Graph Data Model

Graph Theory and how it is applied
to graph databases for social graphing
and more

Introduction

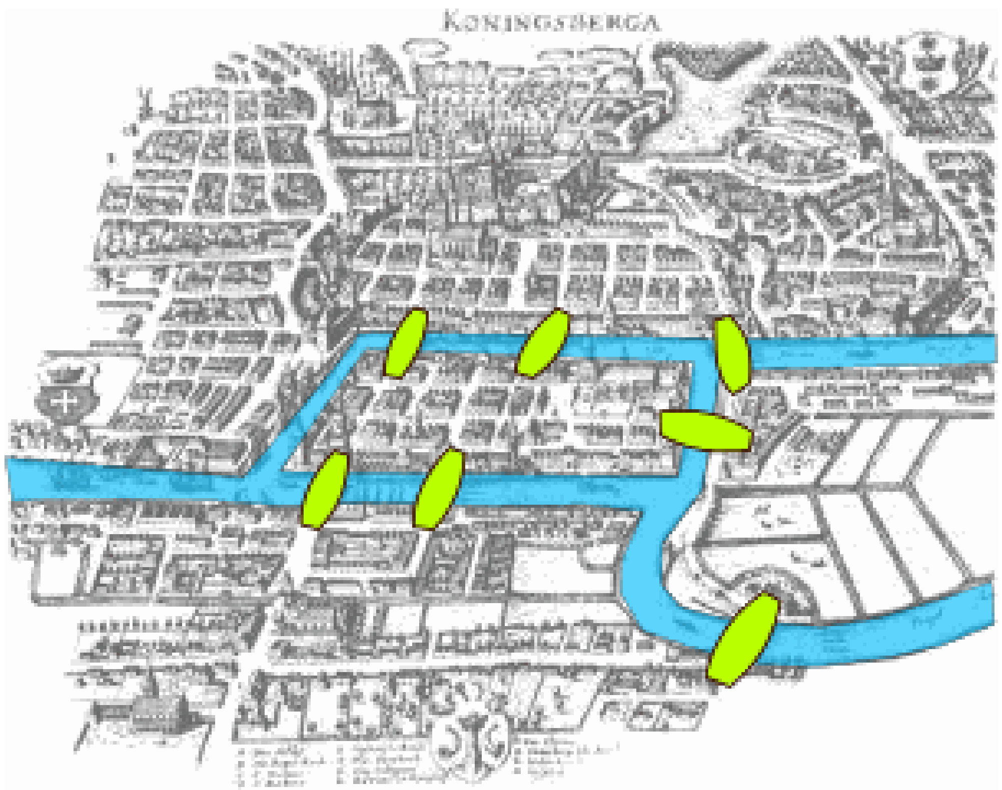
- Graph Theory is studied in Computer Science and Mathematics
- We will see how it is applied to databases
- We will look at its application to social graphs

Graph Theory

- Graph Theory is a subject studied in computer science and mathematics (not related to graphing of functions)
- Has its roots in 1735 with the "Seven Bridges of Königsberg" – now Kaliningrad in Russia.

