

今から理解する NoSQL Databases

～ケーススタディと実践的データモデリング～

上原 友理

技術統括本部 ソリューションアーキテクト
Amazon ワエブ サービス ジャパン合同会社

堤 勇人

技術統括本部 ソリューションアーキテクト
Amazon ワエブ サービス ジャパン合同会社

Who am I ?

上原 友理 (Yuri Uehara)



アマゾン ウェブ サービス ジャパン
ソリューション アーキテクト

AWSの利用を始めるお客様を技術面でサポートしています

好きなAWSサービス : Amazon Neptune

本セッションについて

お話すこと

本セッションではショッピングサイトを例に下記の2点をお伝えします

- 目的別にデータベースを使い分ける具体的なシチュエーション
- DynamoDBを例に、データモデリングの実践的なステップの解説

対象者

- NoSQL データベースの利用を検討されている開発者の方
- リレーショナルデータベースでサービスを構築した経験のある方

本セッションについて

本セッションでお話しないこと

- リレーショナルデータベース と NoSQL データベースの違い
- 各NoSQL データベースサービスの説明
- マネージドサービス 等のAWSの基本的な説明

※上記内容については、以前実施したオンラインセミナーをご観聴ください

- 開発者のための、NoSQL Database の選び方 入門

https://pages.awscloud.com/DevAx_connect_session_archive.html

アジェンダ

- NoSQL データベースが求められる背景
- こんな時に NoSQL データベースを使おう
- 実践的 NoSQL データモデリング
- データモダナイゼーション with AWS

NoSQL データベースが求められる背景

最新のアプリケーション要件

より多くのパフォーマンス、拡張性、可用性が必要

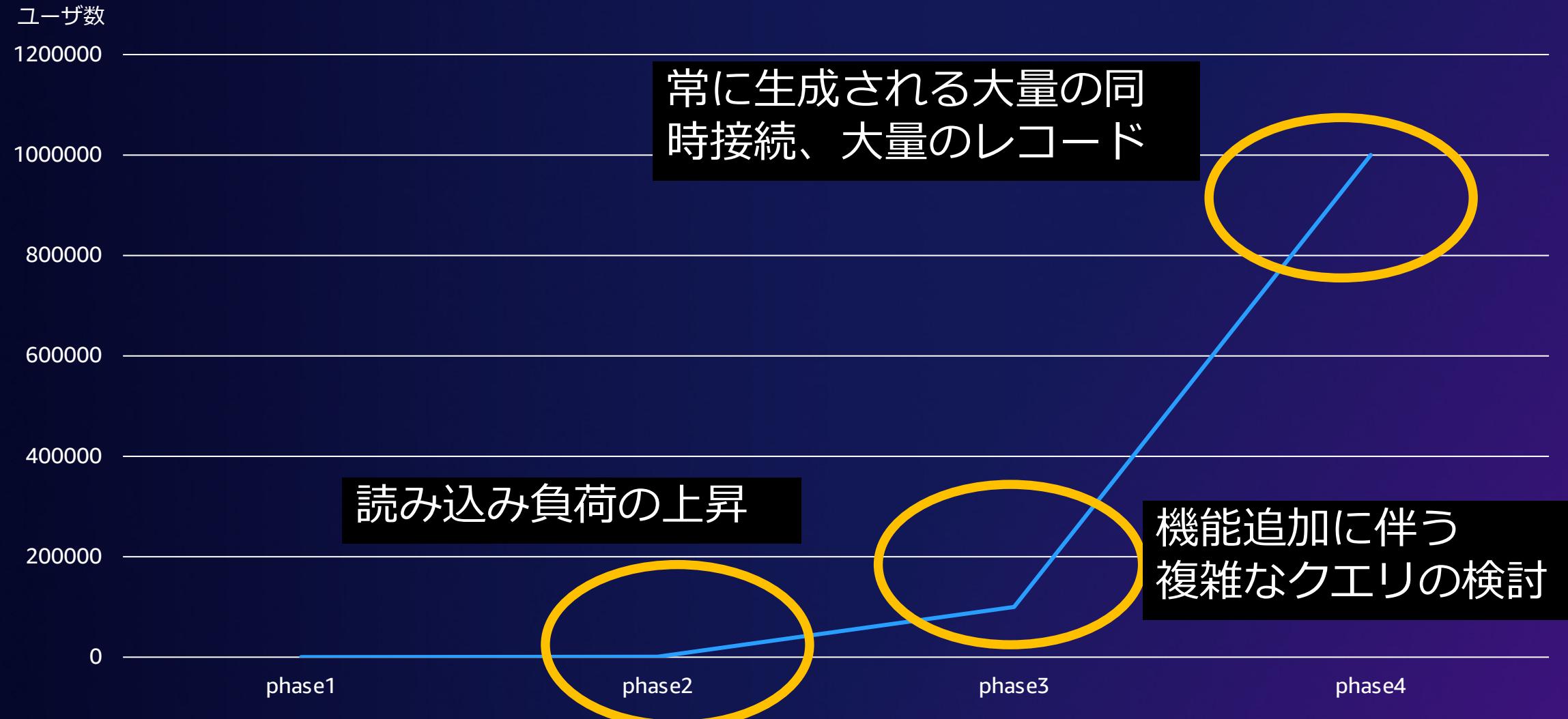


電子商取引 メディア ソーシャル オンライン 共有経済
ストリーミング メディア ゲーム



アクセスユーザー	100万+
データボリューム	テラバイト、ペタバイト
アクセス元	世界中から
パフォーマンス	ミリ～マイクロ秒遅延
リクエストレート	毎秒百万
交通アクセス	ウェブ、モバイル、IoT、デバイス
スケール	スケールアップ/ダウン、スケールアウト/イン
事業マネタイズ	従量制課金制
開発者アクセス	API アクセス
アプリの変更頻度	毎週やそれ以下の場合も

