Graph Database Models

Eduart Uzeir

eduart.uzeir@studio.unibo.it

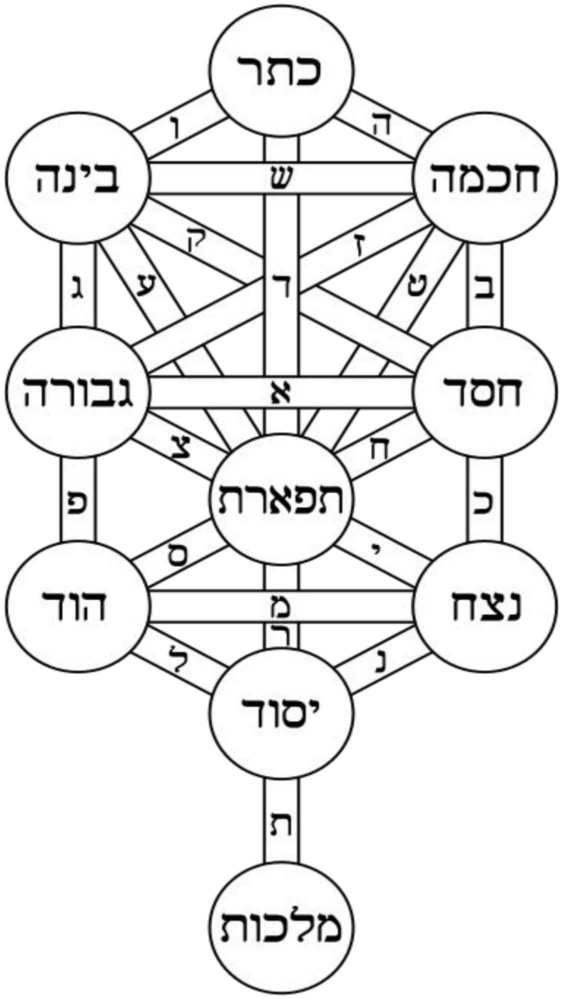
eduart.uzeir@gmail.com

Master Degree in Computer Science

2017-2018

Introduction

“A graph-oriented database is a type of NoSQL database where graph-theory concepts are used to store, model and query data. In this sense a graph database can be viewed as a collection of nodes and edges.”



In the following slides we will talk about:

- Graphs and DB general concepts
- A historical review of the data models
- Graph-oriented data models
- Graph queries classification
- Graph Database Management Systems
- A real world example: Neo4j
- Conclusions

Preliminary

Graph definitions, operations and related terminology:

- A graph is a mathematical structure represented like a set of pairs, “related” in some kind of relationship.

G = [(V(G), E(G)), I(G)]

- Some graph operations:

add_vertex(*G*, *x*): remove_vertex(*G*, *x*): add_edge(*G*, *x*, *y*): remove_edge(*G*, *x*, *y*): join(G, H): and other operations for creating a new graph, finding the shortest path, neighbours, complementation, difference, union etc.

- Terminology:

- **Finite** graph: when the sets of V and E are finite.
- **Path**: is a linear sequence of adjacent vertices.
- **Simple** graph: a graph that has no loops and parallel edges.
- **Connected** graph: if for any partition of the V set in two non empty sets X and Y there is an edge with one end in the X set and the other in the Y set, the graph is called **Disconnected** otherwise.
- **Degree** of a vertex *u* in G: is the number of edges in G incident with *u*.
- The **Order** of a graph is the number of the vertices and **Size** the number of its edges.
- **Directed**: where the edges have directions, otherwise **Undirected**.
- **Labeled** graph: a graph in which the edges or the vertices or both have a label.
- **Hypergraph**: is a pair (X, E) where X is a set of vertices and E is a set of subsets of X.
- **Multigraph**: is a graph that contain multiple edges (parallel edges).
- **Hypernode**: is a graph whose nodes can be themselves graphs.

Preliminary

Database concepts and terminology:

- A **DBMS** is a collection of interrelated data (**Database**) and a set of programs to access and to manipulate those data (**Data Language**) in a safe, convenient and efficient way.
- One of the advantages of DBMS is the fact that it provides **data abstraction** by dividing the system in three levels: physical, logical and view level.
- An **Instance** of a DB is the collection of data stored in the DB in a particular moment.
- The overall design of a DB is called DB **Schema**.
- **NoSQL** is a broad class of DBMS identified by its non-adherence to the widely used Relational model. NoSQL DB are not primarily built using tables and SQL like data language (usually not **ACID** but **BASE** principles).

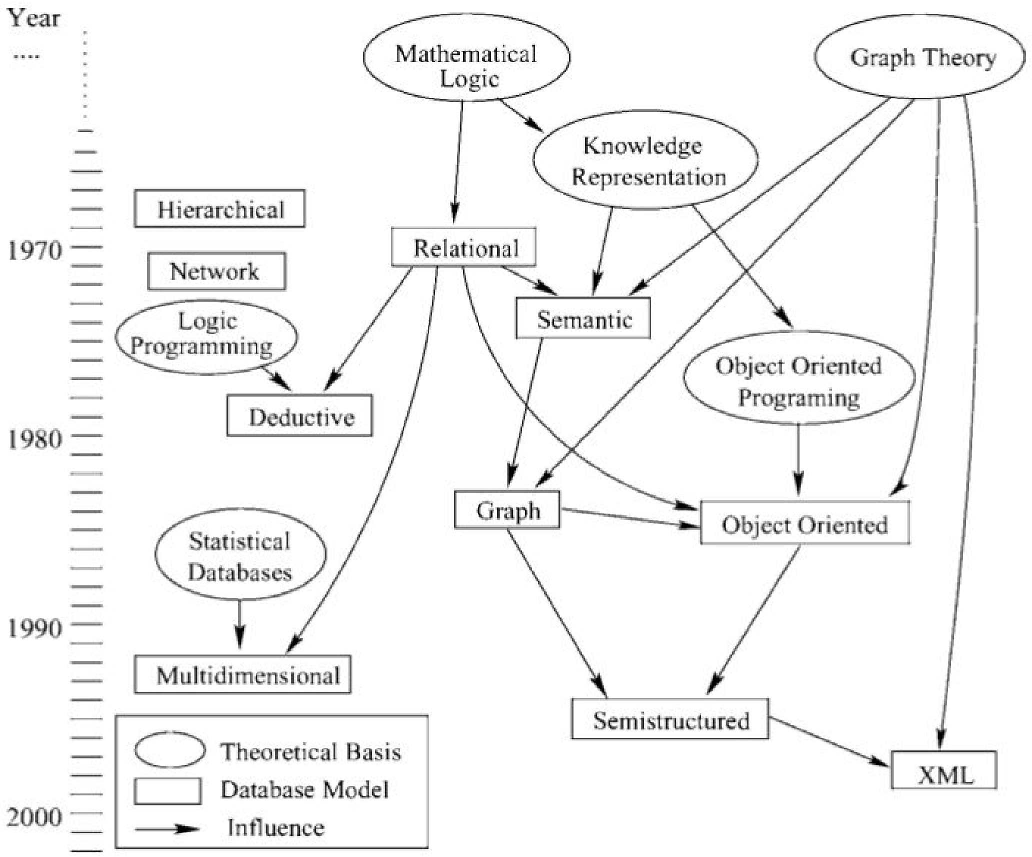
Document	**Key-Value**	**XML**	**Column**	**Graph**
MongoDB CouchDB RavenDB Terrastore	Redis Membase Voldemort MemcacheDB	BaseX eXist	BigTable Hadoop/HBase Cassandra SimpleDB	Neo4j FlockDB InfiniteGraph