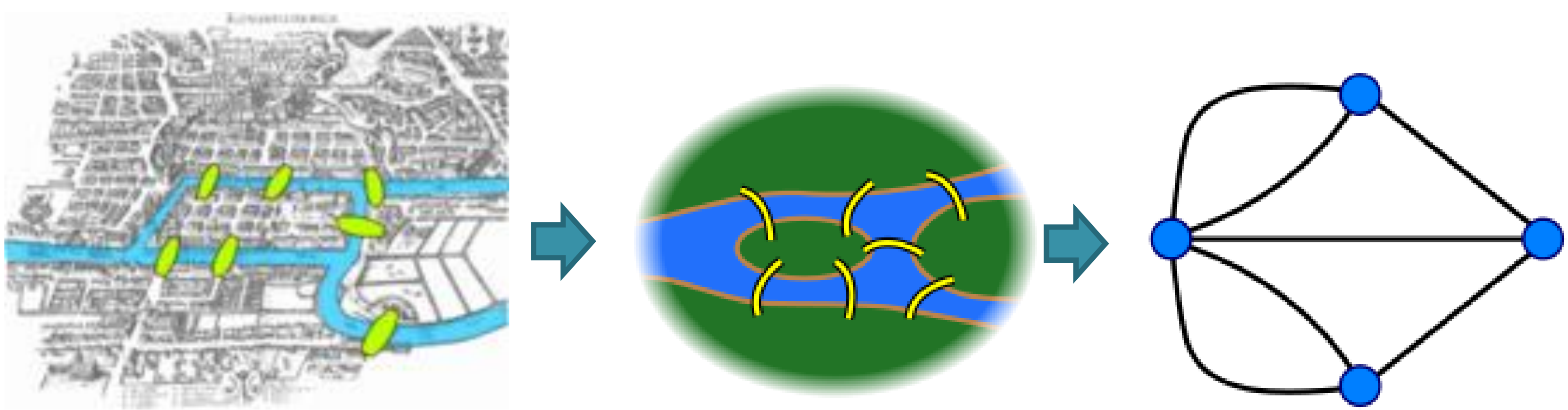
7 Bridges of Konigsberg

- *Find a walk through the city that would cross each bridge once and only once*



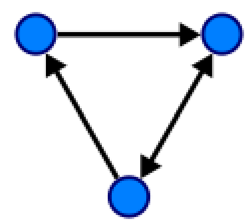
7 Bridges of Kingsberg

- Each land mass became a “vertex” or “node” (we will use the term node)
- Each bridge became an “edge”
- See more on Wikipedia



Directed and Undirected

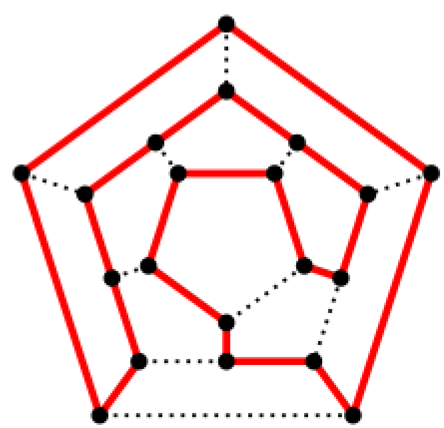
- A directed graph is one where the direction of ***traversing*** edges is set



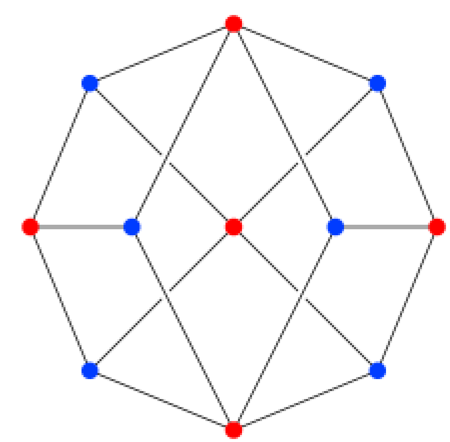
- In an undirected graph, no direction is set

Hamiltonian Path

- Hamiltonian path is a path where each node is visited exactly once
- A graph that contains a Hamiltonian path is called a **traceable graph**



Hamiltonian dodecahedron



Non-Hamiltonian "Herschel Graph"

