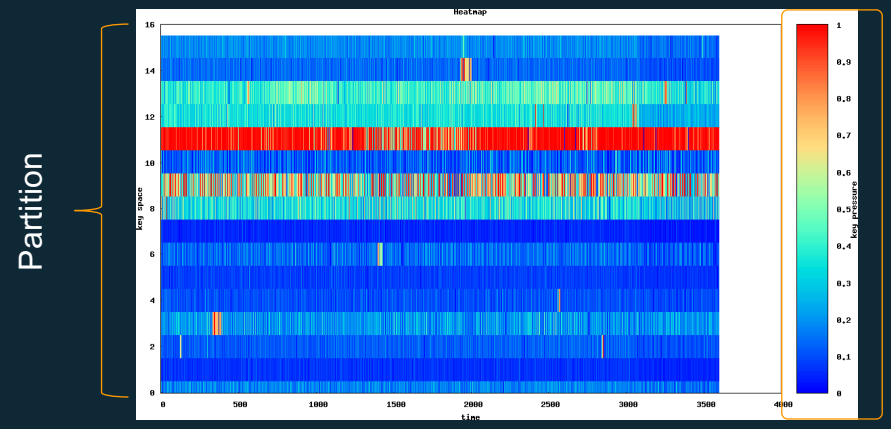
Partitioning



aws

10

What bad NoSQL looks like…



11

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."

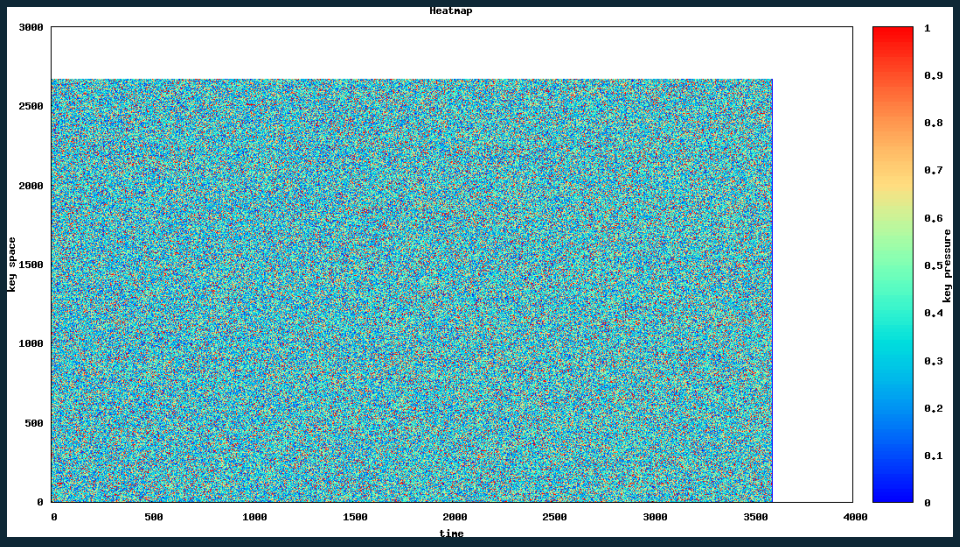
—DynamoDB Developer Guide

Space: access is evenly spread over the keyspace

Time: requests arrive evenly spaced in time

12

Much better picture…



aws

13

Amazon DynamoDB

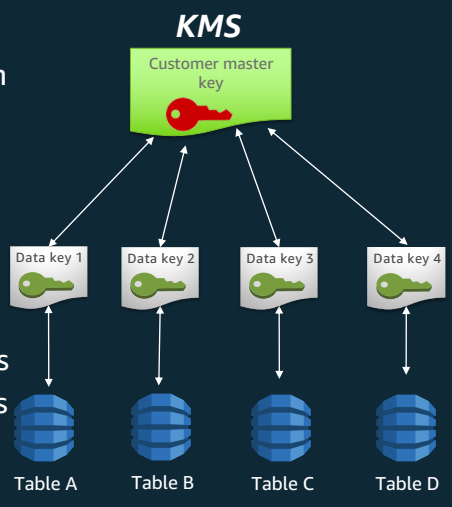
Encryption at rest

Two-tiered key hierarchy using envelope encryption

- Unique data key encrypts customer data
- Customer Master Keys encrypt data keys

Benefits

- Encrypts both base tables and indexes
- Transparent, no application changes
- Better performance for encrypting large tables
- Easier to manage small number of master keys than billions of data keys
- Centralized access and audit of key activity



aws

14

Amazon DynamoDB – Backup and Restore

Only cloud database to provide on demand and continuous backups

On-demand backups for long-term data archival and compliance

Point in time restore for short term retention and data corruption protection (35 days)

Point in time recovery with restore times in a few hours depending on table size

aws

15

Adaptive Capacity

- Dynamically adjusts partition capacity based on real-time traffic
  - To better handle imbalanced workloads
  - Best practices for even load distribution still matter
- Reduces throttling
  - As long as there is enough capacity provisioned for the table
  - Up to the hard limit of partition write capacity
  - Triggered by throttling

aws

16

DynamoDB Global Tables

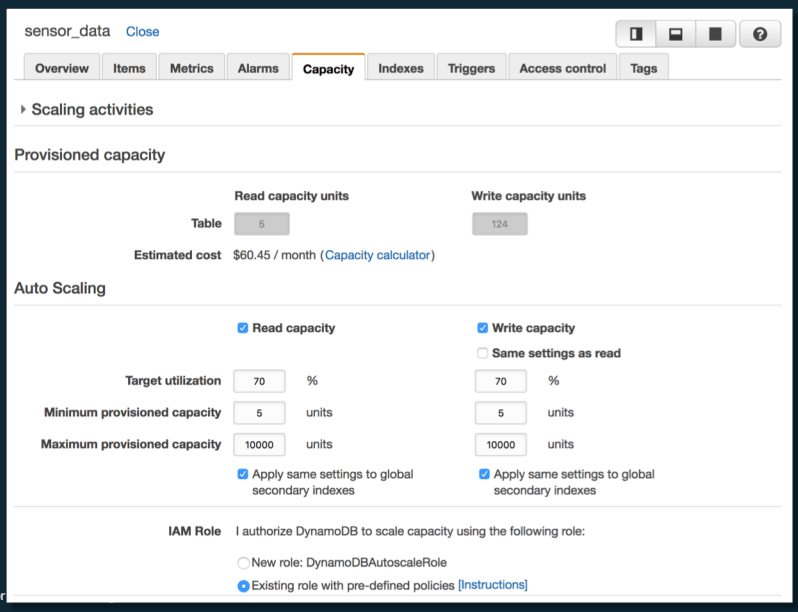
Fully managed, multi-master, multi-region database

- Build high performance, globally distributed applications
- Low latency reads & writes to locally available tables
- Disaster proof with multi-region redundancy
- Easy to setup and no application re-writes required

aws

17

DynamoDB Auto Scaling



aws

18

NoSQL data modeling

aws

19

It’s all about aggregations…

Social network

Document management

Process control

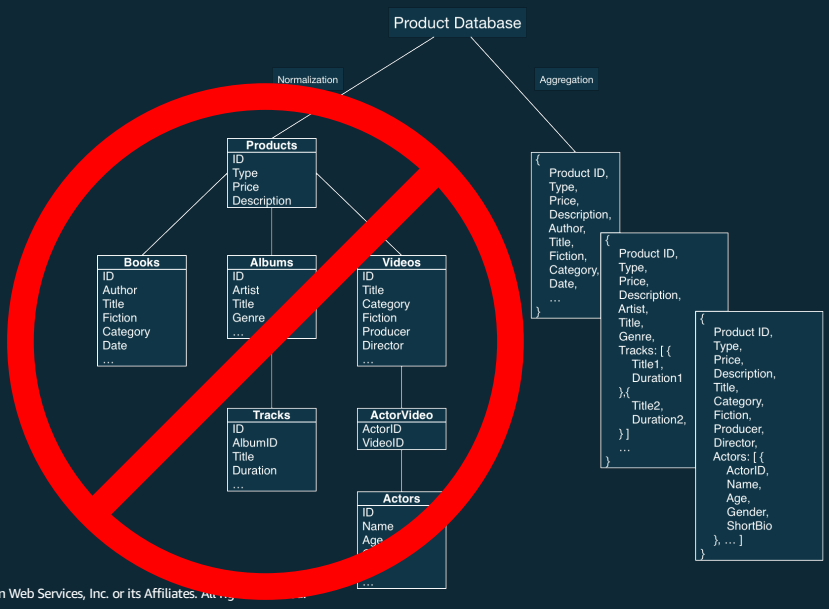
IT monitoring

Data trees

aws

20

SQL vs. NoSQL design pattern



21

Design patterns and best practices

22

TENETS OF NoSQL DATA MODELING

- Understand the use case
- Identify the access patterns
  - Read/Write workloads
  - Query dimensions and aggregations
- Data-modeling
  - Avoid relational design patterns, use one table
- Review -> Repeat -> Review

aws

23

TENETS OF NoSQL DATA MODELING

- Understand the use case
- Identify the access patterns
  - Read/Write workloads
  - Query dimensions and aggregations
- Data-modeling
  - Avoid relational design patterns, use one table
- Review -> Repeat -> Review

- Nature of the application
  - OLTP / OLAP
- Define the Entity-Relationship Model
- Identify Data Life Cycle
  - TTL, Backup/Archival, etc.

aws

24

TENETS OF NoSQL DATA MODELING

- Understand the use case
- Identify the access patterns
  - Read/Write workloads
  - Query dimensions and aggregations
- Data-modeling
  - Avoid relational design patterns, use one table
- Review -> Repeat -> Review

