AWS DEV DAY ONLINE JAPAN

E-2

DynamoDBの初心者に伝えたい初めて触るときの勘所

佐々木 了

In Partnership with intel

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

自己紹介

```
$ curl https://api.marusa.tech/v1/profile
{
   "business_name": "MARUSA TECH",
   "name": "Ryo Sasaki",
   "email": "sasaki.ryo@marusa.tech",
   "website": "https://www.marusa.tech"
}
```

- 佐々木 了
- フリーランス インフラエンジニア@3年目
- OpenStack Love
- Python Love
- Angular Love
- サーバーレス愛好家
- 建築業界向けのWebサービスを運営中
  - このサービスの開発と運用で今回の内容のノウハウをためました

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

本日の内容と対象者

内容

- やってはいけないこと
- 便利な機能の紹介
- 設計のポイント
- 便利なツールの紹介

対象者

- DynamoDBをこれから使いたくて勉強中の人
- DynamoDBを使い始めて間もない人

・DynamoDBって何？みたいな人は対象外
・ゴリゴリ使いまくってる中上級者も対象外

aws
© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.
In Partnership with intel

サービス紹介

- 建築業界向けの施工写真管理アプリ
- AWSネイティブなフルサーバーレス構成

Frontend

AWS Certificate Manager

AWS Amplify

Amazon CloudFront

Amazon Simple Storage Service (S3)

Amazon Cognito

Backend

Amazon DynamoDB

Amazon API Gateway

AWS Lambda

Amazon Simple Storage Service (S3)

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

In Partnership with intel.

最初は本当に辛かったです

- 今までと大きく考え方を変える必要があった
  - RDBはもちろん他のNoSQL DBとも
- DocumentDBを使えば(待てば)良かったと後悔したときもある

ずっと

mongoDB

を愛用していました

・欲しいデータ全然持ってこれないんだけど・・・
・Scanばっかり使うとめっちゃ高くならない？
・LSIとGSI神だけど、上限ががががが

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

aws DEV DAY
ONLINE JAPAN

そんな初心者がDynamoDBを
1年以上使用した上で学んだことを、
これから始める皆さんに色々お伝えしたい

In Partnership with intel

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

前提

- <u>そこまで詳しい内容は話しません！</u>
- 今回説明する内容以外にもDynamoDBにはたくさん仕様や機能が存在します
- でも、突っ込んだ内容はこの場ではしません
- 今回はここだけは覚えてほしい！といった個人的に重要だと考える点のみを紹介します

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

勘所

- ただ1つのやってはいけないこと
- 設計段階からDynamoDBの便利機能を考慮しよう
- アイテムサイズを減らす努力をしよう
- 本当に最新データが必要なのか考え直そう
- 効率的なデータ取得のためにGSI OVERLOADINGを使いこなそう
- DynamoDB LocalやWorkBenchを使って徹底的な設計を行おう

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

In Partnership with intel

一応、基本用語

- **Primary Key**
  - データを一意に識別するためのキー
  - Partition Key単独あるいは後述のSort Keyと組み合わせた複合キーのどちらかで構成される
- **Partition Key** (旧名: Hash Key)
  - DynamoDBのデータは必ずPartitionと呼ばれる保存領域に結びつく
  - どこのPartitionに保存されるか？を決めるのがPartition Key
  - 全てのテーブルに必ず設定され、単独であるならば必ず一意にならなければならない

属性(Attribute)

Partition Key

社員IDなので一意

employee_id	team	employee_name	job	skills	age
0001	hoge	user01	lead_engineer	[“C”, “C++”, “C#”, “Java”, “Rust”]	35
0002	hoge	user02	engineer	[“Java”, “Rust”]	29
0003	fuga	user03	lead_engineer	[“Ruby”, “Node.js”, “Deno”]	30
0004	fuga	user04	engineer	[“Node.js”]	25
0005	fuga	user05	engineer	[“Node.js”, “Python”]	23

項目(Item)

- Partition Keyで検索することで社員毎のデータを取得可能

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

In Partnership with intel.

一応、基本用語

- **複合キーテーブル**
  - Paritition KeyとSort Keyの2つのKeyを使用したテーブル
  - 2つを組み合わせた複合キーで一意なPrimary Keyを構成すれば良い
  - つまりPartition Key単独なら重複OK
- **Sort Key** (旧名: Range Key)
  - 同一Partition Keyのデータは同一Partitionに保存される
  - その中でソートさせるために使用する

Partition Key	Sort Key				
team	employee_id	employee_name	job	skills	age
hoge	0001	user01	lead_engineer	[“C”, “C++”, “C#”, “Java”, “Rust”]	35
hoge	0002	user02	engineer	[“Java”, “Rust”]	29
fuga	0003	user03	lead_engineer	[“Ruby”, “Node.js”, “Deno”]	30
fuga	0004	user04	engineer	[“Node.js”]	24

チーム名なので
一意にはならない

チーム名＋社員IDで
一意になる

- Partition Keyのみで検索することでチーム毎のデータを取得可能
- Partition Key + Sort Keyの検索により特定社員のデータを取得可能

aws

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

In Partnership with intel.

一応、基本用語

- **Secondary Index**
  - 作成したテーブルのPartition KeyあるいはSort Keyを組み合わせた複合キーテーブルでは、取得し辛いデータがある場合などに使用
- **Local Secondary Index**
  - 別のSort Keyを設定可能、Partition Keyはオリジナルのテーブルのものを使用するしかない
  - テーブル初期作成時にしか作成できない
  - 上限5個まで作成可能
  - Keyが重複してもOK、一意である必要はない

Sort Keyを変更

元々のSort Key

Partition Key	Sort Key				
team	job	employee_name	employee_id	skills	age
hoge	lead_engineer	user01	0001	[“C”, “C++”, “C#”, “Java”, “Rust”]	35
hoge	engineer	user02	0002	[“Java”, “Rust”]	29
fuga	lead_engineer	user03	0003	[“Ruby”, “Node.js”, “Deno”]	30
fuga	engineer	user04	0004	[“Node.js”]	25
fuga	engineer	user05	0005	[“Node.js”, “Python”]	23

重複OK

- Partition Keyのみで検索することでチーム毎のデータを取得可能
- Partition Key + Sort Keyの検索により特定チームの特定職の社員を取得可能

aws

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

In Partnership with intel.

一応、基本用語

- **Global Secondary Index**
  - Partition KeyもSort Keyも完全に別のものとして設定可能
  - テーブル初期作成後にも作成可能
  - 上限20個まで作成可能
  - Keyが重複してもOK、一意である必要はない
  - RCU/WCUの設定が必要 (テーブルをもう1つ作るような感覚)

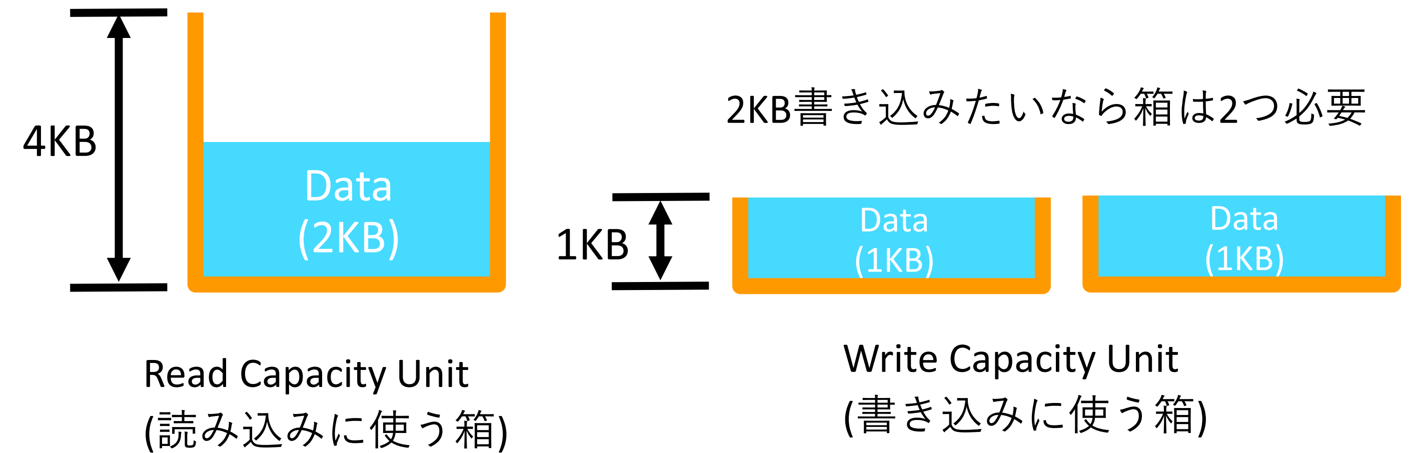
Partition Key					
job	employee_id	employee_name	skills	age	Team
lead_engineer	0001	user01	[“C”, “C++”, “C#”, “Java”, “Rust”]	35	hoge
engineer	0002	user02	[“Java”, “Rust”]	29	hoge
lead_engineer	0003	user03	[“Ruby”, “Node.js”, “Deno”]	30	fuga
engineer	0004	user04	[“Node.js”]	25	fuga
engineer	0005	user05	[“Node.js”, “Python”]	23	fuga

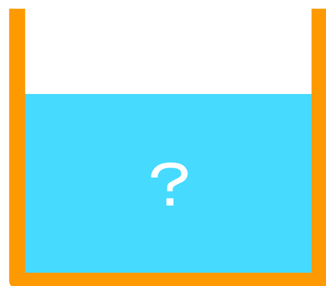
- Partition Keyのみで検索することで特定職の社員を取得可能

aws
© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.
In Partnership with intel.

