データモデリング

パフォーマンスのための設計

河野　泰幸 | ソリューション・エンジニア

2021年 5月 12日

アジェンダ

1. 課題
2. ツール
3. JSON ドキュメントと K-V アクセス
4. エンティティ・リレーションと非正規化
5. 配列と配列インデックス
6. 注意事項

1 課題

Sponsored By

Special Relativity

Limiting the speed of electromagnetic waves through wires in a universe near you

データモデリングの技術

データ モデリングは、選択肢が豊富なタスクです。

その意味で、最適化する対象・目的を把握する事が重要です。

ある目標に対する最適化は、別の目標の最適化の断念を必要とする場合があります。

最適化の選択肢	優先 順位
アプリケーションのパフォーマンス	1
将来のスケーラビリティ	1
リソース要件の最小化	1
複数のアプリケーションまたはユースケースをサポート	1
今後のモデル拡張	1

ここでは、あなたが調整できるいくつかの「ダイヤル」を紹介します。
ある選択肢は他の選択肢に影響を与えます。
単一の「正しい」選択はありませんが、間違った選択はあります

警告：上記のように、この先の議論には、ある程度の確率で「矛盾」が含まれ得ます。

設計上の課題

SLAs

- エンド ユーザーによる評価
- データベースのみでなく外部システムに依存
- 複数のチームによる作業の調整
- 複雑なトラブルシューティング
- 野心的 - 現実に基づいていないかもしれない

コスト

- ハードウェア/ホスティングコスト
- ソフトウェア ライセンス
- サポートコスト
- インシデント・コスト
- SLAからの逸脱
- ブランド・ダメージ

ネットワーク帯域幅

- サーバー ノード間でのデータ移動
- サーバーからクライアントへのデータ移動 (逆も同様)
- ラックアーキテクチャ (ToR, EoR, コアスイッチ)
- データ出力コスト
- CPU on サーバー vs クライアント

キャッシュ無効化

- 複数の製品の統合
- リード・スルー
- ライト・スルー
- バルク操作
- データ同期

Couchbaseの「メモリファースト」アーキテクチャは、この課題を取り除きます

分析のためのETL

- アドホッククエリ
- データマッピング
- バッチ障害
- 同期
- ネットワーク帯域幅

Couchbaseの「NoETL」アーキテクチャは、この課題を取り除きます

2 | ツール

Couchbase パフォーマンスの課題に対処するためのツール

スケーラビリティ

- クラスターへのノードの追加
- より大きなノードを使用
- 複数データ・センター間 XDCR

キー-バリュー アクセス

- 超高速
- 余分なネットワーク ホップなし
- サブドキュメントおよびアレイ操作 API
- ACID トランザクション

N1QL インデックス

- クエリ・チューニング
- レプリカとパーティション
- インデックス・アドバイザー

全文検索

- 反転インデックス
- ランキングとスコアリング
- ステミングとフィルタリング

選択的 非正規化

- 不要な結合を避ける
- 値の事前計算

利用可能なAPI について理解する

Couchbase API	レイテンシ	スループット	スケーラビリティ	適用性
キーバリュー	50µs – 10 ms	> 10M ops/sec	非常に高	高スループット、低遅延が重要なワークロードに最適
N1QL	< 1ms - 2ms+	> 40k クエリ/sec	高	セカンダリルックアップに最適、ロジックをデータベースにプッシュ
Full Text Search	5ms+	> 20k クエリ/sec	高	自然言語テキスト検索、関連性ランキング、ファセット
Analytics	1s – 60s+	< 1-10 クエリ/sec	高	非常に大きなデータ量の複雑な分析またはアドホック分析

そもそも、NoSQL アーキテクチャが考案されたのには、理由があります。

**レガシーデータベースよりも多少の複雑さが予期されますが、
それだけの価値があります。**

SQL

Not Only SQL

通話を開始する方法はいくつあるでしょうか?

