Amazon DynamoDB

Prof. Dr. Alexander Paar

Duale Hochschule Schleswig-Holstein

Amazon DynamoDB

„Amazon DynamoDB is a fully managed NoSQL database service that provides fast and predictable performance with seamless scalability.“

„With DynamoDB, you can create database tables that can store and retrieve any amount of data and serve any level of request traffic. You can scale up or scale down your tables' throughput capacity without downtime or performance degradation.“

„DynamoDB automatically spreads the data and traffic for your tables over a sufficient number of servers to handle your throughput and storage requirements, while maintaining consistent and fast performance. All of your data is stored on solid-state disks (SSDs) and is automatically replicated across multiple Availability Zones in an AWS Region, providing built-in high availability and data durability.“

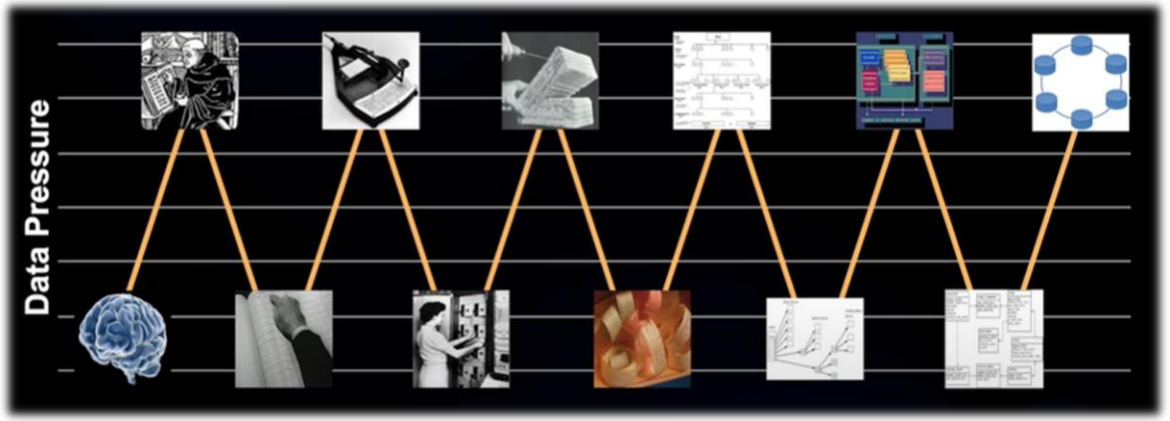
Datenbanken im Laufe der Zeit

Das relationale Datenmodell von Edgar F. "Ted" Codd ermöglicht eine Vielfalt unvorhergesehener Anfragen

Das Mooresche Gesetz hat die daraus resultierende Komplexität ($O(n^2)$) praktikabel gemacht

Aber: Seit ca. 2014 flacht die Kurve der CPU-Geschwindigkeit ab während Speicher weiter immer günstiger wird

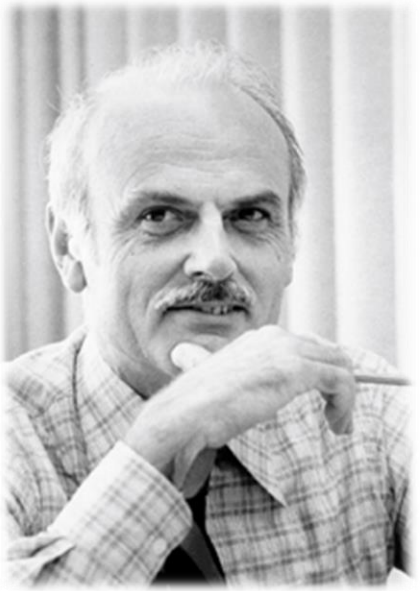
Ein generisches Datenbankschema für komplexe unvorhergesehene Anfragen reicht nicht mehr aus



Nichts Neues…

Ted Codd über Normalisierung:

„The simplicity of the array representation which becomes feasible when all relations are cast in normal form is not only an advantage for storage purposes but also for communication of bulk data between systems which use widely different representations of data.“



Edgar F. “Ted” Codd
* 19. August 1923
† 18. April 2003

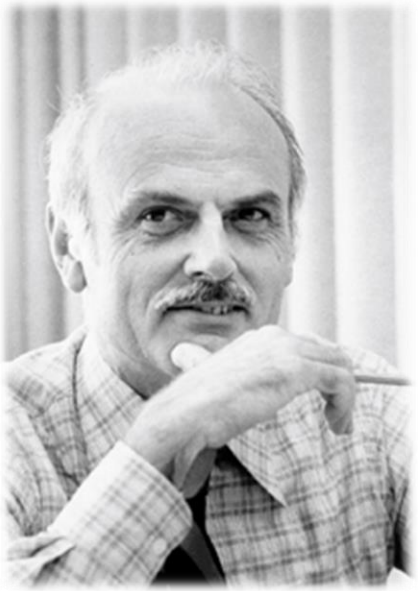
“A Relational Model of Data for Large Shared Data Banks”

Nichts Neues…

Ted Codd über Denormalisierung:

„If the strong redundancies in the named set are directly reflected in strong redundancies in the stored set (or if other strong redundancies are introduced into the stored set), then, generally speaking, extra storage space and update time are consumed with a potential drop in query time for some queries and in load on the central processing units."

→ CPU vs. Speicher!



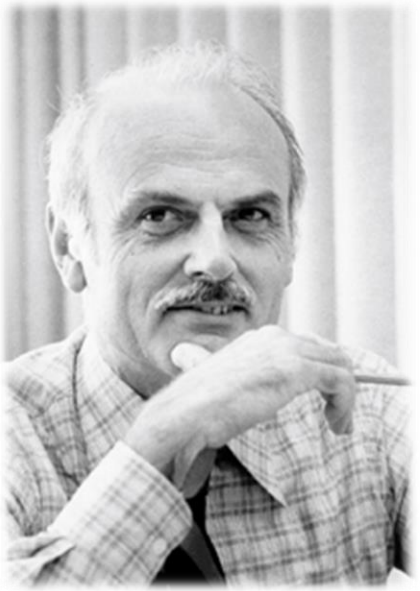
Edgar F. "Ted" Codd
* 19. August 1923
† 18. April 2003

"A Relational Model of Data for Large Shared Data Banks"

Nichts Neues…

Ted Codd über Entity-Relationship-Modelle:

„Ideally, the variety of permitted data representations should be just adequate to cover the spectrum of performance requirements of the total collection of installations. Too great a variety leads to unnecessary overhead in storage and continual reinterpretation of descriptions for the structures currently in effect.“



Edgar F. “Ted” Codd
* 19. August 1923
† 18. April 2003

“A Relational Model of Data for Large Shared Data Banks”

SQL oder NoSQL

Eine relationale Datenbank ist sinnvoll wenn die Zugriffsmuster und die Anfragen unbekannt und variabel sind

Aber: Ein Verbund (engl. join) ist teuer ($O(n^2)$)

In einer NoQL-Datenbank liegen die Objekte mit ihren relevanten Eigenschaften bereits innerhalb eines Index vor

Erforderlich ist nur ein Zugriff auf den Index

Aber: für die Struktur der Objekte und die Indizes müssen die Zugriffsmuster bekannt sein

SQL	NoSQL
Optimiert für Speicher	Optimiert für CPU
Normalisiert / relational	Denormalisiert / hierarchisch
Ad hoc-Anfragen	Instanziierte Sichten
Vertikale Skalierung	Horizontale Skalierung
Geeignet für OLAP	Geeignet für OLTP

Skalierung

DynamoDB bietet eine automatische Anpassung des Durchsatzes an das Datenaufkommen („Auto scaling“)

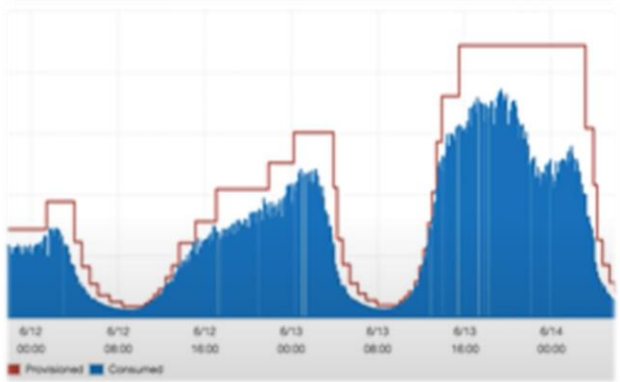
Viele betriebliche Informationssysteme sind überprovisioniert und verbrauchen nur 15 – 25% der bereitgestellten Ressourcen

Ohne automatische Skalierung ist ein Puffer für Lastspitzen erforderlich

Auch provisionierte aber nicht genutzte Ressourcen kosten Geld!

