Amazon DynamoDB Advanced Design Pattern

Solutions Architect 成田　俊
2018/12/25

AWS 公式 Webinar
https://amzn.to/JPWebinar

過去資料
https://amzn.to/JPArchive

AWS Black Belt Online Seminar とは

「サービス別」「ソリューション別」「業種別」のそれぞれのテーマに分かれて、アマゾン ウェブ サービス ジャパン株式会社が主催するオンラインセミナーシリーズです。

質問を投げることができます！

- 書き込んだ質問は、主催者にしか見えません
- 今後のロードマップに関するご質問は
  お答えできませんのでご了承下さい

① 吹き出しをクリック
② 質問を入力
③ Sendをクリック

Twitter ハッシュタグは以下をご利用ください
#awsblackbelt

内容についての注意点

- 本資料では2018年12月25日時点のサービス内容および価格についてご説明しています。最新の情報はAWS公式ウェブサイト(http://aws.amazon.com)にてご確認ください。

- 資料作成には十分注意しておりますが、資料内の価格とAWS公式ウェブサイト記載の価格に相違があった場合、AWS公式ウェブサイトの価格を優先とさせていただきます。

- 価格は税抜表記となっています。日本居住者のお客様が東京リージョンを使用する場合、別途消費税をご請求させていただきます。

- AWS does not offer binding price quotes. AWS pricing is publicly available and is subject to change in accordance with the AWS Customer Agreement available at http://aws.amazon.com/agreement/. Any pricing information included in this document is provided only as an estimate of usage charges for AWS services based on certain information that you have provided. Monthly charges will be based on your actual use of AWS services, and may vary from the estimates provided.

本日のアジェンダ

- はじめに
- 直近のUpdate
- NoSQL データモデリング
- DynamoDB を活用したデザインパターン

本日のアジェンダ

- 想定する参加者
  - Amazon DynamoDB を既に利用している
  - Amazon DynamoDB の基本的な用語や機能を知っている
    - 例)パーティション、GSI, LSI など

- お話しする内容
  - Amazon DynamoDB のテーブル設計のポイント
  - ユースケースに応じて、Amazon DynamoDB を
    どのように利用するかの例

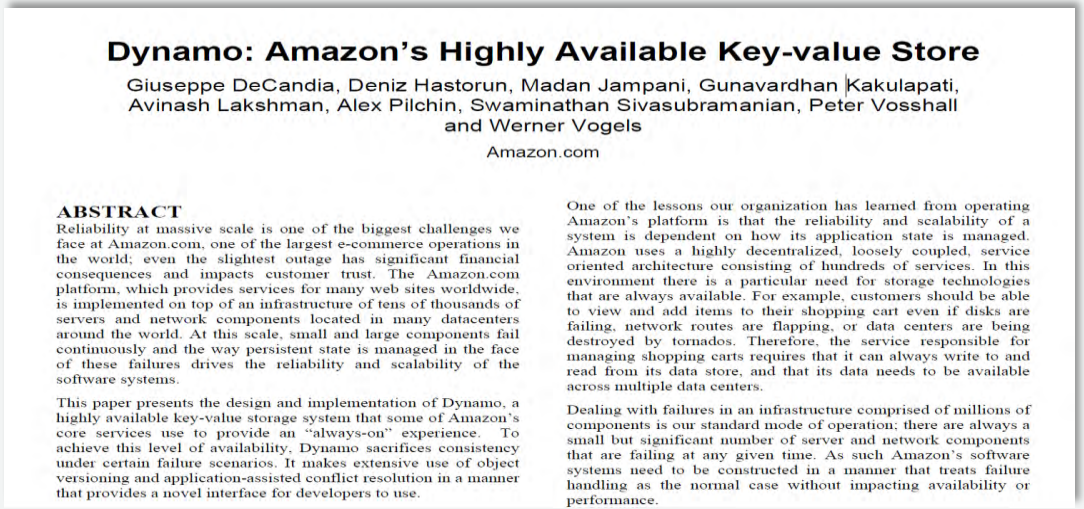
aws

はじめに

Amazon DynamoDBの生い立ち

Amazon.comではかつて全アクセスパターンをRDBMSで処理

RDBMSのスケールの限界を超えるため開発されたDynamoが祖先



Dynamo: Amazon's Highly Available Key-value Store

Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall and Werner Vogels

Amazon.com

ABSTRACT

Reliability at massive scale is one of the biggest challenges we face at Amazon.com, one of the largest e-commerce operations in the world; even the slightest outage has significant financial consequences and impacts customer trust. The Amazon.com platform, which provides services for many web sites worldwide, is implemented on top of an infrastructure of tens of thousands of servers and network components located in many datacenters around the world. At this scale, small and large components fail continuously and the way persistent state is managed in the face of these failures drives the reliability and scalability of the software systems.

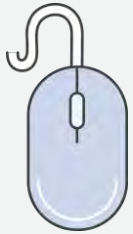
This paper presents the design and implementation of Dynamo, a highly available key-value storage system that some of Amazon's core services use to provide an "always-on" experience. To achieve this level of availability, Dynamo sacrifices consistency under certain failure scenarios. It makes extensive use of object versioning and application-assisted conflict resolution in a manner that provides a novel interface for developers to use.

One of the lessons our organization has learned from operating Amazon's platform is that the reliability and scalability of a system is dependent on how its application state is managed. Amazon uses a highly decentralized, loosely coupled, service oriented architecture consisting of hundreds of services. In this environment there is a particular need for storage technologies that are always available. For example, customers should be able to view and add items to their shopping cart even if disks are failing, network routes are flapping, or data centers are being destroyed by tornados. Therefore, the service responsible for managing shopping carts requires that it can always write to and read from its data store, and that its data needs to be available across multiple data centers.

Dealing with failures in an infrastructure comprised of millions of components is our standard mode of operation; there are always a small but significant number of server and network components that are failing at any given time. As such Amazon's software systems need to be constructed in a manner that treats failure handling as the normal case without impacting availability or performance.

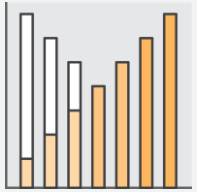
- 結果整合性モデル採用による可用性向上
- HWを追加する毎に性能が向上するスケーラビリティ
- シンプルなクエリモデルによる予測可能な性能

Amazon DynamoDB



完全マネージド型
NoSQL データベースサービス

ドキュメント型 or KVS



高いスケーラビリティ

高速かつ一貫した
パフォーマンス

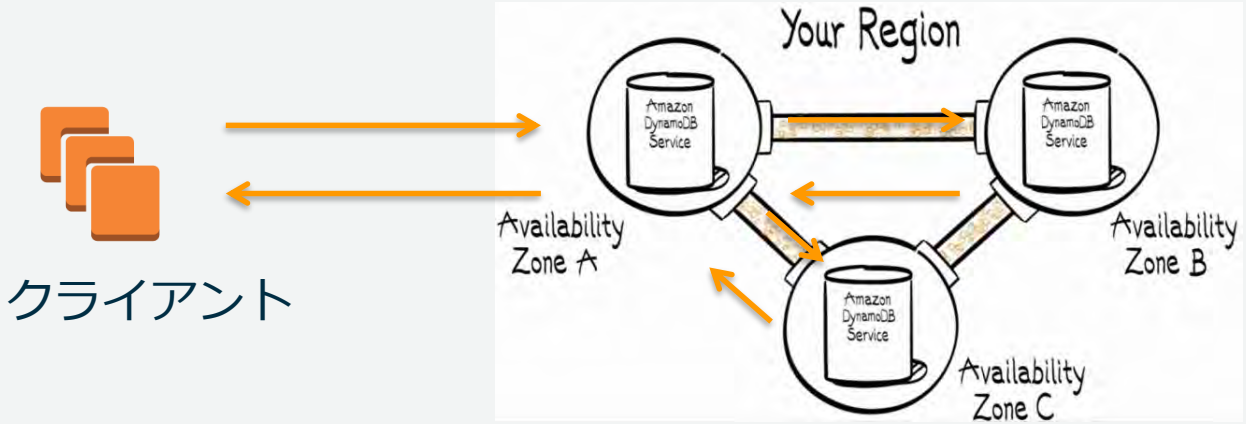
アクセスコントロール

イベントドリブン
プログラミング

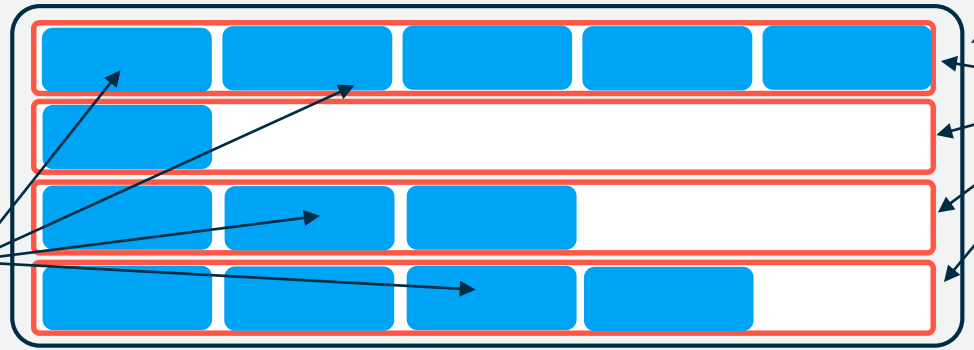
aws

管理不要で高い信頼性

- SPOFの存在しない構成
- データは3箇所のAZに保存されるので信頼性が高い
- ストレージは必要に応じて自動的にパーティショニング



テーブル設計 基礎知識



name/value 型、JSON 型等
アイテム間で不揃いであっても問題ない

Partation Key

Sort Key

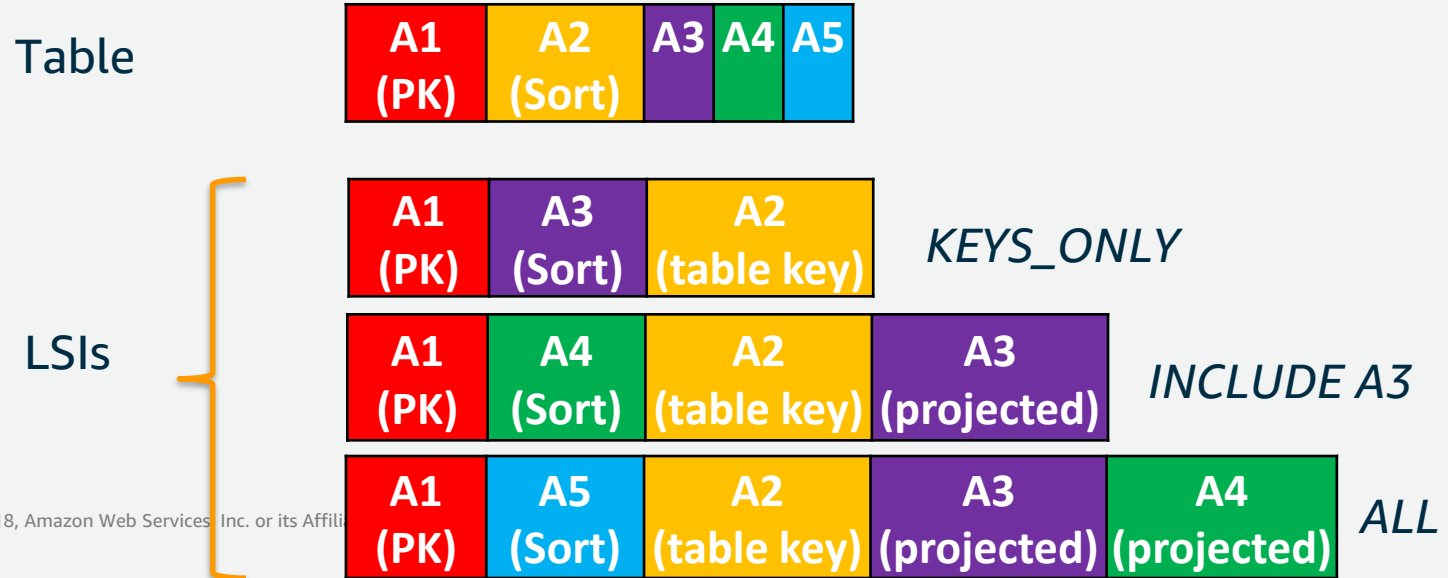
必須
キーバリュー型のアクセスパターン
データ分散に利用される

オプション
1:Nモデルのリレーションシップ
豊富なQueryをサポート
デフォルトの並べ替え順序は昇順

キー検索用
==, <, >, >=, <=
“begins with”
“between”
sorted results
counts
先頭/末尾 N件
ページ単位出力

