

# Neo4j I

HIGHER DIPLOMA IN DATA ANALYTICS



# Why NoSQL databases?



# Why NoSQL databases?

Scalability



# Why NoSQL databases?

Scalability



# Why NoSQL databases?

Scalability



Scale Up



# Why NoSQL databases?

Scalability



Scale Up



Scale Out

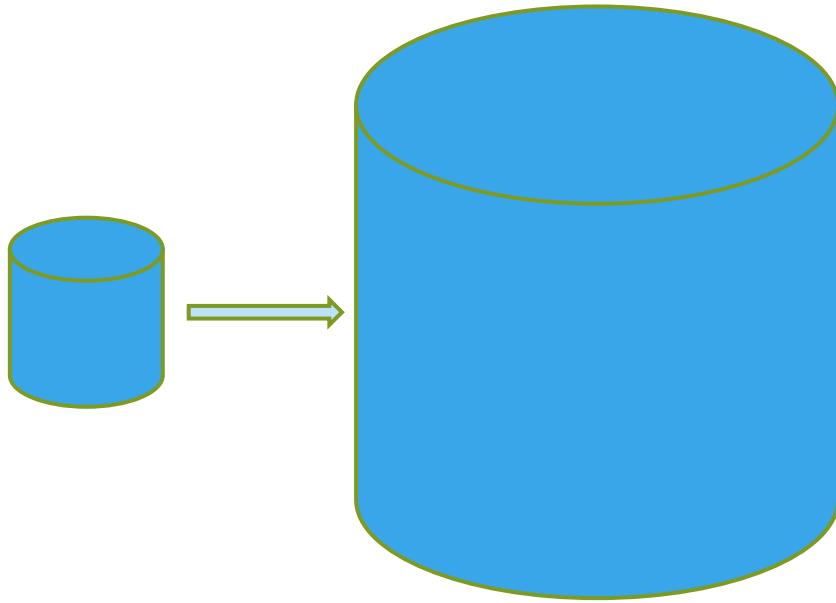


# Scalability



# Scalability

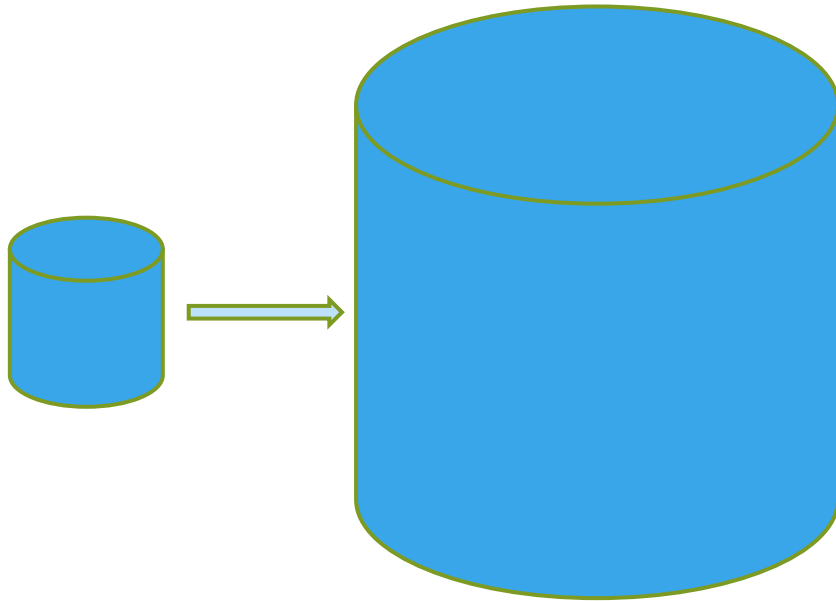
## ► Scale Up/Vertically



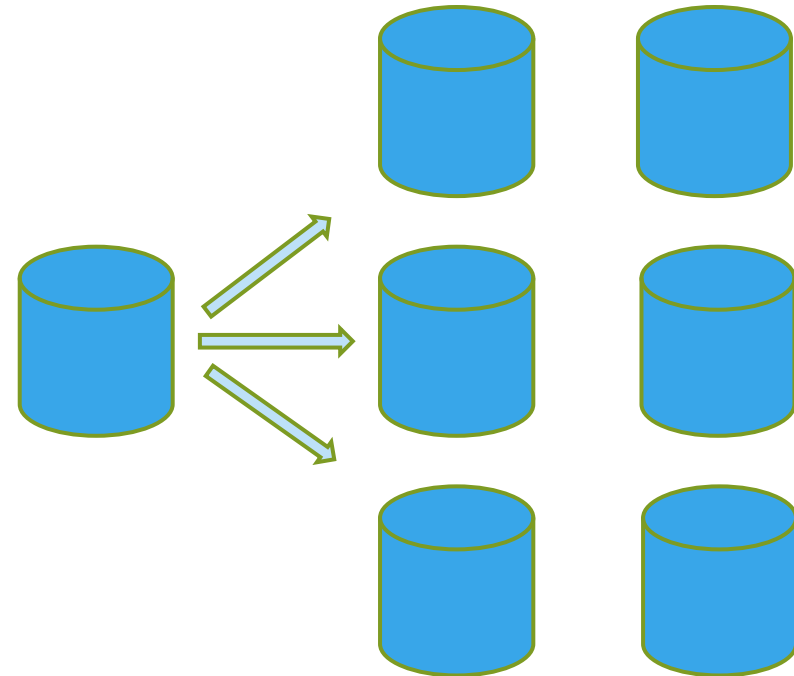


# Scalability

## ► Scale Up/Vertically

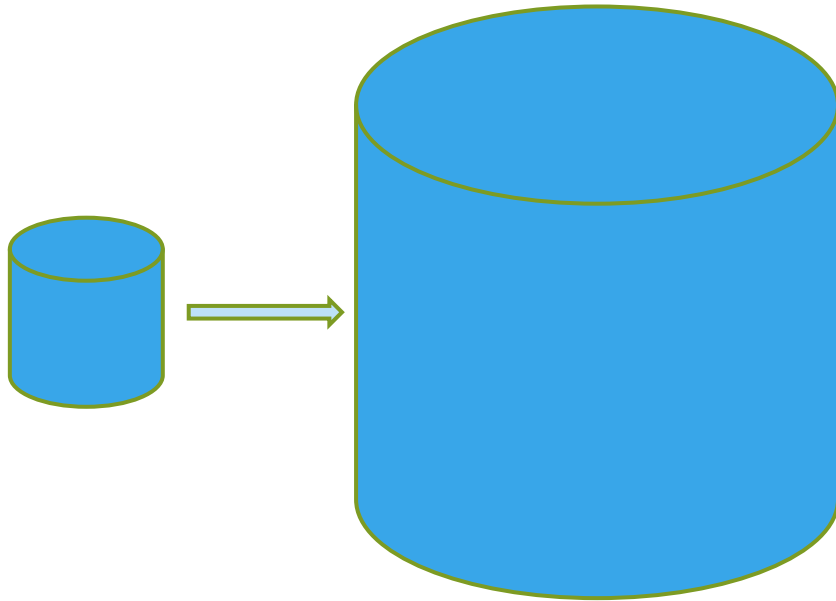


## ► Scale Out/Horizontally

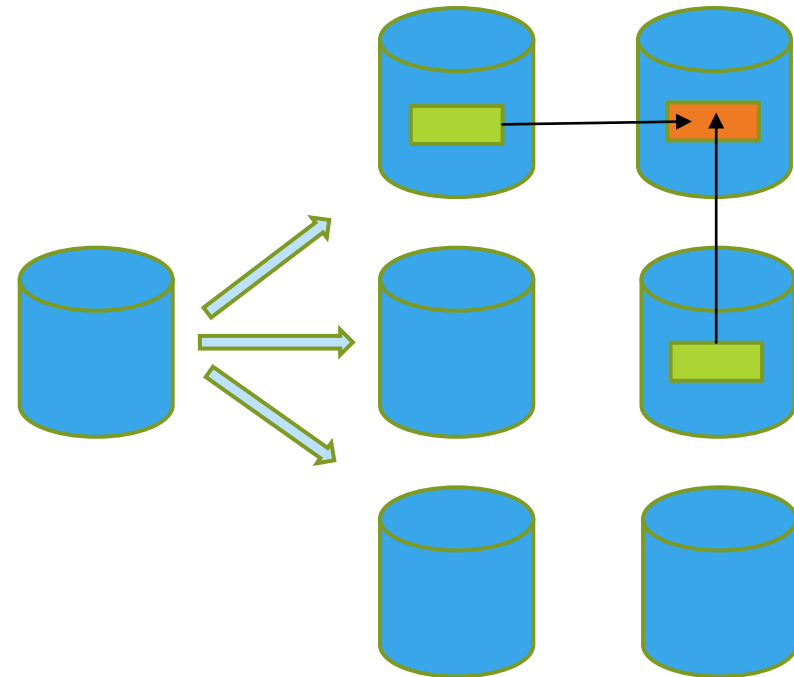


# Scalability

## ► Scale Up/Vertically

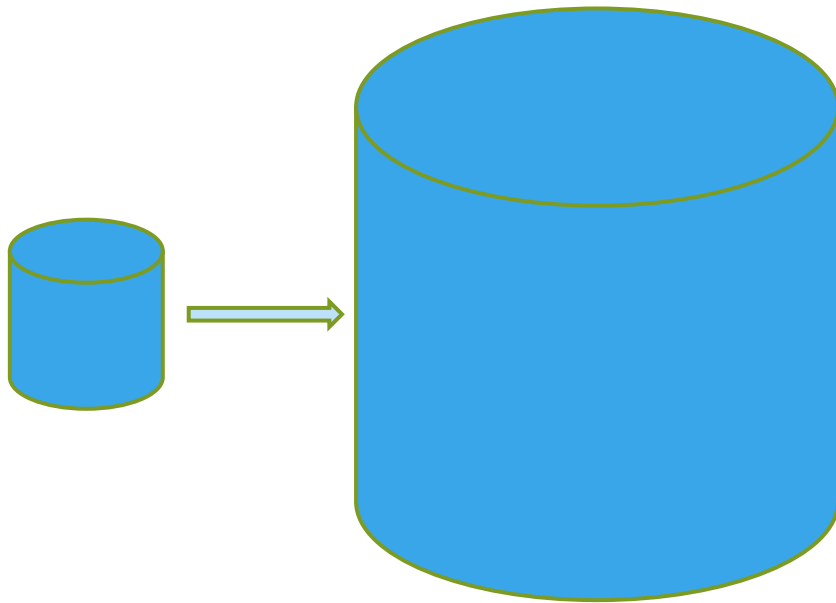


## ► Scale Out/Horizontally

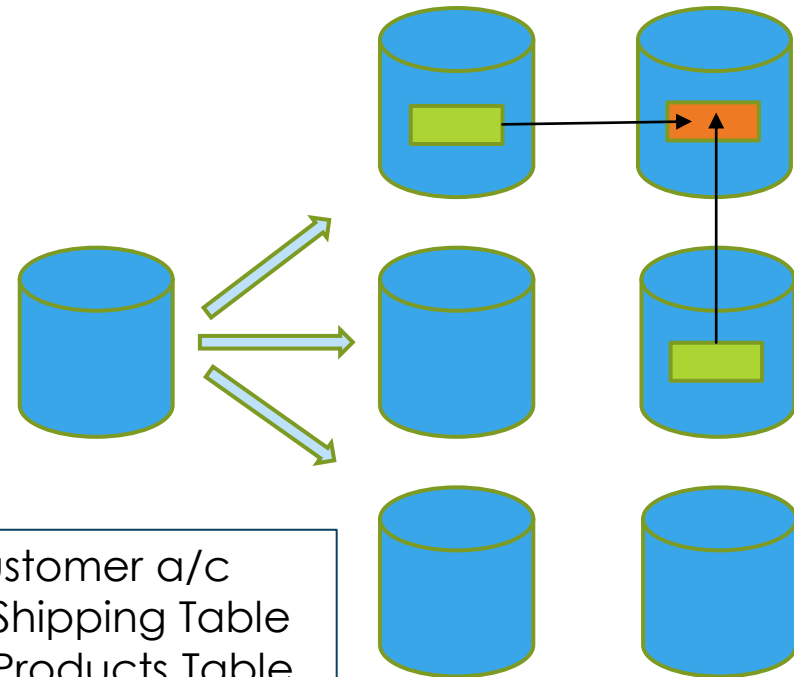


# Scalability

## ► Scale Up/Vertically



## ► Scale Out/Horizontally



- Debit Customer a/c
- Update Shipping Table
- Update Products Table
- Credit Store a/c



# Unstructured Data



# Unstructured Data

- ▶ CustomerID INTEGER
- ▶ Name VARCHAR(20)
- ▶ Phone VARCHAR(20)
- ▶ Address VARCHAR(50)



