Graph Databases

TEAM 4

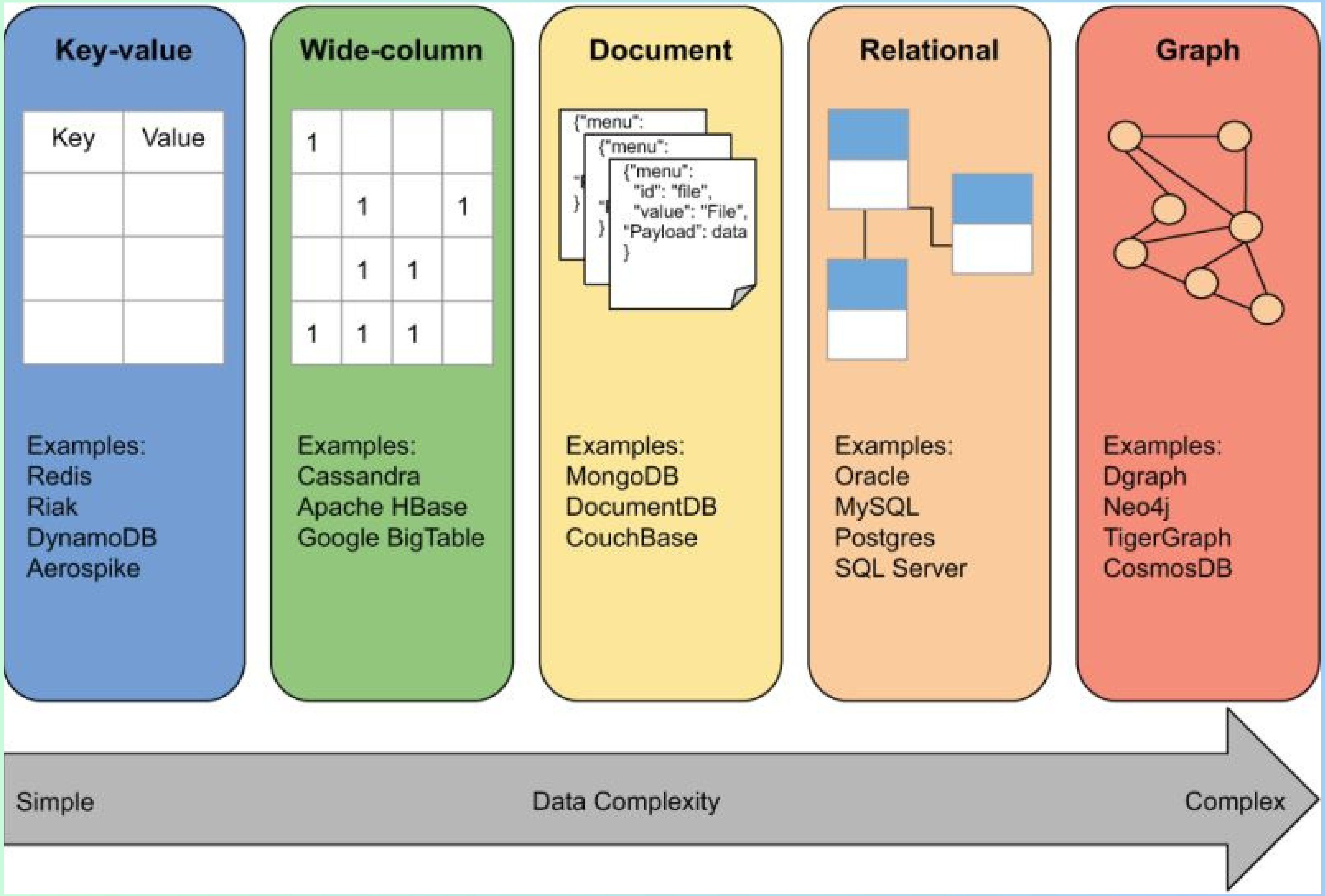
Rajeevan M

Sravya L

Ushasri B

Graph Databases

Data with varying Complexity: Suitable DB Designs



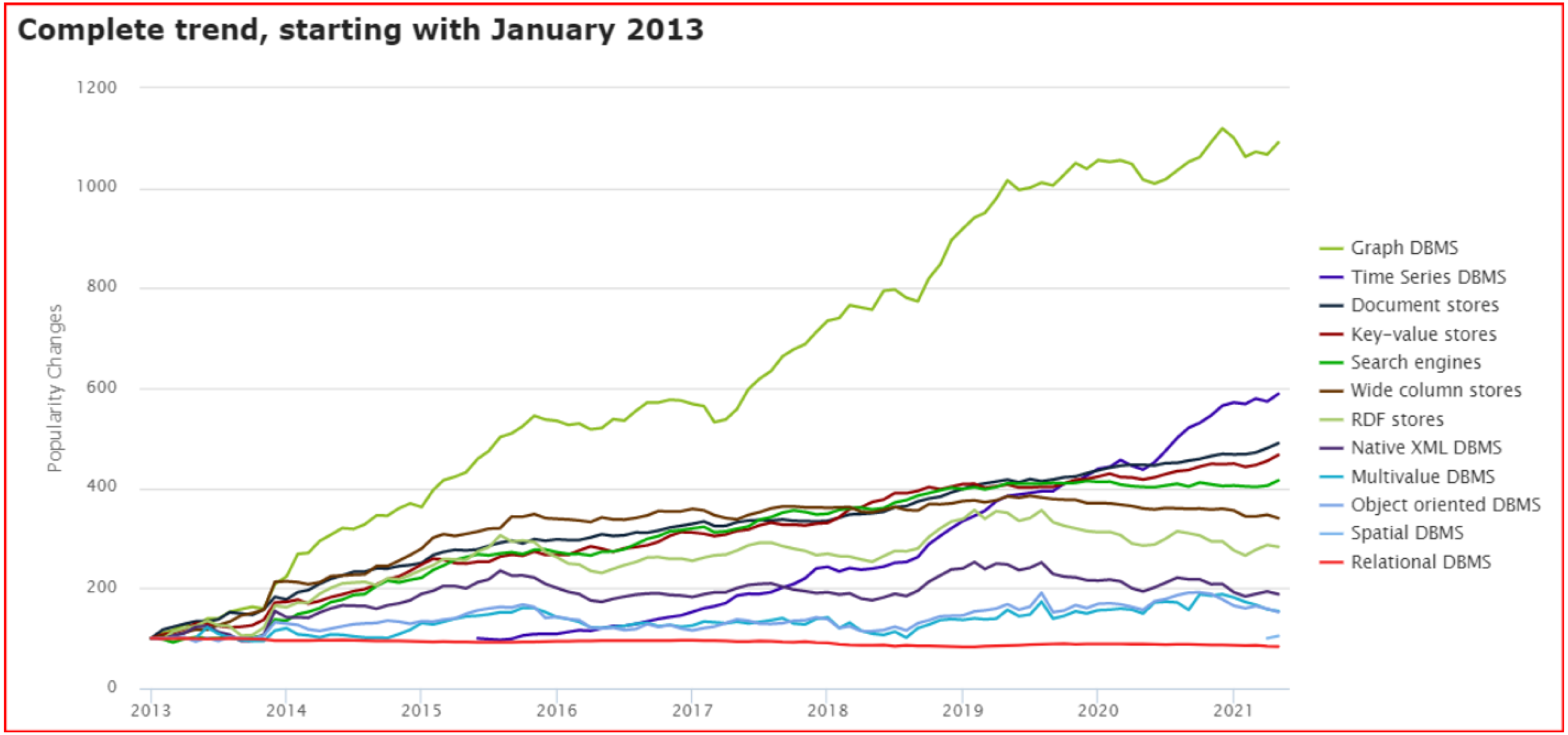
This illustration shows 5 significantly different types of databases that have been designed to handle data with increasing complexity.

Database designs 1,2,3 and 5 fall under the 'NoSQL' hashtag.