- Identify data sources
- Define queries and write patterns
- Document all workflows

aws

25

TENETS OF NoSQL DATA MODELING

- Understand the use case
- Identify the access patterns
  - Read/Write workloads
  - Query dimensions and aggregations
- Data-modeling
  - Avoid relational design patterns, use one table
- Review -> Repeat -> Review

- **1 application service = 1 table**
  - **Reduce round trips**
  - **Simplify access patterns**
- Identify Primary Keys
  - How will items be inserted and read?
  - Overload items into partitions
- Define indexes for secondary access patterns

aws

26

Building Queries

Query filters, composite keys, and sparse indexes

aws

27

Multi-value sorts and filters

Partition key

Sort key

Base table index

GamerID	Date	GameId	Status	Host
Alice	2018-10-02	d9bl3	DONE	David
Carol	2018-10-08	o2pnb	IN_PROGRESS	Bob
Bob	2018-09-30	72f49	PENDING	Alice
Bob	2018-10-03	b932s	PENDING	Carol
Bob	2018-10-03	ef9ca	IN_PROGRESS	David

aws

28

Approach 1: Query filter

```
SELECT * FROM Game
WHERE Opponent='Bob'
ORDER BY Date DESC
FILTER ON Status='PENDING'
```

Base table index

GamerID	Date	GameId	Status	Host
Alice	2018-10-02	d9bl3	DONE	David
Carol	2018-10-08	o2pnb	IN_PROGRESS	Bob
Bob	2018-09-30	72f49	PENDING	Alice
Bob	2018-10-03	b932s	PENDING	Carol
Bob	2018-10-03	ef9ca	IN_PROGRESS	David

(filtered out)

aws

29

Approach 2: Composite key

Status
DONE
IN_PROGRESS
IN_PROGRESS
PENDING
PENDING

+

Date
2018-10-02
2018-10-08
2018-10-03
2018-10-03
2018-09-30

=

StatusDate
DONE_2018-10-02
IN_PROGRESS_2018-10-08
IN_PROGRESS_2018-10-03
PENDING_2018-09-30
PENDING_2018-10-03

aws

30

Approach 2: Composite key

Partition key

Sort key

Base table index

GamerID	StatusDate	GameId	Host
Alice	DONE_2018-10-02	d9bl3	David
Carol	IN_PROGRESS_2018-10-08	o2pnb	Bob
Bob	IN_PROGRESS_2018-10-03	ef9ca	David
Bob	PENDING_2018-09-30	72f49	Alice
Bob	PENDING_2018-10-03	b932s	Carol

aws

31

Approach 2: Composite key

```
SELECT * FROM Game
WHERE Opponent='Bob'
   AND StatusDate BEGINS_WITH 'PENDING'
```

Bob

Base table index

GamerID	StatusDate	GameId	Host
Alice	DONE_2018-10-02	d9bl3	David
Carol	IN_PROGRESS_2018-10-08	o2pnb	Bob
Bob	IN_PROGRESS_2018-10-03	ef9ca	David
Bob	PENDING_2018-09-30	72f49	Alice
Bob	PENDING_2018-10-03	b932s	Carol

aws

32

Design for the access patterns:

```
SELECT * FROM CartTable
WHERE CartID ='BobCT'
   AND StatusDate BEGINS_WITH 'ACTIVE'
```

Bob's cart

Partition Key *Sort Key*

CartID	Status_Date	ItemSKU	Count	UnitPrice	Gift
AliceCart	SAVED_2018-10-02	d9bl3		34.99	
CarolCart	SHIPPED_2018-10-08	o2pnb		16.99	
BobCart	SAVED_2018-10-03	ef9ca		4.99	
BobCart	ACTIVE_2018-09-30	72f49	2	12.49	
BobCart	ACTIVE_2018-10-03	b932s	1	67.99	Alice

GSI SortKey

GSI Partition Key

aws

33

Sparse indexes

Game-scores-table

Id (Partition)	User	Game	Score	Date	Award
1	Bob	G1	1300	2012-12-23	
2	Bob	G1	1450	2012-12-23	
3	Jay	G1	1600	2012-12-24	
4	Mary	G1	2000	2012-10-24	Champ
5	Ryan	G2	123	2012-03-10	
6	Jones	G2	345	2012-03-20	

Scan sparse GSIs

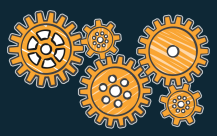
Award-GSI

Award (Partition)	Id	User	Score
Champ	4	Mary	2000

aws

34

Replace filter with indexes



Concatenate attributes to form useful secondary index keys

Take advantage of sparse indexes

Important when: you want to optimize a query as much as possible

aws

35

Complex Queries & Write Sharding

Realtime aggregations
Rebalancing write distribution

aws

36

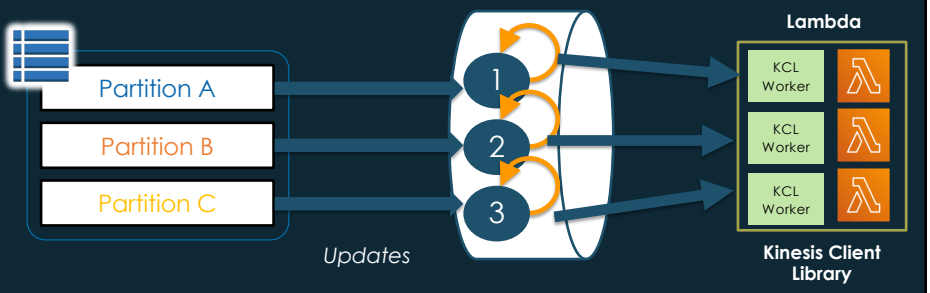
DynamoDB Streams

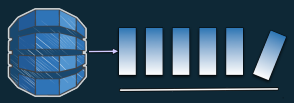
Event-Driven Architecture

Shards have a lineage and automatically close after time or when the associated DynamoDB partition splits

DynamoDB Streams

- ✓Ordered stream of item changes
- ✓Exactly once, strictly ordered by key
- ✓Highly durable, scalable
- ✓24 hour retention
- ✓Sub-second latency
- ✓Compatible with Kinesis Client Library





DynamoDB Table | Shards | Consumers

aws

37

DynamoDB Streams

Anatomy of a Stream Record

```
{
        'eventID': 'ba0aa41d6f9f2d64f',
        'eventName': 'MODIFY',
        'eventVersion': '1.1',
        'eventSource': 'aws:dynamodb',
        'awsRegion': 'us-east-1',
        'dynamodb': {
                'ApproximateCreationDateTime':
                1546020023.0,
                'NewImage': {Map of item}
                'SequenceNumber':
                '113973700000000001108655351',
                'SizeBytes': 116,
                'StreamViewType':
                'NEW_AND_OLD_IMAGES'
        },
        'eventSourceARN':'arn:aws:ddb:reg:acct:tbl/str/'
}
```

