データベース基礎

21新卒データベース研修 座学パート

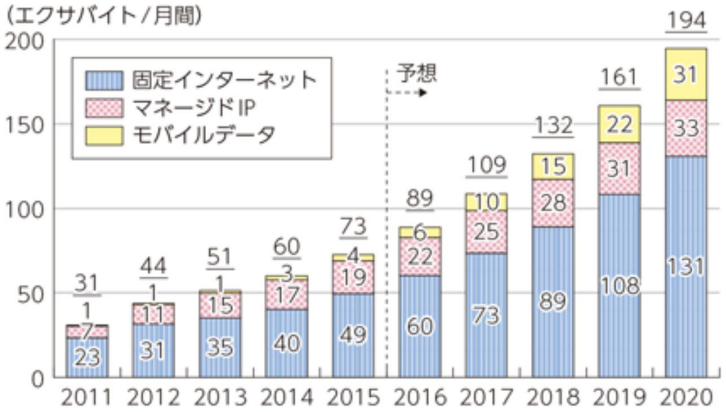
2021.04.16

目次

I. データシステム基礎

0. なぜデータベースを学ぶのか

- 演算指向からデータ指向へシフトしている時代
  - 処理能力ではなく、データの量や複雑さ、変化がボトルネックに
- **トレードオフ**を理解して**データシステム**の適切な技術選定ができるようになる
  - そのための引き出しを増やす



世界のインターネットトラフィックの推移

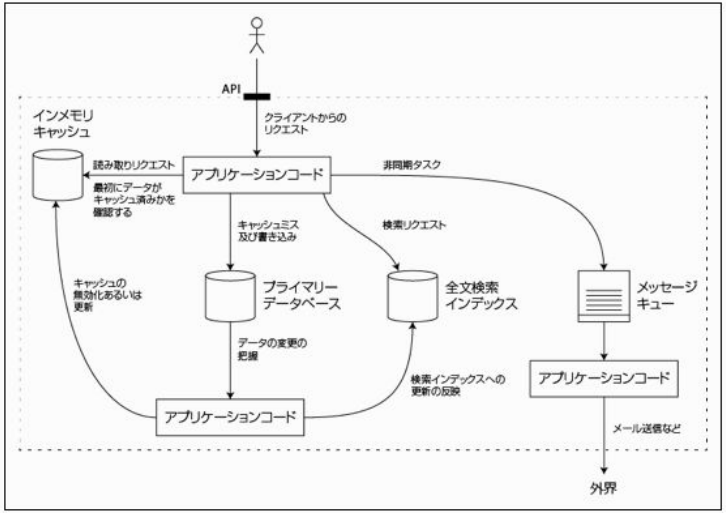
出典: 平成29年版情報通信白書, 総務省
https://www.soumu.go.jp/johotsusintokei/whitepaper/ja/h29/html/nc121210.html

1. データシステム

データシステム設計の機能要件

- データを保存/見つけられるようにする(DB)
- よく参照されるデータを記憶する(Cache)
- キーワードで検索できる(SearchIndex)

アクセスパターン、パフォーマンス特性に違いがある



DB, キャッシュ, 全文検索インデックス...
データシステムには様々な種類があり, 使い分けされている

出典: データ指向アプリケーションデザイン
https://www.oreilly.co.jp/books/9784873118703/

1. データシステム(2)

