AWS re:Invent

DAT403-R

Amazon DynamoDB deep dive: Advanced design patterns

Rick Houlihan
Principal Technologist, NoSQL
Amazon Web Services

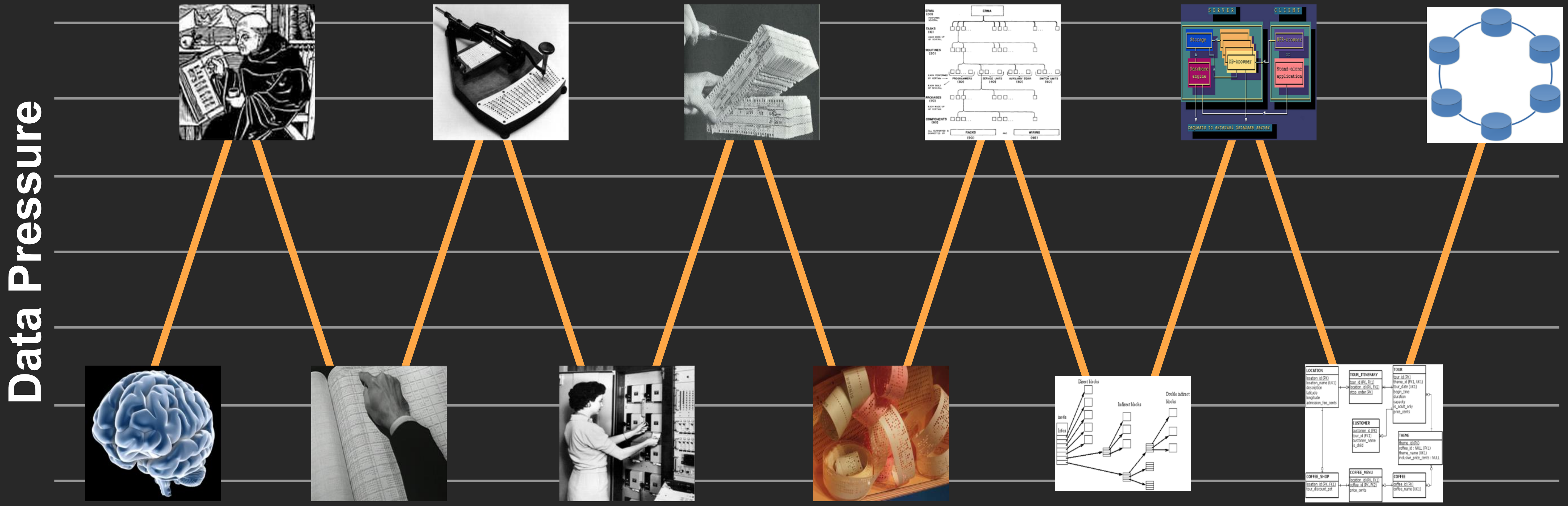
Agenda

- Brief history of data processing (Why NoSQL?)
- Overview of Amazon DynamoDB
- NoSQL data modeling
    - Normalized versus de-normalized schema
- Common NoSQL design patterns
    - Composite keys, hierarchical data, relational data
- Modeling real applications

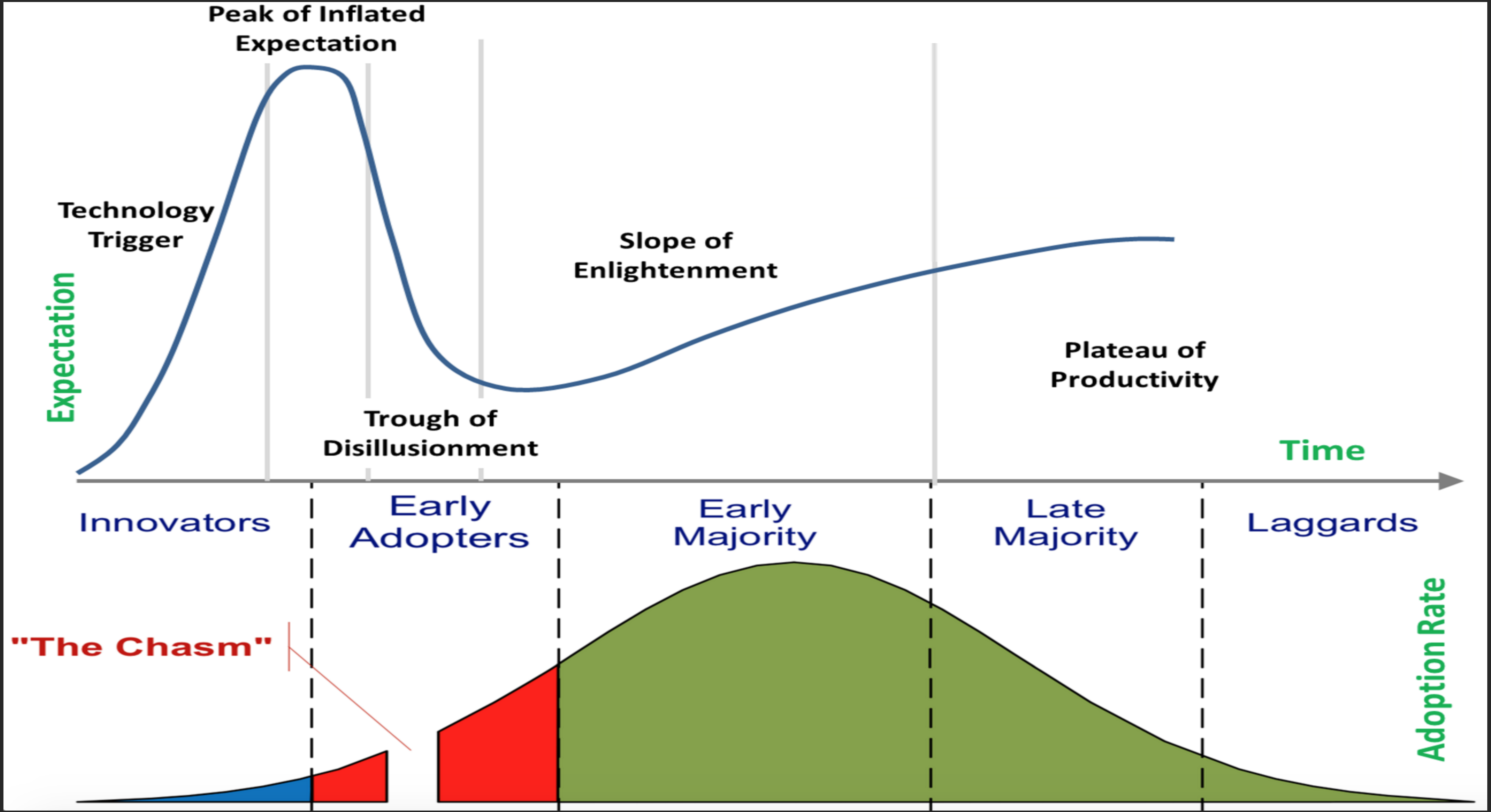
History of data processing

> “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	De-normalized/hierarchical
Ad hoc queries	Instantiated views
Scale vertically	Scale horizontally
Good for OLAP	Built for OLTP at scale

Amazon DynamoDB

Fully managed NoSQL

Document or Wide Column

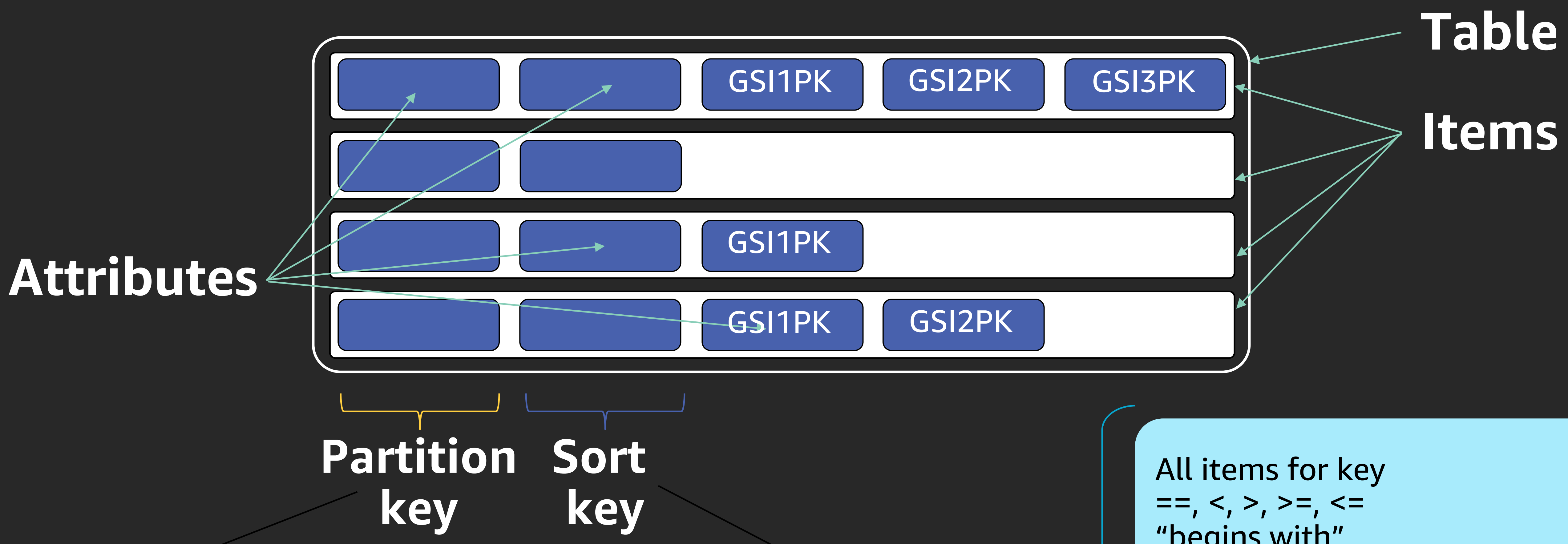
Scales to any workload

Fast and consistent

Access control

Event-driven programming

Table



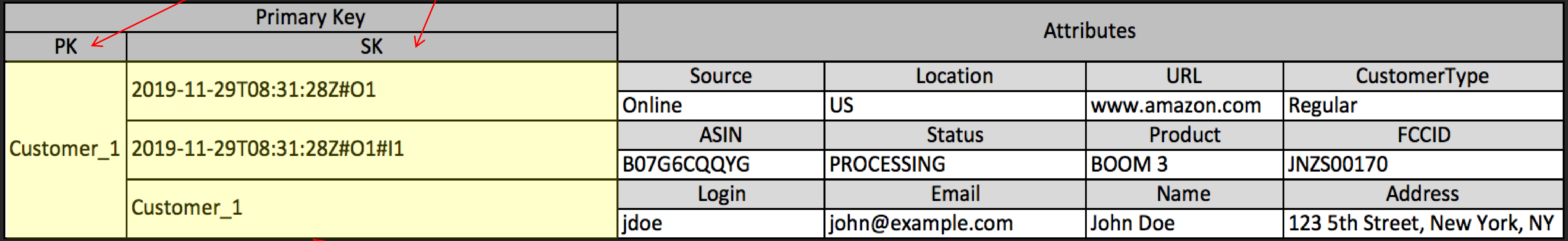
Mandatory
Key-value access pattern
Determines data distribution

Optional
Model 1:N relationships
Enables rich query capabilities

All items for key
==, <, >, >=, <=
“begins with”
“between”
“contains”
“in”
sorted results
counts
top/bottom N values

Partition overloading

Use generic keys to facilitate heterogeneous partitions



Primary Key		Attributes			
PK	SK				
Customer_1	2019-11-29T08:31:28Z#O1	Source	Location	URL	CustomerType
		Online	US	www.amazon.com	Regular
	2019-11-29T08:31:28Z#O1#I1	ASIN	Status	Product	FCCID
		B07G6CQQYG	PROCESSING	BOOM 3	JNZS00170
	Customer_1	Login	Email	Name	Address
		jdoe	john@example.com	John Doe	123 5th Street, New York, NY

SELECT * WHERE PK=Customer_1 AND SK > 2019-10-29

Secondary indexes

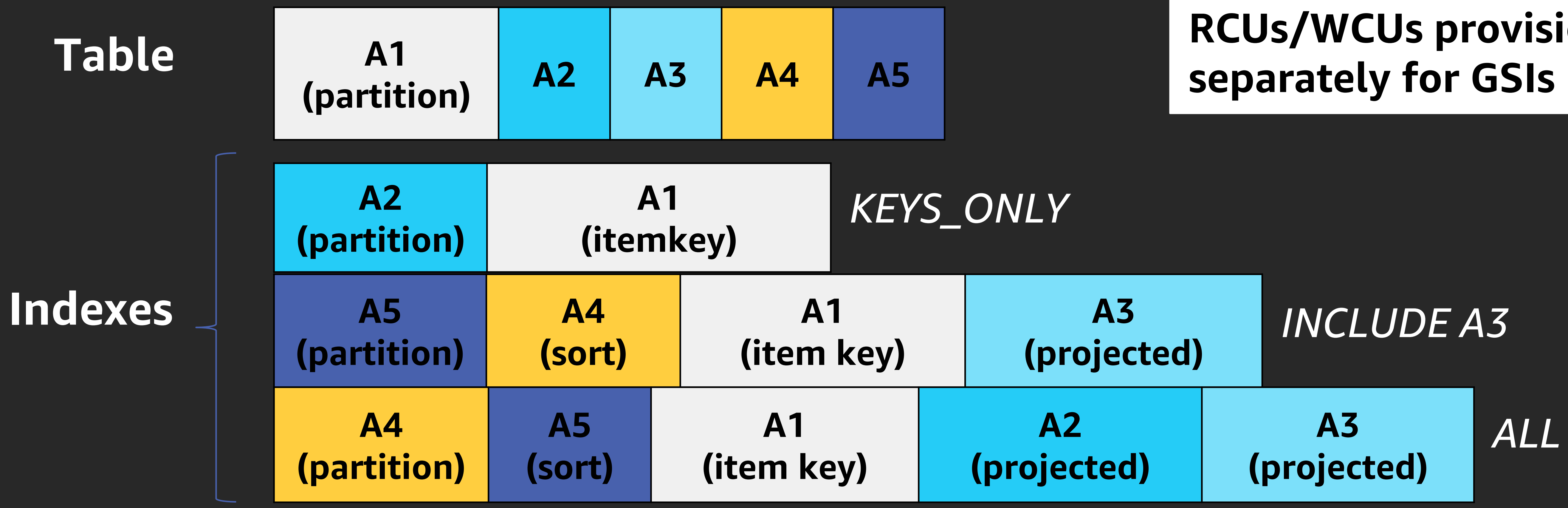
Online indexing

Support secondary access patterns

Index across all partition keys

Use composite sort keys for compound indexes

RCUs/WCUs provisioned separately for GSIs



Table

A1 (partition) | A2 | A3 | A4 | A5

Indexes

A2 (partition) | A1 (itemkey) — *KEYS_ONLY*

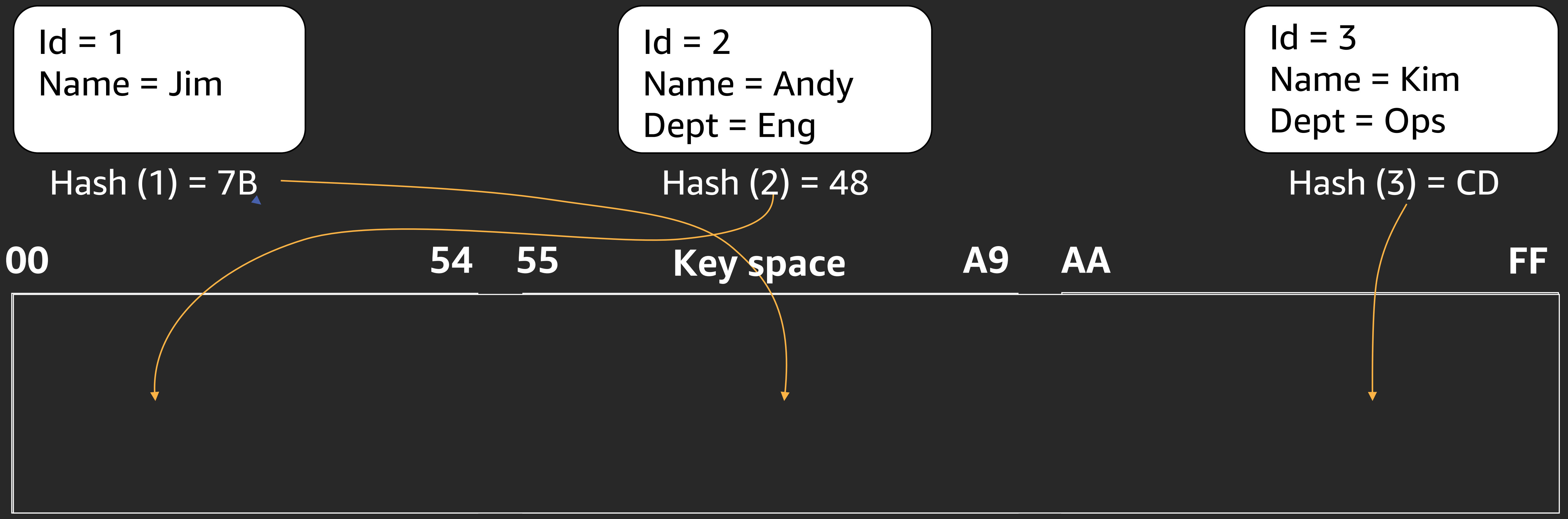
A5 (partition) | A4 (sort) | A1 (item key) | A3 (projected) — *INCLUDE A3*

