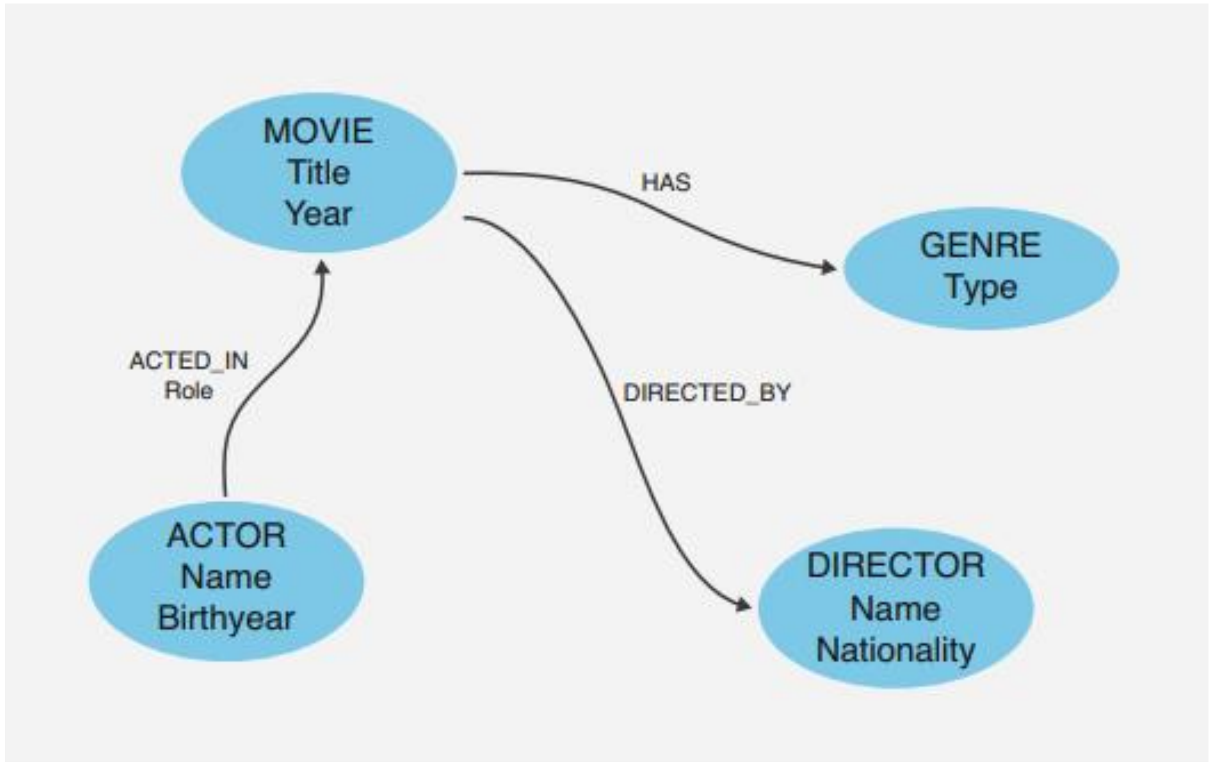
# Lecture 7: Graphical databases pt.2

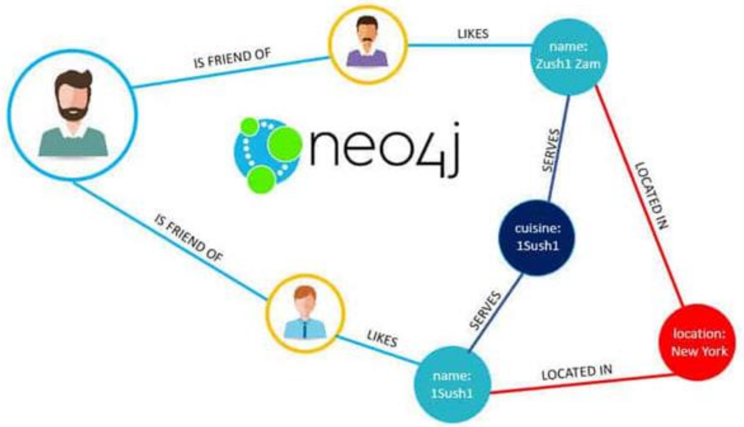
**Dr Anesu Nyabadza**

## Neo4J



Property graphs consist of nodes (concepts, objects) and directed edges (relationships) connecting the nodes. Both nodes and edges are given a label and can have properties. Properties are given as attribute-value pairs with the names of attributes and the respective values