データシステム設計の非機能要件

- 障害発生時でもデータが正しいことを保証したい
- 安定したパフォーマンスを提供したい
- 負荷増大に対応したい

=> **信頼性、スケーラビリティ、メンテナンス性**

1.1 信頼性

「何か問題が生じたとしても正しく動作し続けること」

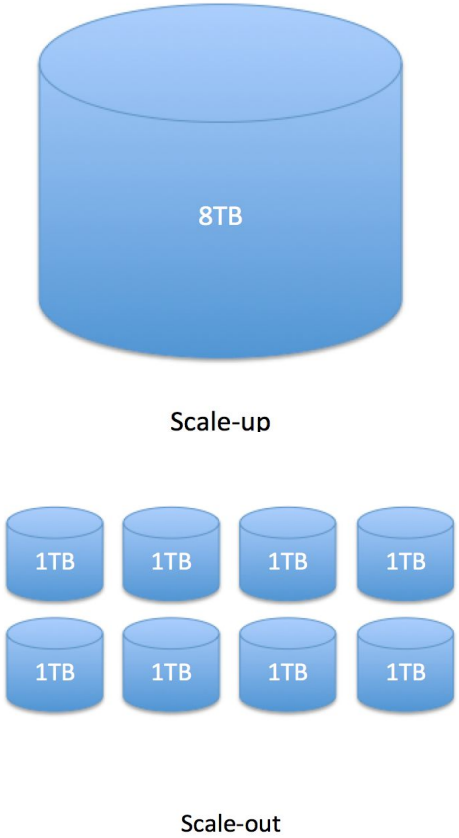
- 問題を起こしうるもの：**フォールト(fault)**
  - フォールトを見越してこれに対処できるシステムは耐障害性を持つ(fault tolerant)という
- フォールトに耐性がないと障害につながる
  - 意図的にフォールトを発生させて耐障害性の仕組みを継続的にテストする => カオスエンジニアリング

信頼性の重要さを訴えかけるイラスト

1.2 スケーラビリティ

「負荷の増大に対してシステムが対応できる能力」

- スケールアップ: マシンを強力に
  - 垂直スケーリング
- スケールアウト: 負荷分散
  - 水平スケーリング
- **one-size-fits-all なアーキテクチャは存在しない**



図引用:https://blogs.saphana.com/2014/12/10/sap-hana-scale-scale-hardware/

1.3 メンテナンス性

「メンテナンスのしやすさ」

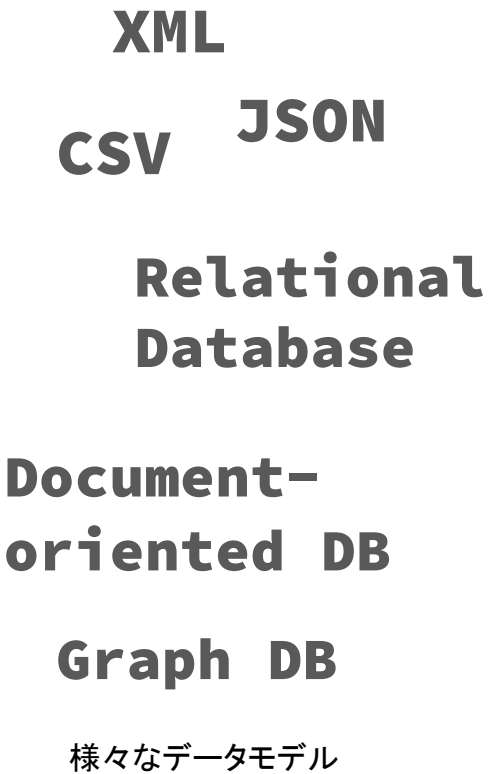
メンテナンス性を高めるための設計原則

- **運用性**
  - 健全性を可視化して効率的な管理方法での運用をしよう
- **単純性**
  - 複雑なシステムはメンテナンスのコストを増大させる
  - 抽象化でクリーンなアーキテクチャに
- **進化性**
  - システムの修正容易性
  - 単純性の高さだけでなく，開発技法も影響

2. データモデル

「データを表現するためのモデル」

- 汎用的なデータモデル
  - リレーショナルモデル
  - ドキュメントモデル
  - グラフモデル
- データモデルは, ソフトウェアのできること/できないことに大きな影響を及ぼす
- **アプリケーションに適したデータモデルを選択することは重要**



様々なデータモデル

2.1 リレーショナルモデル

「SQLのデータモデル」

- すべてのデータはオープンに配置されている
  - データはリレーション (テーブル)として構成
  - リレーションは順序無しタプル (行)の集合
- クエリオプティマイザが実行順序やインデックスを判断
  - 宣言的にデータベースを利用できる
  - (開発者がデータを手続き的に探す必要がない)
  - => アプリケーションに新しい機能を追加するのが容易
- 多対一, 多対多の関係を表現するのに優れている

```
insert into
     users(name, age)
values
     ("21新卒くん", 2021);
```

```
select
  id,name,age
from
  users;

+----+---------------+------+
| id | name          | age  |
+----+---------------+------+
|  1 | 20新卒くん    | 2020 |
|  2 | 21新卒くん    | 2021 |
+----+---------------+------+
```

