Chafik NOUIRA, LesFurets.com
tp-bigdata@lesfurets.com

Plan

1. **Intro**
2. Data storage
3. Use case
4. Cypher
5. Functions & Procedures
6. Neo4j @LesFurets

Intro

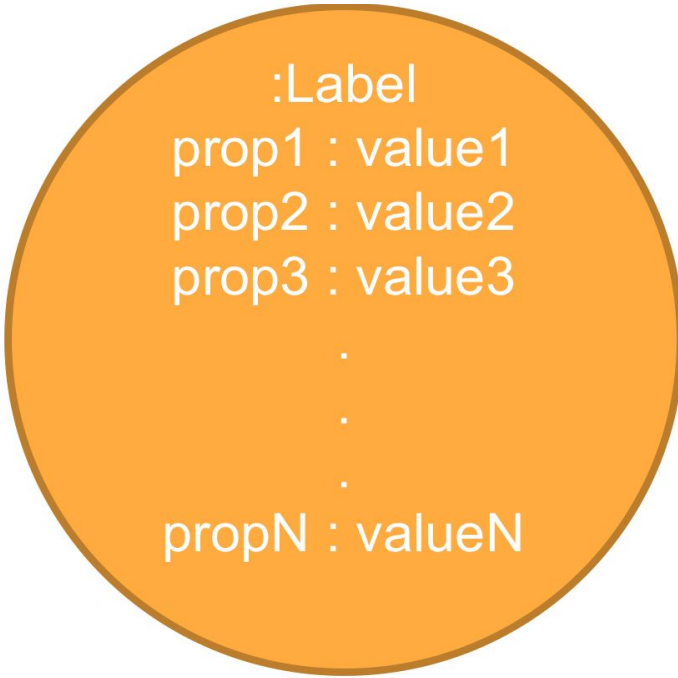
- NoSQL Graph Database Management System (DBMS), developed by Neo Technology, Inc.
- Developed in Java, provides cross platform accessibility
- Adheres to ACID properties
- Flexible schema
- Accessible from different software using Cypher Query Language(CQL) through HTTP

Plan

1. Intro
2. **Data storage**
   - **Node**
   - **Relationship**
   - **Label**
3. Use case
4. Cypher
5. Functions & Procedures
6. Neo4j @LesFurets

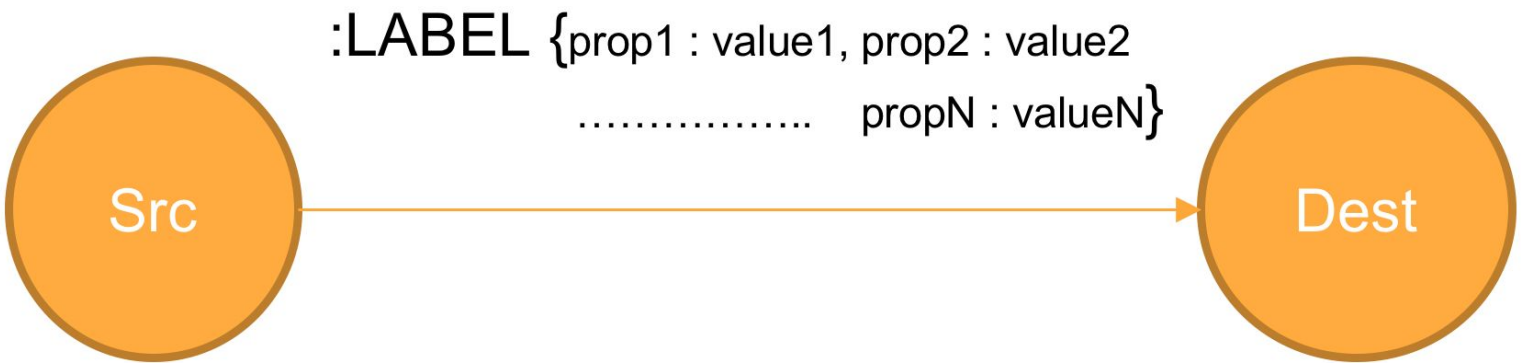
Node

- Basic building block
- Contains properties stored as key-value pairs
- Numeric(integer, float), String, Boolean, List
- Nodes cannot contain Nodes
- Storage capacity : No limitation



Relationship

- Another basic building block, also known as egde
- Contains properties also stored as key-value pairs
- Used to connect two nodes
- Always directed from one node to another
- Storage capacity : No limitation



Label

- A name given to a set of nodes or relationships
- Can be considered as a node or relationship’s type
- Storage capacity : 64k different labels (16M in Entreprise Edition)
- **:NodeLabel** (No limitation of labels per node)
- **:RELATIONSHIP_LABEL** (Only one label per relationship)

Plan

1. Intro
2. Data storage
3. **Use case**
4. Cypher
5. Functions & Procedures
6. Neo4j @LesFurets

Student

Name	Country	Hair	University
Andrei	Romania	Black	Paris Diderot
Alexandre	Canada	Brown	Université de Montréal
Geoffrey	France	Blond	Paris Diderot
Chafik	Tunisia	Black	UPMC
Jonathan	France	Black	UPMC

Country

ID	Name
1	Romania
2	Canada
3	France
4	Tunisia

Country

ID	Name	Capital_city
1	Romania	Bucharest
2	Canada	Ottawa
3	France	Paris
4	Tunisia	Tunis