Enough about the theory

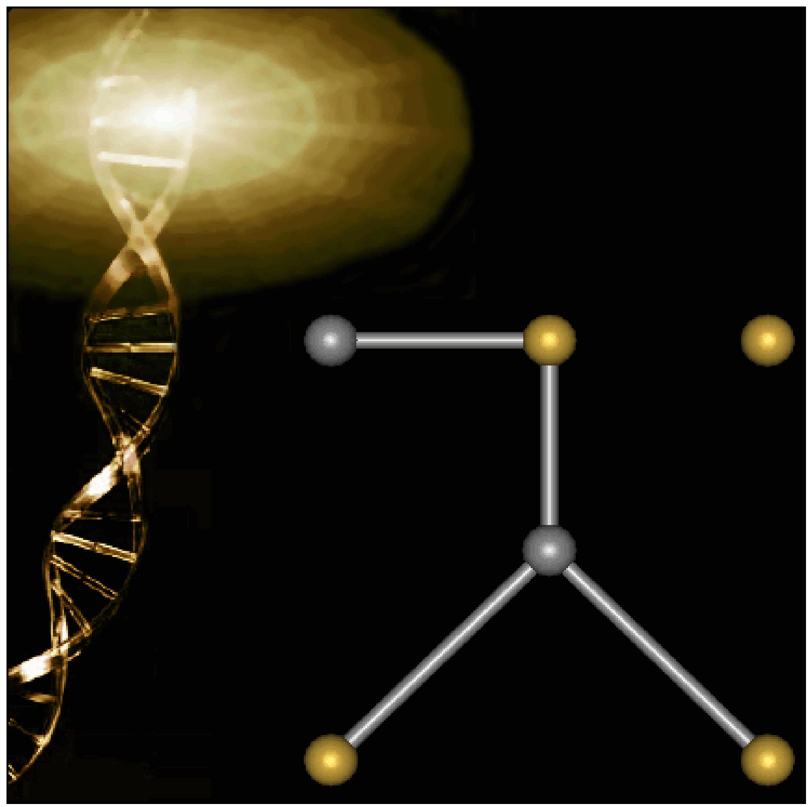
- Previous slides are just a small subset of graph theory
- It is a much wider subject that is way beyond the scope of this module
- There are many applications, some of which may have been staring you in the face