- A **Data Model** is a collection of conceptual tools for describing data, data relationships, data semantics and consistency constraints. A data model provides a way to describe the design of a DB at the physical, logical and view level. Every data model can be characterized by three basic components:

a). **data structures** b). **query and transformation language** c). **integrity constraints**

Overview of Data Models

Evolution of database models.

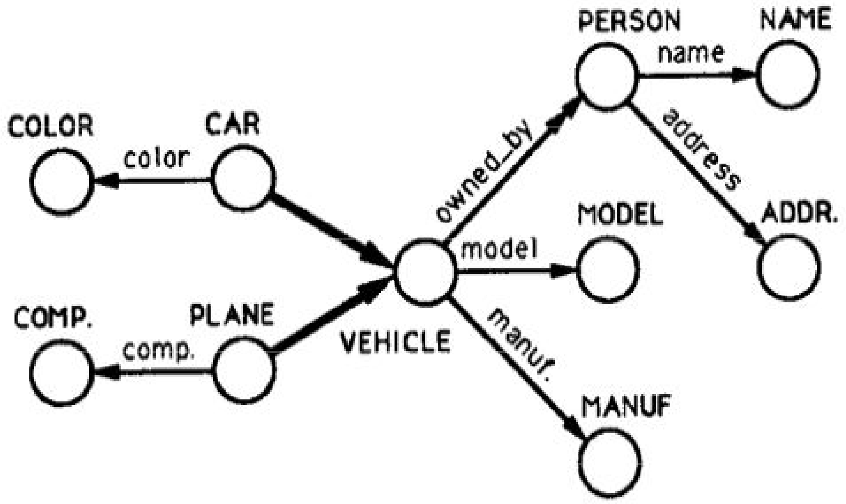


- **Hierarchical** [Tsichritzis, Lochovsky, '78]
- **Network** [Taylor, Frank, '76]
- **Relational** [Codd, '70, '83] introduced the concept of **data abstraction** and the separation of physical and logical level. It is based on **sets** and **relations**. Uses tables to represent data and relationships between data.
- **Semantic DB model** [Peckham, Maryanski, '88] allows to represent objects and their relationships in a natural way in their domains. A well known example of this model is the E-R model [Chen, '76]
- **Object-oriented DB model** [Kim, '90] can be viewed as an extension of the E-R model with notions related to OO programming such as classes, encapsulation ecc.
- **Graph DB model** …
- **Semistructured DB model** [Buneman, '97] model data with flexible structure, **XML** is an example of this model.

Graph data model

Let’s describe the graph data model…

- **Data structures** (data / schema) are represented by graphs or generalization of them (hypergraphs, hypernodes etc).
- **Data manipulation and querying** is based on graph operations and transformations like paths, neighbours, subgraphs, connectivity etc.
- **Integrity constraints** play a fundamental role in all type of DB and are used to enforce data consistency. Some examples are: labels with unique names, domain and range limits and properties.



```
FOR EACH CAR
SUCH THAT
        COLOR(CAR) = "Green" AND
        EXIST PERSON IN IS_OWNED_BY(CAR)
                ADDRESS(PERSON) = "New York"
LIST MODEL(CAR)
```

The figure show a graph modeled schema in Functional Data Model for a vehicle DB and a query in DAPLEX data language that lists the models of green cars owned by New Yorkers. Here an Integrity constraint may be that the color of the car has to be a name, not a number.

Graph Data Models

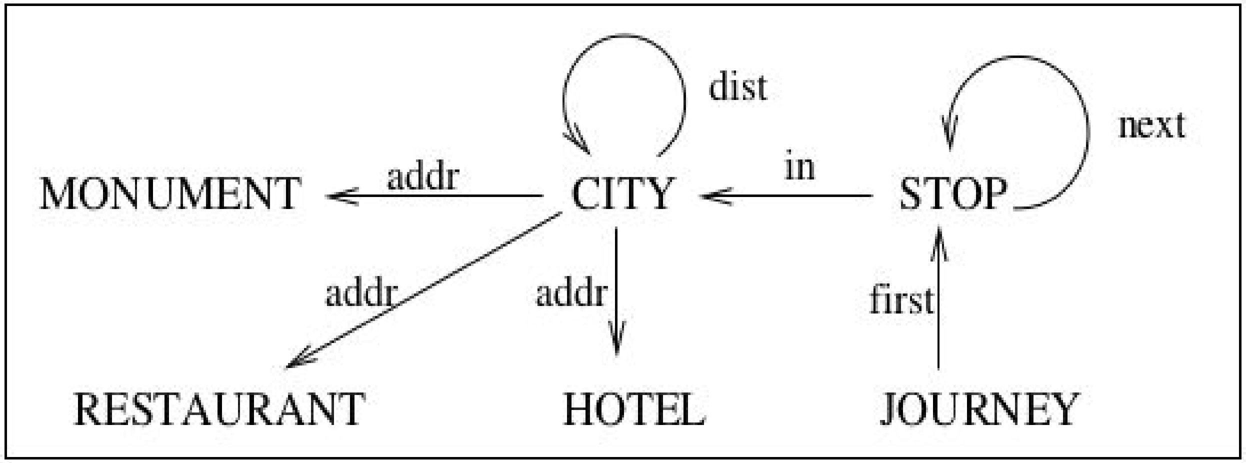
We can classify the graph database models based on the variation of the mathematical definition of the concept of a graph. In this sense the keywords are: directed or undirected graphs, labeled or unlabeled edges and nodes, hypergraphs, hypernode etc.

