

Modelagem com DynamoDB

O que é o DynamoDB?

É um banco NoSQL totalmente gerenciado pela AWS onde podemos usar HTTPS com autenticação IAM, é rápido e consistente mesmo em escala.

Quem usa?

A amazon.com e o Lyft, então dá pra ter uma noção do tanto que ele suporta carga.

Por que usar?

Totalmente gerenciado, multi-master, multi-region. Construído para ter alta performance e é distribuído globalmente. Baixa latência na leitura e escrita para tabelas locais. A prova de desastres com redundância multi-região e fácil configurar.

Conceitos

No DynamoDB temos *Table*, *Item*, *Primary Key* e *Attributes*.

Primary Key	Attributes		
SessionId	Username	CreatedAt	ExpiresAt
23f0578e-25c9-44ff-b230-48239	blackwidow	2019-05-10T22:10:03	2019-06-10T22:10:03
8c9a175e-c394-4183-8869-50bb	thanos	2025-03-04T10:34:02	2030-04-04T10:34:02
d9ba8426-1bd5-4f85-a5f6-6ecf9	ironman	2018-06-07T15:55:12	2019-09-15T19:55:12
3289eeaf-07d2-4e06-93ed-63b1	captainamerica	1945-12-01T10:42:40	2019-12-15T13:33:43

Primary key

Se for simples, ou seja, uma única "coluna", então ela é própria *Partition Key*. Mas ela também pode ser composta, nesse caso a *primary key* é composta da *partition key* + *sort key*.

Primary Key		Attributes		
Actor (Partition key)	Movie (Sort key)			
Tom Hanks	Cast Away	Role	Year	Genre
		Chuck Noland	2000	Drama
Tom Hanks	Toy Story	Role	Year	Genre
		Woody	1995	Children's
Tim Allen	Toy Story	Role	Year	Genre
		Buzz Lightyear	1995	Children's
Natalie Portman	Black Swan	Role	Year	Genre
		Nina Sayers	2010	Drama

API Actions

Item based action

Aqui a gente consegue **escrever, atualizar ou deletar** um item. Aí precisamos fornecer a *Primary Key* completa.

Primary Key		Attributes		
Actor (Partition key)	Movie (Sort key)			
Tom Hanks	Cast Away	Role	Year	Genre
		Chuck Noland	2000	Drama
Tom Hanks	Toy Story	Role	Year	Genre
		Woody	1995	Children's
Tim Allen	Toy Story	Role	Year	Genre
		Buzz Lightyear	1995	Children's
Natalie Portman	Black Swan	Role	Year	Genre
		Nina Sayers	2010	Drama

Query

Pra fazer uma query precisamos passar a *Partition Key*, mas a *Sort Key* é opcional.


Primary Key		Attributes		
Actor (Partition key)	Movie (Sort key)			
Tom Hanks	Cast Away	Role	Year	Genre
		Chuck Noland	2000	Drama
Tom Hanks	Toy Story	Role	Year	Genre
		Woody	1995	Children's
Tim Allen	Toy Story	Role	Year	Genre
		Buzz Lightyear	1995	Children's
Natalie Portman	Black Swan	Role	Year	Genre
		Nina Sayers	2010	Drama

Scan


Nesse caso busca tudo, portanto **evite!** É muito "caro" em **escala**.

Secondary indexes

Vai inverter as chaves, algo assim:



Primary Key		Attributes		
Actor (Partition key)	Movie (Sort key)			
Tom Hanks	Cast Away	Role	Year	Genre
		Chuck Noland	2000	Drama
Tom Hanks	Toy Story	Role	Year	Genre
		Woody	1995	Children's
Tim Allen	Toy Story	Role	Year	Genre
		Buzz Lightyear	1995	Children's
Natalie Portman	Black Swan	Role	Year	Genre
		Nina Sayers	2010	Drama



Repare na nova tabela abaixo que consigo pesquisar pela *Partition Key* (Movie).