システムの成長と共に様々な課題が発生



データベースに求めるものは 利用するアプリケーションによって異なる



適切なデータベースを選択することで
あなたのビジネス要件を満たす
アプリケーションを構築できます



Scale
faster

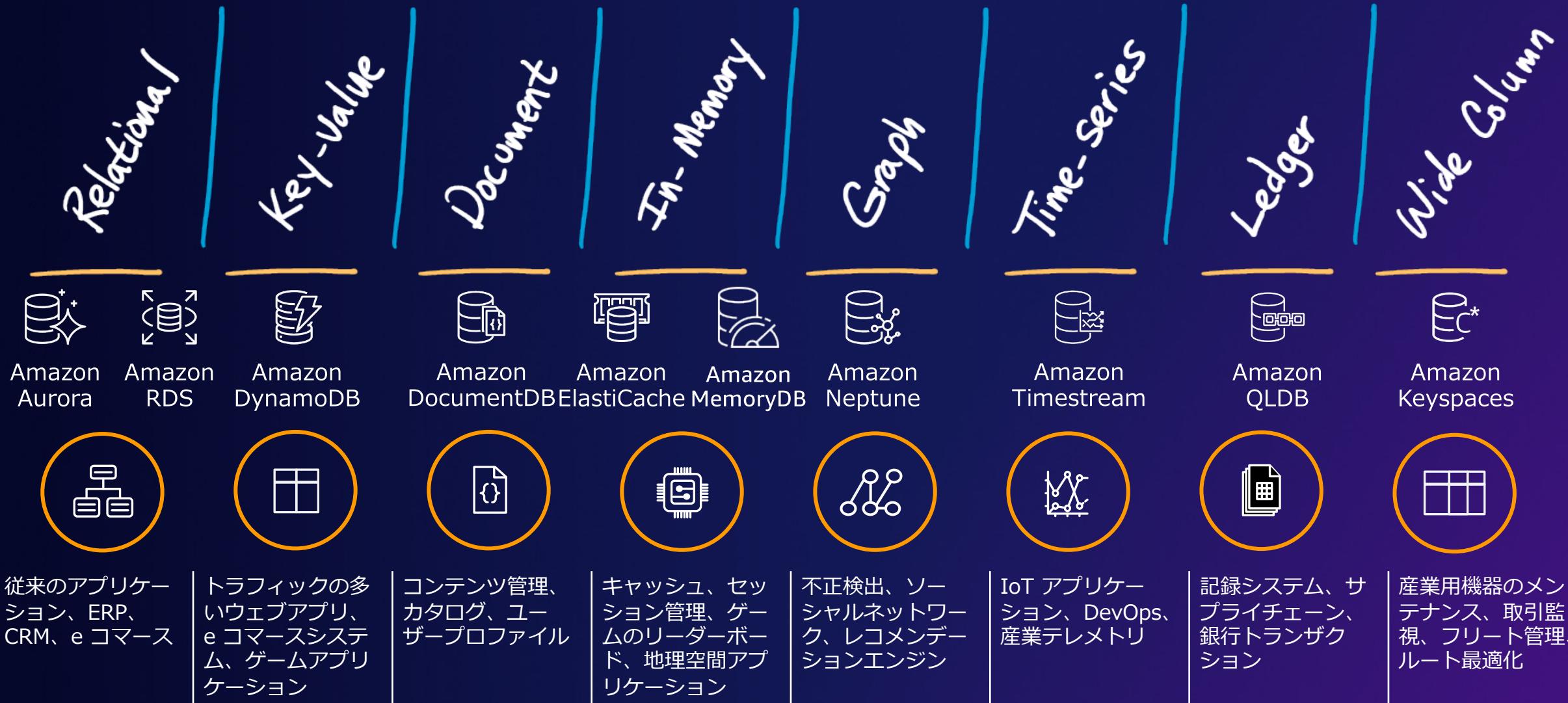


Innovate
more



Accelerate
time to
market

要件に応える幅広いデータベースサービス



世の中の変化に合わせて
NoSQLが必要になったのは分かった
でもどういう時に使うのか
イメージがつかない



【ケーススタディ】 こんな時に NoSQL データベースを使おう

設定



あなたは新しくリリースされた
あるショッピングサイトの開発者です

担当している新サービスが軌道に乗ったある日・・



← 最近ショッピングサイトの
開発を終えたあなた

サイトが表示できないって
問合せが来てるよ
すぐに対応して！

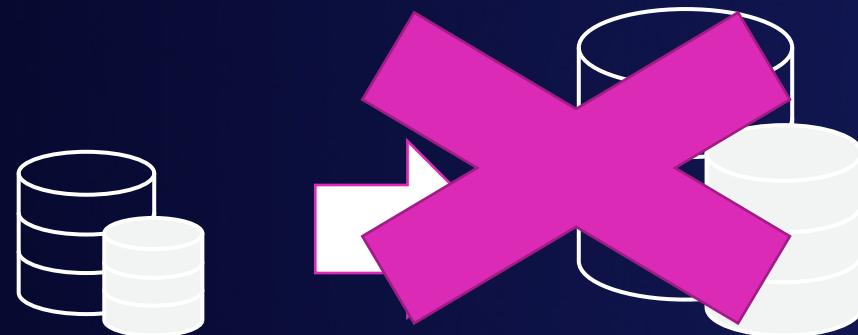


システムの成長と共に様々な課題が発生



データベースの拡張を申請するも・・

Relational
databases



すぐに原因を調査し、読み込み負荷が増えレ
イテンシが上昇していることを発見

よし、データベースを拡張すれば解決だ！

リソースは余裕があるように見
えるけど本当に拡張したら改善
するの？

他にチューニングの余地は？



リーダー

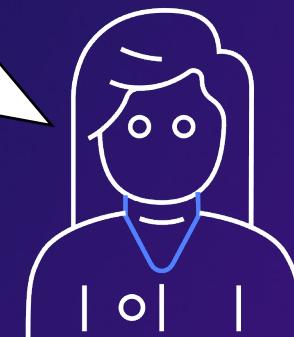
キャッシュを活用して低レイテンシを実現



単純なクエリだからチューニングの余地なんて・・
どうやってレイテンシを下げればいいんだ?

更新の少ないデータならキャッシュさせればいいんじゃない?

インメモリデータベースを調べてごらん

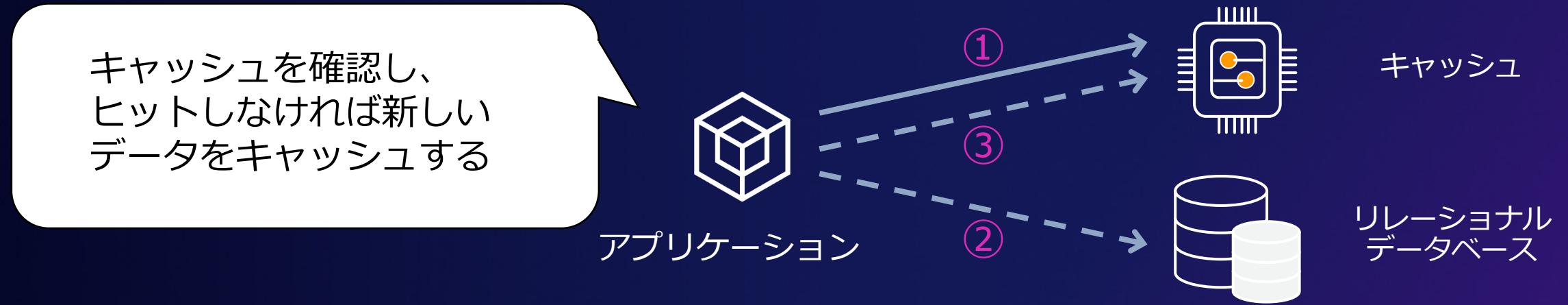


先輩

要件に応える幅広いデータベースサービス



キャッシュを活用して低レイテンシを実現



【イメージ】

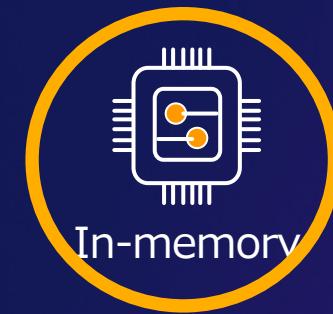
```
get_customer(product_id)
① product_record = cache.get(product_id)
if (product_record == null)
    ② product_record = db.query("SELECT * FROM Products WHERE id = {0}", product_id)
    ③ cache.set(product_id, product_record)
return product_record
```