# Unstructured Data

- ▶ CustomerID INTEGER
- ▶ Name VARCHAR(20)
- ▶ Phone VARCHAR(20)
- ▶ Address VARCHAR(50)

CustomerID*	Name	Phone	Address
100	John	086 3304896	Tuam, Co. Galway
101	Alan	NULL	Athenry, Co. Galway
102	Mary	091 5688874	Galway, Co. Galway
103	Tom	090 6458959	Athlone, Co. Westmeath
104	Alice	094 1245763	Castlebar, Co. Mayo



# Unstructured Data

- ▶ CustomerID INTEGER
- ▶ Name VARCHAR(20)
- ▶ Phone VARCHAR(20)
- ▶ Address VARCHAR(50)
- ▶ Email VARCHAR(50)

CustomerID*	Name	Phone	Address
100	John	086 3304896	Tuam, Co. Galway
101	Alan	NULL	Athenry, Co. Galway
102	Mary	091 5688874	Galway, Co. Galway
103	Tom	090 6458959	Athlone, Co. Westmeath
104	Alice	094 1245763	Castlebar, Co. Mayo



# Unstructured Data

- ▶ CustomerID INTEGER
- ▶ Name VARCHAR(20)
- ▶ Phone VARCHAR(20)
- ▶ Address VARCHAR(50)
- ▶ Email VARCHAR(50)

CustomerID*	Name	Phone	Address	Email
100	John	086 3304896	Tuam, Co. Galway	<u>John@gmail.com</u>
101	Alan	NULL	Athenry, Co. Galway	NULL
102	Mary	091 5688874	Galway, Co. Galway	<u>Mary@yahoo.com</u>
103	Tom	090 6458959	Athlone, Co. Westmeath	NULL
104	Alice	094 1245763	Castlebar, Co. Mayo	NULL





# Unstructured Data

- ▶ CustomerID INTEGER
- ▶ Name VARCHAR(20)
- ▶ Phone VARCHAR(20)
- ▶ Address VARCHAR(50)
- ▶ Email VARCHAR(50)
- ▶ Twitter VARCHAR(50)

CustomerID*	Name	Phone	Address	Email
100	John	086 3304896	Tuam, Co. Galway	<u>John@gmail.com</u>
101	Alan	NULL	Athenry, Co. Galway	NULL
102	Mary	091 5688874	Galway, Co. Galway	<u>Mary@yahoo.com</u>
103	Tom	090 6458959	Athlone, Co. Westmeath	NULL
104	Alice	094 1245763	Castlebar, Co. Mayo	NULL



# Unstructured Data

- ▶ CustomerID INTEGER
- ▶ Name VARCHAR(20)
- ▶ Phone VARCHAR(20)
- ▶ Address VARCHAR(50)
- ▶ Email VARCHAR(50)
- ▶ Twitter VARCHAR(50)

CustomerID*	Name	Phone	Address	Email	Twitter
100	John	086 3304896	Tuam, Co. Galway	<u>John@gmail.com</u>	@John123
101	Alan	NULL	Athenry, Co. Galway	NULL	NULL
102	Mary	091 5688874	Galway, Co. Galway	<u>Mary@yahoo.com</u>	NULL
103	Tom	090 6458959	Athlone, Co. Westmeath	NULL	NULL
104	Alice	094 1245763	Castlebar, Co. Mayo	NULL	@AliceC1965



# NoSQL Database Types



# JSON

6



# JSON

- ▶ JSON – JavaScript Object Notation



# JSON

- ▶ JSON – JavaScript Object Notation
- ▶ Lightweight data-interchange format



# JSON

- ▶ JSON – JavaScript Object Notation
- ▶ Lightweight data-interchange format
- ▶ Machine/Human readable



# JSON

- ▶ JSON – JavaScript Object Notation
- ▶ Lightweight data-interchange format
- ▶ Machine/Human readable
- ▶ Language independent





# JSON

- ▶ JSON – JavaScript Object Notation
- ▶ Lightweight data-interchange format
- ▶ Machine/Human readable
- ▶ Language independent
- ▶ JSON Structure:
  - ▶ Name/Value pairs
  - ▶ Ordered Lists



# JSON Datatypes



# JSON Datatypes

Number

```
{  
  "id" : 1  
}
```



# JSON Datatypes

Number

```
{  
  "id": 1  
}
```



# JSON Datatypes

Number

```
{  
  "id" : 1  
}
```



# JSON Datatypes

Number

```
{  
  "id" : 1  
}
```



# JSON Datatypes

Number

```
{  
  "id" : 1  
}
```

```
{  
  "id" : 3.14  
}
```



# JSON Datatypes

## Number

```
{  
  "id" : 1  
}
```

```
{  
  "id" : 3.14  
}
```

## String

```
{  
  "id" : 1,  
  "fname" : "John"  
}
```





# JSON Datatypes

## Number

```
{  
  "id" : 1  
}
```

```
{  
  "id" : 3.14  
}
```

## String

```
{  
  "id" : 1,  
  "fname" : "John"  
}
```

## Boolean

```
{  
  "reg" : "09-G-13",  
  "hybrid" : false  
}
```



# JSON Datatypes



# JSON Datatypes

Array

```
{  
  "student" : "G00257854",  
  "subjects" : ["Databases", "Java", "Mobile Apps"]  
}
```



# JSON Datatypes

## Objects

```
{  
  "student": "G00257854",  
  "address": {  
    "street": "Castle St",  
    "town": "Athenry",  
    "county": "Galway"  
  }  
}
```



# JSON Datatypes

## Objects

```
{  
  "student": "G00257854",  
  "address": {  
    "street": "Castle St",  
    "town": "Athenry",  
    "county": "Galway"  
  }  
}
```



# JSON Datatypes

## Objects

```
{  
  "student": "G00257854",  
  "address": {  
    "street": "Castle St",  
    "town": "Athenry",  
    "county": "Galway"  
  }  
}
```



# JSON Datatypes

## Objects

```
{  
  "student": "G00257854",  
  "address": {  
    "street": "Castle St",  
    "town": "Athenry",  
    "county": "Galway"  
  }  
}
```



# JSON Datatypes

## Objects

```
{  
  "student": "G00257854",  
  "address": {  
    "street": "Castle St",  
    "town": "Athenry",  
    "county": "Galway"  
  }  
}
```





# JSON Datatypes

## Objects

```
{  
  "student": "G00257854",  
  "address": {  
    "street": "Castle St",  
    "town": "Athenry",  
    "county": "Galway"  
  }  
}
```



# Graphs



# Graphs

- ▶ In Mathematical terms, a Graph is a collection of elements - typically called **Nodes** (also called Vertices or Points) - that are joined together by **Edges**.



# Graphs

- ▶ In Mathematical terms, a Graph is a collection of elements - typically called **Nodes** (also called Vertices or Points) - that are joined together by **Edges**.
- ▶ Each Node represents some piece of information in the Graph.



# Graphs

- ▶ In Mathematical terms, a Graph is a collection of elements - typically called **Nodes** (also called Vertices or Points) - that are joined together by **Edges**.
- ▶ Each Node represents some piece of information in the Graph.
- ▶ Each Edge represents some connection between two Nodes.

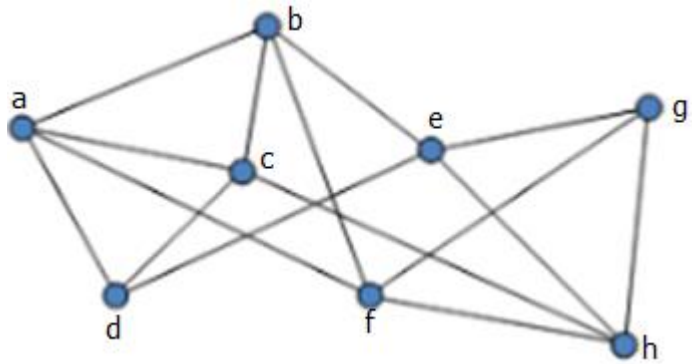


# Graphs

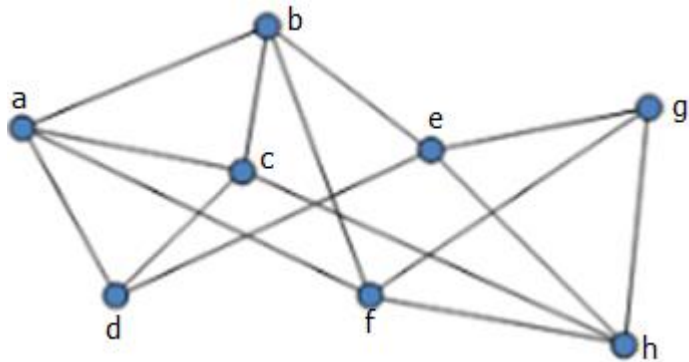
- ▶ In Mathematical terms, a Graph is a collection of elements - typically called **Nodes** (also called Vertices or Points) - that are joined together by **Edges**.
- ▶ Each Node represents some piece of information in the Graph.
- ▶ Each Edge represents some connection between two Nodes.
- ▶ Graphs are a common method to visually illustrate relationships in the data.



# Graphs



# Graphs

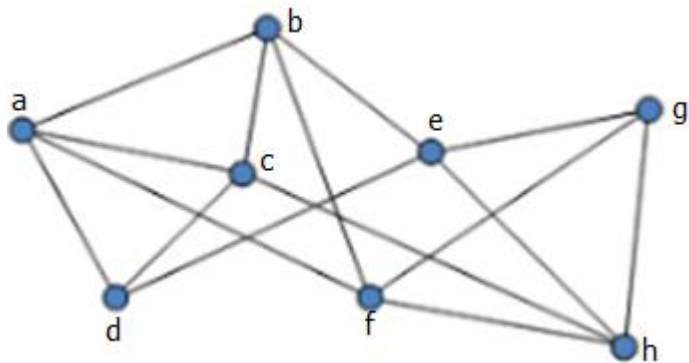


$V = \{a, b, c, d, e, f, g, h\}$





# Graphs

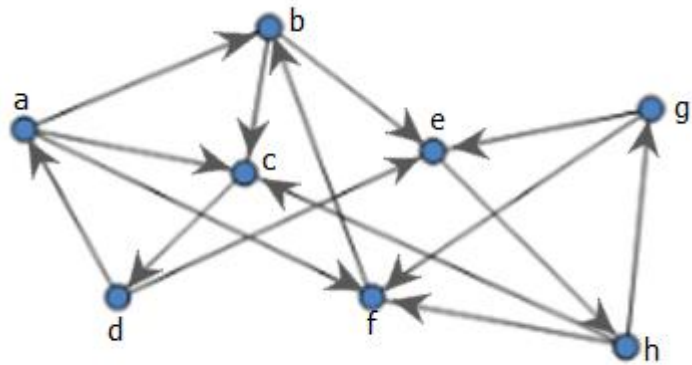


$V = \{a, b, c, d, e, f, g, h\}$

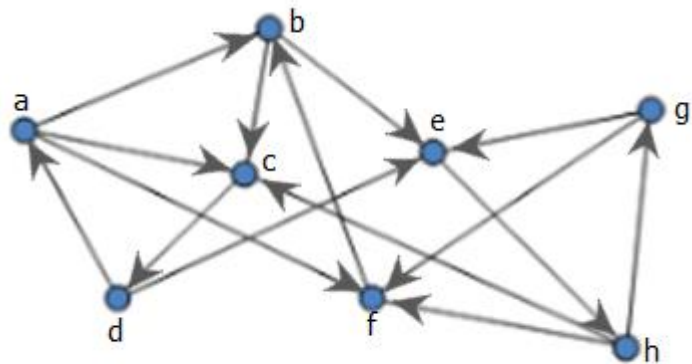
$E = \{\mathbf{ab}, \mathbf{ba}, ac, ca, ad, da, af, fa, bc, cb, be, eb, bf, fb, cd, dc, ch, hc, de, ed, eg, ge, eh, he, fg, gf, fh, hf, gh, hg\}$