A4 (partition) | A5 (sort) | A1 (item key) | A2 (projected) | A3 (projected) — *ALL*

Partition/shard keys in NoSQL

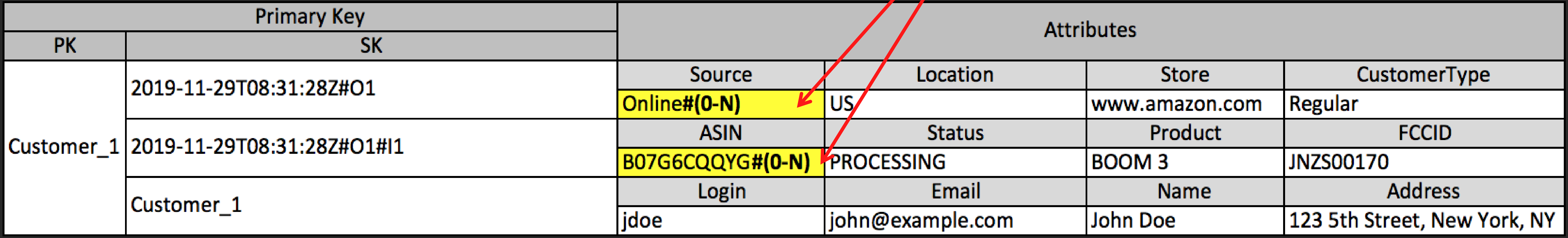
Partition/shard key is used for building an unordered hash index

Allows table to be partitioned for scale

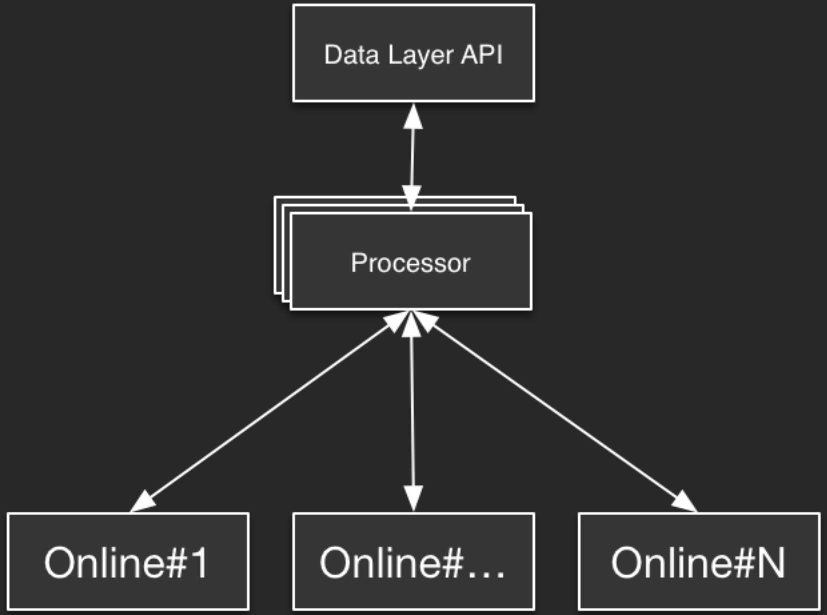


Write sharding

Salt indexed keys to support high-density aggregations on GSIs



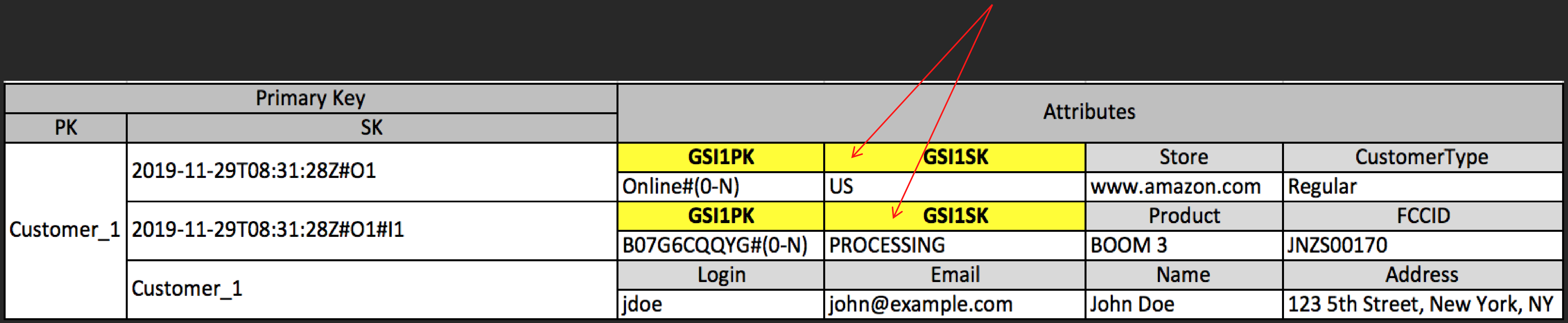
Primary Key		Attributes			
PK	SK				
Customer_1	2019-11-29T08:31:28Z#O1	Source	Location	Store	CustomerType
		Online#**(0-N)**	US	www.amazon.com	Regular
	2019-11-29T08:31:28Z#O1#I1	ASIN	Status	Product	FCCID
		B07G6CQQYG#**(0-N)**	PROCESSING	BOOM 3	JNZS00170
	Customer_1	Login	Email	Name	Address
		jdoe	john@example.com	John Doe	123 5th Street, New York, NY



- Abstract partitioning from clients behind an API
- Write across many partitions
- Use parallel processes to increase read throughput

Index overloading

Use generic keys once more to use indexes for multiple access patterns



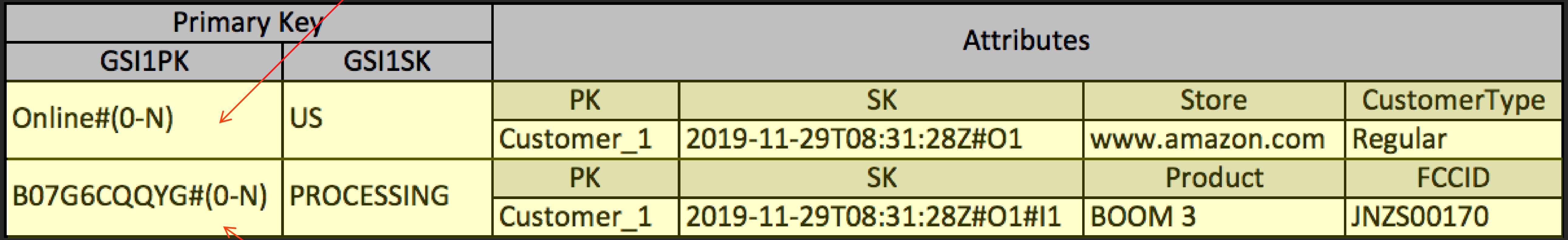
Primary Key		Attributes			
PK	SK				
Customer_1	2019-11-29T08:31:28Z#O1	**GSI1PK**	**GSI1SK**	Store	CustomerType
		Online#(0-N)	US	www.amazon.com	Regular
	2019-11-29T08:31:28Z#O1#I1	**GSI1PK**	**GSI1SK**	Product	FCCID
		B07G6CQQYG#(0-N)	PROCESSING	BOOM 3	JNZS00170
	Customer_1	Login	Email	Name	Address
		jdoe	john@example.com	John Doe	123 5th Street, New York, NY

Index overloading

SELECT * WHERE PK=ONLINE#0 AND SK=US

...

SELECT * WHERE PK=ONLINE#N AND SK=US



Primary Key		Attributes			
GSI1PK	GSI1SK				
Online#(0-N)	US	PK	SK	Store	CustomerType
		Customer_1	2019-11-29T08:31:28Z#O1	www.amazon.com	Regular
B07G6CQQYG#(0-N)	PROCESSING	PK	SK	Product	FCCID
		Customer_1	2019-11-29T08:31:28Z#O1#I1	BOOM 3	JNZS00170

SELECT * WHERE PK=B07G6CQQYG#0 AND SK=PROCESSING

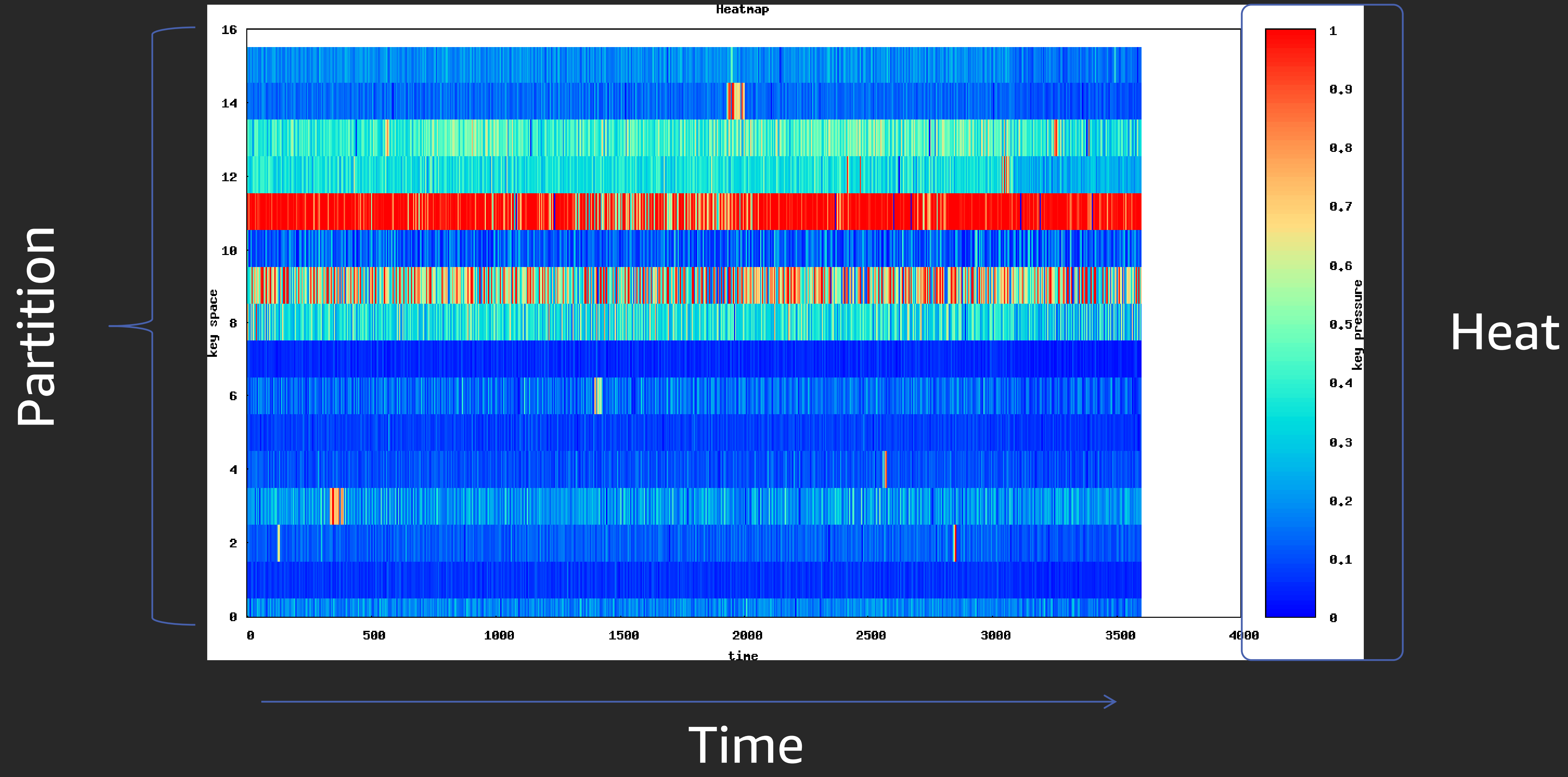
...