- The Basic Graph Data Model [directed, labeled graphs]
- The Hypergraph Data Model [hypergraphs]
- The Hypernode Data Model (Nested Graphs)
- The RDF Data Model
- The Property Graph Data Model [directed, labeled, property graphs]

The Basic Graph Data Model

In this model the data structure is a directed graph with labeled nodes and edges.

- **Gram** is a graph data model that is based on directed, labeled graphs and used for hypertext querying. The nodes are labeled with a symbol called ***type*** that has a domain of possible values and the edges label represent the relationship between this types.
- The basic objects in this model are the ***walks*** (paths); an alternating sequence of nodes and edges that begins and ends with nodes. A set of walks is called a ***hyperwalk***.
- As for data language, Gram employ an algebraic language (**hyperwalk algebra**) based on regular expressions that uses node types and edge labels to select and manipulate walks (**walk expressions**).
- The operations in the hyperwalk algebra are: projection, selection, renaming, join etc.
- The figure shows a schema of a Travel Agency in Gram.

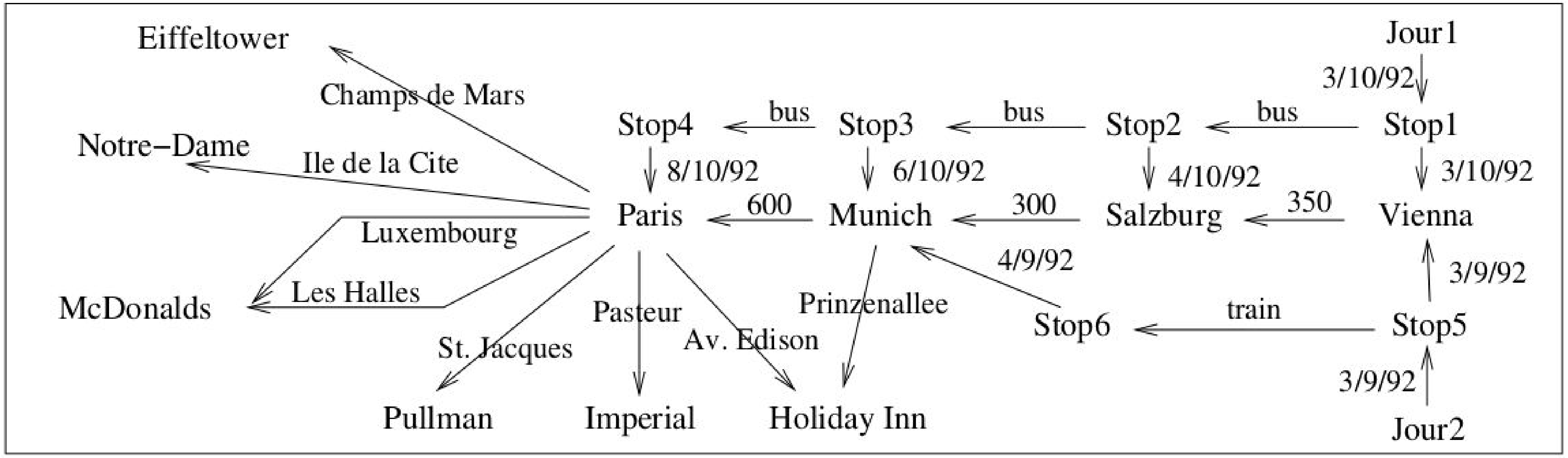


Nodes: Type Document

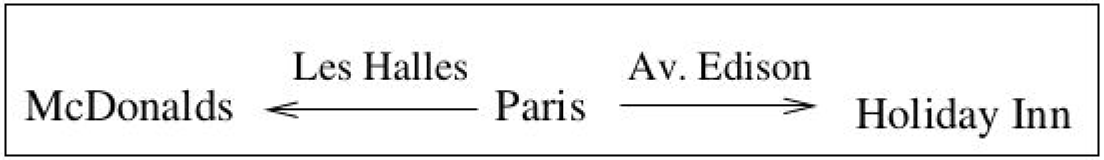
Edges: Type Link

The Basic Graph Data Model

- The figure shows a possible instance of the Travel Agency DB.



- The figure shows a possible hyperwalk in the Travel Agency DB.



- This hyperwalk can be modeled using the following hwe (hyperwalk expression), that is the sum of two we (walk expressions).

CITY addr RESTAURANT + CITY addr HOTEL

The Hypergraph Data Model

This data model is based on hypergraphs that are a sophisticated form of graphs where an edge (hyperedge) can connect any number of node sets.

- **GROOVY** (Graphically Represented Object-Oriented data model with Values) is an object-oriented DB model that is formalized using hypergraphs.
- The data structure of GROOVY is the **object schema** over which the objects that represent real world entities are defined.
- The object schemas include the integrity constraints that are an extension of functional dependencies and can provide data encapsulation.
- The object schema is viewed as a triple <N, F, S> where N is a **value scheme**, F are the **value functional dependencies** and S is **sub-object schema**) that is in a one-to-one correspondence with a hypergraph G = <V, E_d, E_u> where V is the set of vertices, E_d are the directed edges and E_u are the undirected edges.
- The **value scheme** define the real world objects like a pair *(i, u)* where "i" is a unique *object id* and "u" is the current *object value*. For example:

PERSON = {NAME, CHILDRENS} /* this is a value scheme, NAME and CHILDREN are attributes */

O = <i, <John, {Jim, Jill}> > /* this is an example of an object */

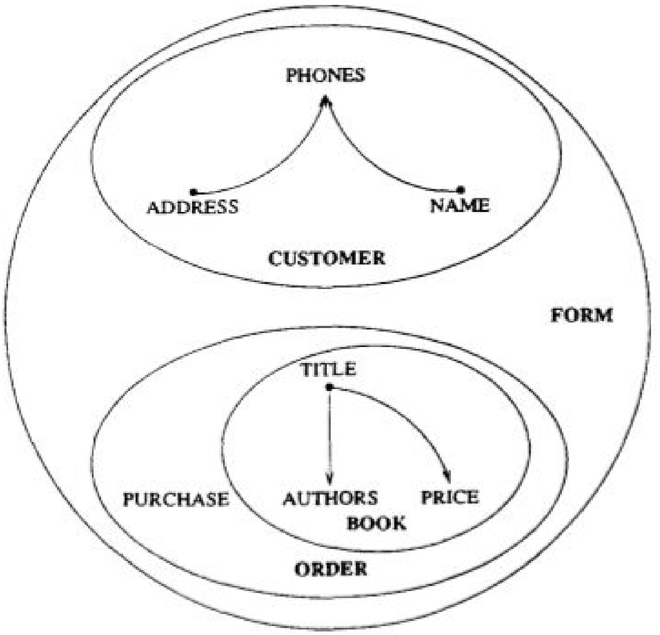
- **VFDs** provides semantic integrity constraints for schema objects in GROOVY and use single-valued or multiple-valued attributes to identify *keys* for data querying.

