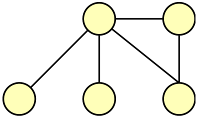


GRAPH DATABASES

A look into the world as a graph

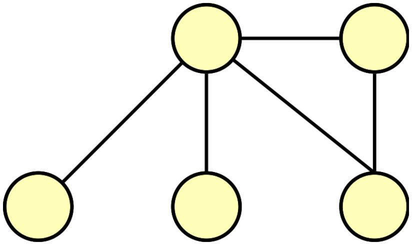
Pedro R. Paredes

December 4, 2015

DCC/FCUP - University of Porto

GRAPH PRELIMINARIES

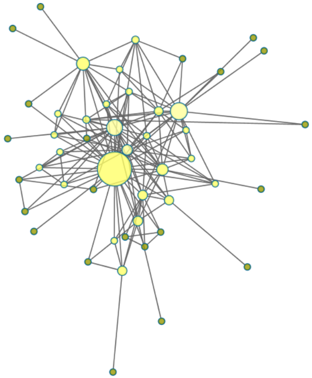
DATABASE OF WHAT?



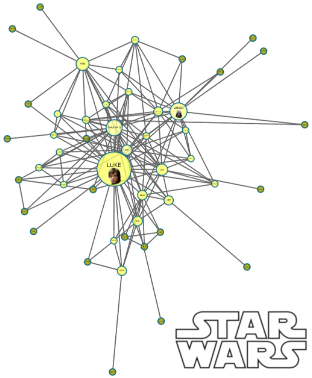
Definition of Graph

$G(V, E)$ is a set of vertices V and edges E

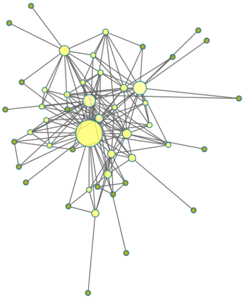
WHAT IS A GRAPH?



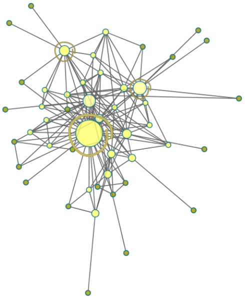
WHAT IS A GRAPH?



WHY A GRAPH?

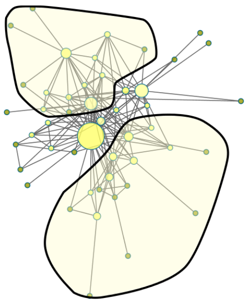


WHY A GRAPH?



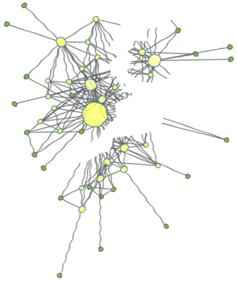
- node centrality

WHY A GRAPH?



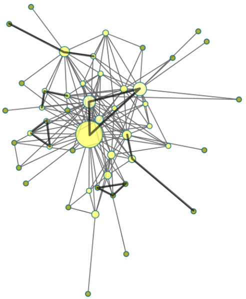
- node centrality
- community detection

WHY A GRAPH?



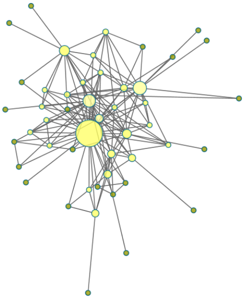
- node centrality
- community detection
- robustness

WHY A GRAPH?



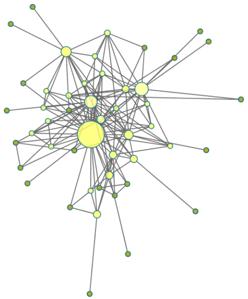
- node centrality
- community detection
- robustness
- find patterns (motifs)

WHY A GRAPH?



- node centrality
- community detection
- robustness
- find patterns (motifs)

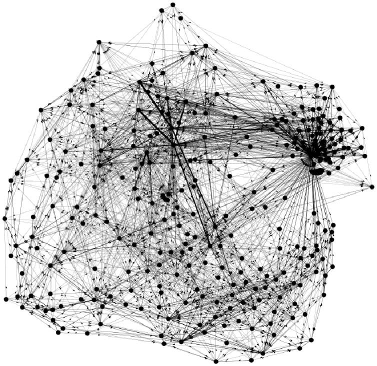
WHY A GRAPH?



