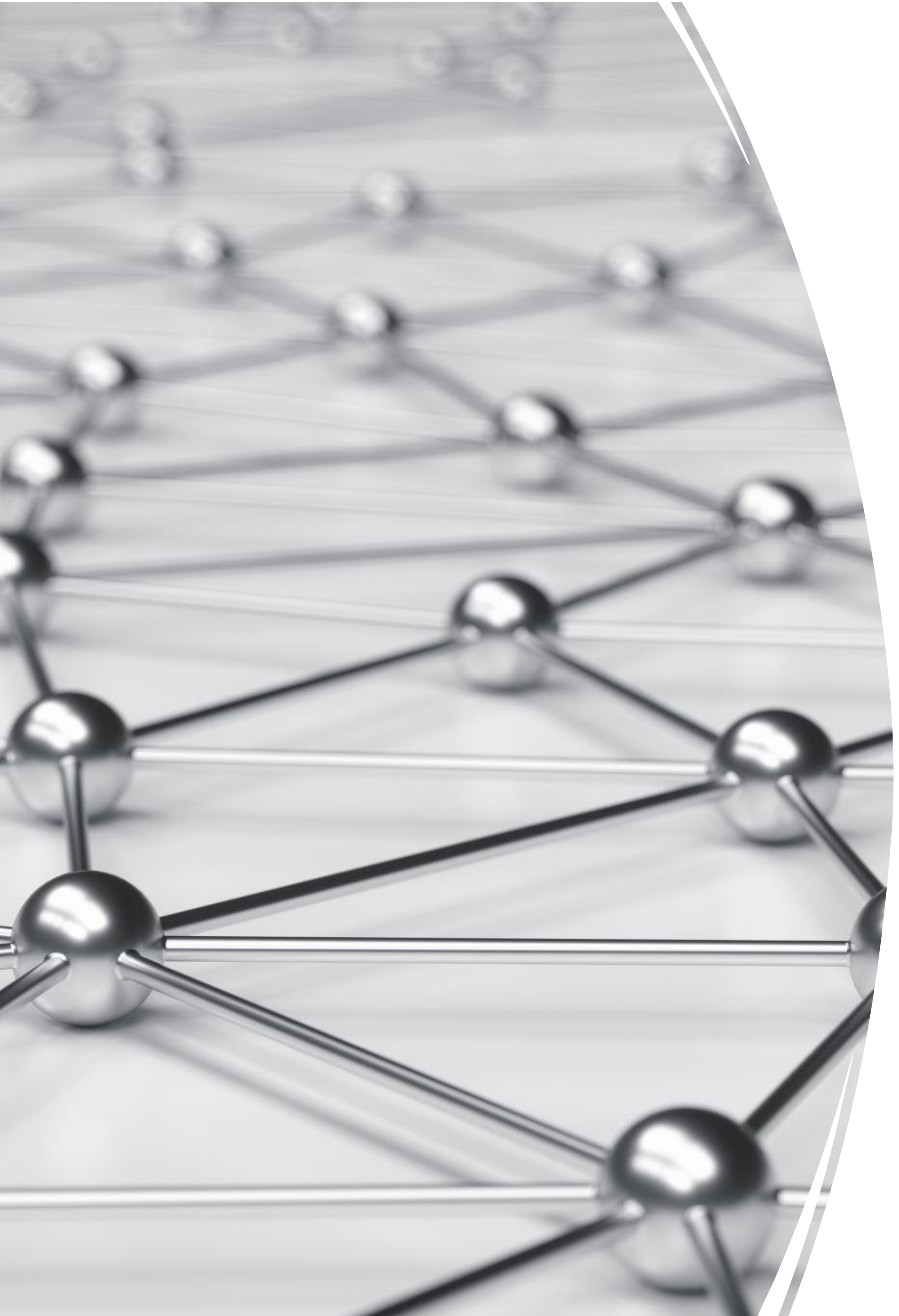


Unlocking the Power of Graph Databases with Neo4j

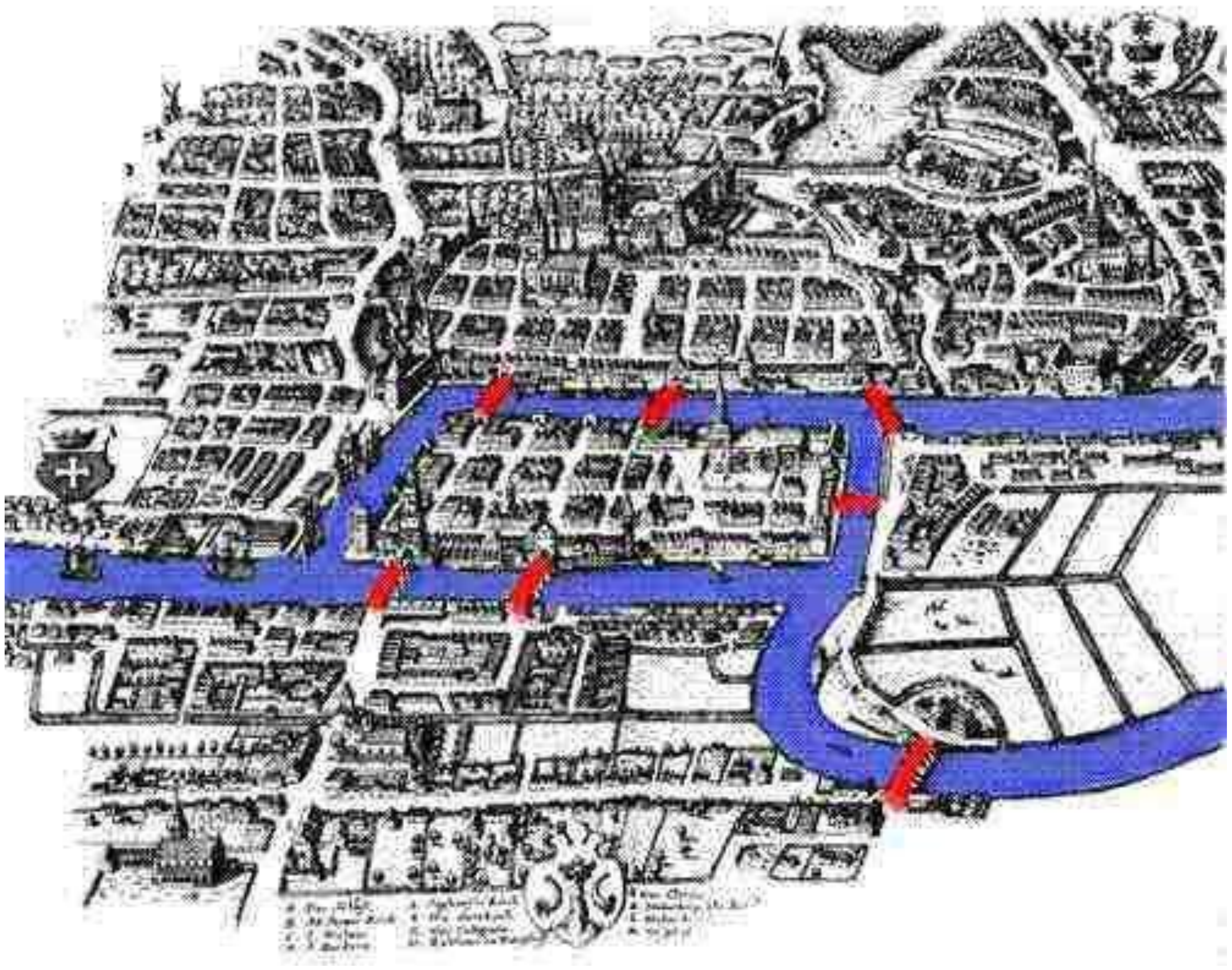
Ilya Verbitskiy

Agenda

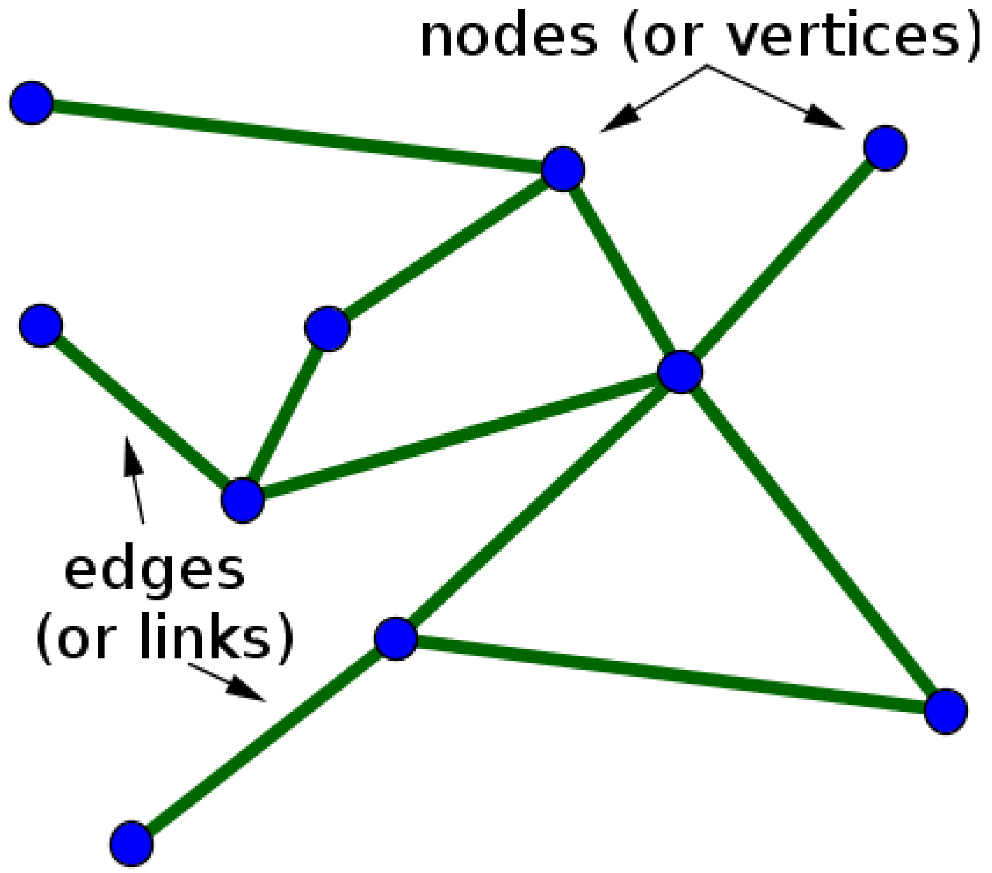
- Introduction to graph theory and Neo4j
- Data Modelling - Northwind
- Setting up the Northwind database on Neo4j AuraDB
- Introduction to Cypher

Königsberg, Prussia, 1736

- Is it possible for us to stroll through the city in such a way that we traverse each of the seven bridges just one time?



