

Banco de dados das coisas

Especialização em Internet das coisas
Prof. Felipe Scheidt

Informações

Carga horária: 15h

Entrega de atividade no moodle da disciplina:

<https://ava.ifpr.edu.br/course/view.php?id=6851>

Material e código: <https://github.com/fscheidt/iotdb-21>

Período de aulas: 09/12 à **30/12**

Sobre a disciplina

Paradigmas de armazenamento e recuperação de dados relacional, distribuídos, tempo real e NoSQL. Características necessárias para aplicações em IoT como, escalabilidade, flexibilidade, disponibilidade, integração e ferramentas de análise de dados

Conteúdos específicos

1. Paradigmas de armazenamento
2. SQL (MySQL/SQLite)
3. NoSQL (MongoDB)
 - a. Projeto Python
 - b. Colab notebook
4. Análise exploratória de dados
 - a. Jupyter notebook
 - b. Pandas, scikit, matplotlib

Tópicos da aula

- Apresentação da disciplina
- Características de aplicações em IoT
- Paradigmas de armazenamento de dados
- Conceitos banco de dados

IoT e a *relação* com banco de dados

IoT == conexão de objetos do mundo físico à Internet.

Inclusão de dispositivos que *usualmente* não estão conectados a Internet: equipamento industrial, casas inteligentes, cidades inteligentes, fazendas inteligentes

Objetivos da IoT:

- Monitorar a infraestrutura
- Coletar dados do ambiente através de sensores

Banco de dados: surge como solução para integrar esses objetivos.

```
graph TD; A[Objetivos da IoT:  
- Monitorar a infraestrutura  
- Coletar dados do ambiente através de sensores] --> B[Banco de dados: surge como solução para integrar esses objetivos.]
```

Características de aplicações em IoT

Pontos a considerar:

- Dispositivos de IoT (microcontroladores) são computacionalmente *limitados*
- Banco de dados precisa ser leve para executar local OU delegar o gerenciamento dos dados a um serviço externo.
- Não há garantia de conectividade de rede.
- Dependendo da distância e localização do dispositivo as restrições podem ser mais limitantes.

Características de aplicações em IoT

- Soluções baseadas em serviço cloud dependem de acesso à Internet.
- Custo de manutenção do serviço cloud.
- Algumas soluções IoT precisam funcionar independentemente de conexão com a internet, por exemplo: veículo autônomo (?)
- Atraso em obter informações na *cloud* pode inviabilizar a tomada de decisão em aplicações real-time.
- Rapidez na análise dos dados: cenários de missão crítica/alta disponibilidade.
- Segurança dos dados de ponta-a-ponta.

Características de aplicações em IoT

- Grande volume de dados coletados de muitos sensores
- Pre-processamento e filtragem de dados local reduzindo o tamanho dos dados armazenados.
- Persistência amostral dos dados por intervalo de tempo (sumarização)
- Cache: armazenamento local

Características de aplicações IoT

Pontos a considerar:

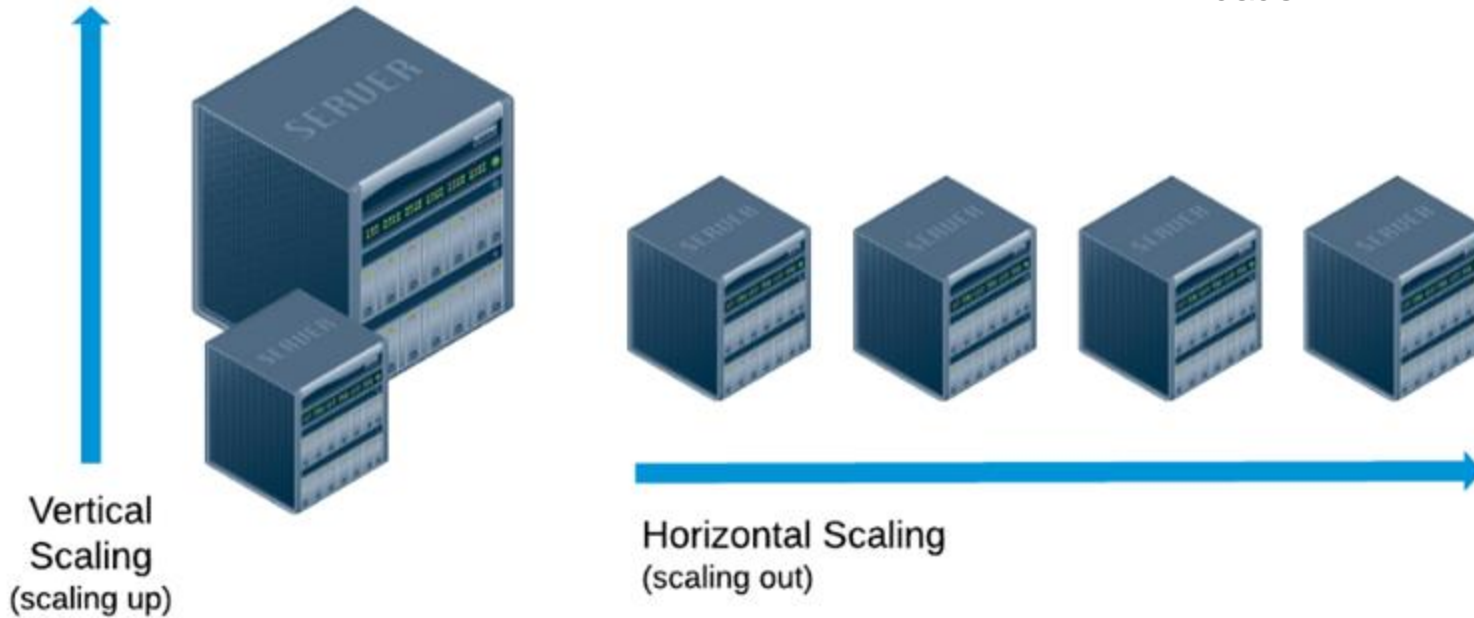
- Escalabilidade: multiplicação de sensores e explosão dos dados
- Flexibilidade: compatibilidade protocolos, tipos de redes, esquemas de dados
- Disponibilidade
- Integração

Escalabilidade

- É medida como a capacidade do sistema em tratar *requisições* simultâneas.
- Todo sistema possui um limite computacional que eventualmente é atingido.
- Escalar os recursos computacionais envolve: mais CPU, memória RAM, hard disk, largura da banda, etc...

Escalabilidade

- Escalabilidade Horizontal ou Vertical.
- Melhor opção de escalabilidade?



Big data

Big data: refere-se a bases de dados *muito grandes* no qual a complexidade para buscar um registro, por exemplo ultrapassa o poder computacional para recuperar a informação dentro de um tempo viável.

Problema: como extrair valor (conhecimento) de um grande dataset? (>**terabytes**)

Como escalar a aplicação?

Serviço cloud, NoSQL, BigQuery, ...

Técnicas: amostragem, achatamento dos dados, memcached, MapReduce, ...

Persistência dos dados

Praticamente toda aplicação necessita armazenar informação.

A informação é armazenada posterior recuperação.

Informações precisam também ser atualizadas e filtradas

Gerenciar essas operações usando arquivos de texto ou log torna-se impraticável

A escolha do **SGBD**/DBMS é um fator de decisão crítico para a aplicação.

Banco de dados (conceito geral)

- É uma coleção de dados armazenados e organizados de acordo com uma estrutura
- O que existem são diferentes variações de esquemas (pré-definido ou dinâmico)
- Um BD é gerenciado por um **DBMS** (*database management system*) ou SGBD (sistema de gerenciamento de banco de dados).
- As principais funções são armazenamento e recuperação dos dados.
- Funcionalidades são acessadas através de uma API.
 - CLI - command line interface
 - Library - implementa driver
 - GUI / Web interface

Sistemas de banco de dados

Recursos básicos esperados de um DBMS:

- Armazenamento dos dados com recuperação e atualização.
- Suporte a autenticação
- Acesso remoto
- Regras básicas de validação.
- Interface/API para manipulação dos dados

Recursos +avançados:

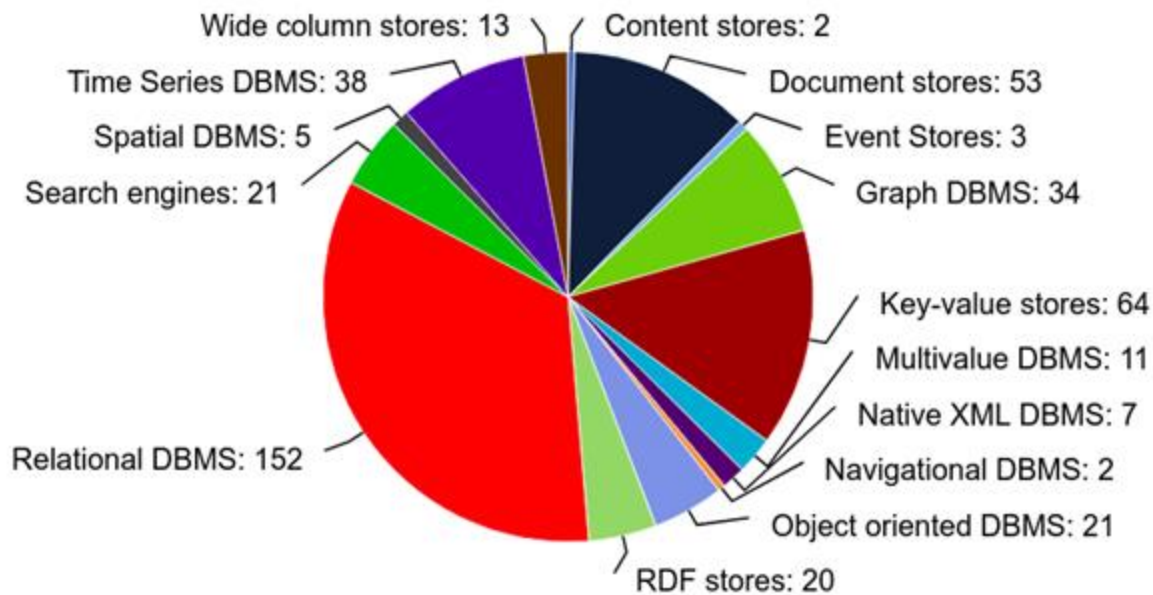
- Suporte a transações e concorrência
- Suporte a esquemas de perfis de autorização
- Escalabilidade
- Sistema distribuído

Paradigmas de armazenamento de dados

Cada paradigma de armazenamento segue uma estrutura específica para representação dos dados. Existem muitas variações, mas resumidamente temos:

1. In-memory (key-value pair)
2. Wide column
3. Document oriented
4. Relational databases
5. Graph databases
6. Time series databases
7. Multi-model

DBMS por categorias



Relational databases (RDBMS)

Paradigma relacional

- Dados armazenados em diversas tabelas
- Tabelas estão relacionadas
- Tabelas armazenam objetos e índices (chave PK e FK)
- Temos diferentes tipos de relacionamento (1-n, 1-1, n-n, etc...)
- O conjunto dessas regras é o que chamamos de *schema*.
- O schema é garantido pelo DBMS*

Estrutura tabular

Conceitos principais:

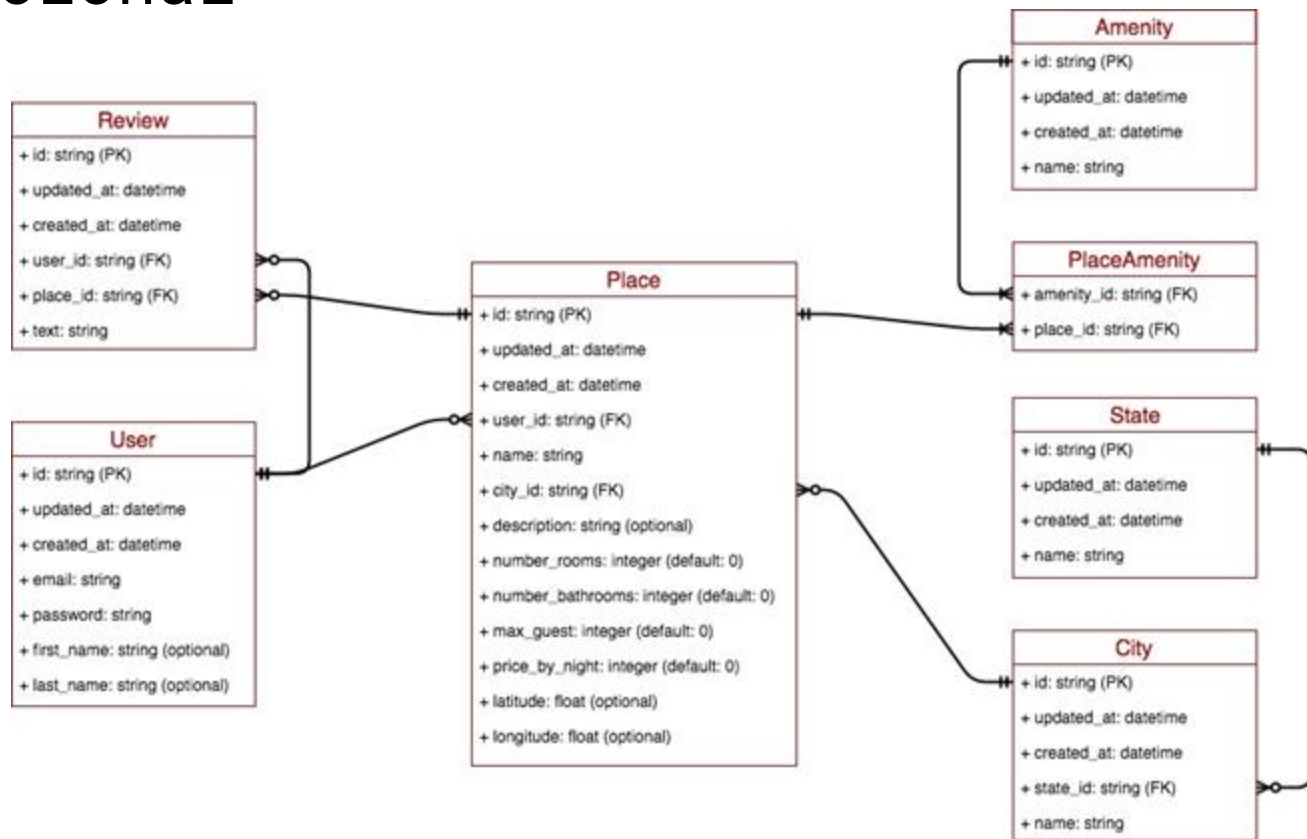
- Tabela
- Coluna
- Linha

Database Table

The diagram illustrates a database table with three columns: ID, firstName, and email. The rows are numbered 1 through 4. A red arrow labeled 'Column' points to the first column (ID). A red arrow labeled 'Row' points to the second row (ID 2). A red box highlights the second row (ID 2), and a red box highlights the first two columns (ID and firstName) for all rows.

ID	firstName	email
1	John	john@email.com
2	Paul	paul@email.com
3	Peter	peter@email.com
4	James	james@email.com

Esquema relacional



Integridade dos dados (RDBMS)

RDBMS endossam a integridade referencial:

- Consistência dos dados (validação valores nulos, tipo de dados)
- Integridade referencial (FK e PK são válidas)
- Garantir que as regras definidas no schema do database sejam respeitadas
- Prevenir que o banco de dados entre em um estado inconsistente

RDBMS

Vantagens:

- Consistência dos dados
- Organização (estrutura fixa)
- Linguagem SQL para manipulação dos dados
- Garantias ACID

Desvantagem:

- Problema com escalabilidade (difícil escalar horizontalmente)
- Consome mais recursos - single server based
- Complexidade de manutenção (esquemas em desenvolvimento)
- Consulta a múltiplas relações gera queries complexas + overhead

ACID

- **Atomicidade:** A transação deve ter todas as suas operações executadas em caso de sucesso ou nada deve ser realizado
- **Consistência:** A execução de uma transação deve levar o banco de dados de um estado consistente.
- **Isolamento:** evitar que transações paralelas interfiram umas nas outras (controle de concorrência)
- **Durabilidade:** Garante que os dados estarão disponíveis em definitivo (armazenados permanentemente)

Paradigma NoSQL

- *Non-relational database, Non structured data, Not only sql*
- Documentos não precisam seguir o mesmo esquema (*schema-less* = sem restrições)
- “Não há relacionamentos”
- A ideia é agrupar todas as informações necessárias em um só lugar.
- Ganho de performance em operações de leitura pois não é preciso buscar em outras tabelas.
- Desvantagem é que há um certo nível de dados duplicados.
- Operações de update devem levar isso em consideração.

