AWS re:Invent

DAT305

Real-world use cases for Amazon DynamoDB

Sean Shriver
Sr. DynamoDB SA
Amazon Web Services

Ankur Kasliwal
Technical Program Manager
Amazon Web Services

Three things to consider

Hi!

URL

Office hours

Agenda

1. DynamoDB fundamentals
2. Three distilled steps for a successful design
3. Three real-world use cases for DynamoDB

Amazon DynamoDB

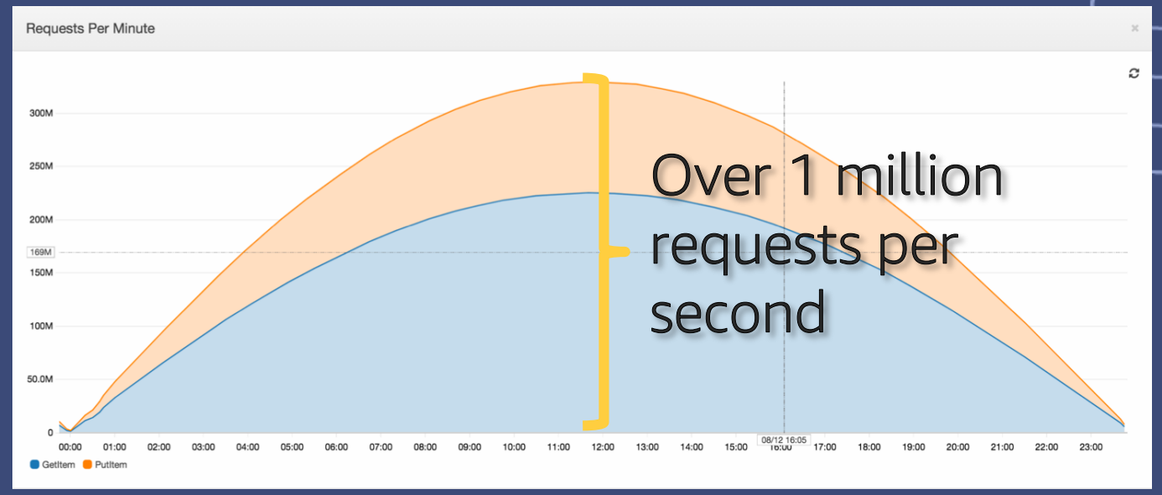
Key value

Durable

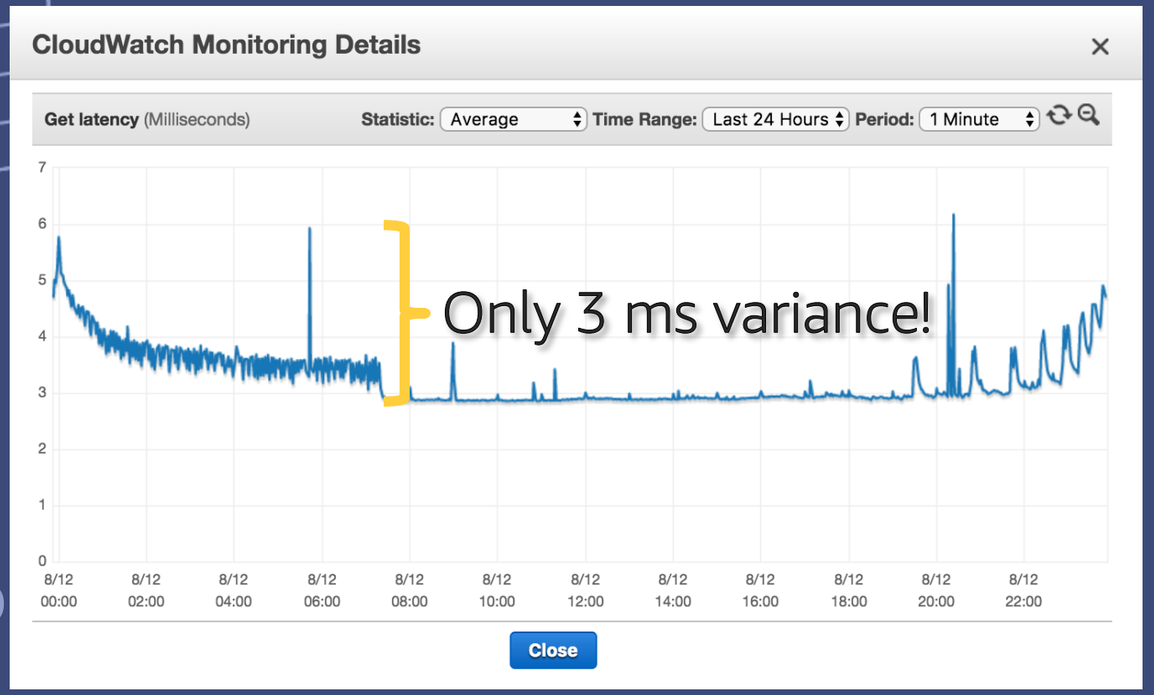
Low latency

Amazon DynamoDB

Request Volume



Latency

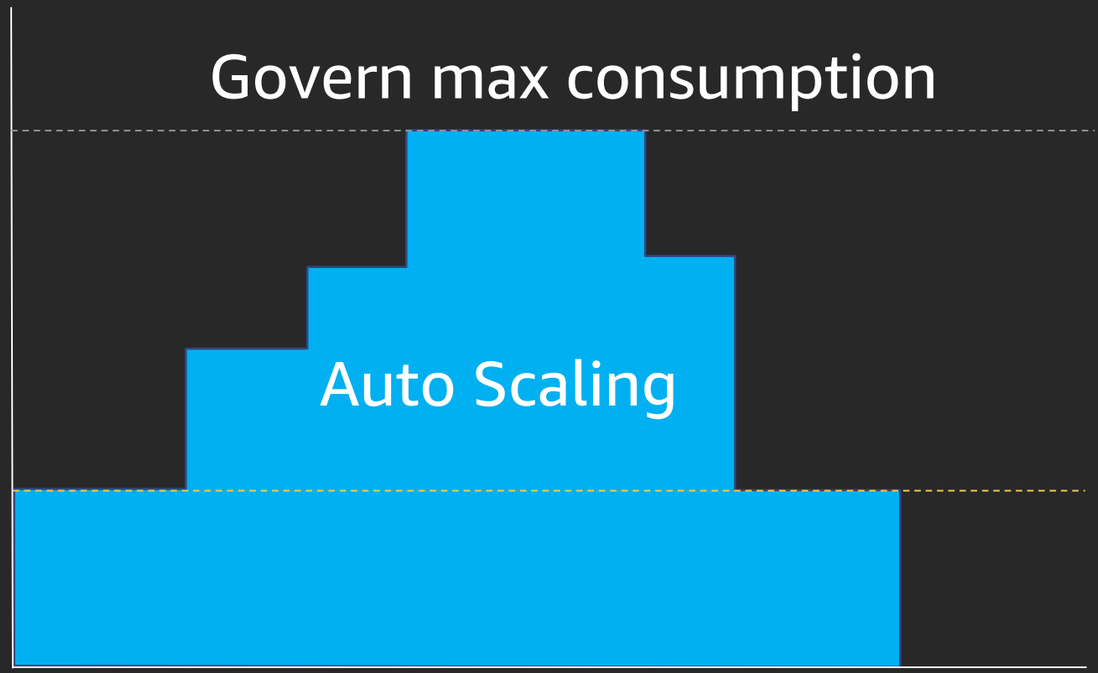


Key Value

Durab

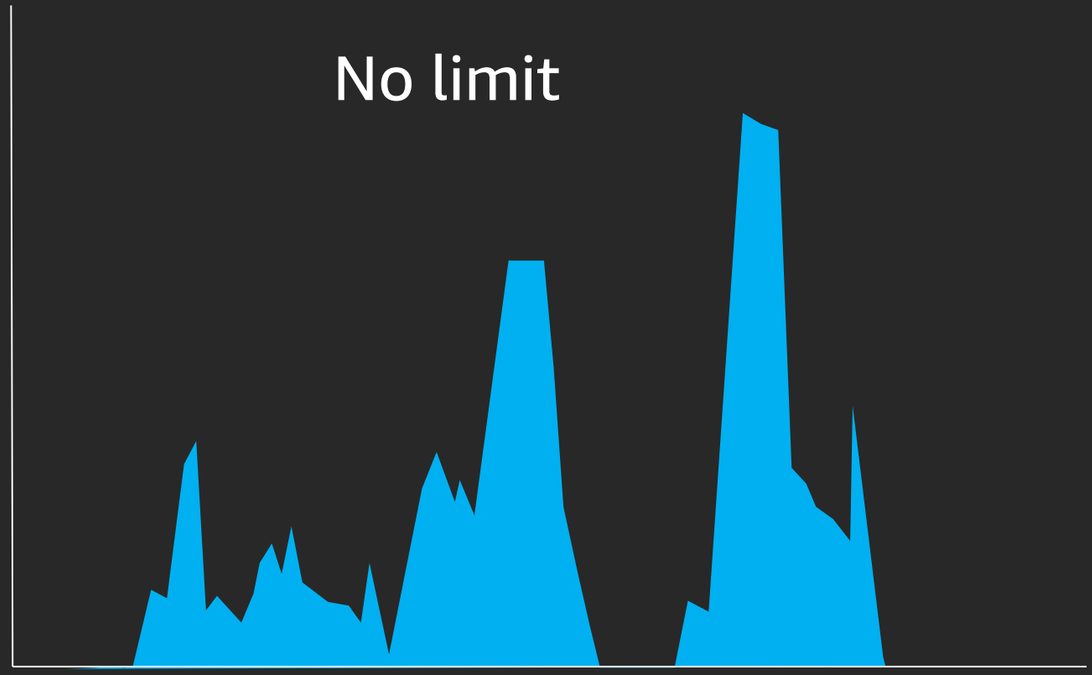
DynamoDB: Capacity managed for you

Provisioned



