Neo4J: Introduction to Advanced Basics

Presented by: Muhammad Usman, Muhammad Ahmad, Abdul Samad, Gohar Hussain, Ahmed Raza

Table of contents

01

Recap

Revisiting Previous Lecture Highlights

What is Neo4j?

- Neo4J is a graph database that was founded by Emil Eifrem in 2000, and it officially launched in 2007.

- It stores data in a unique way compared to traditional databases.

- Instead of relying on tables, it organizes information as a network of nodes and relationships.

Language

- Neo4J uses a query language called Cypher. This language is specifically designed for interacting with graph data.

- It allows users to query, manipulate, and analyze data stored in Neo4J's graph database.

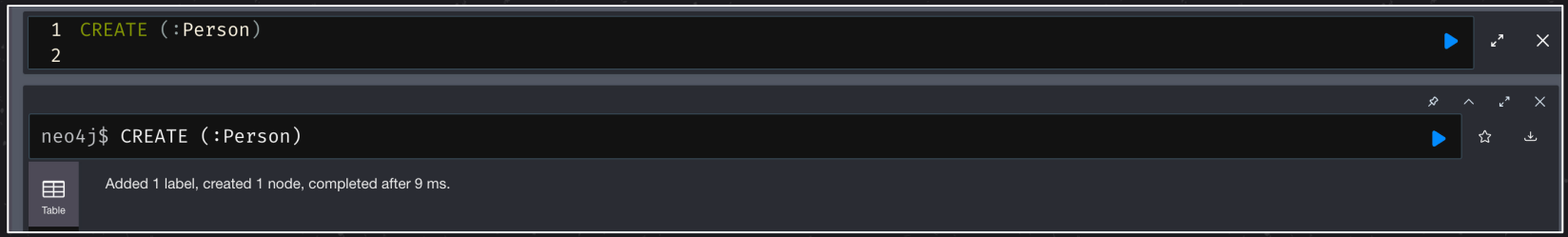
- Cypher provides an intuitive syntax for expressing graph patterns and operations, making it easier for developers and data analysts to work with complex interconnected data structures.

Graph Database vs Traditional Database

Aspect	Graph Database	Traditional Database
Data Organization	Organizes data as nodes and relationships	Organizes data in tables and rows
Representation	Uses a network of dots and lines	Uses tables with rows and columns
Connectivity	Emphasizes relationships between data	Focuses on individual data points
Flexibility	Highly flexible in handling interconnected data	More rigid structure with predefined schemas
Traversal	Efficient traversal of complex relationships	Limited traversal beyond predefined joins
Schema	Schema-less or flexible schema	Requires strict schema definition
Query Language	Uses Cypher, an intuitive graph query language	Typically uses SQL for querying
Use Cases	Ideal for scenarios with complex relationships	Suited for structured data with clear relationships
Example	Social networks, recommendation systems	E-commerce, banking systems

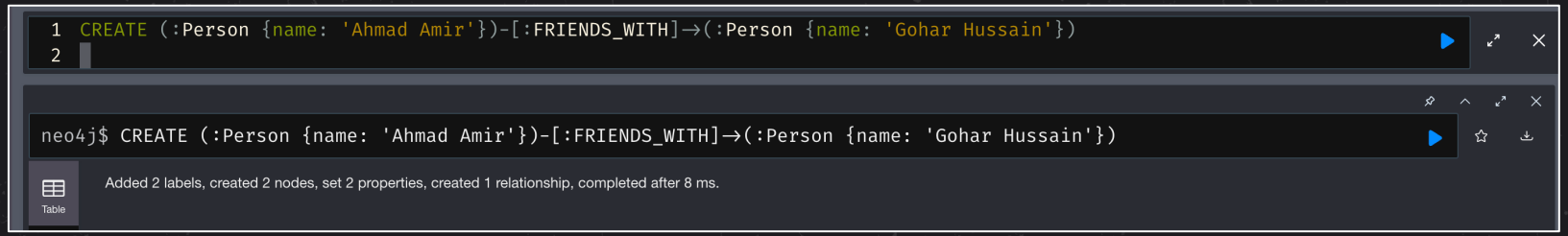
Creating a Node

- To create a single node in Cypher for Neo4J, you simply use the CREATE keyword followed by parentheses containing optional labels and properties for the node.



Creating a Relationship

- To create a relationship between two nodes in Cypher for Neo4J, you use the CREATE keyword followed by the relationship pattern and the nodes involved in the relationship.



02

Advanced Basics

Approaching the Advanced Basics of Neo4J

Adding Multiple Nodes

- Adding multiple nodes creates individual entities in the graph database

- These nodes can have properties associated with them. By creating multiple nodes, you're essentially populating your graph database with data about different entities

- Instead of relying on tables, it organizes information as a network of nodes and relationships.

Cypher of Adding Multiple Nodes

```
// Creating 10 nodes representing friends
CREATE (:Person {name: 'Friend1'})
CREATE (:Person {name: 'Friend2'})
CREATE (:Person {name: 'Friend3'})
CREATE (:Person {name: 'Friend4'})
CREATE (:Person {name: 'Friend5'})
CREATE (:Person {name: 'Friend6'})
CREATE (:Person {name: 'Friend7'})
CREATE (:Person {name: 'Friend8'})
CREATE (:Person {name: 'Friend9'})
CREATE (:Person {name: 'Friend10'})
```

Making Multiple Relationships

- Creating multiple relationships between nodes in a graph database allows for the establishment of multiple connections or associations between entities.

- Each relationship represents a specific interaction, connection, or attribute shared between the connected nodes.

