Querying Graph Databases: Neo4j

Querying Databases: The Non-relational Ways

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

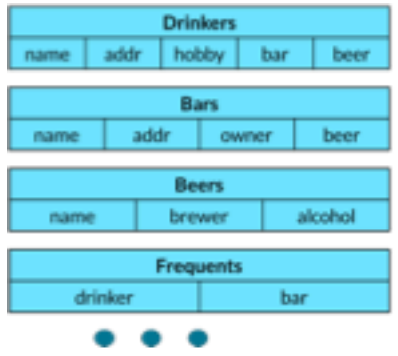
Kevin C.C. Chang, Professor
Computer Science @ Illinois

Learning Objectives

By the end of this video, you will be able to:

- Describe the basic mechanism for querying Neo4j.
- Identify the language used for querying Neo4j and how it is conceptually related to SQL.

FridayNight Database– Now It's a Graph!



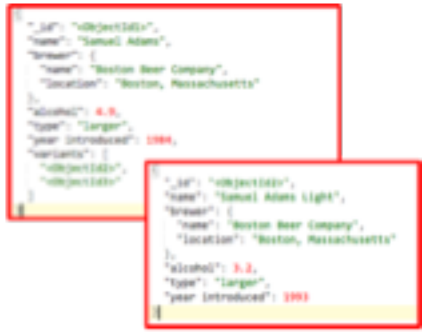
Drinkers				
name	addr	hobby	bar	beer

Bars			
name	addr	owner	beer

Beers		
name	brewer	alcohol

Frequents	
drinker	bar

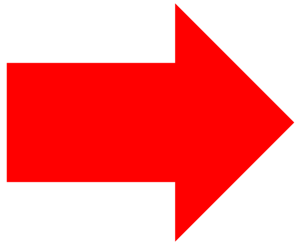
Relational database



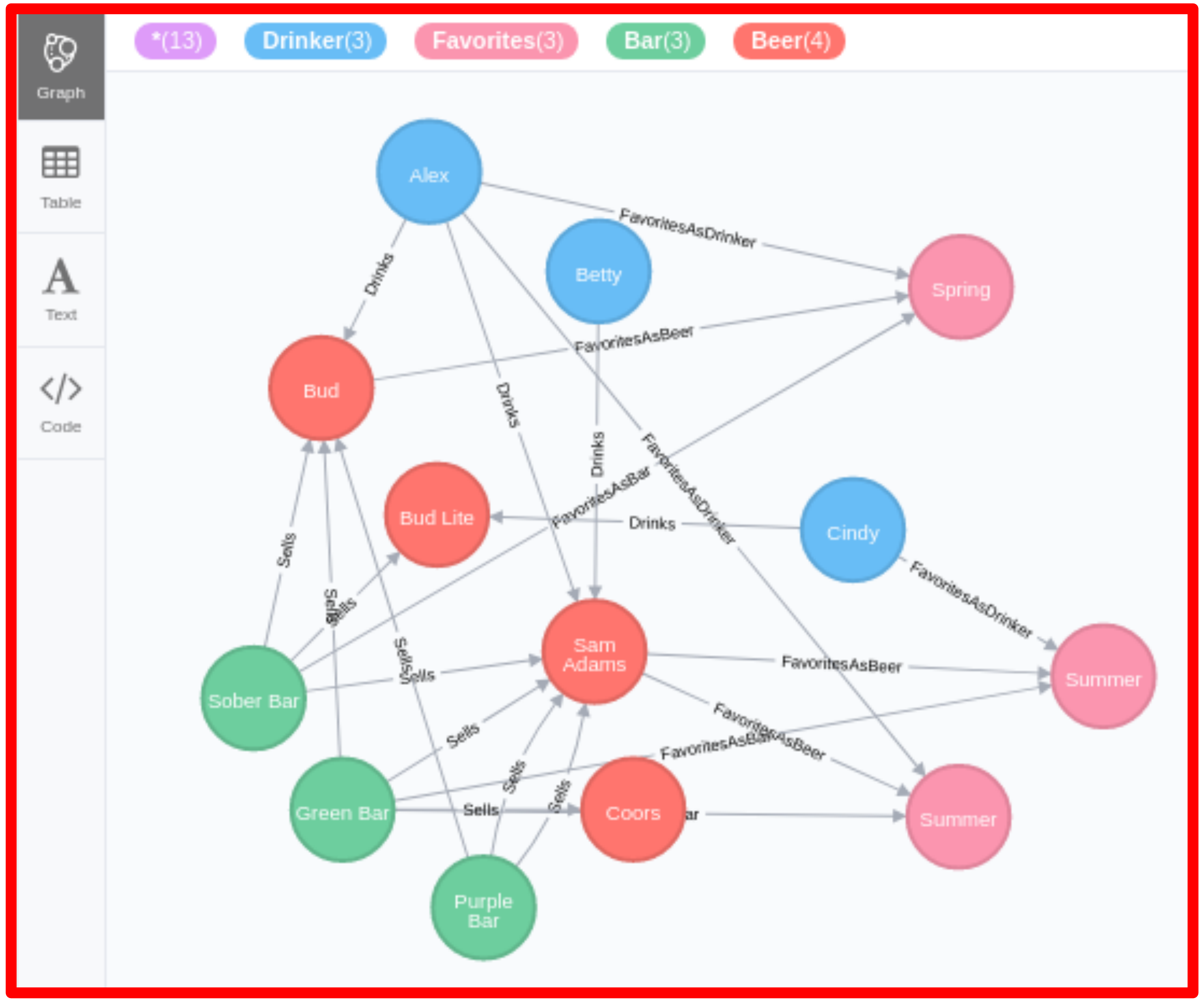
```
{
  "_id": "<ObjectId1>",
  "name": "Samuel Adams",
  "brewer": {
    "name": "Boston Beer Company",
    "location": "Boston, Massachusetts"
  },
  "alcohol": 4.9,
  "type": "larger",
  "year introduced": 1984,
  "variants": [
    "<ObjectId1>",
    "<ObjectId1>"
  ]
}
```

```
{
  "_id": "<ObjectId1>",
  "name": "Samuel Adams Light",
  "brewer": {
    "name": "Boston Beer Company",
    "location": "Boston, Massachusetts"
  },
  "alcohol": 3.2,
  "type": "larger",
  "year introduced": 1993
}
```