## Neo4J-stucture



Neo4j is ACID-compliant (Atomicity, Consistency, Isolation, Durability), which ensures data integrity and consistency even in the face of failures or concurrent transactions.
ACID compliance ensures that transactions are performed consistently and predictably, which is critical for applications that require strict data consistency.

Neo4j is highly scalable and can handle large-scale graph datasets with billions of nodes and relationships.
It supports **horizontal scaling through clustering and sharding**, allowing users to distribute graph data across multiple machines for improved performance and reliability.

Neo4j has a vibrant ecosystem with a rich set of tools, libraries, and integrations that extend its functionality.
This ecosystem includes visualization tools, data import/export utilities, integration with programming languages like Java, Python, and JavaScript, and support for popular frameworks and platforms.

Neo4j is used in a wide range of applications and industries, including **social networking, recommendation systems, fraud detection, network and IT operations management, knowledge graphs, and biomedical research**

Another **EXAM type** example where we store details of movies and their genres. Each movie will have a title, and each genre will have a name

Take 10 min to write down the code and the query to present the results

## Solution

```
//Create movie Node
CREATE (:Movie {title: "Iron Man"}),
       (:Movie {title: "The Avengers"}),
       (:Movie {title: "Guardians of the Galaxy"}),
       (:Movie {title: "Black Panther"})

//Create genre node
CREATE (:Genre {name: "Action"}),
       (:Genre {name: "Adventure"}),
       (:Genre {name: "Sci-Fi"}),
       (:Genre {name: "Superhero"})

//Create relationships
MATCH (movie:Movie),(genre:Genre)
WHERE movie.title = "Iron Man" AND genre.name = "Action"
CREATE (movie)-[:BELONGS_TO]->(genre)

MATCH (movie:Movie),(genre:Genre)
WHERE movie.title = "The Avengers" AND genre.name IN ["Action", "Adventure", "Sci-Fi", "Superhero"]
CREATE (movie)-[:BELONGS_TO]->(genre)

MATCH (movie:Movie),(genre:Genre)
WHERE movie.title = "Guardians of the Galaxy" AND genre.name IN ["Action", "Adventure", "Sci-Fi", "Superhero"]
CREATE (movie)-[:BELONGS_TO]->(genre)

MATCH (movie:Movie),(genre:Genre)
WHERE movie.title = "Black Panther" AND genre.name IN ["Action", "Adventure", "Sci-Fi", "Superhero"]
CREATE (movie)-[:BELONGS_TO]->(genre)
```

## Solution cont….

```
// get the movies and associated genres
// is used to comment code, similar to MongoDB
MATCH (movie:Movie)-[:BELONGS_TO]->(genre:Genre)
RETURN movie.title, genre.name
```

## Homework

**Let's consider an example where we store details of students, courses, and their enrollments.**

## Homework- solution

```
//Create Nodes for Students
CREATE (:Student {name: "Alice"}),
       (:Student {name: "Bob"}),
       (:Student {name: "Charlie"})

//Create Nodes for Courses
CREATE (:Course {title: "Mathematics"}),
       (:Course {title: "History"}),
       (:Course {title: "Computer Science"})

//Relationships
MATCH (student:Student),(course:Course)
WHERE student.name = "Alice" AND course.title = "Mathematics"
CREATE (student)-[:ENROLLED_IN]->(course)

MATCH (student:Student),(course:Course)
WHERE student.name = "Bob" AND course.title IN ["Mathematics", "History"]
CREATE (student)-[:ENROLLED_IN]->(course)

MATCH (student:Student),(course:Course)
WHERE student.name = "Charlie" AND course.title IN ["History", "Computer Science"]
CREATE (student)-[:ENROLLED_IN]->(course)
```