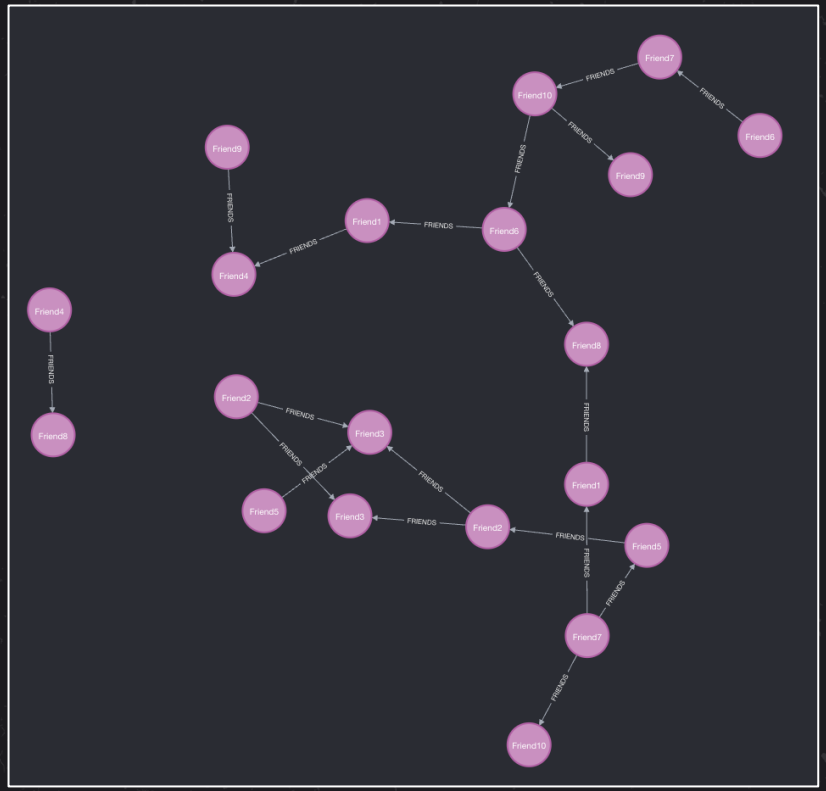
- With multiple relationships, you have the flexibility to capture various aspects of the relationships between nodes. For example, in a social network, you could have relationships representing friendships, professional connections, family ties, etc.

Cypher of Creating Multiple Relationships

```
// Creating relationships between friends
MATCH (a:Person), (b:Person)
WHERE a.name <> b.name
WITH a, b, rand() AS r
ORDER BY r
LIMIT 15 // Adjust this number to create more or fewer relationships
CREATE (a)-[:FRIENDS]→(b)

// Create a specific relationship between Friend2 and Friend3
MATCH (f2:Person {name: 'Friend2'}), (f3:Person {name: 'Friend3'})
CREATE (f2)-[:FRIENDS]→(f3)
```

Created Graph



Exporting Data

- The data from a graph database can be exported in CSV, JSON, PNG, and SVG formats, each presenting information differently.
-

- JSON displays the backend code, while CSV represents the data in a table format.

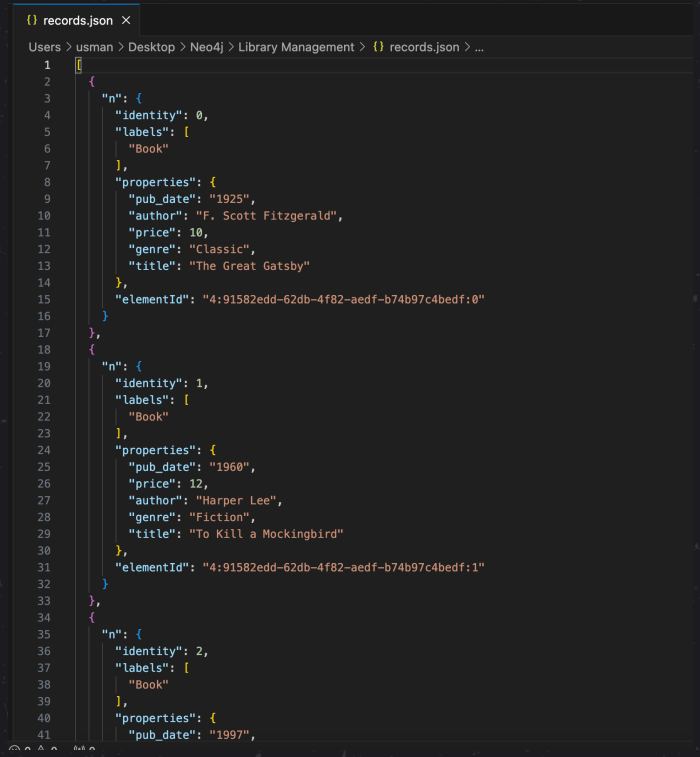
- On the other hand, SVG and PNG are image formats used for graphical representations.

