

Profa. Andreza Leite

NOSQL E NEWSQL

Bancos Relacionais

Opções populares,
como MySQL,
PostgreSQL e
Oracle

Sempre foram a
primeira opção ao
desenvolver
qualquer aplicação

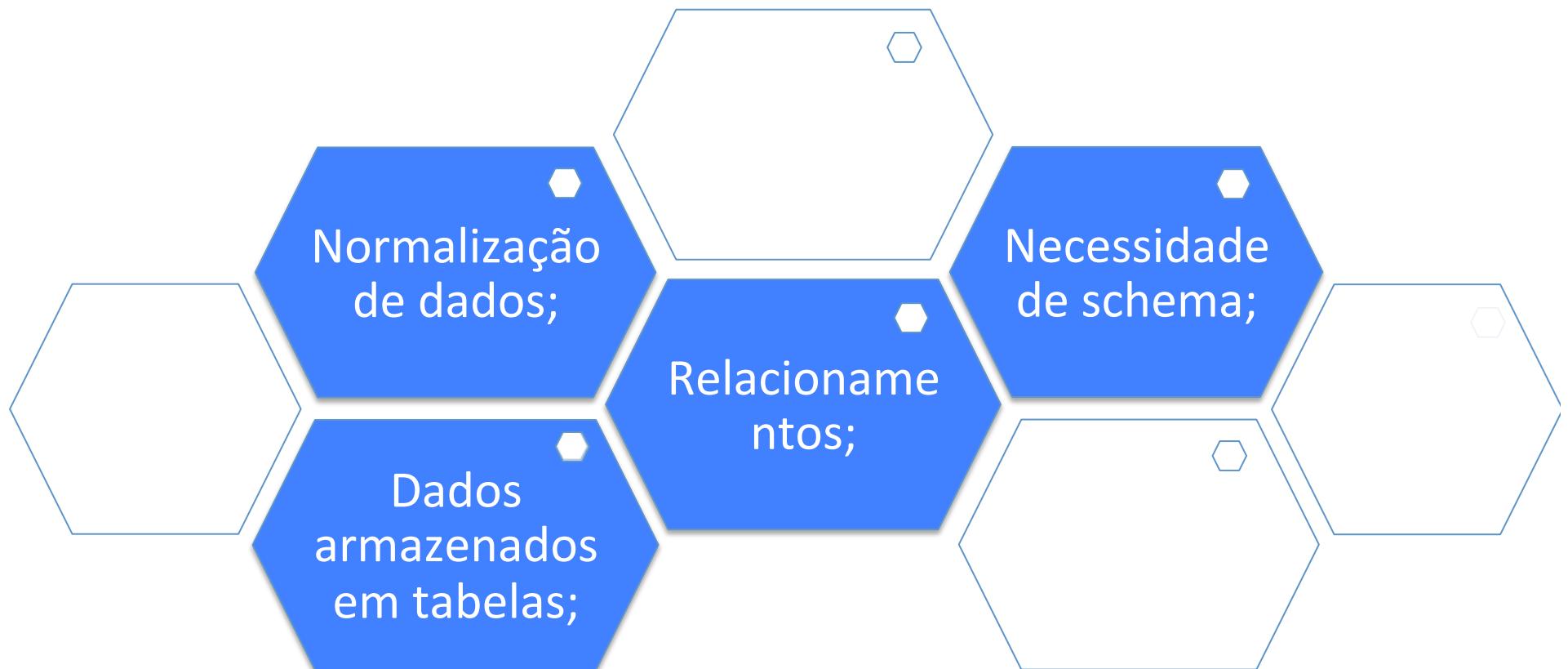
Muitas
ferramentas

Muita gente
qualificada no
mercado

Suporte amplo e
fácil de encontrar,
inclusive do
fabricante

Acabou se
tornando um
“padrão”

Modelo Relacional



Vantagens do Relacional

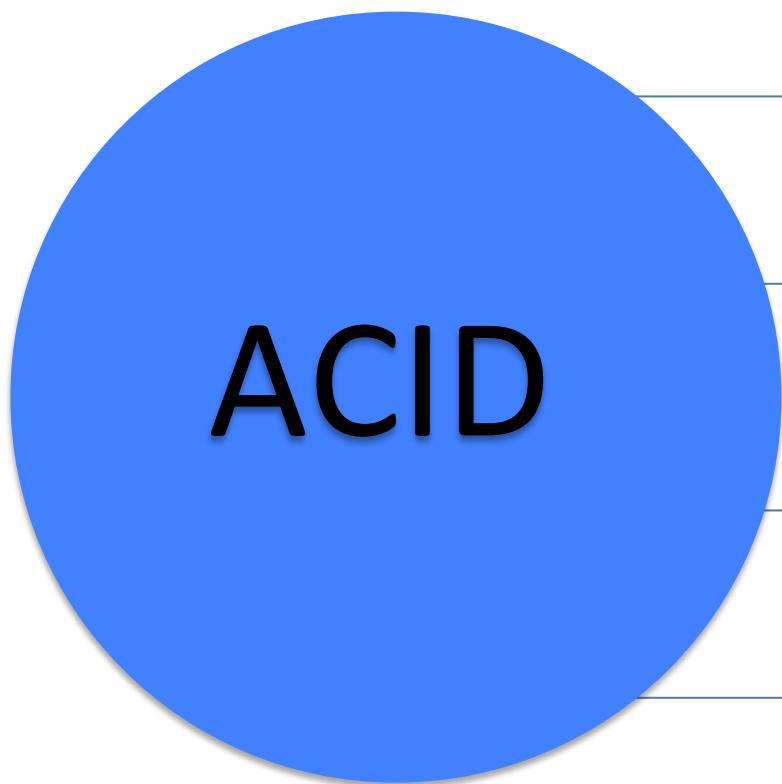
SQL → Quase
todo mundo
conhece;

- Linguagem bem flexível;
- Muitos operadores, stored procedures, boas ferramentas;

Dados bem
padronizados,
normalizados;

- Relacionamento;
- Join, group by, integridade relacional, etc

Vantagens do Relacional



Atomicidade (tudo ou nada)

Consistência (validação, respeita integridade)

Isolamento (concorrência retorna resultado válido)

Durabilidade (uma vez gravado, é definitivo)

Desvantagens Relacional

Dependência da
modelagem

- qualquer alteração, precisa passar por uma migração;

Dificuldade para manter

- aplicações que crescem muito rápido;

Dificuldade na
Escalabilidade

- Vertical (o banco está lento, adiciona memória);
- Compra uma máquina melhor;

E quando o banco começa ficar lento?



Contrata um cara para criar índices



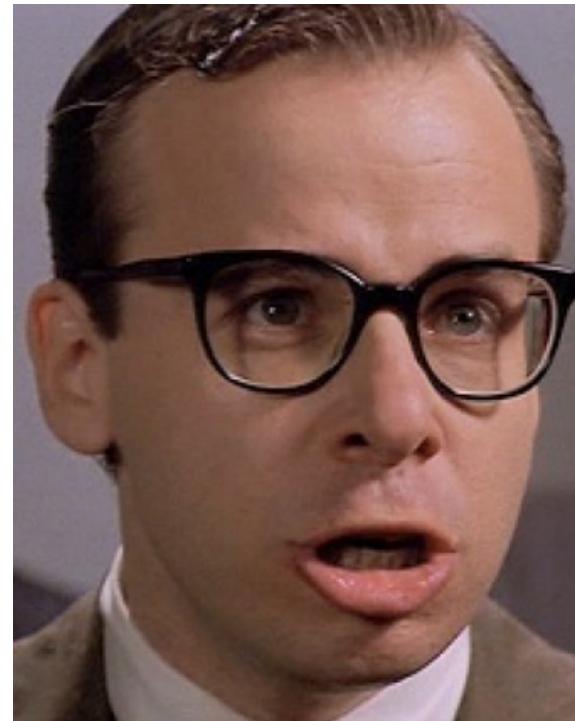
fica um pouco melhor, mas não resolve 100%

Contrata um cara para criar views e queries complexas



Melhora mais um pouco, mas vira um pesadelo para manter...

Contrata um cara para criar cache na app



Fica ainda melhor, mas a informação começa ser duplicada

NoSQL

Novo Paradigma

Não usam SQL

Normalmente open-souce

**Liberdade de schema e
modelagem**

Escalabilidade horizontal

Clusterização

Podem ser máquinas comuns

Persistência Poliglota

Muitos seguem o teorema de CAP
relax ACID

Não é bala de prata

Fazemos três tipos de serviços:

Bom,
Barato,
Rápido

- Se for BOM e BARATO não vai ser RÁPIDO
- Se for BARATO e RÁPIDO não vai ser BOM
- Se for BOM e RÁPIDO não vai ser BARATO

Teorema CAP

Escolha bem os recursos
que precisa.

Abra mão de outros



Leverage the NoSQL boom



Consistency **A**vailability **P**artition Tolerance

“... não é possível que um sistema de dados distribuído preencha estes 3 requisitos em simultâneo ...” Eric Brewer, 2000

CAP::Consistência Eventual

- Quando não há alterações durante um longo período de tempo, eventualmente todas as alterações irão propagar através do sistema e todos os nós serão consistentes
- Para uma determinada atualização e um determinado nó, eventualmente a atualização atinge o nó ou o nó é removido do serviço

BASE(Eventual Consistency)

becomes consistent at some later time

Soft State – não tem que ser consistente o tempo todo

CAP::Disponibilidade

- Ideal = servidor/processo disponíveis a 99.999 %
- Para um grande sistema, em quase qualquer tempo, há uma boa chance de que um nó caia ou haja uma interrupção de rede entre os nós.
 - Requer um sistema resiliente frente a interrupções de rede

BASE(Basic Availability)
system seems to work all the time

Modelo de consistência

- Determina regras para visibilidade e ordem de atualizações
- For example:
 - Linha X é replicada nos nós M and N
 - Cliente A escreve linha X para o nó N
 - Algum período de tempo t decorrido...
 - Cliente B lê linha X do nó M
 - Cliente B vê o que foi escrito pelo cliente A?
- Para NoSQL, a resposta seria: talvez
 - CAP Theorem: Consistência não pode ser alcançada ao mesmo tempo que disponibilidade e tolerância a falha (partição)

Transações em BDs...