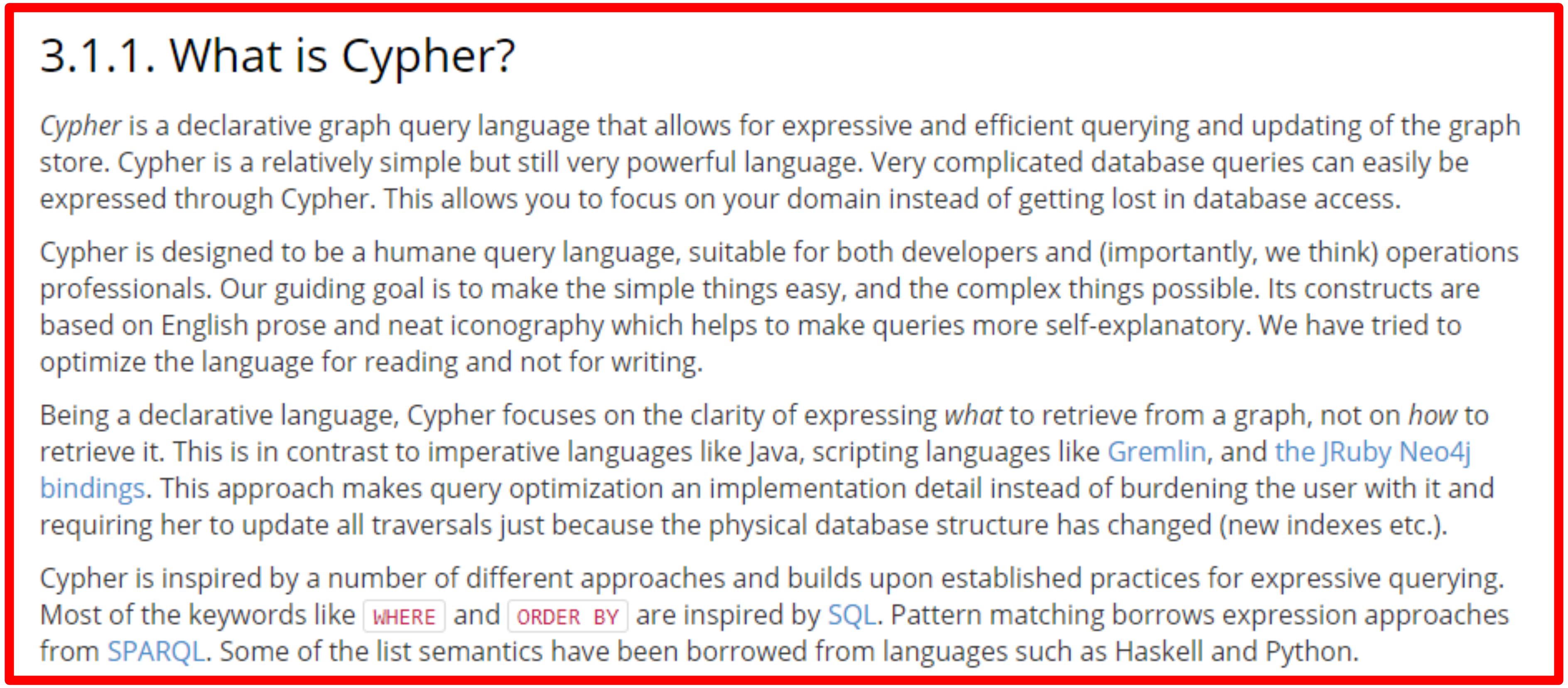
Document database



Graph database



Neo4j Query Language: Cypher



3.1.1. What is Cypher?

Cypher is a declarative graph query language that allows for expressive and efficient querying and updating of the graph store. Cypher is a relatively simple but still very powerful language. Very complicated database queries can easily be expressed through Cypher. This allows you to focus on your domain instead of getting lost in database access.

Cypher is designed to be a humane query language, suitable for both developers and (importantly, we think) operations professionals. Our guiding goal is to make the simple things easy, and the complex things possible. Its constructs are based on English prose and neat iconography which helps to make queries more self-explanatory. We have tried to optimize the language for reading and not for writing.

