POLITECNICO
MILANO 1863

SYSTEMS AND METHODS FOR BIG AND UNSTRUCTURED DATA

Graph Databases – Neo4J

Marco Brambilla

marco.brambilla@polimi.it

@marcobrambi

Agenda

- Graph Theory
- Graph Databases
- Neo4J

1. Graph Theory

Marco Brambilla

@marcobrambi

marco.brambilla@polimi.it

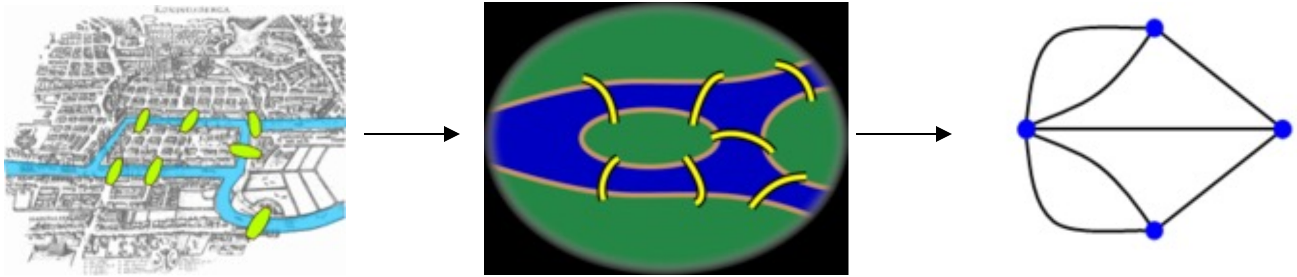
ICO

53

Graph Theory - History

Leonhard Euler's paper on "*Seven Bridges of Königsberg*" ,

published in 1736.



POLITECNICO
MILANO 1863

Famous problems

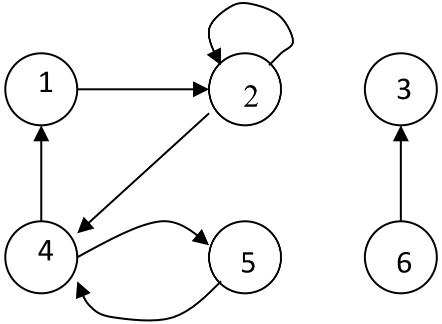
- "The traveling salesman problem"
  - A traveling salesman is to visit a number of cities; how to plan the trip so every city is visited once and just once and the whole trip is as short as possible ?
- In 1852 Francis Guthrie posed the "four color problem" which asks if it is possible to color, using only four colors, any map of countries in such a way as to prevent two bordering countries from having the same color.
- SOLVED ONLY 120 YEARS LATER!

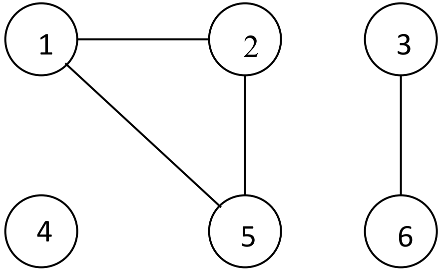
Other Examples

- Cost of wiring electronic components
- Shortest route between two cities.
- Shortest distance between all pairs of cities in a road atlas.
- Matching / Resource Allocation
- Task scheduling
- Visibility / Coverage

What is a Graph?

- Informally a *graph* is a set of nodes joined by a set of lines or arrows.





Definition: Graph

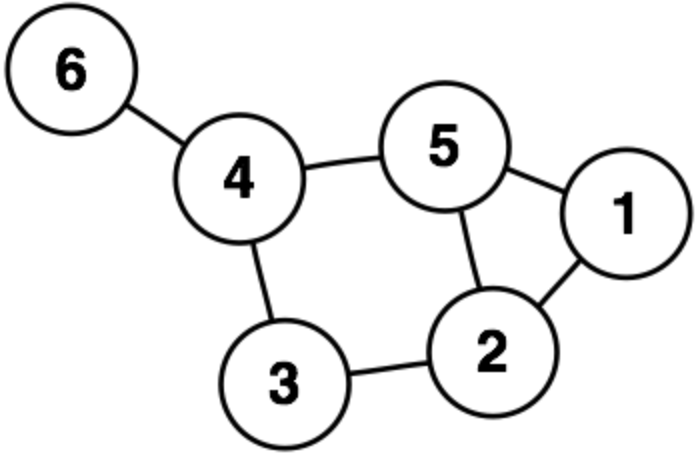
- G is an ordered triple G:=(V, E, f)
  - V is a set of nodes, points, or vertices.
  - E is a set, whose elements are known as edges or lines.
  - f is a function
    - maps each element of E
    - to an unordered pair of vertices in V.

POLITECNICO
MILANO 1863

Definitions

- Vertex
  - Basic Element
  - Drawn as a *node* or a *dot*.
  - Vertex set of G is usually denoted by $V(G)$, or V
- Edge
  - A set of two elements
  - Drawn as a line connecting two vertices, called end vertices, or endpoints.
  - The edge set of G is usually denoted by E(G), or E.

Example



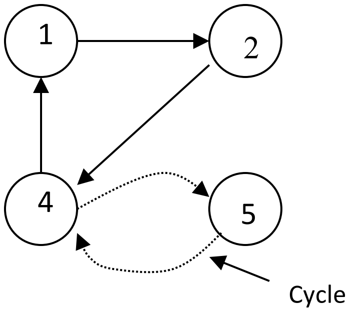
- V:={1,2,3,4,5,6}
- E:={{1,2},{1,5},{2,3},{2,5},{3,4},{4,5},{4,6}}

Simple Graphs

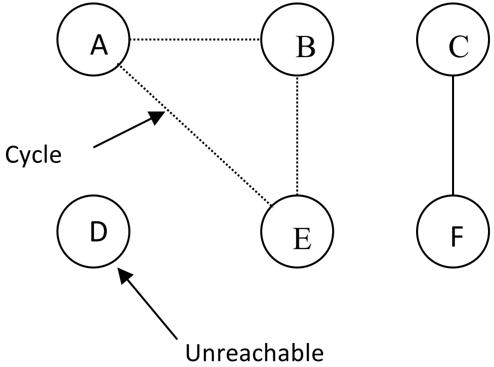
Simple graphs are graphs without multiple edges or self-loops.

Path

- A *path* is a sequence of vertices such that there is an edge from each vertex to its successor.
- A path is ***simple*** if each vertex is distinct.



Simple path from 1 to 5
= [1, 2, 4, 5]
Our text's alternates the vertices and edges.

