



Sistemas de Gerência de Banco de Dados

Bancos de Dados NoSQL

A Motivação...

- Bancos de Dados NoSQL (**N**ot **O**nly **SQL**)
- Dados gerados, armazenados e processados atingiu escalas inéditas com a Web 2.0;
- Quantidade de dados diários em vários domínios de aplicação, tais como: redes sociais, redes de sensores, dados urbanos, etc.;
- Grandes volumes de dados (Big Data).

A Motivação...

1 The accelerating pace of change ...



2 ... and exponential growth in computing power ...

Computer technology, shown here climbing dramatically by powers of 10, is now progressing more each hour than it did in its entire first 90 years

COMPUTER RANKINGS

By calculations per second per \$1,000



Analytical engine
Never fully built, Charles Babbage's invention was designed to solve computational and logical problems



Colossus
The electronic computer, with 1,500 vacuum tubes, helped the British crack German codes during WW II



UNIVAC I
The first commercially marketed computer, used to tabulate the U.S. Census, occupied 943 cu. ft.



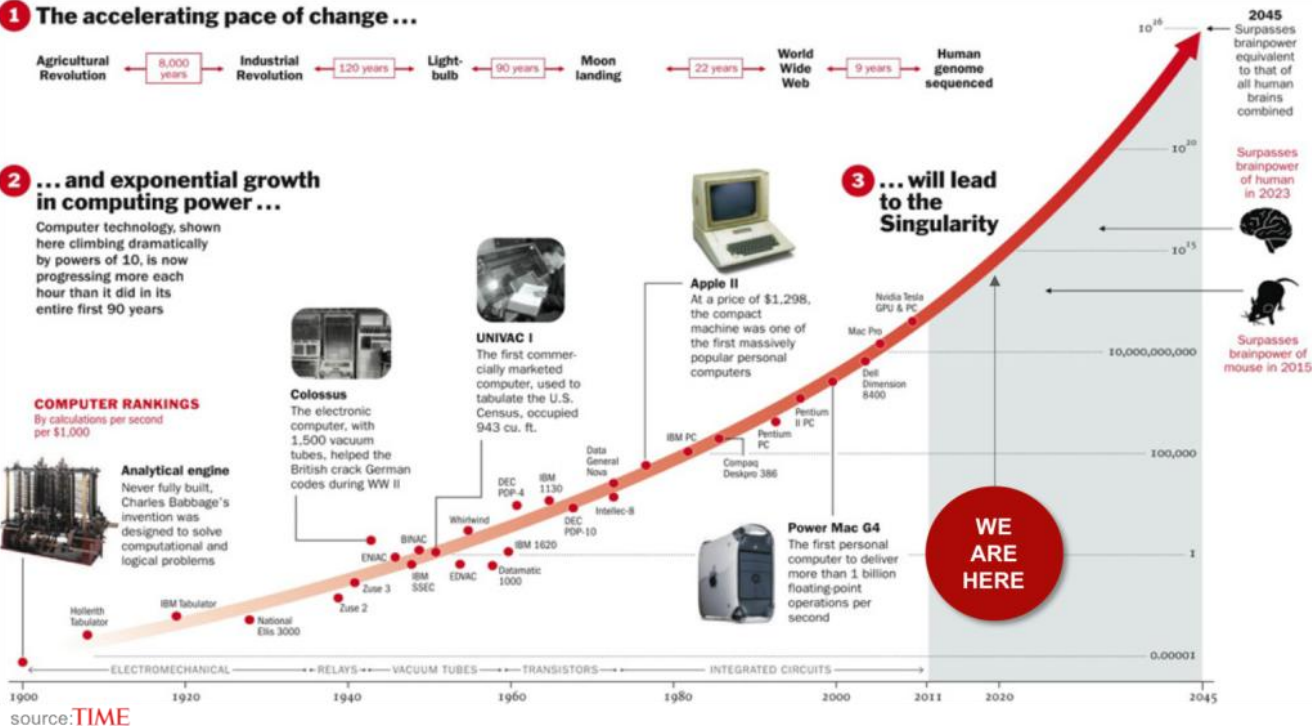
Apple II
At a price of \$1,298, the compact machine was one of the first massively popular personal computers