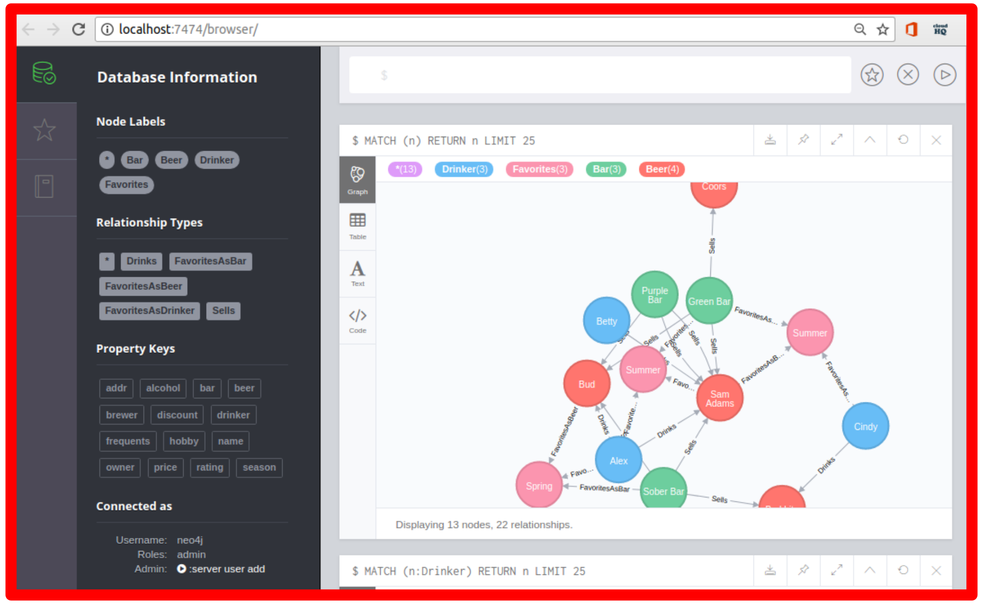
Being a declarative language, Cypher focuses on the clarity of expressing *what* to retrieve from a graph, not on *how* to retrieve it. This is in contrast to imperative languages like Java, scripting languages like Gremlin, and the JRuby Neo4j bindings. This approach makes query optimization an implementation detail instead of burdening the user with it and requiring her to update all traversals just because the physical database structure has changed (new indexes etc.).

Cypher is inspired by a number of different approaches and builds upon established practices for expressive querying. Most of the keywords like `WHERE` and `ORDER BY` are inspired by SQL. Pattern matching borrows expression approaches from SPARQL. Some of the list semantics have been borrowed from languages such as Haskell and Python.