一応、基本用語

- **Capacity Unit**
  - 1秒あたりに使用可能なデータの読み込み/書き込み容量
    - 回数ではなく容量と考えたほうがしっくりきやすい
  - 実際の消費量ではなく、使用可能な上限値
  - DynamoDBにおける一番のコスト要因
  - CUが多ければ、より大量のデータを読み込み/書き込み可能
    - つまりパフォーマンス向上につながる
    - 一方で露骨なコスト増にもつながる
    - それをどうバランシングするかがキモ





x N

事前に何個用意しておく？

or

動的に用意してもらう？

- Provisioned: CUを1秒あたり何個用意するか？を事前定義
  - 用意された箱を使い切ると、データは運べない
  - どの程度のトラフィックが発生し得るか？を必ず予測できている必要がある
  - Reserved Capacityを購入すれば安くできる
- On-Demand: 必要なときに必要な分だけCUを用意してくれる
  - どれだけトラフィックに変化があったとしても足りなくなるということがない
  - ワークロードが急激に増減し得るケースに向く
  - (状況によるが) Provisionedよりもコストが高くつきやすい

参考: https://docs.aws.amazon.com/ja_jp/amazondynamodb/latest/developerguide/HowItWorks.ReadWriteCapacityMode.html

aws

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

In Partnership with intel.

勘所

- **ただ1つのやってはいけないこと**
- 設計段階からDynamoDBの便利機能を考慮しよう
- アイテムサイズを減らす努力をしよう
- 本当に最新データが必要なのか考え直そう
- 効率的なデータ取得のためにGSI OVERLOADINGを使いこなそう
- DynamoDB LocalやWorkBenchを使って徹底的な設計を行おう

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

In Partnership with intel.

先にテーブルを作り始めてはいけません。

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

先にテーブルを作り始めてはいけません

- どんなデータが欲しいか？
- どんなデータアクセスが想定されるか？

が固まっていない限り、作り始めるべきではない

なぜ？

- RDBなら、SQLクエリ次第でどんなデータでも取得できる
- DynamoDBの場合、データ操作アクションが非常に限られているため、欲しいデータを持ってくることができない(場合がよくある)
- 力技で取得することも可能だが、金銭的コストが非常に高くつく

こんな画面/機能が欲しい

こういうデータが必要

```
{
  "hoge": "fuga",
  "foo": [
    "bar",
    "baz",
    "piyo"
  ]
}
```

このデータを得るためにはこういうデータ構造が必要

employee_id	team	employee_name	job
0001	hoge	user01	lead_engineer
0002	hoge	user02	engineer
0003	fuga	user03	lead_engineer
0004	fuga	user04	engineer
0005	fuga	user05	engineer

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

In Partnership with intel.

先にテーブルを作り始めてはいけません

DynamoDBは事前設計が本当に大事。
設計が甘いと、とんでもない金食い虫になる。

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

勘所

- ただ1つのやってはいけないこと
- **設計段階からDynamoDBの便利機能を考慮しよう**
- アイテムサイズを減らす努力をしよう
- 本当に最新データが必要なのか考え直そう
- 効率的なデータ取得のためにGSI OVERLOADINGを使いこなそう
- DynamoDB LocalやWorkBenchを使って徹底的な設計を行おう

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

In Partnership with intel.

設計段階からDynamoDBの便利機能を考慮しよう

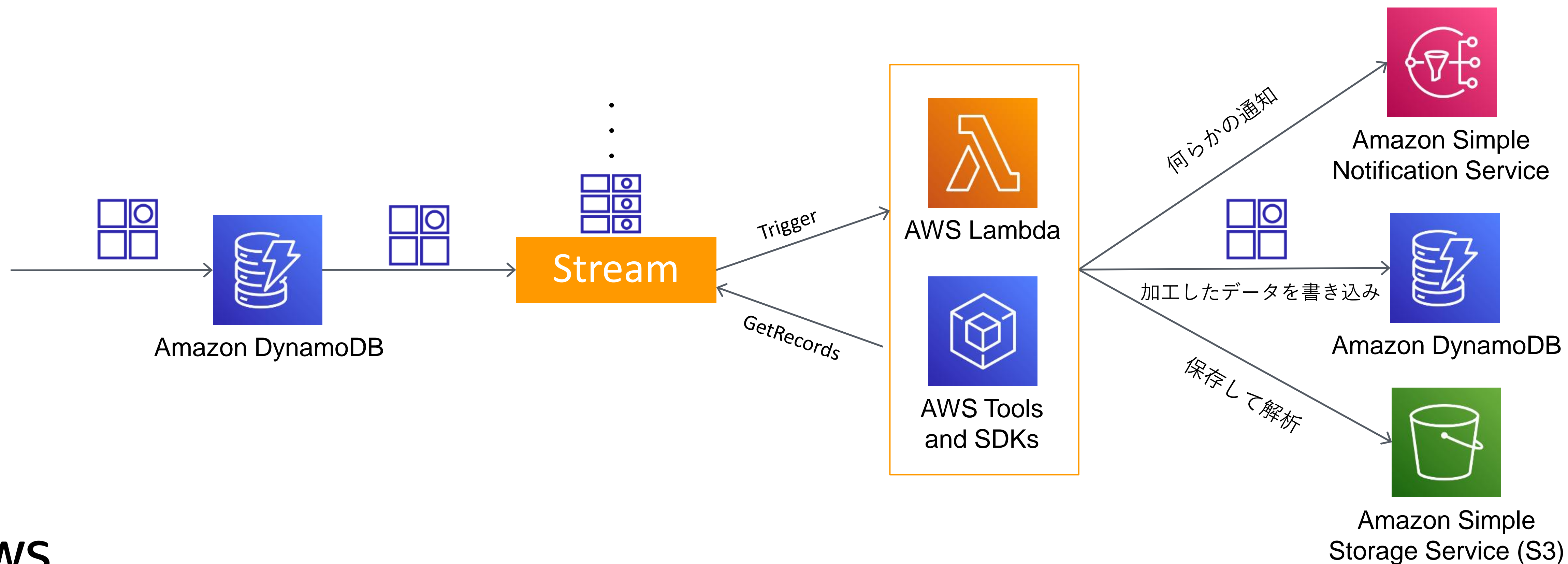
- 是非とも設計段階から意識して使ってほしい機能
  - Streams
  - TTL
  - PITR
- 下記の機能は裏側で自動的に有効化されている機能
  - Adaptive Capacity
  - Burst Capacity

他にもたくさん機能はあるけど
ひとまずこれだけ。

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Streams

- アイテムの追加/削除/変更を検知し、該当アイテムの情報が時系列順にStream(queue)に書き込まれている
- 24時間以内であればStreamから任意に取り出せる
  - 24時間経過すると自動的に削除
- Lambdaなどと連携させることでイベントドリブンに処理可能 = Triggers

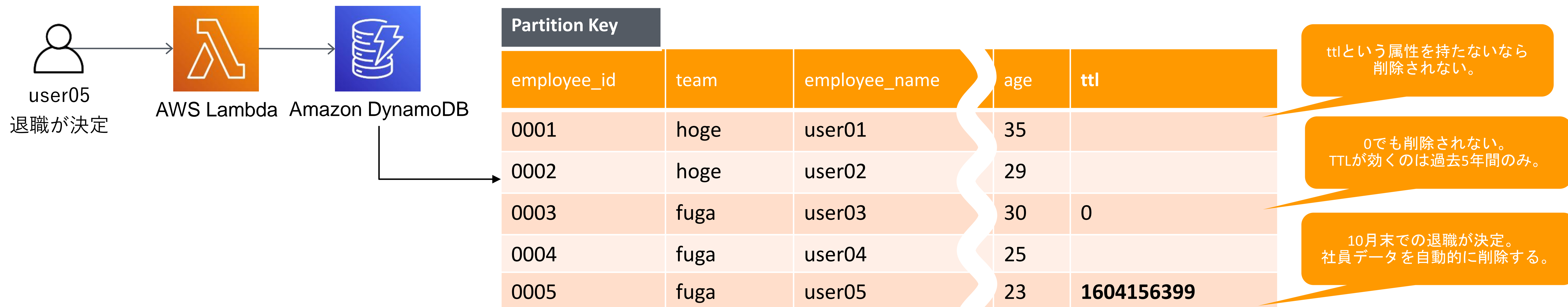


© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

In Partnership with intel.