Before & After

Relational
databases



Purpose-Built Databases



アクセスが多く更新頻度の少ないデータは
キャッシュさせることに
そこでキャッシュに最適な インメモリデータ
ベースである Amazon ElastiCache を追加

ユーザ数が伸び悩んでいるある日・・



← システムも安定稼働して
平和だなと思っていたあなた

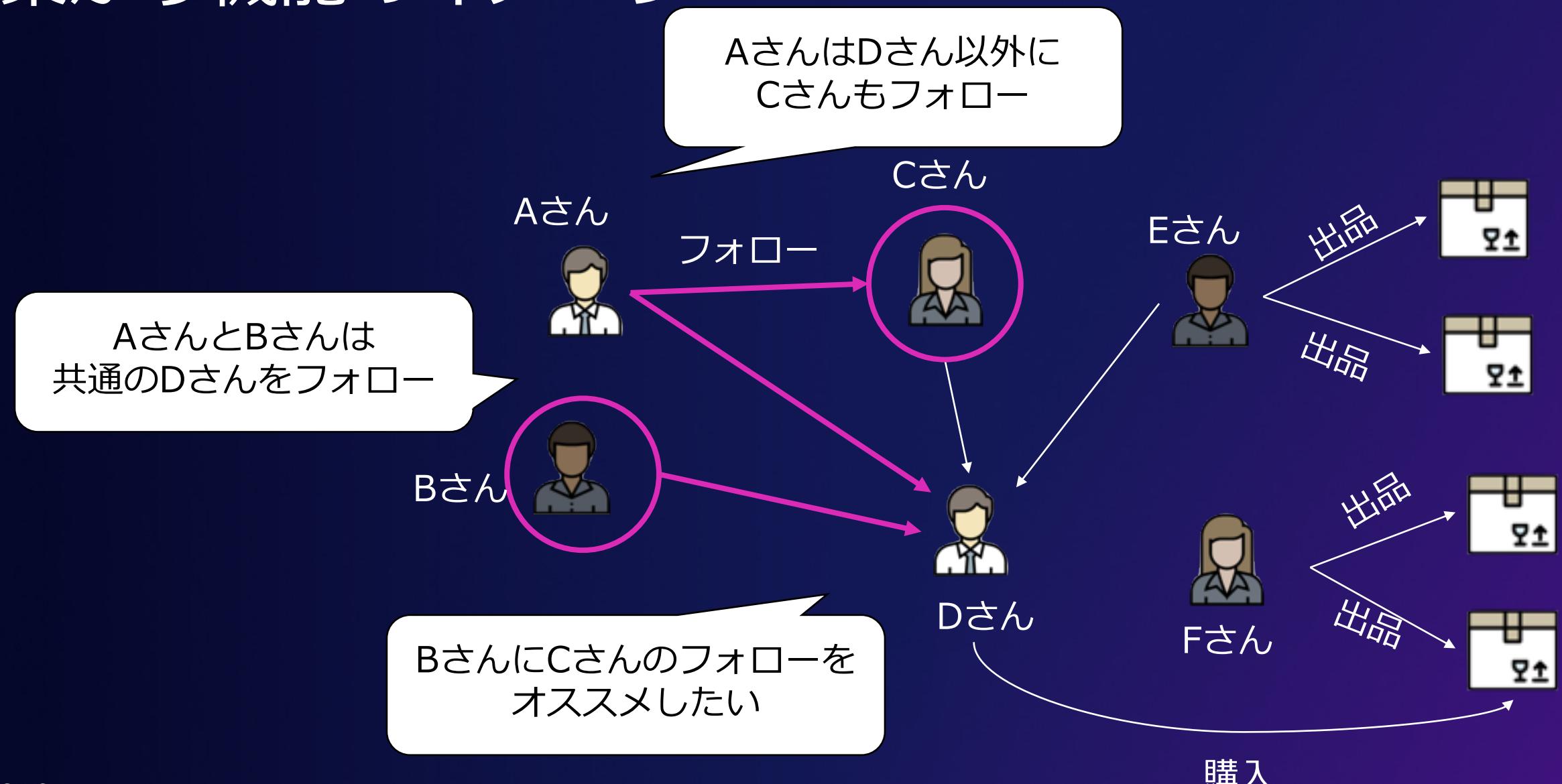
新規ユーザを増やす為に、
ユーザ同士の繋がり機能を追加して！



システムの成長と共に様々な課題が発生



繋がり機能のイメージ



関係表を追加することを検討するも・・



ユーザテーブルは既にあるから、
フォロー関係を表すテーブルを追加しよう

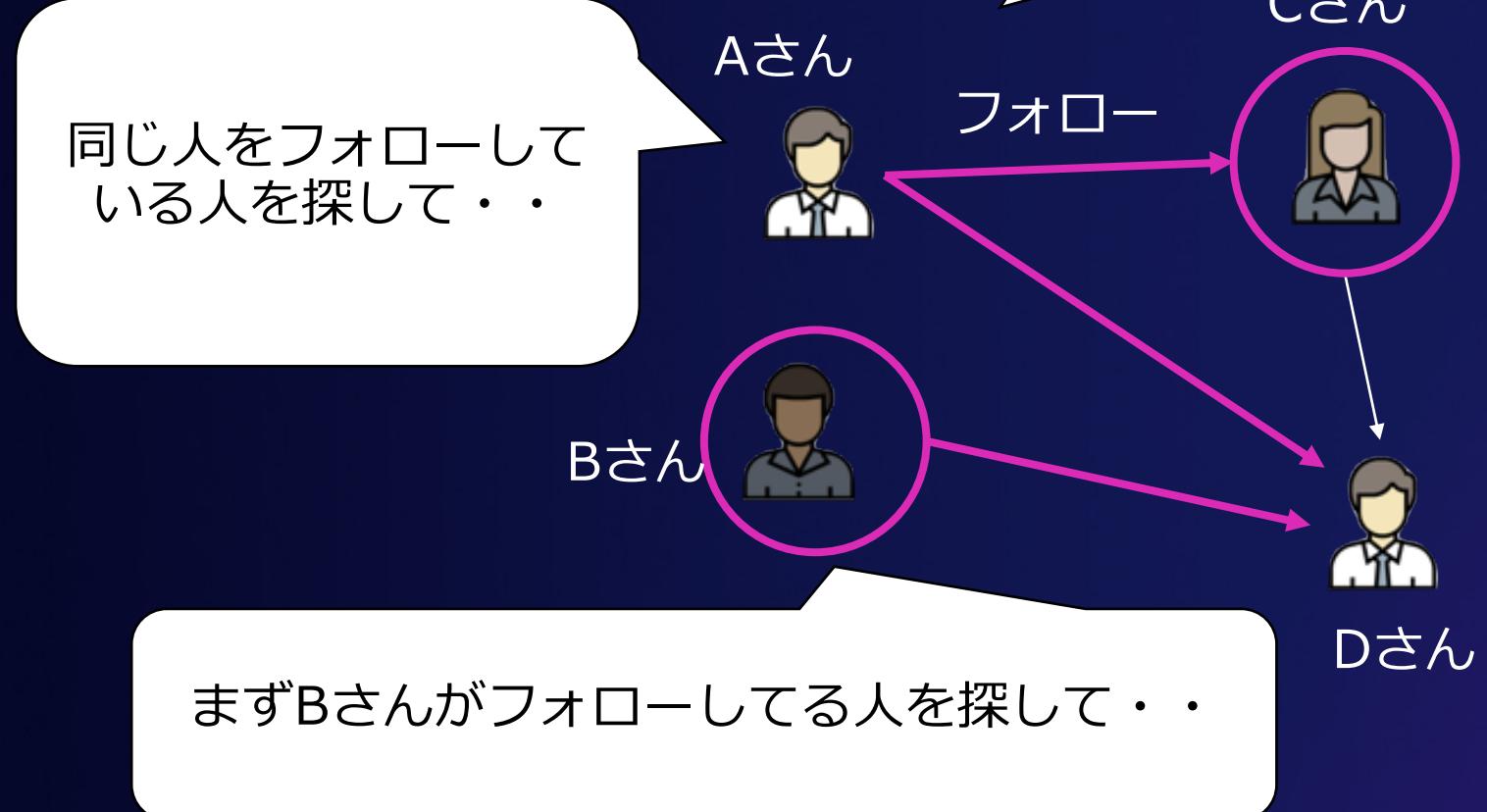
User

- id
- name

Follow

- user_id
- follow_id

関係表を追加することを検討するも・・



```
SELECT follow_id
FROM Follow
WHERE user_id = B
```

```
SELECT user_id
FROM Follow
WHERE follow_id = D
```

```
SELECT follow_id
FROM Follow
WHERE user_id = A
```

```
SELECT name
FROM User
WHERE u.id = C
```

グラフ構造でシンプルにつながりを表現

ユーザ数が増えるとフォロー関係の
階層も増えていくよ

もっとシンプルにクエリできるようにグラフ
データベースを使ったらどうだろう



・・・すぐにインターネット検索・・・

こんなシンプルなクエリで実現できるんですね

```
g.V('B').out('follow').in('follow').out('follow') .values('name')
```