Cypher Manual 2017. Retrieved from https://neo4j.com/docs/developer-manual

Querying Neo4j: Mechanism

- Queries are issued in the Cypher language.
- You can access databases using an interactive GUI shell “Neo4j Browser”.
- Applications can access databases via client language drivers.



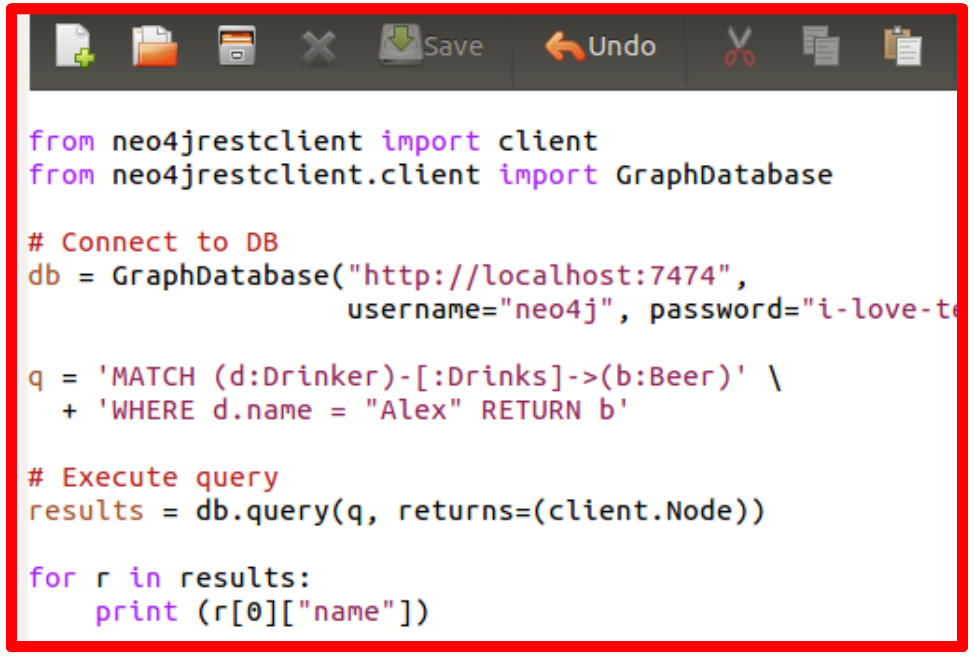
```
$ MATCH (n) RETURN n LIMIT 25
```

Displaying 13 nodes, 22 relationships.

```
$ MATCH (n:Drinker) RETURN n LIMIT 25
```