TTL (Time to Live)

- アイテム毎にそのアイテムの有効期限を決められる
  - 削除して欲しい日時を秒単位のUNIXエポックタイムに変換し、指定した属性のValueとして設定する
- つまりアイテムを自動的に削除できる
- 削除はバックグラウンドで行われるため、WCUを消費しない
- 有効期限を迎えてから48時間以内に削除される仕様なので、即時性が必要なものには向かない



employee_id	team	employee_name	age	ttl
0001	hoge	user01	35	
0002	hoge	user02	29	
0003	fuga	user03	30	0
0004	fuga	user04	25	
0005	fuga	user05	23	**1604156399**

aws
© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.
In Partnership with intel.

PITR (Point-In-Time Recovery)

- 継続的バックアップとも言う
  - 他にもオンデマンドバックアップの機能もある
- 35日間、1秒単位で自動的にバックアップ取得
- もちろん1秒単位でいつでもリストア可能
- 恐ろしく便利なのに安い
  - 東京リージョンの場合: 0.228USD/GB
- 長期保存が必要ならAWS Backupを使うべき

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Adaptive Capacity

よく言われませんか？

Partition Keyは均一なワークロードが
かかるように設計しよう！

参考: https://docs.aws.amazon.com/ja_jp/amazondynamodb/latest/developerguide/bp-partition-key-uniform-load.html

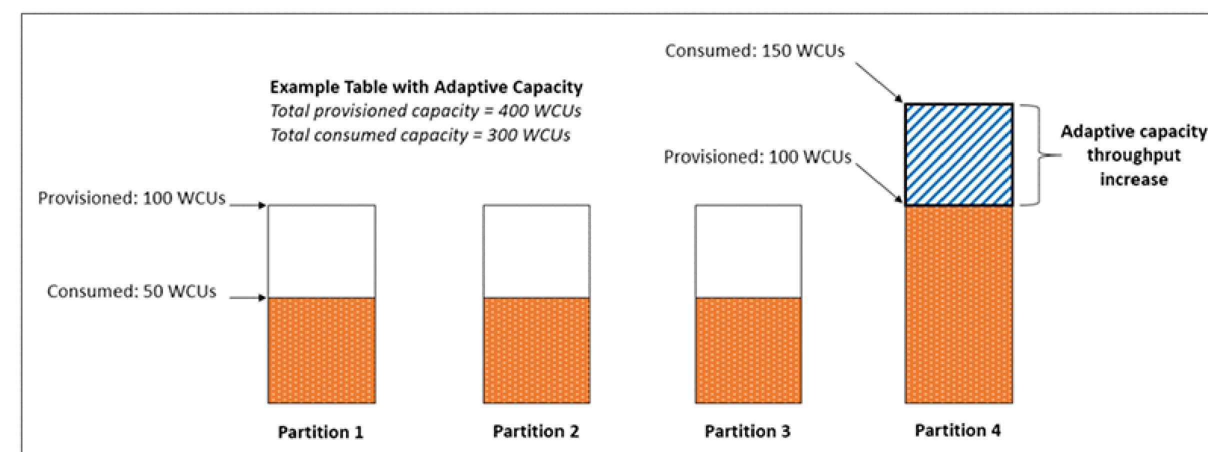
良いPartition Key / 悪いPartition Keyの例が
載っているので必見

いや、ぶっちゃけ難しいでしょ・・・
実際のプロダクション環境のワークロードなんて
わからないよ・・・

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Adaptive Capacity

- ホットパーティションの処理能力を一時的に向上させる機能
- 不均一なワークロードによるスロットリングを防ぐことができる
  - つまり特定パーティションにトラフィックが偏る状態に対応できる
  - もちろん効果は絶対ではない



引用: https://aws.amazon.com/jp/blogs/news/how-amazon-dynamodb-adaptive-capacity-accommodates-uneven-data-access-patterns-or-why-what-you-know-about-dynamodb-might-be-outdated/

明らかに偏りそうな設計はするべきではないが・・・
もうそこまで気にする必要はないと思う。

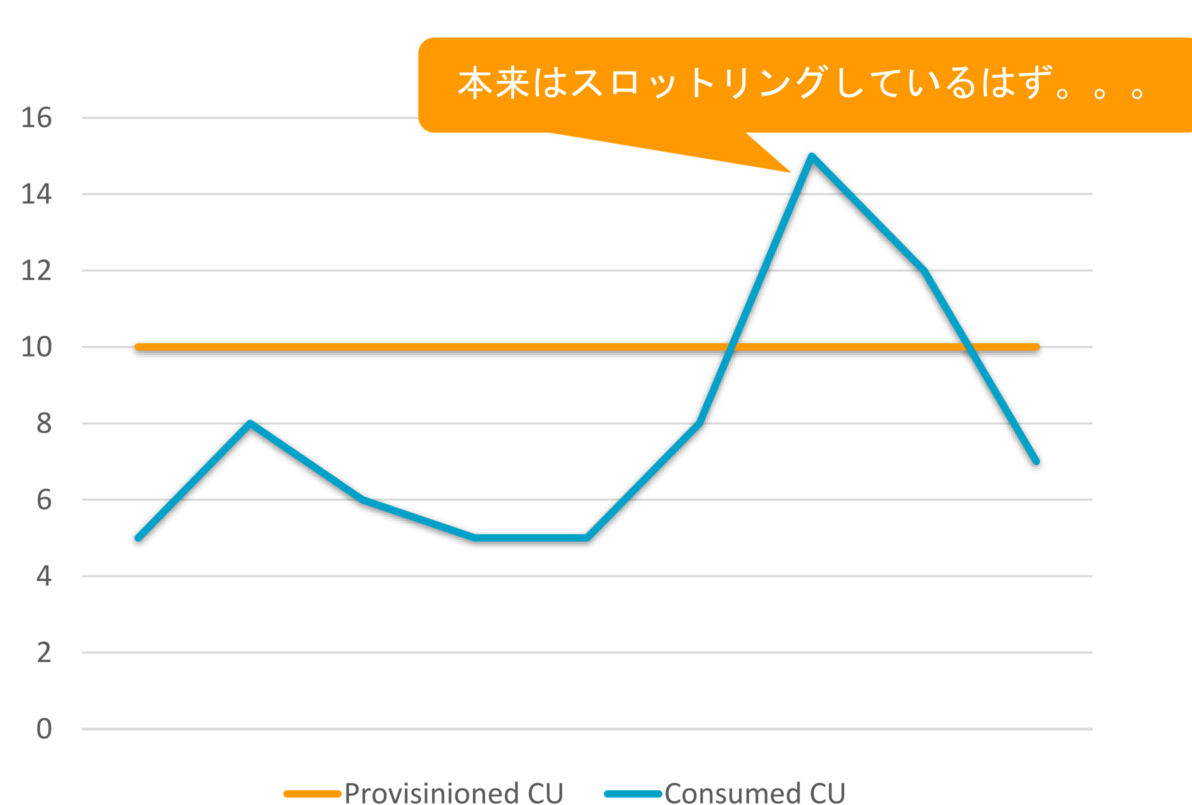
ちなみに小規模環境ではパーティションは1つになる。
下記資料の35ページからパーティションの計算式が載っているので是非参考にしてほしい。
参考: https://www.slideshare.net/AmazonWebServicesJapan/20150805-aws-blackbeltdynamodb