- **A**tomicidade
- Consistência
- **I**solamento
- **D**urabilidade



SQL

- **B**asic
- A**vailability
- **S**oft-state
- Eventual
consistência



NoSQL

Teorema CAP

C

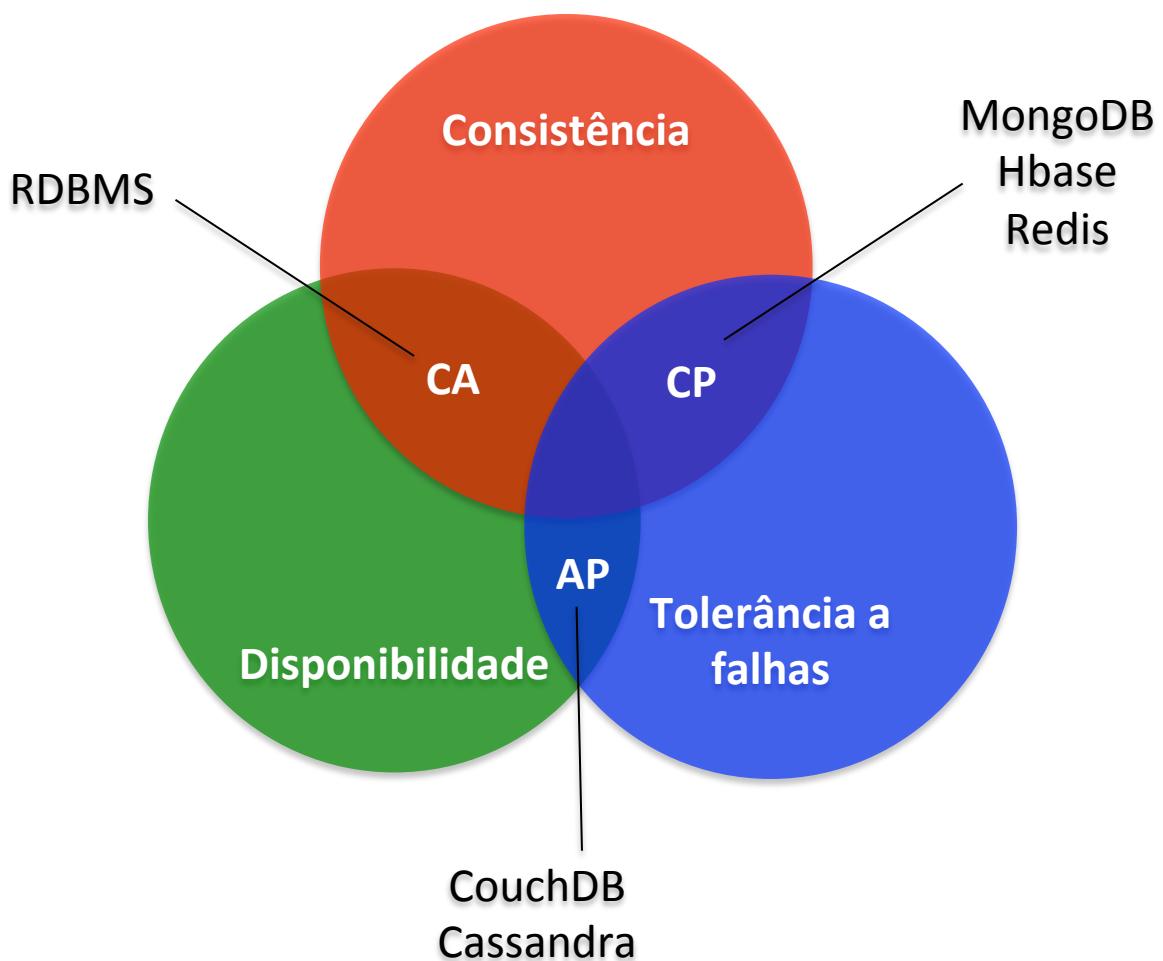
Todos os nós possuem o mesmo dado ao mesmo tempo

A

Garantia que todas as requisições recebam um retorno
true ou false

P

O sistema continua operando mesmo se parte do nó
estiver inacessível





Qual é a sua escolha?

Robustez e segurança estão mais ligados ao projetista+ frameworks/ferramentas utilizadas, do que à linguagem.

Eu+metralhadora

X

Chuck Norris+espoleta





Não há barreiras!

entre a tecnologia e o projetista

**Ser um bom Projetista
só depende de você!**



O que muda?

Desnormalizar

- NoSQL não trabalha de forma normalizada (Duplicidade,Falta de Consistência?)
- Isso pode ser bom ou ruim dependendo da sua aplicação

Novo Modelo
de Consultas

- não trabalham com conceito de queries (filtros)
- integrar novas formas de acesso a dados

Modelagem e
SQL

- Aprenda como o NoSQL funciona, não pense da mesma forma que no relacional
- Concentre-se em Schema Design

O que muda?

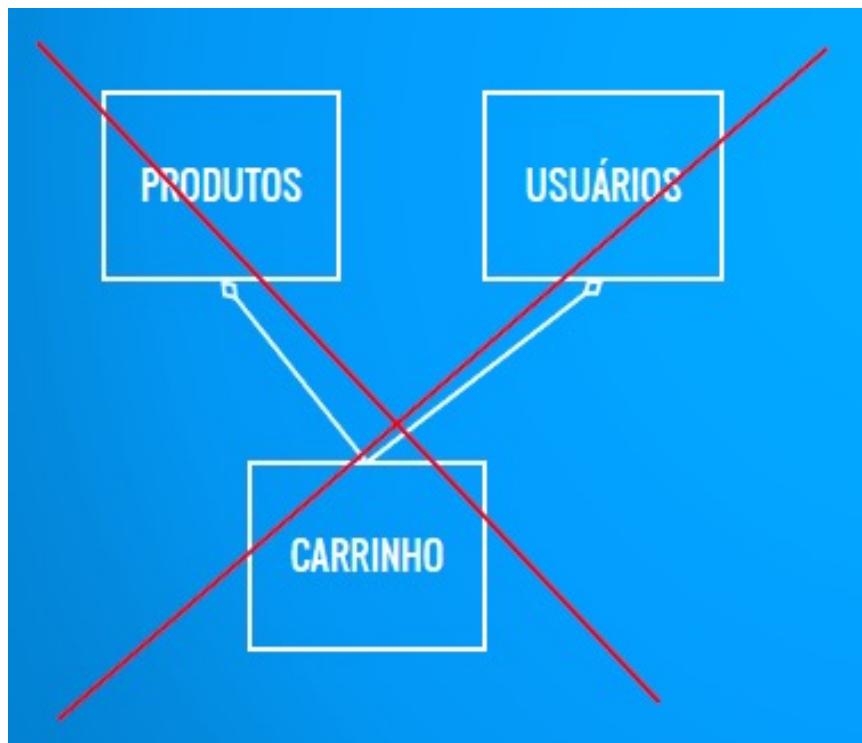
RDBMS

- Uma única máquina
- Escala verticalmente
- Full Index
- SQL

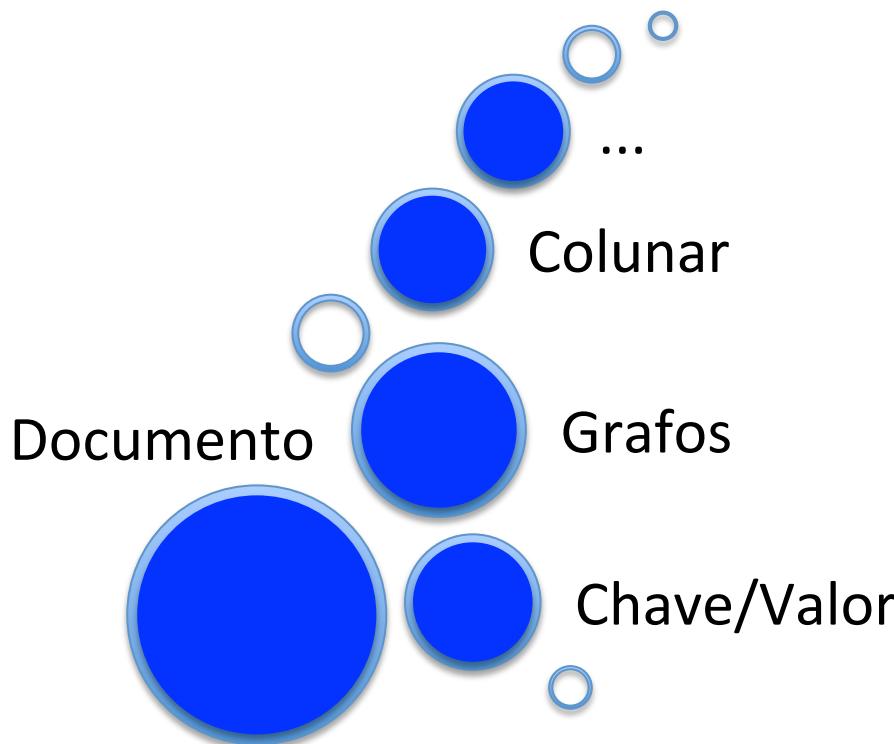
NoSQL

- Um Cluster
- Escala horizontalmente
- Baseado em chaves
- Possui API e filtros de pesquisa

RELAÇÃO X AGREGADOS



Diversos Modelos



Identifique qual
é o mais
apropriado para
sua aplicação

<http://nosql-database.org>



Documento



Colunar



Grafos



Chave/Valor



Ao infinito e além!