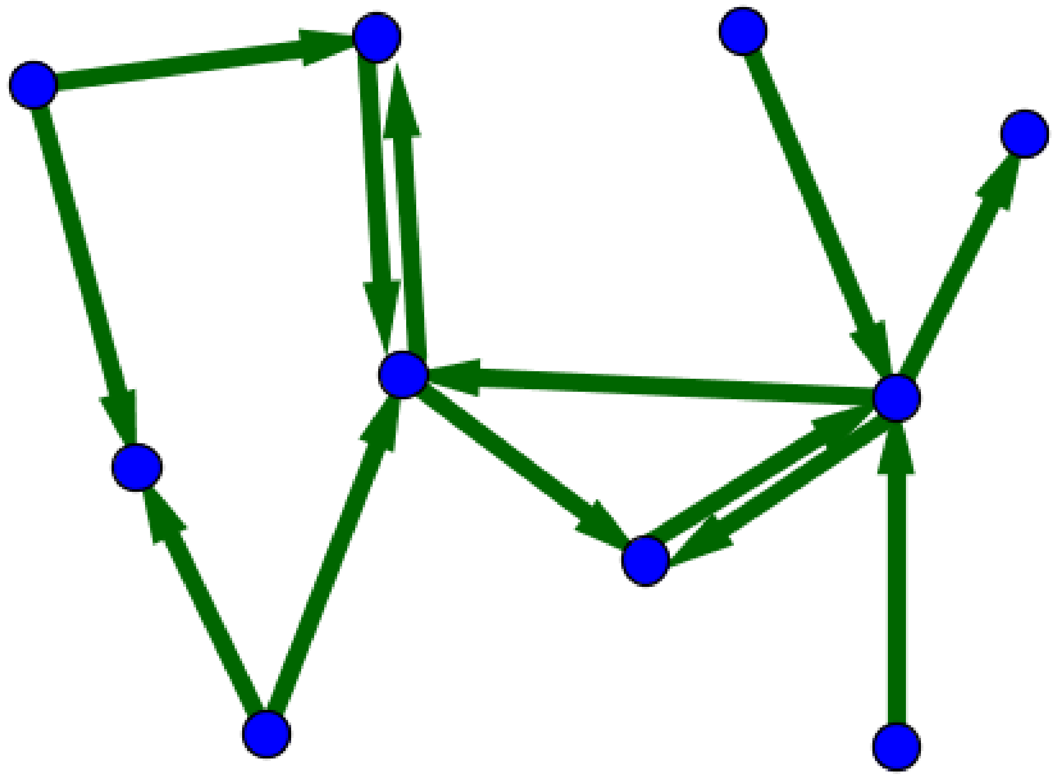
What is a graph



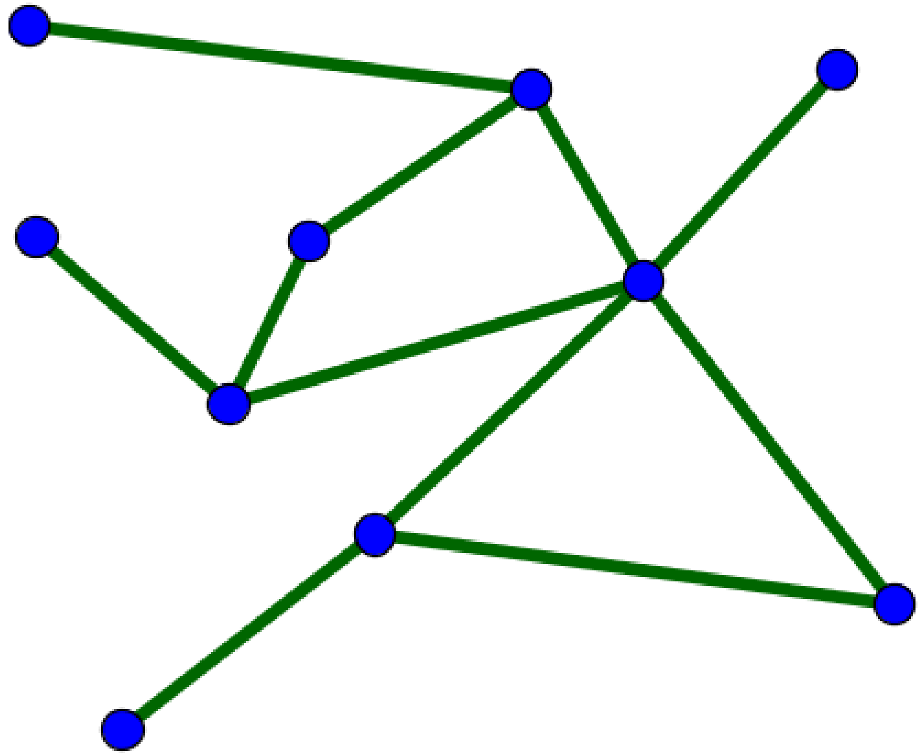
An introduction to networks by Duane Q. Nykamp.

Node relationships

A directed network



An undirected network



An introduction to networks by Duane Q. Nykamp.

Graphs around us

- Logistics
- Finance
- Network and IT operations
- E-Commerce
- Crime investigations
- Generative AI

Why Graph database

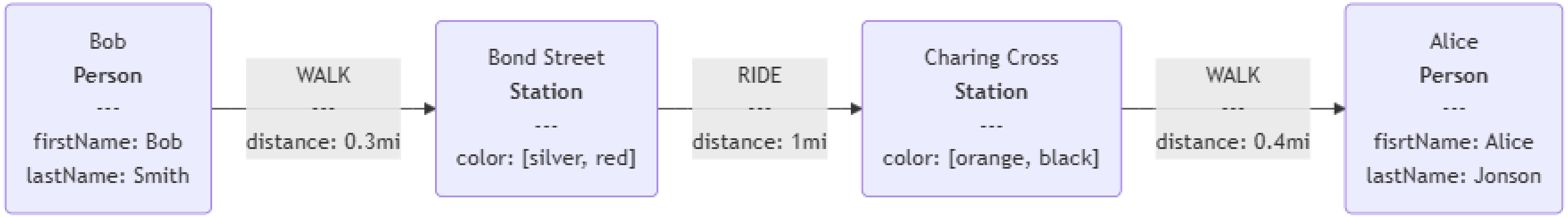
- Navigate deep hierarchies
- Find hidden connections between distant items
- Discover inter-relationships between items

What is Neo4j

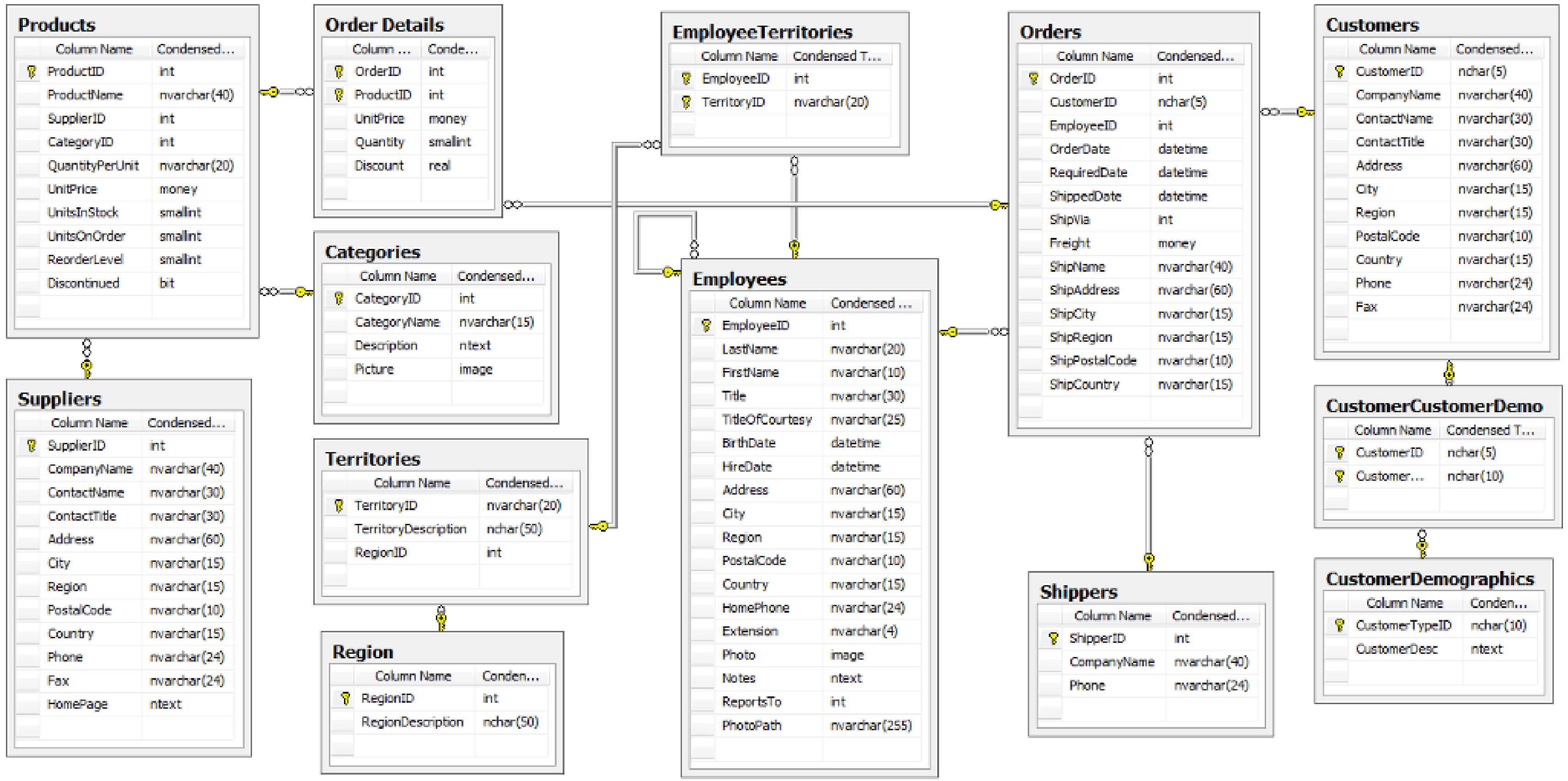
- Native graph database designed specifically with traversal in mind
- High performance read and write scalability
- ACID compliant
- Index-free adjacency (IFA)
- Cypher query language
- Open Source (GPLv3)
- Cloud enabled (AuraDB)