SELECT * WHERE PK=B07G6CQQYG#N AND SK=PROCESSING

Scaling NoSQL

> “We are stuck with technology when what we really want is just stuff that works.”
>
> – Douglas Adams

What bad NoSQL looks like



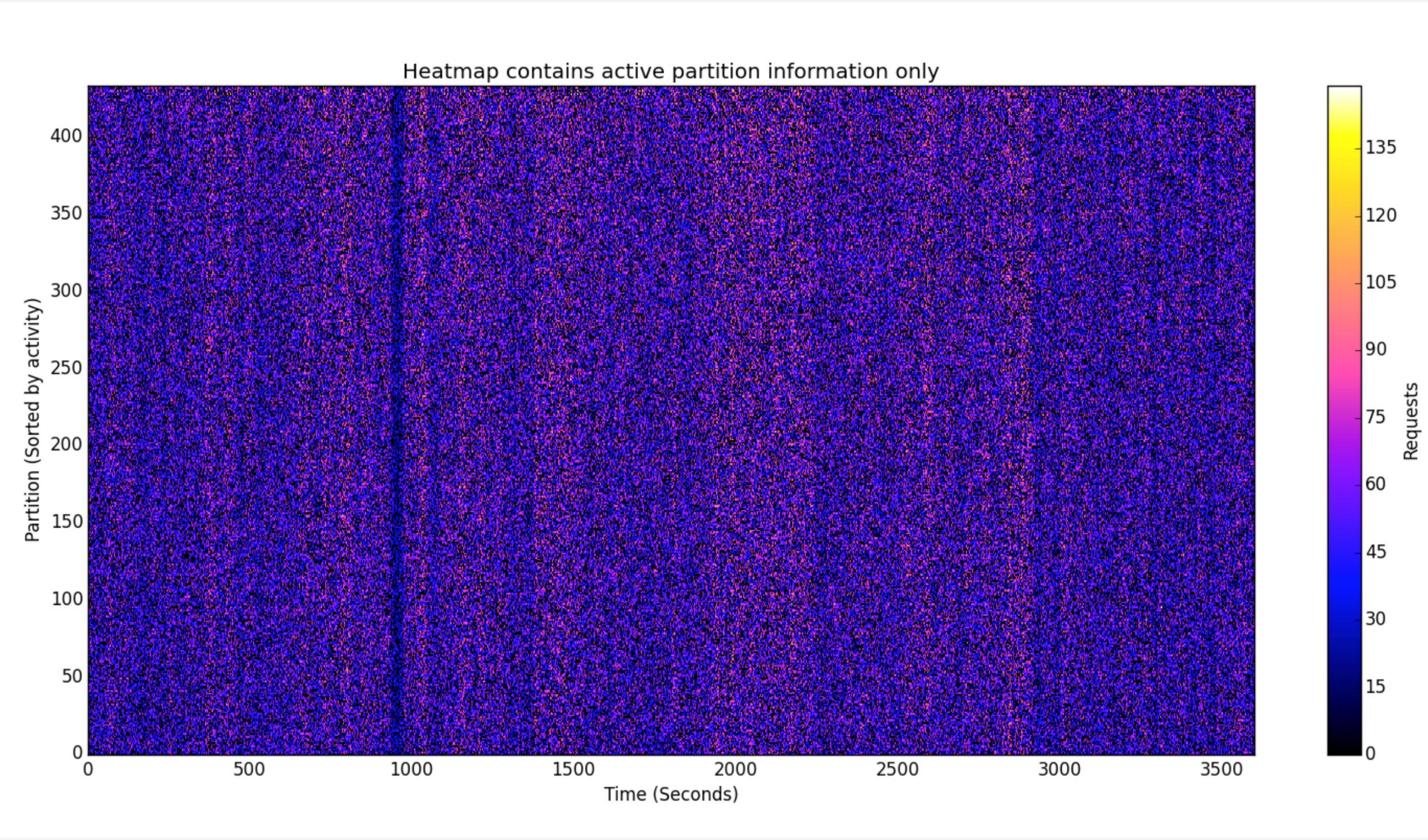
Getting the most out of DynamoDB throughput

> “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 key space

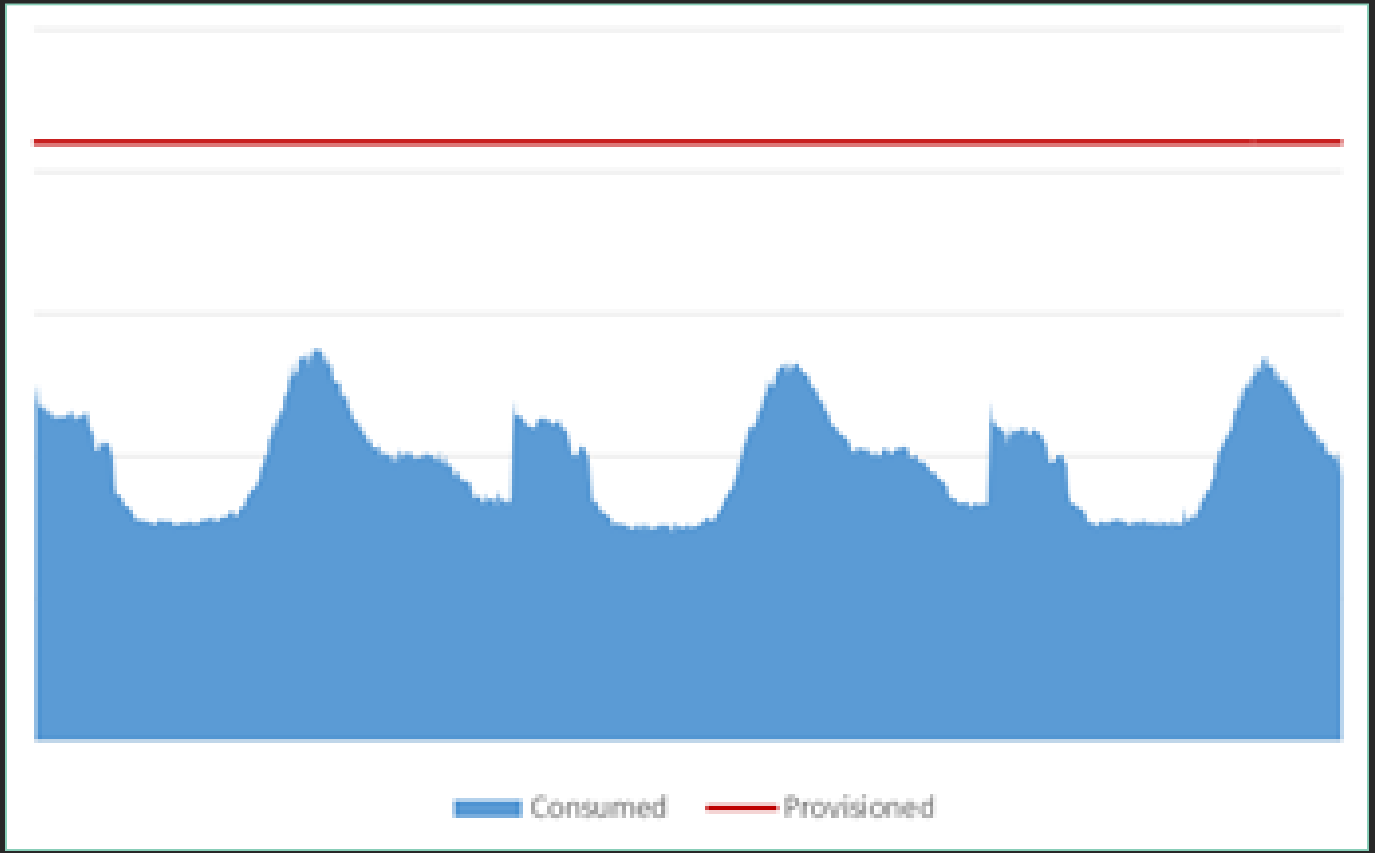
Time: Requests arrive evenly spaced in time

