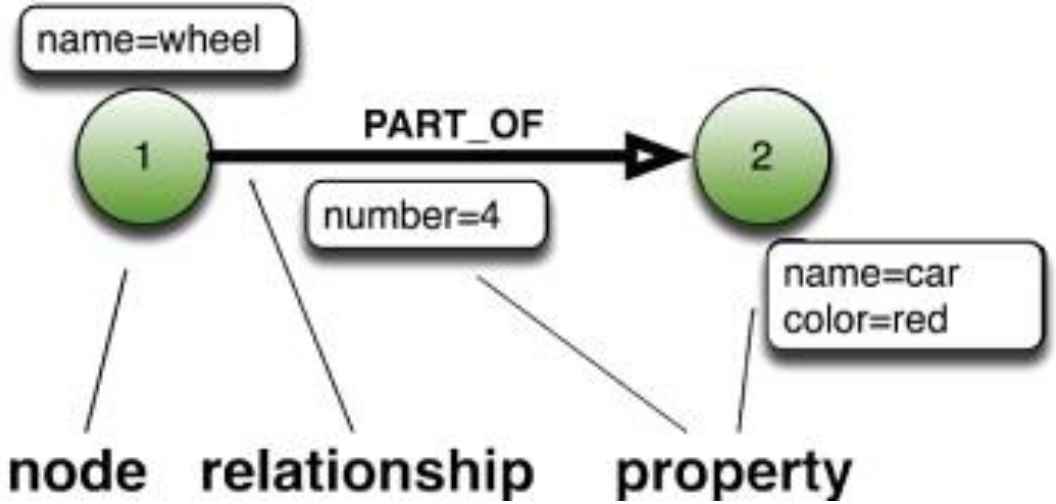
Modèle graphes

- **Principes**
- **Neo4j**
- **OrientDB**
- **Graphes et Bases de Données Relationnelles**

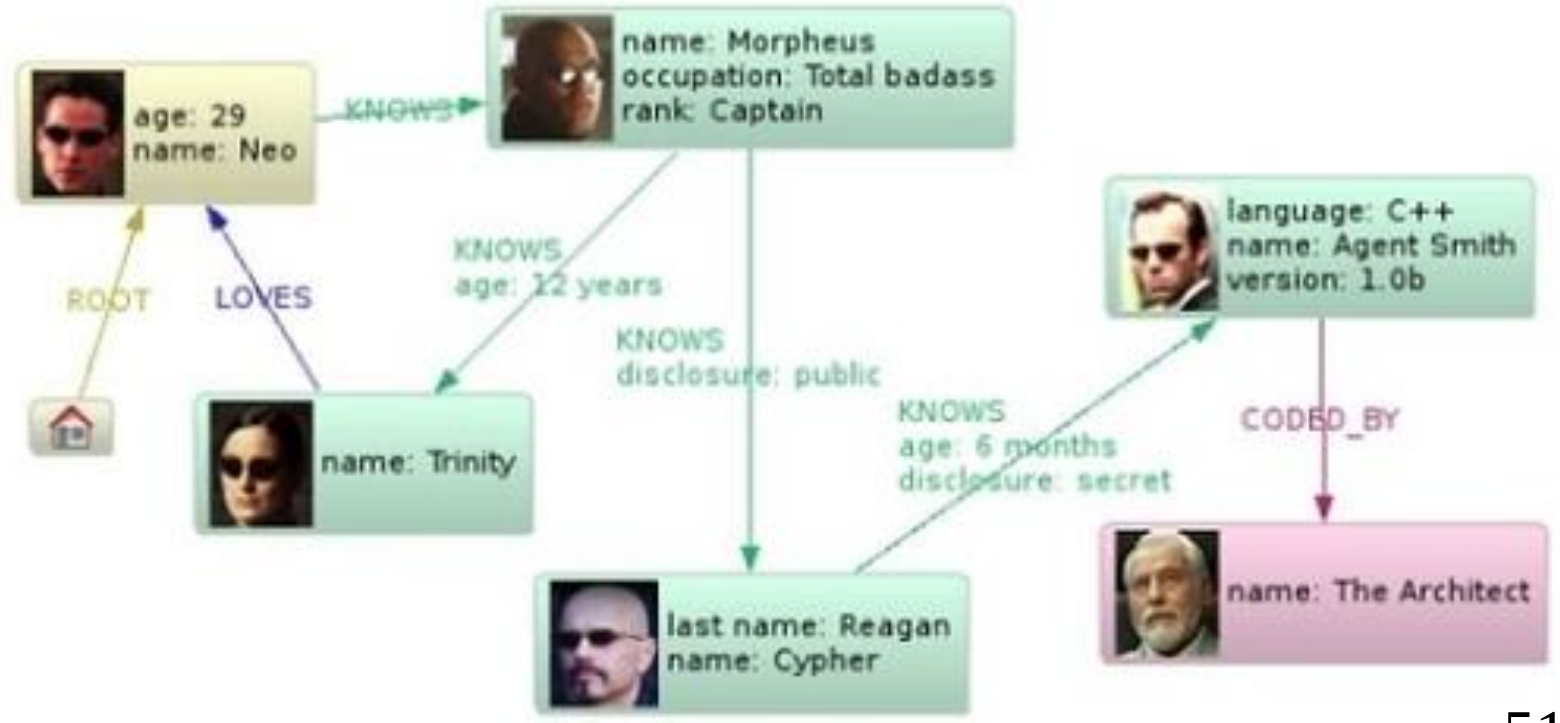
Base de données graphes

- Graphe : représentation d'un réseau
- Graphe : ensemble de nœuds et d'arêtes
- Nœud : représentation d'une entité avec des propriétés
- Arête : directionnelle avec des propriétés
- Exemples de graphes :
  - Réseaux sociaux
  - Métro

Base de données graphes : exemples



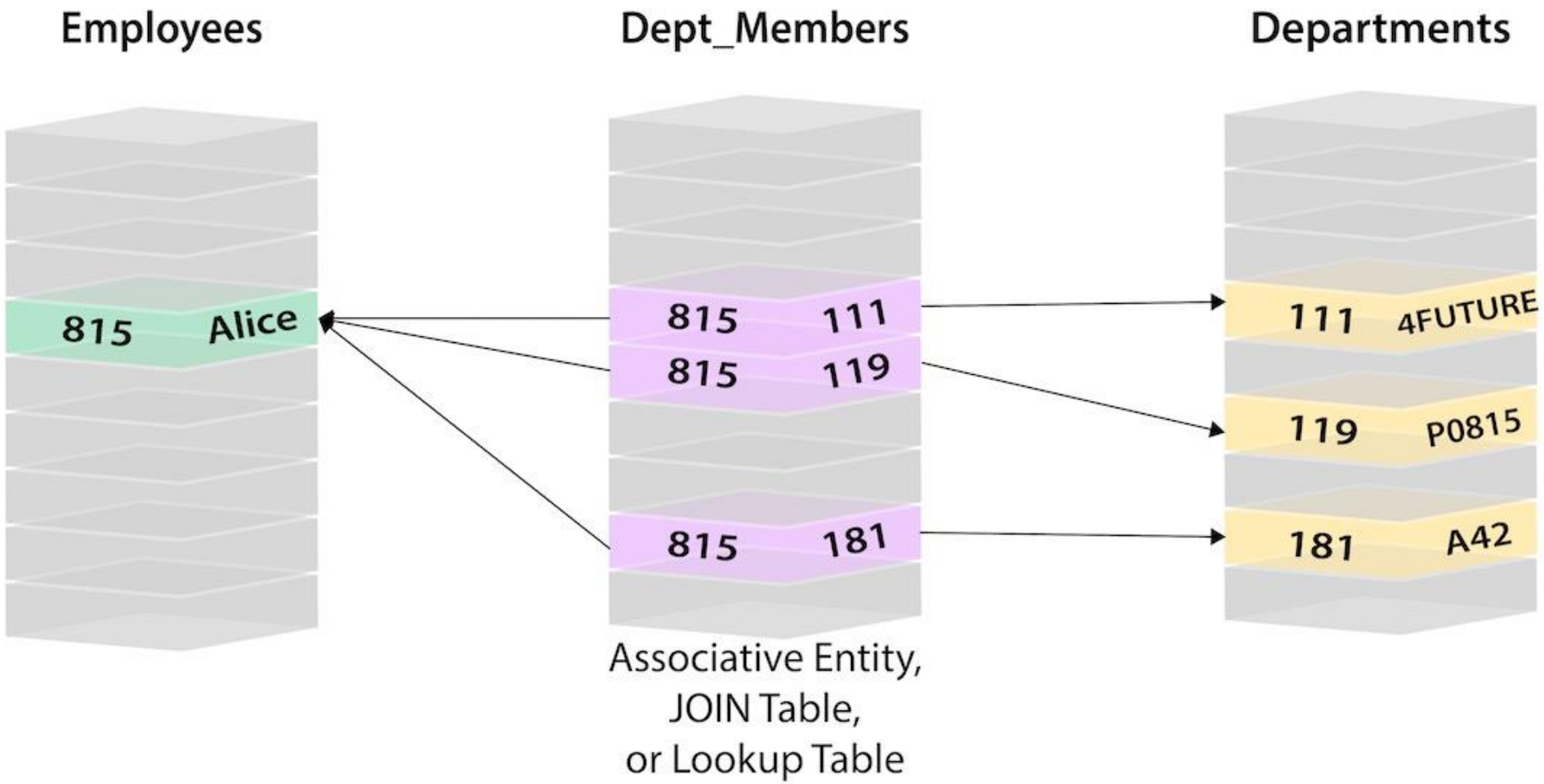
- **Nœud** ou sommet (*node*, *vertex*)
- **Arête** ou **relation** (*relationship*, *edge*), avec une orientation et un type (orienté et marqué)
- **Propriété** ou attribut (*property*, *attribute*), portée par un nœud ou une relation