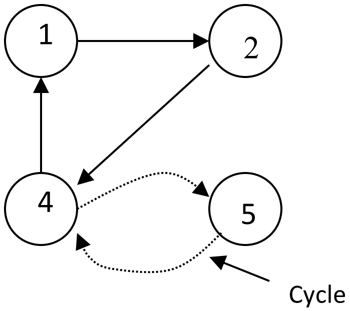


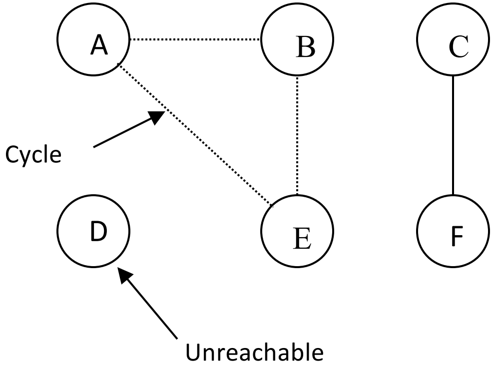
If there is path *p* from *u* to *v* then we say *v* is **reachable** from *u* via *p*.

POLITECNICO
MILANO 1863

Cycle

- A path from a vertex to itself is called a ***cycle***.
- A graph is called ***cyclic*** if it contains a cycle;
  - otherwise it is called ***acyclic***





Connectivity

- A graph is *connected* if
    - you can get from any node to any other by following a sequence of edges OR
    - any two nodes are connected by a path.
- A directed graph is *strongly connected* if there is a directed path from any node to any other node.

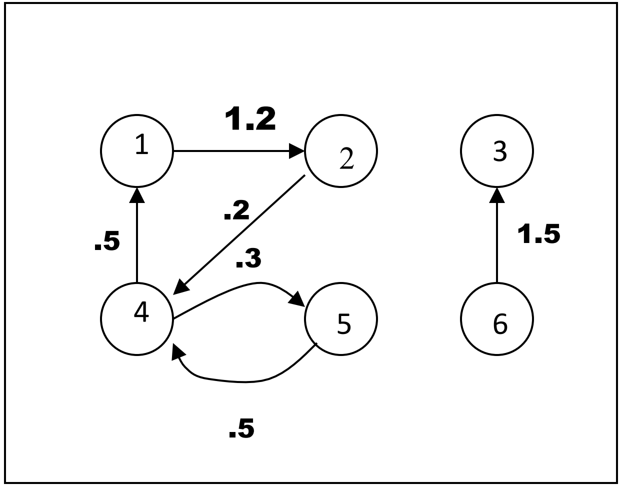
Sparse/Dense

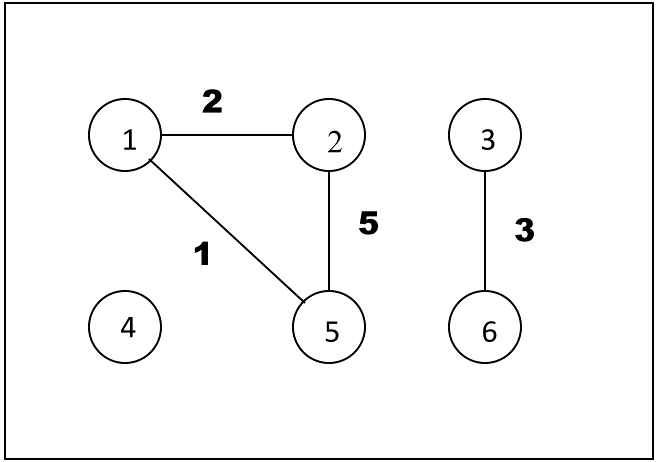
- A graph is ***sparse*** if $|E| \approx |V|$
- A graph is ***dense*** if $|E| \approx |V|^2$.

POLITECNICO
MILANO 1863

A *weighted graph*

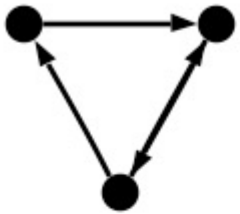
- is a graph for which each edge has an associated ***weight***, usually given by a ***weight function*** $w: E \rightarrow \mathbb{R}$.





Directed Graph (digraph)

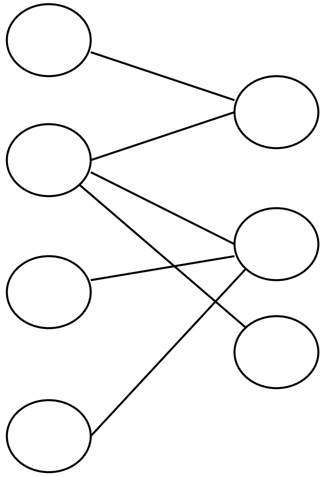
- Edges have directions
  - An edge is an *ordered* pair of nodes



POLITECNICO
MILANO 1863

Bipartite graph

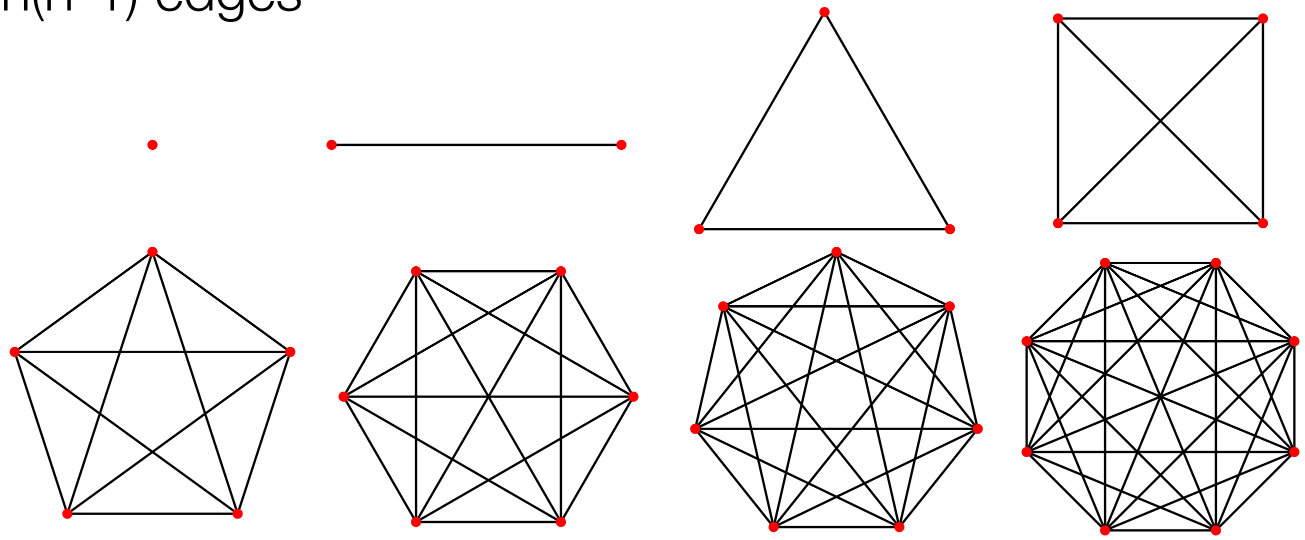
- V can be partitioned into 2 sets V_1 and V_2 such that $(u,v) \in E$ implies
  - either $u \in V_1$ and $v \in V_2$
  - OR $v \in V_1$ and $u \in V_2$.



