

TITAN:DB



UM BANCO DE DADOS TRANSACIONAL ORIENTADO A GRAFOS

ANTONIO CAVALCANTE DE PAULA FILHO

GUILHERME MOREIRA DOS REIS

LUCIANO BONFIM DE AZEVEDO

TÓPICOS

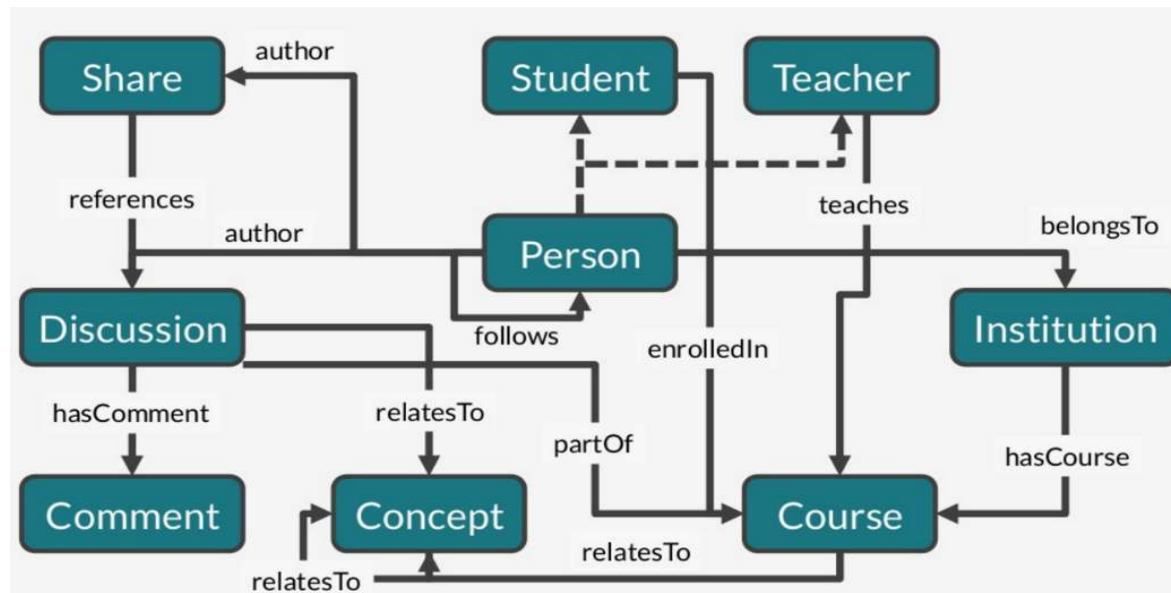


- HISTÓRICO
- CAPACIDADE E BENEFÍCIOS
- TITAN E O TEOREMA CAP
- ARQUITETURA
- INSTALAÇÃO
- GREMLIN
- GREMLIN VS SQL

- TITAN SERVER
- CONFIGURAÇÕES
- ÍNDICES
- TRANSAÇÕES
- CACHE
- LOG DE TRANSAÇÃO
- PARTICIONAMENTO DE GRAFO
- CRUD

HISTÓRICO

- DESENVOLVIDO PELA STARTUP THINKAURELIUS SOB A LICENÇA APACHE 2 EM 2011
- MISSÃO DE APOIAR E MELHORAR OS SEUS SERVIÇOS DE CURSOS ONLINE DA PEARSON
- RESPONSABILIDADE DE REPRESENTAR TODAS AS UNIVERSIDADES, ESTUDANTES, CURSOS, ETC



TIME DE CRIAÇÃO DO TITAN:DB

- COLABORADORES:

- MATTHIAS BROECHELER: DESENVOLVEDOR LÍDER E TEÓRICO.
- DAN LAROCQUE: DESENVOLVEDOR LÍDER COM FOCO ESPECIAL NOS ADAPTADORES DE ARMAZENAMENTO DE BACKEND E IMPLANTAÇÃO EM AMBIENTES DE NUVEM.
- MARKO A. RODRIGUEZ: SUPORTE AO TINKERPOP, TESTES E DOCUMENTAÇÃO.
- PAVEL YASKEVICH: DESENVOLVEDOR COM GRANDES CONTRIBUIÇÕES COM ADAPTADORES BACKEND DO TITAN, CASSANDRA E HBASE.
- STEPHEN MALLETTE: SUPORTE AO TINKERPOP, TESTES E DOCUMENTAÇÃO.
- DANIEL KUPPITZ: TESTES E FEEDBACK GERAL.
- KETRINA YIM: CRIADOR DO LOGO DO TITAN.

- ORGANIZAÇÕES:

- AURELIUS: EQUIPE DE DESENVOLVIMENTO E MANUTENÇÃO DA BASE DE CÓDIGO DO TITAN.
- PEARSON: DESENVOLVIMENTO DE CASO DE USO, FEEDBACK DOS CLIENTES E PRIMEIRO A USAR A TECNOLOGIA.
- INTEL: DESENVOLVIMENTO DE RECURSOS INDIVIDUALMENTE.
- DIGITAL REASONING: DESENVOLVIMENTO DE RECURSOS INDIVIDUALMENTE E TESTE DE CASO DE USO.

HISTÓRICO

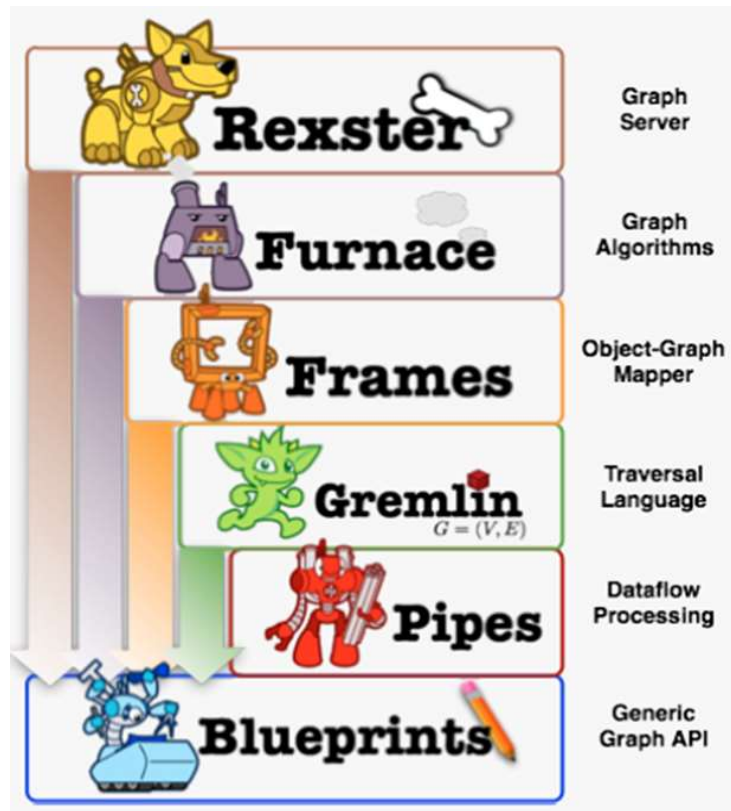
- FEVEREIRO DE 2015 O TITAN É ADQUIRIDO PELA DATASTAX
- INPIRAÇÃO PARA O DSE (DATASTAX ENTERPRISE) LANÇADO EM 12 DE ABRIL DE 2016
- O TITAN É USADO POR: MAGAZINELUIZA, CISCO, PAYPAL, AMAZON ENTRE OUTROS

CAPACIDADE E BENEFÍCIOS

- ALTAMENTE ESCALÁVEL
- É CAPAZ DE LIDAR COM APLICAÇÕES COM MAIS DE 100 BILHÕES DE NÓS E DEZENAS DE MILHARES DE USUÁRIOS CONCORRENTES EXECUTANDO CONSULTAS COMPLEXAS
- SUPORTE À MÚLTIPLAS TRANSAÇÕES SIMULTÂNEAS E PROCESSAMENTO OPERACIONAL
- PARTICIONAMENTO E DISTRIBUIÇÃO DE DADOS EM CLUSTERS, UTILIZA REDUNDÂNCIA DE DADOS

CAPACIDADE E BENEFÍCIOS

- COMPATÍVEL COM TINKERPOP, UM FRAMEWORK PARA TRABALHAR COM GRAFOS





CAPACIDADE E BENEFÍCIOS

- DISTRIBUÍDO COM 3 FERRAMENTAS DE APOIO PARA ARMAZENAMENTO:
 - CASSANDRA
 - HBASE
 - BERKELEYDB
- E DUAS DE INDEXAÇÃO:
 - ELASTICSEARCH
 - LUCENE

