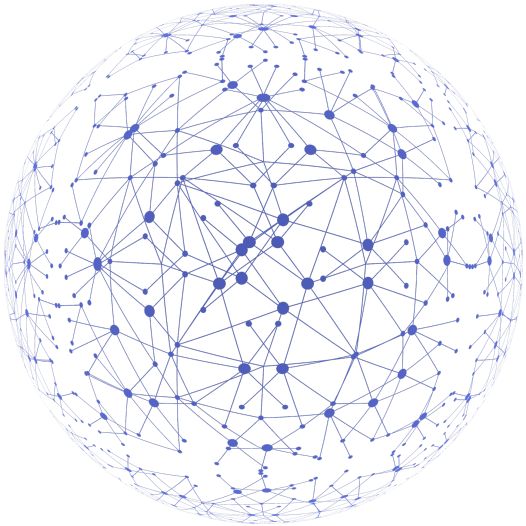
Università di Catania

Dipartimento di Matematica e Informatica
Corso di Big Data
a.a. 2022/2023

Graph DB

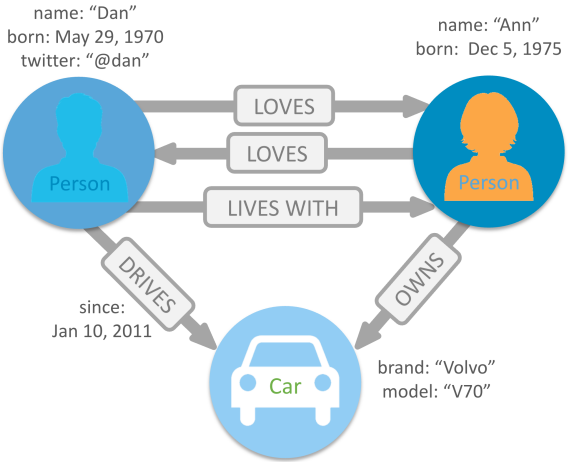
Dott. Roberto Grasso
roberto.grasso@phd.unict.it

Graph DB



- Un graph DB è una piattaforma in grado di creare, manipolare ed interrogare strutture a grafo.
- Il grafo è composto da nodi, archi, etichette e proprietà.
- I dati sono memorizzati in modo tale da agevolare la navigazione del grafo.

Graph DB


name: “Dan”
born: May 29, 1970
twitter: “@dan”
name: “Ann”
born: Dec 5, 1975
LOVES
LOVES
LIVES WITH
Person
Person
DRIVES
since:
Jan 10, 2011
OWNS
Car
brand: “Volvo”
model: “V70”

Graph DB

- I graph DB sono particolarmente utili per la gestione di dati altamente interconnessi e complessi, come le reti sociali e le reti di distribuzione.
- Uno dei principali vantaggi dei graph DB è la loro capacità di gestire grandi quantità di dati interconnessi in modo efficiente. Ciò è dovuto alla loro struttura dati altamente ottimizzata, che consente di recuperare informazioni in modo molto più rapido rispetto ai tradizionali database relazionali.

Graph DB

Quali sono le funzionalità offerte da un graph DB?

- Ricerca di nodi e relazioni: i graph DB consentono di cercare nodi e relazioni in modo efficiente.
- Scoperta di path: si possono analizzare i percorsi tra nodi all'interno del database al fine di trovare, ad esempio, i percorsi più brevi o i percorsi più comuni.
- Community detection: si possono utilizzare algoritmi, come quello di Louvain, al fine di identificare le comunità.
- Similarità: i graph DB consentono di trovare in maniera efficiente nodi con comportamenti o proprietà simili.
- ...

Graph DB

Perché preferire un graph DB ad un database relazionale?

- Gestione di dati altamente interconnessi: i graph DB, come detto in precedenza, sono particolarmente adatti per gestire dati altamente interconnessi, come le reti sociali o le reti stradali. In un database relazionale, la rappresentazione di questa tipologia di dati richiede l'uso di diverse tabelle e relazioni, che possono diventare complesse e difficili da gestire. In un database a grafo, i dati interconnessi sono rappresentati come nodi e relazioni, semplificando la modellazione e l'accesso ai dati.
- Flessibilità: i database relazionali richiedono uno schema rigido per definire le tabelle e le relazioni, che deve essere definito in anticipo. Ciò può limitare la flessibilità del modello dei dati. I graph DB, invece, sono più flessibili e consentono di aggiungere o modificare le entità e le relazioni senza dover modificare lo schema del database.

