Graph Database & neo4j

BY DIPESH RAFALIYA, MOUNIL SHAH, SHYAM KANTESARIYA

What is a graph database?

A graph database is simply a database that is built on top of a graph data structure. Like in a graph, graph databases can store nodes and edges between nodes. An edge also has a label that defines the relationship between two nodes. A graph database will support graph operations such as traversals and shortest path calculations. Because a lot of the data we are interested in can be represented by graphs, a graph database that is able to balance scalability as well as querying functionality is a very powerful tool. Graph databases are very good at representing data that has a lot of many-to-many relationships.

Property of Graph

- Each node/edge is uniquely identified
- Each node has a set of incoming and outgoing edges
- Each node/edge has a collection of properties
- Each edge has a label that defines the relationship between its two nodes

Importance of graphs

Graphs are useful for representing real world data. There are many useful operations and analyses that can be applied. Therefore it is pertinent that large graphs can be represented in a database

The current leading model is the relational model. While graph data can be stored efficiently in relational databases, many of the more powerful relational operations are either hard to implement, or execute inefficiently. Therefore, the concept of graph database was introduced to solve this problems.

- Modeling chemical and biological data
- Social networks
- The Web
- Hierarchical data

Compering with Relational DB

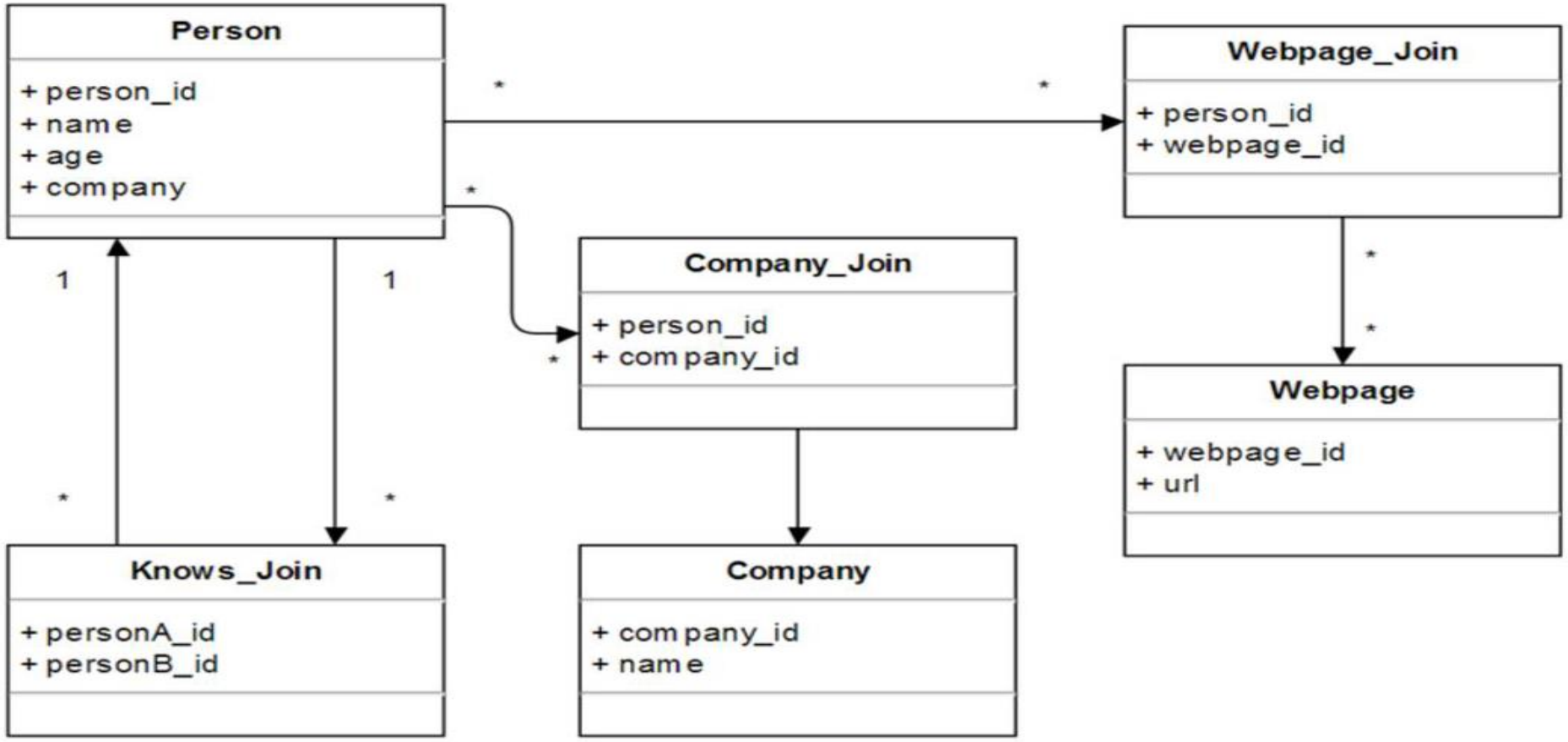
Pros

- Like other NoSQL implementations, graph databases exhibit the same schema flexibility which is a huge advantage given that current schemas are liable to change
- Most graph databases also have their own querying languages that are more intuitive for querying graphs
- Query times are also not affected by the total number of nodes Because graph databases do not have to do scans on tables, adding more nodes will in most cases not affect the execution time.

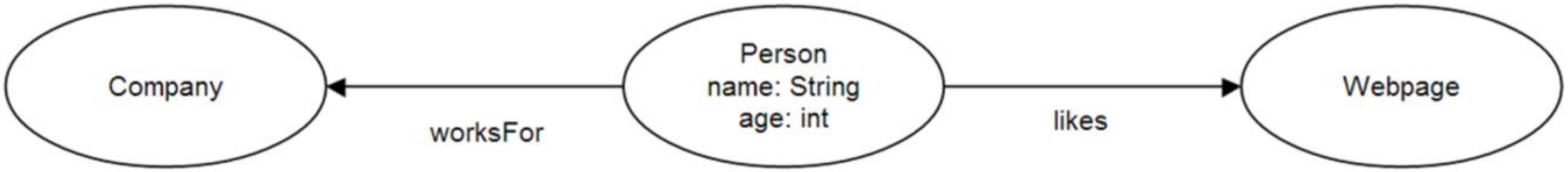
Cons

- However to allow arbitrary initial nodes, graph databases must maintain bi-directional relationships. Likewise with relational databases, certain graph databases are optimized for certain operations and must be tuned for your application.
- graph database querying languages are not unified, so migrating between graph databases will require more effort on the developer's part.

Relational Database Example



Graph database Example



Query Processing in Graph Databases

Graph Queries

- List nodes/edges that have this property
- List matching subgraphs
- Can these two node reach each other?
- How many hops does it take for two nodes to connect?

There are several questions We might want to ask of data stored in a graph. List for me all the nodes or edges that have a certain property. Find subgraphs that match a given set of relationships.

