- Neo4J is for graph databases
- A graph contains nodes and relationships between nodes
- Each relationship has only one direction, and for a bidirectional relationship you need to make two relationships
- Each relationship can have weights and properties associated with it. For example, a road relationship connecting two place nodes can have a cost and a distance associated with it.
- Each node can have many properties, like a name, ID, and specific things depending on what the node is like something like age or price
- Each node can also have labels, to separate different categories of nodes

- Use the Cypher query language to create, read, update, delete, and query data.
- To query for something
    - use the keyword "**MATCH**"
    - A node is represented as "**()**"
    - Inside the parenthesis, put in a variable. This variable can be anything, like "n" or "x"; careful not to use "node", which is a keyword
    - MATCH(n) matches the nodes, but we need to return it.
    - "**RETURN** n" returns all of the nodes in the application
        - Even though we query for relationships here, the relationships will be returned along with the nodes
- To filter the query
    - MATCH (n:LABEL1) RETURN n
        - This will only return the nodes that have the label "LABEL1"
    - MATCH (n:LABEL2) RETURN n.PROPERTY1
        - This will return the value of the property PROPERTY1, if it exists, for each node that has the label LABEL2.'
    - MATCH (n:LABEL3) RETURN n.PROPERTY1, n.PROPERTY2, n.PROPERTY3, …
        - Can query for multiple properties of the node at the same time
    - MATCH (n:LABEL) RETURN n.name AS name, n.PROPERTY2 AS aliasproperty2name
        - Instead of showing the column heading as "n.name", the title of the property is **Aliased** as "name". This makes the data more readable
    - Query based on properties of node
        - MATCH (specificNode:specificNodeLabel)
        - WHERE specificNode.property = value
        - Return specificNode
        - The "**WHERE**" keyword queries the nodes with the specificNodeLabel such that a certain property of that node is equal to a value. The nodes can be filtered on multiple properties. Something like an id can be used as a filter to only return specific nodes, while specific groups of nodes can be returned based on the value of other properties.
            - To query by ID:

        - MATCH (n1:label1)
        - WHERE ID(n1) = value
    - To return a group through use of other properties, do something like:
        - MATCH (node:label)
        - WHERE node.property >= value
        - RETURN node
            - Returns all nodes such that the property is greater than or equal to value
- Another way to do the same thing is :
    - MATCH (specificNode:specificNodeLabel {property: value}) RETURN specificNode
- For more properties, it's just
    - MATCH (specificNode:specificNodeLabel{prop1: value1, prop2: value2, prop3: value3})
- To return all nodes of a specific label such that the node property is NOT a specific value, use "**<>**"
    - MATCH (node:label)
    - WHERE node.property <> value
    - RETURN node
- You can also do arithmetic on the properties of a node to query more specific things
    - For the NBA BMI example,:
    - MATCH (player: PlAYER)
    - WHERE (player.weight / (player.height * player.height)) > 25
        - Returns all the players who have a BMI over 25
- What if you have multiple conditions?
    - simply use keyword "**AND**"
    - NBA Example:
    - MATCH (player: PLAYER)
    - WHERE player.weight >= 100 AND player.height <= 2
    - RETURN player
        - Returns player nodes whose weight is over 100 and height is less than 2
    - Similarly, can use an "**OR**" operator instead of "AND"
    - To return things that don't fulfill the given conditions, simply use the keywords "**WHERE NOT**" instead of "WHERE"
- To limit the number of nodes that are returned, use the "**LIMIT**" keyword, followed by the maximum number of nodes you want returned
- "**SKIP n**" will simple skip the first n nodes that would be returned
- Things can also be ordered based on a certain property. This is using the "**ORDER BY**" keyword.
    - For example, to order NBA players by descending height:
        - ORDER BY player.height DESC

                - DESC implies descending
                - ASC would be ascending
            - To MATCH nodes of different labels at the same time, simply do:
                - MATCH (n1:Label1), (n2:Label2) RETURN n1, n2
                - You can now add WHERE clauses for both n1 and n2
    - Query for nodes based on relationships
        - Specify the two labels that the relationship is between, and then specify the relationship
        - MATCH (n1:LABEL1) -[relationship:relationshipname]-> (n2:LABEL2)
        - Then, can add a WHERE clause to filter based on properties of team and player nodes
        - RETURN n1
            - Now, all the nodes n1 which have the given relationship to nodes n2 such that the properties of the nodes fulfill the WHERE clause filter are returned
        - Can also filter based on relationship properties
            - MATCH (n1:LABEL1) -[relationship:RELATIONSHIPLABEL]-> (n2:LABEL2)
            - WHERE relationship.property > value
            - RETURN n1
        - Aggregation: Getting data from all the relationships attached to a given node through the relationship properties and then combining them to create a new data property of the node
- Delete a node
    - Query for the node, then use the "**DELETE"** keyword
    - MATCH (n {property: value}) DELETE n
        - However, this will give an error if the node has relationships
            - To do this, add the "**DETACH**" keyword
            - DETACH DELETE n
- Delete a relationship
    - Query for the relationship first
    - MATCH (n1 {property:value}) -[rel:relationship]-> (n2:LABEL2)
    - Now, the relationship can just be deleted with DELETE keyword
    - DELETE rel
- To Delete everything,
    - MATCH(n) DETACH DELTE n
- Creating data
    - To create a node, use the "**CREATE**" keyword
    - A node can have multiple labels!
    - A node also has properties
    - CREATE(:LABEL1:LABEL2:LABEL3 {property1: value1, property2: value2, property3: value3})
    - To create a node while specifying relationship:

    - CREATE (:LABEL1) -[:RELATIONSHIPLABEL {property1:value1}]-> (:LABEL {property1: value1, property2: value2})
    - To create a relationship between existing nodes
        - Filter for each node
        - MATCH (n1:label1 {property1:value1}), (n2:label2 (property1:value1})
        - Create relationship
        - CREATE (n1) - [:relationshiplabel {property1:value1}]-> (n2)
- Updating data
    - Using the "**SET**" keyword
    - MATCH and WHERE to get specific nodes
    - MATCH (n1:label1)
    - WHERE ID(n1) = value
    - Then,
    - SET n1.property1 = value1

- Shortcuts
    - Use "*" to trace out across a bunch of relationships
        - match p=(:City{name:"Wuppertal"})-[:IN*]->(:World) return p
    - If you are doing a relationship, and you know that only one possible thing exists, then you can just leave things blank
        - match p=(:Admin2{name:"San Diego County"})-[:HAS_DEMOGRAPHIS]->(:Demographics) return p
        - match p=(:Admin2{name:"San Diego County"})-[:HAS_ECONOMICS]->() return p
            - For economics, we didn't have to specify the economics node
            - Additionally, using "p" is another tip as it is a pair and shows the entire path