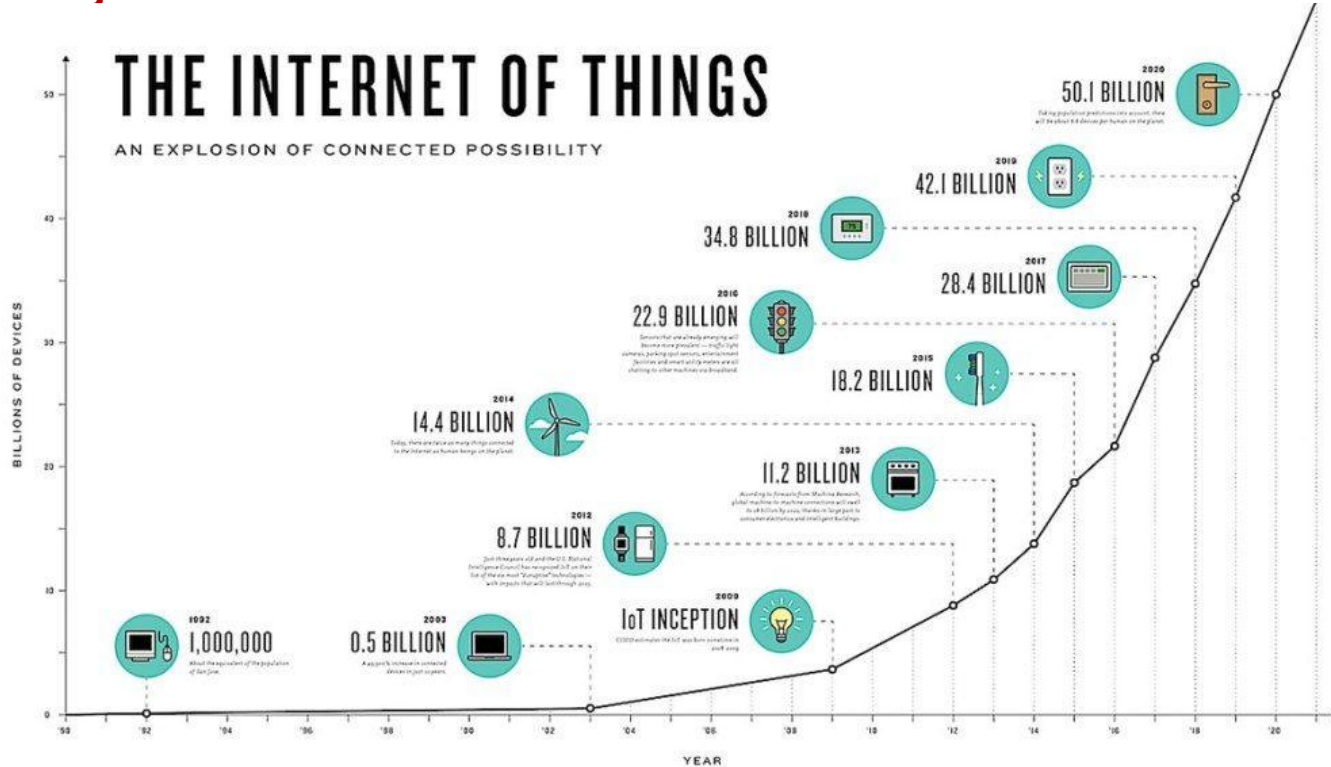
Power Mac G4
The first personal computer to deliver more than 1 billion floating-point operations per second

3 ... will lead to the Singularity

WE ARE HERE



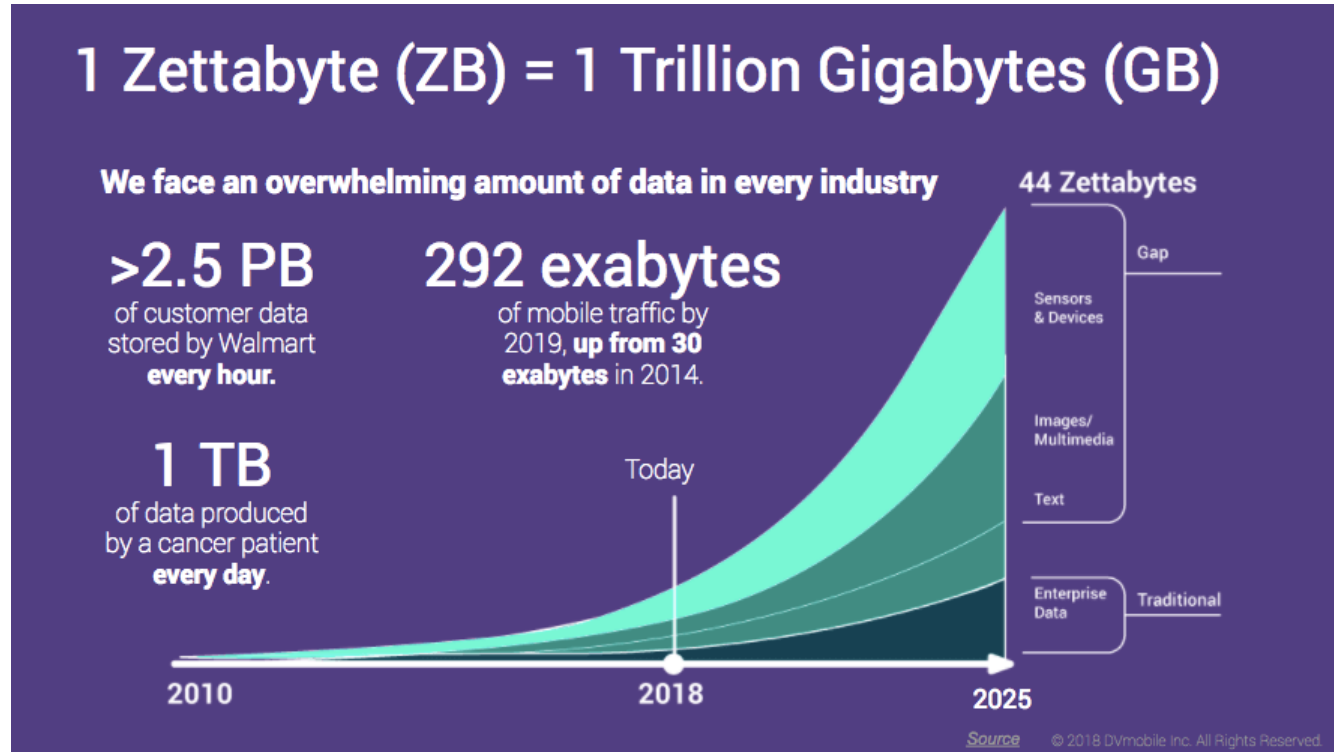
A Motivação...



Fonte: <https://www.gregverdino.com/everything-is-exponential/>

Sistemas de Gerência de Banco de Dados

A Motivação...



A Motivação...

