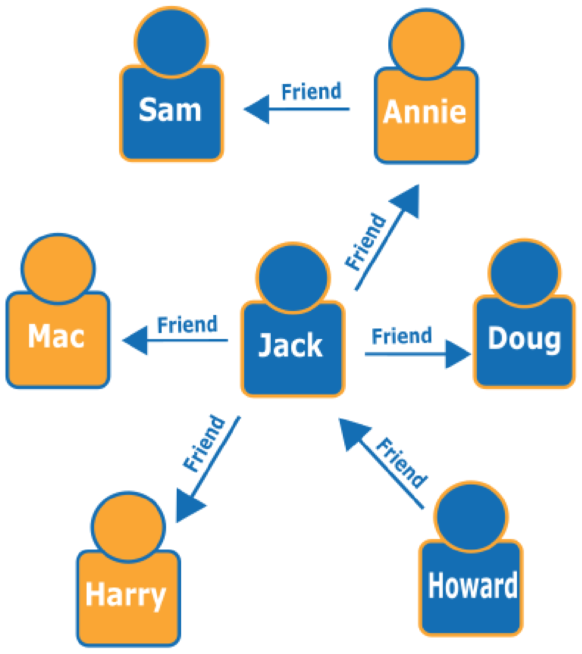
<u>**Unit - 5**</u>

Graph Databases:

What Is a Graph Database, Graph Database using Neo4j, Advantages of Neo4j, CQL DataTypes, Neo4j CQL Operators, Create Nodes, Create Relationships, Index, Constraint, Select data with match, Import data from CSV, Drop an Index, Drop a Constraint, Deleting Nodes, Deleting Relationships. Suitable Use Cases, and When Not to Use.

❖ What Is a Graph Database?

- ➢ Graph databases are purpose-built to store and navigate relationships. Relationships are first-class citizens in graph databases, and most of the value of graph databases is derived from these relationships. Graph databases use nodes to store data entities, and edges to store relationships between entities. An edge always has a start node, end node, type, and direction, and an edge can describe parent-child relationships, actions, ownership, and the like. There is no limit to the number and kind of relationships a node can have.
- ➢ A graph in a graph database can be traversed along specific edge types or across the entire graph. In graph databases, traversing the joins or relationships is very fast because the relationships between nodes are not calculated at query times but are persisted in the database. Graph databases have advantages for use cases such as social networking, recommendation engines, and fraud detection, when you need to create relationships between data and quickly query these relationships.
- ➢ The following graph shows an example of a social network graph. Given the people (nodes) and their relationships (edges), you can find out who the "friends of friends" of a particular person are—for example, the friends of Howard's friends.



❖ Graph Database using Neo4j

- ➢ Neo4j is an open-source, NoSQL, native graph database that provides an ACID-compliant transactional backend for your applications that has been publicly available since 2007.
- ➢ Neo4j is a *native graph database*, which means that it implements a true graph model all the way down to the storage level. The data isn't stored as a "graph abstraction" on top of another technology, it's stored just as you whiteboard it. This is important because it's the reason why Neo4j outperforms other graphs and stays so flexible. Beyond the core graph, Neo4j provides what you'd expect out of a database; ACID transactions, cluster support, and runtime failover. This stability and maturity is why it's been used in production scenarios for large enterprise workloads for years.
- ➢ What makes Neo4j the easiest graph to work with?

- Cypher, a declarative query language similar to SQL, but optimized for graphs. Now used by other databases like SAP HANA Graph and Redis graph via the openCypher project.
- Constant time traversals in big graphs for both depth and breadth due to efficient representation of nodes and relationships. Enables scale-up to billions of nodes on moderate hardware.
- Flexible property graph schema that can adapt over time, making it possible to materialize and add new relationships later to shortcut and speed up the domain data when the business needs change.
- Drivers for popular programming languages, including Java, JavaScript, .NET, Python, and many more.