- node centrality
- community detection
- robustness
- find patterns (motifs)

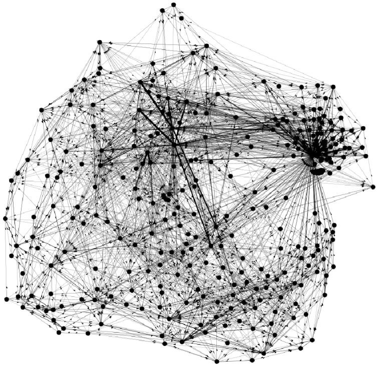
...

WHY A GRAPH?



↑ neural network ↑

WHY A GRAPH?



$\uparrow$ neural network $\uparrow$

expressiveness $\Rightarrow$ power $+$ utility

WHERE ARE GRAPHS USED?

WHERE ARE GRAPHS USED?

WHERE ARE GRAPHS USED?

Walmart

tomtom

WHERE ARE GRAPHS USED?

everywhere!

DATABASE PRELIMINARIES

EVERYBODY KNOWS THESE...

MySQL

PostgreSQL

MariaDB

SQLite

ORACLE DATABASE

Microsoft SQL Server

these are all relational databases

BUT THERE ARE OTHER FLAVORS

mongoDB

memsql®

redis

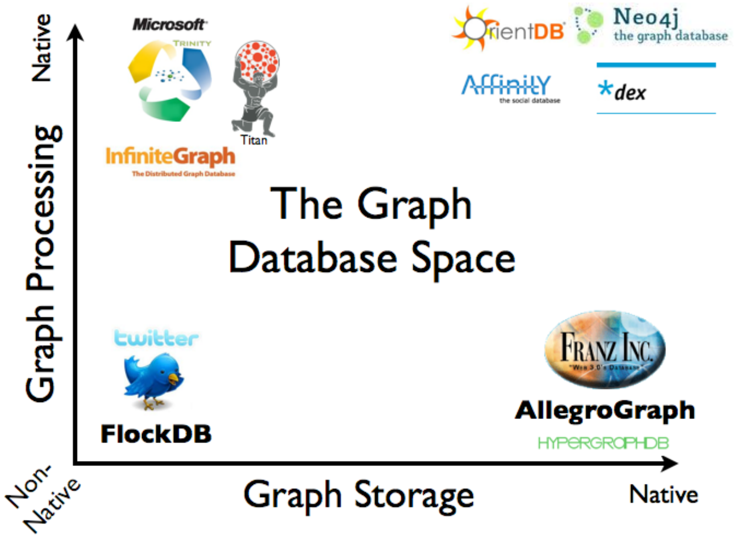
BUT THERE ARE OTHER FLAVORS

mongoDB memsql redis

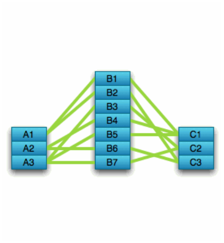
↓ these are graph databases ↓

TITAN neo4j OrientDB

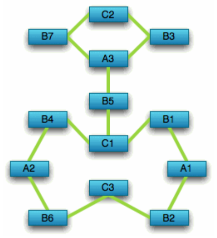
AND WHAT KIND OF GRAPH DATABASES ARE THERE?



AND WHY SHOULD I USE IT?