Name	Symbol	Binary	Decimal
byte	B	$2^0=1$ byte	$10^0=1$
kilobyte	KB	$2^{10}=1.024$ byte (B)	$10^3=1.000$
megabyte	MB	$2^{20}=1.048.576$ B	$10^6=1.000.000$
gigabyte	GB	$2^{30}=1.073.741.824$ B	$10^9=1.000.000.000$
terabyte	TB	$2^{40}=1.099.511.627.776$ B	$10^{12}=1.000.000.000.000$
petabyte	PB	$2^{50}=1.125.899.906.842.624$ B	$10^{15}=1.000.000.000.000.000$
exabyte	EB	$2^{60}=1.152.921.504.606.846.976$ B	$10^{18}=1.000.000.000.000.000.000$
zettabyte	ZB	$2^{70}=1.180.591.620.717.411.303.424$ B	$10^{21}=1.000.000.000.000.000.000.000$
yottabyte	YB	$2^{80}=1.208.925.819.614.629.174.706.176$ B	$10^{24}=1.000.000.000.000.000.000.000.000$
brontobyte	BB	$2^{90}=1.237.940.039.285.380.274.899.124.224$ B	$10^{27}=1.000.000.000.000.000.000.000.000.000$
geopbyte	GeB	$2^{100}=1.267.650.600.228.229.401.496.703.205.376$ B	$10^{30}=1.000.000.000.000.000.000.000.000.000.000$

A Motivação...

- *Big Data é o termo designado para grandes volumes de dados (muitas vezes complexos) em que os algoritmos tradicionais não atendam às necessidades de análise.*
- **Volume:** quantidade de dados geradas em instantes de tempo.
 - Ex.: twitter, vídeos na internet, etc.;
- **Velocidade:** a cada “x” milissegundos, seja uma mensagem no twitter ou facebook, seja uma transação de cartão, entre outros;
- **Variedade:** cerca de 85% dos dados no mundo não seguem um padrão estruturado.
 - Ex.: fotos, vídeos, som, entre outros;
- **Veracidade:** esperam-se dados verdadeiros, ou seja, não é possível controlar uma mensagem no twitter com “hashtag” e esta ser falsa, por isso, as análises e estatísticas são fundamentais para transformá-la numa mensagem verdadeira;
- **Valor:** não adianta uma massa de dados enorme por segundo se esta não traz valor. A palavra aqui é “agregar valor”.

A Motivação...

SGBDR	Gerenciamento Big Data
Grande volume de dados	Muito além da capacidade de um SGBDR
Esquema fixo	Ausência ou esquema simples/flexível
Escalabilidade limitada	Alta escalabilidade
Consistência ACID	Consistência eventual (foco em disponibilidade)
Métodos de acesso com suporte ao processamento de consultas complexas (otimização e processamento de junções)	Métodos de acesso simples (foco em escalabilidade e disponibilidade)

Como armazenar estes dados?

A Motivação...

- **Natureza de dados:** os dados Web não são estruturados;
- **Importância do paralelismo no processamento de grande volume de dados:** uso de hardware comum e barato para operação dos sistemas; e olha que hoje tem-se o conceito de **Computação Voluntária**;
- **Distribuição do sistema em escala global:** o software deve ser robusto o suficiente para tolerar constantes e imprevisíveis falhas de hardware e de infra-estrutura.

No SQL - História

- Banco de Dados multi-valor na TRW em **1965**;
- DBM na AT&T em **1979**;
- Lotus Domino em **1989**;
- Carlo Strozzi usou o termo NoSQL em **1998** para um banco de dados que não usava o SQL;
- Banco de dados Neo4j (orientado a grafo) iniciou em **2000**;
- BigTable da Google iniciou em **2004**. Paper publicado em 2006;
- CouchDB iniciou em **2005**;
- Paper sobre o Amazon Dynamo em **2007**;
- MongoDB iniciou em **2007** como parte de uma solução Cloud;
- Projeto Cassandra pelo Facebook em **2008**;
- Project Voldemort (LinkedIn) iniciou em **2008**;
- O termo NoSQL foi introduzido novamente em **2009**;
- **Dias atuais**... diversos tipos de NoSQL foram introduzidos...

No SQL – O que é?

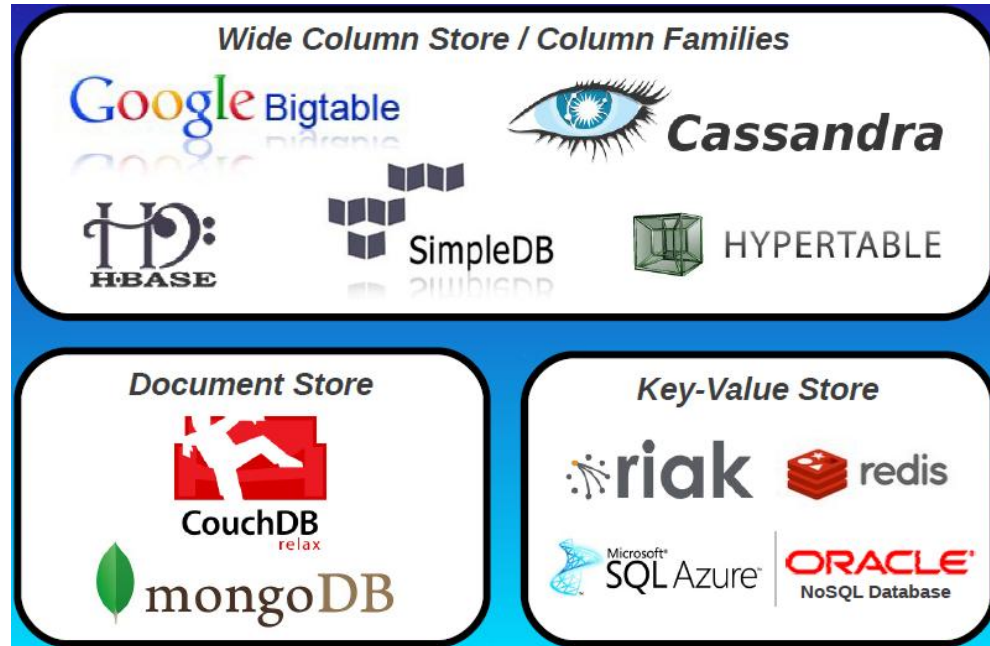
- “Próxima geração” de banco de dados que aborda principalmente alguns pontos:
 - ser não relacional,
 - distribuído,
 - código aberto e
 - escalável horizontalmente.
- A intenção original tem sido os bancos de dados web-escaláveis, além disso, algumas características se aplicam:
 - livre de esquema,
 - facilidade com replicação,
 - API simples e
 - eventualmente consistente.