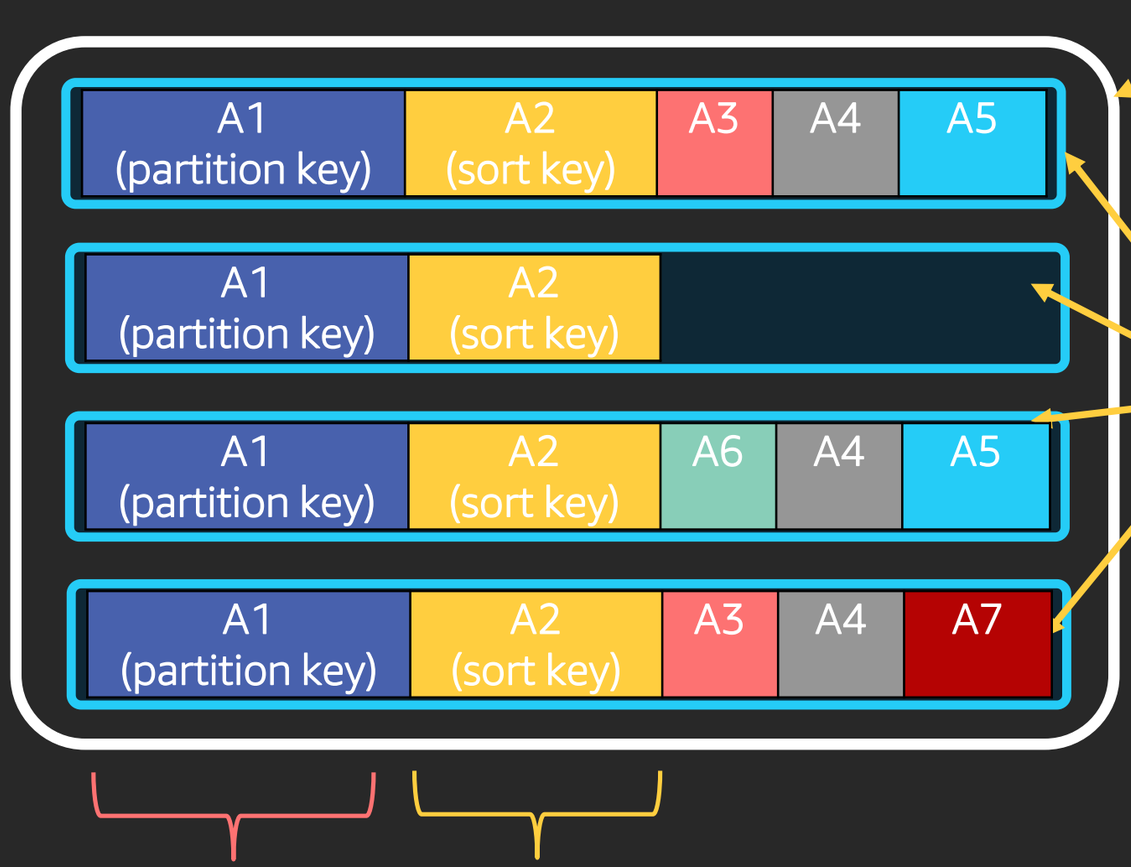
Set a minimum

On-Demand



Start at zero

DynamoDB table



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

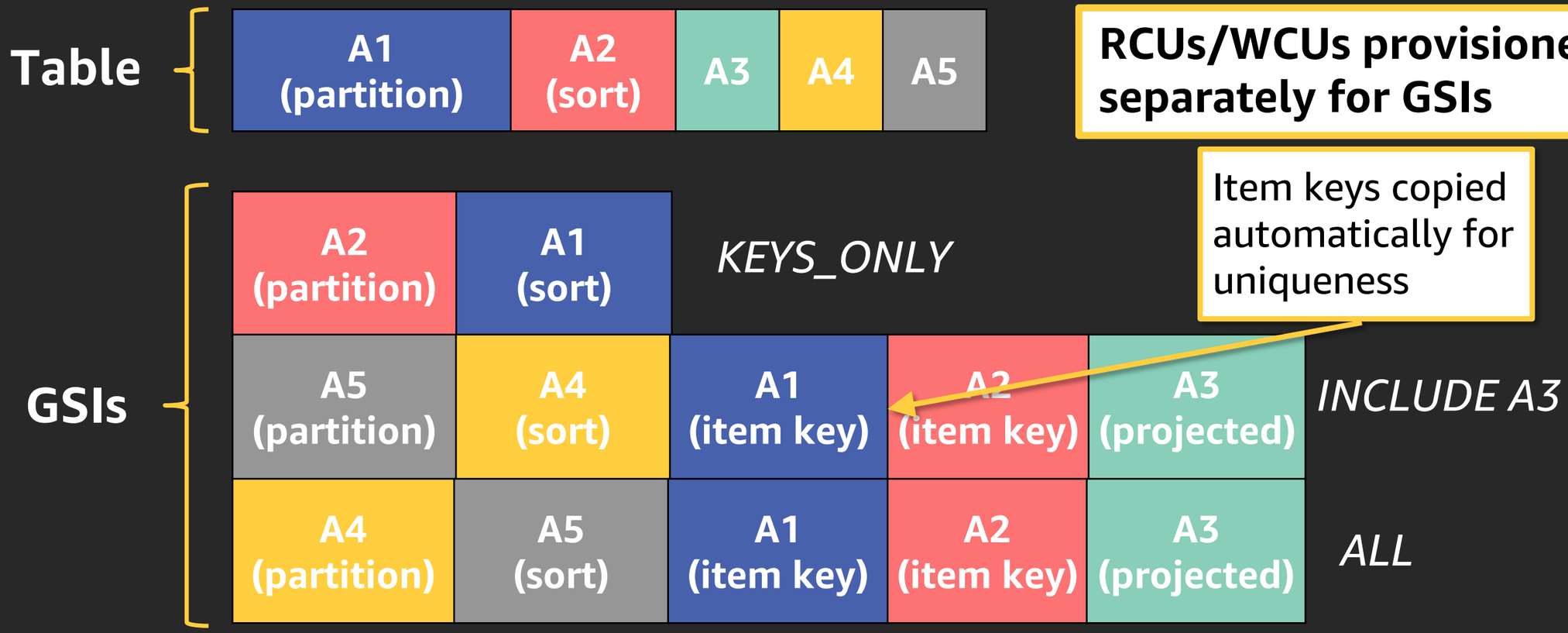
Mandatory
Key-value access pattern
Determines data distribution

Optional
Model 1:N relationships
Enables rich query capabilities

Global secondary index (GSI)

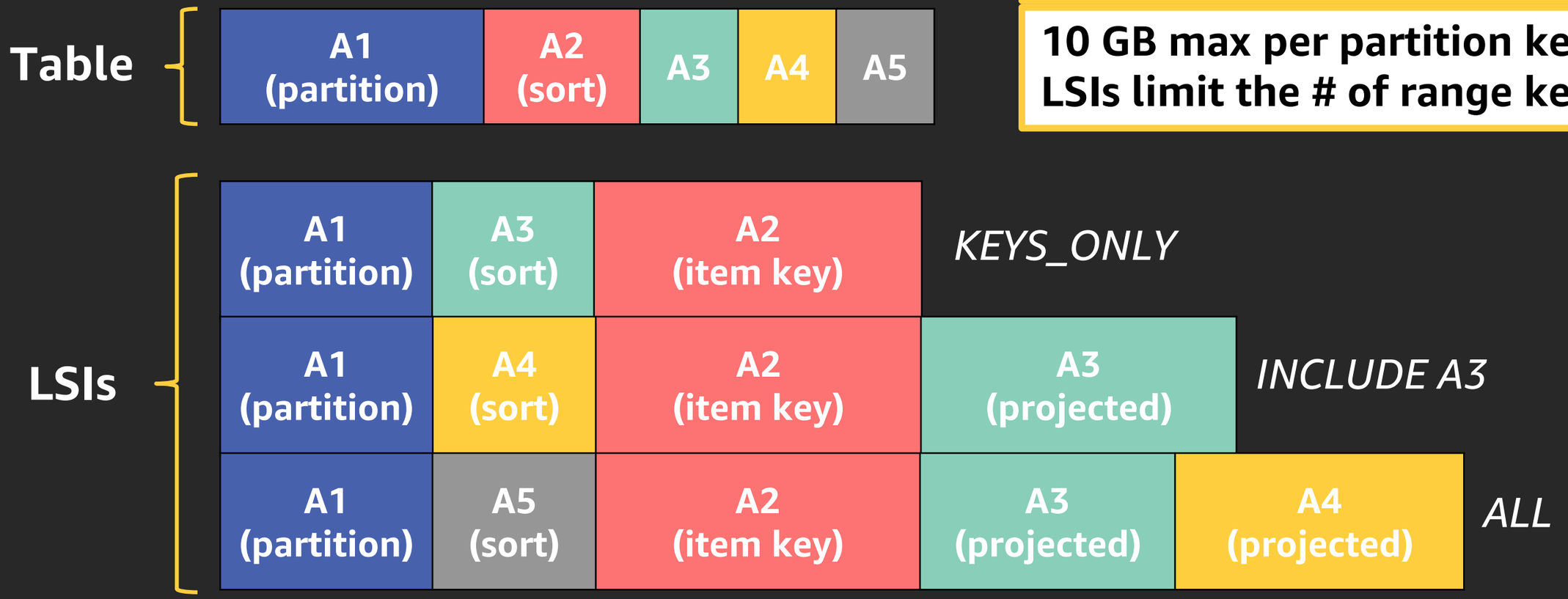
Up to 20 GSIs per table

Alternate partition and/or sort key
Index is across all partition keys
Use composite sort keys for compound indexes



Local secondary index (LSI)

Alternate sort key attribute
Index is local to a partition key



Collection limits
10 GB max per partition key. LSIs limit the # of range keys.