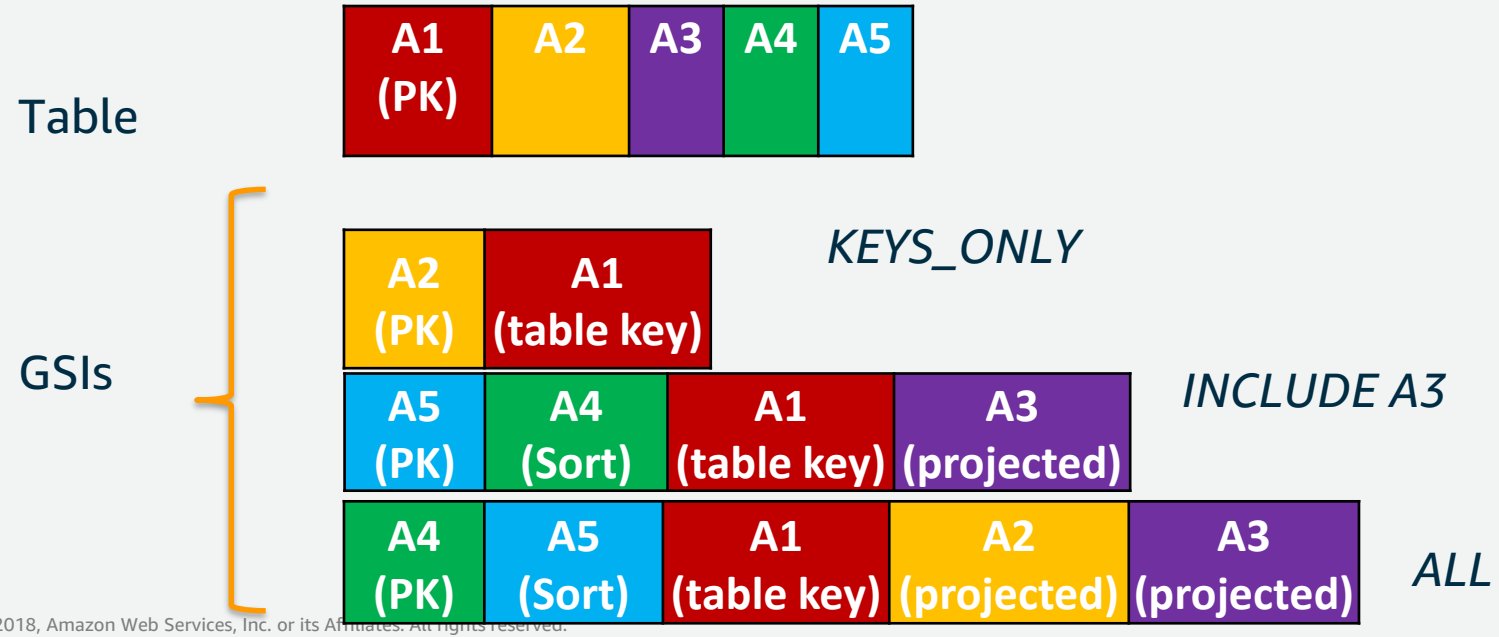
Local Secondary Index (LSI)

- Sort key以外に絞り込み検索を行うkeyを持つことができる
- Partition keyが同一で、他のアイテムからの検索のために利用
- すべての要素（テーブルとインデックス）の合計サイズを、各Partation キーごとに 10 GB に制限



Global Secondary Index (GSI)

Partition Key属性の代わりとなる
Partition Keyをまたいで検索を行うためのインデックス



Update

DynamoDB

Advancements over the last 21 months

February 2017

Time To Live (TTL)

April 2017

VPC endpoints

April 2017

DynamoDB Accelerator (DAX)

June 2017

Auto scaling

November 2017

Global tables

November 2017

On-demand backup

November 2017

Encryption at rest

March 2018

Point-in-time recovery

June 2018

99.999% SLA

August 2018

Adaptive capacity

November 2018

Transactions

November 2018

On-demand

aws

DynamoDB on-demand

DynamoDBで今までcapacity unitをprovisionして利用する必要があったがpay-per-requestモデルでの利用が可能に

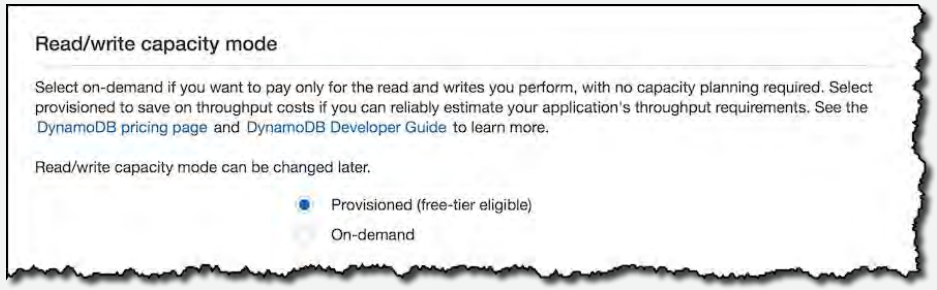
これにより事前のキャパシティユニット設計などが不要

既存テーブルでもどちらを利用するか選択可能

料金例としてVirginia Regionの場合
Write request：100万回につき$1.25
Read request：100万回につき$0.25

利用した分のストレージ課金は従来どおり

全ての商用リージョンで利用可能



DynamoDB Transations

DynamoDBで複数Item、複数tableに対する書き込み操作や読み込み操作でACIDトランザクションが可能に

トランザクション分離レベルはserializableでロックは取らない

トランザクションの進行中にアイテムがトランザクション外で変更された場合、トランザクションはキャンセルされ、例外を発生させたItemまたはItemに関する詳細がスローされる

GlobalTableではデフォルト無効、有効にした場合もトランザクションはリージョン単位

トランザクションを有効にすることに追加料金は無し

全ての商用リージョンで利用可能

Amazon DynamoDB key diagnostics library

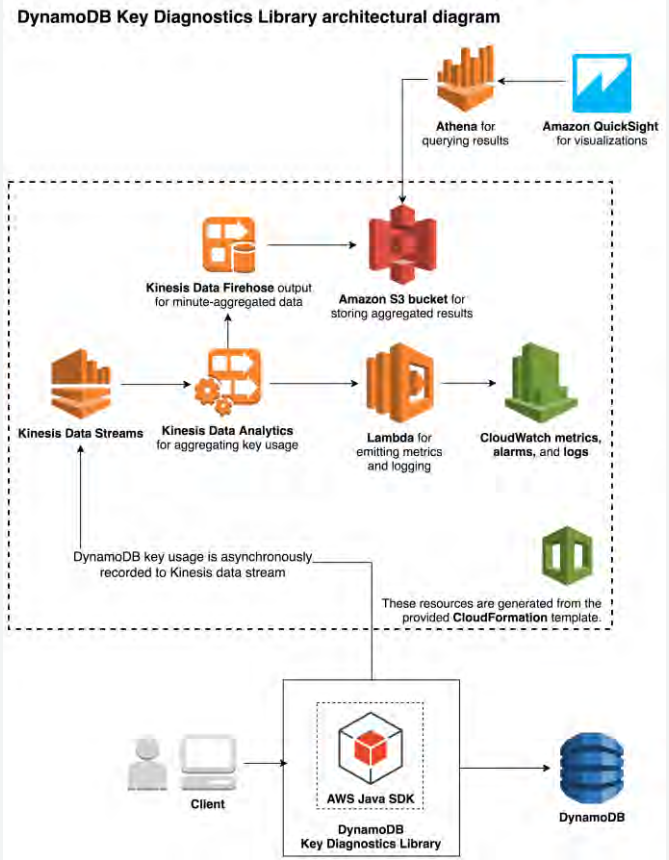
既存のDynamoDB Clientをラップする形で簡単に利用できるライブラリ（Javaのみ）

アクセスパターンの統計情報ログが手軽にkinesisに出力可能

非同期でkinesisに出力なのでパフォーマンス影響は無し

各種解析サービスとの組み合わせでHotkeyなどの発見がより容易に

https://aws.amazon.com/jp/blogs/database/how-to-use-the-new-amazon-dynamodb-key-diagnostics-library-to-visualize-and-understand-your-applications-traffic-patterns/



Global Secondary Indexes の作成数引き上げ

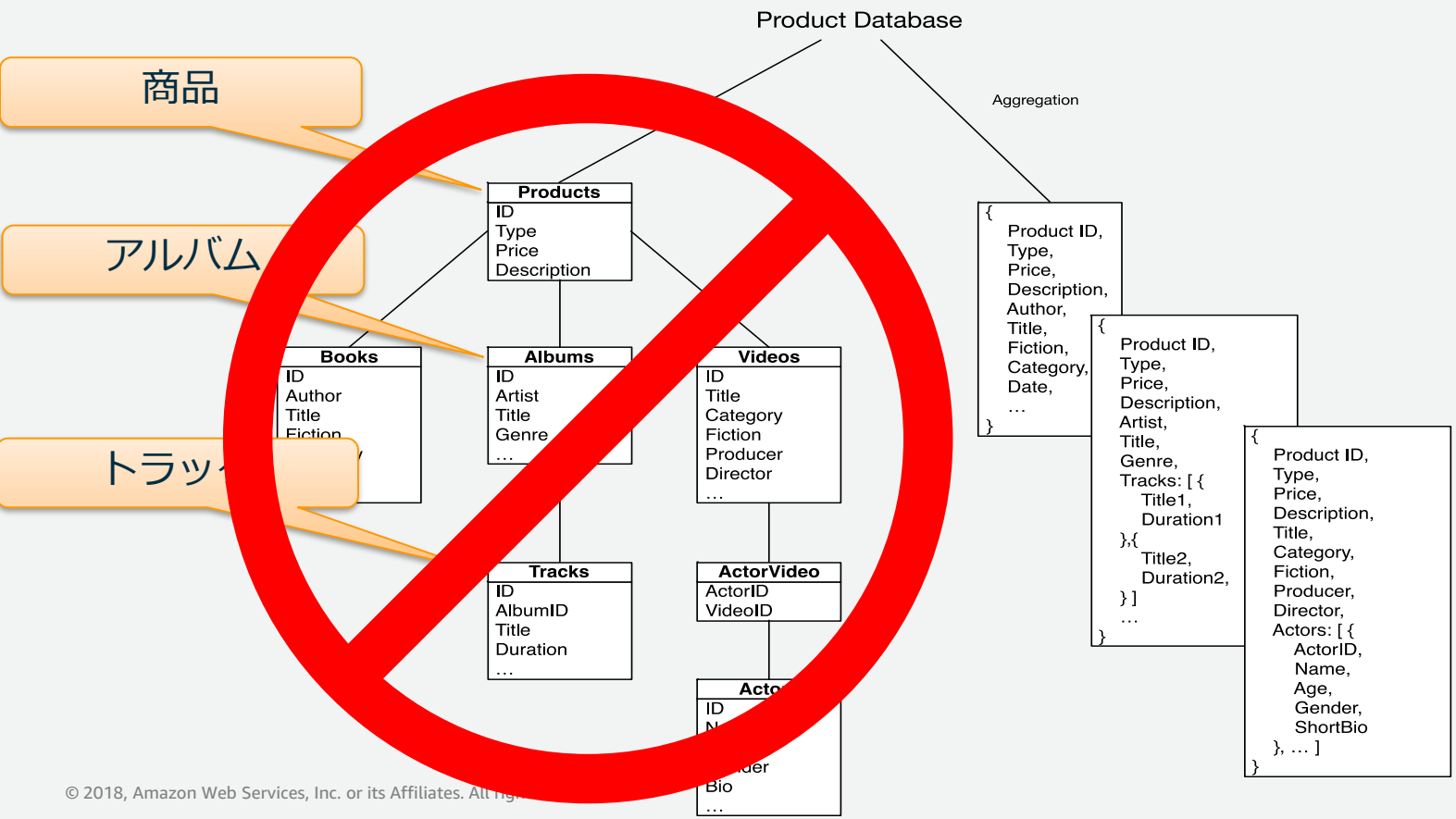
今までTable辺り最大5個までだったGSIが初期20個までに引き上げ

更に必要な場合は上限緩和申請を

GSIへ射影出来る属性が100まで可能に

NoSQL Data Modeling

SQL vs. NoSQL design pattern



1:1 リレーション or キー・バリュー型

- Partition keyを使ったテーブルまたはGSI
- GetItem かBatchGetItem APIを使用
- 例: UserIDやEmailから要素を抽出する場合