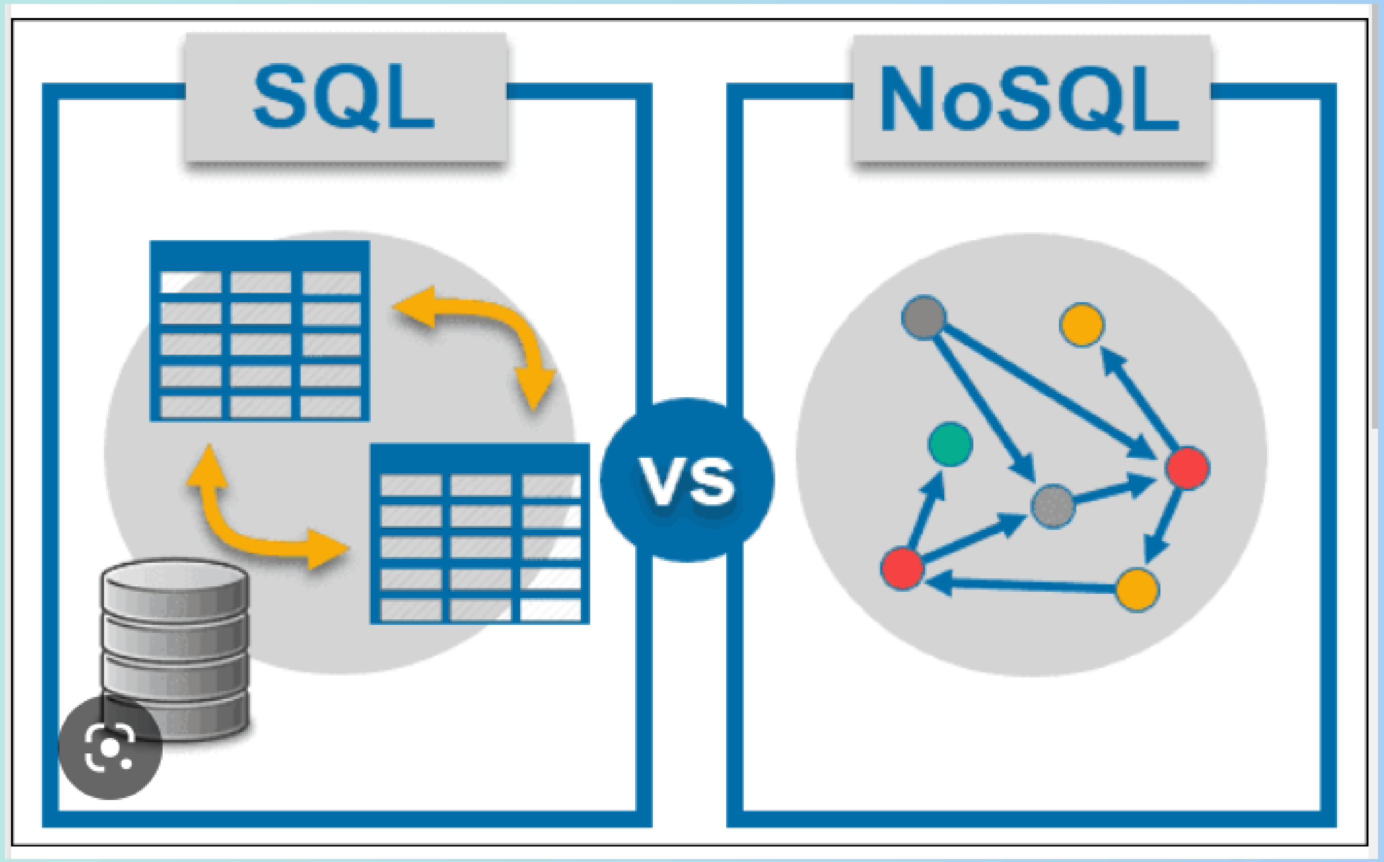
They are faster and easily scaleable compared to the Relational design.

Graph DB Trends from2013 to 2021

Compared to other DBs, Graph DBs are growing at an impressive pace. The following is an image made by DB engine, a popular Database Ranking site. We can say that Relational DBs have attained maturity while Graph DBs are still in the Growth phase.



The SQL and NoSQL Distinction



RELATIONAL DBs

Data Storage as Tables with **Fixed Schema**

Relationship between tables thru Primary and Foreign keys

To access multiple tables in distributed environments, expensive joins are required

Use SQL for CRUD and querying

Uphold ACID Properties, Robust

Suitable for a variety of uses

Impedence mismatch

NoSQL DBs

Data storage as Key-value pairs, column or Document formats with **Flexible Schema**

Documents map to objects in the most popular programming languages thereby supporting the rapid development of applications

Use XML, JSON, or some other API or query language for CRUD operations

Fast, Easily Scalable

Suitable for a variety of uses and distributed environments

Graph Databases

Evolution:

First, as an internal tool at Neo, to make querying existing Databases faster using Graph theory principles.

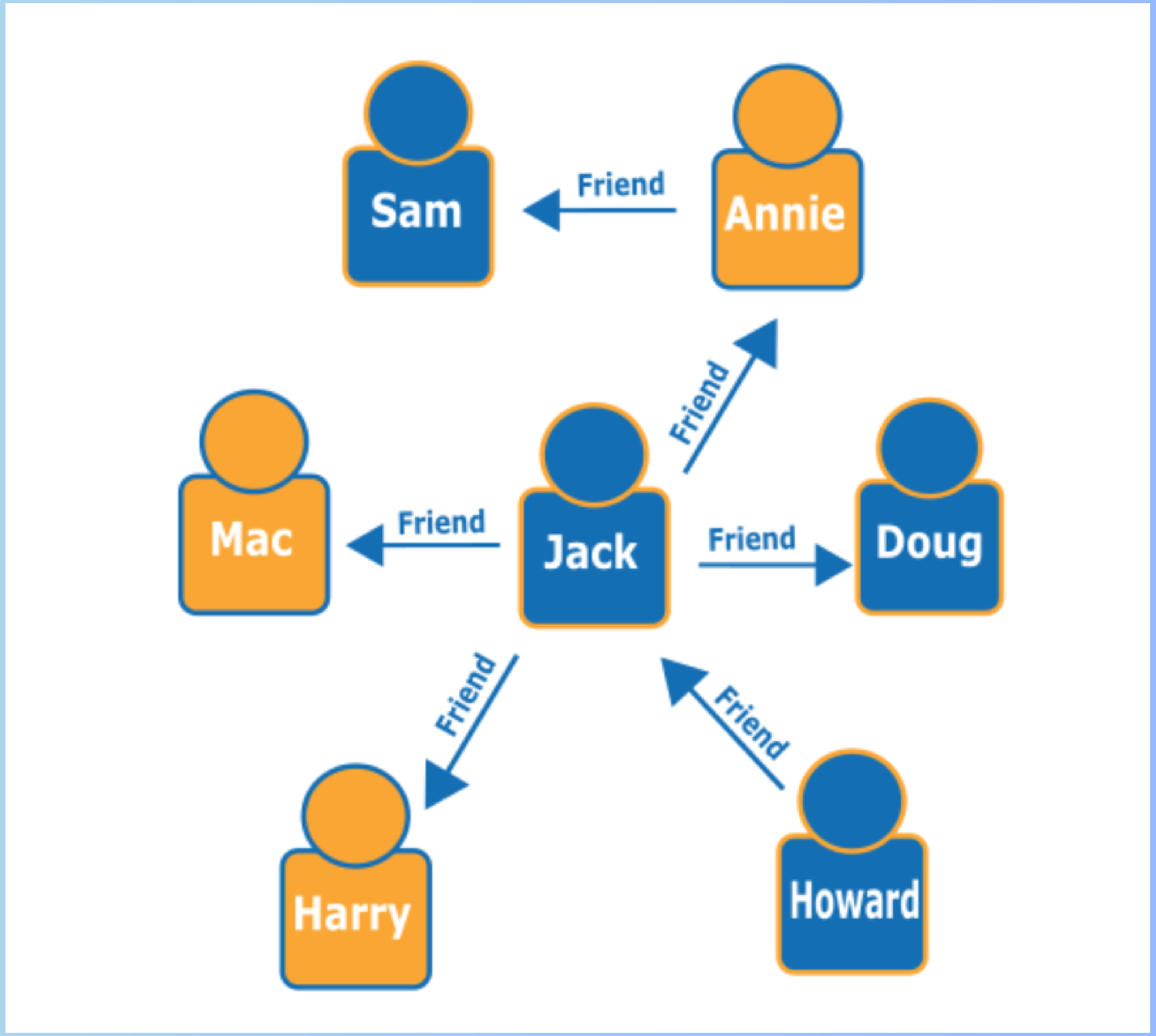
Neo team understood the limitations of SQL and the Relational design when it comes to extracting insights from relationships.

They wanted to make relationships between data first-class citizens in their design.

They called it Graph DB inspired by what Facebook said about itself, ‘A utility for the social graph’ because, GraphDBs were particularly good at solving these kinds of 'Social Graph 'problems.

.

'Friends of friends'