MySQLにおけるデータの挿入と抽出

2.2 ドキュメントモデル

「データをJSONモデルとして保存」

- スキーマを強制しない
- ローカリティ(局所性)に優れている
  - 関連情報が一箇所に集まっている
  - 一対多のツリー構造では結合が必要がない
- 結合のサポートは弱い
- MongoDB, CouchDB, RethinkDB

```
> db.users.insert(
 {"name": "21新卒くん","age": 21}
)
```

```
> db.users.find()

{
  "_id" : ObjectId("60e..."),
  "name" : "21新卒くん",
  "age" : 21
}
```

MongoDBにおけるデータの挿入と抽出

2.3 Schema

- **スキーマオンリード**
  - データベースがスキーマを強制しない
  - データ構造は暗黙 => 読み取り時に解釈
- **スキーマオンライト**
  - スキーマを明示
  - 書き込み時にスキーマに従っていることを保証
  - スキーマの変更にマイグレーションが必要

2.4 クエリ言語

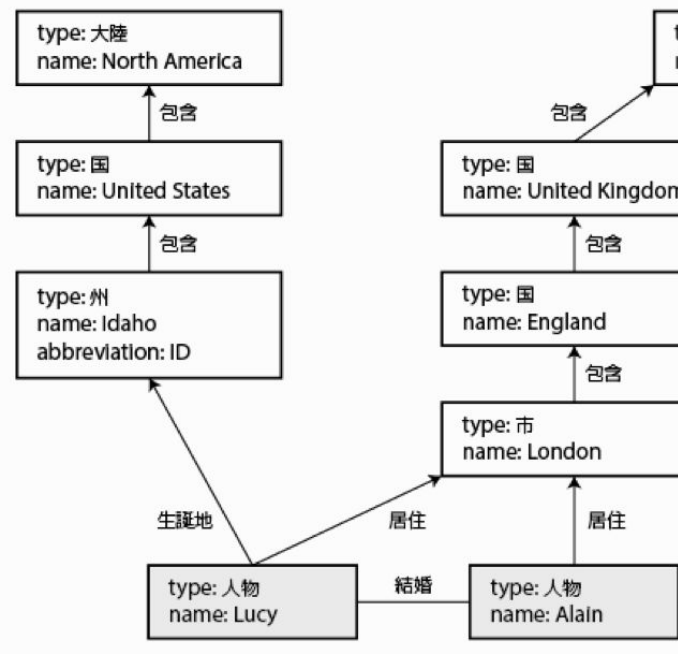
「SQLは宣言的クエリ言語」

- 宣言的なクエリ言語はDBエンジンの実装の詳細を隠蔽
  - クエリの書き換えなしにDBシステムのパフォーマンスを改善できる
- SQL, MapReduce, Cypherなど
  - JSのDOM APIやCSSセレクターも一種のクエリ言語

2.5 グラフデータモデル

「多対多の関係表現に特化したデータモデル」

- データをグラフとして表現
  - 頂点(ノード)が人やWebページなどのエンティティ
  - 辺(エッジ)がノード間の関係
- ソーシャルグラフ, Webグラフなどで利用
- DB: Neo4j, Titan, InfiniteGraph



出典: データ指向アプリケーションデザイン
https://www.oreilly.co.jp/books/9784873118703/

3. ストレージエンジン

「データの保存と取り出しを行う方法を決めるもの」

大きく分けて2種類の用途:

- トランザクション処理用途
  - OLTP: Online Transaction Processing
- 分析処理用途
  - OLAP: Online Analytic processing

特性	トランザクション処理システム（OLTP）	分析処理システム（OLAP）
主な読み取りのパターン	クエリごとに少数のレコードをキーに基づいてフェッチ	大量のレコードを集計
主な書き込みのパターン	ユーザーの入力によるランダムアクセスと低レイテンシの書き込み	バルクインポート（ETL）あるいはイベントストリーム
主な利用者	Webアプリケーションを利用するエンドユーザー/顧客	経営判断を支援する組織内のアナリスト
データの内容	データの最新の状態（現時点）	時間の経過とともに生じた出来事の履歴
データセットのサイズ	ギガバイトからテラバイト	テラバイトからペタバイト

出典: データ指向アプリケーションデザイン
https://www.oreilly.co.jp/books/9784873118703/

3.1 OLTP (Online Transaction Processing)

- エンドユーザーとやり取りするインタラクティブな用途で利用
  - ランダムアクセスと低レイテンシーな書き込みが求められる
- 大量データから特定のキーの値を効率的に見つけことに長けている
  -> インデックス
- OLTPで主流のストレージエンジンは2つ
  - Bツリー系: update-in-place. 最も一般的.
  - log-structured系: ファイルへの追記と削除のみ . 最近開発された
    - LevelDB (google bigtable)
    - Lucene (Elasticsearch)