Sequence number:
- *Strictly ordered within shard*

View options:
- *Old/New Images or Keys Only*

Type of event

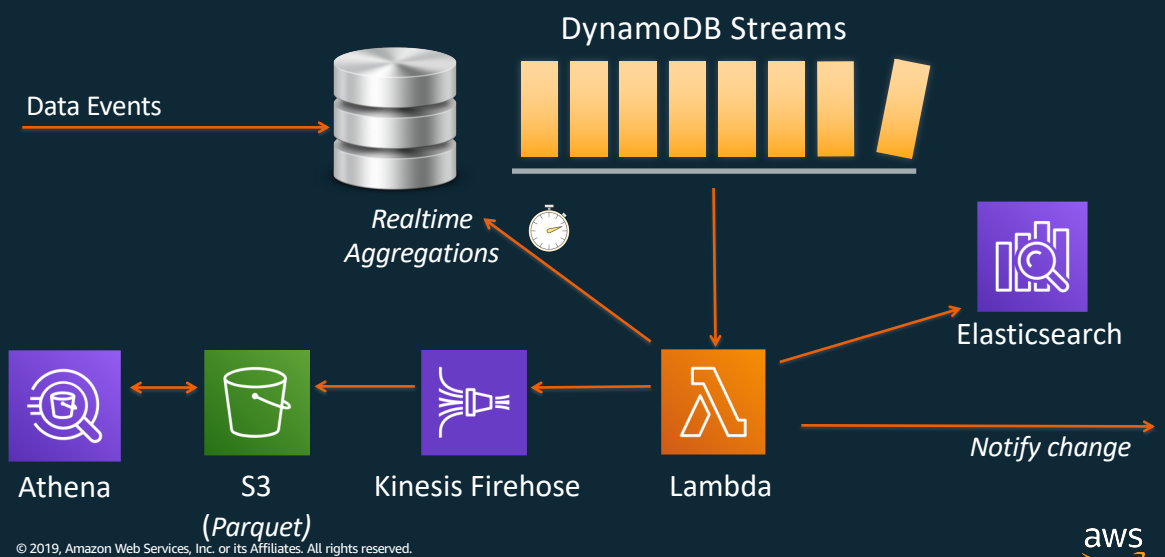
TTL delete: principal = dynamodb

Two consumers per stream

aws

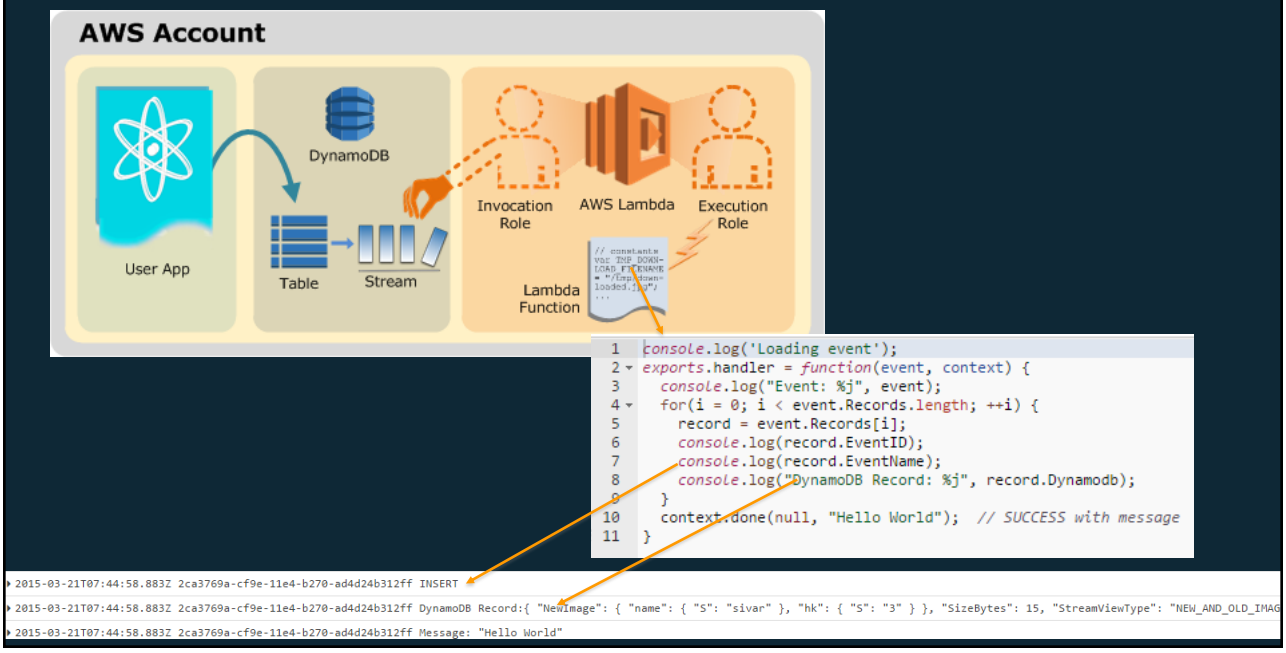
38

Serverless & Event Driven Architecture



39

DynamoDB Streams: Aggregates & Counts



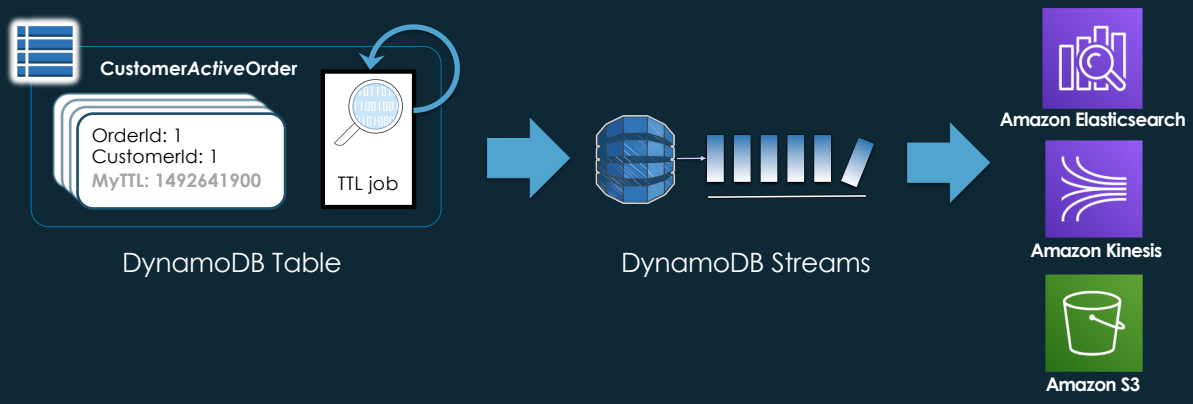
40

Time-To-Live (TTL)

Archive Design Pattern

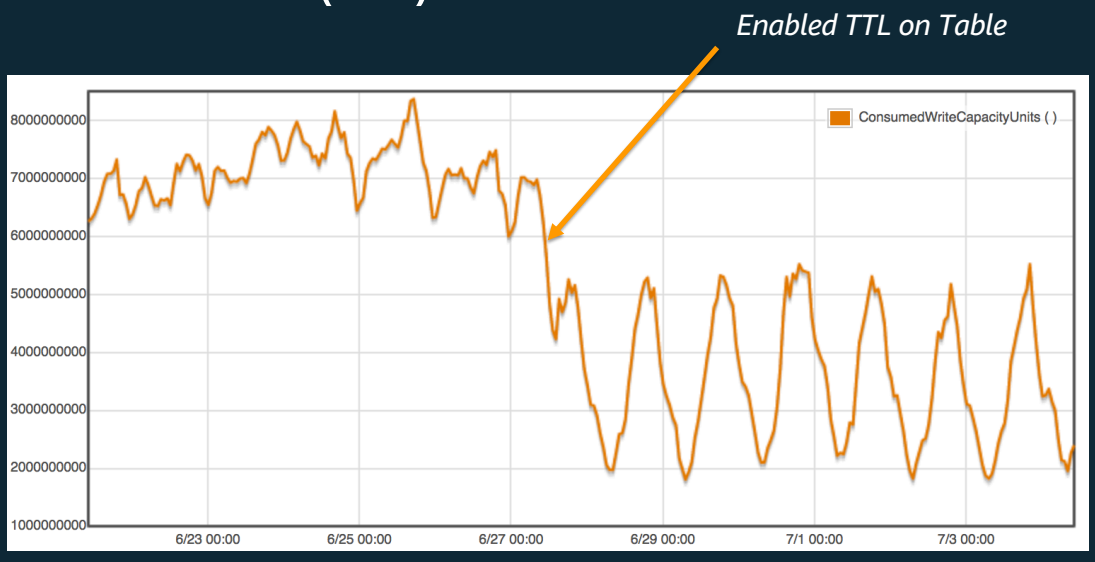
Time-To-Live

An epoch timestamp marking when an item can be deleted by a background process, without consuming any provisioned capacity



41

Time-To-Live (TTL)



42

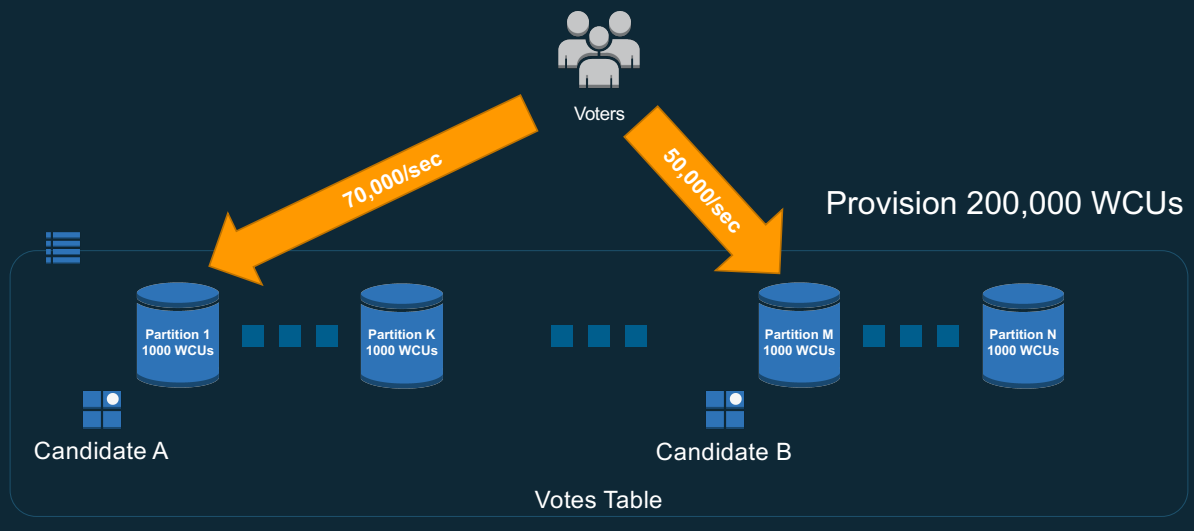
Write sharding

Handling writes with an access distribution skew

aws

43

Scaling bottlenecks

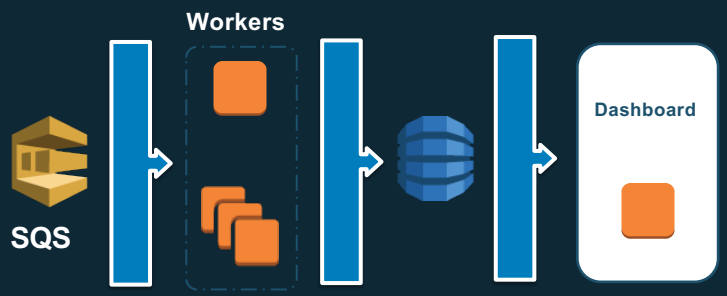


aws

44

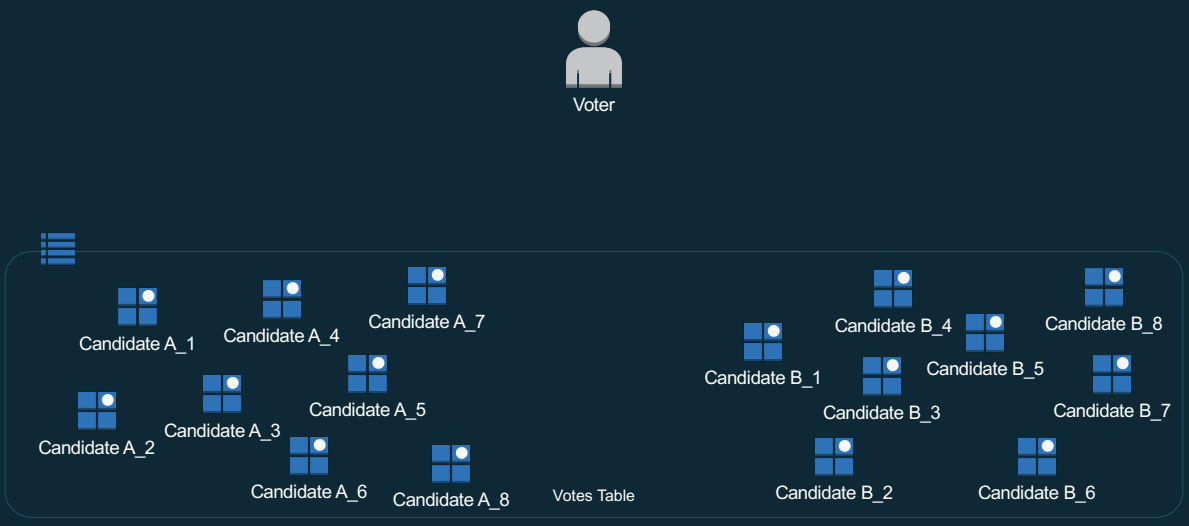
Queue-based load leveling

- Write incoming votes to SQS
- Scale workers as needed to process queue
- Avoid key pressure

