

Bancos de Dados NO-SQL



PROF. ANTONIO GUARDADO

Agenda

- 1 – Conceitos e definições
- 2 – Características
- 3 – Modelos de dados
- 4 -

1 - O que significa?

- É uma denominação para bancos de dados **não-relacionais**.
- Isso não quer dizer que seus modelos não possuem relacionamentos.
- E sim, que não são orientados a tabelas.
- Not Only SQL. (**Não Apenas SQL**)
- A tecnologia NoSQL foi iniciada por companhias líderes da Internet : Google, Facebook, Amazon e LinkedIn - para superar as limitações (45 anos de uso da tecnologia) de banco de dados relacional para aplicações web modernas (2009)

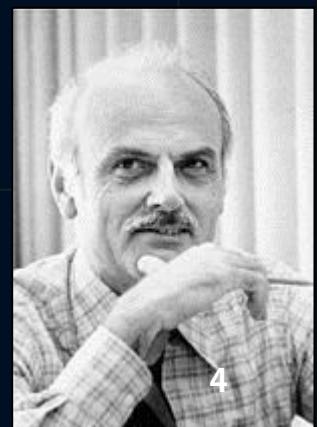
1.1 - Banco de Dados Relacional

- Dados são estruturados de acordo com o modelo relacional
- Padrão para a grande maioria dos SGBDs
SQL Server, Oracle, PostgreSQL, MySQL, DB2, etc.
- Elementos básicos
Relações (tabelas) e registros (tuplas)
- Características fundamentais
Restrições de integridade (PK, FK, UK, CK, NN)
Normalização
Linguagem SQL (*Structured Query Language*)

Edgar F. Codd

*August 23, 1923 +April 18, 2003

Codd, E.F. (1970). "A Relational Model of Data for Large Shared Data Banks". Communications of the ACM 13 (6): 377–387. doi:10.1145/362384.362685.



1.2- Porque NoSQL ?

- ❑ Hoje as empresas estão adotando NoSQL para um número crescente de casos de uso.
- ❑ A escolha que é impulsionada por quatro megatendências inter-relacionadas :
 - ❑ Big Users
 - ❑ Big Data
 - ❑ Internet das coisas
 - ❑ Cloud Computing

1.2.1 - Big Users



3
Billion
Global Online
Population



35
Billion Hrs./Mo.
Spent Online



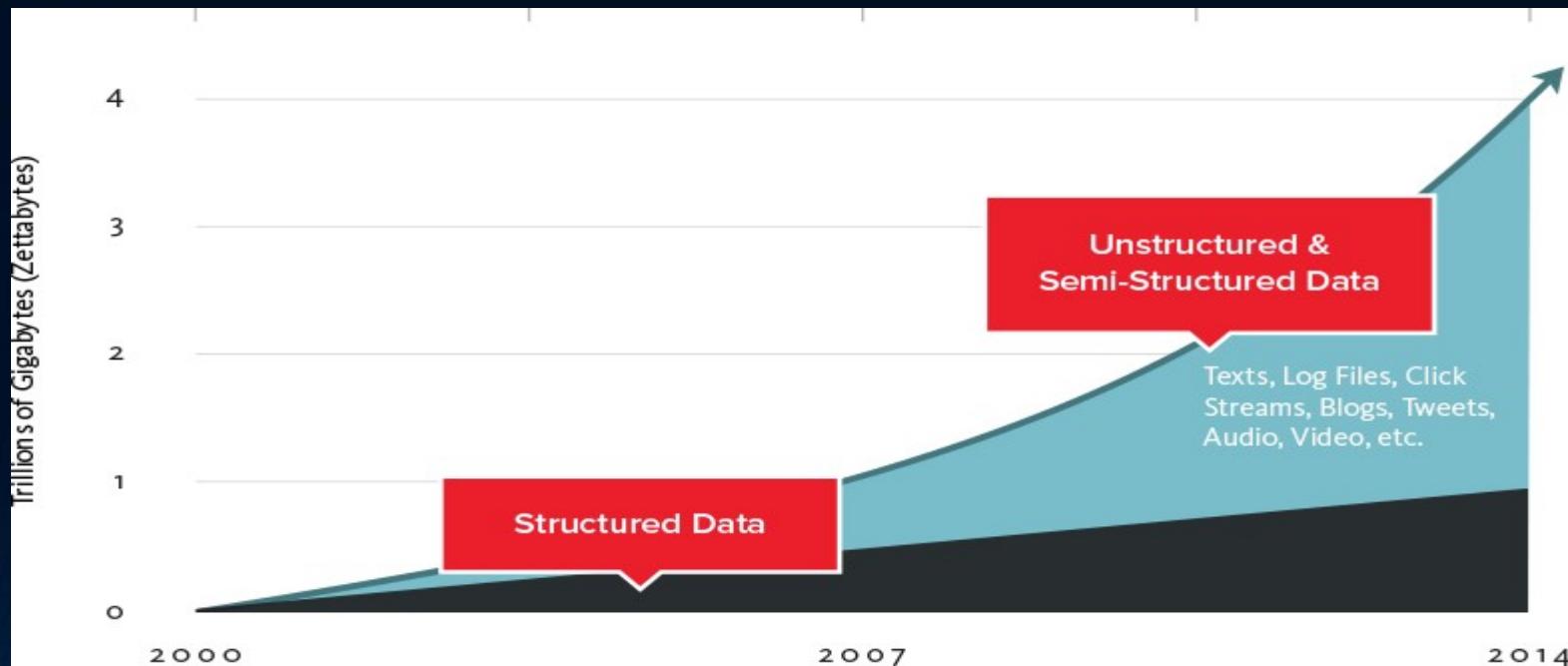
1.75
Billion
Smartphone
Users

O crescente uso de aplicativos online resultou em um número crescente de operações de banco de dados e uma necessidade de uma maneira mais fácil de **escalar** bancos de dados para atender a essas demandas.

Um grande número de usuários, combinados com a natureza dinâmica dos padrões de uso está demandando uma tecnologia de banco de dados mais facilmente **escalável**.

NoSQL é a solução.

1.2.2 - Big Data



É necessário uma solução altamente flexível, que acomode facilmente qualquer novo tipo de dado (não-estruturado e semi-estruturado) e que não seja corrompida por mudanças na estrutura de conteúdo.

NoSQL fornece um modelo de dados **sem esquema** muito mais flexível que mapeia melhor a organização de dados de uma aplicação e simplifica a interação entre a aplicação e o banco de dados, resultando em menos código para escrever, depurar e manter.

1.2.3 - A Internet das Coisas

32 bilhões de coisas vão estar conectadas a internet

10% de todos os dados serão gerados por sistemas embarcados (vs 2% hoje)

21% dos mais valiosos dados serão gerados por sistemas embarcados (vs 8% hoje)

Dados de telemetria - semi- estruturados e contínuos - representam um desafio para bancos de dados relacionais, que exigem um esquema fixo e dados estruturados.

Empresas inovadoras estão utilizando tecnologia NoSQL para dimensionar o acesso simultâneo de dados para milhões de dispositivos e sistemas conectados, armazenar bilhões de pontos de dados e atender aos requisitos de infra-estrutura e operações de missão crítica de performance.

1.2.4 - Cloud Computing

Atualmente a maioria das novas aplicações são executados em um sistema em nuvem privado, público ou híbrido, suportam um grande número de usuários e usam uma arquitetura de internet de três camadas.

Na camada de banco de dados, bancos de dados relacionais são originalmente a escolha popular.

Seu uso é cada vez mais problemático porque eles são uma tecnologia centralizada, cuja escabilidade é vertical ou invés de horizontal.

Isso não os torna adequado para aplicações que requerem escalabilidade fácil e dinâmica.

Bancos de dados NoSQL são construídos a partir do zero para serem distribuídos, escaláveis dinamicamente e são, portanto, mais adequados a natureza altamente distribuída da arquitetura três camadas da internet.