Much better picture

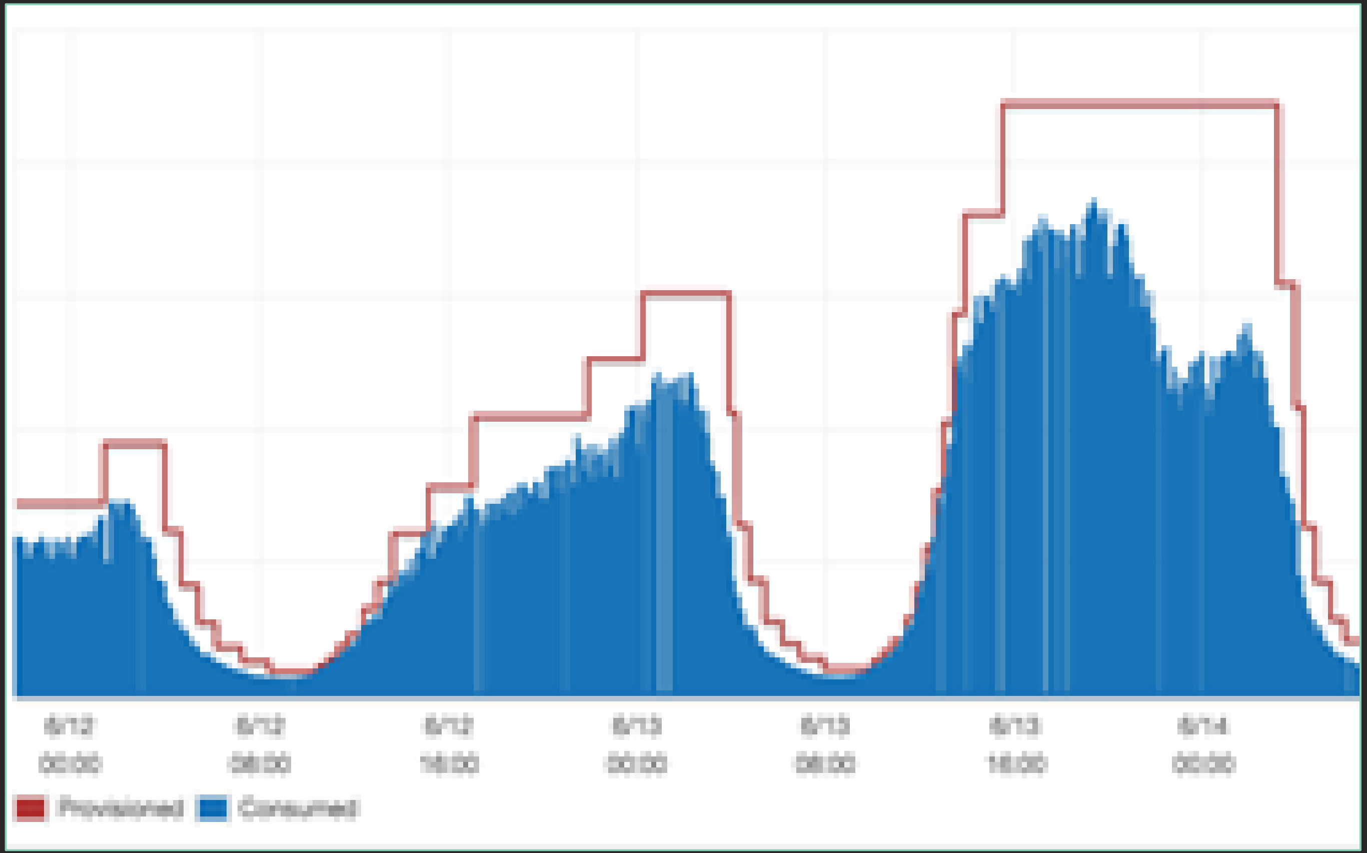


Auto scaling

Throughput automatically adapts to your actual traffic



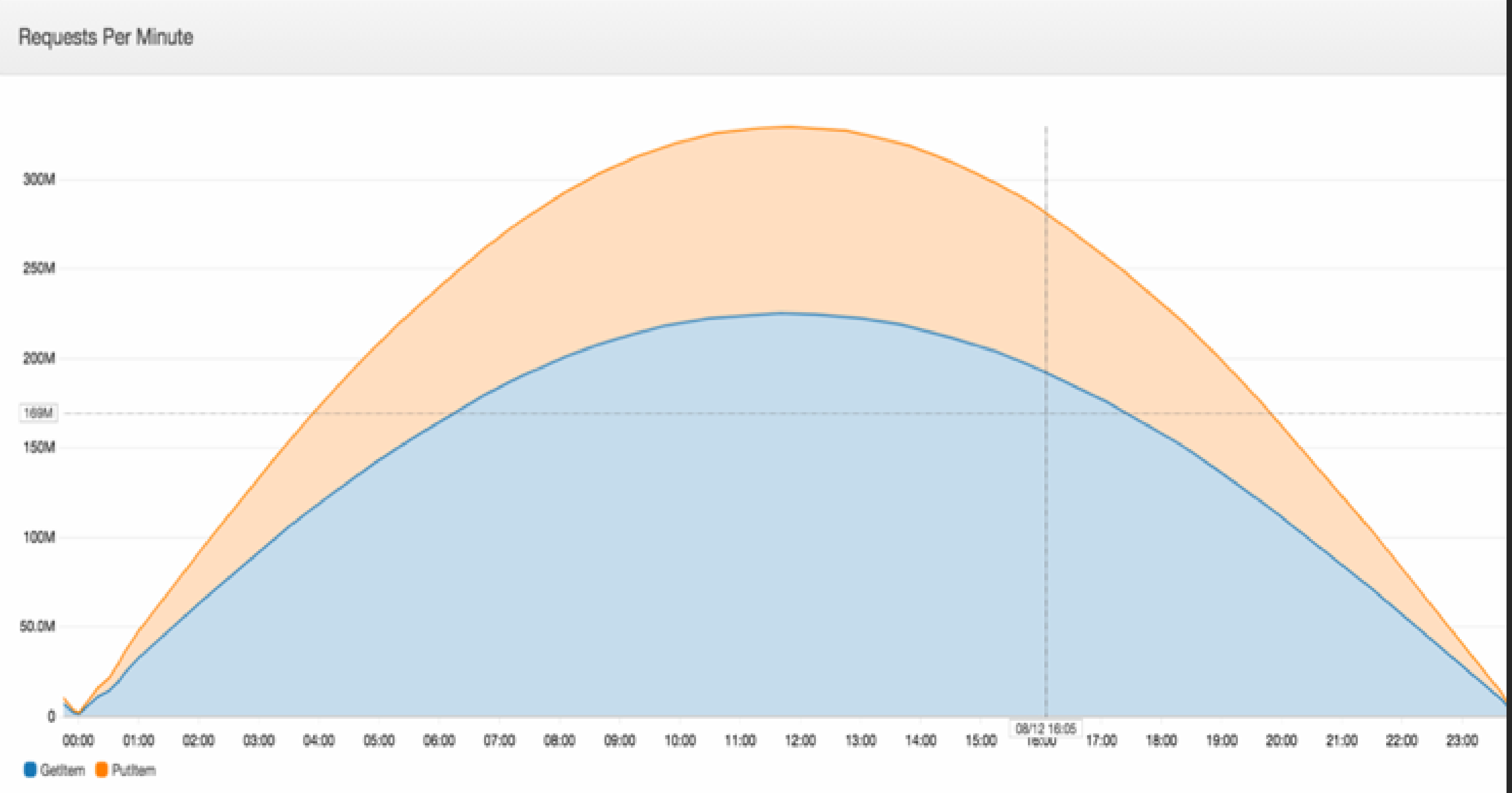
Without auto scaling



With auto scaling

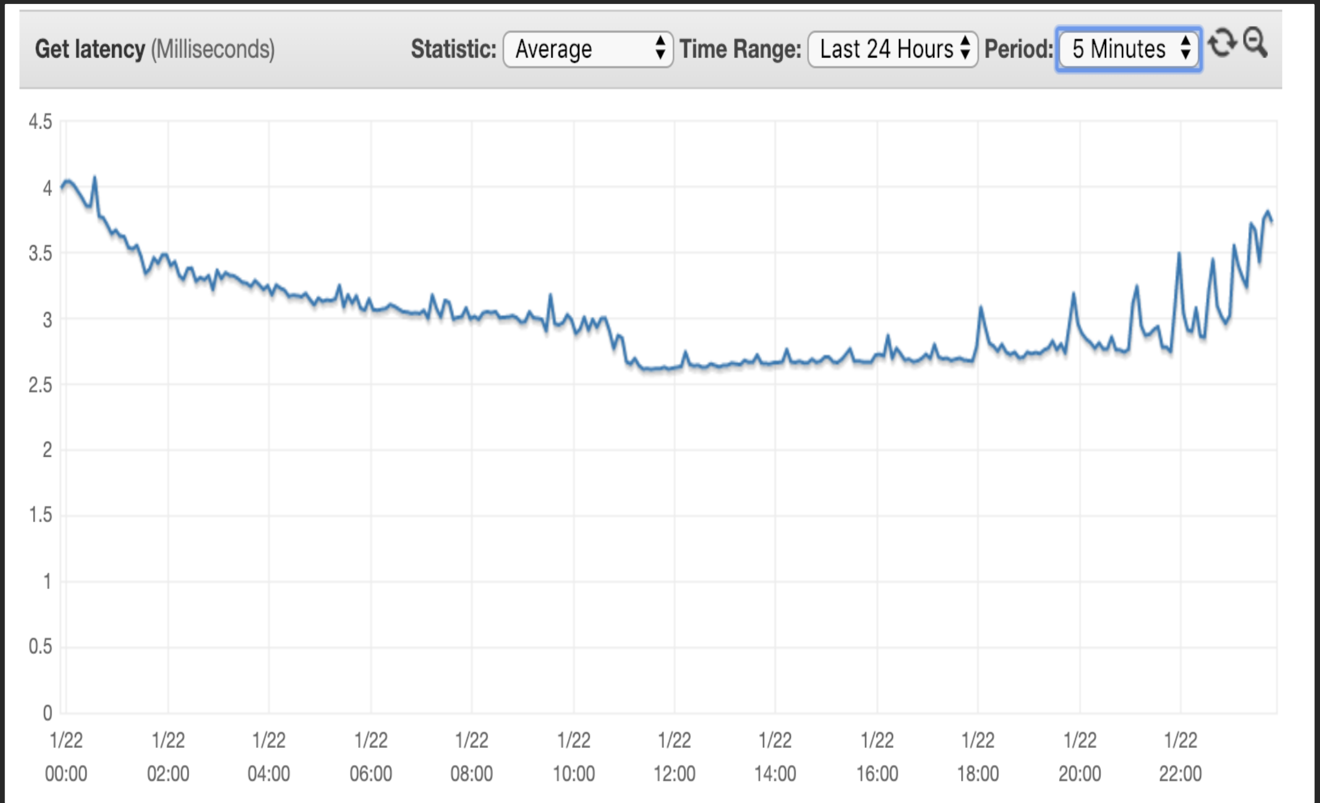
Performance at any scale

High request volume



Many **millions of requests** per second per table

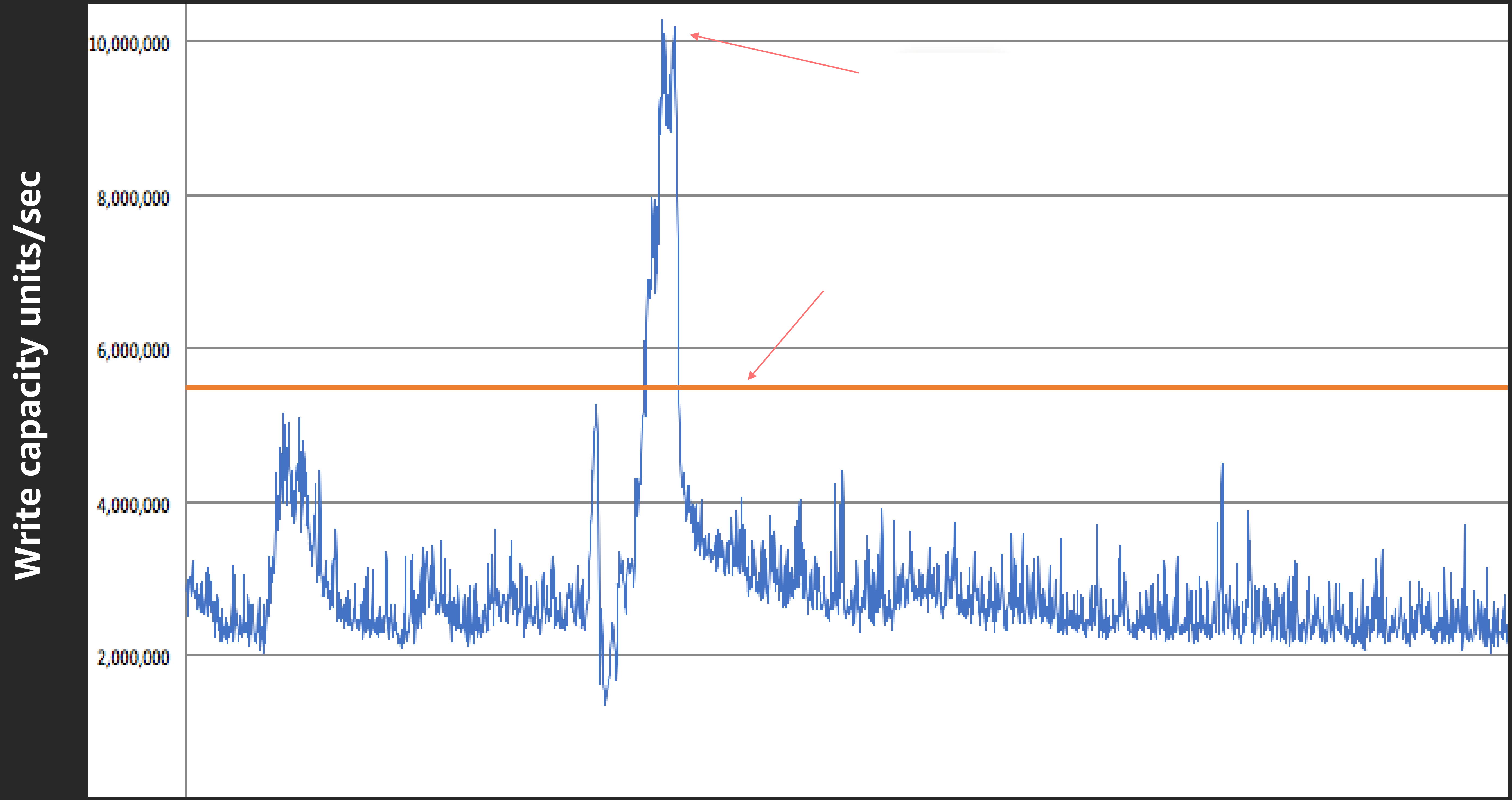
Consistent low latency



Statistic: Average Time Range: Last 24 Hours Period: 5 Minutes

Millisecond variance

Global-scale events: Elastic is the new normal



NoSQL data modeling

“If we have data, let’s look at data. If all we have are opinions, let’s go with mine.”

– Jim Barksdale

It’s all about relationships

Social network

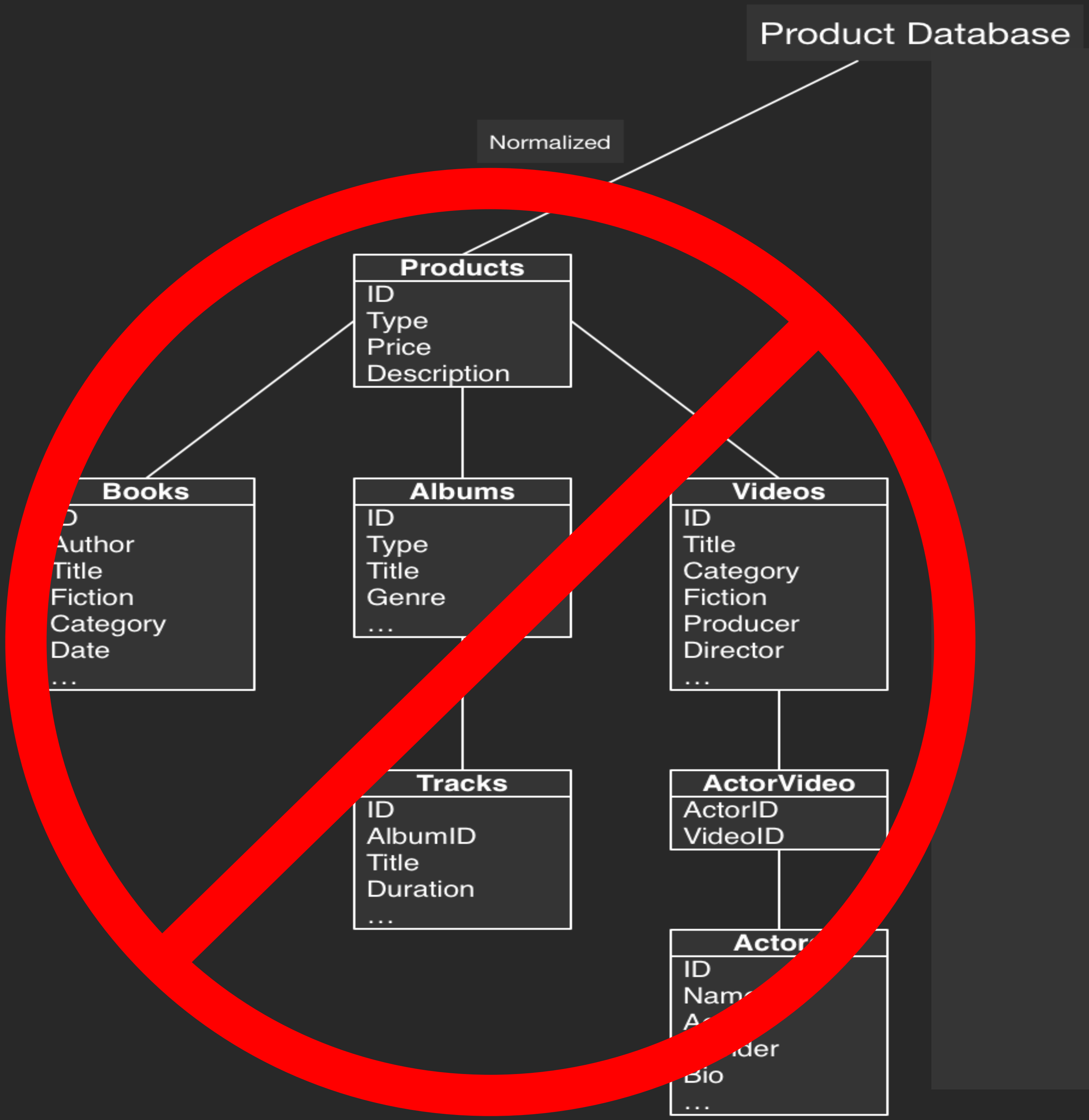
Document management

Process control

IT monitoring

Data trees

SQL vs. NoSQL design pattern

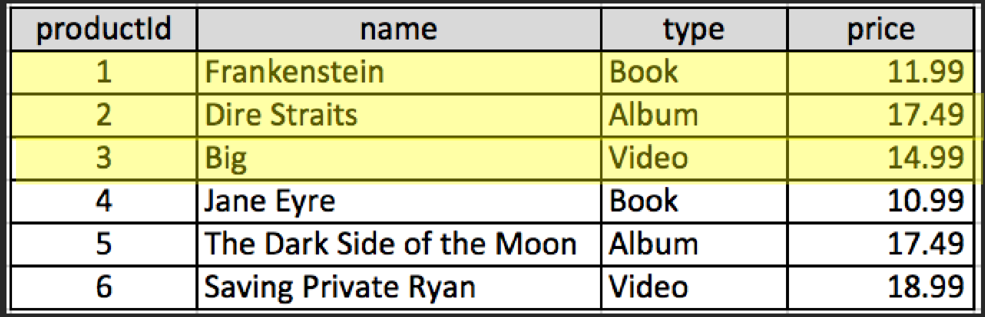


Ad hoc "joins" in SQL

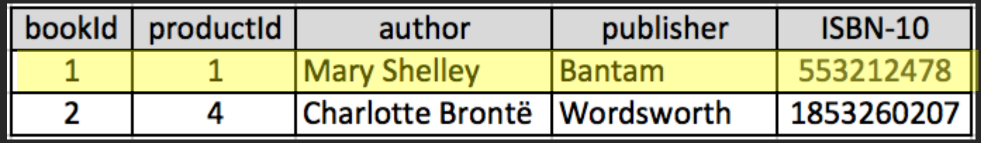
SELECT * FROM PRODUCTS
INNER JOIN BOOKS ON
productId = productId
WHERE name = "Book Title"

SELECT * FROM PRODUCTS
INNER JOIN ALBUMS ON
productId = productId
INNER JOIN TRACKS ON
albumId = albumId
WHERE name = "Album Title"

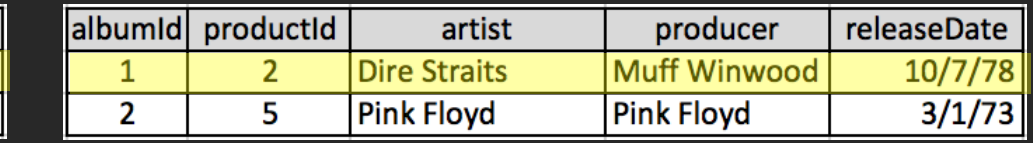
SELECT * FROM PRODUCTS
INNER JOIN VIDEOS ON
productId = productId
INNER JOIN ACTORVIDEO ON
videoId = videoId
INNER JOIN ACTORS ON
actorId = actorId
WHERE name = "Movie Title"



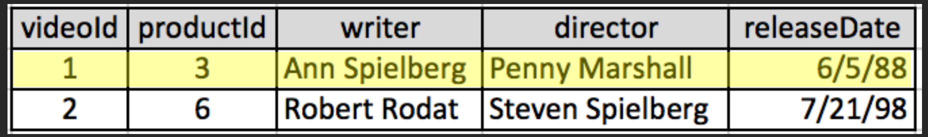
productId	name	type	price
1	Frankenstein	Book	11.99
2	Dire Straits	Album	17.49
3	Big	Video	14.99
4	Jane Eyre	Book	10.99
5	The Dark Side of the Moon	Album	17.49
6	Saving Private Ryan	Video	18.99



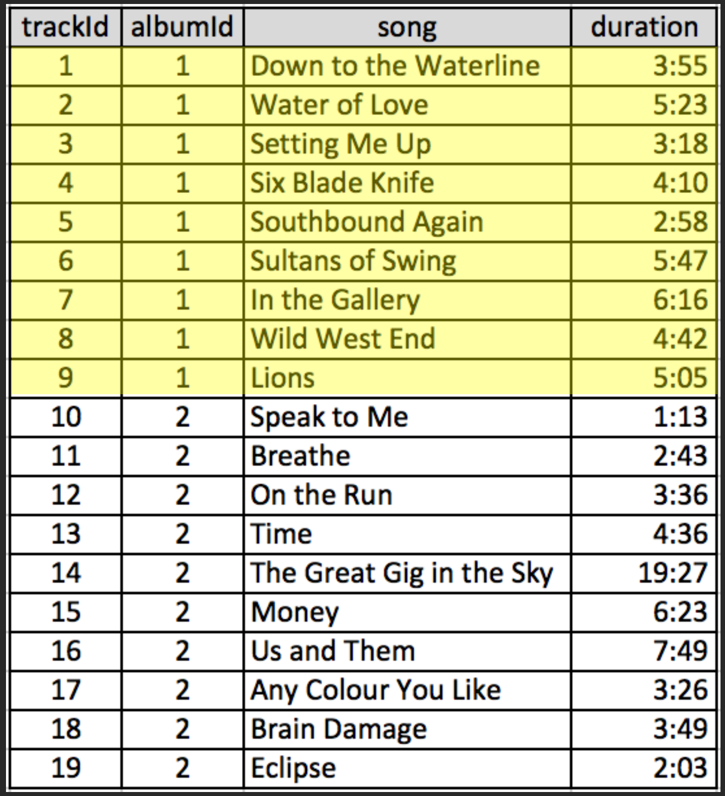
bookId	productId	author	publisher	ISBN-10
1	1	Mary Shelley	Bantam	553212478
2	4	Charlotte Brontë	Wordsworth	1853260207