Graph DB

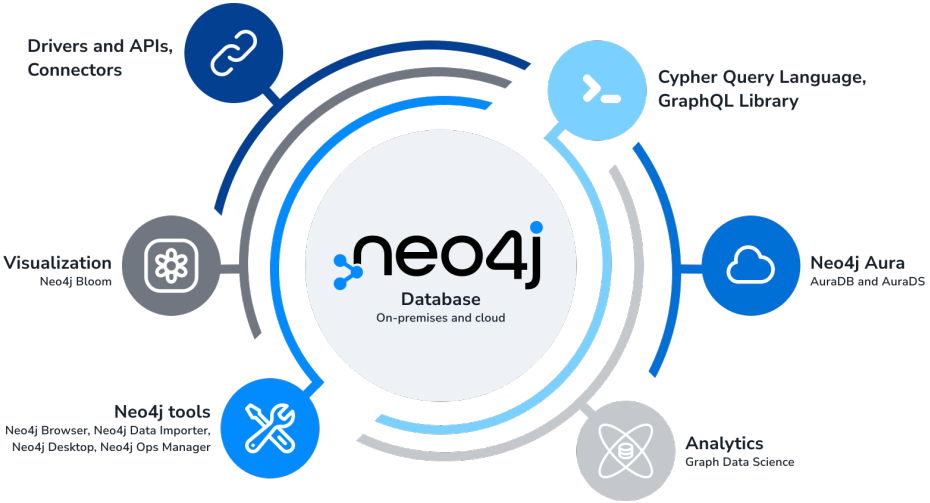
- Prestazioni migliori (per alcune tipologie di query): i database relazionali sono ottimizzati per le query di tipo join, che richiedono di unire più tabelle per recuperare i dati richiesti. Tuttavia, in alcune situazioni, i graph DB possono fornire prestazioni migliori, in particolare per le query che richiedono di esplorare i dati in modo ricorsivo o di identificare le relazioni tra le entità. Inoltre, i graph DB possono supportare query ad hoc più complesse rispetto ai database relazionali.
- ...

Graph DB

Alcuni noti graph DB sono Neo4j, ArangoDB, OrientDB, Titan e Janus Graph.

Neo4j

Noi proveremo Neo4j, uno tra i più usati e performanti graph DB tra quelli che abbiamo citato.



Cypher

- Cypher è il linguaggio utilizzato da Neo4j per interagire con il DB.
- È stato progettato per essere intuitivo, leggibile e facile da usare, anche per gli utenti non esperti di database.
- Il linguaggio Cypher si basa su un'interfaccia a riga di comando, in cui gli utenti possono digitare le loro query direttamente nel prompt.
- Le query Cypher sono divise in clausole, ognuna delle quali esprime un'operazione specifica sui dati. Le clausole più comuni sono MATCH, WHERE, RETURN, CREATE e DELETE.

Rappresentazione dei nodi

(n)	$\rightarrow$	nodo referenziato dalla variabile *n*
(n:Persona)	$\rightarrow$	nodo di tipo *Persona*
(n:Persona:Dirigente)	$\rightarrow$	nodo di tipo *Persona* e *Dirigente*
(n:Persona {nome: 'Marco})	$\rightarrow$	nodo di tipo *Persona* e con proprietà *nome* = Marco
()	$\rightarrow$	nodo non referenziato da una variabile

Rappresentazione degli archi

[]	→	arco anonimo
[r]	→	arco referenziato dalla variabile *r*
[r:AMICO]	→	arco di tipo *AMICO*
[r:AMICO | CONOSCENTE]	→	arco di tipo *AMICO* o *CONOSCENTE*
[r:AMICO {dal: 2020}]	→	arco di tipo *AMICO dal* 2006 (<<*dal* >> è una proprietà)
-[...]->	→	arco uscente
<-[...]-	→	arco entrante
-[...]-	→	arco non orientato

Clausola MATCH

MATCH è la clausola principale in Cypher ed è utilizzata per cercare nodi o relazioni all'interno del database. Consideriamo la seguente query.