Student

Name	Country	Hair	University
Andrei	1	Black	Paris Diderot
Alexandre	2	Brown	Université de Montréal
Geoffrey	3	Blond	Paris Diderot
Chafik	4	Black	UPMC
Jonathan	3	Black	UPMC

University

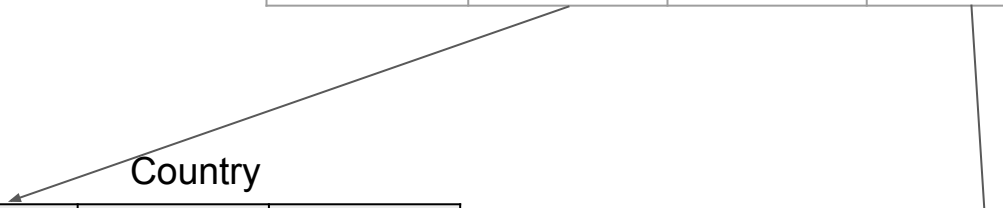
ID	Name
1	Paris Diderot
2	Université de Montréal
3	UPMC

Student

Name	Country	Hair	University
Andrei	1	Black	1
Alexandre	2	Brown	2
Geoffrey	3	Blond	1
Chafik	4	Black	3
Jonathan	3	Black	3

Student

Name	Country	Hair	University
Andrei	1	Black	1
Alexandre	2	Brown	2
Geoffrey	3	Blond	1
Chafik	4	Black	3
Jonathan	3	Black	3



Country

ID	Name	Capital city
1	Romania	Bucharest
2	Canada	Ottawa
3	France	Paris
4	Tunisia	Tunis

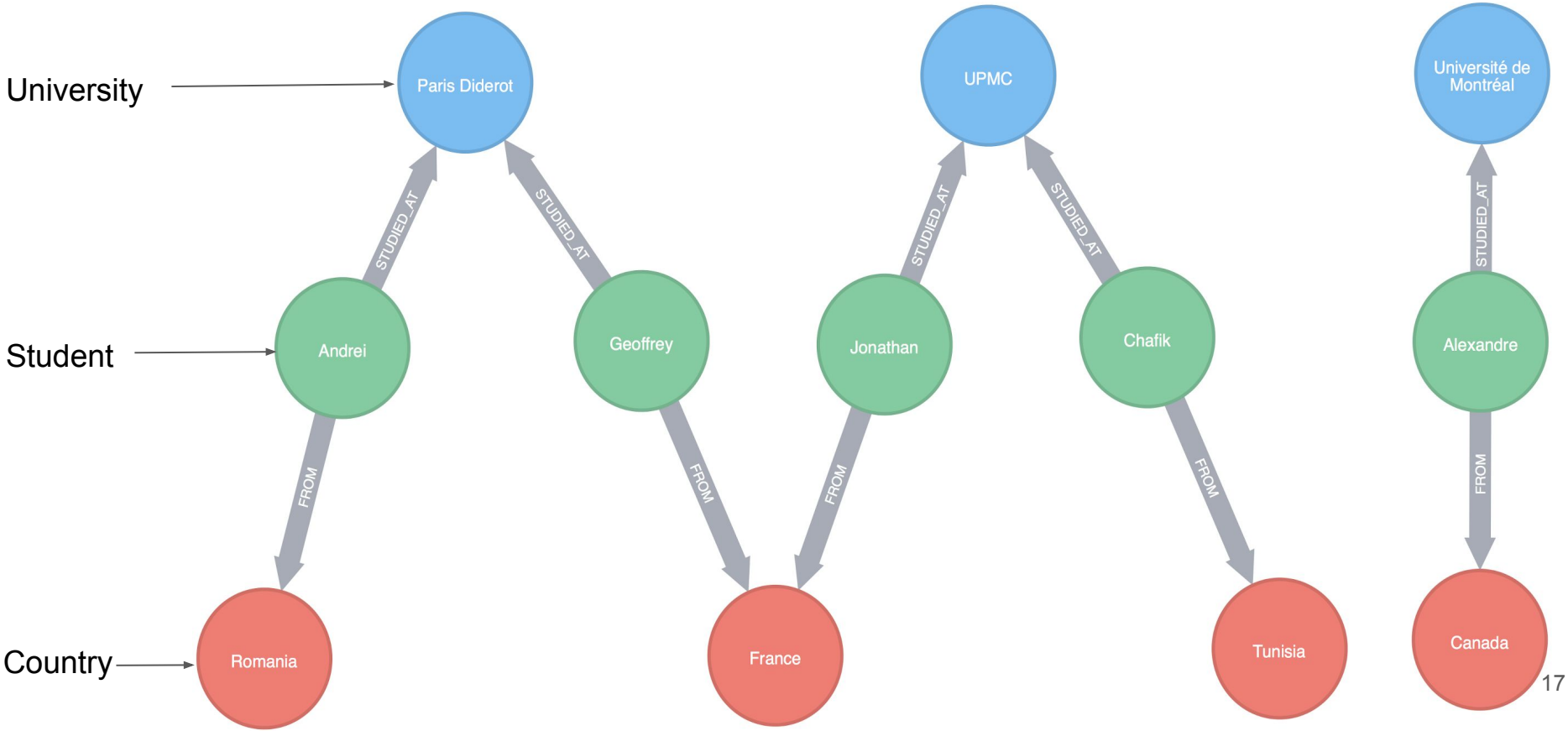
University

ID	Name
1	Paris Diderot
2	Université de Montréal
3	UPMC

Requête SQL

```
SELECT
    s.name, c.name, u.name
FROM
    student s
    LEFT JOIN country c ON c.ID = s.country
    LEFT JOIN university u ON u.ID = s.university
WHERE
    u.name = 'UPMC'
```