Example of CSV Output

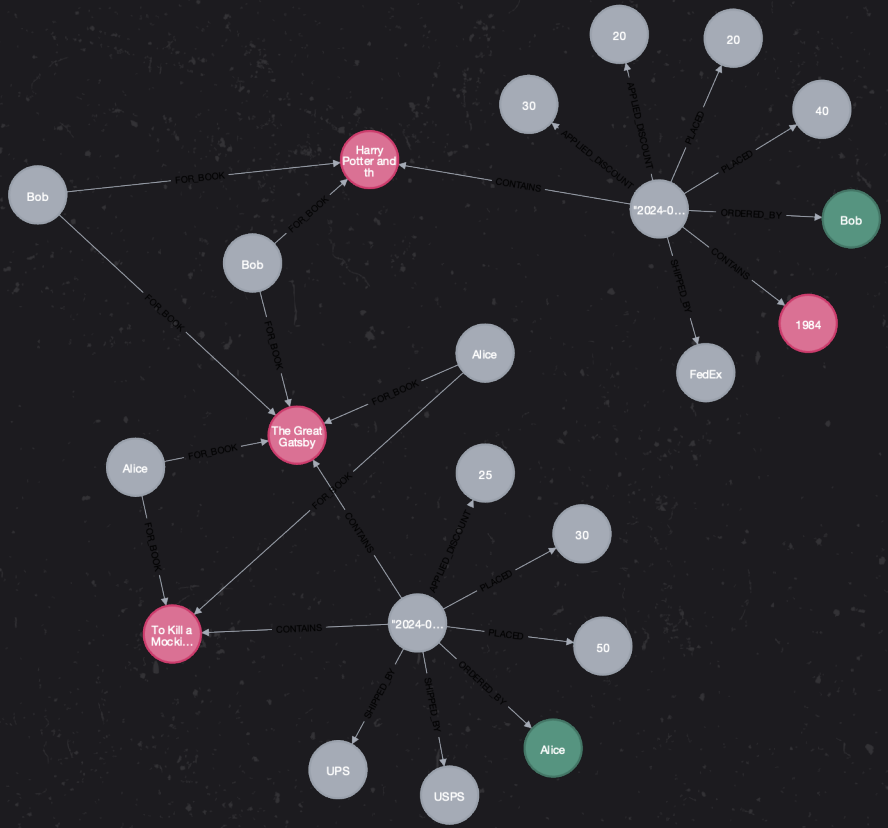
export

n
(:Book {pub_date: "1925",author: "F. Scott Fitzgerald",price: 10,genre: "Classic",title: "The Great Gatsby"})
(:Book {pub_date: "1960",price: 12,author: "Harper Lee",genre: "Fiction",title: "To Kill a Mockingbird"})
(:Book {pub_date: "1997",price: 15,author: "J.K. Rowling",genre: "Fantasy",title: "Harry Potter and the Philosopher's Stone"})
(:Book {pub_date: "1949",price: 11,author: "George Orwell",genre: "Dystopian",title: "1984"})
(:Customer {address: "123 Main St, City, Country",name: "Alice",payment: "Credit Card",email: "alice@example.com"})
(:Customer {address: "456 Oak St, Town, Country",name: "Bob",payment: "PayPal",email: "bob@example.com"})
(:Order {date: "2024-05-15",total: 37,order_id: "ORD001",status: "Processing"})
(:Order {date: "2024-05-16",total: 28,order_id: "ORD002",status: "Shipped"})
(:Inventory {quantity: 50,book_id: 1})
(:Inventory {quantity: 30,book_id: 2})
(:Inventory {quantity: 40,book_id: 3})
(:Inventory {quantity: 20,book_id: 4})
(:Review {rating: 4,comment: "Great book, highly recommended!",book_id: 1,customer_name: "Alice"})
(:Review {rating: 5,comment: "One of the best classics I've read",book_id: 1,customer_name: "Bob"})
(:Review {rating: 5,comment: "Amazing storytelling, a must-read",book_id: 2,customer_name: "Alice"})
(:Review {rating: 4,comment: "Captivating world and characters",book_id: 3,customer_name: "Bob"})
(:Discount {amount: 25,code: "SPRING25"})
(:Discount {amount: 20,code: "SUMMER20"})
(:Discount {amount: 30,code: "FALL30"})
(:ShippingProvider {service_type: "Express",name: "UPS"})
(:ShippingProvider {service_type: "Standard",name: "FedEx"})
(:ShippingProvider {service_type: "Priority Mail",name: "USPS"})

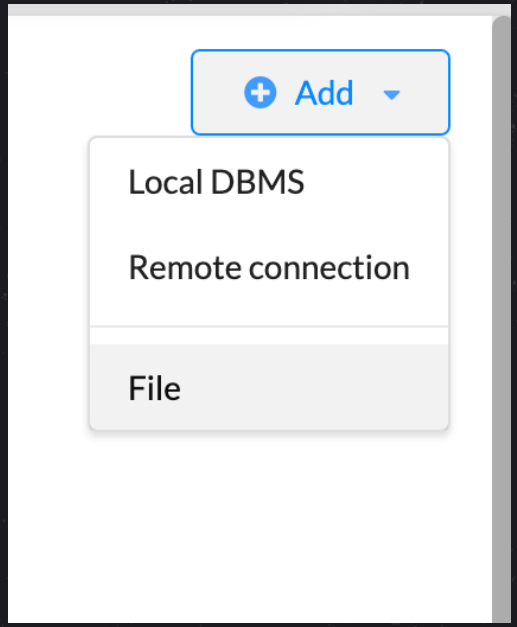
Example of JSON Output



Example of SVG Output



Importing Data



Data can be imported from local database management system (DBMS) files, via a remote connection, or by directly uploading a file.

Graph Modelling

- Create schemas that accurately represent your data and optimize performance for common query patterns.

- Graph databases excel at modeling complex relationships between entities, allowing for flexible schema evolution as your data needs change over time.

- Structure your graph data to minimize traversal and maximize query performance, ensuring fast and efficient data retrieval.

Integration

- Seamlessly connect Neo4j with existing systems and tools to leverage its graph capabilities alongside other technologies in your ecosystem.

- Neo4j can be integrated with popular frameworks and databases to enhance application development and data analysis.

- Examples: Spring framework, relational databases:

03

Advanced Queries

With the Help of Examples

Advanced Functions

- The “COUNT” function is used to count the number of items returned by a query,

- The “SUM” function is used to calculate the sum of numeric values in a result set. It's commonly used with the “RETURN” clause to aggregate and compute totals.

- The “CASE” clause allows you to conditionally execute expressions based on specified conditions.

COUNT Function

- To count the number of users who are friends with another user with a specific name (let's say "John"), you could use:

```
MATCH (:User {name: 'John'})-[:FRIENDS_WITH]→(friend)
RETURN COUNT(friend) AS johnsFriendsCount

```

- This query counts the number of nodes that are connected to "John" through a FRIENDS_WITH relationship.