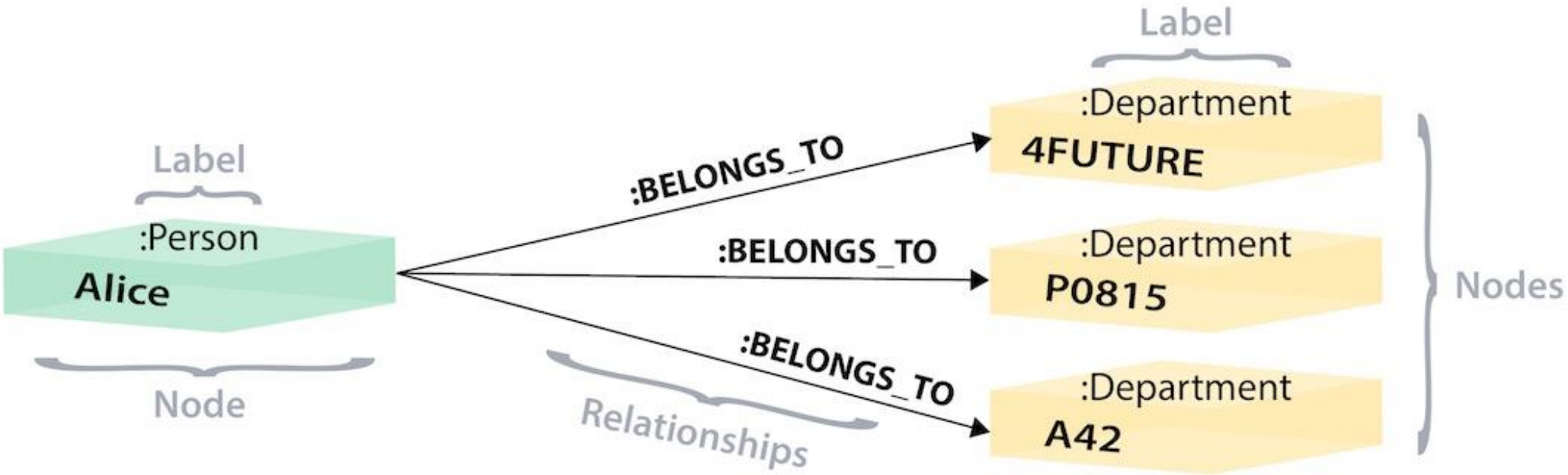
Base de données graphes : cas d'utilisation

- **Cas d'utilisation :** Domaine riche en liens entre des entités (cf. https://neo4j.com/graphgists/)
  - Applications avec des données connectées
  - Applications de routage ou basées sur la localisation
  - Moteurs de recommandation
  - Business Intelligence
- **Non applicable aux:**
  - Applications avec des forts besoins de mises à jour
  - Données non interconnectés

Base de données graphes vs relationnel (1/3)

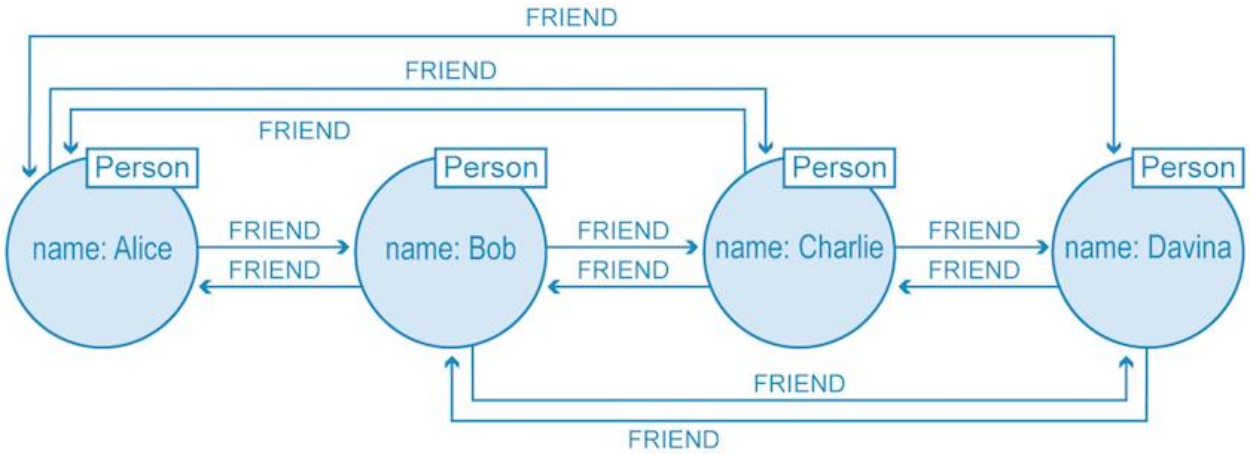


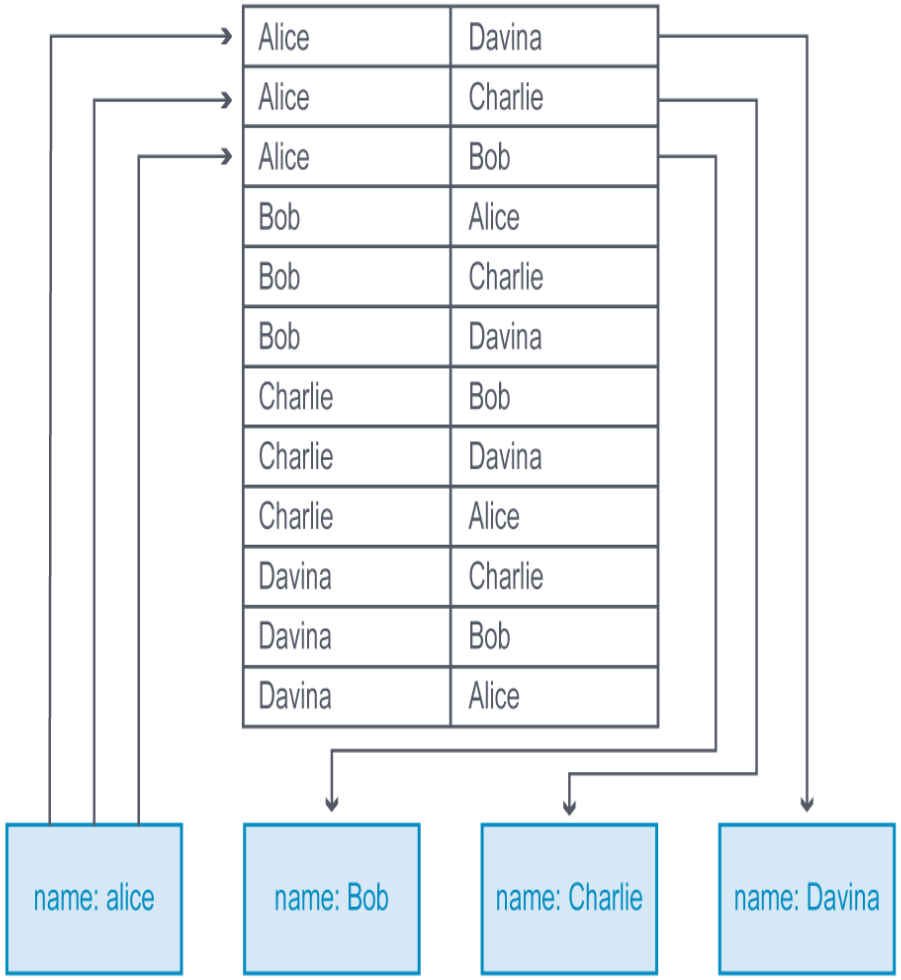
Base de données graphes vs relationnel (2/3)



