# Big Data Storage and Processing

MSc in Data Analytics

CCT College Dublin

## Graph Databases

## Week 11

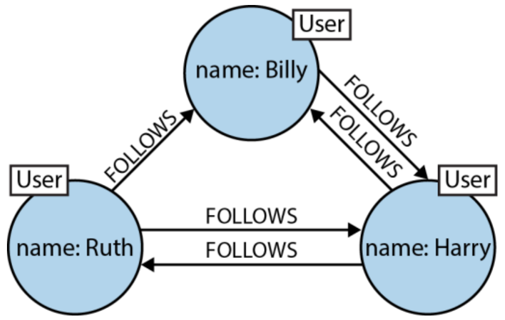
Lecturer: Dr. Muhammad Iqbal*

Email: miqbal@cct.ie

## Agenda

## Introduction

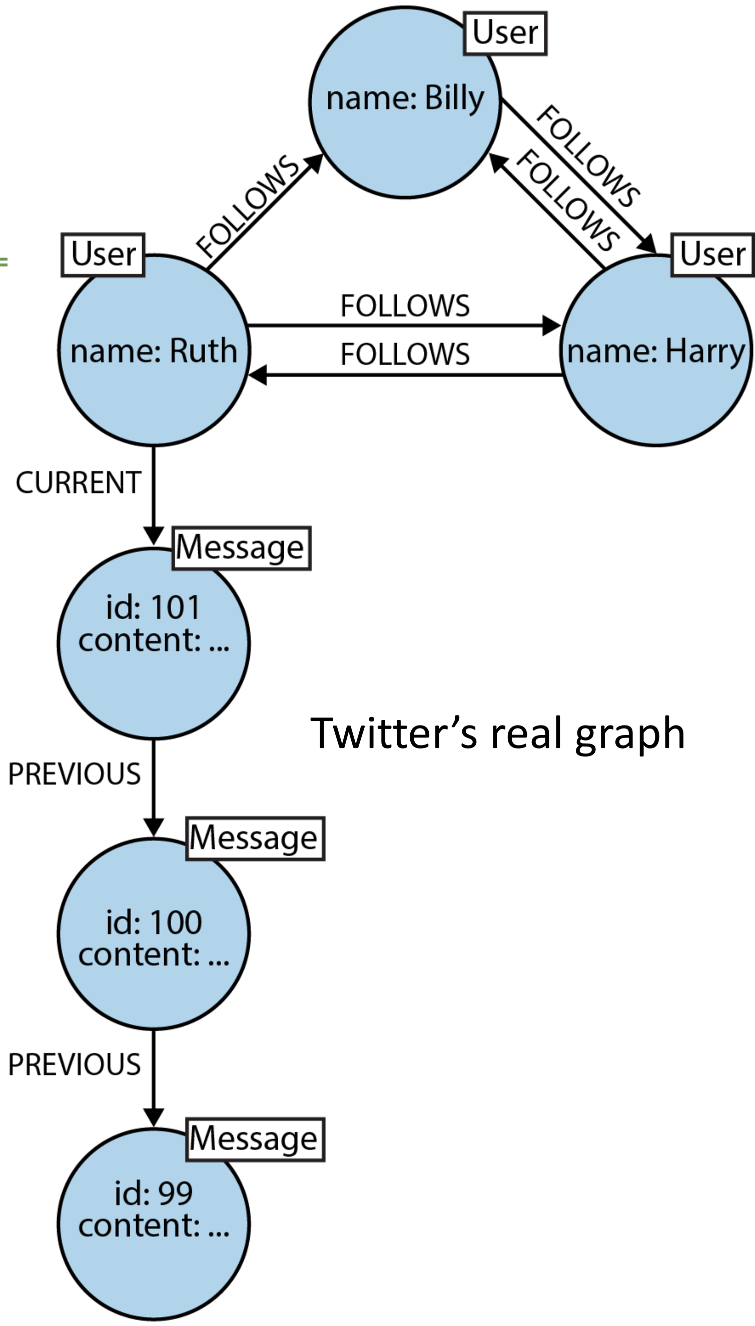
- Graph databases are categorized as NoSQL databases because they use a graph-based data model as opposed to relational databases.
- A graph is a collection of vertices and edges or, in less intimidating language, a set of nodes and the relationships that connect them.
- The main goal is to effectively store, retrieve, and process data that can be shown as a graph with relationships between nodes (entities) and edges.
- Applications like social networks, recommendation engines, fraud detection, knowledge graphs, and network analysis that require data with many-to-many links are especially well-suited for graph databases.
- Graphs represent entities as nodes and the ways in which those entities relate to the world as relationships.
- We can see a small network of Twitter users in Figure. Each node is labeled User, indicating its role in the network. These nodes are then connected with relationships, which help further to establish the semantic context



For example: **Billy** follows **Harry**, and that **Harry** follows **Billy**. **Ruth** and **Harry** likewise follow each other, but sadly, although **Ruth** follows **Billy**, **Billy** hasn't (yet) reciprocated.