## Homework- solution cont…

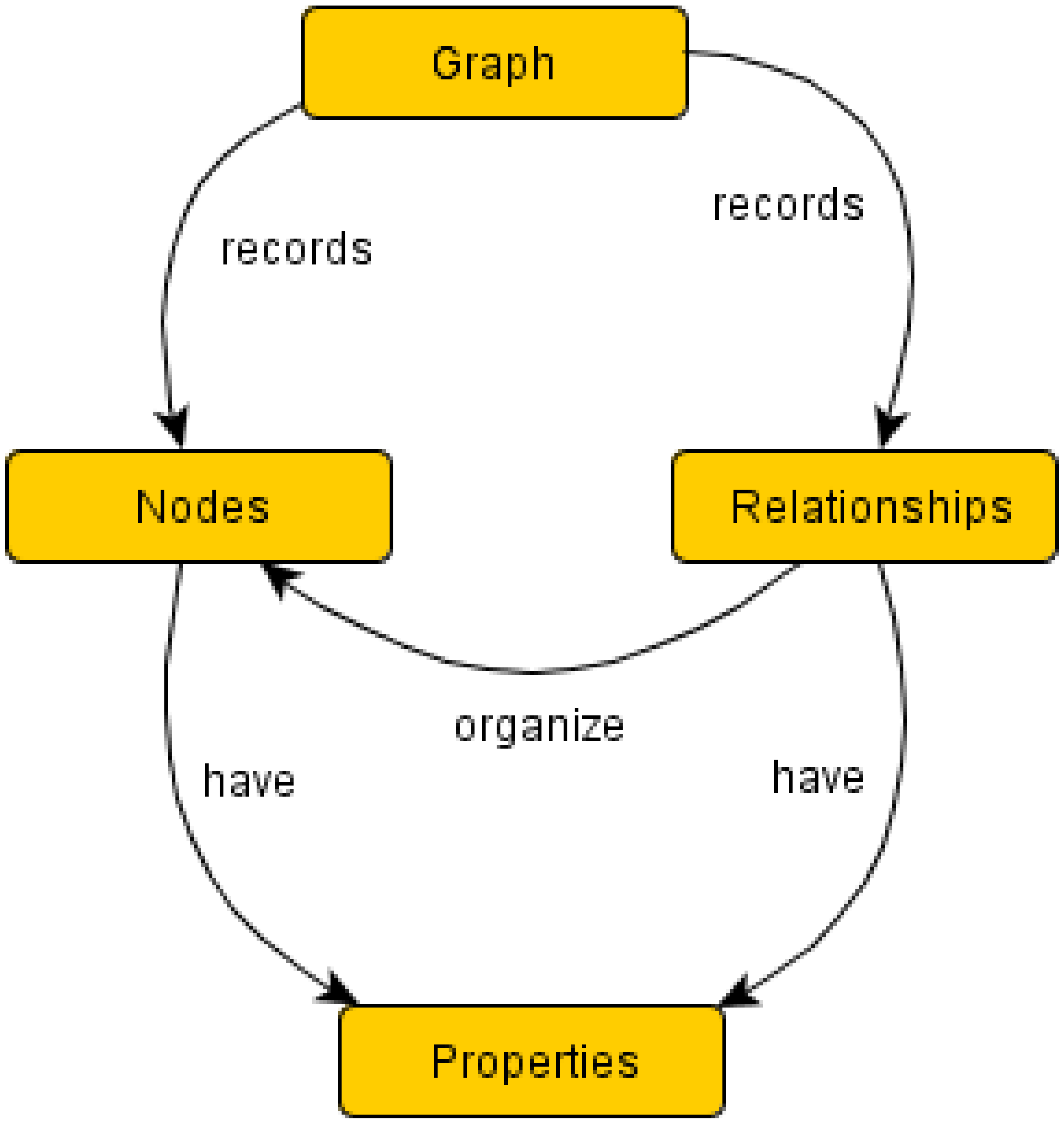
```
//query students information
MATCH (student:Student)-[:ENROLLED_IN]->(course:Course)
RETURN student.name, course.title
```