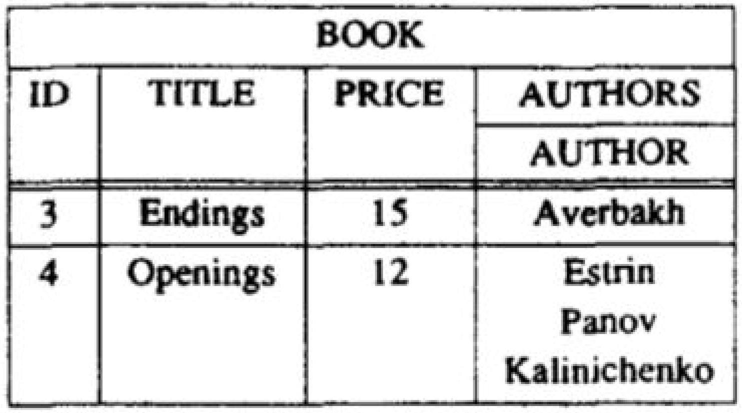
  F = {NAME, ID → STUDENT} /* NAME and ID uniquely identify the STUDENT */
- The **sub-object schema** is the object schema of a sub-object.
- We use **HML** (Hypergraph Manipulating Language) to query and update labelled hypergraphs.

The Hypergraph Data Model

Let’s consider the example of the object schema for a mail order form for books.

- The value schemas are:
  - **CUSTOMER = (NAME, ADDRESS, PHONES}**
  - **BOOK = (TITLE, PRICE, AUTHORS}**
  - **ORDER = (TITLE, PRICE, AUTHORS , PURCHASE} = {attr(BOOK), PURCHASE}**
  - **FORM = (NAME, ADDRESS, PHONES,TITLE, PRICE, AUTHORS , PURCHASE} = { a t t r (C U S T O M E R) , attr(ORDER)}**
- The VFDs for this value schemas are:
  - **F 1 = {NAME,ADDRESS → PHONES}** /* set of VFDs for CONSUMERS */
  - **F 2 = {TITLE → AUTHORS, TITLE → PRICE}** /* VFDs for BOOK and ORDER */
  - **F 3 = {NAME,ADDRESS → PHONES, TITLE → AUTHORS, TITLE --> PRICE}** /* VFDs for FORM */





BOOK			
ID	TITLE	PRICE	AUTHORS
			AUTHOR
3	Endings	15	Averbakh
4	Openings	12	Estrin
			Panov
			Kalinichenko

On the left is the object schema for FORM, on the right an instance of the BOOK value schema.

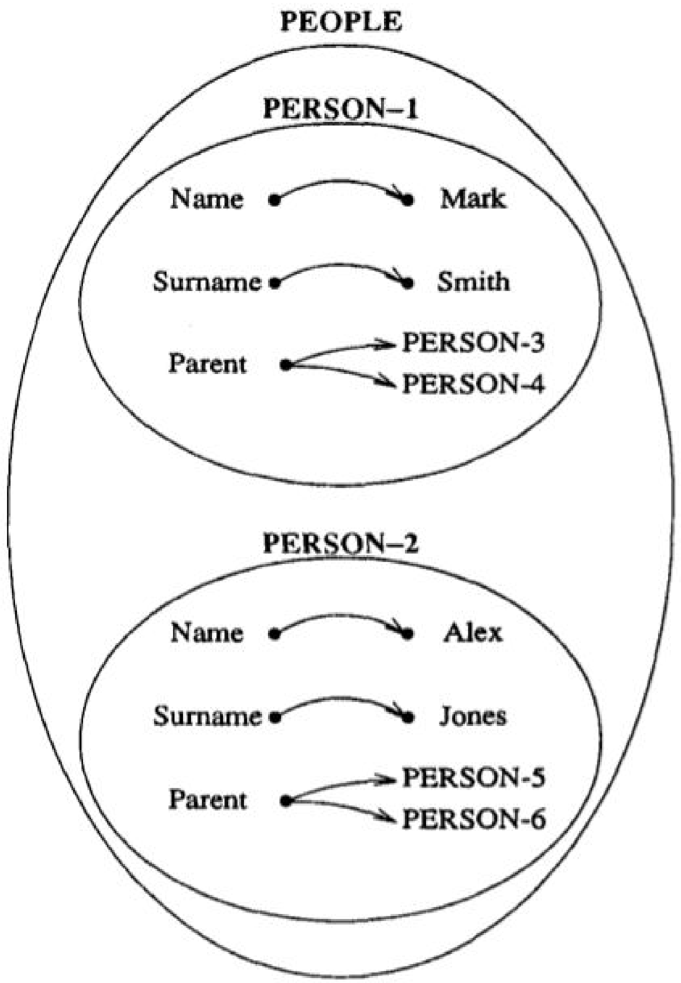
The Hypernode Data Model

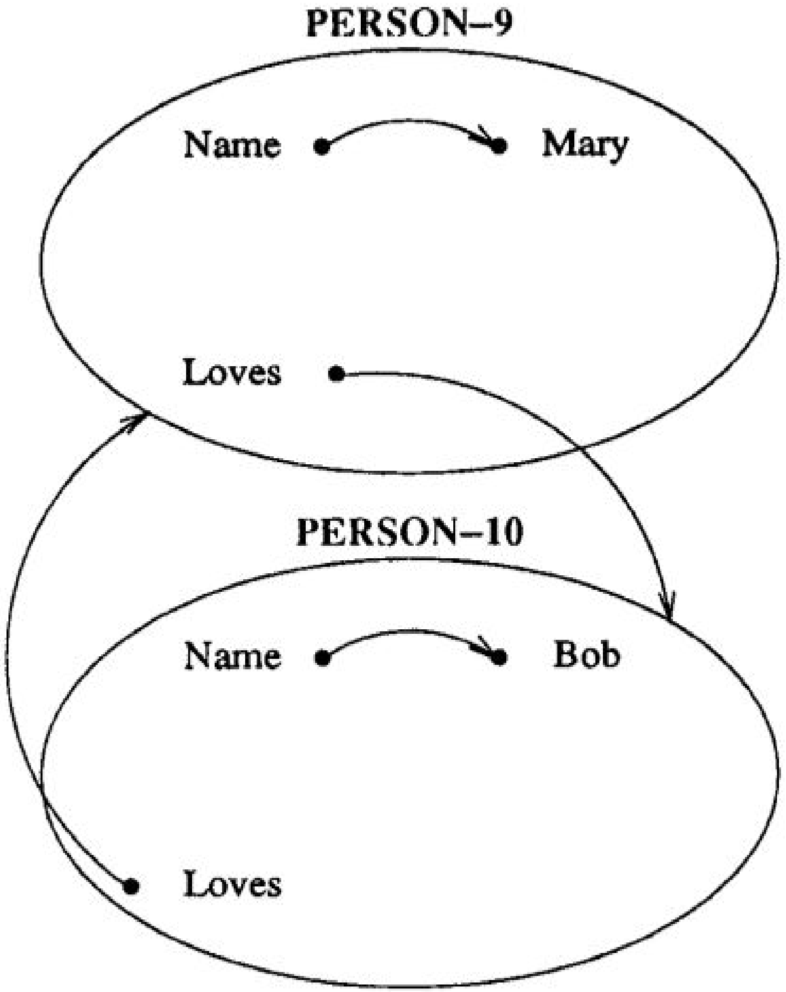
The hypernode data model use as data structure a graph whose nodes can contain nested graphs.