OBS. ENTRE O TITAN E OS DISCOS EXISTEM UM OU MAIS ADAPTADORES DE ARMAZENAMENTO E INDEXAÇÃO



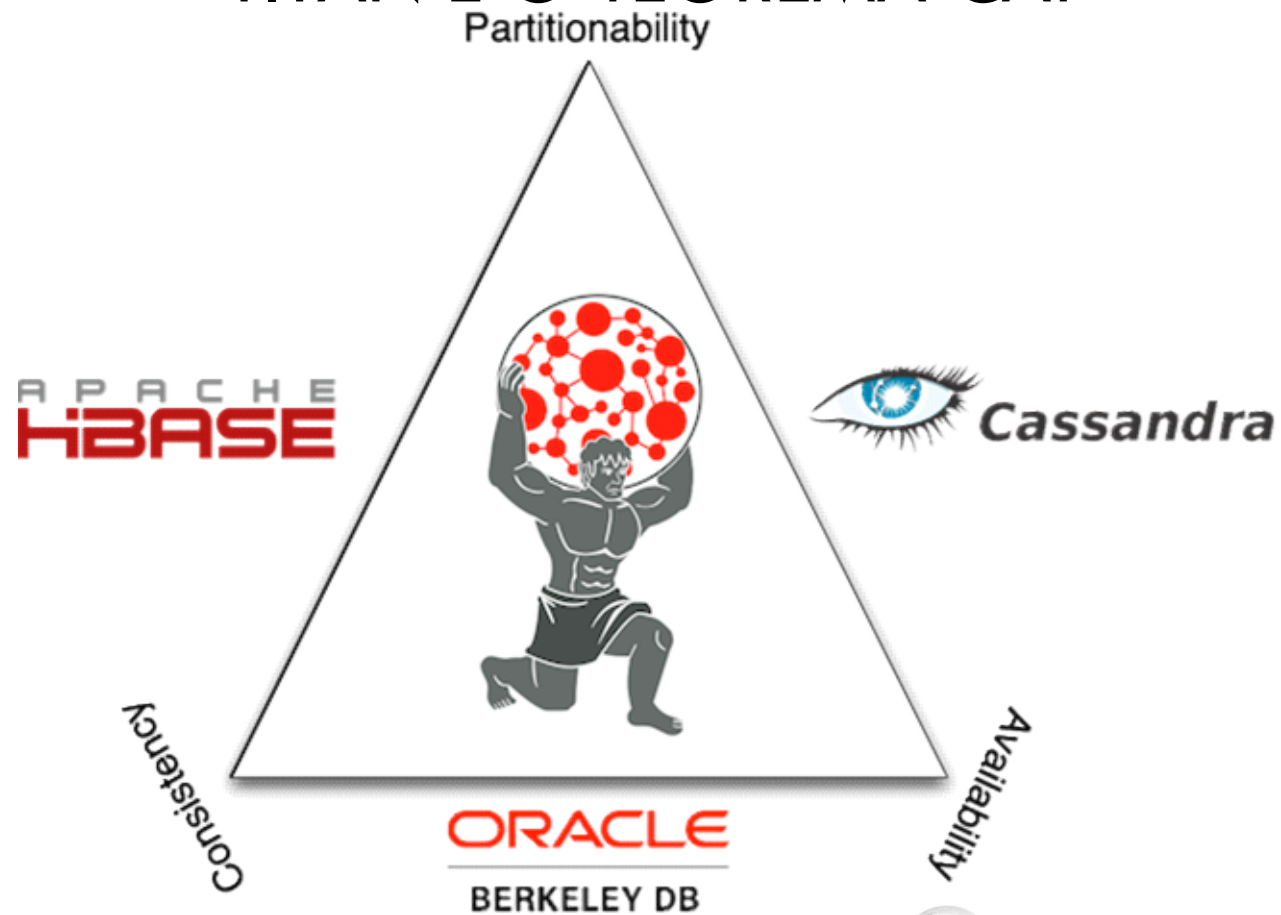
BENEFÍCIOS DO TITAN COM O CASSANDRA

- SEM GARGALOS DE READ / WRITE, NÃO UTILIZA ARQUITETURA MASTER / SLAVE
- CONTINUAMENTE DISPONÍVEL, NÃO HÁ PONTOS ÚNICOS DE FALHA
- ESCALABILIDADE ELÁSTICA PERMITINDO A INTRODUÇÃO E REMOÇÃO DE MÁQUINAS
- CAMADA DE CACHE GARANTE QUE OS DADOS CONTINUAMENTE ACESSADOS ESTEJAM DISPONÍVEIS NA MEMÓRIA
- AUMENTAR O TAMANHO DO CACHE, ADICIONANDO MAIS MÁQUINAS AO CLUSTER
- INTEGRAÇÃO COM O HADOOP
- OPEN SOURCE SOB A LICENÇA APACHE 2.

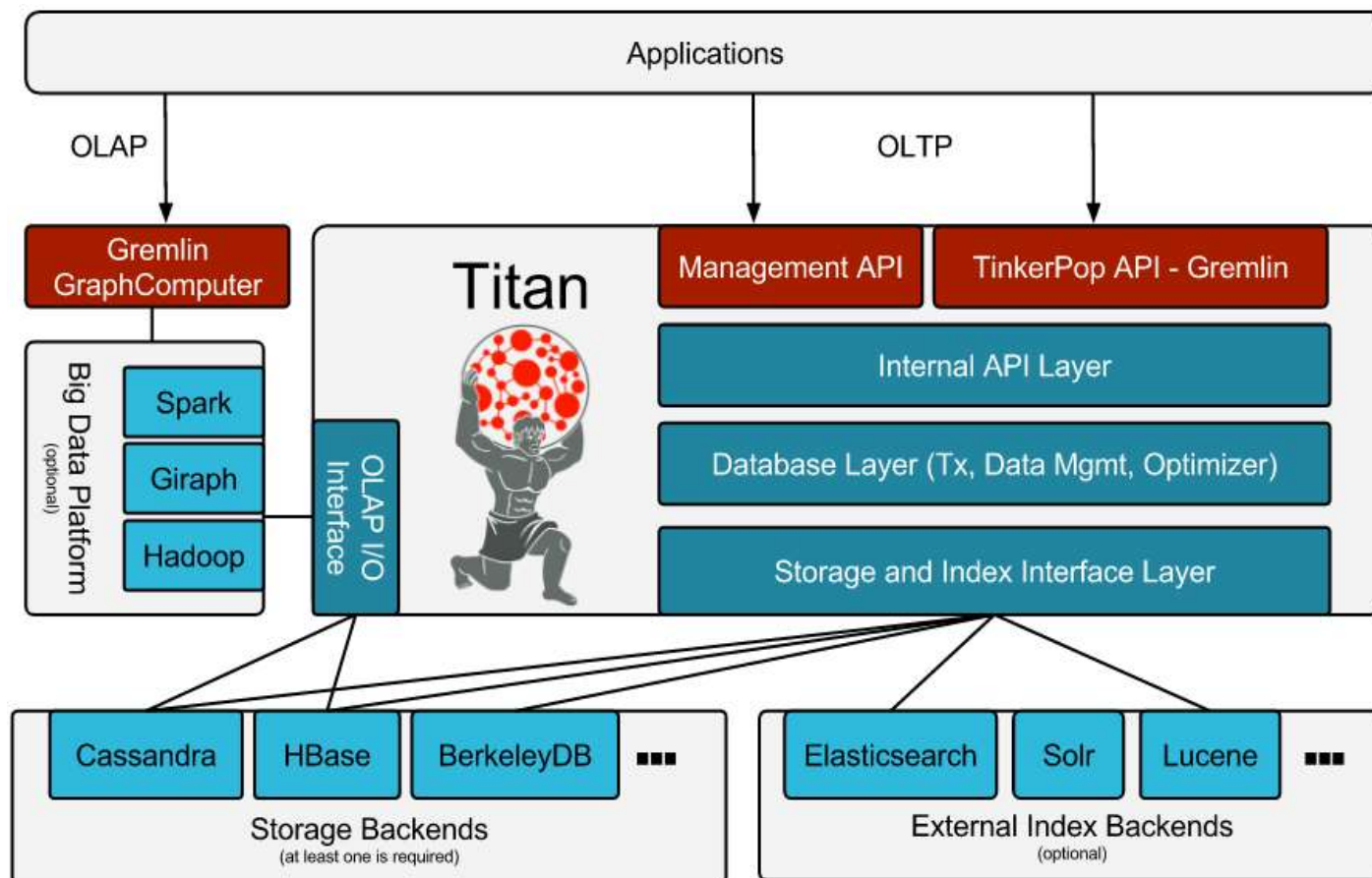
BENEFÍCIOS DO TITAN COM O HBASE

- A FORTE INTEGRAÇÃO COM O ECOSSISTEMA HADOOP
- CONSISTÊNCIA FORTE
- ESCALABILIDADE LINEAR COM A ADIÇÃO DE MAIS MÁQUINAS
- LEITURAS E ESCRITAS RIGOROSAMENTE CONSISTENTES
- CLASSES CONVENIENTES PARA APOIAR OS TRABALHOS HADOOP MAPREDUCE COM TABELAS DO HBASE
- SUPORTE PARA EXPORTAR MÉTRICAS VIA JMX (EXTENSÃO DE GERENCIAMENTO JAVA)
 - FERRAMENTAS PARA GERENCIAMENTO DE MONITORAMENTO DE APLICAÇÕES, OBJETOS DE SISTEMA, DISPOSITIVOS (E.G. IMPRESSORAS) E REDES ORIENTADAS A SERVIÇO
- OPEN SOURCE SOB A LICENÇA APACHE 2