Profa. Andreza Leite

MODELOS

Qual NoSQL escolher?

- Chave-Valor ou ‘*the big hash table*’ .
- *Schema-less* ou column-based,
- document-based ou
- graph-based
- ...

Chave/Valor

- Sistema de BD com o modelo mais simples.
- Estrutura: Pares de chaves e valores (hashtable)
- Muito bom desempenho pela simplicidade do modelo
- Não recomendado com consultas complexas
 - Exemplos: MemcacheDB, Redis, Riak, DynamoDB, Amazon S3 (Dynamo), Voldemort, Scalaris

- Modelo de dados
 - {"**nome**" : "Andreza"}

↓
Chave

↓
Valor

Chave-Valor

Pros:

- Rápido
- Escalável
- Modelo simples
- Distribui horizontalmente

Con:

- Muitas estruturas de dados (objetos) não podem ser facilmente modeladas como pares de chave-valor

Orientado a colunas

- Boa solução para dados esparsos
- Estrutura: entre relacional e chave/valor
- Colunas são armazenadas juntas
 - Exemplos: Hbase, Cassandra, Hypertable

Orientado a colunas

Pros:

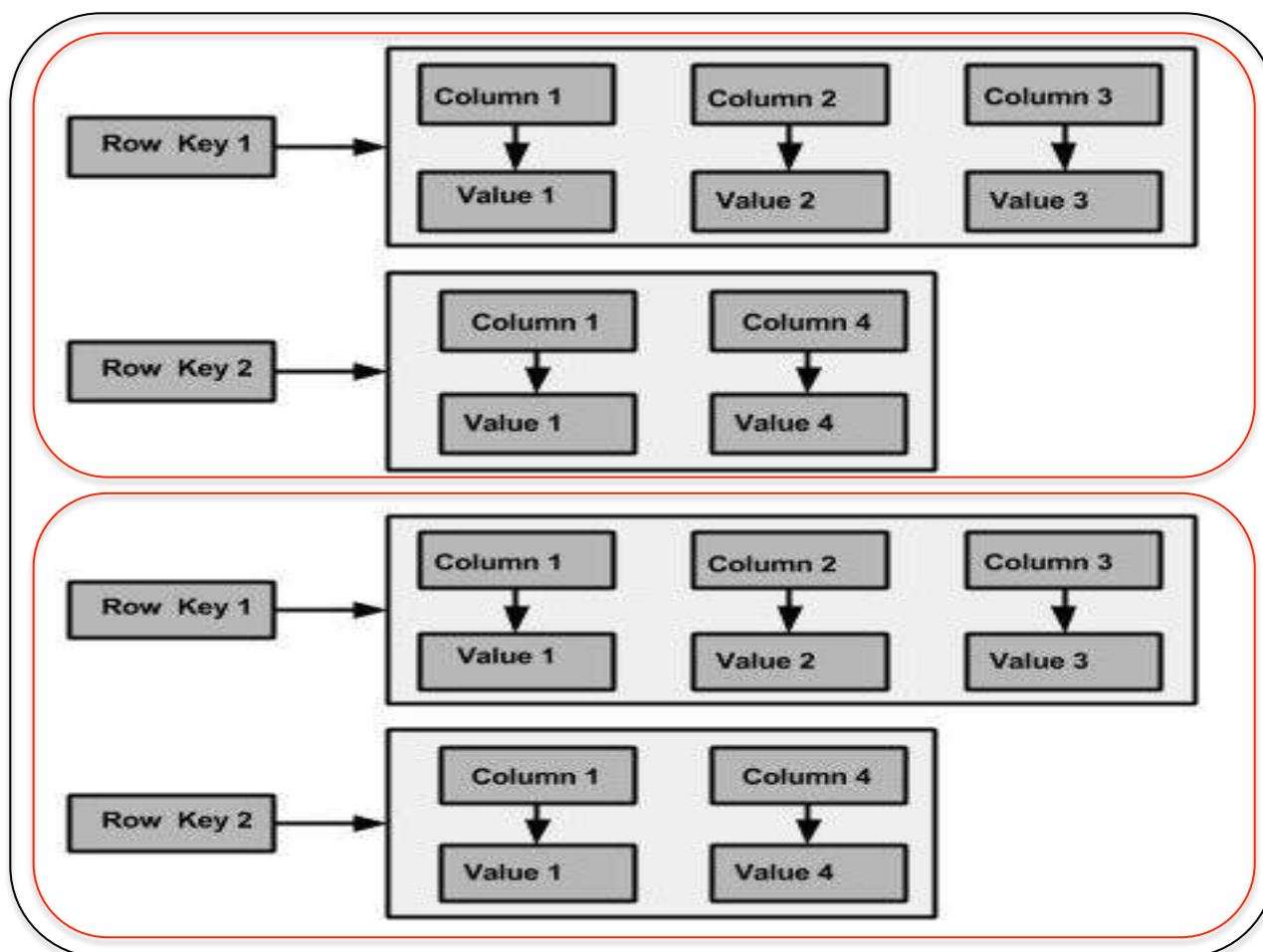
- É mais rico que pares de chave-valor
- Consistência eventual
- Muitos são distribuídos
- Provê performance e escalabilidade

Con:

- tipicamente não possui transações ACID ou joins

Orientado a colunas

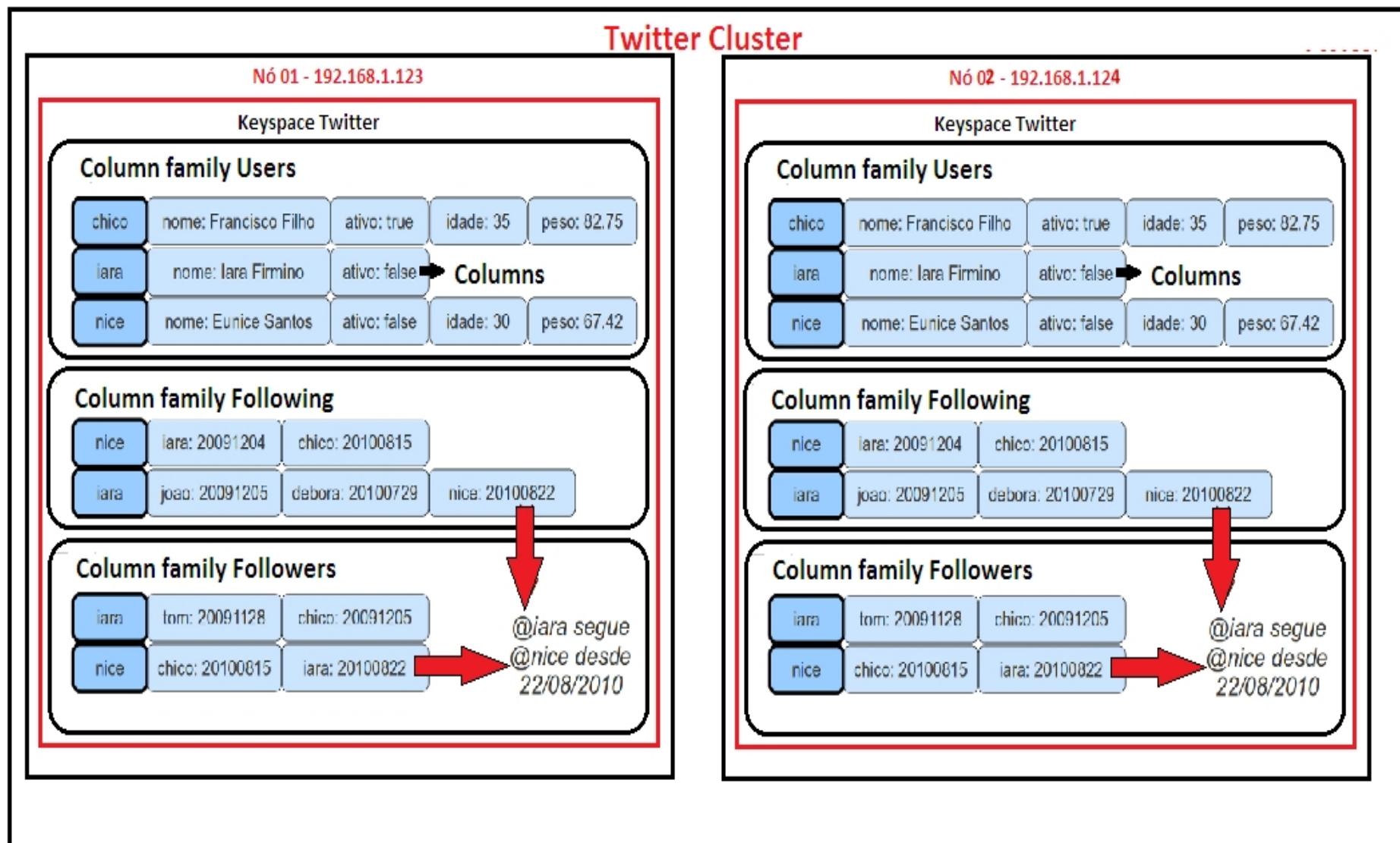
Keyspace



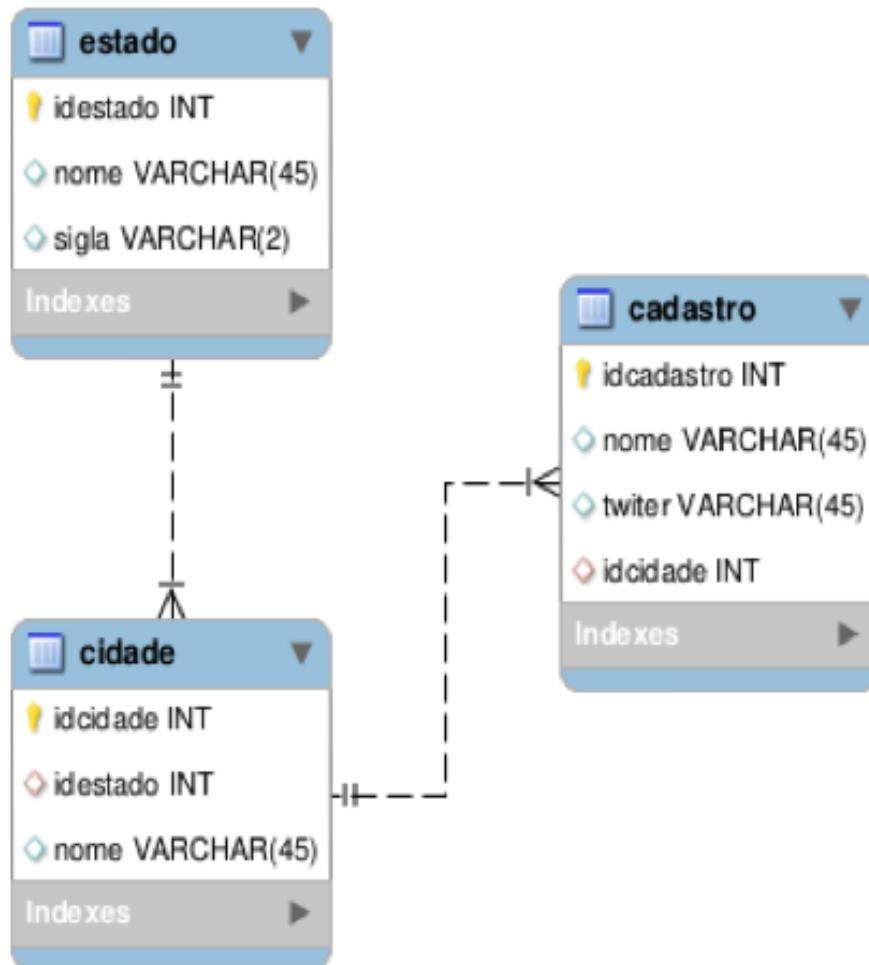
Família de colunas 1

Família de colunas 2

Orientado a colunas



Colunas X Relacional



```
mysql> select * from cadastro;
+----+-----+-----+-----+
| id | nome      | twitter | cidade |
+----+-----+-----+-----+
| 1  | Christiano | dump   | 1       |
+----+-----+-----+-----+
1 row in set (0.00 sec)
```

Informação normalizada,
Relaciona com a tabela de cidade que
relaciona com a tabela de estado

Colunas X Relacional

Família de colunas Cadastro

| | | | |
|---------|--------------|-----------------|---------------|
| Andreza | Nome:Andreza | Twitter:andreza | Cidade:Recife |
| Chico | Nome:Chico | | |

Familia de colunas cidade

| | | |
|--------|-------------|-----------|
| Recife | Nome:Recife | Estado:PE |
|--------|-------------|-----------|

Familia de colunas Estado

| | | |
|----|-----------------|----------|
| PE | Nome:Pernambuco | Sigla:PE |
|----|-----------------|----------|

Orientado a documentos

- Não possuem esquemas
- Estrutura: Documentos que consistem em coleções de atributos e valores
- Ideal para armazenamento de dados semi-estruturados
 - Exemplos: MongoDB, CouchDB

Orientado a documentos

```
{  
    "_id" : ObjectId("541f30d992a2ee25fedaa652"),  
    "nome" : "Andreza",  
    "twitter" : "andreza_paju"  
}
```

↓
Chave

↓
Valor

Orientado a documentos

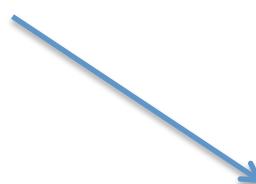
```
{  
    "_id" : ObjectId("541f30d992a2ee25fedaa652"),  
    "nome" : "Andreza",  
    "twitter" : "andreza_paju"  
    "linguagens" : [  
        "Python",  
        "C",  
        "JavaScript",  
        "C++"  
    ]  
}
```



Uma lista ou array

Orientado a documentos

```
{  
    "_id" : ObjectId("541f30d992a2ee25fedaa652"),  
    "nome" : "Andreza",  
    "redes_sociais" : {  
        "Twitter" : "andreza_paju",  
        "Facebook" : "andreza.leite1",  
        "LinkedIn" : "andrezaleite",  
    }  
}
```

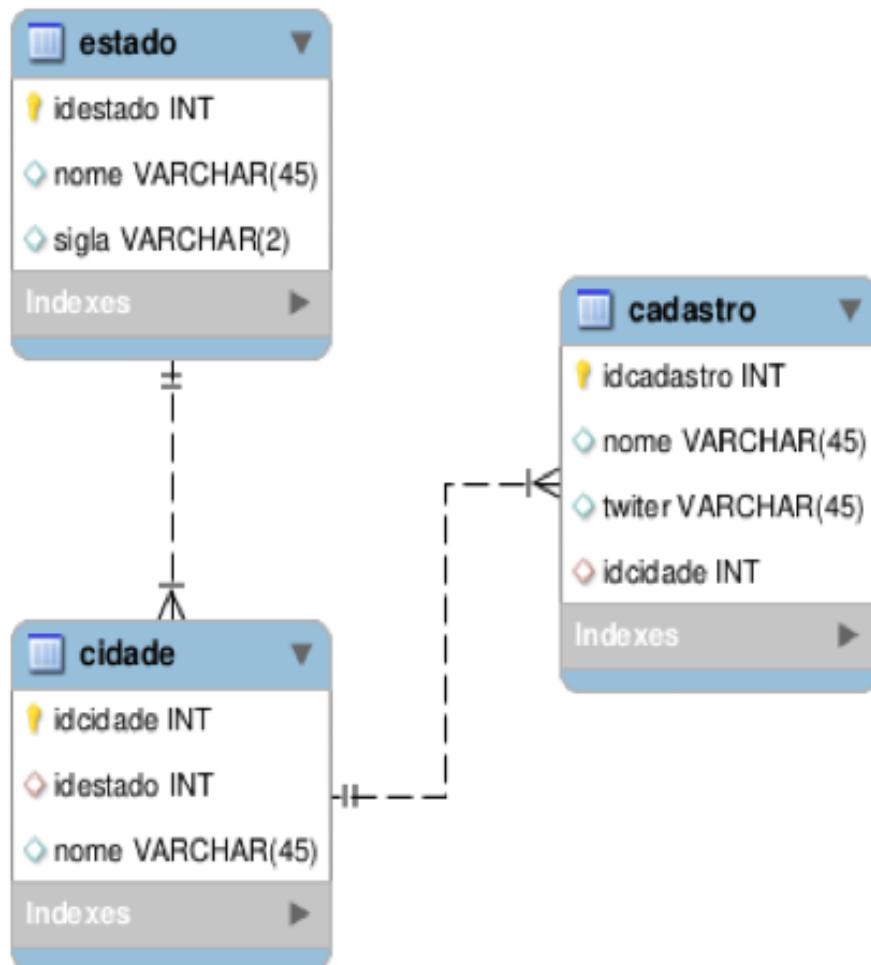


**Uma lista de valores ou
Um documento aninhado**

Documentos X Relacional

- Dados:
 - Nome: Andreza Leite
 - Twitter: andreza_paju
 - Cidade: Recife
 - Estado: PE

Documentos X Relacional



```
mysql> select * from cadastro;
+-----+-----+-----+-----+
| id | nome      | twitter | cidade |
+-----+-----+-----+-----+
| 1  | Christiano | dump    | 1       |
+-----+-----+-----+-----+
1 row in set (0.00 sec)
```

Informação normalizada,
Relaciona com a tabela de cidade que
relaciona com a tabela de estado

Documentos X Relacional

```
{  
    "_id" : ObjectId("541f64d092a2ee25fedaa654"),  
    "nome" : "Andreza",  
    "twitter" : "andreza",  
    "cidade" : "Recife",  
    "estado" : "PE"  
}
```

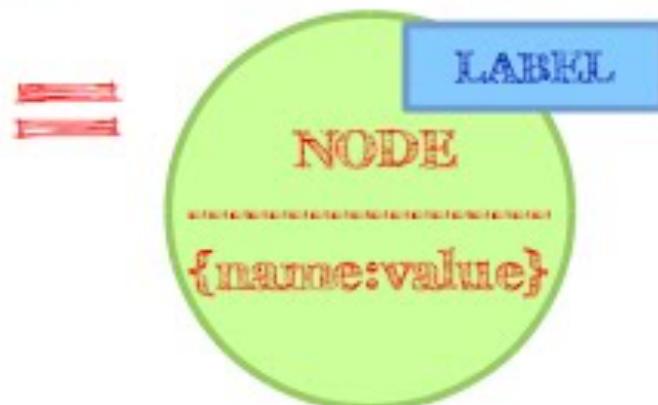
- Coleções não relacionadas - dados desnormalizados
- Pode ocorrer inconsistências – controle feito via código;

Baseado em grafos

- Apropriada para dados altamente interconectados.
 - Interconectividade dos dados é tão ou mais importante quanto os dados em si
- Estrutura: Nós e relações.
- Benefício: a possibilidade de navegar entre nós através das relações
- Exemplos: Neo4J, Infinite Graph, InforGrid