Users Table	
Partition key	**Attributes**
UserId = bob	Email = bob@gmail.com, JoinDate = 2011-11-15
UserId = fred	Email = fred@yahoo.com, JoinDate = 2011-12-01

Users-email-GSI	
Partition key	**Attributes**
Email = bob@gmail.com	UserId = bob, JoinDate = 2011-11-15
Email = fred@yahoo.com	UserId = fred, JoinDate = 2011-12-01

1:N リレーション or 親子関係

- Partition key とSort key を使ったテーブル、GSI
- Query APIを使ってアクセス
- 例:1ユーザでN個のゲームをプレイしている場合

User-Games		
Partition Key	Sort key	Attributes
UserId = bob	GameId = Game1	HighScore = 10500, ScoreDate = 2011-10-20
UserId = fred	GameId = Game2	HIghScore = 12000, ScoreDate = 2012-01-10
UserId = bob	GameId = Game3	HighScore = 20000, ScoreDate = 2012-02-12

N:M リレーション

- table と GSI を使用してPartition key とSort key の要素をスイッチして設計
- Query API を用いてアクセス
- 例: 1ユーザが複数のゲームをプレイし,1ゲームで複数のプレイヤーがゲームをしている場合

User-Games-Table	
Partition Key	**Sort key**
UserId = bob	GameId = Game1
UserId = fred	GameId = Game2
UserId = bob	GameId = Game3

Game-Users-GSI	
Partition Key	**Sort key**
GameId = Game1	UserId = bob
GameId = Game2	UserId = fred
GameId = Game3	UserId = bob

Targeting queries

Query filters, composite keys, and sparse indexes

aws

複数の値を条件にしたソート・絞り込み

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

aws

Approach 1: Query filter(Filter条件の利用)

アクセスイメージ

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

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(キーの結合)

Status
DONE
IN_PROGRESS
IN_PROGRESS
PENDING
PENDING

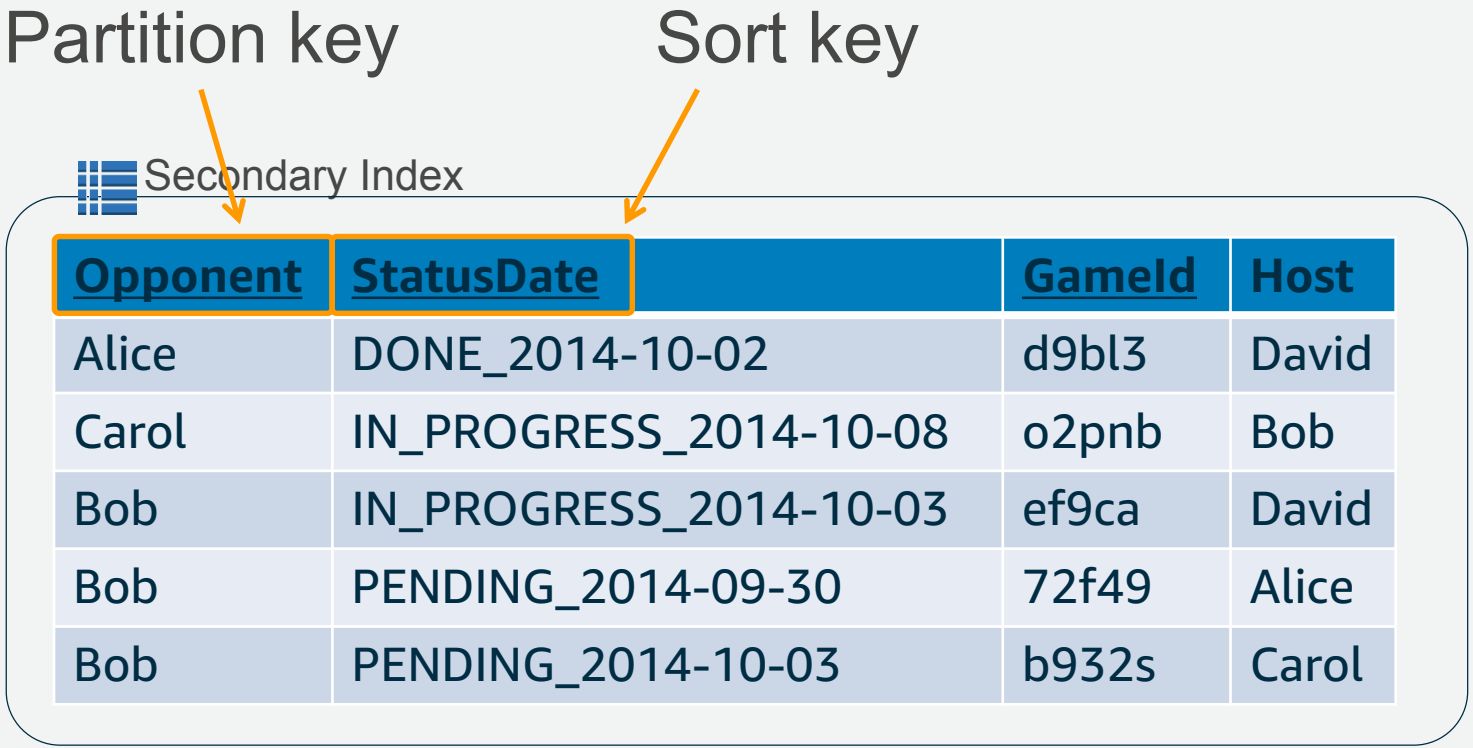
+

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(キーの結合)



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

aws

Approach 2: Composite key(キーの結合)

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

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

Sparse indexes

Game-scores-table

Id (Partition)	User	Game	Score	Date	Award
1	Bob	G1	1300	2012-12-23	
2	Bob	G1	1450	2012-12-23	
3	Jay	G1	1600	2012-12-24	
4	Mary	G1	2000	2012-10-24	Champ
5	Ryan	G2	123	2012-03-10	
6	Jones	G2	345	2012-03-20	

Scan sparse GSIs

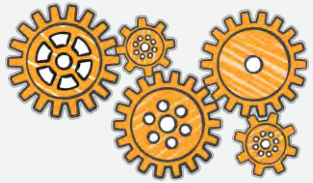
Award-GSI

Award (Partition)	Id	User	Score
Champ	4	Mary	2000

テーブル全体を Scan する場合に比べて
少ないコストで Scan, Query が可能

aws

フィルターの代わりにインデックスを活用



- Attribute の結合(Composite Key)により、
  より効果的なセカンダリインデックスキーを実現

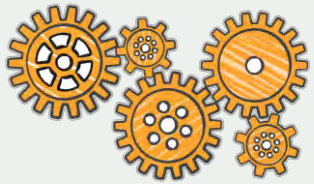
- Sparse indexes によりクエリコストを削減

クエリを最適化したい場面で効果的

aws

GSI Overloading

Holding many different types of data
Beyond GSI limitation



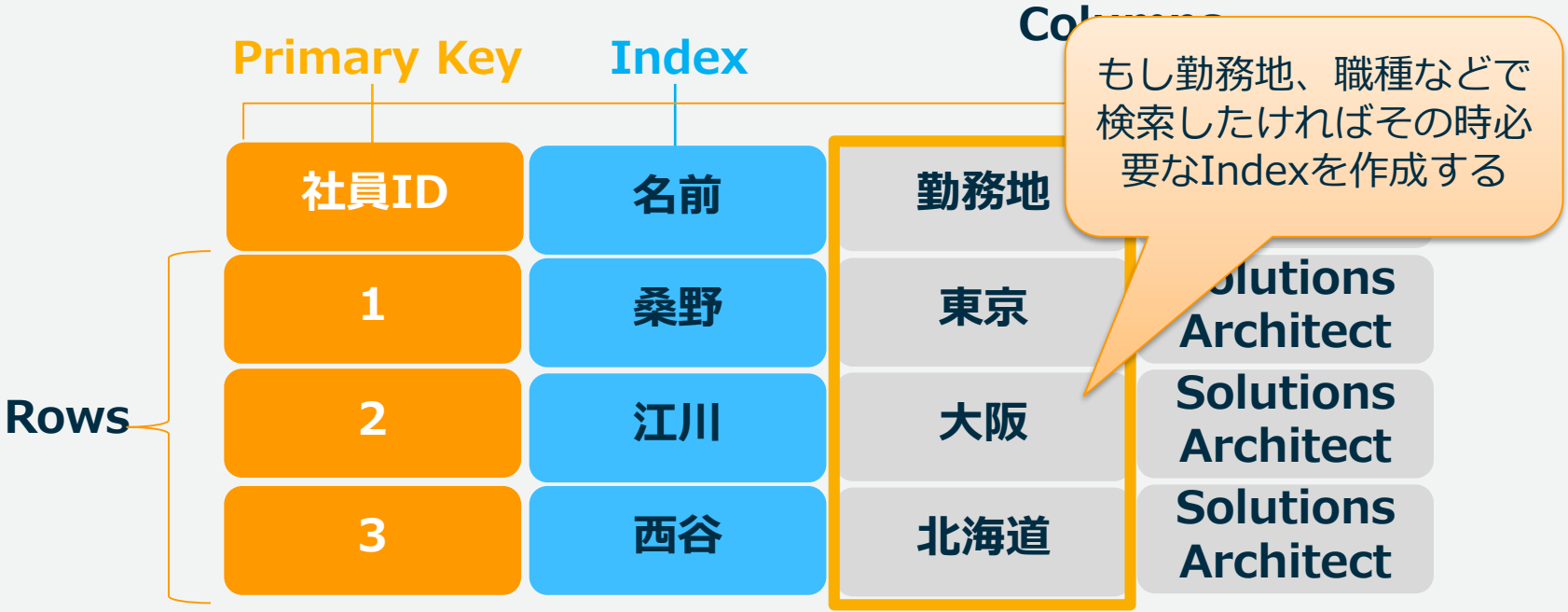
aws

GSI OVERLOADING(GSI の多重定義)