MATCH (n:Persona)-[:AMICO]->(m:Persona) RETURN n, m

Questa query restituisce tutti i nodi di tipo *Persona* che sono collegati tra loro da un arco di tipo AMICO.

Clausola WHERE

La clausola **WHERE** viene utilizzata per applicare condizioni alle query MATCH.
Consideriamo la seguente query.

MATCH (n:Persona)-[:AMICO]->(m:Persona) **WHERE** n.nome = 'Marco' RETURN m

Questa query restituisce tutti i nodi di tipo *Persona* collegati a *Marco* attraverso un arco di tipo *AMICO*.

Clausola RETURN

La clausola **RETURN** viene utilizzata per specificare quali dati restituire nella query. Consideriamo la seguente query

MATCH (n:Persona)-[:AMICO]->(m:Persona) **RETURN** n.nome, m.nome

Questa query restituisce i nomi delle persone collegate da un arco di tipo *AMICO*.

Clausola CREATE

La clausola **CREATE** viene utilizzata per creare nuovi nodi o relazioni all'interno del database. Consideriamo la seguente query.

CREATE (n:Persona {nome: 'Marco', età: 30})

Questa query crea un nuovo nodo di tipo *Persona* con le proprietà *nome* e *età* impostate su *Marco* e *30*, rispettivamente.

Clausola DELETE

La clausola **DELETE** viene utilizzata per eliminare nodi o relazioni all'interno del database. Consideriamo la seguente query.

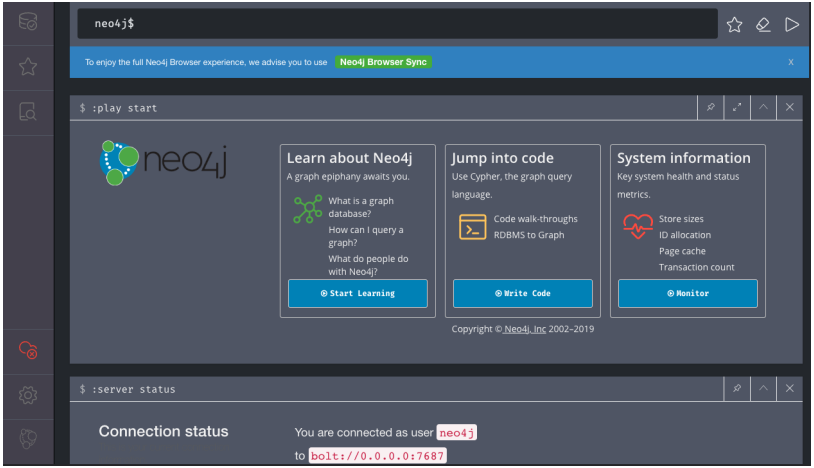
MATCH (n:Persona {nome: 'Marco'}) **DELETE** n

Questa query elimina tutti i nodi di tipo *Persona* con nome uguale a *Marco*.

Nel materiale del corso troverete tutto il necessario per eseguire Neo4j in maniera totalmente containerizzata.

- Per avviare il container basta lanciare (dalla cartella *graphdb*) il seguente comando:
  `docker compose up -d`
- Arrestare il container:
  `docker compose stop`
- Neo4j Web UI:
  `http://localhost:7474/browser/`
  Username: neo4j
  Password: password
- Accedere al container:
  `docker exec -it neo4j-container /bin/bash`
- Accedere alla cypher shell (dal container):
  `$NEO4J_HOME/bin/cypher-shell`
- Uscire dalla cypher shell:
  `:exit`

Neo4J Web UI


neo4j$
To enjoy the full Neo4j Browser experience, we advise you to use Neo4j Browser Sync
$:play start
neo4j
Learn about Neo4j
A graph epiphany awaits you.
What is a graph database?
How can I query a graph?
What do people do with Neo4j?
Start Learning
Jump into code
Use Cypher, the graph query language.
Code walk-throughs
RDBMS to Graph
Write Code
System information
Key system health and status metrics.
Store sizes
ID allocation
Page cache
Transaction count
Monitor
Copyright © Neo4j, Inc 2002–2019
$:server status
Connection status
You are connected as user neo4j
to bolt://0.0.0.0:7687