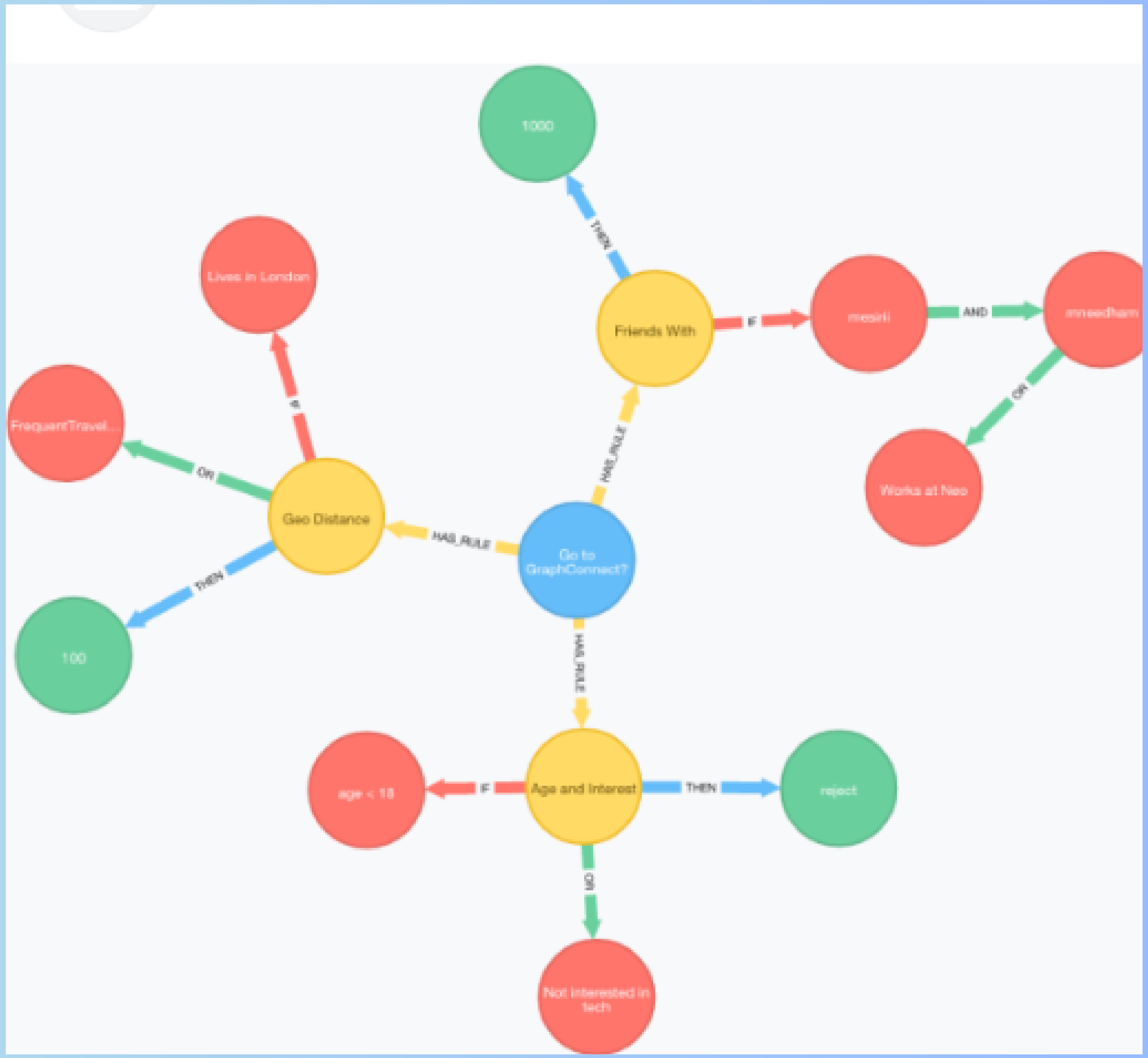
Graph DB for Connected Data

What is a Graph DB?

Graph Databases use Graph theory principles, the mathematical model of Vertices, Edges and relationships to both to store data and query relationships through Graph Traversal.

Data in a Graph DB is always 'connected' and not isolated. Relationships between data objects are established and linked data is stored together.

Graph Traversal involves following established paths, is therefore faster, resulting in a highly available DB suitable for distributed environments.



Graph Traversal

Using SQL

Graph Traversal of a relational model is possible using SQL.
The following is a snippet of SQL needed to find all actors that have worked with Keanu Reeves.
This 5-way join is manageable, but as we traverse more we have to add another 3-joins each time we increase the search depth. Also there is no syntax that allows us to explore an arbitrary depth

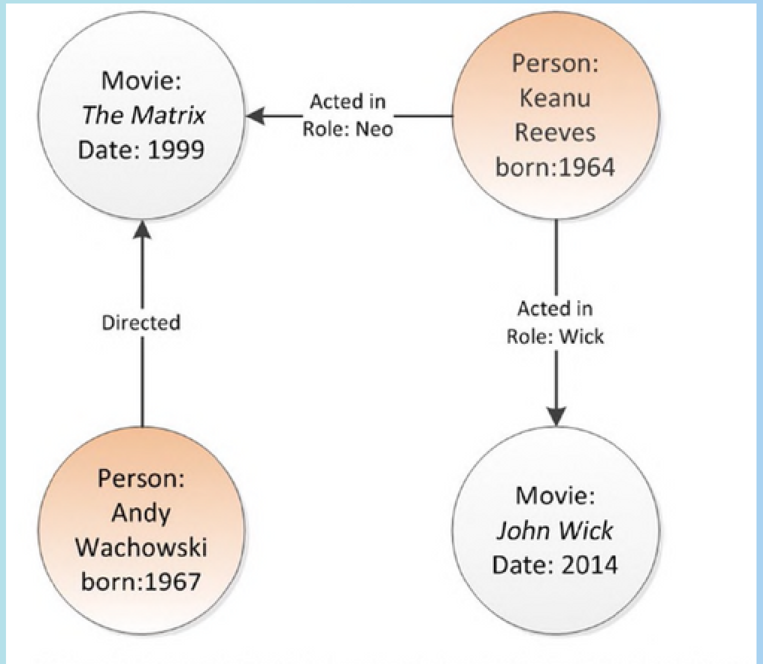
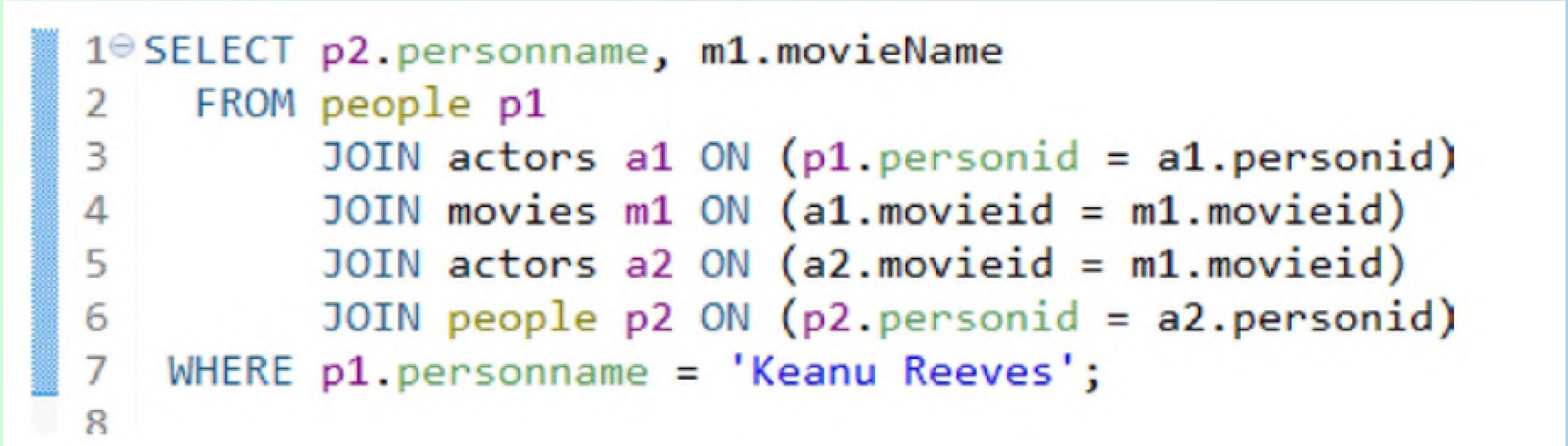

Movie: The Matrix Date: 1999
Acted in Role: Neo
Person: Keanu Reeves born:1964
Directed
Acted in Role: Wick
Person: Andy Wachowski born:1967
Movie: John Wick Date: 2014

Figure 5-1. *Simple graph with four vertices (nodes) and three edges (relationships)*



```
SELECT p2.personname, m1.movieName
  FROM people p1
       JOIN actors a1 ON (p1.personid = a1.personid)
       JOIN movies m1 ON (a1.movieid = m1.movieid)
       JOIN actors a2 ON (a2.movieid = m1.movieid)
       JOIN people p2 ON (p2.personid = a2.personid)
 WHERE p1.personname = 'Keanu Reeves';
```