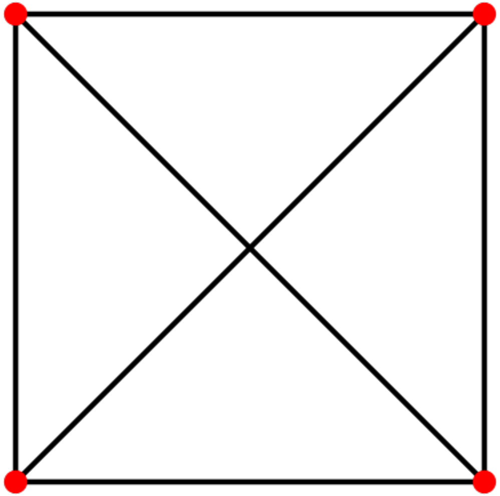
POLITECNICO
MILANO 1863

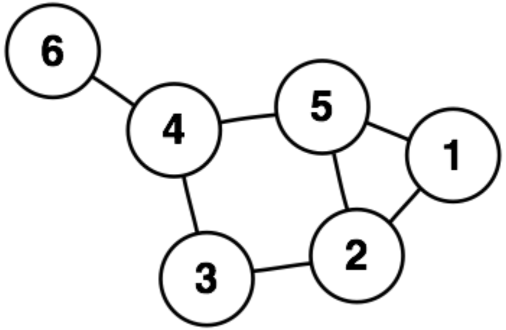
Complete Graph

- Denoted K_n
- Every pair of vertices are adjacent
- Has n(n-1) edges



Planar Graph

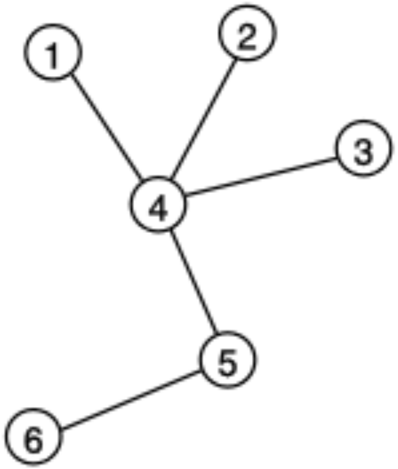




- Can be drawn on a plane such that no two edges intersect
- K_4 is the largest complete graph that is planar

Tree

- Connected Acyclic Graph
- Two nodes have *exactly* one path between them

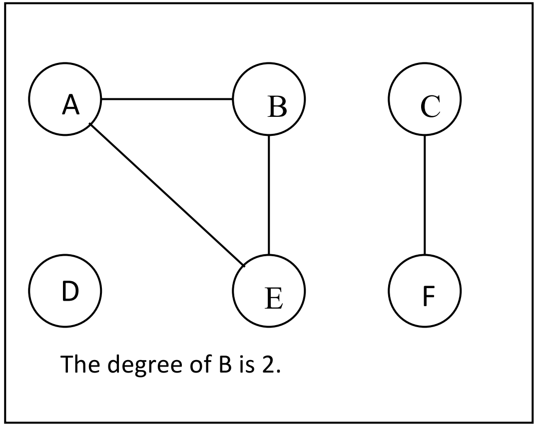


Generalization: Hypergraph

- Generalization of a graph,
    - edges can connect any number of vertices.
- Formally, an hypergraph is a pair (X,E) where
    - X is a set of elements, called nodes or vertices, and
    - E is a set of subsets of X, called hyperedges.
- Hyperedges are arbitrary sets of nodes,
    - contain an arbitrary number of nodes.

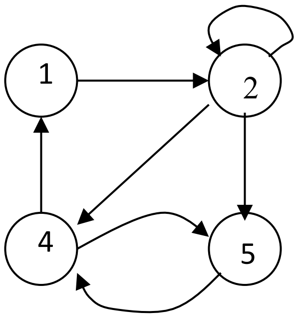
Degree

- Number of edges incident on a node



Degree (Directed Graphs)

- In degree: Number of edges entering
- Out degree: Number of edges leaving
- Degree = indegree + outdegree



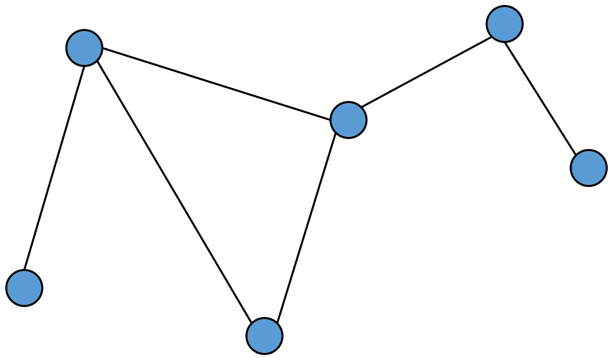
The in degree of 2 is 2 and
the out degree of 2 is 3.

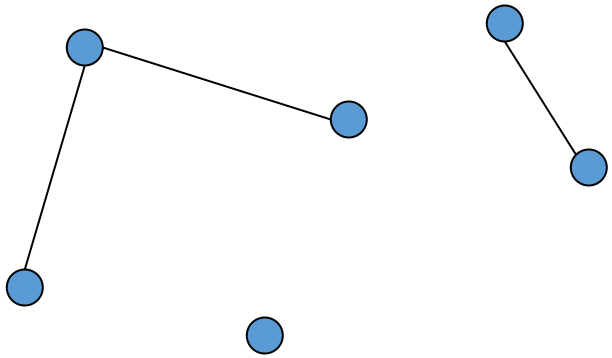
Subgraph

- Vertex and edge sets are subsets of those of G
  - a *supergraph* of a graph G is a graph that contains G as a subgraph.

Spanning subgraph

- Subgraph H has the same vertex set as G.
  - Possibly not all the edges
  - “H spans G”.





Graph ADT

- In computer science, a graph is an abstract data type (ADT)
- that consists of
  - a set of nodes and
  - a set of edges
    - establish relationships (connections) between the nodes.
- The graph ADT follows directly from the graph concept from mathematics.

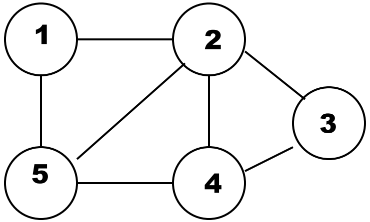
Representation (Matrix)

- Incidence Matrix
  - E x V
  - [edge, vertex] contains the edge's data
- Adjacency Matrix
  - V x V
  - Boolean values (adjacent or not)
  - Or Edge Weights

Representation (List)

- Edge List
  - pairs (ordered if directed) of vertices
  - Optionally weight and other data
- Adjacency List

Adjacency matrix representation



$$
\begin{array}{c|ccccc|}
 & 1 & 2 & 3 & 4 & 5 \\
\hline
1 & 0 & 1 & 0 & 0 & 1 \\
2 & 1 & 0 & 1 & 1 & 1 \\
3 & 0 & 1 & 0 & 1 & 0 \\
4 & 0 & 1 & 1 & 0 & 1 \\
5 & 1 & 1 & 0 & 1 & 0 \\
\hline
\end{array}
$$

- $|V| \times |V|$ matrix $A = (a_{ij})$ such that
  $a_{ij} = 1$ if $(i, j) \in E$ and 0 otherwise.
  We arbitrarily uniquely assign the numbers $1, 2, \ldots, |V|$ to each vertex.

