# COS 326
# Database Systems

## Lecture 14

NoSQL databases – Neo4j (2)

(Ref: Notes)

Thursday 8 September 2016

(Wednesday timetable)

# Admin matters (1)

**Schedule changes:**

| Week | Date | Day | Topic |
|---|---|---|---|
| 8 | 6 Sept | Tues | L13: NoSQL databases (Neo4j)<br>**Presentation: Essay topic 4** |
| | 7 Sept | Wed | UP Spring Day |
| | **8 Sept** | **Thurs** | **Wednesday timetable**<br>**L14: NoSQL databases (Neo4j)**<br>**Presentation: Essay topic 5** |
| | 9 Sept | Fri | ***Practical 6:*** **MongoDB** |
| | | | |
| 9 | 13 Sept | Tues | L15: NoSQL databases (Neo4j)<br>**Presentation: Essay topic 6** |
| | **14 Sept** | **Wed** | **Class Test 2: XML and NoSQL DBs**<br>L16: NoSQL databases (Neo4j) |
| | 16 Sept | Fri | ***Practical 7:*** **Neo4j tutorial** |

# Admin matters (2)

**Schedule changes:**

| Week | Date | Day | Topic |
|---|---|---|---|
| 10 | 20 Sept | Tues | L15: NoSQL databases (Neo4j)<br>**Presentation: Essay topic 7** |
| | **21 Sept** | **Wed** | L16: NoSQL databases (Neo4j)<br>**Presentation: Essay topics 8 & 9** |
| | **22 Sept** | **Thurs** | **Semester Test (evening)** |
| | 23 Sept | Fri | ***Practical 7:*** **Neo4j** |
| | | | |
| 11 | 27 Sept | Tues | L19: Data Analytics: big data |
| | 28 Sept | Wed | L20: Data Analytics: big data<br>**Presentation: Essay topic 1** |
| | **30 Sept** | **Fri** | ***No prac:*** |

# Outline

1. RECAP:  Neo4j DB enviroment

2. Neo4j CRUD operations
    a. WRITE clauses
    b. READ clauses
    c. AGGREGATION clauses

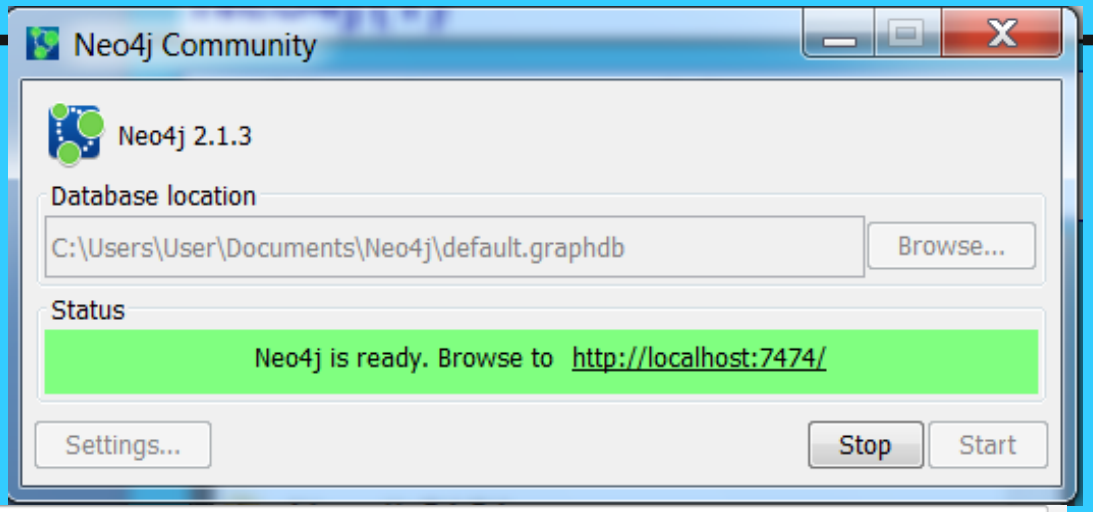
**Reference:  Neo4j 2.1.3 manual (documentation)**