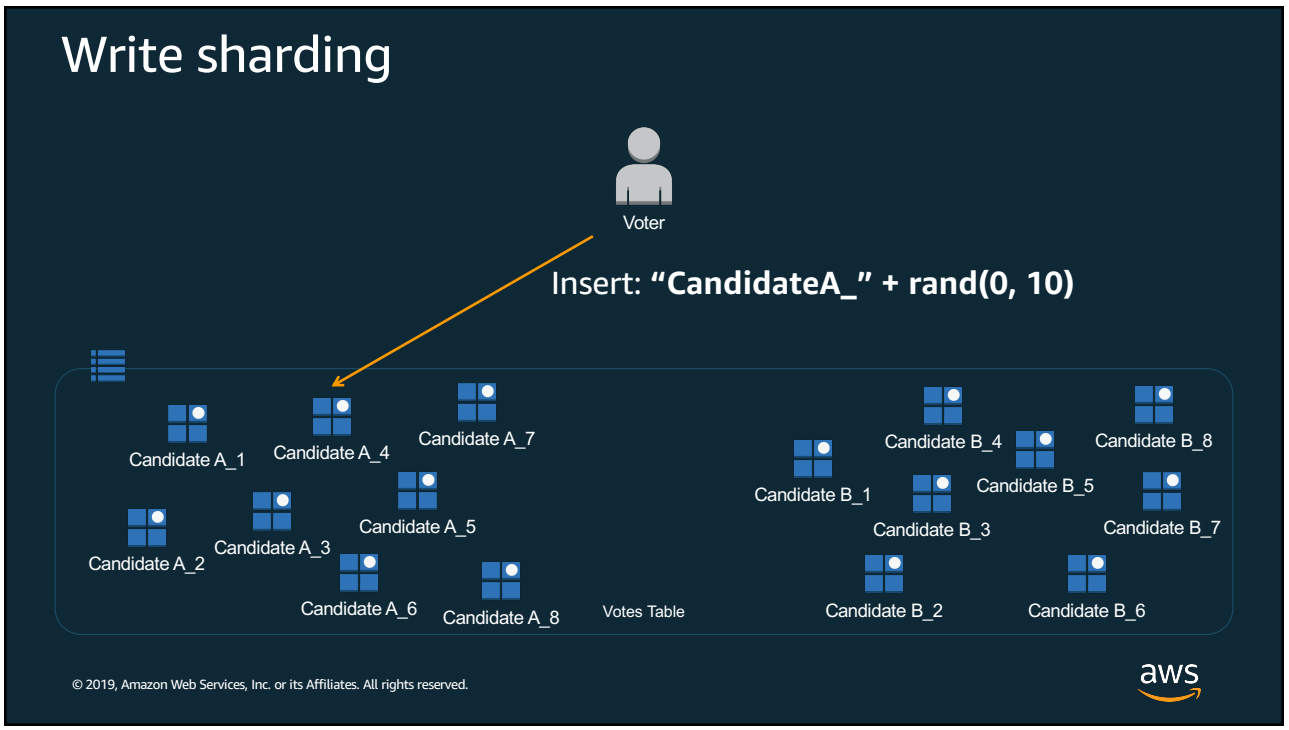


45

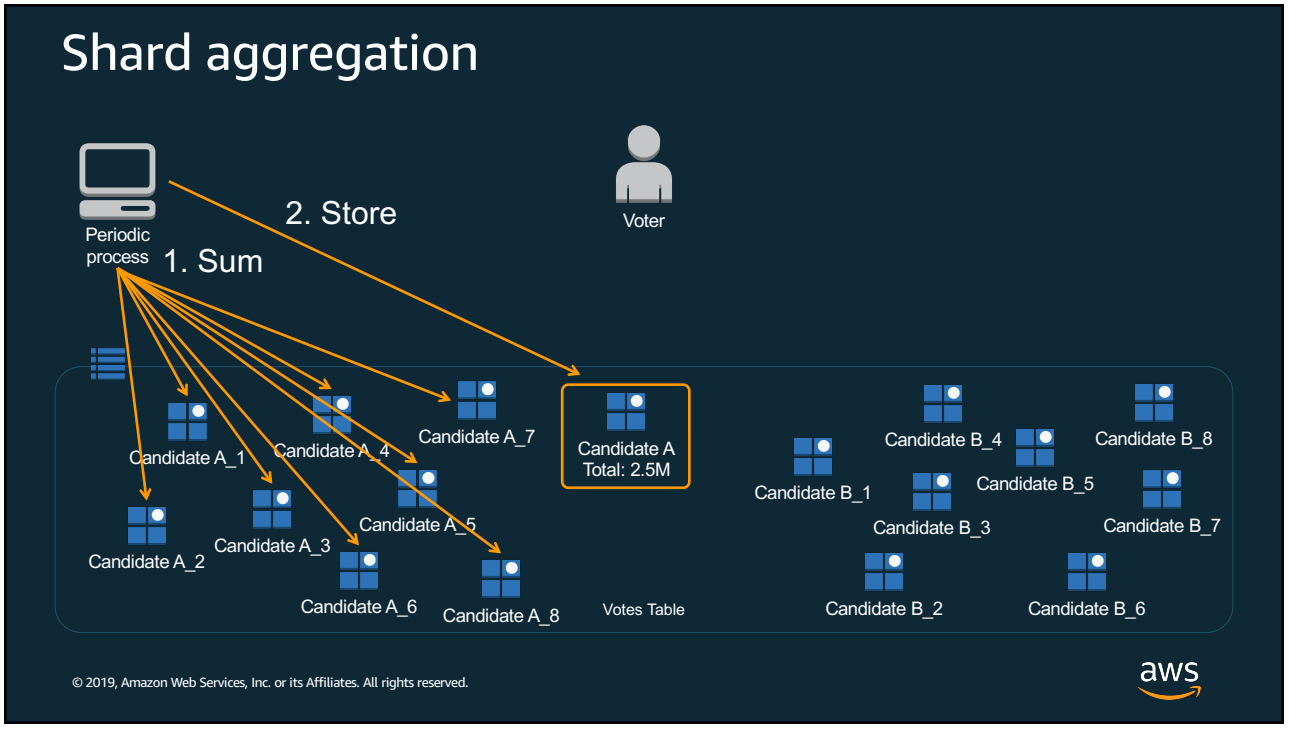
Write sharding



46

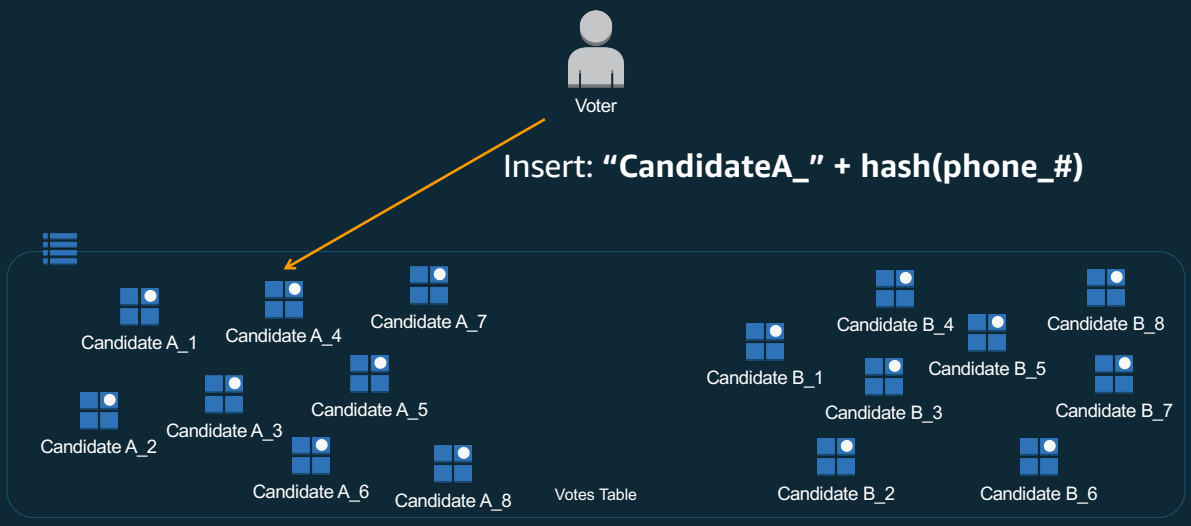

Write sharding
Voter
Insert: “CandidateA_” + rand(0, 10)
Candidate A_1
Candidate A_4
Candidate A_7
Candidate A_2
Candidate A_3
Candidate A_5
Candidate A_6
Candidate A_8
Votes Table
Candidate B_1
Candidate B_4
Candidate B_8
Candidate B_5
Candidate B_3
Candidate B_7
Candidate B_2
Candidate B_6
© 2019, Amazon Web Services, Inc. or its Affiliates. All rights reserved.
aws

47


Shard aggregation
Periodic process
1. Sum
2. Store
Voter
Candidate A_1
Candidate A_4
Candidate A_7
Candidate A_2
Candidate A_3
Candidate A_5
Candidate A_6
Candidate A_8
Votes Table
Candidate A
Total: 2.5M
Candidate B_1
Candidate B_4
Candidate B_8
Candidate B_5
Candidate B_3
Candidate B_7
Candidate B_2
Candidate B_6
© 2019, Amazon Web Services, Inc. or its Affiliates. All rights reserved.
aws

48

Query a single sharded item efficiently



aws

49

Calculating partition counts (reads)

aws

50

Calculating partition counts (writes)

100 K * (itemSize < 1KB ? 1KB : itemSize) / 1000 = 100

Items per second

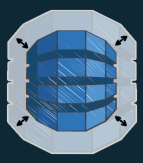
Average item size

Partition max WCU

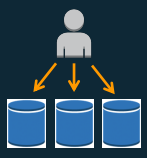
aws

51

Shard write-heavy partition keys



- Increase throughput with concurrency.
- Evaluate RCU/WCU per key, item size and request rate.