DynamoDB bietet außerdem einen „On Demand Capacity“-Modus

Etwas teurer aber voll flexibel von 0 bis ∞



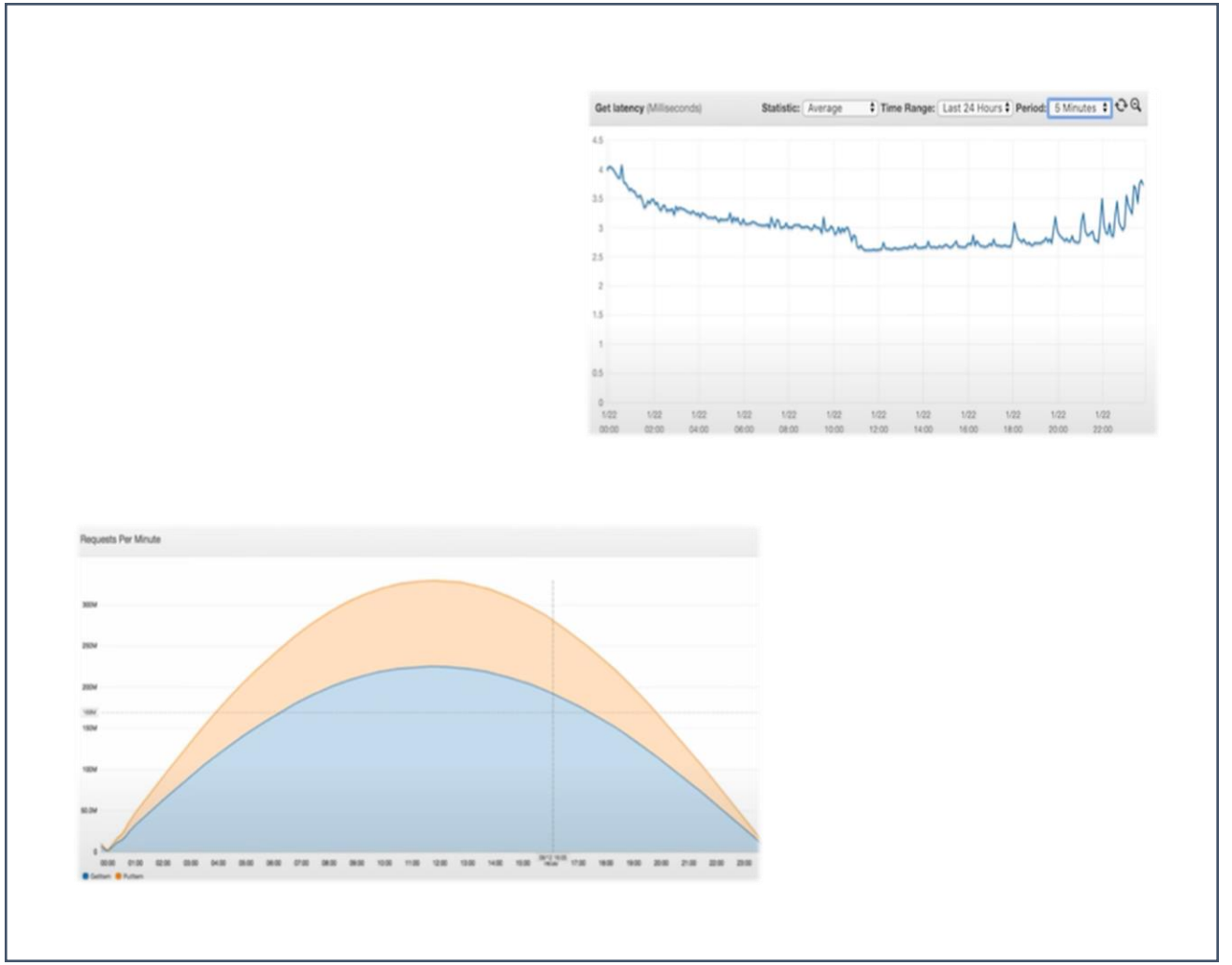
mit automatischer Skalierung

ohne automatische Skalierung

Skalierung

DynamoDB bietet eine konsistent niedrige Latenz

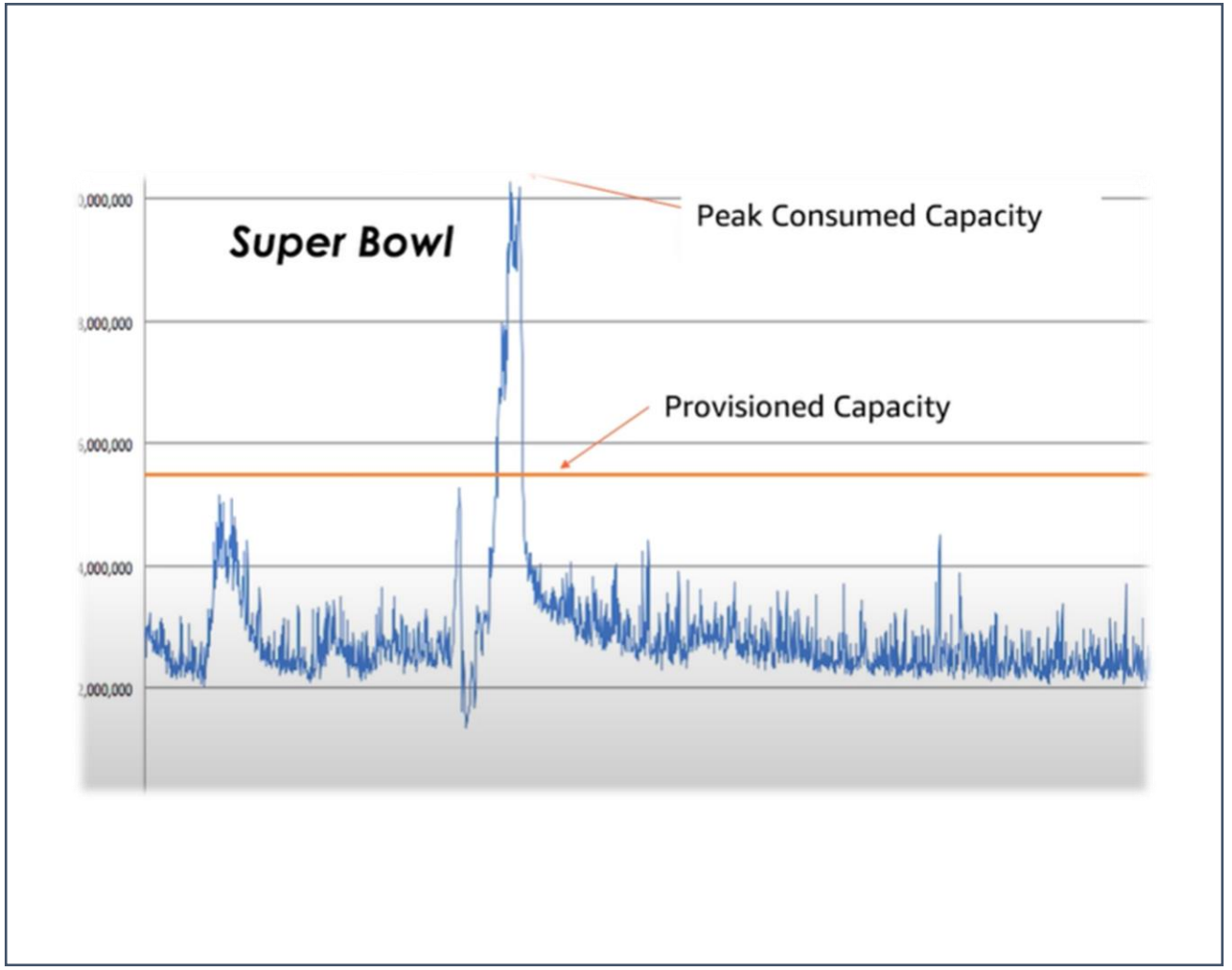
Durch warme Caches und skalierte Ressourcen kann die Latenz bei höherer Last sogar geringer sein



Skalierung

DynamoDB lädt ungenutzte Kapazität der letzten 5 Minuten in ein „Burst Bucket“ und stellt diese Kapazität bei Lastspitzen zur Verfügung und „Throttling“ zu vermeiden!

„Throttling“ wäre die Nichtbehandlung von Zugriffen mangels provisionierter Kapazität



Grundkonzepte

Tabellen (engl. tables) enthalten Items

Items (engl. items) sind Mengen von Schlüssel-Werte-Paaren, ähnlich einem Tupel in einem relationalen Datenbanksystem

Attribute (engl. attributes) sind die fundamentalen Datenelemente

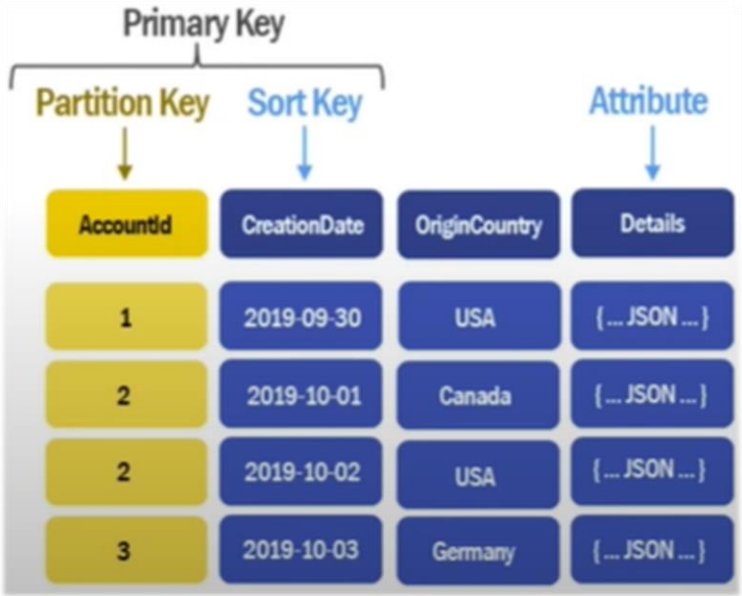
Jedes Item hat einen Primary Key

Für die Attribute eines Primary Key muss der Datentyp String, Number oder Binary sein

Für die anderen Attribute ist DynamoDB schemafrei

Ein Primary Key umfasst immer einen Partition Key

Ein Primary Key kann einen Sort Key (auch: Range Key umfassen)



https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/CheatSheet.html#CheatSheet.ServiceBasics

Grundkonzepte

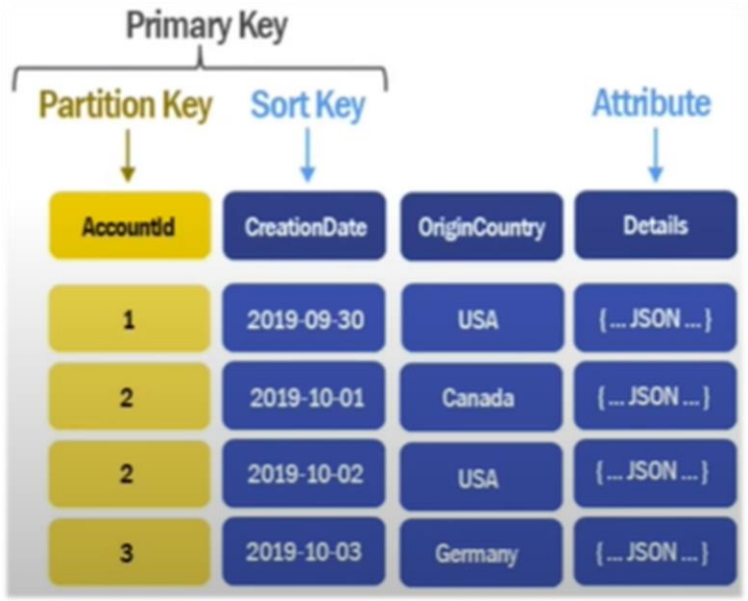
Die Wahl der Schlüssel entscheidet die physische Verteilung und Speicherung der Items und die Effizienz von Abfragen

Eine Filterung nach Partition Key und Sort Key ist effizient

Eine Filterung nach anderen Attributen erfordern einen Scan der kompletten Tabelle

Ein unpassender Schemaentwurf kann…

- zu zuvielen Leseoperationen und damit zu Throttling führen oder
- bestimmte Abfragen unmöglich machen



Partition & Sort Key

Aus dem Partition Key wird mit einer internen Hash-Funktion die Partition, wo das Item physisch gespeichert wird, berechnet

Ohne Sort Key…

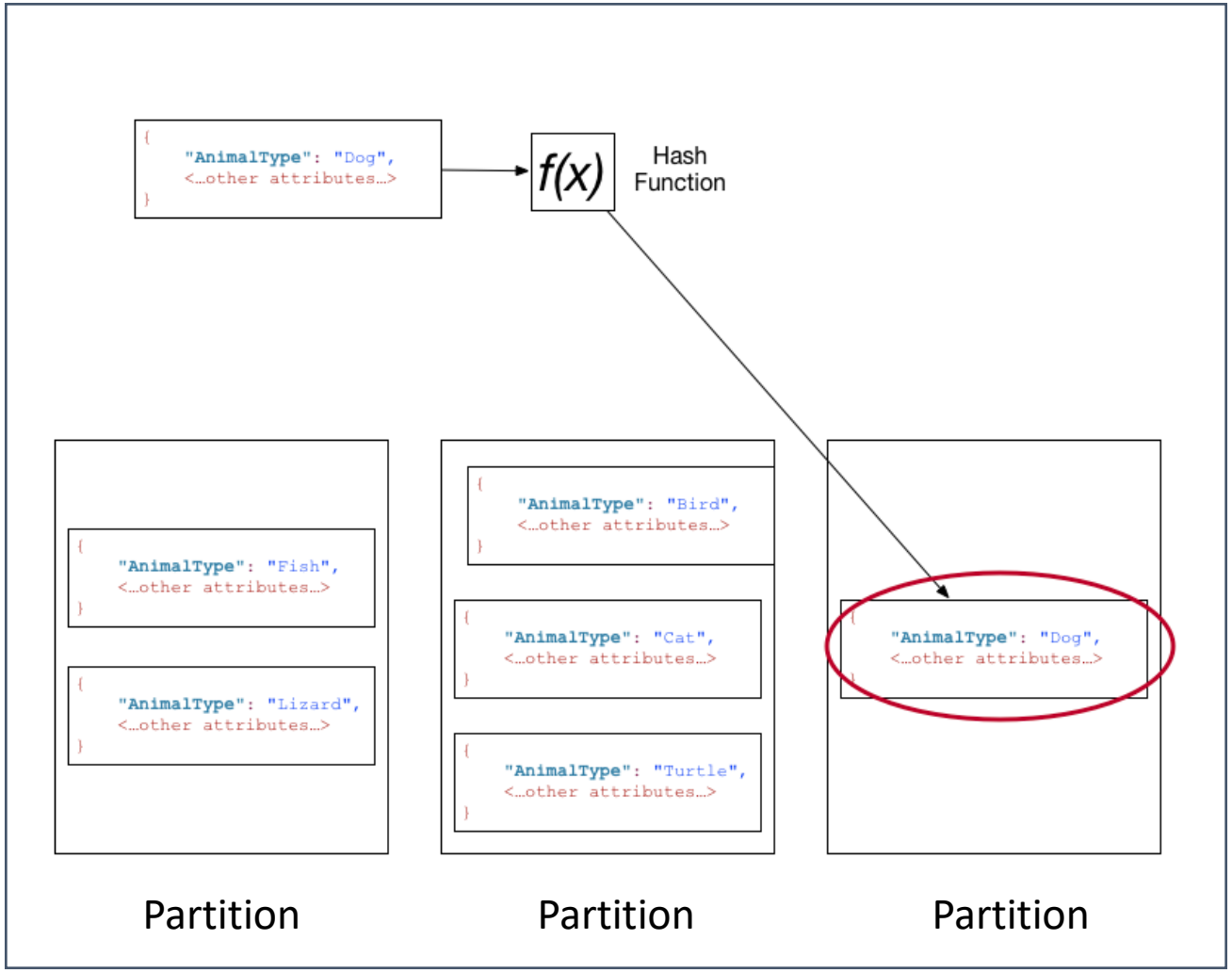
- muss der Partition Key eindeutig sein
- werden die Items innerhalb der Partition in beliebiger Reihenfolge gespeichert