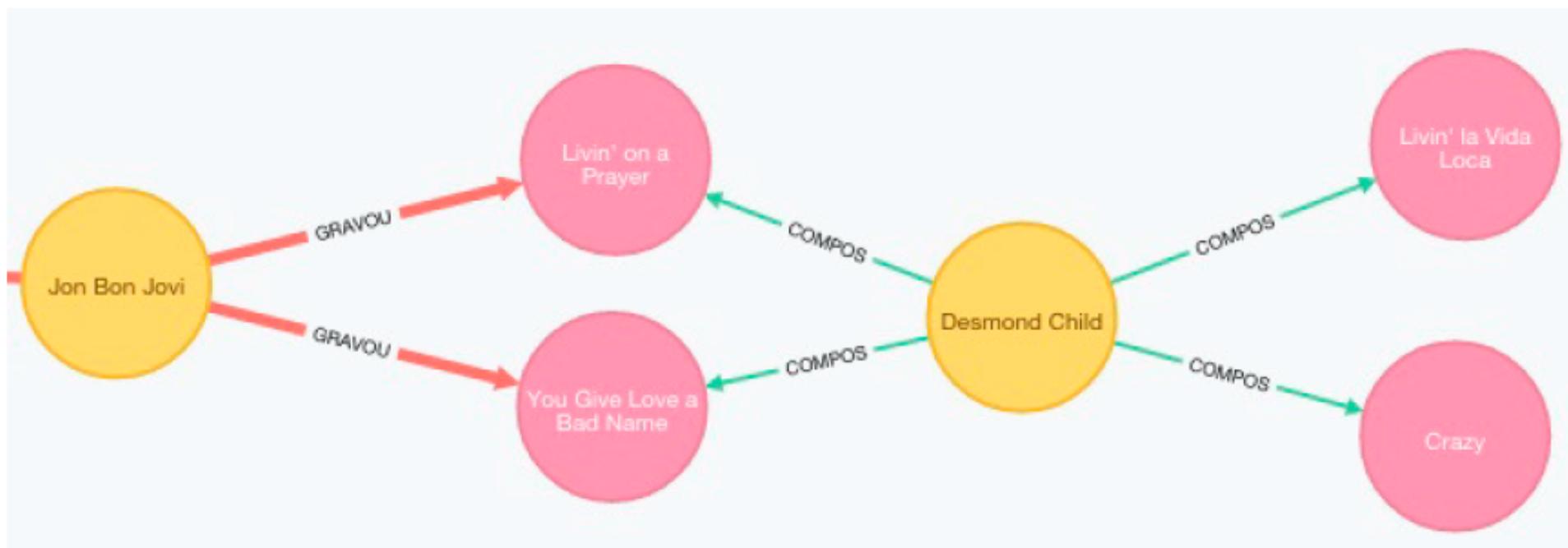
Baseado em grafos

NODE + PROPERTY

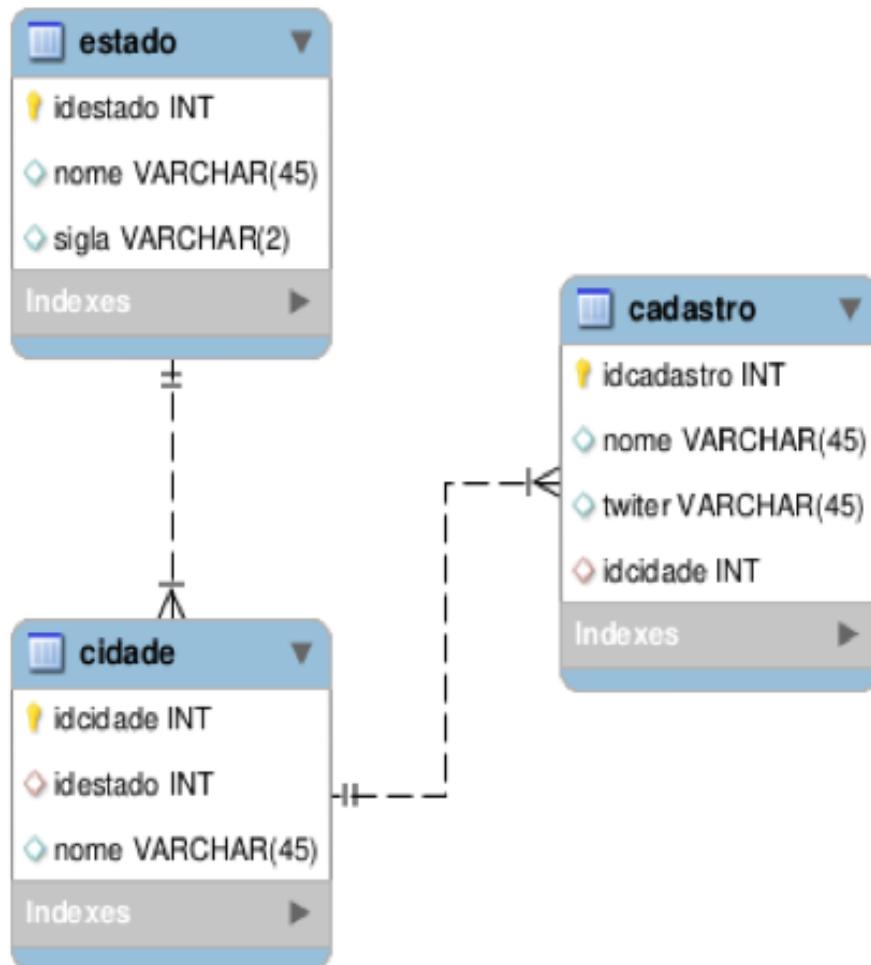


Baseado em grafos

- Grafos – nós(entidades) e relacionamentos podem ter atributos/propriedades



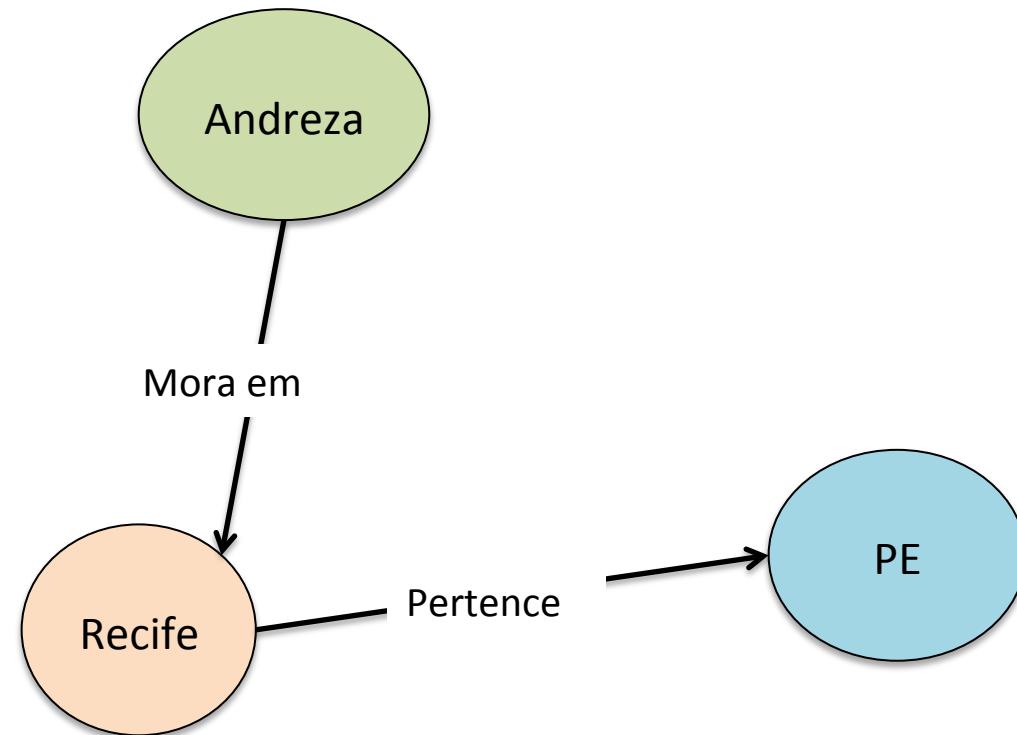
Grafo X Relacional



```
mysql> select * from cadastro;
+----+-----+-----+-----+
| id | nome      | twitter | cidade |
+----+-----+-----+-----+
| 1  | Christiano | dump   | 1       |
+----+-----+-----+-----+
1 row in set (0.00 sec)
```

Informação normalizada,
Relaciona com a tabela de cidade que
relaciona com a tabela de estado

Grafo X Relacional



Vantagens comuns

- Fácil de implementar (open source)
- Dados são replicados para multiplos nós (idênticos e tolerante a falhas) e pode ser particionado
 - Nós que falharem são facilmente substituídos
 - Sem ponto único de falha
- Fácil de distribuir
- Não requer um esquema
- Pode escalar (Up and Down)
- Relaxa a exigência de consistência dos dados(CAP)

Do quê desistimos?