Neo4j data elements

- Nodes
- Relationships
- Labels
- Properties



Data Modeling – Northwind (SQL)



Products

	Column Name	Condensed...
🔑	ProductID	int
	ProductName	nvarchar(40)
	SupplierID	int
	CategoryID	int
	QuantityPerUnit	nvarchar(20)
	UnitPrice	money
	UnitsInStock	smallint
	UnitsOnOrder	smallint
	ReorderLevel	smallint
	Discontinued	bit

Order Details

	Column ...	Conde...
🔑	OrderID	int
🔑	ProductID	int
	UnitPrice	money
	Quantity	smallint
	Discount	real

EmployeeTerritories

	Column Name	Condensed T...
🔑	EmployeeID	int
🔑	TerritoryID	nvarchar(20)

Orders

	Column Name	Condensed...
🔑	OrderID	int
	CustomerID	nchar(5)
	EmployeeID	int
	OrderDate	datetime
	RequiredDate	datetime
	ShippedDate	datetime
	ShipVia	int
	Freight	money
	ShipName	nvarchar(40)
	ShipAddress	nvarchar(60)
	ShipCity	nvarchar(15)
	ShipRegion	nvarchar(15)
	ShipPostalCode	nvarchar(10)
	ShipCountry	nvarchar(15)

Customers

	Column Name	Condensed...
🔑	CustomerID	nchar(5)
	CompanyName	nvarchar(40)
	ContactName	nvarchar(30)
	ContactTitle	nvarchar(30)
	Address	nvarchar(60)
	City	nvarchar(15)
	Region	nvarchar(15)
	PostalCode	nvarchar(10)
	Country	nvarchar(15)
	Phone	nvarchar(24)
	Fax	nvarchar(24)

Categories

	Column Name	Condensed...
🔑	CategoryID	int
	CategoryName	nvarchar(15)
	Description	ntext
	Picture	image

Employees

	Column Name	Condensed ...
🔑	EmployeeID	int
	LastName	nvarchar(20)
	FirstName	nvarchar(10)
	Title	nvarchar(30)
	TitleOfCourtesy	nvarchar(25)
	BirthDate	datetime
	HireDate	datetime
	Address	nvarchar(60)
	City	nvarchar(15)
	Region	nvarchar(15)
	PostalCode	nvarchar(10)
	Country	nvarchar(15)
	HomePhone	nvarchar(24)
	Extension	nvarchar(4)
	Photo	image
	Notes	ntext
	ReportsTo	int
	PhotoPath	nvarchar(255)

Suppliers

	Column Name	Condensed...
🔑	SupplierID	int
	CompanyName	nvarchar(40)
	ContactName	nvarchar(30)
	ContactTitle	nvarchar(30)
	Address	nvarchar(60)
	City	nvarchar(15)
	Region	nvarchar(15)
	PostalCode	nvarchar(10)
	Country	nvarchar(15)
	Phone	nvarchar(24)
	Fax	nvarchar(24)
	HomePage	ntext

Territories

	Column Name	Condensed...
🔑	TerritoryID	nvarchar(20)
	TerritoryDescription	nchar(50)
	RegionID	int

CustomerCustomerDemo

	Column Name	Condensed T...
🔑	CustomerID	nchar(5)
🔑	Customer...	nchar(10)

Region

	Column Name	Conden...
🔑	RegionID	int
	RegionDescription	nchar(50)

Shippers

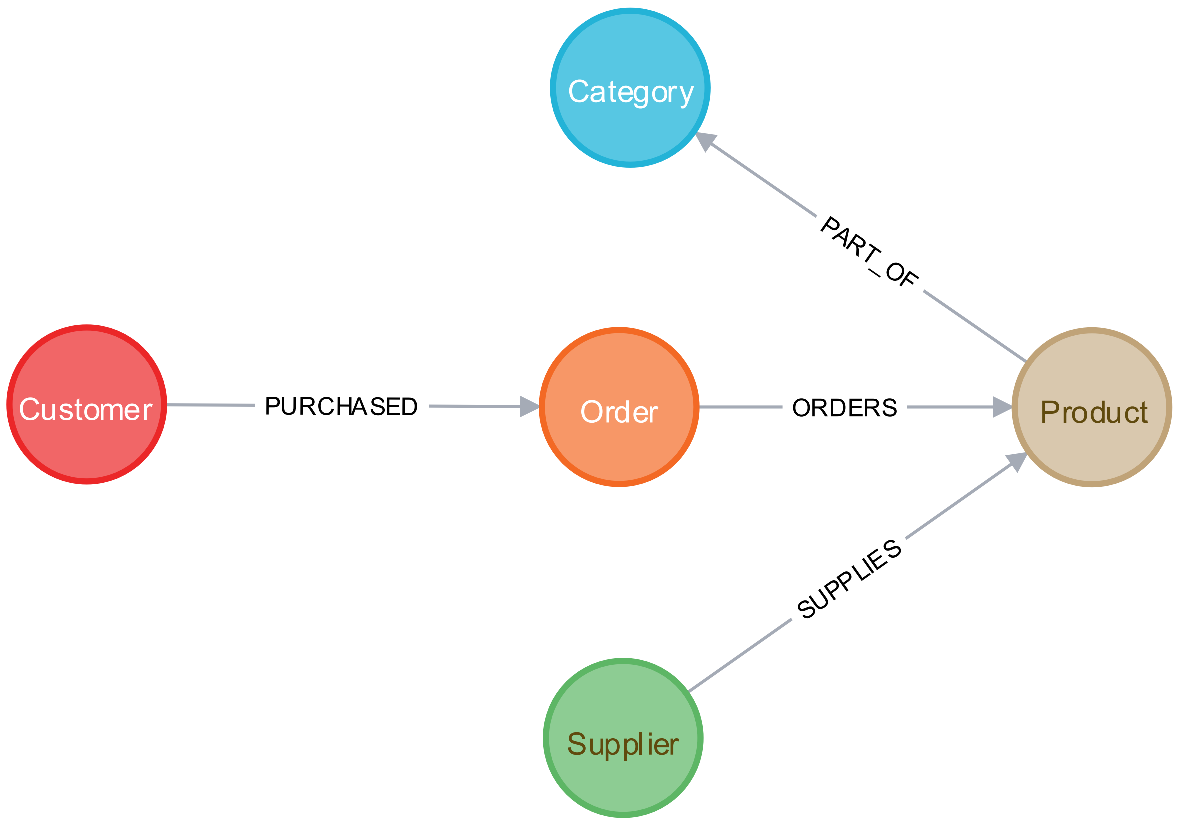
	Column Name	Condensed...
🔑	ShipperID	int
	CompanyName	nvarchar(40)
	Phone	nvarchar(24)

