DAT401

Advanced Design Patterns for DynamoDB

Rick Houlihan
Principal Technologist, NoSQL
AWS

Agenda

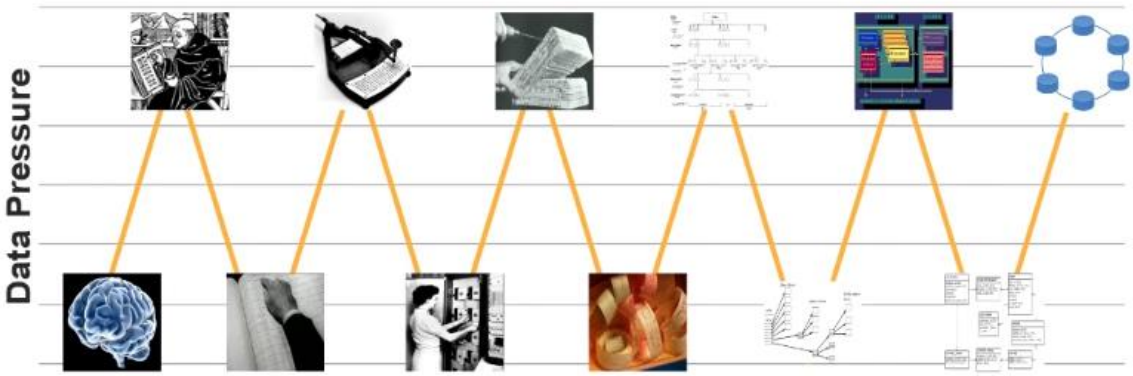
- Brief History of Data Processing (Why NoSQL?)
- Overview of DynamoDB
- NoSQL Data Modeling
  - Normalized versus De-normalized schema
- Common NoSQL Design Patterns
  - Composite Keys, Hierarchical Data, Relational Data
- Modeling Real Applications

History of Data Processing

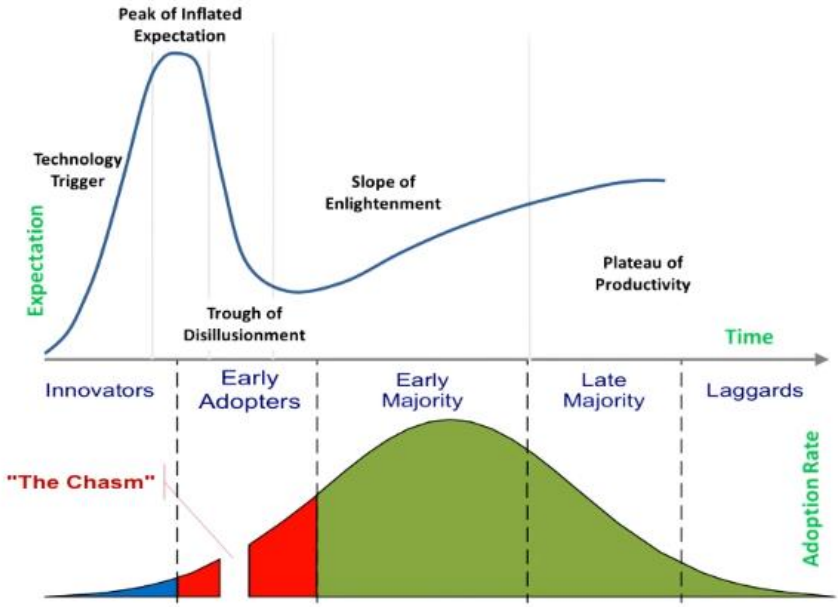
"History repeats itself because nobody was listening the first time"

- Anonymous

Timeline of Database Technology



Technology Adoption and the Hype Curve

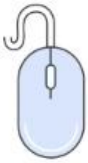


Why NoSQL?

SQL	NoSQL
Optimized for storage	**Optimized for compute**
Normalized/relational	Denormalized/hierarchical
Ad hoc queries	Instantiated views
Scale vertically	Scale horizontally
Good for OLAP	Built for OLTP at scale

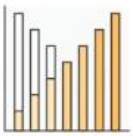
When choosing a database for a system, we need to ask what type of access patterns are we dealing with in our application?

Amazon DynamoDB

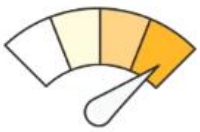


Fully Managed NoSQL

Document or Key-Value



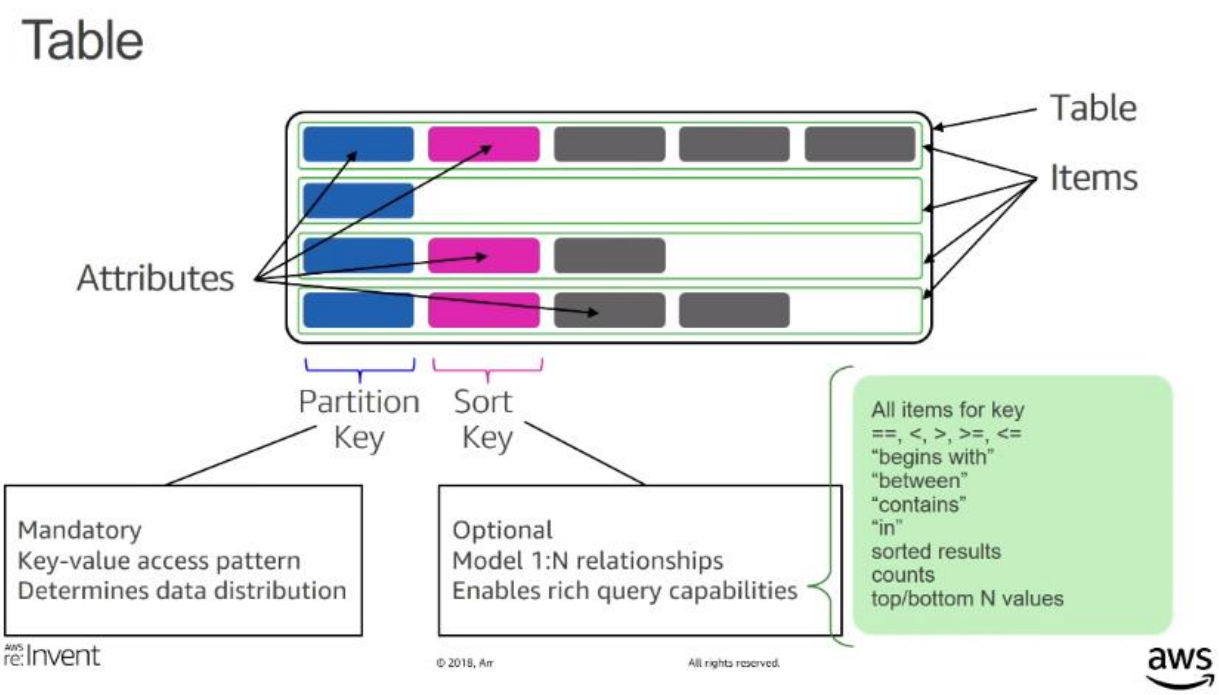
Scales to Any Workload



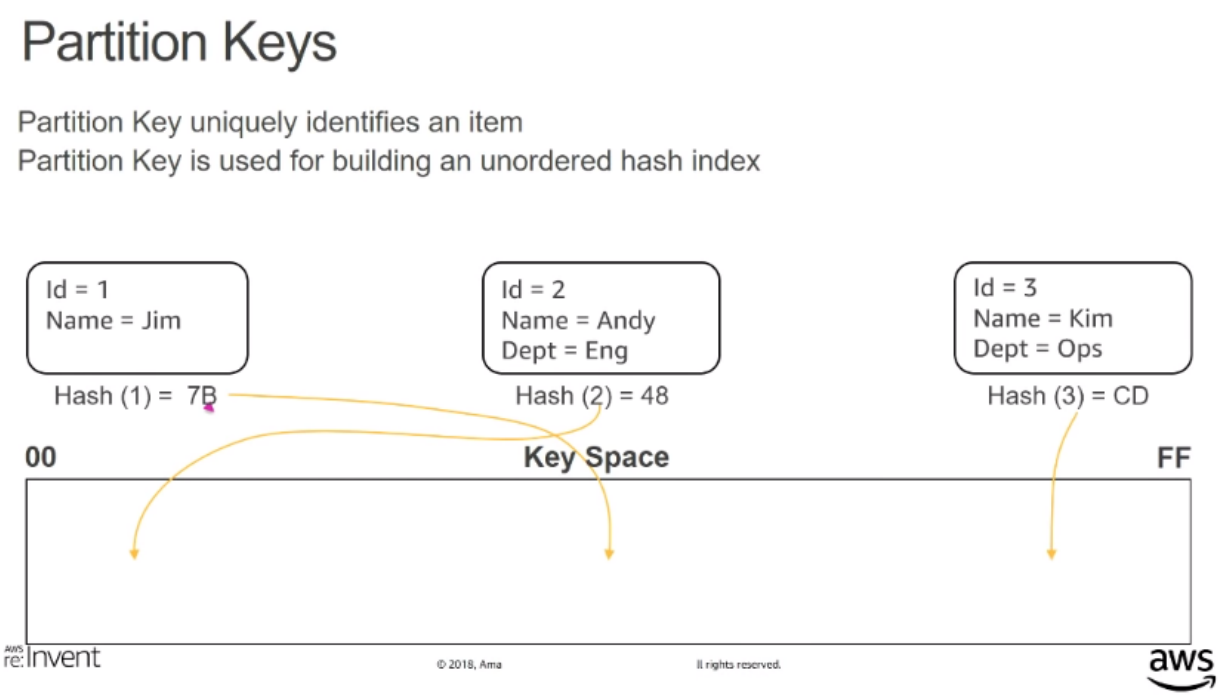
Fast and Consistent

Access Control

Event Driven Programming

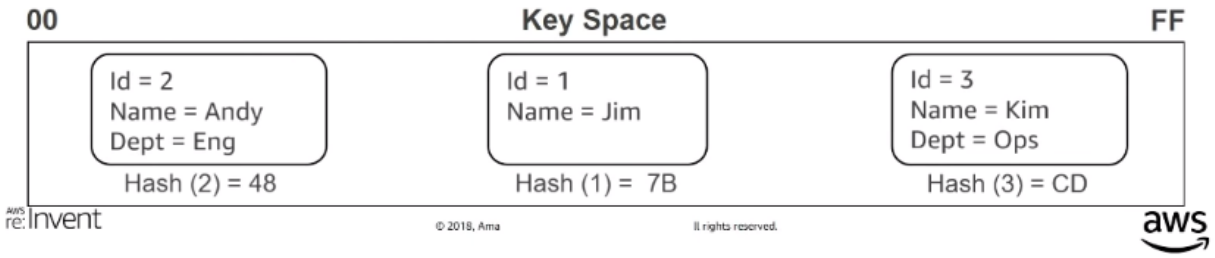


You also need to design the tables to support one of your primary data access patterns



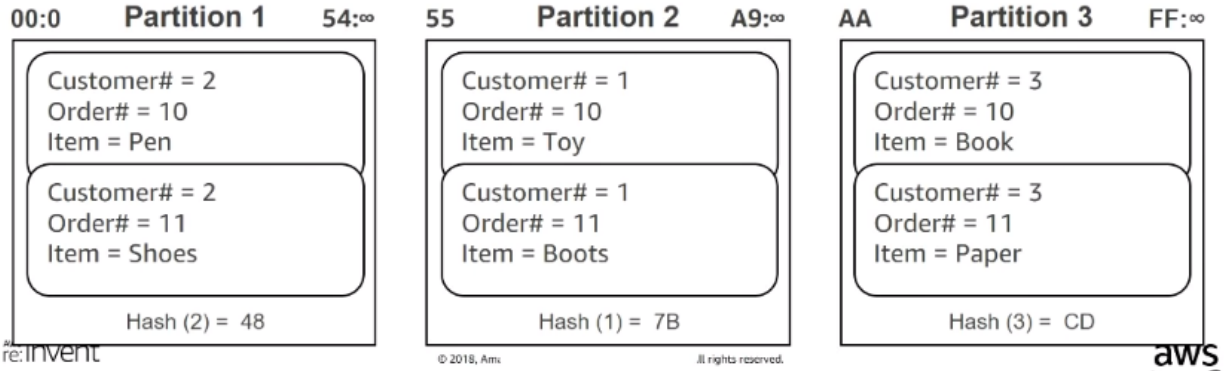
Partition Keys

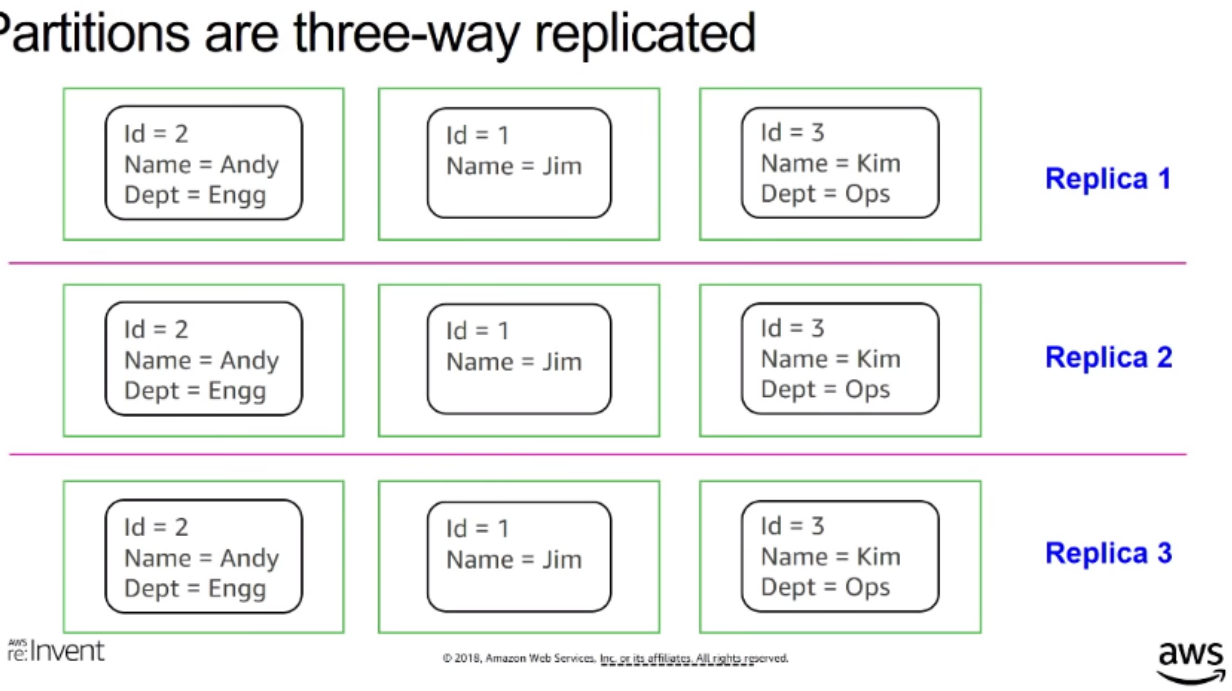
Partition Key uniquely identifies an item
Partition Key is used for building an unordered hash index