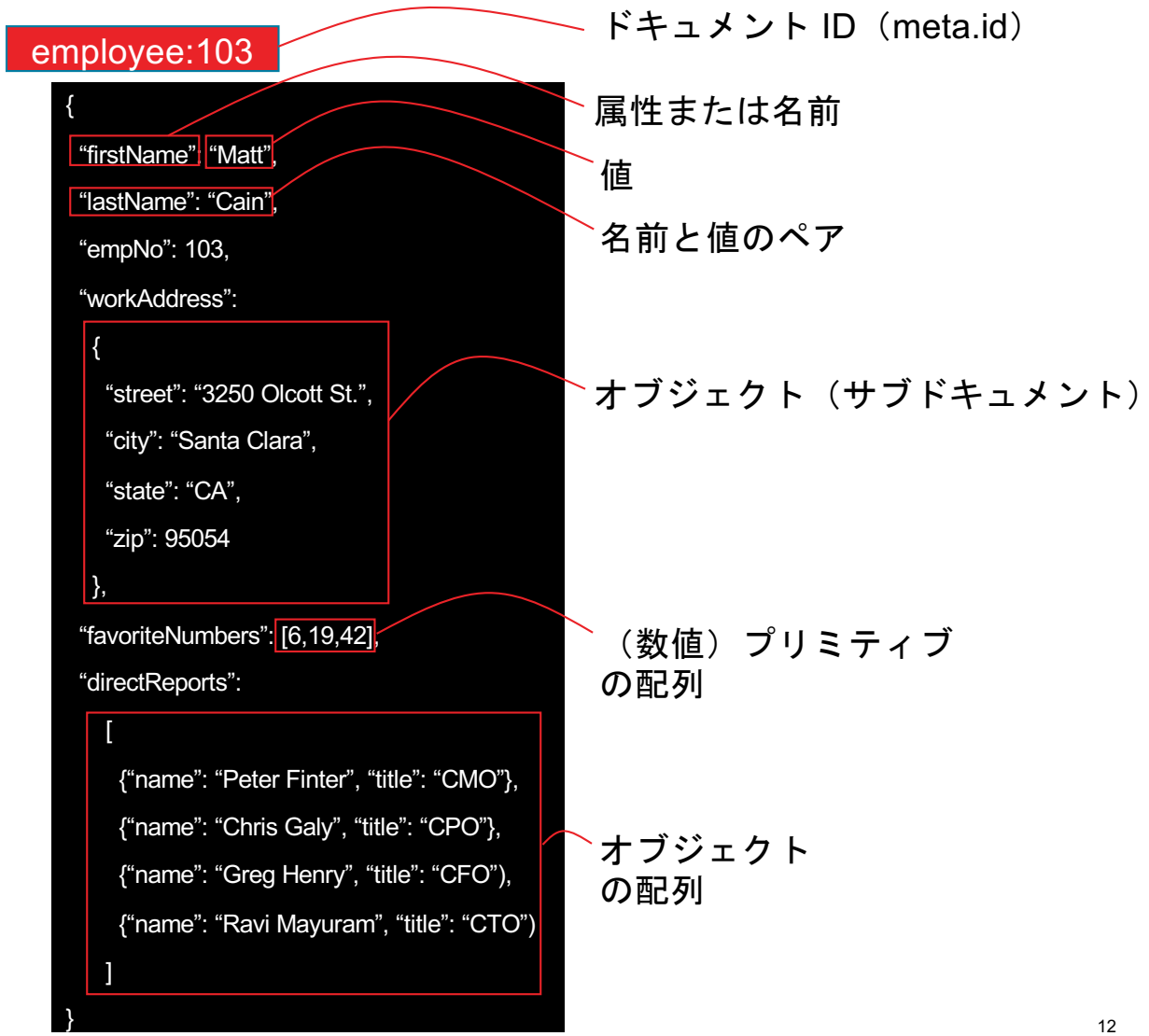
Oracle のようなレガシー データベースは、アプリケーションとデータベースの間のインターフェイスがクエリ言語 (SQL) に制約されるという誤った感覚に陥らせます。

N1QLだけを使っていても、Couchbaseへのアクセスを完結させることはできますが、それは貧弱な「最適化の選択」になります。 最適なアプリケーションでは、適切な目的のために Couchbase インターフェイスの組み合わせを使用します。

3

JSON ドキュメント
そして KEY-VALUE アクセス

JSON ドキュメントの構造



JSON ドキュメントデザインの選択肢

属性名

- 長さ
- スタイル
- 一貫性
- 可読性
- ケース
- 予約語を避ける

ドキュメント メタデータ

- タイプ
- スキーマバージョン
- ドキュメント作成時情報
- ドキュメント修正時情報
- 企業ニーズに応じた追加情報

タイムスタンプ フォーマット

- 長さ
- スタイル
- 一貫性
- 可読性
- ケース
- 予約語を避ける

エンプティ値 vs NULL 値

- データの重複を避ける
- それは未知の値か?
- デフォルト値があるか?
-

ドキュメント キー

- 人が理解可能
- 決定論的
- 生成
- 埋め込み型

属性名
- 長さ
- スタイル
- 構成

簡潔さはスケールにおける美徳 (11文字を3文字に減らせば、 10億のドキュメントで 8 GB の節約)

geo vs *coordinates*

自己説明的な名前は読みやすさを高め、作業量を減らす

userName vs *usysLogintxt*

一貫性によってバグを軽減

firstName または first_name または firstname – 1つを選び、一貫性を保つ

配列フィールドには複数名を使用し、他のフィールドには単数名を使用する

*“phone**s**”: [xxx, yyy], “address”: {…}, “genre”: “comedy”, “scale”: 2.3*

英字、数字、$、またはアンダースコアのみを使用する– 他の特殊文字を避ける

first_name vs *first-name*

属性名で避けるべきこと

- N1QL/SQL++ 予約語 (利用する場合、エスケープが必要になります)

  user, password, type, bucket, cluster, role, select, insert etc.

- JavaScript 予約語

  break, case, catch, continue, debugger, default, delete, do, else, finally, for, function, if, in, instanceof, new, return, switch, this, throw, try, typeof, var, void, while, with

  そして将来の予約語

  class, const, enum, export, extends, import, super

- アンダースコアから始まる名前(Couchbase Mobileとの互換性に問題あり)

  docType vs *_docType*

ケースを述べよ!