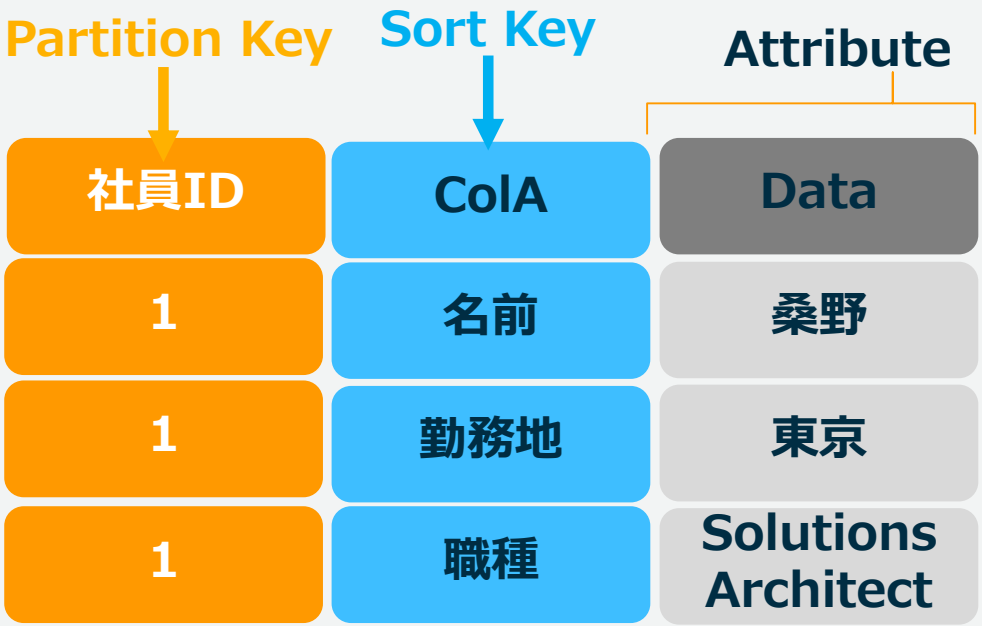
- GSI の制限：テーブルあたりデフォルト最大20まで
- DynamoDB テーブルでは異なるデータを1つの item で保持可能
- 一つのGSIで複数の用途で利用できる様に定義(Overload)
- Overloadするattributeの値はitemのcontextが分かる値にする

例: 従業員テーブルにたいして以下の情報を元にクエリしたい場合
(1) 従業員名 (2)デスク(3) 採用日(4) 四半期ごとの売上高
(5) ジョブロール/役職 (6) 住所 (7)ロケーションion (8) 従業員ID
などなど

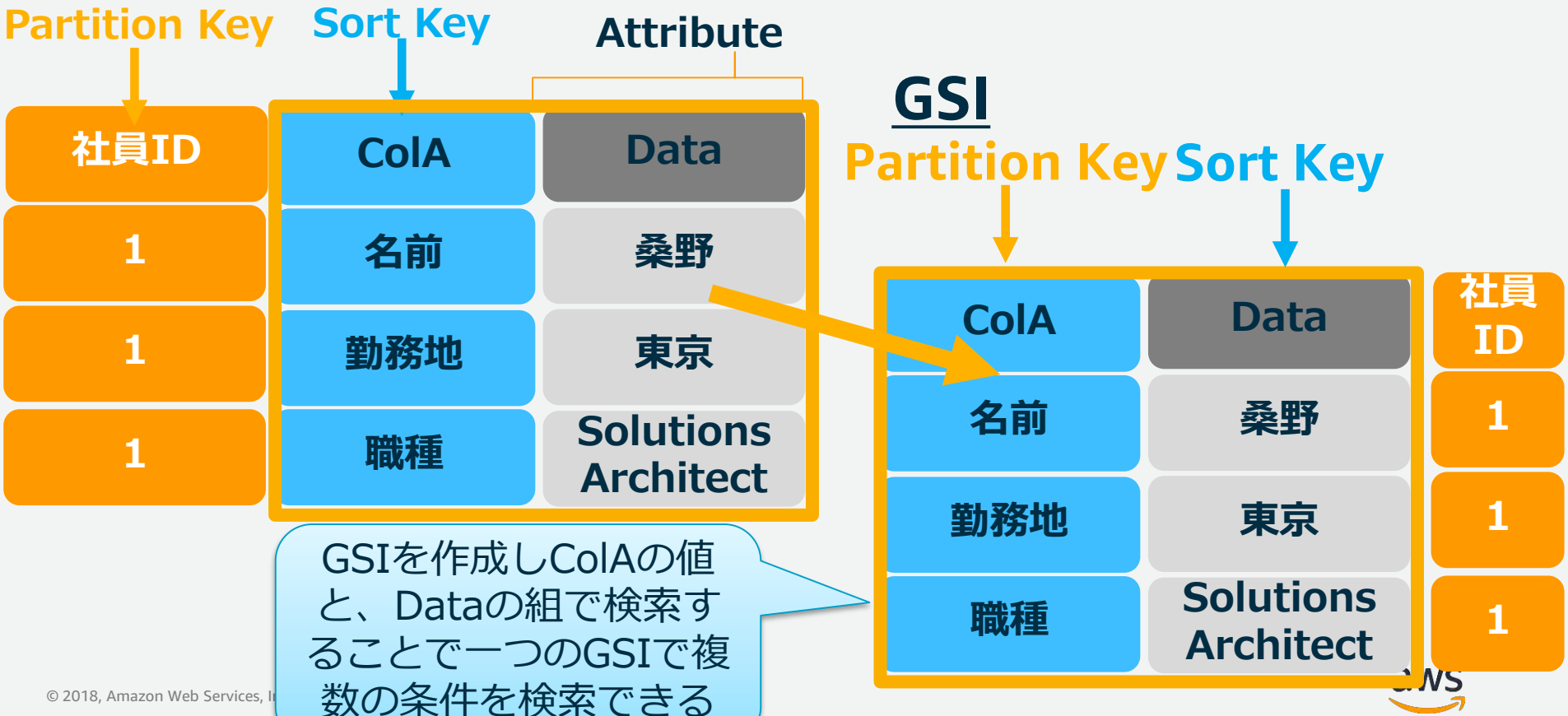
RDBMSの場合



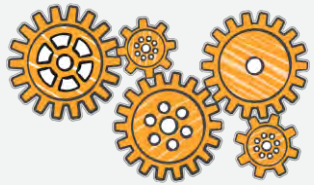
GSI OVERLOADINGの例



GSI OVERLOADINGの例



GSI OVERLOADING(GSI の多重定義)



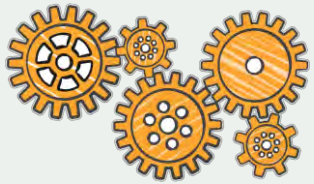
- GSI のテーブルあたりの作成数に関する制限やGSIが検索要件に応じて増えてしまう事を回避
- テーブル構成としては複雑になる点に注意
  - RDBMS で実現する場合に比べて、DynamoDB で得たいメリットを整理

多数の異なるデータをもとにクエリをかけたい場面で効果的

aws

Vertical Pertitioning (メッセージアプリ)
Large Items

Filters vs Indexes
M:N modeling – Inbox & Sent Items



aws

メッセージアプリケーション

Messages App

Inbox

```
SELECT *
FROM Messages
WHERE Recipient='David'
LIMIT 50
ORDER BY Date DESC
```

Messages Table

Outbox

```
SELECT *
FROM Messages
WHERE Sender ='David'
LIMIT 50
ORDER BY Date DESC
```


aws

大小のデータが混在

Partition key

Sort key

Messages Table

Recipient	Date	Sender	Message
David	2014-10-02	Bob	…
… 48 more messages for David …			
David	2014-10-03	Alice	…
Alice	2014-09-28	Bob	…
Alice	2014-10-01	Carol	…

(Many more messages)

Inbox

David

```
SELECT *
FROM Messages
WHERE Recipient='David'
LIMIT 50
ORDER BY Date DESC
```

50 items × 平均 256 KB

大きなメッセージボディーを格納

aws

クエリーコストの計算

$$50_{items} \times 256_{KB} \times \frac{1_{RCU}}{4_{KB}} \times \frac{1_{read}}{2_{e.c.reads}} = \mathbf{1600_{RCU}}$$

1回の問い合わせによって取得されるアイテム数

平均アイテムサイズ

4KB毎に1RCU消費

結果整合性のある読み込み

大きいデータを分けて配置

均等に大きいアイテムを読むように配置

David

(50 sequential items at 128 bytes)

1. Query Inbox-GSI: 1 RCU
2. BatchGetItem Messages: 1600 RCU

(50 separate items at 256 KB)

(Recipientをindex,
Message メタデータを格納)

Inbox-GSI

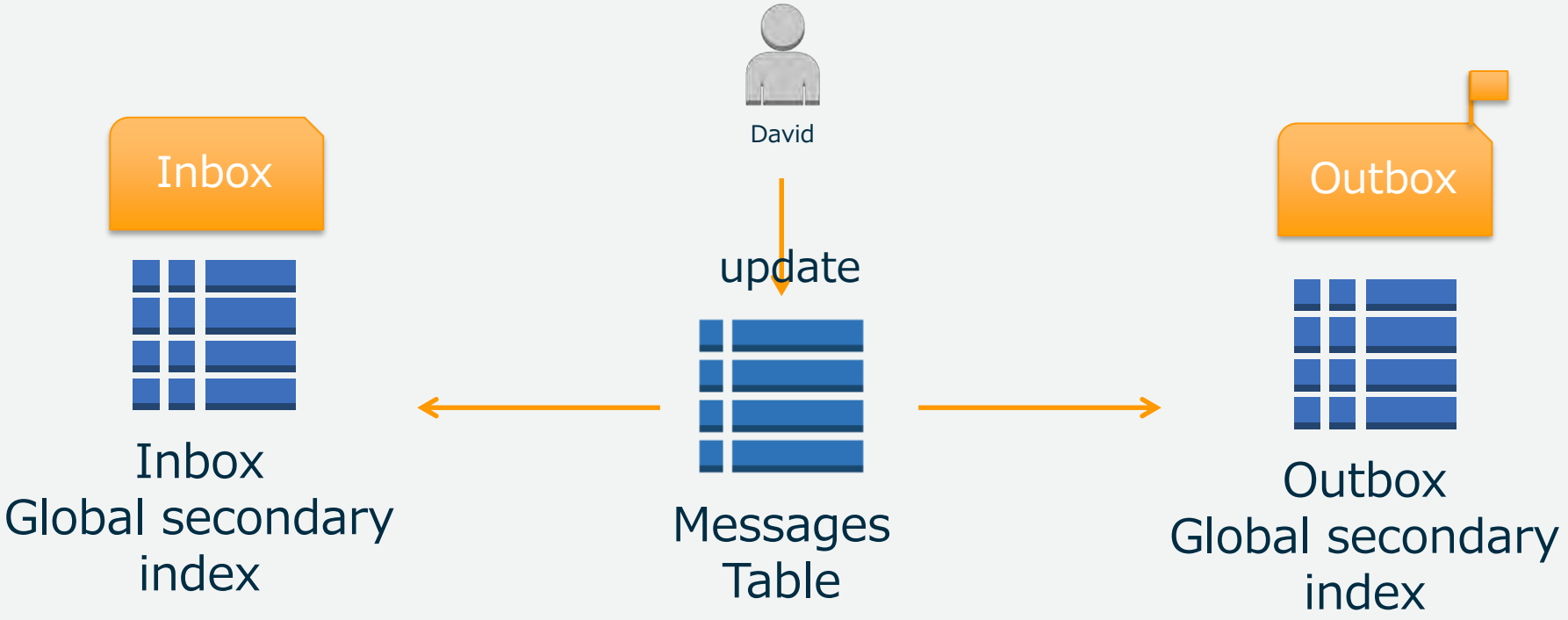
Recipient	Date	Sender	Subject	MsgId
David	2014-10-02	Bob	Hi!...	afed
David	2014-10-03	Alice	RE: The...	3kf8
Alice	2014-09-28	Bob	FW: Ok...	9d2b
Alice	2014-10-01	Carol	Hi!...	ct7r

Messages Table

MsgId	Body
9d2b	...
3kf8	...
ct7r	...
afed	...