- Hypernodes can be used to represent flat, hierarchical, cyclic and other complex objects and are able to provide a natural support for encapsulation of the information at any level.
- A hypernode is viewed as a triple H = (G, N, E) where G is a unique label, N is a finite set of nodes containing simple graphs or other hypernodes and E is the set of edges between members of N.
- A declarative logic-based query and update language called hypernode programs is used as data language. Hypernode programs can reach Turing Completeness and consists of: Datalog + Negation + Label Generator + Predicates + Functions.
- Besides hypenode programs there are other two data languages defined for the hypernode model, Hyperlog and HNQL (HyperNode Query Language).
- HNQL is mainly used to define HFD (Hypernode Functional Dependencies) in an axiomatic maner.

The Hypernode Data Model

Two examples of hypernodes.





The RDF Data Model

Resource Description Framework (RDF) is a W3C recommendation designed to represent metadata.

- RDF is able to interconnect vast quantity of data in a graph-like structure using labeled graphs called RDF Graphs.
- An atomic RDF expression is a triple (S, P, O) where S called the subject is the resource being described, P is a predicate and represent the property and O is the object or the value of the property.
- A set of atomic RDF expressions create a general RDF expression and is viewed as a graph.
- To query the model is used a data language called SPARQL.
- Example of SPARQL query:

data graph:

```
<http://example.org/book/book1> <http://purl.org/dc/elements/1.1/title> "SPARQL Tutorial"
```

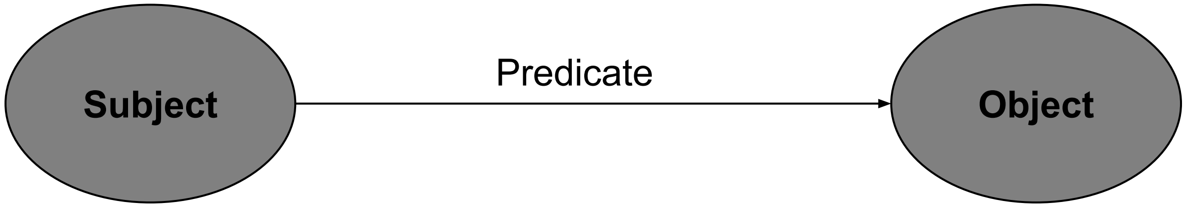
query:

```
SELECT ?title
WHERE
{
  <http://example.org/book/book1> <http://purl.org/dc/elements/1.1/title> ?title
}
```

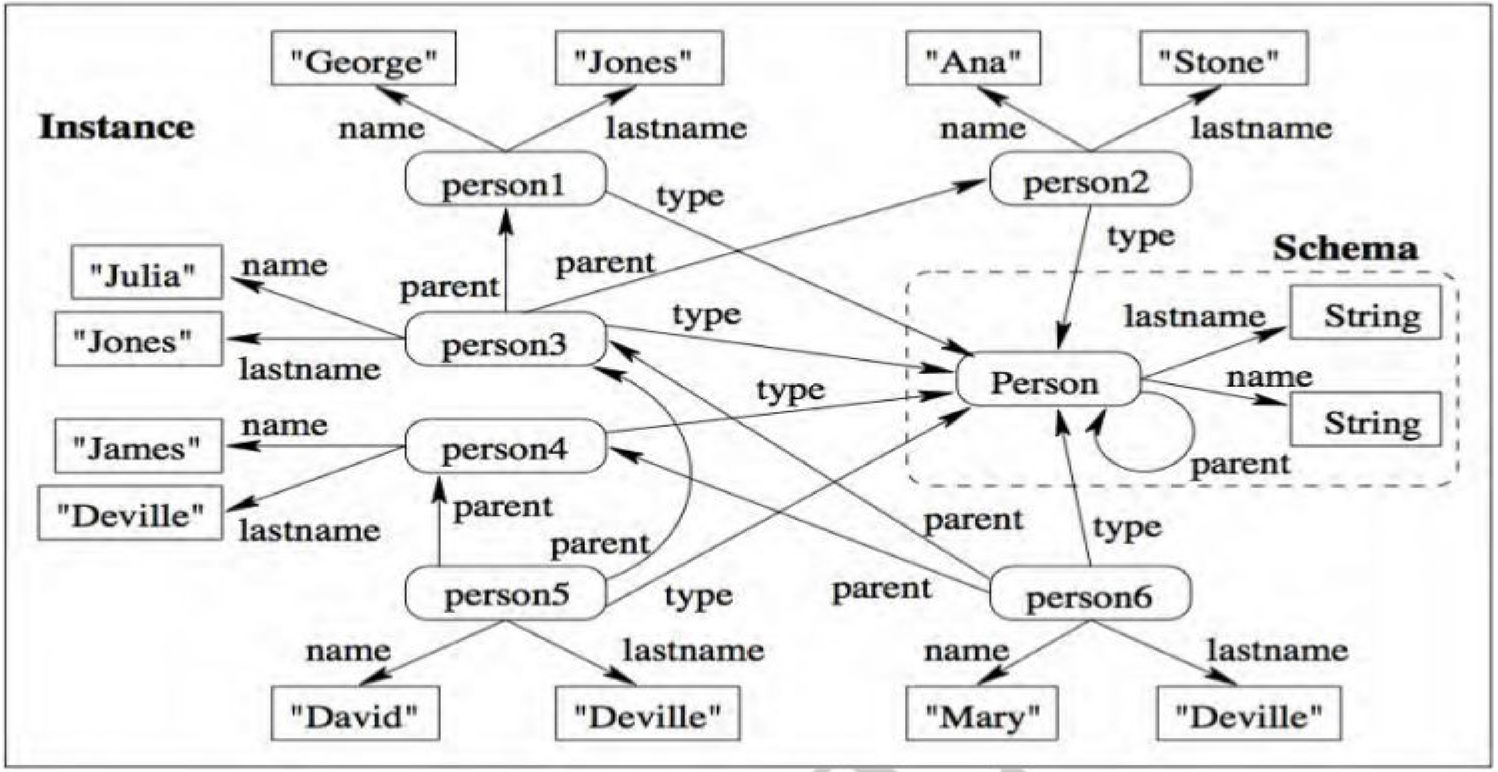
result: title = “SPARQL Tutorial”

The RDF Data Model

The form of a atomic expression in RDF.