Neo4j

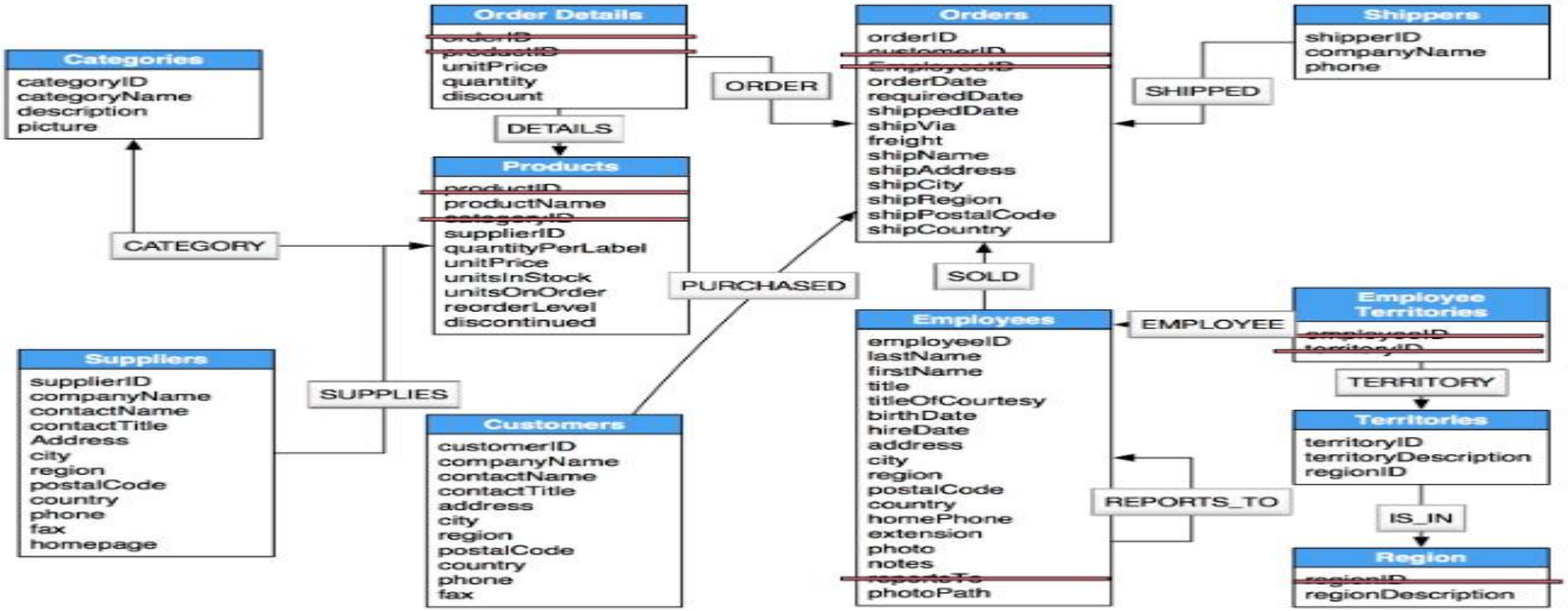
Most popular commercial graph database

Transactional queries

Optimized for single-machine use-case

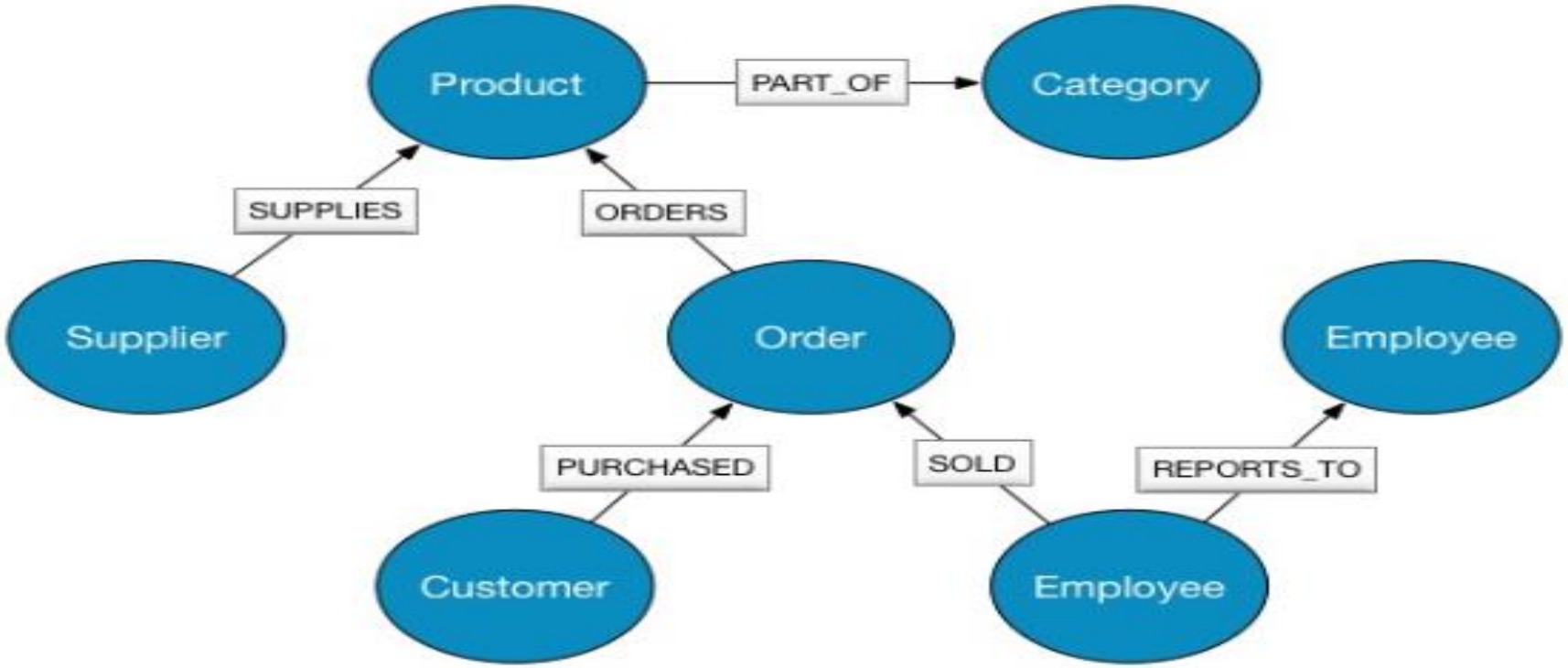
Cypher query language

Example in neo4J (Nortwind Dataset)