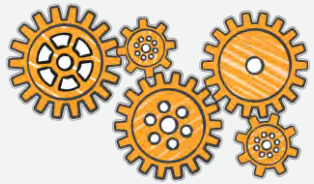
aws

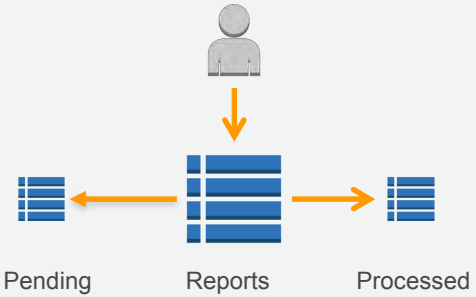
Messaging app



大きい Item の分割

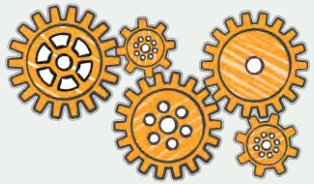
- 1:Nを表す Item サイズの削減
- secondary index の設定
- GSI により M:N リレーションの構築





サイズが大きい Item を一度に多数クエリする際に効果的

Design patterns and best practices

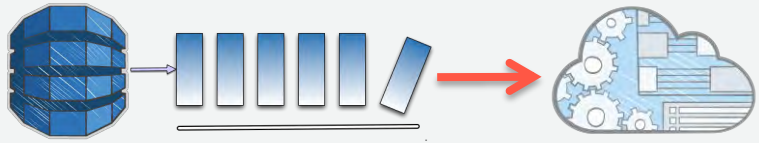


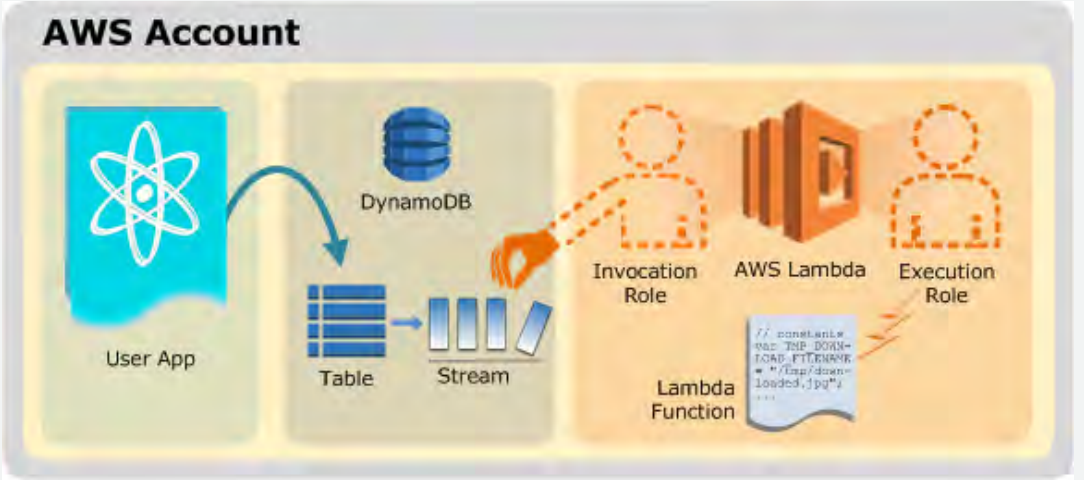
aws

DynamoDB Triggers

DynamoDB + AWS Lambda= DynamoDB Triggers

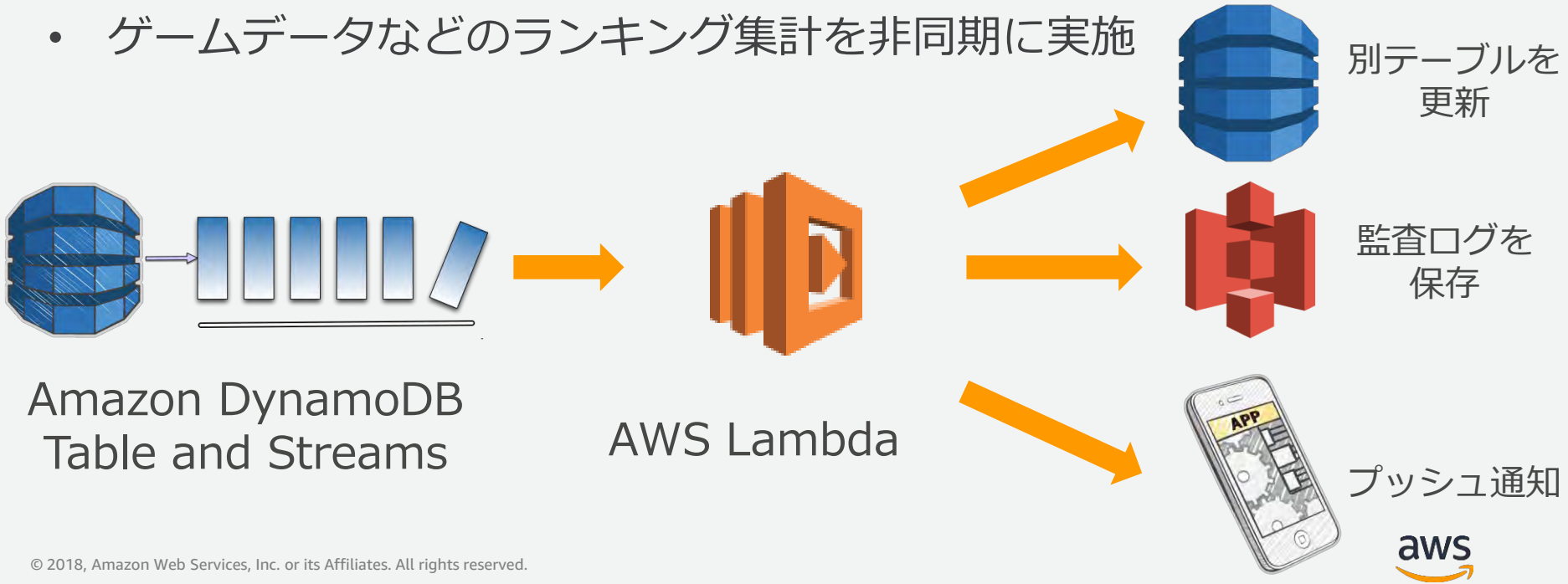
- AWS Lambdaとは
  - OS、キャパシティ等インフラの管理不要
  - S3、Kinesis、SNS等での
    イベント発生を元にユーザが
    用意したコードを実行
  - ユーザアプリからの
    同期/非同期呼び出し





DynamoDB Triggers ユースケース

- DynamoDBへの書き込みに応じて値チェックをしつつ別テーブルの更新やプッシュ通知を実行
- DynamoDBの更新状況の監査ログをS3に保存
- ゲームデータなどのランキング集計を非同期に実施



TTL（Time To Live）

特徴

- Itemの有効期限が切れ、データベースから自動削除されるタイミングを定義可能
- プロビジョニングスループットを使用することなく、関連性のないデータのストレージ使用量と保存コストを削減可能
- 追加料金なし
- 既存・新規のテーブルに設定可能
- **DynamoDB Streamsとの併用可能**

注意点

- 期限切れ後、即削除されるわけではない
  - ➢ 48時間以内に削除
  - ➢ 読み取り時に期限切れのものを取得しないようにするにはQuery or アプリ側でのフィルタが必要

詳細：https://docs.aws.amazon.com/ja_jp/amazondynamodb/latest/developerguide/TTL.html

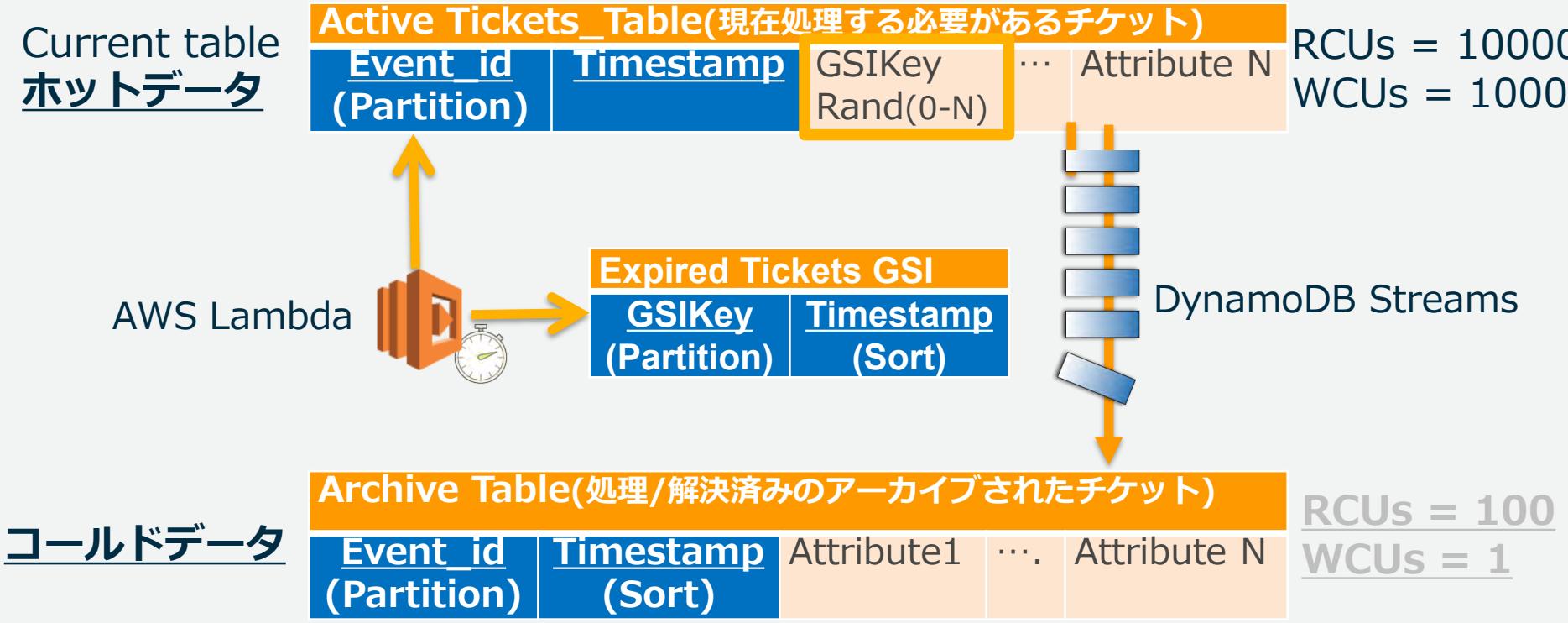
Time-based workflows (チケットシステム)

Processing the entire table efficiently

時系列データが必要なアプリケーション

- ホットデータとコールドデータでテーブルを分ける
- DynamoDB StreamsとGSIを活用し、非同期でホットデータをコールドデータへと移動する
- ホットデータではWCU、RCUを高く設定、コールドデータでは書き込み、読み込みは最低限必要な分のみに限定する
  - コスト最適化