TITAN E O TEOREMA CAP

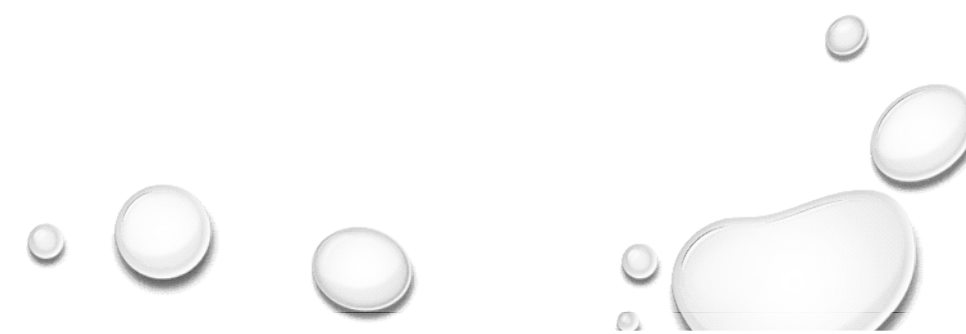


ARQUITETURA

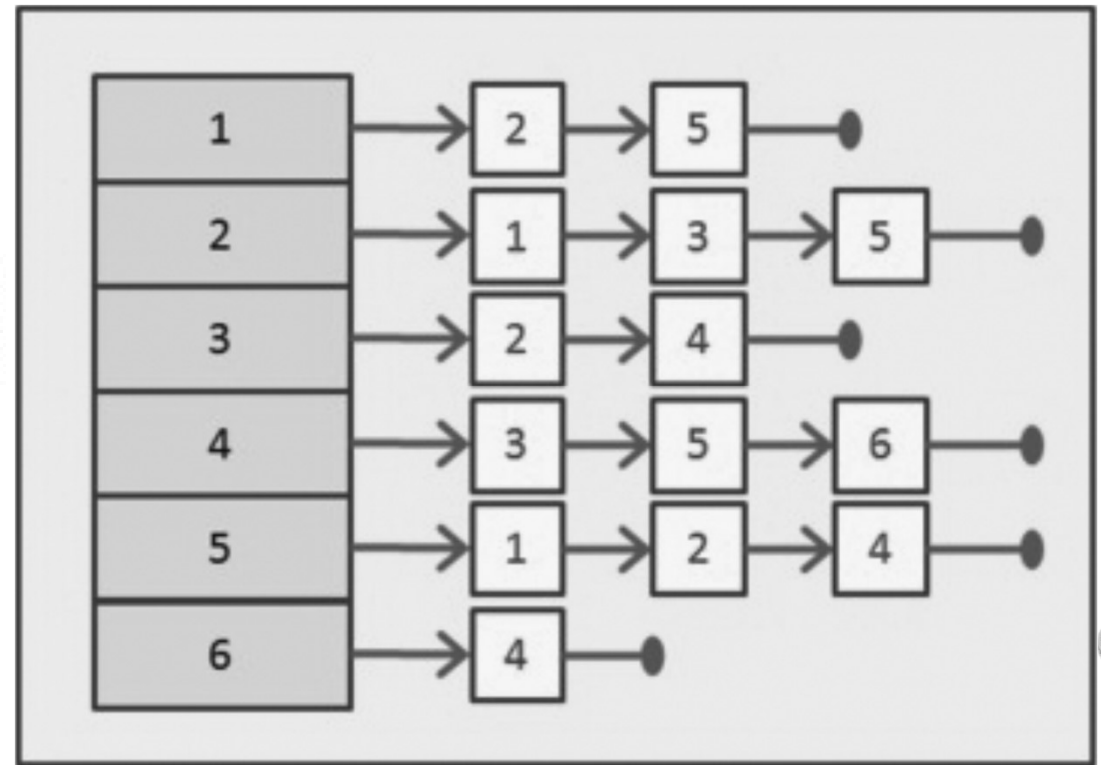
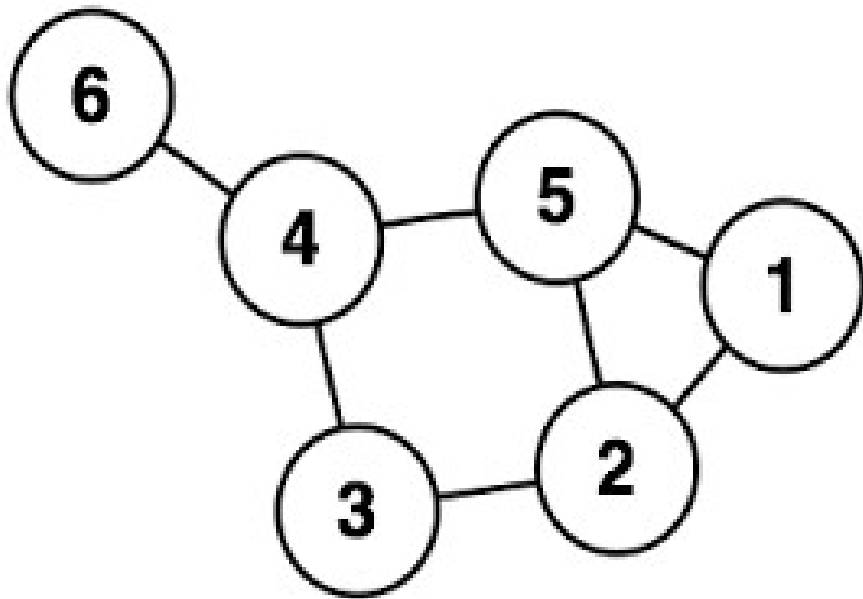




ARQUITETURA

- FOCADO EM SERIALIZAÇÃO COMPACTA DE GRAFOS, MODELAGEM DE DADOS EM GRAFOS E EXECUÇÃO DE CONSULTA EFICIENTE
 - O FOCO DO TITAN NÃO ESTÁ EM COISAS COMO: REPLICAÇÃO, BACKUP E SNAP SHOTS, PORQUE TUDO ISSO É TRATADO PELO STORAGEBACKEND (CASSANDRA, HBASE...) E INDEXBACKEND.
 - OS DADOS DOS GRAFOS NO SÃO ARMAZENADOS EM LISTA DE ADJACÊNCIA
 - DADOS SÃO ARMAZENADOS COMO UMA COLEÇÃO DE VÉRTICES JUNTO COM A LISTA DE ADJACÊNCIA DE CADA VÉRTICE
 - A LISTA DE ADJACÊNCIA DE UM VÉRTICE CONTÉM TODAS AS ARESTAS INCIDENTES NO VÉRTICE (E AS PROPRIEDADES)
- 

ARQUITETURA



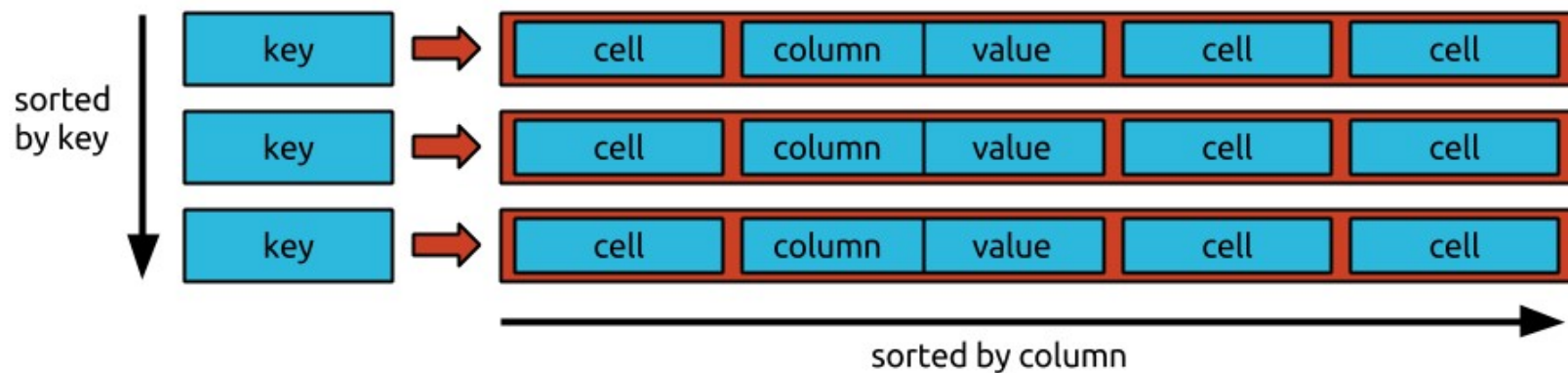
ARQUITETURA

- COM ISSO O TITAN GARANTE QUE TODAS AS ARESTAS INCIDENTES EM UM VÉRTICE E SUAS PROPRIEDADES SEJAM ARMAZENADAS DE FORMA COMPACTA NO BACKEND DE ARMAZENAMENTO
- ELE ARMAZENA A LISTA DE ADJACÊNCIA NO FORMATO BIG TABLE
 - A TABELA É UMA COLEÇÃO DE LINHAS, CADA LINHA IDENTIFICADA POR UMA CHAVE, CADA LINHA FORMADA POR UM NÚMERO ARBITRÁRIO (GRANDE, MAS LIMITADO) DE CÉLULAS
 - UMA CÉLULA É COMPOSTA POR UMA COLUNA E O SEU VALOR
 - CADA CÉLULA É IDENTIFICADA DE FORMA ÚNICA POR UMA COLUNA DENTRO DE UMA DADA LINHA

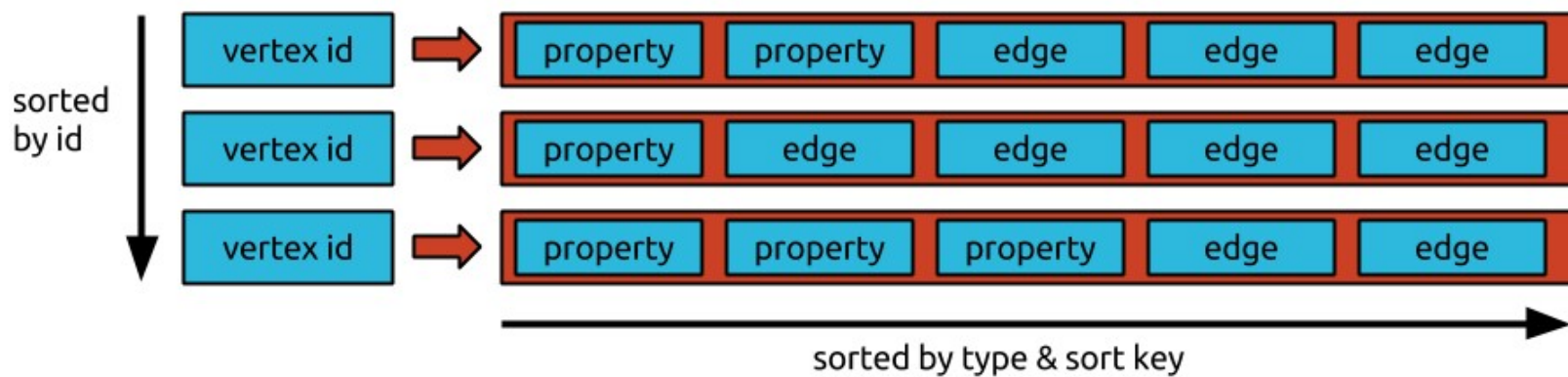
OBS: O TITAN TEM UMA EXIGÊNCIA ADICIONAL PARA O MODELO DE DADOS BIGTABLE, AS CÉLULAS DEVEM SER CLASSIFICADAS POR SUAS COLUNAS.