Username: neo4j
Roles: admin
Admin: :server user add

Accessing Neo4J from Neo4J Browser



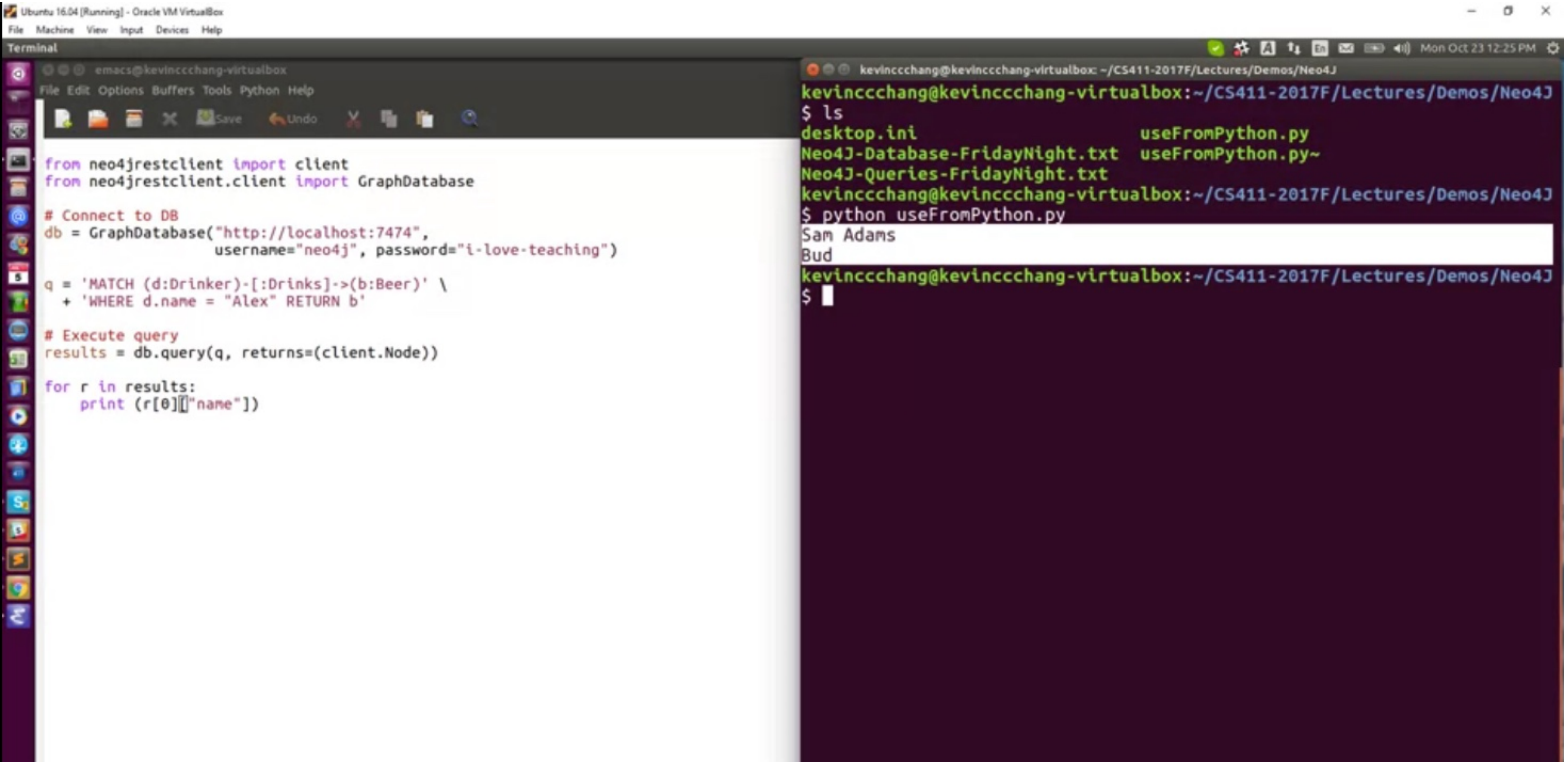
```
from neo4jrestclient import client
from neo4jrestclient.client import GraphDatabase

# Connect to DB
db = GraphDatabase("http://localhost:7474",
                   username="neo4j", password="i-love-te

q = 'MATCH (d:Drinker)-[:Drinks]->(b:Beer)' \
  + 'WHERE d.name = "Alex" RETURN b'

# Execute query
results = db.query(q, returns=(client.Node))

for r in results:
    print (r[0]["name"])
```

Accessing N4o4J from Python

Neo4j Querying Examples

- *Q1: Using Neo4J Browser, explore the FridayNight database.*

- *Q2: Perform some querying from Python.*


Ubuntu 16.04 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Terminal
Mon Oct 23 12:25 PM
emacs@kevincchang-virtualbox
File Edit Options Buffers Tools Python Help
Save Undo
kevinccchang@kevinccchang-virtualbox: ~/CS411-2017F/Lectures/Demos/Neo4J

```python
from neo4jrestclient import client
from neo4jrestclient.client import GraphDatabase

# Connect to DB
db = GraphDatabase("http://localhost:7474",
                   username="neo4j", password="i-love-teaching")

q = 'MATCH (d:Drinker)-[:Drinks]->(b:Beer)' \
  + 'WHERE d.name = "Alex" RETURN b'

# Execute query
results = db.query(q, returns=(client.Node))

for r in results:
    print (r[0]["name"])
```

```
kevinccchang@kevinccchang-virtualbox:~/CS411-2017F/Lectures/Demos/Neo4J
$ ls
desktop.ini                       useFromPython.py
Neo4J-Database-FridayNight.txt   useFromPython.py~
Neo4J-Queries-FridayNight.txt
kevinccchang@kevinccchang-virtualbox:~/CS411-2017F/Lectures/Demos/Neo4J
$ python useFromPython.py
Sam Adams
Bud
kevinccchang@kevinccchang-virtualbox:~/CS411-2017F/Lectures/Demos/Neo4J
$ 
```