Figure 5-3. *SQL to perform first-level graph traversal*

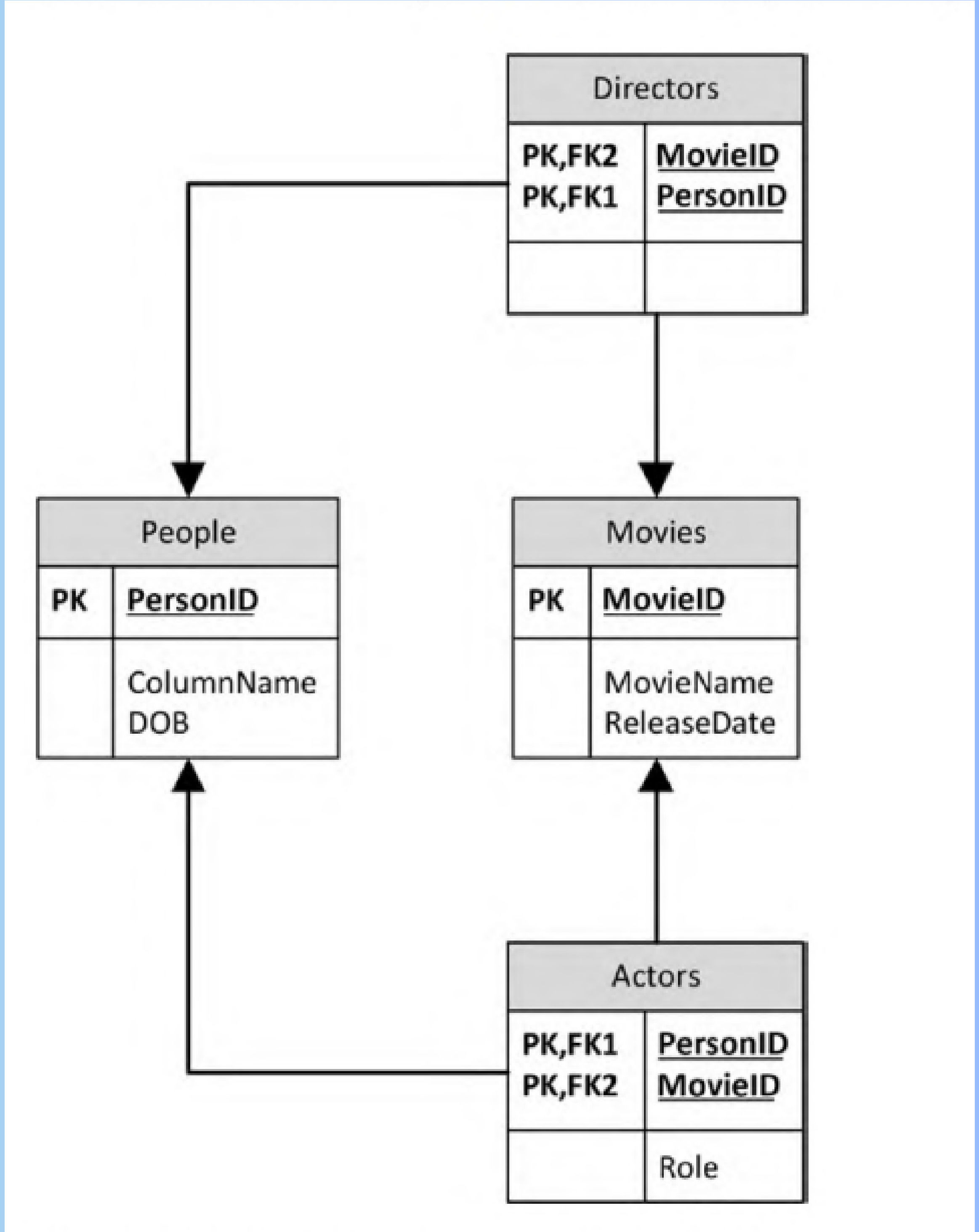


Figure 5-2. *Relational schema to represent our sample graph data*

Why Graph DB?

Because Real-world data is rich and interrelated, it resists being captured by a one-size fits all schema or conveniently split, disconnected aggregates.

Graph DBs are efficient at visual representation as well as querying connected data.

The relational model is not efficient in dealing with complex relationships and neither is the Document model, as it does not support joins.

Graph DBs are mainly used with OLTP systems and hence optimized for preserving transactional integrity and operational availability.

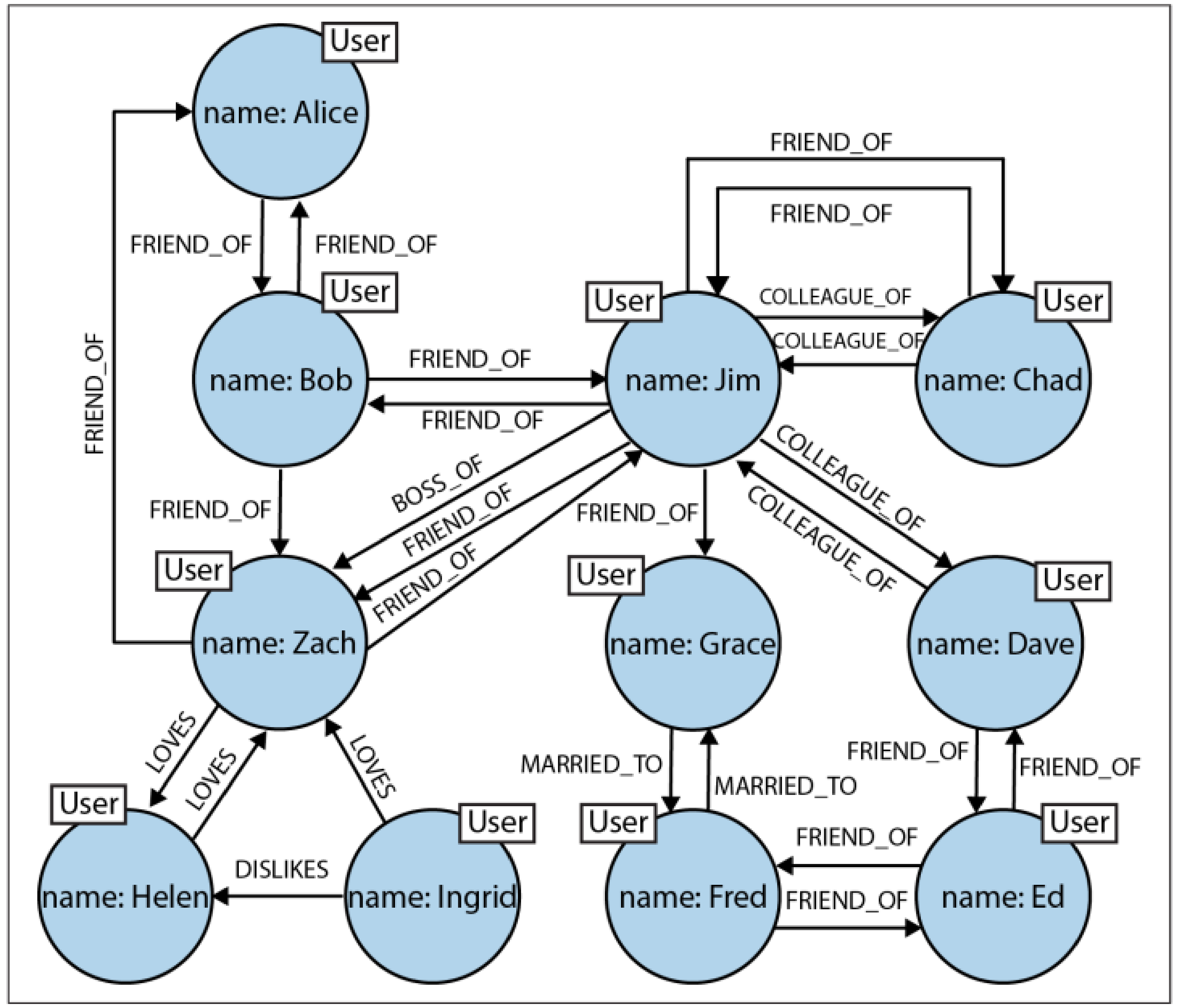


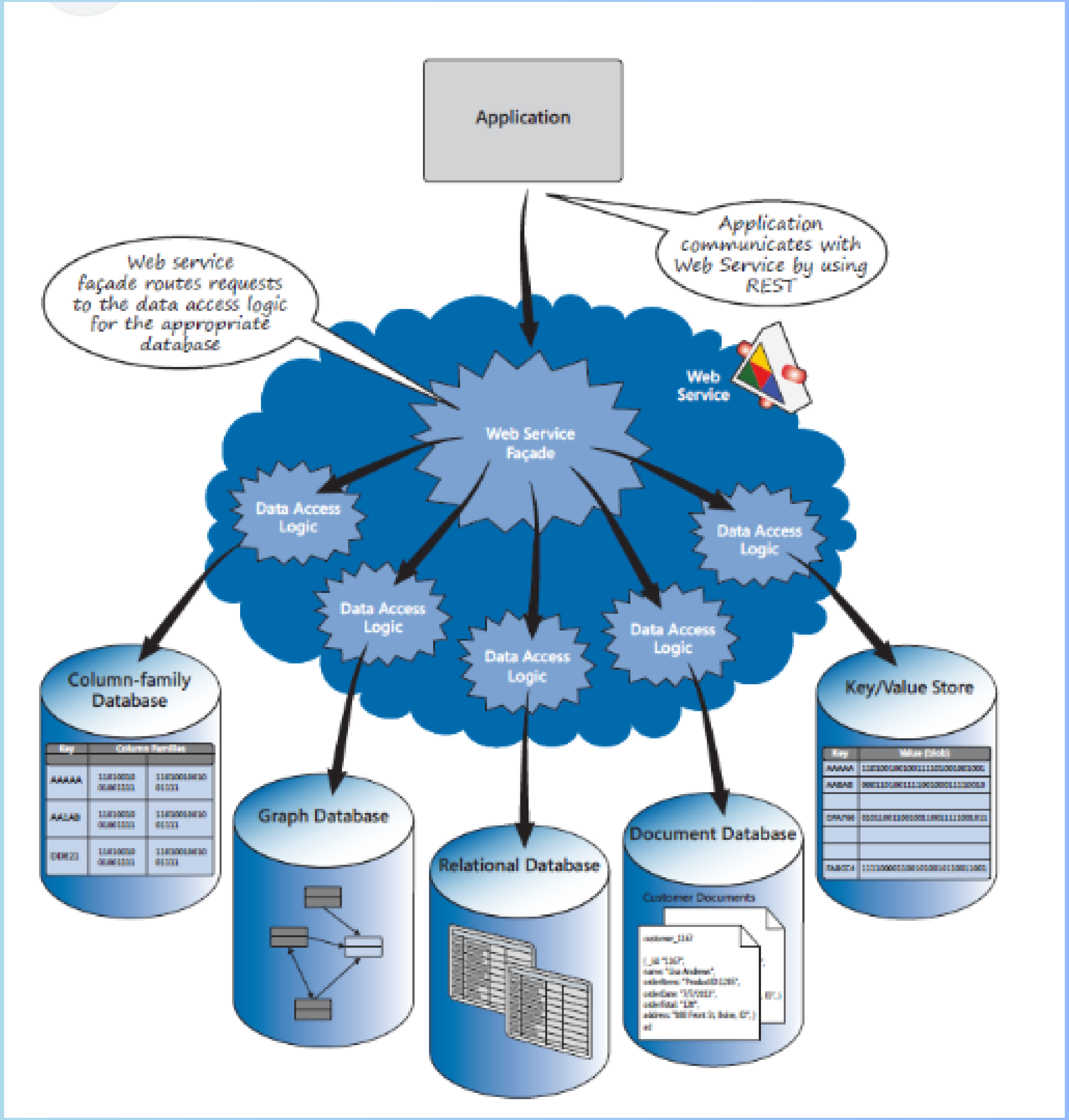
Figure 2-5. Easily modeling friends, colleagues, workers, and (unrequited) lovers in a graph