Movies Graph

Nel seguente esempio, consideriamo un grafo in cui i nodi rappresentano gli attori e i registi, e gli archi rappresentano la collaborazione tra di loro in uno o più film.
In questa esercitazione impareremo a:

- inserire un grafo all'interno del DB;
- cercare attori e registi;
- trovare specifiche relazioni tra attori e registi;
- trovare path di lunghezza n.

Caricamento di un grafo

Per prima cosa vediamo come caricare un grafo.

- Comando per la creazione di un nodo:

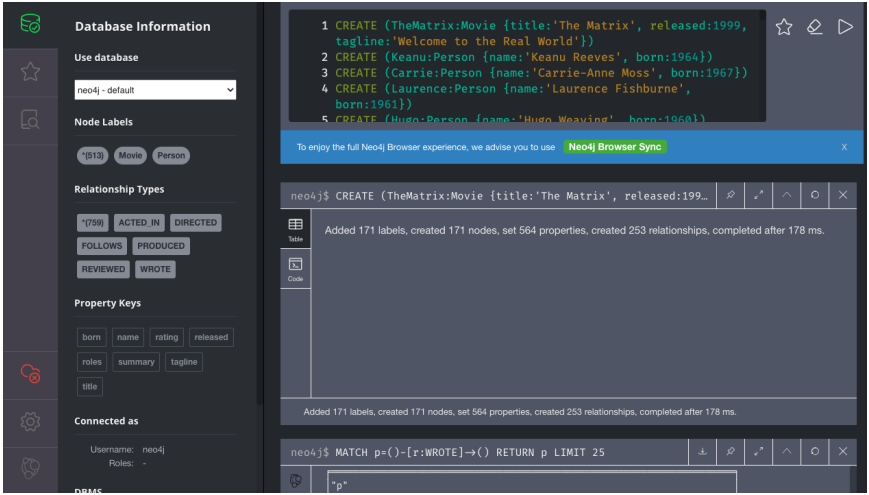
  CREATE (TheMatrix:Movie {title:'The Matrix', released:1999, tagline:'Welcome to the Real World'})

  CREATE (Keanu:Person {name:'Keanu Reeves', born:1964})

- Comando per la creazione di un arco:

  CREATE (Keanu)-[:ACTED_IN {roles:['Neo']}]->(TheMatrix)

Caricamento di un grafo



Nel materiale del corso troverete un file con il grafo completo.

Alcune query

- Cercare l'attore *Tom Hanks*:

  MATCH (tom {name: 'Tom Hanks'}) RETURN tom

- Cercare il film Cloud Atlas:

  MATCH (cloudAtlas {title: 'Cloud Atlas'}) RETURN cloudAtlas

- Cercare 10 persone:

  MATCH (p:Person) RETURN p.name LIMIT 10

Alcune query

- Cercare tutti i film usciti tra il 1990 e il 2000:

  MATCH (m:Movie) WHERE m.released >= 1990 AND m.released < 2000 RETURN m.title

- Cercare tutti i film in cui ha recitato Tom Hanks:

  MATCH (tom:Person {name: 'Tom Hanks'})-[:ACTED_IN]->(tomHanksMovies) RETURN tom,tomHanksMovies

- Cercare il regista di Cloud Atlas:

  MATCH (cloudAtlas {title: 'Cloud Atlas'})<-[:DIRECTED]-(directors) RETURN directors.name

Alcune query

- Cercare tutti gli attori che hanno recitato con Tom Hanks:

  MATCH (tom:Person {name:'Tom Hanks'})-[:ACTED_IN]->(m)<-[:ACTED_IN]-(coActors) RETURN coActors.name

- Trovare i film e gli attori che distano al massimo 4 hop da Kevin Bacon:

  MATCH (bacon:Person {name:'Kevin Bacon'})-[*1..4]-(hollywood) RETURN DISTINCT hollywood

- Trovare il percorso più breve tra Kevin Bacon e Meg Ryan:

  MATCH p=shortestPath((bacon:Person {name:'Kevin Bacon'})-[*]-(meg:Person {name:'Meg Ryan'})) RETURN p

Alcune query

- Cancellare tutti i gli, gli attori e gli archi:

  MATCH (n) DETACH DELETE n

- Verificare che il grafo sia vuoto:

  MATCH (n) RETURN n

Cypher Shell

Tutte le query che abbiamo visto possono essere anche essere eseguite sulla Cypher Shell.