# Graphs are Everywhere

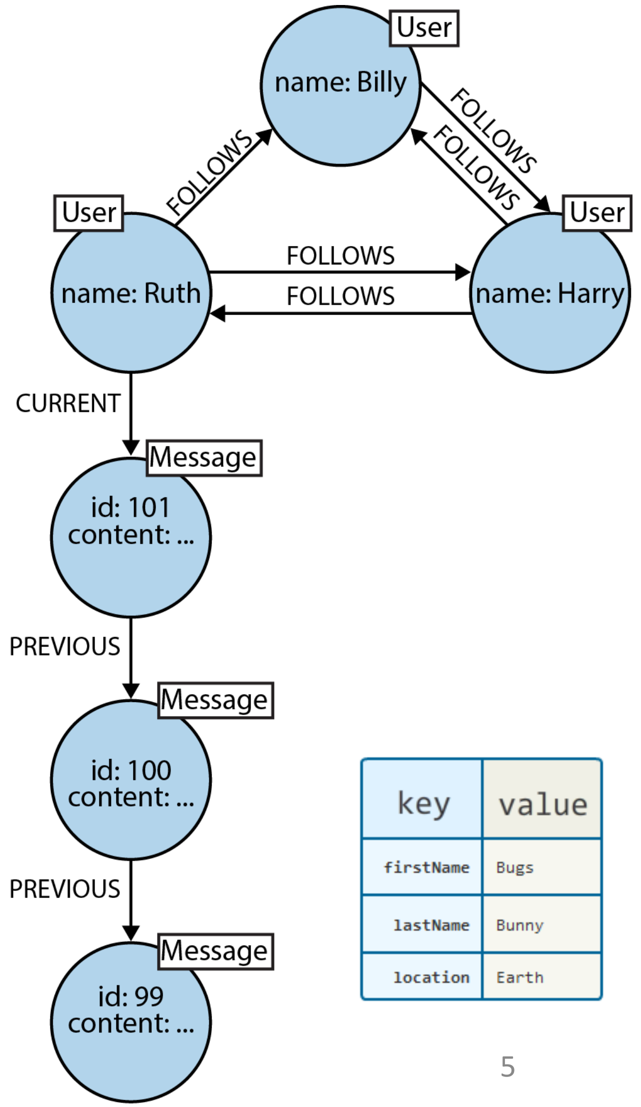
- Graphs are extremely useful in understanding a wide diversity of datasets in fields such as science, government, and business.
- Relationships between data points are easily handled by graph databases. They facilitate quick traversal of relationships, which makes it simpler to respond to inquiries such as "Who are my friends' friends?" and "How are these concepts related?"
- Graph databases, in contrast to conventional relational databases (SQL), are typically not schema-specific. This implies that we do not have to specify your data's strict structure up front. As your application develops, you can add additional kinds of nodes and associations.
- Gartner, for example, identifies five graphs in the world of business: **social, intent, consumption, interest**, and **mobile** and says that the ability to leverage these graphs provides a "sustainable competitive advantage."

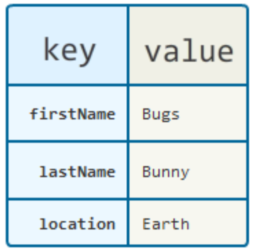


Twitter's real graph

# Labeled Property Graph Model

- We introduced the most popular form of graph model, the labeled property graph. A labeled property graph has the following characteristics
  - ***It contains nodes and relationships.***
  - ***Nodes contain properties (key-value pairs).***
  - ***Nodes can be labeled with one or more labels.***
  - ***Relationships are named and directed, and always have a start and end node.***
  - ***Relationships can also contain properties.***
- The property graph model is intuitive and simple to grasp for most individuals. It can be utilised to explain the vast majority of graph use cases in ways that provide us with relevant data insights.





| key | value |
|---|---|
| firstName | Bugs |
| lastName | Bunny |
| location | Earth |

# High-Level View of the Graph Space

- A plethora of new projects and applications for preserving, altering, and analysing graphs have flooded the market in recent years.

- We discuss a high-level framework for making sense of the emerging graph landscape. The graph space can be divided into two parts.

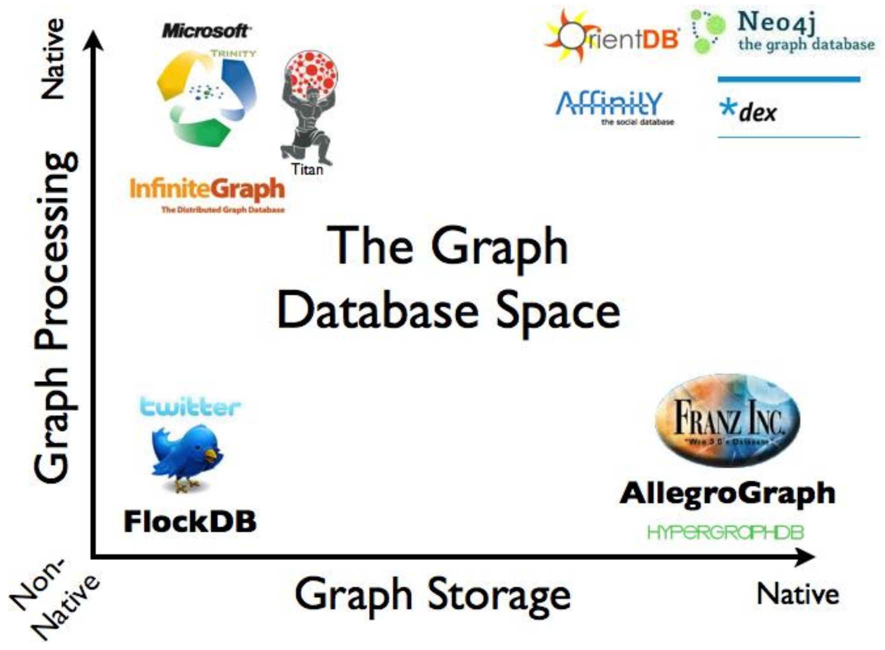
1. Technologies used primarily for transactional online graph persistence, accessed directly in real time from an application.
   - **These technologies are called graph databases. They are the equivalent to “normal” online transactional processing (OLTP) databases in the relational world.**
2. Technologies used primarily for offline graph analytics, performed as a series of batch steps.
   - **These technologies can be called graph compute engines. They can be thought of as being in the same category as other technologies for analysis of data in bulk, such as data mining and online analytical processing (OLAP).**