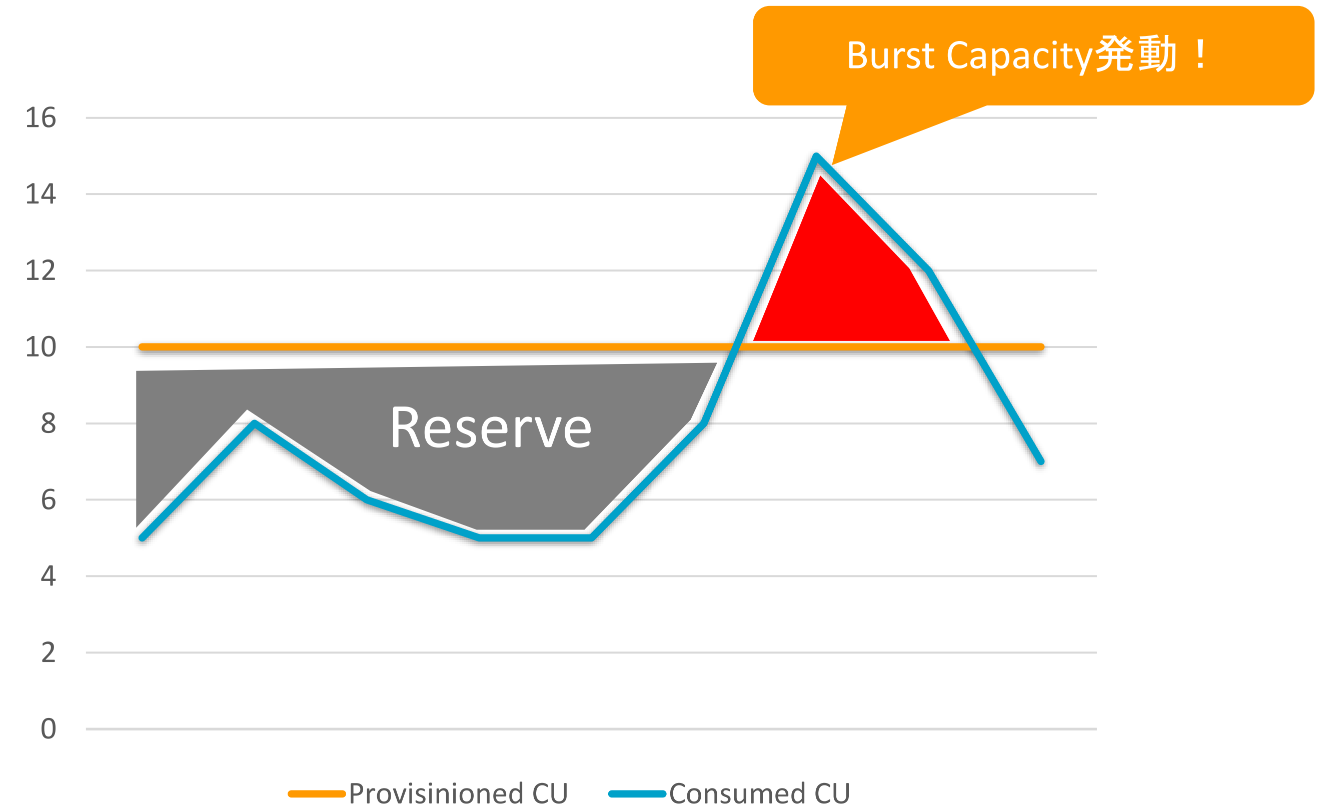
© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

In Partnership with intel.

Burst Capacity

- Provisioned CUの内、過去5分間に使用されなかったCUをリザーブ
- そのリザーブをバーストトラフィック発生時に使用し処理可能
- 右肩上がりなトラフィックをさばくならAuto Scalingを使用すべき
- バーストトラフィックが多発するならCapacity ModeをOn-Demandで運用すべき





© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

aws

In Partnership with intel.

Burst Capacity

テーブルの詳細

テーブル名	Teams
プライマリパーティションキー	team (文字列)
プライマリソートキー	employee_id (数値)
ポイントインタイムリカバリ	無効 有効化
暗号化タイプ	デフォルト 暗号化の管理
KMS マスターキーの ARN	該当しません
暗号化状態	
CloudWatch 投稿者のインサイト	無効 投稿者のインサイトを管理 新規
有効期限 (TTL) 属性	無効 TTL の管理
テーブルの状態	有効
作成日	2020年10月10日 17:27:05 UTC+9
読み込み/書き込みキャパシティーモード	プロビジョンド
オンデマンドモードへの最後の変更	-
プロビジョンド読み込みキャパシティーユニット	1 (Auto Scaling 無効)
プロビジョンド書き込みキャパシティーユニット	1 (Auto Scaling 無効)
最後の減少時刻	-
最後の増加時刻	-
ストレージサイズ（バイト単位）	0 バイト
項目数	0 ライブ項目数の管理
リージョン	Asia Pacific (Tokyo)

```
$ aws dynamodb scan ¥
>    --consistent-read ¥
>    --return-consumed-capacity TOTAL ¥
>    --table Teams
{
```

```
    "Count": 1,
    "ScannedCount": 1,
    "ConsumedCapacity": {
        "TableName": "Teams",
        "CapacityUnits": 71.0
    }
}
```

何回も実行して
リザーブを枯渇させる

```
$ time aws dynamodb scan ¥
>    --consistent-read ¥
>    --return-consumed-capacity TOTAL ¥
>    --table Teams

An error occurred (ProvisionedThroughputExceededException) when
calling the Scan operation (reached max retries: 9): The level of
configured provisioned throughput for the table was exceeded.
Consider increasing your provisioning level with the UpdateTable
API.

real    0m26.961s
user    0m0.927s
sys     0m0.067s
```

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

In Partnership with intel.

勘所

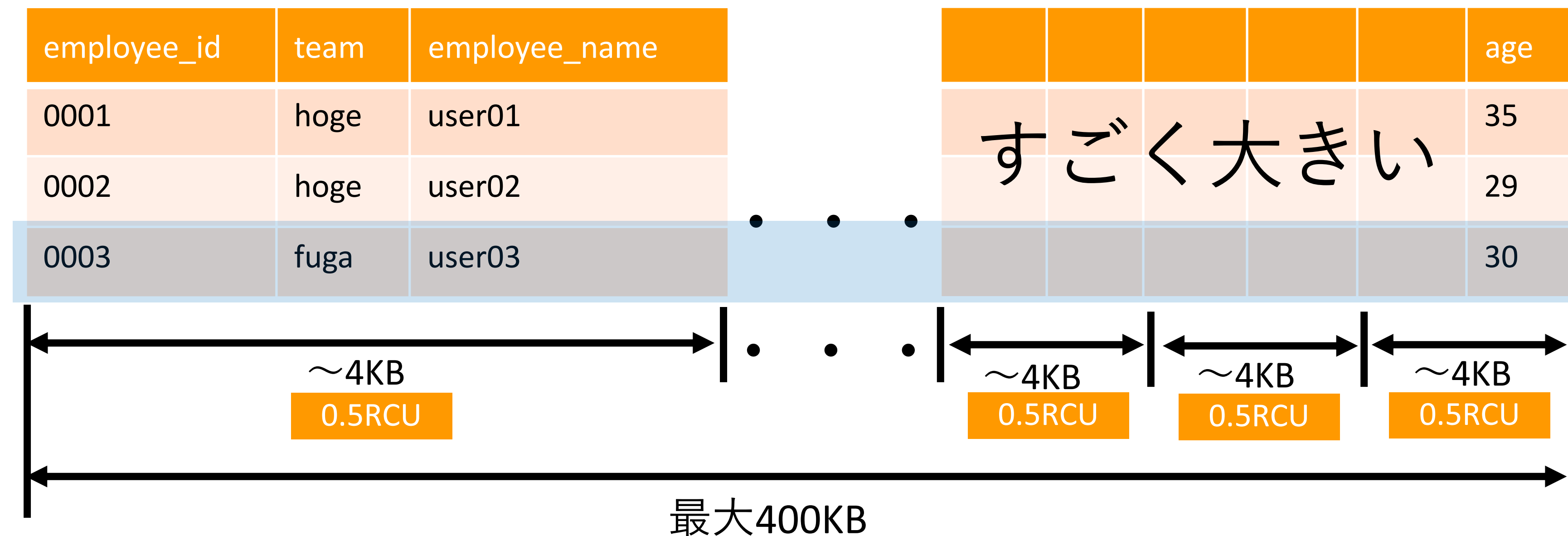
- ただ1つのやってはいけないこと
- 設計段階からDynamoDBの便利機能を考慮しよう
- アイテムサイズを減らす努力をしよう
- 本当に最新データが必要なのか考え直そう
- 効率的なデータ取得のためにGSI OVERLOADINGを使いこなそう
- DynamoDB LocalやWorkBenchを使って徹底的な設計を行おう

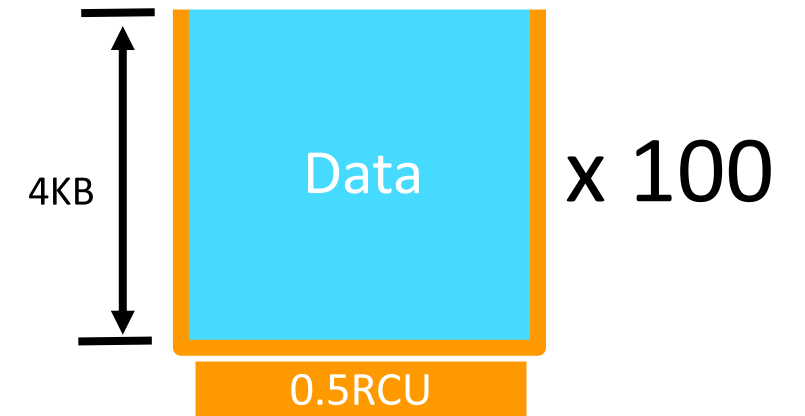
© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

In Partnership with intel.