ARQUITETURA

Modelo big table de dados



Modelo Titan:DB de dados



INSTALAÇÃO

- REQUER O JAVA 8
- A VERSÃO TITAN-1.0.0-HADOOP1.ZIP É A RECOMENDADA SEGUNDO O SITE OFICIAL
- DOWNLOAD NO LINK:

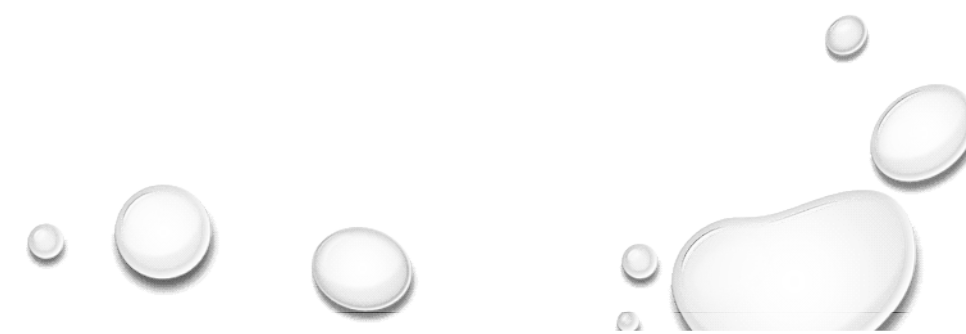
[HTTPS://GITHUB.COM/THINKAURELIUS/TITAN/WIKI/DOWNLOADS](https://github.com/ThinkAurelius/titan/wiki/downloads)

DEPOIS É SÓ DESCOMPACTAR E CONFIGURAR AS VARIÁVEIS DE AMBIENTE

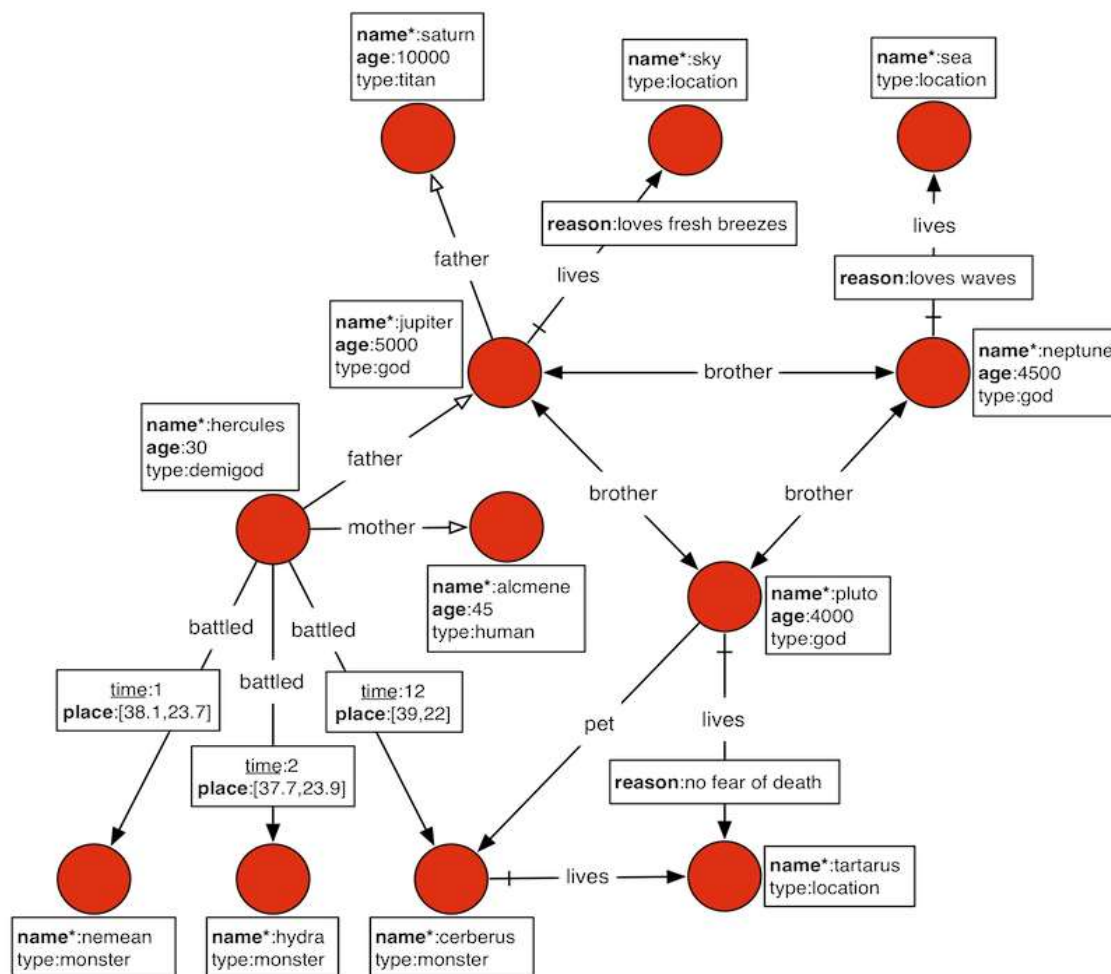
TITAN_HOME, CASSANDRA_HOME, ETC.



GREMLIN

- DSL (LINGUAGEM DE DOMÍNIO ESPECÍFICO)
 - ORIENTADA A CAMINHO
 - UMA LINGUAGEM ESPECIALIZADA EM TRANSITAR EM GRAFOS
 - CONSTRUÍDA EM GROOVY
 - INDEPENDENTE DO TITAN
 - E SUPORTA A MAIORIA DOS BANCOS DE DADOS ORIENTADO GRAFO
 - O USO DO GREMLIN, EVITA FICAR PRESO A UM FABRICANTE, REAPROVEITANDO SCRIPTS E RECOMENDAÇÕES QUE JÁ EXISTAM
- 

GRAFO DOS DEUSES



USANDO O GREMLIN

NO GNU/LINUX

`./bin/gremlin.sh`

OU

NO WINDOWS

`.\bin\gremlin.bat`

PARA SAIR DO GREMLIN

`ctrl d`

USANDO O GREMLIN

//instancia o grafo passando a configuração do berkeleydb como parâmetro.

```
gremlin> graph = TitanFactory.open('conf/titan-berkeleyje.properties')
```

```
==>standardtitangraph[berkeleyje:../db/berkeley]
```

//cria o grafo recém instanciado no storagebackend

```
gremlin> GraphOfTheGodsFactory.load(graph)
```

```
==>null
```

//visita exatamente uma vez cada vértice no grafo, verificando e/ou atualizando cada vértice

```
gremlin> g = graph.traversal()
```

```
==>graphtraversalsource[standardtitangraph[berkeleyje:../db/berkeley], standard]
```

USANDO O GREMLIN

//retorna a quantidade de vértices

```
gremlin> g.V( ).count( )
```

```
==>12
```

//retorna a quantidade de arestas

```
gremlin> g.E( ).count()
```

```
==>17
```

//Retorna o índice de saturn

```
gremlin> saturn = g.V().has('name', 'saturn').next()
```

```
==>v[256]
```

//Retorna as propriedades do vértice saturn

```
gremlin> g.V(saturn).valueMap()
```

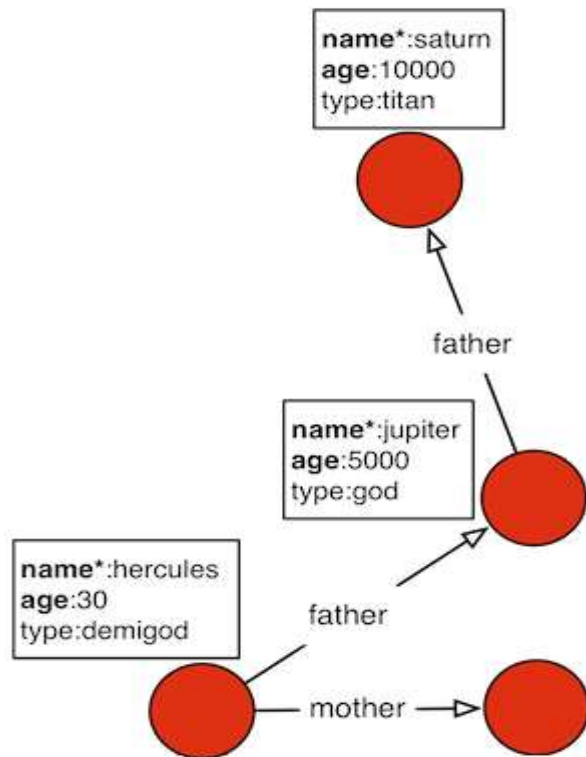
```
==>[name:[saturn], age:[1 0000]]
```

USANDO O GREMLIN

//Retorna o neto de saturn

```
gremlin>g.V(saturn).in('father').in('father').values('name')
```

=>hercules



GREMLIN

- A PROPRIEDADE PLACE É TAMBÉM UM ÍNDICE NO GRAFO, ELA É UMA PROPRIEDADE DE ARESTA
- É POSSÍVEL BUSCAR NO GRAFO DOS DEUSES POR TODOS OS EVENTOS QUE ACONTECERAM DENTRO DE 50KM DE ATENAS, LATITUDE 37.97 E LONGITUDE 23.72

//Retorna os vértices envolvidos nesses eventos

```
gremlin> g.E().has('place', geowithin(geoshape.circle(37.97, 23.72, 50)))
```

```
==>e[6qm-9o0-9hx-36w][12528-battled->4136]
```

```
==>e[74u-9o0-9hx-3bc][12528-battled->4296]
```

```
gremlin> g.V(4136).valueMap()
```

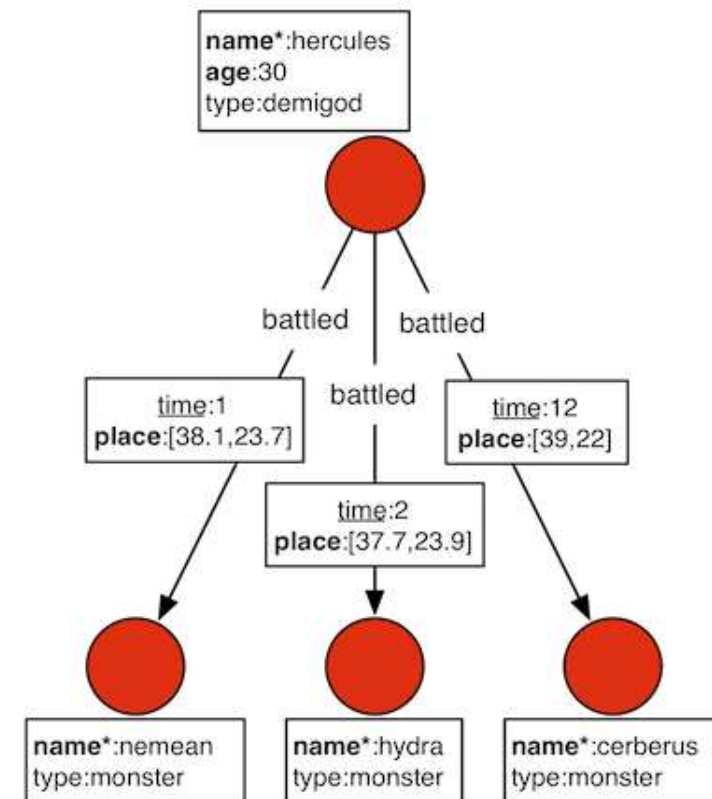
```
==>[name:[nemean]]
```

```
gremlin> g.V(4296).valueMap()
```

```
==>[name:[hydra]]
```