スネークケース
first_name

ケバブケース
first-name

キャメルケース
firstName

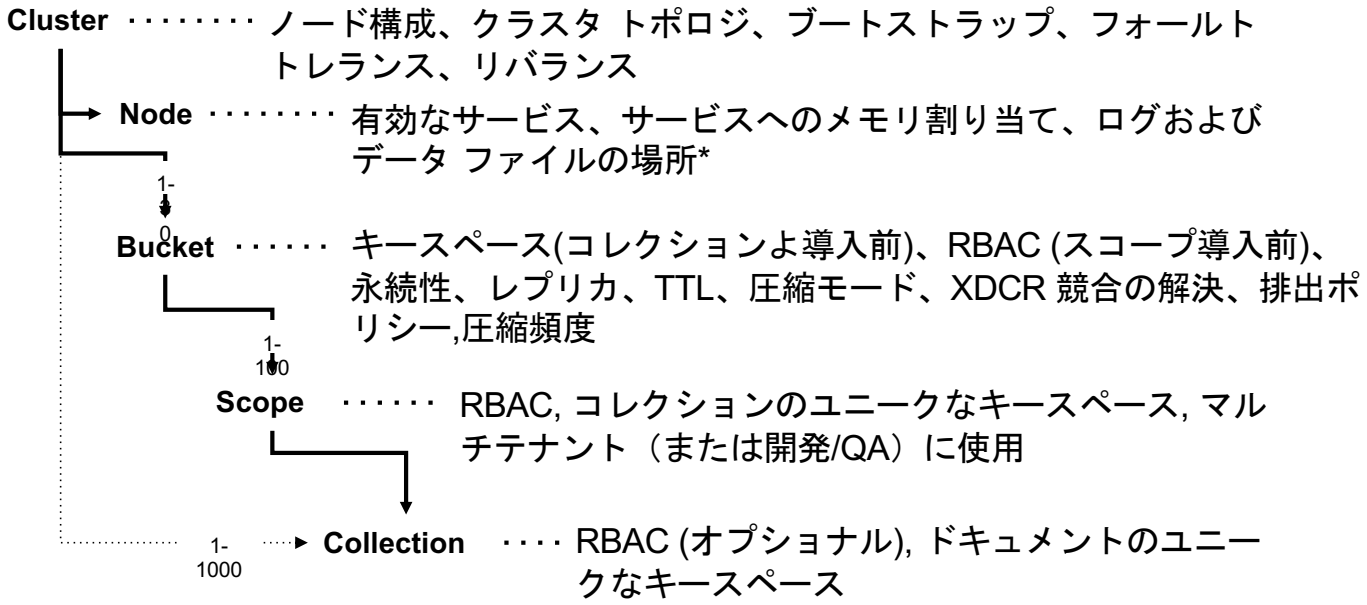
🚫 避ける：アンダースコア分、余計な文字を使うことになる

⊗ ダッシュを使用しない：N1QL パーサーはこれを 'マイナス' と混同します。

✓ ベスト・チョイス (Pythonプログラマには申し訳ない)

バケットとスコープとコレクション

リレーショナル データベース	Couchbase
Database	Bucket
Table	Document w/ type フィールド (< v7) Collections (> v7)
Row	Document
Column	Attribute



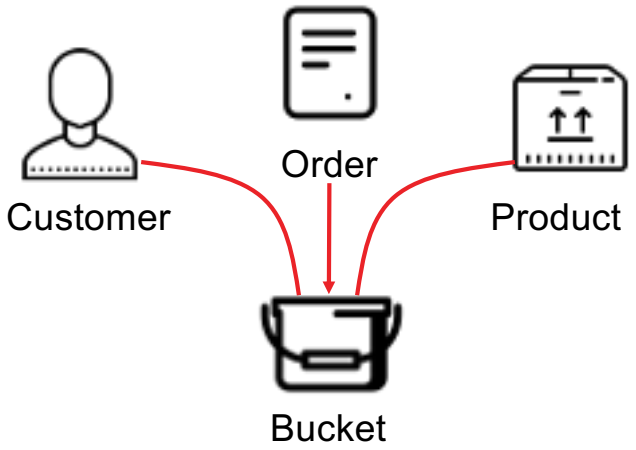
* 推奨されるベスト プラクティスは、すべてのノードでファイルの場所を統一することです。

‘type’ フィールド

バケットは、ドキュメントを格納するために割り当てられたリソース (メモリ) を定義します。

コレクション導入前において、一意のキースペースも定義します。

コレクション導入前において、異なる種類のドキュメントは、一般的に共通のバケットで混在しています。



ルートtype または type フィールド？

```
{
  “order”: {
      “custId”: 845927,
      “id”: 789434,
      “payments”: [
         {”method”: “credit”, “cardNo”: xxxxxxxxxxxxxxx, “amt”:
146.03}
    ]
  }
```

```
{
  “docType”: “order”,
  “custId”: 845927,
  “id”: 789434,
  “payments”: [
      {”method”: “credit”, “cardNo”: xxxxxxxxxxxxxxx, “amt”: 146.03}
  ]
}
```

どちらが良いか?

ドキュメント管理フィールド
(提案)

属性	目的
docType	ドキュメント タイプ(N1QL でクエリ/フィルタリング可能)
schema	ドキュメント スキーマのバージョン(提案: セマンティックバージョニング, タイムスタンプ, git コミット id)
created	作成タイムスタンプ
createdBy	ドキュメント作成ユーザー(システム)
modified	修正タイムスタンプ
modifiedBy	最後にドキュメントを修飾したユーザー（システム）

- ドキュメント全体に分散しないよう、単一のサブドキュメントでドキュメント管理データを保持する
- クエリで使いやすいよう、標準的な、短い、競合にしない名前(例: 'doc')を使用する
- メタデータを示すために、アンダースコアを使用しない
- ‘meta’ という名称を使用しない (Couchbase のドキュメントメタデータとの混同を防ぐ)
- meta.id() は、ユーザーが使用する必要がある唯一の Couchbase のメタデータ フィールドです。

```
{
  “doc”: {
     “docType”: “person”,
     “schema”: “1.3.1”,
     “created”:  1538397313000
   },
  “name”: {
    ”firstName”: “Robert”,
    ”lastName”: ”Downey”,
    “suffix”: “Jr.”
  }
}
```

タイムスタンプ オプション

一般原則:

- 常に UTC でタイムスタンプを保存する
- ミリ秒単位までユースケースに関連しているかを検討する

ISO 8601

"modified": "2020-03-14T07:35:13Z"

これは良い選択ですか?