Fonte: <http://nosql-database.org/>

No SQL não é NewSQL

- Não confundir com NewSQL;
- NewSQL é um movimento para SGBDRs escaláveis e que visam o tratamento de Big Data;
- No NewSQL há suporte para transações e propriedades ACID; não são necessariamente serviços na nuvem;
- Alguns exemplos: VoltDB, MySQL Cluster, etc.

No SQL – Pequena Zoologia



No SQL - Características

- **Escalabilidade:** o sistema pode lidar com a crescente quantidade de dados sem perder o desempenho.
 - **Vertical (scale-up):** adicionar recursos ao hardware (CPU, memória RAM, HD) para um único nó (computador) de forma a utilizar mais threads para lidar com um problema local;
 - **Horizontal (scale-out e utilizado em NoSQL):** adicionar mais nós (computadores) para um sistema distribuído. Atualmente, é possível em função dos custos. Aumenta-se a complexidade à medida que threads/processos sejam criados de forma distribuída, mas neste caso, a ausência de bloqueios facilita o gerenciamento. **Sharding:** dividir o banco de dados em vários nós (desnormalizar para trabalhar de forma distribuída).



No SQL - Características

	Vertical	Horizontal
Custo de hardware	Pode ser caro (linha “top” de equipamentos), além de componentes	Servidores mais baratos
Custo de software	Sem custo	Alguma licença para sistema operacional e banco de dados
Espaço físico	Sem necessidade	Mais servidores podem demandar espaço
Consumo de energia	Pouco (inexpressivo)	Mais servidores demandam mais consumo
Facilidade de implementação	Simple	Pode ser complicado (requer mão-de-obra técnica)
Capacidade de expansão	Há limites de hardware	Não há

No SQL - Características

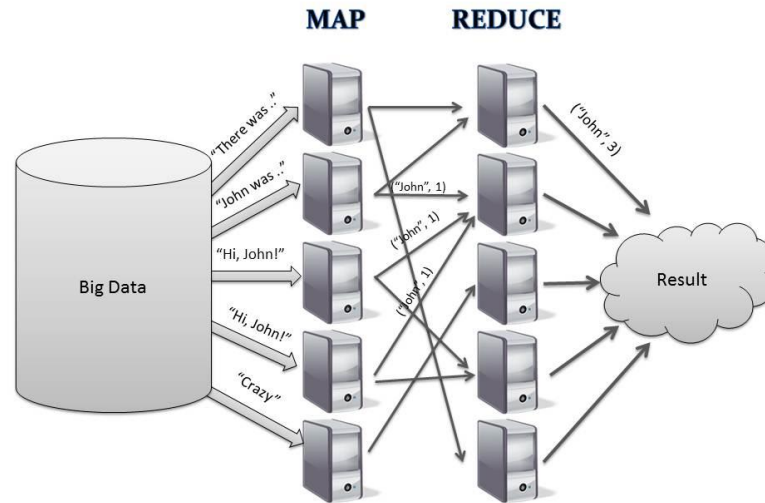
- **Baixa latência:** é obtida através da escalabilidade horizontal e que se caracteriza pela otimização do tempo de um grande processamento, dividindo-o em vários processos que são distribuídos em vários nós, reduzindo o tempo de resposta;
- **Disponibilidade de acesso:** através do grande processamento, permite-se acesso de um número de usuários maior que um SGBDR;
- **Processamento de grandes volumes:** seja através de uma quantidade de ciclos de leitura e escrita ou quantidade massiva de acessos de usuários;

No SQL - Características

- **Ausência de esquema ou esquema flexível:** ausência esta que facilita a escalabilidade horizontal e o aumento da disponibilidade. Em contrapartida, não há garantias da integridade dos dados;
- **Suporte nativo a replicação:** uma forma de prover escalabilidade é prover a replicação, portanto, a replicação nativa diminui o tempo gasto para recuperar informações;
- **API simples para acesso aos dados:** o foco não está em como os dados são armazenados, mas sim em como os mesmos são recuperados;
- **Consistência eventual:** é uma característica do NoSQL, pois nem sempre é mantida nos diversos pontos de distribuição.

No SQL - Técnicas

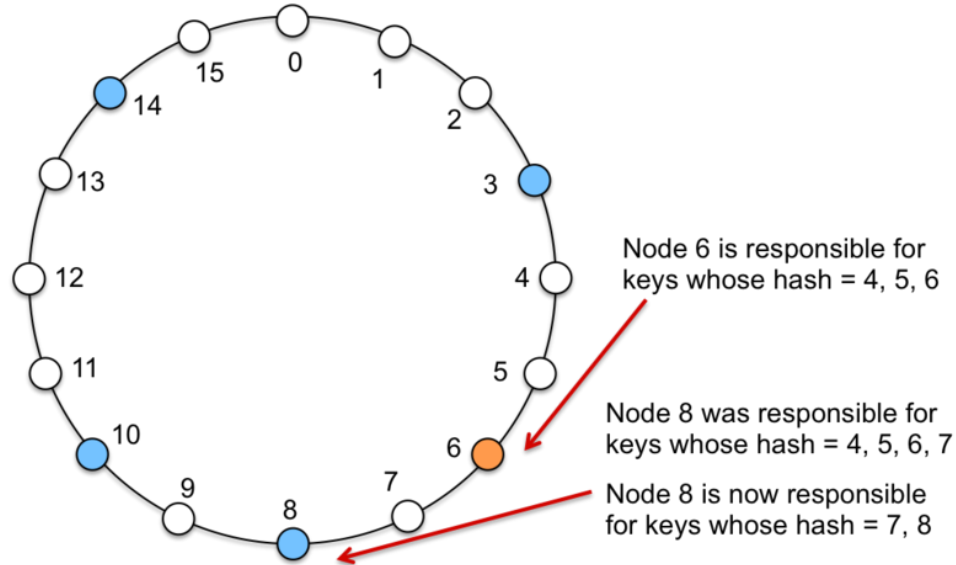
- **Map/reduce:** suporte ao manuseio de grandes volumes de dados distribuídos. Na fase de map os problemas são quebrados em subproblemas que são distribuídos em outros nós da rede.