SUM Function

- Let's say you have a graph representing a sales database where nodes represent orders and have a property “amount” indicating the total amount of each order. To calculate the total sales amount, you would use the following query:

```
1 MATCH (order:Order)
2 RETURN SUM(order.amount) AS totalSales
3
```

- This query will return the sum of the “amount” property of all Order nodes in your graph database, giving you the total sales amount.

CASE Function

- Suppose you have a graph representing a social network, and you want to categorize users based on the number of friends they have. You can use CASE to create different labels for users based on the number of friends they have.

```
MATCH (user:User)
RETURN user.name,
  CASE
    WHEN size((user)-[:FRIENDS_WITH]→()) = 0 THEN 'No friends'
    WHEN size((user)-[:FRIENDS_WITH]→()) ⩽ 5 THEN 'Few friends'
    ELSE 'Many friends'
  END AS friendCategory
```

- For each user, we check the number of outgoing FRIENDS_WITH relationships using the size function.
- If the user has no friends (size((user)-[:FRIENDS_WITH]->()) = 0), we label them as 'No friends'.
- If the user has 1 to 5 friends (size((user)-[:FRIENDS_WITH]->()) <= 5), we label them as 'Few friends'.
- Otherwise, we label them as 'Many friends'.
- The result of the CASE expression is aliased as friendCategory in the returned result set.

Example 1: University Management System

Creating Students and Courses: Students with their names and ages are created, and courses with their names and credit values are created.

Enrolling Students in Courses: Students are enrolled in various courses using the ENROLLED_IN relationship.

Adding Relationships: Additional relationships are created, such as TAUGHT_BY to connect professors to courses and WORKS_IN to connect professors to departments.

Creating Departments and Professors: Departments and professors with their names and department affiliations are created.

Assigning Professors to Departments: Professors are associated with their respective departments using the WORKS_IN relationship.

Creating Courses within Departments: Specific courses are created within their respective departments.

Enrolling Students in Additional Courses: More students are enrolled in different courses.

Assigning Professors to Courses: Professors are assigned to teach specific courses using the TEACHES relationship.

Creating Assignments: Various assignments with their names and due dates are created.

Connecting Assignments to Courses: Assignments are linked to the courses they belong to using the FOR relationship.

Grading Assignments: Students submit assignments, and their grades are recorded.

Using SUM, COUNT, and CASE: Finally, the script uses aggregation functions and conditional logic to calculate total assignments submitted by each student (total_assignments) and the total score obtained by each student across all submitted assignments (total_score). It uses the SUM function to calculate the total score, the COUNT function to count the total assignments, and the CASE statement to handle cases where the grade might be missing or null.

CYPHER

```
// Create students
CREATE (:Student {name: "John", age: 20})
CREATE (:Student {name: "Alice", age: 22})
CREATE (:Student {name: "Bob", age: 21})

// Create courses
CREATE (:Course {name: "Mathematics", credits: 4})
CREATE (:Course {name: "Computer Science", credits: 3})
CREATE (:Course {name: "History", credits: 3})

// Enroll students in courses
MATCH (s:Student {name: "John"}), (c:Course {name: "Mathematics"})
CREATE (s)-[:ENROLLED_IN]→(c)

MATCH (s:Student {name: "John"}), (c:Course {name: "Computer Science"})
CREATE (s)-[:ENROLLED_IN]→(c)

MATCH (s:Student {name: "Alice"}), (c:Course {name: "Computer Science"})
CREATE (s)-[:ENROLLED_IN]→(c)

MATCH (s:Student {name: "Bob"}), (c:Course {name: "History"})
CREATE (s)-[:ENROLLED_IN]→(c)

// Add some more relationships
MATCH (s:Student {name: "John"}), (c:Course {name: "Mathematics"})
CREATE (s)-[:TAUGHT_BY]→(:Professor {name: "Dr. Smith"})

MATCH (s:Student {name: "Alice"}), (c:Course {name: "Computer Science"})
CREATE (s)-[:TAUGHT_BY]→(:Professor {name: "Prof. Johnson"})

MATCH (s:Student {name: "Bob"}), (c:Course {name: "History"})
CREATE (s)-[:TAUGHT_BY]→(:Professor {name: "Dr. Brown"})

```

CONTINUED...

```
// Create departments
CREATE (:Department {name: "Computer Science"})
CREATE (:Department {name: "Mathematics"})
CREATE (:Department {name: "History"})

// Create professors
CREATE (:Professor {name: "Dr. Smith", department: "Computer Science"})
CREATE (:Professor {name: "Prof. Johnson", department: "Computer Science"})
CREATE (:Professor {name: "Dr. Brown", department: "History"})
CREATE (:Professor {name: "Dr. White", department: "Mathematics"})

// Assign professors to departments
MATCH (p:Professor {name: "Dr. Smith"}), (d:Department {name: "Computer Science"})
CREATE (p)-[:WORKS_IN]->(d)

MATCH (p:Professor {name: "Prof. Johnson"}), (d:Department {name: "Computer Science"})
CREATE (p)-[:WORKS_IN]->(d)

MATCH (p:Professor {name: "Dr. Brown"}), (d:Department {name: "History"})
CREATE (p)-[:WORKS_IN]->(d)

MATCH (p:Professor {name: "Dr. White"}), (d:Department {name: "Mathematics"})
CREATE (p)-[:WORKS_IN]->(d)

// Create courses within departments
MATCH (d:Department {name: "Computer Science"})
CREATE (:Course {name: "Introduction to Programming", department: "Computer Science"})
CREATE (:Course {name: "Data Structures", department: "Computer Science"})

MATCH (d:Department {name: "Mathematics"})
CREATE (:Course {name: "Calculus", department: "Mathematics"})

MATCH (d:Department {name: "History"})
CREATE (:Course {name: "World History", department: "History"})
```