Pros:
- 読みやすい
- ソートが容易

Cons:
- スペース効率が悪い
- クエリが難しい
- インデックス作成が困難
-

Time component array

"modified": [2020,3,14,7,35,13, "UTC"]

これは良い選択ですか?

Pros:
- 読みやすい
- インデックス作成が容易

Cons:
- クライアント コードで理解が困難
- オフセットに関する合意が必要

Unix/Epoch

"modified": 1584171313000

これは良い選択ですか?

Pros:
- スペース効率が高い
- ソートが容易
- N1QL 日付関数を使用して変換/抽出
-

Cons:
- 人間が読めない
- タイム ゾーンが明らかでない (UTC を使用する!)

JSON ドキュメント内のデータプレゼンスの 4 つの状態

フィールドに値が指定されているケース

```
SELECT qty
WHERE discount IS VALUED
```

```
{
  “qty”: 6,
  “discount”: 1.12
}
```

フィールドに値がないケース

```
SELECT qty
WHERE discount IS NOT VALUED
```

```
{
  “qty”: 4,
  “discount”: “”
}
```

フィールドが見つからない可能性もあります

```
SELECT qty
WHERE discount IS [NOT] MISSING
```

```
{
  “qty”: 9
}
```

フィールドは明示的に null である可能性があります

```
SELECT qty
WHERE discount IS [NOT] NULL
```

```
{
  “qty”: 3,
  “discount”: null
}
```

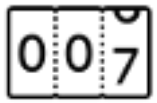
いつ使用すべきか？ :
‘null’ vs ‘missing’ vs ‘not valued’
既知のデフォルト値を保存する必要があるか?

ドキュメント キー

決定論的

- クライアントが持っている可能性が高いデータから構成可能
- ドメイン内で一意
- **不変（IMMUTABLE）でなければなりません**
- PII、機密情報、保護情報の利用を避ける

OR



生成

- カウンター, GUIDS
- 決定論的キーがない場合のみのオプション
-

コレクションが利用可能になるまで (v7) 型識別子と区切り記号をキーの前に付けてください。

cust:john.doe@mail.com

order:jdoe:6

キーに適した区切り文字は何か? 不適切な区切り文字は?

キーに型（接頭辞）が含まれる場合、ドキュメントに type 属性が必要か?

生の GUID を使用する必要がるか (例: 6c80e5ce-a5bb-42f8-bf9d-aaf33e0f5eef)?

ドキュメントにドキュメント ID を含める必要があるでしょうか?

4 エンティティ間の関係と非正規化

従来のリレーショナル構造

正規化されたデータをオブジェクトにマッピングすると、非常にコストが高くなります(多数の結合)

データの複雑さ(異なる支払タイプなど)が、構造の複雑さ（や空の列の発生）につながります。

CustomerID	Name	AddressID	Email
ABC123	BipCo	abc123	
XYZ234	Acme	xyz456	

ProductID	Name	Description	Price
774477	Widget	Fit for all uses.	$1.04
	Gizmo	Perfect for that.	$19.99
	Gadget	Just what I need.	$50.00

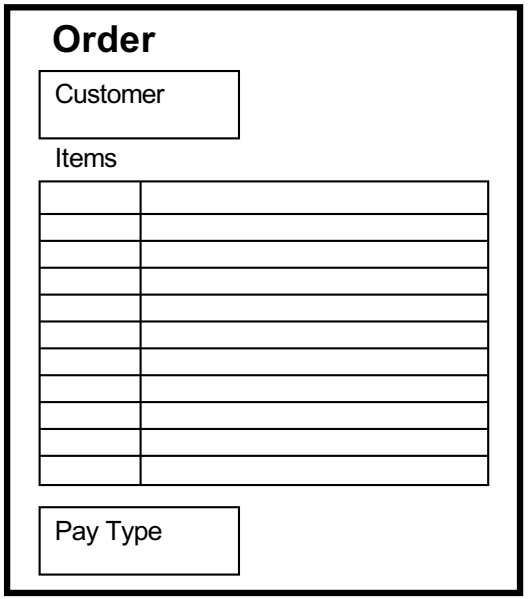
OrderID	CustomerID	ProductID	Qty
2098740	ABC123	774477	1
8308643	ABC123	332299	36

CustomerID	AddressID	Type	Street	City	State	ZIP
XYZ234	477564	Bill	123 Sesame	New York	NY	10128
				New York	NY	10128

PaymentID	Type	Amount
80982	PayPal	$134.67
71954	Visa	$96.30

PaymentID	CardNum	Exp	A
71954	546759312549862	11/22	0
80978	645858963214895	08/24	13876544687435

PaymentID	UserID	Transaction
80982	carlthefrog	46529955364
87894	ironman345	46557945335



埋め込みまたは参照?