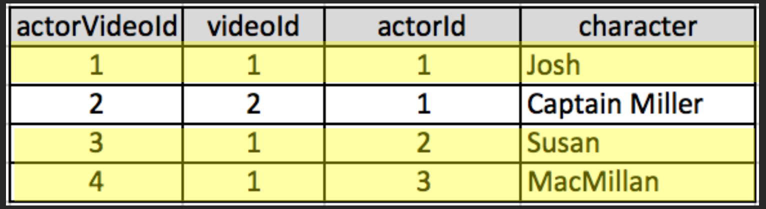
albumId	productId	artist	producer	releaseDate
1	2	Dire Straits	Muff Winwood	10/7/78
2	5	Pink Floyd	Pink Floyd	3/1/73



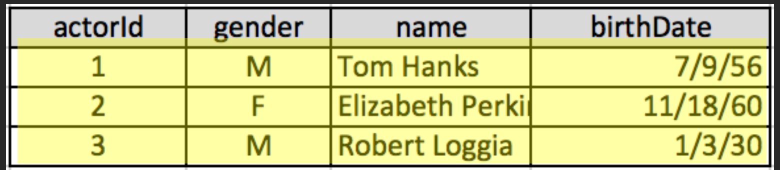
videoId	productId	writer	director	releaseDate
1	3	Ann Spielberg	Penny Marshall	6/5/88
2	6	Robert Rodat	Steven Spielberg	7/21/98



trackId	albumId	song	duration
1	1	Down to the Waterline	3:55
2	1	Water of Love	5:23
3	1	Setting Me Up	3:18
4	1	Six Blade Knife	4:10
5	1	Southbound Again	2:58
6	1	Sultans of Swing	5:47
7	1	In the Gallery	6:16
8	1	Wild West End	4:42
9	1	Lions	5:05
10	2	Speak to Me	1:13
11	2	Breathe	2:43
12	2	On the Run	3:36
13	2	Time	4:36
14	2	The Great Gig in the Sky	19:27
15	2	Money	6:23
16	2	Us and Them	7:49
17	2	Any Colour You Like	3:26
18	2	Brain Damage	3:49
19	2	Eclipse	2:03



actorVideoId	videoId	actorId	character
1	1	1	Josh
2	2	1	Captain Miller
3	1	2	Susan
4	1	3	MacMillan



actorId	gender	name	birthDate
1	M	Tom Hanks	7/9/56
2	F	Elizabeth Perki	11/18/60
3	M	Robert Loggia	1/3/30

Time Complexity

O(log(N) + Nlog(M) + Nlog(M))

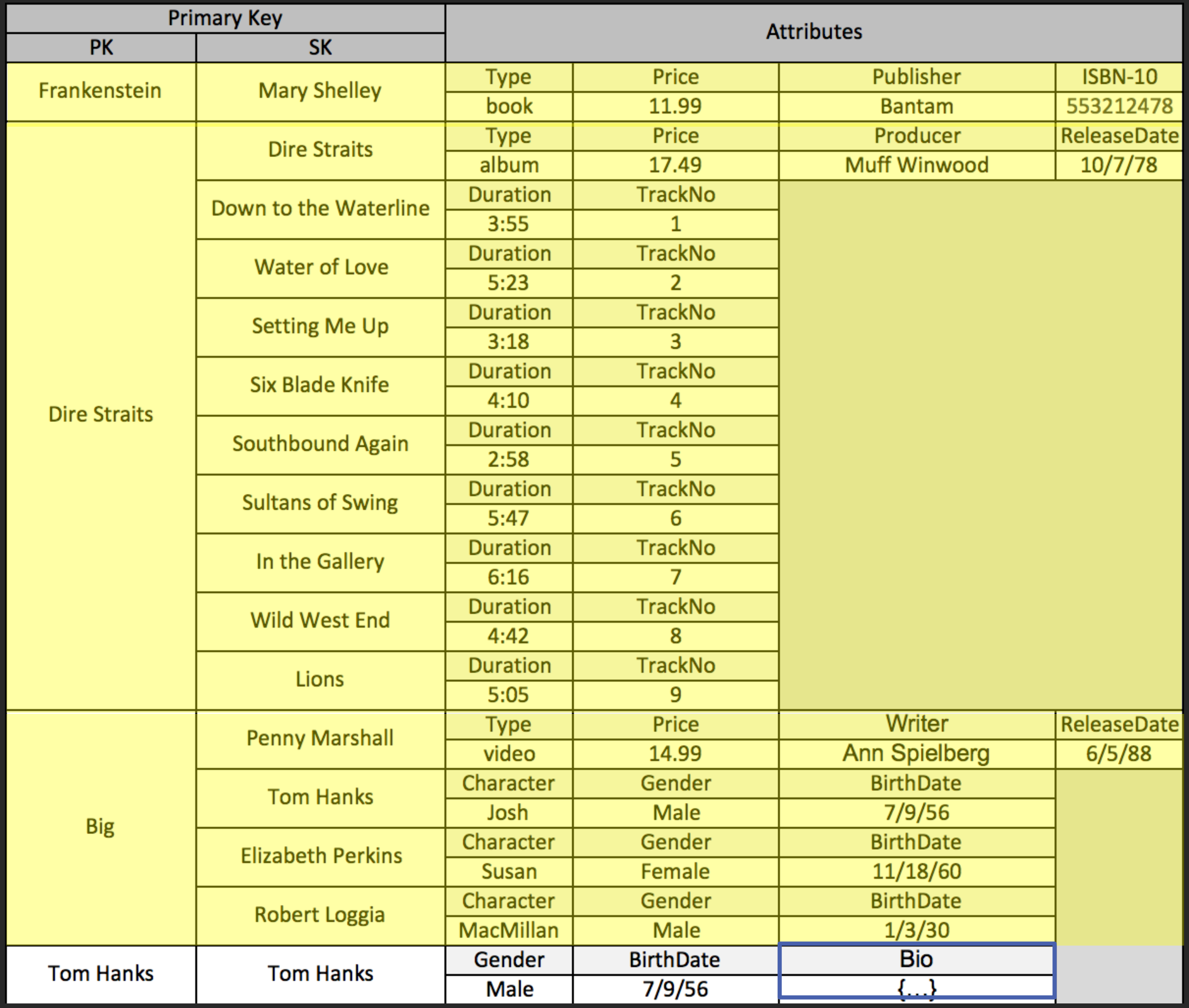
O(log(N) + Nlog(M))

Modeled “joins” in NoSQL

SELECT * WHERE PK=“Book Title”

SELECT * WHERE PK=“Album Title”

SELECT * WHERE PK=“Movie Title”



Primary Key		Attributes			
PK	SK				
Frankenstein	Mary Shelley	Type	Price	Publisher	ISBN-10
		book	11.99	Bantam	553212478
Dire Straits	Dire Straits	Type	Price	Producer	ReleaseDate
		album	17.49	Muff Winwood	10/7/78
	Down to the Waterline	Duration	TrackNo		
		3:55	1		
	Water of Love	Duration	TrackNo		
		5:23	2		
	Setting Me Up	Duration	TrackNo		
		3:18	3		
	Six Blade Knife	Duration	TrackNo		
		4:10	4		
	Southbound Again	Duration	TrackNo		
		2:58	5		
	Sultans of Swing	Duration	TrackNo		
		5:47	6		
	In the Gallery	Duration	TrackNo		
		6:16	7		
	Wild West End	Duration	TrackNo		
		4:42	8		
	Lions	Duration	TrackNo		
		5:05	9		
Big	Penny Marshall	Type	Price	Writer	ReleaseDate
		video	14.99	Ann Spielberg	6/5/88
	Tom Hanks	Character	Gender	BirthDate	
		Josh	Male	7/9/56	
	Elizabeth Perkins	Character	Gender	BirthDate	
		Susan	Female	11/18/60	
	Robert Loggia	Character	Gender	BirthDate	
		MacMillan	Male	1/3/30	
Tom Hanks	Tom Hanks	Gender	BirthDate	Bio	
		Male	7/9/56	{…}	

Time Complexity

O(1)

Modeled “joins” in NoSQL

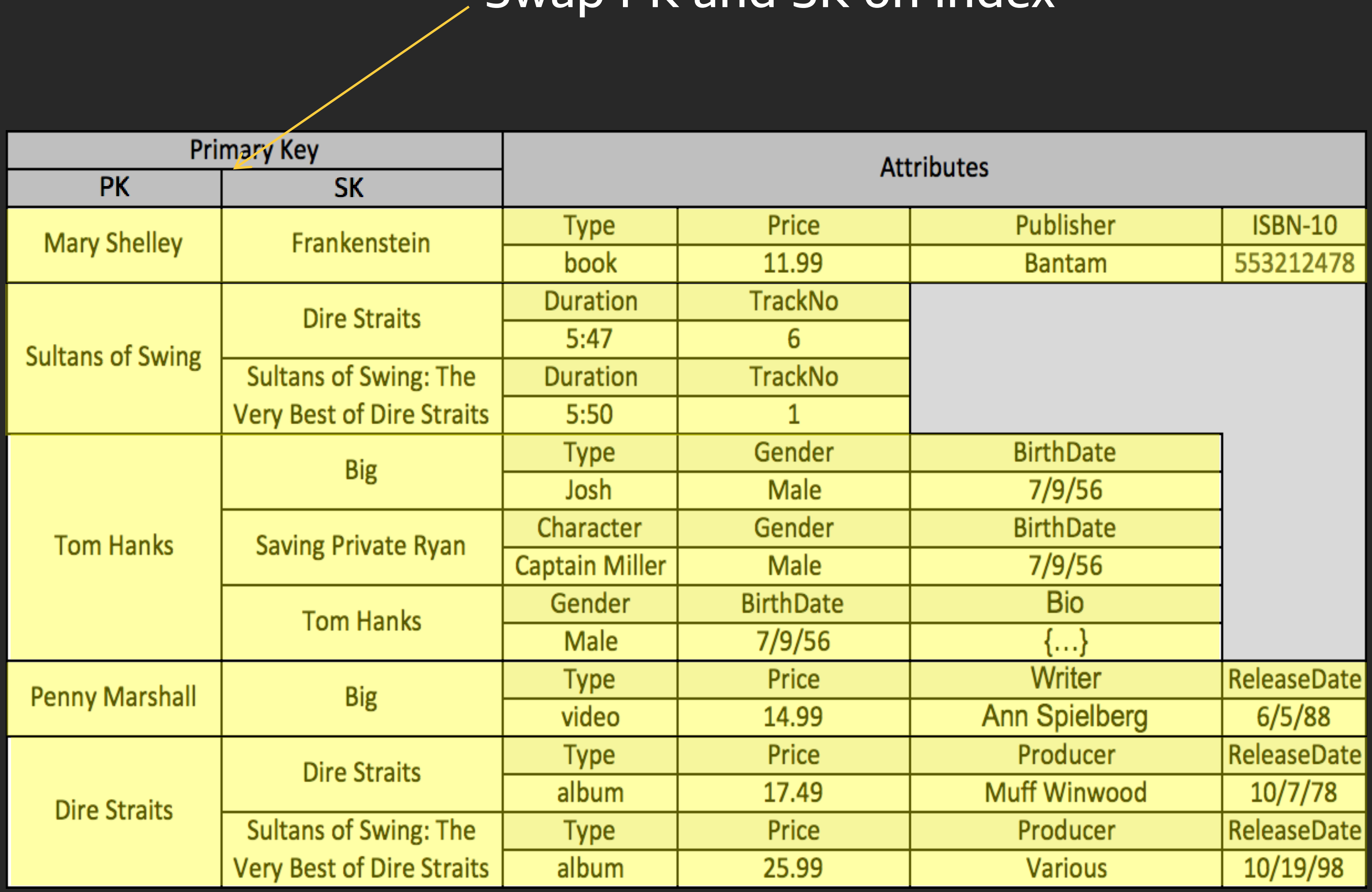
SELECT * WHERE SK=“Author Name”

SELECT * WHERE SK=“Song Title”

SELECT * WHERE SK=“Actor Name”

SELECT * WHERE SK=“Director Name”

SELECT * WHERE SK=“Musician”



Swap PK and SK on index

Primary Key		Attributes			
PK	SK				
Mary Shelley	Frankenstein	Type	Price	Publisher	ISBN-10
		book	11.99	Bantam	553212478
Sultans of Swing	Dire Straits	Duration	TrackNo		
		5:47	6		
	Sultans of Swing: The Very Best of Dire Straits	Duration	TrackNo		
		5:50	1		
Tom Hanks	Big	Type	Gender	BirthDate	
		Josh	Male	7/9/56	
	Saving Private Ryan	Character	Gender	BirthDate	
		Captain Miller	Male	7/9/56	
	Tom Hanks	Gender	BirthDate	Bio	
		Male	7/9/56	{…}	
Penny Marshall	Big	Type	Price	Writer	ReleaseDate
		video	14.99	Ann Spielberg	6/5/88
Dire Straits	Dire Straits	Type	Price	Producer	ReleaseDate
		album	17.49	Muff Winwood	10/7/78
	Sultans of Swing: The Very Best of Dire Straits	Type	Price	Producer	ReleaseDate
		album	25.99	Various	10/19/98

Document vs. wide column data modeling

```
{
        _id: “john@example.com”,
        firstName: “John”,
        lastName: “Doe”,
        address: “123 A Street”,
        city: “Seattle”,
        state: “WA”,
        building: “SEA58”,
        floor: “07.650.O1”
}
```

Default “_id” index supports K/V access patterns, e.g., “Get employee data by email”, etc.