Abfragen mit dem Partition Key sind direkte Lookups: DynamoDB weiß, an welchem Ort die Daten gespeichert sind

Früher waren die Capacity Units statisch auf die Partitionen verteilt

Eine Konzentration von Anfragen auf eine Partition konnte zu Throttling führen

Seit Mai 2019: Adaptive Capacity, aber hartes Limit 3000 RCUs and 1000 WCUs



Partition & Sort Key

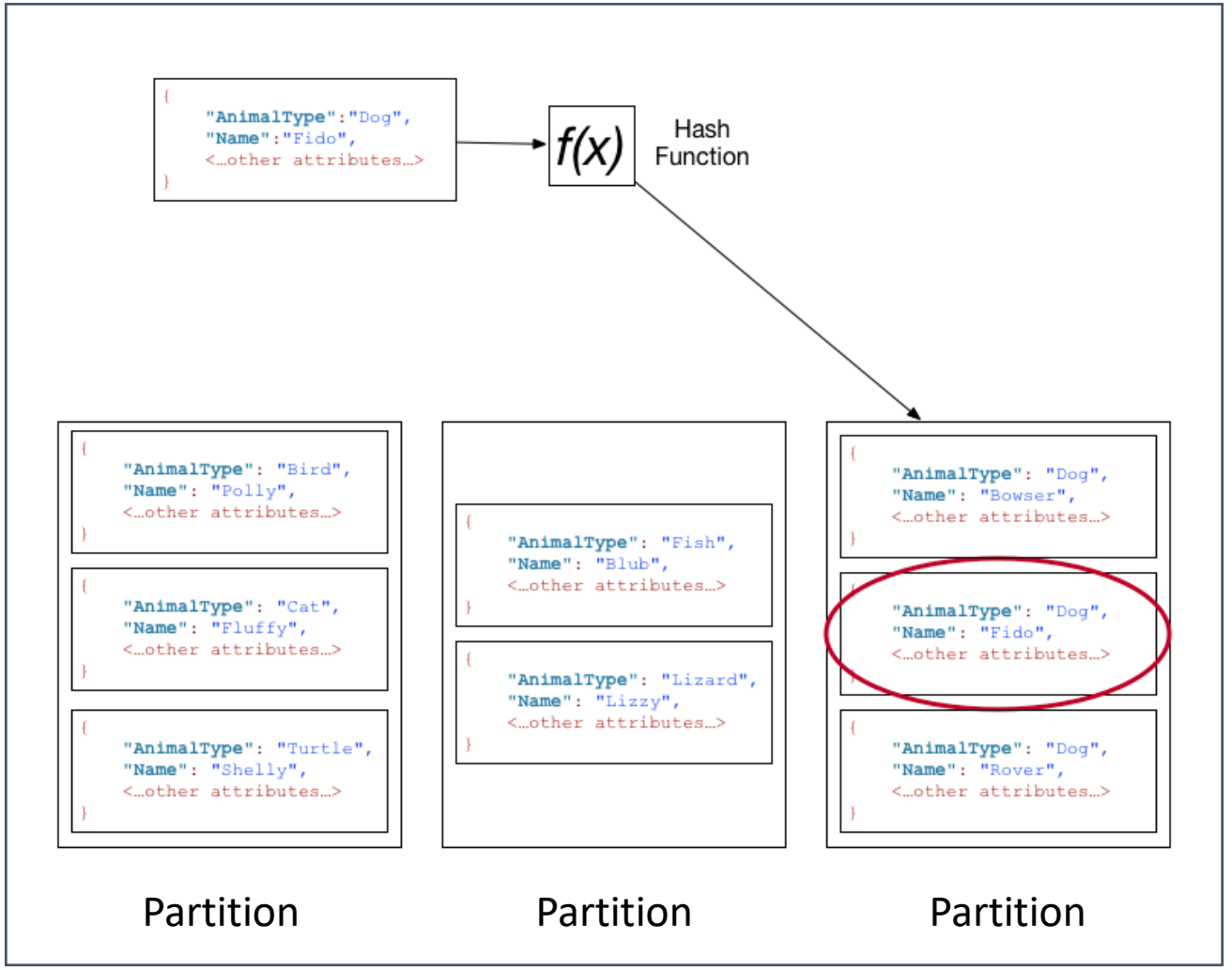
Ein Composite Primary Key umfasst einen Partition Key und einen Sort Key

Die Kombination von Partition Key und Sort Key muss eindeutig sein

Wenn ein Sort Key definiert ist, werden die Items innerhalb einer Partition nach dem Sort Key in aufsteigender Reihenfolge gespeichert

Mit einem Sort Key können effizient Items innerhalb eines Bereichs (engl. range) abgefragt werden (zum Beispiel Zeitangaben nach ISO 8601)

Eine Filter-Operation nach einem anderen Attribut würde alle Items (in der Partition) abfragen



Partition & Sort Key

Wann welcher Schlüssel?

Ein Schlüssel-Attribut ist eindeutig und es erfolgen Abfragen über dieses Attribut:

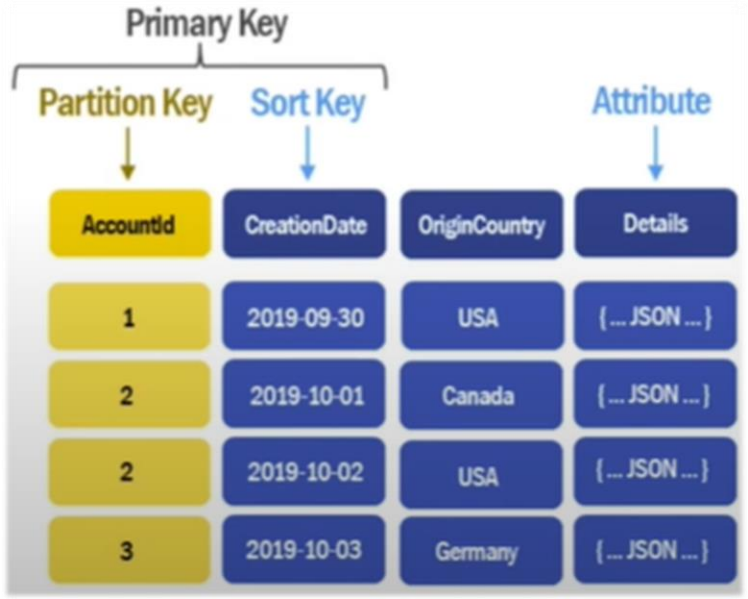
Partition Key

Das Schlüssel-Attribut ist nicht eindeutig und es sind Abfragen innerhalb eines Wertebereichs eines anderen Attributs erforderlich (engl. range queries)

Partition Key + Sort Key

Weitere Strategien:

- Zusammengesetzte Partition Keys
- Präfixe / Suffixe
- DynamoDB Accelerator (DAX), d.h. Caching mit EC2-Instanzen



Überladen von Partitionen und Indizes

In einer Partition kann ein Sort Key mit verschiedenen Werten überladen (engl. overloading) werden

Genauso kann ein sekundärer Index (engl. secondary index) diese verschiedenen Attribute oder zusammengesetzte Attribute (engl. compound keys) als Partition Key haben

Damit sind logisch mehr als 20 Global Secondary Indexes möglich

Primary Key		Data-Item Attributes...		
Partition Key	Sort Key	Attribute 1	Attribute 2	...
HR-974 *(employee ID)*	Employee_Name	Data: Murphy, John *(employee name)*	Start: 2008-11-08 *(start date)*	...etc.
	YYYY-Q1	Data: $5,477 *(order totals in USD)*	Name: Murphy, John *(employee name)*	
	HR_confidential	Data: 2008-11-08 *(hire date)*	Name: Murphy, John *(employee name)*	...etc.
	Warehouse_01	Data: Murphy, John *(employee name)*		
	v0_Job_title	Data: Operator-1 *(job title)*	Start: 2008-11-08 *(start date)*	...etc.
	v1_Job_title	Data: Operator-2 *(job title)*	Start: 2016-11-04 *(start date)*	...etc.
	v2_Job_title	Data: Supervisor-1 *(job title)*	Start: 2017-11-01 *(start date)*	...etc.

select * where PK=HR974 and SK > 2022-Q1

Global Secondary Indexes

Anwendungen können Abfragen über verschiedene Attribute erfordern:

- What is the top score ever recorded for the game Meteor Blasters?
- Which user had the highest score for Galaxy Invaders?
- What was the highest ratio of wins vs. losses?

Effiziente Abfragen (d.h. keine vollständigen Scans) sind nur über Partition Key und Sort Key möglich

→ Erzeuge eine neue Tabelle mit den benötigten Attributen als Partition Key und Sort Key

GameScores

UserId	GameTitle	TopScore	TopScoreDateTime	Wins	Losses	
"101"	"Galaxy Invaders"	5842	"2015-09-15:17:24:31"	21	72	...
"101"	"Meteor Blasters"	1000	"2015-10-22:23:18:01"	12	3	...
"101"	"Starship X"	24	"2015-08-31:13:14:21"	4	9	...
"102"	"Alien Adventure"	192	"2015-07-12:11:07:56"	32	192	...
"102"	"Galaxy Invaders"	0	"2015-09-18:07:33:42"	0	5	...
"103"	"Attack Ships"	3	"2015-10-19:01:13:24"	1	8	...
"103"	"Galaxy Invaders"	2317	"2015-09-11:06:53:00"	40	3	...
"103"	"Meteor Blasters"	723	"2015-10-19:01:13:24"	22	12	...
"103"	"Starship X"	42	"2015-07-11:06:53:00"	4	19	...
...	...	...	...	...	...	