Distilled design concepts

Distilled design concepts

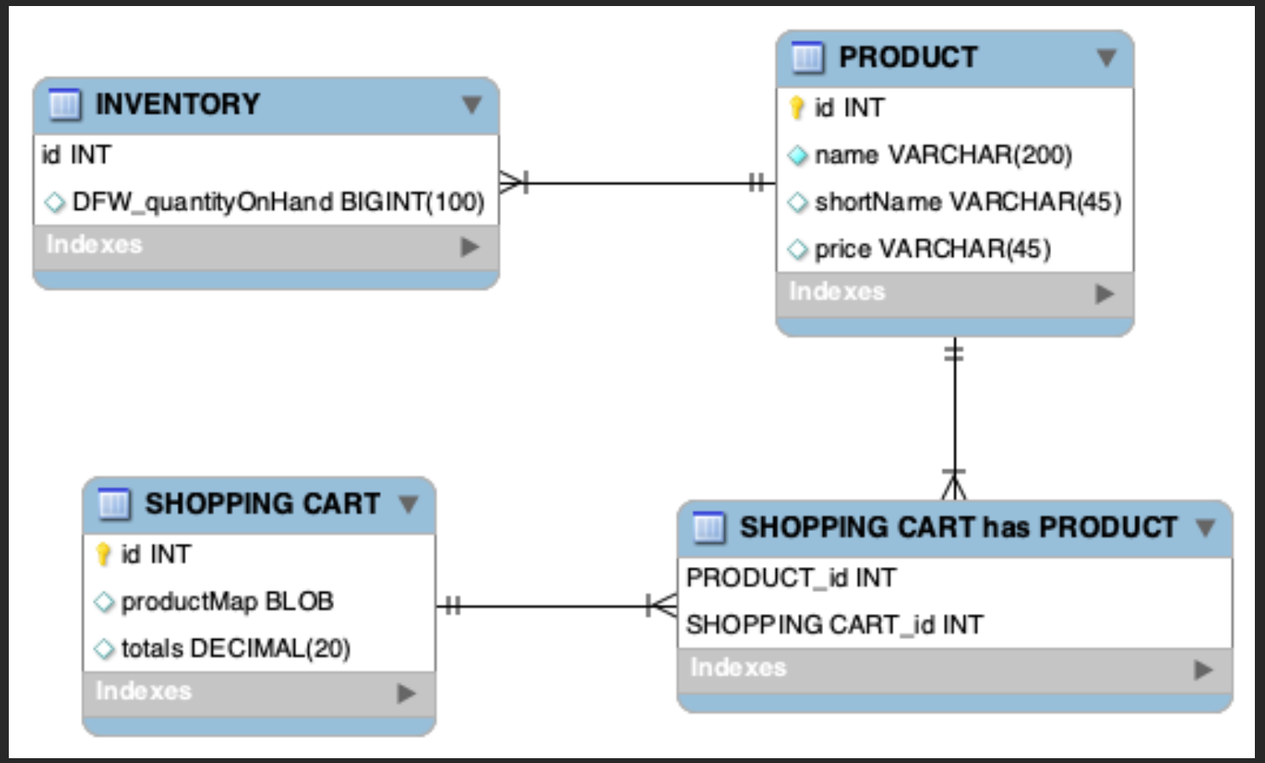
1. Understand the use case
2. Model the entity relationships
3. Write down the queries

1. Understand the use case

- ✓ Industry knowledge
- ✓ Is this a new or existing feature?
- ✓ Is DynamoDB the operational database?
- ✓ Who are the actors querying data from DynamoDB?

2. Model the relationships

- ✓ Entity-relationship diagrams are here to stay



3. Establish the queries

- ✓ **Get all `PRODUCTS` from `SHOPPING CART`**
- ✓ **Get all `SHOPPING CARTs` holding a `PRODUCT`**
- ✓ **Reserve the `PRODUCT` in `INVENTORY` during checkout**

Tip: think in SQL, and then convert to words

3. Establish the queries

- ✓ **Get all `PRODUCTS` from `SHOPPING CART`**

```
SELECT products FROM cart where cart_id = 'YYY';
```

- ✓ **Get all `SHOPPING CARTs` holding a `PRODUCT`**

```
SELECT cart_id FROM products_cart WHERE product_id = 'XXX';
```

- ✓ **Reserve the `PRODUCT` in `INVENTORY` during checkout**

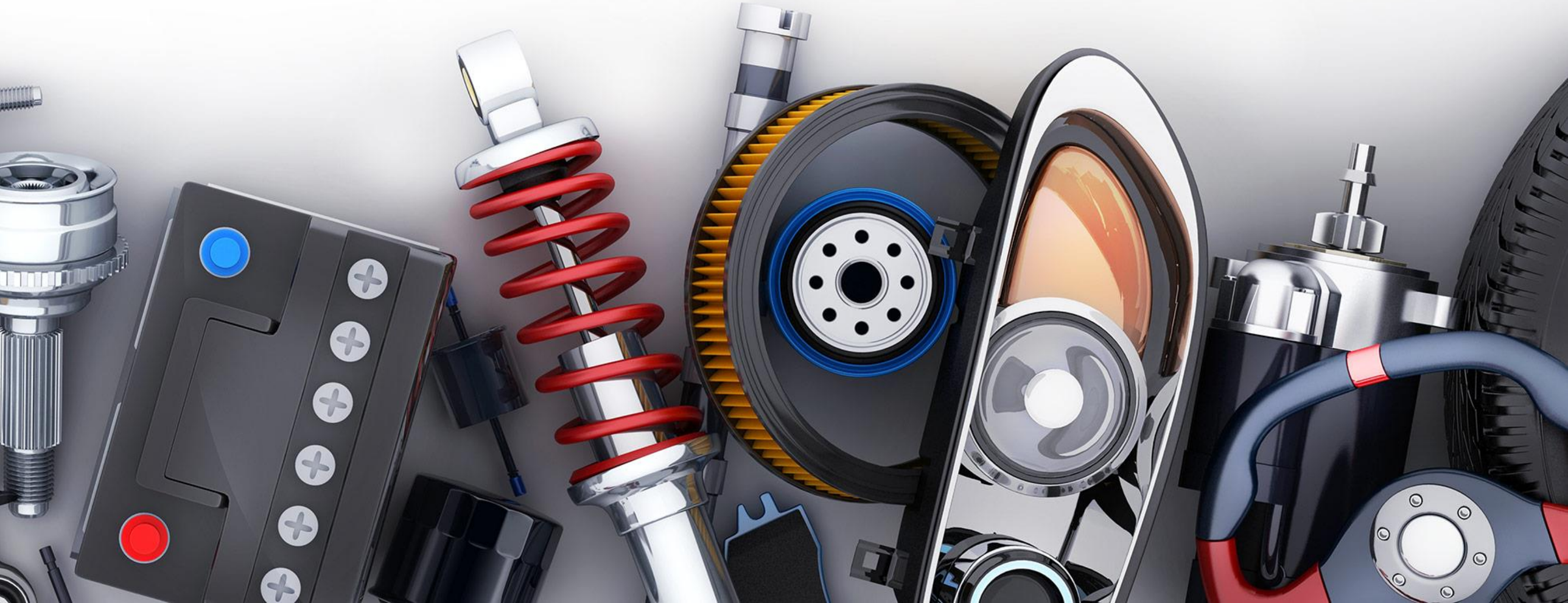
```
UPDATE inventory SET qoh = qoh - 1 WHERE product_id = 'XXX'
      AND qoh > 0;
```

Remember, DynamoDB is API driven and does not use SQL.

Real world use cases

Customer #1: Midwest industrial manufacturer

Use case: n:m relationships, data lake processing



Why Dynamo?

- Low latency
- Need durability
- Operational data

Schedule

8A - Start
10-noon - Lab
Noon-1p - EAT
1p-3p - Design

Possible partition keys:

- Process order id

Possible sort keys:

- ???

Queries:

1. Get all steps for process order

Why Dynamo?

- Low latency
- Need durability
- Operational data

Schedule

8A - Start
10-noon - Lab
Noon-1p - EAT
1p-3p - Design

Possible partition keys:

- Process order id

Possible sort keys:

- ???

Queries:

1. Get all steps for process order

Why Dynamo?

- ✓ Low latency
- ✓ Need durability
- ✓ Operational data

Schedule

8A - Start
10-noon - Lab
Noon-1p - EAT
1p-3p - Design

Possible partition keys:

- ✓ Process order id

Possible sort keys:

- ???

Queries:

1. Get all steps for process order

Real-world additive manufacturing

Process an order for a set of parts, and track it all!

Step 1: Gather "Materials"

e.g. Aluminum

Step 2: Mix "Materials"

Mix, and measure

mass and volume

Step 3: Print "Parts"

2 or more parts

e.g. drill bit

Step 4: Bake "Parts"