CONTINUED...

```
67 CREATE (:Course {name: "World History", department: "History"})
68 CREATE (:Course {name: "European History", department: "History"})
69
70 // Enroll students in courses
71 MATCH (s:Student {name: "John"}), (c:Course {name: "Introduction to Programming"})
72 CREATE (s)-[:ENROLLED_IN]→(c)
73
74 MATCH (s:Student {name: "John"}), (c:Course {name: "Data Structures"})
75 CREATE (s)-[:ENROLLED_IN]→(c)
76
77 MATCH (s:Student {name: "Alice"}), (c:Course {name: "Introduction to Programming"})
78 CREATE (s)-[:ENROLLED_IN]→(c)
79
80 MATCH (s:Student {name: "Bob"}), (c:Course {name: "Calculus"})
81 CREATE (s)-[:ENROLLED_IN]→(c)
82
83 // Assign professors to courses
84 MATCH (p:Professor {name: "Dr. Smith"}), (c:Course {name: "Introduction to Programming"})
85 CREATE (p)-[:TEACHES]→(c)
86
87 MATCH (p:Professor {name: "Prof. Johnson"}), (c:Course {name: "Data Structures"})
88 CREATE (p)-[:TEACHES]→(c)
89
90 MATCH (p:Professor {name: "Dr. White"}), (c:Course {name: "Calculus"})
91 CREATE (p)-[:TEACHES]→(c)
92
93 // Create assignments for courses
94 CREATE (:Assignment {name: "Programming Project 1", due_date: "2024-05-20"})
95 CREATE (:Assignment {name: "Data Structures Assignment 1", due_date: "2024-05-25"})
96 CREATE (:Assignment {name: "Calculus Quiz 1", due_date: "2024-05-30"})
97 CREATE (:Assignment {name: "World History Essay", due_date: "2024-06-01"})
98
99 // Connect assignments to courses
100 MATCH (a:Assignment {name: "Programming Project 1"}), (c:Course {name: "Introduction to Programming"})
101 CREATE (a)-[:FOR]→(c)
```

CONTINUED...

```
// Connect assignments to courses
MATCH (a:Assignment {name: "Programming Project 1"}), (c:Course {name: "Introduction to Programming"})
CREATE (a)-[:FOR]→(c)

MATCH (a:Assignment {name: "Data Structures Assignment 1"}), (c:Course {name: "Data Structures"})
CREATE (a)-[:FOR]→(c)

MATCH (a:Assignment {name: "Calculus Quiz 1"}), (c:Course {name: "Calculus"})
CREATE (a)-[:FOR]→(c)

MATCH (a:Assignment {name: "World History Essay"}), (c:Course {name: "World History"})
CREATE (a)-[:FOR]→(c)

// Grade assignments
MATCH (s:Student {name: "John"}), (a:Assignment {name: "Programming Project 1"})
CREATE (s)-[:SUBMITTED]→(a)
SET a.grade = 85

MATCH (s:Student {name: "Alice"}), (a:Assignment {name: "Programming Project 1"})
CREATE (s)-[:SUBMITTED]→(a)
SET a.grade = 92

MATCH (s:Student {name: "Bob"}), (a:Assignment {name: "Calculus Quiz 1"})
CREATE (s)-[:SUBMITTED]→(a)
SET a.grade = 78

MATCH (s:Student {name: "Bob"}), (a:Assignment {name: "World History Essay"})
CREATE (s)-[:SUBMITTED]→(a)
SET a.grade = 88

// Use SUM, COUNT, and CASE
MATCH (s:Student)-[:SUBMITTED]→(a:Assignment)
WITH s, COUNT(a) AS total_assignments, SUM(CASE WHEN a.grade IS NOT NULL THEN a.grade ELSE 0 END) AS total_score
RETURN s.name, total_assignments, total_score

```

OUTPUT

export

n
(:Student {name: "John",age: 20})
(:Student {name: "Alice",age: 22})
(:Student {name: "Bob",age: 21})
(:Course {credits: 4,name: "Mathematics"})
(:Course {credits: 3,name: "Computer Science"})
(:Professor {name: "Dr. Smith"})
(:Professor {name: "Prof. Johnson"})
(:Department {name: "Computer Science"})
(:Department {name: "Mathematics"})
(:Professor {name: "Dr. Smith",department: "Computer Science"})
(:Professor {name: "Prof. Johnson",department: "Computer Science"})
(:Professor {name: "Dr. White",department: "Mathematics"})
(:Course {name: "Introduction to Programming",department: "Computer Science"})
(:Course {name: "Data Structures",department: "Computer Science"})
(:Course {name: "Calculus",department: "Mathematics"})
(:Course {name: "Linear Algebra",department: "Mathematics"})
(:Assignment {grade: 92,due_date: "2024-05-20",name: "Programming Project 1"})
(:Assignment {due_date: "2024-05-25",name: "Data Structures Assignment 1"})
(:Assignment {grade: 78,due_date: "2024-05-30",name: "Calculus Quiz 1"})