2 - Principais Características

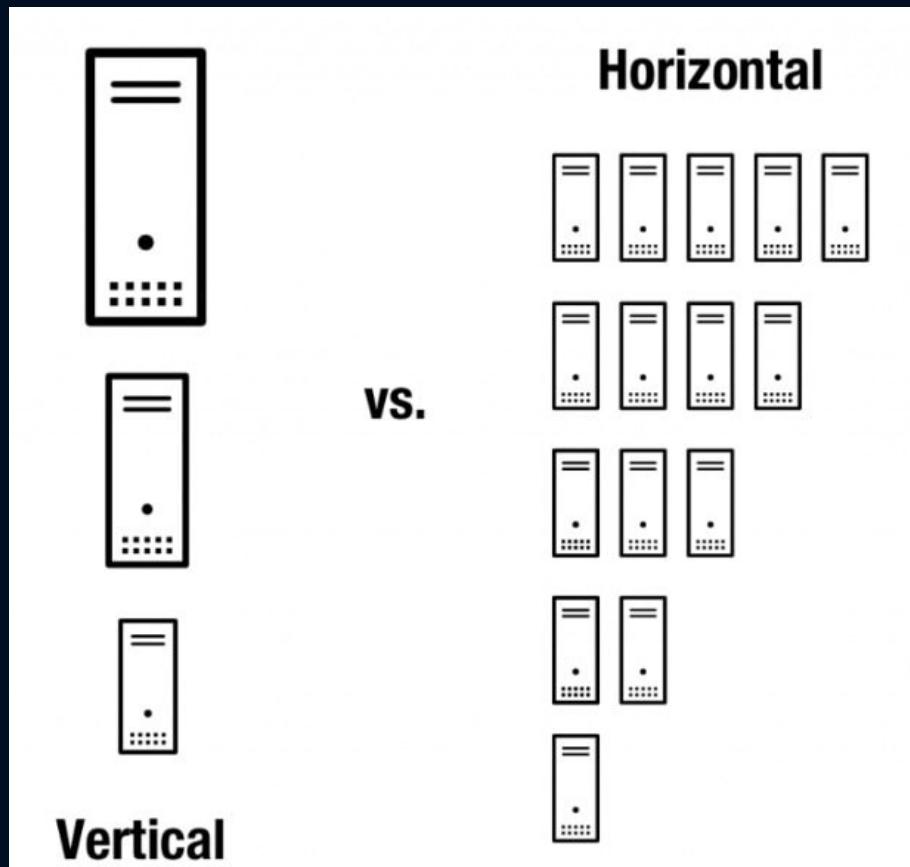
- Escalabilidade Horizontal.
- Ausência de Esquema ou Esquema Flexível.
- Suporte a Replicação.
- API Simples
- Nem Sempre é Consistente.
- Alta disponibilidade (Confiabilidade, recuperabilidade, detecção rápida de erros e operações contínuas)
- Modos de armazenamento

2.1 - Escalabilidade Horizontal

- A escalabilidade Horizontal consiste em aumentar o número de máquinas disponíveis.
- A escalabilidade Horizontal em modelos relacionais seria inviável devido a concorrência.
- Como nos modelos NoSQL não existe bloqueios, esse tipo de escalabilidade é a mais viável.

2.1 – Escalabilidade Vertical x Horizontal

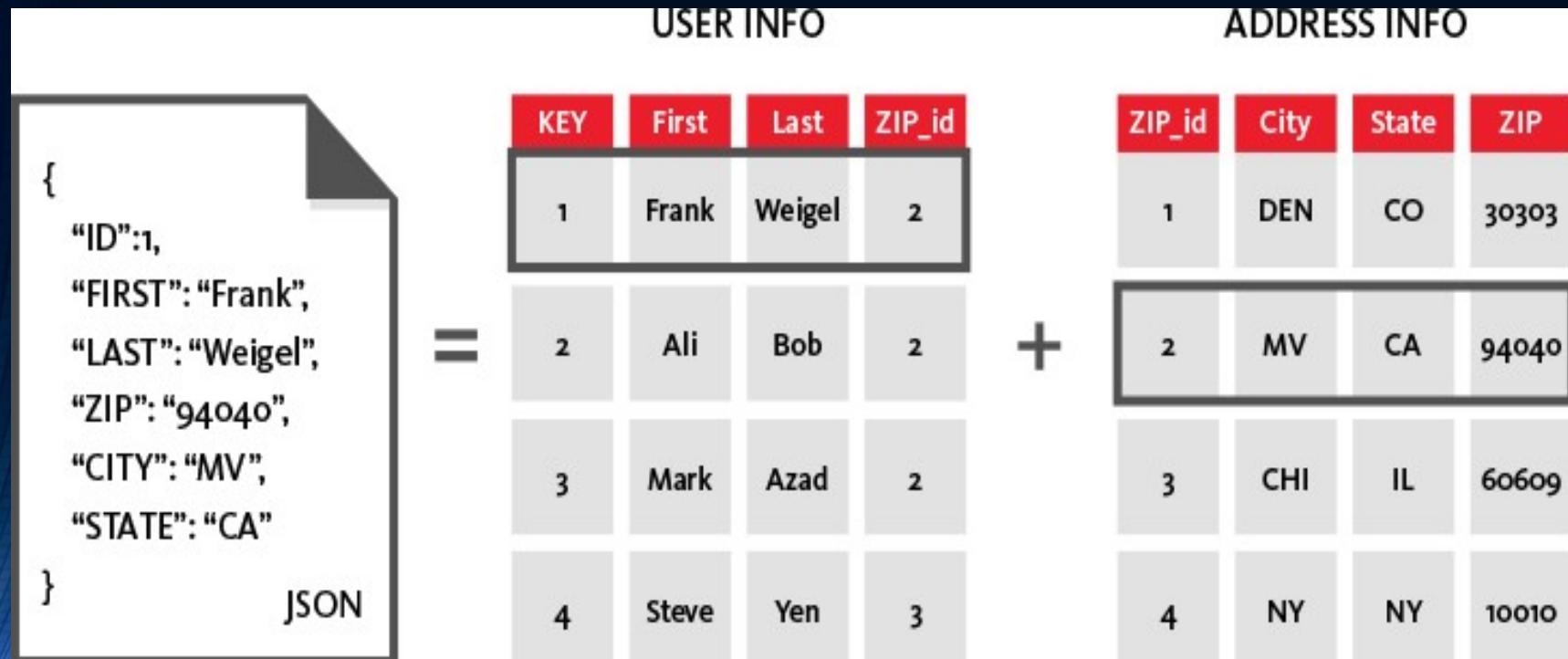
- Vertical :aumentar a capacidade do hardware
-> + memória / + HD / + processador
- Horizontal : mais máquinas com processamento distribuído



2.2 - Ausência de Esquema (Schema Free)

- Apresentam ausência de Esquema ou esquema flexível, isso permite uma fácil aplicação da escalabilidade e também um aumento na disponibilidade dos dados
- Mas também devido a essa ausência, não há garantia da integridade dos dados.

2.2 - Modelo de dados mais flexível



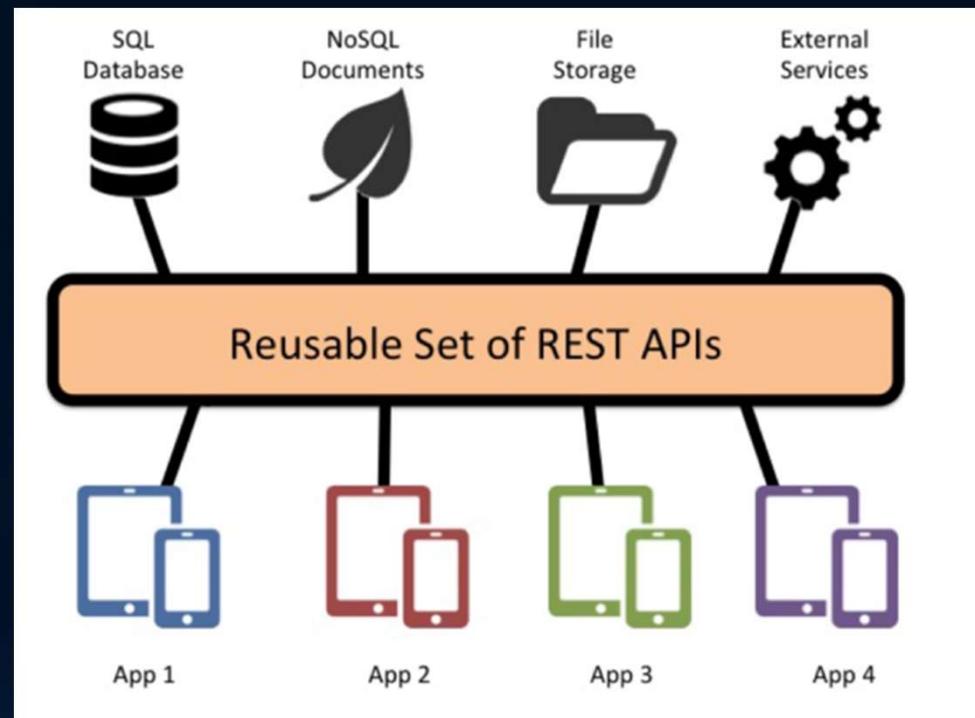
2.3 - Suporte a replicação

- Permitem a replicação de uma forma nativa o que provém uma escalabilidade maior e também uma diminuição do tempo gasto para a recuperação de informações