## Graph Databases

- A graph database management system (a graph database) is an online database management system with **Create, Read, Update, and Delete (CRUD) methods** that expose a graph data model.
- **Graph Databases** are built for use with transactional (**OLTP**) systems. There are two properties of graph databases we should consider when investigating graph database technologies
- **The underlying storage**
  - Some graph databases use native graph storage that is optimized and designed for storing and managing graphs.
  - Some serialize the graph data into a relational database, an object-oriented database, or some other general-purpose data store.
- **The processing engine**
- We take a slightly broader view: any database that from the user's perspective behaves like a graph database (i.e., exposes a graph data model through CRUD operations) qualifies as a graph database.

# Graph Databases

- The graph data model treats relationships as first-class citizens.
- In other DBMSs, where we have to infer connections between entities using things like foreign keys or out-of-band processing such as map-reduce.
- By assembling the simple abstractions of nodes and relationships into connected structures, graph databases enable us to build sophisticated models that map closely to our problem domain.
- Traditional relational databases may not perform as well as graph databases for some kinds of queries involving graph-related processes, particularly when relationship complexity rises.



- Figure 3 shows a pictorial overview of some of the graph databases on the market today, based on their storage and processing models.

# Graph Compute Engines

- A graph compute engine is a technology that enables global graph computational algorithms to be run against large datasets.
- Graph compute engines are designed to identify clusters in your data, or answer questions such as, "**how many relationships, on average, does everyone in a social network have?**"

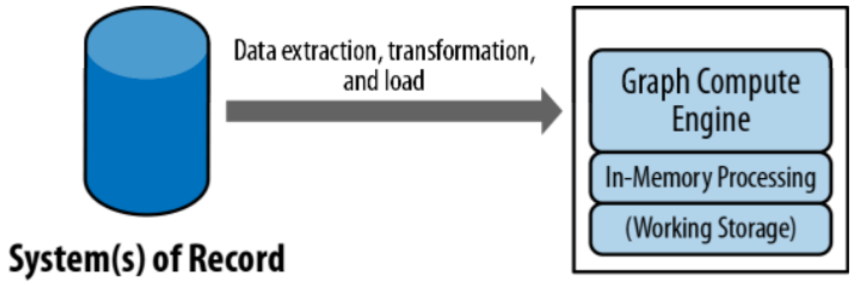


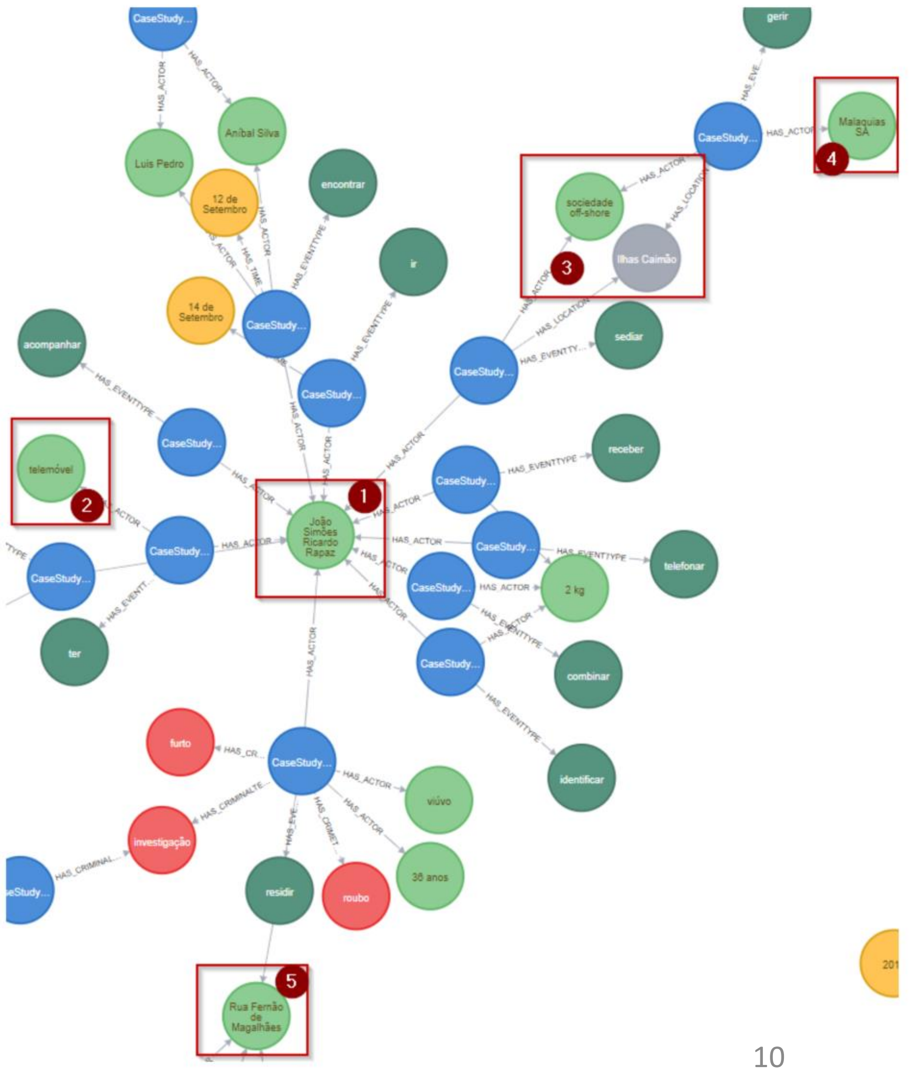
Figure 4. A high-level view of a typical graph compute engine deployment