USE OF COUNT, SUM AND CASE

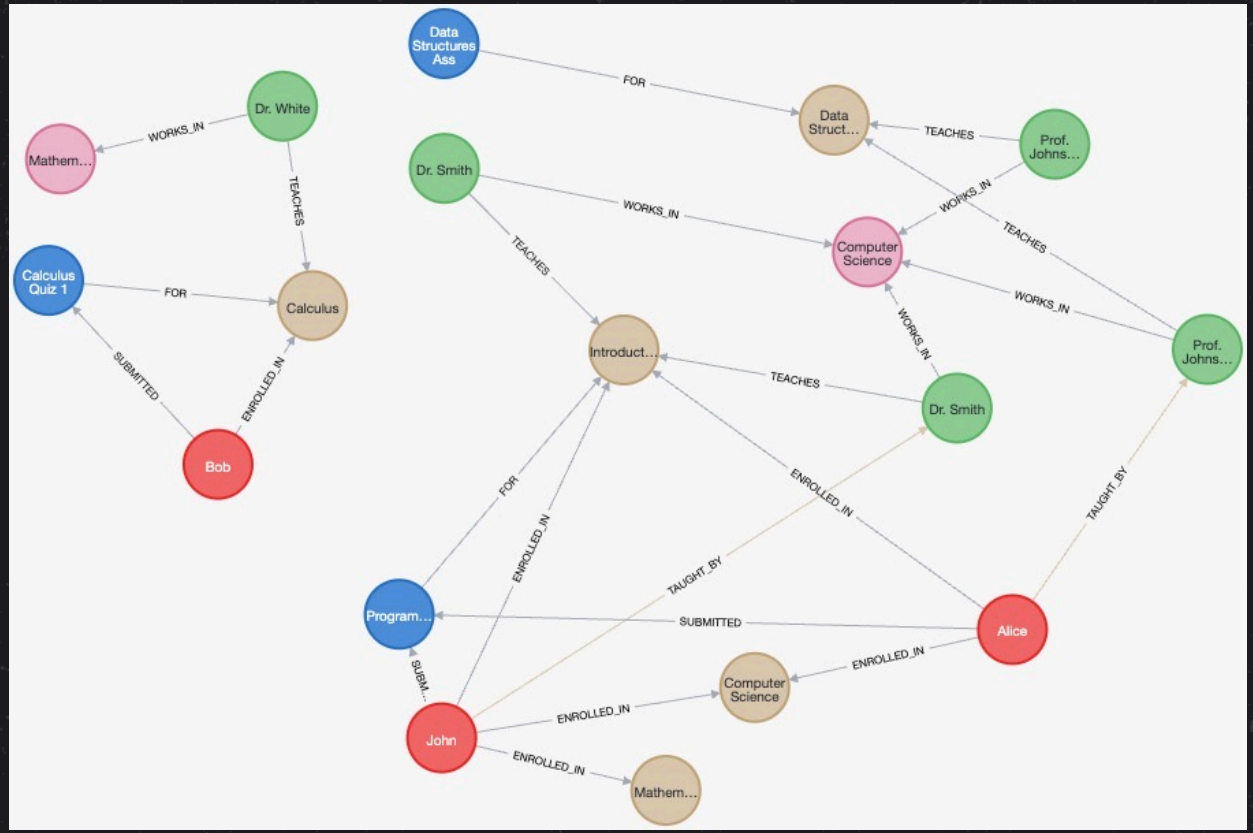
```
neo4j$ MATCH (s:Student)-[:SUBMITTED]→(a:Assignment) WITH s, COUNT(a) AS total_assignment…
```

Table | Text | Code

	s.name	total_assignments	total_score
1	"John"	1	92
2	"Alice"	1	92
3	"Bob"	1	78

Started streaming 3 records after 5 ms and completed after 12 ms.

GRAPH


Data Structures Ass
FOR
Data Struct...
TEACHES
Prof. Johns...
Dr. White
WORKS_IN
Mathem...
TEACHES
Dr. Smith
WORKS_IN
Computer Science
WORKS_IN
TEACHES
Calculus Quiz 1
FOR
Calculus
WORKS_IN
Prof. Johns...
TEACHES
Introduct...
WORKS_IN
SUBMITTED
ENROLLED_IN
TEACHES
Dr. Smith
Bob
ENROLLED_IN
FOR
TAUGHT_BY
ENROLLED_IN
Program...
SUBMITTED
Alice
TAUGHT_BY
ENROLLED_IN
SUBM...
Computer Science
ENROLLED_IN
John
ENROLLED_IN
Mathem...

Example 2: Library Management System

Creating Reviews: Creates reviews for different books, each review containing information like the book ID, customer name, rating, and comment.

Creating Discounts: Creates discount codes with their respective amounts.

Creating Shipping Providers: Creates shipping providers with their names and service types.

Associating Reviews with Books: Matches reviews with books based on certain conditions and creates relationships between them.

Associating Discounts with Orders: Matches discounts with orders based on discount codes and order IDs and creates relationships between them.

Associating Shipping Providers with Orders: Matches shipping providers with orders based on provider names and order IDs and creates relationships between them.

Calculating Average Rating per Book: Calculates the average rating for each book based on the reviews associated with it.

Calculating Total Discounts Applied: Counts the total number of discounts applied to orders.

Calculating Total Orders Shipped by Each Provider: Counts the total number of orders shipped by each shipping provider.

Calculating Total Discount Amount: Calculates the total amount of discounts given across all orders.

CYPHER

```
// Create reviews
CREATE (:Review {book_id: 1, customer_name: "Alice", rating: 4, comment: "Great book, highly recommended!"})
CREATE (:Review {book_id: 1, customer_name: "Bob", rating: 5, comment: "One of the best classics I've read"})
CREATE (:Review {book_id: 2, customer_name: "Alice", rating: 5, comment: "Amazing storytelling, a must-read"})
CREATE (:Review {book_id: 3, customer_name: "Bob", rating: 4, comment: "Captivating world and characters"})

// Create discounts
CREATE (:Discount {code: "SPRING25", amount: 25})
CREATE (:Discount {code: "SUMMER20", amount: 20})
CREATE (:Discount {code: "FALL30", amount: 30})

// Create shipping providers
CREATE (:ShippingProvider {name: "UPS", service_type: "Express"})
CREATE (:ShippingProvider {name: "FedEx", service_type: "Standard"})
CREATE (:ShippingProvider {name: "USPS", service_type: "Priority Mail"})

// Associate reviews with books
MATCH (r:Review), (b:Book {title: "The Great Gatsby"})
WHERE r.customer_name IN ["Alice", "Bob"] AND b.book_id = r.book_id
CREATE (r)-[:FOR_BOOK]→(b)

MATCH (r:Review), (b:Book {title: "To Kill a Mockingbird"})
WHERE r.customer_name = "Alice" AND b.book_id = r.book_id
CREATE (r)-[:FOR_BOOK]→(b)

MATCH (r:Review), (b:Book {title: "Harry Potter and the Philosopher's Stone"})
WHERE r.customer_name = "Bob" AND b.book_id = r.book_id
CREATE (r)-[:FOR_BOOK]→(b)

// Associate discounts with orders
MATCH (d:Discount {code: "SPRING25"}), (o:Order {order_id: "ORD001"})
CREATE (o)-[:APPLIED_DISCOUNT]→(d)