Compound index on “building.floor” supports subtree aggregations for employees by location: SELECT * WHERE building == “SEA58” AND floor startsWith(“07”)

Document vs. wide column

```
{
    _id: “john@example.com”,
    firstName: “John”,
    lastName: “Doe”,
    address: “123 A Street”,
    city: “Seattle”,
    state: “WA”,
    building: “SEA58”,
    floor: “07.650.01”
}
```

PK (_id)	firstName	lastName	address	city	state	GSIPK	GSISK

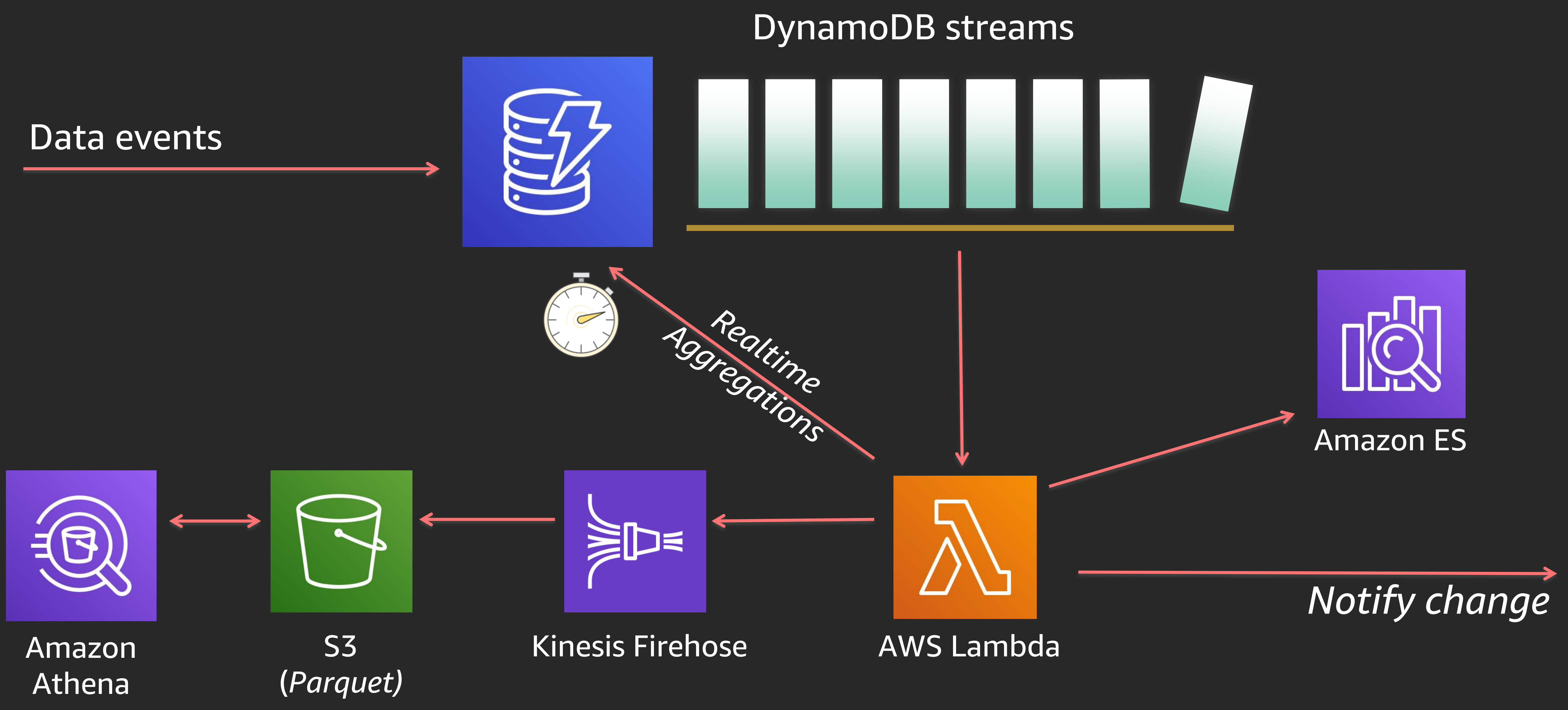
Indexing efficiently in NoSQL

Document	**Wide column**
Default index on **_id**	**Partition Key** defines default index
Query planner selects the index	User specifies the index
Include **Shard Key** or suffer	**Partition Key** value always required
Optimize with **Compound Indexes**	Use **Projections** to “pre-load” the index

Complex queries

> “Computers are useless. They can only give you answers.”
>
> – Pablo Picasso

Serverless & event driven architecture



Composite keys

> “Hierarchies are celestial. In hell all are equal.”
>
> – Nicolás Gómez Dávila

AWS re:Invent

aws

Multi-value sorts and filters

Bob

Partition key

Sort key

Secondary index

Opponent	Date	GameId	Status	Host
Alice	2014-10-02	d9bl3	DONE	David
Carol	2014-10-08	o2pnb	IN_PROGRESS	Bob
Bob	2014-09-30	72f49	PENDING	Alice
Bob	2014-10-03	b932s	PENDING	Carol
Bob	2014-10-03	ef9ca	IN_PROGRESS	David

Approach 1: Query filter

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

Bob

Secondary index

Opponent	Date	GameId	Status	Host
Alice	2014-10-02	d9bl3	DONE	David
Carol	2014-10-08	o2pnb	IN_PROGRESS	Bob
Bob	2014-09-30	72f49	PENDING	Alice
Bob	2014-10-03	b932s	PENDING	Carol
Bob	2014-10-03	ef9ca	IN_PROGRESS	David

(Filtered out)