要件に応える幅広いデータベースサービス

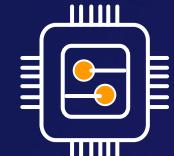


Before & After

Relational
databases



Purpose-Built Databases



In-memory



Graph

多対多の関係をシンプルにクエリし、今後のスケールに対応できる方法を検討

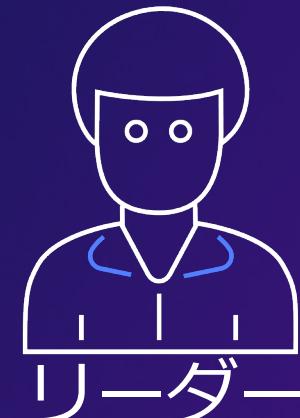
そこでグラフデータベースである Amazon Neptune を利用

順調にユーザが増加したある日・・

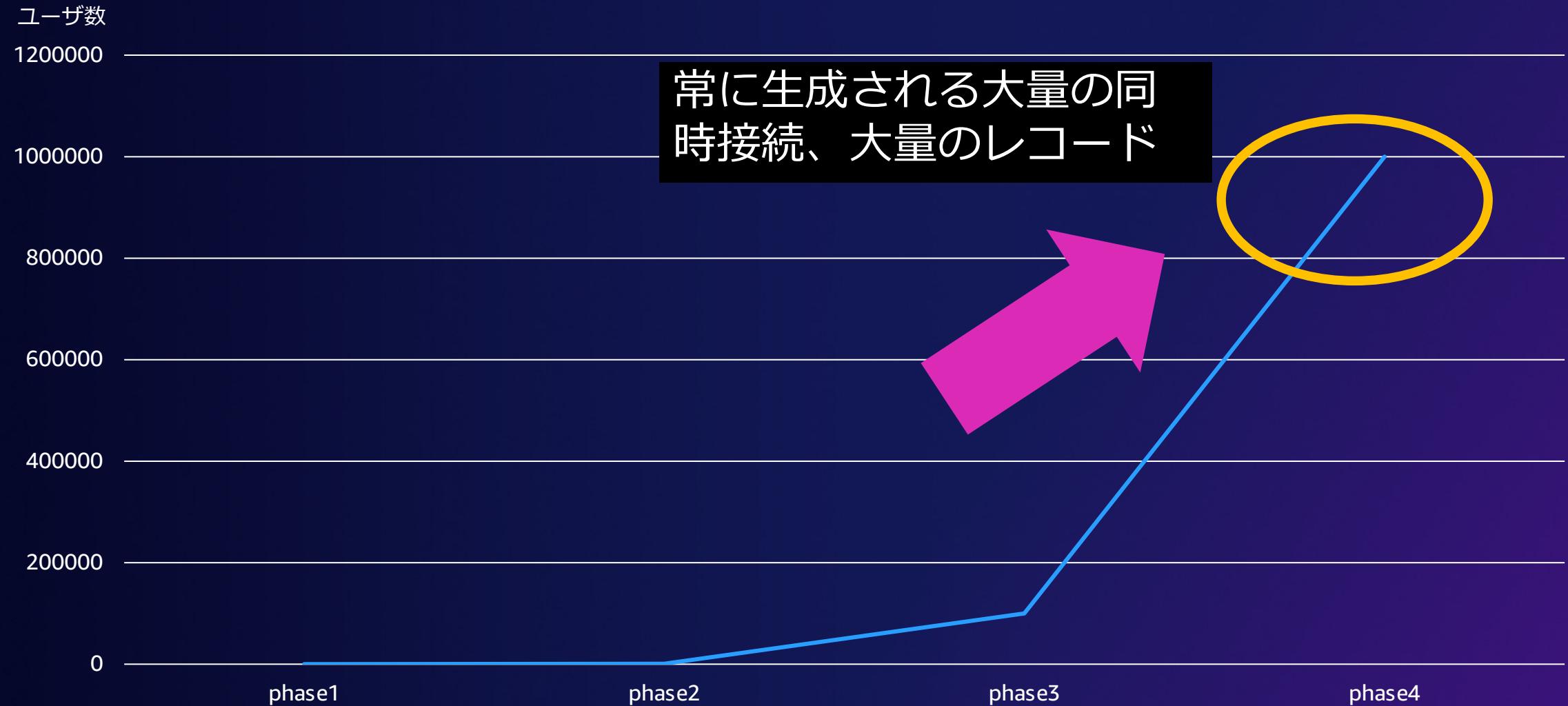


← 大規模サービスになり
感慨にふけるあなた

エラーが頻発してるって
SNSで騒がれてるよ
すぐに対応して！



システムの成長と共に様々な課題が発生

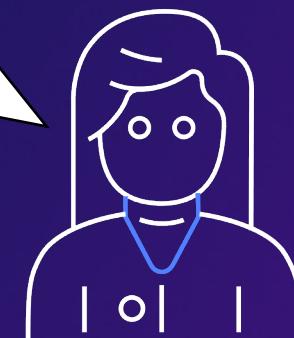


スケーラビリティを求めて

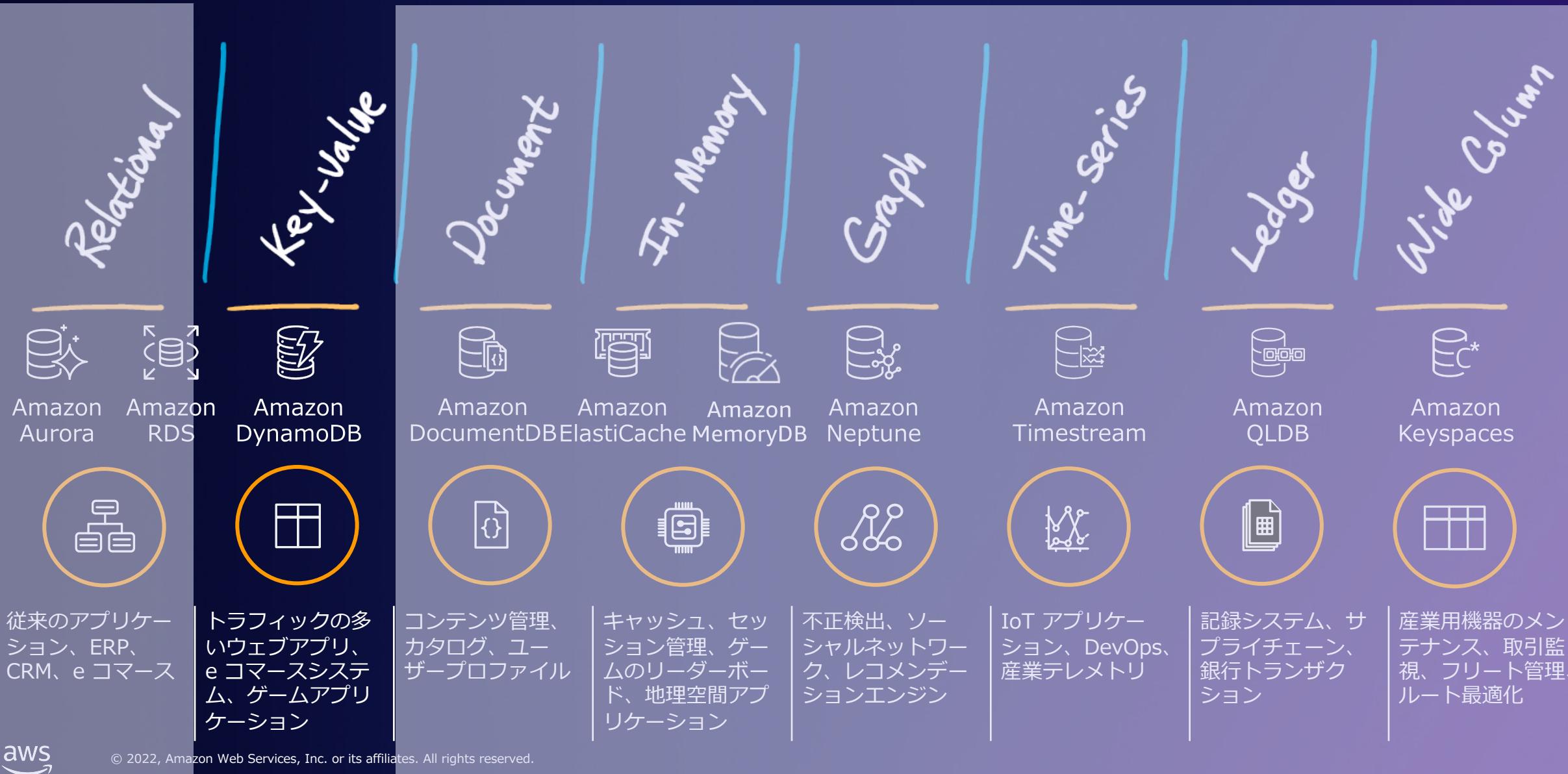


これまでデータベースの分散にも取り組んで来たけど、
ユーザ数の増加と共にレイテンシが増加している
これ以上分散しても改善は難しい・・

シンプルなデータ構造ならもっと高速に処理
できるデータベースもある
仕組みを大きく変える必要があるけれど、今
後の成長のために最適な方法を検討しよう



要件に応える幅広いデータベースサービス



アクセスパターンを整理する