# Graphs



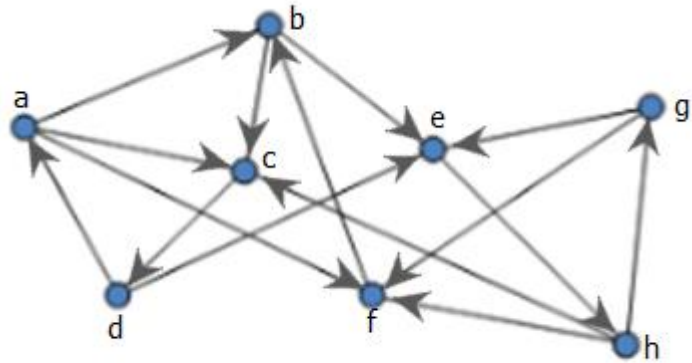
# Graphs



$V = \{a, b, c, d, e, f, g, h\}$



# Graphs



$V = \{a, b, c, d, e, f, g, h\}$

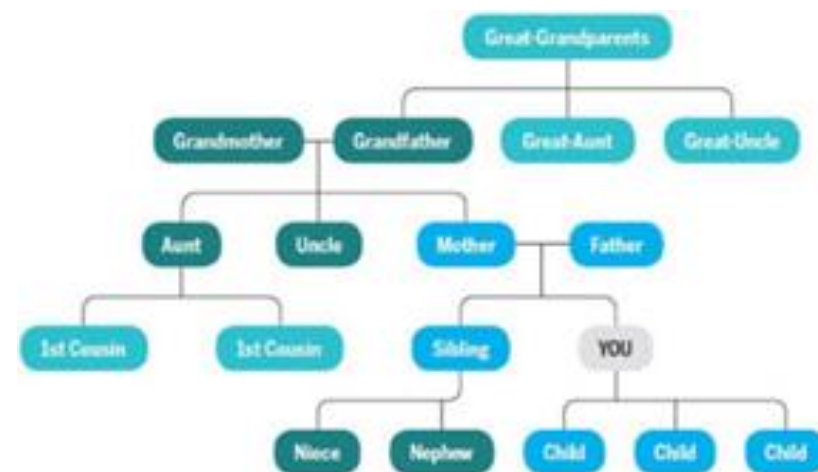
$E = \{ab, ac, af, bc, be, cd, da, de, eh, fb, ge, gf, hc, hf, hg\}$



# Graphs



# Graphs



# Why Graph Databases?



# Why Graph Databases?

- ▶ Intuitive





# Why Graph Databases?

## ► Intuitive

*name: John*

*joined: 2012-04-12*  
*tweetCount: 425*

*name: Bill*

*joined: 2021-01-12*  
*tweetCount: 15561*

*name: Alan*

*joined: 2012-04-05*  
*tweetCount: 1325*

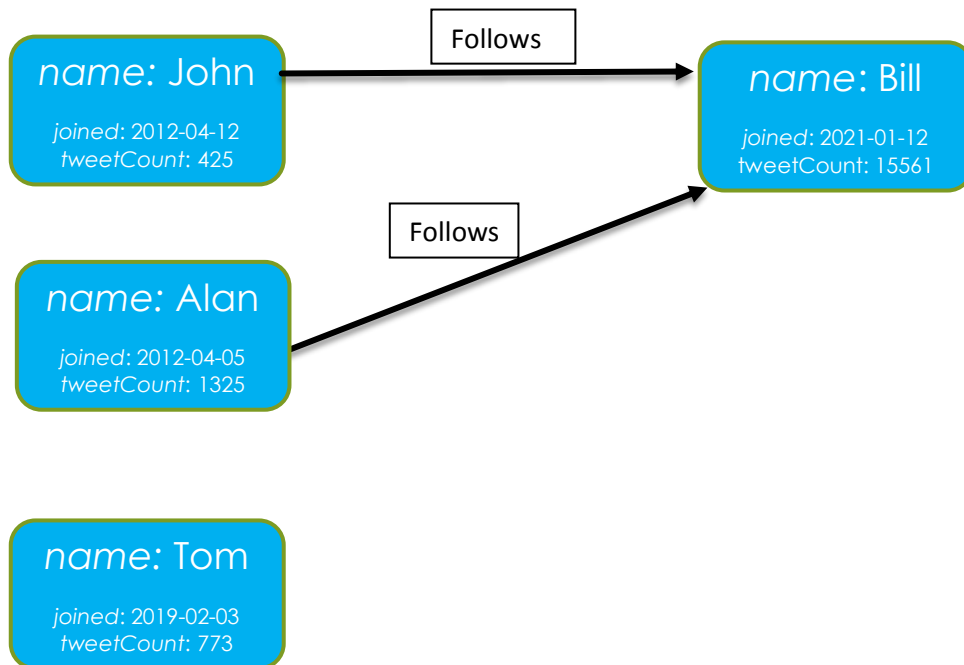
*name: Tom*

*joined: 2019-02-03*  
*tweetCount: 773*



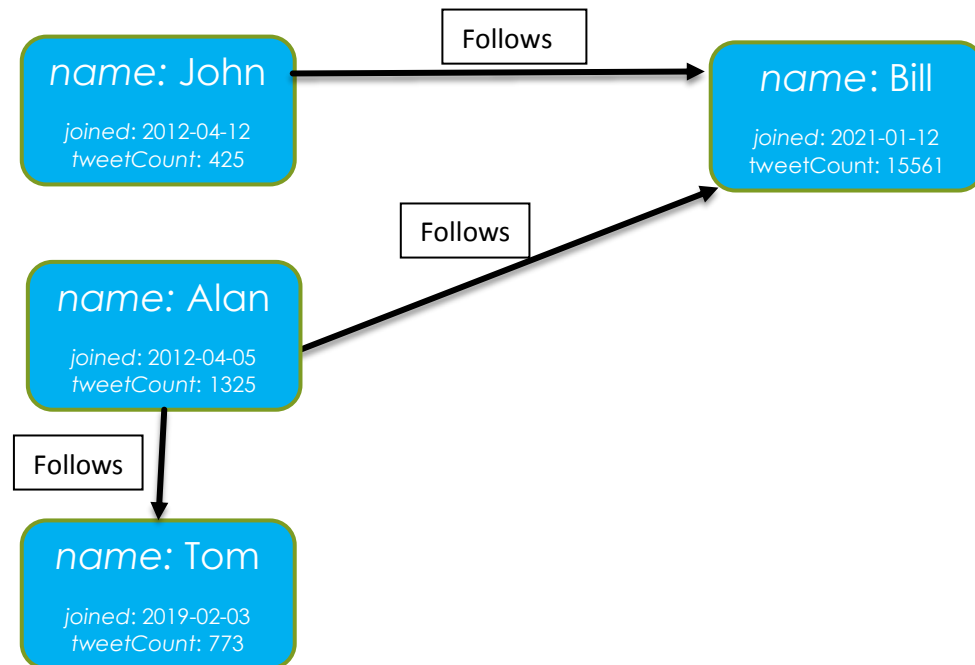
# Why Graph Databases?

## ► Intuitive



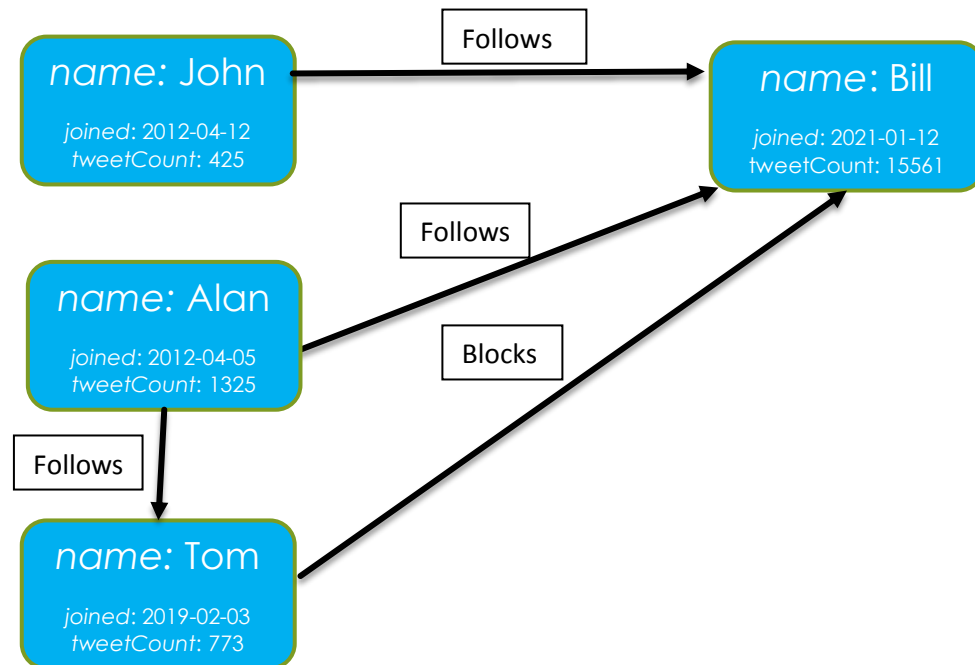
# Why Graph Databases?

## ► Intuitive



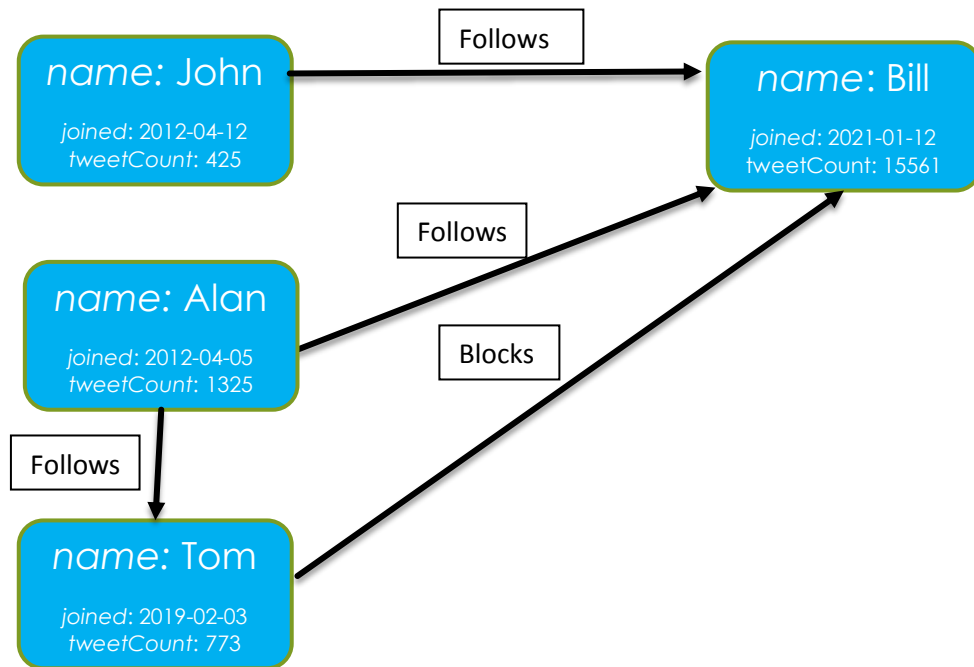
# Why Graph Databases?

## ► Intuitive



# Why Graph Databases?

## ► Intuitive

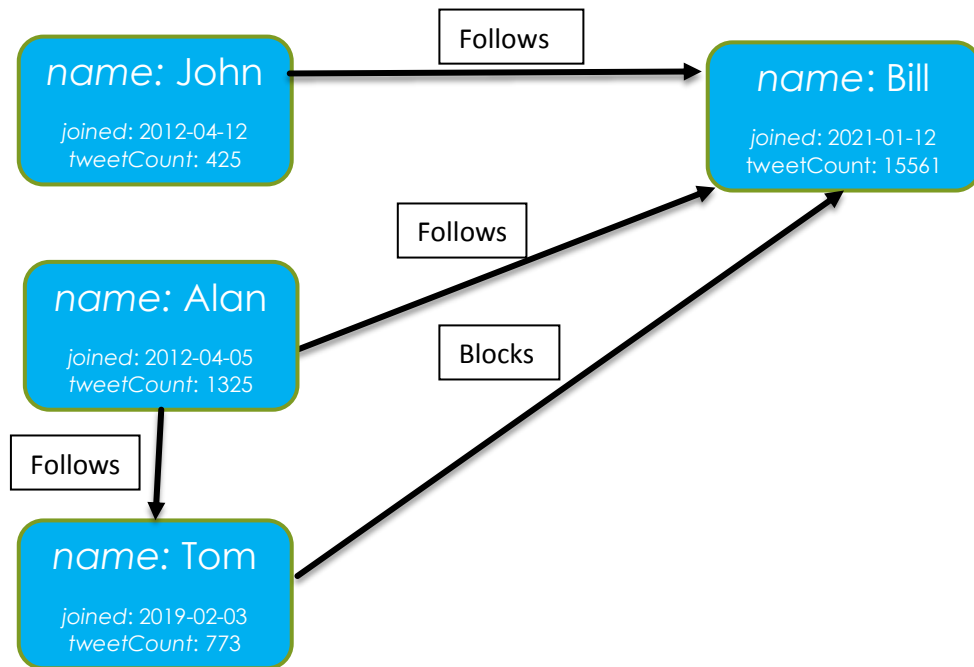


User			
UID	name	joined	Tweet count
100	Tom	2019-02-03	773
101	Alan	2012-04-05	1325
102	John	2012-04-12	425
103	Bill	2021-01-12	15561