Graph Algorithms

- Shortest Path
  - Single Source
  - All pairs (Ex. Floyd Warshall)
- Network Flow
- Matching
  - Bipartite
  - Weighted
- Topological Ordering
- Strongly Connected

Graph Algorithms

- Biconnected Component / Articulation Point
- Bridge
- Graph Coloring
- Euler Tour
- Hamiltonian Tour
- Clique
- Isomorphism
- Edge Cover
- Vertex Cover
- Visibility

2. Graph Databases

Marco Brambilla

@marcobrambi

marco.brambilla@polimi.it

Motivation

- Relational Databases
  - (incredibly!)
- are not good in managing relationships!

Graph Databases

- Database that uses graph structures with **nodes, edges and properties to store data**
- Provides **index-free adjacency**
  - Every node is a pointer to its adjacent element
- Edges hold most of the important information and connect
  - nodes to other nodes
  - nodes to properties

Advantage of Graph Databases

- When there are relationships that you want to analyze, Graph databases become a very nice fit because of the data structure
- Graph databases are very fast for associative data sets
  - Like social networks
- Map more directly to object oriented applications
  - Object classification and Parent->Child relationships

Relational Database Representation

- Sailor(sid:integer, sname:char(10), rating: integer, age:real)
- Boat(bid:integer, bname:char(10), color:char(10))
- Reserve(sid:integer, bid:integer, day:date)