3.2 OLAP (Online Analytic Processing)

「結果がビジネスインテリジェンス(BI)のために利用される」

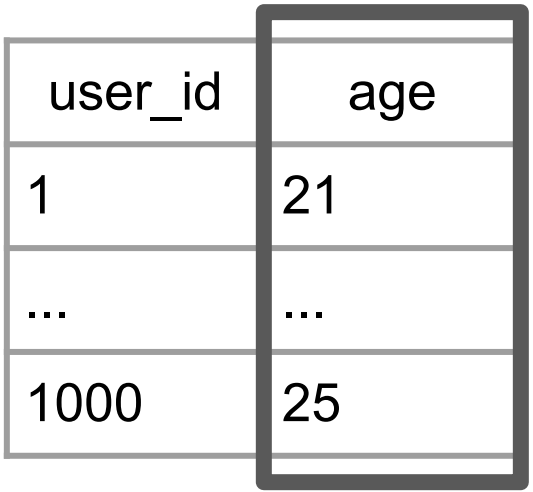
- 一般に分析用途のクエリーは高負荷&データセットの大部分をスキャン
  - パフォーマンスへ大きく影響
- 分析用途に特化した独立したDB -> データウェアハウス
- データウェアハウスでは, リードに最適化され, データのスキャン範囲を抑える工夫がされている

3.2.1 列指向

「データウェアハウスは列指向」

- 列に含まれるすべての値をまとめて保存
  - ⇔ 1つの行に含まれるすべての値をまとめて保存
- 必要な列のデータのみを取り出せる
  - `SELECT SUM(age)` が強い
  - `SELECT *` で全データスキャン
  - 行指向では `SELECT SUM(age)` でデータ全スキャン
- 圧縮しやすい
- 書き込みは苦手

列指向におけるデータの持ち方



user_id file contents: 1, …., 1000
age file contents: 21, …, 25

4. エンコーディングと進化

「アプリケーションの変化に伴い、データも変化する」エンコーディングによって互換性を担保

- **エンコーディング**
  - 「インメモリの表現からバイトの並びへの変換」
  - メモリ内のデータの持ち方とメモリ外でのデータの持ち方は異なる
- **後方互換性**
  - 古いコードによって書かれたデータを新しいコードが読める
- **前方互換性**
  - 新しいコードによって書かれたデータを古いコードが読める
  - 難しい

4.1 様々なデータエンコーディングフォーマット

- プログラミング言語固有のフォーマット
  - java.io.Serializableなど
  - 他プログラミング言語との互換性が (ほぼ)ない
- 標準化されたフォーマット
  - テキストフォーマット : JSON, XML, CSVなど
  - バイナリフォーマット : Thrift, Protocol Buffers, Avro

4.2 JSON, XML, CSV

「多くのプログラミング言語で読み書きできる標準化されたエンコーディング」

- テキストフォーマット
  - JSON, XMLはバイナリ文字列をサポートしていない
  - データサイズが大きくなりがち
- スキーマは組み込まれていない
  - 好きなスキーマでデータを格納できる
  - 合意形成がなされていれば問題は発生しない
  - ただし、それが何であれ、何かについて複数の組織間が合意するのは難しい

4.3 バイナリエンコーディング

「テラバイト級のデータを扱うために開発されたデータエンコーディング」

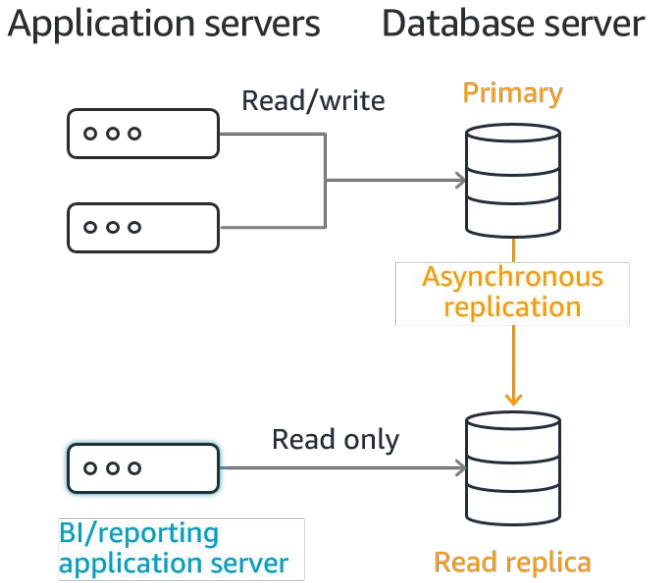
- **Thrift** (Facebook), **Protocol Buffers** (Google), **Avro**(Apache)など
  - スキーマを必要とするエンコーディング
- スキーマ情報はドキュメントやコード生成にも利用可能
- JSON, XML用のバイナリエンコーディングもある
  - JSON用にはMessagePack, BSONなど
  - XML用にはWBXML, Fast Infosetなど