NoSQL

Flexibilidade para realizar atualizações no esquema do database sem quebrar o código.

Robustez para aplicações big data e real-time.

Escalabilidade (horizontal e vertical)

Não há garantia que determinado campo exista ou o tipo de dados.

Exige um tratamento extra na programação.

Overhead em situação de escrita em várias collections.

NoSQL

Vantagens:

- Flexibilidade
- Escalabilidade
- “Velocidade”

Desvantagens:

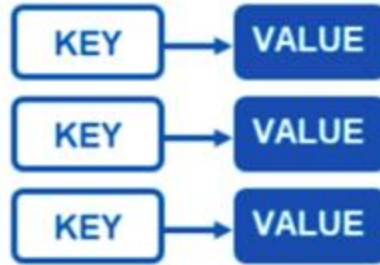
- Documentos de uma mesma collection podem ter campos diferentes, o que pode gerar resultados inesperados (tratar no código)
- Duplicação de dados

Família NoSQL

Paradigmas de modelos da família NoSQL



Column based



Key-value



Graph



Document



















Document oriented databases (NoSQL)

Paradigma orientado a documento.

Documentos representam os dados armazenados usando a notação JSON

```
{
  "_id":4892,
  "pais":"Brasil",
  "densidade": 64.7,
  "is_opec_member": false,
  "test": null,
  "last_census":"2010-10-21",
  "estados":[
    {
      "name":"Parana", "region": "sul"
    },
    {
      "name":"Santa Catarina", "idh": 0.774
    }
  ]
}
```

Ranking - document model database

Rank			DBMS	Database Model	Score Oct 2021
Oct 2021	Sep 2021	Oct 2020			
1.	1.	1.	MongoDB 	Document, Multi-model 	493.55
2.	2.	2.	Amazon DynamoDB 	Multi-model 	76.55
3.	3.	3.	Microsoft Azure Cosmos DB 	Multi-model 	40.29
4.	4.	4.	Couchbase 	Document, Multi-model 	27.91
5.	5.	 6.	Firebase Realtime Database	Document	19.02
6.	6.	 5.	CouchDB	Document, Multi-model 	15.79
7.	7.	7.	MarkLogic 	Multi-model 	9.43
8.	8.	8.	Realm 	Document	9.29
9.	9.	9.	Google Cloud Firestore	Document	8.37
10.	 11.	 20.	Virtuoso 	Multi-model 	4.69

In-memory databases (key-value)

- Simplicidade: baseado no esquema par chave-valor.
- Chave e valor suportam vários tipos de dados
- Útil para armazenamento dados temporários como cache de consultas, resultados de chamadas a API, etc.
- Limitação da quantidade de dados a quantidade de memória RAM
- Não tem suporte a construção de queries (select, join, ...)
- Exemplos:
 - Redis
 - Memcached
 - DynamoDB

MAC table	
Key	Value
10.94.214.172	3c:22:fb:86:c1:b1
10.94.214.173	00:0a:95:9d:68:16
10.94.214.174	3c:1b:fb:45:c4:b1

Ranking popularidade (key-value databases)

Rank			DBMS	Database Model	Score
Oct 2021	Sep 2021	Oct 2020			
1.	1.	1.	Redis +	Key-value, Multi-model ⓘ	171.35
2.	2.	2.	Amazon DynamoDB +	Multi-model ⓘ	76.55
3.	3.	3.	Microsoft Azure Cosmos DB +	Multi-model ⓘ	40.29
4.	4.	4.	Memcached	Key-value	26.02
5.	5.	↑ 6.	etcd	Key-value	10.19
6.	6.	↓ 5.	Hazelcast +	Key-value, Multi-model ⓘ	9.66
7.	7.	7.	Ehcache	Key-value	6.94
8.	8.	↑ 9.	Riak KV	Key-value	5.88
9.	↑ 10.	↑ 12.	Ignite	Multi-model ⓘ	5.41
10.	↓ 9.	↓ 8.	Aerospike +	Key-value, Multi-model ⓘ	5.13