# Why Graph Databases?

## ► Intuitive



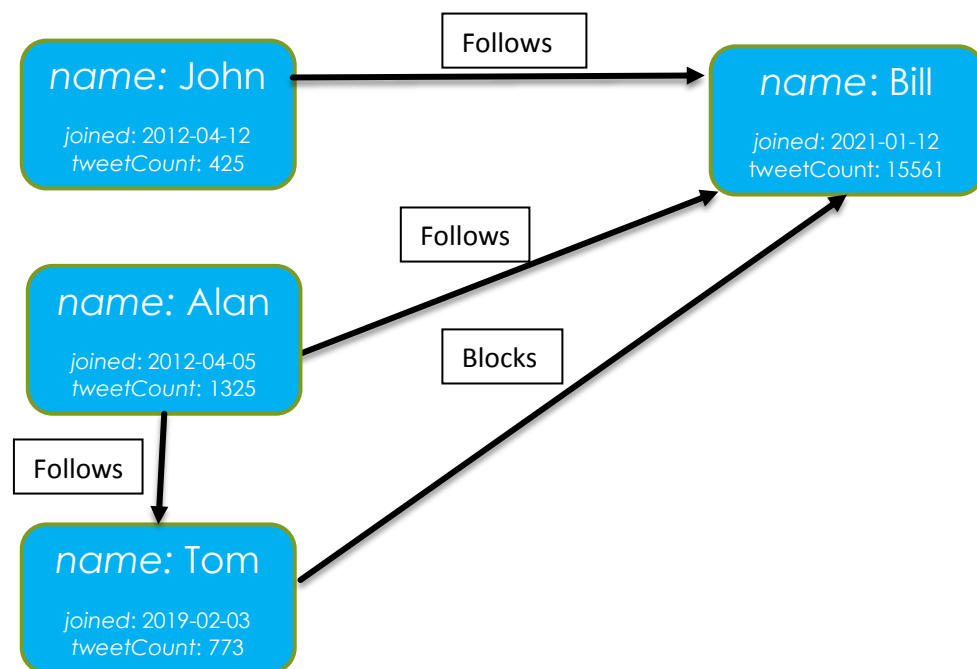
User			
UID	name	joined	Tweet count
100	Tom	2019-02-03	773
101	Alan	2012-04-05	1325
102	John	2012-04-12	425
103	Bill	2021-01-12	15561

Relationship	
RID	name
R1	Follows
R2	Blocks



# Why Graph Databases?

## ► Intuitive



User			
UID	name	joined	Tweet count
100	Tom	2019-02-03	773
101	Alan	2012-04-05	1325
102	John	2012-04-12	425
103	Bill	2021-01-12	15561

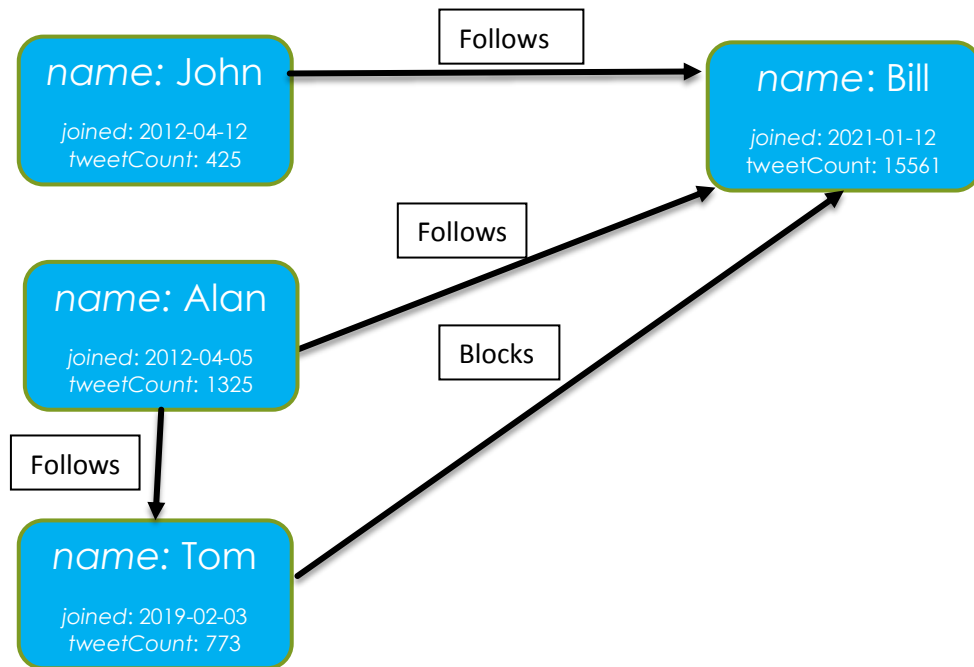
Relationship	
RID	name
R1	Follows
R2	Blocks

User Relationship Table		
User1 ID	User 2 ID	Relationship ID
100	103	R2
101	103	R1
101	100	R1
102	103	R1



# Why Graph Databases?

## ► Intuitive



User			
UID	name	joined	Tweet count
100	Tom	2019-02-03	773
101	Alan	2012-04-05	1325
102	John	2012-04-12	425
103	Bill	2021-01-12	15561

Relationship	
RID	name
R1	Follows
R2	Blocks

User Relationship Table		
User1 ID	User 2 ID	Relationship ID
100	103	R2
101	103	R1
101	100	R1
102	103	R1

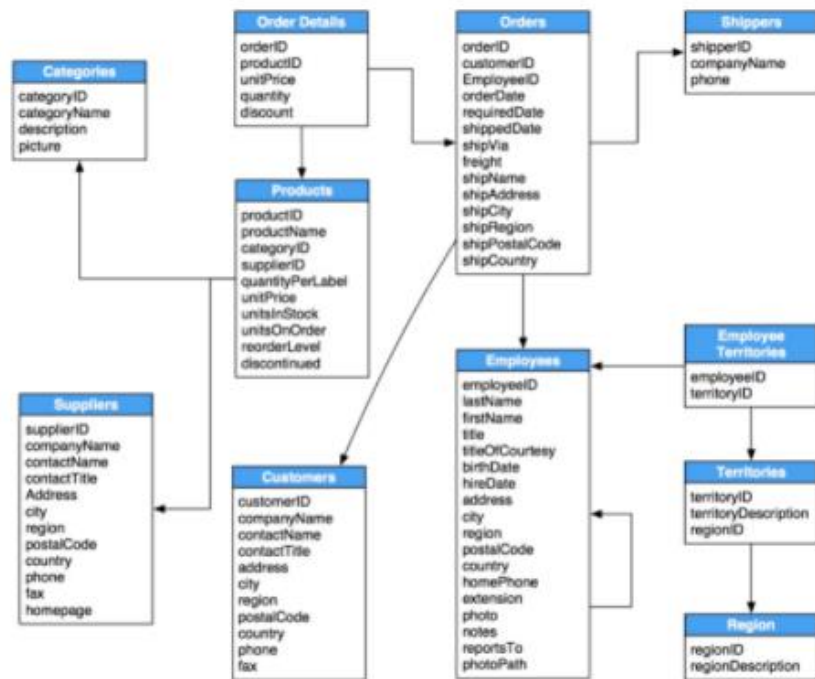




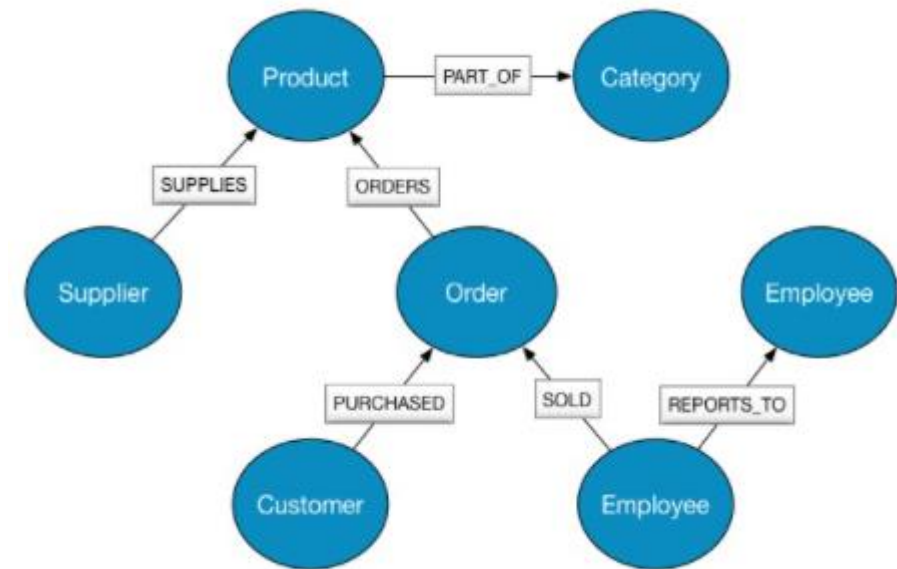
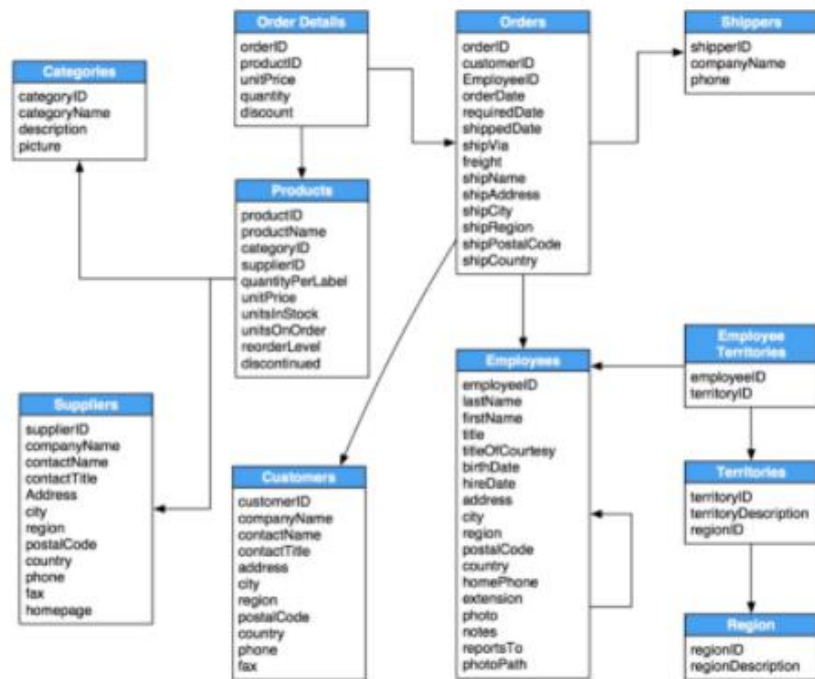
# Why Graph Databases?



# Why Graph Databases?



# Why Graph Databases?



# Why Graph Databases?

- ▶ Relationships are First-Class Citizens
  - ▶ A First-Class citizen is an entity that has an identity independent of any other item.
  - ▶ The identity allows the item to persist when its attributes change.
  - ▶ The identity allows other items to claim relationships with the item.
  - ▶ In a relational database First-Class citizens are entities or “things”, but not the relationships between them.



# Why Graph Databases?

- ▶ Relationships are First-Class Citizens
  - ▶ A First-Class citizen is an entity that has an identity independent of any other item.
  - ▶ The identity allows the item to persist when its attributes change.
  - ▶ The identity allows other items to claim relationships with the item.
  - ▶ In a relational database First-Class citizens are entities or “things”, but not the relationships between them.

User			
UID	name	joined	Tweet count
100	Tom	2019-02-03	773
101	Alan	2012-04-05	1325
102	John	2012-04-12	425
103	Bill	2021-01-12	15561

Relationship	
RID	name
R1	Follows
R2	Blocks

User Relationship Table		
User1 ID	User 2 ID	Relationship ID
100	103	R2
101	103	R1
101	100	R1
102	103	R1



# Why Graph Databases?

- ▶ Unstructured Data



# Why Graph Databases?

## ► Unstructured Data

Customer		
CID	Name	Address
C001	John Smith	1 College Road, Galway
C002	Mary Flynn	16 The Avenue, Tuam
C003	Bill Murphy	Church Road, Mallow, Cork



# Why Graph Databases?

## ► Unstructured Data

Customer			
CID	Name	Address	email
C001	John Smith	1 College Road, Galway	john@gmail.com
C002	Mary Flynn	16 The Avenue, Tuam	NULL
C003	Bill Murphy	Church Road, Mallow, Cork	bm1@gmail.com





# Why Graph Databases?

## ► Unstructured Data

Customer				
CID	Name	Address	email	Messenger ID
C001	John Smith	1 College Road, Galway	john@gmail.com	NULL
C002	Mary Flynn	16 The Avenue, Tuam	NULL	NULL
C003	Bill Murphy	Church Road, Mallow, Cork	bm1@gmail.com	billmurphy173



# Neo4j

- ▶ Neo4j is a popular Graph Database.