2.4 – APIs simples

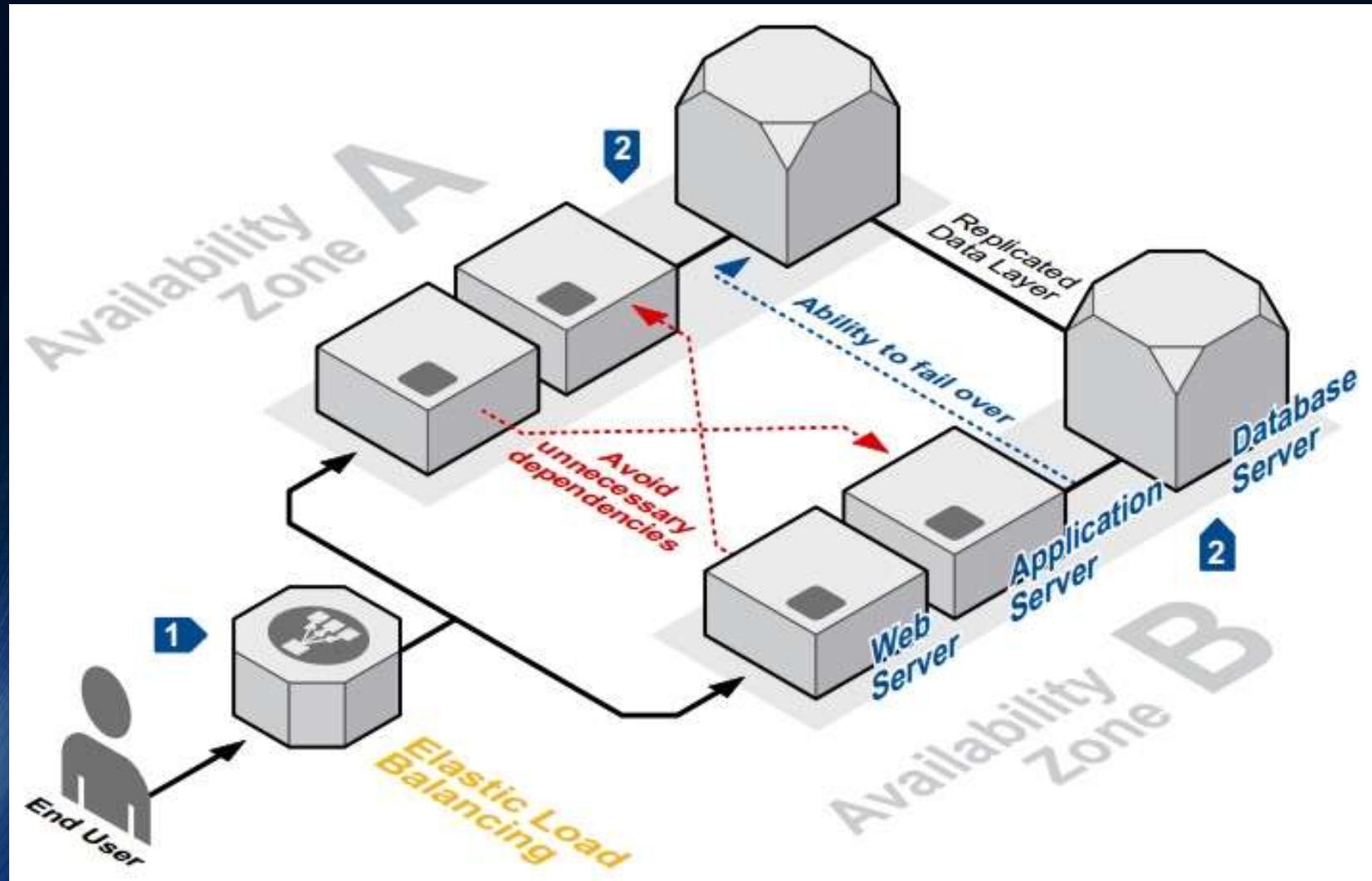
- Para que o acesso às informações seja feito da forma mais rápida possível APIs são desenvolvidas para que qualquer aplicação possa ter acesso aos dados do SGBD.



2.5 - Nem sempre consistente

- Os bancos de dados NoSQL nem sempre conseguem se manter consistentes, ou seja, o valor do dado lido nem sempre é o mais atual.

2.6 – Alta disponibilidade



2.7 - Modo de Armazenamento de Dados

□ Temos os sistemas que...

-mantêm suas informações em memória realizando persistências ocasionais

Scalaris, Redis

-mantêm suas informações em disco

CouchDB, MongoDB, Riak, Voldemort

-são configuráveis

BigTable, Cassandra, Hbase, HyperTable

3 - Modelo de Dados

☐ Existem quatro categorias:

- Sistemas baseados em armazenamento chave-valor
- Sistemas orientados a documentos
- Sistemas orientados à coluna
- Sistemas baseados em grafos

3.1 - Modelo de Dados – Chave-Valor

KEY VALUE

COLUMN

GRAPH

DOCUMENT

- ❑ Coleção de chaves únicas associada a um valor, que pode ser de qualquer tipo (binário, string)

Exemplo:

Key: 123435 Value: Joao da Silva

Key: 334545 Value: Name = Fernando, age = 29

	Key	Value	
	123435	Joao da Silva	
	334545	Name=Fernando, age=29	

3.2 - Modelo de Dados – Orientado a Colunas

KEY VALUE

COLUMN

GRAPH

DOCUMENT

- Famílias de colunas (um repositório para colunas, análogo a uma tabela do Modelo Relacional) e super-colunas (compostas por arrays de colunas)
- o benefício de armazenar dados em colunas, é a busca /acesso rápido e a agregação de dados.

3.2 - Modelo de Dados – Orientado a Colunas

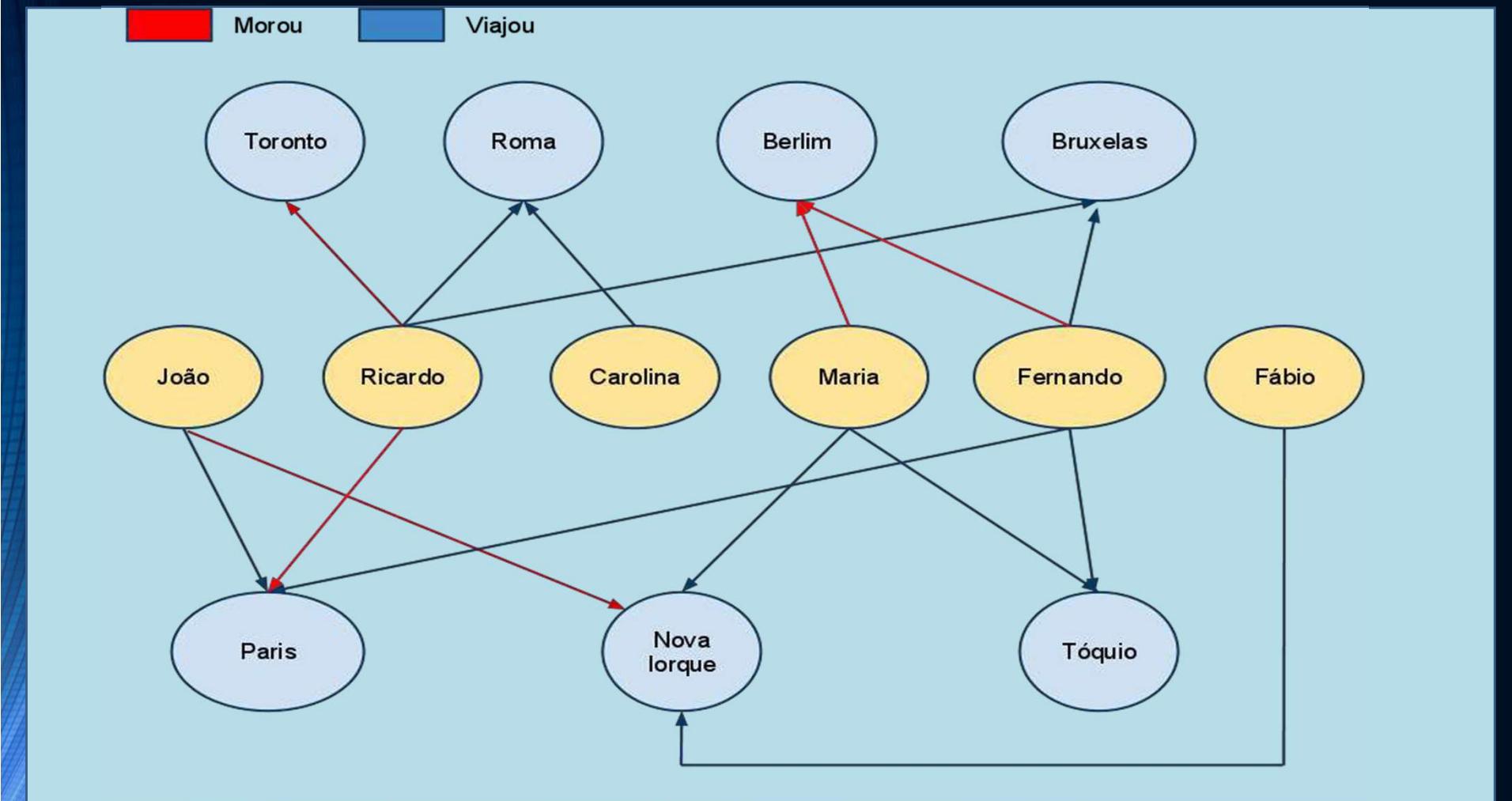
KEY	COLUMN FAMILIES	
ID	CUSTOMERINFO	ADDRESSINFO
1001	FirstName: Tom MiddleName: T LastName: Tester	Address1: 2001 Bayfront Dr. Address2: Suite#813 City: Tampa State: FL Zip: 34637 Country: US
1002	FirstName: Bob MiddleName: B LastName: Builder	Address1: 1234 Sunny Circle City: Beverly Hills State: CA Zip: 90210