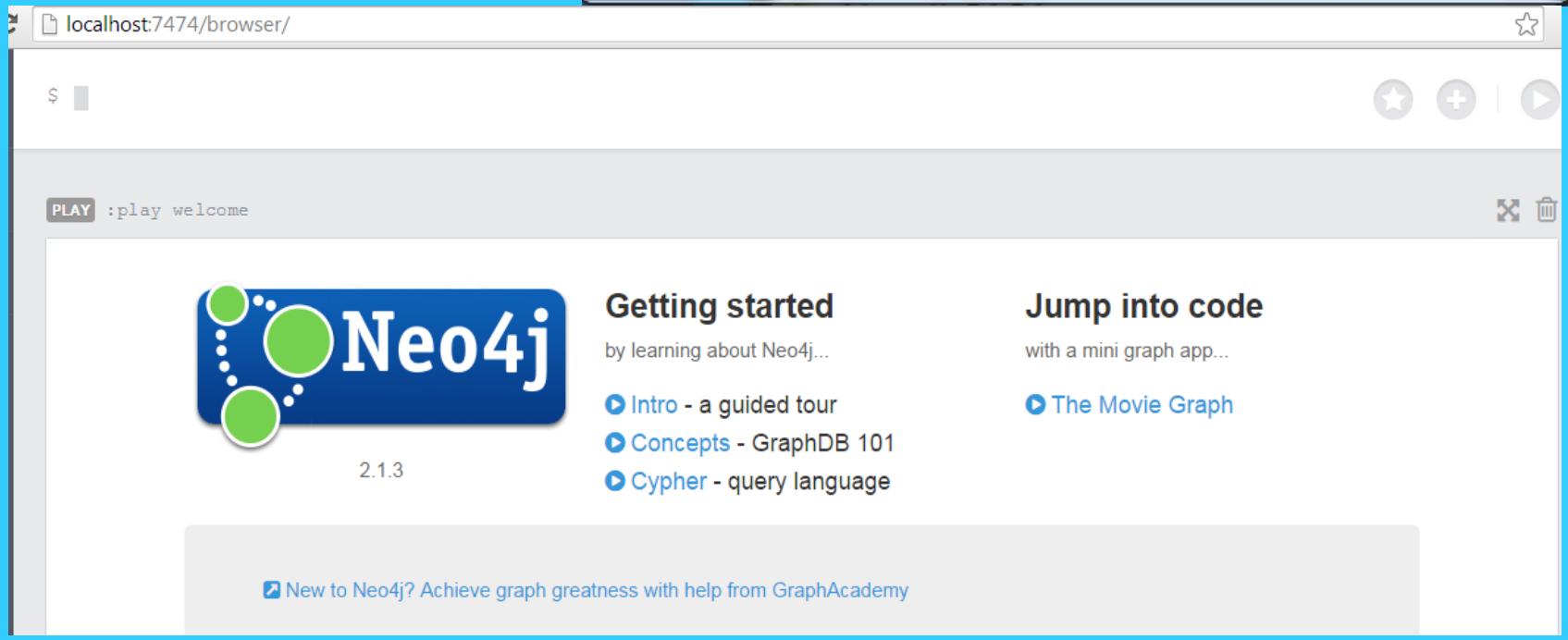
# Recap: Neo4j graph database

**Client / server.**

**connection URL:**

**http://localhost:7474**



# Recap: Neo4j Cypher query language

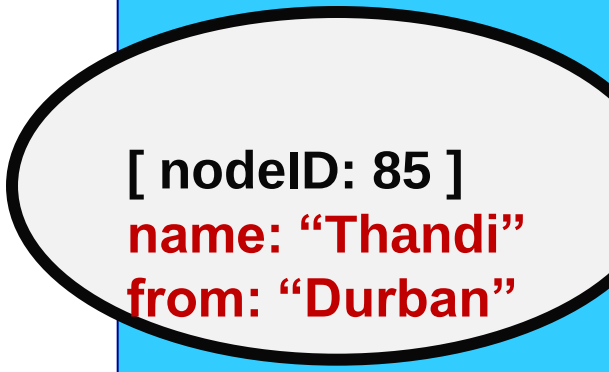
**Cypher:** Neo4j's graph query language

- purpose built for working with graph data.
- uses patterns to describe graph data
- uses familiar SQL-like clauses
- is declarative (like SQL) i.e. describes what to do, not how to do it

- **A Neo4j graph consists of:**
  - **Nodes which store data as Properties**
  - **Properties are simple key/value pairs e.g.**