時系列データが必要なアプリケーション



期限切れのデータをTTLを使用してアーカイブ

AWS Lambda を活用してテーブルを効率良く整理

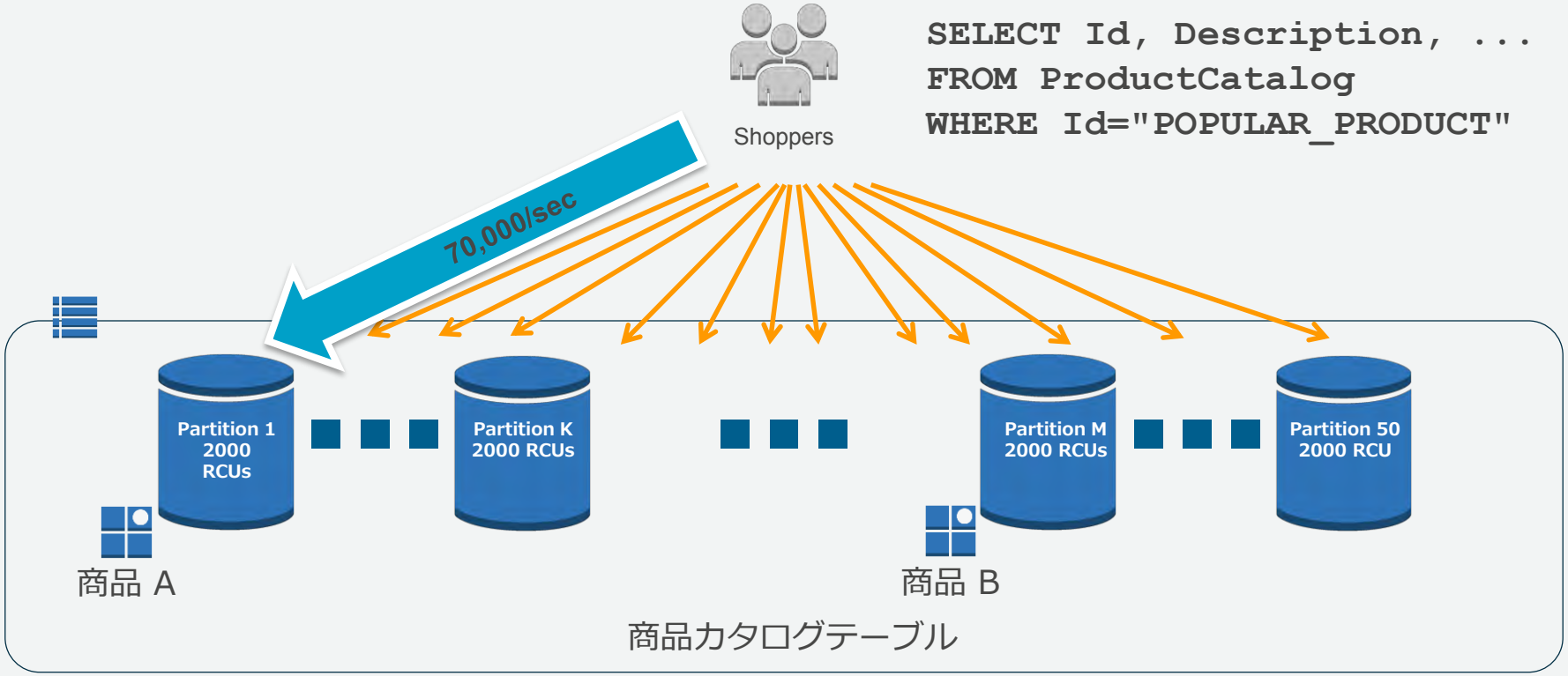
- ホットデータとコールドデータでテーブルを分ける
- テーブル全体に効率的にクエリをかけるために GSI を使用
- Lambda “ストアドプロシージャ”を作成し、Item を遷移
- TTL/DynamoDB Streams/Lambda を利用してテーブル間でデータを移行

全て/多くのItem を効率良くクエリしなければいけない場合に効果的

Product catalog
Popular items (read)

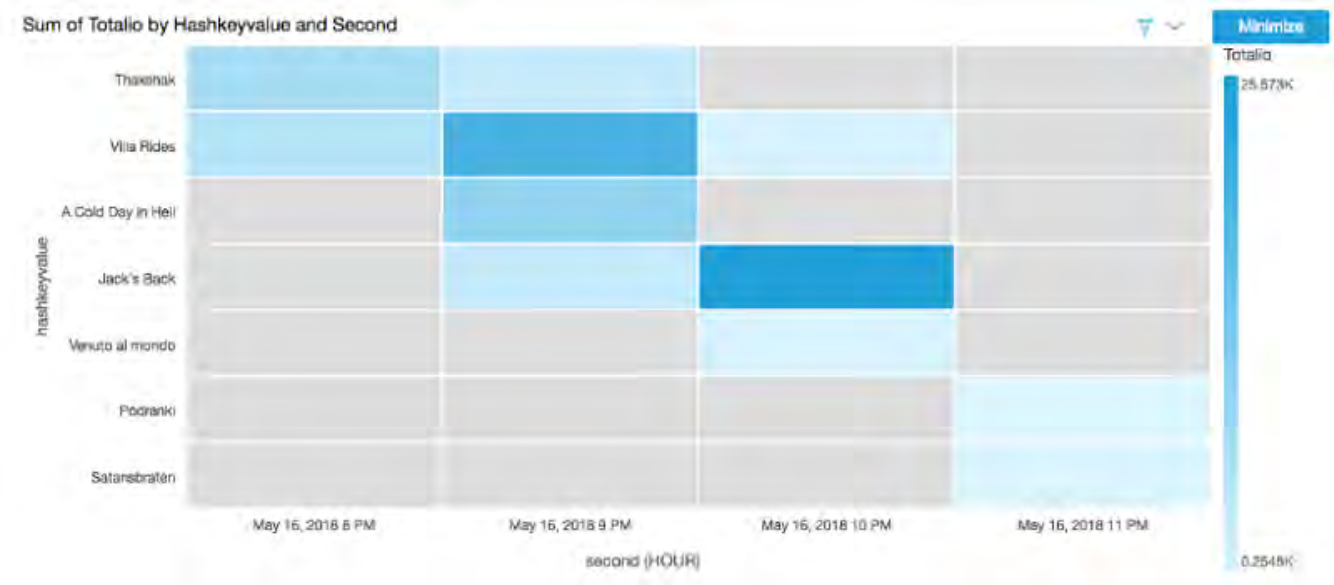
アクセスが集中する特定の Item に対する Read

特定のパーティションに対して集中した読み込み



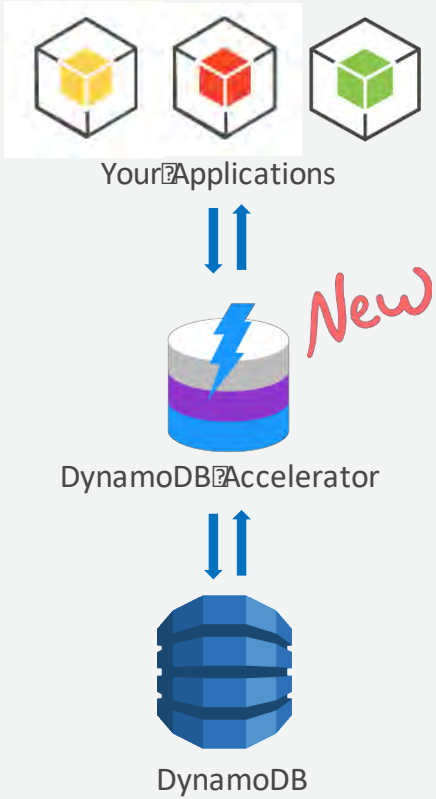
特定のパーティションに対して集中した読み込み

How to use the new Amazon DynamoDB key diagnostics library to visualize and understand your application's traffic patterns



https://aws.amazon.com/jp/blogs/database/how-to-use-the-new-amazon-dynamodb-key-diagnostics-library-to-visualize-and-understand-your-applications-traffic-patterns/

DynamoDB Accelerator (DAX)によるキャッシング



特徴

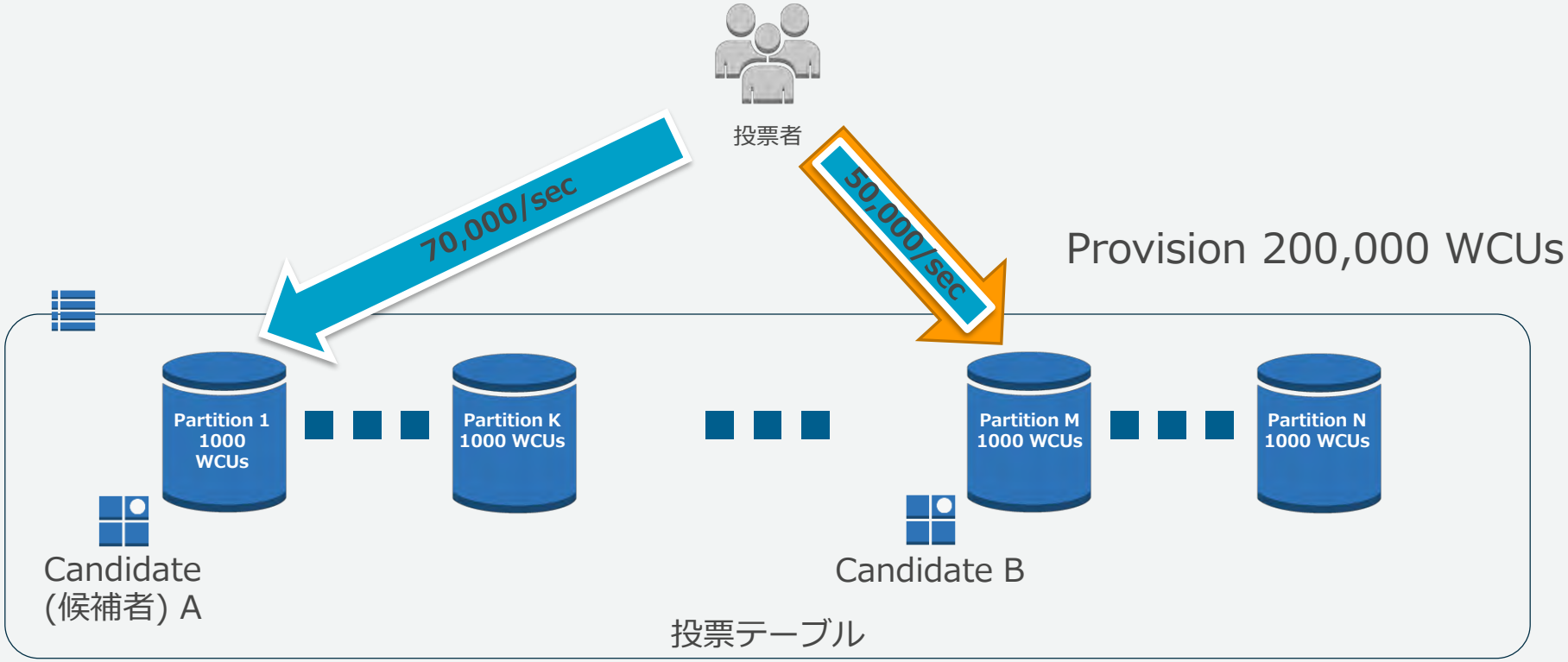
- **フルマネージドかつ高可用性:** リージョン内でマルチAZ構成かつキャッシュ情報のレプリケーション、障害時のフェイルオーバーなどをフルマネージドで実現
- **DynamoDB API互換:** 現在のSDKと互換性を保っているのでコードの大部分は書き直す必要が無い
- **Write-through:** ライトスルーでキャッシュを保持
- **Flexible**: 様々なDynamoDBのテーブル状況に対応
- **Scalable**: 最大10ノードまでのスケールアウトに対応
- **Manageability:** Amazon CloudWatch, Tagging for DynamoDB, AWS Consoleなどとの連携も完備
- **Security:** Amazon VPC, AWS IAM, AWS CloudTrail, AWS Organizationsに対応