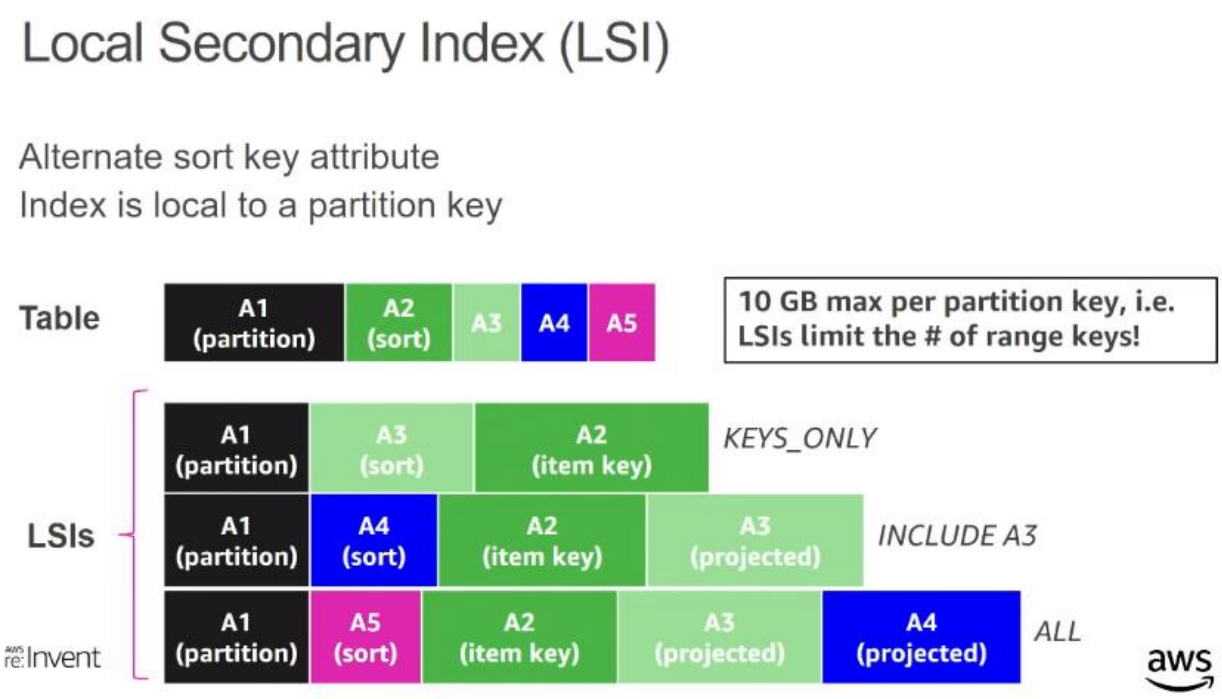
Partition:Sort Key

Partition:Sort Key uses two attributes together to uniquely identify an Item
Within unordered hash index, data is arranged by the sort key
No limit on the number of items (∞) per partition key
- Except if you have local secondary indexes

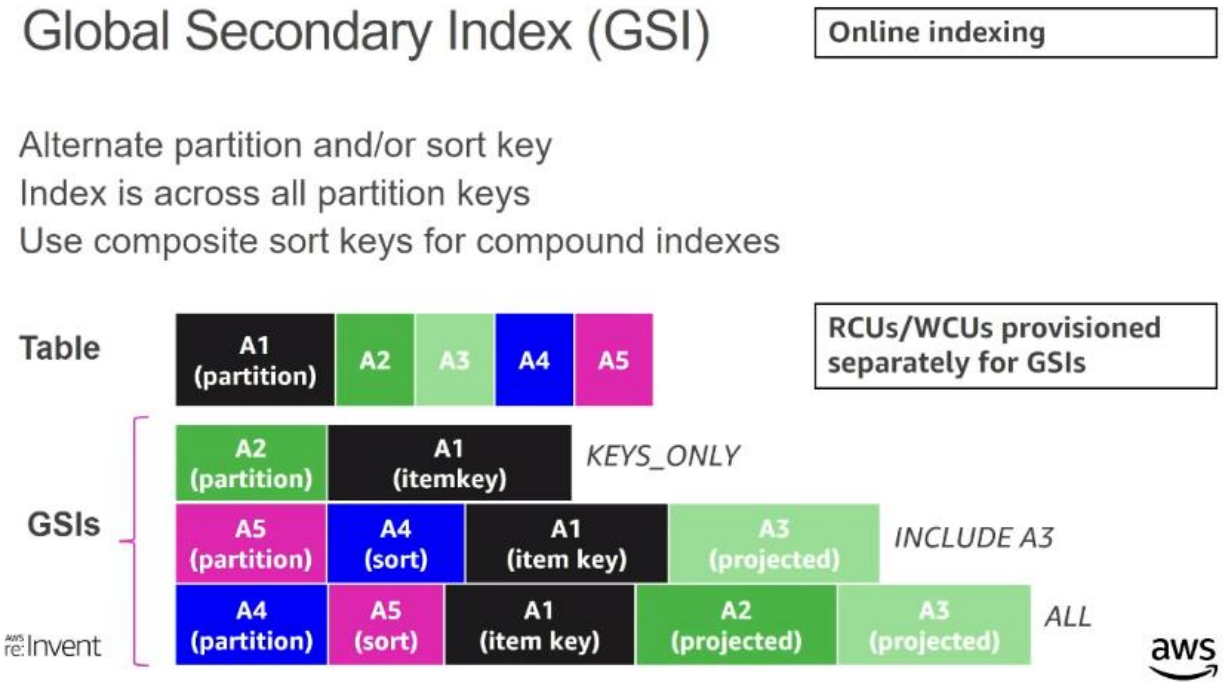




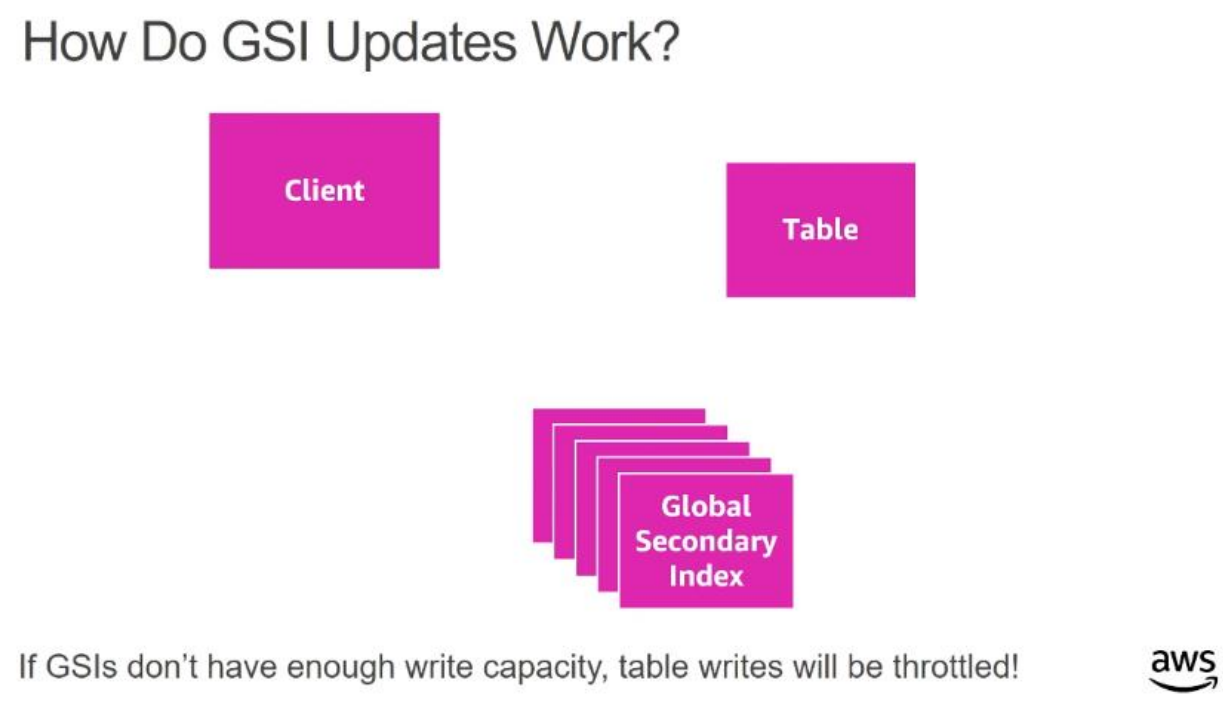
When you read from DynamoDB, you have a choice between eventually consistent and strongly consistent reads. The difference between reading from the primary or the secondary node is a sub-1 millisecond delay, this allows eventually consistent reads to be half the cost of a strongly consistent read because we have more nodes to choose from and randomly read from one of those 3 replica partitions. Making a strongly consistent read means that the data is read from the primary node only since it always accepts the writes. A cheap way to double the capacity of your database/application is to use **ecreads**, **the ecreads parameter is not on by default in your DynamoDB database**.



LSI allows you to re-sort the data in the partitions. If orderID is the primary key for orders data and the orderDate is the sort key, on the GSI or LSI we then create a partition key using the orderID to use to get data related to the orderID directly from the partition quickly. We can now use the LSI to get all the back ordered items for a customer using the orderID or get all the orders in the last 24 hours using the sort key oderDate less than 24 hours.

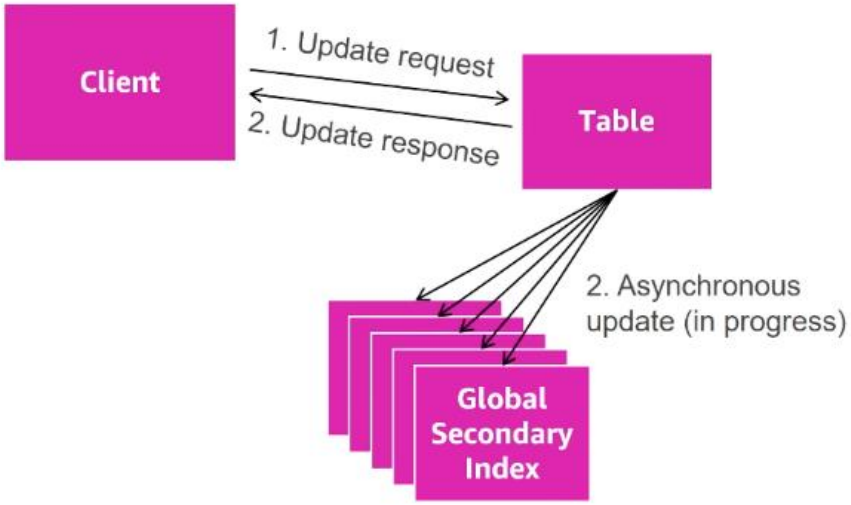


GSI allows us to create a completely new aggregation of the data, since the primary table is grouping the orders by customerID, then the GSI can group the orders by warehouseID. Then the partition key on the GSI will be the warehouseID and the sort key will be the orderDate. We can then get the latest orders in a warehouse using the warehouseID and the orderDate less than 1 hour. **As we begin to model the data, we will start to use the indexes LSI and GSI to re-group, re-sort, re-aggregate the data to support our secondary access patterns but using the same type of key structures.**



GSI updates are eventually consistent, LSI updates are strongly consistent.

How Do GSI Updates Work?



If GSIs don't have enough write capacity, table writes will be throttled!

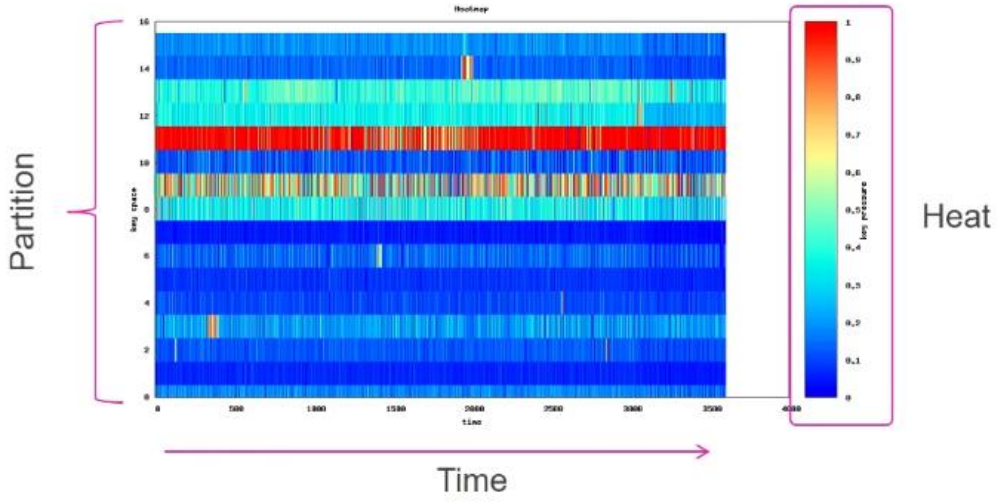
aws

Scaling NoSQL

"We are stuck with technology when what we really want is just stuff that works."