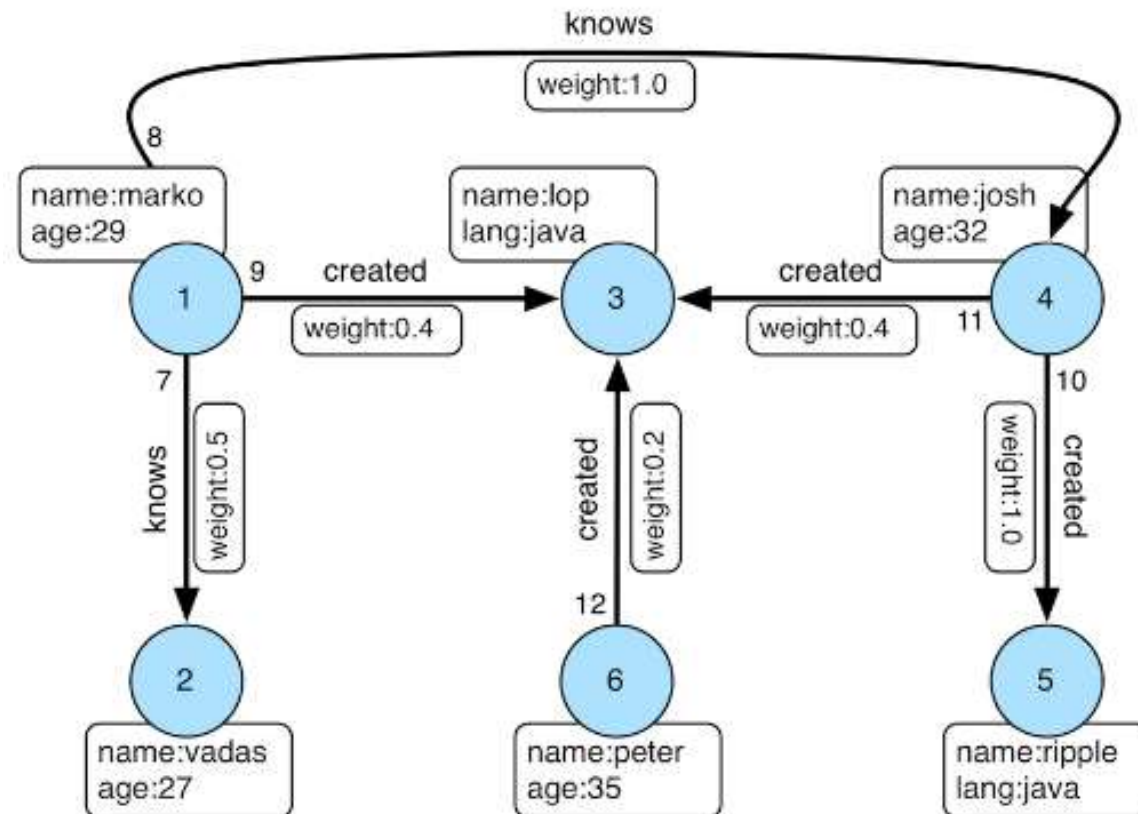
```
gremlin> g.V(12528).valueMap()
```

```
==>[name:[hercules], age:[30]]
```



GREMLIN VS SQL

- ABRINDO E CARREGANDO O GRAFO CLÁSSICO QUE VEM COM O TINKERPOP, VEJA O DIAGRAMA:



GREMLIN

//Cria o grafo padrão do tinkerpops

```
gremlin> g = TinkerFactory.createClassic()
```

```
==>tinkergraph[vertices:6 edges:6]
```

```
gremlin> g = g.traversal( )
```

```
==>graphtraversalsource[tinkergraph[vertices:6 edges:6], standard]
```

```
gremlin> g.V( )
```

```
==>V[1]
```

```
==>V[2]
```

```
==>V[3]
```

```
==>V[4]
```

```
==>V[5]
```

```
==>V[6]
```

GREMLIN

```
gremlin> g.V().valueMap()
```

```
==>[name:[marko], age:[29]]
```

```
==>[name:[vadas], age:[27]]
```

```
==>[name:[lop], lang:[java]]
```

```
==>[name:[josh], age:[32]]
```

```
==>[name:[ripple], lang:[java]]
```

```
==>[name:[peter], age:[35]]
```

```
//removendo o vértice de índice 1 (marko)
```

```
gremlin> grafo.V(1).drop()
```

```
//adiciona de volta o vértice removido
```

```
gremlin> g.addVertex(T.label, "person", T.id, 1, "name", "marko", "age", 29)
```

SQL VS GREMLIN

QUERY	SQL	GREMLIN
Todos os usuários	<pre>select * from tbl</pre>	<pre>g.V().hasLabel().valueMap()</pre>
O nome de todos os usuários	<pre>select name from tbl_users</pre>	<pre>g.V().valueMap('name')</pre>
Nome e idade de todos	<pre>select name, age from users</pre>	<pre>g.V().valueMap('name', 'age')</pre>
Com distinct	<pre>select distinct (lang) From tbl_users</pre>	<pre>g.V().values('lang').dedup()</pre>
Usuário mais velho	<pre>select max(age) from users</pre>	<pre>g.V().values('age').max()</pre>
Select por igualdade	<pre>select * from users where age = 35</pre>	<pre>g.V().has('age', 35).valueMap()</pre>
Select por comparação	<pre>select * from users where age > 21</pre>	<pre>g.V().has('age', gt(21)).valueMap()</pre>

TITAN SERVER

INICIANDO O TITAN

```
$ ./bin/titan.sh start
```

```
Forking Cassandra...
```

```
Running `nodetool statusthrift`.. OK (returned exit status 0 and printed string "running").
```

```
Forking Elasticsearch...
```

```
Connecting to Elasticsearch (127.0.0.1:9300)... OK (connected to 127.0.0.1:9300).
```

```
Forking Gremlin-Server...
```

```
Connecting to Gremlin-Server (127.0.0.1:8182)... OK (connected to 127.0.0.1:8182).
```

```
Run gremlin.sh to connect
```

CONECTANDO COM O GREMLIN SERVER

- O SERVIDOR GREMLIN ESTARÁ PRONTO PARA ESCUTAR CONEXÕES WEBSOCKET QUANDO INICIADO
- UMA FORMA FÁCIL DE TESTAR A CONEXÃO É COM O CONSOLE DO GREMLIN
- O COMANDO `./bin/gremlin.sh` INICIA O CONSOLE GREMLIN
- `:remote` CONECTA, DIZ PARA O CONSOLE CONFIGURAR UMA CONEXÃO REMOTA COM O GREMLIN SERVER USANDO O ARQUIVO `conf/remote.yaml`, ESSE ARQUIVO APONTA PARA UMA INSTÂNCIA DO GREMLIN SENDO EXECUTADA EM `localhost`
- `:>` QUE É O SUBMIT.

```
$ bin/gremlin.sh
```

```
  \___/
```

```
(o o)
```

```
-----oOOo-(3)-oOOo-----
```

```
plugin activated: tinkerpop.server
```

```
plugin activated: tinkerpop.hadoop
```

```
plugin activated: tinkerpop.utilities
```

```
plugin activated: aurelius.titan
```

```
plugin activated: tinkerpop.tinkergraph
```

```
gremlin> :remote connect tinkerpop.server conf/remote.yaml
```

```
==>Connected - localhost/127.0.0.1:8182
```

```
gremlin> :> graph.addVertex("name", "stephen")
```

```
==>v[256]
```

```
gremlin> :> g.V().values('name')
```

```
==>stephen
```

CONFIGURAÇÕES

- TITAN COM CASSANDRA (SERVIDOR LOCAL)

- ELASTIC SEARCH (REMOTO)

`storage.backend=cassandra`

`storage.hostname=localhost`

`index.search.backend=elasticsearch`

`index.search.hostname=100.100.101.1, 100.100.101.2`

`index.search.elasticsearch.client-only=true`

`graph = TitanFactory.open("conf/titan-cassandra.properties")`

- TITAN COM CASSANDRA (SERVIDOR REMOTO)

`TitanGraph graph = TitanFactory.build().`

`set("storage.backend", "cassandra").`

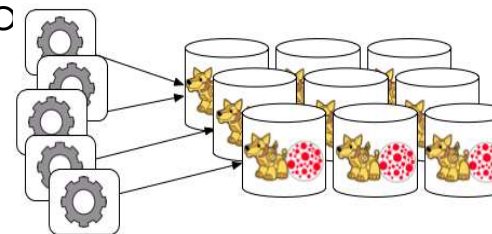
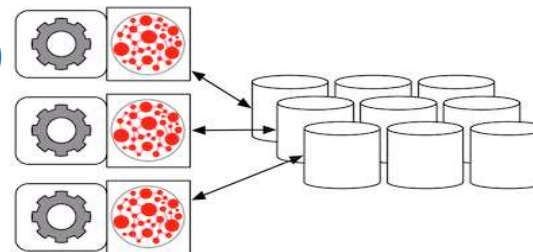
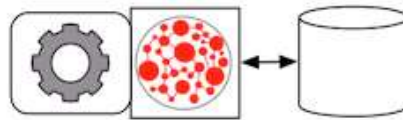
`set("storage.hostname", "77.77.77.77, 77.77.77.78")`

`open()`

- TITAN NO MODO EMBARCADO COM CASSANDRA

O ARQUIVO `cassandra.yaml` É CONFIGURADO USANDO A OPÇÃO
COMO UMA URL COMPLETA

`storage.conf-file = file:///home/cassandra.yaml`



CONFIGURA O ARQUIVO YAML

CONFIGURAÇÕES

- TITAN NO MODO EMBARCADO COM CASSANDRA

O ARQUIVO `cassandra.yaml` É CONFIGURADO USANDO A OPÇÃO `STORAGE.CONF-FILE`, QUE ESPECIFICA O ARQUIVO YAML COMO UMA URL COMPLETA

`storage.conf-file = file:///home/cassandra.yaml`

- TITAN COM HBASE (SERVIDOR LOCAL)

`storage.backend=hbase`

`storage.hostname=100.100.101.1`

`storage.port=2181`

`cache.db-cache = true`

`cache.db-cache-clean-wait = 20`

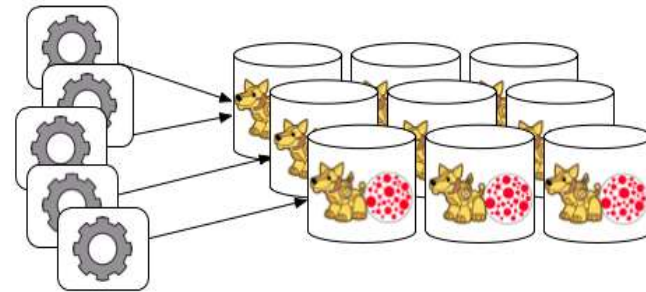
`cache.db-cache-time = 180000`

`cache.db-cache-size = 0.5`

- HBASE PODE SER EXECUTADO COMO UM BANCO DE DADOS AUTÔNOMO NO HOST LOCAL

Para iniciar o HBase é usado:

`./bin/start-hbase.sh`



CONFIGURAÇÕES

- HBASE PODE SER EXECUTADO COMO UM BANCO DE DADOS AUTÔNOMO NO HOST LOCAL

Para iniciar o HBase é usado:

```
./bin/start-hbase.sh
```

Para parar o HBase é usado:

```
stop-hbase.sh
```

Com o HBase rodando, agora é possível criar um grafo Titan com HBase:

```
TitanGraph graph = TitanFactory.build()  
    .set("storage.backend", "hbase")  
    .open();
```

Note que não precisou especificar um nome de host, a conexão localhost é tentada por padrão.