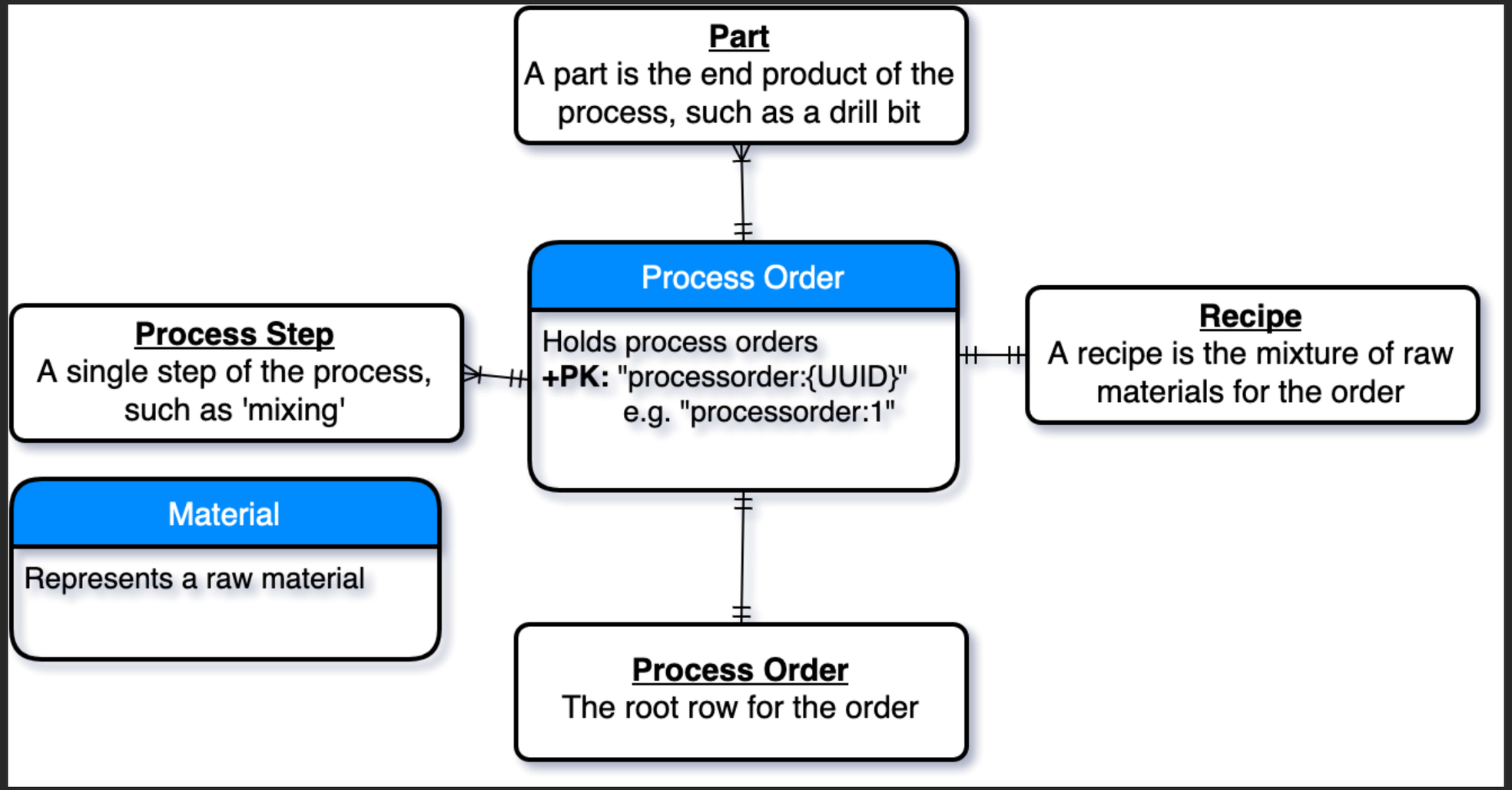
"Light the fires!"

Step 5: QC "Parts"

Microscopy

Measurement

Establish the entity relationships



Queries

- ✓ Get all steps for a `PROCESS ORDER`
- ✓ Get the AVERAGE density of a `MATERIAL`
- ✓ Find the `PROCESS ORDER` for a `PART`

Final schema

Primary Key		Attributes		
PK	SK			
processorder:1	part:1	porosity	mass	GSI_1_PK
		20%	122g	part:1
	processstep:materials:1	materialIds		
		[materialId:1, materialId:2]		
	processstep:mixing:2	mass	volume	density
		130g	26cm^3	5g/cm^3
	processstep:print:3	bedTemp	materiallTemp	adhesionSurface
		120	400	tape
	recipes	processstep:materials		
		recipe:1		

Primary Key		Attributes			
PK	SK				
materialid:1	processorder:1	mass	volume	density	timestamp
		45g	9cm^3	5g/cm^3	1575423000
	processorder:2	mass	volume	density	timestamp
		66g	13.2cm^3	5g/cm^3	1575477000
materialid:2	processorder:1	mass	volume	density	timestamp
		85g	17cm^3	5g/cm^3	1575423000
	processorder:2	mass	volume	density	timestamp
		64g	12.8cm^3	5g/cm^3	1575477000

Adjacency lists

Generic partition and sort key names

Often used with GSI overloading

Primary Key		Attributes		
PK	SK	porosity	mass	GSI_1_PK
processorder:1	part:1	20%	122g	part:1
	processstep:materials:1	materialIds		
	processstep:mixing:2	[materialId:1, materialId:2]	volume	density
	processstep:print:3	130g	26cm^3	5g/cm^3
		bedTemp	materiallTemp	adhesionSurface
		120	400	tape

Soft associations for n:m relationship

Primary Key		Attributes			
PK	SK	mass	volume	density	timestamp
materialid:1	processorder:1	*45g*	9cm^3	5g/cm^3	1575423000
	processorder:2	mass	volume	density	timestamp
materialid:2	processorder:1	*66g*	13.2cm^3	5g/cm^3	1575477000
		mass	volume	density	timestamp
	processorder:2	*85g*	17cm^3	5g/cm^3	1575423000
		mass	volume	density	timestamp
		64g	12.8cm^3	5g/cm^3	1575477000

Queries

Primary Key		Attributes		
PK	SK			
processorder:1	part:1	porosity	mass	GSI_1_PK
		20%	122g	part:1
	processstep:materials:1	materialIds		
		[materialId:1, materialId:2]		
	processstep:mixing:2	mass	volume	density
		130g	26cm^3	5g/cm^3
	processstep:print:3	bedTemp	materiallTemp	adhesionSurface
		120	400	tape
	recipes	processstep:materials		
		recipe:1		

Primary Key		Attributes			
PK					
materialid:1					timestamp
					1575423000
					timestamp
					1575477000
materialid:2					timestamp
					1575423000
					timestamp
					1575477000

Query #1: Get all steps for a `PROCESS ORDER`

Solution: Query with key_condition

```
PK = process_order:1
AND SK starts_with("processstep")
```

Queries

Primary Key		Attributes		
PK	SK			GSI_1_PK
processor				part:1
				density
				5g/cm^3
				adhesionSurface
				tape
	recipes			
		recipe:1		

Query #2: Get the AVERAGE density of a `MATERIAL`

Solution: Query with key_condition `PK = materialId:1` and on the client compute the `AVERAGE`

Primary Key		Attributes			
PK	SK				
materialid:1	processorder:1	mass	volume	density	timestamp
		45g	9cm^3	5g/cm^3	1575423000
	processorder:2	mass	volume	density	timestamp
		66g	13.2cm^3	5g/cm^3	1575477000
materialid:2	processorder:1	mass	volume	density	timestamp
		85g	17cm^3	5g/cm^3	1575423000
	processorder:2	mass	volume	density	timestamp
		64g	12.8cm^3	5g/cm^3	1575477000

Queries

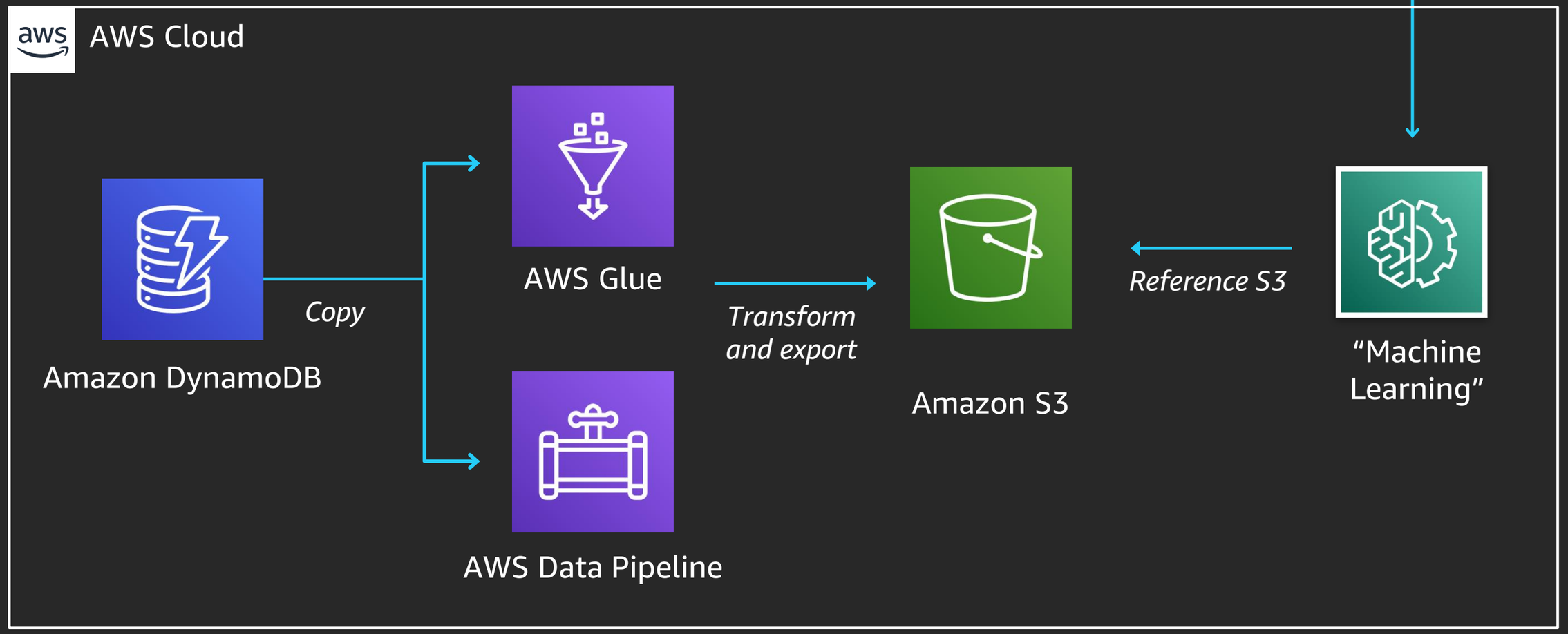
Primary Key		Attributes		
PK	SK			
processorder:1	part:1	porosity	mass	GSI_1_PK
		20%	122g	part:1
	processstep:materials:1	materialIds		
		[materialId:1, materialId:2]		
	processstep:mixing:2	mass	volume	density
		130g	26cm^3	5g/cm^3
	processstep:print:3	bedTemp	materiallTemp	adhesionSurface
		120	400	tape
	recipes	processstep:materials		
		recipe:1		

Primary Key					
PK					
materialid:1					timestamp
					1575423000
					timestamp
					1575477000
materialid:2					timestamp
					1575423000
					timestamp
		64g	12.8cm^3	5g/cm^3	1575477000