詳細は
後半で!



キーバリューデータベースで低レイテンシのままスケールさせることができそうだ
でもトランザクション管理やキー制約はそのまま使える訳ではないんだな・・

アプリケーションの改修で吸収できるか
ユーザ視点で重要かも考えながら
アクセスパターンを整理しましょう



先輩

Before & After

Relational
databases



Purpose-Built Databases



Key	Value
Key	Value

Key	Value
Key	Value



In-memory



Graph

アクセスパターンを分析し、厳密なトランザクション管理や複雑な制約の要否を整理

移行が可能な機能はキーバリュー型の Amazon DynamoDB に置き換えて低レイテンシを実現

ユーザが増えて困ったら
NoSQL を使うの?



今までに あなたがシステム開発をしているとしたら・・

- 最初から全てNoSQLデータベースを使うべきか？
- まずは使い慣れた リレーショナルデータベース？

どちらも正解です！

要件を明確にしておくことが大切

たとえば・・

低レイテンシ



ゲーム

早期リリース



新規サービス

厳密な
トランザクション



金融サービス

複数のデータベースを適材適所で組み合せる
要件の変更に応じた移行も視野に

データベースの選択方法が分かって来た
でも実際に使うとなるとどう扱うのか
データモデリングが分からない



実践的NoSQLデータモデリング

Who am I ?