# Neo4j

- ▶ Neo4j is a popular Graph Database.
  - ▶ Flexible Schema (Schemaless).



# Neo4j

- ▶ Neo4j is a popular Graph Database.
  - ▶ Flexible Schema (Schemaless).
  - ▶ ACID.



# Neo4j

- ▶ Neo4j is a popular Graph Database.
  - ▶ Flexible Schema (Schemaless).
  - ▶ ACID.
  - ▶ Cypher Query Language.



# Neo4j

- ▶ Neo4j is a popular Graph Database.
  - ▶ Flexible Schema (Schemaless).
  - ▶ ACID.
  - ▶ Cypher Query Language.
  - ▶ Integration with several languages.



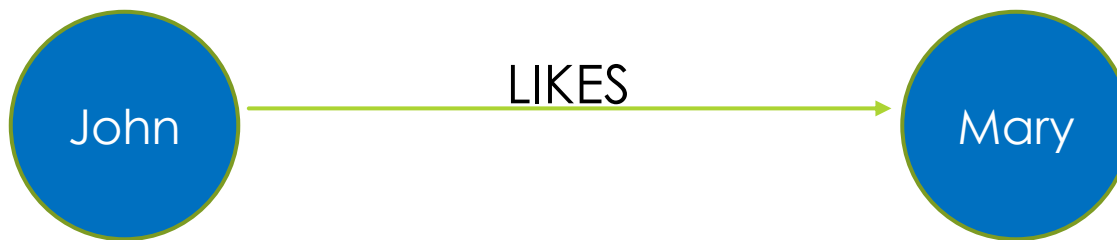
# Neo4j

- ▶ Graphs have:
  - ▶ Nodes (Vertices)
  - ▶ Relationships (Edges)



# Neo4j

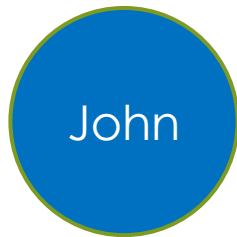
- ▶ Graphs have:
  - ▶ Nodes (Vertices)
  - ▶ Relationships (Edges)





# Neoj4 - Node

- ▶ A *node* is the basic entity of the graph, with the unique attribute of being able to exist in and of itself.
- ▶ A node may be assigned a set of unique labels.
- ▶ A node may have properties.
- ▶ A node may have zero or more outgoing relationships.
- ▶ A node may have zero or more incoming relationships.



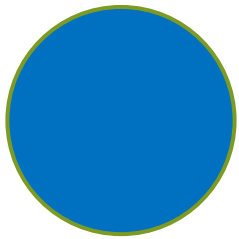
# Neoj4 - Label

- ▶ Labels are used to shape the domain by grouping nodes into sets where all nodes that have a certain label belongs to the same set.
- ▶ A node can have zero or many labels.



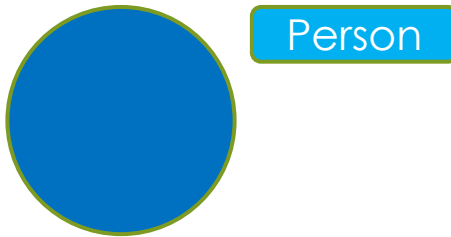
# Neoj4 - Label

- ▶ Labels are used to shape the domain by grouping nodes into sets where all nodes that have a certain label belongs to the same set.
- ▶ A node can have zero or many labels.



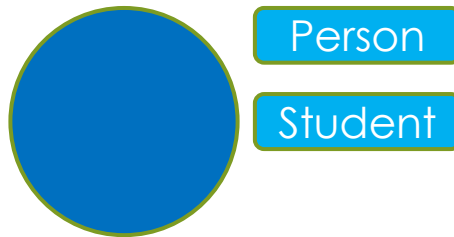
# Neoj4 - Label

- ▶ Labels are used to shape the domain by grouping nodes into sets where all nodes that have a certain label belongs to the same set.
- ▶ A node can have zero or many labels.



# Neoj4 - Label

- ▶ Labels are used to shape the domain by grouping nodes into sets where all nodes that have a certain label belongs to the same set.
- ▶ A node can have zero or many labels.



# Neo4j - Property

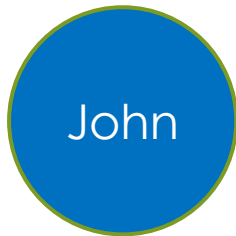
- ▶ Properties are name-value pairs that are used to add qualities to nodes.

```
<id>:1, name:"John", age:39
```



# Neo4j - Property

- ▶ Properties are name-value pairs that are used to add qualities to nodes.



`<id>:1, name:"John", age:39`



# Cypher

- ▶ Cypher is a declarative graph query language that allows for expressive and efficient querying and updating of the graph store.
- ▶ Focuses on *what* to retrieve from a graph, not *how* to retrieve it.
- ▶ Made up of *clauses*.





# Cypher - CREATE

- ▶ `CREATE()` – Creates a Node
- ▶ `CREATE(:Person)` – Creates a Node with the label Person
- ▶ `CREATE(:Person{name:"John"})` – Creates a Node with the label Person and a property key called name and a property value of “John”.



# Cypher - MATCH

- ▶ `MATCH(n) RETURN n` – Match all nodes in the database (and return them).
- ▶ `MATCH(p:Person) RETURN(p)` – Match all nodes in the database with the label Person.
- ▶ `MATCH(p:Person{name:"John"}) RETURN(p)` – Match all nodes in the database with the label Person and who have the following property:
  - ▶ key = name, value="John".



# Cypher - WHERE

- ▶ WHERE adds constraints to the patterns in a MATCH.
- ▶ `MATCH(p:Person{name:"John"}) RETURN(p)`
- ▶ `MATCH(p:Person)`  
`WHERE p.name="John"`  
`RETURN p`



# Cypher - WHERE

- ▶ WHERE adds constraints to the patterns in a MATCH.

- ▶ `MATCH(p:Person{name:"John"}) RETURN(p)`

- ▶ `MATCH(p:Person)`  
`WHERE p.name="John"`  
`RETURN p`

```
MATCH(p:Person)
WHERE p.name="John"
OR p.name="Tom"
RETURN p
```



# Cypher - Property Existence Checking

- ▶ Graph databases are good for storing less structured data.
- ▶ Only need to add a property to a node if necessary.
- ▶ May only be interested in nodes with/without specific properties.



Person

name: **Tom**  
age: 18  
height: 5'9.5"  
sex: M  
weight: 9st 2lb

Person

name: **Mary**  
age: 18  
hair: Brunette  
height: 5'6"  
sex: F  
weight: 8st 11lb

Person

name: **Alan**  
age: 44  
height: 6'1"  
sex: M

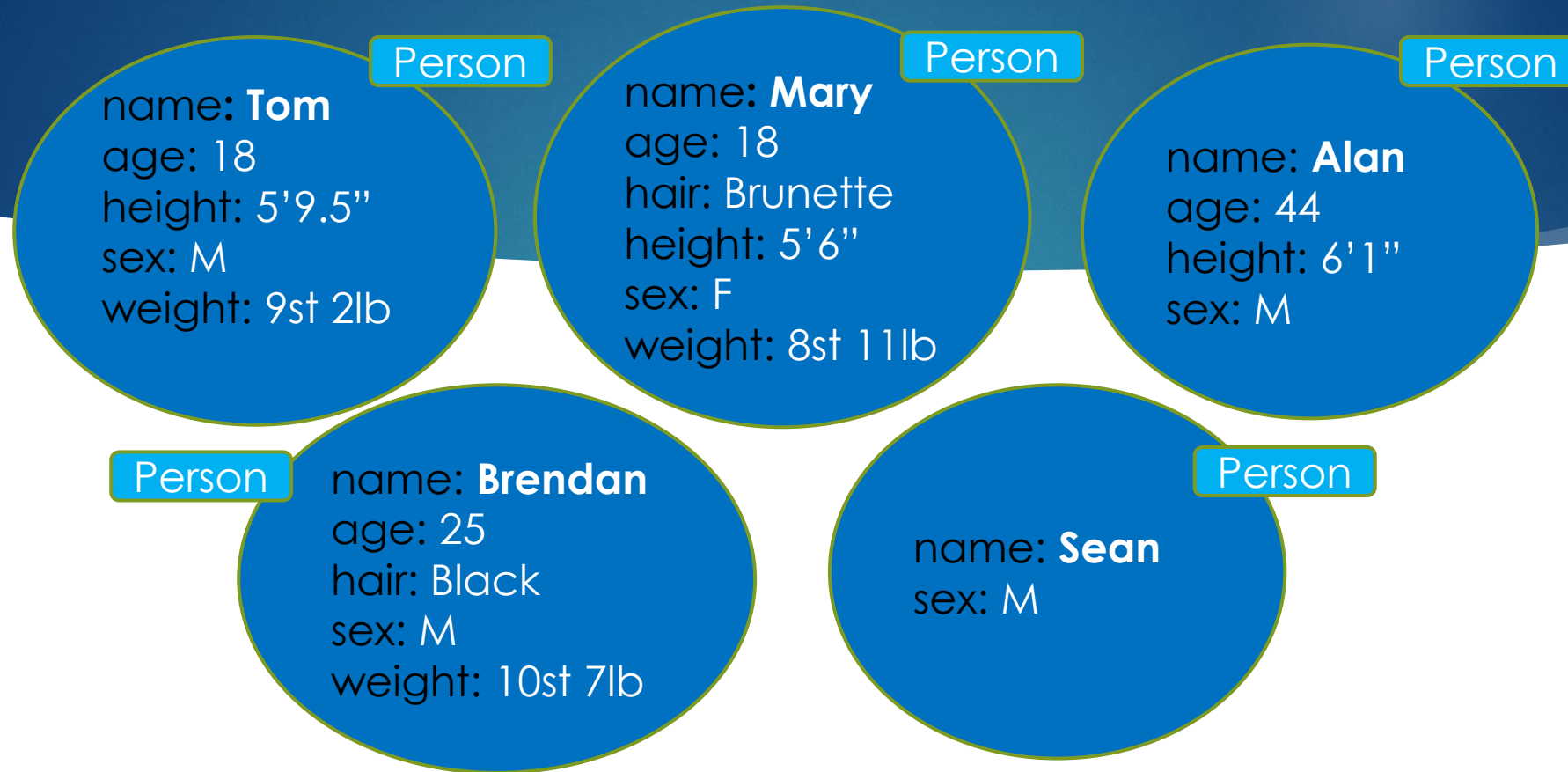
Person

name: **Brendan**  
age: 25  
hair: Black  
sex: M  
weight: 10st 7lb

Person

name: **Sean**  
sex: M

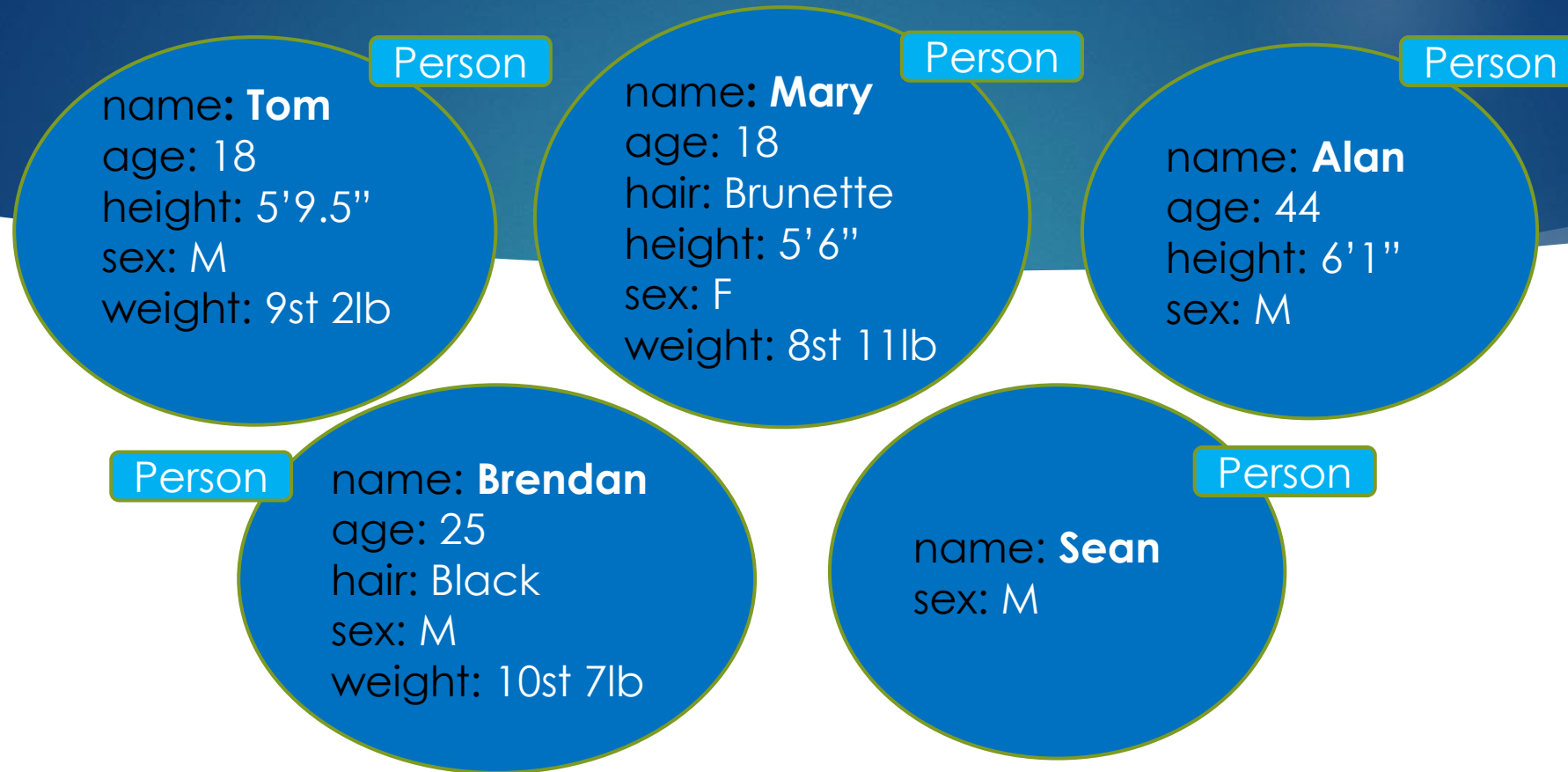




- Find all People who have a weight property:

```
MATCH(p:Person) WHERE p.weight IS NOT NULL RETURN p
```