Sailor

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

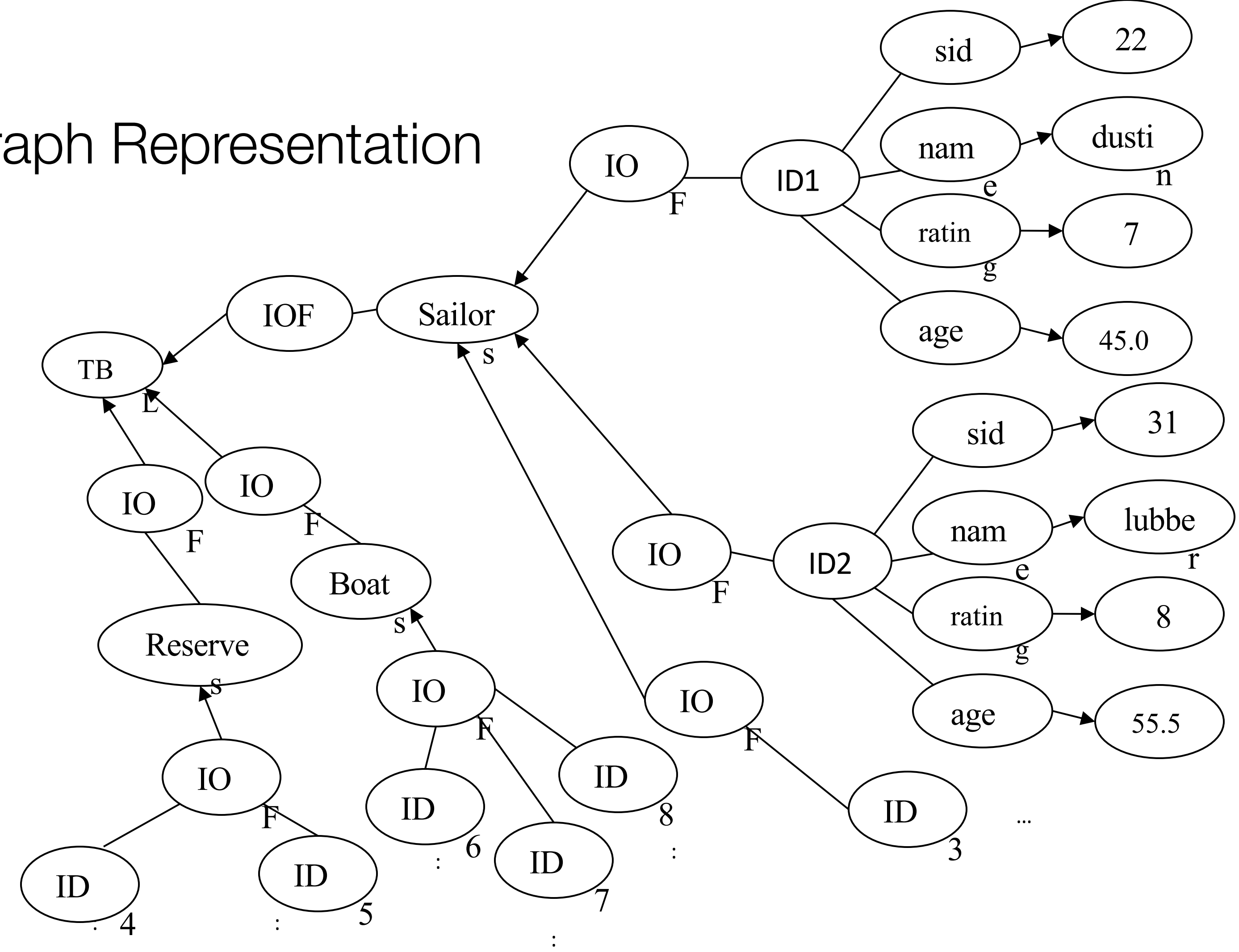
Reserve

sid	bid	day
22	101	10/10/96
58	103	11/12/96

Boat

bid	bname	color
101	Interlake	red
102	Clipper	green
103	Marine	red

Graph Representation



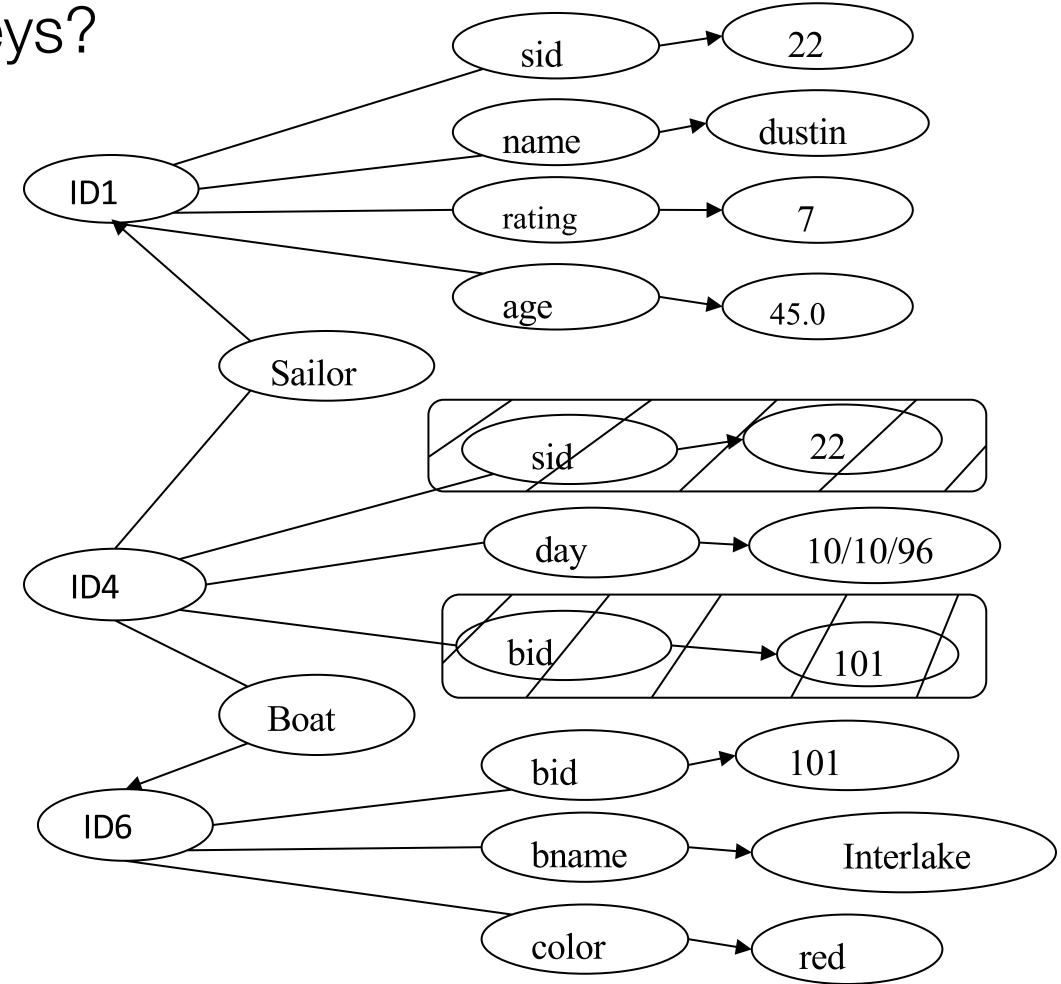
POLITECNICO MILANO 1863

Actual Graph Model

- Sailor [Reserves] Boat

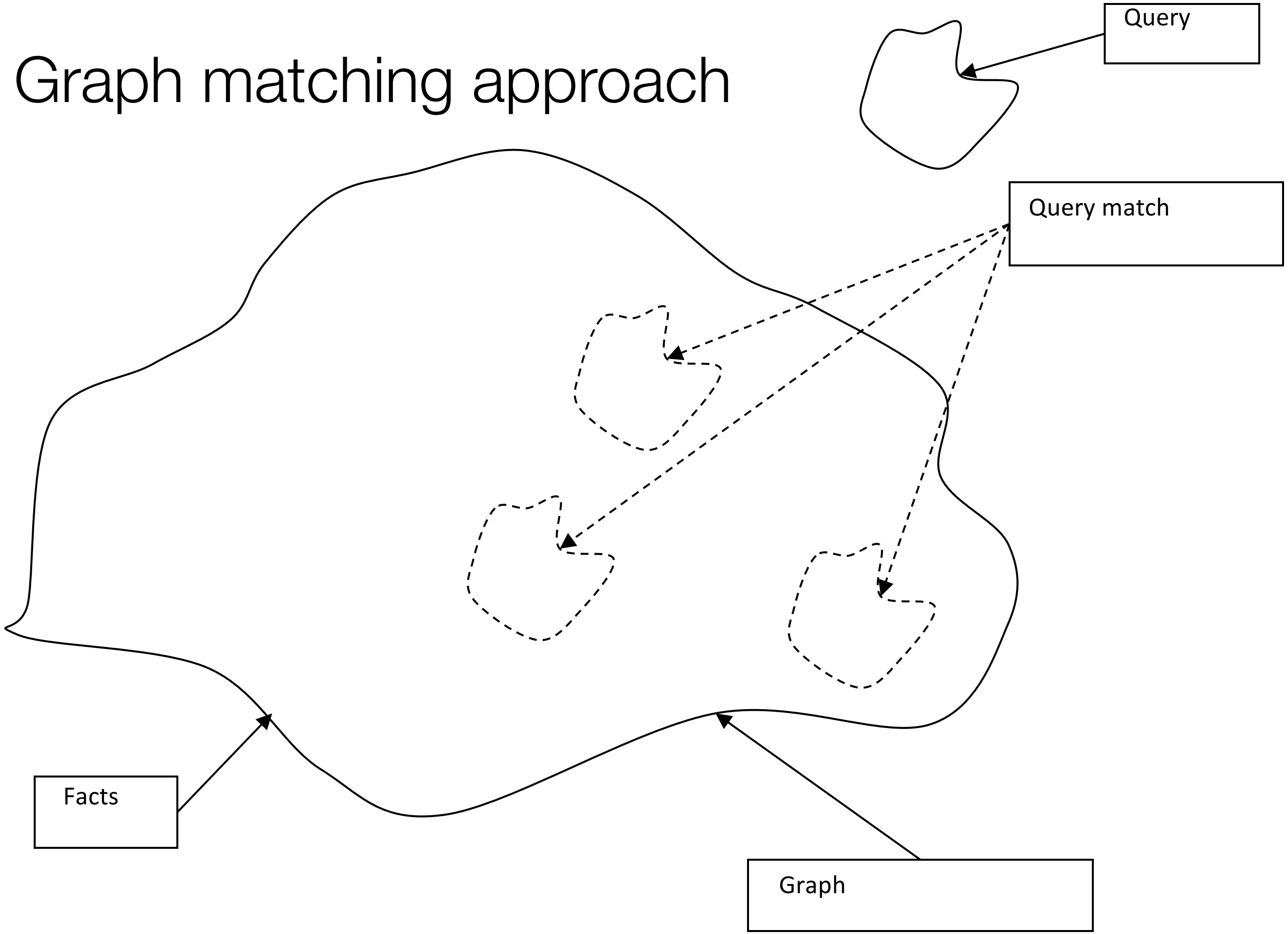
- (:Sailor) –[:reserves]-> (:Boat)