Global Secondary Indexes

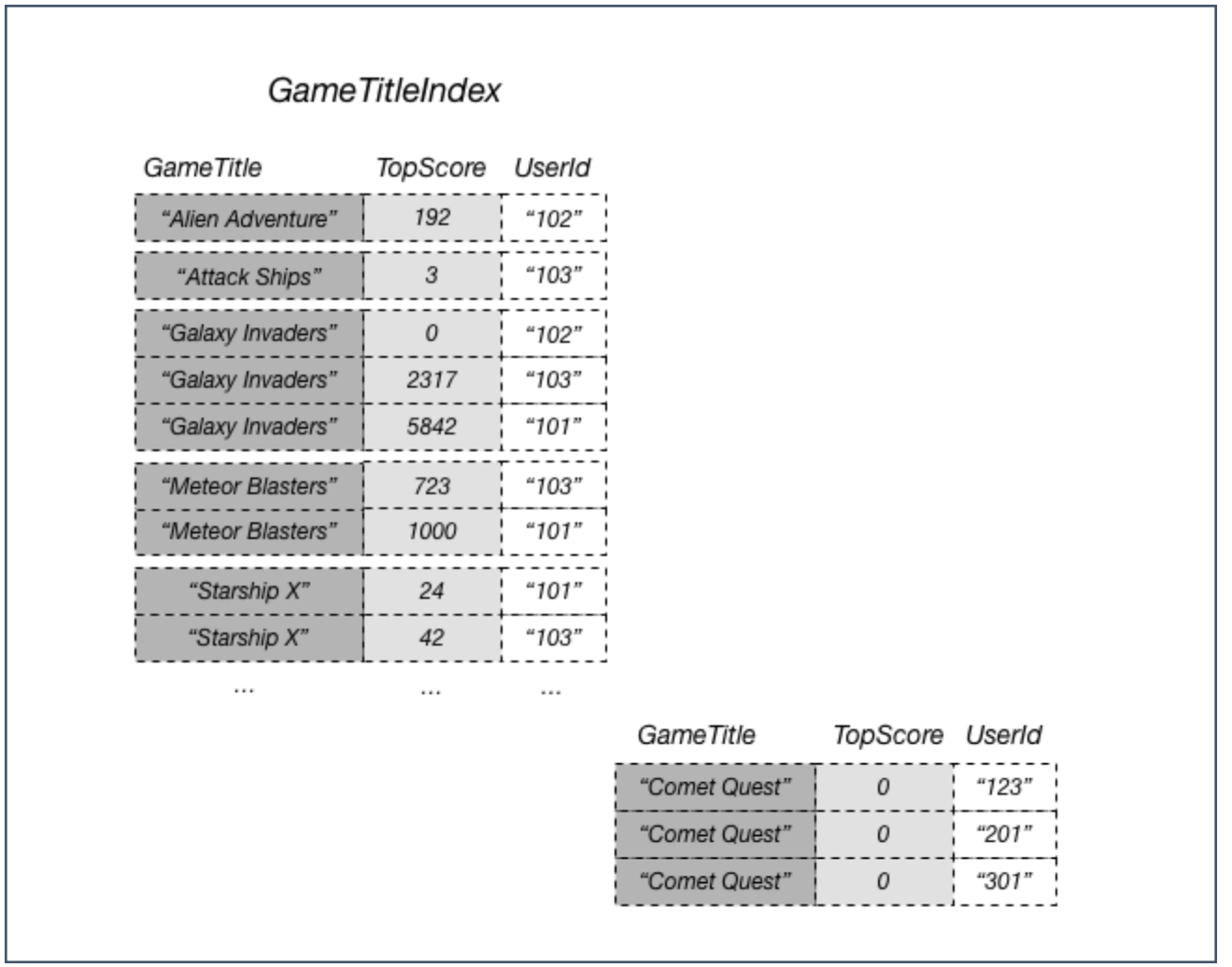
Ein Global Secondary Index…

- beschleunigt Anfragen über Nicht-Schlüssel-Attribute
- enthält eine beliebige Auswahl der Attribute der Basistabelle (engl. base table)
- hat als Attribute immer die Primärschlüsselattribute der Basistabelle
- hat ansonsten beliebige Attribute (mit zulässigem Datentyp) als Partition / Sort Key
- erfordert keine eindeutigen Schlüsselwerte

Wo sind die Joins?

NoSQL hat keine Joins, NoSQL braucht aber auch keine Joins

Ein Join wird durch einen Index ersetzt, in dem die Objekte mit ihren relevanten Eigenschaften enthalten sind



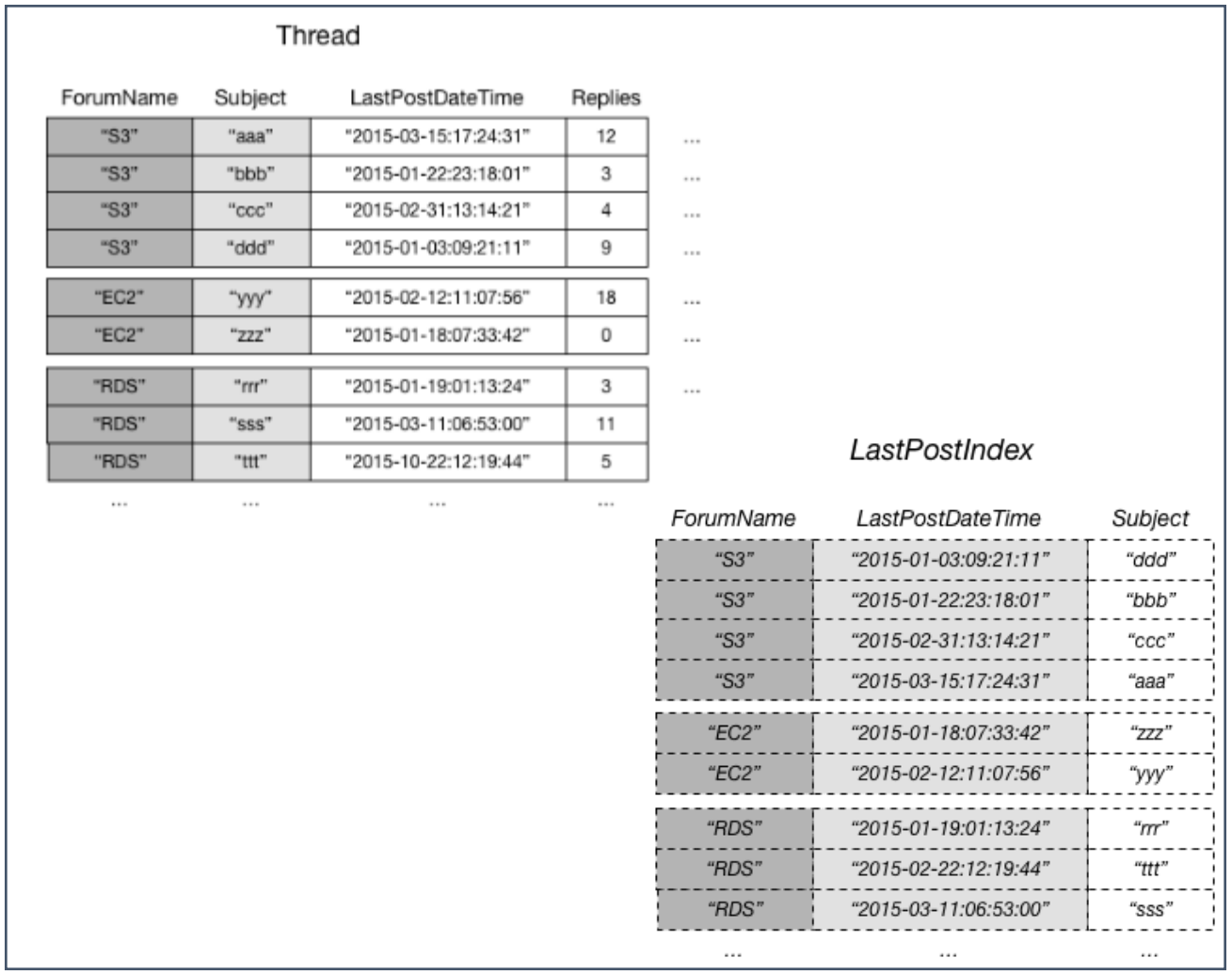
Local Secondary Indexes

Effiziente Abfragen an eine Partition können einen alternativen Sort Key erfordern

- Which forum threads get the most views and replies?
- Which thread in a particular forum has the largest number of messages?
- How many threads were posted in a particular forum within a particular time period?

Ein Local Secondary Index hält einen alternativen Sort Key für einen gegebenen Partition Key vor

Der Sort Key der Basistabelle wird immer als Attribut projiziert

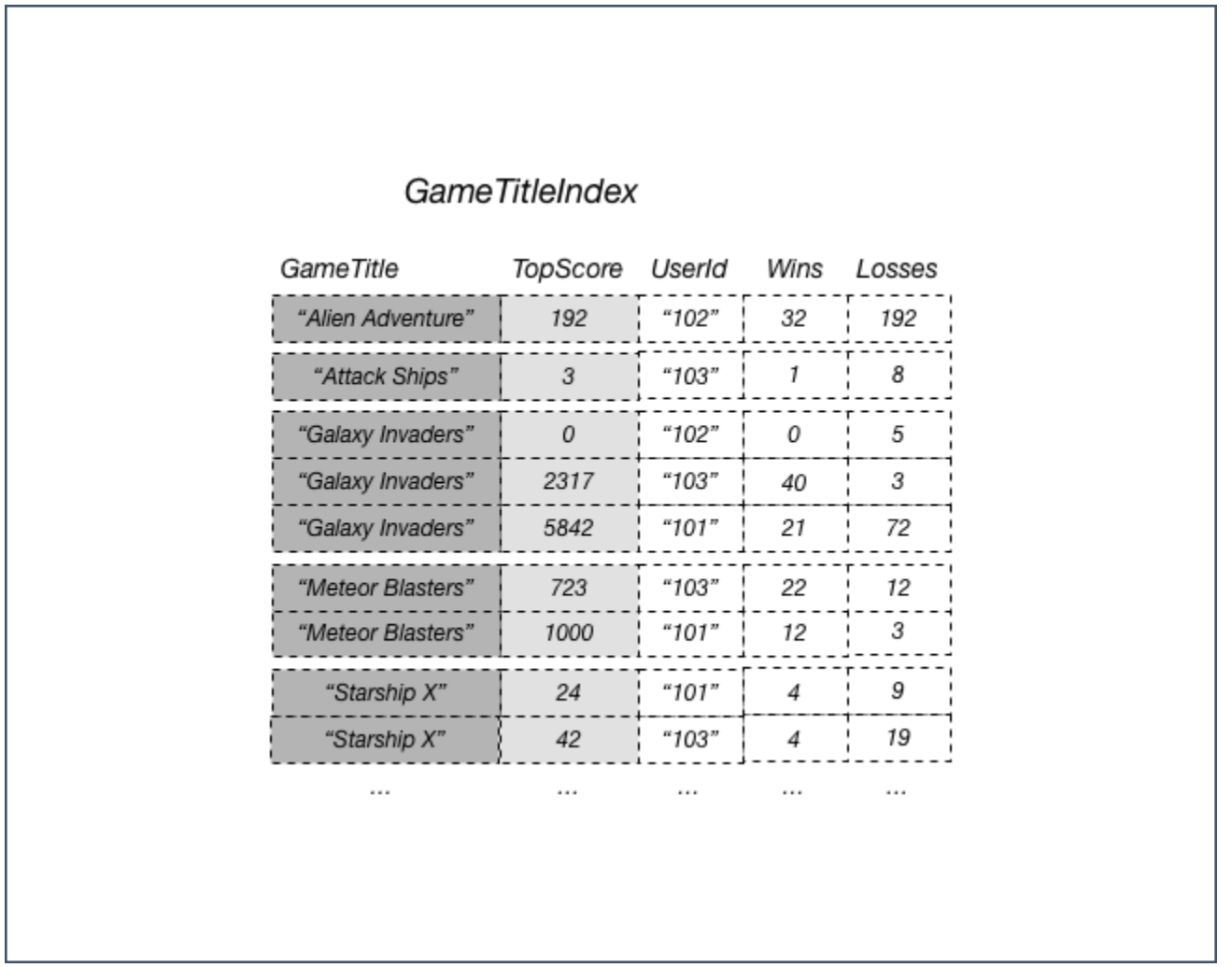


Attributprojektion

Eine Projektion (engl. projection) sind die Attribute, die von der Basistabelle in den Sekundärindex kopiert werden

DynamoDB bietet dafür drei Optionen:

- KEYS_ONLY – Der Index umfasst nur die Schlüsselattribute des Index und der Basistabelle
- INCLUDE – Der Index umfasst zusätzlich zu KEYS_ONLY weitere Attribute
- ALL – Der Index umfasst alle Attribute der Basistabelle



Attributprojektion

Bei der Auswahl der Attribute eines Index sind der Aufwand für Abfragen und der Aufwand für den Speicher abzuwägen. Siehe Amazon DynamoDB Developer Guide:

- If you need to access just a few attributes with the lowest possible latency, consider projecting only those attributes into a global secondary index. The smaller the index, the less that it costs to store it, and the less your write costs are.
- If your application frequently accesses some non-key attributes, you should consider projecting those attributes into a global secondary index. The additional storage costs for the global secondary index offset the cost of performing frequent table scans.
- If you need to access most of the non-key attributes on a frequent basis, you can project these attributes – or even the entire base table – into a global secondary index. This gives you maximum flexibility. However, your storage cost would increase, or even double.
- If your application needs to query a table infrequently but must perform many writes or updates against the data in the table, consider projecting KEYS_ONLY. The global secondary index would be of minimal size but would still be available when needed for query activity.