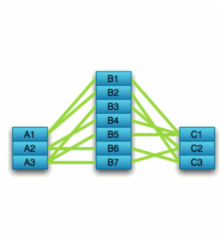


in a relational database

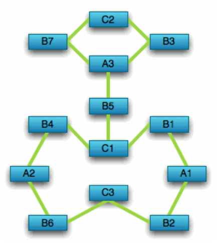


in a graph database

AND WHY SHOULD I USE IT?



in a relational database



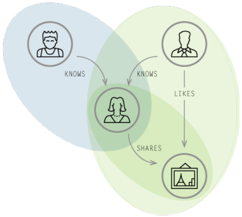
in a graph database

simplify modeling interconnected data

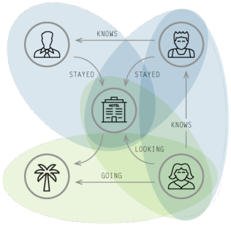
+

optimize querying interconnected data

AND WHY SHOULD I USE IT?



social networks



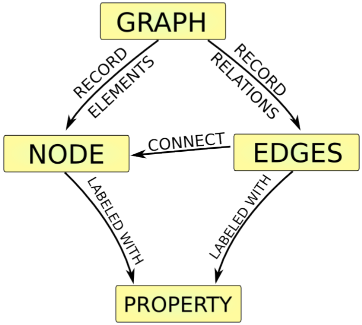
customer purchases

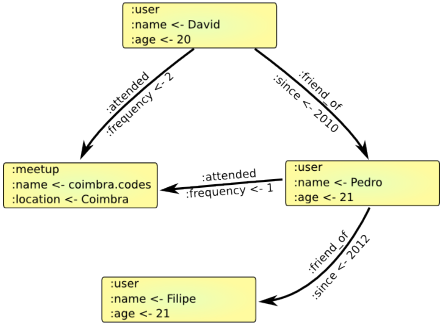
...

most (a lot?) data is interconnected!

GRAPH DATABASES 101

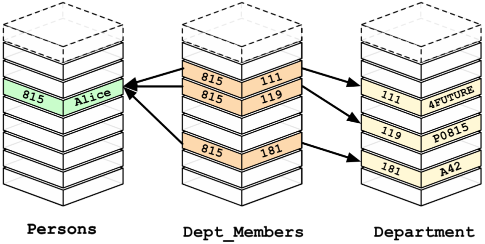
A POSSIBLE REPRESENTATION



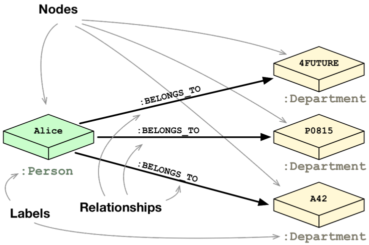


the property graph

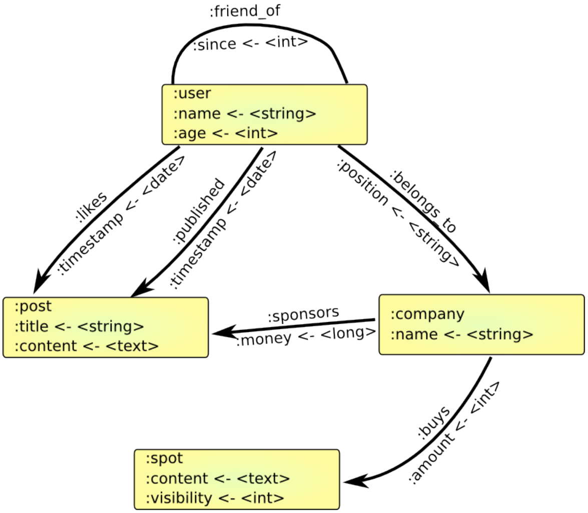
HOW TO REPRESENT MY DATA?