Base de données graphes vs relationnel (3/3)

Bases de données graphes : *index-free adjacency*





Chaque nœud ~ un micro index des nœuds voisins

Base de données graphes : *index-free adjacency*

- Maintien pour chaque nœud des références directes vers ses nœuds adjacents
- Temps de requêtes généralement indépendantes de la taille totale du graphe et proportionnel à la taille du graphe recherché
- En relationnel : jointures bidirectionnelles précalculées et stockées dans la base de données

Classement des moteurs orientés graphe

☐ include secondary database models — 39 systems in ranking, February 2023

Rank Feb 2023	Rank Jan 2023	Rank Feb 2022	DBMS	Database Model	Score Feb 2023	Score Jan 2023	Score Feb 2022
1.	1.	1.	Neo4j	Graph	55.43	-0.41	-2.81
2.	2.	2.	Microsoft Azure Cosmos DB	Multi-model	36.51	-1.45	-3.45
3.	3.	↑ 4.	Virtuoso	Multi-model	6.11	+0.23	+0.72
4.	4.	↓ 3.	ArangoDB	Multi-model	5.29	+0.22	-0.11
5.	5.	5.	OrientDB	Multi-model	4.54	+0.06	-0.49
6.	↑ 7.	↑ 8.	JanusGraph	Graph	2.79	+0.16	+0.44
7.	↓ 6.	↓ 6.	Amazon Neptune	Multi-model	2.73	-0.08	-0.26
8.	8.	↓ 7.	GraphDB	Multi-model	2.44	-0.09	-0.49
9.	9.	9.	TigerGraph	Graph	2.16	-0.04	-0.08
10.	↑ 11.	↑ 12.	Fauna	Multi-model	1.89	+0.12	+0.57
11.	↓ 10.	11.	Dgraph	Graph	1.86	+0.06	+0.14
12.	↑ 14.	↑ 15.	NebulaGraph	Graph	1.79	+0.28	+0.63
13.	↑ 17.	↑ 20.	Memgraph	Graph	1.68	+0.36	+1.30
14.	↓ 12.	↓ 10.	Stardog	Multi-model	1.63	+0.01	-0.35
15.	↓ 13.	↓ 13.	Giraph	Graph	1.59	+0.06	+0.28

Neo4j

- Système de gestion de graphes par *Neo Technology, Inc*
- Interrogation de la base avec un langage à travers HTTP : *Cypher Query Language*
- Développé en Java et Scala
- Existe depuis 2000
- Peut fonctionner en ***stand-alone*** ou en temps que **serveur web**

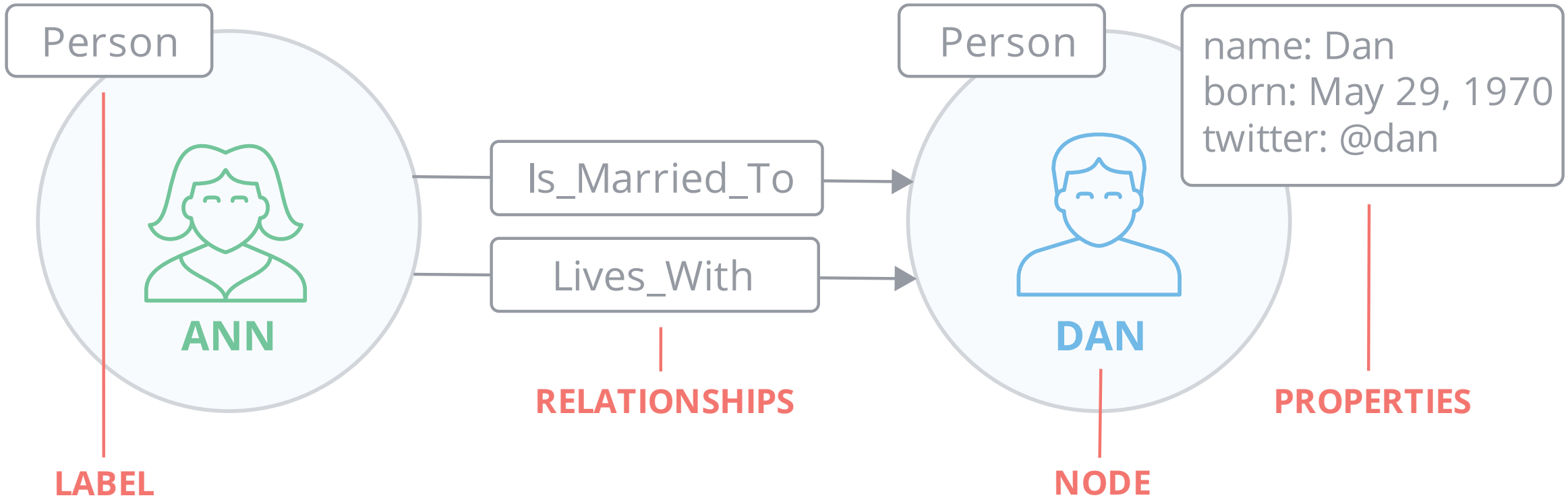
Neo4j : liens utiles

- Site officiel : https://neo4j.com/
- Pour tester en ligne (mais en s'identifiant) : https://neo4j.com/sandbox-v2/?ref=product
- Console en ligne : http://console.neo4j.org/
- Documentation : : https://neo4j.com/docs/
- Exemples d'application :
  - https://neo4j.com/graphgists/
  - Gestion du tour de France : https://neo4j.com/graphgist/modeling-the-tour-de-france-2014-in-a-neo4j-graph-database
- Tutoriels :
  - https://logisima.developpez.com/tutoriel/nosql/neo4j/introduction-neo4j/
  - https://www.tutorialspoint.com/neo4j/
  - https://stph.scenari-community.org/contribs/nos/Neo4j1/co/Neo4j-1_2.html

Neo4j : modèle de données

- Nœud : unité fondamentale d'un graphe, pouvant être associé à des *labels* (~type) et avoir des propriétés
- Relation : connexion entre deux nœuds avec une direction, pouvant avoir des propriétés
- Propriété des nœuds et relations : paires clé-valeur avec des valeurs de type nombre, chaine de caractères, booléen et listes
- Traversée d'un graphe : réponse à une requête (navigation de nœud en nœud exprimée en *Cypher*)
- Chemin : séquence de nœuds avec des relations (utilisé comme résultat d'une requête)

Neo4j : exemple de graphe



Neo4j : Langage *Cypher*

- Deux clauses principales en *Cypher* pour construire des requêtes
  - CREATE pour créer une nouvelle entité
  - MATCH pour chercher/récupérer des entités
- Représentation graphique des entités :
  - Nœud représenté entre parenthèses et deux-points pour *label*
  - Relation représentée avec flèches/tirets et crochets pour détails
  - Propriétés représentées par un dictionnaire à la JSON
- Manuel Cypher : https://neo4j.com/docs/cypher-manual/current/
- Exemples de commandes : https://neo4j.com/graphgist/graphgists-to-learn-the-first-steps-in-the-graph-world-with-the-regesta-of-emperor-frederick-iii

Neo4j : création d'un nœud

```
Query:
CREATE (philip:Person {name:"Philip"})


Query took 12 ms and returned no rows.
Updated the graph - created 1 node set 1 property
```
Result Details

Detailed Query Results

Query Results

```
0 rows
12 ms
+--------------------+
| No data returned. |
+--------------------+
Nodes created: 1
Properties set: 1
```