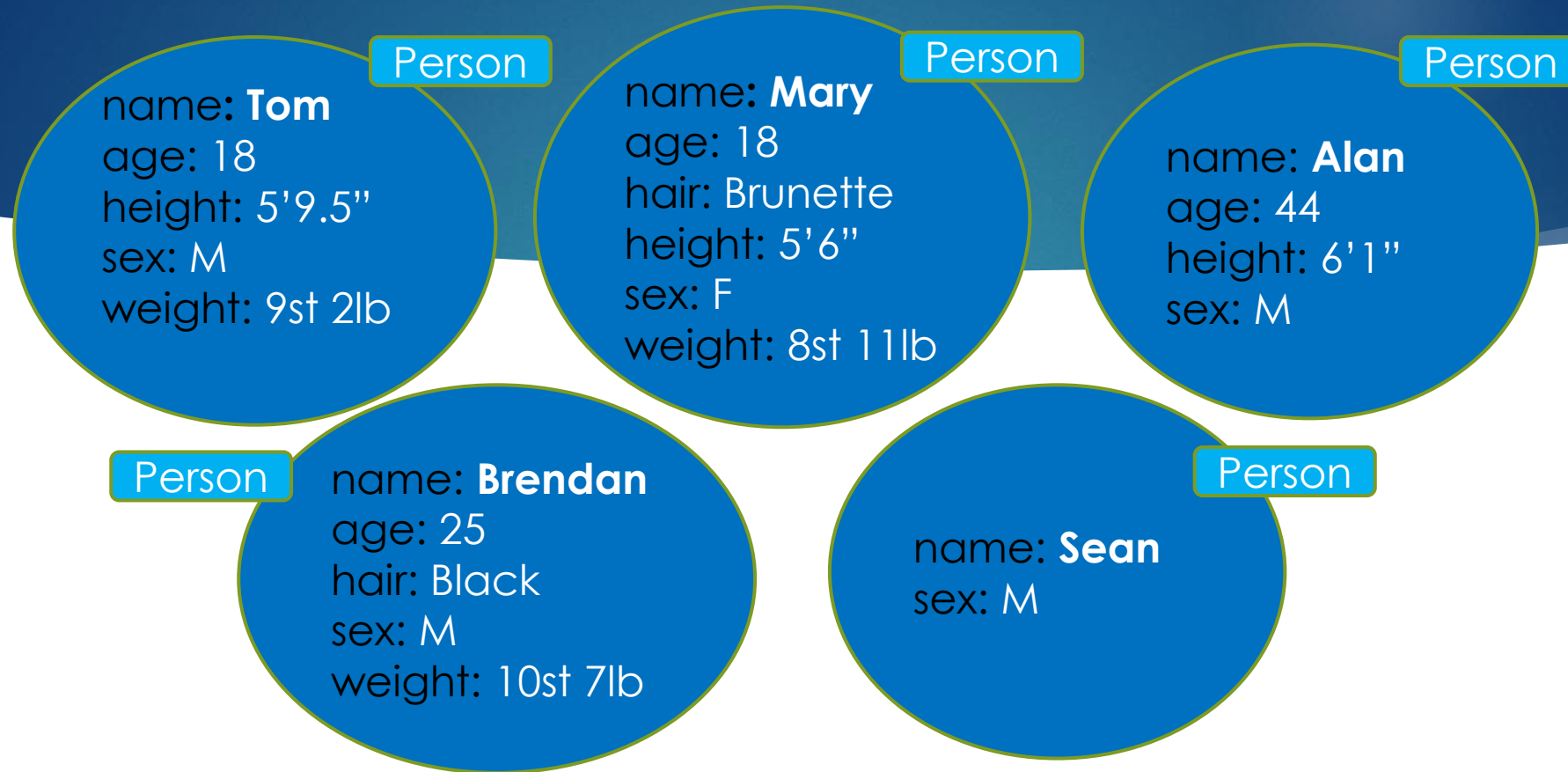




- Find all Males who have a weight property:





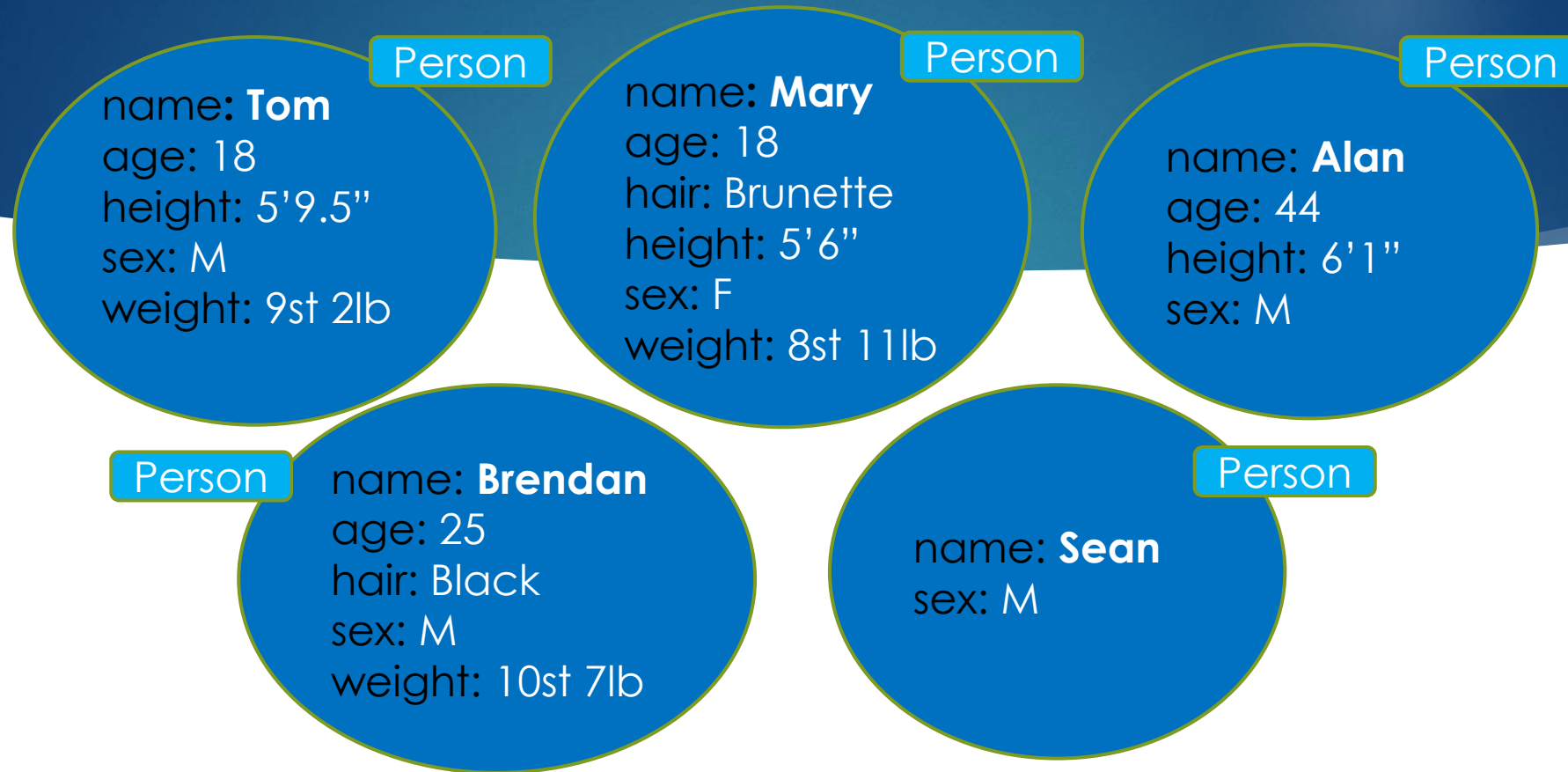


- Find all Males who have a weight property:

```
MATCH(p:Person{sex:"M"})  
WHERE p.weight IS NOT NULL  
RETURN p
```

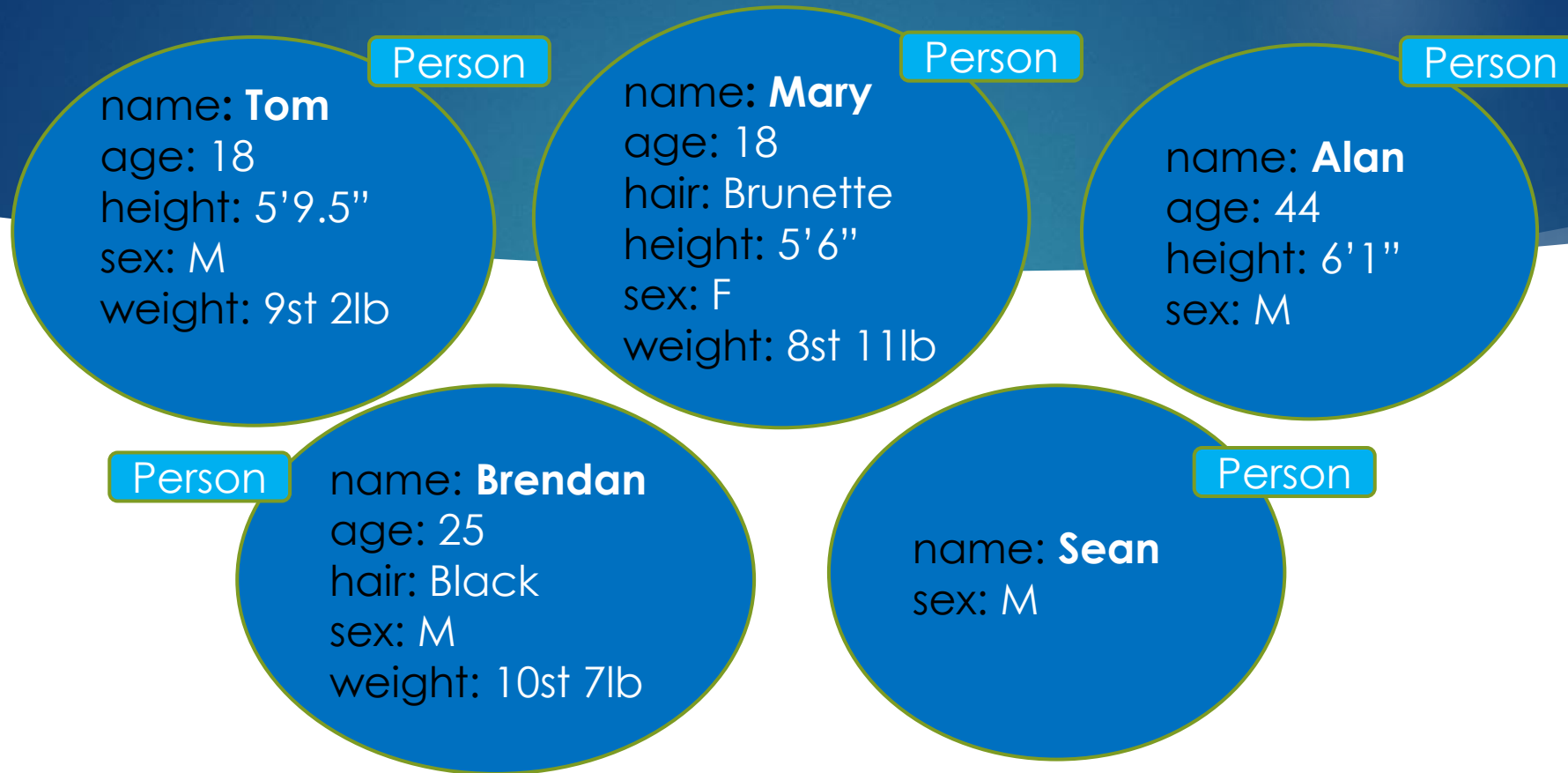
```
MATCH(p:Person)  
WHERE p.weight IS NOT NULL  
AND p.sex="M"  
RETURN p
```





- Find all Brendan's properties:





- Find all Brendan's properties:

```
MATCH (n:Person{name:"Brendan"}) RETURN keys(n)  
["hair", "name", "weight", "age", "sex"]
```



# Constraints

- ▶ A constraint ensures data integrity.