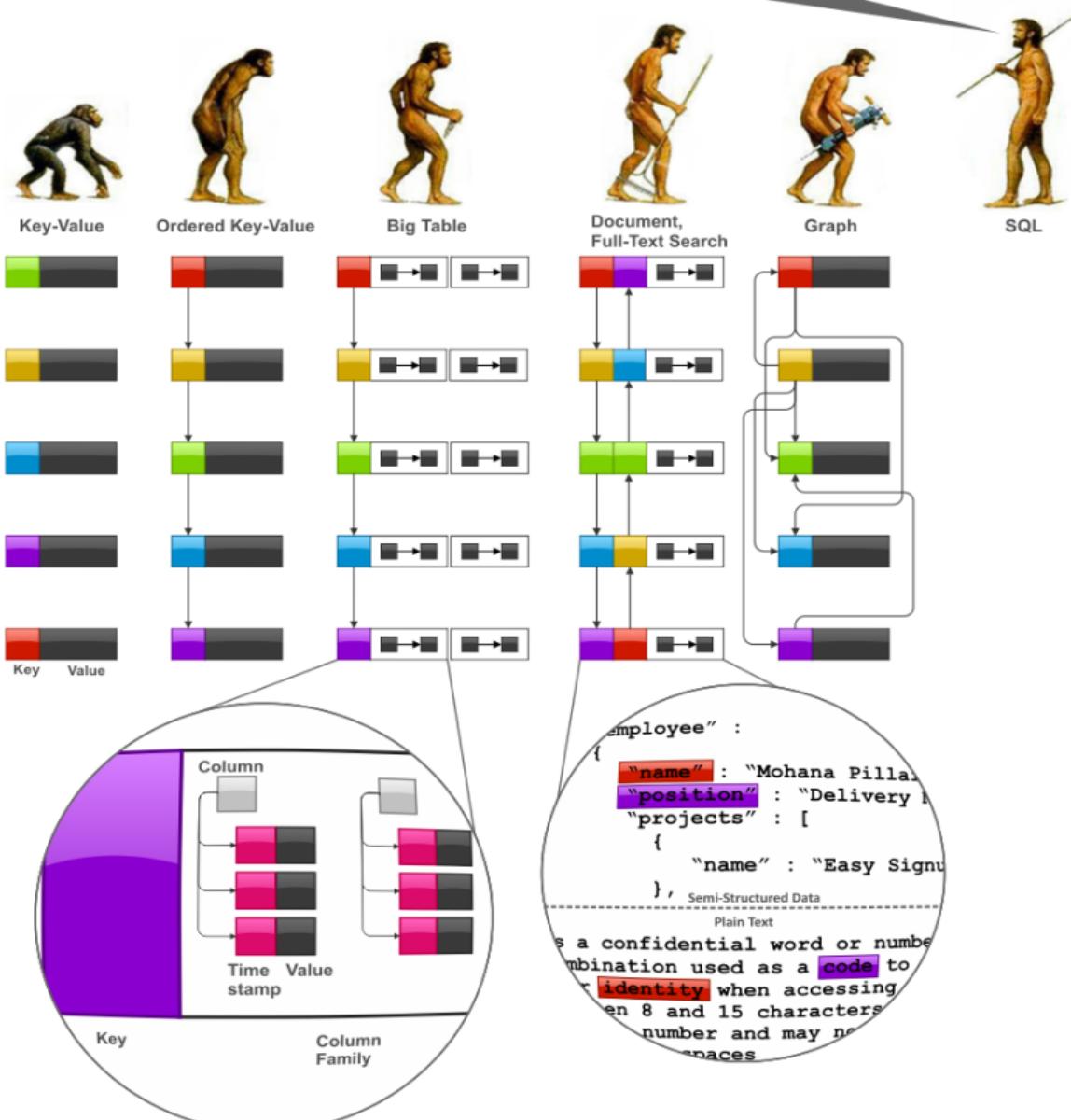
- NoSQL tem pouco suporte a
- joins
- group by
- order by
- ACID transactions
- SQL (powerful query language)
- Integração com outras aplicações que suportam SQL

Profa. Andreza Leite

MODELAGEM NOSQL

Modelagem NoSQL

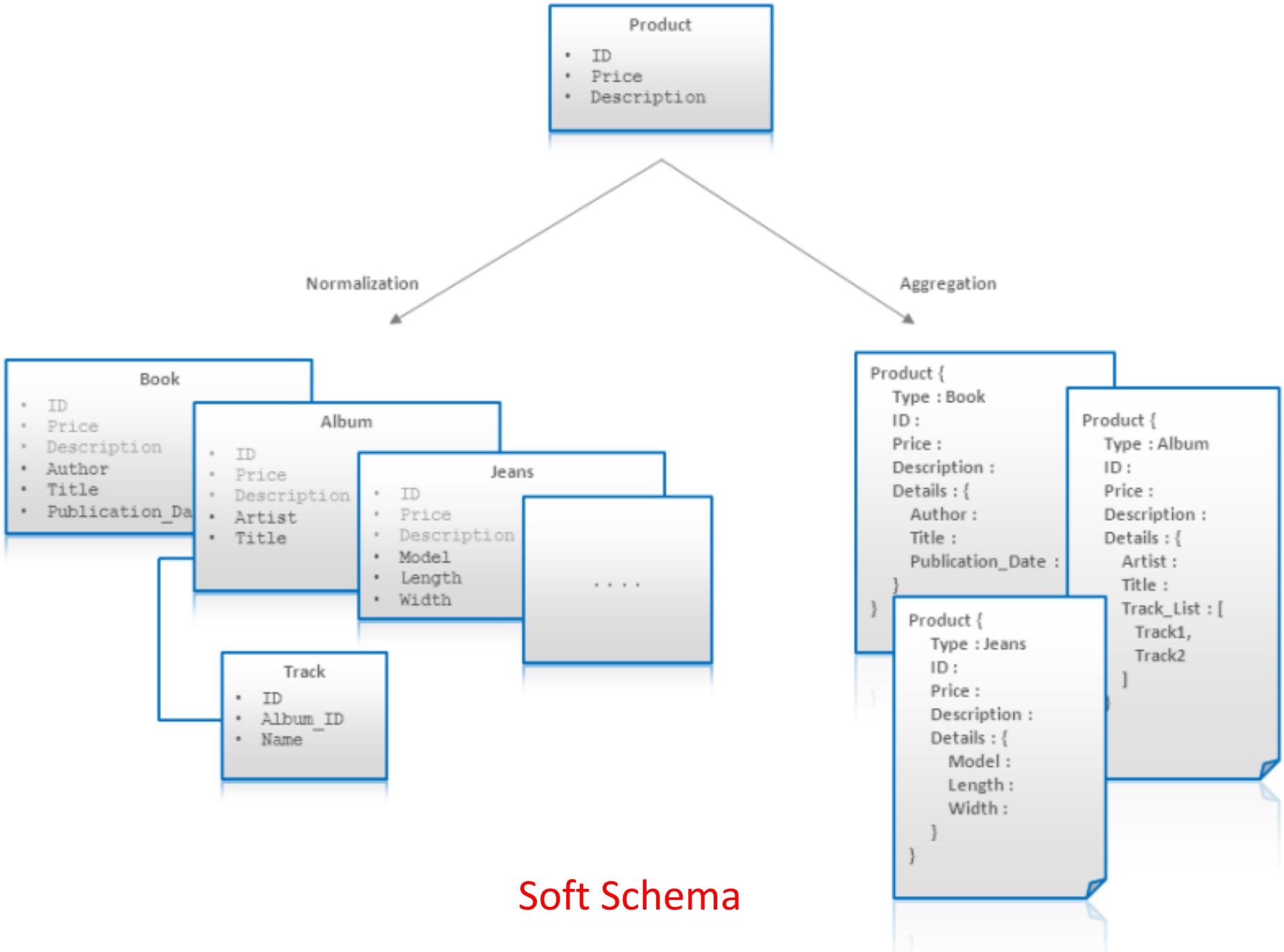
Stop following me, you fucking freaks!



Modelagem NoSQL

Técnicas Conceituais

- Desnormalização
 - Cópia do mesmo dado em vários documentos ou tabelas p/ simplificar/otimizar processamento de consultas ou encaixar dados de usuário em um modelo de dados particular.
 - Key-Value Stores, Document Databases, BigTable-style Databases (Column Family)
- Agregados
 - A maioria dos sistemas NoSQL oferecem a capacidade de esquemas leves (Soft Schema) permitindo formar classes de entidades com estruturas internas complexas(aninhadas) e variar a estrutura de certas entidades.
 - Minimiza relacionamentos 1:M, reduzindo joins
 - Mascaramento de diferenças técnicas entre entidades de negócio e modelagem de entidades de negócio heterogêneas usando uma coleção de documentos ou uma tabela.
 - Key-Value Stores, Document Databases, BigTable-style Databases (Column Family)
- Application Side Joins

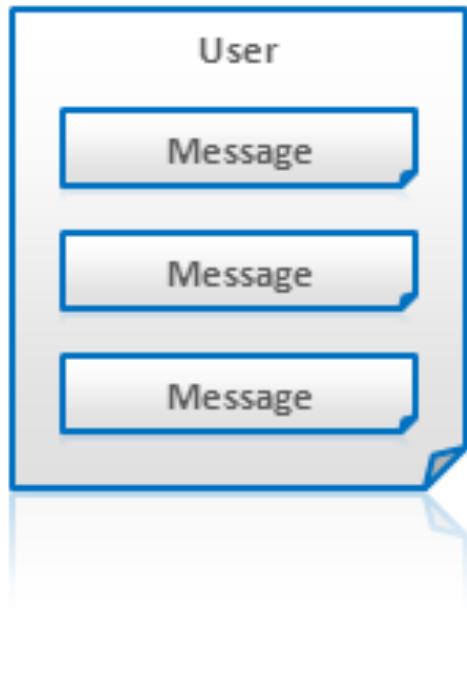


Modelagem NoSQL

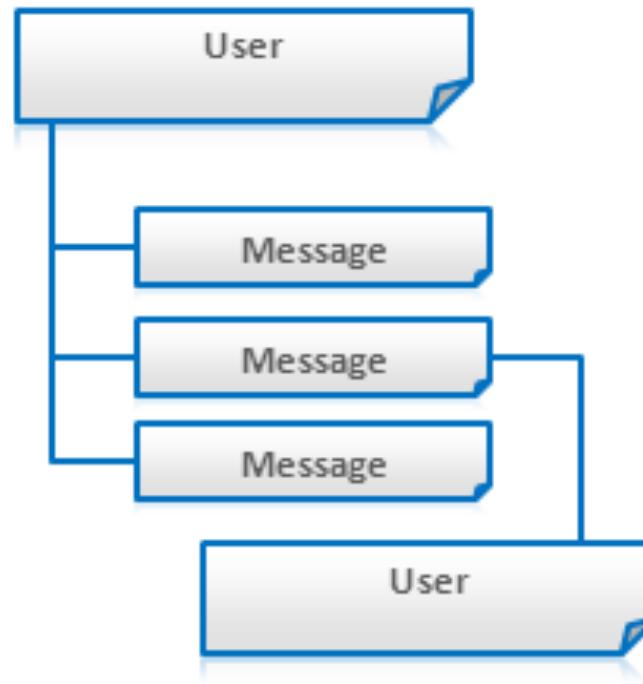
Técnicas Conceituais

- Application Side Joins
 - Junções são raramente suportados por soluções NoSQL
 - Como consequência da natureza NoSQL “question-oriented”, junções são frequentemente tratadas em tempo de design. Em modelos relacionais as junções são tratadas no tempo de execução da consulta.
 - Relacionamentos M:M são modelados por links e requerem junções
 - Agregados são frequentemente inaplicáveis quando entidades internas são objeto de frequentes modificações
 - Key-Value Stores, Document Databases, BigTable-style Databases, Graph Databases

Aggregates



Joins



Static
One-To-Many

Dynamic
Many-To-Many



Um sistema de mensagens pode ser modelado como uma entidade “User” que contém entidades “Message” aninhadas. Mas, se as mensagens forem frequentemente abertas, talvez seja melhor extrair as mensagens como entidades independentes e junta-las ao usuário no momento da consulta.