Execution Plan

```
Compiler CYPHER 3.5

Planner COST

Runtime INTERPRETED

Runtime version 3.5

+-----------------+----------------+------+---------+----------------+------------------+----------------------+-----------+
| Operator        | Estimated Rows | Rows | DB Hits | Page Cache Hits | Page Cache Misses | Page Cache Hit Ratio | Variables |
+-----------------+----------------+------+---------+----------------+------------------+----------------------+-----------+
| +ProduceResults |              1 |    0 |       0 |              0 |                0 |               0.0000 | philip    |
| |               +----------------+------+---------+----------------+------------------+----------------------+-----------+
| +EmptyResult    |              1 |    0 |       0 |              0 |                0 |               0.0000 | philip    |
| |               +----------------+------+---------+----------------+------------------+----------------------+-----------+
| +Create         |              1 |    1 |       4 |              0 |                0 |               0.0000 | philip    |
+-----------------+----------------+------+---------+----------------+------------------+----------------------+-----------+

Total database accesses: 4
```

Nom de variable non obligatoire

```
CREATE (:Person { name:'Philip' })
```

Les nœuds d'un même *label* n'ont pas forcement la même structure :

```
CREATE (:Person { name:'Emil', Age:25 })
CREATE (:Person { name:'Jeanne' })
```

Neo4j : création d'une relation

```
Query:
CREATE (philip:Person {name:"Philip"})-[:IS_FRIEND_OF]->(emil:Person {name:"Emil"})


Query took 79 ms and returned no rows.
Updated the graph - created 2 nodes and 1 relationship set 2 properties
```

Result Details

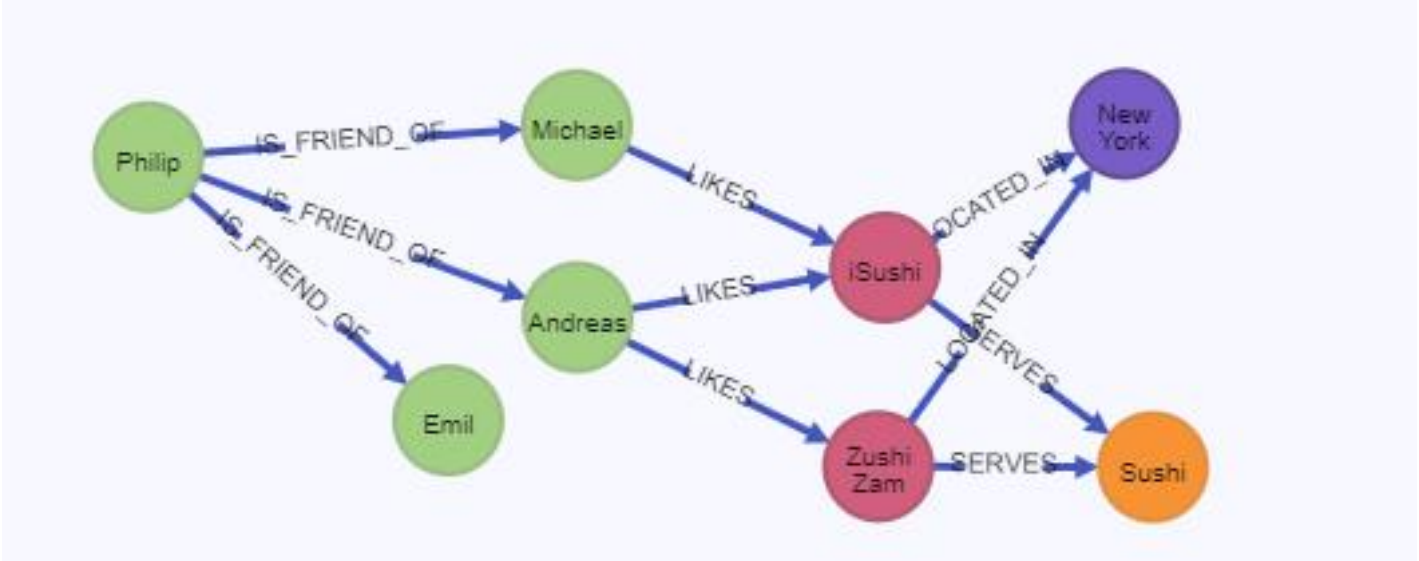
Inutile de créer un arc dans les 2 sens, lors des requêtes le sens des arcs peut être omis.

Autre manière d'écrire l'instruction :

```
CREATE (Philip:Person { name:'Philip' }), (Emil:Person { name:'Emil' }), (Philip)-[:IS_FRIEND_OF]->(Emil)
```

Neo4j : création d'un graphe

```
CREATE (philip:Person {name:"Philip"})-[:IS_FRIEND_OF]->(emil:Person {name:"Emil"}),
       (philip)-[:IS_FRIEND_OF]->(michael:Person {name:"Michael"}),
       (philip)-[:IS_FRIEND_OF]->(andreas:Person {name:"Andreas"})
CREATE (sushi:Cuisine {name:"Sushi"}), (nyc:City {name:"New York"}),
       (iSushi:Restaurant {name:"iSushi"})-[:SERVES]->(sushi),(iSushi)-[:LOCATED_IN]->(nyc),
       (michael)-[:LIKES]->(iSushi),
       (andreas)-[:LIKES]->(iSushi),
       (zam:Restaurant {name:"Zushi Zam"})-[:SERVES]->(sushi),(zam)-[:LOCATED_IN]->(nyc),
       (andreas)-[:LIKES]->(zam)
```



Neo4j : *CREATE*

```
Query:
CREATE (philip:Person {name:"Philip"})

Query took 0 ms and returned no rows.
Updated the graph - created 1 node set 1 property Result Details

Query:
CREATE (philip:Person {name:"Philip"})

Query took 1 ms and returned no rows.
Updated the graph - created 1 node set 1 property Result Details

Query:
CREATE (philip:Person {name:"Philip"})
```

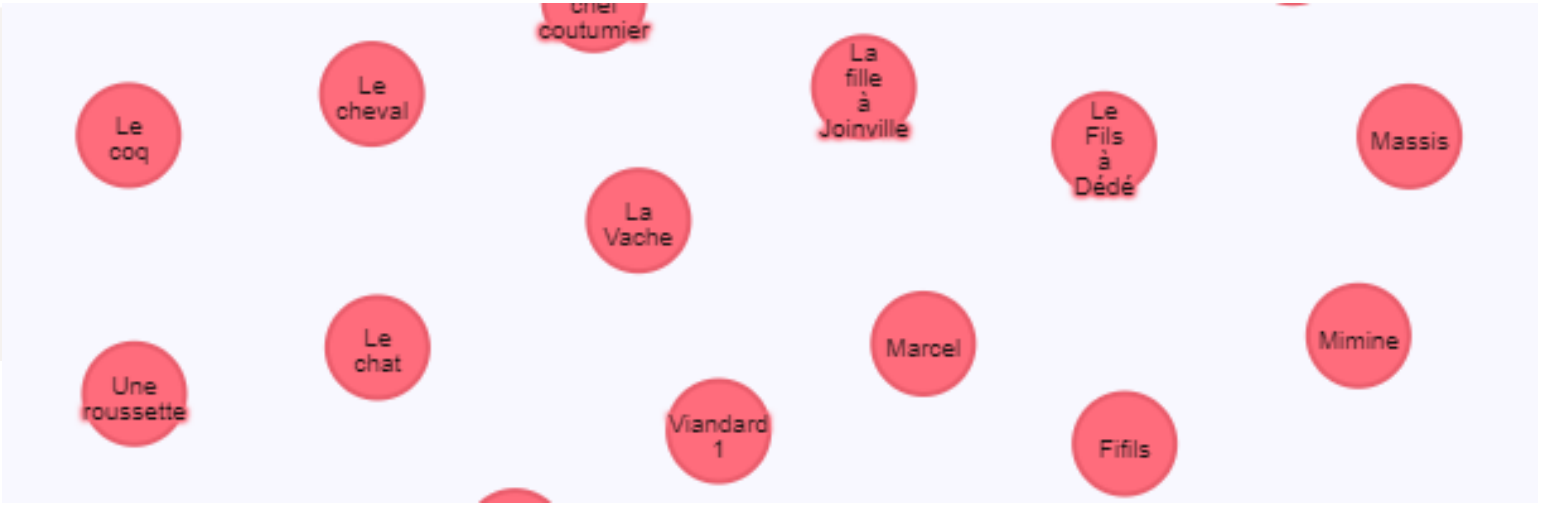
Attention : rien n'empêche de créer plusieurs fois le « même » nœud.