# Constraints

- ▶ A constraint ensures data integrity.
- ▶ `CREATE CONSTRAINT eid_unique ON (e:Employee) ASSERT e.eid IS UNIQUE`



# Constraints

- ▶ A constraint ensures data integrity.
- ▶ `CREATE CONSTRAINT eid_unique ON (e:Employee) ASSERT e.eid IS UNIQUE`

Employee

eid: **E001** ✓  
name: Tom Lawson  
salary: 55,992.92



# Constraints

- ▶ A constraint ensures data integrity.
- ▶ `CREATE CONSTRAINT eid_unique ON (e:Employee) ASSERT e.eid IS UNIQUE`

Employee

eid: **E001** ✓

name: Tom Lawson  
salary: 55,992.92

Employee

eid: **E001** ✗

name: Anne Lyons  
salary: 51,322.23



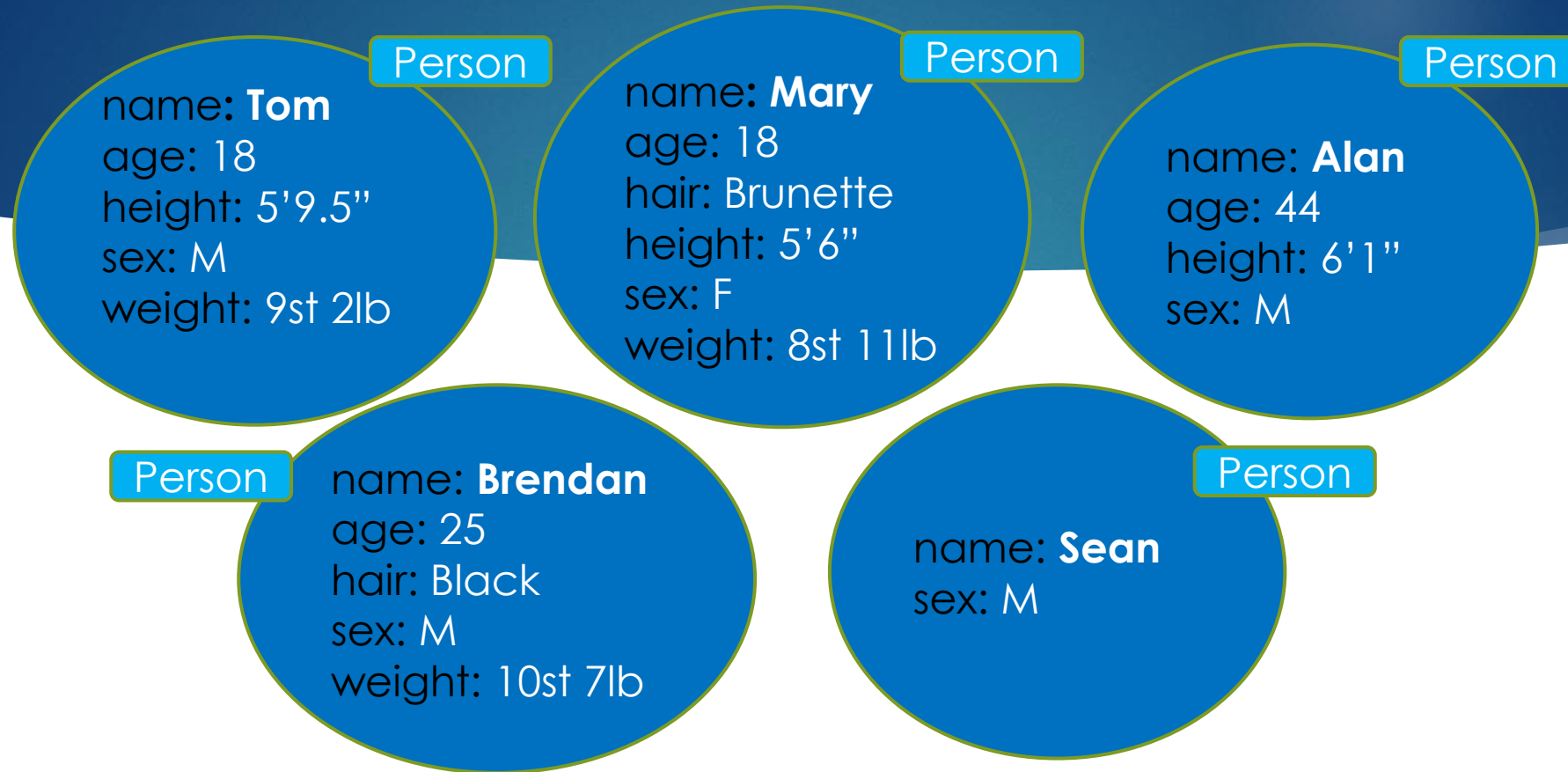
# Cypher – Aggregating Functions

- ▶ Aggregating functions take a set of values and calculate an aggregated value over them.

- ▶ avg()
- ▶ max()
- ▶ min()
- ▶ sum()





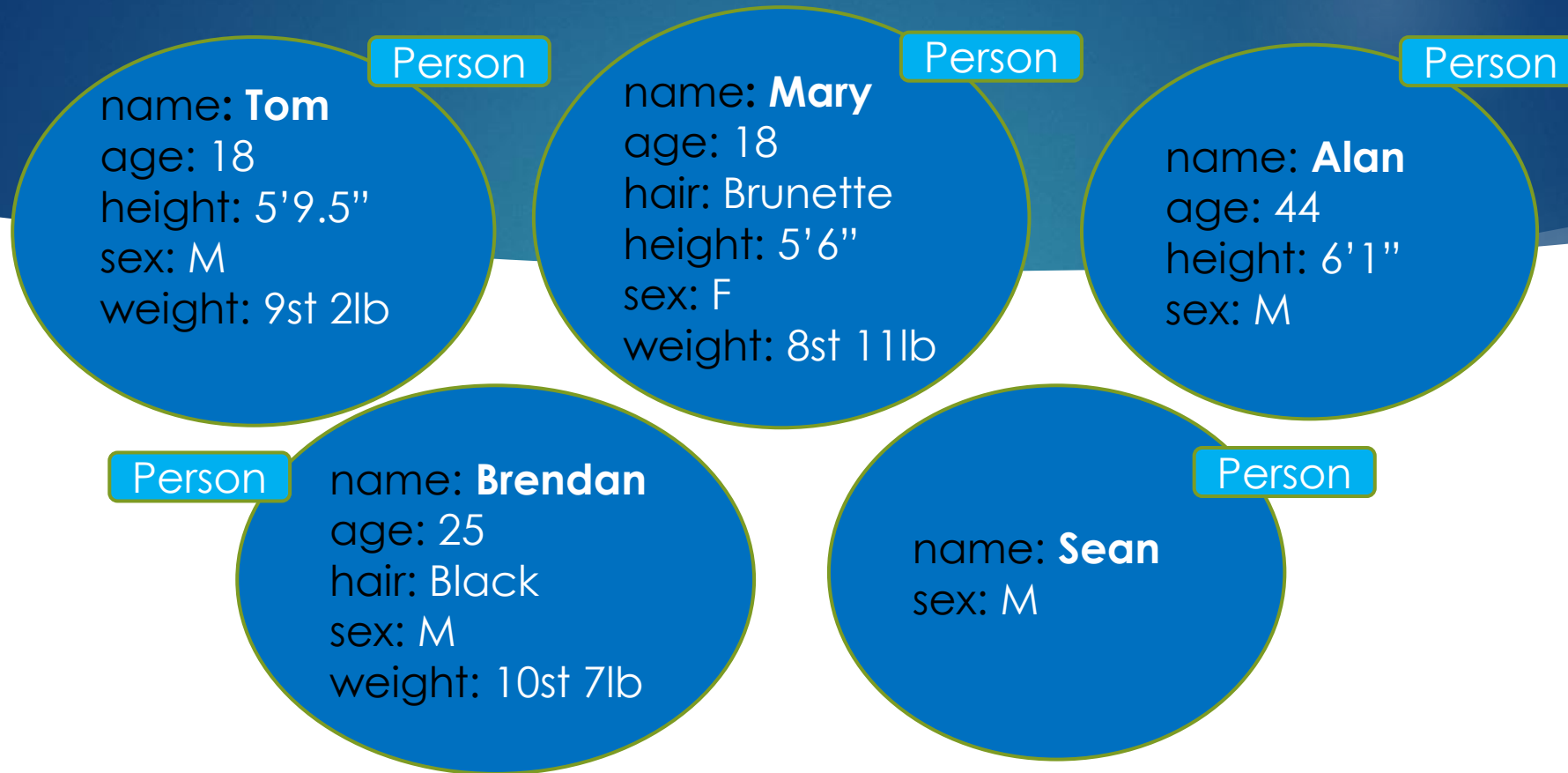


- Find the average age of Males:

```
MATCH(n{sex:"M"}) RETURN avg(n.age)
```

29.0





- Find the average age of Males and Females:

```
MATCH(n) RETURN n.sex, avg(n.age)
```

```
"M"    29.0
```

```
"F"    18.0
```



# Cypher - SET

- ▶ The SET clause is used to update labels on nodes and properties on nodes.



# Cypher - SET

- The SET clause is used to update labels on nodes and properties on nodes.

Person

name: **Brendan**  
age: 25  
hair: Black  
sex: M  
weight: 10st 7lb

Person

name: **Brendan**  
age: 26  
hair: Black  
sex: M  
weight: 10st 7lb

```
MATCH(n{name:"Brendan"})  
SET n.age = n.age+1  
RETURN n
```



# Cypher - SET

- ▶ The SET clause is used to update labels on nodes and properties on nodes.

Person

name: **Brendan**  
age: 25  
hair: Black  
sex: M  
weight: 10st 7lb

```
MATCH(n{name:"Brendan"})  
SET n.age = n.age+1  
RETURN n
```

Person

name: **Brendan**  
age: 26  
hair: Black  
sex: M  
weight: 10st 7lb

```
MATCH(n{name:"Brendan"})  
SET n.height = "6'1\""
```

Person

name: **Brendan**  
age: 26  
hair: Black  
sex: M  
weight: 10st 7lb  
height: 6'1"



# Cypher - REMOVE

- ▶ The REMOVE clause is used to remove labels from nodes and properties from nodes.



# Cypher - REMOVE

- The REMOVE clause is used to remove labels from nodes and properties from nodes.

Person

name: **Brendan**  
age: 26  
hair: Black  
sex: M  
weight: 10st 7lb  
height: 6'1"

Person

name: **Brendan**  
age: 26  
hair: Black  
sex: M  
weight: 10st 7lb

```
match(n{name:"Brendan"})  
remove n.height  
return n
```



# Cypher - REMOVE

- The REMOVE clause is used to remove labels from nodes and properties from nodes.

Person

name: **Brendan**  
age: 26  
hair: Black  
sex: M  
weight: 10st 7lb  
height: 6'1"

```
match(n{name:"Brendan"})  
remove n.height  
return n
```

Person

name: **Brendan**  
age: 26  
hair: Black  
sex: M  
weight: 10st 7lb

```
match(n{name:"Brendan"})  
remove n.height  
return n
```

Person

name: **Brendan**  
age: 26  
hair: Black  
sex: M  
weight: 10st 7lb





# Cypher - DELETE

- ▶ The DELETE clause is used to delete nodes, relationships or paths.

```
MATCH(p:Person) DELETE p
```

```
MATCH(p:Person) WHERE p.weight IS NULL DELETE p
```



# Cypher - MERGE

- ▶ The MERGE clause ensures that a pattern exists in the graph.
- ▶ Either the pattern already exists, or it needs to be created.



# Cypher - MERGE

- ▶ The MERGE clause ensures that a pattern exists in the graph.
- ▶ Either the pattern already exists, or it needs to be created.