Applied Graph Theory

- GPS Navigation for shortest path home

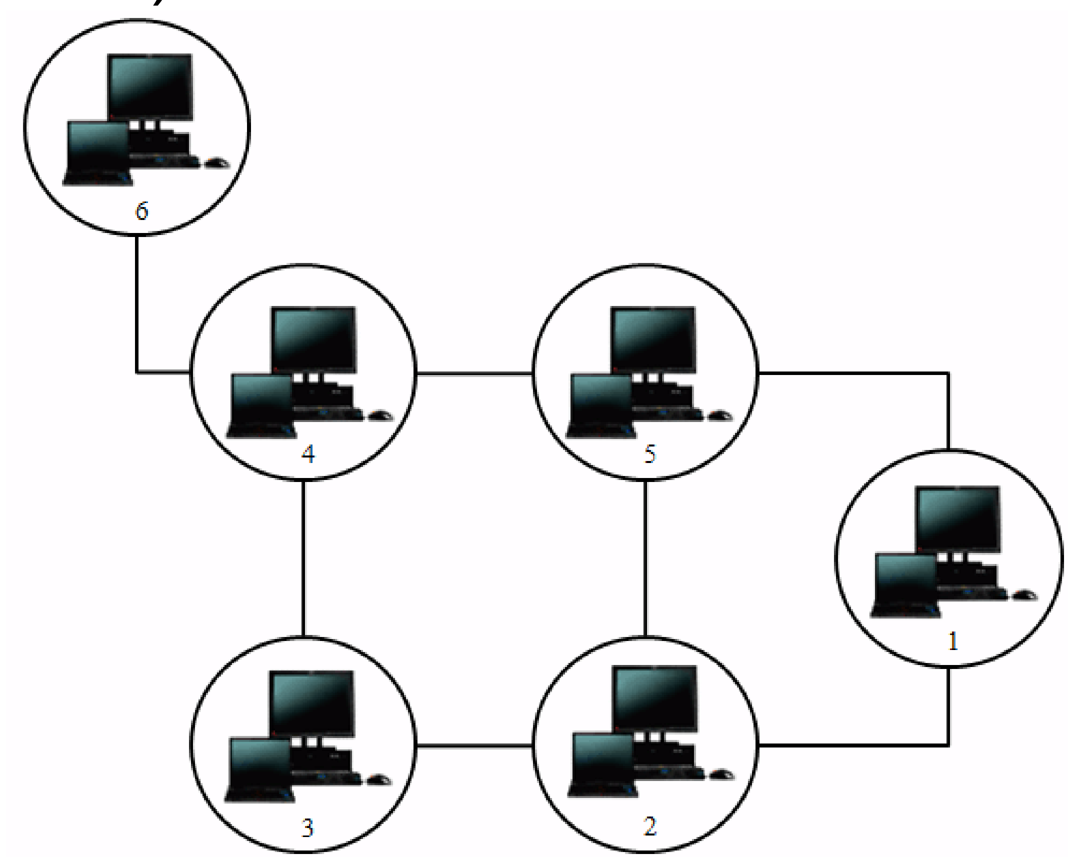
Applied Graph Theory

- Computational Biochemistry (looking for conflicts in DNA sequences)



Applied Graph Theory

- Network Security (e.g. propagation of worms)

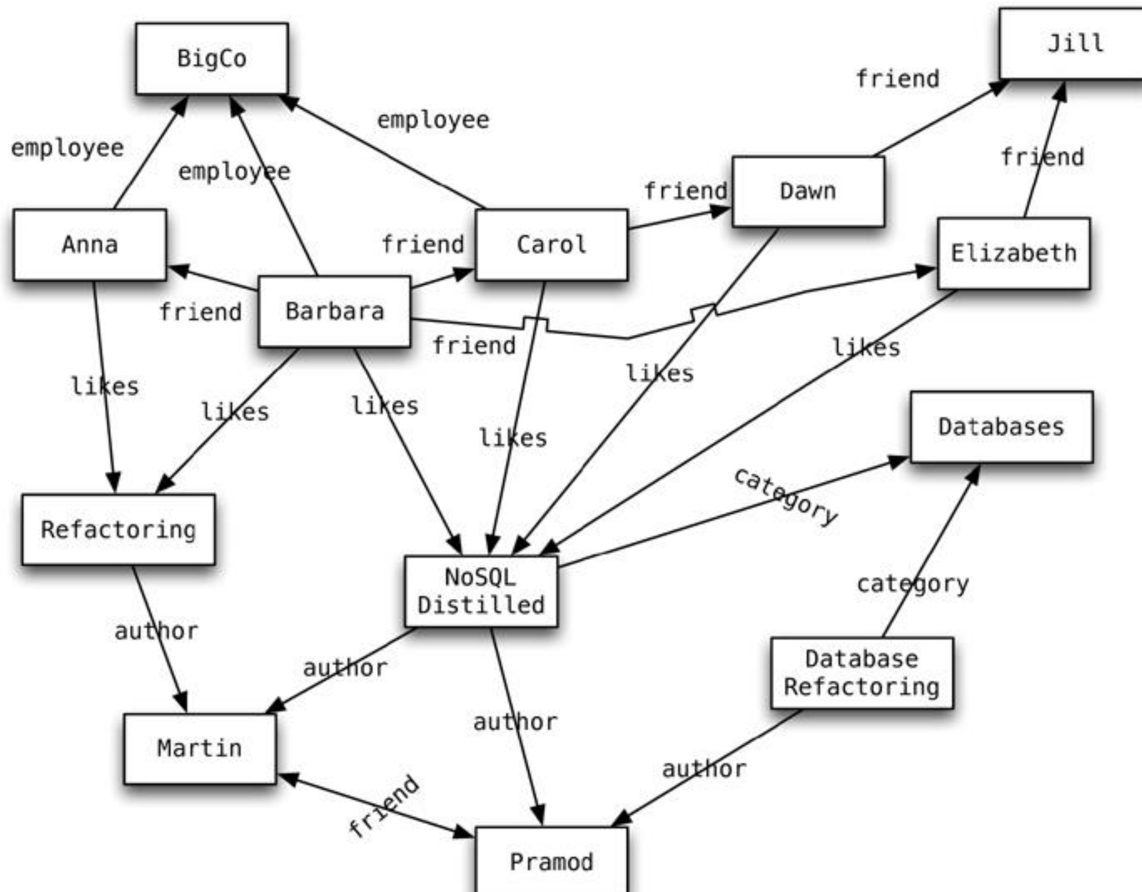


Applied Graph Theory

- Map region colouring (e.g. 4 colours, no regions of same colour touching)