Primary Key		Attributes		
Movie (Partition key)	Actor (Sort key)			
Cast Away	Tom Hanks	Role	Year	Genre
		Chuck Noland	2000	Drama
Toy Story	Tom Hanks	Role	Year	Genre
		Woody	1995	Children's
Toy Story	Tim Allen	Role	Year	Genre
		Buzz Lightyear	1995	Children's
Black Swan	Natalie Portman	Role	Year	Genre
		Nina Sayers	2010	Drama

Modelando um sistema como exemplo

Básico

1. Comece com um ERD (Entity Relationship Diagram)
2. Defina seu padrão de acesso - isso é escrever, em português mesmo quais as consultas serão feitas
3. Design a *primary key* e *secondary indexes* - isso irá montar as queries do DynamoDB

Importante - esqueça sua experiência relacional:

- Normalização
- JOINS
- Um tipo de entidade por tabela

Setup

- Uma loja e-commerce
- Usuários (*users*) fazem pedidos (*orders*)
- Um pedido (*order*) pode ter muitos itens (*items*)

Começando com o ERD:



Identifique os padrões de acesso:

1. Get user profile
2. Get orders for user
3. Get single order and order items
4. Get orders for user by status

Design a primary key e secondary indexes:

Primary Key		Attributes			
PK	SK				
USER#junior	#PROFILE#junior	Username	FullName	Email	CreatedAt
		junior	Sarah Mamede	sarah@email.com	2018-10-03
USER#sarah	#PROFILE#sarah	Username	FullName	Email	CreatedAt
		sarah	Normandes Jr	normandes@email.com	2018-10-03

PK = USER#junior AND SK = #PROFILE#JUNIOR

One-To-Many relationships

Para fazer o relacionamento de *user* com *user_address* podemos pensar que um usuário não terá dezenas de endereço, mas apenas alguns poucos, como de casa, do trabalho e talvez o da mãe.

Sendo assim podemos desnormalizar (colocar na mesma tabela) e usar um *type document*, ou seja, uma nova coluna do tipo JSON.

Primary Key		Attributes	
PK	SK	Username	Addresses
USER#junior	#PROFILE#junior	junior	{ "Home": { "StreetAddress": "Rua X, 10", "State": "Rio de Janeiro" } }
		sarah	{ "Home": { "StreetAddress": "Rua X, 10", "State": "Rio de Janeiro" }, "Business": { "StreetAddress": "Rua Floriano, 304 ap 302", "State": "Rio de Janeiro" } }

Mas no caso do one-to-many das *orders* não seria legal colocar um JSON em um atributo, pois um *user* pode ter várias *orders* com o tempo e também podemos querer fazer uma pesquisa pela *order* em algum momento.

Então, uma outra forma de implementar o one-to-many seria usando a *sort key*.

Primary Key		Attributes				
PK	SK					
USER#junior	#PROFILE#junior	Username	FullName	Email	CreatedAt	Addresses
		junior	Sarah Mamede	sarah@...	2018-10-03	{"Home"...
	ORDER#5eaf12	Username	OrderId	Status	CreatedAt	Addresses
		junior	5eaf12	PLACED	2019-03-22	{"Home"...
	ORDER#ac630a	Username	OrderId	Status	CreatedAt	Addresses
		junior	ac630a	PLACED	2019-05-10	{"Business"...
	ORDER#f3990a	Username	OrderId	Status	CreatedAt	Addresses
		junior	f3990a	SHIPPED	2019-11-23	{"Home"...
USER#sarah	#PROFILE#sarah	Username	FullName	Genre	CreatedAt	Addresses
		sarah	Normandes Jr	norm@...	2018-10-03	{"Business"...
	ORDER#B39AD	Username	OrderId	Status	CreatedAt	Addresses
		sarah	B39AD	PLACED	2019-02-03	{"Business"...

A query para buscar as *orders* de um *user* seria assim:

"PK = USER#junior AND BEGINS_WITH(SK, 'ORDER#')"

Agora vamos para o mapeamento do **Order e OrderItems**, mas ainda usando apenas uma tabela.