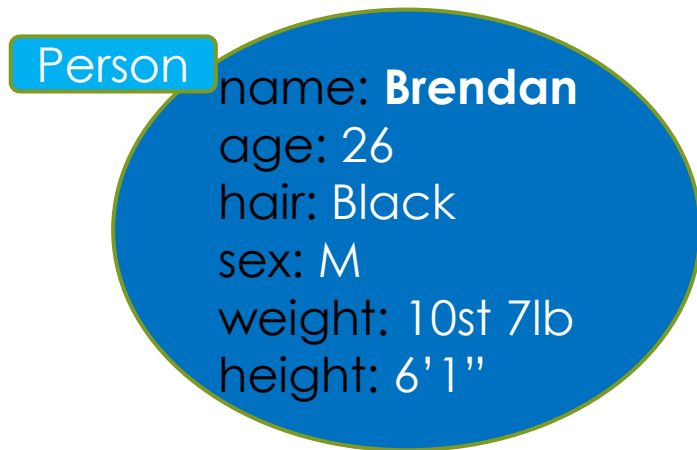
Person

name: **Brendan**  
age: 26  
hair: Black  
sex: M  
weight: 10st 7lb  
height: 6'1"



# Cypher - MERGE

- ▶ The MERGE clause ensures that a pattern exists in the graph.
- ▶ Either the pattern already exists, or it needs to be created.

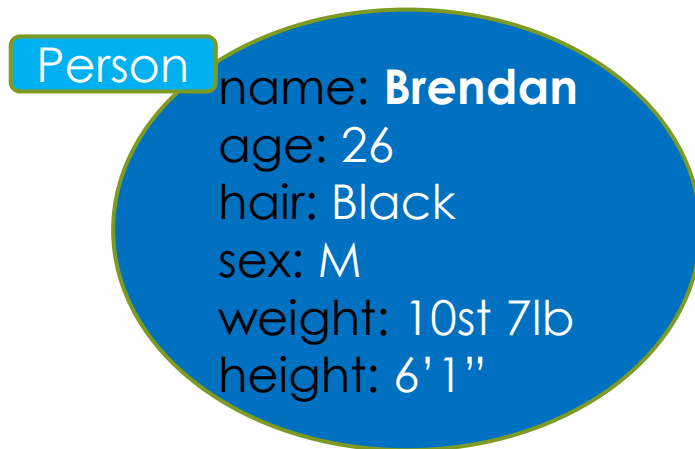


```
MERGE (p:Person{name:"Brendan"})  
RETURN p
```

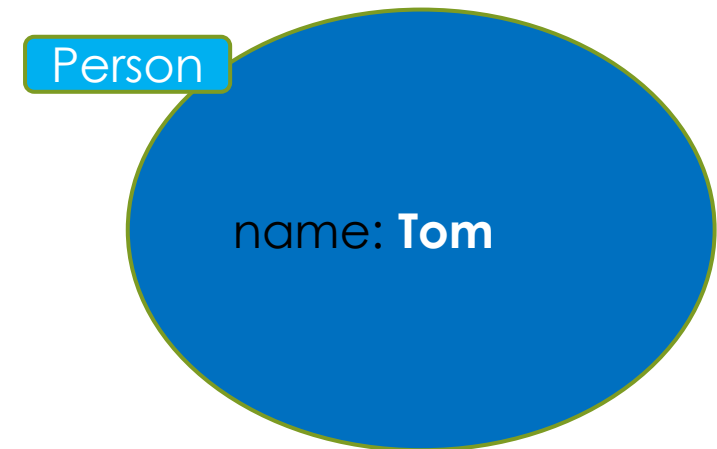


# Cypher - MERGE

- ▶ The MERGE clause ensures that a pattern exists in the graph.
- ▶ Either the pattern already exists, or it needs to be created.



```
MERGE (p:Person{name:"Brendan"})  
RETURN p
```

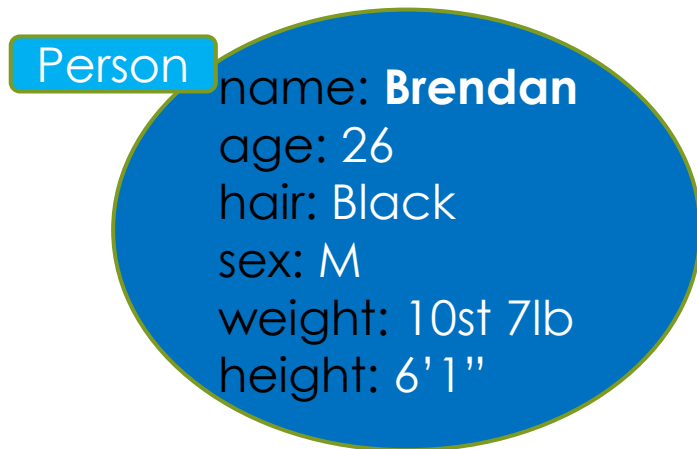


```
MERGE (p:Person{name:"Tom"})  
RETURN p
```

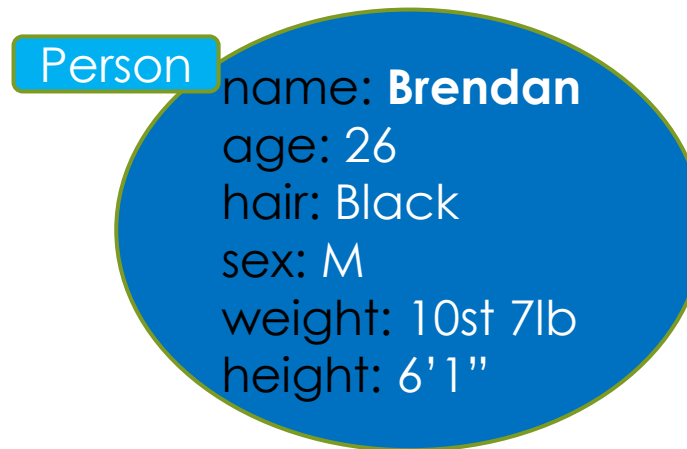


# Cypher - MERGE

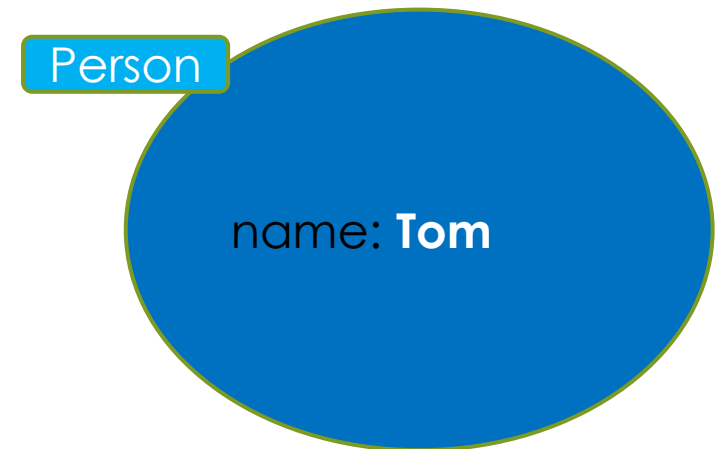
- ▶ The MERGE clause ensures that a pattern exists in the graph.
- ▶ Either the pattern already exists, or it needs to be created.



```
MERGE (p:Person{name:"Brendan"})  
RETURN p
```



```
MERGE (p:Person{name:"Tom"})  
RETURN p
```



## WITH

- ▶ The WITH clause allows query parts to be chained together, piping the results from one part of the query to the next.



Person

name: **Tom**  
age: 19  
height: 5'9.5"  
sex: M  
weight: 9st 2lb

Person

name: **Mary**  
age: 18  
hair: Brunette  
height: 5'6"  
sex: F  
weight: 8st 11lb

Person

name: **Alan**  
age: 44  
height: 6'1"  
sex: M

Person

name: **Alan**  
age: 21  
hair: Black  
sex: M





Person

name: **Tom**  
age: 19  
height: 5'9.5"  
sex: M  
weight: 9st 2lb

Person

name: **Mary**  
age: 18  
hair: Brunette  
height: 5'6"  
sex: F  
weight: 8st 11lb

Person

name: **Alan**  
age: 44  
height: 6'1"  
sex: M

Person

name: **Alan**  
age: 21  
hair: Black  
sex: M

MATCH(n:Person) RETURN avg(n.age)

25.5



Person

name: **Tom**  
age: 19  
height: 5'9.5"  
sex: M  
weight: 9st 2lb

Person

name: **Mary**  
age: 18  
hair: Brunette  
height: 5'6"  
sex: F  
weight: 8st 11lb

Person

name: **Alan**  
age: 44  
height: 6'1"  
sex: M

Person

name: **Alan**  
age: 21  
hair: Black  
sex: M

MATCH(n:Person) RETURN avg(n.age) **25.5**

MATCH(n:Person) RETURN n.name AS Name, avg(n.age) AS Avg

Name	Avg
Tom	19.0
Mary	18.0
Alan	32.5



Person

name: **Tom**  
age: 19  
height: 5'9.5"  
sex: M  
weight: 9st 2lb

Person

name: **Mary**  
age: 18  
hair: Brunette  
height: 5'6"  
sex: F  
weight: 8st 11lb

Person

name: **Alan**  
age: 44  
height: 6'1"  
sex: M

Person

name: **Alan**  
age: 21  
hair: Black  
sex: M

MATCH(n:Person) RETURN avg(n.age) **25.5**

MATCH(n:Person) RETURN n.name AS Name, avg(n.age) AS Avg

Name	Avg
Tom	19.0
Mary	18.0
Alan	32.5

MATCH(n:Person) WITH avg(n.age) as averageAGE

MATCH(n1:Person) WHERE n1.age < averageAGE

RETURN n1.name AS Name

Name
Tom
Mary
Alan



Person

name: **Tom**  
age: 19  
height: 5'9.5"  
sex: M  
weight: 9st 2lb

Person

name: **Mary**  
age: 18  
hair: Brunette  
height: 5'6"  
sex: F  
weight: 8st 11lb

Person

name: **Alan**  
age: 44  
height: 6'1"  
sex: M

Person

name: **Alan**  
age: 21  
hair: Black  
sex: M

- Return the number of Males (as *Num\_Younger*) who are less than the average Male age.



Person

name: **Tom**  
age: 19  
height: 5'9.5"  
sex: M  
weight: 9st 2lb

Person

name: **Mary**  
age: 18  
hair: Brunette  
height: 5'6"  
sex: F  
weight: 8st 11lb

Person

name: **Alan**  
age: 44  
height: 6'1"  
sex: M

Person

name: **Alan**  
age: 21  
hair: Black  
sex: M

- Return the number of Males (as *Num\_Younger*) who are less than the average Male age.

```
MATCH(p:Person{sex:"M"}) WITH avg(p.age) AS avgAge
```



Person

name: **Tom**  
age: 19  
height: 5'9.5"  
sex: M  
weight: 9st 2lb

Person

name: **Mary**  
age: 18  
hair: Brunette  
height: 5'6"  
sex: F  
weight: 8st 11lb

Person

name: **Alan**  
age: 44  
height: 6'1"  
sex: M

Person

name: **Alan**  
age: 21  
hair: Black  
sex: M

- Return the number of Males (as *Num\_Younger*) who are less than the average Male age.

```
MATCH(p:Person{sex:"M"}) WITH avg(p.age) AS avgAge
```

```
MATCH(p1:Person{sex:"M"}) WHERE p1.age < avgAge
```



Person

name: **Tom**  
age: 19  
height: 5'9.5"  
sex: M  
weight: 9st 2lb

Person

name: **Mary**  
age: 18  
hair: Brunette  
height: 5'6"  
sex: F  
weight: 8st 11lb

Person

name: **Alan**  
age: 44  
height: 6'1"  
sex: M

Person

name: **Alan**  
age: 21  
hair: Black  
sex: M

- Return the number of Males (as *Num\_Younger*) who are less than the average Male age.

```
MATCH(p:Person{sex:"M"}) WITH avg(p.age) AS avgAge
```

```
MATCH(p1:Person{sex:"M"}) WHERE p1.age < avgAge
```

```
RETURN count(p1) as Num_Younger
```



Person

name: **Tom**  
age: 19  
height: 5'9.5"  
sex: M  
weight: 9st 2lb

Person

name: **Mary**  
age: 18  
hair: Brunette  
height: 5'6"  
sex: F  
weight: 8st 11lb

Person

name: **Alan**  
age: 44  
height: 6'1"  
sex: M

Person

name: **Alan**  
age: 21  
hair: Black  
sex: M

- Return the number of Males (as *Num\_Younger*) who are less than the average Male age.

```
MATCH(p:Person{sex:"M"}) WITH avg(p.age) AS avgAge
```

```
MATCH(p1:Person{sex:"M"}) WHERE p1.age < avgAge
```

```
RETURN count(p1) as Num_Younger
```

Num\_Younger

2