GameTitleIndex

GameTitle	TopScore	UserId	Wins	Losses
"Alien Adventure"	192	"102"	32	192
"Attack Ships"	3	"103"	1	8
"Galaxy Invaders"	0	"102"	0	5
"Galaxy Invaders"	2317	"103"	40	3
"Galaxy Invaders"	5842	"101"	21	72
"Meteor Blasters"	723	"103"	22	12
"Meteor Blasters"	1000	"101"	12	3
"Starship X"	24	"101"	4	9
"Starship X"	42	"103"	4	19
...	...	...	...	...

Datensynchronisation

Schreibzugriffe auf die Basistabelle werden automatisch „within a fraction of a second, under normal conditions“ mit den Indizes synchronisiert

Ein Index ist eventually consistent

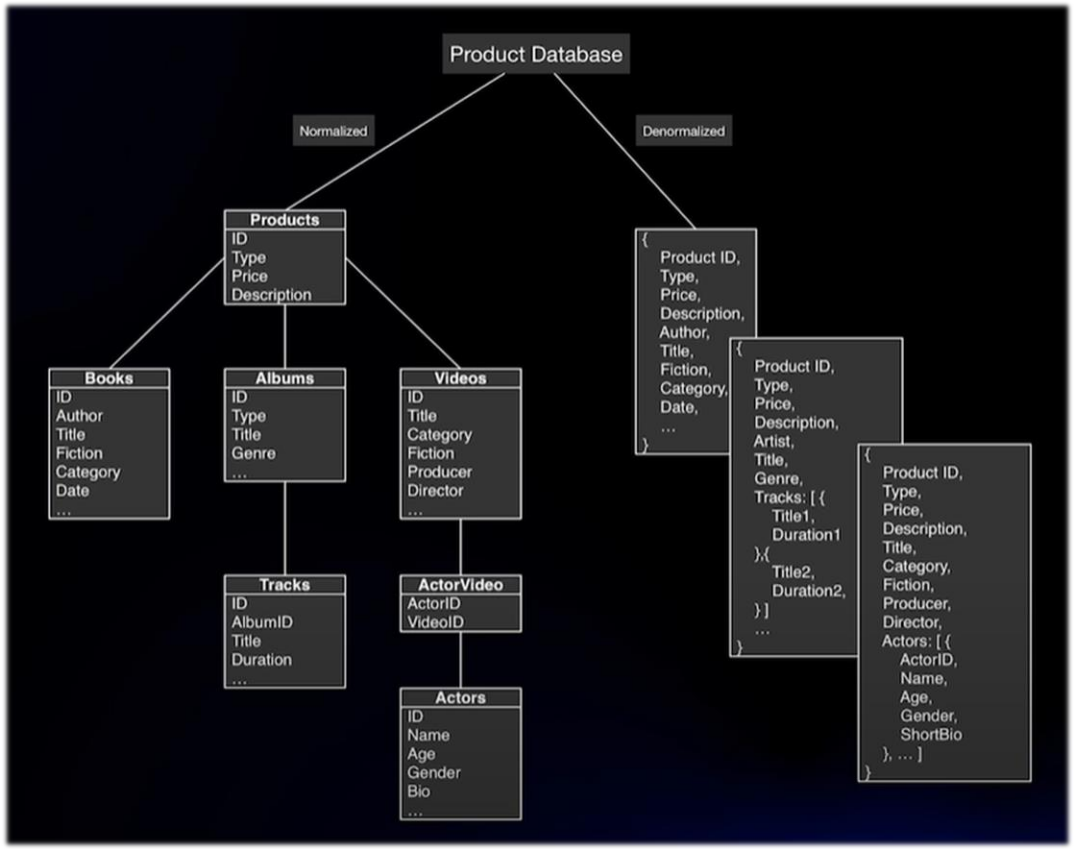
Ein Schreibzugriff auf die Basistabelle braucht Schlüsselattribute der Indizes nicht berücksichtigen

GameTitleIndex

GameTitle	TopScore	UserId	Wins	Losses
"Alien Adventure"	192	"102"	32	192
"Attack Ships"	3	"103"	1	8
"Galaxy Invaders"	0	"102"	0	5
"Galaxy Invaders"	2317	"103"	40	3
"Galaxy Invaders"	5842	"101"	21	72
"Meteor Blasters"	723	"103"	22	12
"Meteor Blasters"	1000	"101"	12	3
"Starship X"	24	"101"	4	9
"Starship X"	42	"103"	4	19
...	...	...	...	...

Relationen

Relationen spielen auch mit NoSQL-Datenbanken eine wichtige Rolle



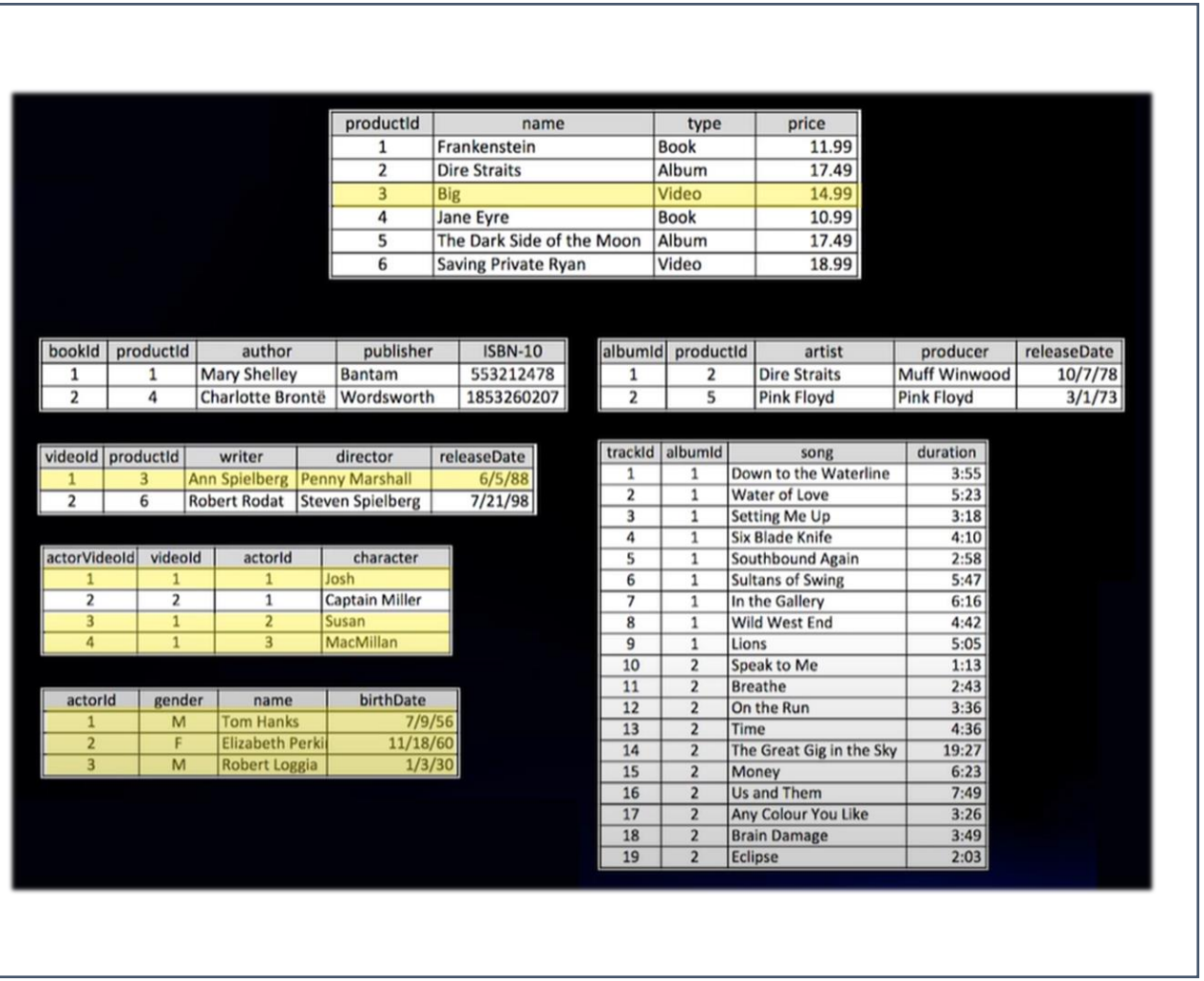
Relationen

Der Aufwand von „ad hoc“-Joins in einer relationalen Datenbank kann sehr groß sein

Beispiel

```
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"
```

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



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

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

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

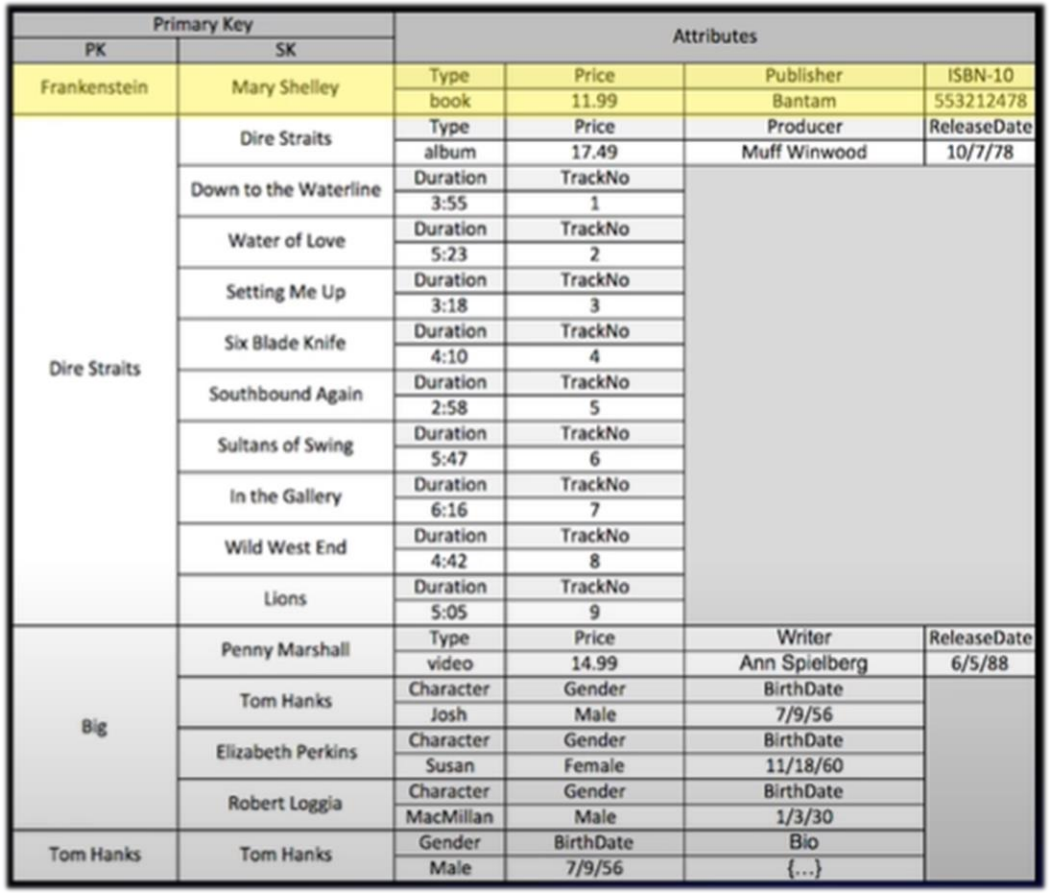
„Joins“ in NoSQL

Die Komplexität bei Abfragen einer (überladenen) Partition ist O(log(N))