❖ Advantages of Neo4j

- ➢ It is very easy to represent connected data.
- ➢ It is very easy and faster to retrieve/traversal/navigation of more Connected data.
- ➢ It represents semi-structured data very easily.
- ➢ Neo4j CQL query language commands are in humane readable format and very easy to learn.
- ➢ It uses simple and powerful data model.
- ➢ It does NOT require complex Joins to retrieve connected/related data as it is very easy to retrieve it's adjacent node or relationship details without Joins or Indexes.

❖ CQL DataTypes

- ➢ CQL stands for Cypher Query Language. It is a query language for Neo4j just like SQL is a query language for Oracle or MySQL.
- ➢ Neo4j CQL Features
  - CQL is a query language for Neo4j Graph Database.
  - Is a declarative pattern-matching language.
  - The syntax of CQL is same like SQL syntax.
  - Syntax of CQL is very simple and in human readable format.
- ➢ Similarity between Oracle SQL and Neo4j CQL
  - Oracle and Neo4j CQL both has simple commands to do database operations.
  - Both support clauses like WHERE, ORDER BY, etc., to simplify complex queries.
  - Oracle and Neo4j CQL supports some Relationship Functions and functions such as String, Aggregation.
- ➢ The Neo4j CQL data types are similar to Java language data types. They are used to define properties of a node or a relationship.
- ➢ A list of Neo4j CQL data types:

CQL Data Type	**Usage**
Boolean	It is used to represent Boolean literals: True, False.
byte	It is used to represent 8-bit integers.
short	It is used to represent 16-bit integers.
int	It is used to represent 32-bit integers.
long	It is used to represent 64-bit integers.
float	Float is used to represent 32-bit floating-point numbers.
double	Double is used to represent 64-bit floating-point numbers.
char	Char is used to represent 16-bit characters.

String	String is used to represent strings.

❖ **Neo4j CQL Operators**

➢ Neo4j CQL Operators can be categorized in following types:

- Mathematical Operators: i.e. +, -, *, /, %, ^
- Comparison Operators: i.e. +, <>, <, >, <=, >=
- Boolean Operators: i.e. AND, OR, XOR, NOT
- String Operators: i.e. +
- List Operators: i.e. +, IN, [X], [X?..Y]
- Regular Expression: i.e. =-
- String matching: i.e. STARTS WITH, ENDS WITH, CONSTRAINTS

➢ Let's see the two most used Neo4j CQL Operators:

➢ Boolean Operators

Following is a list of Boolean operators which are used in Neo4j CQL WHERE clause to support multiple conditions:

Boolean Operators	**Description**
AND	It is a neo4j CQL keyword to support AND operation. It is like SQL AND operator.
OR	It is a Neo4j CQL keyword to support OR operation. It is like SQL AND operator.
NOT	It is a Neo4j CQL keyword to support NOT operation. It is like SQL AND operator.
XOR	It is a Neo4j CQL keyword to support XOR operation. It is like SQL AND operator.

Comparison Operators

➢ A list of Neo4j CQL Comparison Operators used with WHERE clause:

Boolean operators	**Description**
=	It is a Neo4j CQL "equal to" operator.
< >	It is a Neo4j CQL "not equal to" operator.
<	It is a Neo4j CQL "less than" operator.
>	It is a Neo4j CQL "greater than" operator.
<=	It is a Neo4j CQL "less than or equal to" operator.
> =	It is a Neo4j CQL"greater than or equal to" operator.

❖ Create Nodes

➢ *The CREATE clause is used to create nodes and relationships.*

- In the CREATE clause, patterns are used extensively. Read Patterns for an introduction.
- Create nodes
- Create single node
- Creating a single node is done by issuing the following query:
  - CREATE (n)
- Create multiple nodes
  - Creating multiple nodes is done by separating them with a comma.
  - CREATE (n), (m)
- Create a node with a label
  - To add a label when creating a node, use the syntax below:
  - CREATE (n:Person)
  - Create a node with multiple labels
- To add labels when creating a node, use the syntax below. In this case, we add two labels.
  - CREATE (n:Person:Swedish)
  - Create node and add labels and properties
  - When creating a new node with labels, you can add properties at the same time.
  - CREATE (n:Person {name: 'Andy', title: 'Developer'})
  - Return created node
- Creating a single node is done by issuing the following query:
  - CREATE (a {name: 'Andy'})
  - RETURN a.name
  - The name of the newly-created node is returned.