Write sharding

Handling high velocity writes

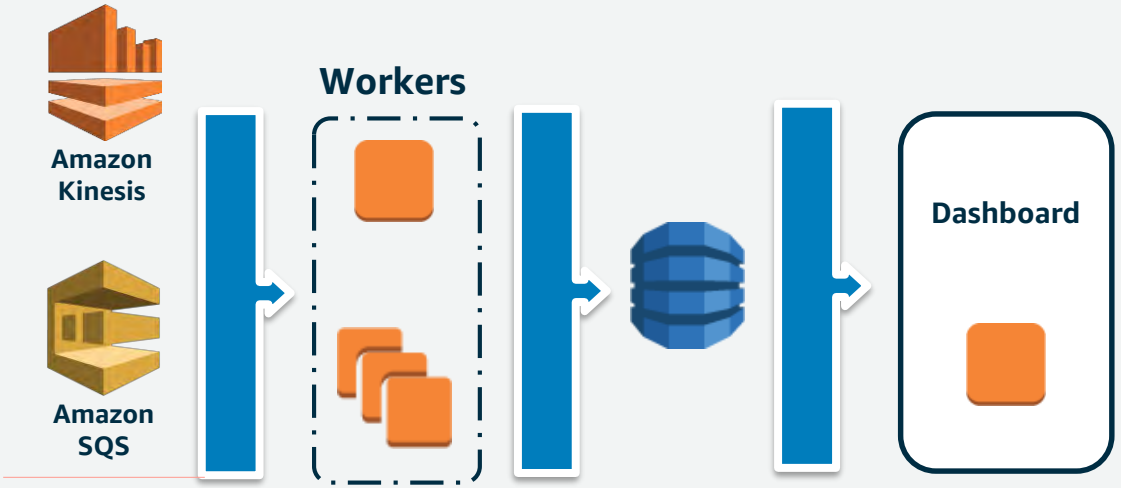
aws

特定のパーティションに対して集中した書き込み

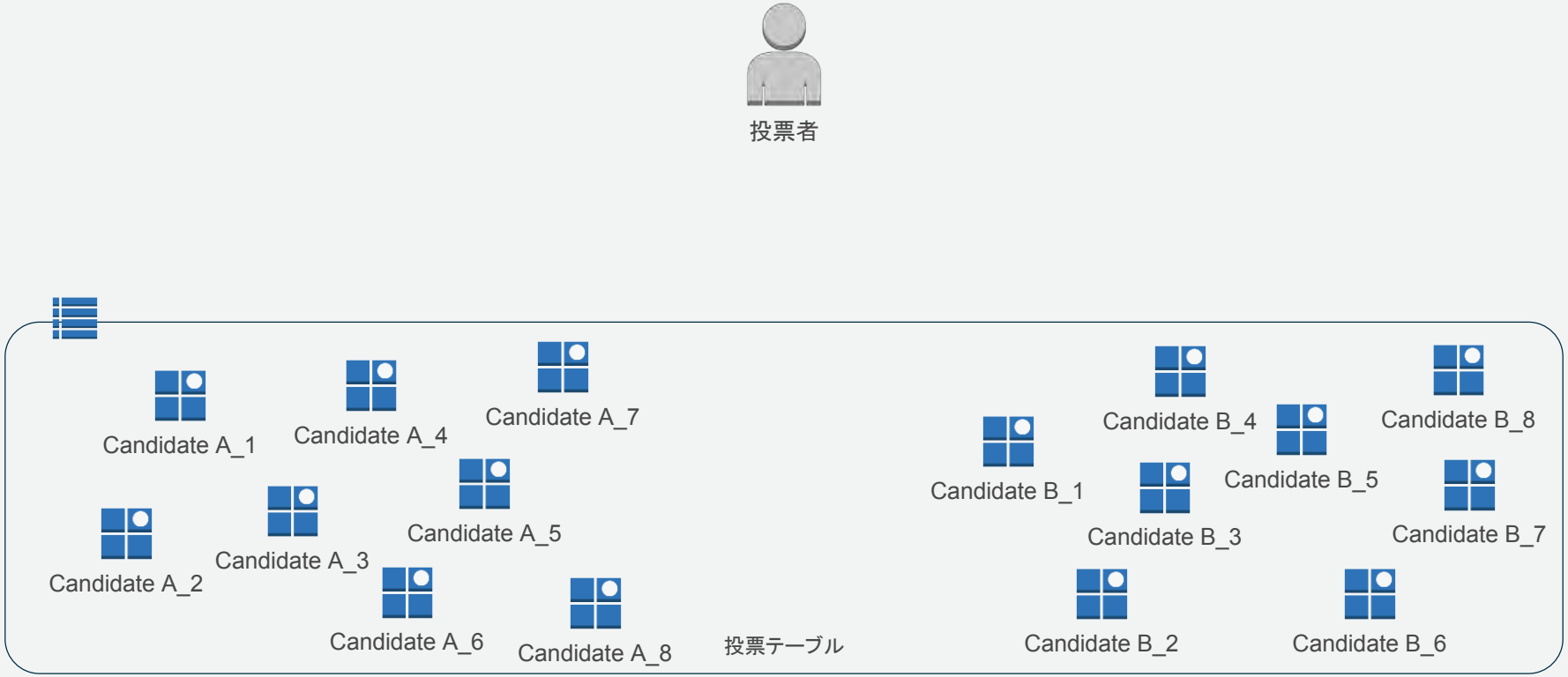


キュー、ストリームによる負荷調整

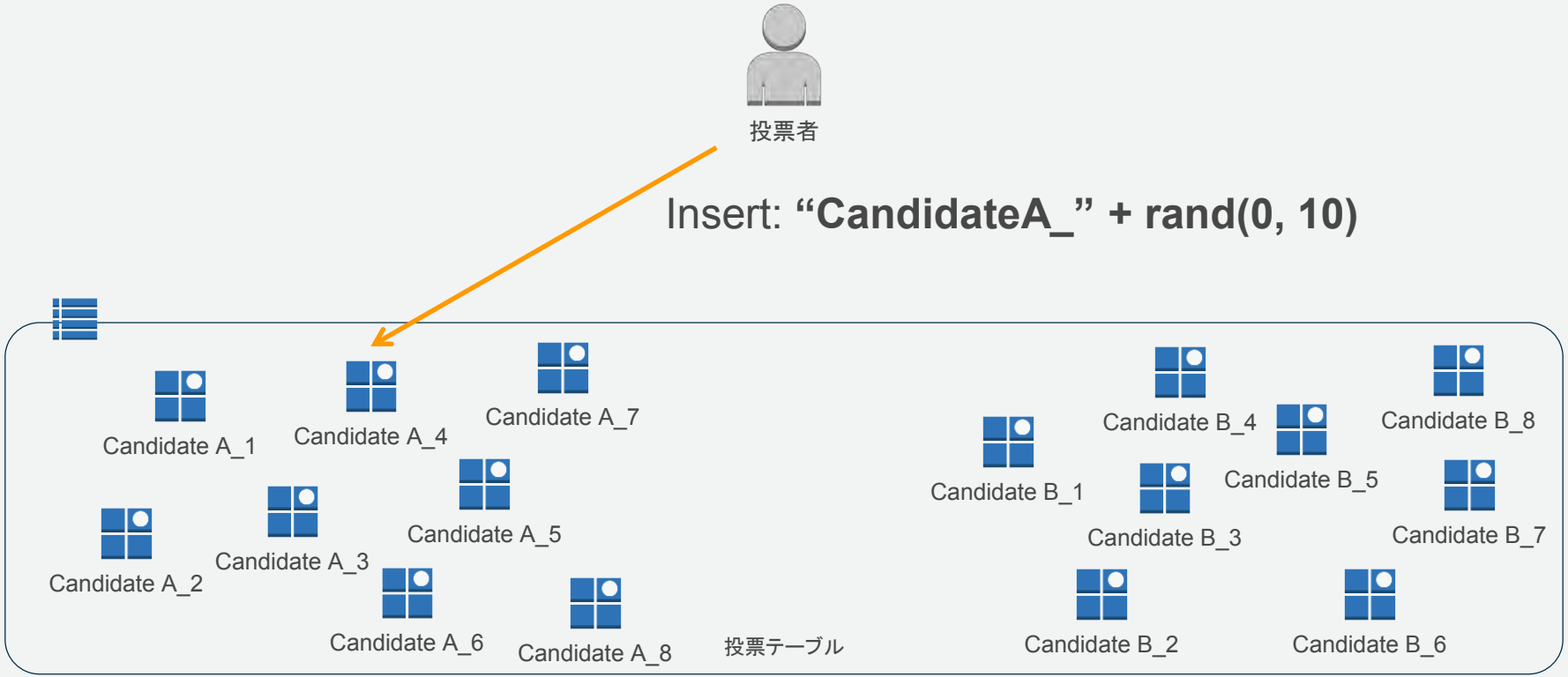
- 送信される投票内容を
一旦、SQSやkinesis で受ける
- SQSなどからメッセージを取得してDynamoDB へ書き込みを実施する Worker を必要に応じてスケールさせ、
キュー内のメッセージを処理
- 瞬間的な特定キーへの偏りを回避