最初に「**埋め込み**」を検討します。
合理的な理由がある場合に「**参照**」の利用を考えます。

埋め込み

- 所有関係
- 読み取り頻度が、書き込み頻度を大幅に上回る
- データが小さい
- データ毎のわずかな不整合(inconsistency)は問題ない
- 速度に最適化
-

OR

参照

- 非所有関係
- データの整合性(consistency)が重要
- ドキュメントは頻繁に更新される
- キャッシュメモリ利用の最適化
- ドキュメントサイズの削減
- 一意のキーが必要

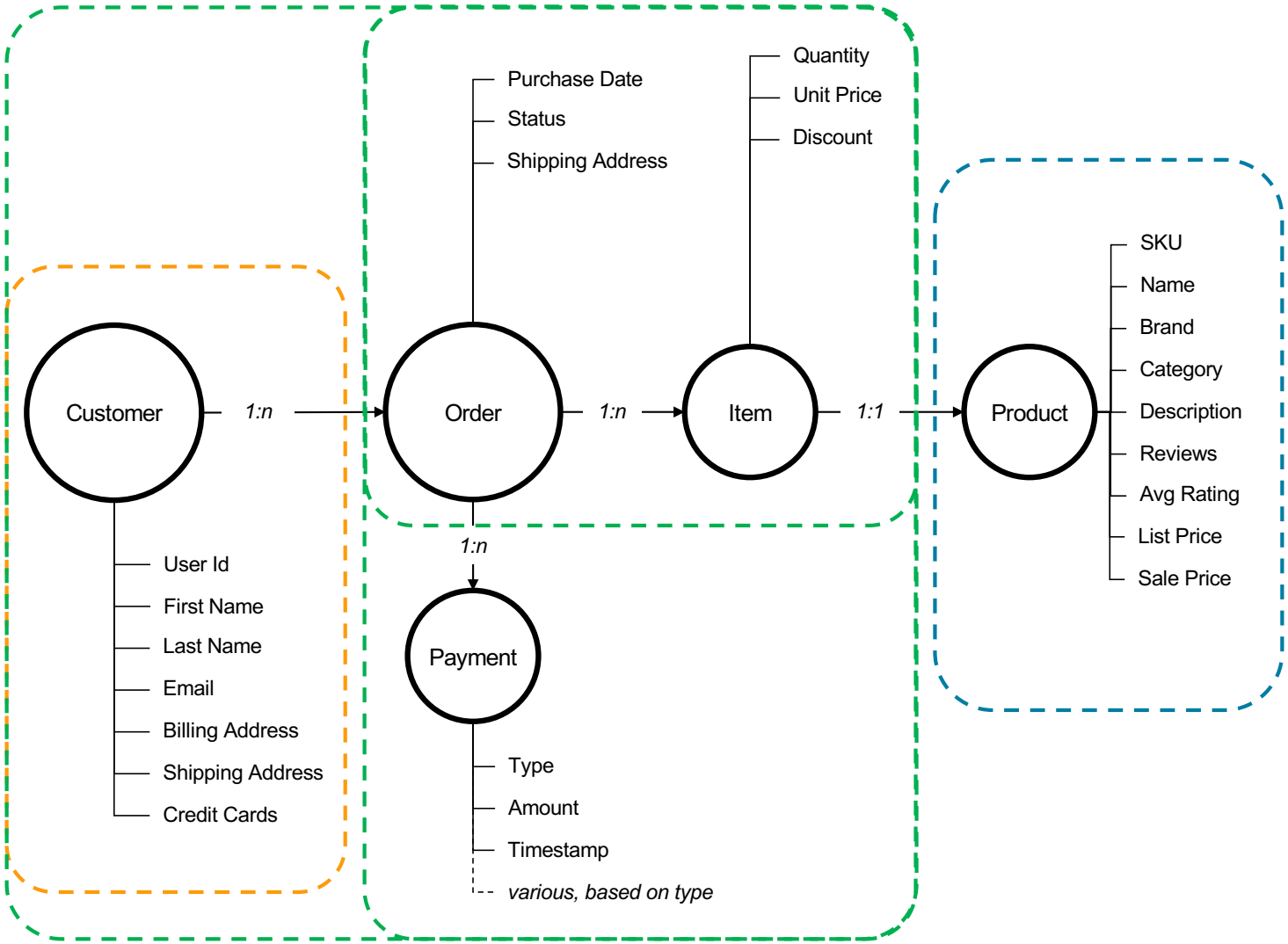
Couchbase は、以下をサポート:

- 単一ドキュメントのアトミックな更新
- サブドキュメントの読み取りと更新
- 複数ドキュメントACID トランザクション

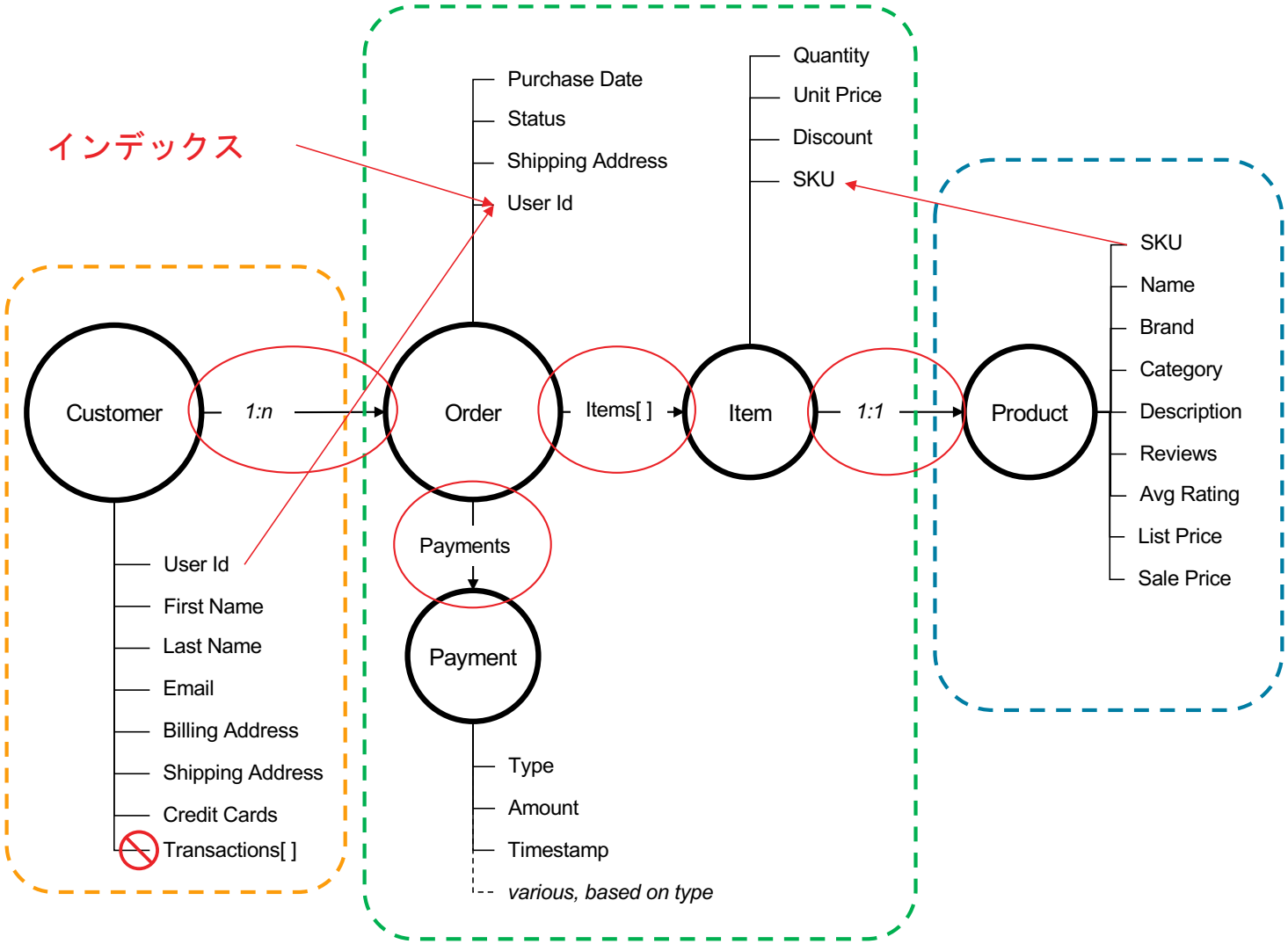
プライマリ オブジェクトの選択

主な考慮事項:

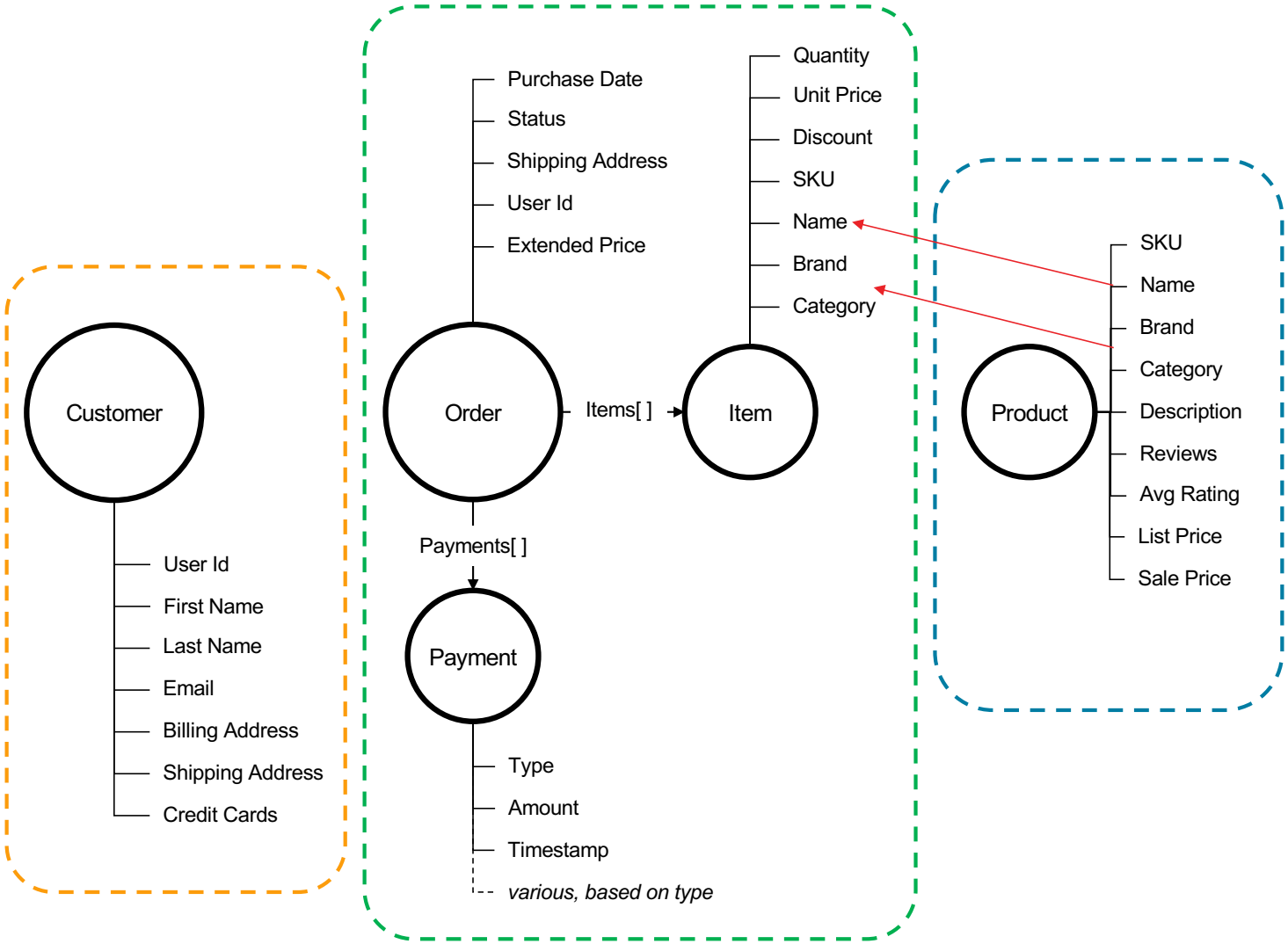
- 所有関係
- 同時データ更新
- オブジェクト サイズ
-



プライマリ オブジェクトの選択



緩和された正規化



5 配列と配列のインデックス

配列の適切な使用

N1QL には、配列にアクセスして操作するための関数が多数あります。
Couchbase SDK は、配列に対して直接データ追加(append/prepend)できます。

もし … 順序は重要であるなら、配列が適切である可能性が高い

しかし … 他の構造の方が、クエリが簡潔になったり、パフォーマンスが高い場合は、配列を使用しないでください。

配列には、同じ型の要素だけを含める必要があります。配列内に複数の型要素を持つことはレッドフラグです。

配列は小さくする必要があり、そうでなければ親オブジェクトが膨らむ危険性があります。– 参照の配列に置き換えます。

設計時に最大配列長という概念を持っている必要があります。最大配列長制限を持つ事ができない場合は、外部ドキュメントへの参照を使用するか、子から親への逆参照を使用します。

配列要素の繰り返しを探す（要素を取り出す）

```
{
  “dealerName”: “Joe’s Cars”,
  “inventory”: [
     {“format”: “SUV”, “manu”: “BMW”, ”model”: “X1”, “doors”: 5, “price”: 62567},
     {“format”: “sedan”, “manu”: “Mercedes”, ”model”: “E300”, “doors”: 4  “price”: 83465},
     {“format”: “sedan”, “manu”: “Lexus”, ”model”: “X1”, “doors”: 4, “price”: 51342},
     {“format”: “SUV”, “manu”: “Ford”, ”model”: “Escape”, “doors”: 5, “price”: 42993}
  … + 100 more
  ]
}
```

可能性の高い検索条件は何ですか??

どのように、データモデルを最適化/簡素化する事ができるでしょうか？

それぞれの（car）オブジェクトが大幅に大きかった場合はどうなりますか?

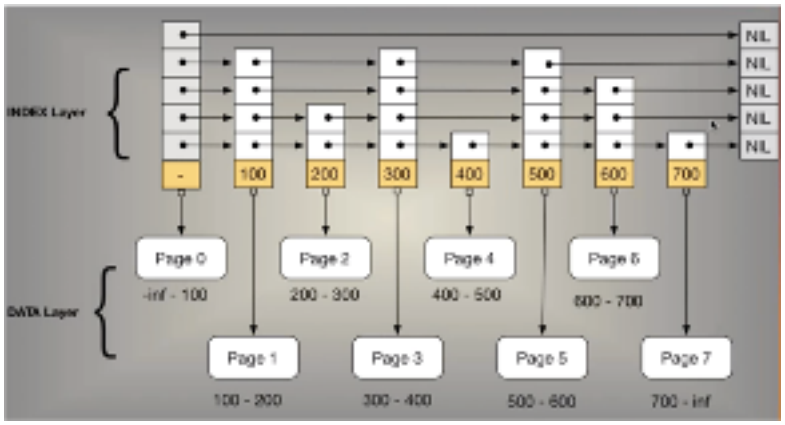
Plasmaを用いたグローバルセカンダリインデックス

Couchbase GSI (Plasma) 主要な設計基準:

- 書き込み最適化、ロックフリー、メモリとディスクの両方を活用
-

B ツリー構造は書き込みの競合が多すぎます。

そのため、Couchbase Plasmaストレージはロックフリーである、スキップリストを使用します



理解すべきことは、インデックス内では、各ドキュメントに対して、インデックス指定された各フィールドのコピーを持つ個別のレコードが作成されることです。
インデックスは、情報を「フォーク」に格納するのではなく、「葉 (ページ)」 にのみ格納します。