CustomerDemographics

	Column Name	Conden...
🔑	CustomerTypeID	nchar(10)
	CustomerDesc	ntext

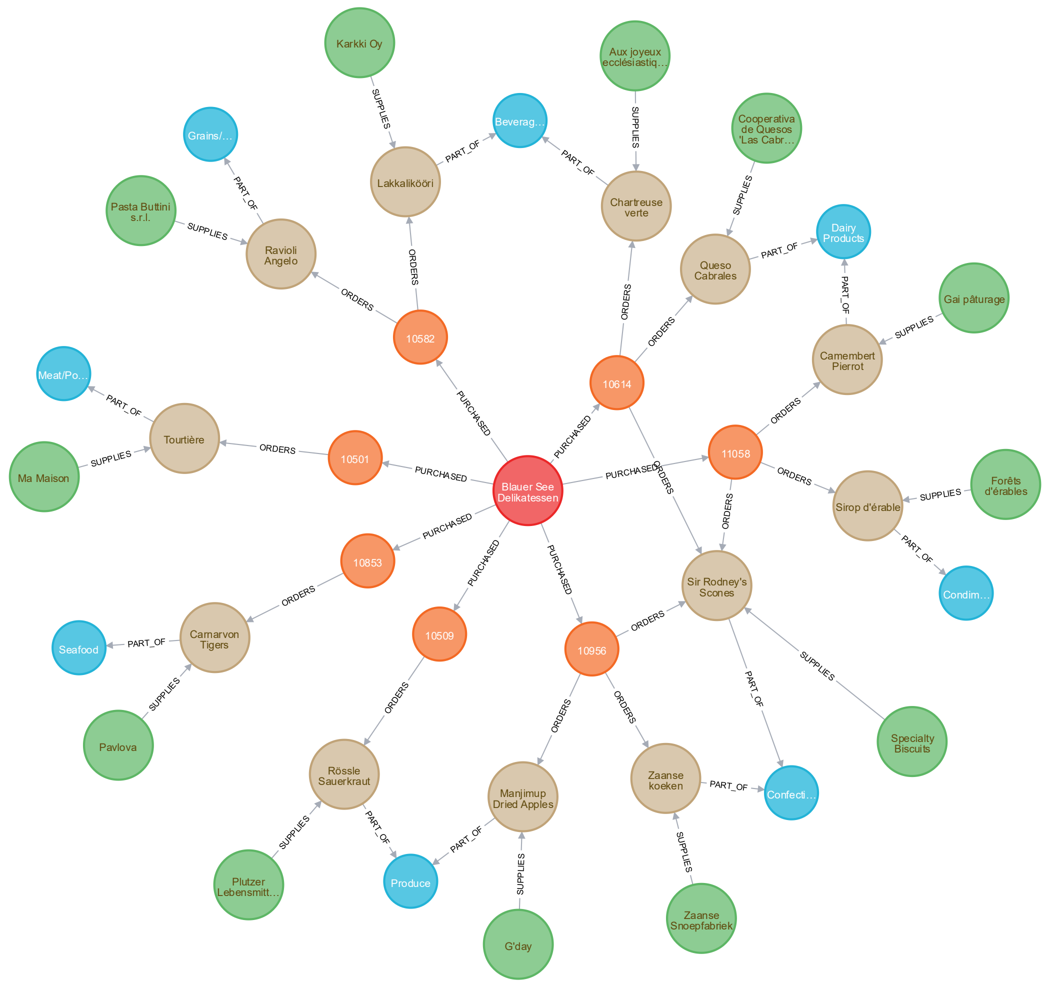
https://github.com/eirkostop/SQL-Northwind-exercises

Data Modeling – Northwind (Neo4j)



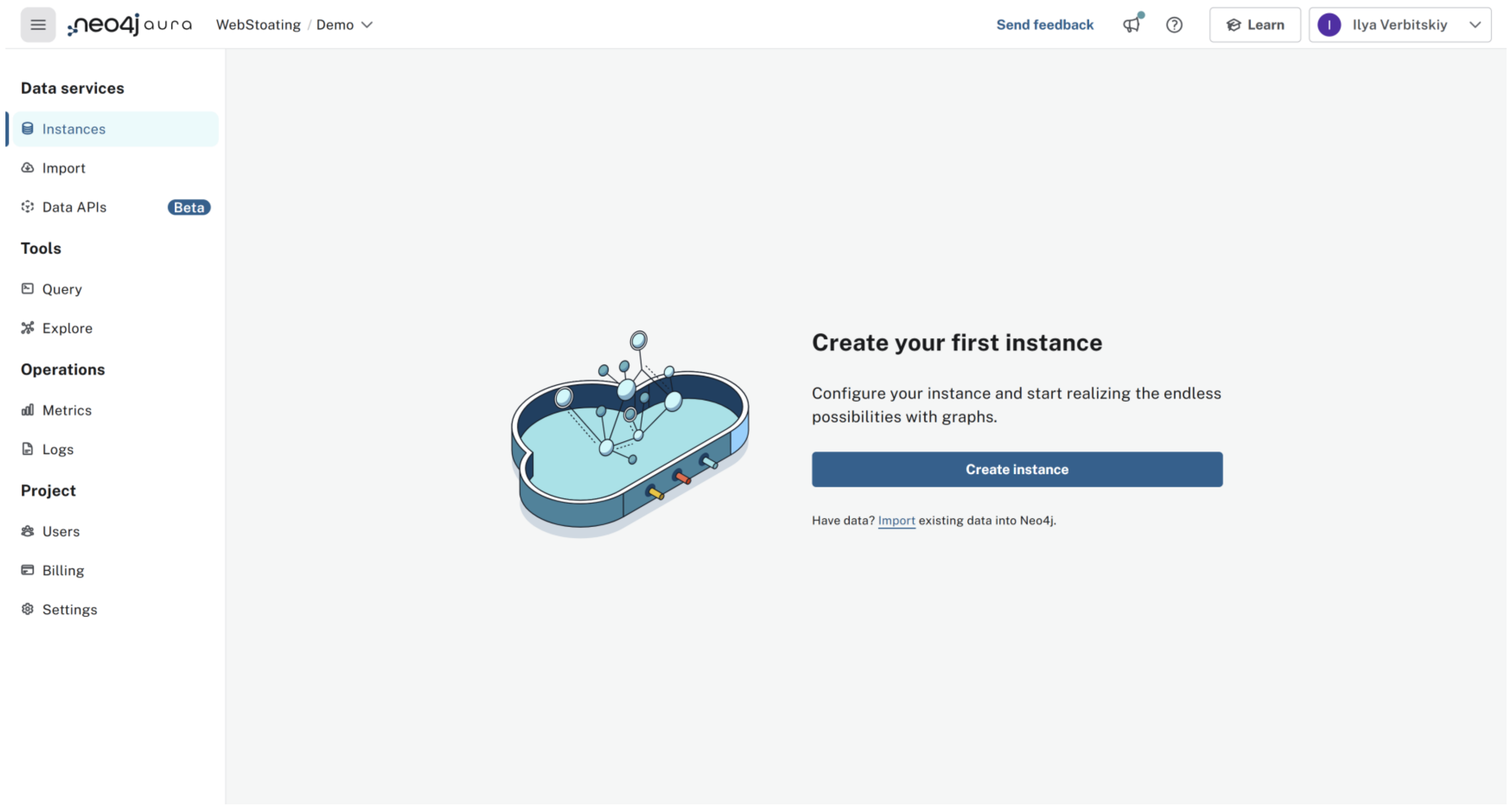
https://github.com/neo4j-graph-examples/northwind

Data Modelling – Northwind (Neo4j)



https://github.com/neo4j-graph-examples/northwind

Setting up Northwind database

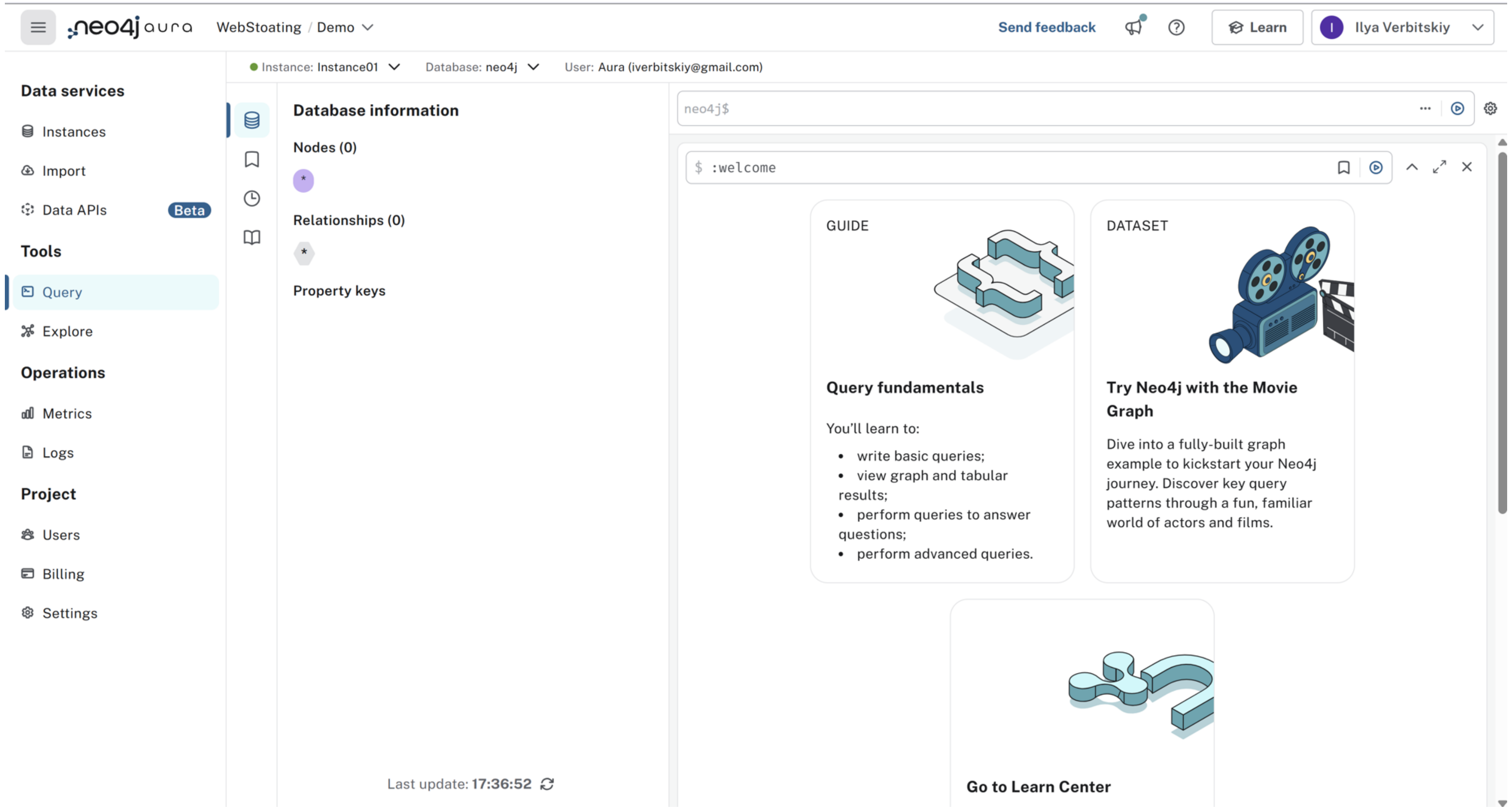


Create your first instance

Configure your instance and start realizing the endless possibilities with graphs.