Neo4j : Créer des nœuds à partir d'un fichier csv (1/2)

Query 1

```
LOAD CSV WITH HEADERS FROM "https://raw.githubusercontent.com/adriens/brousse-en-folie-network/master/nodes.csv" AS csvLine
CREATE (p:Hero {id: csvLine.id, name: csvLine.label});
```

```
0 rows
93 ms
+--------------------+
| No data returned. |
+--------------------+
Nodes created: 30
Properties set: 60
```



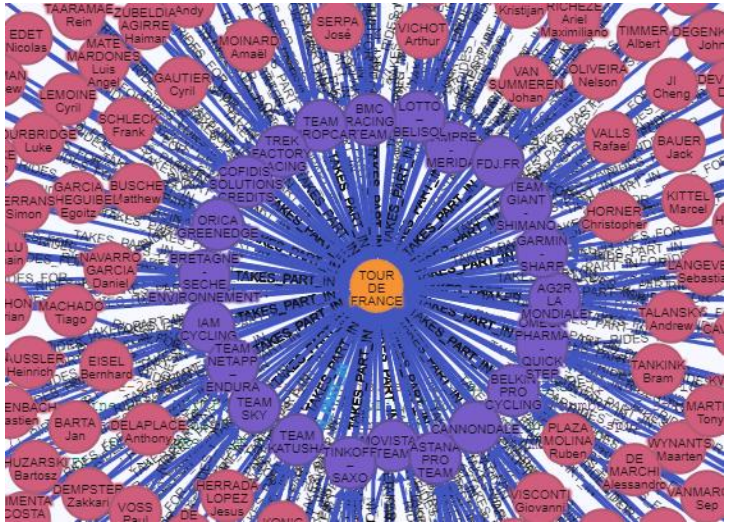
Extrait du contenu du fichier csv
https://github.com/adriens/brousse-en-folie-network/blob/master/nodes.csv

id	label
dede	Dédé
tathan	Tathan
joinville	Joinville
marcel	Marcel
lecteur	Le Lecteur

Neo4j : Créer un graphe à partir d'un fichier csv (2/2)

```
LOAD CSV WITH HEADERS FROM "https://raw.githubusercontent.com/inserpio/tour-de-
france-2014/master/tour-de-france-2014-0001-teams-and-riders.csv" AS csvLine
MERGE (r:Race { id: toInt(csvLine.RACE_ID), name: csvLine.RACE_NAME, from:
csvLine.RACE_FROM, to: csvLine.RACE_TO, edition: csvLine.RACE_EDITION, distance:
csvLine.RACE_DISTANCE, number_of_stages: csvLine.RACE_NUMBER_OF_STAGES, website:
csvLine.RACE_WEBSITE })
MERGE (t:Team { id: toInt(csvLine.TEAM_ID), name: csvLine.TEAM_NAME, country:
csvLine.TEAM_COUNTRY, sportingDirectors: csvLine.TEAM_MANAGERS })
MERGE (p:Rider { name: csvLine.RIDER_NAME, country: csvLine.RIDER_COUNTRY })
CREATE (t)-[:TAKES_PART_IN]->(r)<-[:TAKES_PART_IN { number:
toInt(csvLine.RIDER_NUMBER), info: csvLine.RIDER_INFO }]-(p), (p)-[:RIDES_FOR {
year: toInt(csvLine.RACE_YEAR) }]->(t);
```

Pour voir le contenu du fichier :
https://raw.githubusercontent.com/inserpio/tour-de-france-2014/master/tour-de-france-2014-0001-teams-and-riders.csv



Neo4j : recherche de nœuds

```
Query:
MATCH (p:Person {name:'Philip'}) RETURN p
```

p
(0:Person {name:"Philip"})

Query took 18 ms and returned 1 rows. Result Details

Detailed Query Results

Query Results

```
+-------------------------+
| p                       |
+-------------------------+
| Node[0]{name:"Philip"} |
+-------------------------+
1 row
18 ms
```

Execution Plan

```
Compiler CYPHER 3.5

Planner COST

Runtime INTERPRETED

Runtime version 3.5

+------------------+----------------+------+---------+----------------+----------------+---------------------+-----------+--------------------------+
| Operator         | Estimated Rows | Rows | DB Hits | Page Cache Hits | Page Cache Misses | Page Cache Hit Ratio | Variables | Other                    |
+------------------+----------------+------+---------+----------------+----------------+---------------------+-----------+--------------------------+
| +ProduceResults  |              0 |    1 |       0 |              0 |                0 |              0.0000 | p         |                          |
| |                +----------------+------+---------+----------------+----------------+---------------------+-----------+--------------------------+
| +Filter          |              0 |    1 |       4 |              0 |                0 |              0.0000 | p         | p.name = $`  AUTOSTRING0` |
| |                +----------------+------+---------+----------------+----------------+---------------------+-----------+--------------------------+
| +NodeByLabelScan |              4 |    4 |       5 |              0 |                0 |              0.0000 | p         | :Person                  |
+------------------+----------------+------+---------+----------------+----------------+---------------------+-----------+--------------------------+

Total database accesses: 9
```

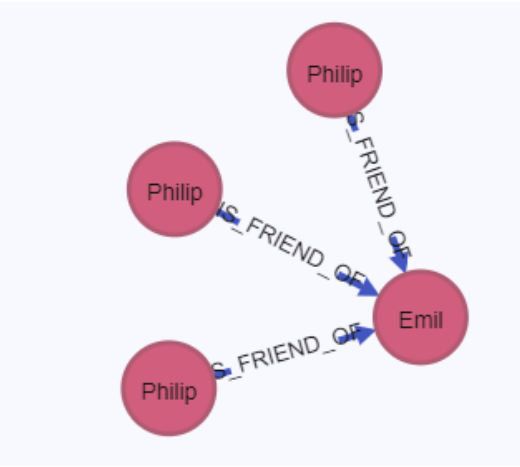
Neo4j : CREATE et MATCH (1/2)

```
CREATE (Philip:Person { name:'Philip' }), (Emil:Person { name:'Emil' })
```


```
MATCH (Philip:Person { name:'Philip' }), (Emil:Person { name:'Emil' }) CREATE (Philip)-[:IS_FRIEND_OF]->(Emil)
```

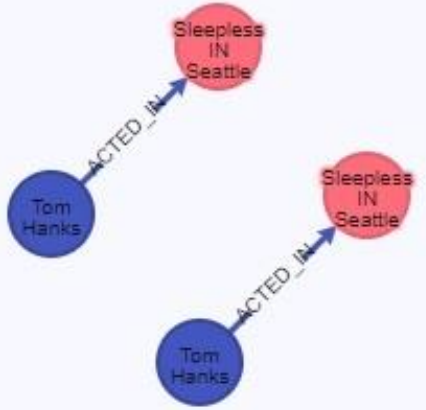
Attention si vous avez plusieurs nœuds répondant au *MATCH*


```
MATCH (Philip:Person { name:'Philip' }), (Emil:Person { name:'Emil' }) CREATE (Philip)-[:IS_FRIEND_OF]->(Emil)
```



Neo4j : CREATE et MATCH (2/2)

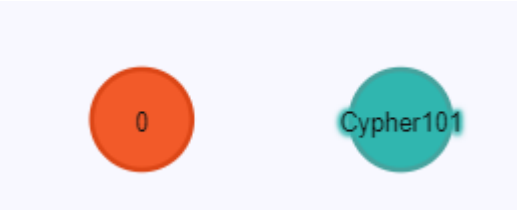
```
Query:
CREATE (n:Actor { name:'Tom Hanks' });

Query took 35 ms and returned no rows.
Updated the graph - created 1 node set 1 property Result Details

Query:
CREATE (n:Actor { name:'Tom Hanks' });

Query took 1 ms and returned no rows.
Updated the graph - created 1 node set 1 property Result Details

Query:
MATCH (actor:Actor)  WHERE actor.name = 'Tom Hanks'  CREATE (movie:Movie { title:'Sleepless IN Seattle' })  CREATE (actor)-[:ACTED_IN]->(movie);

Query took 34 ms and returned no rows.
Updated the graph - created 2 nodes and 2 relationships set 2 properties Result Details
```




```
MATCH (actor:Actor)  WHERE actor.name = 'Tom Hanks'  CREATE (movie:Movie { title:'Sleepless IN Seattle' })  CREATE (actor)-[:ACTED_IN]->(movie);
```