堤 勇人 (Hayato Tsutsumi)

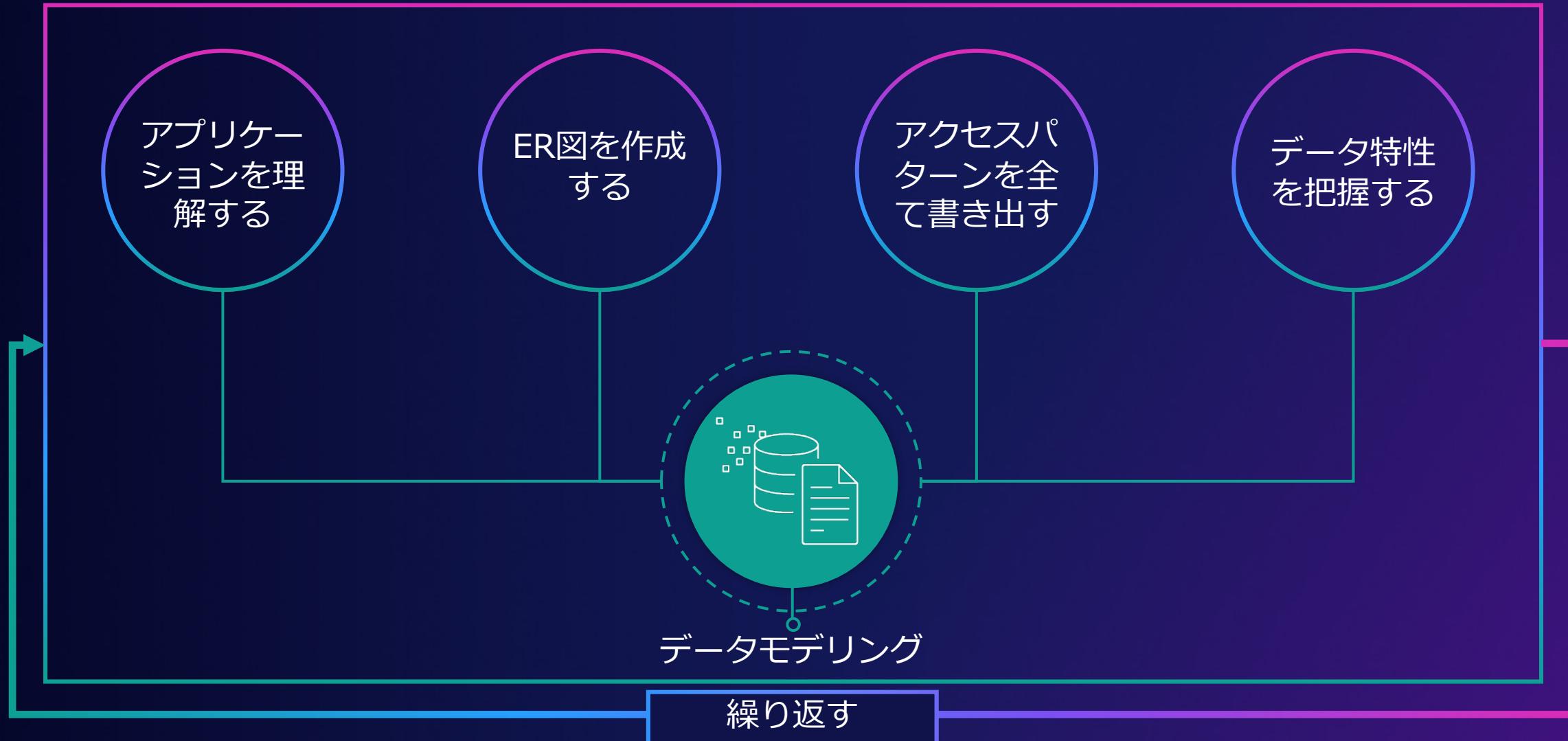
アマゾン ウェブ サービス ジャパン
ソリューション アーキテクト

お客様のビジネスを加速させる技術的なサポート

好きなAWSサービス : Amazon DynamoDB, Amazon Keyspaces



データモデリングのステップ



アプリケーションを理解する



ユーザーの体験を理解する



担当者・開発者とミーティングを持つ

ER図を作成する



アクセスパターンを全て書き出す

No.	Access Pattern Name	Input Parameters	Return	Order
1	PutToCart	UserId, ProductId, Count, Price, TimeStamp	isSucceed	
2	PurchaseCartItems	UserId	OrderId	
3	ListActiveProducts	UserId	List[ProductId, Count, Price, CreatedAt]	CreatedAt
4	ListCartItems	UserId	List[ProductId, Count, Price, CreatedAt]	CreatedAt
5	ListPurchasedItems	UserId	List[ProductId, Count, Price, CreatedAt]	CreatedAt
6	ListUsersByCart	ProductId	List[UserId]	CreatedAt
7	ListUsersByPurchased	ProductId	List[UserId]	CreatedAt
8	FullTextSearchProducts	UserId, Keywords	List[ProductId, Count, Price, CreatedAt]	CreatedAt
9	AnalyzeDataForBI	various	various	various