Have data? Import existing data into Neo4j.

Setting up Northwind database

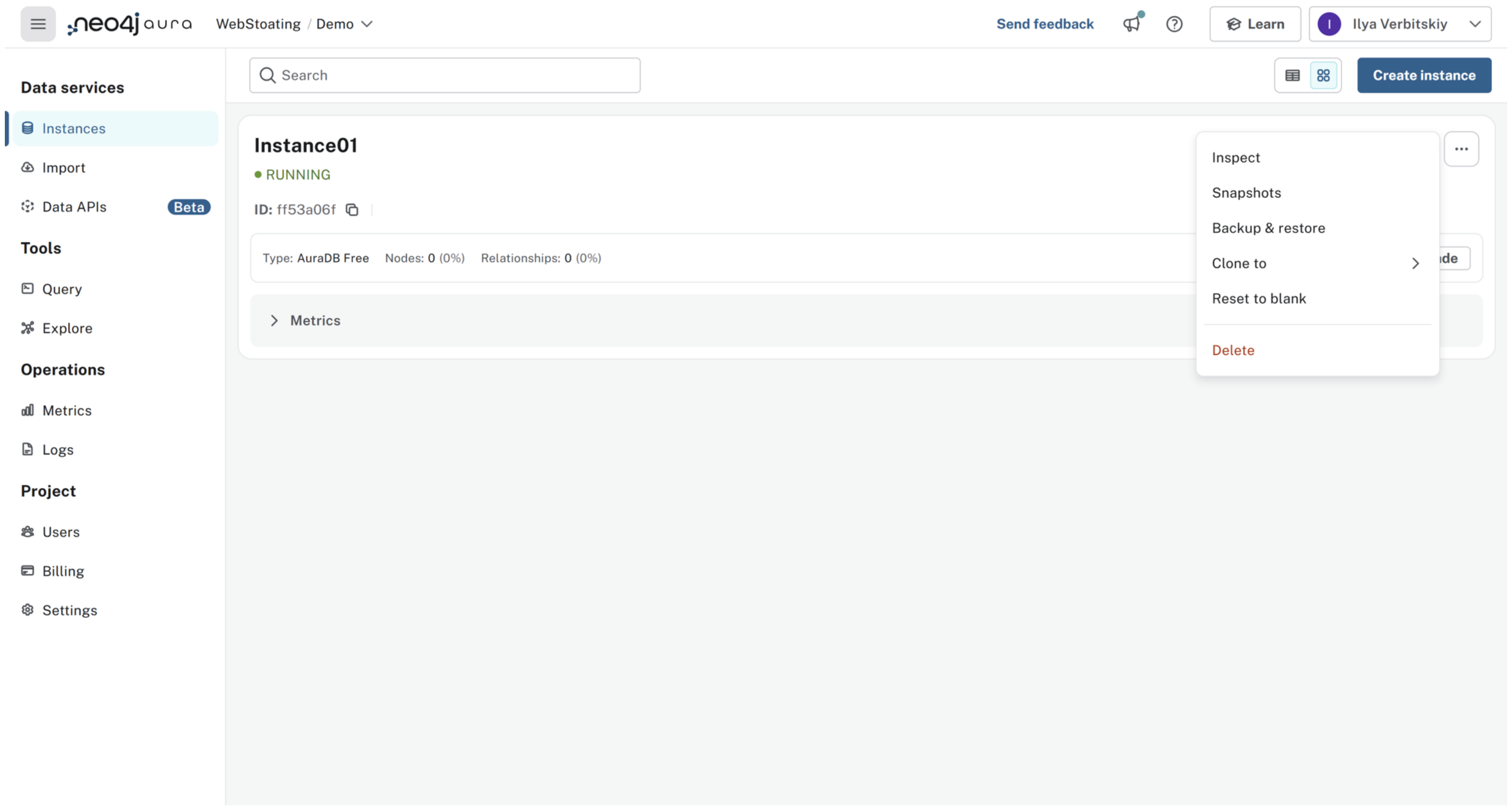


You'll learn to:

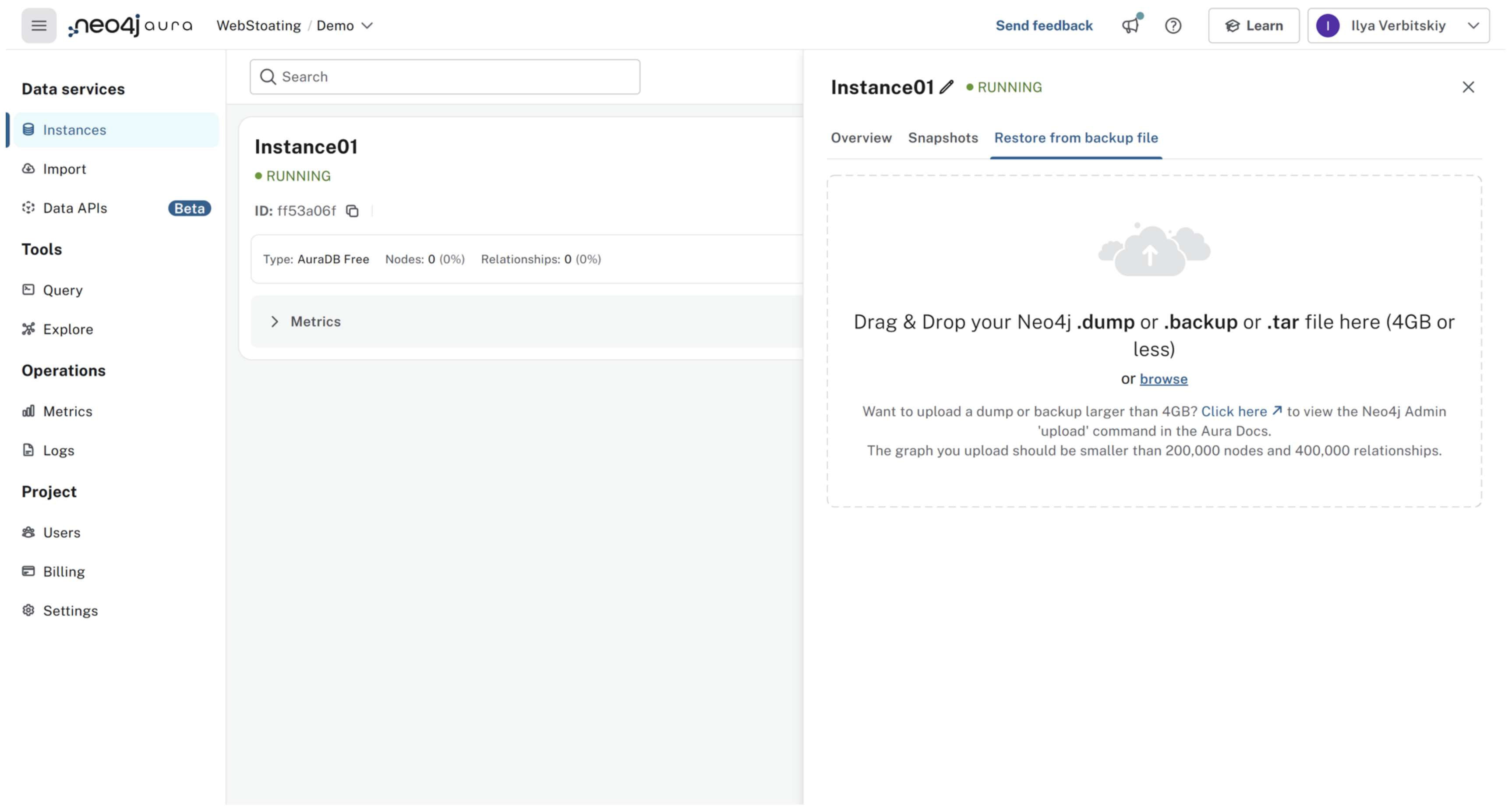
- write basic queries;
- view graph and tabular results;
- perform queries to answer questions;
- perform advanced queries.

Dive into a fully-built graph example to kickstart your Neo4j journey. Discover key query patterns through a fun, familiar world of actors and films.