Técnicas de Modelagem Geral

- Atomic Aggregates
 - Key-Value Stores, Document Databases, BigTable-style Databases
- Enumerable Keys
 - Key-Value Stores
- Dimensionality Reduction
 - permite mapear dados multidimensionais para um modelo de valor-chave ou para outros modelos não multidimensionais
 - Key-Value Stores, Document Databases, BigTable-style Databases
- Index Table
 - considerada como um análogo de visões materializadas em BDRs
 - BigTable-style Databases
- Composite Key Index
 - Um tipo de index multidimensional. Pode-se usar chaves em um formato *State:City:UserID*. Permite interagir com vários registros de uma mesma cidade ou estado (partial key match)
 - BigTable-style Databases

Projeto de BD

- Modelo Conceitual
 - abstração de mais alto nível
- Lógico
 - representação da modelagem conceitual em um modelo de BD
- Físico
 - Implementação

Nova geração de sistemas web

Escalabilidade

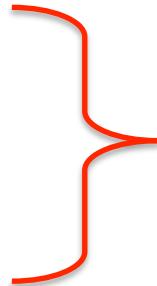
+

Performance

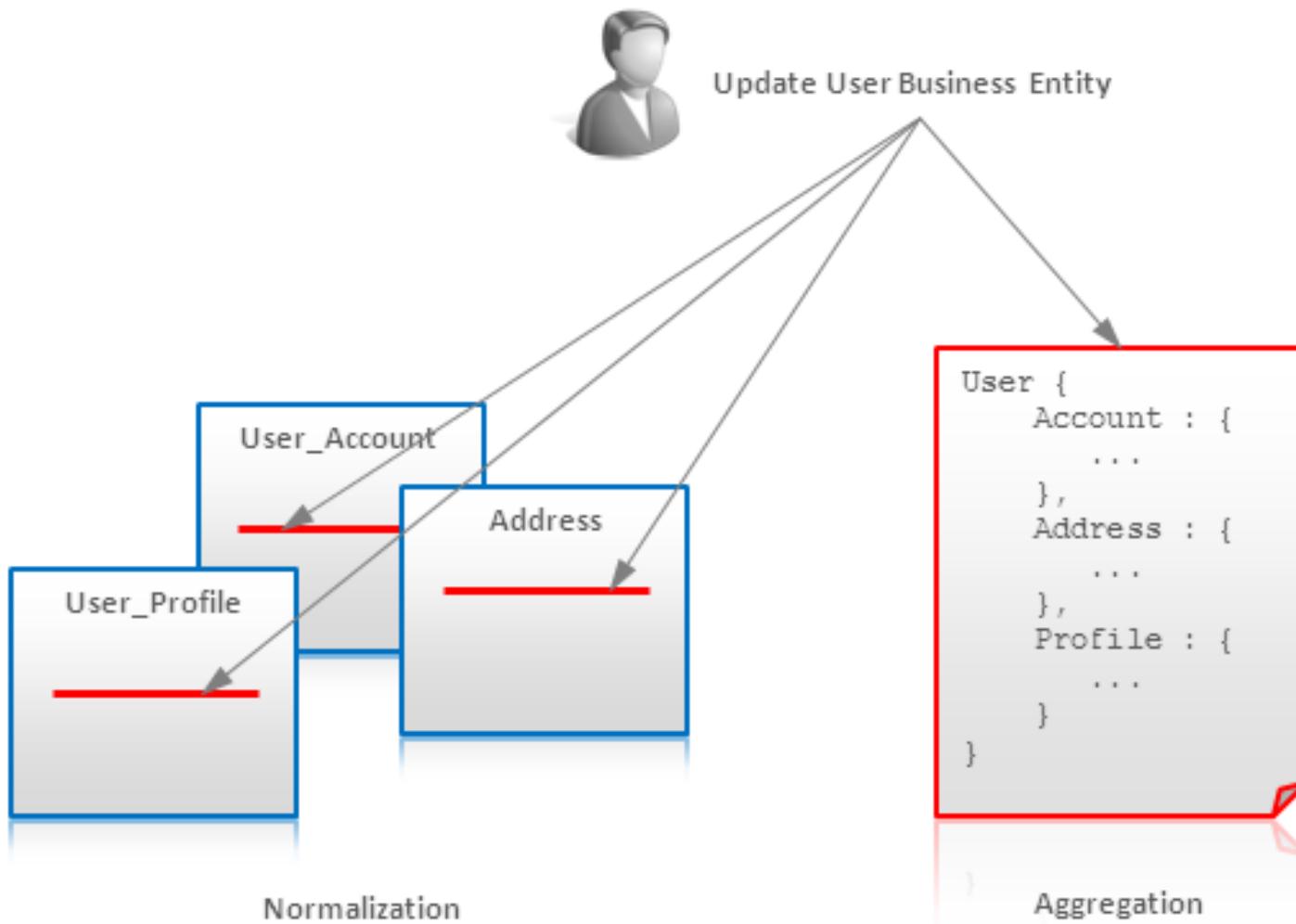
+

Consistência

Projeto de BD

- Modelo Conceitual
 - abstração de mais alto nível
 - Diagramas ER, Classe, UML
 - Lógico
 - modelo de BD
 - Agregados
 - Físico
 - Implementação
 - SGBD
- 
- É comum modelar dados usando uma técnica de Agregados para garantir algumas das propriedades ACID

AGREGADOS ATOMICOS



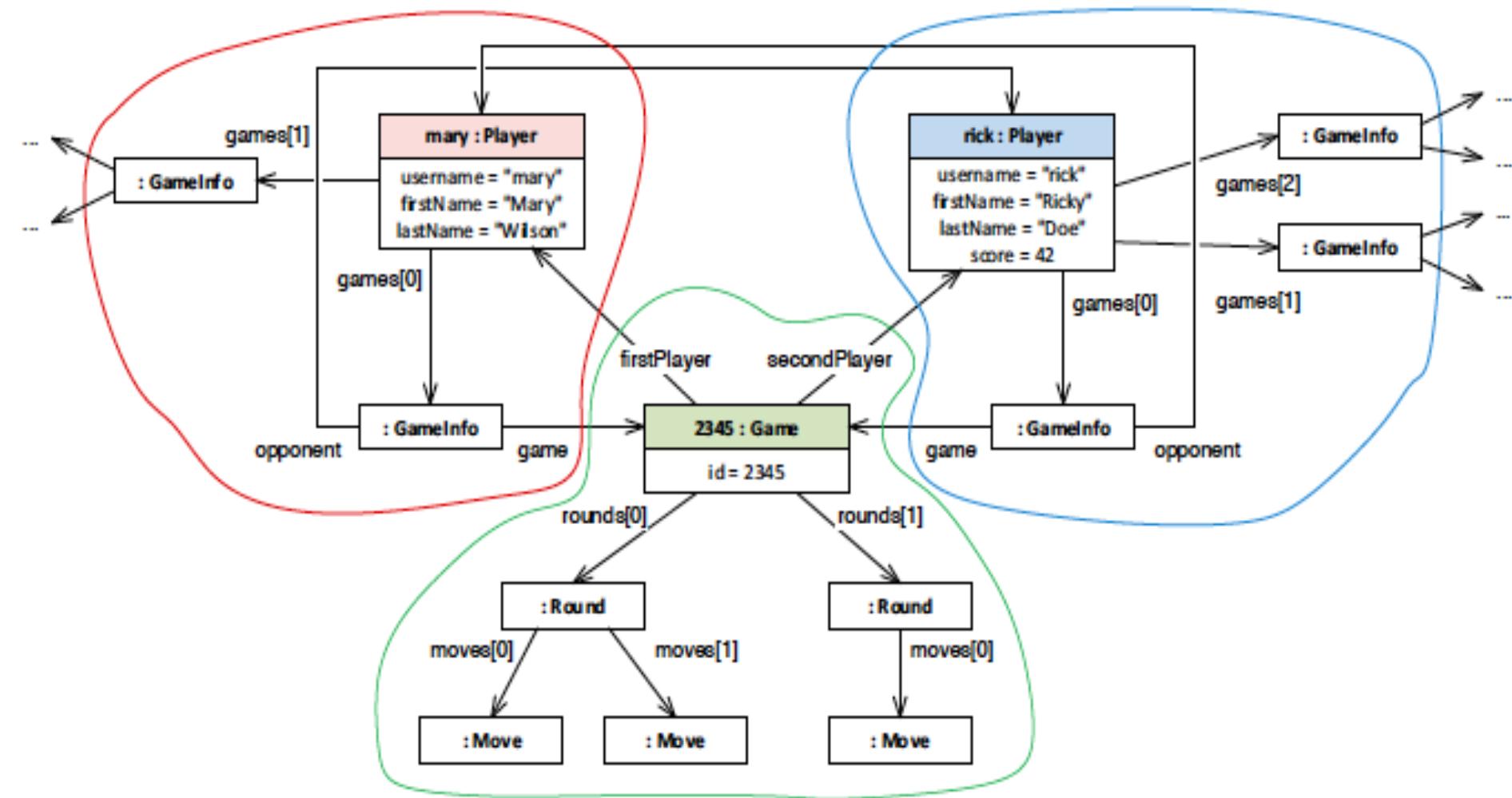
Projeto de BD

- Modelo Conceitual
 - Lógico
 - modelo de BD
 - Design de Agregados
 - Particionamento de Agregados
 - Físico
 - Implementação
 - Modelo específico do NoSQL de destino
- 
- Simplifica a implementação p/ NoSQL

Projeto de BD

- Particionamento de Agregados
 - Se o agregado for **pequeno**, ou todos ou a maioria dos seus dados são **acessados ou modificados juntamente**, isso deve ser uma **única entrada**.
 - Um agregado deve ser **particionado** em varias entradas se ele for **grande** e houver operações que frequentemente **acessam ou modificam somente partes específicas** do agregado.
 - Dois ou mais elementos de dados devem pertencer a mesma entrada se eles são frequentemente acessados ou modificados juntos.