Ferramentas Apache Hadoop e Spark.

No SQL - Técnicas

- **Consistent hashing:** mecanismos de armazenamento e recuperação em bancos de dados distribuídos onde a quantidade de nós está em constante modificação. Ex.: DHT (Distributed Hash Table).



No SQL - Técnicas

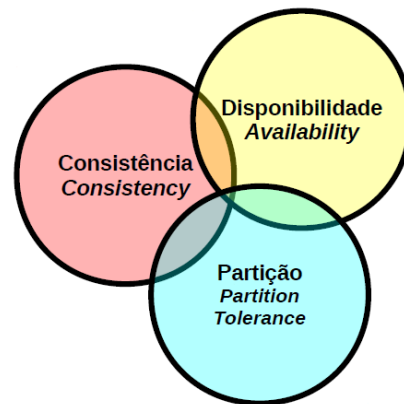
- **Multiversion Concurrency Control (MVCC):** mecanismo que dá suporte a transações paralelas em um banco de dados. Ao contrário do esquema clássico de gerenciamento de transações, como não faz uso de *locks*, permite que operações de leitura e escrita sejam feitas simultaneamente. Este mecanismo está relacionado diretamente com a característica “consistência eventual”;
- **Vector clocks:** mecanismo que permite ordenar os eventos acontecidos em um sistema. Como há possibilidade de várias operações estarem acontecendo ao mesmo tempo sobre o mesmo item de dado distribuído, o uso de um log de operações e as respectivas datas é importante para determinar qual versão de um dado é mais atual.

Teorema CAP (ou Brewer)

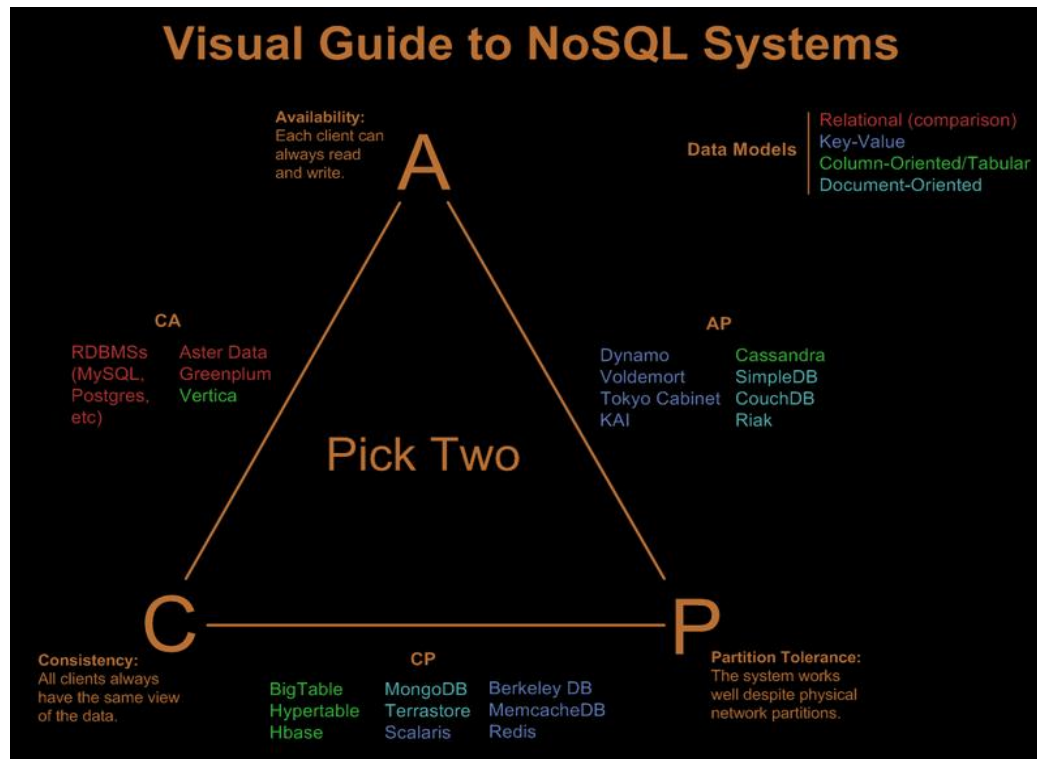
- Proposto por Brewer (ACM PODC'2000), e que afirma que é **impossível** um sistema computacional distribuído fornecer simultaneamente as três garantias a seguir:
- Consistência (consistency): todos os nós terem ao mesmo tempo o mesmo dado);
- Disponibilidade (availability): todas as requisições recebidas têm uma resposta (sucesso ou falha);
- Partição (partition tolerance): o sistema

continua operando apesar de mensagens de erro de uma parte do sistema).

- (Gilbert, Lynch SIGACT'2002) provaram a conjectura de Brewer.