Query #3: Find the `PROCESS ORDER` for a `PART`

Solution: Query GSI 1 with key_condition `GSI_1_PK = part:1`

Data lake: Export to Amazon S3

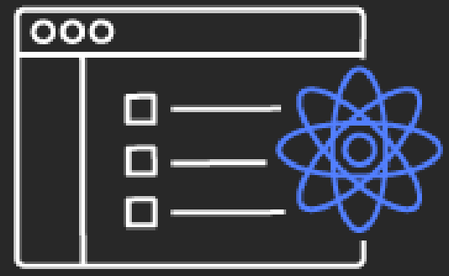


Recap

✓ Adjacency lists

Good for:

- n:m relationships
- Relational data



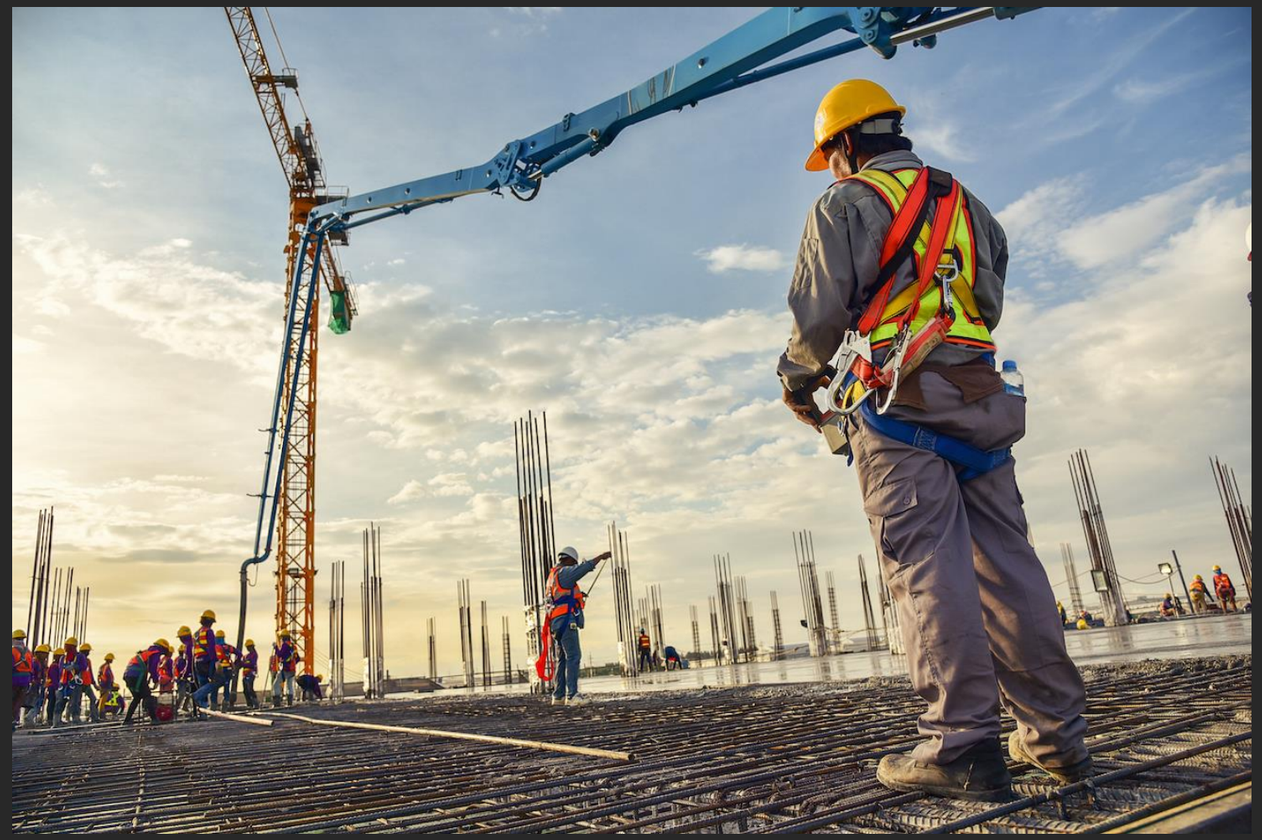
✓ Export data to S3 with AWS

Good for:

- Machine learning
- Data lakes

Customer #2: Heavy machinery

Use case: access control database, n:m consistency



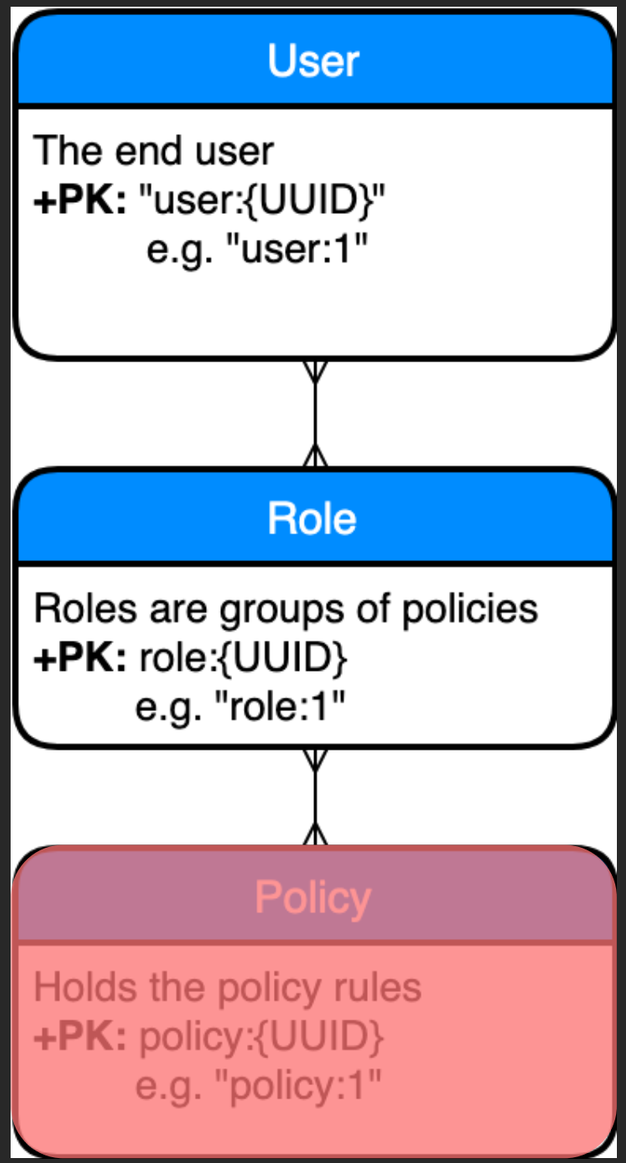
Use case

User

Role

Policy

Entity relationships (original)