```

CONTINUED...

```
MATCH (d:Discount {code: "SUMMER20"}), (o:Order {order_id: "ORD002"})
CREATE (o)-[:APPLIED_DISCOUNT]→(d)

MATCH (d:Discount {code: "FALL30"}), (o:Order {order_id: "ORD002"})
CREATE (o)-[:APPLIED_DISCOUNT]→(d)

// Associate shipping providers with orders
MATCH (s:ShippingProvider {name: "UPS"}), (o:Order {order_id: "ORD001"})
CREATE (o)-[:SHIPPED_BY]→(s)

MATCH (s:ShippingProvider {name: "USPS"}), (o:Order {order_id: "ORD001"})
CREATE (o)-[:SHIPPED_BY]→(s)

MATCH (s:ShippingProvider {name: "FedEx"}), (o:Order {order_id: "ORD002"})
CREATE (o)-[:SHIPPED_BY]→(s)

// Calculate average rating per book
MATCH (b:Book)←[:FOR_BOOK]-(r:Review)
RETURN b.title,
       AVG(r.rating) AS average_rating,
       COUNT(r) AS total_reviews

// Calculate total discounts applied
MATCH (o:Order)-[:APPLIED_DISCOUNT]→(d:Discount)
RETURN COUNT(d) AS total_discounts_applied

// Calculate total orders shipped by each provider
MATCH (s:ShippingProvider)←[:SHIPPED_BY]-(o:Order)
RETURN s.name,
       COUNT(o) AS total_orders_shipped

// Calculate the total amount of discounts given
MATCH (o:Order)-[:APPLIED_DISCOUNT]→(d:Discount)
RETURN SUM(d.amount) AS total_discount_amount
```

OUTPUT

export

	n
1	(:Book {pub_date: "1925",author: "F. Scott Fitzgerald",price: 10,genre: "Classic",title: "The Great Gatsby"})
2	(:Book {pub_date: "1960",price: 12,author: "Harper Lee",genre: "Fiction",title: "To Kill a Mockingbird"})
3	(:Book {pub_date: "1997",price: 15,author: "J.K. Rowling",genre: "Fantasy",title: "Harry Potter and the Philosopher's Stone"})
4	(:Book {pub_date: "1949",price: 11,author: "George Orwell",genre: "Dystopian",title: "1984"})
5	(:Customer {address: "123 Main St, City, Country",name: "Alice",payment: "Credit Card",email: "alice@example.com"})
6	(:Customer {address: "456 Oak St, Town, Country",name: "Bob",payment: "PayPal",email: "bob@example.com"})
7	(:Order {date: "2024-05-15",total: 37,order_id: "ORD001",status: "Processing"})
8	(:Order {date: "2024-05-16",total: 28,order_id: "ORD002",status: "Shipped"})
9	(:Inventory {quantity: 50,book_id: 1})
10	(:Inventory {quantity: 30,book_id: 2})
11	(:Inventory {quantity: 40,book_id: 3})
12	(:Inventory {quantity: 20,book_id: 4})
13	(:Review {rating: 4,comment: "Great book, highly recommended!",book_id: 1,customer_name: "Alice"})
14	(:Review {rating: 5,comment: "One of the best classics I've read",book_id: 1,customer_name: "Bob"})
15	(:Review {rating: 5,comment: "Amazing storytelling, a must-read",book_id: 2,customer_name: "Alice"})
16	(:Review {rating: 4,comment: "Captivating world and characters",book_id: 3,customer_name: "Bob"})
17	(:Discount {amount: 25,code: "SPRING25"})
18	(:Discount {amount: 20,code: "SUMMER20"})
19	(:Discount {amount: 30,code: "FALL30"})
20	(:ShippingProvider {service_type: "Express",name: "UPS"})
21	(:ShippingProvider {service_type: "Standard",name: "FedEx"})
22	(:ShippingProvider {service_type: "Priority Mail",name: "USPS"})

USE OF COUNT, SUM, CASE

```
neo4j$ MATCH (s:ShippingProvider)←[:SHIPPED_BY]-(o:Order) RETURN s.name, COUNT(o) AS total_orders_shipped
```

Table | Text | Code

	s.name	total_orders_shipped
1	"UPS"	1
2	"FedEx"	1
3	"USPS"	1

Started streaming 3 records after 1 ms and completed after 2 ms.

```
neo4j$ MATCH (b:Book)←[:FOR_BOOK]-(r:Review) RETURN b.title, AVG(r.rating) AS average_rating, COUNT(r) AS total_reviews
```

Table | Text | Code

	b.title	average_rating	total_reviews
1	"The Great Gatsby"	4.5	4
2	"To Kill a Mockingbird"	4.5	2
3	"Harry Potter and the Philosopher's Stone"	4.5	2

Started streaming 3 records after 1 ms and completed after 3 ms.