Important when: A write workload is not horizontally scalable.

aws

52

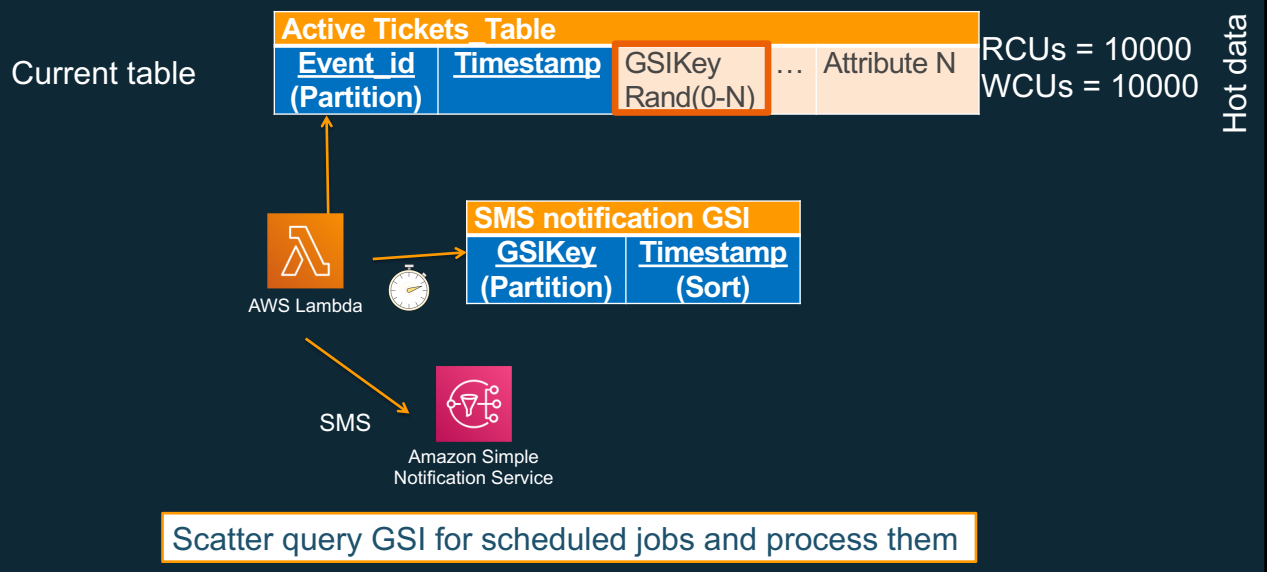
Time-based workflows

Processing the entire table efficiently

aws

53

Job scheduling



aws

54

Find items efficiently with AWS Lambda

- Write-shard a GSI to selectively query the entire table.
- Use Lambda "stored procedure" to process items.
- Migrate data between tables with AWS Lambda.

Important when: You need a massive horizontally scaled index

aws

55

Product catalog

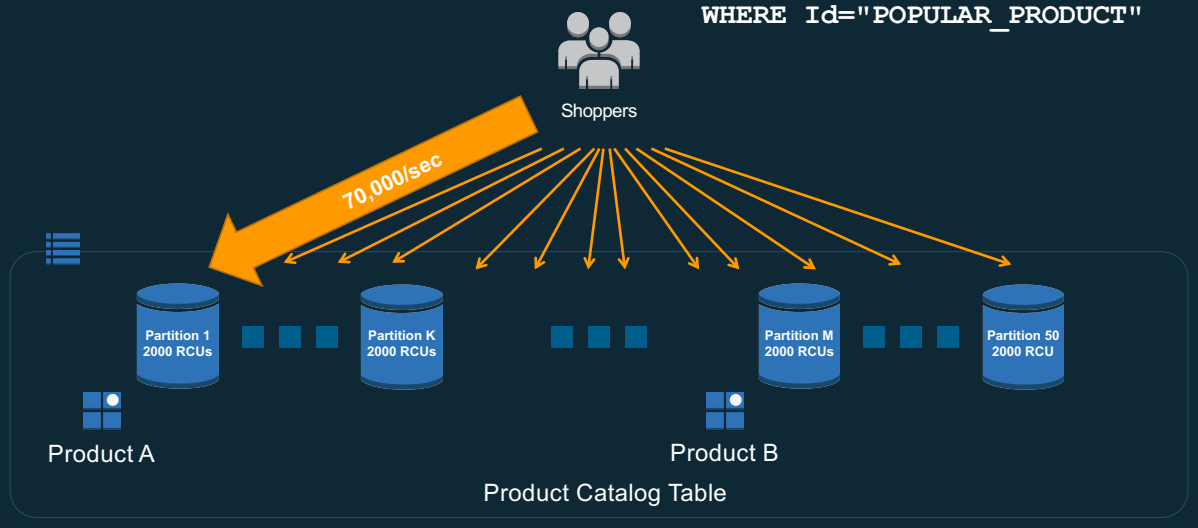
Popular items (***read heavy***)

aws

56

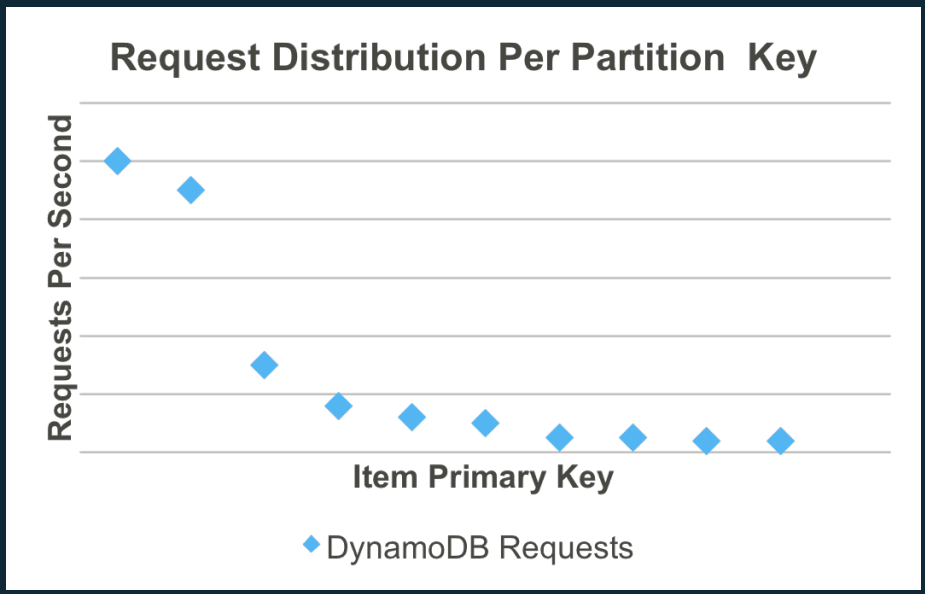
Scaling bottlenecks

```
SELECT Id, Description, ...
FROM ProductCatalog
WHERE Id="POPULAR_PRODUCT"
```



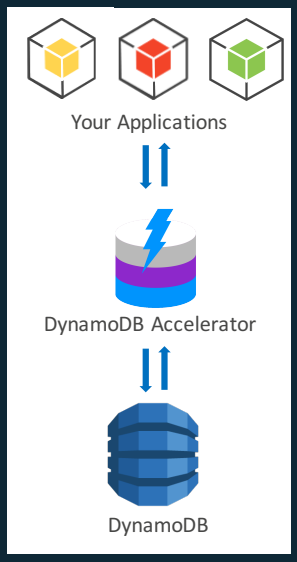
aws

57



aws

58

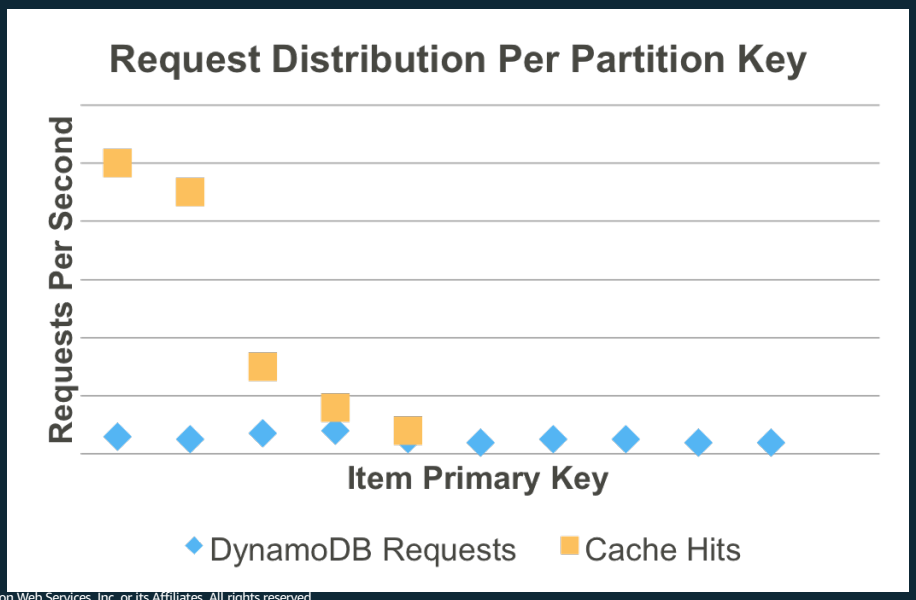


Features

- **Fully managed, highly available:** handles all software management, fault tolerant, replication across multi-AZs within a region
- **DynamoDB API compatible:** seamlessly caches DynamoDB API calls, no application re-writes required
- **Write-through:** DAX handles caching for writes
- **Flexible**: configure DAX for one table or many
- **Scalable**: scales-out to any workload with up to 10 read replicas
- **Manageability:** fully integrated AWS service: Amazon CloudWatch, tagging for DynamoDB, AWS Console
- **Security:** Amazon VPC, AWS IAM, AWS CloudTrail, AWS Organizations