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

Approach 2: Composite key

Partition key

Sort key

Secondary index

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'
```

Bob

Secondary index

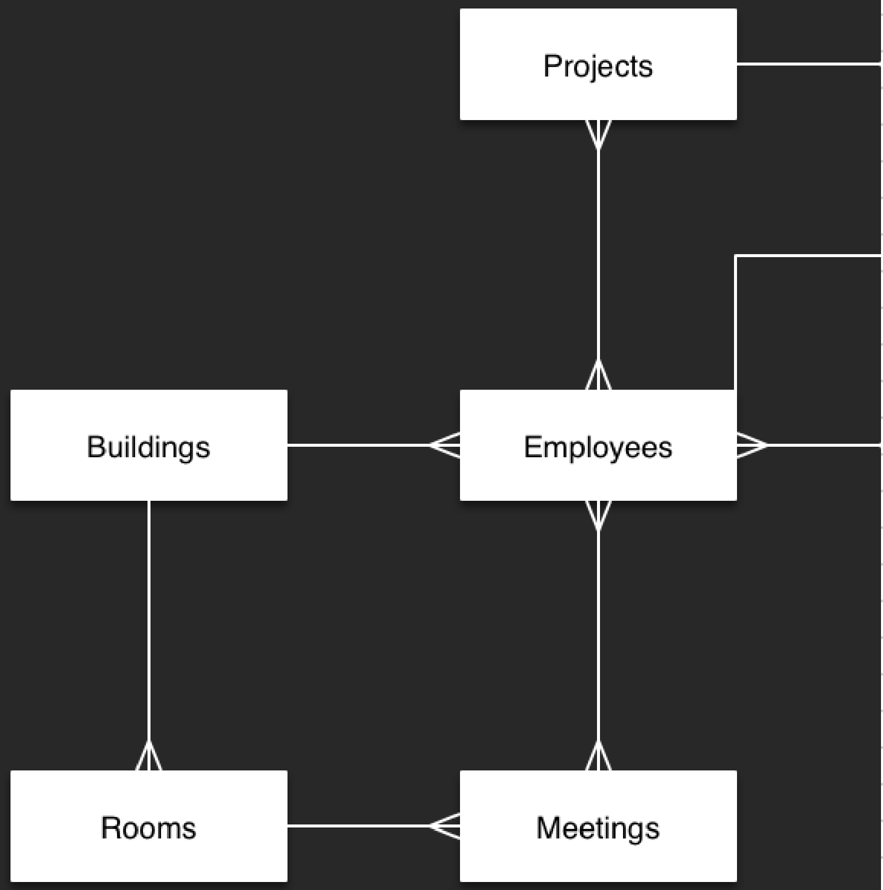
Opponent	StatusDate	GameId	Host
Alice	DONE_2014-10-02	d9bl3	David
Carol	IN_PROGRESS_2014-10-08	o2pnb	Bob
Bob	IN_PROGRESS_2014-10-03	ef9ca	David
Bob	PENDING_2014-09-30	72f49	Alice
Bob	PENDING_2014-10-03	b932s	Carol

Modeling relational data

“Dude, where’s my lookup table?”

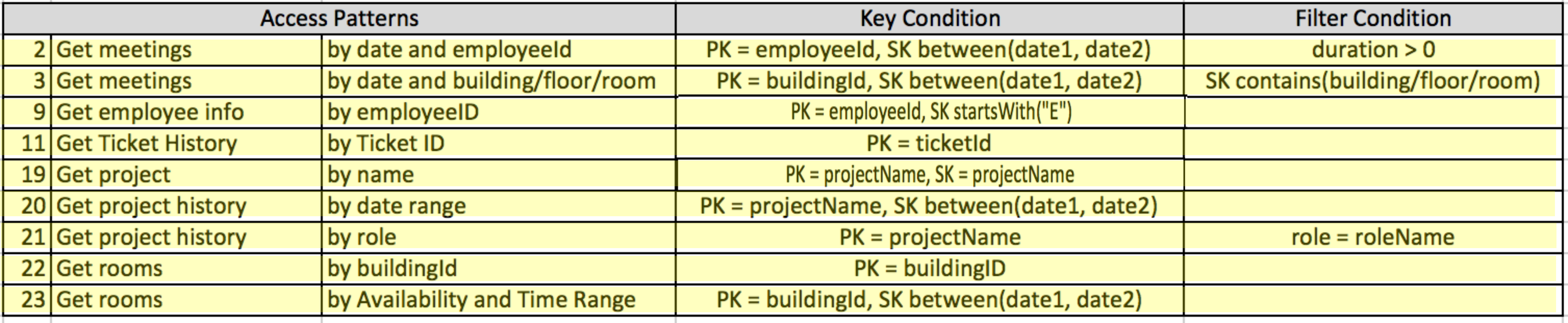
- Anonymous Amazon SDE

Modeling complex relationships

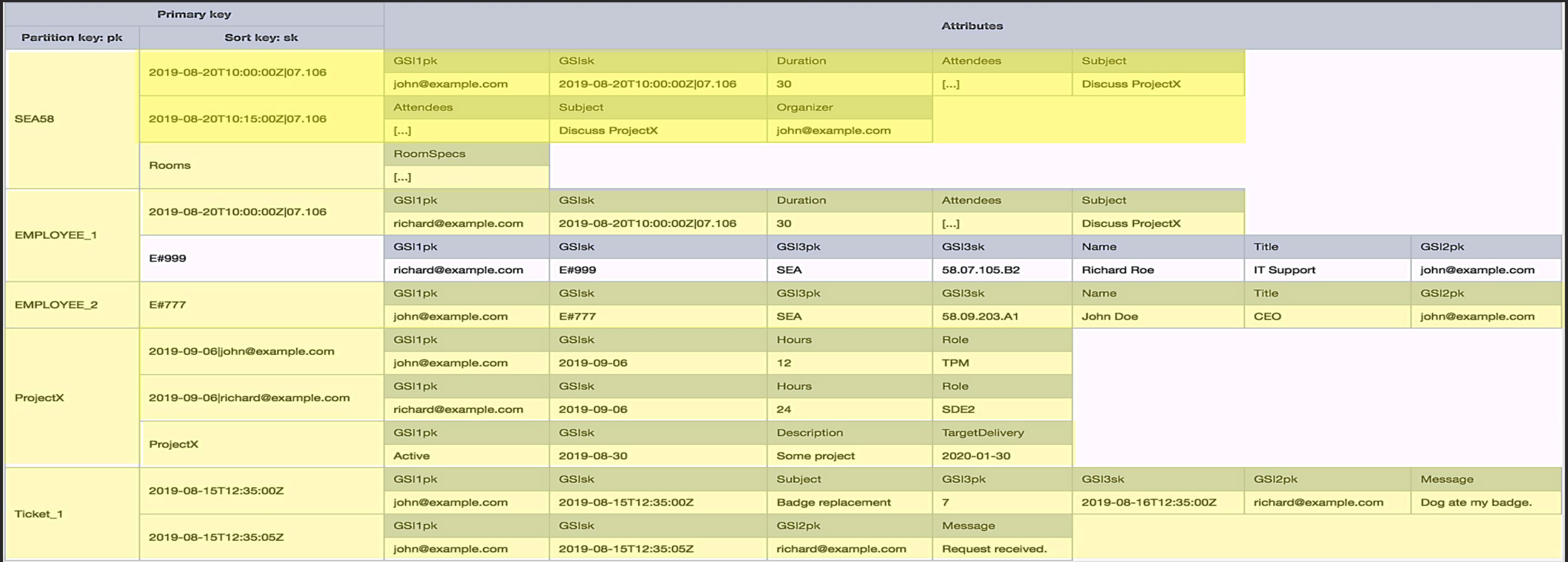


Access Patterns		
	Get meetings	
1		by date and email
2		by date and employeeId
3		by date and building/floor/room
	Load employee dashboard by email	
4		Get employee data
5		Get meetings
6		Get tickets
7		Get reservations
8		Get time cards
	Get employee info	
9		by employeeID
10		by email
	Get Ticket history	
11		by Ticket ID
12		by employee email
13		by assignee email
	Get employees	
14		by city, building, floor, aisle, desk
15		by manager
	Get assigned tickets	
16		by email
	Get Tickets	
17		by last touched > 24 hours
	Get project(s)	
18		by status, start and target date
19		by name
	Get project history	
20		by date range
21		by role
	Get Rooms	
22		by buildingID
23		by availability and time range

The table

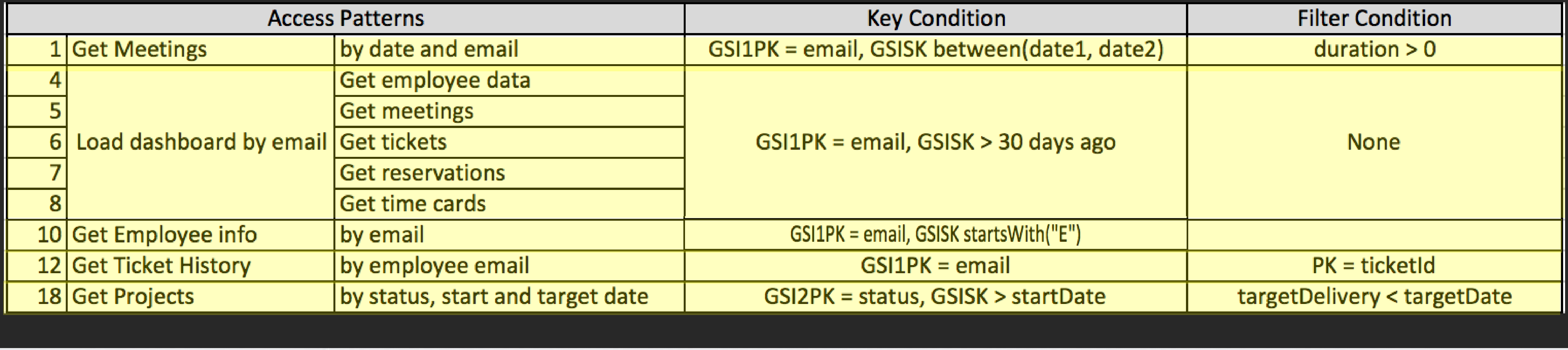


Access Patterns			Key Condition	Filter Condition
2	Get meetings	by date and employeeId	PK = employeeId, SK between(date1, date2)	duration > 0
3	Get meetings	by date and building/floor/room	PK = buildingId, SK between(date1, date2)	SK contains(building/floor/room)
9	Get employee info	by employeeID	PK = employeeId, SK startsWith("E")	
11	Get Ticket History	by Ticket ID	PK = ticketId	
19	Get project	by name	PK = projectName, SK = projectName	
20	Get project history	by date range	PK = projectName, SK between(date1, date2)	
21	Get project history	by role	PK = projectName	role = roleName
22	Get rooms	by buildingId	PK = buildingID	
23	Get rooms	by Availability and Time Range	PK = buildingId, SK between(date1, date2)	

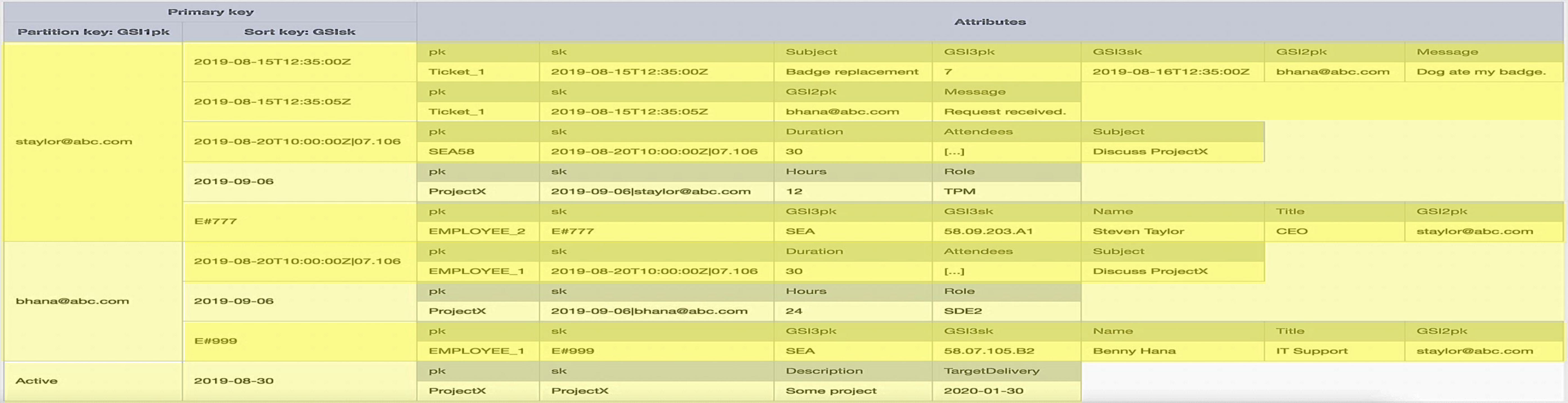


Primary key		Attributes						
Partition key: pk	Sort key: sk							
SEA58	2019-08-20T10:00:00Z|07.106	GSI1pk	GSIsk	Duration	Attendees	Subject		
		john@example.com	2019-08-20T10:00:00Z|07.106	30	[...]	Discuss ProjectX		
	2019-08-20T10:15:00Z|07.106	Attendees	Subject	Organizer				
		[...]	Discuss ProjectX	john@example.com				
	Rooms	RoomSpecs						
		[...]						
EMPLOYEE_1	2019-08-20T10:00:00Z|07.106	GSI1pk	GSIsk	Duration	Attendees	Subject		
		richard@example.com	2019-08-20T10:00:00Z|07.106	30	[...]	Discuss ProjectX		
	E#999	GSI1pk	GSIsk	GSI3pk	GSI3sk	Name	Title	GSI2pk
		richard@example.com	E#999	SEA	58.07.105.B2	Richard Roe	IT Support	john@example.com
EMPLOYEE_2	E#777	GSI1pk	GSIsk	GSI3pk	GSI3sk	Name	Title	GSI2pk
		john@example.com	E#777	SEA	58.09.203.A1	John Doe	CEO	john@example.com
ProjectX	2019-09-06|john@example.com	GSI1pk	GSIsk	Hours	Role			
		john@example.com	2019-09-06	12	TPM			
	2019-09-06|richard@example.com	GSI1pk	GSIsk	Hours	Role			
		richard@example.com	2019-09-06	24	SDE2			
	ProjectX	GSI1pk	GSIsk	Description	TargetDelivery			
		Active	2019-08-30	Some project	2020-01-30			
Ticket_1	2019-08-15T12:35:00Z	GSI1pk	GSIsk	Subject	GSI3pk	GSI3sk	GSI2pk	Message
		john@example.com	2019-08-15T12:35:00Z	Badge replacement	7	2019-08-16T12:35:00Z	richard@example.com	Dog ate my badge.
	2019-08-15T12:35:05Z	GSI1pk	GSIsk	GSI2pk	Message			
		john@example.com	2019-08-15T12:35:05Z	richard@example.com	Request received.			

The index schema (GSI1)

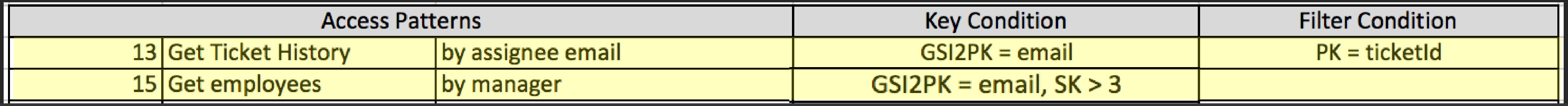


Access Patterns			Key Condition	Filter Condition
1	Get Meetings	by date and email	GSI1PK = email, GSISK between(date1, date2)	duration > 0
4	Load dashboard by email	Get employee data	GSI1PK = email, GSISK > 30 days ago	None
5		Get meetings		
6		Get tickets		
7		Get reservations		
8		Get time cards		
10	Get Employee info	by email	GSI1PK = email, GSISK startsWith("E")	
12	Get Ticket History	by employee email	GSI1PK = email	PK = ticketId
18	Get Projects	by status, start and target date	GSI2PK = status, GSISK > startDate	targetDelivery < targetDate

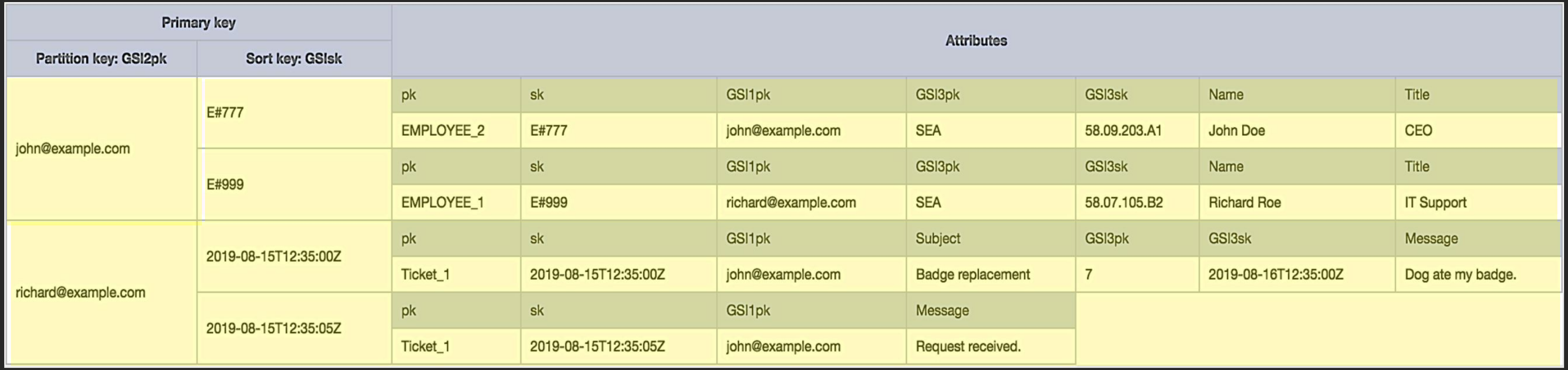


Primary key		Attributes						
Partition key: GSI1pk	Sort key: GSIsk							
staylor@abc.com	2019-08-15T12:35:00Z	pk	sk	Subject	GSI3pk	GSI3sk	GSI2pk	Message
		Ticket_1	2019-08-15T12:35:00Z	Badge replacement	7	2019-08-16T12:35:00Z	bhana@abc.com	Dog ate my badge.
	2019-08-15T12:35:05Z	pk	sk	GSI2pk	Message			
		Ticket_1	2019-08-15T12:35:05Z	bhana@abc.com	Request received.			
	2019-08-20T10:00:00Z|07.106	pk	sk	Duration	Attendees	Subject		
		SEA58	2019-08-20T10:00:00Z|07.106	30	[...]	Discuss ProjectX		
	2019-09-06	pk	sk	Hours	Role			
		ProjectX	2019-09-06|staylor@abc.com	12	TPM			
	E#777	pk	sk	GSI3pk	GSI3sk	Name	Title	GSI2pk
		EMPLOYEE_2	E#777	SEA	58.09.203.A1	Steven Taylor	CEO	staylor@abc.com
bhana@abc.com	2019-08-20T10:00:00Z|07.106	pk	sk	Duration	Attendees	Subject		
		EMPLOYEE_1	2019-08-20T10:00:00Z|07.106	30	[...]	Discuss ProjectX		
	2019-09-06	pk	sk	Hours	Role			
		ProjectX	2019-09-06|bhana@abc.com	24	SDE2			
	E#999	pk	sk	GSI3pk	GSI3sk	Name	Title	GSI2pk
		EMPLOYEE_1	E#999	SEA	58.07.105.B2	Benny Hana	IT Support	staylor@abc.com
Active	2019-08-30	pk	sk	Description	TargetDelivery			
		ProjectX	ProjectX	Some project	2020-01-30			

The index schema (GSI2)

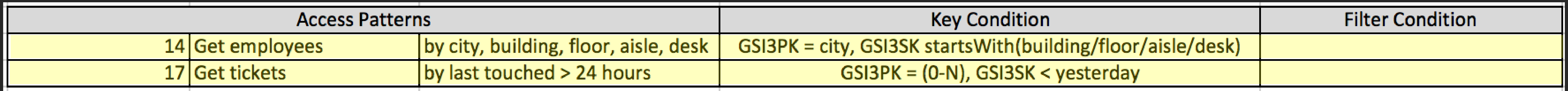


Access Patterns			Key Condition	Filter Condition
13	Get Ticket History	by assignee email	GSI2PK = email	PK = ticketId
15	Get employees	by manager	GSI2PK = email, SK > 3	

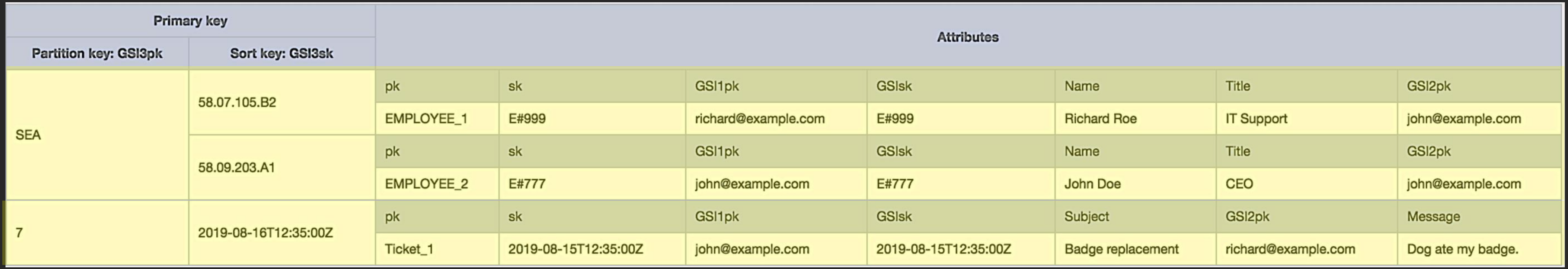


Primary key		Attributes						
Partition key: GSI2pk	Sort key: GSIsk							
john@example.com	E#777	pk	sk	GSI1pk	GSI3pk	GSI3sk	Name	Title