Primary Key		Attributes				
PK	SK					
USER#junior	#PROFILE#junior	Username	FullName	Email	CreatedAt	Addresses
		junior	Sarah Mamede	sarah@...	2018-10-03	{"Home"...
	ORDER#5eaf12	Username	OrderId	Status	CreatedAt	Addresses
		junior	5eaf12	PLACED	2019-03-22	{"Home"...
USER#sarah	#PROFILE#sarah	Username	FullName	Genre	CreatedAt	Addresses
		sarah	Normandes Jr	norm@...	2018-10-03	{"Business"...
	ORDER#B39AD	Username	OrderId	Status	CreatedAt	Addresses
		sarah	B39AD	PLACED	2019-02-03	{"Business"...
ITEM#28291	ORDER#5eaf12	ItemId	OrderId	ProductName	Price	Status
		28291	5eaf12	Echo Dot	59,99	FAILED
ITEM#99834	ORDER#5eaf12	ItemId	OrderId	ProductName	Price	Status
		99834	5eaf12	Macbook	1999	SUCCESS
ITEM#17333	ORDER#ac630a	ItemId	OrderId	ProductName	Price	Status
		17333	ac630a	Mouse	100	FAILED

Se a gente inverter a PK com a SK criando um GSI (Global Secondary Index), veja que conseguiremos buscar os itens de uma order.

Primary Key	
SK (SERÁ PK)	PK (SERÁ SK)
#PROFILE#junior	USER#junior
#PROFILE#sarah	USER#sarah
ORDER#B39AD	
ORDER#5eaf12	USER#junior
	ITEM#28291
	ITEM#99834
ORDER#ac630a	ITEM#17333

PK = ORDER#5eaf12 and START_WITH(SK, ITEM#)

Portanto, temos 3 formas para mapearmos um one-to-many relationships:

1. Attribute (list or map)
2. Primary key + query
3. Secondary index + query

Dos nossos padrões de acesso, já resolvemos 3:

- ~~1. Get user profile~~
- ~~2. Get orders for user~~
- ~~3. Get single order and order items~~
4. Get orders for user by status

Vamos pensar agora na *Get orders for user by status*, que na nossa cabeça SQL seria algo como:

```
SELECT * FROM ORDERS
WHERE USERNAME = 'junior'
AND STATUS = 'SHIPPED'
```

Se criarmos um novo atributo chamado *OrderStatusDate* que junta o status com a data (devemos incluir mais alguma informação para não termos problemas na primary key, talvez a hora que foi feito o pedido).

Status	CreatedAt	OrderStatusDate
PLACED	2019-02-03	PLACED#2019-02-03
Status	CreatedAt	OrderStatusDate
SHIPPED	2019-12-22	SHIPPED#2019-12-22

Agora podemos criar um GSI2 com a PK do usuário e o OrderStatusDate sendo a SK:

Primary Key		Attributes				
PK	OrderStatusDate	Username	Order Id	Status	CreatedAt	OrderStatusDate
USER#junior	PLACED#2019-02-03	junior	2las2a	PLACED	2018-10-03	PLACED#2019-02-03
	SHIPPED#2019-12-22	junior	5eaf12	SHIPPE D	2019-03-22	SHIPPED#2019-12-22

Agora filtramos pelo padrão de *Composite Sort Key*:

PK = USER#junior AND BEGINS_WITH(OrderStatusDate, 'SHIPPED#')

E com isso matamos mais um padrão de acesso:

1. ~~Get user profile~~
2. ~~Get orders for user~~
3. ~~Get single order and order items~~
4. ~~Get orders for user by status~~

Nota sobre GSI

Apesar do GSI ser muito útil, existe uma limitação, podemos ter 20 por tabela.

E, caso existam vários GSI, a manutenção poderá ficar complexa, portanto é muito importante estudar sobre GSI Overloading para um projeto um pouco maior.

Basicamente termos um único GSI com um atributo "coringa", ou seja, o valor armazenado nele será a SK e poderá ser usado para montar diversas queries.