3.3 - Modelo de Dados - Grafos

• KEY VALUE COLUMN GRAPH DOCUMENT

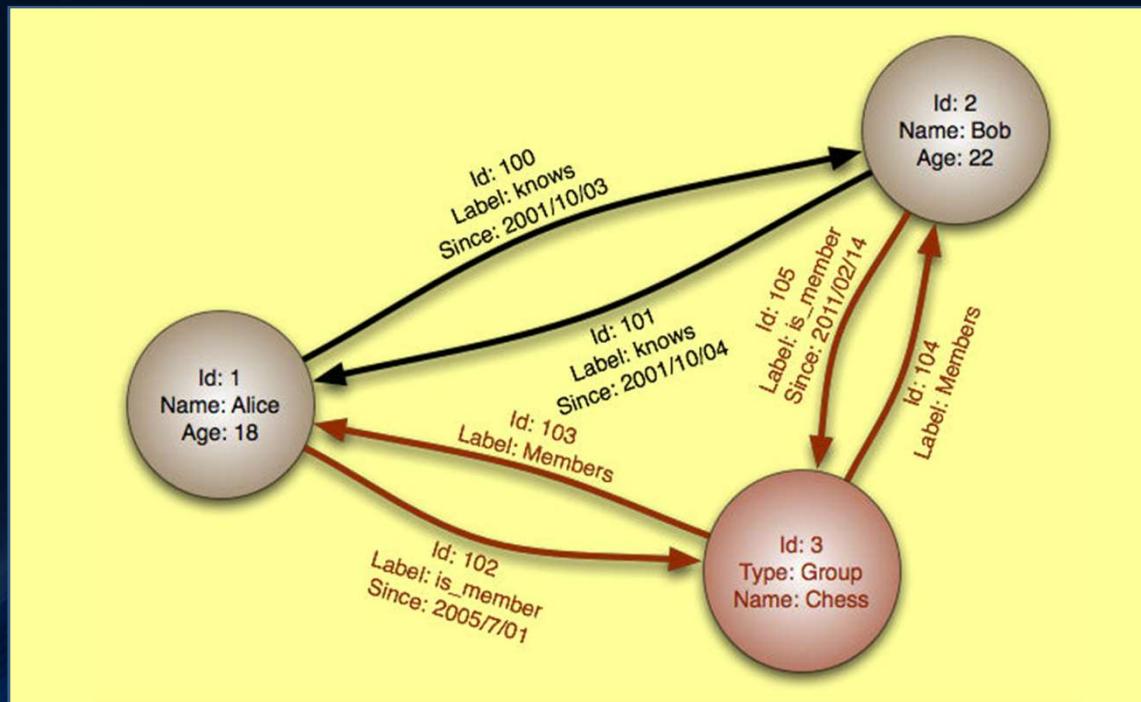
- Banco de dados baseado em grafos, nele temos as entidades chamadas de vértices (ou node) que são ligadas entre elas pelas arestas (ou relationships) cada um podendo guardar dados entre os relacionamentos e cada relacionamento pode ter uma direção.

3.3 - Modelo de Dados - Grafos



3.3 - Modelo de Dados - Grafos

- Exemplo:
- - Vértice: Chave->Valor representa entidade. Nome: Alice
- - Aresta: relacionamentos
- Ex: Vértice Alice conhece o Vértice Bob desde 2001



3.4 - Modelo de Dados -Documentos

KEY VALUE COLUMN GRAPH DOCUMENT

- Os documentos são as unidades básicas de armazenamento e estes não utilizam qualquer tipo de estruturação pré-definida
- São baseados em JSON (JavaScript Object Notation)

Exemplo:

```
{"user":{  
    "id": "123",  
    "name": "Emmanuel",  
    "addresses": [  
        {"city": "Paris"},  
        {"city": "Sao Paulo"}]
```

3.4 - Modelo de Dados -Documentos

KEY VALUE COLUMN GRAPH DOCUMENT

Document 1

```
{
  "id": "1",
  "name": "John Smith",
  "isActive": true,
  "dob": "1964-30-08"
}
```

Document 2

```
{
  "id": "2",
  "fullName": "Sarah Jones",
  "isActive": false,
  "dob": "2002-02-18"
}
```

Document 3

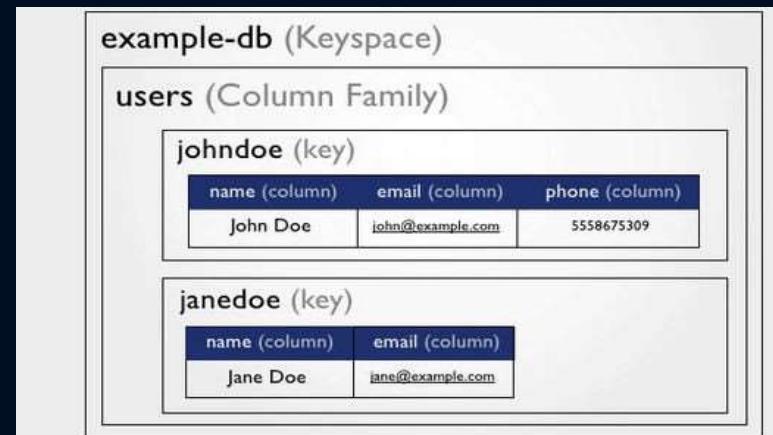
```
{
  "id": "3",
  "fullName": {
    "first": "Adam",
    "last": "Stark"
  },
  "isActive": true,
  "dob": "2015-04-19"
}
```

3.5 - Classificação NoSQL

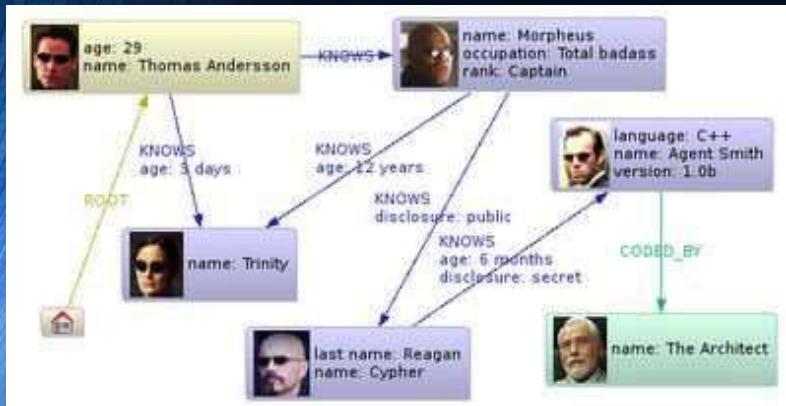
Key - Value

Key	Value
123435	Joao da Silva
334545	Name=Fernando, age=29

Column



Graph



Document



3.6–Consultas em BDs NoSQL

Amazon DynamoDB– Key-Value

SQL Query

```
SELECT _id, name, address ← projection
      FROM users           ← table
      WHERE age > 18        ← select criteria
      LIMIT 5              ← cursor modifier
```

AWS Query

Amazon DynamoDB Explore Table: Reply

List Tables **Browse Items**

Scan Query Go New Item Edit Item Copy to New Delete Item

Index Name: [Table] Reply: Id, ReplyDateTime

Hash Key (*): Id equal to Amazon DynamoDB#DynamoDB Thre

Range Key: ReplyDateTime greater than 2014-01-12

Sort: Ascending Descending

3.6–Consultas em BDs NoSQL

MongoDB - Document

SQL Query

```
SELECT _id, name, address ← projection
      FROM users           ← table
      WHERE age > 18        ← select criteria
      LIMIT 5              ← cursor modifier
```

Operation Find

```
db.users.find(
  { age: { $gt: 18 } },
  { name: 1, address: 1 }
).limit(5)                                ← collection
                                            ← query criteria
                                            ← projection
                                            ← cursor modifier
```

3.6–Consultas em BDs NoSQL

Neo4j- Graph

SQL Query

```
SELECT _id, name, address ← projection
FROM   users           ← table
WHERE  age > 18          ← select criteria
LIMIT  5                ← cursor modifier
```

Cyber query

```
MATCH a
WHERE a.age>18
RETURN a.id, a.name, a.address
LIMIT 5
```

3.6–Consultas em BDs NoSQL

Cassandra - Column

SQL Query

```
SELECT _id, name, address ← projection
      FROM users           ← table
      WHERE age > 18        ← select criteria
      LIMIT 5              ← cursor modifier
```

Comandos CRUD

(Create, Read, Update, Delete)

CQL – Cassandra Query Language

são iguais

```
SELECT _id, name, address ← projection
      FROM users           ← table
      WHERE age > 18        ← select criteria
      LIMIT 5              ← cursor modifier
```

3.7 - Quais linguagens suportam NoSQL?