❖ Create Relationships

➢ Create a relationship between two nodes

- To create a relationship between two nodes, we first get the two nodes. Once the nodes are loaded, we simply create a relationship between them.

  -
    ```
    MATCH
      (a:Person),
      (b:Person)
    WHERE a.name = 'A' AND b.name = 'B'
    CREATE (a)-[r:RELTYPE]->(b)
    RETURN type(r)
    ```
- The created relationship is returned by the query.

  -
    ```
    type(r)
    "RELTYPE"
    ```

- Create a relationship and set properties
  - Setting properties on relationships is done in a similar manner to how it's done when creating nodes. Note that the values can be any expression.
    ```
    MATCH
      (a:Person),
      (b:Person)
    WHERE a.name = 'A' AND b.name = 'B'
    CREATE (a)-[r:RELTYPE {name: a.name + '<->' + b.name}]->(b)
    RETURN type(r), r.name
    ```

❖ Index

➢ A database index is a redundant copy of some of the data in the database for the purpose of making searches of related data more efficient. This comes at the cost of additional storage space and slower writes, so deciding what to index and what not to index is an important and often non-trivial task.

➢ Once an index has been created, it will be managed and kept up to date by the DBMS. Neo4j will automatically pick up and start using the index once it has been created and brought online.

➢ There are multiple index types available:

- Range index.

        - Lookup index.
        - Text index.
        - Point index.
        - Full-text index.

❖ Constraint

- ➢ In Neo4j, a constraint is used to place restrictions over the data that can be entered against a node or a relationship.
    - ➢ There are two types of constraints in Neo4j:
        - Uniqueness Constraint: It specifies that the property must contain a unique value. (For example: no two nodes with an player label can share a value for the Goals property.)
        - Property Existence Constraint: It makes ensure that a property exists for all nodes with a specific label or for all relationships with a specific type.
        - Create a Uniqueness Constraint
            - CREATE CONSTRAINT ON statement is used to create a uniqueness constraint in Neo4j.
            - CREATE CONSTRAINT ON (Kalam:president) ASSERT Kalam.Name IS UNIQUE
        - Property Existence Constraint
            - Property existence constraint is used to make ensure that all nodes with a certain label have a certain property.
    - ➢ Note: exists property constraint are only available in the Neo4j Enterprise Edition.

❖ Select data with match

- ➢ The MATCH clause allows you to specify the patterns Neo4j will search for in the database. This is the primary way of getting data into the current set of bindings. It is worth reading up more on the specification of the patterns themselves in Patterns.
- ➢ MATCH is often coupled to a WHERE part which adds restrictions, or predicates, to the MATCH patterns, making them more specific. The predicates are part of the pattern description, and should not be considered a filter applied only after the matching is done. *This means that WHERE should always be put together with the MATCH clause it belongs to.*
- ➢ MATCH can occur at the beginning of the query or later, possibly after a WITH. If it is the first clause, nothing will have been bound yet, and Neo4j will design a search to find the results matching the clause and any associated predicates specified in any WHERE part. This could involve a scan of the database, a search for nodes having a certain label, or a search of an index to find starting points for the pattern matching. Nodes and relationships found by this search are available as *bound pattern elements,* and can be used for pattern matching of paths. They can also be used in any further MATCH clauses, where Neo4j will use the known elements, and from there find further unknown elements.