Polyglot Persistence

Different applications have different requirements and the best choice of technology varies between applications.

Though rigid, Relational DBs are robust and are able to generalize well, because of which they are used in a variety of applications.

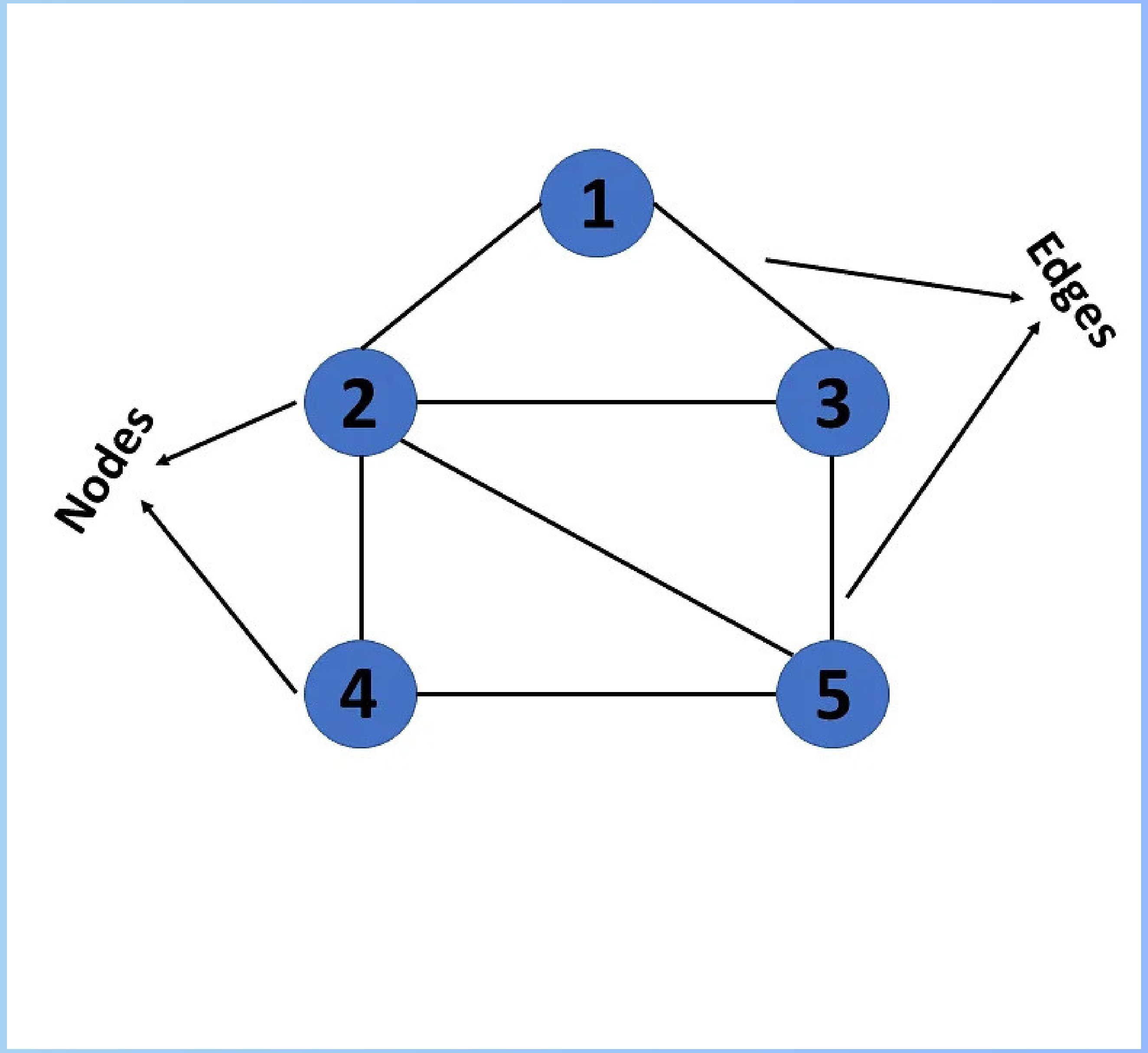
Therefore, in the foreseeable future, Relational Databases may be used alongside a broad variety of non-relational data stores. This idea is called 'Polyglot Persistence'



Graph Database Fundamentals

Graph Theory Basics:

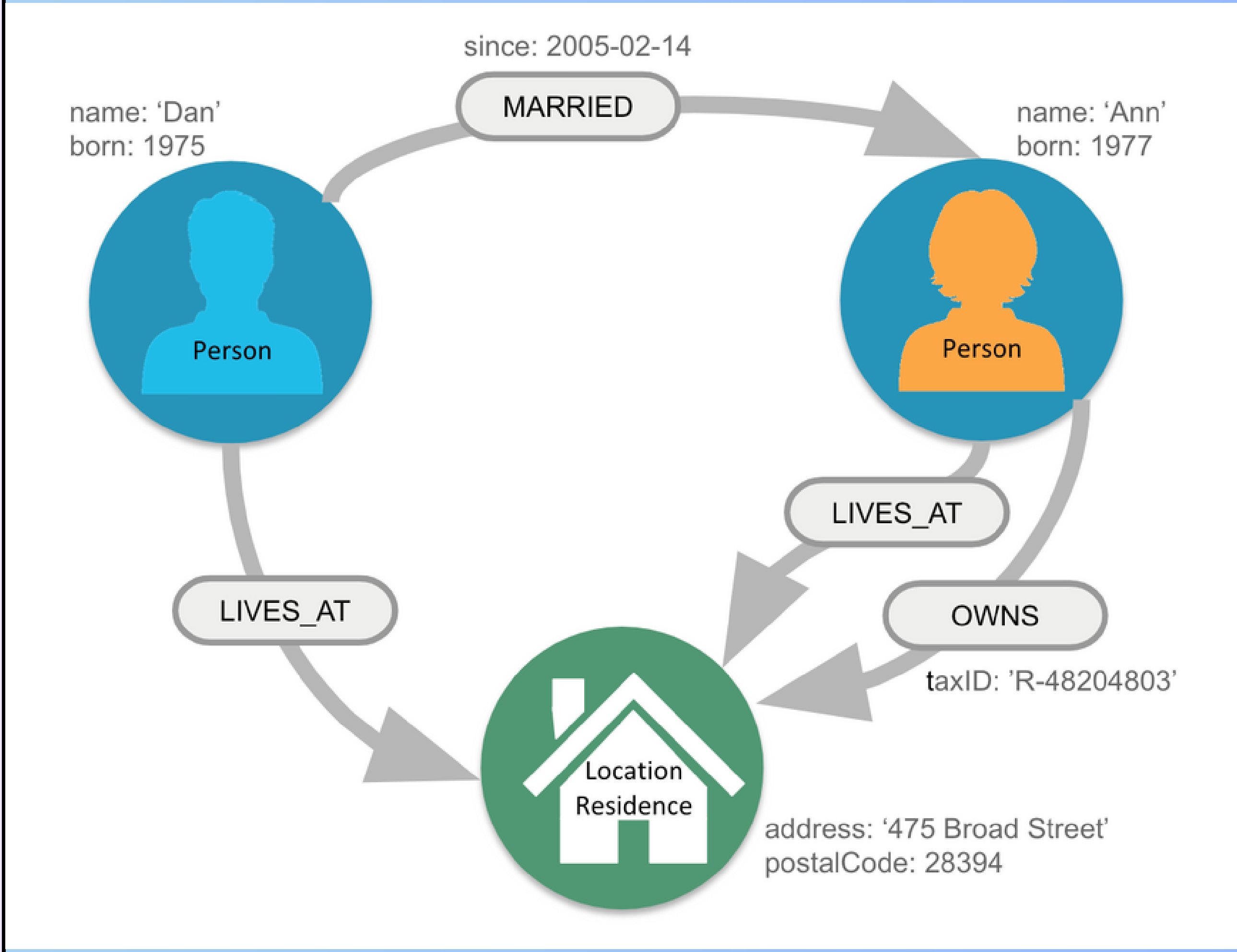
- A graph is a data structure.
- It represents:
  - A set of objects (nodes/vertices)
  - Their relationships (edges).
- Graphs can represent various types of data and are useful for visualizing and analyzing complex systems.
- Graph databases are a type of database that use graph theory principles to store, manage, and query data.
- Nodes represent entities or objects and edges to represent relationships or connections between them.
- Graph, Vertex, Edge, Path , Loop , Isomorphism, Order, Size, Degree, Closeness, Betweenness, Traversal.