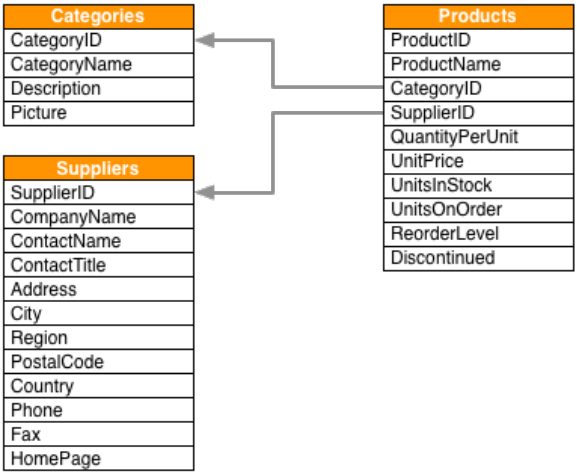
```
root@9f09795db3ab:/var/lib/neo4j# $NEO4J_HOME/bin/cypher-shell
username: neo4j
password: ********
Connected to Neo4j 4.0.0 at neo4j://localhost:7687 as user neo4j.
Type :help for a list of available commands or :exit to exit the shell.
Note that Cypher queries must end with a semicolon.
neo4j@neo4j> MATCH (bacon:Person {name:'Kevin Bacon'})-[*1..4]-(hollywood) RETURN DISTINCT hollywood LIMIT 2;
+--------------------------------------------------+
| hollywood                                        |
+--------------------------------------------------+
| (:Person {name: "Frank Darabont", born: 1959})   |
| (:Person {name: "Tom Hanks", born: 1956})        |
+--------------------------------------------------+

2 rows available after 147 ms, consumed after another 15 ms
neo4j@neo4j> :exit

Bye!
root@9f09795db3ab:/var/lib/neo4j#
```

Migrazione da Database Relazionale a Graph DB

Vediamo un esempio in cui migriamo da un database relazionale a un graph DB.



Caricamento da file CSV

Vediamo un altro modo per importare un grafo: caricamento da file CSV.

- Caricamento dei prodotti:

```
LOAD CSV WITH HEADERS FROM "file:///examples/northwind/products.csv" AS row
CREATE (n:Product)
SET n = row,
n.unitPrice = toFloat(row.unitPrice),
n.unitsInStock = toInteger(row.unitsInStock), n.unitsOnOrder = toInteger(row.unitsOnOrder),
n.reorderLevel = toInteger(row.reorderLevel), n.discontinued = (row.discontinued <> "0")
```

Caricamento da file CSV

- Caricamento delle categoria:

  LOAD CSV WITH HEADERS FROM "file:///examples/northwind/categories.csv" AS row
  CREATE (n:Category)
  SET n = row

- Caricamento dei fornitori:

  LOAD CSV WITH HEADERS FROM "file:///examples/northwind/suppliers.csv" AS row
  CREATE (n:Supplier)
  SET n = row

Creazione degli indici

- Indice per i prodotti:

  CREATE INDEX FOR (n:Product) ON (n.productID)

- Indice per le categorie:

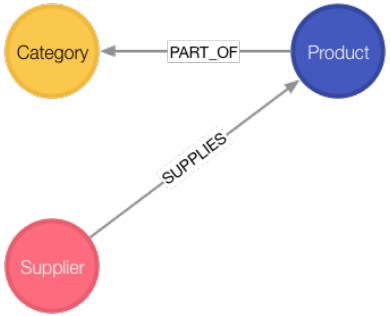
  CREATE INDEX FOR (n:Category) ON (n.categoryID)

- Indice per i fornitori:

  CREATE INDEX FOR (n:Supplier) ON (n.supplierID)

Creazione degli archi

Nel database relazionale avevamo delle relazioni tra prodotti, categorie e fornitori. Dobbiamo dunque creare gli archi del nostro grafo sfruttando queste relazioni. Anche gli archi si possono caricare tramite file CSV. In questo caso li creiamo tramite delle relazioni perché stiamo simulando una migrazione da un database relazionale.



