1. Graph Data Model Fundamentals

- **Nodes (Vertices):** Represent entities with properties (key-value pairs), such as a person with attributes like "name" and "occupation."
- **Edges (Relationships):** Connect nodes and represent relationships between them, also with properties, like "since" in a "KNOWS" relationship.
- **Labels:** Used to categorize nodes (e.g., "person" or "car").
- **Paths:** An ordered sequence of connected nodes with edges.
- **Flavors of Graphs:**
    - Connected vs. Disconnected
    - Weighted vs. Unweighted
    - Directed vs. Undirected
    - Acyclic vs. Cyclic
    - Sparse vs. Dense
    - Trees (a special case of connected, acyclic graphs)
- **Applications:** Social networks, web data, and biological/chemical systems.

2. Graph Algorithms

- **Pathfinding:** Common operations like finding the shortest path between nodes, using algorithms like BFS (Breadth-First Search) and DFS (Depth-First Search).
- **Centrality & Community Detection:** Identifying important nodes and detecting clusters of strongly connected nodes.
- **Famous Graph Algorithms:**
    - **Dijkstra's Algorithm** for shortest paths in weighted graphs.
    - *A Algorithm** (a variation of Dijkstra's).
    - **PageRank** for determining node importance based on incoming relationships.

3. Neo4j: A Graph Database System

- **Features:**
    - ACID compliance (Atomicity, Consistency, Isolation, Durability).
    - Distributed computing support.
    - Schema-optional, though schemas can be imposed.
- **Query Language – Cypher:** A declarative language for graph queries, similar to SQL but designed for graphs.
- **APOC Plugin:** Extends Cypher with additional functions and procedures.
- **Graph Data Science Plugin:** Provides efficient implementations of common graph algorithms.

4. Neo4j Setup with Docker Compose

- **Docker Compose:** Manages multi-container applications with a YAML configuration file. Helps in setting up a Neo4j instance with persistent data storage and plugins.
- **Configuration:** The `docker-compose.yaml` file defines Neo4j's settings, including ports, environment variables (like authentication credentials), and volumes for data persistence.
- **Commands:** Useful commands include `docker-compose up` for starting the container and `docker-compose down`for stopping it.

5. Interacting with Neo4j

- **Creating Nodes and Relationships:**
    - Nodes can be created with properties using the `CREATE` command.
    - Relationships are created with the `MATCH` and `CREATE` commands.

Example Cypher query to create a relationship:

```
MATCH (alice:User {name:"Alice"})
MATCH (bob:User {name: "Bob"})
CREATE (alice)-[:KNOWS {since: "2022-12-01"}]->(bob)
```

    -

Querying Data: Example for matching nodes based on properties and returning results:

```
MATCH (usr:User {birthPlace: "London"})
RETURN usr.name, usr.birthPlace
```

-

6. Importing Data into Neo4j

- **CSV Import:**
    - Data can be imported from CSV files using the `LOAD CSV WITH HEADERS` command.

Example for importing movie data:

```
LOAD CSV WITH HEADERS FROM 'file:///netflix_titles.csv' AS line
CREATE(:Movie {id: line.show_id, title: line.title, releaseYear:
line.release_year})
```

    -

Importing Relationships: Data involving relationships (e.g., directors and movies) can be handled using `MERGE` to avoid duplicates and `MATCH` to link nodes. Example:

```
MATCH (m:Movie {title: "Ray"})
MATCH (p:Person {name: "Director"})
```

- `CREATE (p)-[:DIRECTED]->(m)`