アイテムサイズを減らす努力をしよう

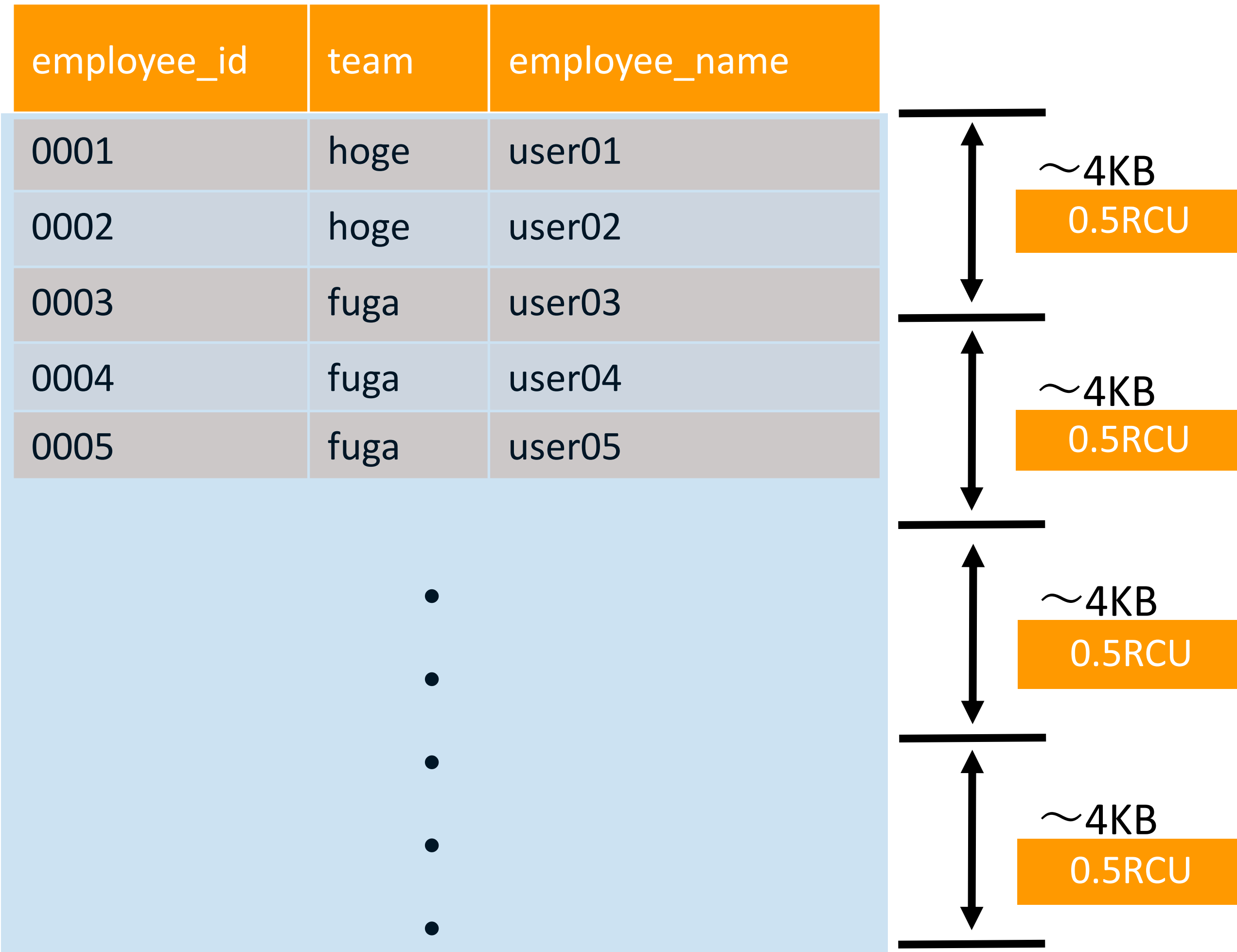
- 1アイテムの最大サイズ: 400KB
- 1 Read Capacity Unitの境界(最大サイズ): 4KB
  - 結果整合性のある読み込み(デフォルト)であるなら0.5RCU
- 1 Write Capacity Unitの境界(最大サイズ): 1KB
- つまり大きなアイテムサイズにするとCUの消費が尋常じゃない
- ちなみにトランザクション処理させるとCUを2倍消費





© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

アイテムサイズを減らす努力をしよう



employee_id	team	employee_name
0001	hoge	user01
0002	hoge	user02
0003	fuga	user03
0004	fuga	user04
0005	fuga	user05

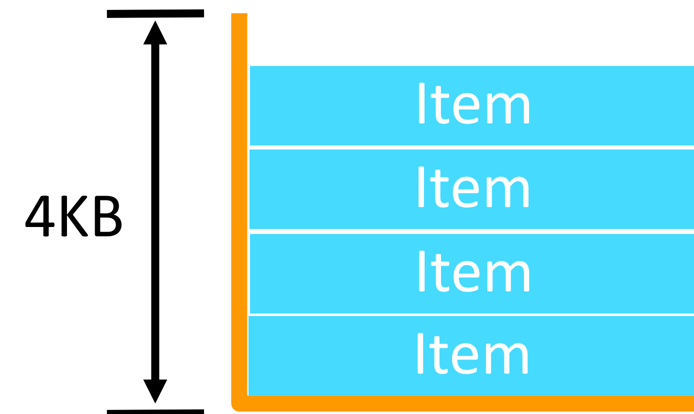
1つのアイテムサイズが小さくても、Queryなどの範囲取得の場合、塵も積もれば何とやらでCUを大量消費しやすい。

これはある程度仕方ない面が大きいが・・・

だからこそ1アイテムサイズは極力小さくするべき。

そして、全データ取得のScanは使うべきではない。

極力小さいまとまりで取得できるようにテーブル設計をしよう。



アイテムサイズが小さければ・・・

1RCUで複数取得も十分可能

aws

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

In Partnership with intel.

アイテムサイズを減らす努力をしよう

- Capacity Unitは容量で計算するので4KBを下回っているなら複数のアイテムを収容できる
- 極端な話、1件あたり0.1KBのアイテムなら40件含められる

ただしあくまでQueryなどの範囲取得のときの話。
書き込みの場合はBatchWriteItemを使ったとしても、
アイテム1件毎に処理されてしまうので、こうはならない。

employee_id	team	employee_name		age
0001	hoge	user01		35
0002	hoge	user02		29
0003	fuga	user03		30
0049	fuga	user49		25
0050	fuga	user50		23

0.1KB

0.1KB * 50件 = 5KB

つまり1RCUで足りる

※0.5RCUで4KBなので、1RCUあれば8KB収容可能

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

アイテムサイズを減らす努力をしよう

employee_id	team	employee_name	icon	job	skills	age
0001	hoge	user01	[base64 encoded image]	lead_engineer	[“C”, “C++”, “C#”, “Java”, “Rust”]	35
0002	hoge	user02		engineer	[“Java”, “Rust”]	29

```
$ aws dynamodb get-item ¥
>    --endpoint-url http://127.0.0.1:8000 ¥
>    --return-consumed-capacity TOTAL ¥
>    --table Teams ¥
>    --key '{ "team": {"S": "hoge" }, "employee_id": {"N": "2"} }'
{
    "Item": {
        "skills": {
            "SS": [
                "C#",
                "Java",
                "Rust"
            ]
    },
    "ConsumedCapacity": {
        "TableName": "Teams",
        "CapacityUnits": 0.5
    }
}
```

```
$ aws dynamodb get-item ¥
>    --endpoint-url http://127.0.0.1:8000 ¥
>    --return-consumed-capacity TOTAL ¥
>    --table Teams ¥
>    --key '{ "team": {"S": "hoge" }, "employee_id": {"N": "1"} }'
{
    "Item": {
        "icon": {
            "B": "QUFBQUFBQUFBQ ...
    },
    "ConsumedCapacity": {
        "TableName": "Teams",
        "CapacityUnits": 47.5
    }
}
```

約380KB

約100倍

4 * 47.5 * 2 = 380

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

In Partnership with intel.