Graph


University
Student
Country
Paris Diderot
UPMC
Université de Montréal
Andrei
Geoffrey
Jonathan
Chafik
Alexandre
Romania
France
Tunisia
Canada
STUDIED_AT
STUDIED_AT
STUDIED_AT
STUDIED_AT
STUDIED_AT
FROM
FROM
FROM
FROM
FROM

Requête CYPHER

```
MATCH
    (u:University{name:"UPMC"})<-[:STUDIED_AT]-(s:Student)-[:FROM]->(c:Country)
RETURN
    s.name AS Student, c.name AS Country, u.name AS University
```

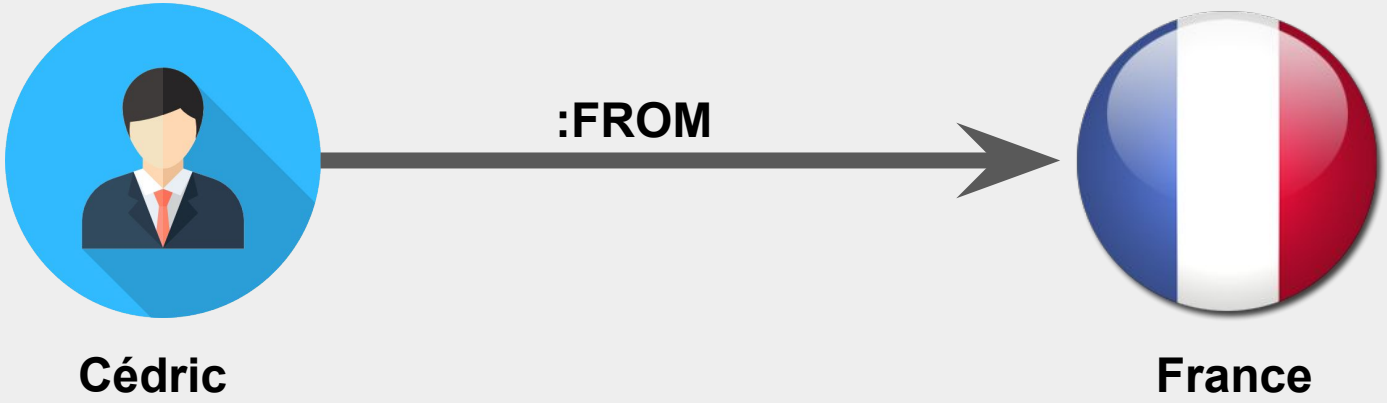
Plan

1. Intro
2. Data storage
3. Use case
4. **Cypher**
   - **Updating clauses**
   - **Reading clauses**
   - **Sub-queries**
5. Functions & Procedures
6. Neo4j @LesFurets

Cypher

- Declarative graph query language
- Borrows its structure from SQL
- Expressing what to retrieve from graph, not how to retrieve it
- Reads, writes and updates
- Query optimization
- MATCH <pattern> WHERE <condition> RETURN <expression>

Graph representation



Cypher – Syntax

NODE RELATIONSHIP NODE

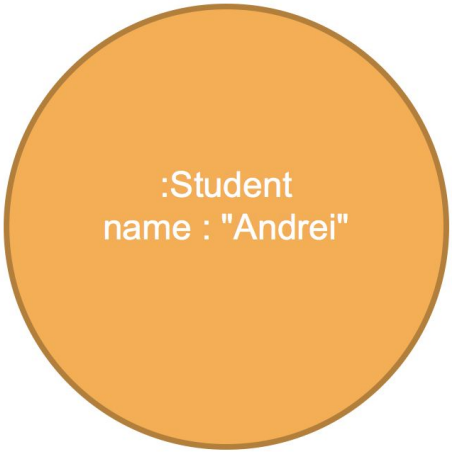
```
CREATE (:Student {name : "Cédric"}) - [:FROM]-> (:Country {name : "France"})
```

Clause LABEL PROPERTY LABEL PROPERTY

Updating clauses

- **CREATE** : creates nodes, relationships and patterns

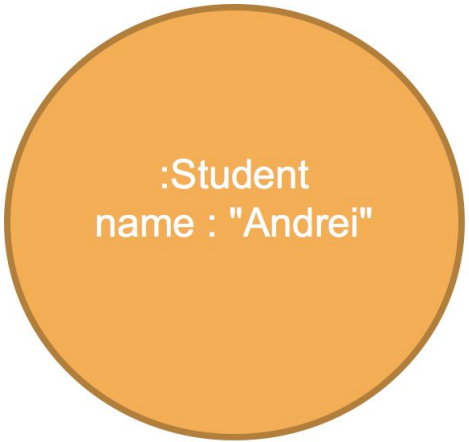
  CREATE (s:Student {name : "Andrei"})

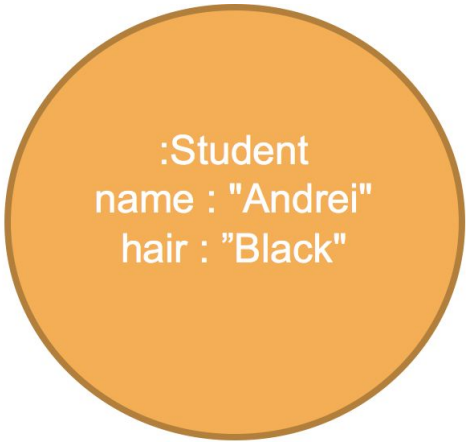


Updating clauses

- **MERGE** : merges nodes, relationships and patterns.
  Either a pattern exists or it needs to be created.

