KNOWLEDGE GRAPH:

- **NEO4J:**
  - **Import Data:**
    - CSV
    - API
  - **Graph Data Modeling**
  - **Uses cypher query language**
    - human-friendly, declarative query language which uses ASCII-Art to represent visual graph patterns for finding or updating data in Neo4j.
  - **Graph Visualization**
    - Neo4j offers methods for visualizing data, such as Neo4j Browser for developers, Neo4j Bloom for analysts and others looking for natural language search, and libraries for developers to embed graphs directly into their applications.
  - **Drivers for Popular Programming Languages**
    - Uses the binary "Bolt" protocol.
    - provides officially-supported drivers for languages such as .NET, Java (also Spring), JavaScript, and Python.

  - **IMPORT CSV: HOW DATA FROM CSV IS MAPPED TO GRPAH NODES:**
    - Using the python driver, we can connect to our neo4j database [browser] with the URI for a graph database service and the configuration and authentication details.
    - Once the session is loaded, we can import or query the graph database.
    - Once the csv is loaded it return the data in the form of dict with each csv's Node and key mapping.
    - For example:

Products.csv	**Orders.csv**	**Order-details.csv [DEFINING RELATIONSHIP BETWEEN PRODUCTS AND ORDERS]**
Productid,productname,unitcost 1,Chai,18 2,Chang,19	orderID,orderDate,shipCountry 10248,1996-07-04 00:00:00.000,France 10249,1996-07-05 00:00:00.000,Germany	orderID,productID,quantity 10248,1,12 10248,2,10
	QUERY:	
LOAD CSV FROM 'file:///products.csv' AS row WITH toInteger(row[0]) AS productId, row[1] AS productName, toFloat(row[2]) AS unitCost MERGE (p:Product {productId: productId}) SET p.productName = productName, p.unitCost = unitCost RETURN p	LOAD CSV WITH HEADERS FROM 'file:///orders.csv' AS row WITH toInteger(row.orderID) AS orderId, datetime(replace(row.orderDate,' ','T')) AS orderDate, row.shipCountry AS country MERGE (o:Order {orderId: orderId}) SET o.orderDateTime = orderDate, o.shipCountry = country RETURN o	LOAD CSV WITH HEADERS FROM 'file:///order-details.csv' AS row WITH toInteger(row.productID) AS productId, toInteger(row.orderID) AS orderId, toInteger(row.quantity) AS quantityOrdered MATCH (p:Product {productId: productId}) MATCH (o:Order {orderId: orderId}) MERGE (o)-[rel:CONTAINS {quantityOrdered: quantityOrdered}]->(p) RETURN rel
	Returns a neo4j.Record object	
	RETURN DATA FROM SERVER:	
• {'p': <Node id=0 labels={'Product'} properties={'unitCost': 18.0, 'productId': 1, 'productName': 'Chai'}>} • {'p': <Node id=1 labels={'Product'} properties={'unitCost': 19.0, 'productId': 2, 'productName': 'Chang'}>}	• {'o': <Node id=20 labels={'Order'} properties={'orderDateTime': neotime.DateTime(1996, 7, 4, 0, 0, 0.0, tzinfo=<UTC>), 'shipCountry': 'France', 'orderId': 10248}>} • {'o': <Node id=21 labels={'Order'} properties={'orderDateTime': neotime.DateTime(1996, 7, 5, 0, 0, 0.0,	• {'rel': <Relationship id=0 nodes=(<Node id=20 labels=set() properties={}>, <Node id=0 labels=set() properties={}>) type='CONTAINS' properties={'quantityOrdered': 12}>} • {'rel': <Relationship id=1 nodes=(<Node id=20 labels=set() properties={}>, <Node id=1 labels=set() properties={}>) type='CONTAINS' properties={'quantityOrdered': 10}>}

	tzinfo=<UTC>), 'shipCountry': 'Germany',
	'orderId': 10249}>}	

Where: • p – key of the record. • Node id • Label of the Node • And respective columns in CSV as its properties of the Node.	Where: • o–key of the record • Node id • Label of the Node • And respective columns in CSV as its properties of the Node.	Where: • rel-key of the record • Relationship id • Nodes- list of nodes in that relationship, ○ Inside each node's id, labels and properties. • Relationship type-name of the relationship • Properties of the relationship- Properties are name-value pairs that are used to add qualities to nodes and relationships.

MATCH QUERY [GRAPH]:

```
MATCH (o:Order)-[rel:CONTAINS]->(p:Product) RETURN p, rel, o
```

- {'rel': <Relationship id=0 nodes=(<Node id=20 labels={'Order'} properties={'orderDateTime': neotime.DateTime(1996, 7, 4, 0, 0, 0.0, tzinfo=<UTC>), 'shipCountry': 'France', 'orderId': 10248}>, <Node id=0 labels={'Product'} properties={'unitCost': 18.0, 'productId': 1, 'productName': 'Chai'}>) type='CONTAINS' properties={'quantityOrdered': 12}>, 'o': <Node id=20 labels={'Order'} properties={'orderDateTime': neotime.DateTime(1996, 7, 4, 0, 0, 0.0, tzinfo=<UTC>), 'shipCountry': 'France', 'orderId': 10248}>, 'p': <Node id=0 labels={'Product'} properties={'unitCost': 18.0, 'productId': 1, 'productName': 'Chai'}>}

- {'rel': <Relationship id=1 nodes=(<Node id=20 labels={'Order'} properties={'orderDateTime': neotime.DateTime(1996, 7, 4, 0, 0, 0.0, tzinfo=<UTC>), 'shipCountry': 'France', 'orderId': 10248}>, <Node id=1 labels={'Product'} properties={'unitCost': 19.0, 'productId': 2, 'productName': 'Chang'}>) type='CONTAINS' properties={'quantityOrdered': 10}>, 'o': <Node id=20 labels={'Order'} properties={'orderDateTime': neotime.DateTime(1996, 7, 4, 0, 0, 0.0, tzinfo=<UTC>), 'shipCountry': 'France', 'orderId': 10248}>, 'p': <Node id=1 labels={'Product'} properties={'unitCost': 19.0, 'productId': 2, 'productName': 'Chang'}>}

  - We can also get summary of our transaction as neo4j.BoltStatementResultSummary object.
  - The data from match query is being sent to data visualization tools.