Querying Graph Databases: Basic Operations

Querying Databases: The Non-relational Ways

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

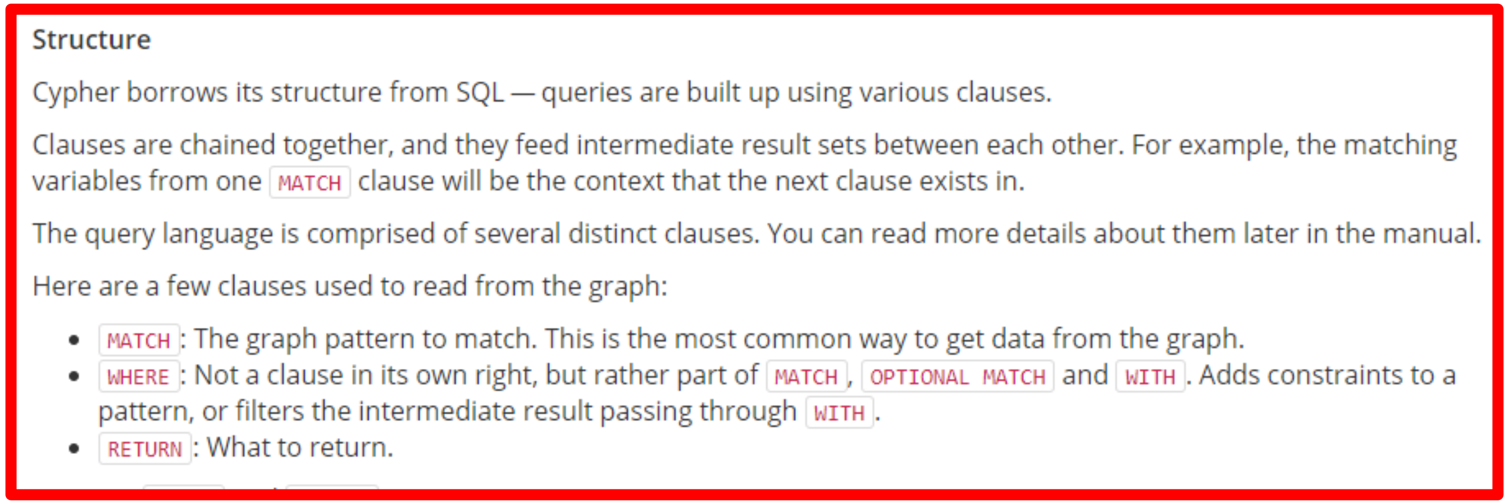
Kevin C.C. Chang, Professor
Computer Science @ Illinois

Learning Objectives

By the end of this video, you will be able to:

- Describe the basic query operations in Neo4j Cypher with MATCH-WHERE-RETURN.
- Explain how these constructs are related to SQL.
- Describe how to use patterns in Neo4j Cypher.
- Write graph queries with simple operations.

Cypher Structure: **MATCH-WHERE-RETURN** vs. SQL: FROM-WHERE-SELECT

- 

Structure

Cypher borrows its structure from SQL — queries are built up using various clauses.

Clauses are chained together, and they feed intermediate result sets between each other. For example, the matching variables from one `MATCH` clause will be the context that the next clause exists in.

The query language is comprised of several distinct clauses. You can read more details about them later in the manual.

Here are a few clauses used to read from the graph:

- `MATCH`: The graph pattern to match. This is the most common way to get data from the graph.
- `WHERE`: Not a clause in its own right, but rather part of `MATCH`, `OPTIONAL MATCH` and `WITH`. Adds constraints to a pattern, or filters the intermediate result passing through `WITH`.
- `RETURN`: What to return.

Cypher Manual 2017. Retrieved from https://neo4j.com/docs/developer-manual

- Correspondence:

SQL	Neo4j Cypher
FROM	MATCH
WHERE	WHERE
SELECT	RETURN

Patterns

- Nodes: (b:Beer {name:”Bud”})
- Relationships: (bar:Bar)-[s:Sells] -> (beer:Beer)
- Patterns can be written continuously or separated with commas.
  - (bar:Bar)-[s:Sells] -> (beer:Beer)<-[d:Drinks]-(drinker:Drinker)
  - (bar:Bar)-[s:Sells] -> (beer:Beer), (bar:Bar {addr:”Green St”})

- Variables
  - You can refer to variables declared earlier or introduce new ones.
  - You can post conditions on these variables.
    - E.g., b.name = “Bud”.