**name: "Thandi" from: "Durban"**

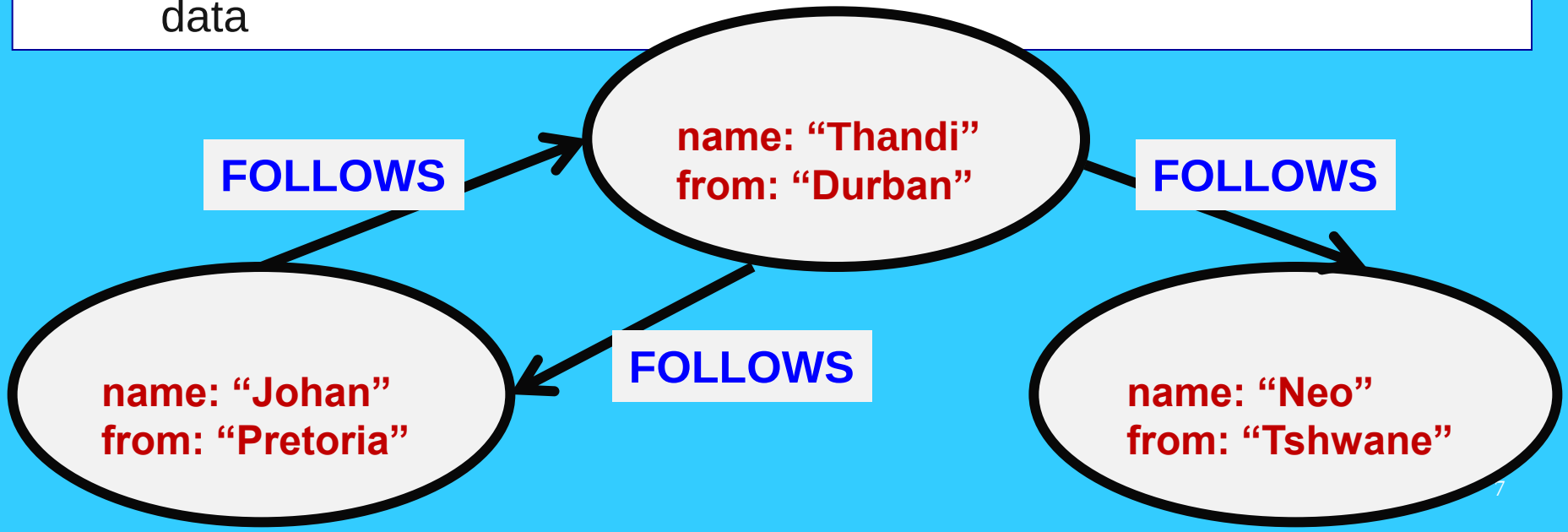
  - **Plus optional components:**
    - **Constraints & indexes**
  - **System assigns unique node IDs**



# Nodes and Relationships (1)

**Relationships:** Connect nodes in the graph

- The real power of a graph database e.g. Neo4j is in connected data.
- **Relationships describe how the records are related**. e.g. the **FOLLOWS** relationship
  - **Thandi FOLLOWS Neo and Johan**
  - **Johan FOLLOWS Thandi**
- Relationships **always have direction & a type** & form patterns of data



# Nodes and Relationships (2)

- **Relationship properties**
  - Store information shared by two nodes.
  - In a property graph, **relationships are data records** that can also contain properties. e.g
    - **Thandi FOLLOWS Neo since 2009**
    - **Thandi FOLLOWS Johan since 2012**
    - **Johan FOLLOWS Thandi since 2010**

# CRUD operations: WRITING clauses

- Writing clauses (ch. 11, pages 160-181)

| Writing clause | Definition |
|---|---|
| **CREATE** | Creating graph elements:  nodes, relationships, constraints |
| **MERGE** | a combination of MATCH and CREATE that additionally allows you to specify what happens if the data was matched or created. |
| **SET** | for **updating labels** on nodes and **properties** on nodes and relationships |
| **DELETE** | Deleting graph elements:  nodes and relationships |
| **REMOVE** | for removing properties and labels from graph elements |
| **FOREACH** | use for updating commands on elements in a collection: a path, or a collection created by aggregation. |
| **CREATE UNIQUE** | **Create unique nodes. Create node if missing.** If the pattern described needs a node, and it can't be matched, a new node will be created. |

# CRUD operations: CREATE (1)

**A CREATE clause can create many nodes and relationships at once.**

```
CREATE
    ( psT:Person { name: “Thandi", from: “Durban",  hobby: “singing" } ),
    ( psJ:Person { name: "Johan", from: "Pretoria",  hobby: "surfing" } ),
     (psN:Person { name: "Neo", from: "Tshwane", hobby: "soccer“ } ),
     (psT)-[:FOLLOWS { since: 2012 } ]->(psJ ),
     (psT)-[:FOLLOWS { since: 2009 } ]->(psN ),
     (psJ)-[:FOLLOWS { since: 2010 } ]->(psT )
```

# CRUD operations: CREATE (2)

**A CREATE clause can create a constraint and an index**

**CREATE CONSTRAINT ON ( psn: Person)**

**ASSERT psn.name IS UNIQUE**

**Note:** adding the unique constraint will add an index on that property

**Alternatively:** **CREATE INDEX ON : Person(name)**;

```
CYPHER CREATE CONSTRAINT ON ( psn: Person) ASSERT psn.name IS UNIQUE;
```

✔ Added 1 constraint, returned 0 rows in 1140 ms

# CRUD operations: SET

for **updating labels** on nodes and **properties** on **nodes** & **relationships**

```
MATCH (psT)
WHERE  psT.name =  "Thandi"   SET psT.age = 20   RETURN psT
```