Foreign Keys?
No thanks



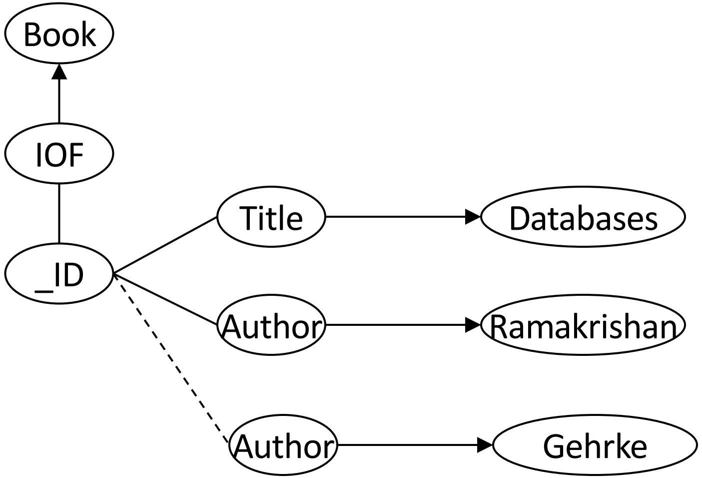
POLITECNICO
MILANO 1863

Query: Graph matching approach

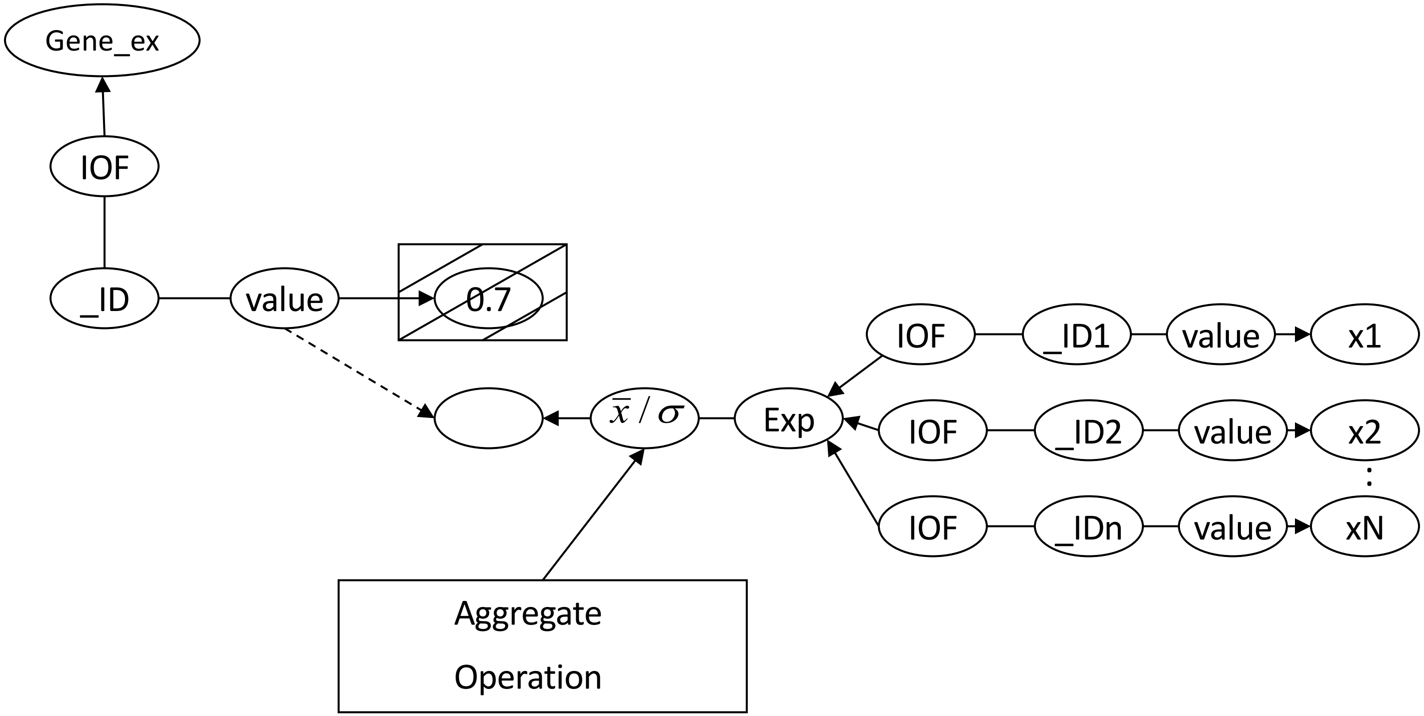


POLITECNICO
MILANO 1863

Easy to Extend



Easy to Change



POLITECNICO
MILANO 1863

3. Neo4J

Marco Brambilla

@marcobrambi

marco.brambilla@polimi.it

What is Neo4j

- Developed by Neo Technologies
- Most Popular Graph Database
- Implemented in Java
- Open Source

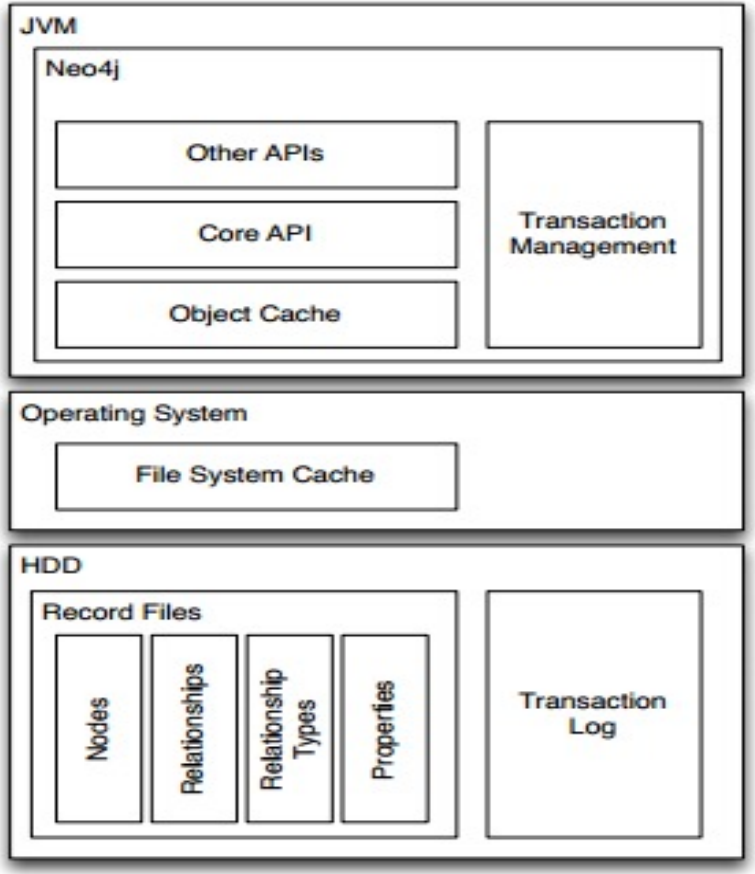
(www.neo4j.org)

Salient features of Neo4j

- Neo4j is schema free – Data does not have to adhere to any convention
- ACID – atomic, consistent, isolated and durable for logical units of work
- Easy to get started and use
- Well documented and large developer community
- Support for wide variety of languages
  - Java, Python, Perl, Scala, Cypher, etc