	Amazon Dynamo	Neo4j	Cassandra	MongoDB
C				X
C#				X
C++			X	X
Go			X	X
Java	X	X	X	X
Javascript	X			X
Node.js	X	X	X	X
Perl			X	X
PHP	X	X	X	X
Python		X	X	X
Ruby	X	X	X	X
Scala		X	X	X

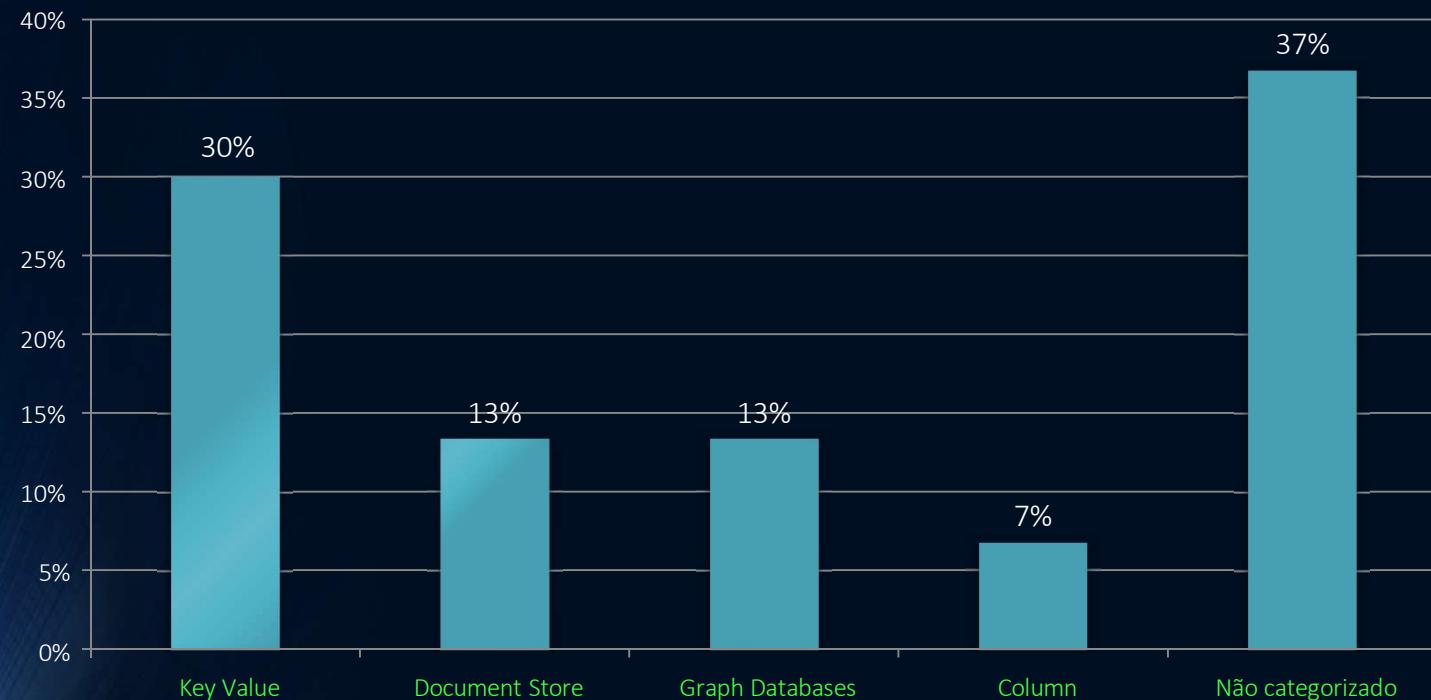
3.8 - DB Ranking

342 systems in ranking, April 2018

Rank			DBMS	Database Model	Score		
Apr 2018	Mar 2018	Apr 2017			Apr 2018	Mar 2018	Apr 2017
1.	1.	1.	Oracle 	Relational DBMS	1289.79	+0.18	-112.21
2.	2.	2.	MySQL 	Relational DBMS	1226.40	-2.46	-138.22
3.	3.	3.	Microsoft SQL Server 	Relational DBMS	1095.51	-9.28	-109.26
4.	4.	4.	PostgreSQL 	Relational DBMS	395.47	-3.88	+33.69
5.	5.	5.	MongoDB 	Document store	341.41	+0.89	+15.98
6.	6.	6.	DB2 	Relational DBMS	188.95	+2.28	+2.29
7.	7.	7.	Microsoft Access	Relational DBMS	132.22	+0.27	+4.04
8.	↑ 9.	↑ 11.	Elasticsearch 	Search engine	131.36	+2.81	+25.69
9.	↓ 8.	9.	Redis 	Key-value store	130.11	-1.12	+15.75
10.	10.	↓ 8.	Cassandra 	Wide column store	119.09	-4.40	-7.10
11.	11.	↓ 10.	SQLite 	Relational DBMS	115.99	+1.17	+2.19
12.	12.	12.	Teradata	Relational DBMS	73.68	+1.21	-2.88
13.	13.	↑ 17.	Splunk	Search engine	65.06	-0.61	+9.55
14.	↑ 15.	↑ 18.	MariaDB 	Relational DBMS	64.56	+1.45	+15.83
15.	↓ 14.	↓ 14.	Solr	Search engine	63.21	-1.60	-1.16

<http://db-engines.com/en/ranking>

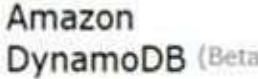
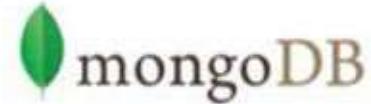
3.9 - Classificação de SGBDs NoSQL



150 tipos de banco de dados NOSQL

- Fonte: <http://nosql-database.org/>
- Dados compilados manualmente

3.9.1- Classificação NoSQL e Produtos

KEY VALUE	COLUMN	GRAPH	DOCUMENT
  	 	  	 
Amazon DynamoDB (Beta) ORACLE BERKELEY DB 11g 	BigTable(column) Google Desenvolvido em: C++ Quem Usa: Gmail Google Maps, YouTube	Neo4j (graph) Desenvolvido em: Java Quem Usa? -WalMart -National Geographic -Ebay	MongoDB (Document) Desenvolvido em: C Quem Usa: -Globo.com _Apontador -Forbes -New York Times
	Cassandra (column) Desenvolvido em: Java Quem Usa? Twitter NetFlix Facebook		

3.10 - Quando e qual utilizar?



Fonte: Martin Fowler

Fonte: <http://www.martinfowler.com/bliki/PolyglotPersistence.html>

4 – Propriedades Transacionais

- Aplicações de grande escala se preocupam com disponibilidade e redundância
- Bancos de dados relacionais tem problemas :
 - em sistemas distribuídos - particionar as informações em várias máquinas, torna a manutenção das tabelas bem difícil e complicado;
 - alto volume de dados com baixa velocidade de resposta - a medida que os dados crescem na base, fica cada vez mais difícil a disponibilização rápida das informações;
 - os dados mudam constantemente - alterar a entidade no mundo relacional em algumas situações não é fácil, dependendo do tipo do dado alguns bancos simplesmente bloqueiam o acesso a tabela para realizar uma simples alteração de coluna;

4 - BASE x ACID

- Propriedades ACID:

- **Atomicidade**: A transação será executada totalmente ou não será executada.
- **Consistência**: Garante que o banco de dados passará de uma forma consistente para outra forma consistente após uma transação.
- **Isolamento**: Garante que uma transação não tem interferência de nenhuma outra transação concorrente, são isoladas.
- **Durabilidade**: Garante que o que foi salvo, não será mais perdido.

4- BASE x ACID

- Propriedades **BASE**:
 - Basically Available – Basicamente Disponível.
 - Soft-State – Estado Leve
 - Eventually Consistent – Eventualmente Consistente.
- Uma aplicação funciona basicamente todo o tempo (Basicamente Disponível), não tem de ser consistente todo o tempo (Estado Leve) e o sistema torna-se consistente no momento devido (Eventualmente Consistente).

4.1 – Comparativo BASE x ACID

ACID	BASE
Consistência forte	Fraca consistência
Isolamento	Foco em Disponibilidade
Concentra-se em "commit"	Melhor esforço
Transações aninhadas	Respostas aproximadas
Disponibilidade	Mais simples e mais rápido
Conservador (pessimista)	Agressivo (otimista)
Evolução difícil (por exemplo, esquema)	Evolução mais fácil

4.2 – Teorema CAP

- Consistência – **Consistency**.
- Disponibilidade – **Availability**.
- Tolerância ao Particionamento - **Partition tolerance**.
- Teorema CAP – é impossível garantir essas três propriedades ao mesmo tempo
- é possível garantir quaisquer duas dessas propriedades ao mesmo tempo

Gilbert, S.; Lynch, N. A. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. SIGACT News, 33(2):51–59, 2002.

4.2.1 - Consistency - Consistência

O sistema garante a leitura do dado mais atualizado, quanto ele foi escrito.

O cliente pode ler o dado no mesmo nó que este dado foi escrito ou de um nó diferente, o mesmo dado será retornado para a aplicação cliente. Mesmo que alguém tenha mudado o estado do domínio da aplicação com novas informações, o comportamento de consistência, garantirá que o cliente não verá dados velhos, apenas os novos dados serão visualizados.