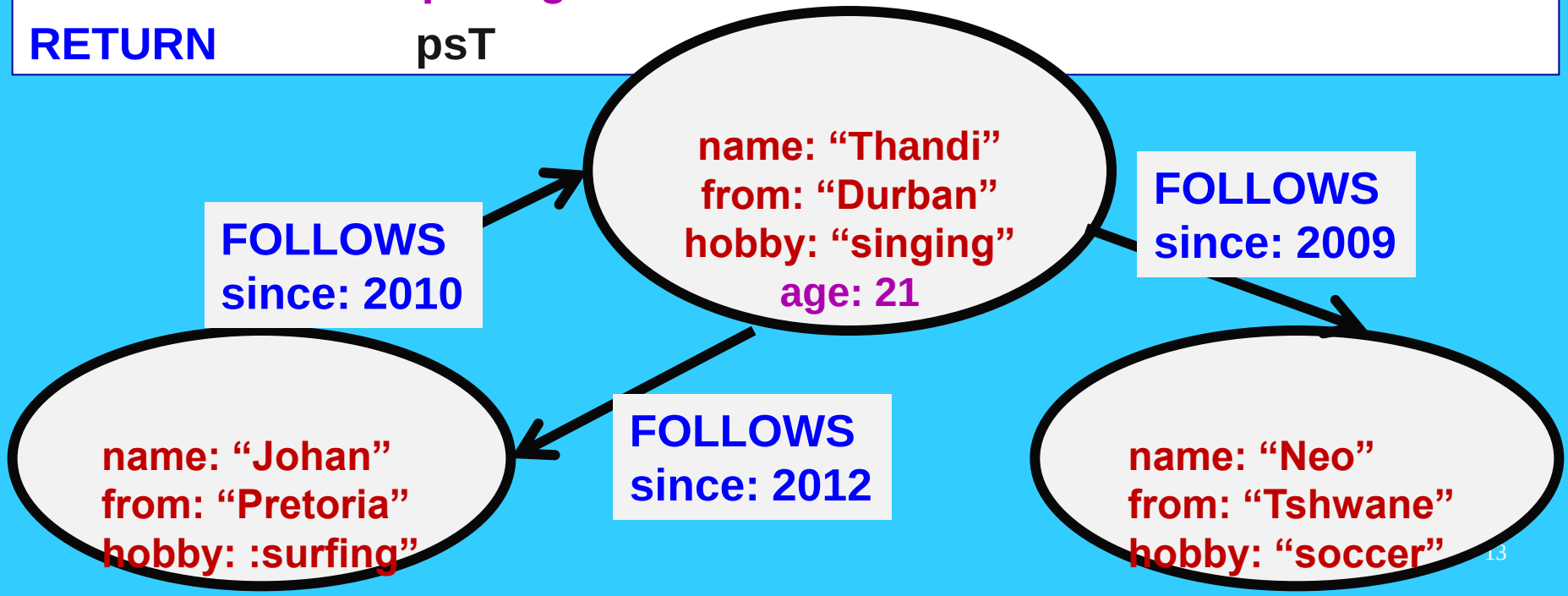
**BUT for relationships:**

```
MATCH (psN),  (psT)
WHERE psN.name =  "Neo" AND  psT.name =   "Thandi"
CREATE  (psN)-[:FOLLOWS { since: 2014 } ]->(psT )
```

# CRUD operations: MERGE

either matches existing node and binds it, or creates new node and binds it. **(Can use only ON CREATE or only ON MATCH or both)**