❖ Import data from CSV

- ➢ CSV is a file of comma-separated values, often viewed in Excel or some other spreadsheet tool. There can be other types of values as the delimiter, but the most standard is the comma. Many systems and processes today already convert their data into CSV format for file outputs to other systems, human-friendly reports, and other needs. It is a standard file format that humans and systems are already familiar with using and handling.
- ➢ Giving Neo4j the ability to read and load a CSV file helps reduces the friction of getting data from various formats and systems into Neo4j.
- ➢ Ways to Import CSV Files
    - There are a few different approaches to get CSV data into Neo4j, each with varying criteria and functionality. The option you choose will depend on the data set size, as well as your degree of comfort with various tools.
    - Let us see some of the ways Neo4j can read and import CSV files.
        - LOAD CSV Cypher command: this command is a great starting point and handles small- to medium-sized data sets (up to 10 million records). *Works with any setup, including AuraDB.*
        - neo4j-admin bulk import tool: command line tool useful for straightforward loading of large data sets. *Works with Neo4j Desktop, Neo4j EE Docker image and local installations.*

            - Kettle import tool: maps and executes steps for the data process flow and works well for very large data sets, especially if developers are already familiar with using this tool. *Works with any setup, including AuraDB.*

Drop an Index

- In Neo4j, "DROP INDEX ON" statement is used to drop an index from database. It will permanently remove the index from the database
    - Example:
        - DROP INDEX ON :player(Goals)

Drop a Constraint

- DROP CONSTRAINT statement is used to drop or remove a constraint from the database as well as its associated index.
    - Example:
    - Use the following statement to drop the previously created constraint and its associated index,
        - DROP CONSTRAINT ON (Kalam:president) ASSERT Kalam.Name IS UNIQUE

Deleting Nodes

- In Neo4j, DELETE statement is always used with MATCH statement to delete whatever data is matched. The DELETE command is used in the same place we used the RETURN clause in our previous examples
    - Example
        - MATCH (Kohli:person {Name: "Virat Kohli"}) DELETE Kohli

Deleting Relationships

- Deleting relationship is as simple as deleting nodes. Use the MATCH statement to match the relationships you want to delete.
- You can delete one or many relationships or all relationships by using one statement.
    - Example:
        - Delete the relationship named "PLAYER_OF" from the database:
            - MATCH (Raul)-[r:PLAYER_OF]->(It)
            - DELETE r

Suitable Use Cases, and When Not to Use.

- Let’s look at some suitable use cases for graph databases.
    - **Connected Data Social networks** are where graph databases can be deployed and used very effectively. These social graphs don’t have to be only of the friend kind; for example, they can represent employees, their knowledge, and where they worked with other employees on different projects. Any link-rich domain is well suited for graph databases. If you have relationships between domain entities from different domains (such as social, spatial, commerce) in a single database, you can make these relationships more valuable by providing the ability to traverse across domains.
    - **Routing, Dispatch, and Location-Based Services** Every location or address that has a delivery is a node, and all the nodes where the delivery has to be made by the delivery person can be modeled as a graph of nodes. Relationships between nodes can have the property of distance, thus allowing you to deliver the goods in an efficient manner. Distance and location properties can also be used in graphs of places of interest, so that your application can provide recommendations of good restaurants or entertainment options nearby. You can also create nodes for your points of sales, such as bookstores or restaurants, and notify the users when they are close to any of the nodes to provide location-based services.
    - **Recommendation Engines** As nodes and relationships are created in the system, they can be used to make recommendations like “your friends also bought this product” or “when invoicing this item, these other items are usually invoiced.” Or, it can be used to make recommendations to travelers mentioning that when other visitors come to Barcelona they usually visit Antonio Gaudi’s creations. An interesting side effect of using the graph databases for recommendations is that as the data size grows, the number of nodes and relationships available to make the recommendations quickly increases. The same data can also be used to mine information—for example, which products are always bought together, or which items are always invoiced together; alerts can be raised when these conditions are not met. Like other

recommendation engines, graph databases can be used to search for patterns in relationships to detect fraud in transactions.

- **When Not to Use** In some situations, graph databases may not appropriate. When you want to update all or a subset of entities—for example, in an analytics solution where all entities may need to be updated with a changed property—graph databases may not be optimal since changing a property on all the nodes is not a straightforward operation. Even if the data model works for the problem domain, some databases may be unable to handle lots of data, especially in global graph operations (those involving the whole graph).