- Some graph compute engines have a graph storage layer, whereas others are concerned with processing data from an external source and providing the results for storage elsewhere.
- Figure 4 shows a common architecture for deploying a graph compute engine. The architecture includes a system of record (**SOR**) database with **OLTP** properties (such as MySQL, Oracle, or Neo4j), which services requests and responds to queries from the application (and ultimately the users) at runtime.
- Periodically, an **Extract, Transform, and Load (ETL)** job moves data from the system of record database into the graph compute engine for offline querying and analysis.

# Characteristics of Graph Databases

## Performance

- We can increase the performance of the applications using a graph database that is dealing with connected data versus relational databases and NoSQL stores.
- In contrast to relational databases, where **join-intensive query performance** deteriorates as the dataset gets bigger, with a graph database performance tends to remain relatively constant, even as the dataset grows.
- This is because queries are localized to a portion of the graph.
- As a result, the execution time for each query is proportional only to the size of the part of the graph traversed to satisfy that query, rather than the size of the overall graph.



## Characteristics of Graph Databases

### Flexibility

- Developers and data architects manage the domain requirements in Graph databases nicely and modify the schema in coordination with our growing demands.
- ***<u>The graph data model effectively conveys and accommodates business requirements, allowing IT to operate at business speed.</u>***
- Graphs are additive by nature, we can add new types of associations, new nodes, new labels, and new subgraphs to an existing structure without affecting current queries and application functionality.
- These things have positive implications for developer productivity and project risk.
- The flexibility of the graph model allows us to provide dynamic modelling approach, which would be difficult in the face of changing business requirements with other techniques.
- The maintenance costs to manage the Graph databases is significantly lower as compared to traditional databases, which reduces risk and costs.

# Characteristics of Graph Databases

## Flexibility

- A social network is a popular example of a densely connected, variably-structured network, one that resists being captured by a one-size-fits-all schema or conveniently split across disconnected aggregates.

- The **flexibility** of the graph model has allowed us to add new nodes and new relationships without compromising the existing network or migrating data, the original data and its intent remain intact.

- The graph offers a much richer picture of the network.

- We can see who **LOVES** whom (and whether that love is required). We can see who is a **COLLEAGUE_OF** whom, and who is **BOSS_OF** them all. We can see who is off the market, because they are **MARRIED_TO** someone else

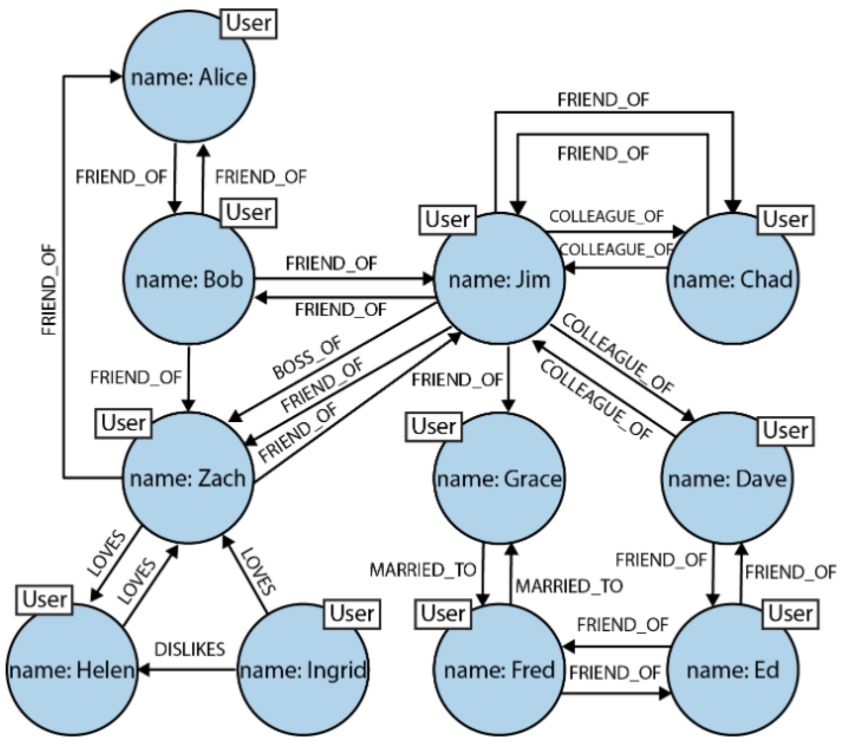


Figure 5. Easily modeling friends, colleagues, workers, and (unrequited) lovers in a graph

Simple network of friends has grown in size (there are now potential friends up to six degrees away) and expressive richness.