Graph Database Fundamentals

Nodes, Relationships, and Properties:

- Nodes represent entities such as people, places, or things.
- Relationships represent the connections between them.
- Nodes can have properties and key-value pairs that provide additional information about the entity.
- Relationships can also have properties, which describe the characteristics of the connection.
- The combination of nodes, relationships, and properties creates a rich, connected data model that is well-suited for complex data scenarios.



Graph Database Fundamentals

Labeling and Indexing:

- Labels are used to group nodes together based on their type.
- For example, you might have labels for "Person," "Organization," or "Location."
- This makes it easy to query for all nodes of a certain type. Indexing is used to optimize queries by creating an index on one or more properties.
- This allows for fast lookups of nodes based on their properties, rather than having to traverse the entire graph.
- By using labeling and indexing, graph databases can efficiently handle large and complex data sets.

Graph Database Implementation

- Data Modeling
- Querying Languages
- Graph Database Implementations
- Best Practices

Data Modeling

- In data modeling for graph databases, it is important to think about the relationships between entities and how they connect to each other.
- It is also important to consider the different types of queries that will be run on the data and how the data model can be optimized for those queries.

Graph Database Implementation

Querying Languages:

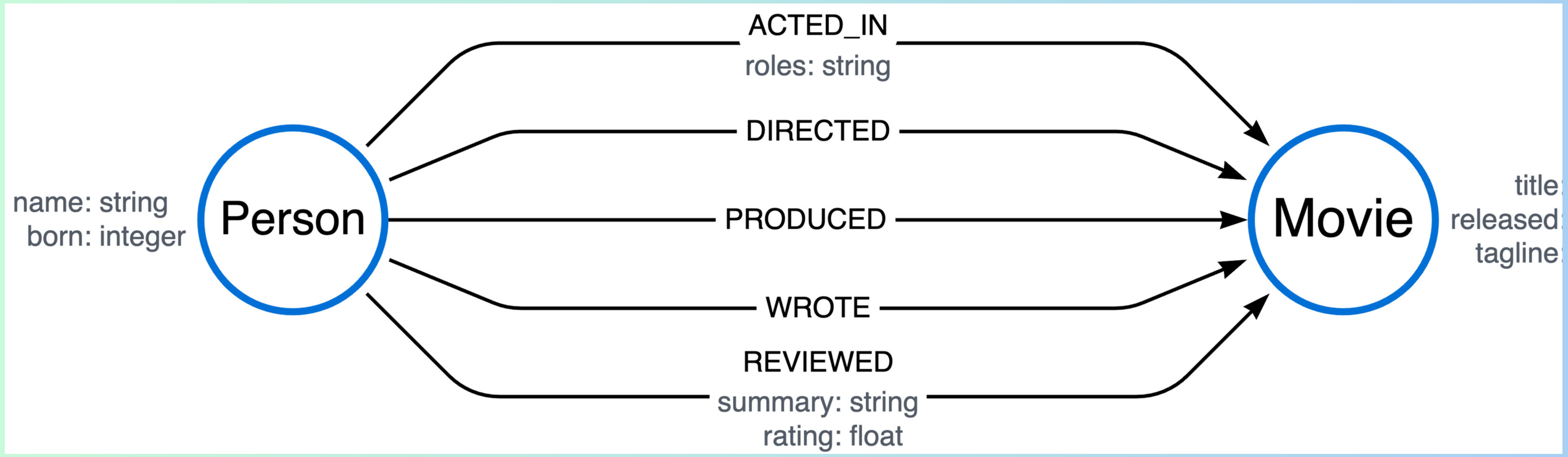
- Graph databases use their own querying languages to retrieve and manipulate data.
  - Cypher: Cypher is a declarative querying language used by Neo4j, the most popular graph database. It is a highly expressive and easy-to-use language that allows users to describe their queries in a natural language-like syntax.
  - Gremlin: Gremlin is a graph traversal language used by Apache TinkerPop, a popular graph computing framework. It is a functional language that allows users to write complex graph traversals using a combination of pipes and steps.
  - SPARQL: SPARQL is a querying language used by RDF databases, which store data in a graph-like format. It is a powerful language that allows users to query and manipulate data stored in RDF graphs.

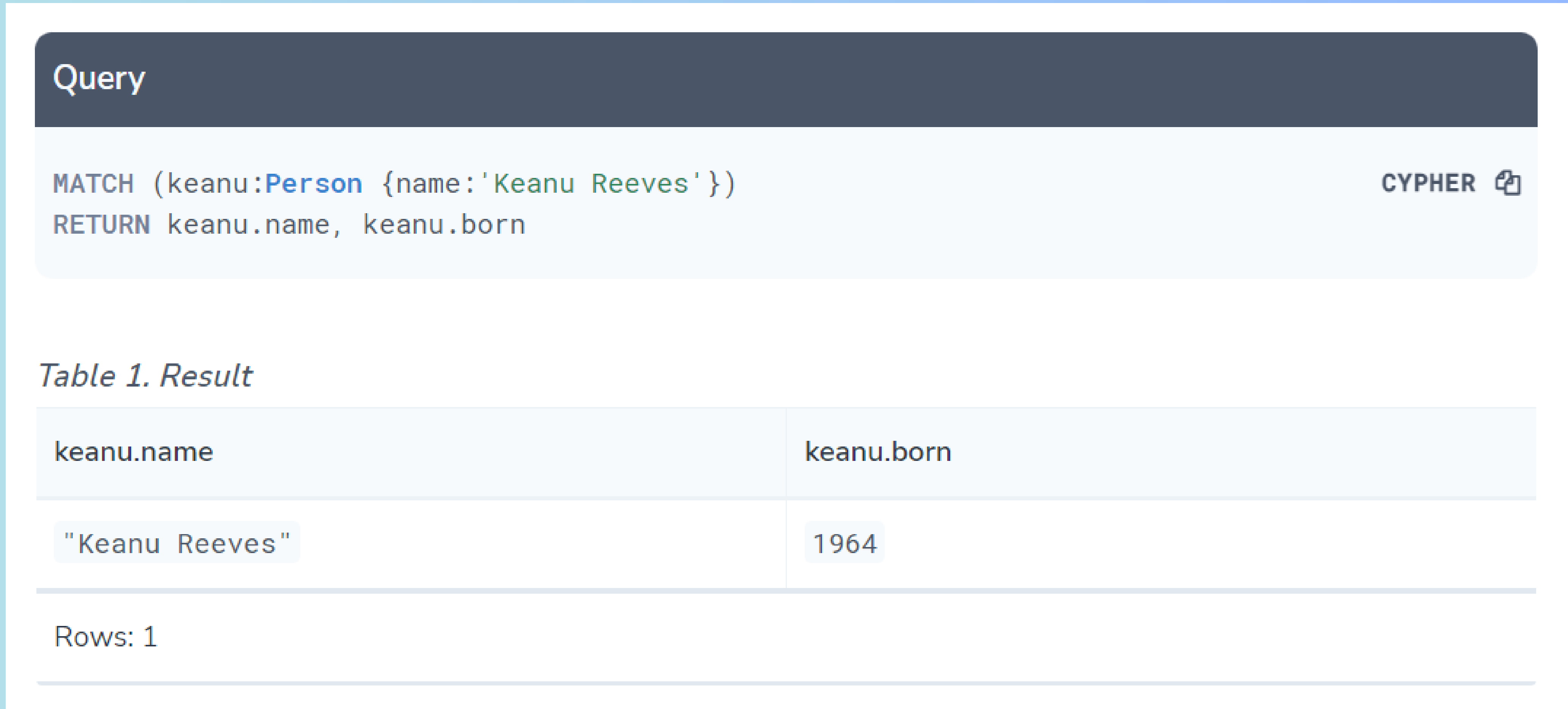
Graph Database Implementation

Graph Database Implementations:

- There are many graph database software available such as Neo4j Graph Database, ArangoDB, Amazon Neptune, Dgraph, OrientDB. And there are different graph query languages such as :
- GraphQL - It is best for complex systems and microservices.
- Gremlin - It allows the users to do procedural and descriptive graph traversals.
- Cypher - It is well-suited for data analytics and application development.
- SPARQL - It is best for integrating data from various intersecting domains.
- But we will be talking about one as an example to understand a little more on how it works.