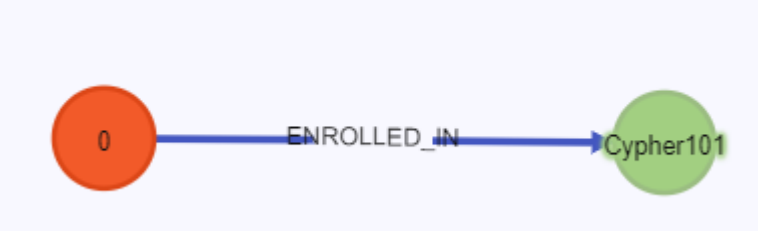
Neo4j : CREATE et MERGE (1/2)

```
CREATE (student:Student{id:123})
CREATE  (class:Class{name:'Cypher101'})
```

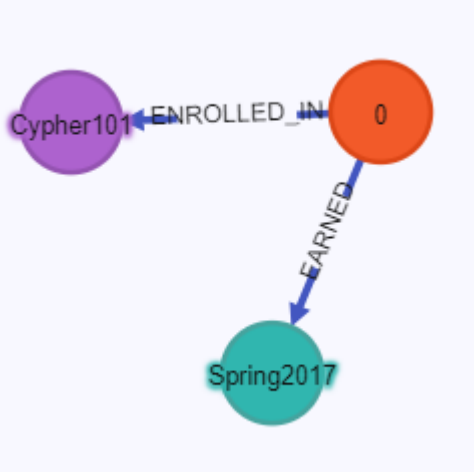


Si on réitère ces instructions on peut créer plusieurs nœuds « identiques »

```
MATCH (student:Student{id:123})
MATCH (class:Class{name:'Cypher101'})
MERGE (student)-[:ENROLLED_IN]->(class)
```



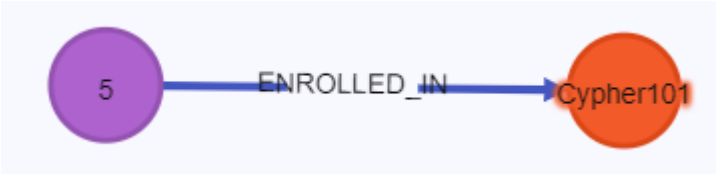

```
MATCH (student:Student{id:123})
MERGE (reportCard:ReportCard{term:'Spring2017'})
MERGE (student)-[:EARNED]->(reportCard)
```



`MERGE` récupère un nœud s'il existe déjà et le crée sinon

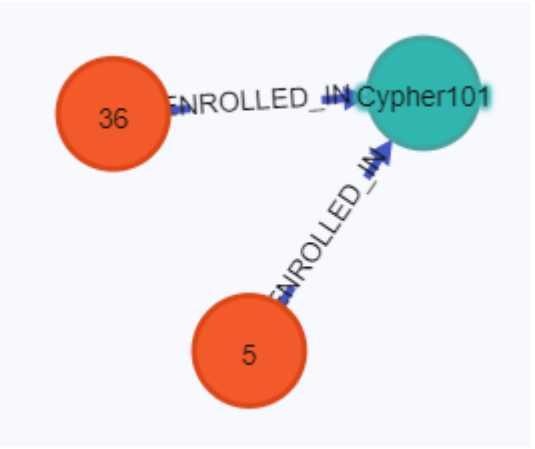
Neo4j : CREATE et MERGE (2/2)

```
CREATE (student:Student{id:123})-[:ENROLLED_IN]->(class:Class{name:'Cypher101'})
```




```
Query:
MATCH (student:Student{id:222}) MATCH (class:Class{name:'Cypher101'}) MERGE (student)-[:ENROLLED_IN]->(class)

Query took 6 ms and returned no rows. Result Details
```

```
MERGE (student:Student{id:222}) MERGE (class:Class{name:'Cypher101'}) MERGE (student)-[:ENROLLED_IN]->(class)
```



`MERGE` récupère un nœud s'il existe (~`MATCH`) ou le crée s'il n'existe pas (~`CREATE`)

Neo4j : recherche de nœuds / sous-graphes (1/2)

```
Query:
MATCH (r:Restaurant) WHERE r.name='iSushi' RETURN r
```

r
(75:Restaurant {name:"iSushi"})

Query took 22 ms and returned 1 rows. Result Details

```
Query:
MATCH (philip:Person {name:'Philip'})-[:IS_FRIEND_OF]-(person) RETURN person.name
```

person.name
Michael
Andreas
Emil

Query took 68 ms and returned 3 rows. Result Details

```
Query:
MATCH (nyc:City {name:'New York'})<-[:LOCATED_IN]-(restaurant)-[:SERVES]->(cuisine) RETURN nyc, restaurant, cuisine
```

nyc	restaurant	cuisine
(74:City {name:"New York"})	(76:Restaurant {name:"Zushi Zam"})	(73:Cuisine {name:"Sushi"})
(74:City {name:"New York"})	(75:Restaurant {name:"iSushi"})	(73:Cuisine {name:"Sushi"})

Query took 197 ms and returned 2 rows. Result Details

Neo4j : mise à jour de nœuds / sous-graphes (2/2)

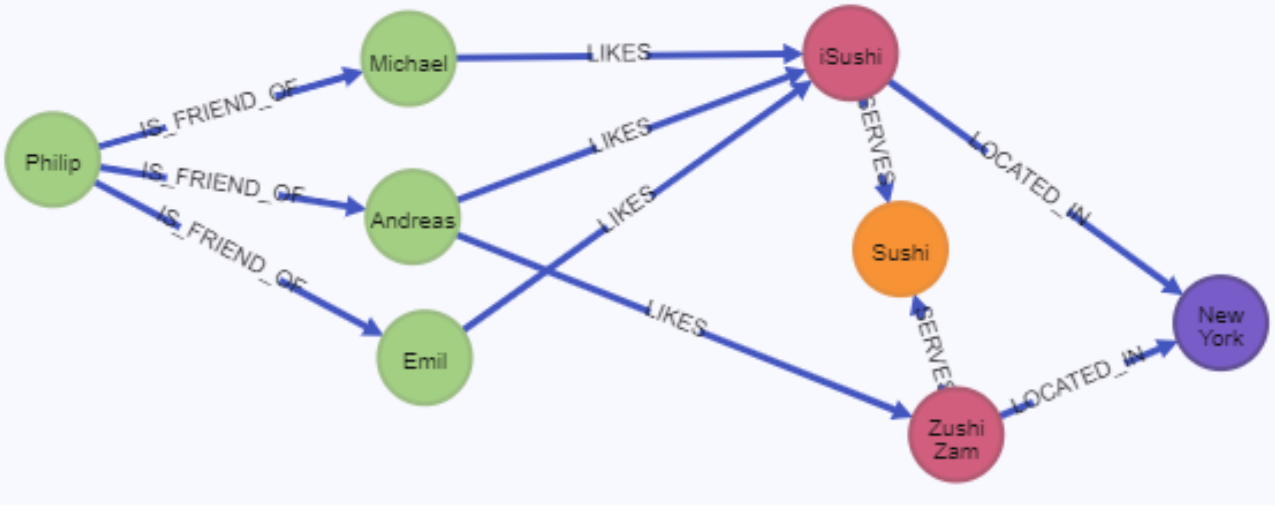
```
Query:
MATCH (p:Person {name:'Philip'}) SET p.Age= 47 RETURN p
```

p
(0:Person {Age:47, name:"Philip"})

Query took 67 ms and returned 1 rows.
Updated the graph - set 1 property Result Details

```
Query:
MATCH (p:Person {name:'Emil'}) MATCH (r:Restaurant {name:'iSushi'}) CREATE (p)-[:LIKES]->(r)
```

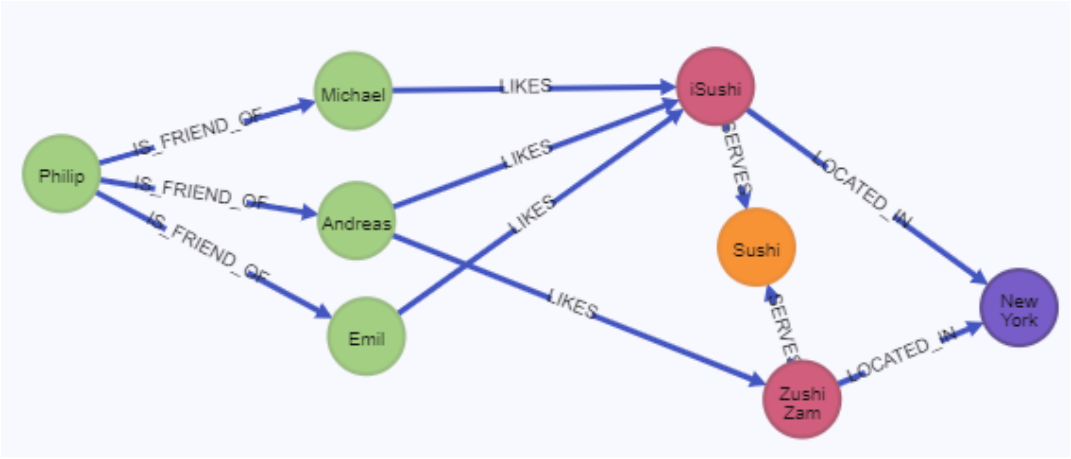
Query took 77 ms and returned no rows.
Updated the graph - created 1 relationship Result Details