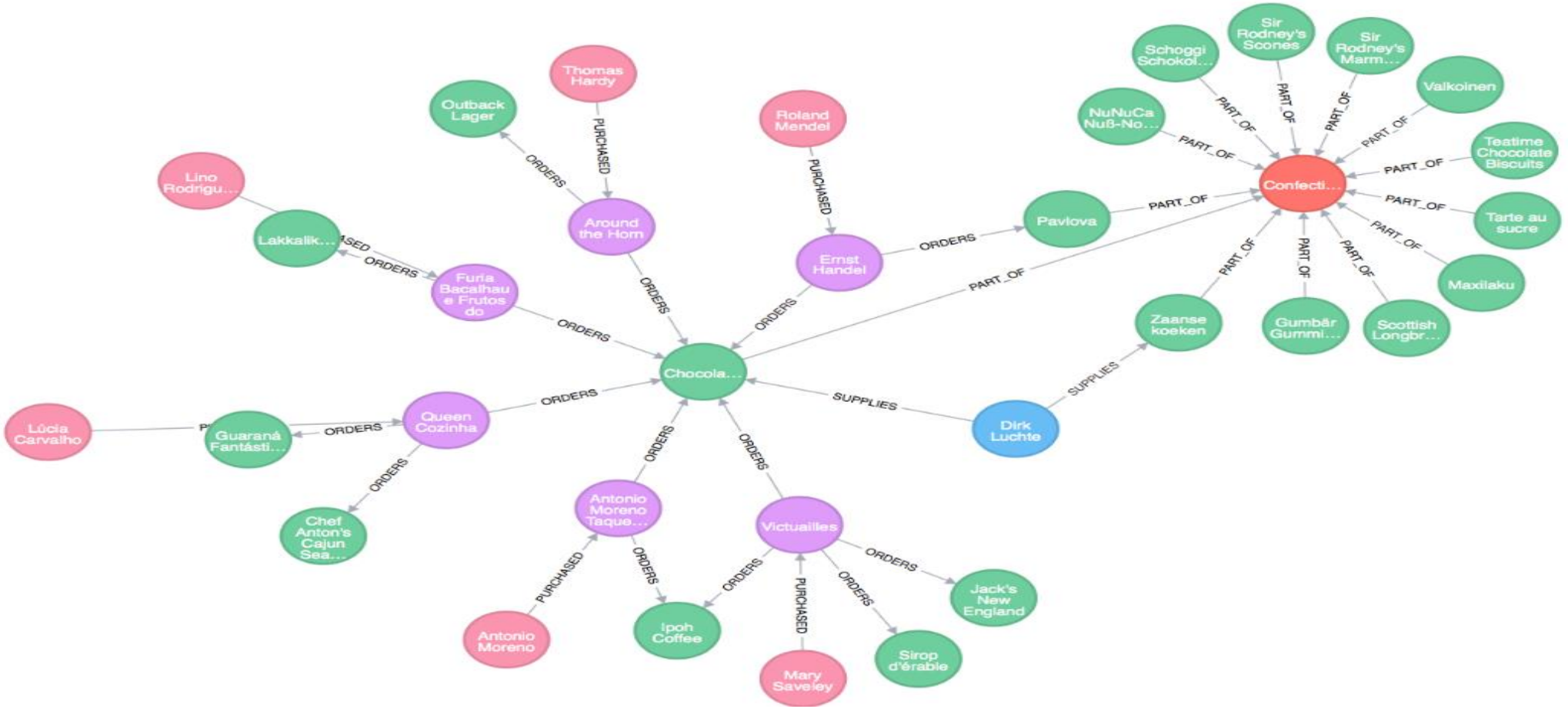


Simple JOIN Tables Become Relationships

The Northwind Graph Data Model



Some relationship of graph data



Relational quires and graph queries side by side

Find All Products

SQL	Cypher
SELECT p.* FROM products AS p;	MATCH (p:Product) RETURN p;

Field Access, Ordering, and Paging

SQL	Cypher
SELECT p.ProductName, p.UnitPrice	MATCH (p:Product)
FROM products AS p ORDER BY p.UnitPrice DESC LIMIT 10;	RETURN p.productName, p.unitPrice ORDER BY p.unitPrice DESC LIMIT 10;

Filter by Equality

SQL	Cypher
SELECT p.ProductName, p.UnitPrice FROM **products** AS p WHERE p.ProductName = 'Chocolade';	MATCH (p:Product) WHERE p.productName = "Chocolade" RETURN p.productName, p.unitPrice;

JOIN Records, DISTINCT Results

```
SELECT DISTINCT c.CompanyName
FROM customers AS c
JOIN orders AS o ON (c.CustomerID = o.CustomerID)
JOIN order_details AS od ON (o.OrderID = od.OrderID)
JOIN products AS p ON (od.ProductID = p.ProductID)
WHERE p.ProductName = 'Chocolade';
```

```
MATCH (:Product {productName:"Chocolade"})
    <-[:ORDERS]-(:Order)<-[:PURCHASED]-(c:Customer)
RETURN DISTINCT c.companyName AS Company;
```

Outer JOINs and Aggregation

```
SELECT p.ProductName, sum(od.UnitPrice * od.Quantity) AS Volume
FROM customers AS c
LEFT OUTER JOIN orders AS o ON (c.CustomerID = o.CustomerID)