Graph Database Implementation





Query

```
MATCH (keanu:Person {name:'Keanu Reeves'})
RETURN keanu.name, keanu.born
```

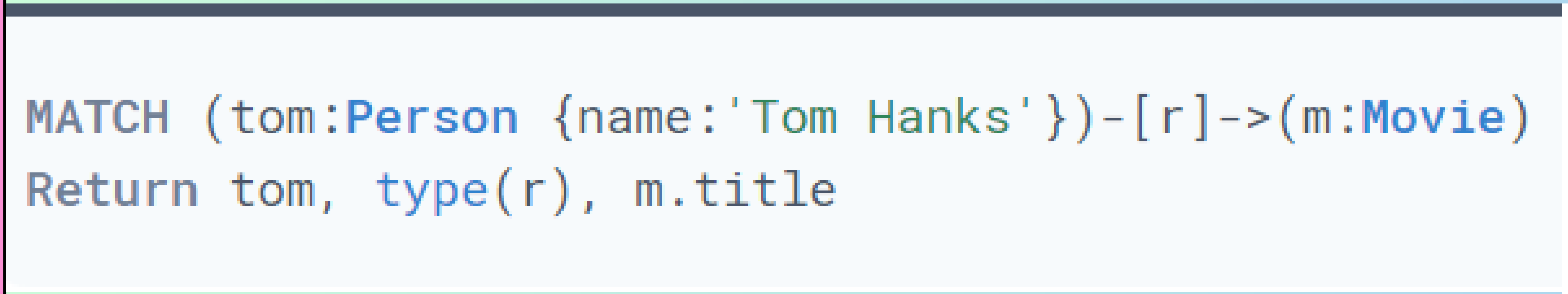
CYPHER

Table 1. Result

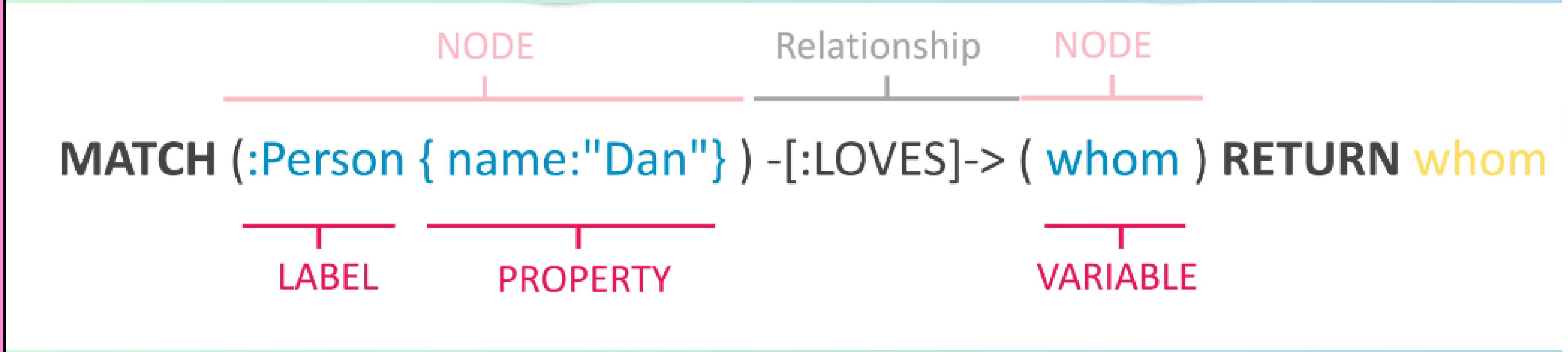
keanu.name	keanu.born
"Keanu Reeves"	1964

Rows: 1

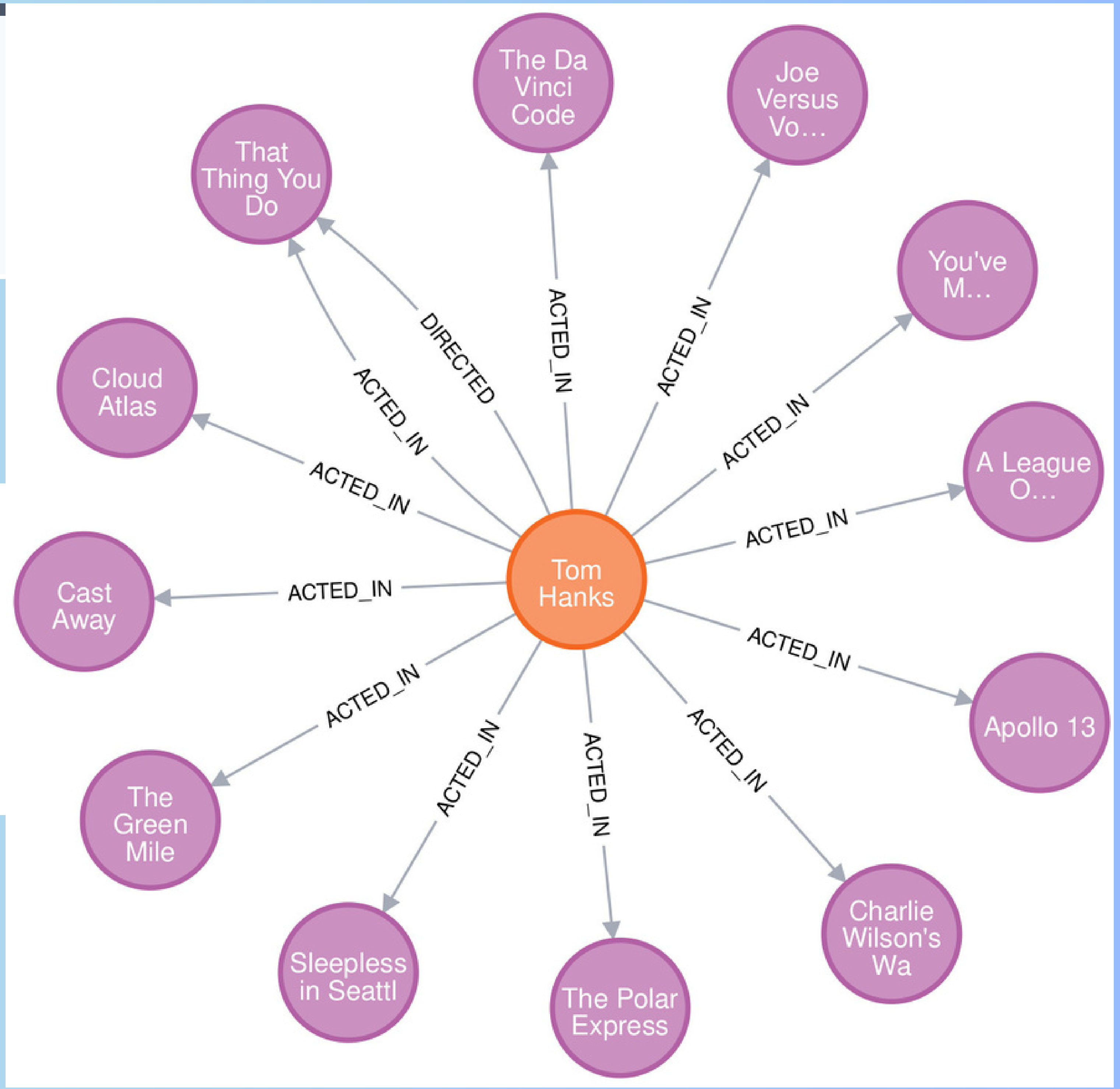
Graph Database Implementation



```
MATCH (tom:Person {name:'Tom Hanks'})-[r]->(m:Movie)
Return tom, type(r), m.title
```