Teorema CAP (ou Brewer)



Fonte: <http://blog.nahurst.com/visual-guide-to-nosql-systems>

Sistemas de Gerência de Banco de Dados

Relacional vs NoSQL

	Relacional	NOSQL
Escalonamento	Possível, mas complexo. Devido à natureza estruturada do modelo, a adição de forma dinâmica e transparente de novos nós no <i>grid</i> não é realizada de modo natural.	Uma das principais vantagens desse modelo. Por não possuir nenhum tipo de esquema pré-definido, o modelo possui maior flexibilidade o que favorece a inclusão transparente de outros elementos.
Consistência	Ponto mais forte do modelo relacional. As regras de consistência presentes propiciam um maior grau de rigor quanto à consistência das informações.	Realizada de modo eventual no modelo: só garante que, se nenhuma atualização for realizada sobre o item de dados, todos os acessos a esse item devolverão o último valor atualizado.
Disponibilidade	Dada a dificuldade de se conseguir trabalhar de forma eficiente com a distribuição dos dados, esse modelo pode não suportar a demanda muito grande de informações do banco.	Outro fator fundamental do sucesso desse modelo. O alto grau de distribuição dos dados propicia que um maior número de solicitações aos dados seja atendida por parte do sistema e que o sistema fique menos tempo não disponível.

BASE

- Para garantir escalabilidade e disponibilidade, sacrifica-se o paradigma do modelo relacional denominado ACID (Atomicidade, consistência, isolamento e durabilidade – veremos adiante!);
- Por isso, foi utilizado outro paradigma denominado **B**asically **A**vailable, **S**oft state, **E**ventual Consistency (BASE);
- Basically available: o banco de dados está disponível todo o tempo (apesar das falhas de partição);
- Soft state: é um fluxo e, portanto, não determinístico. Como “abre-se” mão da consistência, o dado pode estar em constante alteração;
- Eventual consistency: o dado será consistente em um determinado ponto no futuro.

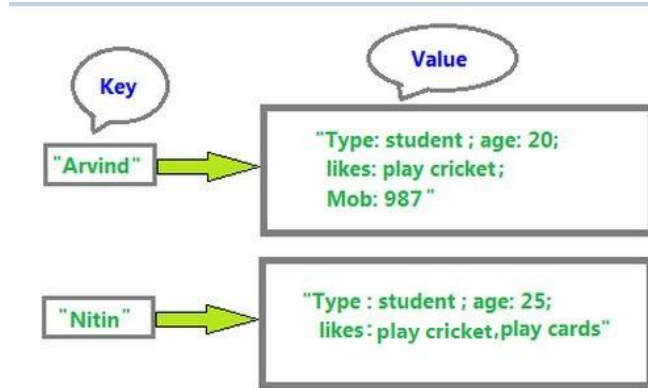
BASE vs ACID

ACID	BASE
Consistência forte	Consistência fraca
Isolamento	Últimas escritas vencem
Transação	Gerenciado pelo programa
Banco de dados robusto	Banco de dados simples
SQL (código simples)	Código complexo
Disponibilidade e consistência	Disponibilidade e partição
Escalonamento vertical	Escalonamento horizontal
Compartilhamento (disco, memória, processador, etc.)	Paralelizado (sem compartilhamento)

Modelos: ou Arquiteturas

- Chave-valor: modelo simples e, geralmente, é baseado em estruturas hash como chave. Possui uma relação chave-valor, onde a chave é única e que aponta para um item em particular. Além disso, as operações são bem simples, do tipo “get”, “set”, “delete”.

- Ex.:

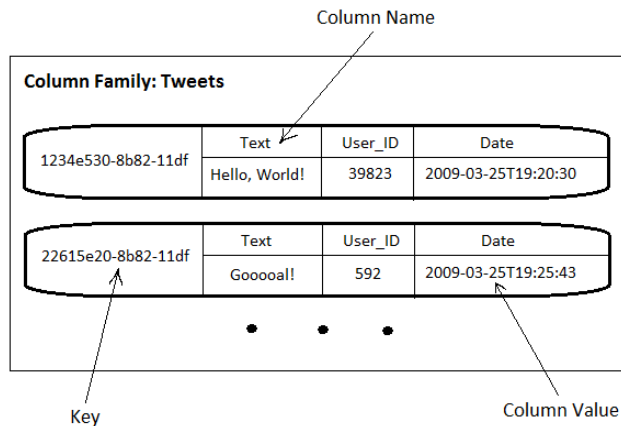


- Bancos: Dynamo, Redis, GenieDB, Voldemort.

Modelos: ou Arquiteturas

- Orientado a colunas (ou famílias de colunas): como o nome diz, é um modelo orientado a colunas, diferentemente do modelo relacional que é orientado a tuplas. Neste modelo, há o conceito de “triplas” que são baseadas em linha, coluna e timestamp.

- Ex.:



- Bancos: BigTable, Cassandra, HBase, Hadoop.

Modelos: ou Arquiteturas

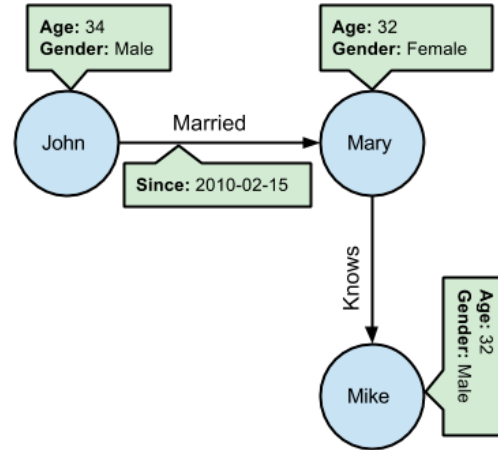
- Documentos: é um modelo que armazena coleções de documentos. Observa-se que um documento, geralmente, é um objeto com um identificador único com um conjunto de campos e que podem ser strings, listas ou documentos aninhados. Quando da inserção de um documento, é criado um identificador hash e, destaca-se que este modelo não possui um esquema rígido.
- Ex.:

```
{
  "firstName": "John",
  "lastName": "Smith",
  "age": 25,
  "address": {
    "streetAddress": "21 2nd Street",
    "city": "New York",
    "state": "NY",
    "postalCode": "10021"
  },
  "phoneNumber": [
    { "type": "home", "number": "212 555-1234" },
    { "type": "fax", "number": "646 555-4567" }
  ]
}
```
- Bancos: MongoDB, CouchDB, SimpleDB.