Source: <https://db-engines.com/en/ranking/key-value+store>

Graph databases

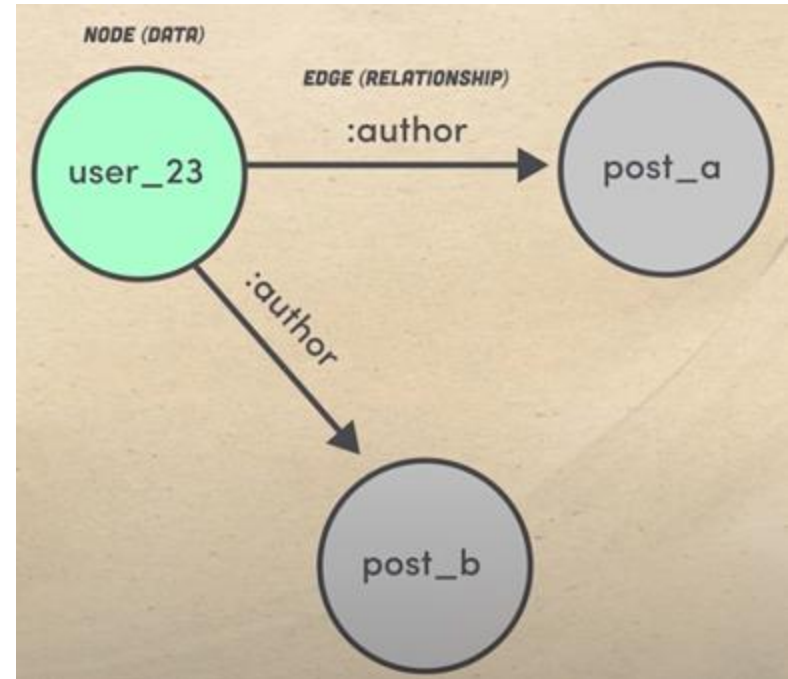
Dados são armazenados em grafos

Databases:

- Neo4j

Aplicações:

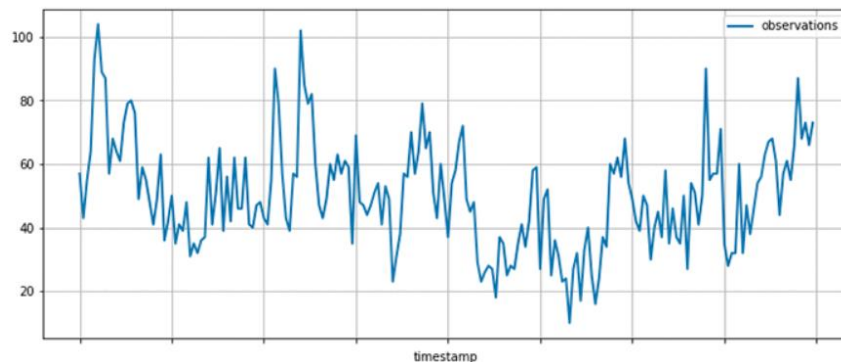
- redes sociais
- bases de conhecimento
- sistemas de recomendação.



Time series databases

1. DBMS especializado em armazenar dados associados a um tempo, também chamado de séries temporais.
2. Dados provenientes de equipamentos e sensores
3. Suporte a algoritmos de compressão de dados para gerenciamento de grande quantidades de dados.

Exemplo: Influxdb



Melhor solução?

Cada paradigma de armazenamento deve ser analisado levando-se em consideração os **requisitos** da aplicação.

Não existe o melhor paradigma.

Nos casos onde múltiplas soluções atendem aos requisitos da aplicação? Buscar a solução mais simples.

Fato é que muitos DBMS já implementam soluções de diferentes paradigmas, que podemos chamar de multi-model databases.

Comparação SQL vs. NoSQL

Não há uma melhor solução. Analise dos requisitos da aplicação.