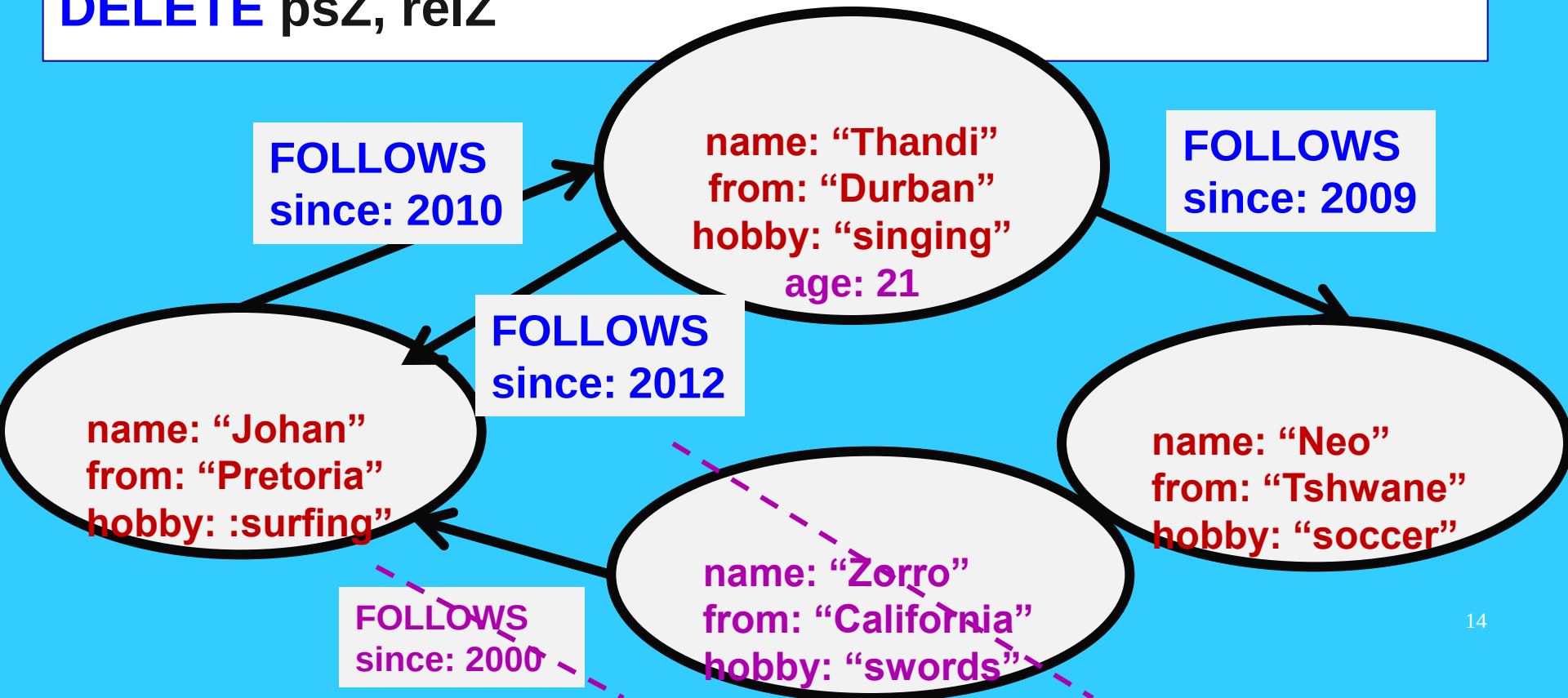
```
MERGE           ( psT:Person { name: "Thandi"} )
ON CREATE SET psT.name = "Thandi", psT.from = "Durban",
                 psT.hobby = "singing", psT.age = 21
ON MATCH SET    psT.age = 21
RETURN          psT
```



# CRUD operations: DELETE (1)

- **Deleting graph elements: nodes and relationships**
- e.g. Delete a node and connected relationships

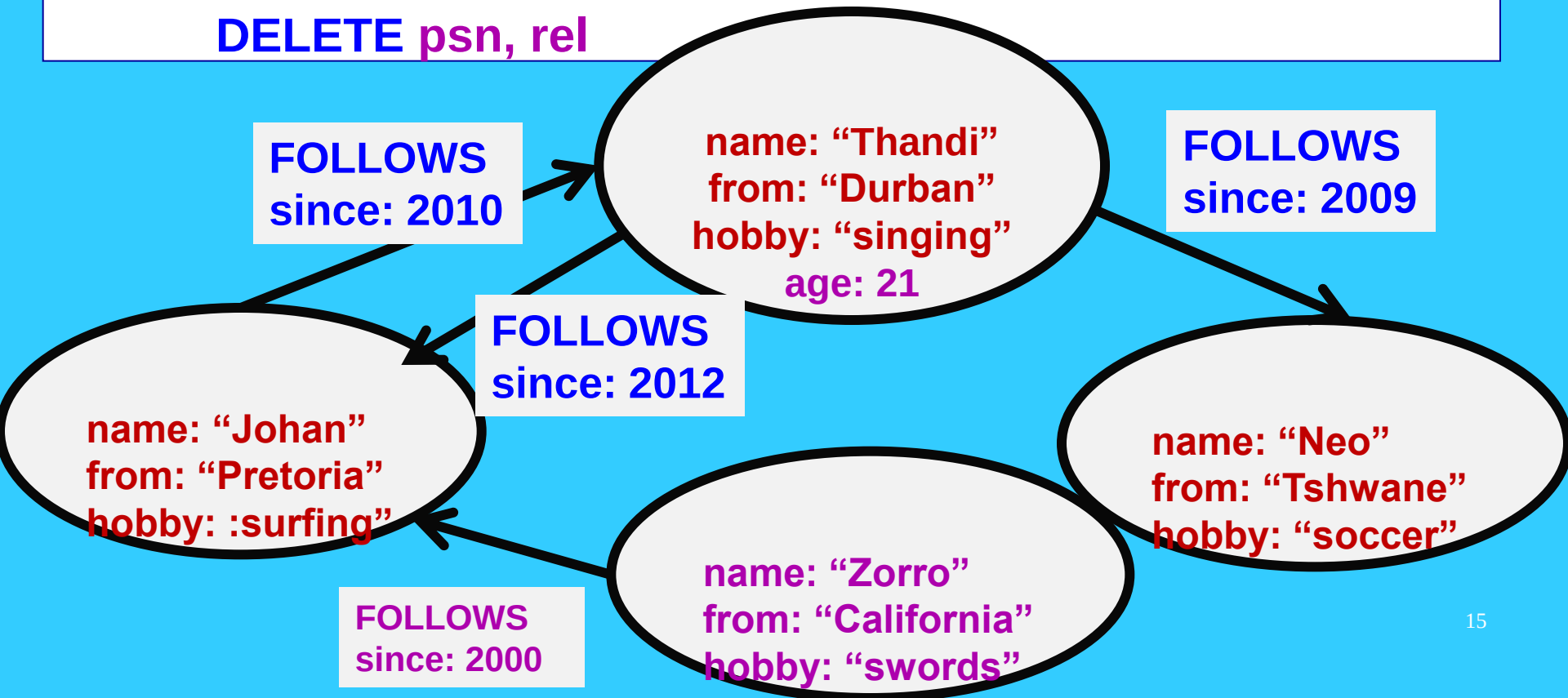
**MATCH ( psZ:Person {name: “Zorro”})-[ relZ]-( )**
**DELETE psZ, relZ**