4.2.2- Availability - Disponibilidade

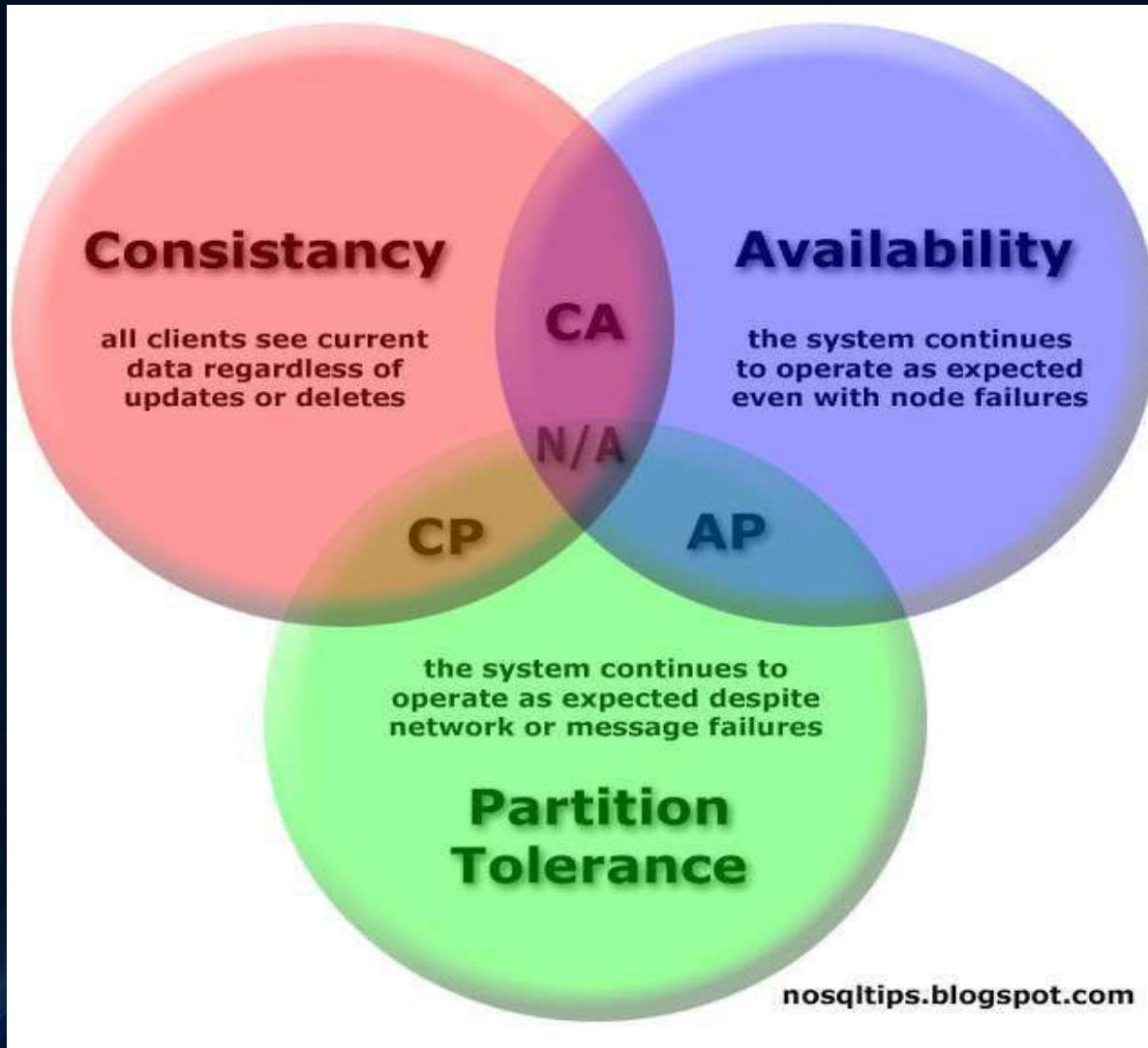
- Quando o cliente lê ou escreve um dados em um dos nós, o nó que está sendo utilizado pelo cliente, pode estar indisponível.
- Isso não quer dizer que um nó não pode falhar, o que esse comportamento quer dizer é que, um nó pode sim ficar offline/cair/falhar, mas o sistema/aplicação continuaria disponível.
- Se um sistema é capaz de obter acesso de leitura/escrita em um nó que não possui falhas e este responde em um tempo razoável, temos aqui a garantia de disponibilidade.
- Mas pode trazer um dado não atual

4.2.3- Partition Tolerance – Tolerância a falhas

O sistema continua operando, mesmo que aconteça alguma falha na rede (quebra de conectividade).

Por exemplo: Temos um banco de dados distribuído em alguns nós da rede (Ex.: 10 NÓS) e, estes nós, estão ligados através de um cluster. Caso uma das partes do cluster falhe, o sistema continuaria operando sem nenhuma falha, porque a outra parte, que se encontra operacional do cluster, garantirá que, pelo menos, as operações de leitura, continuem funcionando normalmente.

4.2 - CAP



4.3 – Combinações CAP

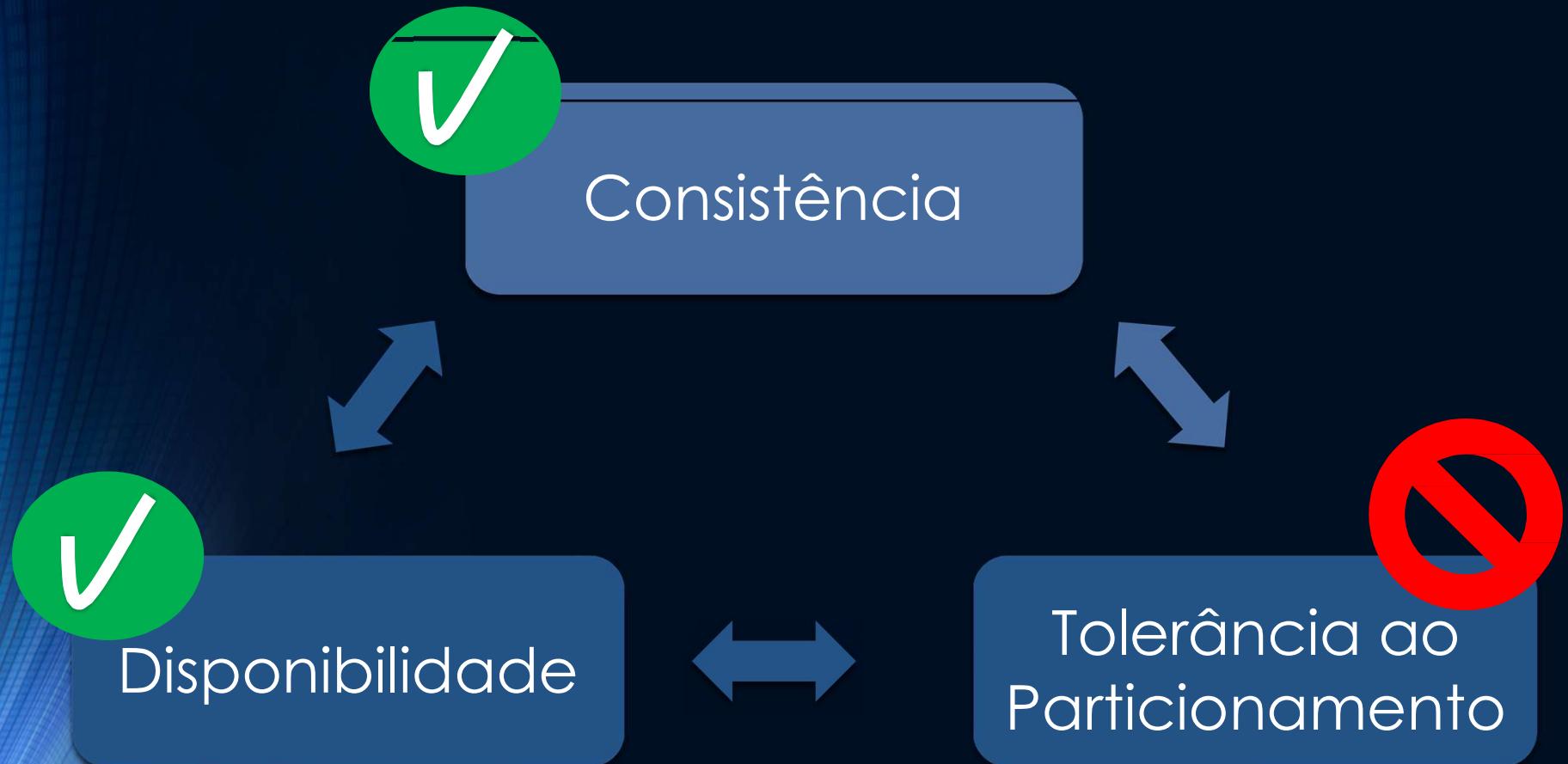
- Dependendo da aplicação escolher entre consistência forte, alta disponibilidade e tolerância ao particionamento
- **Sistemas CA (Consistência e Disponibilidade)**
- **Sistemas CP (Consistência e Particionamento)**
- **Sistemas AP (Disponibilidade e Particionamento)**

4.3.1 - Sistemas CA

- Os sistemas com consistência forte e alta disponibilidade não sabem lidar com a possível falha de uma partição.
- Caso ocorra, sistema inteiro pode ficar indisponível até o membro do cluster voltar.