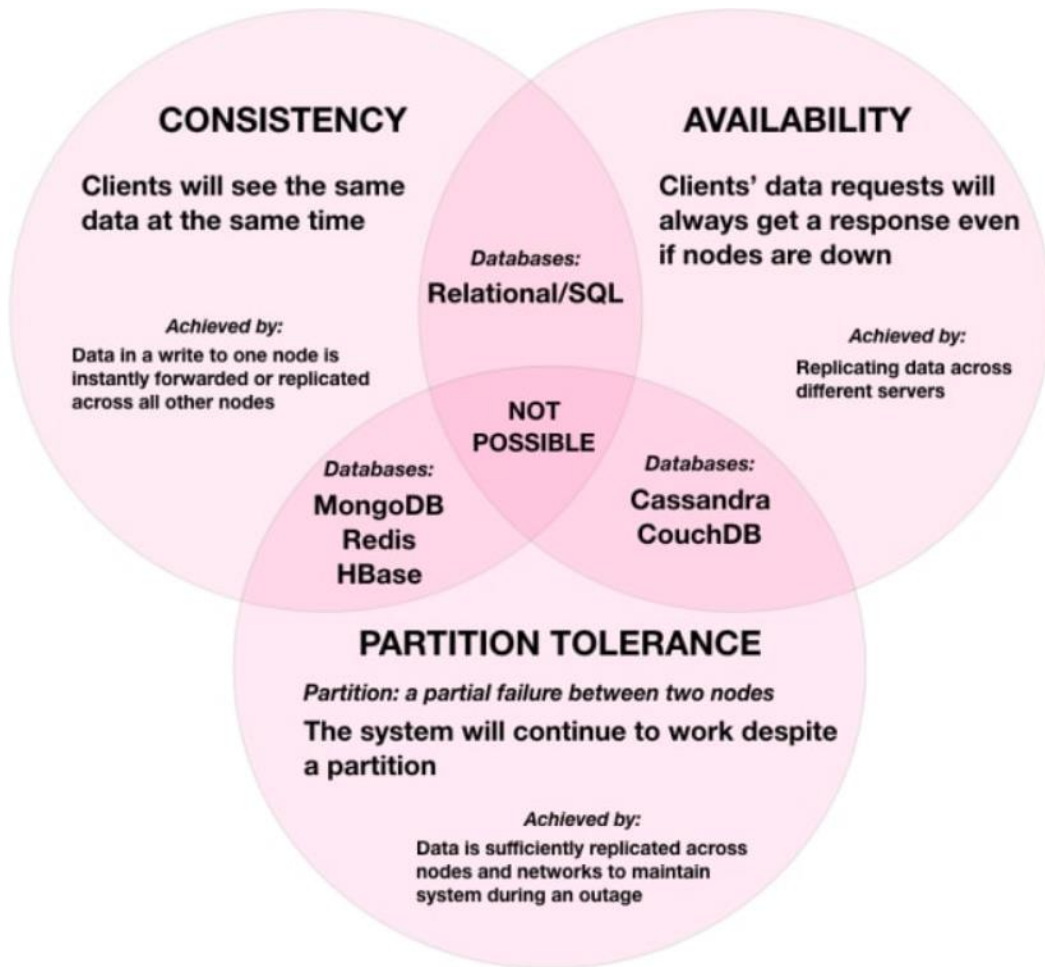
Responder?	SQL	NoSQL
Esquema de dados dinâmico/em evolução?		✓
Operações de update são frequentes	✓	
Muitos dados e constante crescimento		✓
Garantia de consistência dos dados?	✓	

Processo de decisão

- Preciso de um mecanismo ACID?
- Escalabilidade
- Disponibilidade
- É tolerável sacrificar eventualmente consistência?
- Qual a estrutura/formato dos dados que preciso armazenar?
- Performance de leitura dos dados é crítico?
- Performance de escrita? (update)

CAP Theorem

- Consistency
- Availability
- Partition Tolerance



Setup de serviços cloud

Criar conta nas plataformas e serviços que serão usados na disciplina:

- Google colab - gmail account
 - Acessar: <https://colab.research.google.com/>
- Atlas MongoDB
 - <https://www.mongodb.com/cloud/atlas>
- DBHub.io
 - <https://dbhub.io/>
 - Criar database usando arquivo:
 - <https://www.sqlitetutorial.net/wp-content/uploads/2018/03/chinook.zip>
 - <https://github.com/fscheidt/iotdb-21/raw/master/dataset/chinook.zip>

Teste do DBHub

Após importação do dataset chinook, escreva as queries SQL para obter os seguintes resultados:

- Quantidade total de artistas
- Listar os artistas que começam com a letra "F"
- As 10 músicas (tracks) mais longas em milliseconds
- As músicas cujo genero é Comedy
- O total de música por genero

Testar

- MySQL
 - Criar database
 - Importar base customersdb.sql
- MongoDB
 - Logar e testar datasets