- Douglas Adams

What bad NoSQL looks like …



All of our access here is hitting a single storage node within the sharded keyspace. This is an anti-pattern in NoSQL databases. We should be distributing the access patterns to spread out to the data storage nodes or partitions.

Getting the most out of Amazon DynamoDB throughput

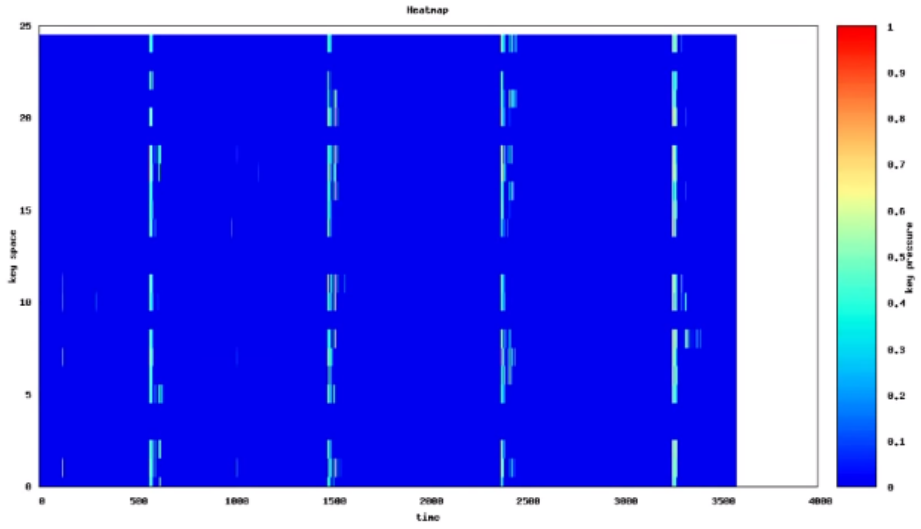
"To get the most out of DynamoDB throughput, create tables where the partition key element has a large number of distinct values, and values are requested fairly uniformly, as randomly as possible."

—DynamoDB Developer Guide

Space: access is evenly spread over the key-space

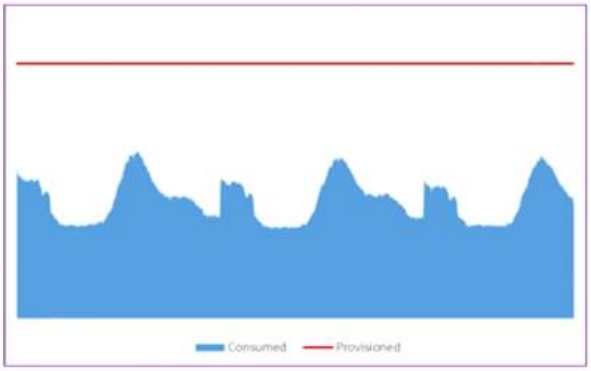
Time: requests arrive evenly spaced in time

Much better picture …

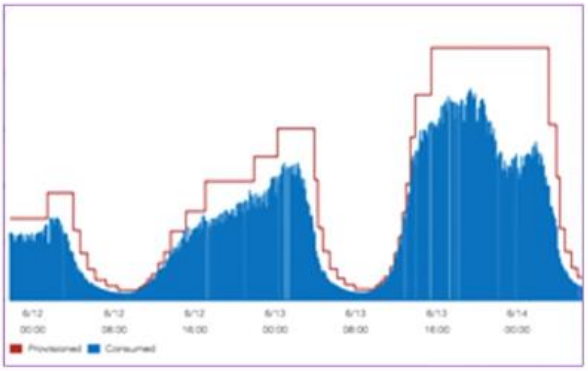


Auto Scaling

Throughput automatically adapts to your actual traffic



Without Auto Scaling



With Auto Scaling

NoSQL Data Modeling

"A ship in port is safe, but that's not what ships were built for."

- Grace Hopper

NoSQL data modelling design is to maximize the efficiency of your access patterns

It's all about relationships…

Social network

Document management

Process control

IT monitoring

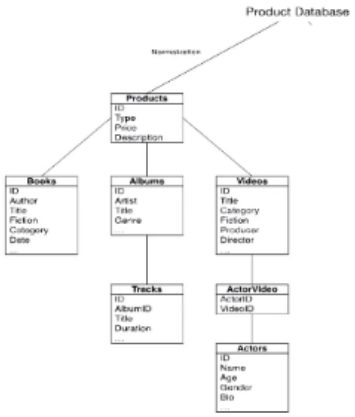
Data trees

SQL vs. NoSQL design pattern

SELECT * FROM Products
INNER JOIN Books

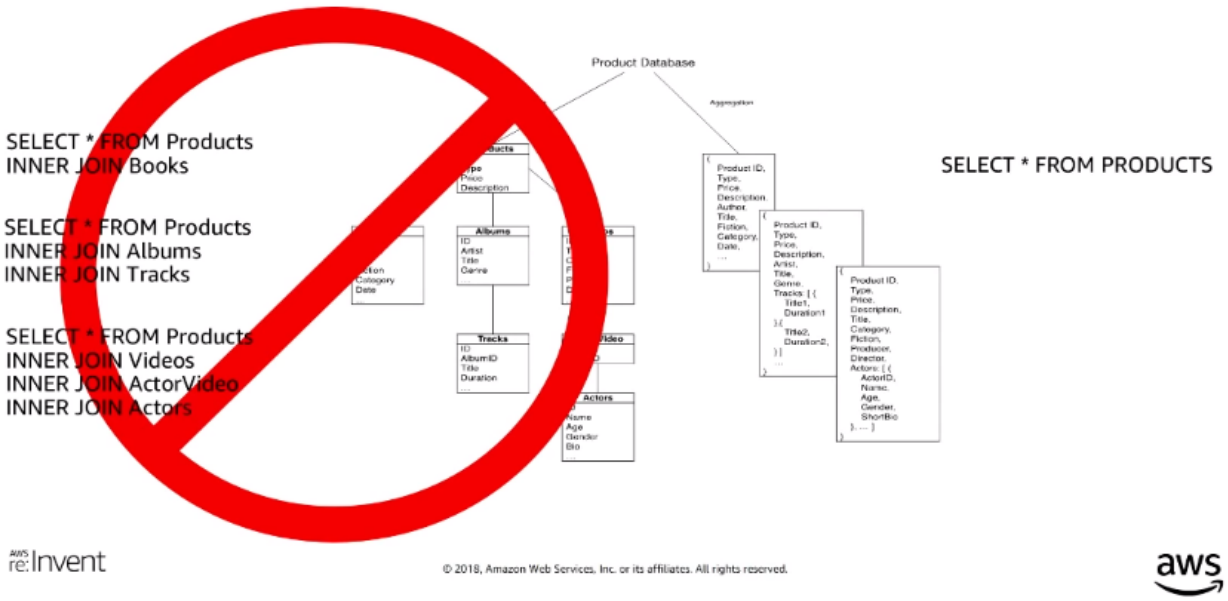
SELECT * FROM Products
INNER JOIN Albums
INNER JOIN Tracks

SELECT * FROM Products
INNER JOIN Videos
INNER JOIN ActorVideo
INNER JOIN Actors



We used normalized modelling approach in the past with relational databases using complex queries and joins. The data ends up living in different tables and we need ACID transactions to make updates later

SQL vs. NoSQL design pattern



But we can use NoSQL databases with hierarchical data models and create simpler queries to get back the data.

AMAZON DYNAMODB - KEY CONCEPTS

Selecting a Partition Key

- Large number of distinct values
- Items are uniformly requested and randomly distributed

Examples:

- Bad: Status, Gender
- Good: CustomerId, DeviceId

Selecting a Sort Key

- Model 1:n and n:n relationships
- Efficient/selective patterns
  - Query multiple entities
- Leverage range queries

Examples:

- Orders and OrderItems
- Hierarchical relationships

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

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 / DSS
- Define the Entity-Relationship Model
- Identify Data Life Cycle
  - TTL, Backup/Archival, etc.

TENETS OF NoSQL DATA MODELING

- Understand the use case
- Define the access patterns
  - Read/Write workloads
- Data-modeling
  - Avoid relational design patterns, use one table
- Review -> Repeat -> Review

- Identify data sources
- Define query aggregations
- Document all workflows

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

TENETS OF NoSQL DATA MODELING

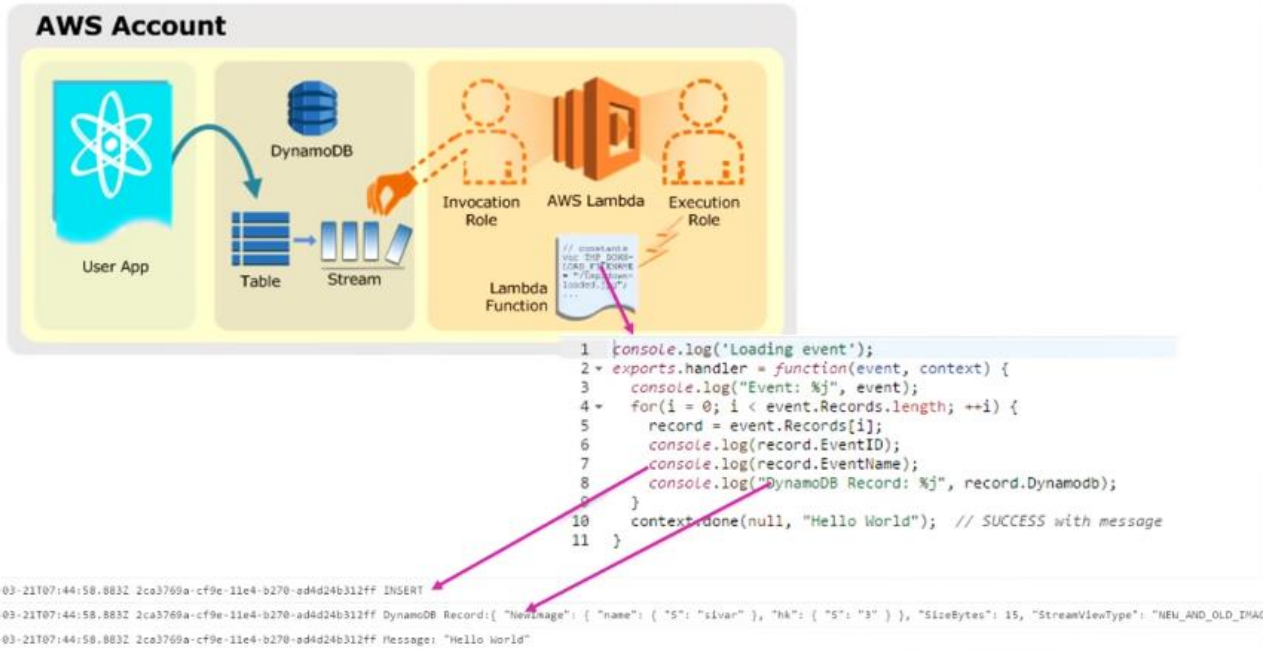
- Understand the use case
- Identify the access patterns
  - Read/Write workloads
  - Query dimensions and aggregations
- Data-modeling
  - Avoid relational design patterns, use one table
- Review -> Repeat-> Review

Complex Queries

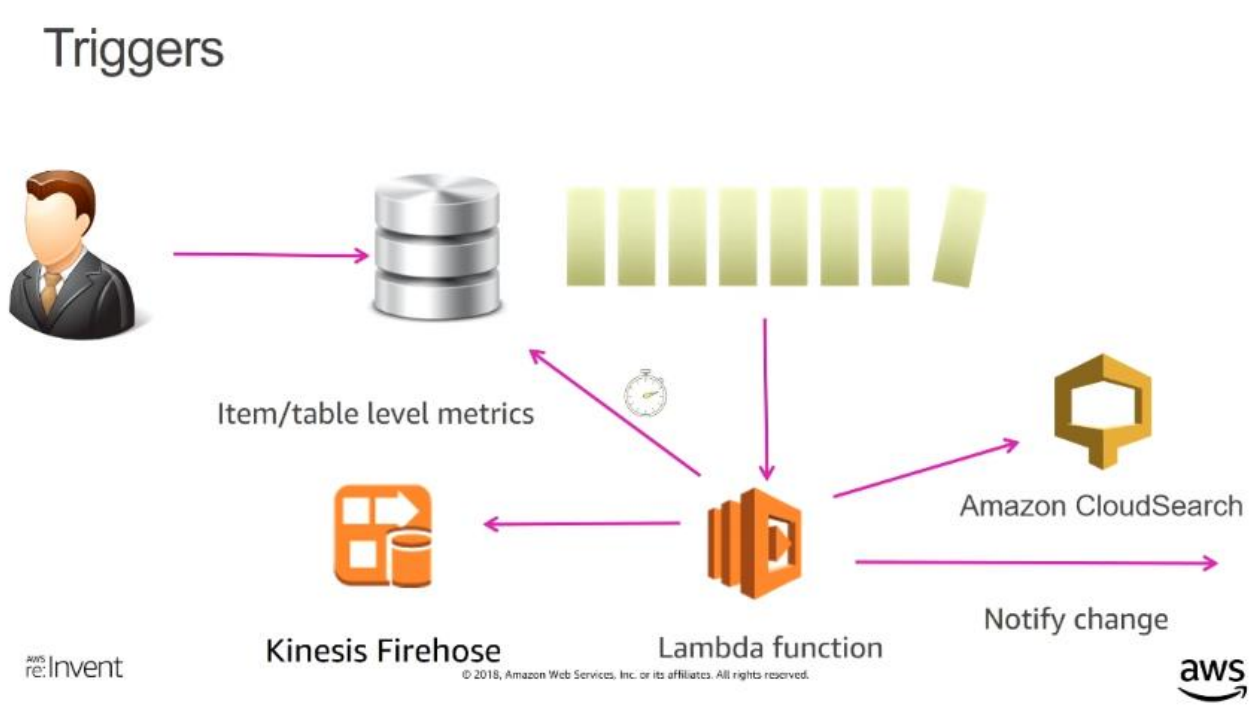
"Computers are useless. They can only give you answers."