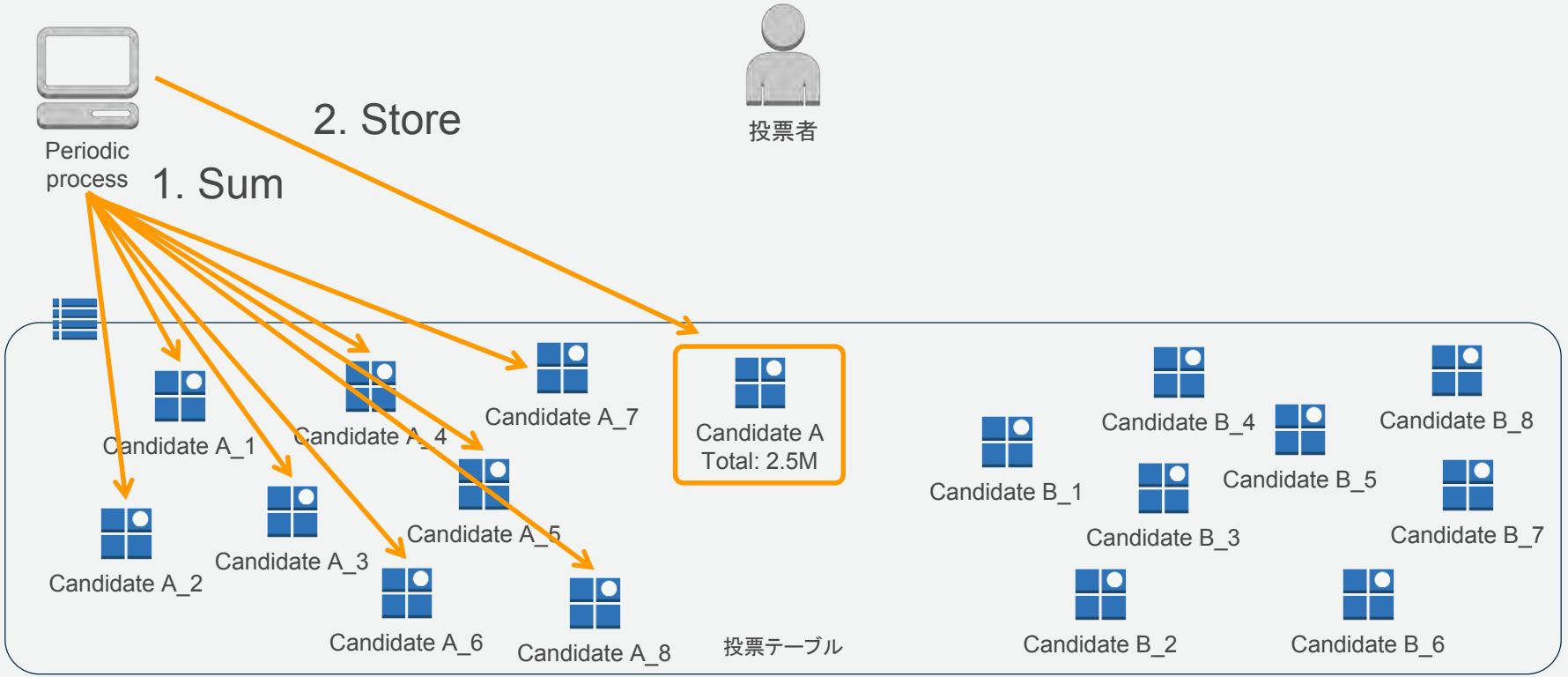
書き込みのシャーディング



書き込みのシャーディング

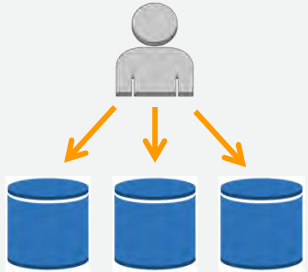


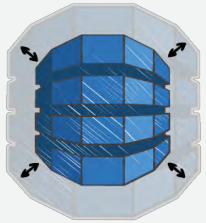
結果のマージ/集計



書き込みが偏るパーティションキーをシャーディング

- 同時アクセス数に応じてスループットを調整(増加)
- キー、Itemサイズ、リクエストレートごとのRCU/WCU を考慮





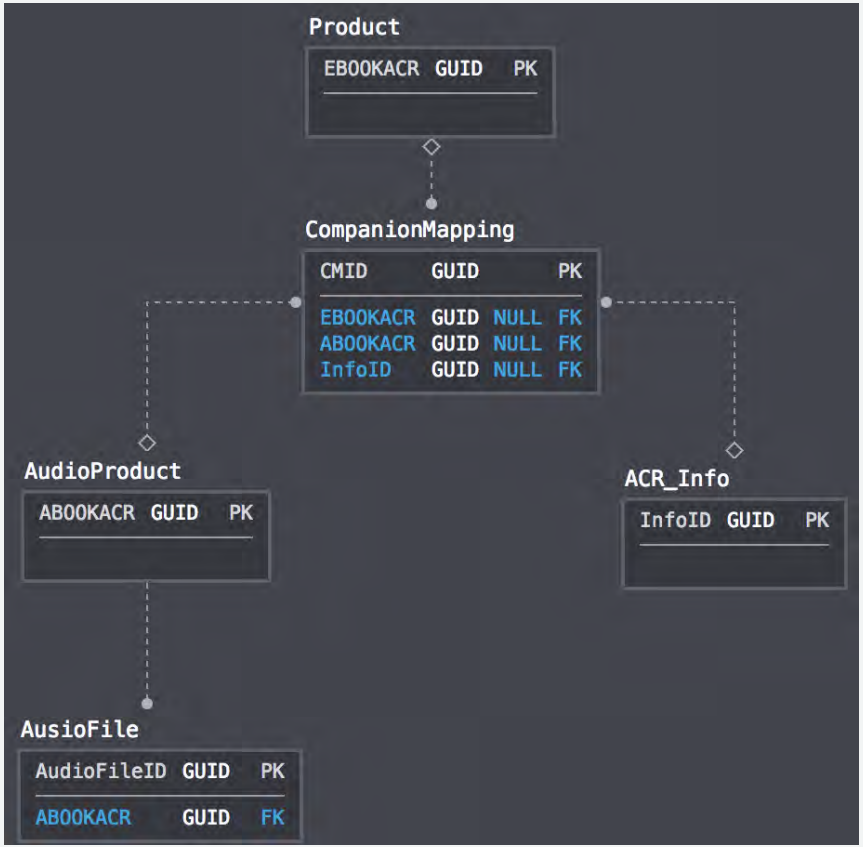
水平方向(均等)に書き込みワークロードがスケールできない場合に効果的

A Real World Example

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

- Philip K. Dick

Audible eBook Sync Service



- ユーザーがAudible eBooksのセッション情報を保存出来る
- eBook及びaudio productのユーザー毎にマッピング
- スパイクする負荷に対応するため多くのキャパシティが重要
- 多数のアクセスパターン対応

Access Patterns

#	USE CASE	
1	CompanionMapping	getCompanionMappingsByAsin
2	CompanionMapping	getCompanionMappingsByEbookAndAudiobookContentId
3	CompanionMapping	getCompanionMappingsFromCache
4	CompanionMapping	getCompanionMappings
5	CompanionMapping	getCompanionMappingsAvailable
6	AcrInfo	getACRInfo
7	AcrInfo	getACRs
8	AcrInfo	getACRInfos
9	AcrInfo	getACRInfosbySKU
10	AudioProduct	getAudioProductsForACRs
11	AudioProduct	getAudioProducts
12	AudioProduct	deleteAudioProductsMatchingSkuVersions
13	AudioProduct	getChildAudioProductsForSKU
14	Product	getProductInfoByAsins
15	Product	getParentChildDataByParentAsins
16	AudioFile	getAudioFilesForACR
17	AudioFile	getAudioFilesForChildACR
18	AudioFile	getAudioFilesByParentAsinVersionFormat
19	AudioFile	getAudioFiles
20	AudioFile	getAudioFilesForChildAsin

Primary Table

TABLE	Primary Key		Attributes	
	PK	**SK (GSI 3)**		
	ABOOKACR1	v0#ABOOKACR1	**GSI-1**	**GSI-2**
			ABOOK-ASIN1	ABOOK-SKU1
		EBOOKACR1	**GSI-1**	**GSI-2**
			SyncFileAcr	ABOOK-ASIN1
		ABOOKACR1#TRACK#1	**GSI-1**	**GSI-2**
			ABOOK-ASIN1	ABOOK-SKU1
		ABOOKACR1#TRACK#2	**GSI-1**	**GSI-2**
			ABOOK-ASIN1	ABOOK-SKU1
	EBOOKACR1	EBOOKACR1	**GSI-1**	**EBookAsin**
			EBOOK-SKU1	ASIN

※ASINは「Amazon Standard Identification Number」の略で、Amazonグループが取り扱う、書籍以外の商品を識別する10けたの番号

Indexes

GSI 1	Partition Key	Projected Attributes	
	ABOOK-ASIN1	ABOOKACR1	ABOOKACR1-v1
			ABOOKACR1#TRACK#1
			ABOOKACR1#TRACK#2
	SyncFileAcr	ABOOKACR1	MAP-EBOOKACR1
	EBOOK-SKU1	ABOOKACR1	EBOOKACR1

GSI 2	Partition Key	Projected Attributes	
	ABOOK-ASIN1	ABOOKACR1	MAP-EBOOKACR1
	ABOOK-SKU1	ABOOKACR1	ABOOKACR1-v1
			ABOOKACR1#TRACK#1
			ABOOKACR1#TRACK#2

GSI 3	GSI Partition Key	Projected Attributes	
	V0#ABOOKACR1	ABOOKACR1	ABOOKACR1-v1
	EBOOKACR1	ABOOKACR1	MAP-EBOOKACR1
	ABOOKACR1#TRACK#1	ABOOKACR1	ABOOKACR1#TRACK#1
	ABOOKACR1#TRACK#2	ABOOKACR1	ABOOKACR1#TRACK#2
	EBOOKACR1	ABOOKACR1	EBOOKACR1

Query Conditions

#	USE CASE		Lookup parameters	INDEX	Key Conditions	Filter Conditions
1	CompanionMapping	getCompanionMappingsByAsin	audiobookAsin/ebookSku	GSI2	GSI-2=ABOOK-ASIN1	None
2	CompanionMapping	getCompanionMappingsByEbookAndAudiobookContentId	ebookAcr/sku,version,format or audiobookAcr/asin,version,format	GSI-3 on TargetACR attribute OR PrimaryKey on Table	GSI-3=MAP-EBOOKACR1	version=v and format=f
3	CompanionMapping	getCompanionMappingsFromCache	ebookAcr/sku,version,format or audiobookAcr/asin,version,format	GSI-3 on TargetACR attribute OR PrimaryKey on Table	GSI-3=MAP-EBOOKACR1	version=v and format=f
4	CompanionMapping	getCompanionMappings	syncfileAcr, ebookAcr?, audiobookAcr?	GSI1	GSI-1=SyncFileAcr	None
5	CompanionMapping	getCompanionMappingsAvailable	ebookAcr, audiobookAcr	Primary Key on Table	Acr=ABOOKACR1 and TargetACR beginswith "MAP-"	
6	AcrInfo	getACRInfo	acr	Primary Key on Table	Acr=ABOOKACR1 and TargetACR beginswith "ABOOKACR1-v"	
7	AcrInfo	getACRs	acr / asin,version,format	Primary Key on Table	Acr=ABOOKACR1	version=v and format=f
8	AcrInfo	getACRInfos	acr	Primary Key on table	Acr=ABOOKACR1 and TargetACR beginswith "ABOOKACR1"	
9	AcrInfo	getACRInfosbySKU	sku	GSI2	GSI-2=ABOOK-SKU1	
10	AudioProduct	getAudioProductsForACRs	acr	Primary Key on table	Acr=ABOOKACR1 and TargetACR beginswith "ABOOKACR1"	
11	AudioProduct	getAudioProducts	sku, version, format	GSI2	GSI-2=ABOOK-SKU1	version=v and format=f
12	AudioProduct	deleteAudioProductsMatchingSkuVersions	sku, version	GSI2	GSI-2=ABOOK-SKU1	version=v
13	AudioProduct	getChildAudioProductsForSKU	sku	GSI2	GSI-2=ABOOK-SKU1	
14	Product	getProductInfoByAsins	asin	GSI1	GSI-1=ABOOK-ASIN1	
15	Product	getParentChildDataByParentAsins	asin	GSI1	GSI-1=ABOOK-ASIN1	
16	AudioFile	getAudioFilesForACR	acr	Primary Key on table	Acr=ABOOKACR1 and TargetACR beginswith "ABOOKACR1#"	
17	AudioFile	getAudioFilesForChildACR	acr, parent_asin	Primary Key on table	Acr=ABOOKACR1	version=v and format=f
18	AudioFile	getAudioFilesByParentAsinVersionFormat	parent_asin, version, format	GSI1	GSI-1=ABOOK-ASIN1	version=v and format=f
19	AudioFile	getAudioFiles	sku, version, format	GSI2	GSI-2=ABOOK-SKU1	version=v and format=f
20	AudioFile	getAudioFilesForChildAsin	asin, parent_asin, version, format	GSI1	GSI-1=ABOOK-ASIN1	version=v and format=f

まとめ

- NoSQLはRDBとは全く特性が異なるので、それらを理解した上で適所でうまく活用する！
- NoSQL, DynamoDB の特徴を理解した上でテーブルを設計する
  - 以下のドキュメントも参考に
    https://docs.aws.amazon.com/ja_jp/amazondynamodb/latest/developerguide/best-practices.html
  - https://www.youtube.com/watch?v=HaEPXoXVf2k
  - https://aws.amazon.com/jp/blogs/news/how-to-use-dynamodb-global-secondary-indexes-to-improve-query-performance-and-reduce-costs/
- 運用はゼロではないが、拡張性の面や性能面ではとても楽ができるのがNoSQLの中でのDynamoDBの特徴

- 構築・運用にかかる時間を、**本来のビジネス価値を高めることに投資できる！**

Q&A

お答えできなかったご質問については
AWS Japan Blog 「https://aws.amazon.com/jp/blogs/news/」にて
後日掲載します。

AWS Well-Architected 個別技術相談会

毎週”W-A個別技術相談会”を実施中

- AWSのソリューションアーキテクト(SA)に対策などを相談することも可能
- 申込みはイベント告知サイトから

(https://aws.amazon.com/jp/about-aws/events/)

AWS イベント で[検索]

ご視聴ありがとうございました

AWS 公式 Webinar
https://amzn.to/JPWebinar

過去資料
https://amzn.to/JPArchive