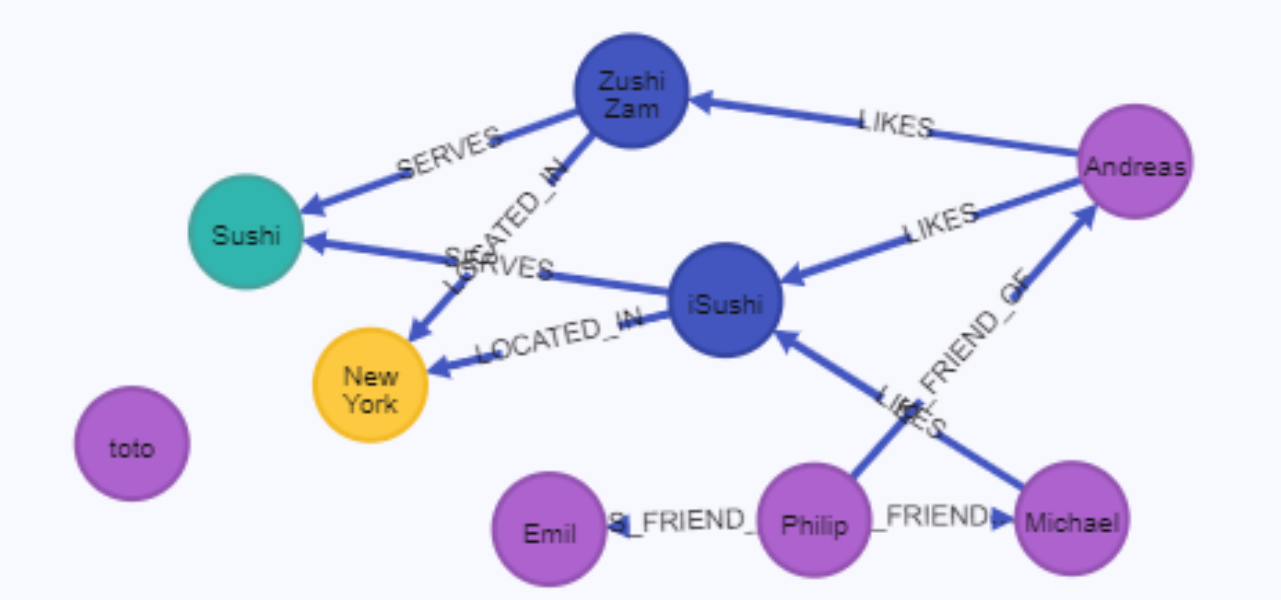
Neo4j : requêtes complexes

```
MATCH (philip:Person {name: 'Philip'}),
      (philip)-[:IS_FRIEND_OF]-(friend),
      (restaurant:Restaurant)-[:LOCATED_IN]->(:City {name: 'New York'}),
      (restaurant)-[:SERVES]->(:Cuisine {name: 'Sushi'}),
      (friend)-[:LIKES]->(restaurant)
RETURN restaurant.name AS restaurantName, collect(friend.name) AS recommendedBy, count(*) AS numberOfRecommendations
ORDER BY numberOfRecommendations DESC
```

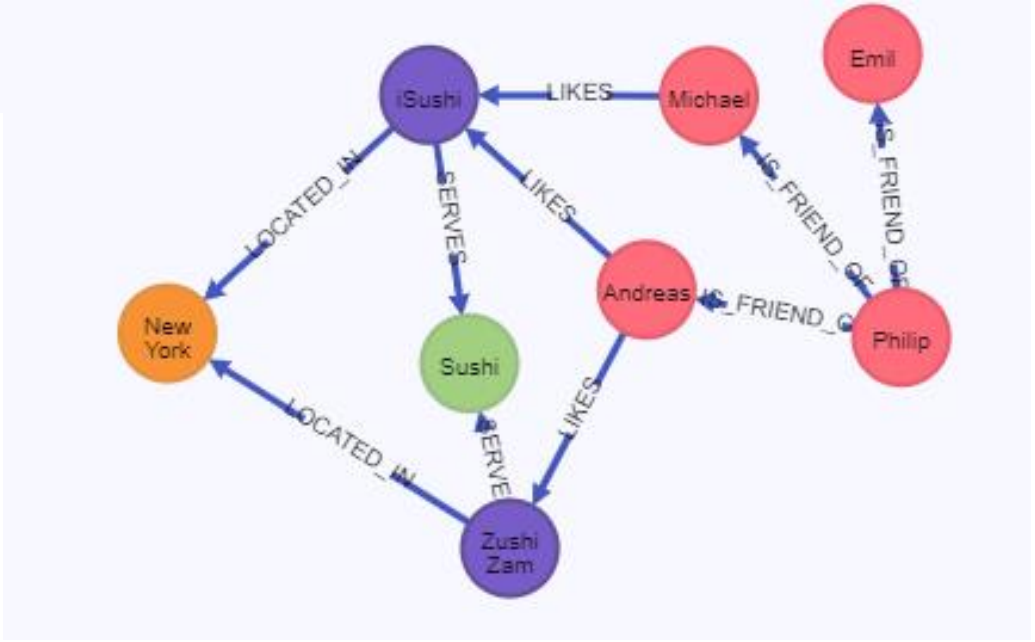
restaurantName	recommendedBy	numberOfRecommendations
iSushi	[Emil, Andreas, Michael]	3
Zushi Zam	[Andreas]	1



Neo4j : supprimer un nœud sans relation (1/2)




```
Query:
MATCH (n:Person {name:"toto"}) DELETE n

Query took 29 ms and returned no rows.
Updated the graph - deleted 1 node  Result Details
```



Neo4j : supprimer un nœud sans relation (2/2)

Pour supprimer tous les nœuds d'un même *label* : `MATCH (p:Person) DELETE p`

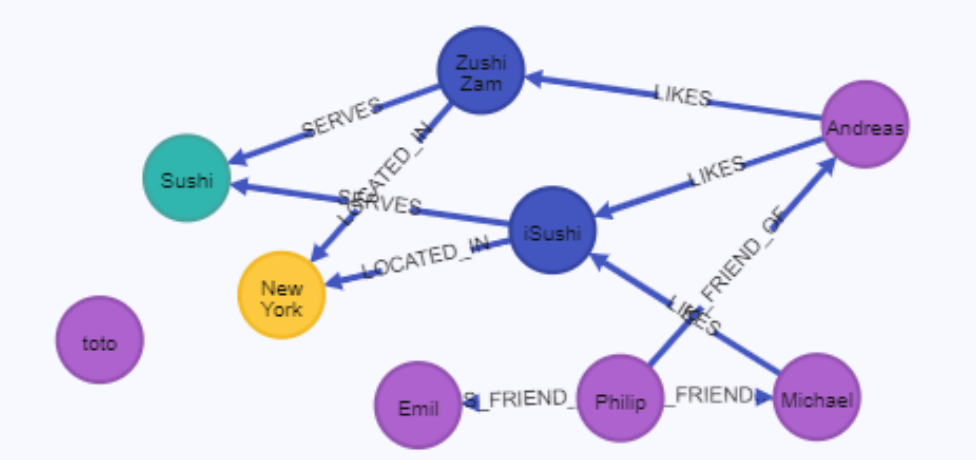
Erreur si les nœuds sont liés à d'autres :

```
Query:
CREATE (Philip:Person { name:'Philip' }), (Emil:Person { name:'Emil' }), (Philip)-[:IS_FRIEND_OF]->(Emil)

Query took 4 ms and returned no rows.
Updated the graph - created 2 nodes and 1 relationship set 2 properties Result Details
```