- Pablo Picasso

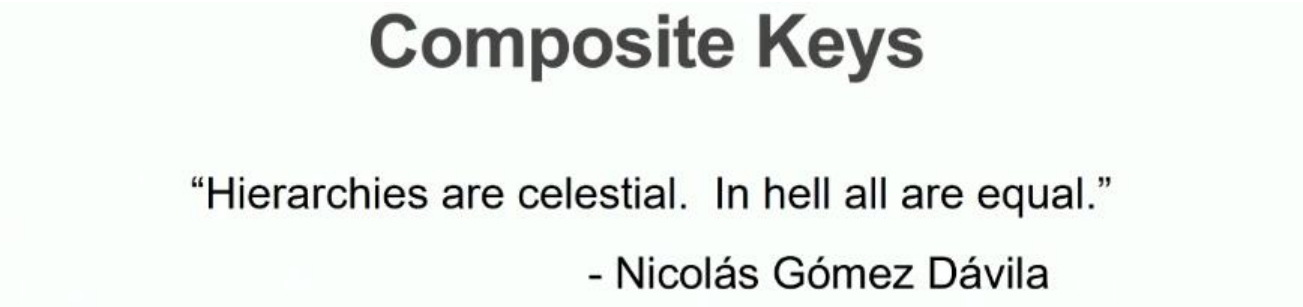
DynamoDB Streams and AWS Lambda

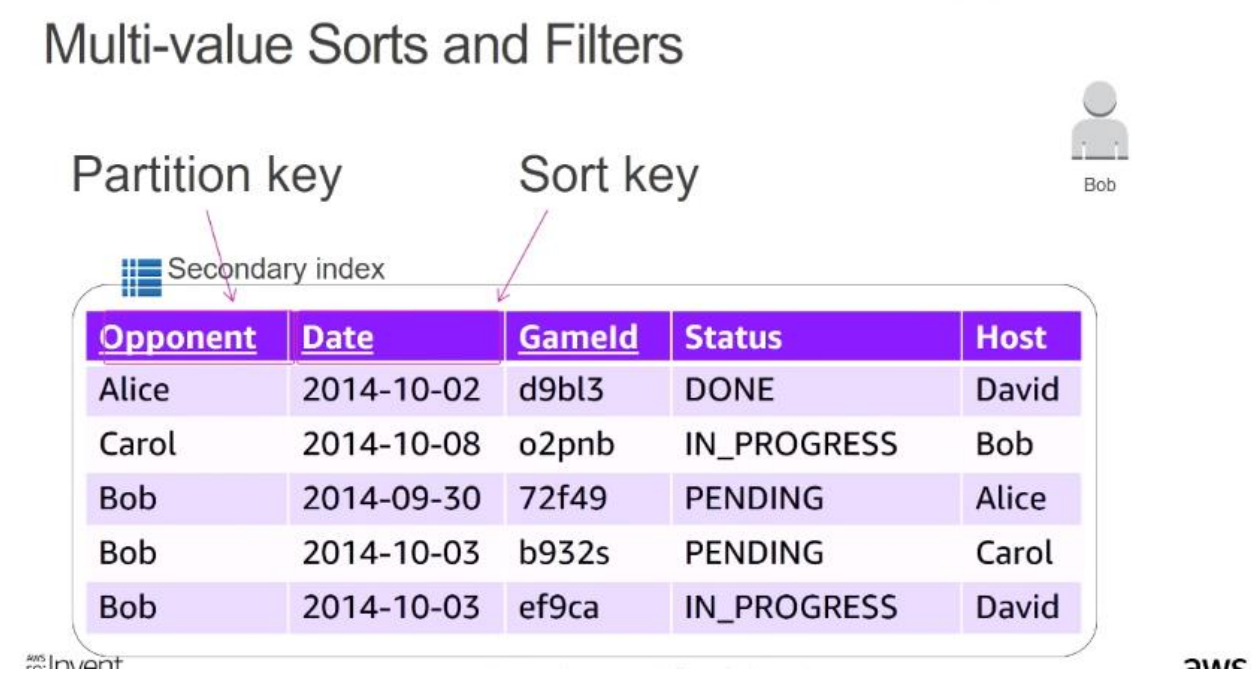


Using DynamoDB Streams (the change log of the DynamoDB tables and we can kick off Lambdas on the events we want as they come through the stream) and Lambda is like the best stored procedure engines available because they decouple your implementation from the databases



Using this pattern for stored procedure type problems, we can do computed running aggregations like counts and sums in DynamoDB Streams with Lambda, then write the metric like count back to the table as a metadata item.





We can store hierarchical data using the primary and secondary indexes for our range queries

Approach 1: Query Filter

This is not good because you still get to pay for the full query that happens before the filtering

Approach 2: Composite Key

Status
DONE
IN_PROGRESS
IN_PROGRESS
PENDING
PENDING

+

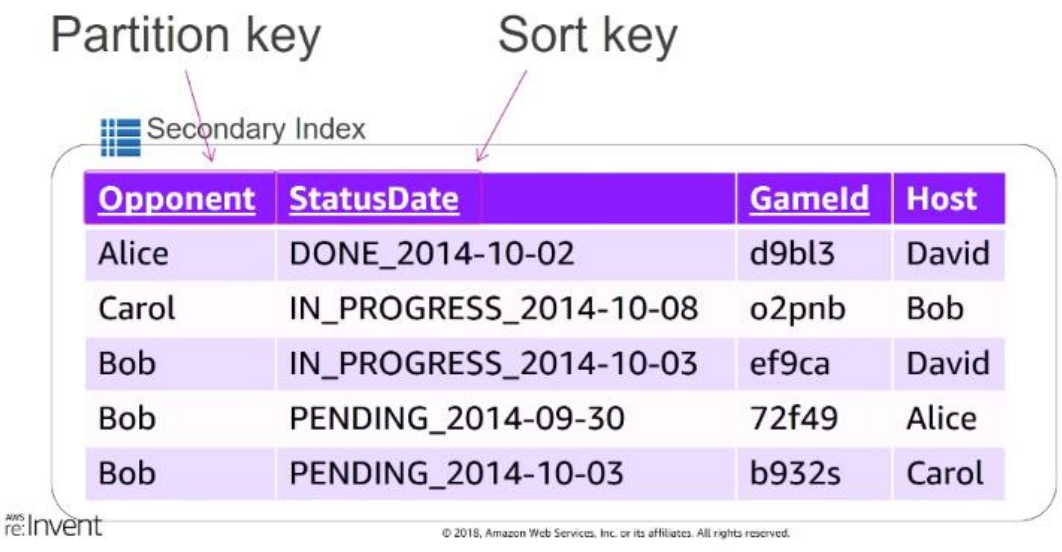
Date
2014-10-02
2014-10-08
2014-10-03
2014-10-03
2014-09-30

=

StatusDate
DONE_2014-10-02
IN_PROGRESS_2014-10-08
IN_PROGRESS_2014-10-03
PENDING_2014-09-30
PENDING_2014-10-03

We can use this approach of creating and using a Composite key.

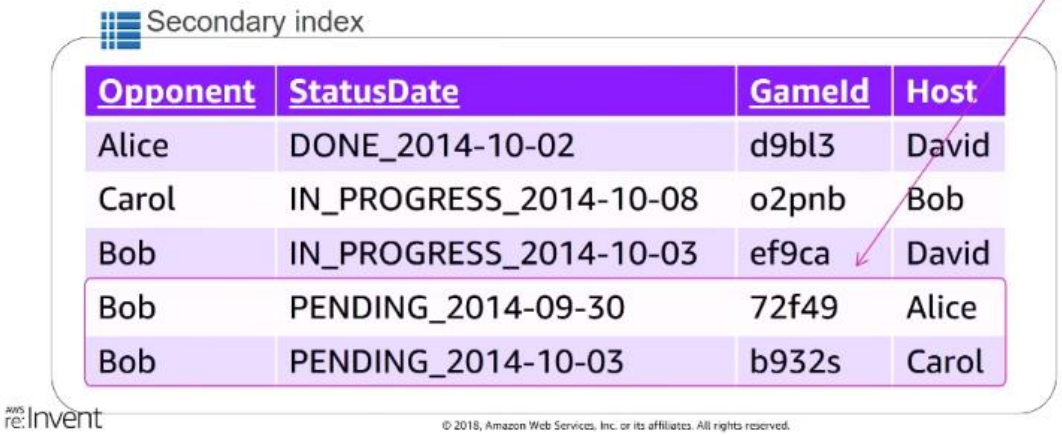
Approach 2: Composite Key



Opponent	StatusDate	GameId	Host
Alice	DONE_2014-10-02	d9bl3	David
Carol	IN_PROGRESS_2014-10-08	o2pnb	Bob
Bob	IN_PROGRESS_2014-10-03	ef9ca	David
Bob	PENDING_2014-09-30	72f49	Alice
Bob	PENDING_2014-10-03	b932s	Carol

Approach 2: Composite Key

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



Opponent	StatusDate	GameId	Host
Alice	DONE_2014-10-02	d9bl3	David
Carol	IN_PROGRESS_2014-10-08	o2pnb	Bob
Bob	IN_PROGRESS_2014-10-03	ef9ca	David
Bob	PENDING_2014-09-30	72f49	Alice
Bob	PENDING_2014-10-03	b932s	Carol

This gives us a selective read querying that will cost less.

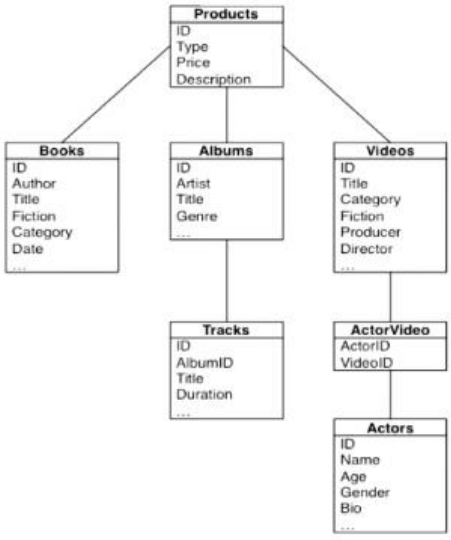
Advanced Data Modeling

"The great myth of our times is that technology is communication."