## Characteristics of Graph Databases

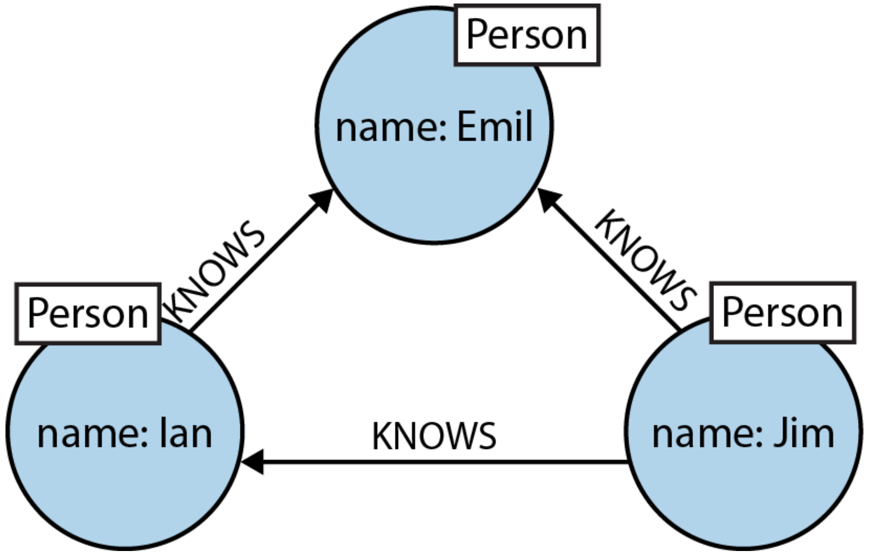
### Agility

- Our goal is to be able to modify our data model in conjunction with the rest of our application using technology that is compatible with the incremental and iterative software development techniques of today.
- Modern graph databases equip us to perform frictionless development and graceful systems maintenance.
- In particular, the schema-free nature of the graph data model, coupled with the testable nature of a graph database’s application programming interface (API) and query language, empower us to evolve an application in a controlled manner.
- Graph databases lack the kind of schema-oriented data governance mechanisms we are familiar with in the relational world.
- Governance is applied in a programmatic fashion, using tests to drive out the data model and queries, as well as assert the business rules that depend upon the graph.

# Data Modelling

## Cypher Philosophy

- Cypher is designed to be easily read and understood by developers, database professionals, and business stakeholders.
- Cypher enables a user to ask the database to find data that matches a specific pattern.
- We describe what “things like this” look like is to draw them, using ASCII art.
- This pattern describes a path that connects a node we call **Jim** to two nodes we call **Ian** and **Emil**, and which also connects the Ian node to the **Emil** node. **Ian**, **Jim**, and **Emil** are identifiers.
- Identifiers allow us to refer to the same node more than once when describing a pattern, a trick that helps us to get round the fact that a query language has only one dimension, whereas a graph diagram can be laid out in two dimensions.



A simple graph pattern, expressed using a diagram

- This pattern describes three mutual friends. The equivalent ASCII art representation in Cypher:

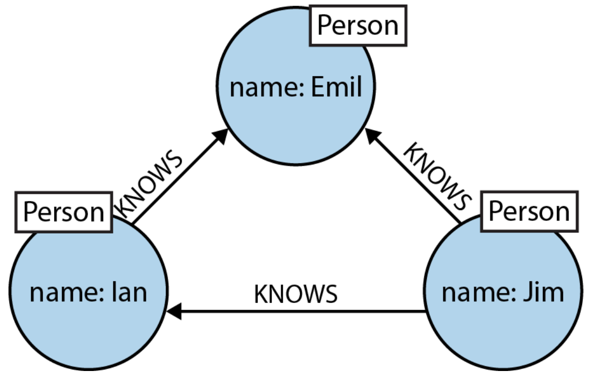
```
(emil)<-[:KNOWS]-(jim)-[:KNOWS]->(ian)-[:KNOWS]->(emil)
```

# Data Modelling

## Cypher Philosophy

- The previous Cypher pattern describes a simple graph structure, but it does not refer to any particular data in the database.
- To bind the **pattern** to **specific nodes** and **relationships** in an existing dataset, we must specify some **property values** and **node labels** that help to locate the relevant elements in the dataset. For example

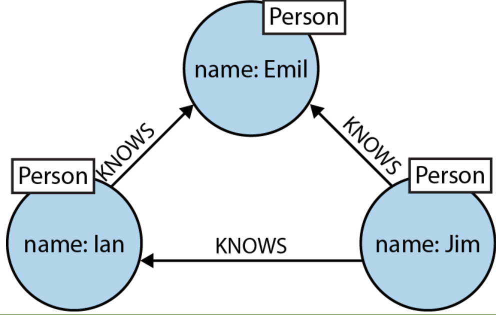
```
(emil:Person {name:'Emil'})
  <-[:KNOWS]-(jim:Person {name:'Jim'})
  -[:KNOWS]->(ian:Person {name:'Ian'})
  -[:KNOWS]->(emil)
```



- We have bound each node to its identifier using its **name property** and **Person label**.
- The **Emil** identifier, for example, is bound to a node in the dataset with a label Person and a name property whose value is **Emil**. Anchoring parts of the pattern to real data in this way is normal Cypher practice.