  MERGE (s:Student {name : "Andrei", hair : "Black"})





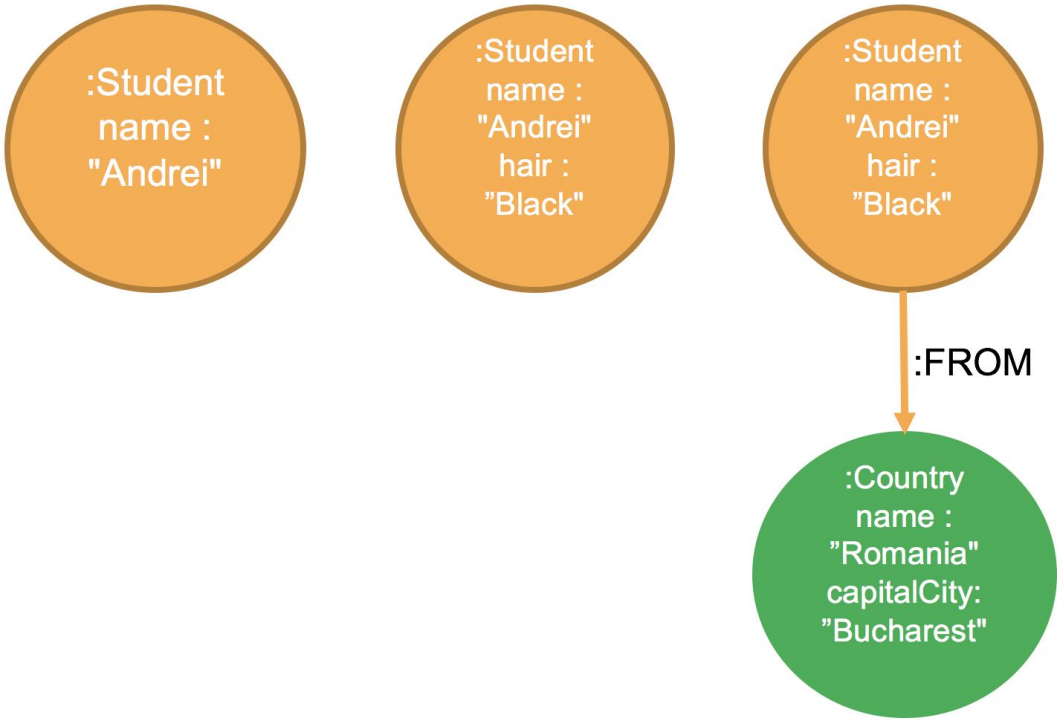
Updating clauses

```
MERGE (s:Student {name : "Andrei"})
      ON CREATE SET s.hair = "Black"
      ON MATCH SET s.hair = "Black"
```

Updating clauses

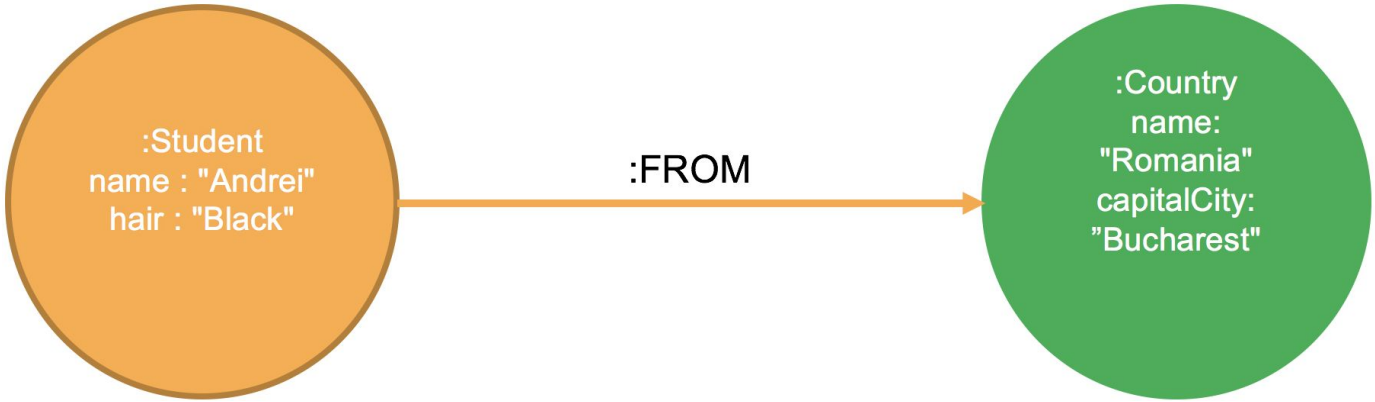
MERGE

(s:Student {name : "Andrei"})-[:FROM]->(c:Country {name : "Romania"})



Updating clauses

```
MERGE (s:Student {name : "Andrei"})
      ON CREATE SET s.hair = "Black"
      ON MATCH SET s.hair = "Black"
MERGE (s)-[r:FROM]->(c:Country {name: "Romania"})
RETURN s, r, c
```



Updating clauses

- **MERGE** : when using constraints, the first example will work just fine.

```
CREATE CONSTRAINT ON (s:Student) ASSERT s.name IS UNIQUE
```

- **SET** : Adds or updates properties (on nodes and relationships) and labels (only on nodes)
  Used with **MATCH** or **MERGE**

  Example

```
MATCH (s:Student {name : "Andrei"})
    SET s:Person
    SET s.birthYear = 1990
```

Updating clauses

- **DELETE** : Delete nodes and relationships
  When deleting a node, use DETACH to delete its relationships first.