Setting up Northwind database



Setting up Northwind database



Instance01
RUNNING
ID: ff53a06f

Type: AuraDB Free   Nodes: 0 (0%)   Relationships: 0 (0%)

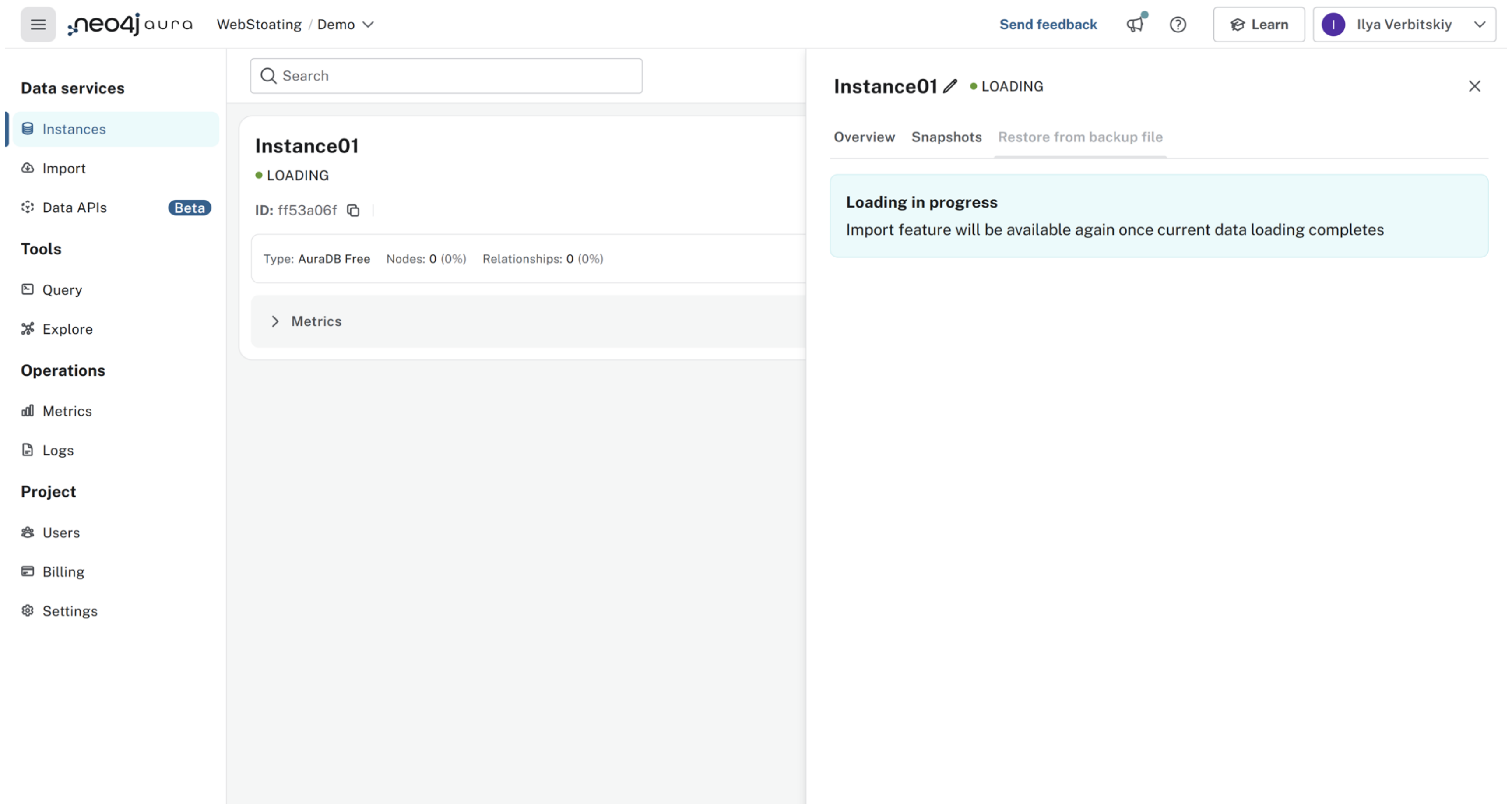
Instance01 • RUNNING

Drag & Drop your Neo4j **.dump** or **.backup** or **.tar** file here (4GB or less)

or browse

Want to upload a dump or backup larger than 4GB? Click here ↗ to view the Neo4j Admin 'upload' command in the Aura Docs.
The graph you upload should be smaller than 200,000 nodes and 400,000 relationships.

Setting up Northwind database

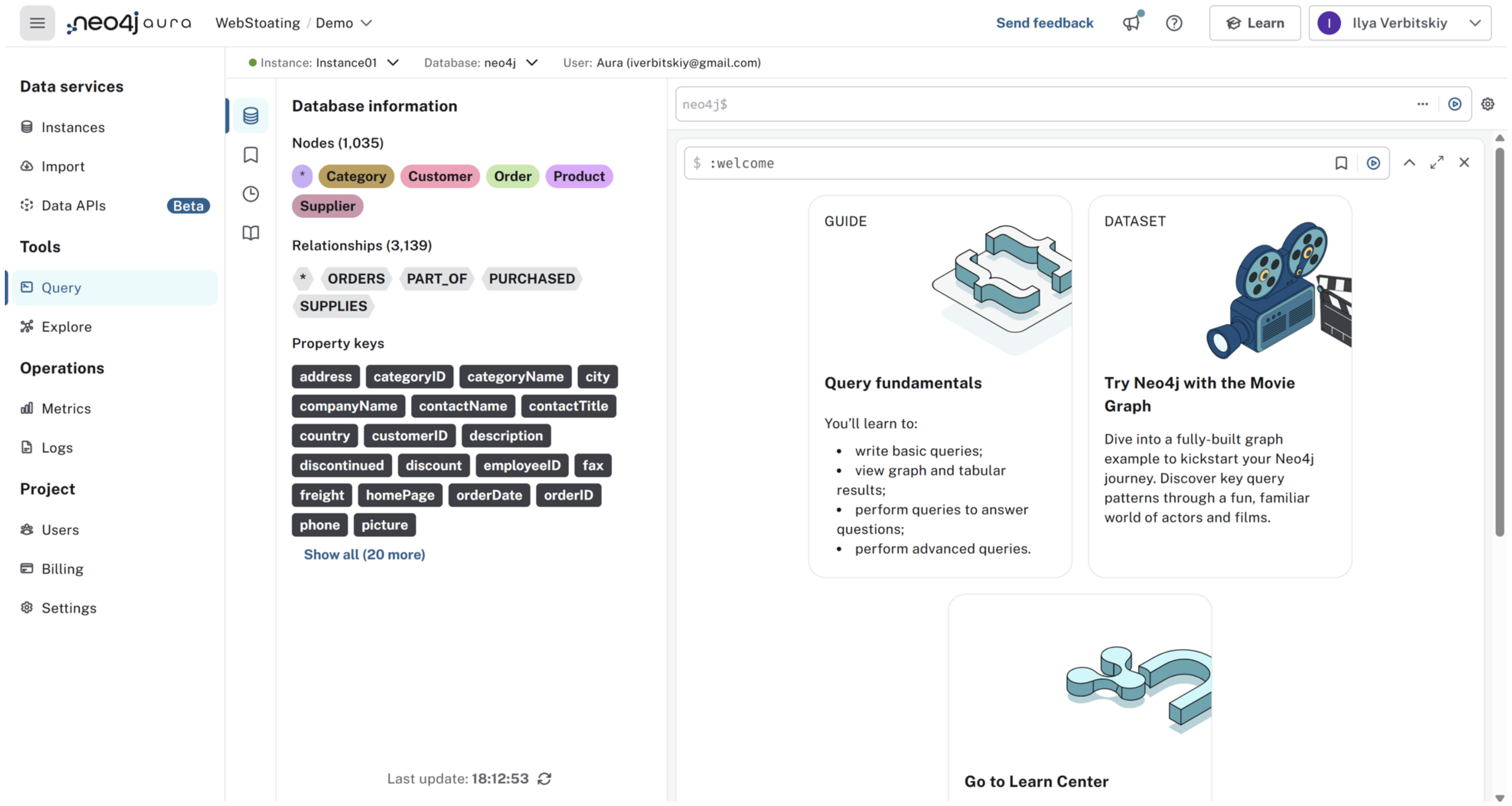


Instance01
LOADING
ID: ff53a06f
Type: AuraDB Free Nodes: 0 (0%) Relationships: 0 (0%)

Instance01 LOADING

Loading in progress
Import feature will be available again once current data loading completes

Setting up Northwind database



neo4j aura WebStoating / Demo

Send feedback Learn Ilya Verbitskiy

Instance: Instance01 Database: neo4j User: Aura (iverbitskiy@gmail.com)

Data services

- Instances
- Import
- Data APIs Beta

Tools

- Query
- Explore

Operations

- Metrics
- Logs

Project

- Users
- Billing
- Settings

Database information

Nodes (1,035)

* Category Customer Order Product Supplier

Relationships (3,139)

* ORDERS PART_OF PURCHASED SUPPLIES

Property keys

address categoryID categoryName city companyName contactName contactTitle country customerID description discontinued discount employeeID fax freight homePage orderDate orderID phone picture

Show all (20 more)

Last update: 18:12:53

```
neo4j$
```

```
$ :welcome
```

GUIDE

Query fundamentals

You'll learn to:

- write basic queries;
- view graph and tabular results;
- perform queries to answer questions;
- perform advanced queries.

DATASET

Try Neo4j with the Movie Graph

Dive into a fully-built graph example to kickstart your Neo4j journey. Discover key query patterns through a fun, familiar world of actors and films.