Veja na documentação da AWS para aprender mais:

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

Links importantes

- <https://www.dynamodbguide.com/what-is-dynamo-db>
- <https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/bp-indexes-general-sparse-indexes.html>
- <https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/bp-gsi-overloading.html>

Comandos

Existe uma imagem oficial AWS com o Dynamo para utilizarmos em tempo de desenvolvimento.

```
docker run -p 8000:8000 --rm amazon/dynamodb-local
```

Os comandos abaixo foram retirados da documentação oficial da AWS, consulte para ver mais detalhes:

<https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/GettingStarted.CoreComponents.html>

Criar tabela:

```
aws dynamodb create-table \
  --table-name Music \
  --attribute-definitions \
    AttributeName=Artist,AttributeType=S \
    AttributeName=SongTitle,AttributeType=S \
  --key-schema \
    AttributeName=Artist,KeyType=HASH \
    AttributeName=SongTitle,KeyType=RANGE \
  --provisioned-throughput \
    ReadCapacityUnits=10,WriteCapacityUnits=5 \
  --endpoint-url http://localhost:8000
```

Scan:

```
aws dynamodb scan \  
  --table-name Music \  
  --endpoint-url http://localhost:8000
```

Escrever dados:

```
aws dynamodb put-item \  
  --table-name Music \  
  --item \  
    '{"Artist": {"S": "No One You Know"}, "SongTitle": {"S": "Call Me  
Today"}, "AlbumTitle": {"S": "Somewhat Famous"}, "Awards": {"N":  
"1"}}' \  
  --endpoint-url http://localhost:8000
```

```
aws dynamodb put-item \  
  --table-name Music \  
  --item \  
    '{"Artist": {"S": "Acme Band"}, "SongTitle": {"S": "Happy Day"},  
"AlbumTitle": {"S": "Songs About Life"}, "Awards": {"N": "10"} }' \  
  --endpoint-url http://localhost:8000
```

Lendo um item:

```
aws dynamodb get-item --consistent-read \  
  --table-name Music \  
  --key '{"Artist": {"S": "Acme Band"}, "SongTitle": {"S": "Happy  
Day"}}' \  
  --endpoint-url http://localhost:8000
```

Update item:

```
aws dynamodb update-item \  
  --table-name Music \  
  --key '{"Artist": {"S": "Acme Band"}, "SongTitle": {"S": "Happy  
Day"}}' \  
  --update-expression "SET AlbumTitle = :newval" \  
  --expression-attribute-values '{":newval":{"S":"Updated Album  
Title"}}' \  
  --return-values ALL_NEW \  
  --endpoint-url http://localhost:8000
```

Query:

```
aws dynamodb query \  
  --table-name Music \  
  --key-condition-expression "Artist = :keyval" \  
  --expression-attribute-values '{":keyval":{"S":"Acme Band"}}' \  
  --scan-index-name "SecondaryIndex" \  
  --limit 100
```

```
--key-condition-expression "Artist = :name" \  
--expression-attribute-values '{":name":{"S":"Acme Band"}}' \  
--endpoint-url http://localhost:8000
```

Criando um Global Secondary Index (GSI):

```
aws dynamodb update-table \  
  --table-name Music \  
  --attribute-definitions AttributeName=AlbumTitle,AttributeType=S  
\  
  --global-secondary-index-updates \  
    "[{\"Create\":{\"IndexName\":  
\"AlbumTitle-index\", \"KeySchema\": [{\"AttributeName\": \"AlbumTitle\"  
, \"KeyType\": \"HASH\"}], \  
    \"ProvisionedThroughput\": {\"ReadCapacityUnits\": 10,  
\"WriteCapacityUnits\": 5  
}, \"Projection\": {\"ProjectionType\": \"ALL\"}}}]" \  
  --endpoint-url http://localhost:8000
```

Consultando no Global Secondary Index:

```
aws dynamodb query \  
  --table-name Music \  
  --index-name AlbumTitle-index \  
  --key-condition-expression "AlbumTitle = :name" \  
  --expression-attribute-values '{":name":{"S":"Somewhat  
Famous"}}' \  
  --endpoint-url http://localhost:8000
```