- Libby Larsen

How OLTP Apps Use Data

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



Maintaining Version History

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
```

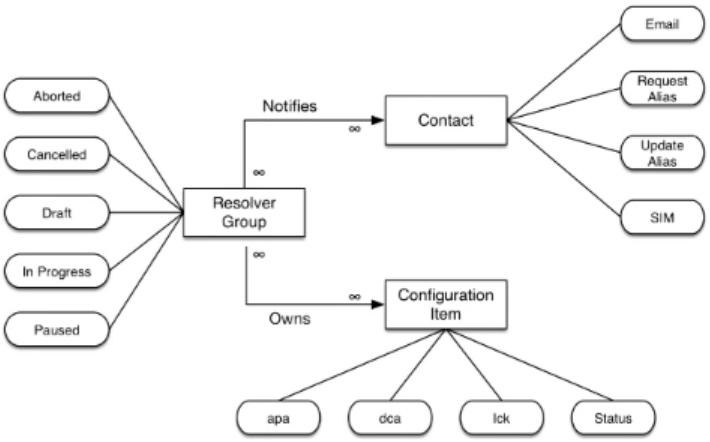
ItemID (PK)	Version (SK)	CurVer	Attrs
1	v0	2	…
	v1	…	
	v2	…	
	v3	…	

(Many more item partitions)

Overwrite v0 Item to Commit changes

Item versions

Managing Relational Transactions



- Configuration Management Service
  - Resolver Groups
  - Contacts
  - Configuration Items
- Transactional Workflows
  - Add Config Items to Resolver Groups
  - Update Config Item status
  - Add Contacts to Resolver Groups

DynamoDB Transactions API

- TransactWriteItems
  - Synchronous update, put, delete, and check
    - Atomic
    - Automated Rollbacks
  - 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

DynamoDB Table Schema

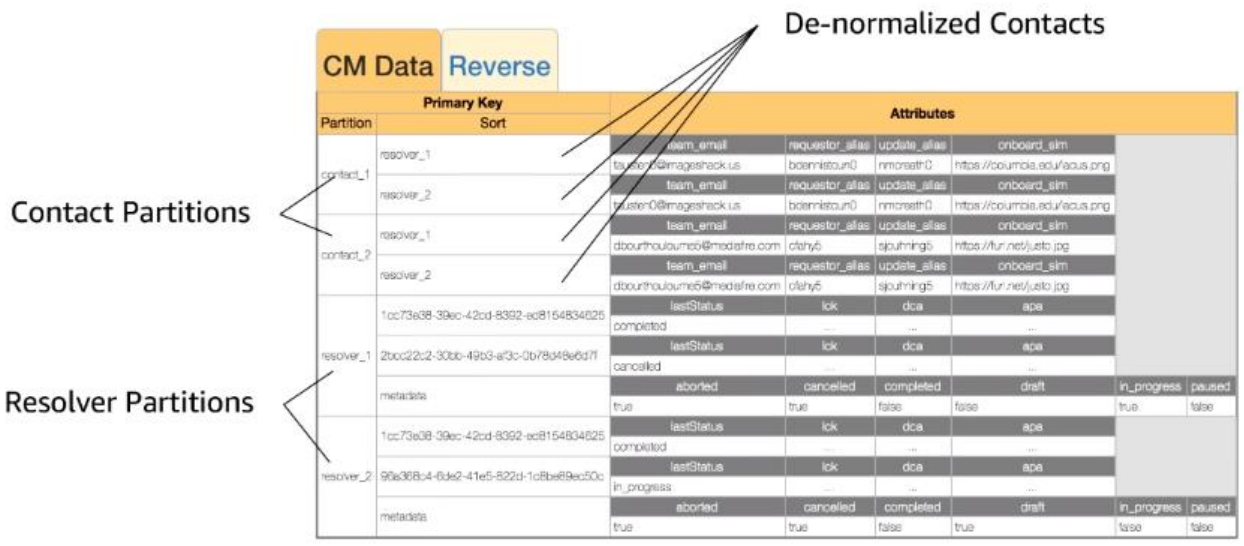
CM Data | Reverse

Primary Key		Attributes					
Partition	Sort						
contact_1	resolver_1	team_email	requestor_alias	update_alias	onboard_slm		
		tauster0@mageshack.us	bdennistoun0	nmoreath0	https://columbia.edu/acus.png		
	resolver_2	team_email	requestor_alias	update_alias	onboard_slm		
		tauster0@mageshack.us	bdennistoun0	nmoreath0	https://columbia.edu/acus.png		
contact_2	resolver_1	team_email	requestor_alias	update_alias	onboard_slm		
		dbourthoulourne6@medafre.com	cfehy6	sjouhning5	https://fur.net/justo.jpg		
	resolver_2	team_email	requestor_alias	update_alias	onboard_slm		
		dbourthoulourne6@medafre.com	cfehy6	sjouhning5	https://fur.net/justo.jpg		
resolver_1	1cc73e38-39ec-42cd-8392-ed8154834625	lastStatus	lck	dca	apa		
		completed	...	...	...		
	2bcc22c2-30bb-49b3-a3c-0b78d48e6d7f	lastStatus	lck	dca	apa		
		cancelled	...	...	...		
	metadata	aborted	cancelled	completed	draft	in_progress	paused
		true	true	false	false	true	false
resolver_2	1cc73e38-39ec-42cd-8392-ed8154834625	lastStatus	lck	dca	apa		
		completed	...	...	...		
	96a368c4-6de2-41e5-822d-1c8be89ec50c	lastStatus	lck	dca	apa		
		in_progress	...	...	...		
	metadata	aborted	cancelled	completed	draft	in_progress	paused
		true	true	false	true	false	false

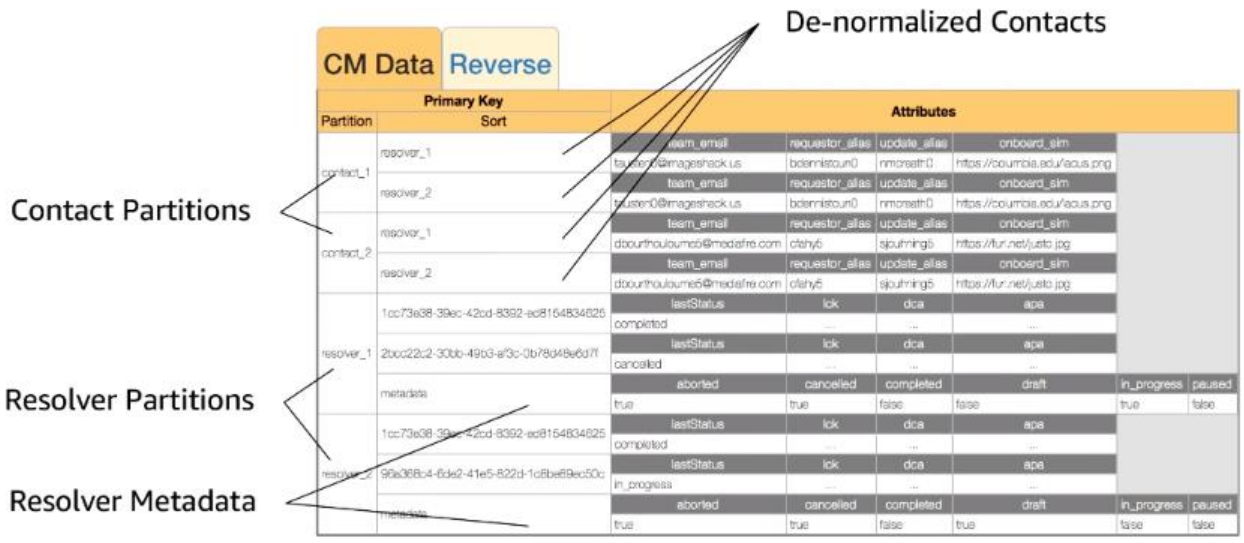
Contact Partitions

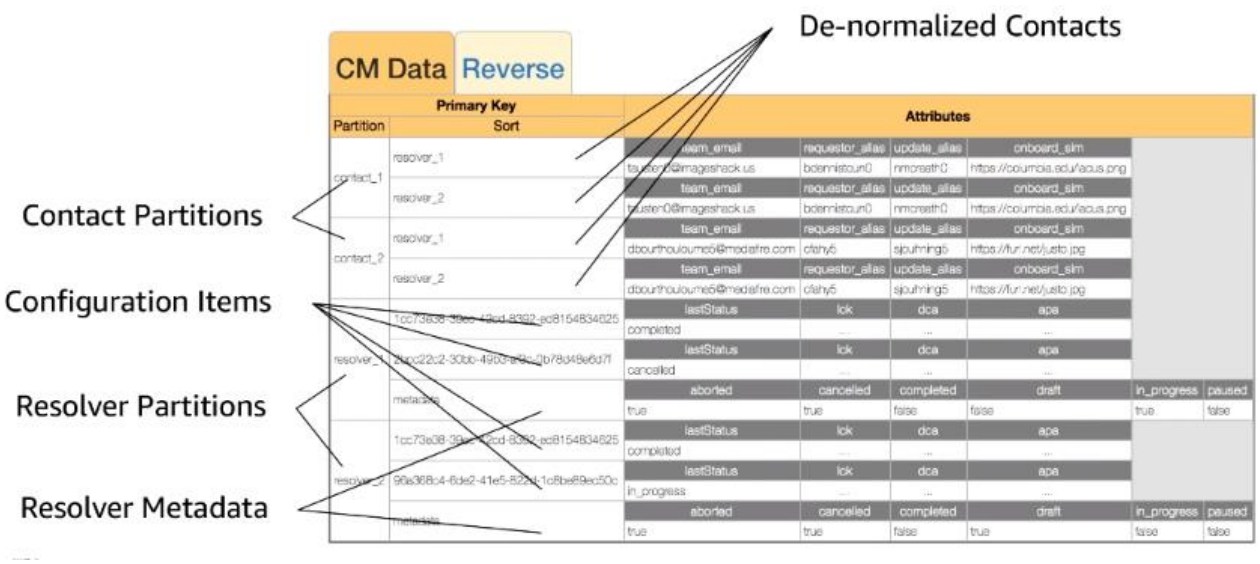
Resolver Partitions

DynamoDB Table Schema

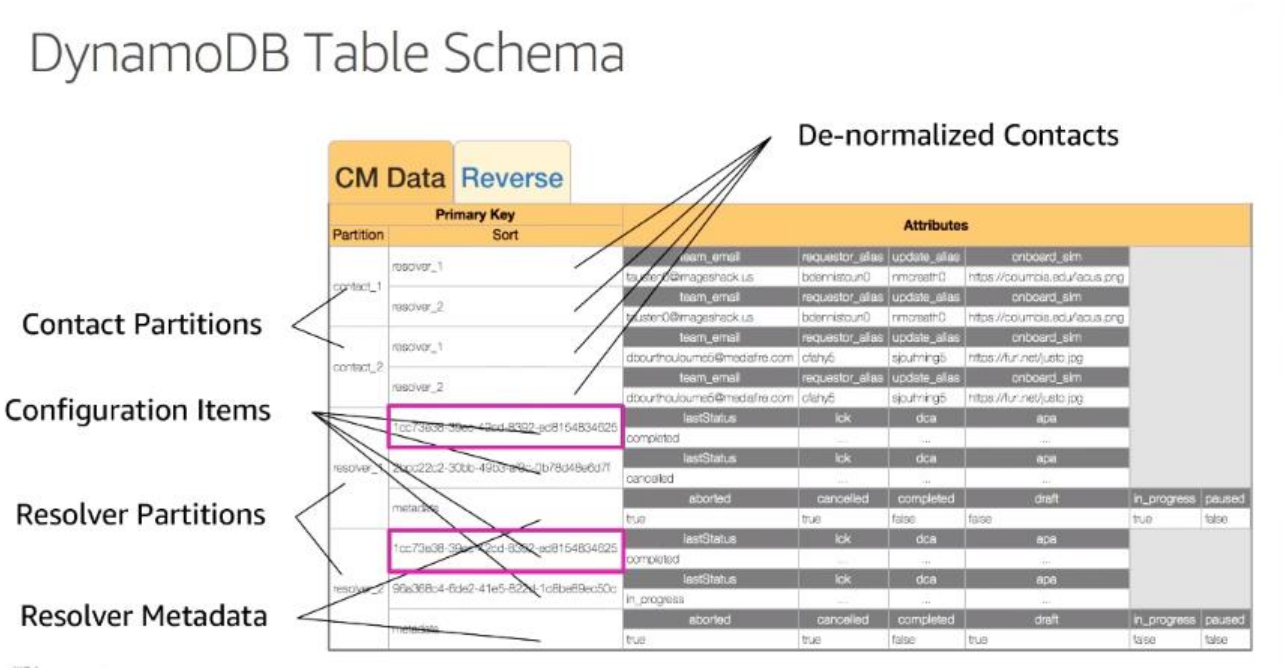


DynamoDB Table Schema

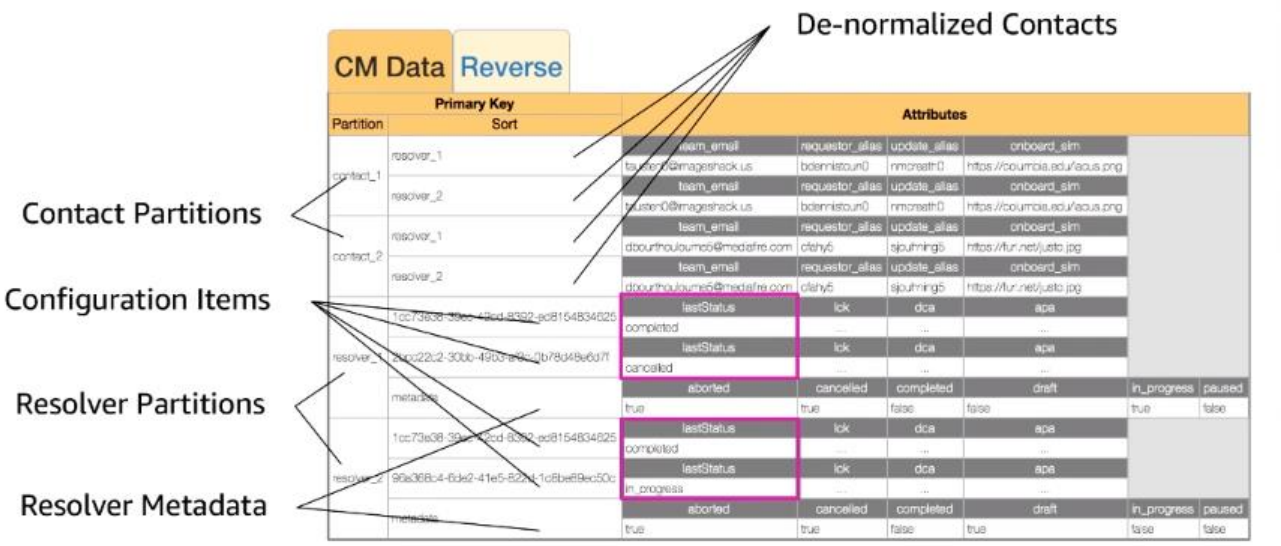




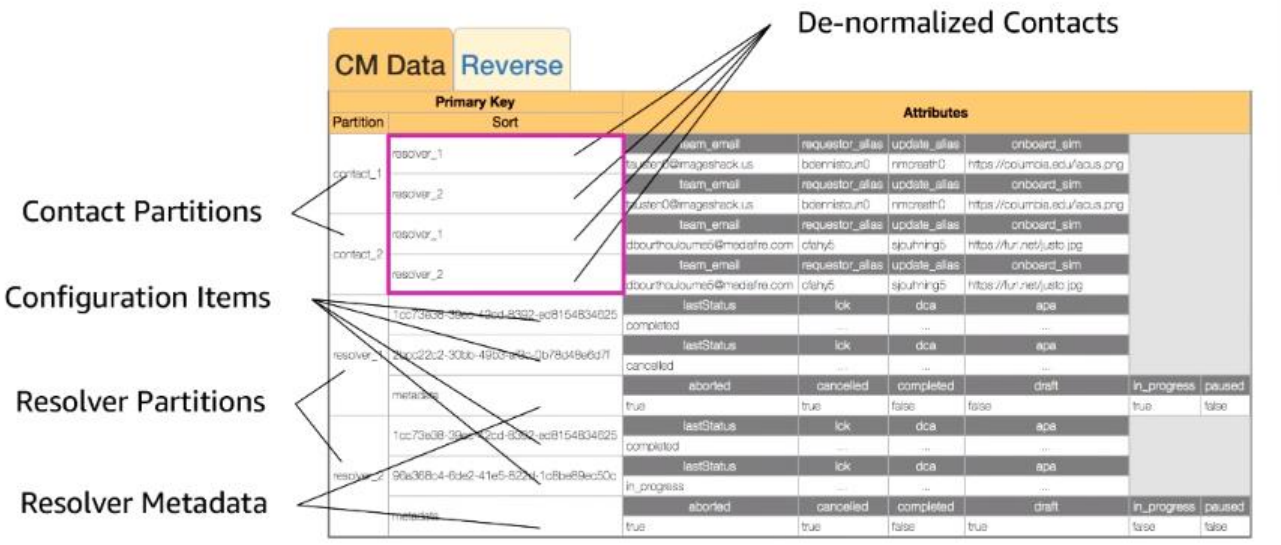
We can also add configuration items into those resolver groups when needed.



For transaction updates, we can execute inserts to both of those partitions and guarantee if they will succeed or not.

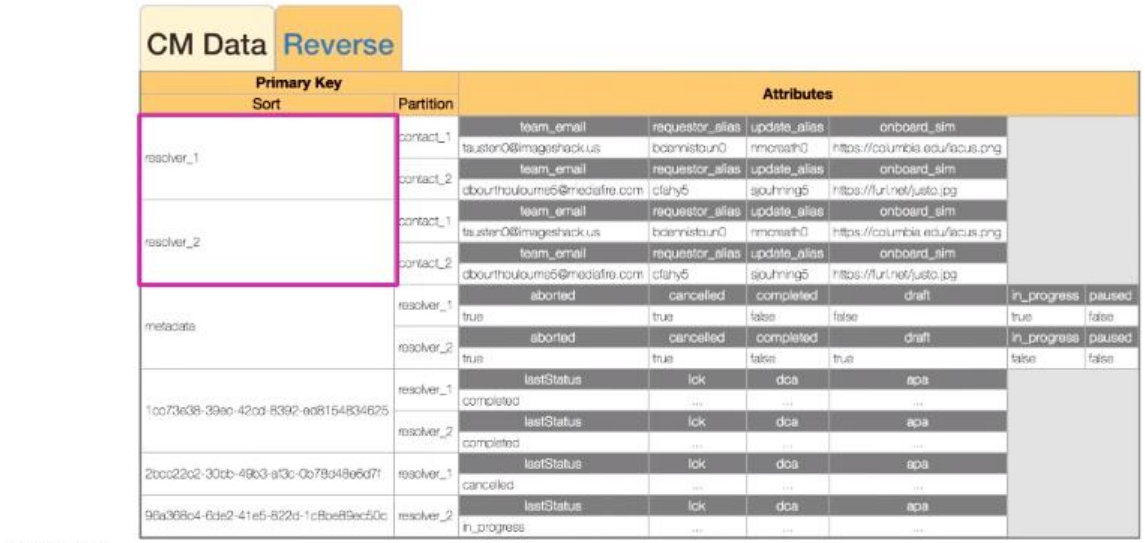


We can also update the transaction status for those transaction updates.



We can also update contact emails across a transaction group.

Reverse Lookup GSI



We can add one GSI to do a reverse lookup by reading the partition

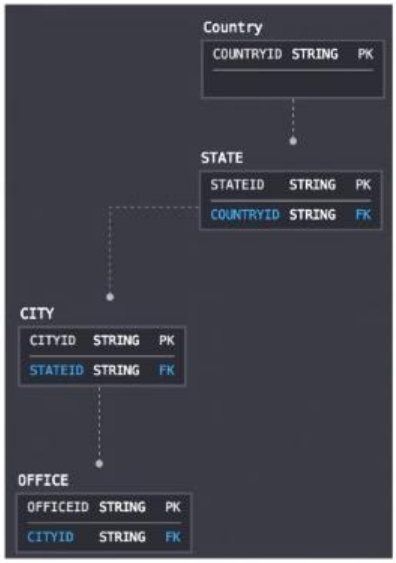
Reverse Lookup GSI

Or we can also do a lookup using the sort key

Hierarchical Data

Joins? We don't need no stinkin' joins!

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

aws

This model allows us to search for an office location. This allows us to use a single table to support multiple access patterns using sort keys and composite keys.

Modeling Relational Data

Dude, where's my Lookup Table?

Modeling a Delivery Service – GetMeThat!

The Business Model:

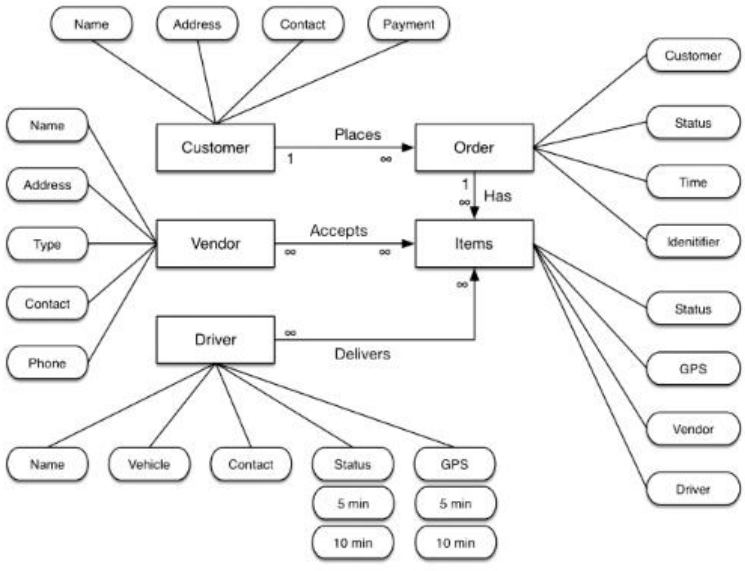
Customers want things, we get them things.

How it works:

People are busy, they need stuff. GetMeThat! lets users create lists of stuff they need and have it dropped where and when they need it.

- Download GetMeThat!
- Browse Stuff
- Get stuff
- Tell us where to put your stuff

The Entity Model