# CRUD operations: DELETE (2)

- **Deleting graph elements: nodes and relationships**
- e.g. Delete all nodes and connected relationships

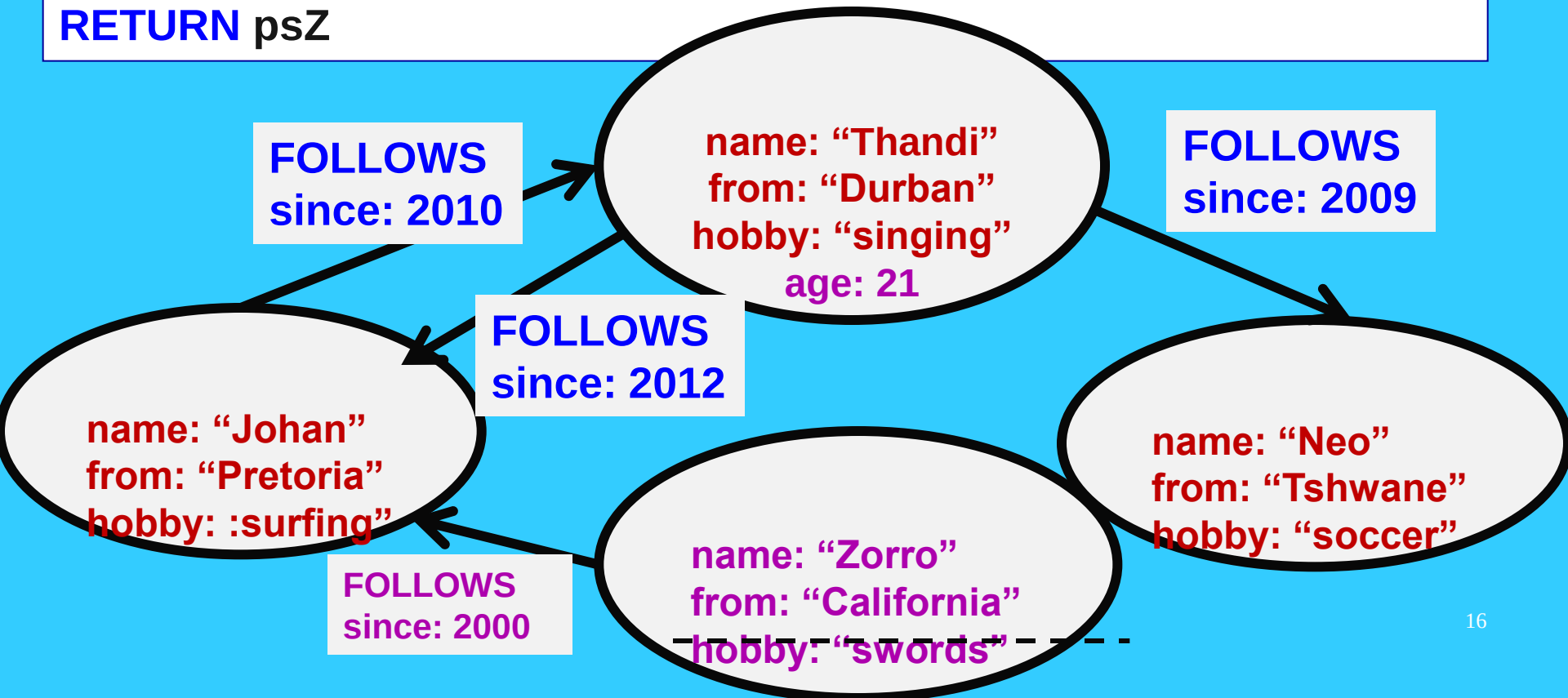
```
MATCH ( psn:Person )
OPTIONAL MATCH (psn)-[rel]-( )    //equiv to outer join
DELETE psn, rel
```



# CRUD operations: REMOVE

- for removing **properties** and labels from graph elements (nodes & relationships) e.g.

```
MATCH   ( psZ:Person) WHERE psZ.name = “Zorro”
REMOVE psZ.hobby
RETURN psZ
```