Ⅱ 分散データの取り扱い

5. レプリケーション

「複数のマシンに同じデータのコピーを保持しておくこと」

- レプリケーションの目的
  - レイテンシを下げる
  - 障害があってもシステムを動作させる（可用性）
  - スケールアウトさせる（スループット）
- 大きく分けて3つのアプローチ
  - シングルリーダー, マルチリーダー, リーダーレス
- レプリケーションにおけるトレードオフ
  - 同期的か非同期か
  - 障害を起こしたレプリカの扱い

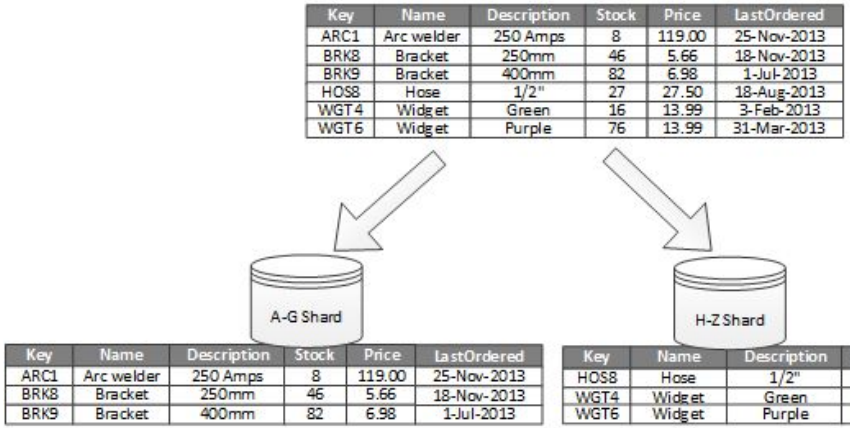


Amazon RDSにおけるレプリケーションのイメージ
https://aws.amazon.com/jp/rds/features/read-replicas/

6. パーティショニング

「データを分割して保存」

- パーティショニング = シャーディング
- スキュー (skew) : パーティショニングに偏りがある状態
  - ホットスポット: 負荷が集中しているパーティション
  - ホットスポットを作らないようなキーの設定が重要
    - [スキーマ設計のベスト プラクティス | Cloud Spanner | Google Cloud]
    - [パーティション分割テーブルの概要 | BigQuery | Google Cloud]



Key	Name	Description	Stock	Price	LastOrdered
ARC1	Arc welder	250 Amps	8	119.00	25-Nov-2013
BRK8	Bracket	250mm	46	5.66	18-Nov-2013
BRK9	Bracket	400mm	82	6.98	1-Jul-2013
HOS8	Hose	1/2"	27	27.50	18-Aug-2013
WGT4	Widget	Green	16	13.99	3-Feb-2013
WGT6	Widget	Purple	76	13.99	31-Mar-2013

Data partitioning guidance - Best practices for cloud applications | Microsoft Docs
https://docs.microsoft.com/en-us/azure/architecture/best-practices/data-partitioning

7. トランザクション

「複数の読み書きを論理的な単位としてまとめる方法」

- すべての読み書きを1つの操作として実行
- 最終的な実行結果は3種類
  - 成功 (commit) , 失敗 (abord, rollback)
  - 一部の操作だけ成功という状態は存在しない
- 保証するトランザクションの強さ(分離レベル)とパフォーマンスはトレードオフ
  - ダーティーリード : コミットされていないデータが見れてしまうこと
  - ダーティーライト : コミットされていないデータを上書きすること

参考書籍

- データ指向アプリケーションデザイン——信頼性、拡張性、保守性の高い分散システム設計の原理
  - https://www.oreilly.co.jp/books/9784873118703/