The Access Patterns

- Get Order(s)
  - by date range
    - by Customer
    - by Vendor
- Get Order Details
  - Status
  - Items
- Get Deliveries
  - by date range
    - by Driver

- Get available drivers
- by Geo and Time

- Get Customer Data
- Get Vendor Data

There are about 10 – 12 different access patterns that needs to be supported for this application

The Relational Approach

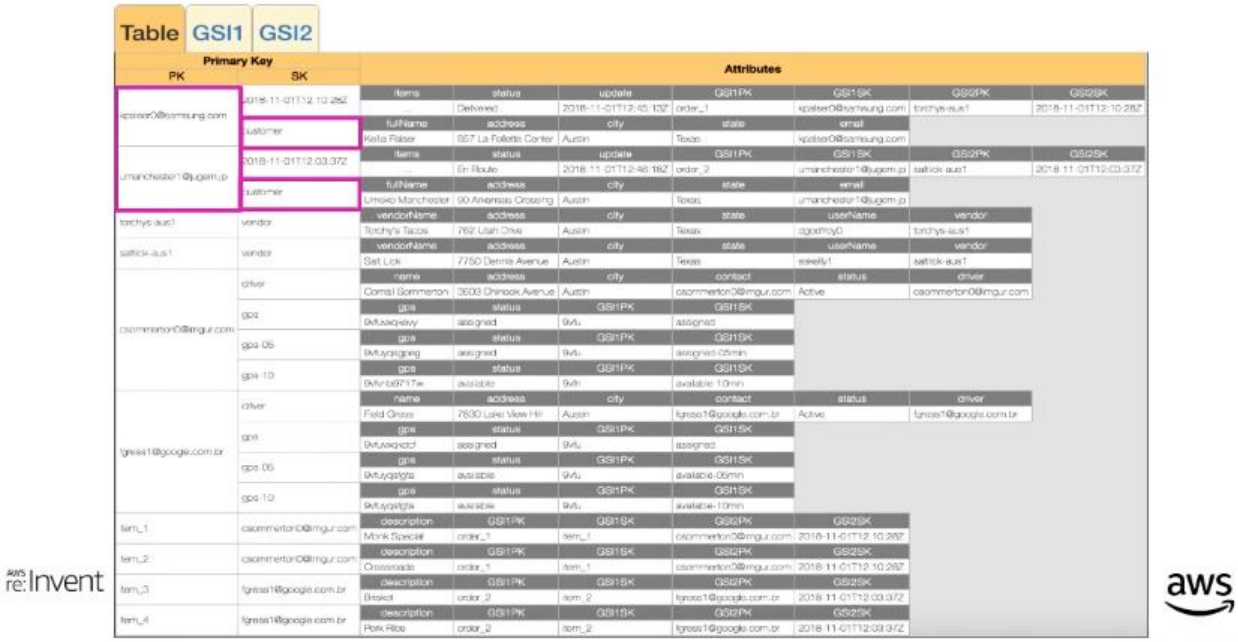
The NoSQL Approach

Table | GSI1 | GSI2

Primary Key		Attributes						
PK	SK	items	status	update	GSI1PK	GSI1SK	GSI2PK	GSI2SK
kpalser0@samsung.com	2018-11-01T12:10:28Z		Delivered	2018-11-01T12:45:13Z	order_1	kpalser0@samsung.com	torchys-aus1	2018-11-01T12:10:28Z
	customer	fullName	address	city	state	email		
		Kelly Palser	927 La Follette Center	Austin	Texas	kpalser0@samsung.com		
umanchester1@ujoom.jp	2018-11-01T12:03:37Z	items	status	update	GSI1PK	GSI1SK	GSI2PK	GSI2SK
			En Route	2018-11-01T12:46:18Z	order_2	umanchester1@ujoom.jp	saltick-aus1	2018-11-01T12:03:37Z
	customer	fullName	address	city	state	email		
		Linnea Manchester	90 Arkansas Crossing	Austin	Texas	umanchester1@ujoom.jp		
torchys-aus1	vendor	vendorName	address	city	state	userName	vendor	
		Torchy's Tacos	762 Utah Drive	Austin	Texas	dgodfroy0	torchys-aus1	
saltick-aus1	vendor	vendorName	address	city	state	userName	vendor	
		Salt Lick	7750 Dennis Avenue	Austin	Texas	sreilly1	saltick-aus1	
csommerton0@mgur.com	driver	name	address	city	contact	status	driver	
		Cornal Sommerton	3500 Oninook Avenue	Austin	csommerton0@mgur.com	Active	csommerton0@mgur.com	
	gps	gps	status	GSI1PK	GSI1SK			
		9vfuxeqvkvy	assigned	9vfu	assigned			
	gps-05	gps	status	GSI1PK	GSI1SK			
		9vfuyqigqng	assigned	9vfu	assigned-05min			
	gps-10	gps	status	GSI1PK	GSI1SK			
		9vfvnb9717w	available	9vfv	available-10min			
fgrass1@google.com.br	driver	name	address	city	contact	status	driver	
		Field Grass	7830 Lake View Hill	Austin	fgrass1@google.com.br	Active	fgrass1@google.com.br	
	gps	gps	status	GSI1PK	GSI1SK			
		9vfuxeqrzcf	assigned	9vfu	assigned			
	gps-05	gps	status	GSI1PK	GSI1SK			
		9vfuyqstgta	available	9vfu	available-05min			
	gps-10	gps	status	GSI1PK	GSI1SK			
		9vfuyqwtgta	available	9vfu	available-10min			
item_1	csommerton0@mgur.com	description	GSI1PK	GSI1SK	GSI2PK	GSI2SK		
		Monk Special	order_1	item_1	csommerton0@mgur.com	2018-11-01T12:10:28Z		
item_2	csommerton0@mgur.com	description	GSI1PK	GSI1SK	GSI2PK	GSI2SK		
		Crossroads	order_1	item_1	csommerton0@mgur.com	2018-11-01T12:10:28Z		
item_3	fgrass1@google.com.br	description	GSI1PK	GSI1SK	GSI2PK	GSI2SK		
		Brisket	order_2	item_2	fgrass1@google.com.br	2018-11-01T12:03:37Z		
item_4	fgrass1@google.com.br	description	GSI1PK	GSI1SK	GSI2PK	GSI2SK		
		Pork Ribs	order_2	item_2	fgrass1@google.com.br	2018-11-01T12:03:37Z		

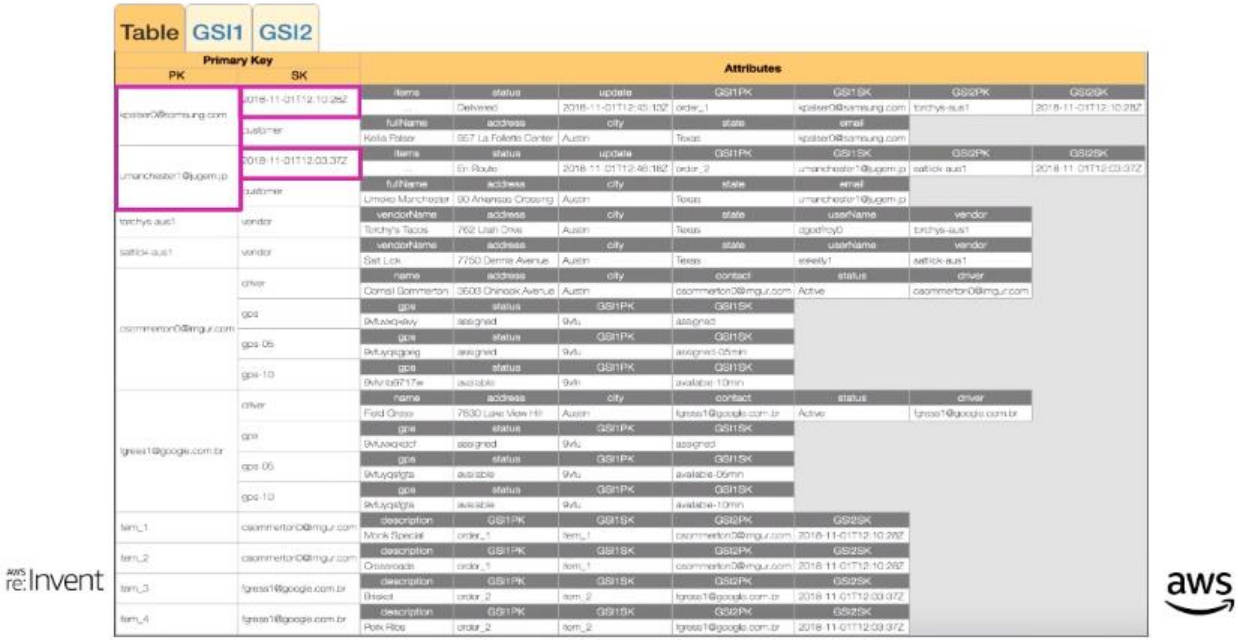
We will design and use a single DynamoDB table that will satisfy all those different access patterns using a combination of a primary key, sort keys and composite keys.

The NoSQL Approach



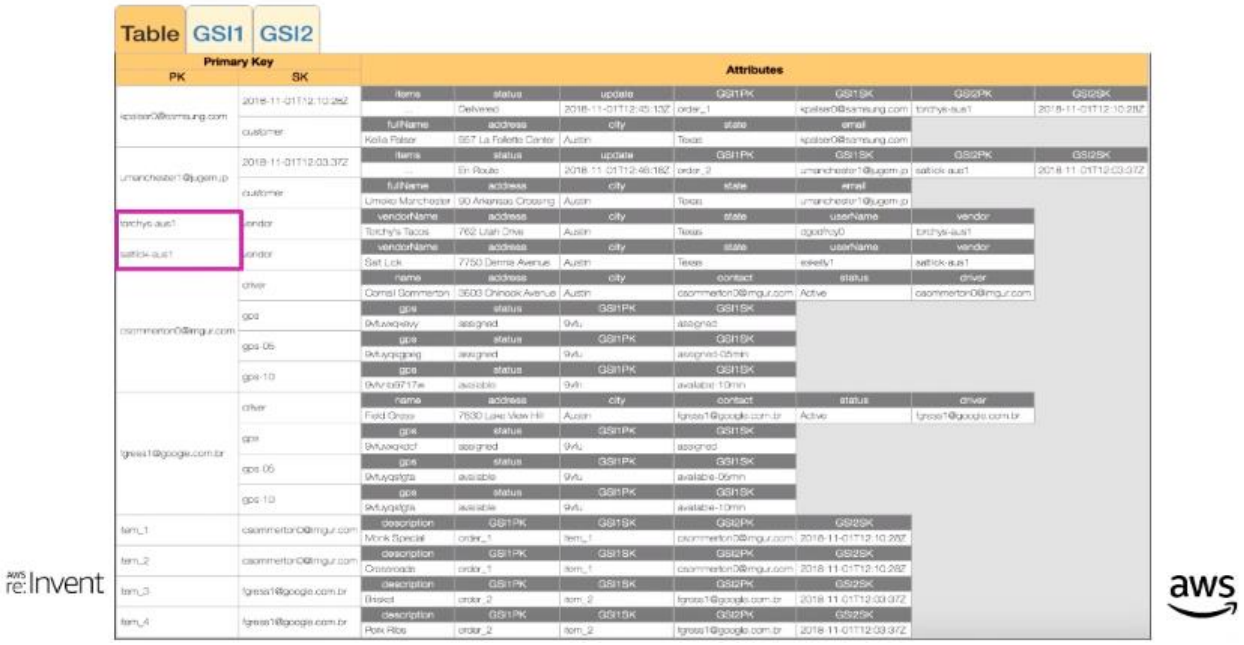
We can get a **customer information** by simply querying the table by the customer's email address as the partition key and the sort key will be the **customer** value for that email.

The NoSQL Approach



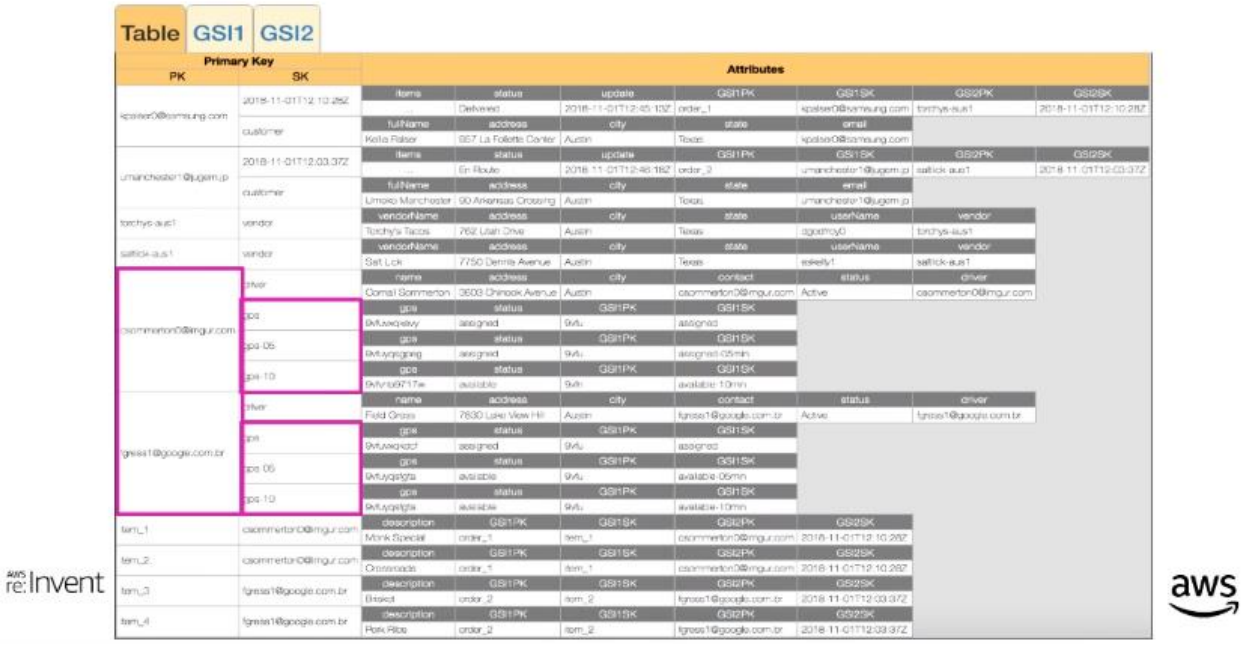
We can also use the **orderdate** value as a sort key to **get orders for that email address** in a particular data or date range.

The NoSQL Approach



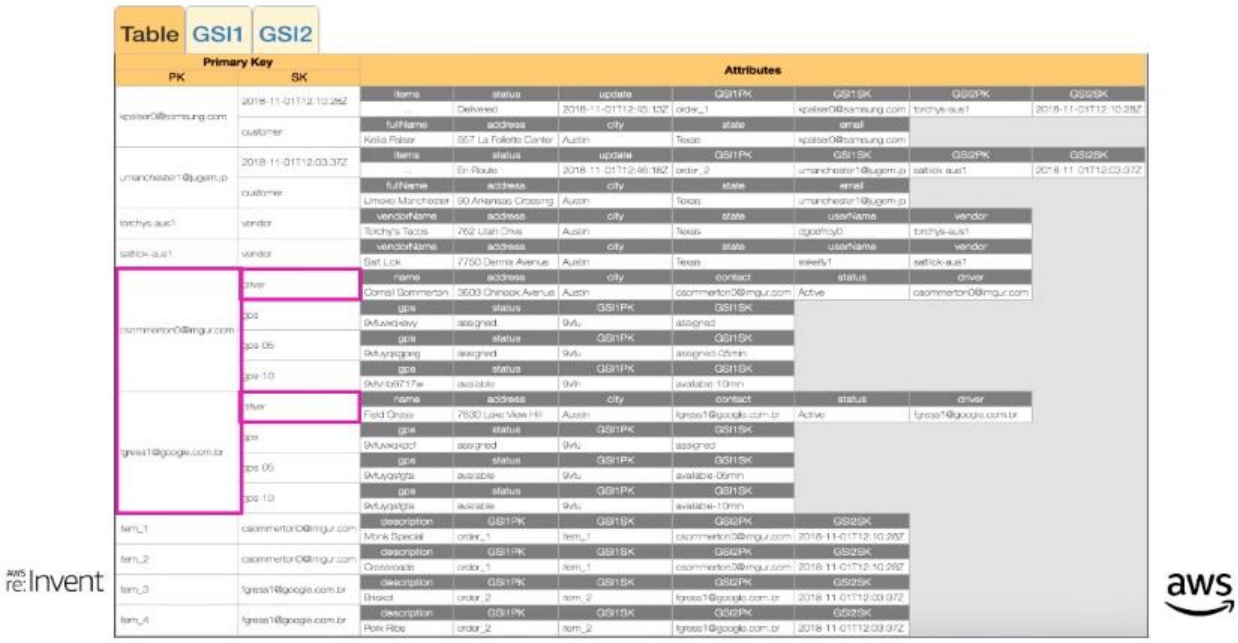
We can also get vendor data using the vendorID as the primary key as above

The NoSQL Approach

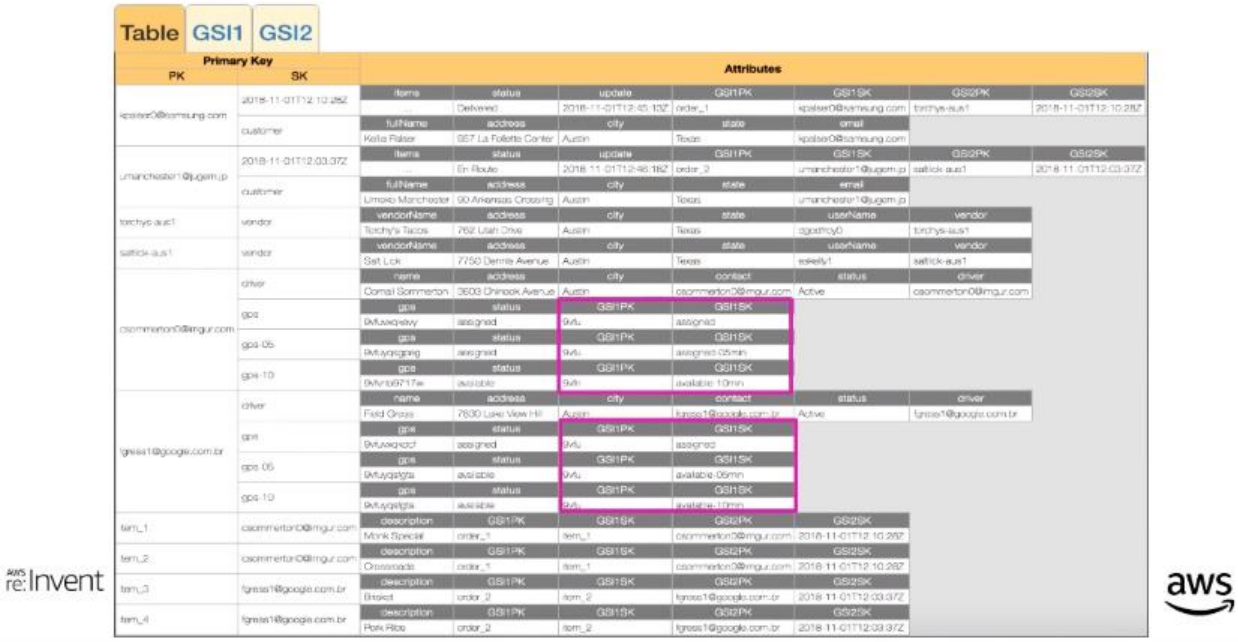


We can **get driver data** by selecting using the **driver's email as the primary key** and the **driver** value as the sort key.