Schema and instance of a RDF Graph.



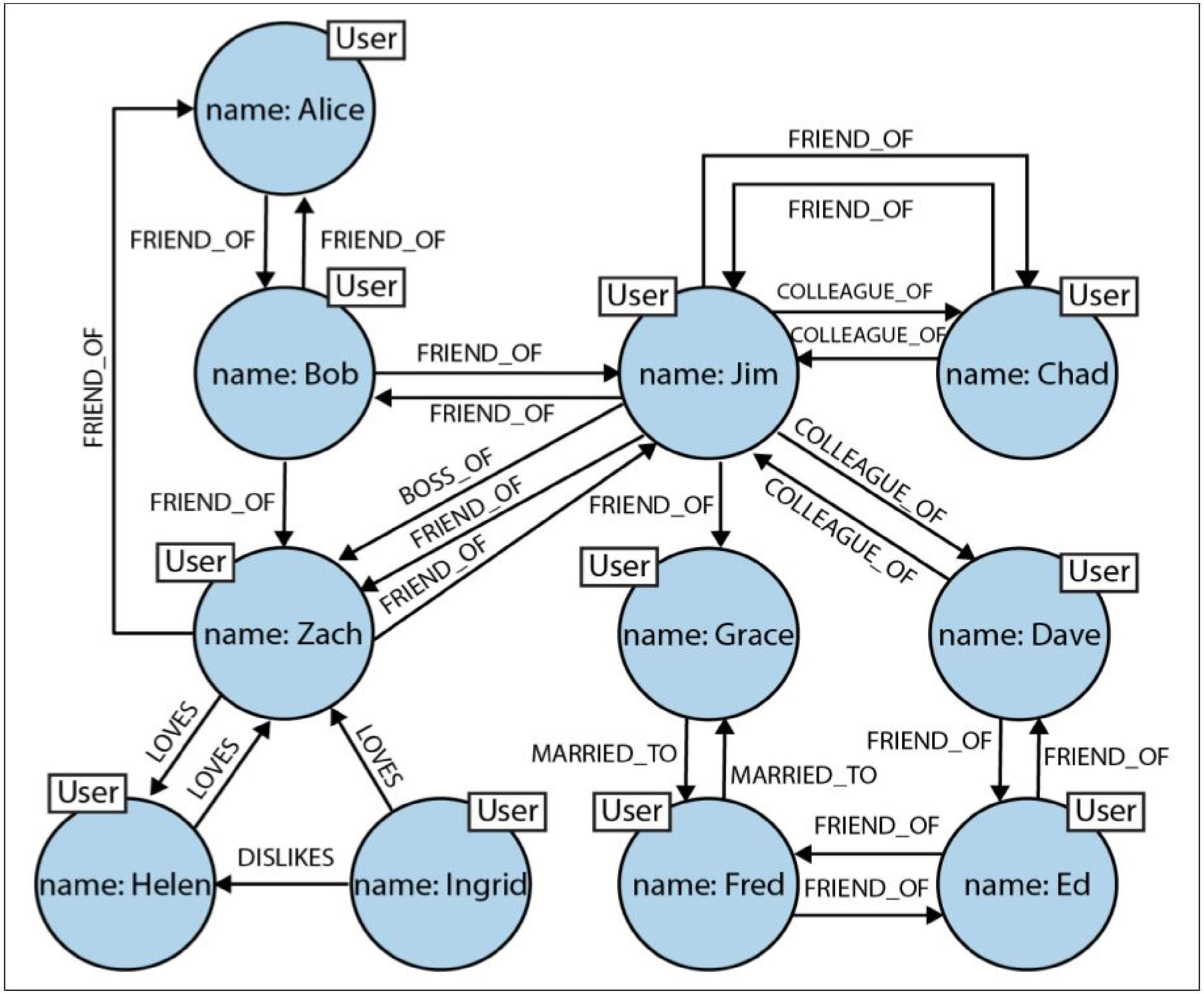
The Property Graph Data Model

The Property Graph Data Model is the most diffuse graph data model. Some notable implementations are Neo4j, InfiniteGraph, Titan, *dex etc.

- The data model is based on directed, labeled multigraphs with properties.
- The properties (attributes) can be assigned to the nodes and to the edges (relationships) and are a <*key:value*> pair.
- There is no required data schema.
- There is not a standard data language for the property graph but there are some candidate implementations like Neo4j’s Cypher, Gremlin etc.