4.3 - Sistemas CA

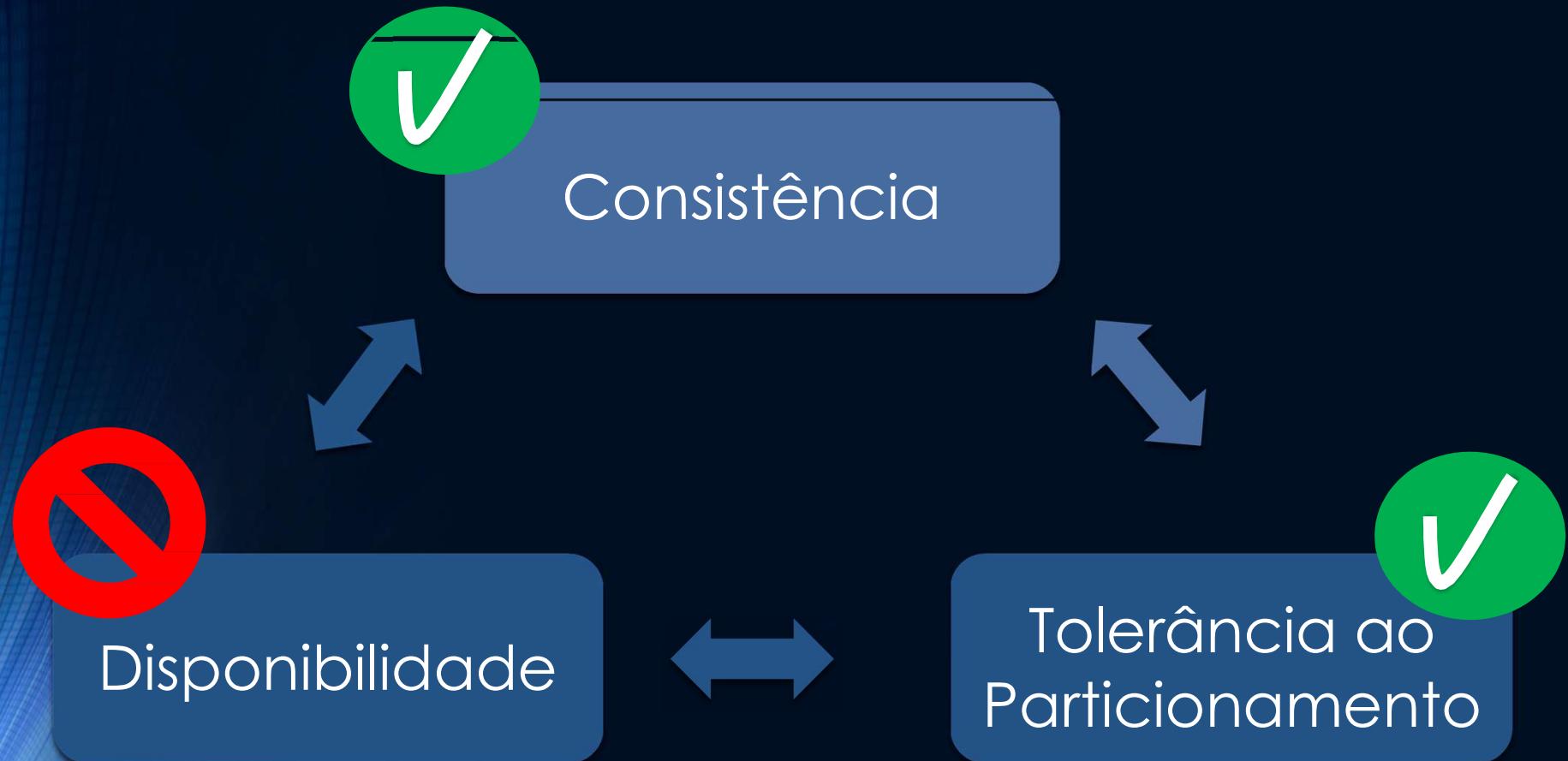


4.3.2 - Sistemas CP

- Para sistemas que precisam da consistência forte e tolerância a particionamento é necessário abrir a mão da disponibilidade (um pouco).
- Exemplos são BigTable, HBase ou
- MongoDB entre vários outros.

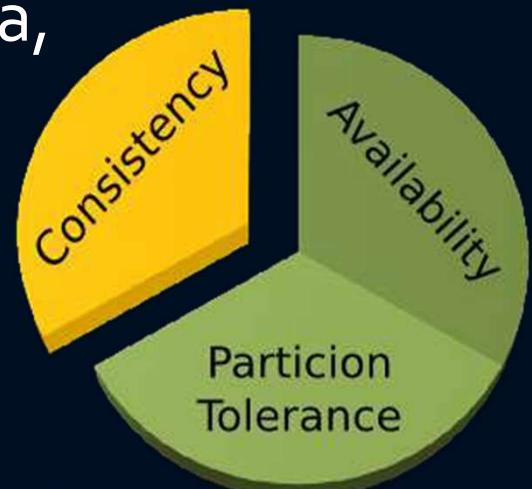


4.3.2 - Sistemas CP



4.3.3 - Sistemas AP

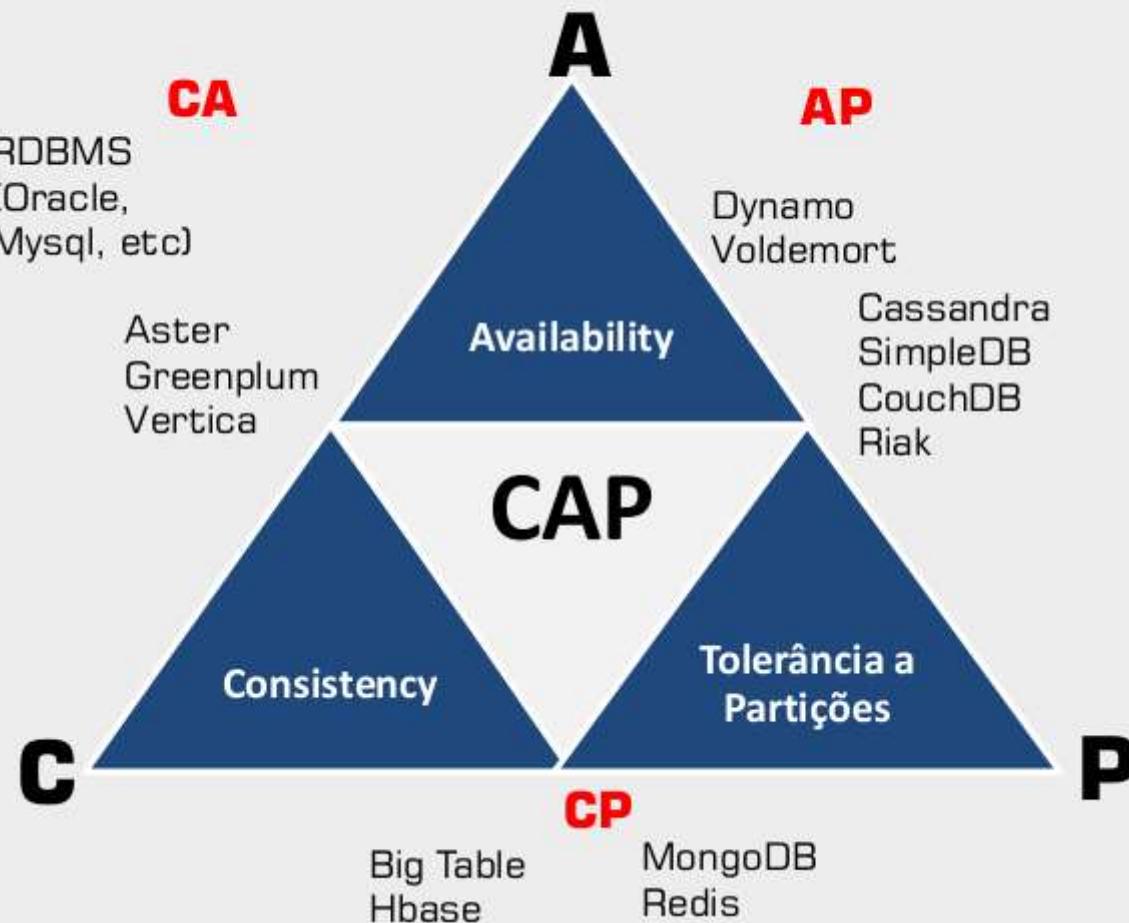
- Há sistemas que jamais podem ficar offline, portanto não desejam sacrificar a disponibilidade. Para ter alta disponibilidade mesmo com um tolerância a particionamento é preciso prejudicar a consistência.
- Exemplos de Bancos são: Cassandra,
- MongoDB, Voldemort