# CRUD operations: READING clauses (1)

- Reading clauses (ch. 10, pages 131-157)

| Reading clause | Definition |
|---|---|
| **MATCH** | The MATCH clause allows you to specify the patterns Cypher will search for in the database. |
| **OPTIONAL MATCH** | if no matches are found, OPTIONAL MATCH will use NULLs for missing parts of the pattern. OPTIONAL MATCH could be considered the Cypher equivalent of the outer join in SQL. |
| **WHERE** | **WHERE is not a clause in it's own right**, rather, it's part of MATCH, OPTIONAL MATCH, START and WITH.<br>In the case of WITH and START, WHERE simply filters the results. |
| **AGGREGATION** | To calculate aggregated data, Cypher offers aggregation, much like SQL's aggregates and GROUP BY. |

# CRUD operations: READING clauses (2)

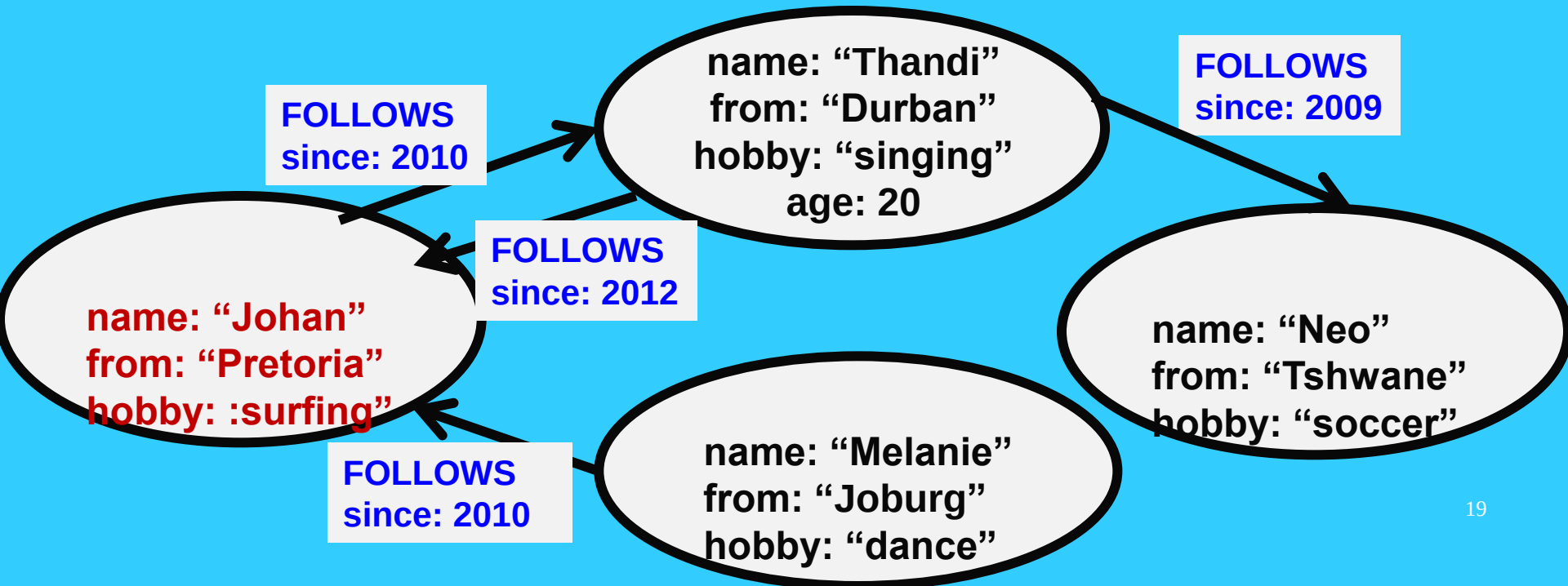
- **Patterns for MATCH**

| Pattern | meaning |
|---|---|
| **MATCH (<var>)** | any node |
| **MATCH** (<var>: <label> ) | a single node pattern with label <label> which will assign matches to the variable <var> |
| **MATCH (<var1>:<label>)-[:<relationship]-(<var2>)** | |
| **MATCH (<var1> { <key>: <value> })-->(<var2> )** | |
| **etc** | |
| | |

# CRUD operations: MATCH

```
MATCH (psT:Person) RETURN psT;          //returns nodes

MATCH (psT:Person) RETURN psT.name;     //returns names
```



# CRUD operations: MATCH & WHERE

specify selection criteria for patterns

e.g. show Melanie's details

```
MATCH (psM)   WHERE psM.name =  "Melanie"
 RETURN psM.name,    psM.from,   psM.hobby
```

e.g. find the people that Thandi follows

```
MATCH (psn:Person)-[:FOLLOWS]->(followed)
WHERE psn.name = "Thandi"
RETURN psn.name,  followed.name
```