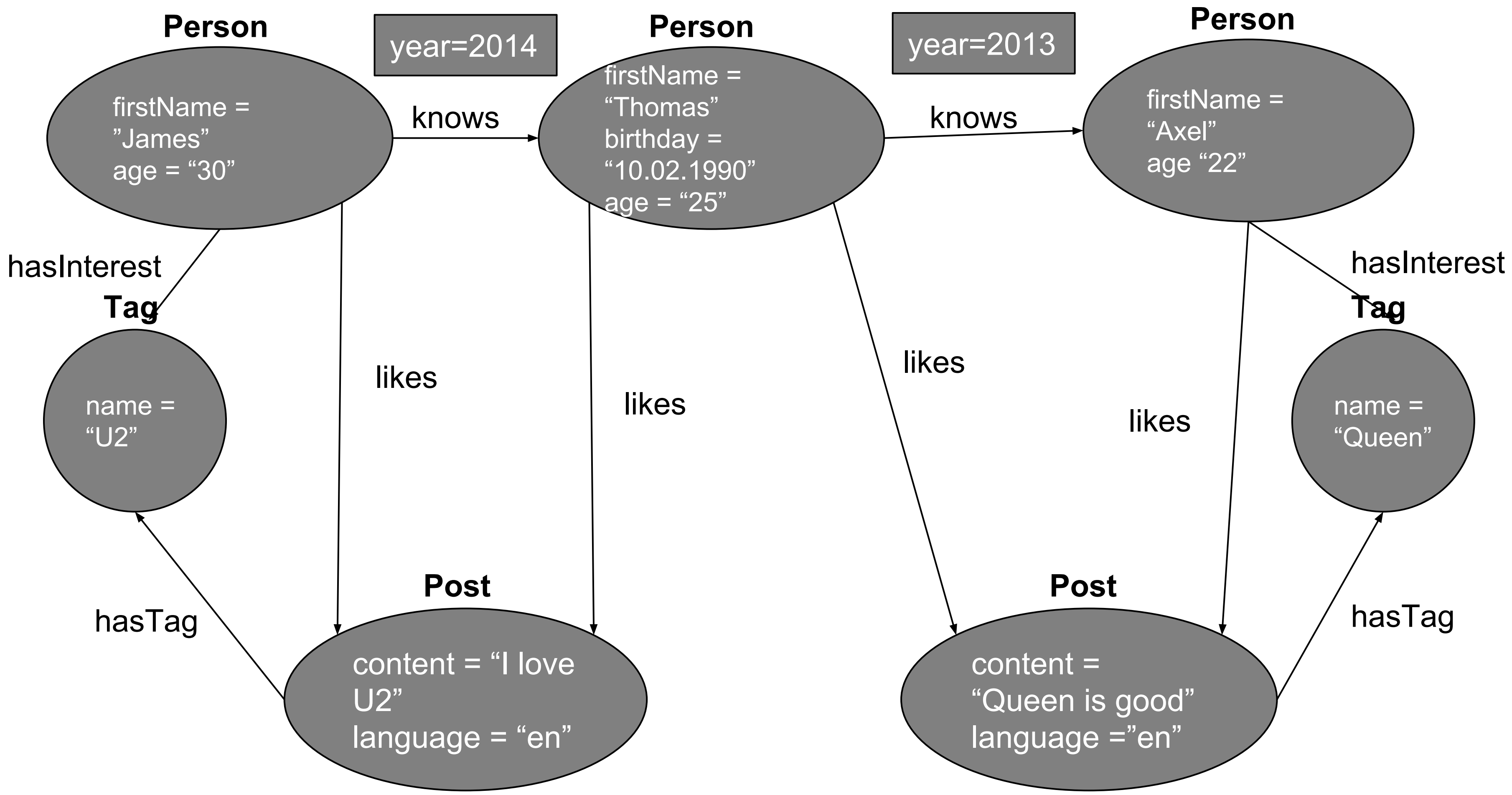
The Property Graph Data Model

Example of a property graph DB modeling friends, colleagues, workers and lovers.



The Property Graph Data Model

Another example of property graph DB.


Person
firstName = "James" age = "30"
year=2014
knows
Person
firstName = "Thomas" birthday = "10.02.1990" age = "25"
year=2013
knows
Person
firstName = "Axel" age "22"
hasInterest
Tag
name = "U2"
likes
likes
likes
likes
hasInterest
Tag
name = "Queen"
hasTag
Post
content = "I love U2" language = "en"
Post
content = "Queen is good" language ="en"
hasTag

A Classification of Graph Queries

- Pattern matching queries: based on the graph definition and search for a subgraph that satisfy the pattern.

  "Return the persons whose attribute firstName is James"

  ```
  MATCH (person : Person)
  WHERE person.firstName = "James"
  RETURN person
  ```

- Adjacency queries: checks if two nodes are connected by edges.

  "Return the pairs of person related by the "knows" relationship"

  ```
  MATCH (person1 : Person)-[: knows] → (person2 : Person)
  RETURN person1, person2
  ```

- Reachability queries: checks if two given nodes are connected by a path.

  "Return the first name of people that can be reached from James by relation knows.

  ```
  MATCH (james : Person) - [: knows* ] - (reachablePerson : Person)
  WHERE mario.firstName = "James"
  Return DISTINCT reachablePerson
  ```

- Summarization queries: summarize the query result (order by, degree, min, max, length, count etc).

  "Return the number of friends of James"

  ```
  MATCH (james : Person) - [: FRIENDS] → (friend : Person)
  WHERE james.firstName = "James"
  RETURN count(friend)
  ```

Graph Database Management Systems

- GDBMS can be divided in two groups:
  - Graph Databases (GDB).
  - Graph Processing Frameworks (GPF).
- GDB is a system designed to manage graph data in a persistent way providing all the tools for creating, saving and querying data. Considering the internal implementation this systems can be classified in two main classes:
  - native graph database (AllegroGraph, InfiniteGraph, Neo4j)
  - non-native graph database (Titan, ArangoDB, OrientDB)
- GPF are used for batch processing and analysis of large and distributed graphs.
  - Some GPF imprelentations are: Pregel, Giraph, GraphLab.

Neo4j

- Neo4j was launched in 2000 and was to result of the effort of a group of 3 people with the aim to ease the performance issues of the relational DB they were using at the time.
- The project evolve from a simple library to a full flavour database system based on the property graph data model, providing better way to model, store and retrieve data in the same time being ACID compliant.
- Neo4j as the name suggest is based in the Java ecosystem and JDK is required for it to work properly.
- Neo4j is a browser-based system and provides a localhost address (http://localhost:7474/browser/) to work with the DB.
- Neo4j is design keeping in mind scalability, fault-tolerance and high availability.
- Neo4j provides his own data language called Cypher.
- Some case of use of Neo4j are:
  - Real-time Recommendation systems
  - Fraud Detection systems
  - Identity & Access Management
  - Social Networking systems
  - Knowledge Graphs (content management, inventory etc)
  - Network & IT Operations (topology analysis)
  - Data Visualization as a way to search for data patterns

SQL vs. Sypher

INSERT INTO User (username) VALUES ("greg")	**CREATE** (u:User {username:"greg"})

SELECT *
 FROM User
 WHERE username = "greg"
	START user=node:User(username="greg")
	 RETURN user