NoSQL

- Example graph from "NoSQL Distilled" book



NoSQL Graph Model

- The model is very simple
- Nodes are connected by edges
- Database implementations will vary in how much further they take it
- FlockDB (used by Twitter) doesn't go much beyond that – no attributes are attached to nodes
- Neo4J allows attributes attached to nodes **and** edges – does this using Java objects in a schema-less way (i.e. each object can have different attributes)

NoSQL Graph Model

- Specific types of queries (relationship-focused) on these networks of nodes and edges can be very fast
- RDBMS will be expensive at read time when using foreign keys to join
- Highly connected data in RDBMS can be expensive to query
- Graph shifts the burden to insert time to make query time fast

NoSQL Graph Model

- There usually needs to be a starting point
- Can index nodes
- Therefore, can ask questions like:
  - Which of my friends like movies rated 4 stars or better in my favourite category
  - Which of my friends' friends liked Lady Gaga's latest song

NoSQL Graph Model

- Whereas with RDBMS, joins are computed at query time…
- Graph databases write those joins as edges at insert time, so query time does not have the join overhead
- Graph databases also make it inexpensive to have multiple relationships between nodes (can be disruptive with RDBMS)

NoSQL Graph Model

- Relationships can be real world style (likes, works at, reviewed, visited, etc) or can be there for performance reasons (spatial indexing or sorted linked lists, for example)

Available Graph Databases

- There are several graph databases, including:
  - Neo4J
  - FlockDB
  - OrientDB
  - AllegroGraph