Modelos: ou Arquiteturas

- Grafos: é um modelo composto por três elementos: nós (vértices), relacionamentos (arestas) e propriedades (atributos) dos nós e relacionamentos. Pode ser visto como um multigrafo rotulado e direcionado. Escalabilidade neste modelo é problemático.

- Ex.:



- Bancos: Neo4J, GraphBase, InfoGrip.

Exemplo do Twitter

Users

john	name: John Doe	pass: swordfish	joined: 20091115
paul	name: Paul Lane	pass: thepass	joined: 20091129

Following

john	paul: 20091204	brigitte: 20100815	
paul	john: 20091205	debora: 20100729	brigitte: 20100822

Followers

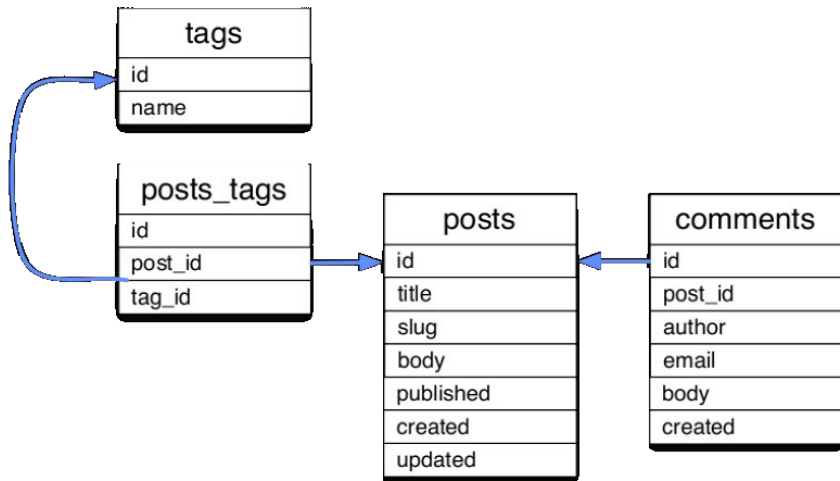
john	tom: 20091128	paul: 20091205
brigitte	john: 20100815	paul: 20100822

*@paul segue
@brigitte desde
22/08/2010*

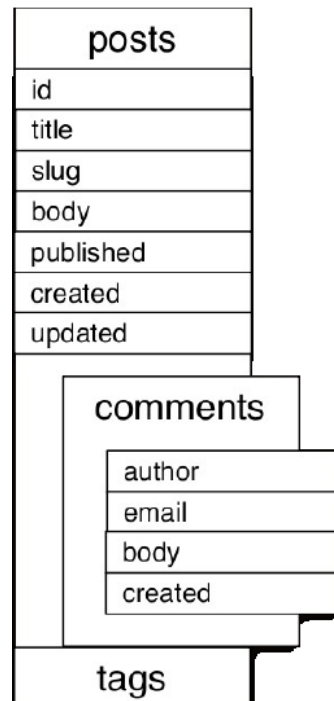
Comparativo Modelos

Modelo	Desempenho	Escalabilidade	Flexibilidade	Complexidade	Funcionalidade
Chave-valor	Alto	Alta	Alta	Nenhuma	Variável (nenhuma)
Família de colunas	Alto	Alta	Moderada	Baixa	Mínimo
Documento	Alto	Variável (alta)	Alta	Baixa	Variável (baixa)
Grafo	Variável	Variável	Alta	Alta	Teoria de Grafos
Banco de Dados Relacional	Variável	Variável	Baixa	Moderada	Álgebra Relacional