(Mistry, Deep, 2013)

Neo4j Software Architecture



POLITECNICO
MILANO 1863

Purpose

- Meant to be an operational DB, not specifically for analytics
- ACID
- Efficient on nodes
- Not so efficient in whole-graph analysis

Data Model

- Nodes – with labels (type) and attributes
- Edges

- Indexes

Cypher

- Query Language for Neo4j
- Declarative language
- Easy to formulate queries based on relationships
- Many features stem from improving on pain points with SQL such as join tables

(Hunger, Michael 2013)

Cypher – data creation

```
CREATE (Neo:Crew { name:'Neo' })
```

```
(Neo)-[:KNOWS]->(Morpheus)
```

(www.neo4j.org/learn/cypher)

Cypher – data creation

- Multiple labels:
- Create (n:Actor:Director {name:'Clint Eastwood'})

- Importing:
- USING PERIODIC COMMIT
- LOAD CSV WITH HEADERS FROM "file:customers.csv" AS row
- CREATE (:Customer {companyName: row.CompanyName,
- customerID: row.CustomerID, phone: row.Phone});

Cypher - Merge

- ID-based

```
LOAD CSV WITH HEADERS FROM "file:///transfers.csv" AS row
MERGE (player:Player {id: row.playerUri})
ON CREATE SET player.name =  row.playerName,
              player.position = row.playerPosition
```

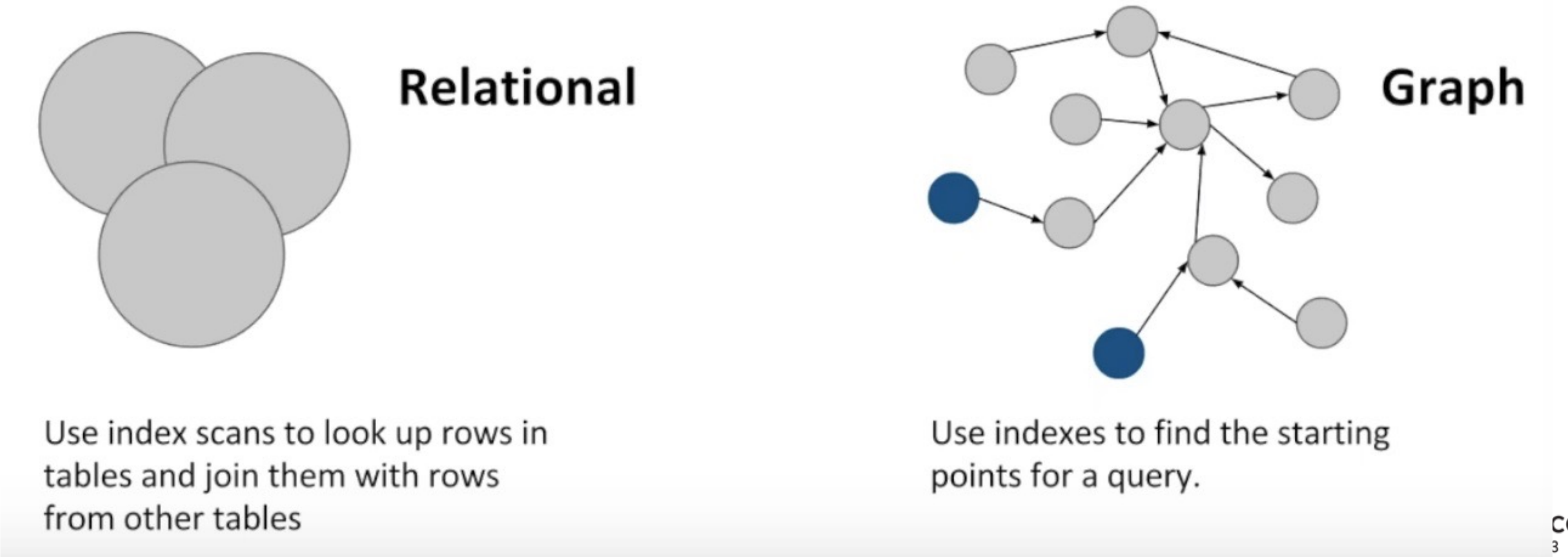
POLITECNICO
MILANO 1863

Importing Edges

```
LOAD CSV WITH HEADERS FROM "file:///transfers.csv" AS row
MATCH (player:Player {id: row.playerUri})
MATCH (source:Club {id: row.sellerClubUri})
MATCH (destination:Club {id: row.buyerClubUri})
MERGE (t:Transfer {id: row.transferUri})
ON CREATE SET t.season = row.season, t.rank = row.transferRank,
              t.fee = row.transferFee
MERGE (t)-[:OF_PLAYER { age: row.playerAge }]->(player)
MERGE (t)-[:FROM_CLUB]->(source)
MERGE (t)-[:TO_CLUB]->(destination)
```

Cypher - indexes

- CREATE INDEX ON :Customer(customerID);



Cypher - constraints

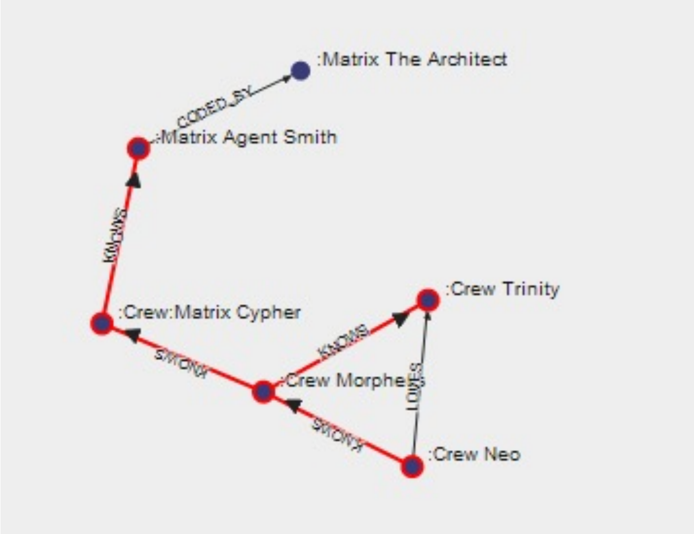
- CREATE CONSTRAINT ON (c:Customer)
- ASSERT c.customerID IS UNIQUE;

Cypher - Queries

- START
- MATCH Pattern Matching
- WHERE Expressions, Predicates
- RETURN Output

Cypher

```
Query:
MATCH (n:Crew)-[r:KNOWS*]-m
WHERE n.name='Neo'
RETURN n AS Neo,r,m
```



Neo	r	m
{name:"Neo"}	[(0)-[0:KNOWS]->(1)]	(1:Crew {name:"Morpheus"})
{name:"Neo"}	[(0)-[0:KNOWS]->(1), (1)-[2:KNOWS]->(2)]	(2:Crew {name:"Trinity"})
{name:"Neo"}	[(0)-[0:KNOWS]->(1), (1)-[3:KNOWS]->(3)]	(3:Crew:Matrix {name:"Cypher"})
{name:"Neo"}	[(0)-[0:KNOWS]->(1), (1)-[3:KNOWS]->(3), (3)-[4:KNOWS]->(4)]	(4:Matrix {name:"Agent Smith"})

(www.neo4j.org/learn/cypher)

General Query Format

- MATCH (user)-[:FRIEND]-(friend)
- WITH user, count(friend) AS friends
- ORDER BY friends DESC
- SKIP 1   LIMIT 3
- RETURN user

- Aggregation can be used (count).
- WITH separates query parts explicitly, to declare the variables for the next part.
- SKIP skips results at the top and LIMIT limits the number of results.