aws

59



aws

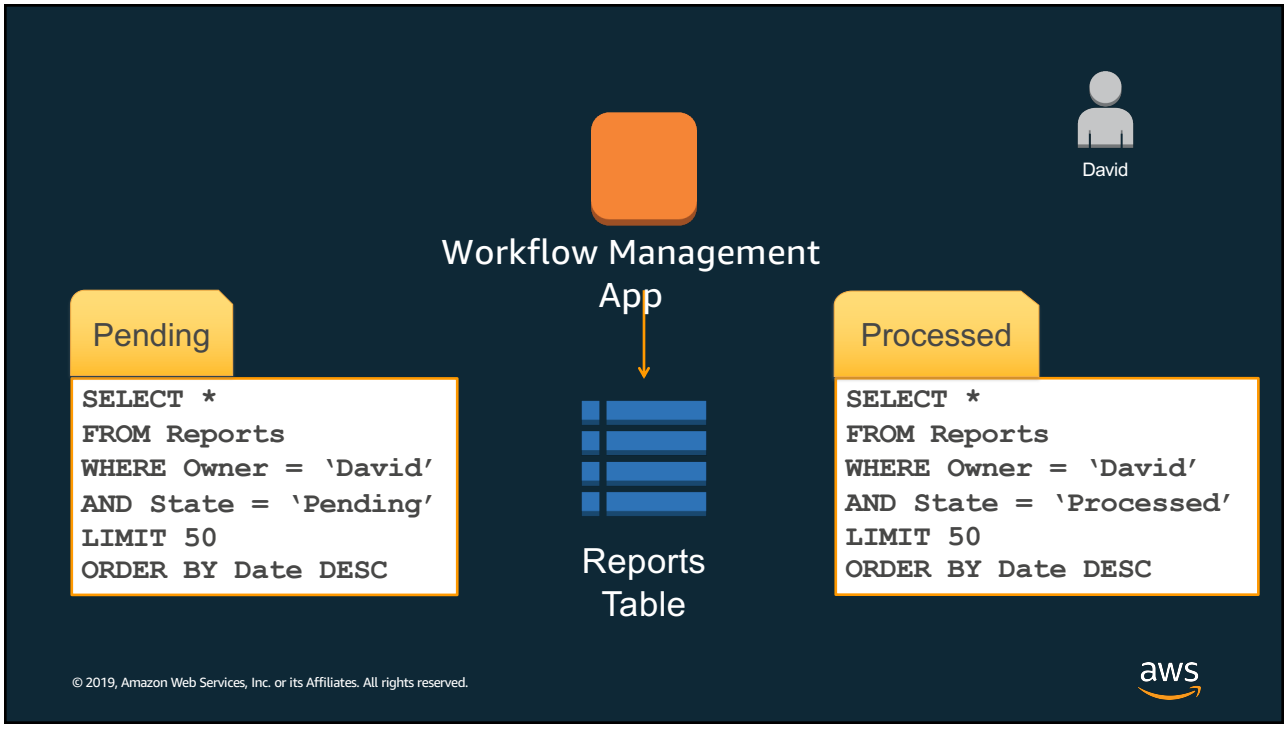
60

Vertical Partitioning

Large items
Filters vs. indexes
M:N modeling

aws

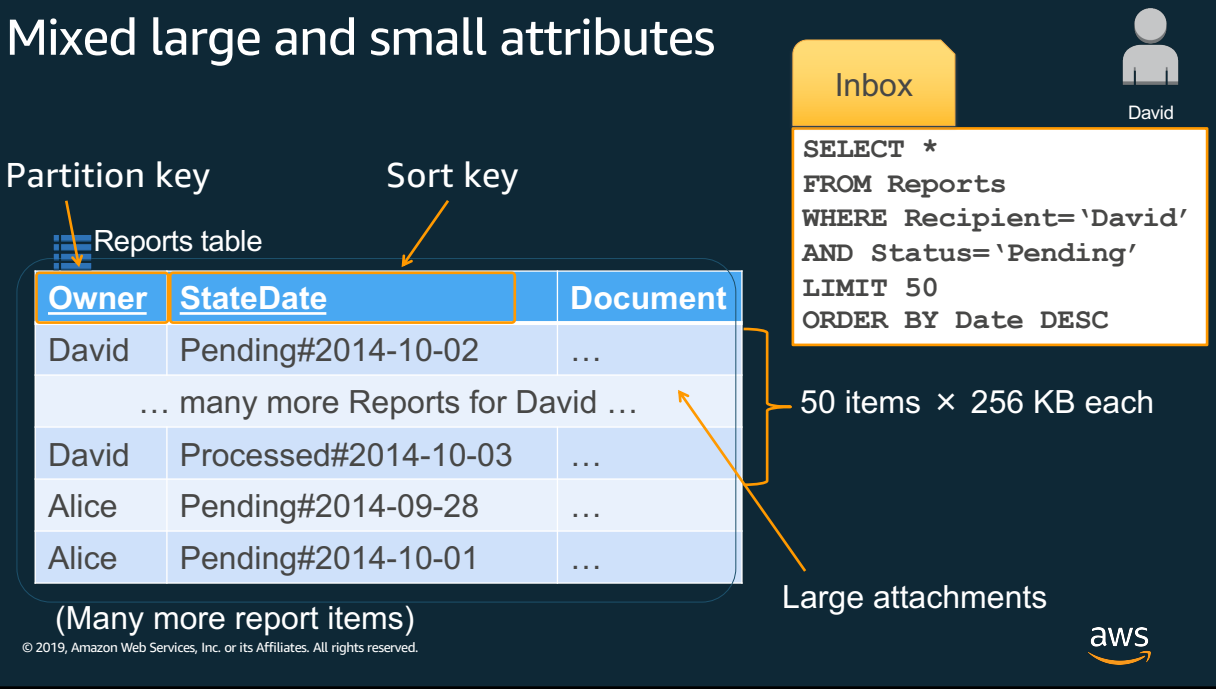
61



aws

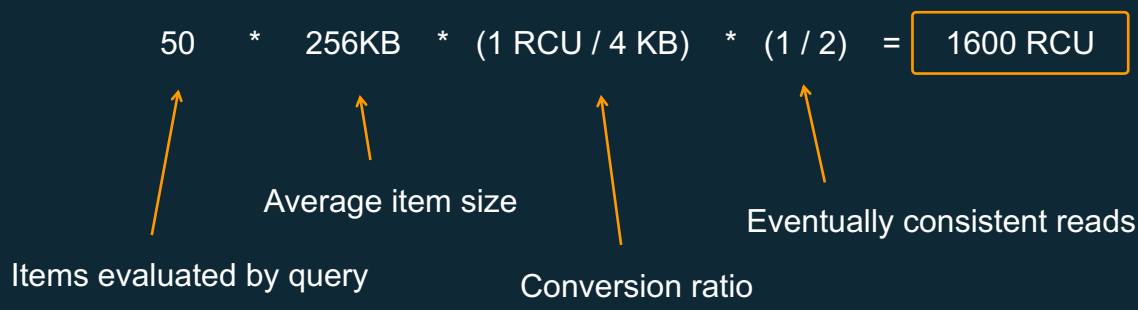
62

Mixed large and small attributes

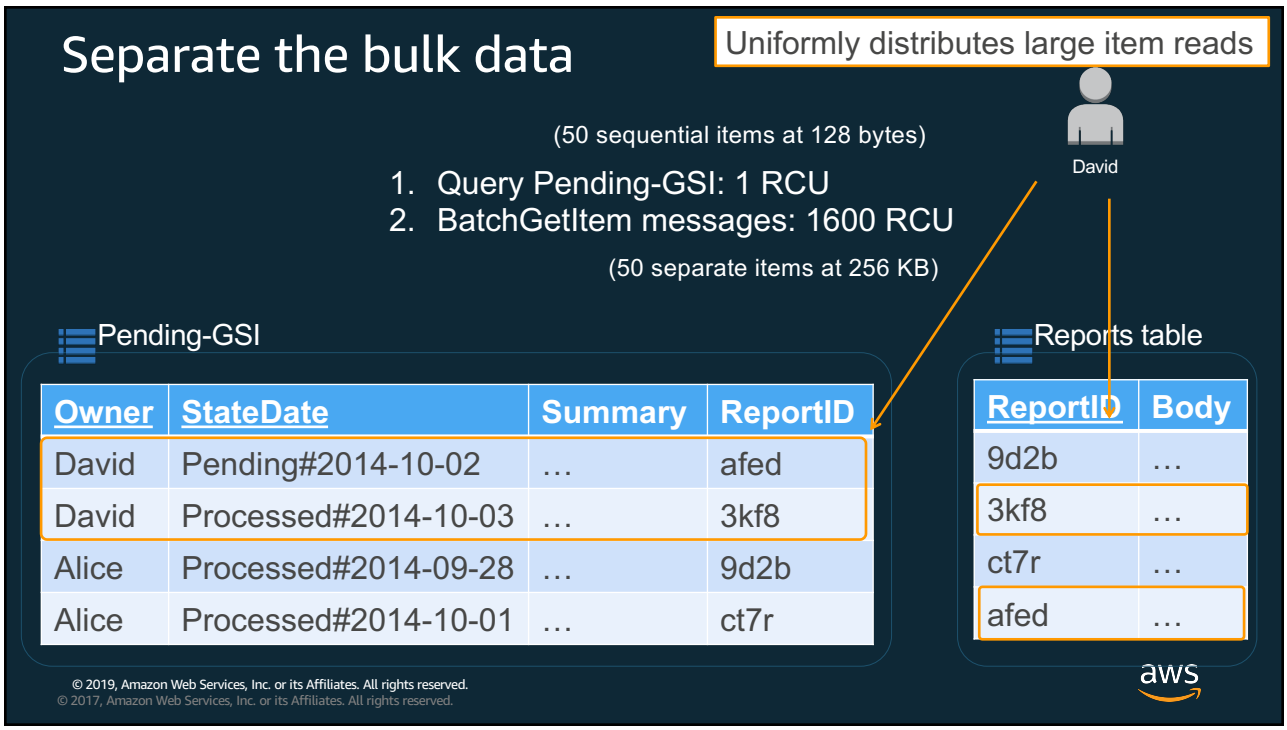


63

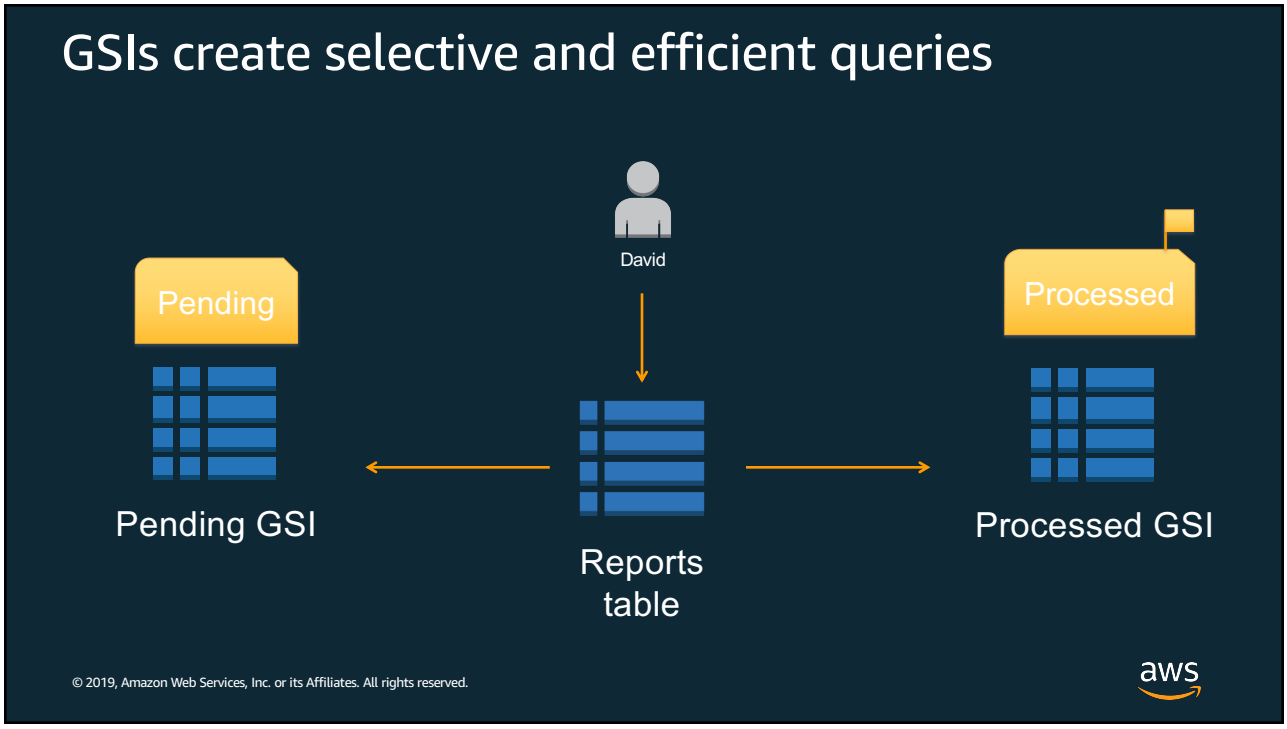
Computing read query cost



64

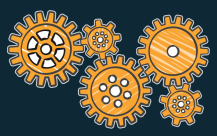

Separate the bulk data
Uniformly distributes large item reads
David
(50 sequential items at 128 bytes)
1. Query Pending-GSI: 1 RCU
2. BatchGetItem messages: 1600 RCU
(50 separate items at 256 KB)
Pending-GSI
Owner | StateDate | Summary | ReportID
David | Pending#2014-10-02 | … | afed
David | Processed#2014-10-03 | … | 3kf8
Alice | Processed#2014-09-28 | … | 9d2b
Alice | Processed#2014-10-01 | … | ct7r
Reports table
ReportID | Body
9d2b | …
3kf8 | …
ct7r | …
afed | …
© 2019, Amazon Web Services, Inc. or its Affiliates. All rights reserved.
© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.
aws

65

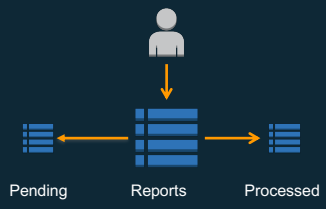

GSIs create selective and efficient queries
David
Pending
Processed
Pending GSI
Reports table
Processed GSI
© 2019, Amazon Web Services, Inc. or its Affiliates. All rights reserved.
aws

66

Distribute large items



- Reduce one-to-many item sizes.
- Configure secondary index projections.
- Use GSIs to model M:N relationship.



Important when: Querying many large items at once.

aws

67

Relational Data Modeling

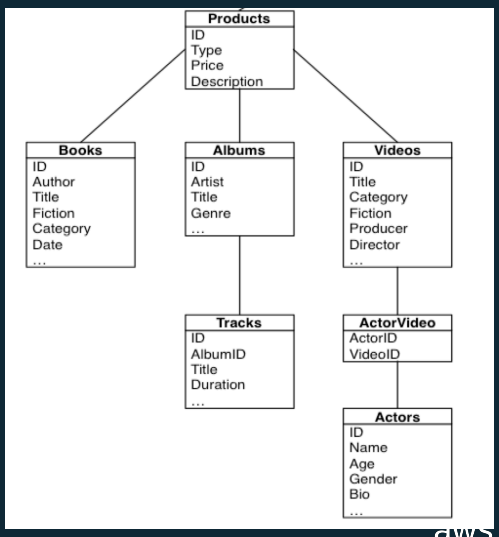
Hierarchies & Compound Keys
Transactions
Bringing it all together

aws

68

How OLTP apps use data

- Mostly hierarchical structures
- Entity driven workflows
- Data spread across tables
- Requires complex queries
- Primary driver for ACID



aws

69

Multi-version concurrency