  Example

  ```
  MATCH (s:Student {name : "Andrei"})
  DETACH DELETE s
  ```

- **REMOVE** : Removes properties from a node or a relationship.

  Example

  ```
  MATCH (s:Student {name : "Andrei"})
  REMOVE s.birthYear
  ```

Reading clauses

- **MATCH** : simplest way to get data from the graph

```
MATCH (s:Student)
RETURN s.name AS Name, s.hair AS Hair
```

- **WHERE** : Filter results using conditions

```
MATCH (s:Student)
WHERE s.name = "Andrei"
RETURN s.hair AS Hair
```

```
MATCH (s:Student {name : "Andrei"})
RETURN s.hair AS Hair
```

Reading clauses

- **RETURN** : Results to be returned. Can be used with **SKIP**/**LIMIT** and **ORDER BY**

```
MATCH (s:Student)
RETURN s.name, s.birthDate
ORDER BY s.birthDate DESC
LIMIT 10
```

Reading clauses

- Returned data can be : nodes, relationships, properties and patterns

```
MATCH p = (:Student)-[:FROM]->(:Country)
RETURN p
```

Sub-queries

- **WITH** : allows query parts to be chained together by passing the results from one to another.
  Can be used with **WHERE**, **ORDER BY**, and **SKIP/LIMIT**.

```
MATCH (u:University)-[:STUDIED_AT]-(s:Student)-[:FROM]-(c:Country)
WITH u, count(DISTINCT c) AS nbCountries
WHERE nbCountries > 1
RETURN u.name AS University, nbCountries
ORDER BY nbCountries DESC, University
```

Other clauses

- **UNWIND** : expands a list into a sequence of rows.

```
UNWIND [1, 1, 2, 3] AS x
RETURN DISTINCT x
```

- **UNION** : used to combine the results of multiple queries.

```
MATCH (s:Student)
RETURN s.name AS name
UNION
MATCH (c:Country)
RETURN c.capitalCity AS name
```

Other clauses

- **FOREACH** : used to update data within a list.

```
MATCH p = (s:Student)--(u:University)
FOREACH (n IN nodes(p) | SET n.marked = TRUE)
```

- Other clauses exist
  see https://neo4j.com/docs/developer-manual/current/cypher/clauses/

Plan

1. Intro
2. Data storage
3. Use case
4. Cypher
5. **Functions & Procedures**
6. Neo4j @LesFurets

Functions & Procedures

- Predicate, scalar, aggregating, list, mathematical, string and spatial.

```
MATCH (c:Country)
RETURN count(c) AS nbCountries
```

- **APOC** : Awesome Procedures On Cypher
  - **Procedure**

```
CALL apoc.algo.wcc() YIELD nodeIds, stats
RETURN nodeIds, stats
```

  - **Function**

```
RETURN apoc.version()
```

Functions & Procedures

- **APOC** : Awesome Procedures On Cypher

https://github.com/neo4j-contrib/neo4j-apoc-procedures

- **Neo4j Graph Algorithms**

https://github.com/neo4j-contrib/neo4j-graph-algorithms

Plan

1. Intro
2. Data storage
3. Use case
4. Cypher
5. Functions & Procedures
6. **Neo4j @LesFurets**

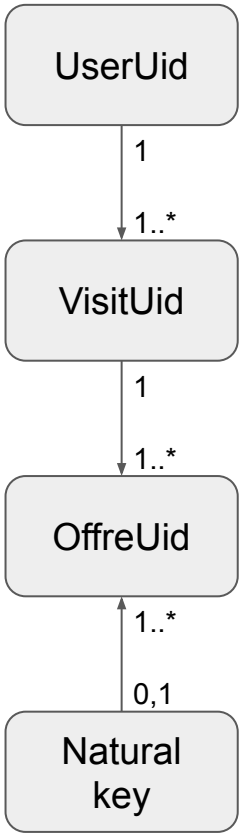
Problematic

- How to identify a user and rebuild his journey on our website LesFurets.com ?

Context

- Form to fill out
- Non-centralized data (Forms & Tracking)
- Unauthenticated users
- Cookie for each web browser
- Invalid email addresses