# CRUD operations: MATCH & ORDER BY

## General clauses (ch. 10, pages 115-129)

- **ORDER BY** is used to sort the output.

```
MATCH (psn)   RETURN psn.name
ORDER BY psn.name
```

- Sorted list of names is returned

# CRUD operations: AGGREGATION (1)

To calculate aggregated data, Cypher offers aggregation, much like SQL's aggregates and GROUP BY.

| Aggregation function | example |
|---|---|
| **'GROUP BY'** | **MATCH (n { name: 'A' })-->(x)**<br>**RETURN n.name, count(x)**<br>***n.name is the grouping key, count(x) is the aggregate*** |
| **count** | RETURN count(*) |
| **sum** | RETURN sum(n.property) |
| **avg** | RETURN avg(n.property) |
| **stdev** | RETURN stdev(n.property) |
| **max** | RETURN max(n.property) |
| **min** | RETURN min(n.property) |
| **collect** | RETURN collect(n.property) |
| **distinct** | RETURN count(DISTINCT b.eyes) |

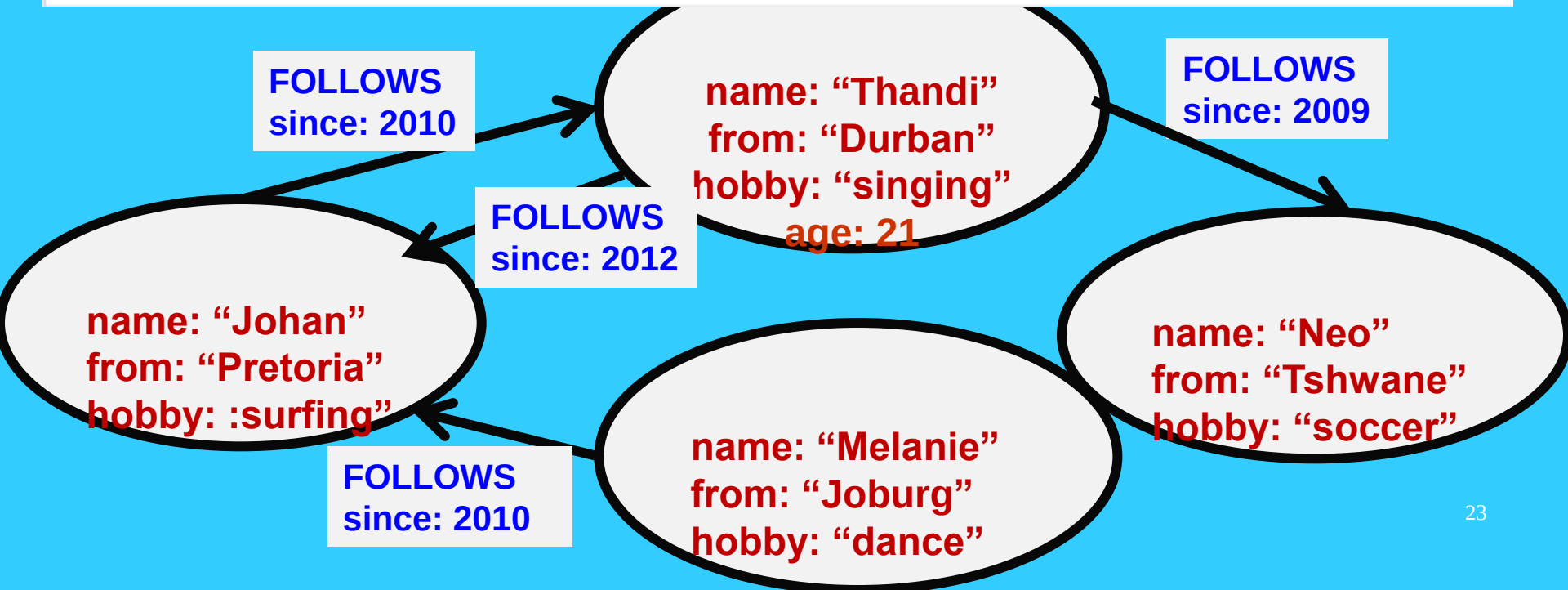
# CRUD operations: AGGREGATION (2)

e.g.1 **count the people that Thandi follows:**

```
MATCH   (psT {name: "Thandi"} )-[:FOLLOWS]->  followed
RETURN  psT.name, count(*) AS count_follows;
```

```
CYPHER MATCH (psT {name: "Thandi"} )-[:FOLLOWS]-> followed RETURN psT.name, count(*) AS count_follows;
```

| psT.name | count_follows |
|---|---|
| Thandi | 2 |



# CRUD operations: AGGREGATION (3)

**e.g.2 Count the number of relationships in the graph**

```
MATCH   (psn )-[relation]->( )
RETURN  type(relation), count(*)
```

# Essay topic presentation

- Topic 5