Es sind hier keine Joins erforderlich, weil alle Objekte mit ihren relevanten Eigenschaften in der (überladenen) Partition vorhanden sind

Beispiel

```
select * where PK = "Book Title"
```



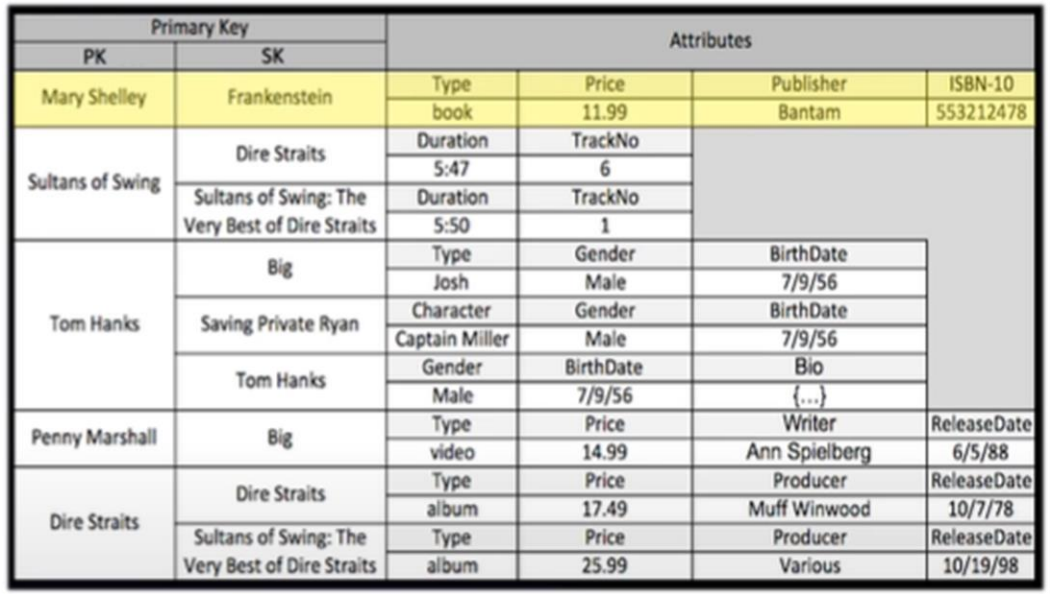
Primary Key		Attributes			
PK	SK				
Frankenstein	Mary Shelley	Type	Price	Publisher	ISBN-10
		book	11.99	Bantam	553212478
Dire Straits	Dire Straits	Type	Price	Producer	ReleaseDate
		album	17.49	Muff Winwood	10/7/78
	Down to the Waterline	Duration	TrackNo		
		3:55	1		
	Water of Love	Duration	TrackNo		
		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	{...}	

„Joins“ in NoSQL

Ein Index ermöglicht effiziente Anfragen über andere Attribute

Beispiel

select * where PK = "Author Name"



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

Aggregationen

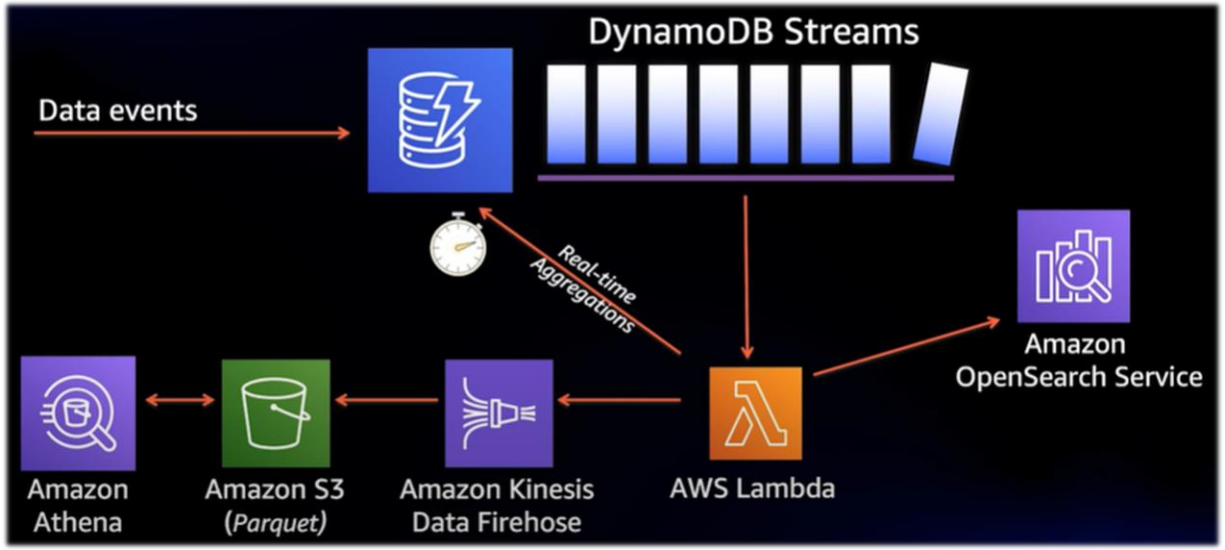
Komplexe Aggregationen können in SQL- und NoSQL-Datenbanken aufwändig sein

Idee: Anstatt Aggregationen auf Anfrage zu berechnen, halte sie schon bei ihrer Entstehung aktuell

Compute aggregations → Maintain aggregations (in realm time)

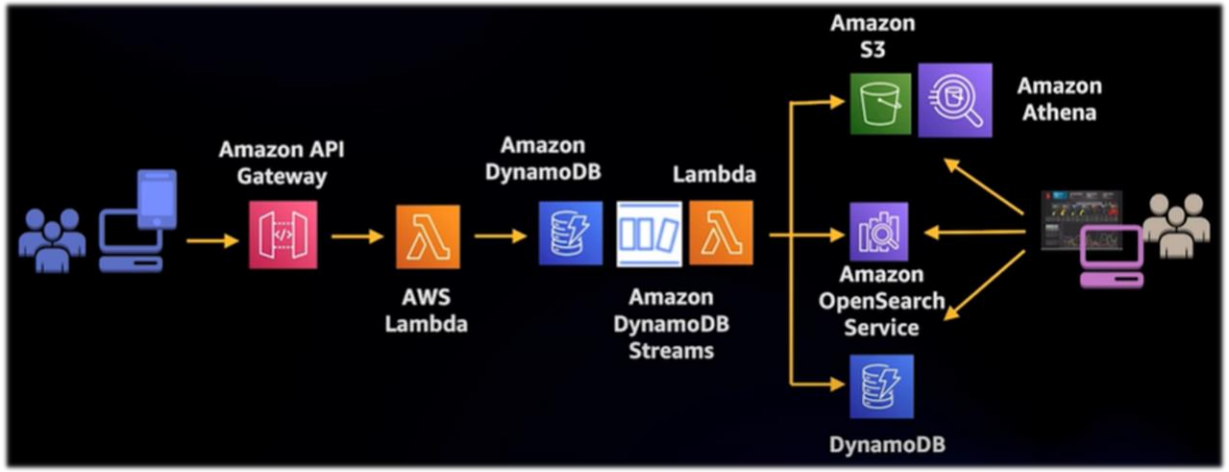
DynamoDB Streams und AWS Lambda bieten „at least once“-Auslieferung

Aber: viele Schreibzugriffe sind idempotent



Serverless Applications

Um Amazon DynamoDB kann eine umfassende serverlose Anwendung gebaut werden



Relationale Daten

In einem RDBMS werden Daten in einer normalisierten relationalen Struktur gespeichert

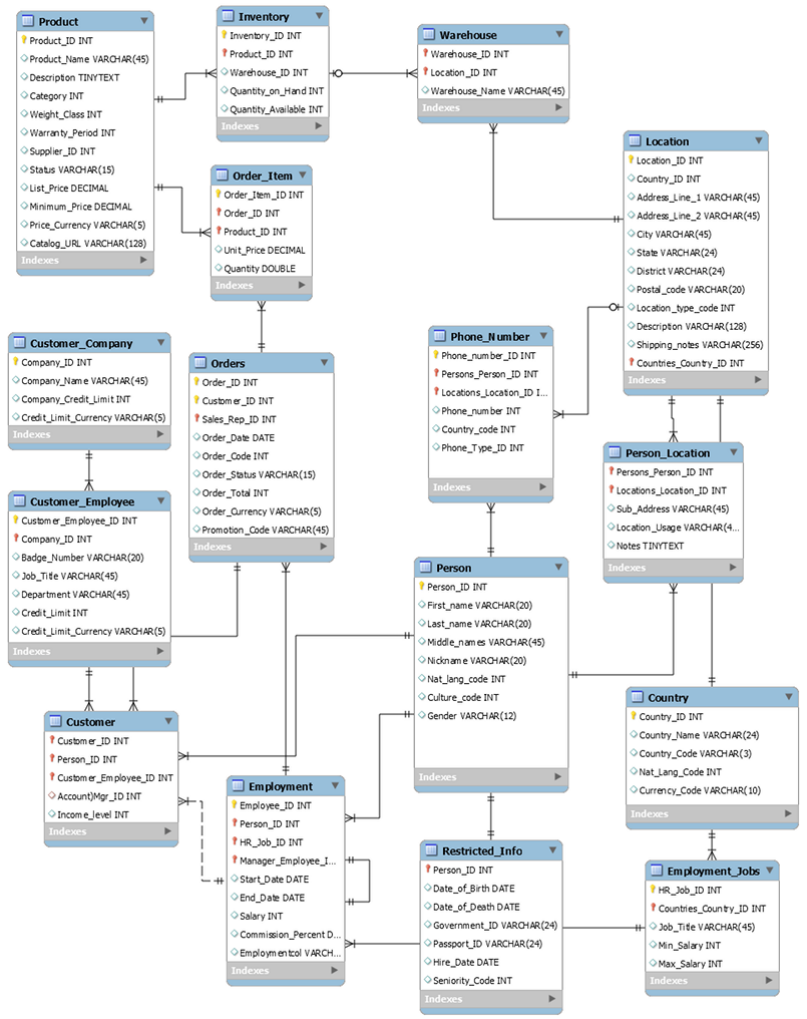
Beispiel: Auftragseingangssystem

Aus einem Entity-Relationship-Modell kann algorithmisch ein normalisiertes relationales Datenbankschema erzeugt werden

Der Entwurfsprozess für ein RDBMS ist mit dem Entity-Relationship-Modell damit praktisch abgeschlossen

Die resultierende Datenbank erlaubt mit SQL vielseitige ad hoc-Abfragen (z.B. für OLAP)

Aber: Erforderliche Join-Operationen sind aufwändig



Relationale Daten

Für den Entwurf einer DynamoDB-Basistabelle und Secondary Indexes werden die erforderlichen Zugriffsmuster (engl. access patterns) erhoben und spezifiziert