Para usar de forma escalar, o Titan com HBase em modo Remote Server, é usado:

```
TitanGraph g = TitanFactory.build()  
    .set("storage.backend", "hbase")  
    .set("storage.hostname", "77.77.77.77, 77.77.77.78, 77.77.77.79")  
    .open();
```


CONFIGURAÇÕES

Em modo Remote Serve com o Gremlin Server:

<http://rexster.titan.machine1/mygraph/vertices/1>

[http://rexster.titan.machine2/mygraph/tp/gremlin?script=g.v\(1\).out\('follows'\).out\('created'\)](http://rexster.titan.machine2/mygraph/tp/gremlin?script=g.v(1).out('follows').out('created'))

O fragmento abaixo mostra a configuração do servidor Gremlin:

...

```
graphs: {  
  g: conf/titan-hbase.properties}
```

```
plugins:  
  - aurelius.titan
```

...

- **TITAN COM O BERKELEYDB**

O Berkeley DB é executado no mesmo JVM com o Titan, por isso requer apenas uma configuração simples:

```
TitanGraph g = TitanFactory.build().  
set("storage.backend", "berkeleyje").  
set("storage.directory", "/tmp/graph").  
open();
```

CONFIGURAÇÕES

- SERVIDOR TITAN

Acesso remoto através de um cliente

- Configuração é feita através do arquivo de configuração do GremlinServer

...

```
graphs: {  
  graph: conf/titan-berkeleyje.properties  
}
```

plugins:

- aurelius.titan

...

- TITAN TEM DUAS OPÇÕES DE CONFIGURAÇÕES DISTINTAS, A LOCAL E A GLOBAL

- Local é aplicada a uma instância individual do Titan
- Global é aplicada a todas as instâncias em um cluster

CONFIGURAÇÕES

- E CINCO ESCOPOS DE OPÇÕES DE CONFIGURAÇÃO:

- LOCAL: aplicada a uma instância individual do Titan e são especificadas na configuração fornecida na inicialização da instância do Titan;
- MASKABLE: essa opção de configuração pode ser sobrescrita por uma instância individual do Titan pelo arquivo de configuração local. Se o arquivo de configuração local não especificar a opção, seu valor é lido da configuração global do cluster Titan;
- GLOBAL: essas opções são sempre lidas da configuração do cluster e não podem ser sobrescrita por uma instância básica;
- GLOBAL_OFFLINE: como a GLOBAL, mas, mudar essa opção requer a reinicialização do cluster para assegurar que o valor é o mesmo através de todo o cluster;
- FIXED: como a GLOBAL, mas, o valor não pode ser mudado depois que o cluster tiver inicializado.

MULTIPLICIDADE DE ARESTA

- **MULTI:** Permite múltiplas arestas do mesmo label entre qualquer par de vértices.
- **SIMPLE:** Permite no máximo uma aresta de determinado rótulo, entre qualquer par de vértices.
- **MANY2ONE:** Permite no máximo uma aresta de saída de um label em qualquer vértice no gráfico, mas não coloca nenhuma restrição sobre arestas de entrada.
- **ONE2MANY:** Permite no máximo uma aresta de entrada de um label em qualquer vértice no gráfico, mas não coloca nenhuma restrição sobre arestas de saída.
- **ONE2ONE:** Permite no máximo uma aresta de entrada e uma aresta de saída de tal label em qualquer vértice no gráfico.

MULTIPLICIDADE DE ARESTA

```
mgmt = graph.openManagement()  
follow = mgmt.makeEdgeLabel('follow').multiplicity(MULTI).make()  
mother = mgmt.makeEdgeLabel('mother').multiplicity(MANY2ONE).make()  
mgmt.commit( )
```

ÍNDICES

- **Graph Index:** operações de recuperação global eficientes em grafos muito grandes
 - **Composite:** rápido e eficiente, mas, limitado a casos particulares onde às combinações de propriedades chave foram definidas previamente
 - **Mixed:** usado em qualquer combinação de chaves indexadas e suporta múltiplas condições predicadas, além de depender de um sistema armazenamento de índice externo
- **vertex-centric index:** acelera o processo de percorrer o grafo, sobretudo quando a travessia será através de vértices com muitas arestas incidentes

COMPOSITE

graph.tx().rollback() //Nunca cria novos índices enquanto uma transação está ativa

mgmt = graph.openManagement()

name = mgmt.getPropertyKey('name')

age = mgmt.getPropertyKey('age')

mgmt.buildIndex('byNameComposite', Vertex.class).addKey(name).buildCompositeIndex()

mgmt.buildIndex('byNameAndAgeComposite',

Vertex.class).addKey(name).addKey(age).buildCompositeIndex()

mgmt.commit()

//Espera pelo índice para ficar disponível

mgmt.awaitGraphIndexStatus(graph, 'byNameComposite').call()

mgmt.awaitGraphIndexStatus(graph, 'byNameAndAgeComposite').call()

//Reindexa os dados existentes

mgmt = graph.openManagement()

mgmt.updateIndex(mgmt.getGraphIndex("byNameComposite"), SchemaAction.REINDEX).get()

mgmt.updateIndex(mgmt.getGraphIndex("byNameAndAgeComposite"), SchemaAction.REINDEX).get()

mgmt.commit()

ÍNDICE MIXED

- **Índices mixed** recuperam vértices ou arestas por qualquer combinação de propriedades chave previamente adicionadas. Exigem uma configuração de um backend de indexação e usa essa infraestrutura de indexação para executar operações de pesquisa

```
graph.tx().rollback() //Nunca cria novos índices enquanto uma transação está ativa
mgmt = graph.openManagement()
name = mgmt.getPropertyKey('name')
age = mgmt.getPropertyKey('age')
mgmt.buildIndex('nameAndAge',Vertex.class).
addKey(name).addKey(age).buildMixedIndex("search")
mgmt.commit()
//Espera pelo índice para ficar disponível
mgmt.awaitGraphIndexStatus(graph, 'nameAndAge').call()
//Reindexa os dados existentes
mgmt = graph.openManagement()
mgmt.updateIndex(mgmt.getGraphIndex("nameAndAge"), SchemaAction.REINDEX).get()
mgmt.commit()
```


ÍNDICE MIXED

- EMBORA A DEFINIÇÃO DO ÍNDICE DO EXEMPLO ACIMA É SEMELHANTE AO ÍNDICE COMPOSITE, ELE FORNECE MAIOR APOIO A CONSULTA E PODE RESPONDER A QUALQUER UMA DAS SEGUINTE CONSULTAS.

```
g.V().has('name', textContains('hercules')).has('age', inside(20, 50))
```

```
g.V().has('name', textContains('hercules'))
```

```
g.V().has('age', lt(50))
```

OBS: Índices mixed suportam pesquisa de texto completo, pesquisa de intervalo, pesquisa geo e outros.

COMPOSITE VS MIXED

- USE UM ÍNDICE COMPOSITE PARA RECUPERAR A COMBINAÇÃO EXATA DO ÍNDICE.
- ÍNDICES COMPOSITE NÃO REQUEREM CONFIGURAÇÃO OU UM SISTEMA DE ÍNDICE EXTERNO
- MUITAS VEZES SÃO SIGNIFICATIVAMENTE MAIS RÁPIDOS DO QUE OS ÍNDICES MIXED
- A EXCEÇÃO É UTILIZA-SE DE UM ÍNDICE MIXED PARA CORRESPONDÊNCIAS EXATAS
 - QUANDO O NÚMERO DE VALORES DISTINTOS DE RESTRIÇÃO DE CONSULTA É RELATIVAMENTE PEQUENO
 - OU QUANDO SE ESPERA QUE UM VALOR SEJA ASSOCIADO COM MUITOS ELEMENTOS NO GRAFO (ISTO É, NO CASO DE UMA BAIXA SELETIVIDADE).