The NoSQL Approach



The NoSQL Approach

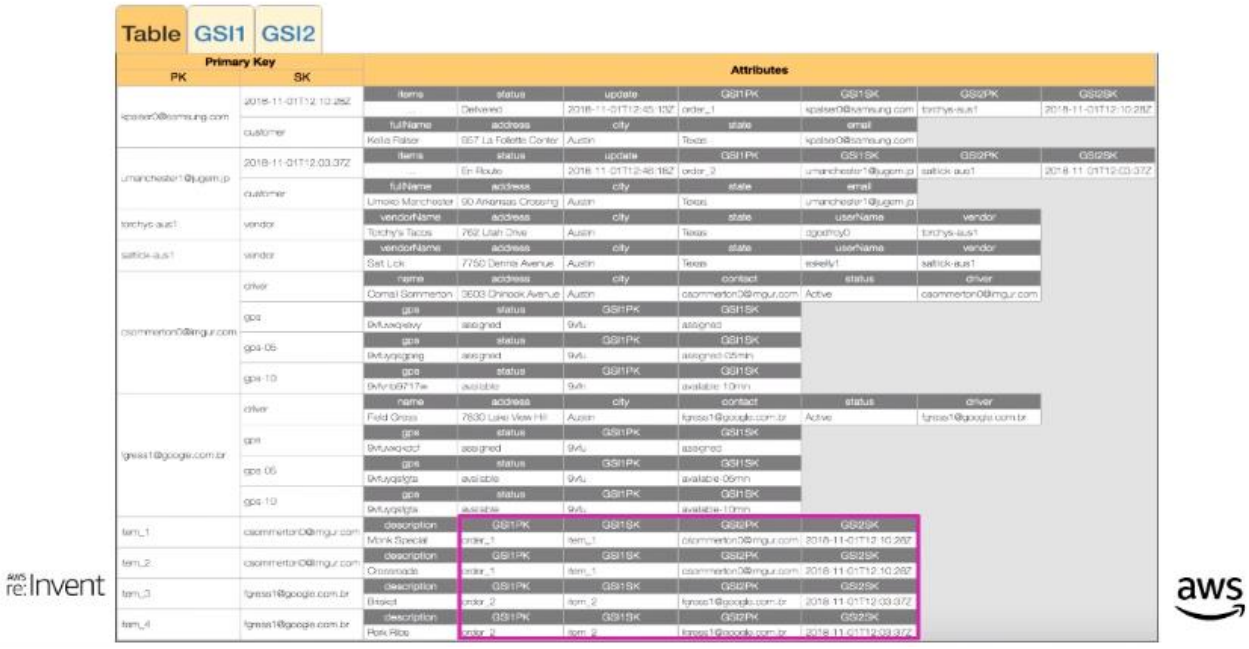


Indexing is about overloading the key attributes like order_1

The NoSQL Approach

We can also index the drivers GPS coordinates and store the data for easy access later

The NoSQL Approach



We can index the ordered items across those GSI's too using timestamp by customers and vendors

The NoSQL Approach (Orders and Drivers GSI)

Table | GSI1 | GSI2

Primary Key		Attributes							
GSI1PK	GSI1SK								
order_1	item_1	PK	SK	description	GSI2PK	GSI2SK			
		item_1	csommerton0@imgur.com	Monk Special	csommerton0@imgur.com	2018-11-01T12:10:28Z			
	item_1	PK	SK	description	GSI2PK	GSI2SK			
		item_2	csommerton0@imgur.com	Crossroads	csommerton0@imgur.com	2018-11-01T12:10:28Z			
	kpaiser0@samsung.com	PK	SK	items	status	update	GSI2PK	GSI2SK	
		kpaiser0@samsung.com	2018-11-01T12:10:28Z	...	Delivered	2018-11-01T12:45:13Z	torchys-aus1	2018-11-01T12:10:28Z	
order_2	item_2	PK	SK	description	GSI2PK	GSI2SK			
		item_3	fgress1@google.com.br	Brisket	fgress1@google.com.br	2018-11-01T12:03:37Z			
	item_2	PK	SK	description	GSI2PK	GSI2SK			
		item_4	fgress1@google.com.br	Pork Ribs	fgress1@google.com.br	2018-11-01T12:03:37Z			
	umanchester1@jugem.jp	PK	SK	items	status	update	GSI2PK	GSI2SK	
		umanchester1@jugem.jp	2018-11-01T12:03:37Z	...	En Route	2018-11-01T12:46:18Z	satlick-aus1	2018-11-01T12:03:37Z	
9vfu	assigned	PK	SK	gps					
		csommerton0@imgur.com	gps	9vfuwcjkevy					
	assigned	PK	SK	gps	status				
		fgress1@google.com.br	gps	9vfuwcjkdcf	assigned				
	assigned 05min	PK	SK	gps	status				
		csommerton0@imgur.com	gps-05	9vfuyqsgpeg	assigned				
	available-05min	PK	SK	gps	status				
		fgress1@google.com.br	gps-05	9vfuyqsfgts	available				
	available-10min	PK	SK	gps	status				
		fgress1@google.com.br	gps-10	9vfuyqsfgts	available				
9vfn	available-10min	PK	SK	gps	status				
		csommerton0@imgur.com	gps-10	9vfvnb971rw	available				

We can now use the GSI and the orderId like **order_1** to get all order items from a specific customer

The NoSQL Approach (Orders and Drivers GSI)

Table | GSI1 | GSI2

Primary Key		Attributes							
GSI1PK	GSI1SK								
order_1	item_1	PK	SK	description	GSI2PK	GSI2SK			
		item_1	csommerton0@imgur.com	Monk Special	csommerton0@imgur.com	2018-11-01T12:10:28Z			
	item_1	PK	SK	description	GSI2PK	GSI2SK			
		item_2	csommerton0@imgur.com	Crossroads	csommerton0@imgur.com	2018-11-01T12:10:28Z			
	kpaiser0@samsung.com	PK	SK	items	status	update	GSI2PK	GSI2SK	
		kpaiser0@samsung.com	2018-11-01T12:10:28Z	...	Delivered	2018-11-01T12:45:13Z	torchys-aus1	2018-11-01T12:10:28Z	
order_2	item_2	PK	SK	description	GSI2PK	GSI2SK			
		item_3	fgress1@google.com.br	Brisket	fgress1@google.com.br	2018-11-01T12:03:37Z			
	item_2	PK	SK	description	GSI2PK	GSI2SK			
		item_4	fgress1@google.com.br	Pork Ribs	fgress1@google.com.br	2018-11-01T12:03:37Z			
	umanchester1@jugem.jp	PK	SK	items	status	update	GSI2PK	GSI2SK	
		umanchester1@jugem.jp	2018-11-01T12:03:37Z	...	En Route	2018-11-01T12:46:18Z	satlick-aus1	2018-11-01T12:03:37Z	
9vfu	assigned	PK	SK	gps					
		csommerton0@imgur.com	gps	9vfuwcjkevy					
	assigned	PK	SK	gps	status				
		fgress1@google.com.br	gps	9vfuwcjkdcf	assigned				
	assigned 05min	PK	SK	gps	status				
		csommerton0@imgur.com	gps-05	9vfuyqsgpeg	assigned				
	available-05min	PK	SK	gps	status				
		fgress1@google.com.br	gps-05	9vfuyqsfgts	available				
	available-10min	PK	SK	gps	status				
		fgress1@google.com.br	gps-10	9vfuyqsfgts	available				
9vfn	available-10min	PK	SK	gps	status				
		csommerton0@imgur.com	gps-10	9vfvnb971rw	available				

We can also query by sector to get driver details also

The NoSQL Approach (Vendor and Deliveries GSI)

Table | GSI1 | GSI2

Primary Key		Attributes						
GSI2PK	GSI2SK							