Projeto::Exemplo



Projeto::Agregados

```
Player:mary : (
    username : "mary",
    firstName : "Mary",
    lastName : "Wilson",
    games : {
        ⟨ game : Game:2345, opponent : Player:rick ⟩,
        ⟨ game : Game:2611, opponent : Player:ann ⟩
    }
}

Player:rick : (
    username : "rick",
    firstName : "Ricky",
    lastName : "Doe",
    score : 42,
    games : {
        ⟨ game : Game:2345, opponent : Player:mary ⟩,
        ⟨ game : Game:7425, opponent : Player:ann ⟩,
        ⟨ game : Game:1241, opponent : Player:johnny ⟩
    }
)
```

```
Game:2345 : (
    id : "2345",
    firstPlayer : Player:mary,
    secondPlayer : Player:rick,
    rounds : {
        ⟨ moves : ..., comments : ... ⟩,
        ⟨ moves : ..., actions : ..., spell : ... ⟩
    }
)
```



Projeto::Key-Value

| <i>key (/major/key/-)</i> | <i>value</i> |
|---------------------------|---|
| <i>/Player/mary/-</i> | { <i>username</i> : "mary", <i>firstName</i> : "Mary", ... } |
| <i>/Player/rick/-</i> | { <i>username</i> : "rick", <i>firstName</i> : "Ricky", ... } |
| <i>/Game/2345/-</i> | { <i>id</i> : "2345", <i>firstPlayer</i> : "Player:mary", ... } |

- ✧ Partes do valor podem ser acessadas separadamente pela aplicação?
- ✧ Pode ser particionado?

Projeto::Key-Value::P

| <i>key</i> (/major/key/-/minor/key) | <i>value</i> | <i>key</i> (/major/key/-/minor/key) | <i>value</i> |
|-------------------------------------|--|-------------------------------------|-------------------------------|
| <i>Player/mary/-/username</i> | "mary" | <i>/Player/mary/-/username</i> | mary |
| <i>Player/mary/-/firstName</i> | "Mary" | <i>/Player/mary/-/firstName</i> | Mary |
| <i>Player/mary/-/lastName</i> | "Wilson" | <i>/Player/mary/-/lastName</i> | Wilson |
| <i>Player/mary/-/games[0]</i> | {game: "Game:2345", opponent: "Player:rick"} | <i>/Player/mary/-/games</i> | [{ ... }, { ... }] |
| <i>Player/mary/-/games[1]</i> | {game: "Game:2611", opponent: "Player:ann"} | <i>/Player/rick/-/username</i> | rick |
| <i>Player/rick/-/username</i> | "rick" | <i>/Player/rick/-/firstName</i> | Ricky |
| <i>Player/rick/-/firstName</i> | "Ricky" | <i>/Player/rick/-/lastName</i> | Doe |
| <i>Player/rick/-/lastName</i> | "Doe" | <i>/Player/rick/-/score</i> | 42 |
| <i>Player/rick/-/score</i> | 42 | <i>/Player/rick/-/games</i> | [{ ... }, { ... }, { ... }] |
| <i>Player/rick/-/games[0]</i> | {game: "Game:2345", opponent: "Player:mary"} | <i>/Game/2345/-/id</i> | 2345 |
| <i>Player/rick/-/games[1]</i> | {game: "Game:7425", opponent: "Player:ann"} | <i>/Game/2345/-/firstPlayer</i> | Player:mary |
| <i>Player/rick/-/games[2]</i> | {game: "Game:1241", opponent: "Player:johnny"} | <i>/Game/2345/-/secondPlayer</i> | Player:rick |
| <i>Game/2345/-/id</i> | 2345 | <i>/Game/2345/-/rounds</i> | [{ ... }, { ... }] |
| <i>Game/2345/-/firstPlayer</i> | "Player:mary" | | |
| <i>Game/2345/-/secondPlayer</i> | "Player:rick" | | |
| <i>Game/2345/-/rounds[0]</i> | {moves: ..., comments: ...} | | |
| <i>Game/2345/-/rounds[1]</i> | {moves: ..., actions: ..., spell: ...} | | |

Projeto::Família de Colunas

table Player

| username | firstName | lastName | score | games[0] | games[1] | games[2] |
|----------|-----------|----------|-------|------------------------------|----------|----------|
| "mary" | "Mary" | "Wilson" | | { game: ..., opponent: ... } | { ... } | |
| "rick" | "Ricky" | "Doe" | 42 | { game: ..., opponent: ... } | { ... } | { ... } |

table Game

| id | firstPlayer | secondPlayer | rounds[0] | rounds[1] | rounds[2] |
|------|-------------|--------------|-------------------------------|--|-----------|
| 2345 | Player:mary | Player:rick | { moves: ..., comments: ... } | { moves: ..., actions: ..., spell: ... } | |

Projeto::Documento

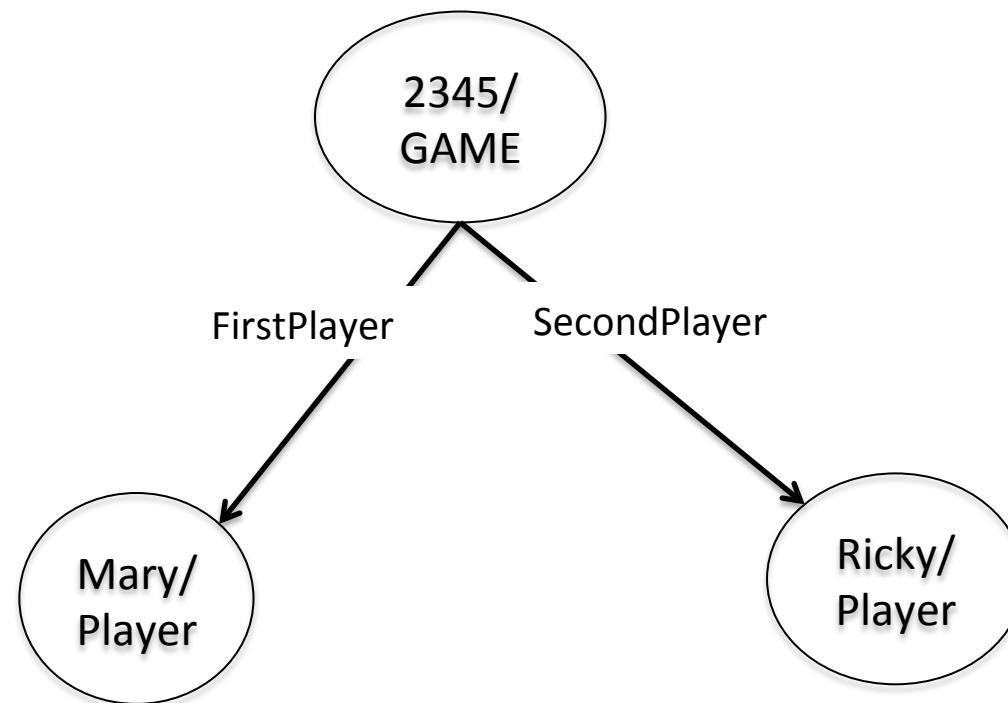
collection Player

| <i>id</i> | <i>document</i> |
|-------------|--|
| <i>mary</i> | { <i>_id</i> : "mary", <i>username</i> : "mary", <i>firstName</i> : "Mary", <i>lastName</i> : "Wilson", <i>games</i> : [{ <i>game</i> : "Game:2345", <i>opponent</i> : "Player:rick" }, { <i>game</i> : "Game:2611", <i>opponent</i> : "Player:ann" }] } |

collection Player

| <i>id</i> | <i>document</i> |
|-------------|--|
| <i>mary</i> | { <i>_id</i> : "mary", <i>username</i> : "mary", <i>firstName</i> : "Mary", <i>lastName</i> : "Wilson", <i>games[0]</i> : { <i>game</i> : "Game:2345", <i>opponent</i> : "Player:rick" }, <i>games[1]</i> : { <i>game</i> : "Game:2611", <i>opponent</i> : "Player:ann" } } |

Projeto::Grafo





Projeto!!!

Projeto em Equipe

- Escolha um domínio/problema/aplicação
- Faça o modelo conceitual (ER, classe, UML)
- Faça o mapeamento para agregados
- Faça o mapeamento de agregado para o modelo NoSQL que mais se adequa aos requisitos da aplicação/consultas (Família de Colunas ou Documento)