ÍNDICE CENTRADO NO VÉRTICE

- ESTRUTURAS DE ÍNDICE LOCAL CONSTRUÍDAS INDIVIDUALMENTE POR VÉRTICE
- AS ARESTAS INCIDENTES TEM QUE SER RECUPERADAS E EM SEGUIDA FILTRADAS NA MEMÓRIA PARA COINCIDIR COM AS CONDIÇÕES DA TRAVESSIA
- ACELERA AS TRAVESSIAS USANDO ESTRUTURAS LOCALIZADAS DE ÍNDICE PARA RECUPERAR APENAS AS ARESTAS QUE PRECISAM SER PERCORRIDAS

Suponhamos que Hércules lutou contra centenas de monstros, além dos três capturado no Grafo introdutório dos Deuses. . Sem um índice vertex-centric, uma consulta pedindo esses monstros que lutaram entre o ponto de tempo 10 e 20, exigiria a recuperação de todas as arestas battled, mesmo que haja apenas algumas arestas correspondentes.

```
h = g.V().has('name', 'hercules').next()
```

```
g.V(h).outE('battled').has('time', inside(10, 20)).inV()
```

ÍNDICE CENTRADO NO VÉRTICE

```
graph.tx().rollback() //Never create new indexes while a transaction is active
mgmt = graph.openManagement()
time = mgmt.getPropertyKey('time')
battled = mgmt.getEdgeLabel('battled')
mgmt.buildEdgeIndex(battled, 'battlesByTime', Direction.BOTH, Order.decr, time)
mgmt.commit()

//Wait for the index to become available
mgmt.awaitGraphIndexStatus(graph, 'battlesByTime').call()

//Reindex the existing data
mgmt = graph.openManagement()
mgmt.updateIndex(mgmt.getGraphIndex("battlesByTime"), SchemaAction.REINDEX).get()
mgmt.commit()
```

TRANSAÇÕES

- CADA OPERAÇÃO EM GRAFO TITAN, OCORRE DENTRO DO CONTEXTO DE UMA TRANSAÇÃO. DE ACORDO COM A ESPECIFICAÇÃO DO BLUEPRINTS, CADA THREAD ABRE SUA PRÓPRIA TRANSAÇÃO NO BANCO DE DADOS DE GRAFO COM A PRIMEIRA OPERAÇÃO (ISTO É, A RECUPERAÇÃO OU MUTAÇÃO) NO GRAFO.

```
graph = TitanFactory.open("berkeleyje:/tmp/titan")
```

```
juno = graph.addVertex() //Automatically opens a new transaction
```

```
juno.property("name", "juno")
```

```
graph.tx().commit() //Commits transaction
```

CACHE

- TITAN EMPREGA MÚLTIPLAS CAMADAS DE ARMAZENAMENTO DE DADOS EM CACHE, PARA FACILITAR O PROCESSO TRANSITAR O GRAFO
 - TRANSACTION-LEVEL CACHING
 - **VERTEX CACHE:** CACHES DE VÉRTICES ACESSADOS E SUA LISTA DE ADJACÊNCIA
 - **INDEX CACHE:** ARMAZENA EM CACHE OS RESULTADOS DE CONSULTAS DE ÍNDICE PARA QUE AS CHAMADAS SUBSEQUENTES DO ÍNDICE POSSA ACESSAR A PARTIR DA MEMÓRIA

configurado via `cache.tx-cache-size` ou `graph.buildTransaction()` ou ainda `setVertexCacheSize (int)`

- **DATABASE LEVEL CACHE:** O DATABASE LEVEL CACHE MANTÉM LISTAS DE ADJACÊNCIA (OU SUBCONJUNTOS DOS MESMOS) EM VÁRIAS OPERAÇÕES, ALÉM DA DURAÇÃO DE UMA ÚNICA TRANSAÇÃO. O CACHE DE NÍVEL DE BANCO DE DADOS É COMPARTILHADO POR TODAS AS TRANSAÇÕES ATRAVÉS DE UM BANCO DE DADOS
- **TEMPO DE EXPIRAÇÃO DO CACHE:** A CONFIGURAÇÃO MAIS IMPORTANTE PARA O COMPORTAMENTO DE DESEMPENHO E DE CONSULTA É O TEMPO DE EXPIRAÇÃO DE CACHE QUE É CONFIGURADO ATRAVÉS `CACHE.DB-CACHE-TIME`
- **TAMANHO DO CACHE:** A OPÇÃO DE CONFIGURAÇÕES `CACHE.DB-CACHE-SIZE` CONTROLA A QUANTIDADE DE ESPAÇO NA PILHA, QUE O TITAN PODERÁ CONSUMIR PARA ARMAZENAR O CACHE. QUANTO MAIOR O CACHE, MAIS EFICAZ ELE SERÁ

CACHE

- TRANSACTION-LEVEL CACHING

- **CLEAN UP WAIT TIME:** QUANDO UM VÉRTICE É MODIFICADO LOCALMENTE (POR EXEMPLO, UMA ARESTA É ADICIONADA) TODAS AS ENTRADAS DE CACHE DE NÍVEL DE BANCO DE DADOS RELACIONADAS A ESSE VÉRTICE SÃO MARCADAS COMO EXPIRADAS E EVENTUALMENTE DESPEJADOS, CONFIGURANDO CACHE.DB-CACHE-CLEAN-WAIT, O CACHE IRÁ ESPERAR PELO MENOS UM NÚMERO DE MILISSEGUNDOS ANTES DE PREENCHER O CACHE COM A ENTRADA RECUPERADA DO BACKEND DE ARMAZENAMENTO
- **STORAGE BACKEND CACHING:** CADA BACKEND DE ARMAZENAMENTO MANTÉM A SUA PRÓPRIA CAMADA DE CACHE DE DADOS. ESTES CACHES SE BENEFICIAM DE COMPRESSÃO, COMPACTAÇÃO DE DADOS, EXPIRAÇÃO COORDENADA E MUITAS VEZES SÃO MANTIDOS FORA HEAP, O QUE SIGNIFICA QUE GRANDES CACHES PODEM SER USADOS SEM SE PREOCUPAR COM QUESTÕES DE COLETA DE LIXO DE MEMÓRIA

LOG DE TRANSAÇÃO

- O TITAN PODE AUTOMATICAMENTE GUARDAR AS ALTERAÇÕES TRANSACIONAIS PARA PROCESSAMENTO ADICIONAL OU COMO UM REGISTRO DE MUDANÇA EM UM LOG
- O TITAN PODE AUTOMATICAMENTE GUARDAR AS ALTERAÇÕES TRANSACIONAIS PARA PROCESSAMENTO ADICIONAL OU COMO UM REGISTRO DE MUDANÇA EM UM LOG

```
tx = graph.buildTransaction().logIdentifier('addedPerson').start()  
u = tx.addVertex(label, 'human')  
u.property('name', 'proteros')  
u.property('age', 36)  
tx.commit()
```

- APÓS O COMMIT, TODAS AS ALTERAÇÕES FEITAS DURANTE A TRANSAÇÃO SÃO REGISTRADAS NO SISTEMA DE REGISTRO DE USUÁRIO EM UM LOG CHAMADO ADDEDPERSON

LOG DE TRANSAÇÃO

- **REGISTRO DE MUDANÇA:** O LOG DE TRANSAÇÕES DO USUÁRIO PODE SER USADO PARA MANTER UM REGISTRO DE TODAS AS ALTERAÇÕES FEITAS EM UM GRAFO. AO USAR IDENTIFICADORES DE LOG SEPARADOS, AS MUDANÇAS PODEM SER GRAVADAS EM DIFERENTES REGISTOS PARA DISTINGUIR TIPOS DE TRANSAÇÕES SEPARADAS
- **ATUALIZAÇÕES DOWNSTREAM:** MUITAS VEZES O CASO DE QUANDO UM CLUSTER DE UM GRAFO TITAN É PARTE DE UMA ARQUITETURA MAIOR. O LOG DE TRANSAÇÕES DO USUÁRIO E DO FRAMEWORK DE PROCESSADOR DE LOG FORNECEM AS FERRAMENTAS NECESSÁRIAS PARA TRANSMITIR ALTERAÇÕES EM OUTROS COMPONENTES DO SISTEMA GLOBAL SEM AFETAR AS OPERAÇÕES ORIGINAIS CAUSANDO A MUDANÇA
- **TRIGGERS:** O LOG DE TRANSAÇÕES DO USUÁRIO FORNECE A INFRAESTRUTURA BÁSICA PARA IMPLEMENTAR GATILHOS QUE PODEM SER DIMENSIONADOS PARA UM GRANDE NÚMERO DE TRANSAÇÕES SIMULTÂNEAS EM GRAFOS MUITO GRANDES. UM GATILHO É REGISTRADO COM UMA MUDANÇA PARTICULAR DE DADOS DISPARANDO UM EVENTO EM UM SISTEMA EXTERNO OU ALTERAÇÕES ADICIONAIS AO GRAFO

PARTICIONAMENTO DE GRAFO

- QUANDO O CLUSTER TITAN CONSISTE EM VÁRIAS INSTÂNCIAS DE ARMAZENAMENTO DE BACKEND, O GRAFO PODE SER DIVIDIDO ENTRE VÁRIAS MÁQUINAS
- UMA VEZ QUE O TITAN ARMAZENA O GRAFO EM UMA REPRESENTAÇÃO DE LISTA DE ADJACÊNCIA, A ATRIBUIÇÃO DE VÉRTICES PARA MÁQUINAS DETERMINA O PARTICIONAMENTO.
- POR PADRÃO, O TITAN UTILIZA UMA ESTRATÉGIA DE PARTICIONAMENTO ALEATÓRIO QUE ATRIBUI ALEATORIAMENTE VÉRTICES PARA MÁQUINAS
- PARTICIONAMENTO ALEATÓRIO RESULTA EM UM PROCESSAMENTO DE CONSULTAS MENOS EFICIENTE QUANDO O CLUSTER DO TITAN CRESCE PARA ACOMODAR MAIS DADOS
- PARTICIONAMENTO EXPLÍCITO DE GRAFOS PODE GARANTIR QUE SUBGRAPHS FREQUENTEMENTE PERCORRIDOS SEJAM ARMAZENADOS NA MESMA INSTÂNCIA, REDUZINDO ASSIM A SOBRECARGA DE COMUNICAÇÃO SIGNIFICATIVAMENTE

PARTICIONAMENTO DE GRAFO

- PARA ATIVAR O PARTICIONAMENTO GRAFO EXPLÍCITO NO TITAN, AS SEGUINTE OPÇÕES DE CONFIGURAÇÃO DEVEM SER DEFINIDAS QUANDO O CLUSTER TITAN É INICIALIZADO:

CLUSTER.PARTITION = TRUE

CLUSTER.MAX-PARTITIONS = 32

IDS.FLUSH = FALSE

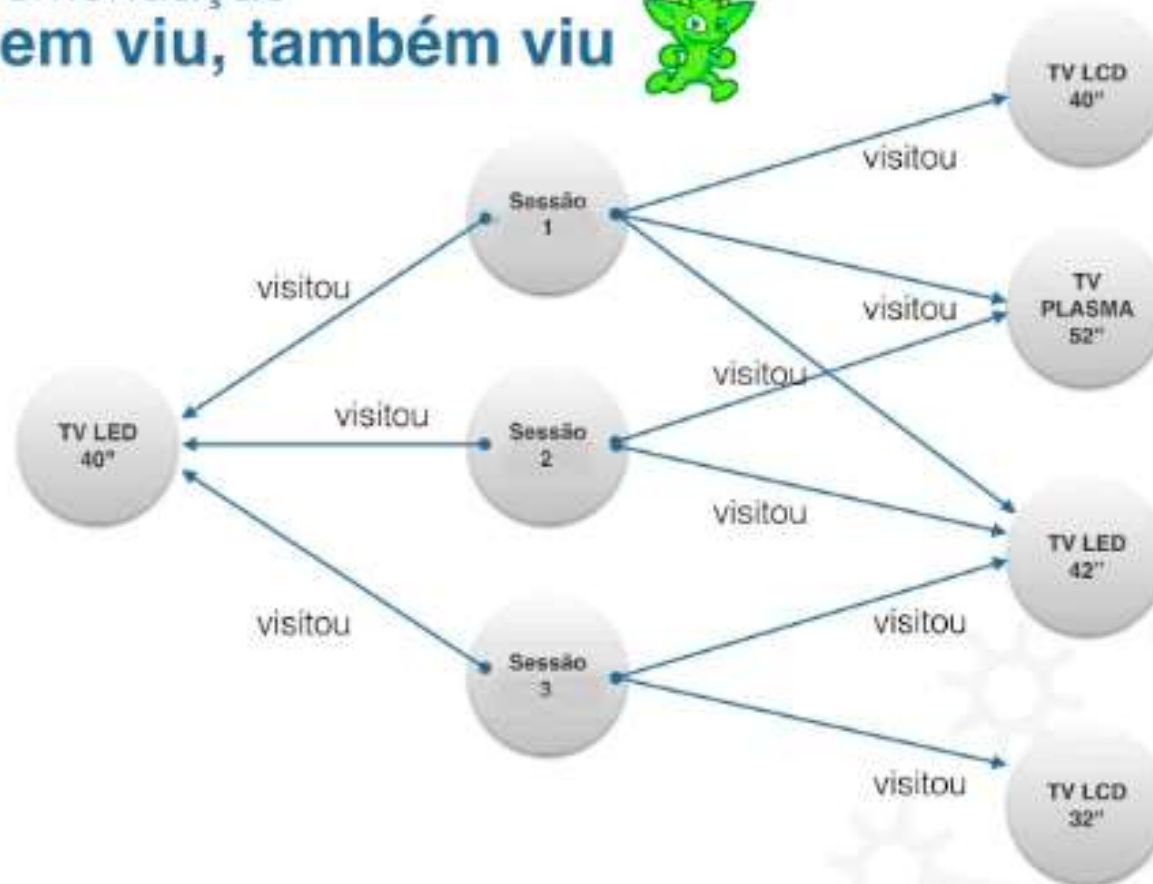
- MAX-PARTITIONS CONTROLA QUANTAS PARTIÇÕES VIRTUAIS
- ESTE NÚMERO DEVE SER CERCA DE DUAS VEZES O NÚMERO DE INSTÂNCIAS DE ARMAZENAMENTO DE BACKEND
- PARTICIONAMENTO EXPLÍCITO SÓ PODE SER HABILITADO EM INFRAESTRUTURAS DE ARMAZENAMENTO QUE SUPORTE DE ARMAZENAMENTO DE CHAVE ORDENA
 - **HBASE:** SUPORTA PARTICIONAMENTO DE GRAFOS EXPLÍCITO
 - **CASSANDRA:** DEVE SER CONFIGURADO PARA USAR BYTEORDEREDPARTITIONER A PARA SUPORTAR O **PARTICIONAMENTO** GRAFO EXPLÍCITO

QUANDO O GRAFO É PEQUENO OU ACOMODADO EM POUCAS INSTÂNCIAS DE ARMAZENAMENTO, É MELHOR USAR O PARTICIONAMENTO ALEATÓRIO PELA SUA SIMPLICIDADE

CRUD COM O EXEMPLO LUIZALABS

Recomendação

Quem viu, também viu



CRUD COM O EXEMPLO LUIZALABS

//inicia o gremlin-server, cassandra e o elasticsearch

`./bin/titan.sh start`

//Inicia o gremlin shell

`./bin/gremlin.sh`

//Cria um grafo vazio usando cassandra como storage backend

`gremlin>g = TitanFactory.open('cassandra:localhost')`

`==>standardtitangraph[cassandra:[localhost]`

//Outra opção de criação do grafo com o Cassandra

`gremlin>g = TitanFactory.build().set('storage.backend','cassandra').open()`

`==>standardtitangraph[cassandra:[127.0.0.1]]`

CRUD COM O EXEMPLO LUIZALABS

//Adiciona os vértices

```
gremlin>sessao1 = g.addVertex(T.label, "sessao", "name", "sessao1")
```

```
gremlin>sessao2 = g.addVertex(T.label, "sessao", "name", "sessao2")
```

```
gremlin>sessao3 = g.addVertex(T.label, "sessao", "name", "sessao3")
```

```
gremlin>produto1 = g.addVertex(T.label, "produto", "name", "TV LED 40")
```

```
gremlin>produto2 = g.addVertex(T.label, "produto", "name", "TV LCD 40")
```

```
gremlin>produto3 = g.addVertex(T.label, "produto", "name", "TV PLASMA 52")
```

```
gremlin>produto4 = g.addVertex(T.label, "produto", "name", "TV LED 42")
```

```
gremlin>produto5 = g.addVertex(T.label, "produto", "name", "TV LCD 32")
```

//Carrega todo o grafo e o associa a variável grafo

```
gremlin>grafo = g.traversal()
```

CRUD COM O EXEMPLO LUIZALABS

//Definindo as variáveis

```
gremlin>sessao1 = grafo.V().has('name', 'sessao1').next()
```

```
==>v[4320]
```

```
gremlin>sessao2 = grafo.V().has('name', 'sessao2').next()
```

```
==>v[4256]
```

```
gremlin>sessao3 = grafo.V().has('name', 'sessao3').next()
```

```
==>v[8312]
```

//Vértices produtos

```
gremlin>tvLed40 = grafo.V().has('name', 'TV LED 40').next()
```

```
==>v[4192]
```

```
gremlin>tvLcd40 = grafo.V().has('name', 'TV LCD 40').next()
```

```
==>v[8288]
```

```
gremlin>tvPlasma52 = grafo.V().has('name', 'TV PLASMA 52').next()
```

```
==>v[4304]
```

```
gremlin>tvLed42 = grafo.V().has('name', 'TV LED 42').next()
```

```
==>v[4328]
```

```
gremlin>tvLcd32 = grafo.V().has('name', 'TV LCD 32').next()
```

```
==>v[4216]
```

CRUD COM O EXEMPLO LUIZALABS

//Checando os vértices

```
gremlin> grafo.V(tvLed40)
```

```
==>v[4192]
```

```
gremlin> grafo.V(tvLed40).valueMap()
```

```
==>[name:[TV LED 40]]
```

```
gremlin> grafo.V(tvPlasma52).valueMap()
```

```
==>[name:[TV PLASMA 52]]
```


CRUD COM O EXEMPLO LUIZALABS

//Adicionando arestas que saem do vertice sessao1

```
gremlin>sessao1.addEdge('visitou', tvLed40)
```

```
==>e[odxcs-3c0-27th-38g][4320-visitou->4192]
```

```
gremlin>sessao1.addEdge('visitou', tvLcd40)
```

```
==>e[1cruak-3c0-27th-6e8][4320-visitou->8288]
```

```
gremlin>sessao1.addEdge('visitou', tvPlasma52)
```

```
==>e[1cruos-3c0-27th-3bk][4320-visitou->4304]
```

```
gremlin>sessao1.addEdge('visitou', tvLed42)
```

```
==>e[1crv30-3c0-27th-3c8][4320-visitou->4328]
```

//Arestas da sessao2

```
gremlin>sessao2.addEdge('visitou', tvLed40);
```

```
==>e[odxck-3a8-27th-38g][4256-visitou->4192]
```

```
gremlin>sessao2.addEdge('visitou', tvLed42);
```

```
==>e[odxqs-3a8-27th-3c8][4256-visitou->4328]
```

```
gremlin>sessao2.addEdge('visitou', tvPlasma52);
```

```
==>e[ody50-3a8-27th-3bk][4256-visitou->4304]
```

CRUD COM O EXEMPLO LUIZALABS

//Arestas da sessao3

```
gremlin>sessao3.addEdge('visitou', tvLed40);
```

```
==>e[odxcf-6ew-27th-38g][8312-visitou->4192]
```

```
gremlin>sessao3.addEdge('visitou', tvLed42);
```

```
==>e[odxqn-6ew-27th-3c8][8312-visitou->4328]
```

```
gremlin>sessao3.addEdge('visitou', tvLcd32);
```

```
==>e[ody4v-6ew-27th-394][8312-visitou->4216]
```

//Grafo completo 8 vértices e dez arestas

```
gremlin>grafo.V().valueMap();
```

```
==>[name:[sessao1]]
```

```
==>[name:[sessao2]]
```

```
==>[name:[sessao3]]
```

```
==>[name:[TV PLASMA 52]]
```

```
==>[name:[TV LED 40]]
```

```
==>[name:[TV LCD 32]]
```

```
==>[name:[TV LED 42]]
```

```
==>[name:[TV LCD 40]]
```

CRUD COM O EXEMPLO LUIZALABS

```
gremlin> grafo.V().count()
```

```
==>8
```

```
gremlin> grafo.E().count()
```

```
==>10
```

```
//Quem visitou tvLed40?
```

```
gremlin> grafo.V(tvLed40).in('visitou').valueMap()
```

```
==>[name:[sessao2]]
```

```
==>[name:[sessao1]]
```

```
==>[name:[sessao3]]
```

CRUD COM O EXEMPLO LUIZALABS

//Quais produtos a sessao1 visitou?

```
gremlin> grafo.V(sessao1).out('visitou').valueMap()
```

```
==>[name:[TV LED 40]]
```

```
==>[name:[TV PLASMA 52]]
```

```
==>[name:[TV LED 42]]
```

```
==>[name:[TV LCD 40]]
```

//Recomendação

```
gremlin> grafo.V(tvLed40).in('visitou').out('visitou').where(is(neq(tvLed40)))valueMap().groupCount()
```

```
==>[[name:[TV LED 42]]:3, [name:[TV LCD 40]]:1, [name:[TV LCD 32]]:1, [name:[TV PLASMA 52]]:2]
```

//Para exportar o grafo em JSON

```
gremlin> g.io(graphson()).writeGraph('/tmp/grafo_magazine.json')
```

//Para importar de JSON

```
gremlin> novoGrafo.io(loCore.graphson()).readGraph('/vagrant/grafo_magazine.json');
```



FIM



The image features a minimalist design with the word "OBRIGADO" centered on a white background. The corners are decorated with realistic water droplets of various sizes, some partially cut off by the edges, creating a clean and fresh aesthetic.

OBRIGADO