アイテムサイズを減らす努力をしよう: 対策

- 400KBに収まるとしても、Binaryなどの大きなデータは格納しないほうが無難
  - 画像データなどの大きなデータはS3に置いて配置先情報だけをDynamoDB内に置こう

employee_id	team	employee_name	icon	job	skills	age
0001	hoge	user01	/icon-images/001.jpg	lead_engineer	[“C”, “C++”, “C#”, “Java”, “Rust”]	35
0002	hoge	user02	/icon-images/002.jpg	engineer	[“Java”, “Rust”]	29

4KB以内に十分収まる

- Binaryを含めない場合でもアイテムが大きくなってしまったら？
  - 極力、属性名に短縮名を使用する
  - LSI, GSIにはProjection(射影)を適用する
  - あるいは、gzipなどで圧縮して格納、というやり方も

参考: https://docs.aws.amazon.com/ja_jp/amazondynamodb/latest/developerguide/bp-use-s3-too.html

aws

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

In Partnership with intel.

おまけ: Scanは極力使うべきではない

- Scan: テーブル内の全アイテムを取得
  = 取得データサイズが非常に大きくなる
  = RCUの消費が激しい
- フィルタはCUの節約には全くならない
- 使うならせめてSparce Indexテクニックを利用しよう

参考: https://docs.aws.amazon.com/ja_jp/amazondynamodb/latest/developerguide/bp-indexes-general-sparse-indexes.html

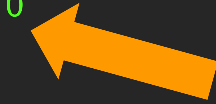
Scanには頼らず、QueryやGetItemによって、
目的のデータを取得できるようなテーブル設計を行うべき

```
$ aws dynamodb scan ¥
>     --endpoint-url http://127.0.0.1:8000 ¥
>     --return-consumed-capacity TOTAL ¥
>     --table Teams
{

     "Count": 16,
     "ScannedCount": 16,
     "ConsumedCapacity": {
         "TableName": "Teams",
         "CapacityUnits": 142.0
     }
}
```

```
$ aws dynamodb scan ¥
>     --endpoint-url http://127.0.0.1:8000 ¥
>     --return-consumed-capacity TOTAL ¥
>     --table Teams ¥
>     --scan-filter '{
         "job":{
             "AttributeValueList":[ {"S":"engineer"} ],
             "ComparisonOperator": "EQ"
         }
     }'

     "Count": 16,
     "ScannedCount": 16,
     "ConsumedCapacity": {
         "TableName": "Teams",
         "CapacityUnits": 142.0
     }
}
```



© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

勘所

- ただ1つのやってはいけないこと
- 設計段階からDynamoDBの便利機能を考慮しよう
- アイテムサイズを減らす努力をしよう
- 本当に最新データが必要なのか考え直そう
- 効率的なデータ取得のためにGSI OVERLOADINGを使いこなそう
- DynamoDB LocalやWorkBenchを使って徹底的な設計を行おう

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

In Partnership with intel.

本当に最新データが必要なのか考え直そう

- そもそもいつでも最新データは必要なのか？
- 毎回DBから読み込む必要があるのか？
- 本当に強力な整合性は必要なのか？

原則的にいつでも結果整合性のある読み込みをし、
どうしても必要なときのみ強力な整合性のある読み込みをしよう。
※いつも強力な整合性で、必要ないときに結果整合性、ではない

社員の増減なんて
頻繁には発生しない

スキルだって
そうそう変わるものではない

employee_id	team	employee_name	job	skills	age
0001	hoge	user01	lead_engineer	[“C”, “C++”, “C#”, “Java”, “Rust”]	35
0002	hoge	user02	engineer	[“Java”, “Rust”]	29
0003	fuga	user03	lead_engineer	[“Ruby”, “Node.js”, “Deno”]	30
0004	fuga	user04	engineer	[“Node.js”]	25
0005	fuga	user05	engineer	[“Node.js”, “Python”]	23

aws

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

In Partnership with intel.