Go to Learn Center

Cypher basics

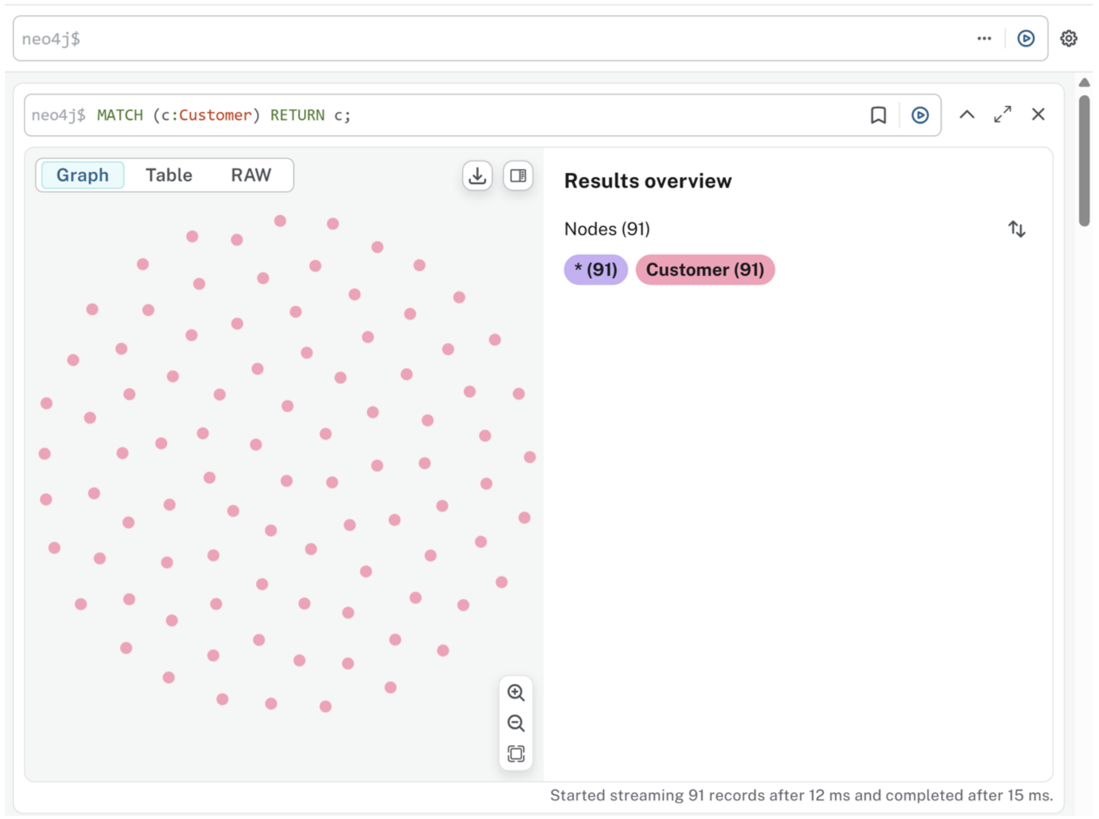
- **Nodes** are represented by parentheses ().
- **Labels** are specified using a colon :, e.g., (:Category).
- **Relationships** between nodes use two dashes --, e.g., (:Order)--(:Product).
- **Direction of relationships** is indicated with < or >, e.g., (:Order)-->(:Product).
- **Relationship types** are enclosed in square brackets [ ], e.g., [:ORDERS].
- **Properties** are key-value pairs, e.g., {employeeID: 4}.

SELECT #1

Get all columns from the Customers table.

SELECT * FROM Customers

MATCH (c:Customer) RETURN c



```
neo4j$
neo4j$ MATCH (c:Customer) RETURN c;
```

Results overview

Nodes (91)

* (91) Customer (91)

Started streaming 91 records after 12 ms and completed after 15 ms.

SELECT #2

Get the top 25 Customers alphabetically by Country and name.

SELECT TOP 25 *

FROM Customers

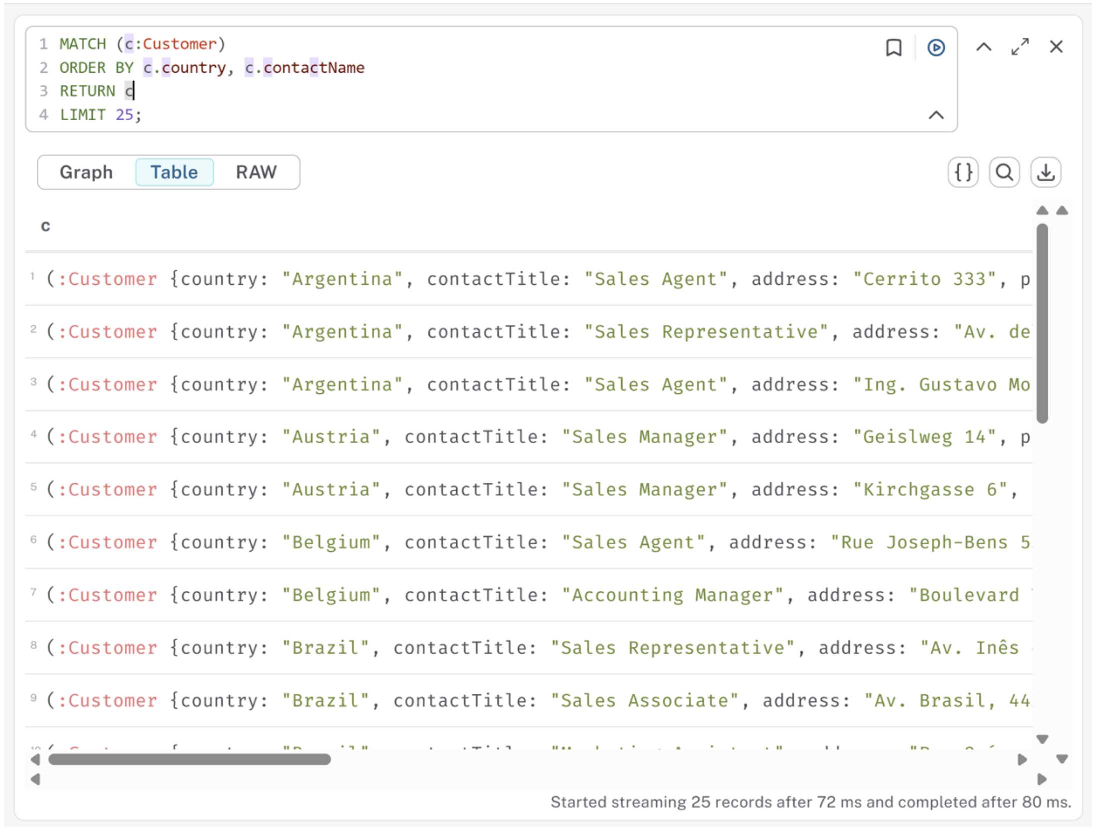
ORDER BY Country, ContactName

MATCH (c:Customer)

ORDER BY c.country, c.contactName

RETURN c

LIMIT 25;



```
MATCH (c:Customer)
ORDER BY c.country, c.contactName
RETURN c
LIMIT 25;
```

```
c
1 (:Customer {country: "Argentina", contactTitle: "Sales Agent", address: "Cerrito 333", p
2 (:Customer {country: "Argentina", contactTitle: "Sales Representative", address: "Av. de
3 (:Customer {country: "Argentina", contactTitle: "Sales Agent", address: "Ing. Gustavo Mo
4 (:Customer {country: "Austria", contactTitle: "Sales Manager", address: "Geislweg 14", p
5 (:Customer {country: "Austria", contactTitle: "Sales Manager", address: "Kirchgasse 6",
6 (:Customer {country: "Belgium", contactTitle: "Sales Agent", address: "Rue Joseph-Bens 5
7 (:Customer {country: "Belgium", contactTitle: "Accounting Manager", address: "Boulevard
8 (:Customer {country: "Brazil", contactTitle: "Sales Representative", address: "Av. Inês
9 (:Customer {country: "Brazil", contactTitle: "Sales Associate", address: "Av. Brasil, 44
```

Started streaming 25 records after 72 ms and completed after 80 ms.

SELECT #3

Get the count of all Orders made during 1997.

```
SELECT COUNT(*)
FROM Orders
WHERE YEAR(OrderDate) = 1997
```

```
MATCH (o:Order)
WHERE o.orderDate.year = 1997
RETURN COUNT(o);
```

SELECT #4

Get all orders placed on the 19th of May, 1997.

```
SELECT *
FROM Orders
WHERE OrderDate = '19970319'
```

```
MATCH (o:Order)
WHERE o.orderDate = date('1997-03-19')
RETURN o;
```

SELECT #5

Create a report for all the orders of 1996 and their Customers.

```
SELECT *
FROM Orders o
INNER JOIN Customers c ON o.CustomerID = c.CustomerID
WHERE YEAR(o.OrderDate) = 1996
```

```
MATCH p=(c:Customer)-[:PURCHASED]->(o:Order)
WHERE o.orderDate.year = 1996
RETURN p;
```

SELECT #6

Create a report for all 1996 orders and their Customers. Return only the Order ID, Order Date, Customer ID, Name, and Country.

```
SELECT o.OrderID, o.OrderDate, c.CustomerID, c.ContactName, c.Country
FROM Orders o
INNER JOIN Customers c ON o.CustomerID = c.CustomerID
WHERE YEAR(o.OrderDate) = 1996
```

```
MATCH (c:Customer)-[:PURCHASED]->(o:Order)
WHERE o.orderDate.year = 1996
RETURN o.orderID, o.orderDate, c.customerID, c.contactName, c.country;
```

SELECT #7

Create a report that shows the number of customers from each city.

```
SELECT c.City, COUNT(*)
FROM Orders o
INNER JOIN Customers c ON o.CustomerID = c.CustomerID
GROUP BY c.City
```

```
MATCH (c:Customer)-[:PURCHASED]->(o:Order)
RETURN c.city, COUNT(*);
```

SELECT #8

Create a report that shows the total quantity of products ordered. Only show records for products for which the quantity ordered is fewer than 200

```
SELECT p.ProductName, SUM(od.Quantity) as Quantity
FROM OrderDetails od
INNER JOIN Products p ON od.ProductID = p.ProductID
GROUP BY p.ProductName
HAVING SUM(od.Quantity) < 200
ORDER BY Quantity
```

SELECT #8

```
MATCH (o:Order)-[od:ORDERS]->(p:Product)
WITH p.productName as productName, SUM(od.quantity) as quantity
WHERE quantity < 200
ORDER BY quantity
RETURN productName, quantity;
```

WITH Clause

- WITH Clause allows chaining query parts, passing results to the next step.
- Scope Control: Only variables included in WITH are carried forward; others are discarded.
- Wildcard * can be used to include all currently scoped variables.
- Data Manipulation: WITH enables modifying output before passing it to the following query part.
- Flexible Results: You can reshape data or filter the number of entries in the result set.

Aggregating functions

1. **Counting and Grouping**
   - COUNT(*) – Counts all records.
   - COUNT(expression) – Counts non-null values of an expression.
   - COLLECT(expression) – Gathers values into a list.
2. **Statistical Aggregates**
   - AVG(expression) – Calculates the average of numeric values.
   - SUM(expression) – Computes the sum of numeric values.
   - MIN(expression) – Returns the minimum value.
   - MAX(expression) – Returns the maximum value.
   - STDEV(expression) – Computes the standard deviation.
   - STDEVP(expression) – Computes the population standard deviation.
   - PERCENTILE_CONT(expression, percentile) – Returns the continuous percentile.
   - PERCENTILE_DISC(expression, percentile) – Returns the discrete percentile.