Patterns

`(n:Person)`
Node with Person label.

`(n:Person:Swedish)`
Node with both Person and Swedish labels.

`(n:Person {name: $value})`
Node with the declared properties.

`()-[r {name: $value}]-()`
Matches relationships with the declared properties.

`(n)-->(m)`
Relationship from n to m.

`(n)--(m)`
Relationship in any direction between n and m.

`(n:Person)-->(m)`
Node n labeled Person with relationship to m.

Patterns

`(m)<-[:KNOWS]-(n)` Relationship of type KNOWS from n to m.	
`(n)-[:KNOWS|:LOVES]->(m)` Relationship of type KNOWS or of type LOVES from n to m.	
`(n)-[r]->(m)` Bind the relationship to variable r.	
`(n)-[*1..5]->(m)` Variable length path from 1 to 5 rels. from n to m.	
`(n)-[*]->(m)` Variable length path of any number of rels. from n to m	
`(n)-[:KNOWS]->(m {property: $value})` A relationship of type KNOWS from a node n to a node m with the declared property.	

Stored Procedures

- You can add Java functions
- Simply move your JAR to a folder!
- Use the function in Cypher

Paths

```
shortestPath((n1:Person)-[*..6]-(n2:Person))
```

Find a single shortest path.

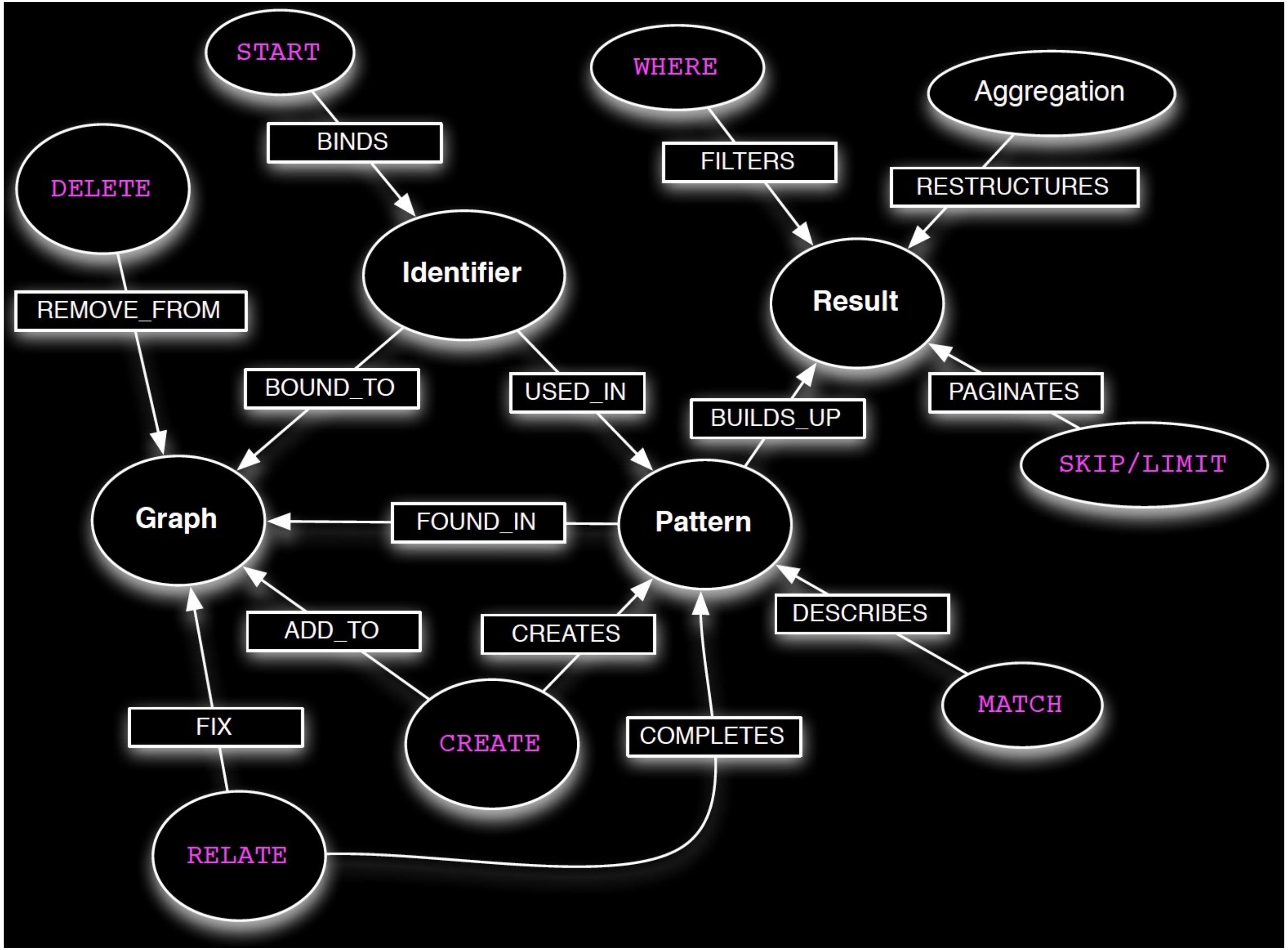
```
allShortestPaths((n1:Person)-[*..6]->(n2:Person))
```

Find all shortest paths.

```
size((n)-->()-->())
```

Count the paths matching the pattern.

Core operators and impact



Hints

- **Use parameters instead of literals** when possible. This allows Cypher to re-use your queries instead of having to parse and build new execution plans.
- **Always set an upper limit for your variable length patterns.** It's easy to have a query touch all nodes in a graph by mistake.
- Return **only the data you need.** Avoid returning whole nodes and relationships
- Use **PROFILE / EXPLAIN** to analyze the performance of your queries.

POLITECNICO
MILANO 1863

References

- https://neo4j.com/

POLITECNICO
MILANO 1863

SYSTEMS AND METHODS FOR BIG AND UNSTRUCTURED DATA

Graph Databases – Neo4J

Marco Brambilla

marco.brambilla@polimi.it

@marcobrambi

POLITECNICO
MILANO 1863