本当に最新データが必要なのか考え直そう

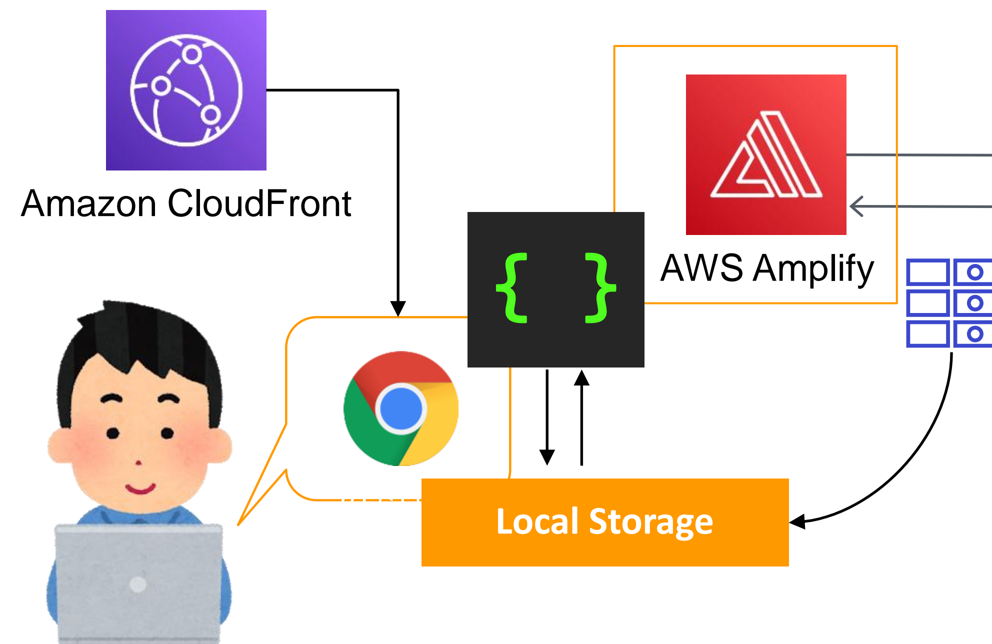
データの特性にもよるが・・・アプリレベルでのキャッシング処理も効果的。

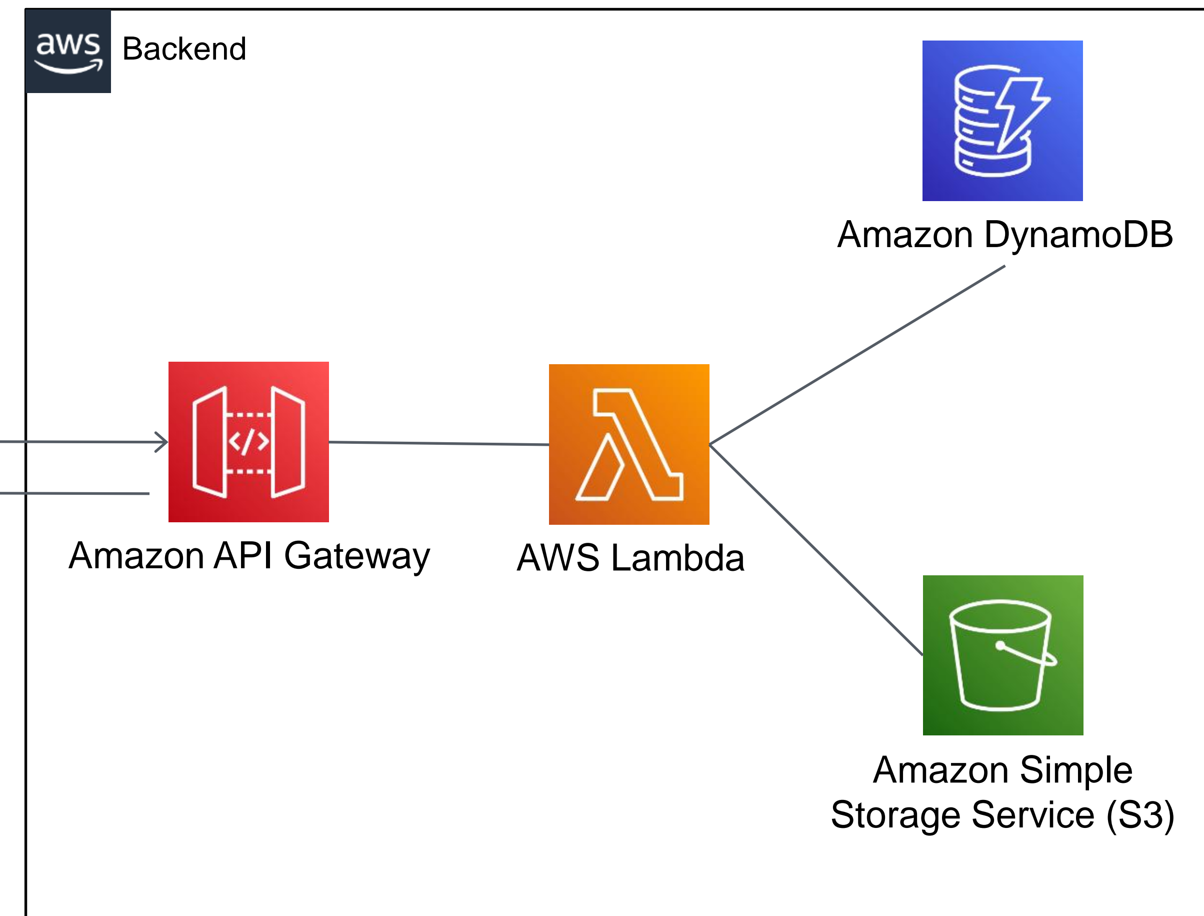
ある程度時間が経つか、人間が"最新データを取得"ボタンを実行するまでは、

基本的にローカルにためたデータを使う

＝ 毎回DynamoDBにアクセスするわけではないのでRCUを節約できる

※この構成の場合、ついでにAPI GatewayやLambdaの実行回数も節約





© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

本当に最新データが必要なのか考え直そう

- ある程度の規模になったらDynamoDB Accelerator (DAX)を使おう
  - スモールスタートなサービスだと逆に高くつくかも
  - 多くのプロダクション環境ならかなりのRCUの節約になる
    - つまりコストの節約や読み込みレイテンシの低減(パフォーマンスの向上)につながる

aws

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

In Partnership with intel.

勘所

- ただ1つのやってはいけないこと
- 設計段階からDynamoDBの便利機能を考慮しよう
- アイテムサイズを減らす努力をしよう
- 本当に最新データが必要なのか考え直そう
- **効率的なデータ取得のためにGSI OVERLOADINGを使いこなそう**
- DynamoDB LocalやWorkBenchを使って徹底的な設計を行おう

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

In Partnership with intel.

GSI OVERLOADINGを使いこなそう

- GSIは最大20個までしか作れない
  - 無限に作成できるわけではない
  - コストも高くつきやすい
- より柔軟にデータ取得できるようにする手法 **= GSI OVERLOADING**

・結果的にGSIの節約になる
・アプリ側の実装コードもシンプルになる(と思う)

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

GSI OVERLOADING

- 1つのGSIで複数用途のデータ取得に対応できるように定義する設計手法
- 属性(Attribute)名を保持する専用のKeyを用意する
  - 横を縦に持ってくるイメージ
- そのテーブルに対してGSIを定義するとデータ取得の柔軟性が非常に向上する

employee_id	team	employee_name	job
0001	hoge	user01	lead_engineer
0002	hoge	user02	engineer
0003	fuga	user03	lead_engineer
0004	fuga	user04	engineer
0005	fuga	user05	engineer

employee_id	context	data
0001	team	hoge
0001	employee_name	user01
0001	job	lead_engineer

GSI

context	data	employee_id
team	hoge	0001
employee_name	user01	0001
job	lead_engineer	0001

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

GSI OVERLOADINGを使いこなそう

Partition Key	Sort Key				
team	employee_id	employee_name	job	skills	age
hoge	0001	user01	lead_engineer	[“C”, “C++”, “C#”, “Java”, “Rust”]	35
hoge	0002	user02	engineer	[“Java”, “Rust”]	29
fuga	0003	user03	lead_engineer	[“Ruby”, “Node.js”, “Deno”]	30
fuga	0004	user04	engineer	[“Node.js”]	24

- Partition Keyのみで検索することでチーム毎のデータを取得可能
- Partition Key + Sort Keyの検索により特定社員のデータを取得可能

?

- 社内に存在するチームの一覧
- 社員の一覧
- どちらもScanするかGSIを作れば可能
- この例ではGSIの上限20以内で収めることも可能だろう
- もっと複雑だと・・・？

aws

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

In Partnership with intel

GSI OVERLOADINGを使いこなそう

Partition Key	
context	data
team	hoge
employee_name	user01
job	lead_engineer
skills	[“C”, “C++”, “C#”, “Java”, “Rust”]
age	35
team	fuga
employee_name	user04
job	engineer
skills	[“Node.js”]
age	24

- 1つのGSIで非常に多くの範囲データを取得可能
- GSIの節約 = Provisioned CUの節約

チーム一覧

context	data
team	hoge
team	fuga

社員一覧

context	data
employee_name	user01
employee_name	user04

aws

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

In Partnership with intel.

GSI OVERLOADINGを使いこなそう

Partition Key	Sort Key					
context	employee_id	team	employee_name	job	skills	age
team	0001	hoge				
employee_name	0001		user01			
job	0001			lead_engineer		
skills	0001				[“C”, “C++”, “C#”, “Java”, “Rust”]	
age	0001					35
team	0002	fuga				
employee_name	0002		user02			
job	0002			engineer		
skills	0002				[“Java”, “Rust”]	
age	0002					29

こんなパターンもあり。
気持ち悪いかもしれないがDynamoDBなら全然アリ。

これなら各種別の一覧も取得できるし、
社員毎の属性もピンポイントに取得可能。

先程の例のような汎用的な属性名でもないので、
わかりやすい

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

In Partnership with intel

勘所

- ただ1つのやってはいけないこと
- 設計段階からDynamoDBの便利機能を考慮しよう
- アイテムサイズを減らす努力をしよう
- 本当に最新データが必要なのか考え直そう
- 効率的なデータ取得のためにGSI OVERLOADINGを使いこなそう
- **DynamoDB LocalやWorkBenchを使って徹底的な設計を行おう**

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

DynamoDB Localは素晴らしい

- AWS謹製のローカル環境向けDynamoDB
  - 詳しい使い方: https://docs.aws.amazon.com/ja_jp/amazondynamodb/latest/developerguide/DynamoDBLocal.UsageNotes.html
- Dockerコンテナとして提供されているので、今すぐ簡単に使い始められる
  - https://hub.docker.com/r/amazon/dynamodb-local
- TTLやStreamsにも対応
- aws-sam-cliと組み合わせればローカル環境でサーバーレス開発を簡単に始められる！！
  - Triggersっぽいことも可能

1 データが揮発しても良いなら

```
$ docker run -d ¥
> -p 8000:8000 ¥
> amazon/dynamodb-local:latest ¥
> -jar DynamoDBLocal.jar ¥
> -sharedDb
```

オススメ

2 データを永続したいなら

```
$ docker run -d ¥
> -p 8000:8000 ¥
> -v `pwd`/db:/home/dynamodblocal/db ¥
> amazon/dynamodb-local:latest ¥
> -jar DynamoDBLocal.jar ¥
> -sharedDb ¥
> -dbPath /home/dynamodblocal/db
```

コレ

コレ

/home/dynamodblocal/shared-local-instance.db
を外に出してあげる。
ただし、/home/dynamodblocalを
そのままマウントすると問題がある。
※DynamoDBLocal.jarが存在しないことになる。
なので”-dbPath”でDB保存先を切り替えてあげると良い。

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

頭の整理にWorkBenchを使ってみよう

- 正式名称: NoSQL WorkBench for Amazon DynamoDB
  - ダウンロード: https://docs.aws.amazon.com/ja_jp/amazondynamodb/latest/developerguide/workbench.settingup.html
- 設計を手助けしてくれるデータモデリングツール
- 多くの場合、エクセルなどの表計算ツールでデータモデリングしていたのではないか？
  - 悪いことは言わないから、試しにWorkBench使ってみて！！！！

WorkBenchの機能

- **Data modeler**
  - 自由自在にデータモデルを作成したり編集できる機能
  - GSIも作成可能、LSIは非対応
  - 既存のデータモデルをインポートしたり、作成したデータモデルをエクスポート可能
- **Visualizer**
  - Data modelerで作成したデータモデルに実データを投入できる
  - 投入したデータを可視化
  - リモート/ローカルのDynamoDBにデータを反映可能
- **Operation builder**
  - リモート/ローカルのDynamoDBからテーブルを読み込み各種操作が可能
  - 対象テーブル操作のサンプルコードの生成が可能

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

In Partnership with intel

頭の整理にWorkBenchを使ってみよう

NoSQL Workbench for Amazon DynamoDB

Application Edit View Window Help

aws NoSQL Workbench

- Data modeler
- Visualizer
- Operation builder

Data modeler

+ Create data model

Import data model　Export data model

Recent data models

Employee Data Model

Edit metadata

Author: Amazon Web Services, Inc.
Created: Sep 05, 2019, 11:50 AM
Last modified: Oct 04, 2020, 11:39 AM
Description: This data model represents an Amazon DynamoDB schema for an employee database application.