Get user → Get roles for user → Get policies for role

10ms per read, limited scalability

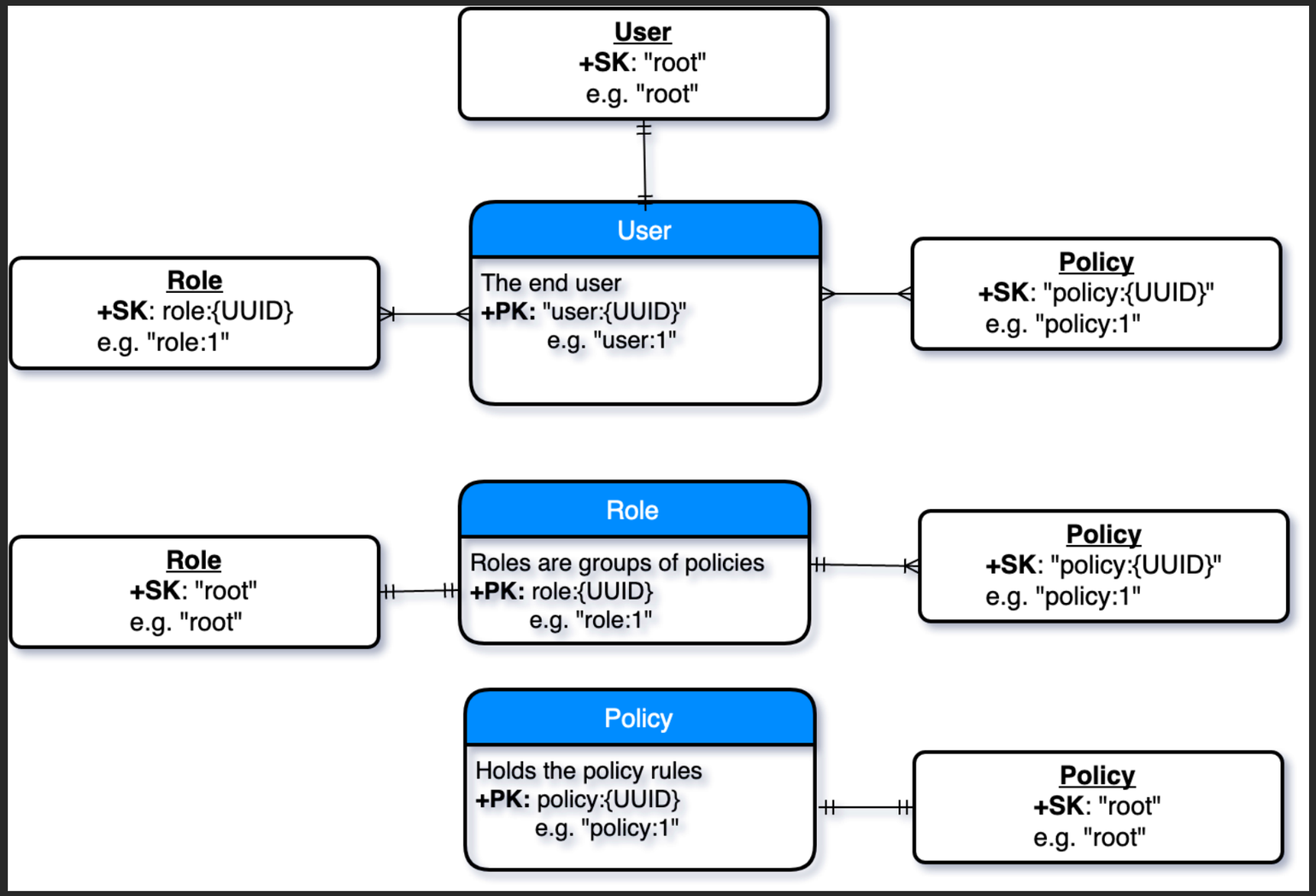
All data is relational

IT monitoring

Social graph

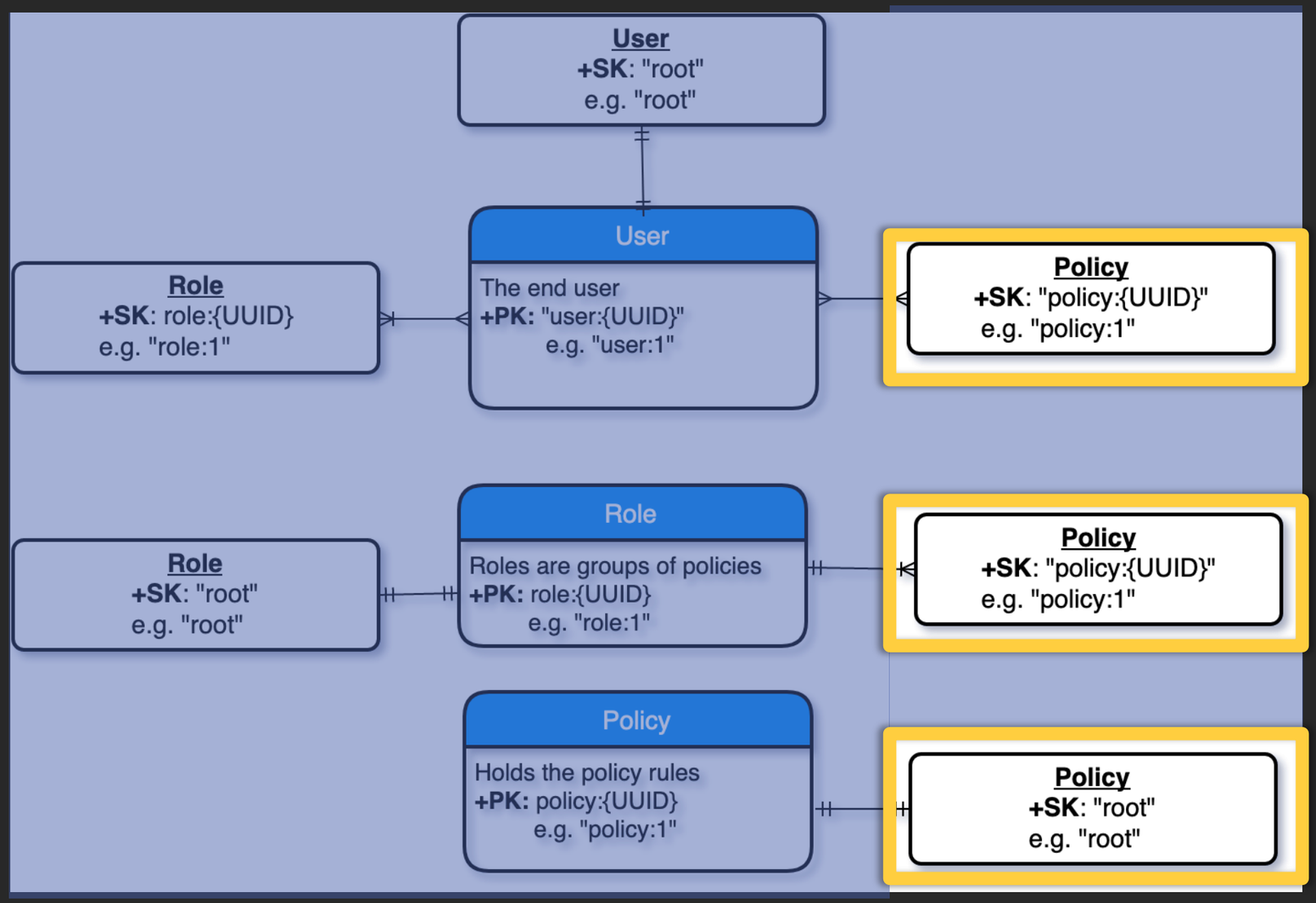
Documents

Entity relationships (fixed)



Aggregation → scalability

Entity relationships



Aggregation → scalability

Queries

- ✓ Get all policies for a `USER`
- ✓ Find all `POLICIES` for a `ROLE`
- ✓ Find all `USERS` in an OrgId

Final schema

Primary Key		Attributes	
PK	SK		
policy:1	root	policyName	policy
		Admin	{ … }
policy:2	root	policyName	policy
		RejectWrites	{ … }
policy:3	root	policyName	policy
		RevokeAssumeRole	{ … }

Primary Key		Attributes	
PK	SK		
role:1	root	roleName	
		Default role	
	policy:1	policyName	GSI_1_PK
		Admin	role:policy:1
	policy:2	policyName	GSI_1_PK
		RejectWrites	role:policy:2
	policy:3	policyName	GSI_1_PK
		RevokeAssumeRole	role:policy:3

Primary Key		Attributes		
PK	SK			
user:1	root	userName	orgId	GSI_1_PK
		John Doe	org:1	org:1
	role:1	roleName	GSI_1_PK	
		Default role	role:1	
	policy:1	policyName	policy	GSI_1_PK
		Admin	{ … }	user:policy:1
	policy:2	policyName	policy	GSI_1_PK
		RejectWrites	{ … }	user:policy:2
	policy:3	policyName	policy	GSI_1_PK
		RevokeAssumeRole	{ … }	user:policy:3

Queries

Query #1: Get all policies for a `USER`

Solution: Query with key_condition:
`PK = user:1 AND SK starts_with("policy")`

Primary Key		Attributes		
PK	SK			
	root	userName	orgId	GSI_1_PK
		John Doe	org:1	org:1
	role:1	roleName	GSI_1_PK	
		Default role	role:1	
user:1	policy:1	policyName	policy	GSI_1_PK
		Admin	{ … }	user:policy:1
	policy:2	policyName	policy	GSI_1_PK
		RejectWrites	{ … }	user:policy:2
	policy:3	policyName	policy	GSI_1_PK
		RevokeAssumeRole	{ … }	user:policy:3

Queries

Primary Key		Attributes	
PK	SK		
	root	roleName	
		Default role	
role:1	policy:1	policyName	GSI_1_PK
		Admin	role:policy:1
	policy:2	policyName	GSI_1_PK
		RejectWrites	role:policy:2
	policy:3	policyName	GSI_1_PK
		RevokeAssumeRole	role:policy:3

Query #2: Find all `POLICIES` for a `ROLE`

Solution: Query with key_condition:

```
PK = role:1 AND SK starts_with("policy")
```

Queries

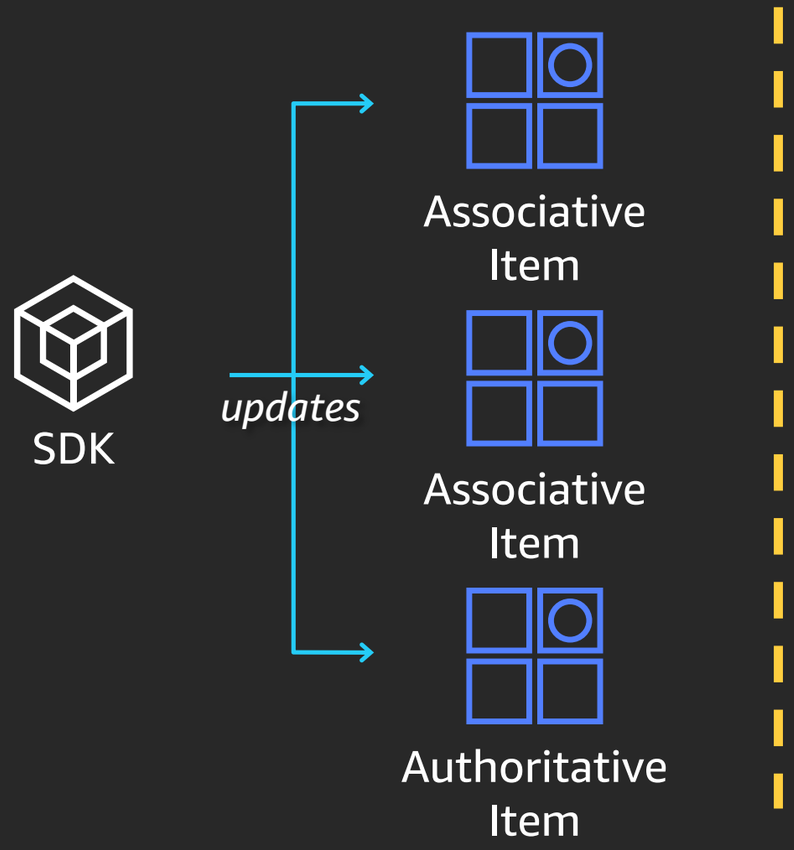
Query #3: Find all `USERS` in an OrgId

Solution: Query GSI 1 with key_condition: `PK = org:1`
Note: The full row is projected into the GSI