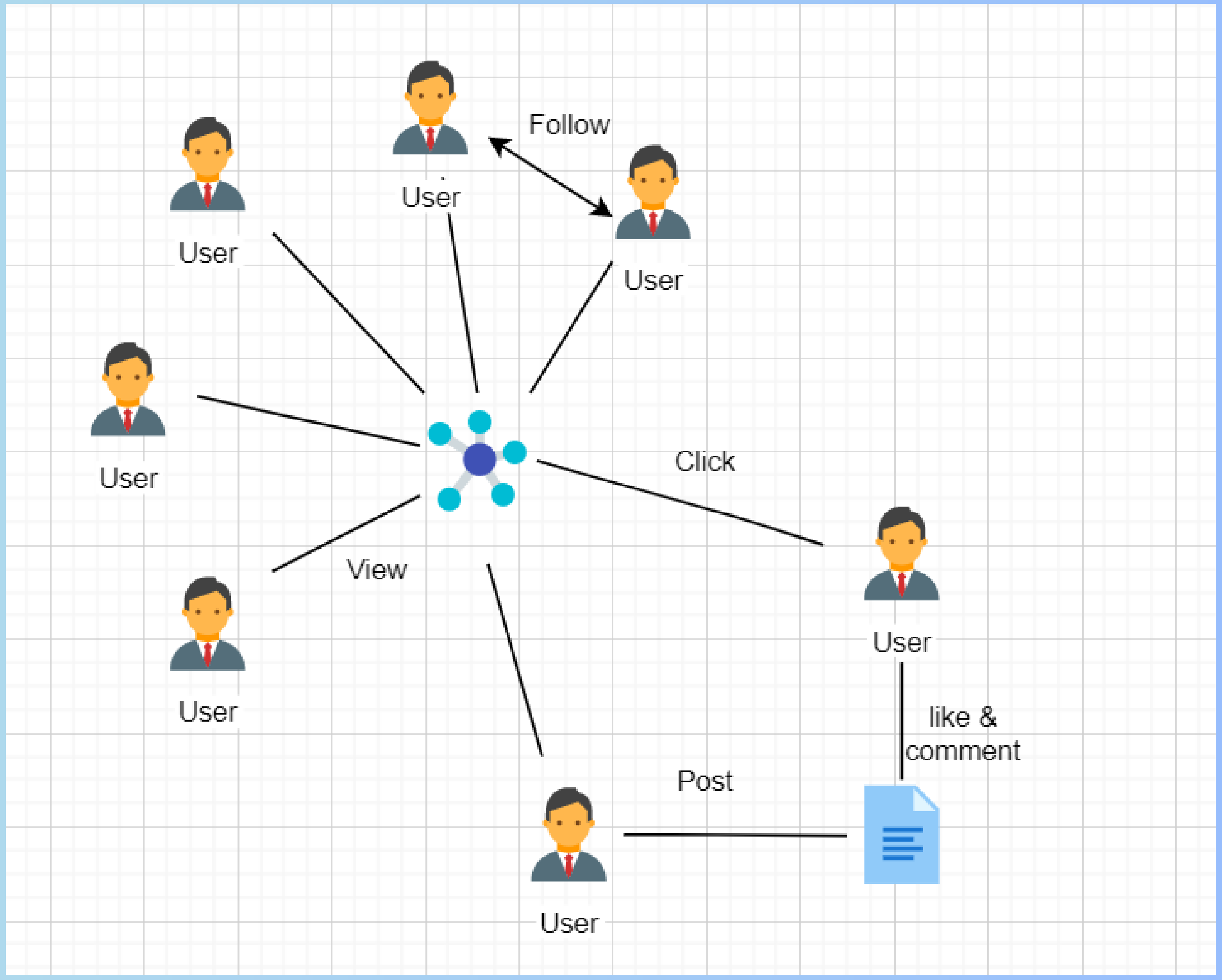
```
MATCH (:Person { name:"Dan"} ) -[:LOVES]-> ( whom ) RETURN whom
```



Graph Database Implementation

Best Practices:

- Keep your data model simple and intuitive.
- Use labeling and indexing to optimize your queries.
- Use the appropriate querying language for your use case.
- Use a highly scalable graph database implementation.
- Monitor and optimize the performance of your graph database regularly.



Use cases of GraphDB

Some use cases include:

Fraud detection: GraphDBs provide near real-time capabilities that can be deployed for fraud detection. The approaches include anomaly detection, pattern recognition, and network community analysis. For example, if multiple fraudulent credit card transactions have been made using the same IP address or shipping address, it may indicate the involvement of a fraud ring.

Recommendation Engines: By enabling storing relationships between information categories such as customer interests, friends, and purchase history, Graph DBs can make product recommendations.

Social Graph: Social network administrators can better understand the relationships between members which helps to surface content relevant to a member and ensure loyalty to the site.

When not to use GraphDB

Graph databases are not ideal for all use cases, especially if:

- Simple data structures:
  - If data has a simple structure with few relationships between entities, a graph database may be overkill.
- Limited budget:
  - Graph databases can be expensive to implement and maintain, especially if you need to store large amounts of data or require a lot of processing power.
- Limited scalability:
  - While GraphDB is highly scalable and can handle large amounts of data, it may not be the best choice for very large datasets that require extreme scalability or very high throughput. In these cases, a distributed data storage solution may be more appropriate.
- Large data sets with low complexity:
  - If your data set is large but has a low level of complexity, other databases may be more appropriate. Graph databases are optimized for data with complex relationships, and may not be as efficient for simple data sets.

Instances of bad implementation of Graph DB

- Using a graph database to store and manage user authentication data, such as usernames and passwords.
    - While graph databases are designed to handle complex relationships between data points, they are not typically used for storing sensitive data like passwords. Instead, traditional relational databases with built-in encryption and other security features are better suited for this type of data.
- Another bad use case for graph databases would be for applications with highly transactional workloads. Graph databases can handle transactions, but they may not be the best choice for applications that require very high levels of transactional consistency or atomicity. In these cases, relational databases are generally better suited for these types of applications.

Advantages

- Simplified Data Modeling:
  - Graph databases allow for flexible data modeling, as they are designed to represent complex relationships between data points using edges instead of tables.
- Performance:
  - Faster query response times to find connections or links, compared to traditional relational databases.
- Scalability:
  - Graph databases are horizontally scalable, meaning they can be easily distributed across multiple servers to handle large and complex datasets.
- Reduced complexity:
  - Graph DB simplifies application development by reducing the need for complex join queries and data normalization.
- Better insights:
  - Better understand patterns and trends in their data, and make more informed decisions based on that information.

Drawbacks

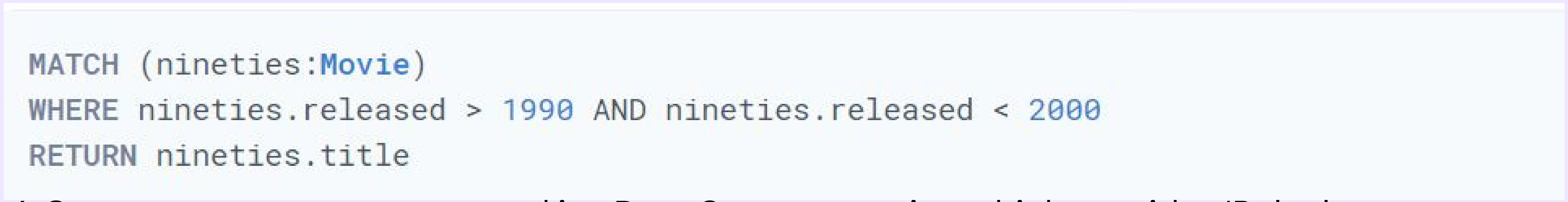
- Learning Curve:
  - Requires a different way of thinking about data modeling and querying compared to traditional relational databases.
- Complexity:
  - Can become very complex as the data model grows in size, making it challenging to manage and optimize performance.
- Scalability:
  - Although graph databases can scale horizontally, adding more servers to the cluster, there can be limitations in terms of scalability due to the nature of graph queries and the need to traverse multiple nodes and edges.
- Cost:
  - Generally more expensive than traditional relational databases due to licensing fees and the cost of hardware required to handle complex queries and large datasets.
- Limited Tooling:
  - Graph databases have fewer tools and integrations available, which can make it harder to work with and integrate into existing workflows.

Key Takeaways & Challenges

1. Graph databases are not a replacement for RDBMS but only an alternative when the relationships between objects are of as much significance as the objects themselves.
2. Graph Compute Engines vs Graph Databases
    a. Graph-only databases excel at graph processing but they are rarely optimal for all the workloads in a typical application. Conversely, many applications—even if their predominant workloads are satisfied by a non-graph database technology—sometimes need to be able to perform efficient graph traversals.
    b. Graph compute engines provide a solution for databases that want to merge graph processing with other data models and for graph processing that needs to work across massive distributed datasets.
3. Transactional(OLTP) & Analytical(OLAP) Graph databases
    a. OLTP databases are typically used for applications that require high-speed data processing, such as financial trading systems or e-commerce platforms
    b. OLAP databases are typically used for business intelligence and data warehousing applications.

Questions?

1. Which database would you use for the following examples? (RDBMS/GraphDB)
   a. Social networking platform
   b. Financial application
   c. Human Resource Management
2. While using Relational DBs, we can find links by joining tables, why is it more advantageous when using GraphDB?
3. What would be the output of this query?



```
MATCH (nineties:Movie)
WHERE nineties.released > 1990 AND nineties.released < 2000
RETURN nineties.title
```

4. Can you name a company and its Data Storage service which provides 'Polyglot Persistence?'

References:

- Harrison, G. (2015). Next generation databases: Nosql, NewSQL, and Big Data. Apress.
- Kleppmann, M. (2021). Designing data-intensive applications: The big ideas behind reliable, scalable, and maintainable systems. O'Reilly.
- Forbes article: The birth of graph databases: How neo4j built its product and category. Neo4j Graph Data Platform. (2019, August 23). Retrieved April 1, 2023, from https://neo4j.com/news/birth-graph-databases-neo4j-built-product-category/
- eBook: Robinson, I., Webber, J., & Eifrem, E. (n.d.). Graph databases: New opportunities for connected data. O'Reilly Media, Inc.
- Slide 11 Image: https://www.simplilearn.com/tutorials/data-structure-tutorial/graphs-in-data-structure
- Slide 12 Image: https://neo4j.com/graphacademy/training-intro-35/01-introneo-3-5-intro-graph-databases/
- Slide 17- 18 Images: https://neo4j.com/docs/cypher-manual/current/introduction/cypher_tutorial/
- Slide 19 Image: https://www.nebula-graph.io/posts/review-on-graph-databases

Thank you for your attention!