torchys-aus1	2018-11-01T12:10:28Z	PK	SK	Items	status	update	GSI1PK	GSI1SK
		kpaiser0@samsung.com	2018-11-01T12:10:28Z		Delivered	2018-11-01T12:45:13Z	order_1	kpaiser0@samsung.com
satlick-aus1	2018-11-01T12:03:37Z	PK	SK	Items	status	update	GSI1PK	GSI1SK
		umanchester1@jugem.jp	2018-11-01T12:03:37Z		En Route	2018-11-01T12:46:16Z	order_2	umanchester1@jugem.jp
csommerton0@imgur.com	2018-11-01T12:10:28Z	PK	SK	description	GSI1PK	GSI1SK		
		item_1	csommerton0@imgur.com	Monk Special	order_1	item_1		
	2018-11-01T12:10:28Z	PK	SK	description	GSI1PK	GSI1SK		
		item_2	csommerton0@imgur.com	Crossroads	order_1	item_1		
fgrass1@google.com.br	2018-11-01T12:03:37Z	PK	SK	description	GSI1PK	GSI1SK		
		item_3	fgrass1@google.com.br	Brisket	order_2	item_2		
	2018-11-01T12:03:37Z	PK	SK	description	GSI1PK	GSI1SK		
		item_4	fgrass1@google.com.br	Pork Ribs	order_2	item_2		

We can also query GSI2 by vendor and date

The NoSQL Approach (Vendor and Deliveries GSI)

Table | GSI1 | GSI2

Primary Key		Attributes						
GSI2PK	GSI2SK							
torchys-aus1	2018-11-01T12:10:28Z	PK	SK	Items	status	update	GSI1PK	GSI1SK
		kpaiser0@samsung.com	2018-11-01T12:10:28Z		Delivered	2018-11-01T12:45:13Z	order_1	kpaiser0@samsung.com
satlick-aus1	2018-11-01T12:03:37Z	PK	SK	Items	status	update	GSI1PK	GSI1SK
		umanchester1@jugem.jp	2018-11-01T12:03:37Z		En Route	2018-11-01T12:46:16Z	order_2	umanchester1@jugem.jp
csommerton0@imgur.com	2018-11-01T12:10:28Z	PK	SK	description	GSI1PK	GSI1SK		
		item_1	csommerton0@imgur.com	Monk Special	order_1	item_1		
	2018-11-01T12:10:28Z	PK	SK	description	GSI1PK	GSI1SK		
		item_2	csommerton0@imgur.com	Crossroads	order_1	item_1		
fgrass1@google.com.br	2018-11-01T12:03:37Z	PK	SK	description	GSI1PK	GSI1SK		
		item_3	fgrass1@google.com.br	Brisket	order_2	item_2		
	2018-11-01T12:03:37Z	PK	SK	description	GSI1PK	GSI1SK		
		item_4	fgrass1@google.com.br	Pork Ribs	order_2	item_2		

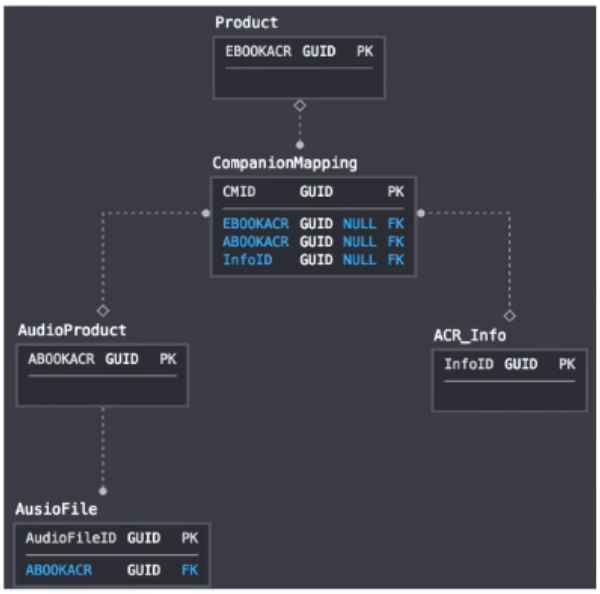
We can also query by email and date

A Real World Example

"Reality is that which, when you stop believing in it, does not go away."

- Philip K. Dick

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

Access Patterns

#	USE CASE	
1	CompanionMapping	getCcompanionMappingsByAsin
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

We have 20 access patterns and we have 5 GSIs to use in our single DynamoDB table implementation.

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

This is the proposed table structure we implemented. The current version of a book is always the v0 version. We have 2 partitions on the table, the ABOOK and the EBOOK partitions. We then insert an item into the ABOOK called ABOOKACR partition that associates the ABOOK to the EBOOK. We end up with 3 GSIs/indexes on this table that don't always contain the same data type because we have so many access data patterns.

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

	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

This is what ends up happening on the table, the GSI entries look meaning less because we have items sorted by arbitrary values.

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

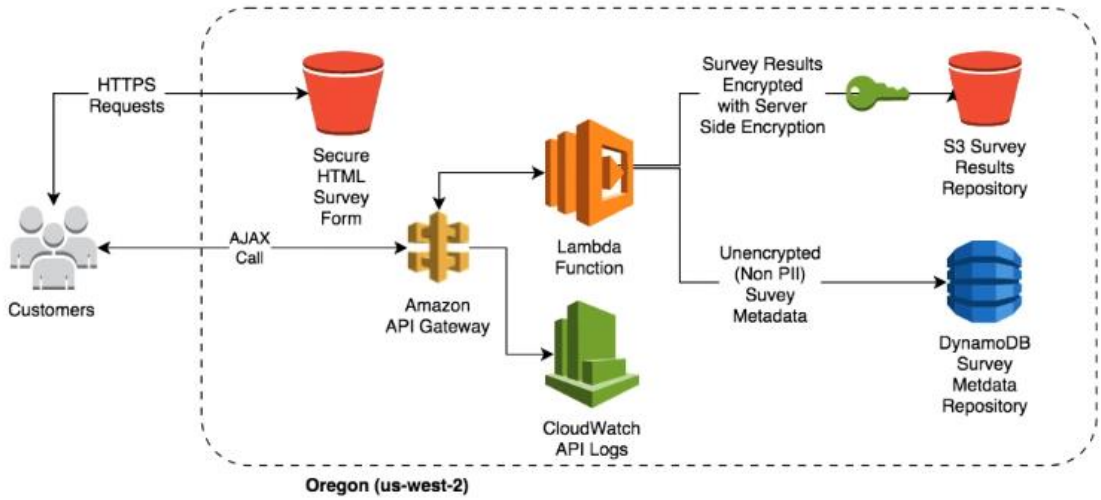
We ended up expanding the tables to show the needed sort key and filter conditions for the 20 different access patterns are satisfied using the proposed single table.

The Serverless Paradigm

"Fairly cheap home computing was what changed my life."

- Linus Torvalds

Elastic Serverless Applications



This is the rate my response interaction service for Amazon services feedback feature.

Conclusions

- NoSQL does not mean non-relational
- The ERD still matters
- RDBMS is not deprecated by NoSQL
- Use NoSQL for OLTP or DSS at scale.
- Use RDBMS for OLAP or OLTP when scale is not important.