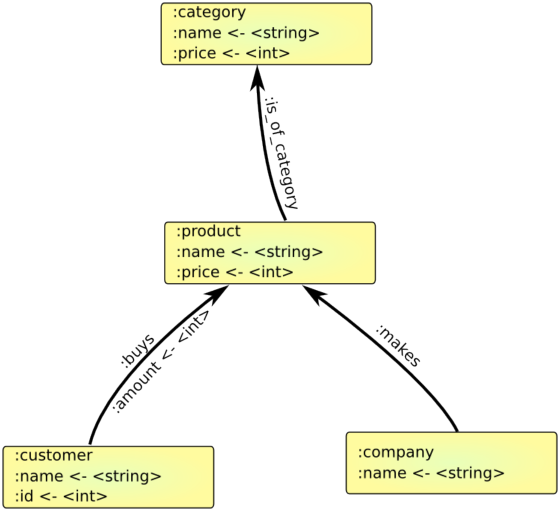
⇓



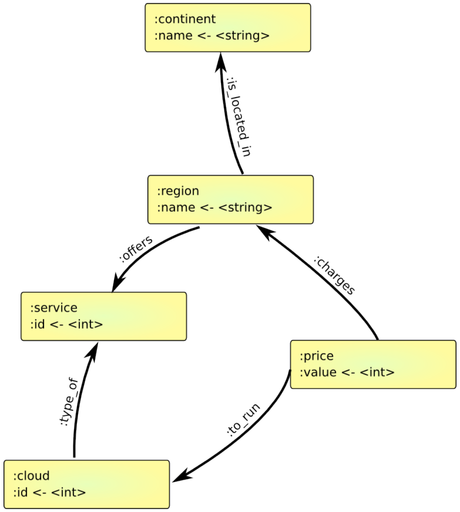
LET'S MODEL SOCIAL DATA



LET'S MODEL SALES DATA



LET'S MODEL SERVER DATA



QUERYING THE GRAPH

THE JOIN FRENZY

```
SELECT * FROM Customers AS C
JOIN Customer_Product AS CP
  ON CP.customer_id = C.customer_id
JOIN Product AS P
  ON P.product_id = CP.product_id
JOIN Category AS CT
  ON CT.category_id = CP.category_id
WHERE CT.name = ''Movie''
```

```
SELECT * FROM Actors AS A
JOIN Actors_Movies AS AM
  ON AM.actor_id = A.actor_id
JOIN Movies AS M
  ON M.movie_id = AM.movie_id
WHERE M.name LIKE ''Star Wars%''
```

THE JOIN FRENZY

```
SELECT * FROM Customers AS C
JOIN Customer_Product AS CP
  ON CP.customer_id = C.customer_id
JOIN Product AS P
  ON P.product_id = CP.product_id
JOIN Category AS CT
  ON CT.category_id = CP.category_id
WHERE CT.name = ''Movie''
```

```
SELECT * FROM Actors AS A
JOIN Actors_Movies AS AM
  ON AM.actor_id = A.actor_id
JOIN Movies AS M
  ON M.movie_id = AM.movie_id
WHERE M.name LIKE ''Star Wars%''
```

. . . ZZZzzz

NO MORE JOINS

```
MATCH (c:Customer)
   -[:Buys]->(p:Product)
   -[:Of_Type]->(ct:Category)
WHERE ct.name = ``Movie''
RETURN c.name
```

```
MATCH (a:Actor)-[:Starred]->(p:Movie)
WHERE p.name ~= ``Star Wars.*''
RETURN a.name
```

NO MORE JOINS

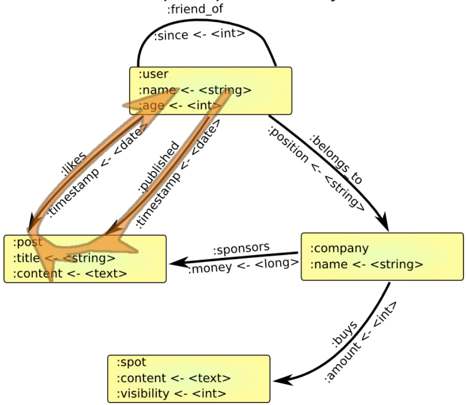
```
MATCH (c:Customer)
   -[:Buys]->(p:Product)
   -[:Of_Type]->(ct:Category)
WHERE ct.name = ``Movie''
RETURN c.name
```

```
MATCH (a:Actor)-[:Starred]->(p:Movie)
WHERE p.name ~= ``Star Wars.*''
RETURN a.name
```

Much simpler! (and more efficient)

RETHINKING QUERYING

users that liked a post published by a certain user



a query follows the graph in a organic fashion

NEO4J AND CYPHER

WHAT'S NEO4J?

- most used graph database

WHAT'S NEO4J?

- most used graph database
- full ACID transactions

WHAT'S NEO4J?

- most used graph database
- full ACID transactions
- scalable up to billions of nodes/relationships

WHAT'S NEO4J?

- most used graph database
- full ACID transactions
- scalable up to billions of nodes/relationships
- native storage engine

WHAT'S NEO4J?