Data model



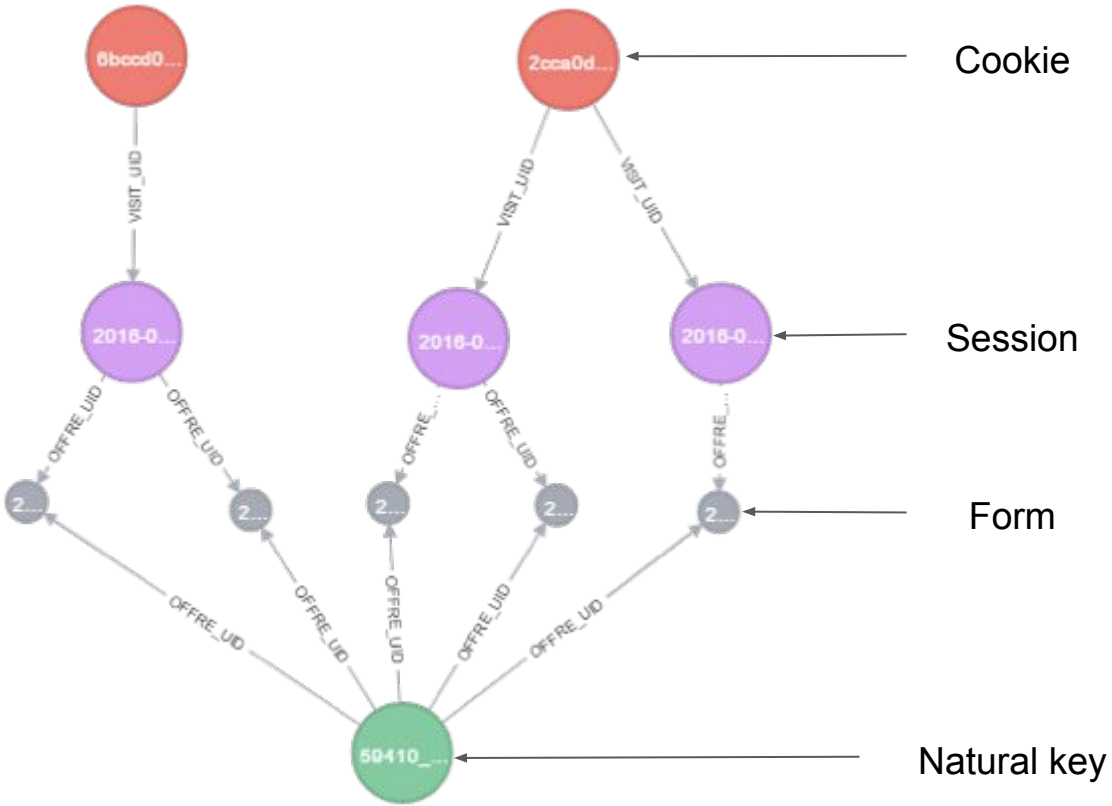
Cookie

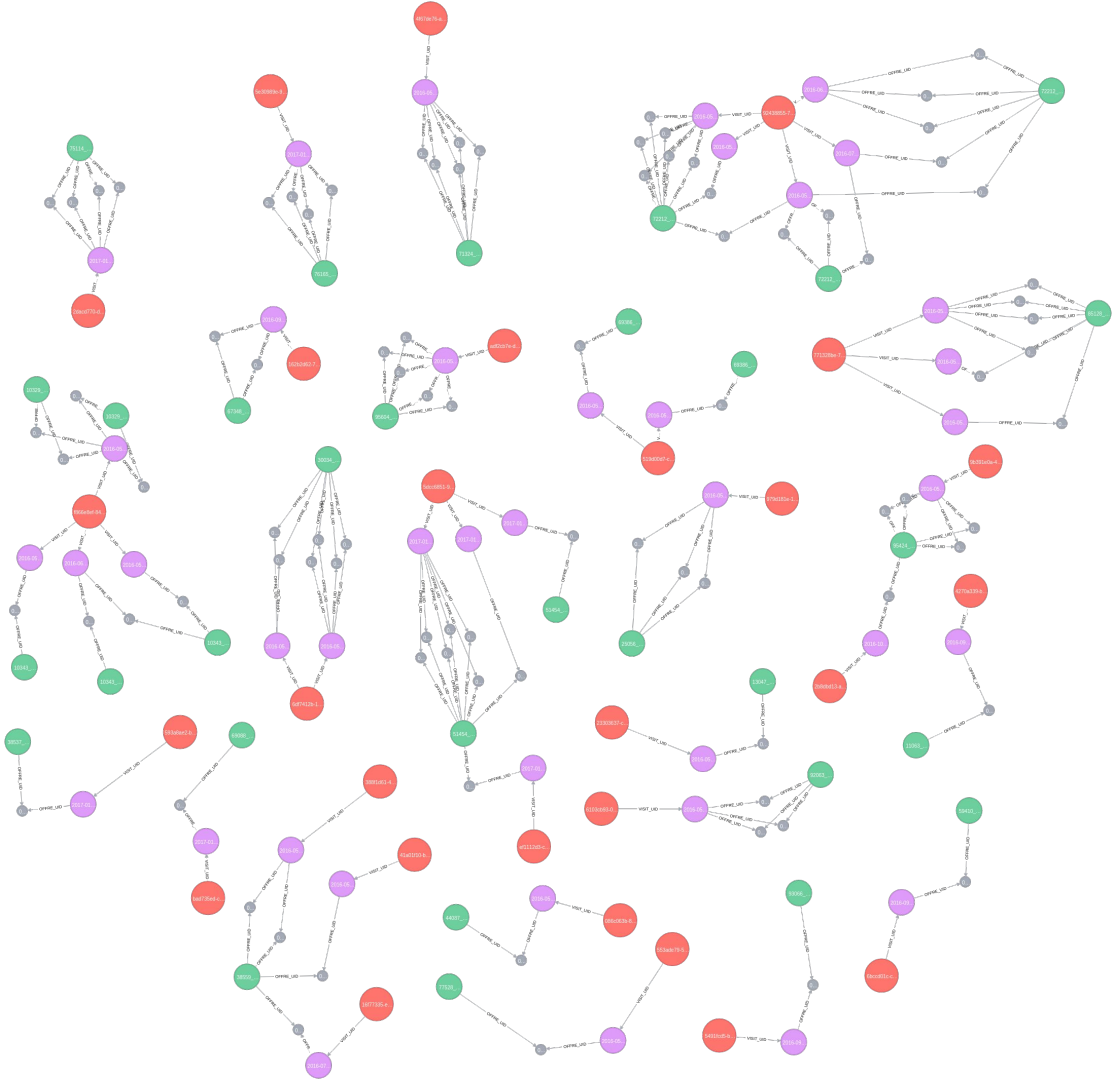
Session

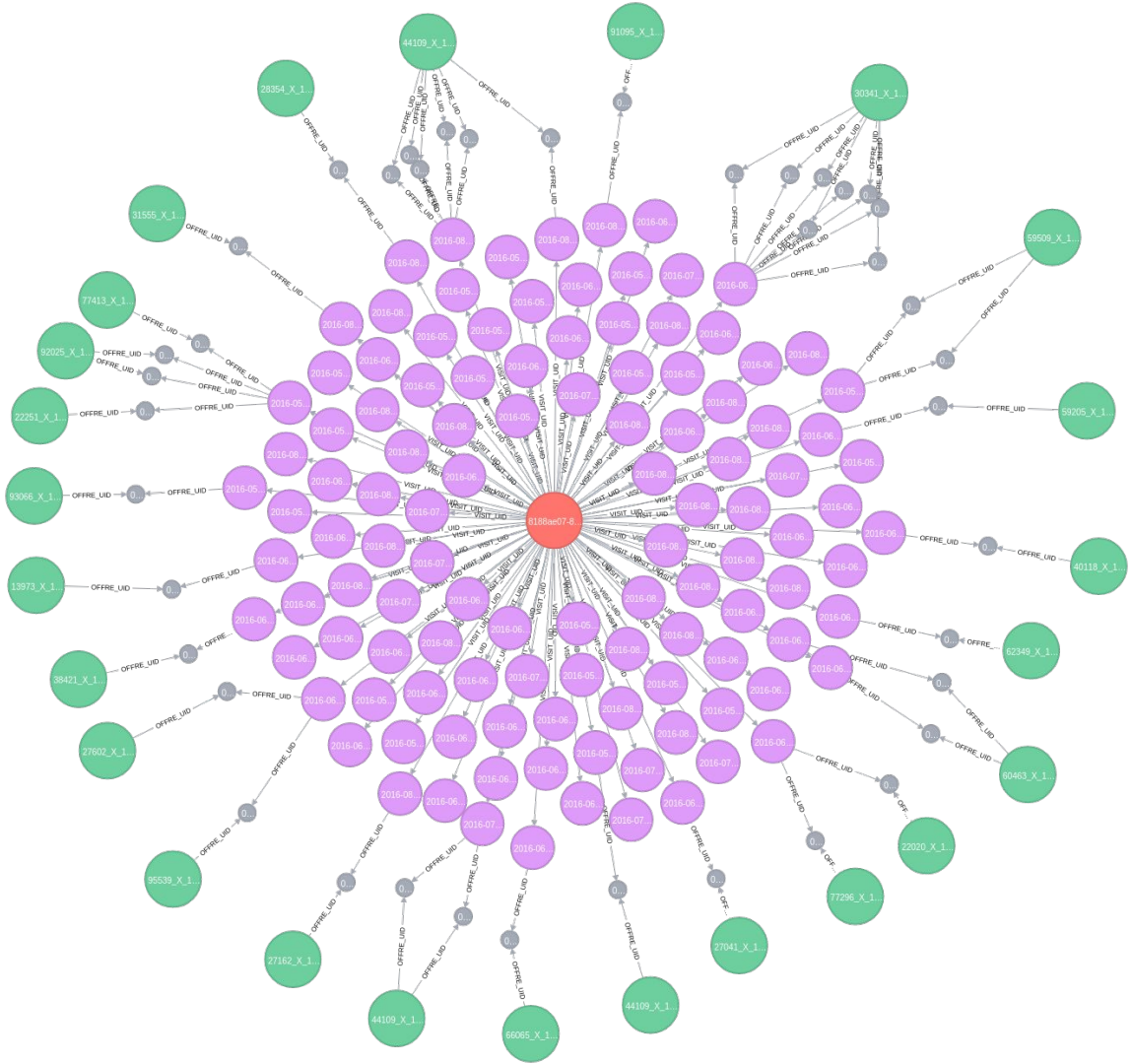
Form

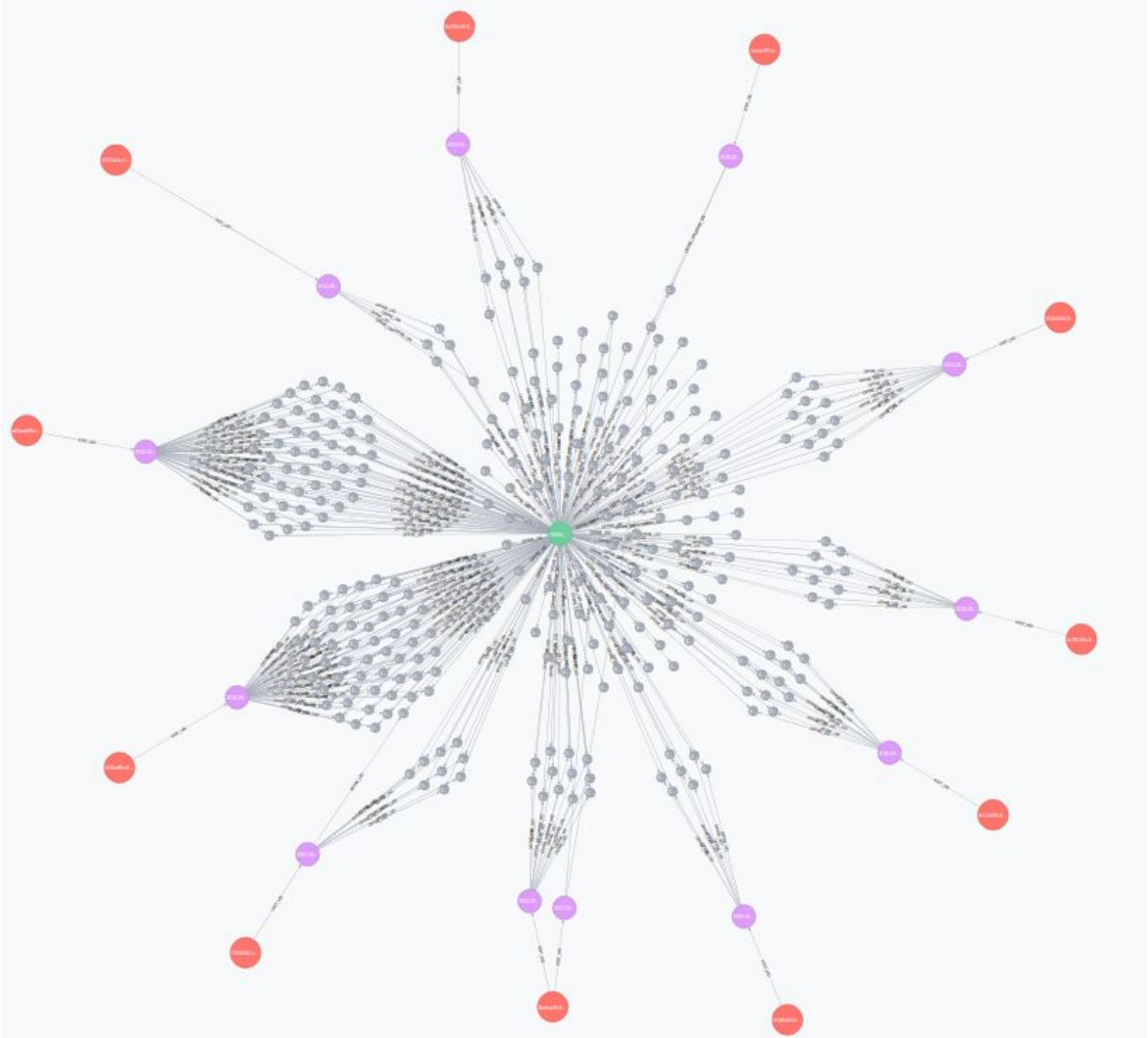
User’s unique identifier

Example


6bccd0...
2cca0d...
VISIT_UID
VISIT_UID
VISIT_UID
2016-0...
2016-0...
2016-0...
OFFRE_UID
OFFRE_UID
OFFRE_UID
OFFRE_UID
OFFRE_...
2...
2...
2...
2...
2...
OFFRE_UID
OFFRE_UID
OFFRE_UID
OFFRE_UID
OFFRE_UID
59410_...
Cookie
Session
Form
Natural key




8188ae07-8...
44109_X_1...
91095_X_1...
28354_X_1...
30341_X_1...
31555_X_1...
59509_X_1...
77413_X_1...
92025_X_1...
59205_X_1...
22251_X_1...
93066_X_1...
40118_X_1...
13973_X_1...
62349_X_1...
38421_X_1...
60463_X_1...