The important access patterns facilitated by this data model are:
* Retrieval of an employee record using the employee's login alias, facilitated by a table called Employee
* Search for employees by name, facilitated by the Employee

Delete　Open

Teams

Edit metadata

Author: Ryo Sasaki
Created: Oct 07, 2020, 08:57 PM
Last modified: Oct 07, 2020, 09:51 PM
Description: Sample Table for AWS DevDay 2020

Delete　Open

自作データモデル

サンプルデータもいくつか用意されている

Sample data models

Introductory

AWS Discussion Forum Data Model

Author: Amazon Web Services, Inc.
Created: Sep 05, 2019, 11:50 AM
Last modified: Sep 05, 2019, 11:50 AM
Description: This data model represents Amazon DynamoDB schema for AWS discussion forums, an example of an application for discussion forums or message boards. Using AWS discussion forums, customers can engage with the developer community, ask questions, and reply to other customers' posts. Each AWS service has a dedicated forum. Anyone can start a new discussion thread by posting a message in a forum, and each

Open

Bookmarks Data Model

Author: Amazon Web Services, Inc.
Created: May 1, 2020, 11:00 AM
Last modified: May 1, 2020, 11:00 AM
Description: This model is about storing URL bookmarks for customers.

Even if the use case is relatively simple, there are still many interesting considerations to make. A customer can have many bookmarks and a bookmark can be for many customers. So we have to model a "many to many" relationship. A first approach is to create two tables, one for customers, and one for

Open

Employee Data Model

Author: Amazon Web Services, Inc.
Created: Sep 05, 2019, 11:50 AM
Last modified: Sep 05, 2019, 11:50 AM
Description: This data model represents an Amazon DynamoDB schema for an employee database application.

The important access patterns facilitated by this data model are:
* Retrieval of an employee record using the employee's login alias, facilitated by a table called Employee

Open

Ski Resort Data Model

Author: Amazon Web Services, Inc.
Created: May 1, 2020, 11:00 AM
Last modified: May 1, 2020, 11:00 AM

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

In Partnership with

頭の整理にWorkBenchを使ってみよう

NoSQL Workbench for Amazon DynamoDB

Application Edit View Window Help

aws NoSQL Workbench

Data modeler

Visualizer

Operation builder

Data modeler + Create data model

Import data model Export data model

Data model

hoge

Tables Add table

Visualize data model

Add DynamoDB table

* Table name Please enter table name

Primary key attributes

* Partition key Partition key String

Add sort key

Other attributes

* Attribute name Attribute name String

+ Add other attribute

Add facets

Global secondary indexes

Global secondary indexes

* Global secondary index name Global secondary index name

* Partition key Attribute

Add sort key

Projection type ALL

作成したデータモデルは
エクスポート可能

GSIも設定可能

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

頭の整理にWorkBenchを使ってみよう

NoSQL Workbench for Amazon DynamoDB

Application Edit View Window Help

aws NoSQL Workbench

- Data modeler
- Visualizer
- Operation builder

Visualizer

Import data model | Export data model

Data model

Teams

Teams Update

Aggregate view

Commit to DynamoDB

FACET Teams

Add data | Edit data

team (Partition key) : String	employee_id (Sort key) : Number	employee_name : String	job : String	skills : String Set	age : Number
hoge	0001	user_01	manager	["Management"]	41
hoge	0002	user_02	lead_engineer	["C","C++","C#","Java"…	39
hoge	0003	user_03	engineer	["Java","C#","Rust"]	34
hoge	0004	user_04	engineer	["Java","C#"]	33
hoge	0005	user_05	engineer	["Java"]	32
fuga	0006	user_06	manager	["Management"]	39
fuga	0007	user_07	lead_engineer	["PHP","Ruby","Node.js"]	34
fuga	0008	user_08	engineer	["PHP","Ruby","Node.js"]	29
fuga	0009	user_09	engineer	["Ruby","Node.js"]	29
fuga	0010	user_10	engineer	["PHP","Node.js"]	27
foo	0011	user_11	manager	["Management"]	35
foo	0012	user_12	lead_engineer	["Python","Node.js","D…	30

自由にデータを
追加/更新できる

リモート/ローカルの
DynamoDBに
反映できる

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

In Partnership with intel.

頭の整理にWorkBenchを使ってみよう

NoSQL Workbench for Amazon DynamoDB

Application Edit View Window Help

aws NoSQL Workbench

Data modeler

Visualizer

Operation builder

Operation builder + Add connection

Active connections

Remote Prod

Open

Local Local

Open

Add a new database connection

A remote connection lets you access the DynamoDB web service in different AWS Regions.
A DynamoDB local connection lets you access the DynamoDB local server running on your computer.

Remote DynamoDB local

* Connection name: Connection 1

* Default AWS Region: Default AWS Region

* Access key ID: AWS Access key ID

* Secret access key: AWS secret access key

Session token: AWS session token

IAM role ARN: IAM role ARN

Persist connection

If you select this check box, AWS connection secrets will be persisted in C:¥Users¥gzock¥.aws¥credentials

Cancel Reset Connect

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

In Partnership with intel.

頭の整理にWorkBenchを使ってみよう

NoSQL Workbench for Amazon DynamoDB

Application Edit View Window Help

aws NoSQL Workbench

- Data modeler
- Visualizer
- Operation builder

Operation builder

+ Add connection

Connection

Name

Local

Tables

Search table

Teams

TABLE Teams

Build operations

Items Metadata

Negate Attribute name = Attribute type Attribute value Scan

#	team	employee_id	skills	employee_name	job	age
1	fuga	10	["Node.js","PHP"]	user_10	engineer	27
2	fuga	9	["Node.js","Ruby"]	user_09	engineer	29
3	fuga	8	["Node.js","PHP","R···	user_08	engineer	29
4	fuga	7	["Node.js","PHP","R···	user_07	lead_engineer	34
5	fuga	6	["Management"]	user_06	manager	39

リモート/ローカルの
DynamoDBからテー
ブルを読み込める

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

In Partnership with intel.

頭の整理にWorkBenchを使ってみよう

NoSQL Workbench for Amazon DynamoDB

Application Edit View Window Help

aws NoSQL Workbench

Data modeler

Operation builder + Add connection

Query operation builder

* Partition key team

Sort key

Projection expression

Filter expression + Condition + Child expression

And Or Negate

* Condition Negate Attribute name = Attribute type Attribute value

Other parameters

Consistent read

Scan Index Forward

Limit

Exclusive start key

Cancel Clear form Execute Generate code

様々な条件をGUIで設定

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

頭の整理にWorkBenchを使ってみよう

NoSQL Workbench for Amazon DynamoDB

Application Edit View Window Help

Projection expression

Filter expression + Condition + Child expression

Other parameters

Consistent read

Scan Index Forward

Limit

Exclusive start key

複数の言語に対応。
今後のVerUPで増えていくかも？

Generated code

Python JavaScript (Node.js) Java

```
                error_message=error_message))

def main():
    # Create the DynamoDB Client with the region you want
    dynamodb_client = create_dynamodb_client(region="eu-west-1")

    # Create the dictionary containing arguments for query call
    query_input = create_query_input()

    # Call DynamoDB's query API
    execute_query(dynamodb_client, query_input)


if __name__ == "__main__":
    main()
```

サンプルコードを自動的に生成可能。
※慣れてきたら自分で書いたほうが早いけど、
最初は十分参考になるのでは？

Cancel Clear form Execute Generate code

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

勘所

- ただ1つのやってはいけないこと
- 設計段階からDynamoDBの便利機能を考慮しよう
- アイテムサイズを減らす努力をしよう
- 本当に最新データが必要なのか考え直そう
- 効率的なデータ取得のためにGSI OVERLOADINGを使いこなそう
- DynamoDB LocalやWorkBenchを使って徹底的な設計を行おう

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

In Partnership with intel.

参考

余裕が出てきたら、下記資料を是非ご確認ください。非常に参考になります。

- DynamoDB を使用した設計とアーキテクチャの設計に関するベストプラクティス
  https://docs.aws.amazon.com/ja_jp/amazondynamodb/latest/developerguide/best-practices.html

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

aws DEV DAY
ONLINE JAPAN

Thank you!

In Partnership with intel

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.