## Data Modelling

### MATCH



- The **MATCH** clause is at the centre of most Cypher queries. Using **ASCII** characters to represent nodes and relationships, we draw the data we are interested in.
- We draw nodes with parentheses, and relationships using pairs of dashes with ***greater-than*** or ***less-than signs (--> and <--)***.
- The **<** and **>** signs indicate relationship direction. Between the dashes, set off by square brackets and prefixed by a colon, we put the relationship name. Node labels are similarly prefixed by a colon. Node (and relationship) property key-value pairs are then specified within curly braces.

```
MATCH (a:Person)-[:KNOWS]->(b)-[:KNOWS]->(c), (a)-[:KNOWS]->(c)
WHERE a.name = 'Jim'
RETURN b, c
```

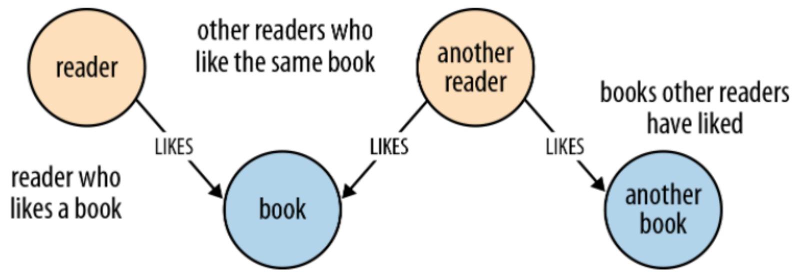
- In example query, we look for a node labeled **Person** with a name property whose value is **Jim**. The return value from this lookup is bound to the identifier **a**. This identifier allows us to refer to the node that represents **Jim** throughout the rest of the query.
- This start node is part of a simple pattern **(a)-[:KNOWS]->(b)-[:KNOWS]->(c), (a)-[:KNOWS]->(c)** that describes a path comprising three nodes, one of which we have bound to the identifier **a**, the others to **b** and **c**. These nodes are connected by way of several **KNOWS** relationships.

# Cypher Clauses

- Other clauses can be used in a Cypher query
  - **WHERE**
- It provides criteria for filtering pattern matching results.
  - **CREATE and CREATE UNIQUE**
- Create nodes and relationships.
  - **MERGE**
- Ensures that the supplied pattern exists in the graph, either by reusing existing nodes and relationships that match the supplied predicates, or by creating new nodes and relationships.
  - **DELETE**
- Removes nodes, relationships, and properties.
  - **SET**
- Sets property values.
  - **FOREACH**
- Performs an updating action for each element in a list.
  - **UNION**
- Merges results from two or more queries.
  - **WITH**
- Chains subsequent query parts and forwards results from one to the next. Similar to piping commands in Unix.
  - **START**
- Specifies one or more explicit starting points, nodes or relationships, in the graph.
- (**START** is deprecated in favor of specifying anchor points in a **MATC**H clause.)

# Building Graph Database Application

- We can identify entities and relationships in the data by asking the right questions of it.
- Agile user stories provide a concise means for expressing an outside-in, user-centered view of an application's needs, and the questions that arise in the course of satisfying this need.



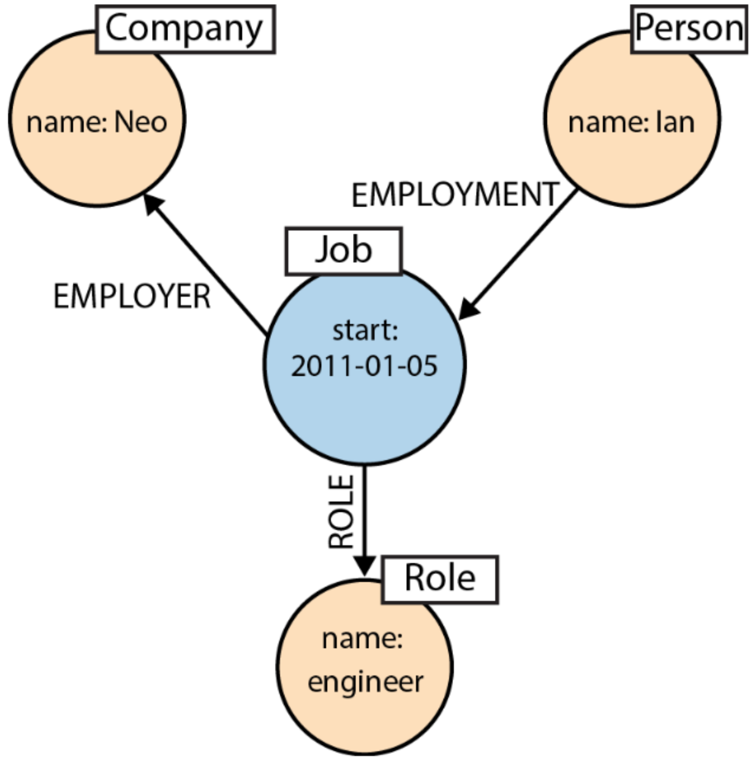
Data model for the book reviews user story

- **A user story for a book review web application:**
- As a reader who likes a book, I **WANT** to know which books other readers who like the same book have liked, SO THAT I can find other books to read.
- This story expresses a user need, which motivates the shape and content of our data model. From a data modeling point of view, the AS A clause establishes a context comprising two entities, a reader and a book plus the **LIKES** relationship that connects them.
- The I WANT clause then poses a question: which books have the readers who like the book I am currently reading also liked? This question exposes more **LIKES** relationships, and more entities: other readers and other books.