Creazione degli archi

- Archi Product-Category:

```
MATCH (p:Product),(c:Category)
WHERE p.categoryID = c.categoryID
CREATE (p)-[:PART_OF]->(c)
```

- Archi Product-Supplier:

```
MATCH (p:Product),(s:Supplier)
WHERE p.supplierID = s.supplierID
CREATE (s)-[:SUPPLIES]->(p)
```

Alcune query

- Categorie dei prodotti di ogni fornitore:

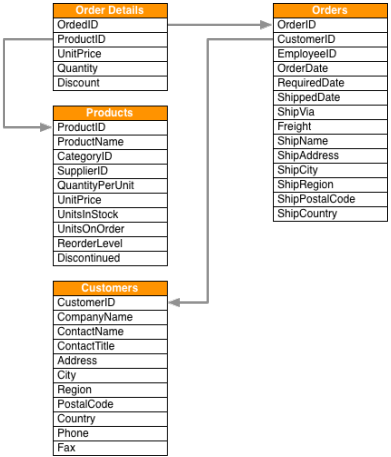
  MATCH (s:Supplier)-[]->(:Product)-[]->(c:Category)
  RETURN s.companyName as Company, collect(distinct c.categoryName) as Categories

- Fornitori per ciascun prodotto:

  MATCH (c:Category categoryName:"Produce")<-[]-(:Product)<-[]-(s:Supplier)
  RETURN DISTINCT s.companyName as ProduceSuppliers

Completamento della migrazione

Per completare la migrazione, aggiungiamo altre tre entità.



Completamento della migrazione

- Caricamento dei clienti:

  LOAD CSV WITH HEADERS FROM "file:///examples/northwind/customers.csv" AS row
  CREATE (n:Customer)
  SET n = row

- Caricamento degli ordini:

  LOAD CSV WITH HEADERS FROM "file:///examples/northwind/orders.csv" AS row
  CREATE (n:Order)
  SET n = row

Completamento della migrazione

- Indice per i clienti:

  CREATE INDEX FOR (n:Customer) ON (n.customerID)

- Indice per gli ordini:

  CREATE INDEX FOR (n:Order) ON (n.orderID)

- Archi Customer-Order:

  MATCH (c:Customer),(o:Order)
  WHERE c.customerID = o.customerID
  CREATE (c)-[:PURCHASED]->(o)

Completamento della migrazione

- Caricamento dei dettagli di ciascun ordine

```
LOAD CSV WITH HEADERS FROM "file:///examples/northwind/order-details.csv" AS row
MATCH (p:Product), (o:Order)
WHERE p.productID = row.productID AND o.orderID = row.orderID
CREATE (o)-[details:ORDERS]->(p)
SET details = row,
details.quantity = toInteger(row.quantity)
```

Esempio

- Numero totale di prodotti acquistati da ogni cliente:
  MATCH (cust:Customer)-[:PURCHASED]->(:Order)-[o:ORDERS]-¿(p:Product),
  (p)-[:PART_OF]->(c:Category categoryName:"Produce")
  RETURN DISTINCT cust.contactName as CustomerName, SUM(o.quantity) AS
  TotalProductsPurchased

Link Utili

Algoritmi

URL: https://neo4j.com/docs/graph-data-science/current/algorithms/

Neo4j Desktop

URL: https://neo4j.com/developer/neo4j-desktop/

Graph Data Science

URL: https://neo4j.com/docs/graph-data-science-client/current/getting-started/

Neo4j su cloud

URL: https://neo4j.com/developer/guide-cloud-deployment/

Northwind

URL 1: https://github.com/neo4j-graph-examples/northwind

Northwind Recommendation

URL 2: https://neo4j.com/graphgists/northwind-recommendation-engine/