## Structure of Neo4j

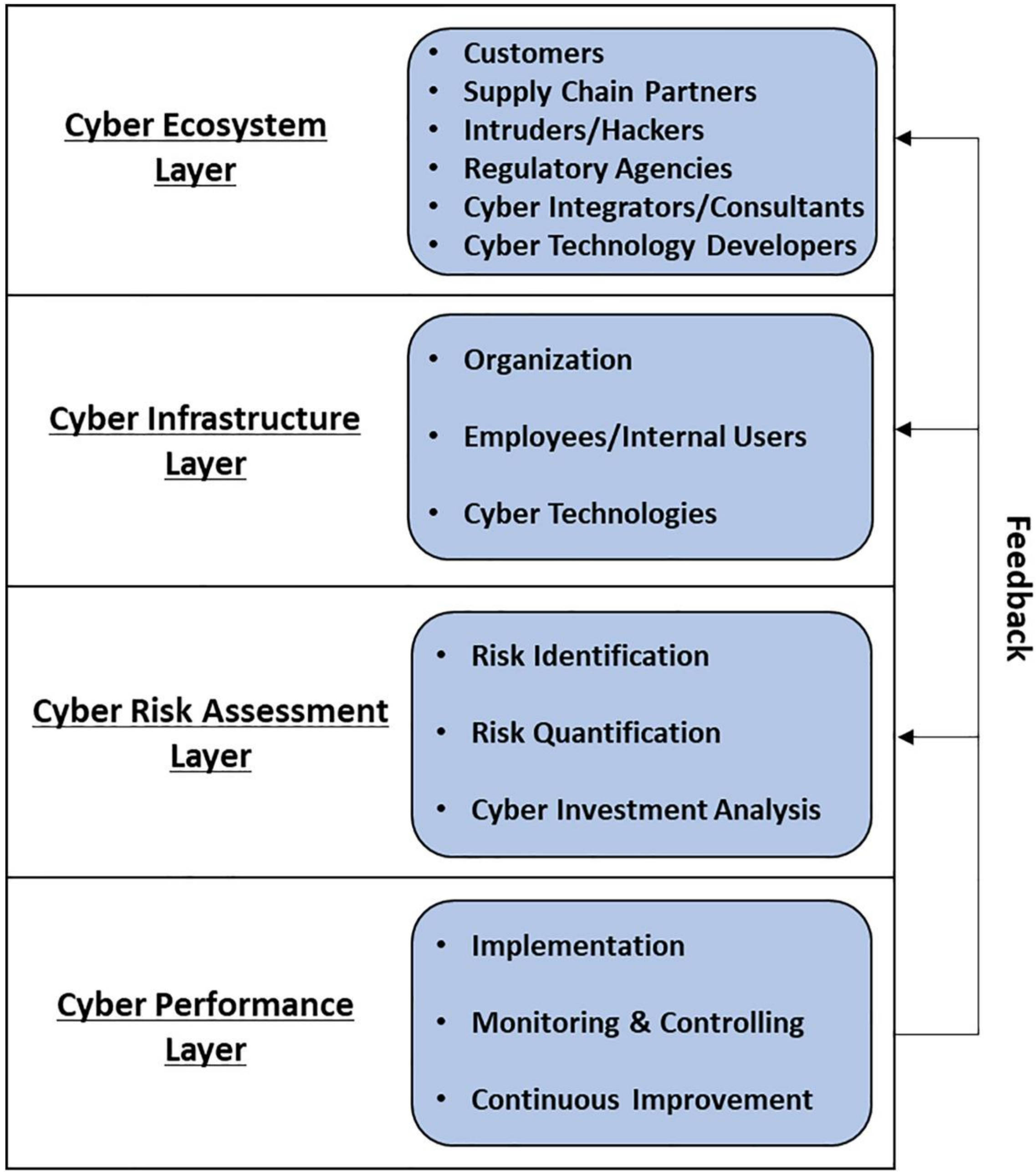
Neo4j is a **graph database** that organizes data using **nodes, relationships, and properties**.
Nodes represent entities like people, products, and events.
Relationships represent links between nodes, indicating how entities are related to one another.
Properties are key-value pairs associated with nodes and relationships that provide further information about entities and connections.