Für neue Anwendungen können zum Beispiel User Stories analysiert werden

Für bestehende Anwendungen können zum Beispiel Query Logs analysiert werden

→ Identifiziere Entitätsobjekte und denormalisiere für einfache Anfragen

Verwende…

- zusammengesetzte Schlüssel
- überladene Tabellen und Indizes

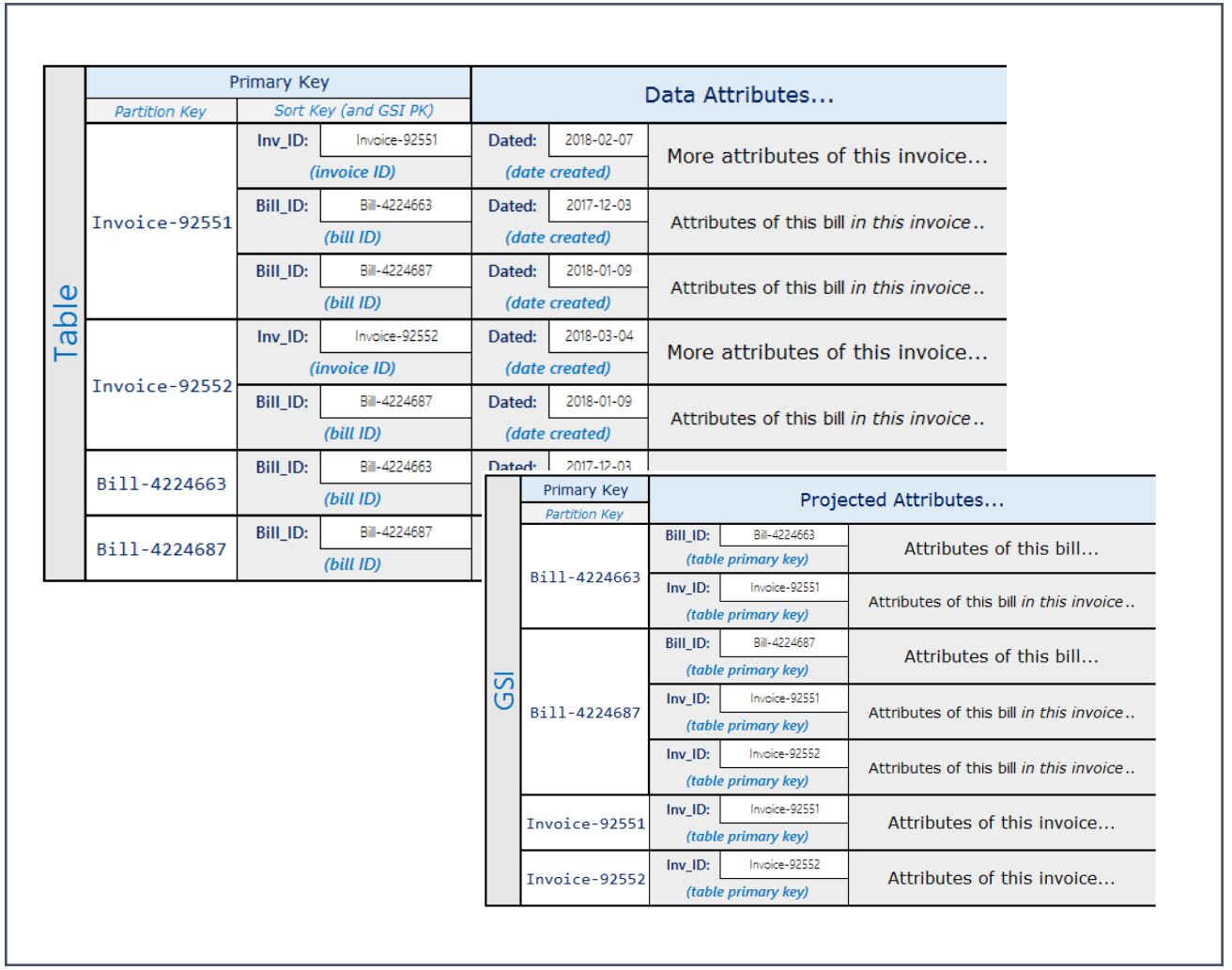
	Most Common/Import Access Patterns in Our Organization
1	Look up employee details by employee ID
2	Query employee details by employee name
3	Find an employee's phone number(s)
4	Find a customer's phone number(s)
5	Get orders for a given customer within a given date range
6	Show all open orders within a given date range across all customers
7	See all employees hired recently
8	Find all employees working in a given warehouse
9	Get all items on order for a given product
10	Get current inventories for a given product at all warehouses
11	Get customers by account representative
12	Get orders by account representative and date
13	Get all employees with a given job title
14	Get inventory by product and warehouse
15	Get total product inventory
16	Get account representatives ranked by order total and sales period

Entwurfsmuster „Adjazenzliste“

N-zu-N-Beziehungen könnten als Adjazenzliste (engl. adjacency list) modelliert werden

Alle Entitäten der obersten Ebene werden durch den Partition Key repräsentiert

Die Beziehungen mit anderen Entitäten (die adjazenten Objekte in einer Darstellung als Graph) sind Items in der Partition mit ihren IDs als Sort Keys



https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/bp-adjacency-graphs.html

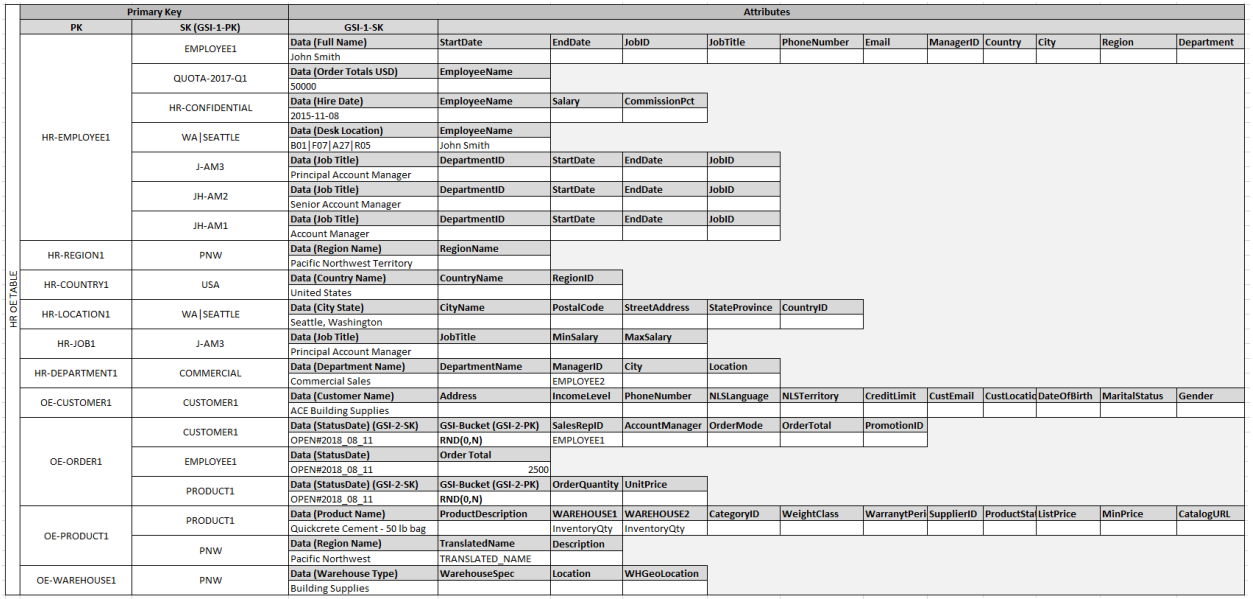
Relationale Daten

Definiere die Entitätstypen

- HR-Employee - PK: EmployeeID, SK: Employee Name
- HR-Region - PK: RegionID, SK: Region Name
- HR-Country - PK: CountryId, SK: Country Name
- HR-Location - PK: LocationID, SK: Country Name
- HR-Job - PK: JobID, SK: Job Title
- HR-Department - PK: DepartmentID, SK: DepartmentID
- OE-Customer - PK: CustomerID, SK: AccountRepID
- OE-Order - PK OrderID, SK: CustomerID
- OE-Product - PK: ProductID, SK: Product Name
- OE-Warehouse - PK: WarehouseID, SK: Region Name

Füge diese Entitäten in die Tabelle ein

Definiere die Beziehungen zu adjazenten Entitäten indem diese Entitäten den Partitionen hinzugefügt werden