MATCH-WHERE-RETURN: Over a Node

- A table of an entity becomes a node in a graph DB.
- Querying over a node: **(var: Label)**

- *Q: Find the beers with alcohol greater than 5%.*

SQL	Neo4j Cypher
FROM Beers beer	**MATCH** (beer:Beer)
WHERE beer.alcohol > 0.05	**WHERE** beer.alcohol > 0.05
SELECT beer.name, beer.alcohol	**RETURN** beer.name, beer.alcohol

MATCH-WHERE-RETURN: Over a Relationship

- A table of a relationship becomes, naturally, a relationship in a graph DB.
- Querying over a node: **(…)-[var: Label]->(…)**

- *Q: Find the prices and bars to get the Sam Adams beer.*

SQL	Neo4j Cypher
FROM Sells s	**MATCH** (bar:Bar)-[s:Sells]->(beer:Beer)
WHERE s.beer = “Sam Adams”	**WHERE** beer.name = “Sam Adams”
SELECT s.price, s.bar	**RETURN** s.price, bar.name

Using Properties of Nodes and Relationships

- Node: **(var: Label {Properties})**
- Relationship: **(…)-[var: Label {Properties}]->(…)**

- *Q: Find the prices and bars to get the Sam Adams beer.*

```
MATCH (bar:Bar)-[sell:Sells]->(beer:Beer)
WHERE beer.name = "Sam Adams"
RETURN sell.price, bar.name
```

```
MATCH (bar:Bar)-[sell:Sells]->(beer:Beer {name:"Sam Adams"})
RETURN sell.price, bar.name
```

MATCH-WHERE-RETURN: Join Queries

- In SQL, you assemble the tables involved using joins.
- In a graph DB, you specify a pattern to traverse the graph.

- *Q: Find the beers brewed by AB InBev and sold at a price less than $5.*

SQL	**Neo4j Cypher**
FROM Beers b, Sells s	**MATCH** (bar:Bar)-[sell:Sells]->(beer:Beer {brewer : "AB InBev" })
WHERE b.name = s.beer AND b.brewer = "AB InBev" AND s.price < 5	**WHERE** sell.price < 5
SELECT b.name, s.price, s.bar	**RETURN** beer.name, sell.price, bar.name

Basic Query Examples

- *Q1: Find the beers with alcohol greater than 5%.*
- *Q2: Find the prices and bars to get the Sam Adams beer.*
- *Q3: Find the beers brewed by AB InBev and sold at a price less than $5.*

Q1

```
Match(beer:Beer) where beer.alcohol > 0.03 return beer.name
```

Q2

```
match (bar:Bar)-[s:sells]->(beer:Beer) where beer.name = “Sam
Adams” return S.price, bar.name
```

Q3

```
match (bar:Bar)-[s:sells]->(beer:Beer{brewer: “Ab InBev”}) where
price < 5 return beer.name, s.price, bar.name
```

Food for Thought

Why does a graph database need “patterns”?
What does this notion correspond in SQL?

Querying Graph Databases: Advanced Capabilities

Querying Databases: The Non-relational Ways

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

Kevin C.C. Chang, Professor
Computer Science @ Illinois

Learning Objectives

By the end of this video, you will be able to:

- Describe the aggregate framework in Neo4j Cypher.
- Explain how chaining of subqueries works.
- Write queries with these advanced capabilities.

Aggregate

```
SELECT beer.name, AVG(sell.price)
FROM Beers beer, Sells sell
WHERE beer.name = sell.beer
        and beer.brewer = "AB InBev"
GROUP BY beer.name
HAVING COUNT(sell.bar) >= 2
```

- Similar to SQL.
    - By applying aggregate functions over groups.
- No separate “GROUP BY” clause.
    - Simply put grouping attributes in “RETURN” together with aggregates.
    - RETURN $A_1, \ldots, A_n, F_1, .., F_m$
        - Equivalent to:

          SELECT $A_1, \ldots, A_n, F_1, .., F_m$

          GROUP BY $A_1, \ldots, A_n$
- No separate HAVING clause– use another “WHERE” by chaining.