Distributed Transactions in NoSQL

aws

70

Distributed Transactions & Version History

ItemID (PK)	Version (SK)	CurVer	Attrs
1	v0	2	…
	v1	…	
	v2	…	
	v3	…	

(Many more item partitions)

Transaction

```
COPY Item.v0 -> Item1.v3 IF Item.v3 == NULL
UPDATE Item1.v3 SET Attr1 += 1
UPDATE Item1.v3 SET Attr2 = …
UPDATE Item1.v3 SET Attr3 = …
COPY Item1.v3 -> Item1.v0 SET CurVer = 3
```

Overwrite v0 Item to Commit changes

Item versions

aws

71

DynamoDB Transactions API

- TransactWriteItems
  - Synchronous update, put, delete, and check
    - Atomic
  - Up to 10 items within a transaction
  - Supports multiple tables
  - Complex conditional checks
- Good Use Cases
  - Commit changes across items
  - Conditional batch inserts/updates
- Bad Use Case
  - Maintaining normalized data

aws

72

Game state - Transactions API

Atomic update of gamer "Hammer57's" Health & Coins

```
{ "TransactItems" :  [ {
   "Update ": {
    "TableName": "Gamers",
    "Key" :{"GamerTag" : {"S": "Hammer57"},
    "Type" : {"S" : "Status"}},
    "UpdateExpression" : "Set health = :nhealth",
    "ExpressionAttributeValues":{":nhealth":{"N":"100"}}
   },
  {
   "Update ": {
    "TableName": "Gamers",
    "Key" :{"GamerTag" : {"S": "Hammer57"},
    "Type" : {"S" : "Assets"} }
     "ConditionExpression" : "coins > :cost",
     "UpdateExpression" : "Set coins = coins - :cost",
      "ExpressionAttributeValues" :{":cost":{"N":"400"}}
   }
 }]
}
```

Gamers				
Primary Key		Attributes		
Gamer Tag	Type			
Hammer57	Rank	Level	Points	Tier
		87	4050	Elite
	Status	Health	Progress	
		100%	30	
	Weapon	Class	Damage	Range
		Taser	87%	50
	Assets	Coins		
		1000		
FluffyDuffy	Rank	Level	Points	Tier
		5	1072	Trainee
	Status	Health	Progress	
		37	8	
	Assets	Coins		
		0		

aws

73

Manage transactional workflows

Distributed Transactions:

- Manage versioning across items with metadata
- Code your app to recognize when updates are in progress
- Implement app layer error handling and recovery logic

DynamoDB Native Transactions:

- ACID mutations across 10 items

Important when: transactional writes are required

aws

74

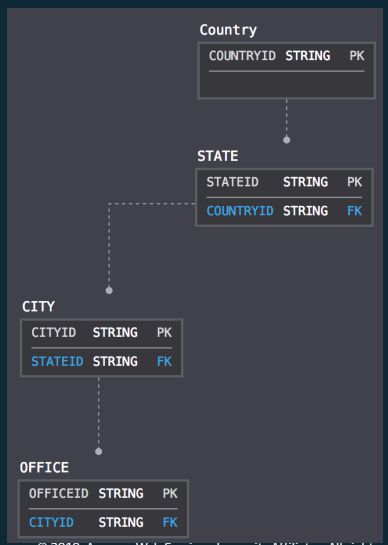
Hierarchical data

Composite key modeling

aws

75

Hierarchical data structures as items

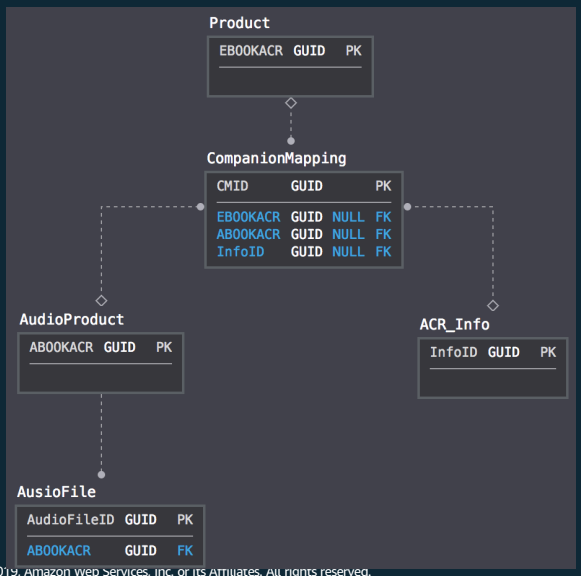


- Use composite sort key to define a hierarchy
- Highly selective queries with sort conditions
- Reduce query complexity

Primary Key		Attributes		
COUNTRY	LOCATION			
USA	NY#NYC#JFK11	Address	EmployeeCount	BuildingManager
	NY#NYC#JFK14	Address	EmployeeCount	BuildingManager
	WA#SEA#BLACKFOOT	Address	EmployeeCount	BuildingManager
	WA#SEA#KUMO	Address	EmployeeCount	BuildingManager
	WA#SEA#MAYDAY	Address	EmployeeCount	BuildingManager

76

Audible eBook sync service



- Allows users to save session state for Audible eBooks
- Maintains mappings per user for eBooks and audio products
- Spikey load patterns require significant overprovisioning
- Large number of access patterns

78

Access patterns

#	USE CASE	
1	CompanionMapping	getCompanionMappingsByAsin
2	CompanionMapping	getCompanionMappingsByEbookAndAudiobookContentId
3	CompanionMapping	getCompanionMappingsFromCache
4	CompanionMapping	getCompanionMappings
5	CompanionMapping	getCompanionMappingsAvailable
6	AcrInfo	getACRInfo
7	AcrInfo	getACRs
8	AcrInfo	getACRInfos
9	AcrInfo	getACRInfosbySKU
10	AudioProduct	getAudioProductsForACRs
11	AudioProduct	getAudioProducts
12	AudioProduct	deleteAudioProductsMatchingSkuVersions
13	AudioProduct	getChildAudioProductsForSKU
14	Product	getProductInfoByAsins
15	Product	getParentChildDataByParentAsins
16	AudioFile	getAudioFilesForACR
17	AudioFile	getAudioFilesForChildACR
18	AudioFile	getAudioFilesByParentAsinVersionFormat
19	AudioFile	getAudioFiles
20	AudioFile	getAudioFilesForChildAsin

79

Primary table

TABLE	Primary Key		Attributes	
	PK	**SK (GSI 3)**		
	ABOOKACR1	v0#ABOOKACR1	**GSI-1**	**GSI-2**
			ABOOK-ASIN1	ABOOK-SKU1
		EBOOKACR1	**GSI-1**	**GSI-2**
			SyncFileAcr	ABOOK-ASIN1
		ABOOKACR1#TRACK#1	**GSI-1**	**GSI-2**
			ABOOK-ASIN1	ABOOK-SKU1
		ABOOKACR1#TRACK#2	**GSI-1**	**GSI-2**
			ABOOK-ASIN1	ABOOK-SKU1
	EBOOKACR1	EBOOKACR1	**GSI-1**	**EBookAsin**
			EBOOK-SKU1	ASIN

aws

80

Indexes

GSI 1	Partition Key	Projected Attributes	
	ABOOK-ASIN1	ABOOKACR1	ABOOKACR1-v1
			ABOOKACR1#TRACK#1
			ABOOKACR1#TRACK#2
	SyncFileAcr	ABOOKACR1	MAP-EBOOKACR1
	EBOOK-SKU1	ABOOKACR1	EBOOKACR1

GSI 2	Partition Key	Projected Attributes	
	ABOOK-ASIN1	ABOOKACR1	MAP-EBOOKACR1
	ABOOK-SKU1	ABOOKACR1	ABOOKACR1-v1
			ABOOKACR1#TRACK#1
			ABOOKACR1#TRACK#2

GSI 3	GSI Partition Key	Projected Attributes	
	V0#ABOOKACR1	ABOOKACR1	ABOOKACR1-v1
	EBOOKACR1	ABOOKACR1	MAP-EBOOKACR1
	ABOOKACR1#TRACK#1	ABOOKACR1	ABOOKACR1#TRACK#1
	ABOOKACR1#TRACK#2	ABOOKACR1	ABOOKACR1#TRACK#2
	EBOOKACR1	ABOOKACR1	EBOOKACR1

aws

81

Query conditions

#	USE CASE		Lookup parameters	INDEX	Key Conditions	Filter Conditions
1	CompanionMapping	getCompanionMappingsByAsin	audiobookAsin/ebookSku	GSI2	GSI-2=ABOOK-ASIN1	None
2	CompanionMapping	getCompanionMappingsByEbookAndAudiobookContentId	ebookAcr/sku,version,format or audiobookAcr/asin,version,format	GSI-3 on TargetACR attribute OR PrimaryKey on Table	GSI-3=MAP-EBOOKACR1	version=v and format=f
3	CompanionMapping	getCompanionMappingsFromCache	ebookAcr/sku,version,format or audiobookAcr/asin,version,format	GSI-3 on TargetACR attribute OR PrimaryKey on Table	GSI-3=MAP-EBOOKACR1	version=v and format=f
4	CompanionMapping	getCompanionMappings	syncfileAcr, ebookAcr?, audiobookAcr?	GSI1	GSI-1=SyncFileAcr	None
5	CompanionMapping	getCompanionMappingsAvailable	ebookAcr, audiobookAcr	Primary Key on Table	Acr=ABOOKACR1 and TargetACR beginswith "MAP-"	
6	AcrInfo	getACRInfo	acr	Primary Key on Table	Acr=ABOOKACR1 and TargetACR beginswith "ABOOKACR1-v"	
7	AcrInfo	getACRs	acr / asin,version,format	Primary Key on Table	Acr=ABOOKACR1	version=v and format=f
8	AcrInfo	getACRInfos	acr	Primary Key on table	Acr=ABOOKACR1 and TargetACR beginswith "ABOOKACR1"	
9	AcrInfo	getACRInfosbySKU	sku	GSI2	GSI-2=ABOOK-SKU1	
10	AudioProduct	getAudioProductsForACRs	acr	Primary Key on table	Acr=ABOOKACR1 and TargetACR beginswith "ABOOKACR1"	
11	AudioProduct	getAudioProducts	sku, version, format	GSI2	GSI-2=ABOOK-SKU1	version=v and format=f
12	AudioProduct	deleteAudioProductsMatchingSkuVersions	sku, version	GSI2	GSI-2=ABOOK-SKU1	version=v
13	AudioProduct	getChildAudioProductsForSKU	sku	GSI2	GSI-2=ABOOK-SKU1	
14	Product	getProductInfoByAsins	asin	GSI1	GSI-1=ABOOK-ASIN1	
15	Product	getParentChildDataByParentAsins	asin	GSI1	GSI-1=ABOOK-ASIN1	
16	AudioFile	getAudioFilesForACR	acr	Primary Key on table	Acr=ABOOKACR1 and TargetACR beginswith "ABOOKACR1#"	
17	AudioFile	getAudioFilesForChildACR	acr, parent_asin	Primary Key on table	Acr=ABOOKACR1	version=v and format=f
18	AudioFile	getAudioFilesByParentAsinVersionFormat	parent_asin, version, format	GSI1	GSI-1=ABOOK-ASIN1	version=v and format=f
19	AudioFile	getAudioFiles	sku, version, format	GSI2	GSI-2=ABOOK-SKU1	version=v and format=f
20	AudioFile	getAudioFilesForChildAsin	asin, parent_asin, version, format	GSI1	GSI-1=ABOOK-ASIN1	version=v and format=f

82

Recap – what did we cover?

- DynamoDB 101
- Tenants of NoSQL Data Modeling
  - Work backwards from access pattern.
- Common Design Patterns
  - How to build keys to support queries.
  - Indexes, Filters, and Hierarchies:
    - *large items & M:N relationships,*
  - Write Sharding, Distributed Transactions, & Complex Queries
- DynamoDB in the AWS Ecosystem

aws

83

Thank you!

aws

84