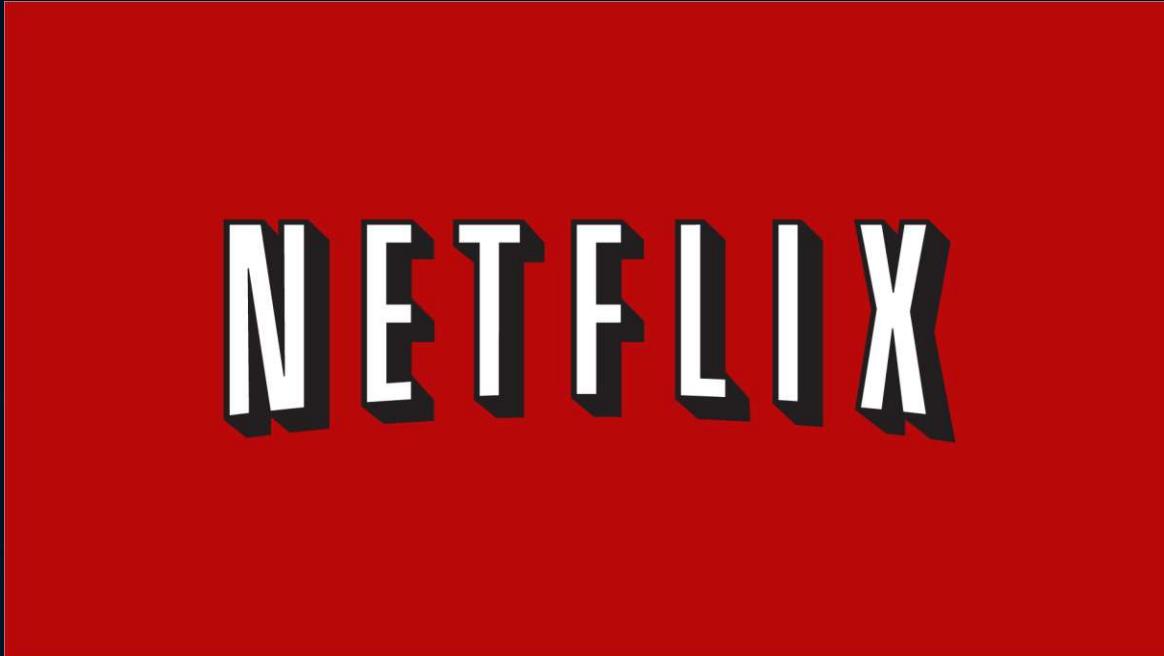
4.3.3 - Sistemas AP



Somente é possível obter 2 delas.



5 - Cases



SGBD:

sistema de processamento de faturas mensais

NOSQL:

Sistema focado em recomendações de melhores filmes.

5 - Cases



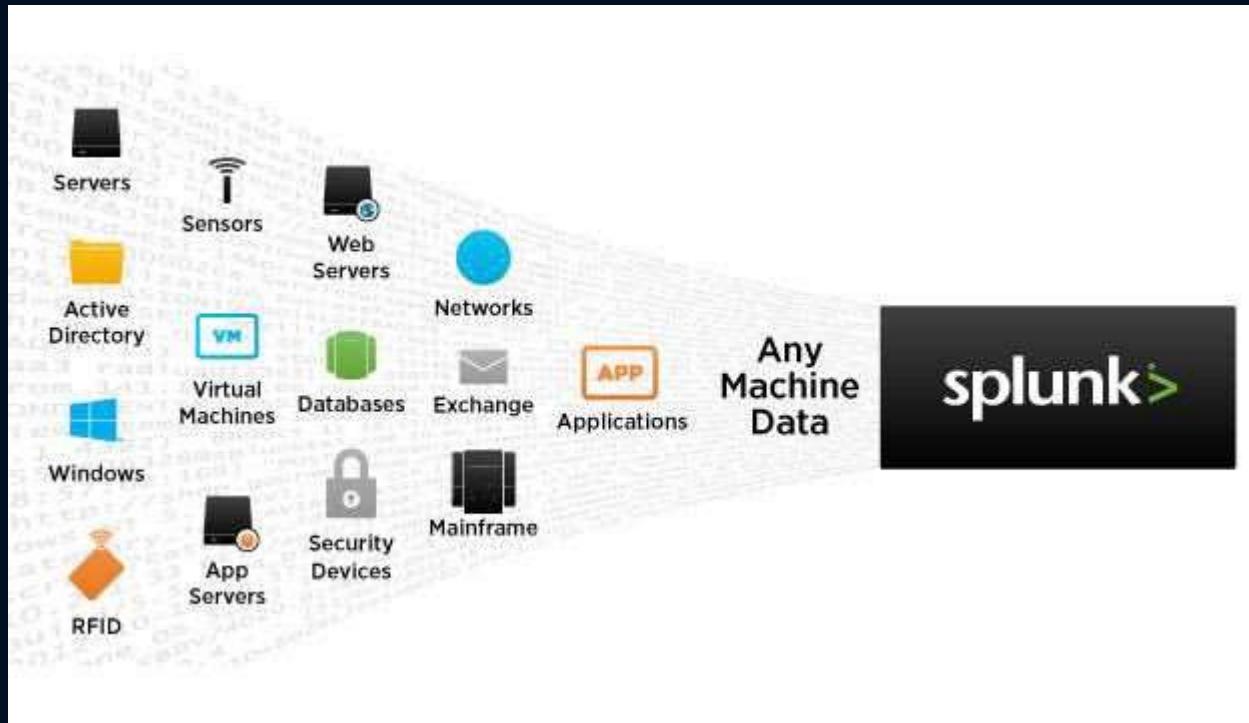
SGBD:

Sistemas de processamento de ordem de venda

NOSQL:

Sistema de pesquisa, recomendações e adaptações de preços em tempo real

5 - Cases



Plataforma para inteligência operacional

SGBD:

Dados de clientes, produtos e RH

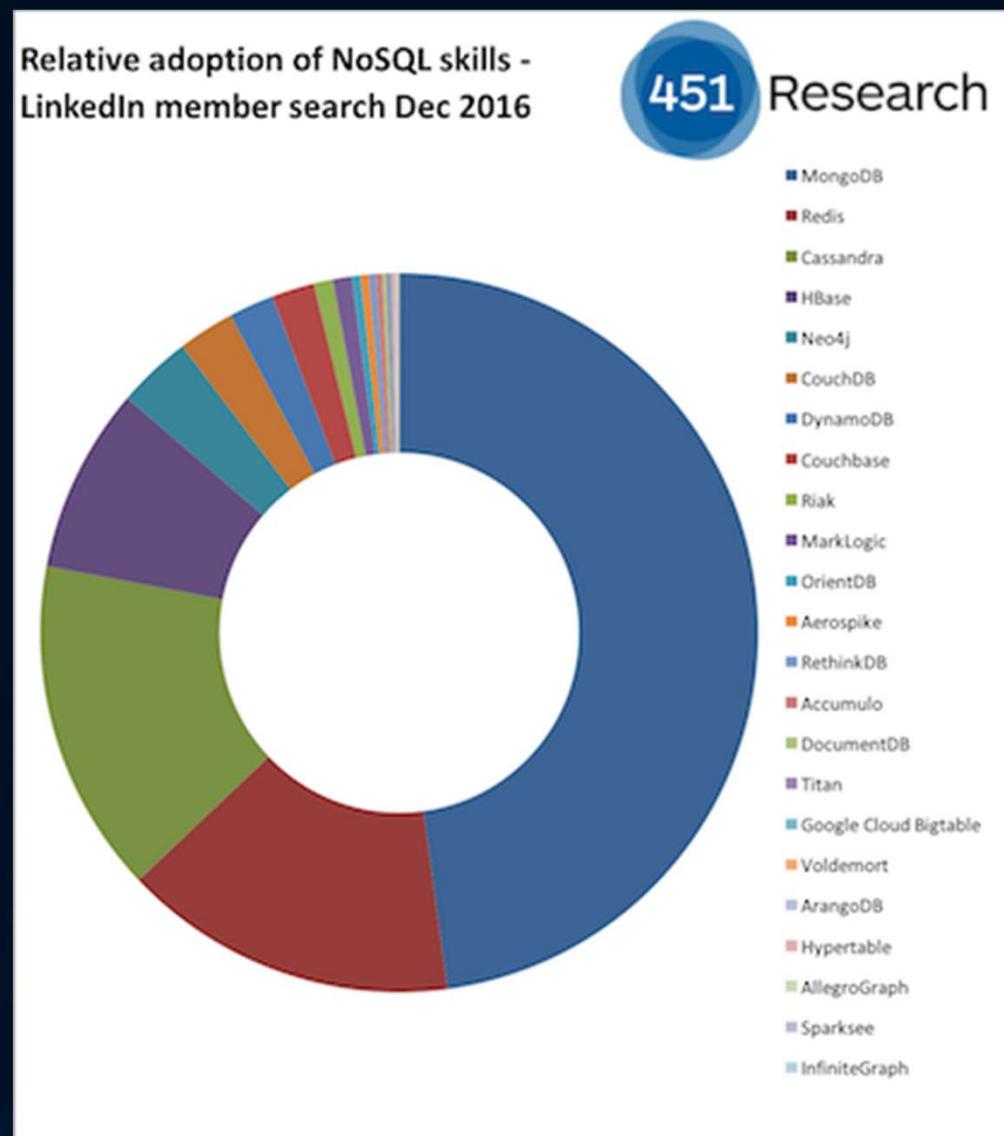
NOSQL:

Explorar, analisar e virtualização de dados

6 - Oportunidades no mercado



6.1 - Profissionais no mercado



https://blogs.the451group.com/information_management/?s=NoSQL+LinkedIn+Skills