27602_X_1...
22020_X_1...
95539_X_1...
77296_X_1...
27162_X_1...
27041_X_1...
44109_X_1...
66065_X_1...
44109_X_1...
OFFRE_UID
VISIT_UID



Students database

```
// Students nodes
CREATE (andrei:Student {name : "Andrei", hair : "Black"})
CREATE (alexandre:Student {name : "Alexandre", hair : "Brown"})
CREATE (geoffrey:Student {name : "Geoffrey", hair : "Blond"})
CREATE (chafik:Student {name : "Chafik", hair : "Black"})
CREATE (jonathan:Student {name : "Jonathan", hair : "Black"})

// Countries nodes
CREATE (romania:Country {name : "Romania", capitalCity : "Bucharest"})
CREATE (canada:Country {name : "Canada", capitalCity : "Ottawa"})
CREATE (france:Country {name : "France", capitalCity : "Paris"})
CREATE (tunisia:Country {name : "Tunisia", capitalCity : "Tunis"})

// Universities nodes
CREATE (diderot:University {name : "Paris Diderot"})
CREATE (montreal:University {name : "Université de Montréal"})
CREATE (upmc:University {name : "UPMC"})
```

```
// Student --> Country Relationships
CREATE (andrei)-[:FROM]->(romania)
CREATE (alexandre)-[:FROM]->(canada)
CREATE (geoffrey)-[:FROM]->(france)
CREATE (chafik)-[:FROM]->(tunisia)
CREATE (jonathan)-[:FROM]->(france)

// Student --> University Relationships
CREATE (andrei)-[:STUDIED_AT]->(diderot)
CREATE (alexandre)-[:STUDIED_AT]->(montreal)
CREATE (geoffrey)-[:STUDIED_AT]->(diderot)
CREATE (chafik)-[:STUDIED_AT]->(upmc)
CREATE (jonathan)-[:STUDIED_AT]->(upmc)
```