SELECT #9

Create a report that shows the total number of orders by Customer since December 31, 1996. The report should only return rows for which the total number of orders is greater than 15

```
SELECT c.ContactName, COUNT(o.OrderID) as TotalOrders
FROM Orders o
INNER JOIN Customers c ON o.CustomerID = c.CustomerID
WHERE OrderDate > '1996-12-31'
GROUP BY c.ContactName
HAVING COUNT(o.OrderID) > 15
```

SELECT #9

```
MATCH (c:Customer)-[:PURCHASED]->(o:Order)
WHERE o.orderDate > date("1996-12-31")
WITH c.contactName as contactName, COUNT(o.orderID) as totalOrders
WHERE totalOrders > 15
RETURN contactName, totalOrders;
```

SELECT #10

Which UK Customers have paid more than 1000 dollars

```
SELECT c.ContactName, SUM(od.UnitPrice * od.Quantity * (1 - od.Discount)) as Paid
FROM Customers c
INNER JOIN Orders o ON c.CustomerID = o.CustomerID
INNER JOIN OrderDetails od ON o.OrderID = od.OrderID
WHERE c.Country = 'UK'
GROUP BY c.ContactName
HAVING SUM(od.UnitPrice * od.Quantity * (1 - od.Discount)) > 1000
```

SELECT #10

```
MATCH (c:Customer)-[:PURCHASED]->(o:Order)-[od:ORDERS]->(p:Product)
WHERE c.country = 'UK’
WITH c.contactName as contactName, SUM(toFloat(od.unitPrice) * od.quantity * (1 -
toFLoat(od.discount))) as paid
WHERE paid > 1000
RETURN contactName, paid;
```

INSERT/UPSERT #1

Insert yourself into the Customers table Include the following fields: CustomerID, CompanyName, ContactName, ContactTitle, Address, City, Region, PostalCode, Country, Phone, Fax

```
INSERT INTO Customers (CustomerID, CompanyName, ContactName, ContactTitle,
Address, City, Region, PostalCode, Country, Phone, Fax)

VALUES

('ILYA1', 'Acme Corp', 'Ilya Verbitskiy', 'Manager', '123 Main St', 'New York', 'NY', '10001',
'USA', '555-1234', '555-5678')
```

INSERT/UPSERT #1

```
CREATE (c:Customer {
        customerID: 'ILYA1',
        companyName: 'Acme Corp',
        contactName: 'Ilya Verbitskiy',
        contactTitle: 'Manager',
        address: '123 Main St',
        city: 'New York',
        region: 'NY',
        postalCode: '10001',
        country: 'USA',
        phone: '555-1234',
        fax: '555-5678’
});
```

INSERT/UPSERT #1

```
MERGE (c:Customer {customerID: 'ILYA1'})
SET c.companyName = 'Acme Corp',
        c.contactName = 'Ilya Verbitskiy',
        c.contactTitle = 'Manager',
        c.address = '123 Main St',
        c.city = 'New York',
        c.region = 'NY',
        c.postalCode = '10001',
        c.country = 'USA',
        c.phone = '555-1234',
        c.fax = '555-5678’;
```

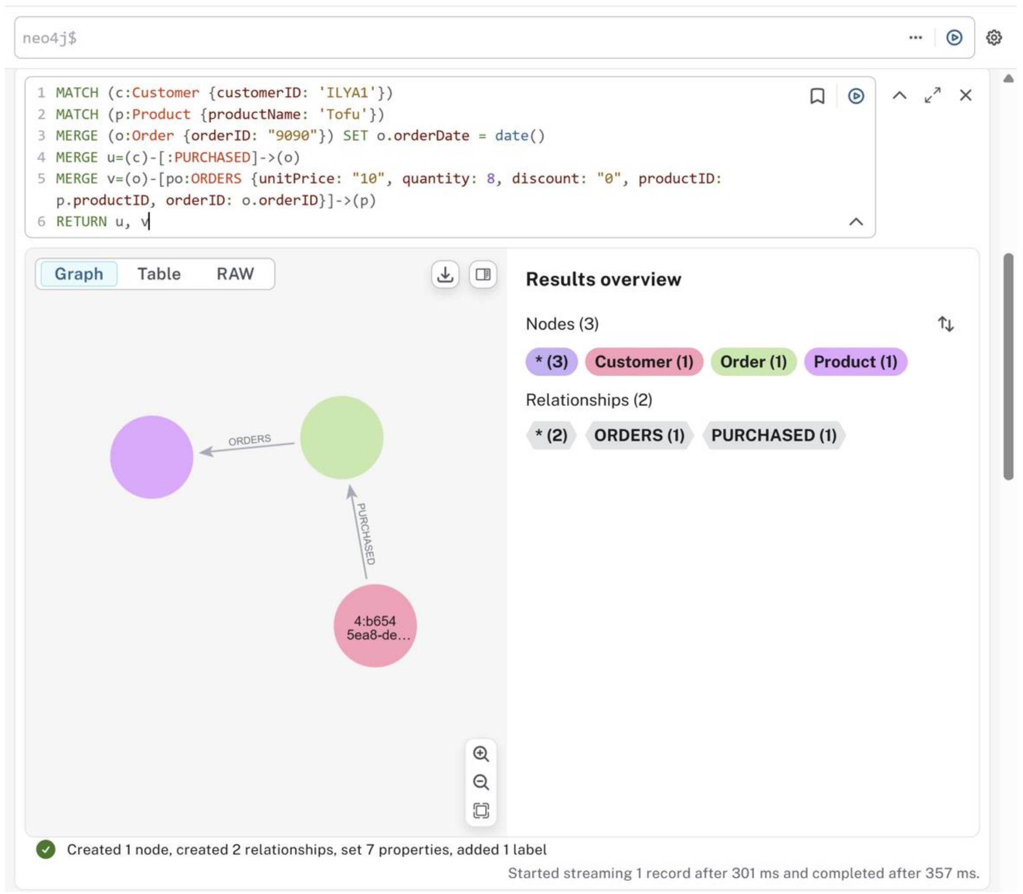
INSERT/UPSERT #2

```
BEGIN TRANSACTION

INSERT Orders(CustomerID, EmployeeID, OrderDate)
VALUES ('ILYA1', 5, GETDATE())

DECLARE @LastOrderID INT = SCOPE_IDENTITY()
DECLARE @ProductId INT
SELECT @ProductId = ProductID FROM Products WHERE ProductName = 'Tofu'

INSERT OrderDetails(OrderID, ProductID, UnitPrice, Quantity, Discount)
VALUES (@LastOrderID, @ProductId, 10, 8, 0)

COMMIT
```

INSERT/UPSERT #2

```
MATCH (c:Customer {customerID: 'ILYA1’})
MATCH (p:Product {productName: 'Tofu’})
MERGE (o:Order {orderID: "9090"}) SET o.orderDate = date()
MERGE u=(c)-[:PURCHASED]->(o)
MERGE v=(o)-[po:ORDERS {unitPrice: "10", quantity: 8, discount: "0", productID:
p.productID, orderID: o.orderID}]->(p)
RETURN u, v;
```

INSERT/UPSERT #2



neo4j$

```
MATCH (c:Customer {customerID: 'ILYA1'})
MATCH (p:Product {productName: 'Tofu'})
MERGE (o:Order {orderID: "9090"}) SET o.orderDate = date()
MERGE u=(c)-[:PURCHASED]->(o)
MERGE v=(o)-[po:ORDERS {unitPrice: "10", quantity: 8, discount: "0", productID:
p.productID, orderID: o.orderID}]->(p)
RETURN u, v
```

Results overview

Nodes (3)

* (3) Customer (1) Order (1) Product (1)

Relationships (2)

* (2) ORDERS (1) PURCHASED (1)

Created 1 node, created 2 relationships, set 7 properties, added 1 label

Started streaming 1 record after 301 ms and completed after 357 ms.

UPDATE #1

Change Order.orderDate data type from String to Date

```
MERGE (o:Order)
SET o.orderDate = date(substring(o.orderDate, 0, 10));
```

UPDATE #2

Update the phone number.

```
UPDATE Customers SET Phone = '000-4321' WHERE CustomerID = 'ILYA1'
```

```
MERGE (c:Customer {customerID: 'ILYA1'})
SET c.phone = '000-4321';
```

UPDATE #3

Double the quantity of the order details record you inserted before

```
UPDATE od
SET Quantity = od.Quantity * 2
FROM OrderDetails od
INNER JOIN Orders o ON od.OrderID = o.OrderID
WHERE o.OrderID = 11084
```

```
MATCH (o:Order {orderID: "9090"})-[od:ORDERS]->(p:Product {productID: "14"})
SET od.quantity = od.quantity * 2
RETURN o, od, p;
```

DELETE #1

Delete the records you inserted before. Don't delete any other records!

DELETE #1

```
BEGIN TRANSACTION
DELETE od
FROM OrderDetails od
INNER JOIN dbo.Orders O on O.OrderID = od.OrderID
INNER JOIN Customers c ON o.CustomerID = c.CustomerID
WHERE c.CustomerID = 'ILYA1'

DELETE o
FROM Orders o INNER JOIN Customers c ON o.CustomerID = c.CustomerID
WHERE c.CustomerID = 'ILYA1'

DELETE Customers WHERE CustomerID = 'ILYA1'
COMMIT
```

DELETE #1

```
MATCH (c:Customer {customerID: "ILYA1"})-[:PURCHASED]->(o:Order)-[:ORDERS]-
>(p:Product)
DETACH DELETE c, o;
```

Let’s keep in touch!