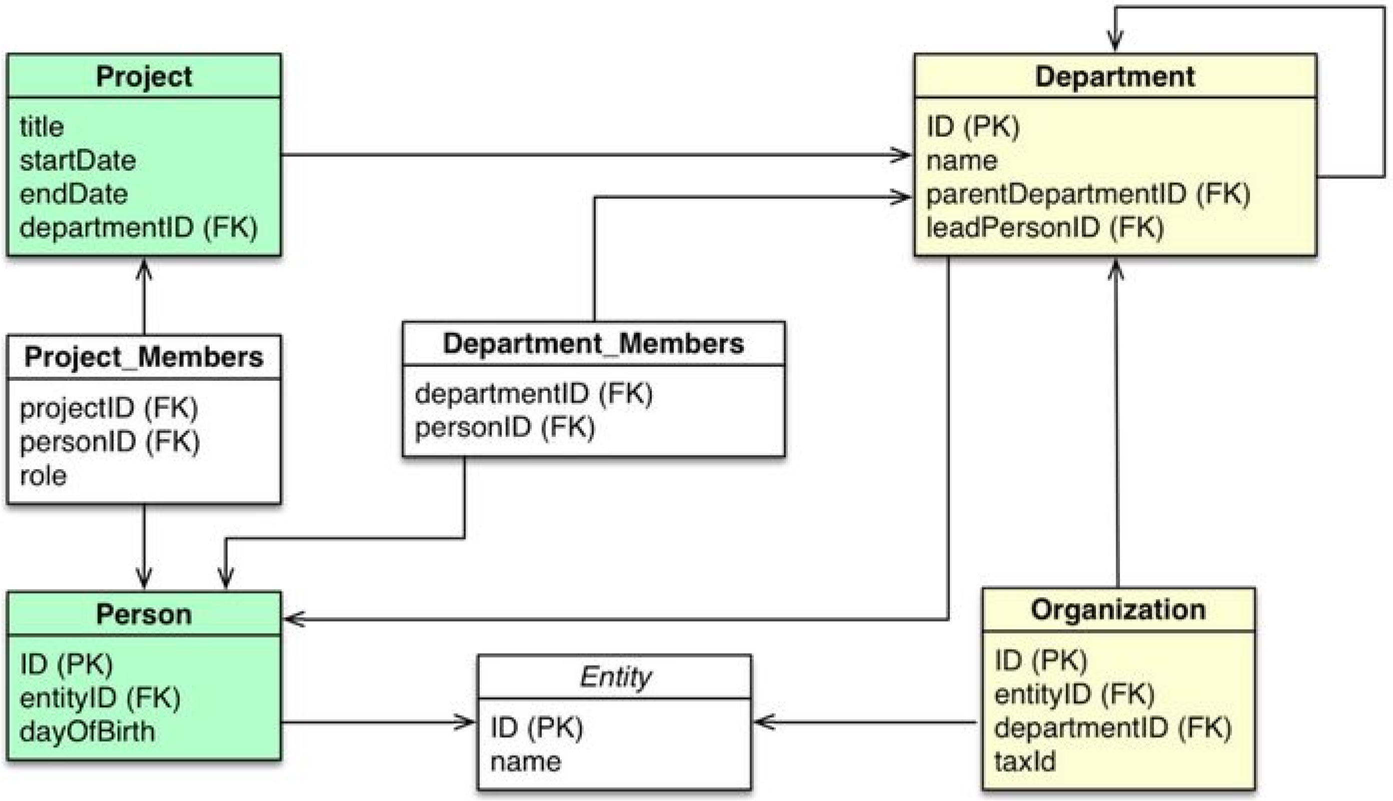
SELECT fullname, email, username
 FROM User
 WHERE username = "greg"
	MATCH (u:User {username: "greg"})
	 RETURN u.fullname, u.email, u.username

UPDATE User
 SET fullname="Greg Jordan"
 WHERE username="greg"
	MATCH (u:User {username: "greg"})
	 SET u.fullname = 'Greg Jordan'
	 RETURN u

DELETE FROM User
 WHERE username="greg"
	MATCH (u:User {username: "greg"})
	 DELETE u

SLQ vs. Cypher - example

The relational schema of a Organization.



A relational data model of an organizational domain.

SLQ vs. Cypher - example 1

“List the employees in the IT Department”.

SQL	Sypher

SELECT name
FROM Person
LEFT JOIN Person_Department
  ON Person.Id = Person_Department.PersonId
LEFT JOIN Department
  ON Department.Id =
Person_Department.DepartmentId
WHERE Department.name = "IT Department"
	MATCH (p:Person)<-[:EMPLOYEE]-(d:Department)
	WHERE d.name = "IT Department"
	RETURN p.name

SLQ vs. Cypher - example 2

"For each customer who bought a product, look at the products that peer customers have purchased and add them as recommendations".

SQL

```
SELECT product.product_name as Recommendation,
count(1) as Frequency
FROM product, customer_product_mapping, (SELECT
cpm3.product_id, cpm3.customer_id
FROM Customer_product_mapping cpm,
Customer_product_mapping cpm2,
Customer_product_mapping cpm3
WHERE cpm.customer_id = ‘customer-one’
and cpm.product_id = cpm2.product_id
and cpm2.customer_id != ‘customer-one’
and cpm3.customer_id = cpm2.customer_id
and cpm3.product_id not in (select distinct product_id
FROM Customer_product_mapping cpm
WHERE cpm.customer_id = ‘customer-one’)
) recommended_products
WHERE customer_product_mapping.product_id =
product.product_id
and customer_product_mapping.product_id in
recommended_products.product_id
and customer_product_mapping.customer_id =
recommended_products.customer_id
GROUP BY product.product_name
ORDER BY Frequency desc
```

Sypher

```
MATCH (u:Customer
{customer_id:'customer-one'})-[:BOUGHT]->(p:Produ
ct)<-
[:BOUGHT]-(peer:Customer)-[:BOUGHT]->(reco:Pro
duct)
WHERE not (u)-[:BOUGHT]->(reco)
RETURN reco as Recommendation, count(*) as
Frequency
ORDER BY Frequency DESC LIMIT 5;
```

Conclusion

- Graph data model provide a very powerful conceptual model for the implementation of NoSQL databases.
- The main purpose is to ease performance bottlenecks, to increase semantics and in the same time tackle the increasing data complexity.
- Graphs have an enormous potential to represent entities and relationships between them in a very natural and intuitive manner.
- Usually GDBMS provide a good scalability, availability and fault-tolerance.
- The query languages are graph-operation based and permits a greater expressiveness than the standard query languages (e.g SQL).
- Graph visualization helps the better understanding of the data interconnectivity and the study of patterns.

References

[1] Survey of Graph Database Models - R. Angles, C. Gutierrez {Main source}
[2] An introduction to Graph Data Management - R. Angles, C. Gutierrez
[3] A Graph-Oriented Object Database Model - M. Gyssene et al.
[4] A Graph-Theoretic Data Model for Genome Mapping Databases - M. Graves et al.
[5] A Graph-Based Data Model and its Ramifications - M. Levene, G. Loizou.
[6] Gram: A Graph Data Model and Query Language - B. Amann, M. Scholl.
[7] https://neo4j.com/
[8] Neo4j Essentials - Sumit Gupta
[9] Professional NoSQL - Shashank Tiwari
[10] Learning Neo4j - Rik Van Bruggen
[11] Practical Neo4j - Greg Jordan
[12] Database System Concepts - A. Silberschatz et al.
[13] ***The Definitive Guide to Graph Databases for the RDBMS Developer***