# Graph Database
## Employment

- Figure shows how the fact of **Ian** being employed by Neo Technology in the role of engineer can be represented in the graph.
- In Cypher, this can be expressed as:

-
  ```
  CREATE (:Person {name:'Ian'})-[:EMPLOYMENT]->(employment:Job {start_date:'2011-01-05'})-[:EMPLOYER]->(:Company {name:'Neo'}),(employment)-[:ROLE]->(:Role {name:'engineer'})
  ```



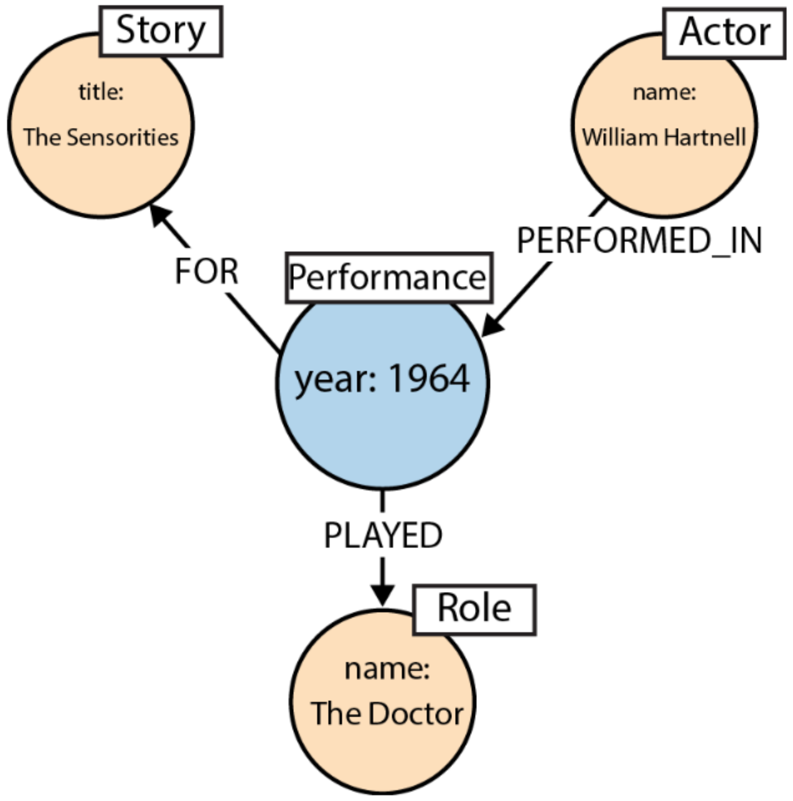
Ian began employment as an engineer at Neo Technology

# Graph Database

## Performance

- Figure shows how the fact that William Hartnell played The Doctor in the story The Sensorites can be represented in the graph.
- In Cypher, this can be expressed as

-
  ```
  CREATE (:Actor {name:'William
  Hartnell'})-[:PERFORMED_IN]->
  (performance:Performance {year:1964})-
  [:PLAYED]->(:Role {name:'The Doctor'})
  (performance)-[:FOR]->(:Story
  {title:'The Sensorites’})
  ```



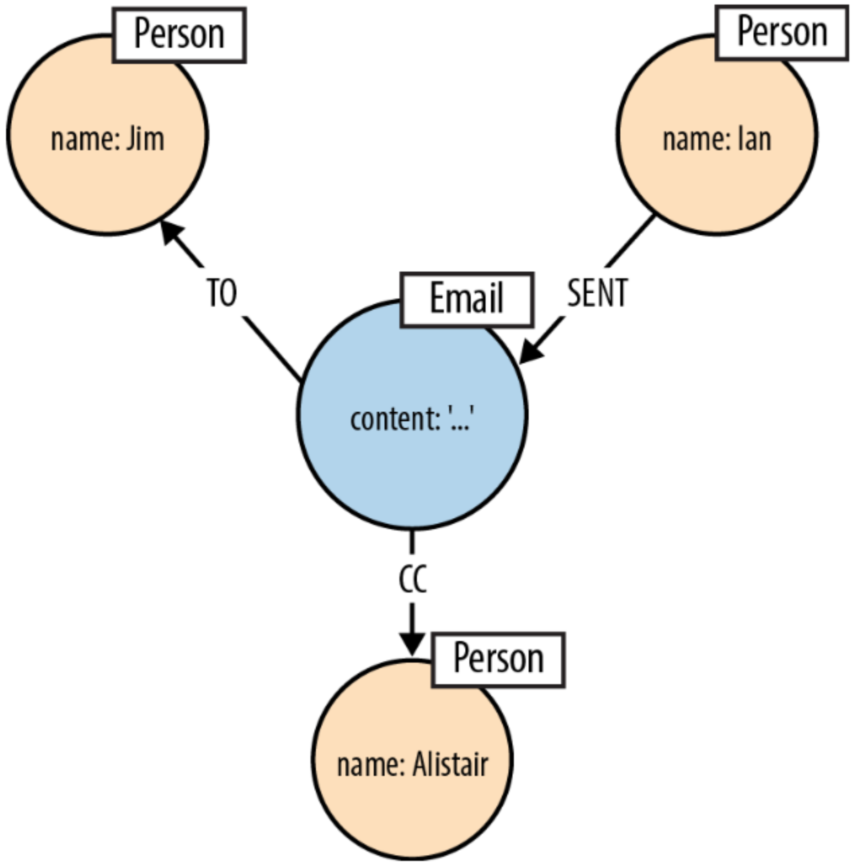
William Hartnell played The Doctor in the story *The Sensorites*