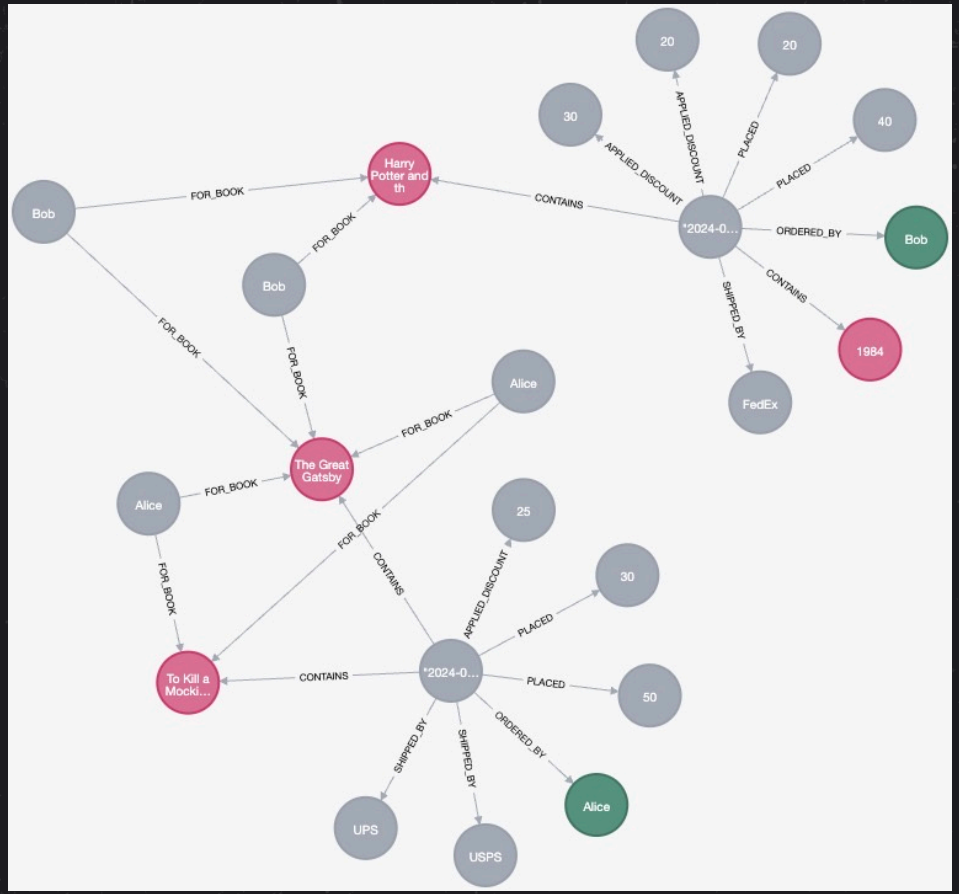
```
neo4j$ MATCH (o:Order)-[:APPLIED_DISCOUNT]→(d:Discount) RETURN SUM(d.amount) AS total_discount_amount
```

Table | Text | Code

	total_discount_amount
1	75

Started streaming 1 records after 1 ms and completed after 3 ms.

GRAPH


20
20
30
40
APPLIED_DISCOUNT
APPLIED_DISCOUNT
PLACED
PLACED
Harry Potter and th
CONTAINS
*2024-0...
ORDERED_BY
Bob
Bob
FOR_BOOK
FOR_BOOK
Bob
CONTAINS
SHIPPED_BY
1984
FedEx
FOR_BOOK
FOR_BOOK
Alice
FOR_BOOK
The Great Gatsby
Alice
FOR_BOOK
25
FOR_BOOK
CONTAINS
APPLIED_DISCOUNT
30
PLACED
To Kill a Mocki...
CONTAINS
*2024-0...
PLACED
50
SHIPPED_BY
SHIPPED_BY
ORDERED_BY
Alice
UPS
USPS

04

Examples

Real Life Examples

Social Network Analysis:
Example: Facebook's Friend Network
Neo4j is used to model relationships between users (nodes) and their friendships (edges). This allows for efficient querying of connections and analysis of network structures.

Recommendation Systems:
Example: Netflix's Movie Recommendations
Neo4j can model user preferences, movie attributes, and historical viewing patterns. By analyzing the graph structure, personalized recommendations can be generated efficiently.

Fraud Detection:
Example: Banking Transactions
Explanation: Neo4j can represent banking transactions and account relationships. By analyzing patterns such as unusual connections or behaviors, fraud detection algorithms can identify potential fraudulent activities.

Network and IT Operations:
Example: IT Infrastructure Management
Neo4j can model network topologies, dependencies between servers, and configurations. This helps in troubleshooting, capacity planning, and optimizing IT operations.

Practical Examples

Content Management:
Example: Spotify Music Recommendations
Neo4j can model relationships between songs, artists, genres, user preferences, and listening history. By analyzing this graph, Spotify can generate personalized music recommendations for its users, enhancing their listening experience and increasing engagement on the platform.

Healthcare Systems:
Example: Patient Care Coordination
Neo4j can model patient records, medical facilities, healthcare providers, and treatment pathways. This facilitates care coordination, referrals, and patient outcomes analysis in healthcare systems, improving efficiency and quality of care.

Transportation and Logistics:
Example: Route Optimization
Neo4j can model transportation networks, including roads, railways, ports, and airports, along with factors like traffic conditions and vehicle capacities. This enables route optimization for delivery fleets, public transit systems, and logistics operations.

05

Conclusion

CONCLUSION

- In summary, Neo4j is a game-changer for handling data relationships across various fields. Its special graph database design makes complex relationships clear and manageable. Whether it's analyzing social networks, optimizing transportation routes, or streamlining healthcare systems, Neo4j offers flexible solutions.

- By adopting Neo4j, organizations can gain insights, make better decisions, and improve user experiences. In the fast-paced world of data tech, Neo4j leads the charge, paving the way for smarter data management and analysis.

Thank You!