		EMPLOYEE_2	E#777	john@example.com	SEA	58.09.203.A1	John Doe	CEO
	E#999	pk	sk	GSI1pk	GSI3pk	GSI3sk	Name	Title
		EMPLOYEE_1	E#999	richard@example.com	SEA	58.07.105.B2	Richard Roe	IT Support
richard@example.com	2019-08-15T12:35:00Z	pk	sk	GSI1pk	Subject	GSI3pk	GSI3sk	Message
		Ticket_1	2019-08-15T12:35:00Z	john@example.com	Badge replacement	7	2019-08-16T12:35:00Z	Dog ate my badge.
	2019-08-15T12:35:05Z	pk	sk	GSI1pk	Message			
		Ticket_1	2019-08-15T12:35:05Z	john@example.com	Request received.			

The index schema (GSI3)

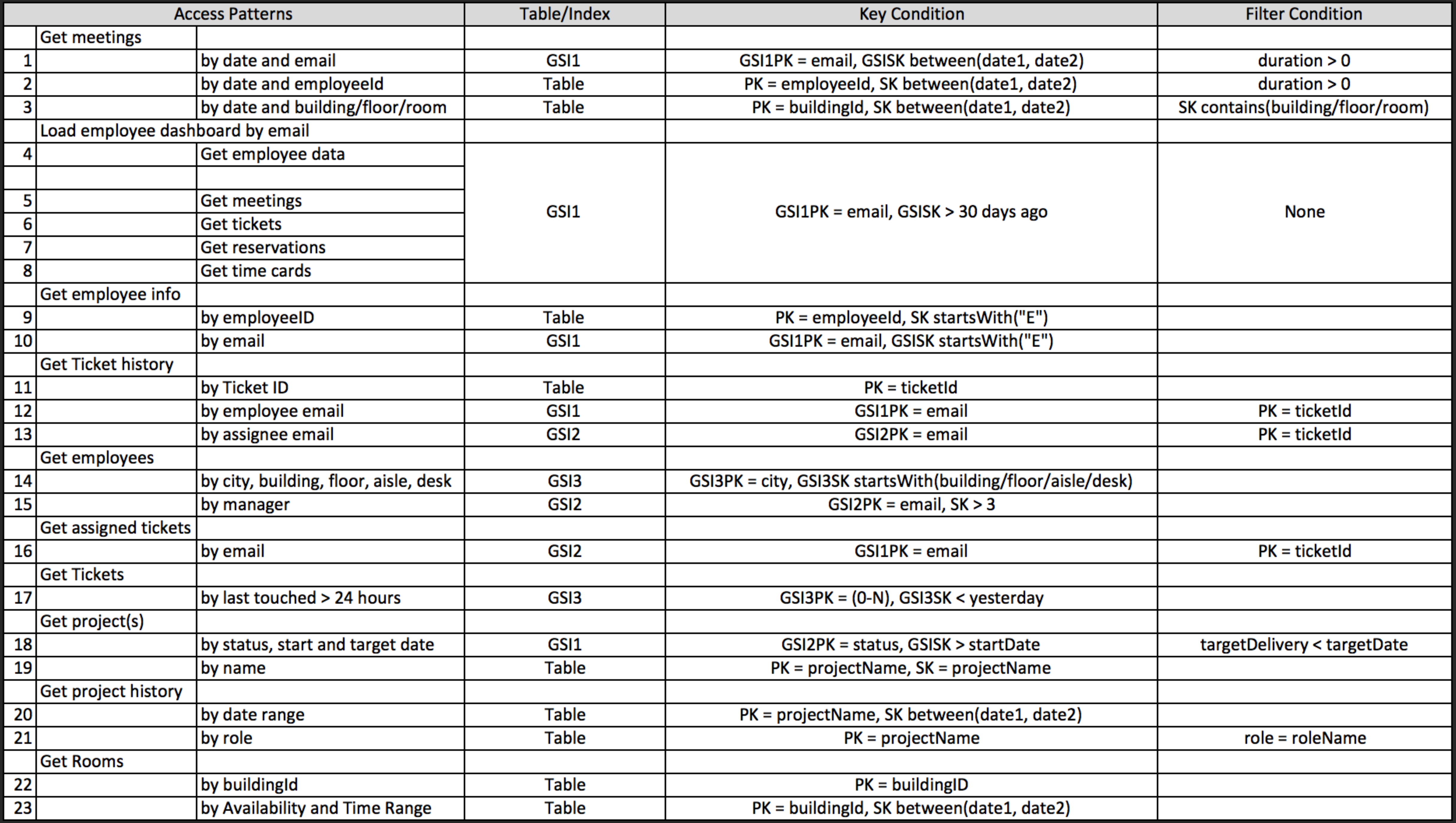


Access Patterns			Key Condition	Filter Condition
14	Get employees	by city, building, floor, aisle, desk	GSI3PK = city, GSI3SK startsWith(building/floor/aisle/desk)	
17	Get tickets	by last touched > 24 hours	GSI3PK = (0-N), GSI3SK < yesterday	



Primary key		Attributes						
Partition key: GSI3pk	Sort key: GSI3sk							
SEA	58.07.105.B2	pk	sk	GSI1pk	GSIsk	Name	Title	GSI2pk
		EMPLOYEE_1	E#999	richard@example.com	E#999	Richard Roe	IT Support	john@example.com
	58.09.203.A1	pk	sk	GSI1pk	GSIsk	Name	Title	GSI2pk
		EMPLOYEE_2	E#777	john@example.com	E#777	John Doe	CEO	john@example.com
7	2019-08-16T12:35:00Z	pk	sk	GSI1pk	GSIsk	Subject	GSI2pk	Message
		Ticket_1	2019-08-15T12:35:00Z	john@example.com	2019-08-15T12:35:00Z	Badge replacement	richard@example.com	Dog ate my badge.

The final result



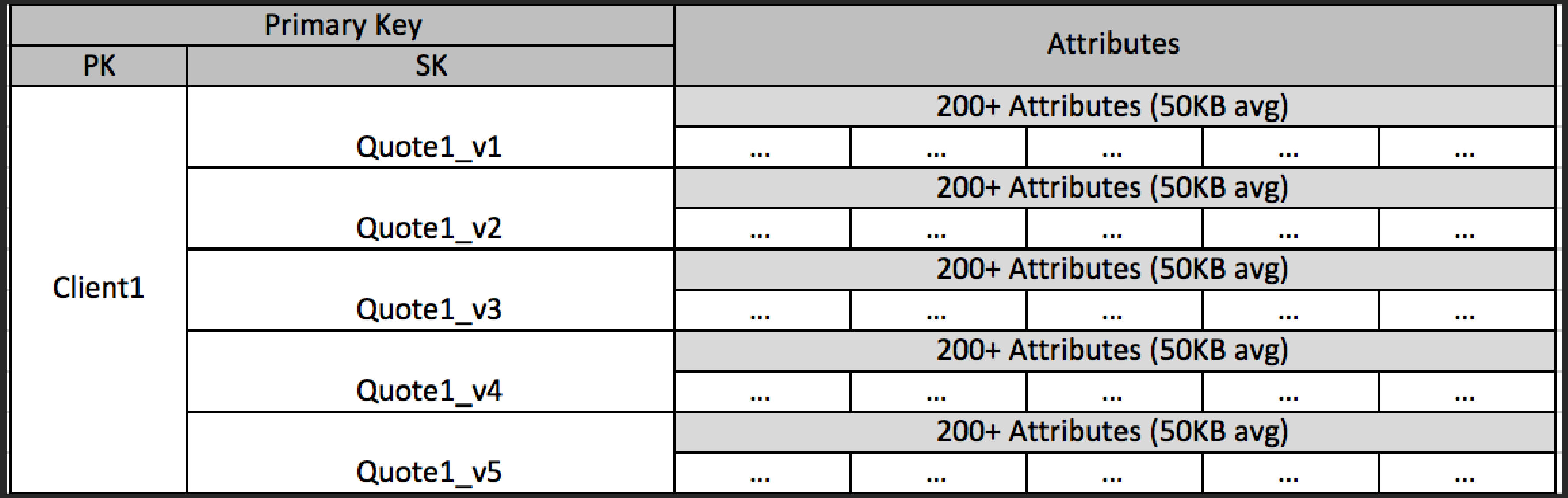
Access Patterns			Table/Index	Key Condition	Filter Condition
	Get meetings				
1		by date and email	GSI1	GSI1PK = email, GSISK between(date1, date2)	duration > 0
2		by date and employeeId	Table	PK = employeeId, SK between(date1, date2)	duration > 0
3		by date and building/floor/room	Table	PK = buildingId, SK between(date1, date2)	SK contains(building/floor/room)
	Load employee dashboard by email				
4		Get employee data	GSI1	GSI1PK = email, GSISK > 30 days ago	None
5		Get meetings			
6		Get tickets			
7		Get reservations			
8		Get time cards			
	Get employee info				
9		by employeeID	Table	PK = employeeId, SK startsWith("E")	
10		by email	GSI1	GSI1PK = email, GSISK startsWith("E")	
	Get Ticket history				
11		by Ticket ID	Table	PK = ticketId	
12		by employee email	GSI1	GSI1PK = email	PK = ticketId
13		by assignee email	GSI2	GSI2PK = email	PK = ticketId
	Get employees				
14		by city, building, floor, aisle, desk	GSI3	GSI3PK = city, GSI3SK startsWith(building/floor/aisle/desk)	
15		by manager	GSI2	GSI2PK = email, SK > 3	
	Get assigned tickets				
16		by email	GSI2	GSI1PK = email	PK = ticketId
	Get Tickets				
17		by last touched > 24 hours	GSI3	GSI3PK = (0-N), GSI3SK < yesterday	
	Get project(s)				
18		by status, start and target date	GSI1	GSI2PK = status, GSISK > startDate	targetDelivery < targetDate
19		by name	Table	PK = projectName, SK = projectName	
	Get project history				
20		by date range	Table	PK = projectName, SK between(date1, date2)	
21		by role	Table	PK = projectName	role = roleName
	Get Rooms				
22		by buildingId	Table	PK = buildingID	
23		by Availability and Time Range	Table	PK = buildingId, SK between(date1, date2)	

Design for common patterns

“To understand is to perceive patterns.”

– Isaiah Berlin

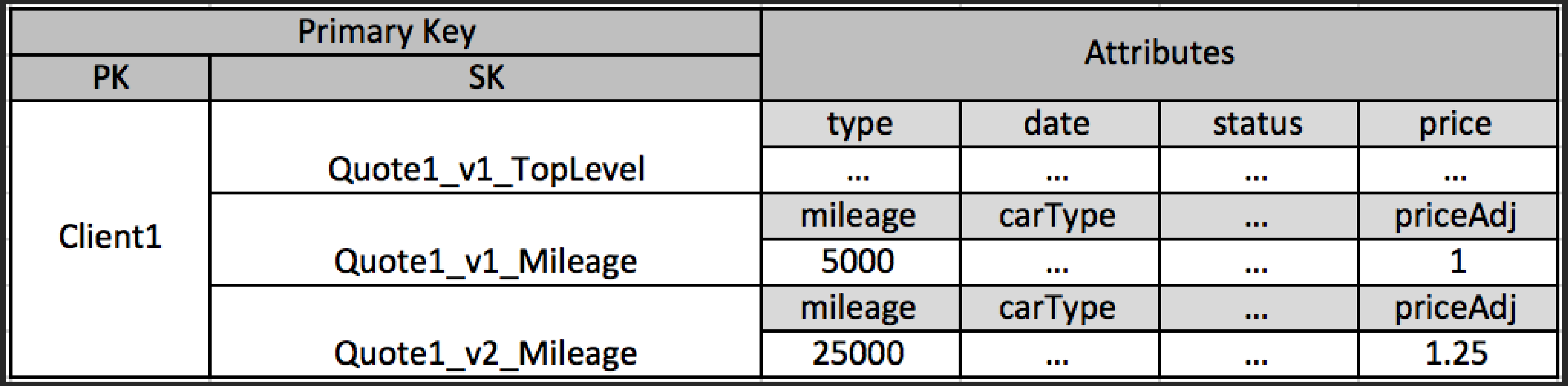
Access patterns matter



Primary Key		Attributes				
PK	SK					
Client1	Quote1_v1	200+ Attributes (50KB avg)				
		...	...	...	...	...
	Quote1_v2	200+ Attributes (50KB avg)				
		...	...	...	...	...
	Quote1_v3	200+ Attributes (50KB avg)				
		...	...	...	...	...
	Quote1_v4	200+ Attributes (50KB avg)				
		...	...	...	...	...
	Quote1_v5	200+ Attributes (50KB avg)				
		...	...	...	...	...

- Insurance quote service
- Store all versions
- 200+ attributes per quote
- 50KB average record size
- 800 quotes-per-minute peak
- 1K WCU provisioned

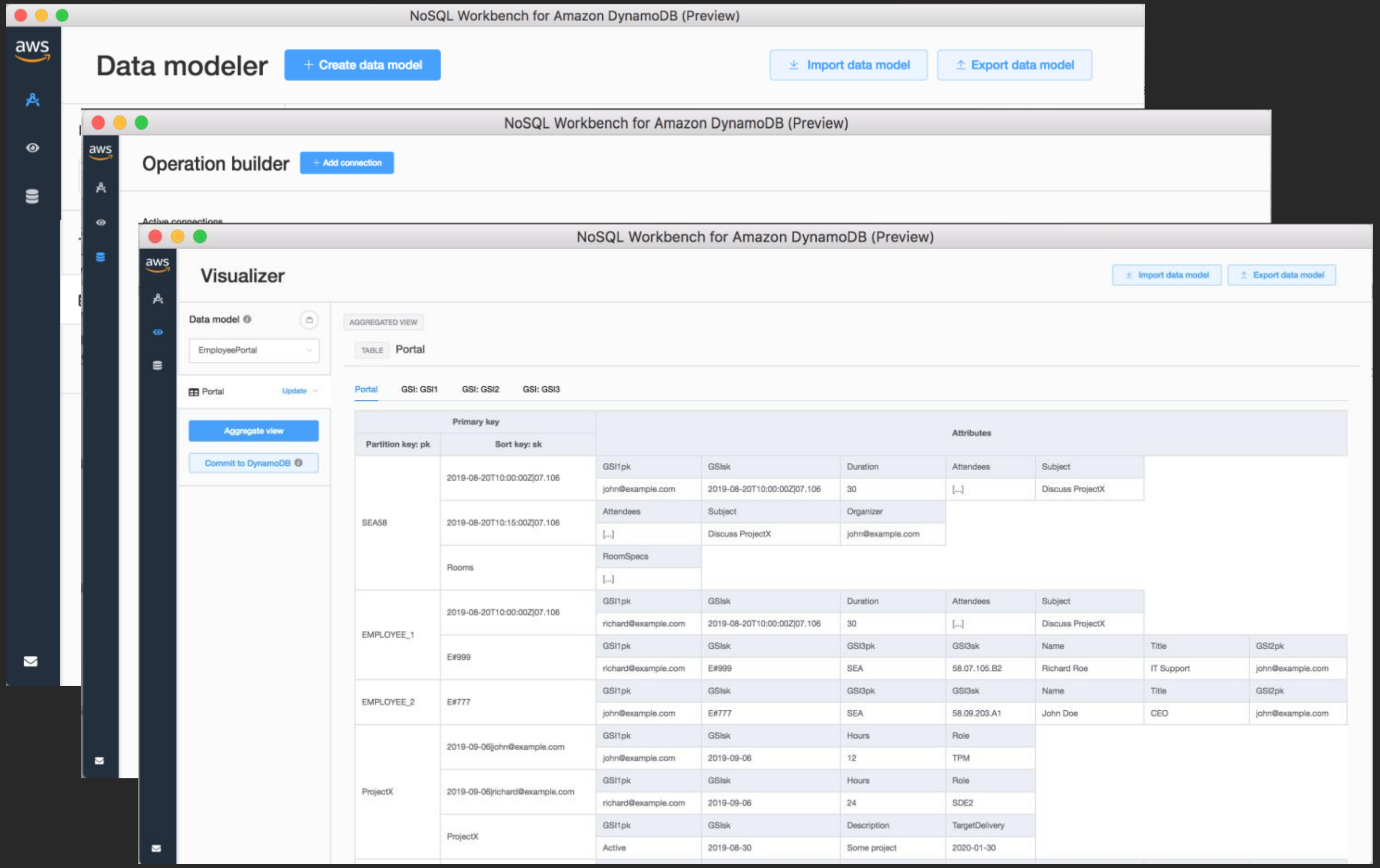
Optimized for writes



Primary Key		Attributes			
PK	SK				
Client1	Quote1_v1_TopLevel	type	date	status	price
		...	...	...	...
	Quote1_v1_Mileage	mileage	carType	...	priceAdj
		5000	...	...	1
	Quote1_v2_Mileage	mileage	carType	...	priceAdj
		25000	...	...	1.25

- Version items as categories are updated
- Send all versions when queried
- Process with client-side logic
- 50 WCU provisioned

NoSQL Workbench for DynamoDB



Primary key		Attributes					
Partition key: pk	Sort key: sk						
SEA58	2019-08-20T10:00:00Z|07.106	GSI1pk	GSIsk	Duration	Attendees	Subject	
		john@example.com	2019-08-20T10:00:00Z|07.106	30	[...]	Discuss ProjectX	
	2019-08-20T10:15:00Z|07.106	Attendees	Subject	Organizer			
		[...]	Discuss ProjectX	john@example.com			
	Rooms	RoomSpecs					
		[...]					
EMPLOYEE_1	2019-08-20T10:00:00Z|07.106	GSI1pk	GSIsk	Duration	Attendees	Subject	
		richard@example.com	2019-08-20T10:00:00Z|07.106	30	[...]	Discuss ProjectX	
	E#999	GSI1pk	GSIsk	GSI3pk	GSI3sk	Name	Title	GSI2pk
		richard@example.com	E#999	SEA	58.07.105.B2	Richard Roe	IT Support	john@example.com
EMPLOYEE_2	E#777	GSI1pk	GSIsk	GSI3pk	GSI3sk	Name	Title	GSI2pk
		john@example.com	E#777	SEA	58.09.203.A1	John Doe	CEO	john@example.com
ProjectX	2019-09-06|john@example.com	GSI1pk	GSIsk	Hours	Role		
		john@example.com	2019-09-06	12	TPM		
	2019-09-06|richard@example.com	GSI1pk	GSIsk	Hours	Role		
		richard@example.com	2019-09-06	24	SDE2		
	ProjectX	GSI1pk	GSIsk	Description	TargetDelivery		
		Active	2019-08-30	Some project	2020-01-30		

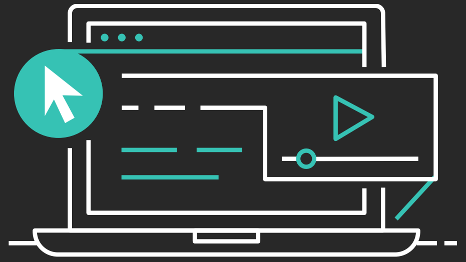
- Use the tool designed by and for the AWS specialist SA team
- Model your data, visualize your designs, generate your code
- https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/workbench.html

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

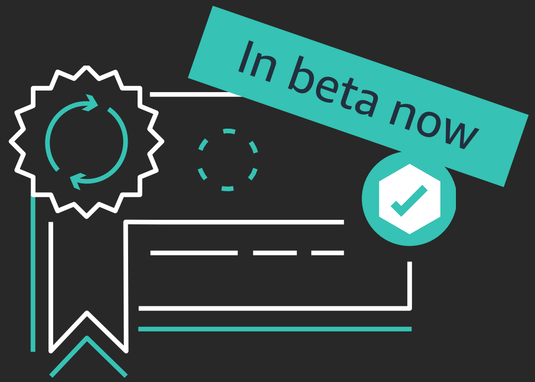
Learn databases with AWS Training and Certification

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

Thank you!

Please complete the session survey in the mobile app.

AWS re:Invent

aws