# Graph Database

## Emailing

- Figure shows the act of Ian emailing Jim and copying in Alistair.
- In Cypher, this can be expressed as:

-
  ```
  CREATE (:Person {name:'Ian'})-
  [:SENT]->(e:Email {content:'...'})-
  [:TO]->(:Person {name:'Jim’}),(e)-
  [:CC]->(:Person {name:'Alistair'})
  ```



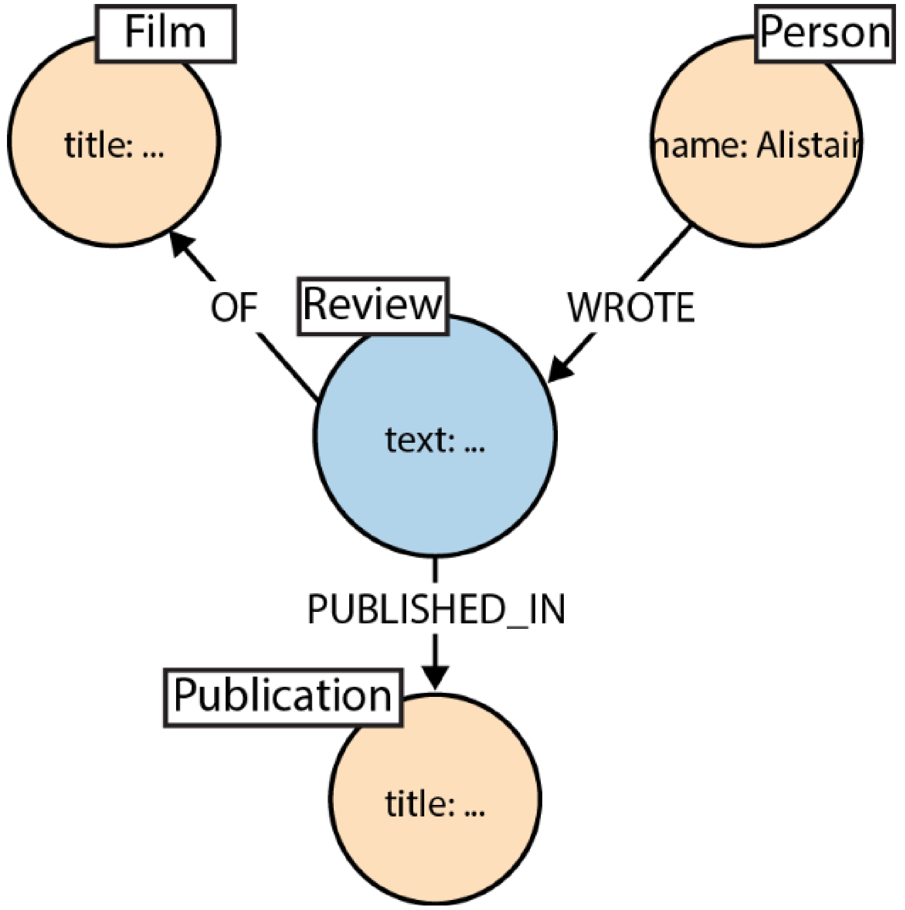
Ian emailed Jim, and copied in Alistair

# Graph Database

## Reviewing

- Figure shows how the act of Alistair reviewing a film can be represented in the graph.
- In Cypher, this can be expressed as:

-
  ```
  CREATE (:Person {name:'Alistair'})-
  [:WROTE]->(review:Review
  {text:'...'})-[:OF]->(:Film
  {title:'...'}),(review)-
  [:PUBLISHED_IN]->(:Publication
  {title:'...'})
  ```



Alistair wrote a review of a film, which was published in a magazine

# Complex Value Types as Nodes

- **Data structures and objects with numerous qualities or attributes might be considered complex value types.**

- **It is possible to represent and maintain intricate relationships between these value kinds by storing them as nodes in a graph database. *Value types are things that do not have an identity, and whose equivalence is based solely on their values.*** Examples include money, address, and SKU.

- Complex value types are value types with more than one field or property.

- **Address**, for example, is a complex value type. Such multiproperty value types may be usefully represented as separate nodes

- **`MATCH (:Order {orderid:13567})-[:DELIVERY_ADDRESS]->(address:Address) RETURN address.first_line, address.zipcode;`**

## Resources/ References

CCT College Dublin
Computing • IT • Business

- Graph Databases, 2nd Edition, Ian Robinson, Jim Webber, Emil Eifrem, O'Reilly Media, Inc., June 2015.
- https://neo4j.com/developer/cypher/
- https://neo4j.com/developer/get-started/
- Some images are used from Google search repository (https://www.google.ie/search) to enhance the level of learning.