- most used graph database
- full ACID transactions
- scalable up to billions of nodes/relationships
- native storage engine
- powerful traversal framework

WHAT'S NEO4J?

- most used graph database
- full ACID transactions
- scalable up to billions of nodes/relationships
- native storage engine
- powerful traversal framework
- native querying language (Cypher)

WHAT'S CYPHER?

- Neo4j's query language

WHAT'S CYPHER?

- Neo4j's query language
- declarative graph query language (analogue to SQL)

WHAT'S CYPHER?

- Neo4j's query language
- declarative graph query language (analogue to SQL)
- expressive and efficient querying and updating

WHAT'S CYPHER?

- Neo4j's query language
- declarative graph query language (analogue to SQL)
- expressive and efficient querying and updating
- simple but powerful

WHAT'S CYPHER?

- Neo4j's query language
- declarative graph query language (analogue to SQL)
- expressive and efficient querying and updating
- simple but powerful

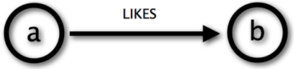
WHAT'S CYPHER?

- Neo4j's query language
- declarative graph query language (analogue to SQL)
- expressive and efficient querying and updating
- simple but powerful

there are alternatives (like Gremlin) but Cypher is simple and practical

CYPHER BASICS

Cypher using relationship ‘likes’



Cypher

```
(a) -[:LIKES]-> (b)
```

```
MATCH  (node1:Label1)−[rel:REL_TYPE]−>(node2:Label2)
WHERE  node1.property = ‘‘value’’
RETURN  node2.propertyA, node2.propertyB, rel.property
```

LET'S QUERY SOCIAL DATA

simple "friend of friend" query

```
MATCH (u1:user)
   -[:friend_of]->(u2:user)
   -[:friend_of]->(u3:user)
WHERE u1.name = ''Pedro''
RETURN u3.name
```

⇑

```
SELECT * FROM users AS u1
JOIN user_friend_user AS ufu1
  ON u1.user_id = ufu1.user_1_id
JOIN users AS u2
  ON u2.user_id = ufu1.user_2_id
JOIN user_friend_user AS ufu2
  ON u2.user_id = ufu2.user_1_id
JOIN users AS u3
  ON u3.user_id = ufu2.user_2_id
WHERE u1.name = ''Pedro''
```

LET'S QUERY SALES DATA

simple recommendation engine query

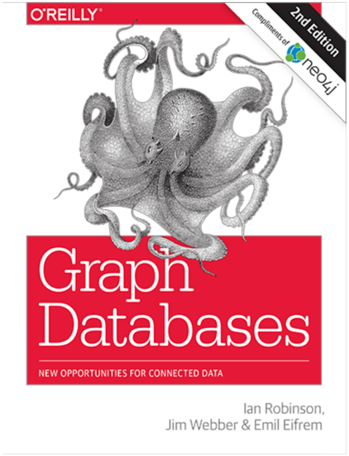
```
MATCH (c:customer)-[:friend_of]->(f),
   (f)-[:buys]->(p:product),
   (p)-[:is_of_category]->(ct:category),
   (p)<-[:makes]-(cp:company)

WHERE c.name = ''Pedro''
  AND ct.category = ''Movie''
  AND cp.name = ''Disney''
  AND p.price < 30

RETURN p.name, count(*) as occurrence
ORDER BY occurrence DESC
LIMIT 5
```

$\Uparrow$

```
SELECT * FROM ...
TOO MUCH STUFF!!
```

REFERENCES

A COOL BOOK (AND FREE)



O'Reilly's Graph Databases, 2nd Edition

SOME USEFUL WEBSITES

http://neo4j.com/

http://neo4j.com/developer/

http://neo4j.com/docs/stable/cypher-refcard/

http://www.opencypher.org/

https://tinkerpop.incubator.apache.org/

NOT FOR TODAY, BUT MAYBE SOMEDAY

- advanced queries (regexp, variable paths, indices, . . .)
- large scale applications (sharding)
- the underlying implementation (how does Neo4j work?)
- more graph metrics

Questions?

That’s all Folks!