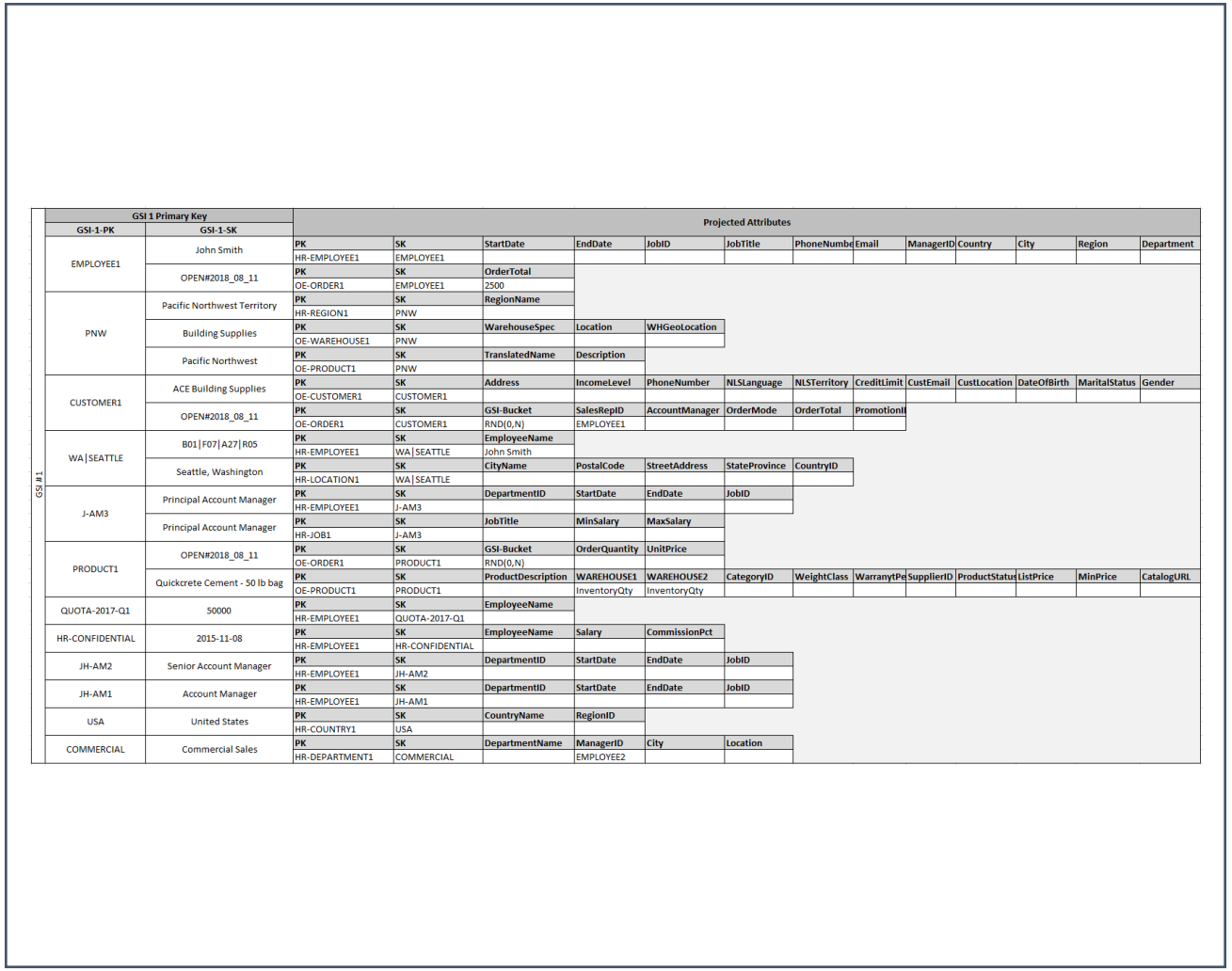
HR OE TABLE

Primary Key		Attributes											
PK	SK (GSI-1-PK)	GSI-1-SK											
HR-EMPLOYEE1	EMPLOYEE1	Data (Full Name)	StartDate	EndDate	JobID	JobTitle	PhoneNumber	Email	ManagerID	Country	City	Region	Department
		John Smith											
	QUOTA-2017-Q1	Data (Order Totals USD)	EmployeeName										
		50000											
	HR-CONFIDENTIAL	Data (Hire Date)	EmployeeName	Salary	CommissionPct								
		2015-11-08											
	WA|SEATTLE	Data (Desk Location)	EmployeeName										
		B01|F07|A27|R05	John Smith									
	J-AM3	Data (Job Title)	DepartmentID	StartDate	EndDate	JobID							
		Principal Account Manager											
	JH-AM2	Data (Job Title)	DepartmentID	StartDate	EndDate	JobID							
		Senior Account Manager											
	JH-AM1	Data (Job Title)	DepartmentID	StartDate	EndDate	JobID							
		Account Manager											
HR-REGION1	PNW	Data (Region Name)	RegionName										
		Pacific Northwest Territory											
HR-COUNTRY1	USA	Data (Country Name)	CountryName	RegionID									
		United States											
HR-LOCATION1	WA|SEATTLE	Data (City State)	CityName	PostalCode	StreetAddress	StateProvince	CountryID						
		Seattle, Washington											
HR-JOB1	J-AM3	Data (Job Title)	JobTitle	MinSalary	MaxSalary								
		Principal Account Manager											
HR-DEPARTMENT1	COMMERCIAL	Data (Department Name)	DepartmentName	ManagerID	City	Location							
		Commercial Sales		EMPLOYEE2									
OE-CUSTOMER1	CUSTOMER1	Data (Customer Name)	Address	IncomeLevel	PhoneNumber	NLSLanguage	NLSTerritory	CreditLimit	CustEmail	CustLocatic	DateOfBirth	MaritalStatus	Gender
		ACE Building Supplies											
OE-ORDER1	CUSTOMER1	Data (StatusDate) (GSI-2-SK)	GSI-Bucket (GSI-2-PK)	SalesRepID	AccountManager	OrderMode	OrderTotal	PromotionID					
		OPEN#2018_08_11	RND(0,N)	EMPLOYEE1									
	EMPLOYEE1	Data (StatusDate)	Order Total										
		OPEN#2018_08_11	2500										
	PRODUCT1	Data (StatusDate) (GSI-2-SK)	GSI-Bucket (GSI-2-PK)	OrderQuantity	UnitPrice								
		OPEN#2018_08_11	RND(0,N)										
OE-PRODUCT1	PRODUCT1	Data (Product Name)	ProductDescription	WAREHOUSE1	WAREHOUSE2	CategoryID	WeightClass	WarranytPeri	SupplierID	ProductSta	ListPrice	MinPrice	CatalogURL
		Quickcrete Cement - 50 lb bag		InventoryQty	InventoryQty								
	PNW	Data (Region Name)	TranslatedName	Description									
		Pacific Northwest	TRANSLATED_NAME										
OE-WAREHOUSE1	PNW	Data (Warehouse Type)	WarehouseSpec	Location	WHGeoLocation								
		Building Supplies											

Relationale Daten

Definiere für weitere Zugriffsmuster einen Global Secondary Index (GSI)

Ein Global Secondary Index ermöglicht verschiedene Anfragen wenn die Entitäten unterschiedliche Schlüsselwerte haben (d.h. Global Secondary Index overloading)



GSI #1

GSI 1 Primary Key		Projected Attributes												
GSI-1-PK	GSI-1-SK													
EMPLOYEE1	John Smith	PK	SK	StartDate	EndDate	JobID	JobTitle	PhoneNumbe	Email	ManagerID	Country	City	Region	Department
		HR-EMPLOYEE1	EMPLOYEE1											
	OPEN#2018_08_11	PK	SK	OrderTotal										
		OE-ORDER1	EMPLOYEE1	2500										
PNW	Pacific Northwest Territory	PK	SK	RegionName										
		HR-REGION1	PNW											
	Building Supplies	PK	SK	WarehouseSpec	Location	WHGeoLocation								
		OE-WAREHOUSE1	PNW											
	Pacific Northwest	PK	SK	TranslatedName	Description									
		OE-PRODUCT1	PNW											
CUSTOMER1	ACE Building Supplies	PK	SK	Address	IncomeLevel	PhoneNumber	NLSLanguage	NLSTerritory	CreditLimit	CustEmail	CustLocation	DateOfBirth	MaritalStatus	Gender
		OE-CUSTOMER1	CUSTOMER1											
	OPEN#2018_08_11	PK	SK	GSI-Bucket	SalesRepID	AccountManager	OrderMode	OrderTotal	PromotionII					
		OE-ORDER1	CUSTOMER1	RND(0,N)	EMPLOYEE1									
WA|SEATTLE	B01|F07|A27|R05	PK	SK	EmployeeName										
		HR-EMPLOYEE1	WA|SEATTLE	John Smith										
	Seattle, Washington	PK	SK	CityName	PostalCode	StreetAddress	StateProvince	CountryID						
		HR-LOCATION1	WA|SEATTLE											
J-AM3	Principal Account Manager	PK	SK	DepartmentID	StartDate	EndDate	JobID							
		HR-EMPLOYEE1	J-AM3											
	Principal Account Manager	PK	SK	JobTitle	MinSalary	MaxSalary								
		HR-JOB1	J-AM3											
PRODUCT1	OPEN#2018_08_11	PK	SK	GSI-Bucket	OrderQuantity	UnitPrice								
		OE-ORDER1	PRODUCT1	RND(0,N)										
	Quickcrete Cement - 50 lb bag	PK	SK	ProductDescription	WAREHOUSE1	WAREHOUSE2	CategoryID	WeightClass	WarranytPe	SupplierID	ProductStatus	ListPrice	MinPrice	CatalogURL
		OE-PRODUCT1	PRODUCT1		InventoryQty	InventoryQty								
QUOTA-2017-Q1	50000	PK	SK	EmployeeName										
		HR-EMPLOYEE1	QUOTA-2017-Q1											
HR-CONFIDENTIAL	2015-11-08	PK	SK	EmployeeName	Salary	CommissionPct								
		HR-EMPLOYEE1	HR-CONFIDENTIAL											
JH-AM2	Senior Account Manager	PK	SK	DepartmentID	StartDate	EndDate	JobID							
		HR-EMPLOYEE1	JH-AM2											
JH-AM1	Account Manager	PK	SK	DepartmentID	StartDate	EndDate	JobID							
		HR-EMPLOYEE1	JH-AM1											
USA	United States	PK	SK	CountryName	RegionID									
		HR-COUNTRY1	USA											
COMMERCIAL	Commercial Sales	PK	SK	DepartmentName	ManagerID	City	Location							
		HR-DEPARTMENT1	COMMERCIAL		EMPLOYEE2									

Relationale Daten

Definiere einen weiteren GSI, um alle Aufträge (d.h. Orders) in einem bestimmten Zustand (z.B. OPEN) abzufragen

Viele Aufträgen können denselben Zustand haben. Das würde zu einer „hot partition“ führen.

→ Randomisiere den Partition Key, damit Aufträge über die Partitionen verteilt werden (engl. write sharding)

Verwende eine Zufallszahl zwischen 0 – N:

1. ItemsPerRCU = 4 KB / AvgItemSize
2. PartitionMaxReadRate = 3 K x ItemsPerRCU
3. N = MaxRequiredIO / PartitionMaxReadRate

GSI #2

GSI 2 Primary Key		Projected Attributes						
GSI-2-PK	GSI-2-SK							
[0..N]	OPEN#2018_08_11	PK	SK	SalesRepID	AccountManager	OrderMode	OrderTotal	PromotionID
		OE-ORDER1	CUSTOMER1	EMPLOYEE2				
[0..N]	OPEN#2018_08_11	PK	SK	OrderQuantity	UnitPrice			
		OE-ORDER1	PRODUCT1					

Beispiel:

- Up to 2 million orders will be in the system, growing to 3 million in 5 years.
- Up to 20 percent of these orders will be in an OPEN state at any given time.
- The average order record is around 100 bytes, with three OrderItem records in the order partition that are around 50 bytes each, giving you an average order entity size of 250 bytes.

1. ItemsPerRCU = 4 KB / 250 B = 16
2. PartitionMaxReadRate = 3 K * 16 = 48 K
3. N = (0.2 x 3 M) / 48 K = 13

Relationale Daten

Die Basistabelle und die beiden Global Secondary Indexes ermöglichen effiziente Abfragen der DynamoDB

	Access patterns	Query conditions
1	Look up Employee Details by Employee ID	Primary Key on table, ID="HR-EMPLOYEE"
2	Query Employee Details by Employee Name	Use GSI-1, PK="Employee Name"
3	Get an employee's current job details only	Primary Key on table, PK=HR-EMPLOYEE-1, SK starts with "JH"
4	Get Orders for a customer for a date range	Use GSI-1, PK=CUSTOMER1, SK="STATUS-DATE", for each StatusCode
5	Show all Orders in OPEN status for a date range across all customers	Use GSI-2, PK=query in parallel for the range [0..N], SK between OPEN-Date1 and OPEN-Date2
6	All Employees hired recently	Use GSI-1, PK="HR-CONFIDENTIAL', SK > date1
7	Find all Employees in specific Warehouse	Use GSI-1, PK=WAREHOUSE1
8	Get all Orderitems for a Product including warehouse location inventories	Use GSI-1, PK=PRODUCT1
9	Get customers by Account Rep	Use GSI-1, PK=ACCOUNT-REP
10	Get orders by Account Rep and date	Use GSI-1, PK=ACCOUNT-REP, SK="STATUS-DATE", for each StatusCode
11	Get all employees with specific Job Title	Use GSI-1, PK=JOBTITLE
12	Get inventory by Product and Warehouse	Primary Key on table, PK=OE-PRODUCT1,SK=PRODUCT1
13	Get total product inventory	Primary Key on table, PK=OE-PRODUCT1,SK=PRODUCT1
14	Get Account Reps ranked by Order Total and Sales Period	Use GSI-1, PK=YYYY-Q1, scanIndexForward=False

PartiQL

Eine SQL-kompatible Abfragesprache

PartiQL

PartiQL ist eine zu SQL-92 rückwärtskompatible Anfragesprache für…

- strukturierte,
- semistrukturierte,
- geschaltete und
- Graph-Daten (experimentell)

PartiQL Core ist eine funktionale Programmiersprache

PartiQL ist „syntactic Sugar“, der SQL auf PartiQL abbildet

PartiQL DML ist noch experimentell

```
value            = absent_value
                 | scalar_value
                 | tuple_value
                 | collection_value ;

absent_value     = "NULL"
                 | "MISSING" ;

scalar_value     = "`", ? ion literal ?, "`"
                 | ? sql literal ? ;

tuple_value       = "{" [ string_value, ":", value { ",",
string_value, ":", value }] "}" ; (1)

collection_value = array_value
                  | bag_value ;

array_value       = "[", [ value , { "," , value } ], "]" ;

bag_value         = "<<", [ value , { "," , value } ], ">>" ;
```

EBNF-Grammatik für PartiQL-Werte

https://partiql.org/

DH DUALE
SH HOCHSCHULE SH

Bildnachweis

https://en.wikipedia.org/wiki/Amazon_DynamoDB#/media/File:DynamoDB.png

Rick Houlihan, AWS re:Invent 2021 – Dynamo DB deep dive: Advanced design patterns

https://youtu.be/2k2GINpO308?si=4QMJXbW98awTWoIl

https://www.youtube.com/watch?v=XvD2FrS5yYM&list=WL&index=4

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

https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/bp-gsi-overloading.html

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

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

https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/bp-relational-modeling.html