データの特性を把握する



- カーディナリティ



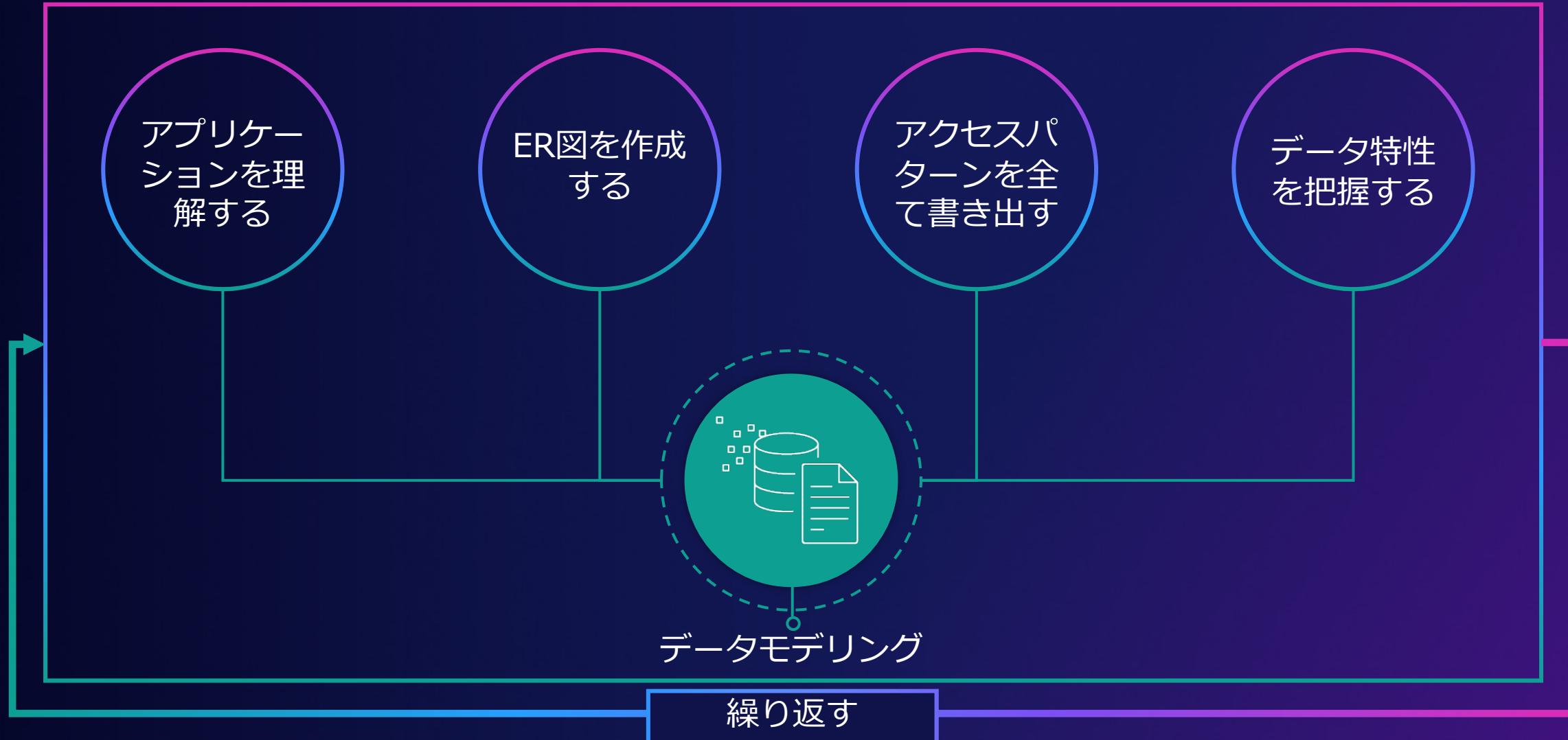
- サイズ



- リクエストレート、
許容レイテンシ

Name	Ammount
Number of Users	1M -
Cardinality of Users	High
Number of Products	1K -
Cardinality of Products	High
Number of Products in Cart	1 - 15
Latency of top page	100 - 500ms
Latency of Cart operation	About 1 sec
Read requests per sec	1,000 rps
Write requests per sec	10 rps

データモデリングのステップ(再掲)



アクセスパターンをそのままテーブルにする

No.	Access Pattern Name	Input Parameters	Return	Order
1	PutToCart	UserId, ProductId, Count, Price, TimeStamp	isSucceed	
2	PurchaseCartItems	UserId	OrderId	
3	ListActiveProducts	UserId	List[ProductId, Count, Price, CreatedAt]	CreatedAt
4	ListCartItems	UserId	List[ProductId, Count, Price, CreatedAt]	CreatedAt
5	ListPurchasedItems	UserId	List[ProductId, Count, Price, CreatedAt]	CreatedAt
6	ListUsersByCart	ProductId	List[UserId, ProductId, Count, Price, CreatedAt]	createdAt
7	ListUsersByPurchased	ProductId	List[UserId, ProductId, Count, Price, CreatedAt]	createdAt
8	FullTextSearchProduct	UserId, Keywords	List[ProductId, Count, Price, CreatedAt]	createdAt
9	AnalyzeDataForBI	various	various	various

書き下したテーブル

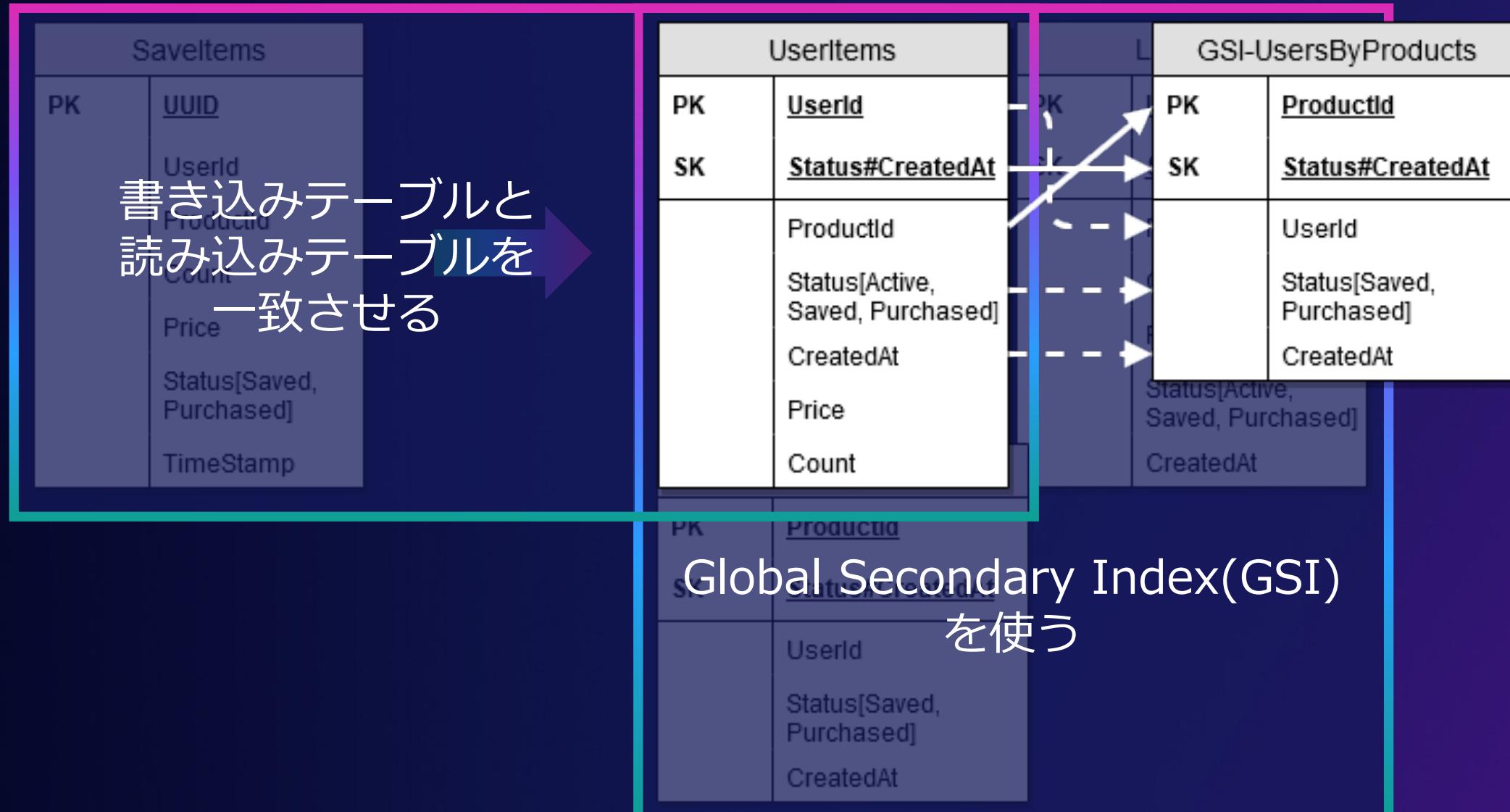
PutToCart		ListActiveProducts		ListCartItems		ListPurchasedItems	
PK	<u>UUID</u>	PK	<u>UserId</u>	PK	<u>UserId</u>	PK	<u>UserId</u>
	UserId		Order		Order		Order
	ProductId				ProductId		ProductId
	Count				Count		Count
	Price				Price		Price
	TimeStamp				CreatedAt		CreatedAt

PurchaseCartItems		ListUsersByCart		ListUsersByPurchased		FullTextSearchProducts		AnalyzeDataForBI	
PK	<u>UUID</u>	PK	<u>ProductId</u>	PK	<u>ProductId</u>	PK	<u>UserId,</u> <u>Keywords</u>	PK	<u>???</u>
	UserId		Order		Order		Order		???
	ProductId								???
	Count								???
	Price								???
	TimeStamp				UserId				???