Estudo de Caso MongoDB

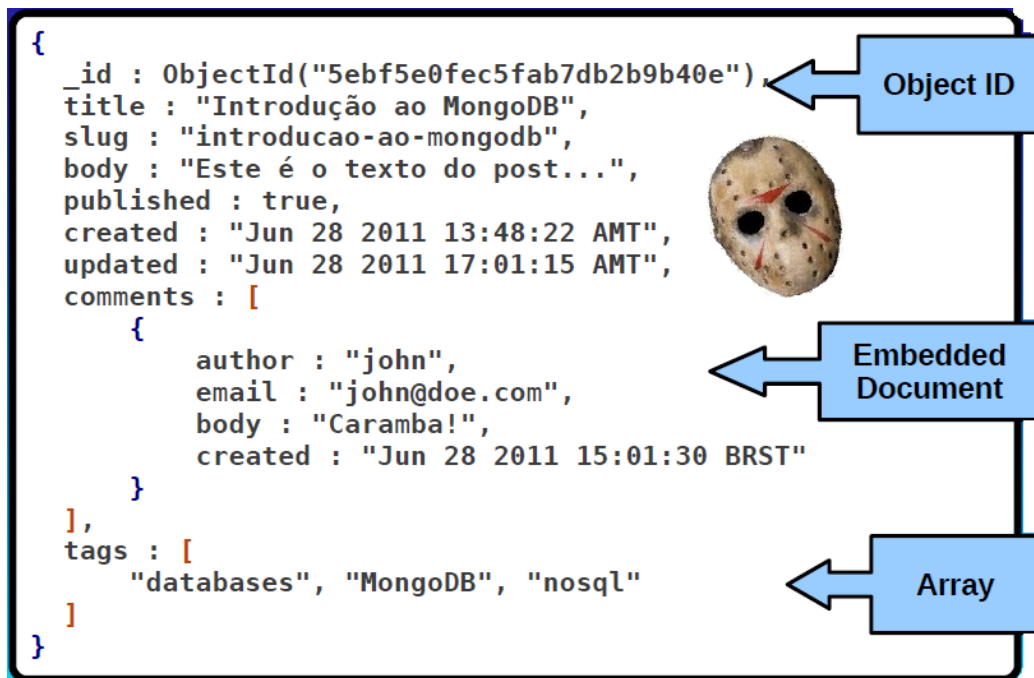


Modelo Relacional



Modelo Orientado a Documento

Estudo de Caso MongoDB



Estudo de Caso MongoDB

- Criar a coleção
db.createCollection("usuarios") ou
db.usuarios.insert(JSON)
- Adicionar um novo atributo à coleção
db.usuarios.update({}, {\$set: {"novo_atributo": "info"}}, {multi: true})
- Excluir um atributo
db.usuarios.update({}, {\$unset: {"novo_atributo": ""}}, {multi: true})
- Criar um índice
db.usuarios.ensureIndex({"title": 1})
- Excluir coleção
db.usuarios.drop()

Estudo de Caso MongoDB

```
SELECT * FROM usuarios  
> db.usuarios.find()
```

```
SELECT nome FROM usuarios  
> db.usuarios.find({}, {"nome": 1})
```

```
SELECT * FROM usuarios WHERE idade = 29  
> db.usuarios.find({"idade": 29})
```

```
SELECT * FROM usuarios WHERE idade = 29 AND ativo = true  
> db.usuarios.find({"idade": 29, "ativo": true})
```

```
SELECT * FROM usuarios WHERE idade >= 18 AND idade <= 30  
> db.usuarios.find({"idade": {"$gte": 18, "$lte": 30}})
```

```
SELECT * FROM usuarios WHERE nome LIKE "%admin%"  
> db.usuarios.find({"nome": /admin/i})
```

Estudo de Caso MongoDB

```
SELECT * FROM usuarios ORDER BY nome  
> db.usuarios.find().sort({"nome": 1})
```

```
SELECT * FROM usuarios ORDER BY idade DESC, nome  
> db.usuarios.find().sort({"idade": -1, "nome": 1})
```

```
SELECT * FROM usuarios LIMIT 3  
> db.usuarios.find().limit(3)
```

```
SELECT * FROM usuarios OFFSET 5  
> db.usuarios.find().skip(5)
```

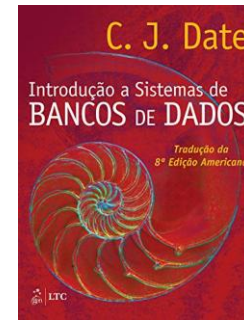
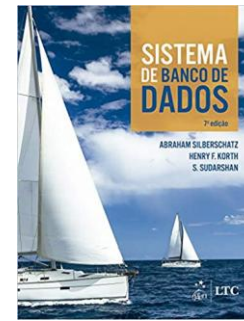
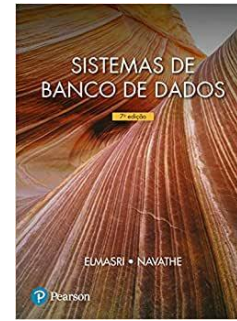
```
SELECT * FROM usuarios LIMIT 3 OFFSET 5  
> db.usuarios.find().limit(3).skip(5)
```

```
SELECT * FROM usuarios ORDER BY nome LIMIT 3  
> db.usuarios.find().sort({"nome": 1}).limit(3)
```

Links Interessantes

- Open Data Porto Alegre
<http://www.datapoa.com.br/dataset/poatransporte>
- Open Data NY
<https://data.ny.gov/>
<https://data.cityofnewyork.us/Transportation/Real-Time-Traffic-Speed-Data/xsat-x5sa>
- Open Data San Francisco
<https://data.sfgov.org/>
- Open Data Chicago
<https://data.cityofchicago.org/>
- Open Data (mais de 500)
<http://www.socrata.com/>
- Open Data (vários)
<http://dataportals.org/>
- Twitter
<https://dev.twitter.com/rest/reference/get/search/tweets>
- Facebook
<https://developers.facebook.com/tools/explorer/>

Bibliografia



- MULLER, Gilberto I.; **Apresentação e Notas de Aula**. Unisinos, 2021.
- **Slides Minicurso Interoperabilidade entre Bancos de Dados Relacionais e Bancos de Dados NoSQL**. Professores Geomar Schreiner e Ronaldo Mello, UFSC. Acessado em 25/08/2015.
- Grolinger, K.; Higashino, W.; Tiwari, A.; Capretz, M. **Data Management in Cloud Environments: NoSQL and NewSQL Data Stores**. *Journal of Cloud Computing: Advances, Systems and Applications*. 2013.
- **Slides Advances Databases and Data Models**. Professor Fang Wei-Kleiner, Linköping University. Acessado em 24/08/2015.
- **Slides NoSQL: onde, como e por quê?** Professor Rodrigo Hjort. Acessado em 24/08/2015.
- **Slides Minicurso Banco de Dados NoSQL: Conceitos, Ferramentas, Linguagens e Estudos de Casos no Contexto de Big Data**. Professores Marcos Vieira, Josiel Figueiredo, Gustavo Liberatti, Alvaro Fellipe, UFMT. Acessado em 24/08/2015.
- Almeida, R.; Brito, Parcilene. **Utilização da Classe de Banco de Dados NoSQL como Solução para Manipulação de Diversas Estruturas de Dados**. CEULP/ULBRA.



Sistemas de Gerência de Banco de Dados

Bancos de Dados NoSQL