Aggregate Query Examples

- *Q1: Find the highest price of beers on the market~~ and the most expensive beers with that price~~.*

- *Q2: Find the average price of each beer brewed by "AB InBev"~~ if it is sold at multiple bars~~.*

Q1

```
Match() - [s:sells] - () return max(s.price)
```

Q2

```
Match() - [s:sells] - (beer: Beer {brewer : “Ab InBev”}) beer.name,
avg(s.price)
```

Chaining – Connecting Subqueries

- MATCH-WHERE-RETURN can be extended to a pipeline.
- MATCH-WHERE-(WITH-MATCH-WHERE)*-RETURN
- WITH clause is the connection from one stage to the next.
  - Specifies variables/functions to pass on, and give them aliases to refer to.
- E.g.,
  - … **WITH** beer.name **AS** beer, COUNT(*) **AS** countSales, AVG(s.price) **AS** avgPrice

Chained Query Examples

- *Q1: Find the highest price of beers on the market* ***and the most expensive beers with that price.***

- *Q2: Find the average price of each beer brewed by “AB InBev”* ***if it is sold at multiple bars.***

Q1

```
Match() - [s:sells] - () return max(s.price) as maxPrice match ()-
[s:sells] -> (beer:Beer) where s.price = max.price
return beer.name, s.price
```

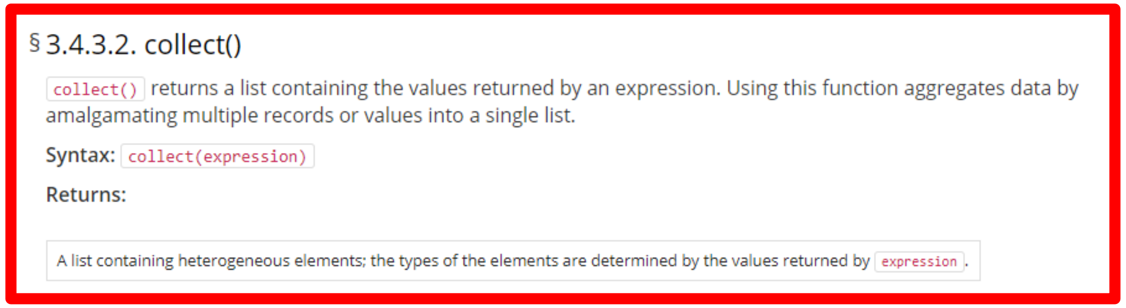
Q2

```
Match() - [s:sells] - (beer: Beer {brewer : “Ab InBev”}) with
beer.name as beer, avg(s.price) as avgPrice, count(*) as countSales
where countSales >= 2
return beer, avgPrice
```

Food for Thought

In Neo4j Cypher, there is an interesting aggregate function COLLECT– which gathers elements of a group into a list.
Can you imagine how it may be useful?

Hint: There is an UNWIND clause (like in MongoDB) to flatten a list, as the reverse of COLLECT, to access/use each element collected.



§ 3.4.3.2. collect()

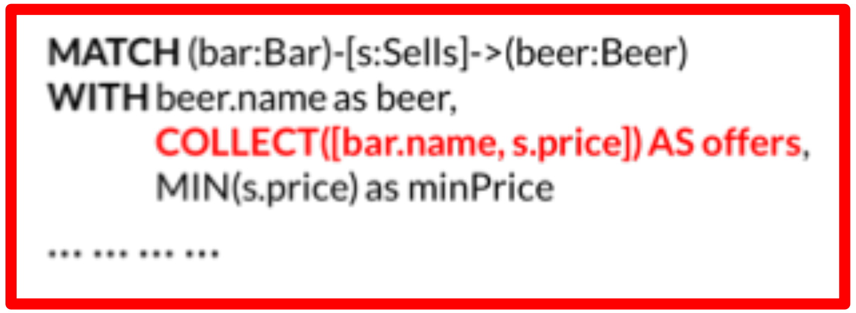
`collect()` returns a list containing the values returned by an expression. Using this function aggregates data by amalgamating multiple records or values into a single list.

Syntax: `collect(expression)`

Returns:

A list containing heterogeneous elements; the types of the elements are determined by the values returned by `expression`.

Cypher Manual 2017. Retrieved from https://neo4j.com/docs/developer-manual



```
MATCH (bar:Bar)-[s:Sells]->(beer:Beer)
WITH beer.name as beer,
        COLLECT([bar.name, s.price]) AS offers,
        MIN(s.price) as minPrice
... ... ... ...
```

Example query using COLLECT()