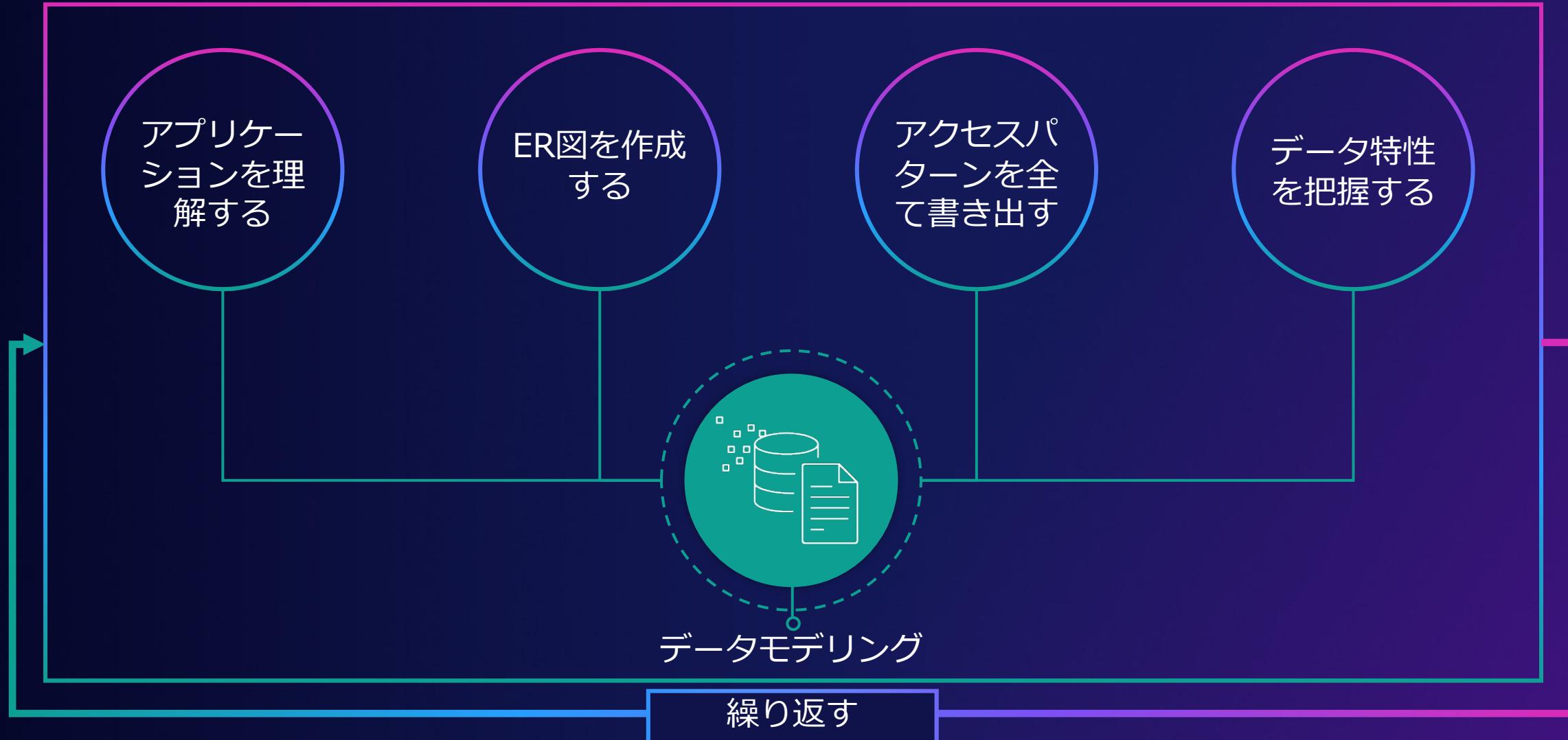
テーブルを減らしていく

SaveItems		ListActiveProducts		ListCartItems		ListItems	
PK	UUID	PK	UserId	PK	UserId	PK	UserId
	UserId		Order		Order		
	ProductId		CreatedAt		CreatedAt		
	Count		ProductId		ProductId		
	Price		Count		Count		
	Status[Saved, Purchased]		Price		Price		
	TimeStamp		CreatedAt		CreatedAt		
マージする		マージする		マージする		DynamoDBには向いていない OLAPやアドホックなクエリは 別のデータベースへ	
PK	UUID	PK	ProductId	PK	ProductId	PK	???
	UserId		Order		Status#CreatedAt		
	ProductId		CreatedAt		UserId		
	Count		UserId		Status[Saved, Purchased]		
	Price				CreatedAt		
	TimeStamp						
マージする		マージする		マージする		マージする	
PK	UserId, Keywords	PK	???	PK	???	PK	???
	Order		CreatedAt		ProductId		
	ProductId		UserId		Count		
	Count				Price		
	Price				CreatedAt		
	TimeStamp						
マージする		マージする		マージする		マージする	
avlis		© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.		50		50	

テーブルを減らしていく



データモデリングのステップ(再掲)



データモデリングのループ



今回触れていない課題

データ移行

バージョン
アップ戦略

キャパシ
ティ予測

監視メトリ
クス

データモダナイゼーション with AWS



必要なインサイトを安全かつ大規模にもたらす データ戦略

マネージド
サービス
への移行

レガシーデー
タベースから
の脱却

統合された
データ
エコシステム
の構築

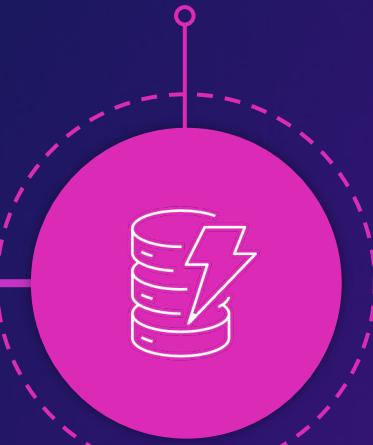
Step 1
データインフラの
モダナイゼーション

データモダナイゼーションのあらゆるステージで お客様を支援

データインフラをクラウド化し、コ
ストを削減する速度と柔軟さを獲得



purpose-build database により
様々なアプリケーションを構築



マネージドデータベースの利用に
より、運用の重労働を自動化

拡張性、信頼性、安全性に優れたクラウドプロバイダーでデータ基盤をモダナイズしまじょう



特性に合った正しい
データベースの利用
15以上 purpose-built cloud
databases



時間とコストの節約
商用データベースの10分の1のコスト



ユースケースに沿ったスケ
ーラブルで分散されたアプ
リケーション



AWS はクラウドデータベ
ースに対して常にオープン
なマインドを持っています

Thank you!



© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.