Neo4j's data model is based on the property graph model, which uses nodes, relationships, and properties as its primary building parts.
To categorize and offer extra semantics, nodes, relationships, and properties can be assigned labels and types.
**Cypher** is the query language used to communicate with Neo4j; it allows users to execute graph traversal, pattern matching, and property-based searches.

## Structure of Neo4j



## Database security again!



## Backup strategies in Neo4j- database security

Neo4j offers a **variety of backup techniques** to meet different deployment scenarios and requirements. These strategies include **Full backups, Incremental Backups and Scheduled Backups**

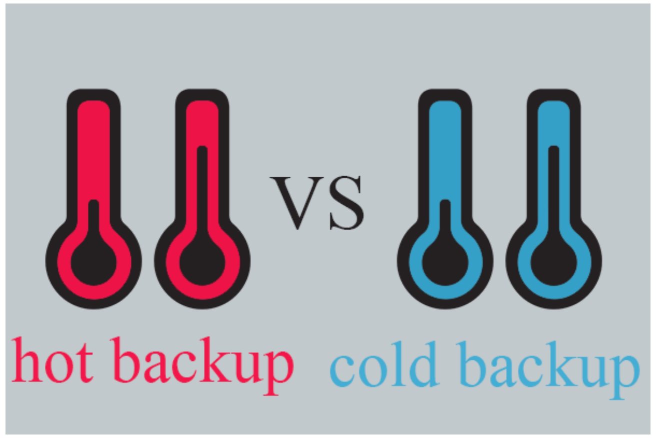
1. **Full backups** preserve the whole graph database, including nodes, relationships, properties, and indexes. They provide a complete snapshot of the database at a given point in time.
2. **Incremental backups** simply record changes made since the previous backup, lowering backup times and storage needs. They are especially effective for huge datasets that require regular updates.
3. **Scheduled Backups,** Neo4j allows users to set up automated backups at regular intervals to maintain data security and consistency.

## Backup strategies in Neo4j- database security

Neo4j provides built-in tools and utilities to facilitate backup operations

**Online Backup Tool:** Neo4j features an online backup tool that allows users to backup their databases while they are still online and accessible. This minimizes downtime during backup operations.

**Hot Backup:** Users can create backups of a running Neo4j database without disrupting current transactions or processes. This provides ongoing availability and data integrity during the backup process.



**Hot backup (Minimizes downtime, Facilitates continuous availability but increases complexity In infrastructure**) , also known as online backup, involves creating a backup of a database while it is still running and accessible to users.
**Cold backup (Simplicity, Reduced resource usage, but requires downtime and stale data**) also known as offline backup, involves shutting down the database before creating a backup.

Generally, The choice between hot and cold backups depends on factors such as the criticality of the data, acceptable downtime, resource availability, and infrastructure constraints.

## Backup strategies in Neo4j- database security

Neo4j offers **flexible configuration** options to customize backup settings according to specific requirements. Users can configure parameters such as backup frequency, retention policies, compression settings, and backup storage locations.

Neo4j supports **a variety of backup storage solutions**, including local disk storage, network-attached storage (NAS), cloud storage providers, and distributed file systems. This flexibility enables users to select the storage solution that best meets their requirements in terms of performance, scalability, and cost effectiveness.

Neo4j includes detailed documentation and guidance for backup and restoration methods, as well as best practices, recommendations, and troubleshooting tips. These resources enable users to properly manage backup operations and recover data in the event of data loss or corruption.

## Backup strategies in Neo4j- database security

Neo4j offers **flexible configuration** options to customize backup settings according to specific requirements. Users can configure parameters such as backup frequency, retention policies, compression settings, and backup storage locations.

Neo4j supports **a variety of backup storage solutions**, including local disk storage, network-attached storage (NAS), cloud storage providers, and distributed file systems. This flexibility enables users to select the storage solution that best meets their requirements in terms of performance, scalability, and cost effectiveness.

Neo4j includes detailed documentation and guidance for backup and restoration methods, as well as best practices, recommendations, and troubleshooting tips. These resources enable users to properly manage backup operations and recover data in the event of data loss or corruption.