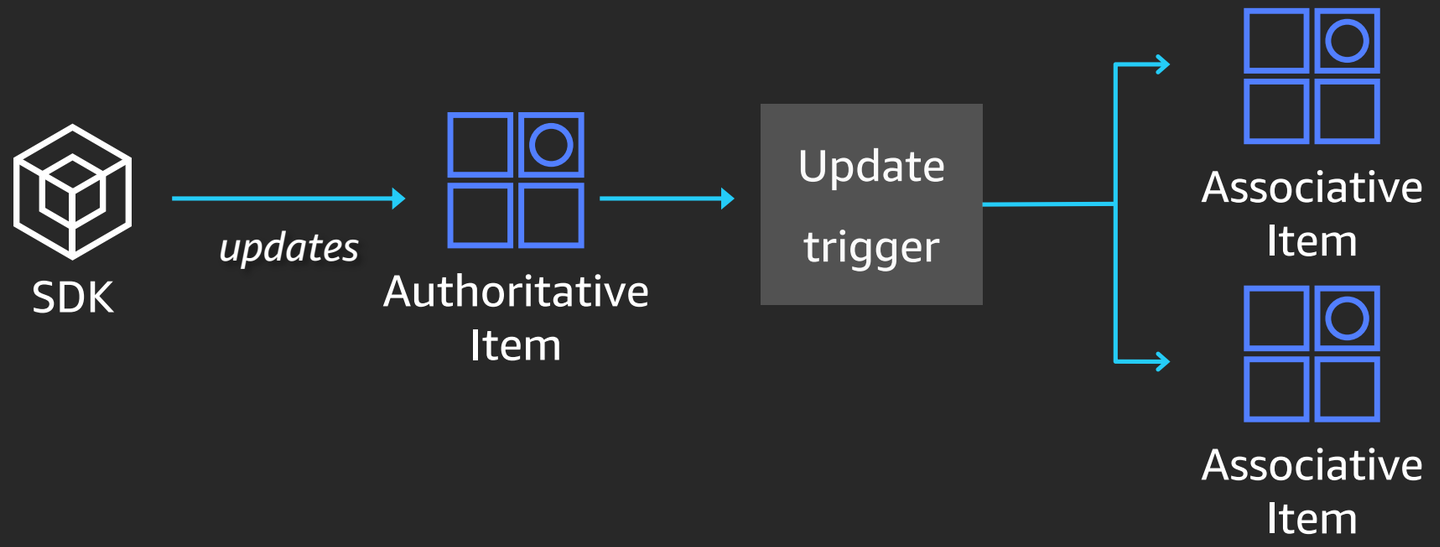
Primary Key		Attributes		
PK	SK			
	root	userName	orgId	GSI_1_PK
		John Doe	org:1	org:1
user:1	role:1	roleName	GSI_1_PK	
		Default role	role:1	
	policy:1	policyName	policy	GSI_1_PK
		Admin	{ … }	user:policy:1
	policy:2	policyName	policy	GSI_1_PK
		RejectWrites	{ … }	user:policy:2
	policy:3	policyName	policy	GSI_1_PK
		RevokeAssumeRole	{ … }	user:policy:3

Update strategies for references

Immediate writes

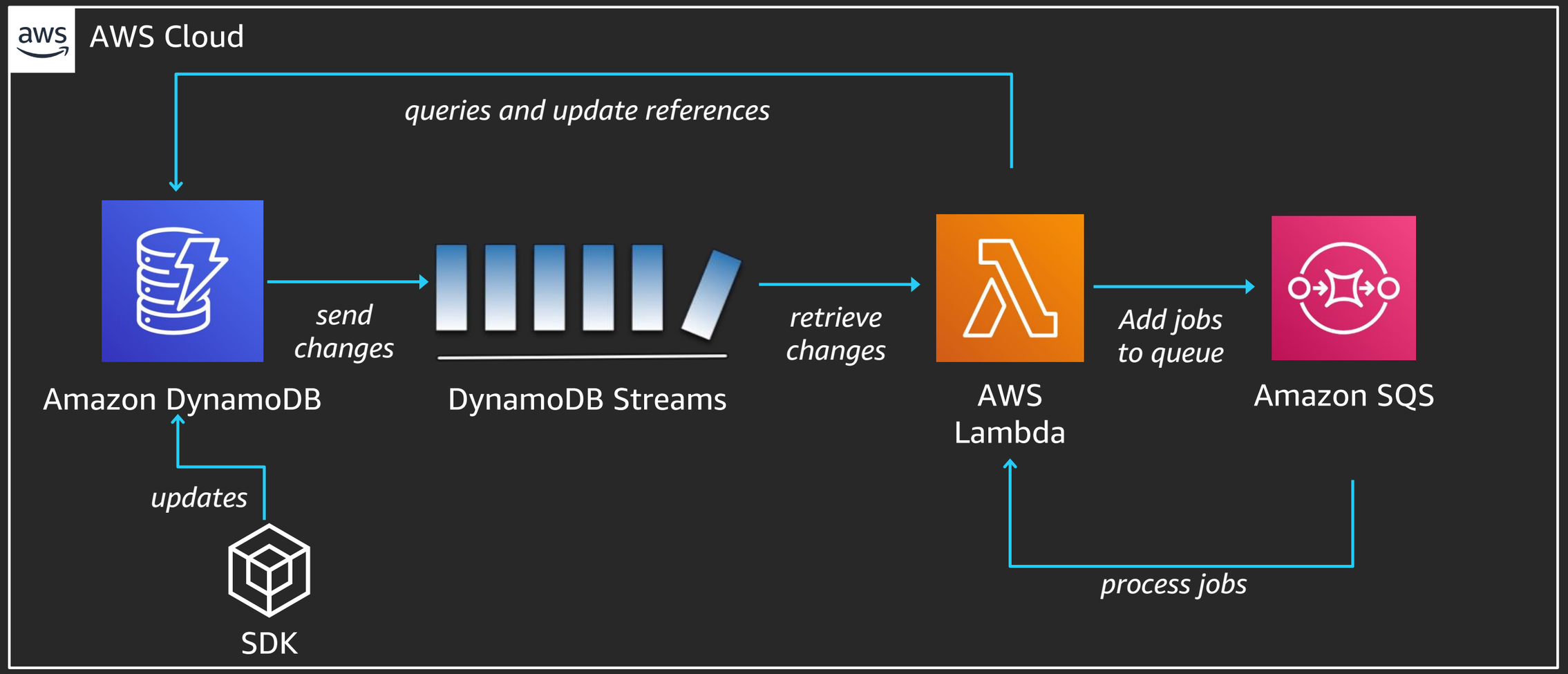


Triggers with DynamoDB streams



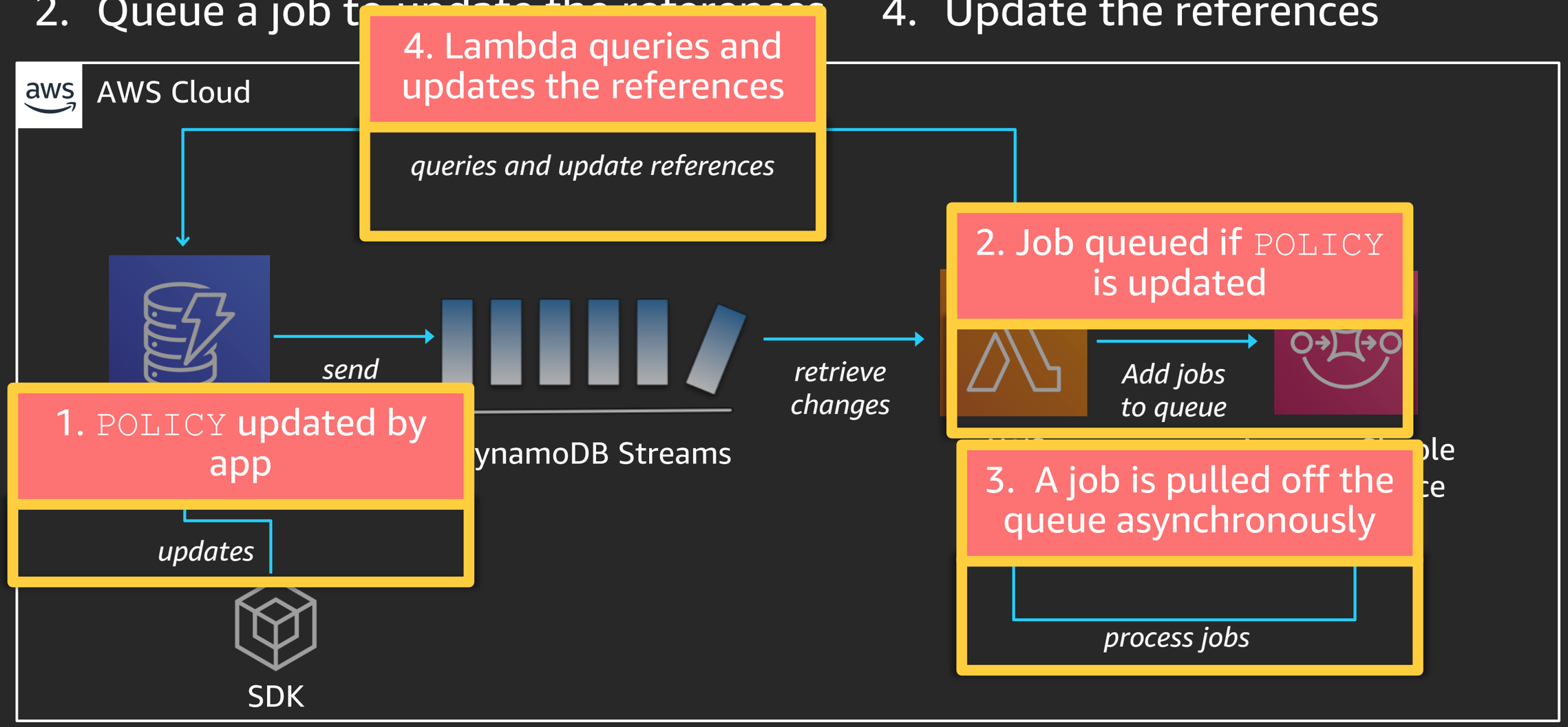
Update triggers

1. Update a `POLICY` in DynamoDB
2. Queue a job to update the references
3. Process the job
4. Update the references



Update triggers

1. Update a `POLICY` in DynamoDB
2. Queue a job to update the references
3. Process the job
4. Update the references



Recap

- ✓ Aggregate for performance

Good for:

- Relational data

- ✓ Trigger based updates

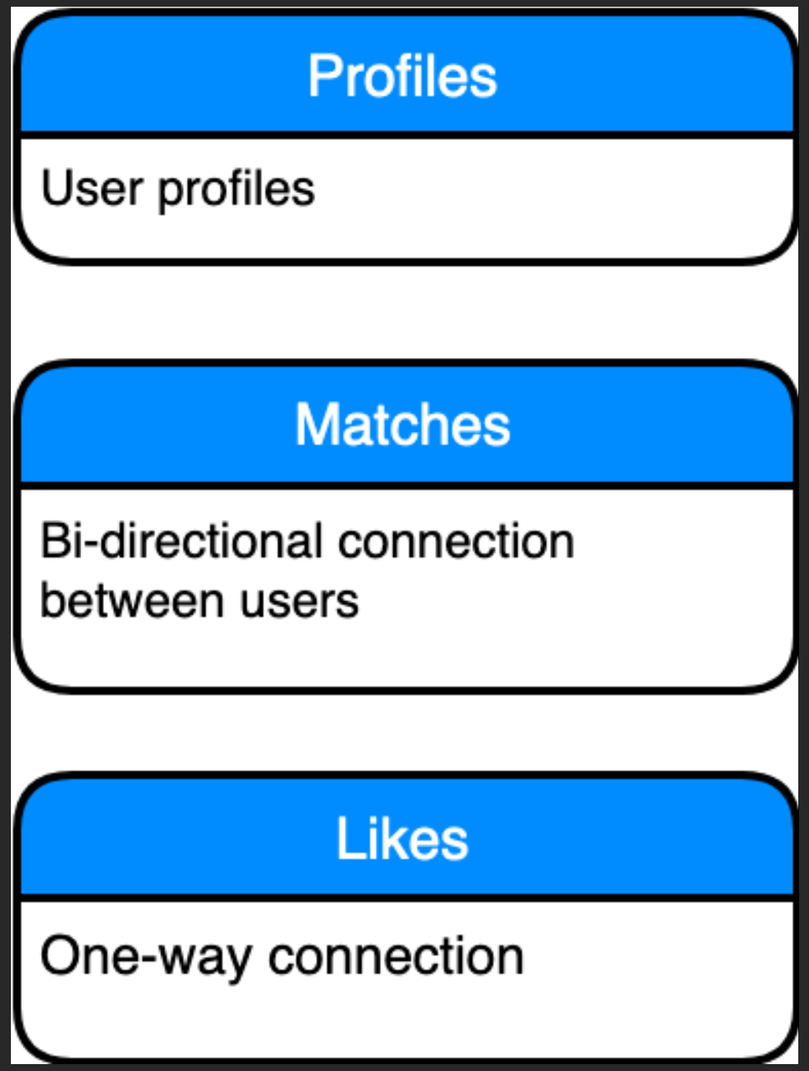
Good for:

- Horizontal scaling
- Relational data

Customer #3: Dating app

Use case: job scheduling

Entity relationships



Queries

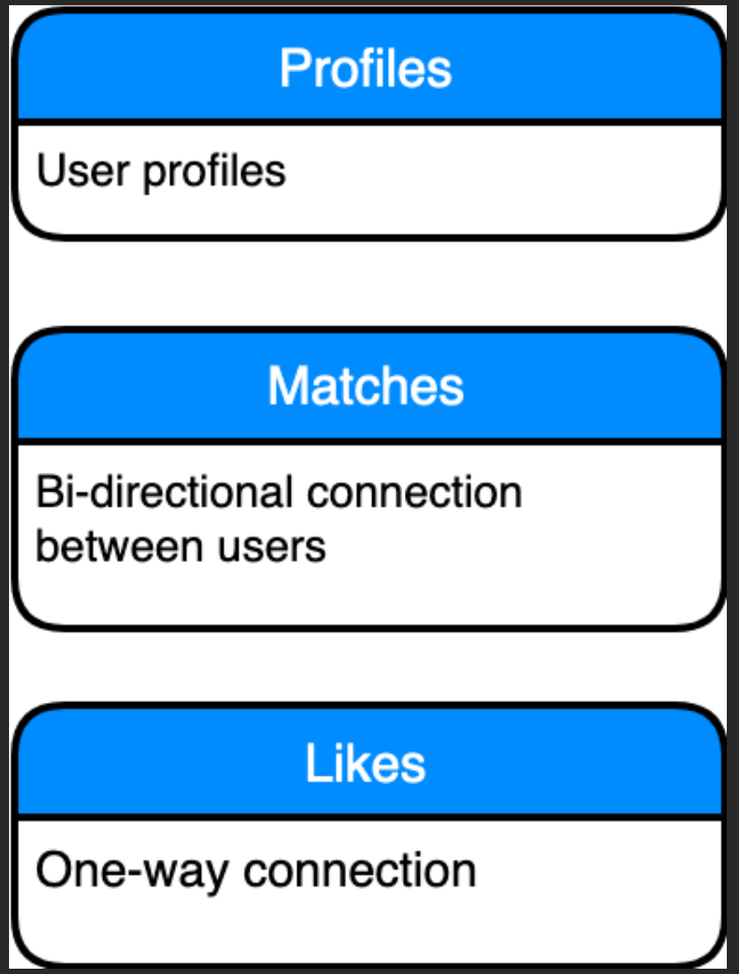
- ✓ Get `MATCHES` for a user
- ✓ For a user, find who `LIKED` them
- ✓ Find `MATCHES` that are stale

Table schema

All together, now

Keep in mind:

- Establish the primary keys first, and then add the GSIs
- Not all entities will be used to establish the queries
- The solution will use one or two queries
- A stale match is one that has not been seen by either party in the last 30 days



✓ Get `MATCHES` for a user

✓ For a user, find who `LIKED` them

✓ Find `MATCHES` that are stale

Final table schema

Primary Key		Attributes					
PK	SK						
match:1	root	user1	user2	GSI_1_PK	GSI_2_PK	GSI_3_PK	GSI_3_SK
		user:1234	user:4567	user:1234	user:4567	Match#(0-N)	2019-12-31T03:44
match:2	root	user1	user2	GSI_1_PK	GSI_2_PK	GSI_3_PK	GSI_3_SK
		user:9833	user:4590	user:9833	user:4590	Match#(0-N)	2019-12-31T03:44

Primary Key		Attributes			
PK	SK				
user:1234	user:4567	Status	deleted	GSI_1_PK	GSI_1_SK
		active	FALSE	user:like:4567	1577829985
user:4567	user:1234	Status	deleted	GSI_1_PK	GSI_1_SK
		active	FALSE	user:like:1234	1577833583

Table schema

Query #1: Get `MATCHES` for a user
Solution: : Query GSI 1 with key_condition `GSI_PK_1 = user:1234`, Query GSI 2 with key_condition `GSI_PK_2 = user:1234`

Primary Key		Attributes					
PK	SK						
match:1	root	user1	user2	GSI_1_PK	GSI_2_PK	GSI_3_PK	GSI_3_SK
		user:1234	user:4567	user:1234	user:4567	Match#(0-N)	2019-12-31T03:44
match:2	root	user1	user2	GSI_1_PK	GSI_2_PK	GSI_3_PK	GSI_3_SK
		user:9833	user:4590	user:9833	user:4590	Match#(0-N)	2019-12-31T03:44

#1

#3

Primary Key		Attributes			
PK	SK				
user:1234	user:4567	Status	deleted	GSI_1_PK	GSI_1_SK
		active	FALSE	user:like:4567	1577829985
user:4567	user:1234	Status	deleted	GSI_1_PK	GSI_1_SK
		active	FALSE	user:like:1234	1577833583

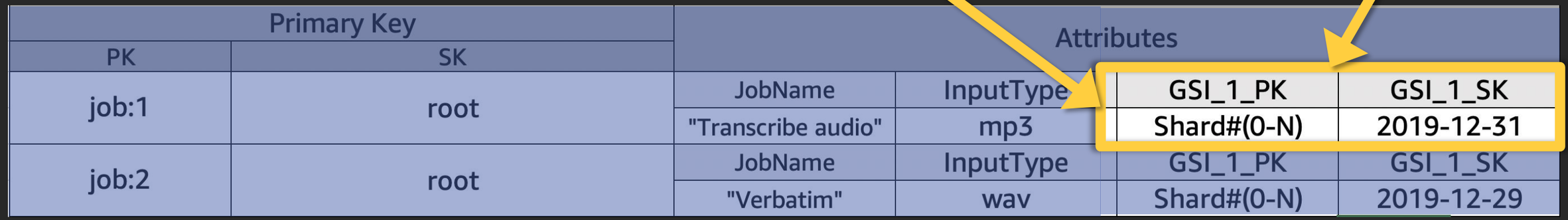
#2

Query #2: For a user, find who `LIKED` them
Solution: : Query GSI 1 with key_condition `GSI_PK_1 = user::like:1234`

Query #3: Find `MATCHES` that are stale
Solution: : Query every Match shard of GSI 3 with key_condition `GSI_PK_1 = Shard#(SHARDNUMBER) AND GSI_1_SK < 2020-01-15`

GSI overloading

Reuse generic index keys for different access patterns in the same table



Primary Key		Attributes			
PK	SK				
job:1	root	JobName	InputType	GSI_1_PK	GSI_1_SK
		"Transcribe audio"	mp3	Shard#(0-N)	2019-12-31
job:2	root	JobName	InputType	GSI_1_PK	GSI_1_SK
		"Verbatim"	wav	Shard#(0-N)	2019-12-29

Recap

- ✓ Migrate to DynamoDB

  Good for:

  - Operational workloads

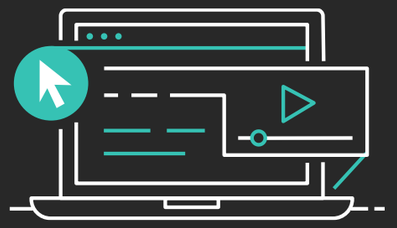
- ✓ Scalable search with GSI write sharding

  Good for:

  - Scheduled jobs
  - Custom expiry rules

Learn databases with AWS Training and Certification

Resources created by the experts at AWS to help you build and validate database skills



25+ free digital training courses cover topics and services related to databases, including:

- Amazon Aurora
- Amazon Neptune
- Amazon DocumentDB
- Amazon DynamoDB
- Amazon ElastiCache
- Amazon Redshift
- Amazon RDS

Validate expertise with the new **AWS Certified Database - Specialty** beta exam

Visit aws.training

aws training and certification

Thank you!

Sean Shriver & Ankur Kasliwal

@sean_shriver, @ankurrk, and
@DynamoDB

Please complete the session survey in the mobile app.