LEFT OUTER JOIN order_details AS od ON (o.OrderID = od.OrderID)
LEFT OUTER JOIN products AS p ON (od.ProductID = p.ProductID)
WHERE c.CompanyName = 'Drachenblut Delikatessen'
GROUP BY p.ProductName
ORDER BY Volume DESC;
```

```
MATCH (c:Customer {companyName:"Drachenblut Delikatessen"})
OPTIONAL MATCH (p:Product)<-[pu:ORDERS]-(:Order)<-[:PURCHASED]-(c)
RETURN p.productName, sum(pu.unitPrice * pu.quantity) as volume
ORDER BY volume DESC;
```

Some interesting Queries

How are Employees Organized? Who Reports to Whom?

```
MATCH path = (e:Employee)<-[:REPORTS_TO]-(sub) RETURN e.employeeID AS manager, sub.employeeID AS employee;
```

Which Employees Report to Each Other Indirectly?

```
MATCH path = (e:Employee)<-[:REPORTS_TO*]-(sub) WITH e, sub, [person in NODES(path) | person.employeeID][1..-1] AS path RETURN e.employeeID AS manager, sub.employeeID AS employee, CASE WHEN LENGTH(path) = 0 THEN "Direct Report" ELSE path END AS via ORDER BY LENGTH(path);
```

How Many Orders were Made by Each Part of the Hierarchy?

```
MATCH (e:Employee) OPTIONAL MATCH (e)<-[:REPORTS_TO*0..]-(sub)-[:SOLD]->(order) RETURN e.employeeID, [x IN COLLECT(DISTINCT sub.employeeID) WHERE x <> e.employeeID] AS reports, COUNT(distinct order) AS totalOrders ORDER BY totalOrders DESC;
```