配列の配列

```
{
  “dealerName”: “Joe’s Cars”,
  “inventory”: [
     {“format”: “SUV”, “manu”: “BMW”, ”model”: “X1”, “doors”: 5, “price”: 62567,
      “prevOwners” [
         {“name”: “John Doe”, “purchaseDate”:1583798400, “zipCode”: 90877},
         {“name”: “Jane Smith”, “purchaseDate”:1522627200, “zipCode”: 90766},
         {“name”: “Bill Jones”, “purchaseDate”:1453637200, “zipCode”: 70893},
         {“name”: “Gail Peters”, “purchaseDate”:142398200, “zipCode”: 71844}
         ]
     },
     {“format”: “sedan”, “manu”: “Ford”, ”model”: “Escort”, “doors”: 4  “price”: 23465,
      “prevOwners” [
         {“name”: “Jim Black”, “purchaseDate”:1572652800, “zipCode”: 50981},
         {“name”: “Sue Cook”, “purchaseDate”:1459555200, “zipCode”: 51876}
         ]
     },
… + 100 more
  ]
}
```

Array Level 1

Array Level 2

配列インデックスとデータストレージ

ディーラーの名前(dealerName)に加え、そのディーラーが持っているメーカー(inventory.manu)、モデル(inventory.model)、および以前の所有者の名前(inventory.prevOwners)に対してインデックスを作成すると想像してみてください.

```
CREATE INDEX idx_prevOwner ON cars(
   dealerName,
   ALL ARRAY i.manu FOR i IN inventory END,
   ALL ARRAY i.model FOR i IN inventory END,
   All ARRAY
         (DISTINCT ARRAY p.name for p IN i.prevOwners END)
       FOR i in inventory END ;
```

この結果、次のインデックス エントリが作成されます。

Joe's	BMW	X1	John Doe
Joe's	BMW	X1	Jane Smith
Joe's	BMW	X1	Bill Jones
Joe's	BMW	X1	Gail Peters
Joe's	Mercedes	E300	Jim Black
Joe's	Mercedes	E300	Sue Cook

大きな配列では、インデックス内のデータの繰り返しが大量のリソースを消費する可能性があります。

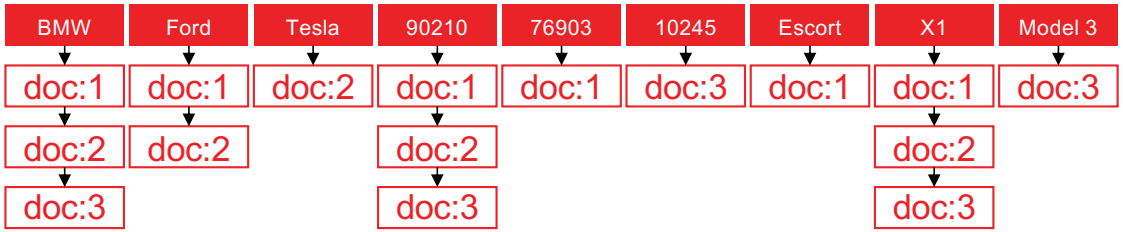
どのようなオプションがこのリソースの圧力を軽減しますか?

全文検索

フルテキスト検索 (FTS) は、'Bleve' (BLEV-ee) と呼ばれる反転インデックスを使用します。

GSI インデックスとは異なり、個々のドキュメントは FTS インデックスに複数のエントリを持つことができるため、配列やその他のデータセットのインデックス作成に適したオプションが得られる**場合があります**。
FTS クエリは、コーダーの基準に基づいてドキュメントを「スコアリング」するようにも設計されています。こうした非常に（全文検索）特有の要件を実現するために設計されていますが、より一般的なクエリや「あいまい」クエリにも優れています。

“doc:1”: {“dealerName”: “Joe’s Cars”, “inventory” …}
“doc:2”: { ”dealerName”: “Phil’s Cars”, “inventory” …}
“doc:3”: { “dealerName”: “Irma’s Cars”, “inventory” …}”



FTS タームは、N1QL WHERE 句の一部として埋め込むことができます。
かつ同時に
K-V、N1QL、および FTS という異なるAPIの種類を組み合わせて、クライアントから使用することができます。

6 注意事項

警告サイン

ドキュメントサイズ> 5k

小さなドキュメントにリファクタリングする必要があるかどうかを判断するために、モデルを再検討しましょう。

最大ドキュメント サイズの推奨

紛らわされないでください。アクセスパターンと頻度を考慮する必要があります。そもそも、シンプルなほど良いと思いませんか?

%LIKE% クエリ

フルテキスト検索をオプションとして考慮する事ができます（日本語の扱いに注意）

各クエリには対応するインデックスが必要です。

データモデルは「ストレージ」バイアスで、「アクセスパターン」に最適化されていない可能性があります。
最適化のために、インデックスアドバイザを使用する事ができます。

SELECT * FROM …

すべてのフィールドに対して、クエリする必要がありますか? それならば、クライアントはクエリで、ドキュメントのキーのリストを取得し、バルク操作として K-V get を使用するべきです。

SDK タイムアウト

タイムアウト待ち時間設定を長くすることで、実際に発生しているタイムアウト問題が解決される事は、ほとんどありません

質疑応答

利用可能なその他のリソース

Couchbase Download:

https://www.couchbase.com/downloads?utm_source=field_event&utm_medium=workshop&utm_campaign=developerworkshop

For More Information on:

- Couchbase Cloud: https://www.couchbase.com/products/cloud
- Couchbase Academy: https://couchbase.com/academy
  - Certification: https://couchbase.com/academy/certification
- Couchbase Professional Services: https://www.couchbase.com/professional-services

Our Website: http://Couchbase.com