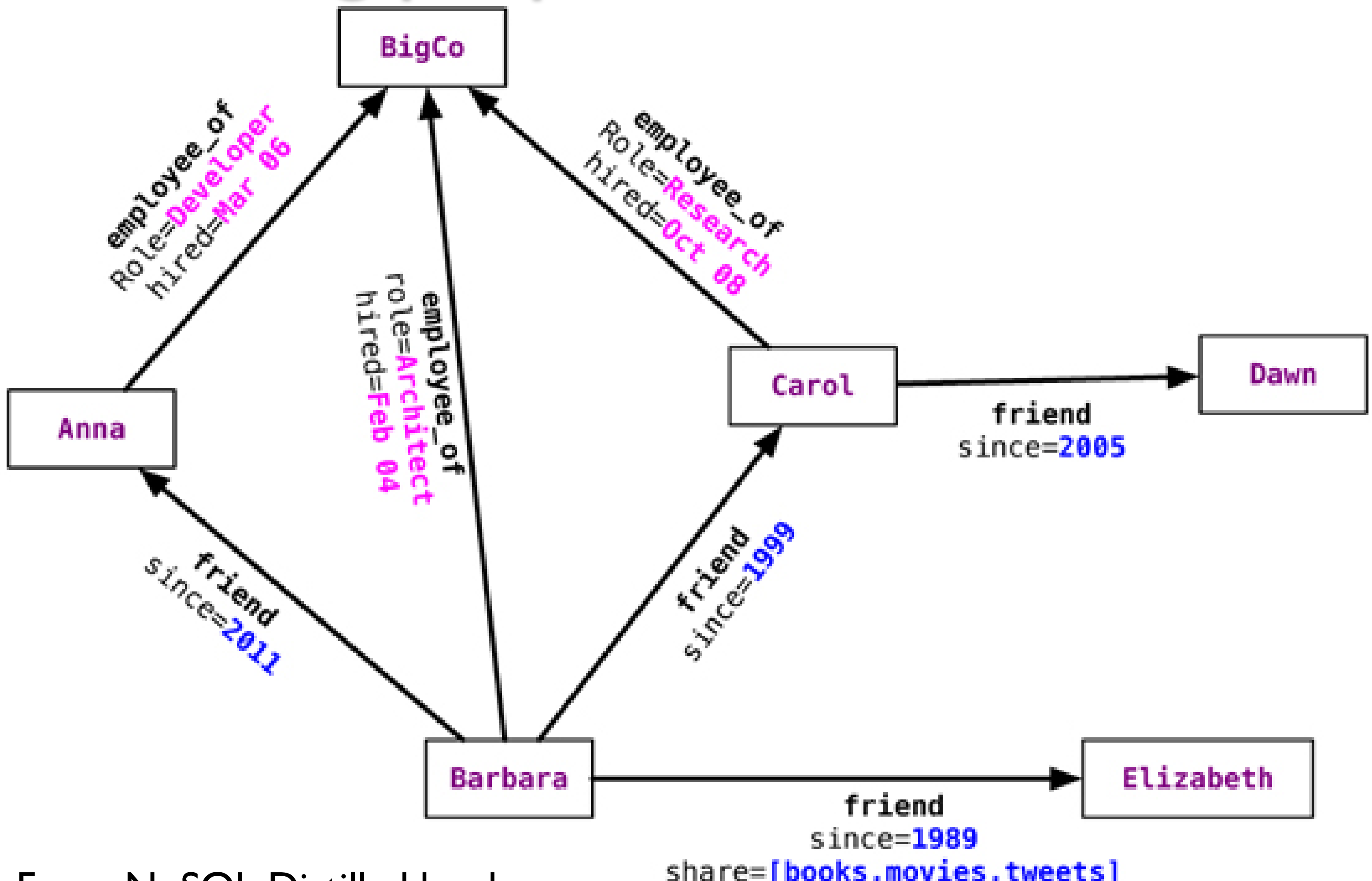
Neo4J

- Our focus will be on the popular Neo4J database (http://www.neo4j.org/)
- This is well supported in Java…

```
Node larkin = graphDb.createNode();
larkin.setProperty("name", "Larkin Cunningham");
Node robocop = graphDb.createNode();
robocop.setProperty("title", "Robocop");
larkin.createRelationshipTo(robocop, WATCHED);
Robocop.createRelationshipTo(larkin, WATCHED_BY);
```

Note the bi-directionality of the WATCHED-WATCHED_BY relationship

Graph db with nodes and edges containing properties



From: NoSQL Distilled book

Relationships are key

- The whole point of graph databases is in modelling the relationships between entitites - care needs to be taken to get the edges right
- ER Modelling not really suitable
- Previous diagrams show that graph modelling tends to be “whiteboard-friendly” and are easily understood by all stakeholders – model entity instances rather than entity types

Graph DB Consistency

- Distribution usually not well supported
- Neo4J is fully ACID-compliant
- Can run Neo4J in a cluster, but other solutions might be better suited to distribution, such as Infinite Graph
- Graph databases are consistent because you cannot have a dangling edge – edges must always be between nodes

Graph DB

- Transactions
  - Depends on the database
  - Neo4J supports transations – can begin, commit and rollback
- Availability
  - Some support master-slave replication (e.g. Neo4J can have slaves auto-elect a new master if the master goes down)
  - Others support distributed nodes

Graph Querying

- Some support Gremlin if they implement a certain type of property graph
- Neo4J has its own Cypher query language (a.k.a. CypherQL)
- Indexes are supported on properties of nodes and edges to allow you search for a start node
- We will examine querying of Neo4J in this next week's lab document

Graph DB Scaling

- Depends on the database
- Cannot easily shard when the db is relationship rather than aggregate-oriented
- Traversing a graph across db nodes is an expensive operation
- Vertical is usually the best way to scale
- Can also have many read-only slaves to improve read performance
- Might by able to shard by having sub-graphs – e.g. European graph and American graph – but not easy to do

Graph DB Use Cases

- We saw some applications of graph theory earlier
- When we need connected data, graph databases are very appropriate, e.g. social networks, genealogy, bioinformatics, recommendations, location-based services