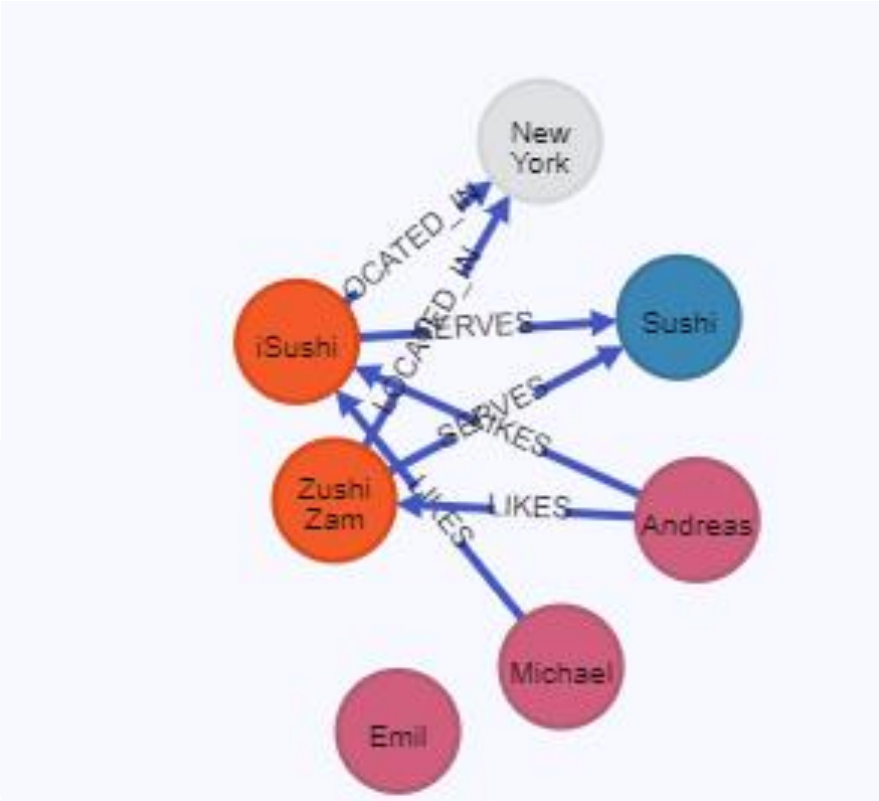

```
Query:
MATCH (p:Person) DELETE p
Error: org.neo4j.graphdb.ConstraintViolationException: Cannot delete node<0>, because it still has relationships. To delete this node, you must first delete its relationships.
```

Neo4j : supprimer un nœud avec relation

```
Query:
MATCH (n:Person {name:"Philip"}) DELETE n
Error: org.neo4j.graphdb.ConstraintViolationException: Cannot delete node<151>, because it still has relationships. To delete this node, you must first delete its relationships.
```




```
Query:
MATCH (n:Person {name:"Philip"}) DETACH DELETE n


Query took 30 ms and returned no rows.
Updated the graph - deleted 1 node and 3 relationships  Result Details
```



Neo4j : comparaison SQL / Cypher

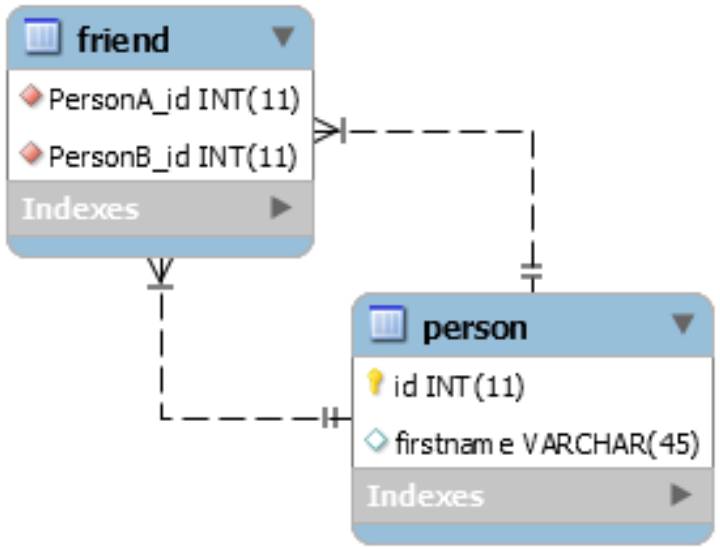
SQL Statement

```
SELECT name FROM Person
LEFT JOIN Person_Department
  ON Person.Id = Person_Department.PersonId
LEFT JOIN Department
  ON Department.Id = Person_Department.DepartmentId
WHERE Department.name = "IT Department"
```

Cypher Statement

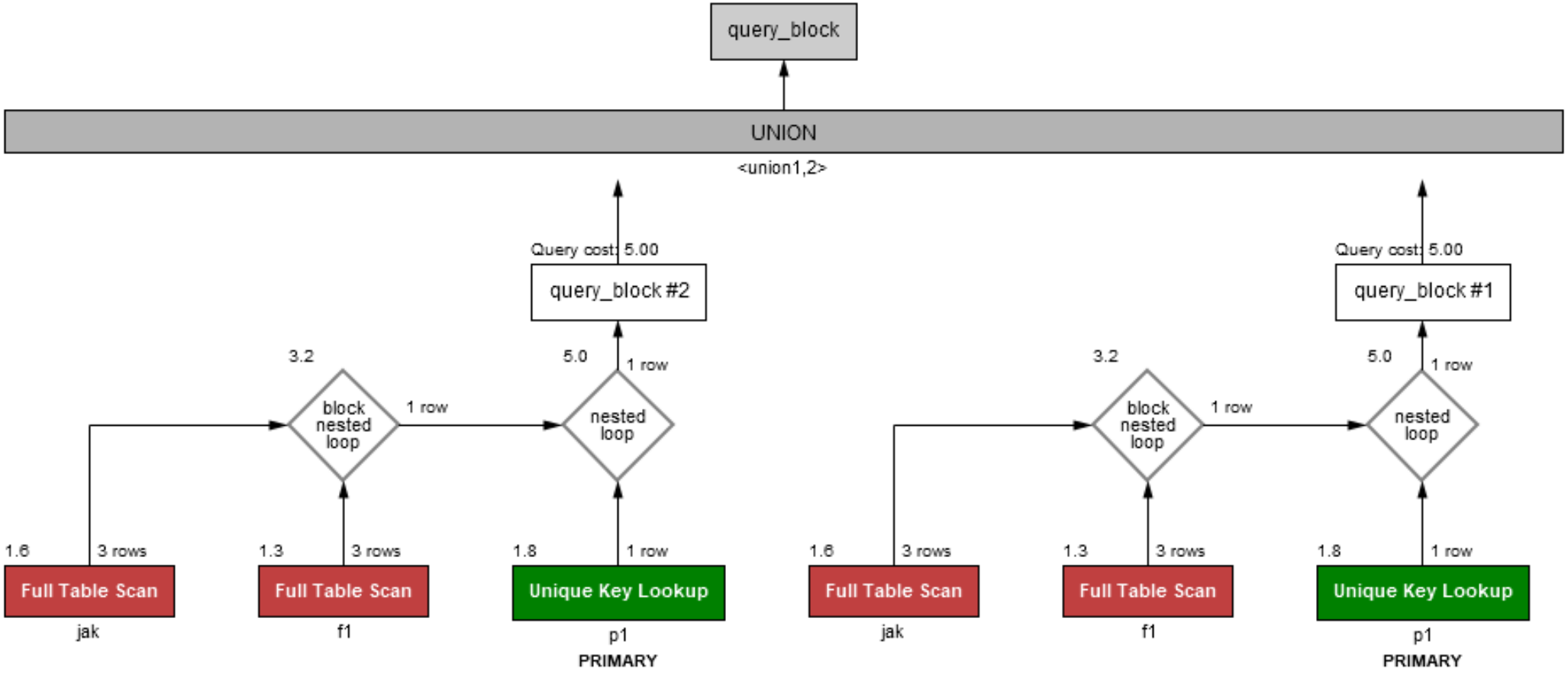
```
MATCH (p:Person)-[:WORKS_AT]->(d:Dept)
WHERE d.name = "IT Department"
RETURN p.name
```

Neo4j : exemple d'union en SQL




```sql
SELECT p1.firstname FROM person jak
INNER JOIN friend f1
ON f1.PersonA_id = jak.id
INNER JOIN person p1
ON p1.id = f1.PersonB_id
WHERE jak.firstname = "jak"

UNION

SELECT p1.firstname FROM person jak
INNER JOIN friend f1
ON f1.PersonB_id = jak.id
INNER JOIN person p1
ON p1.id = f1.PersonA_id
WHERE jak.firstname = "jak"
```

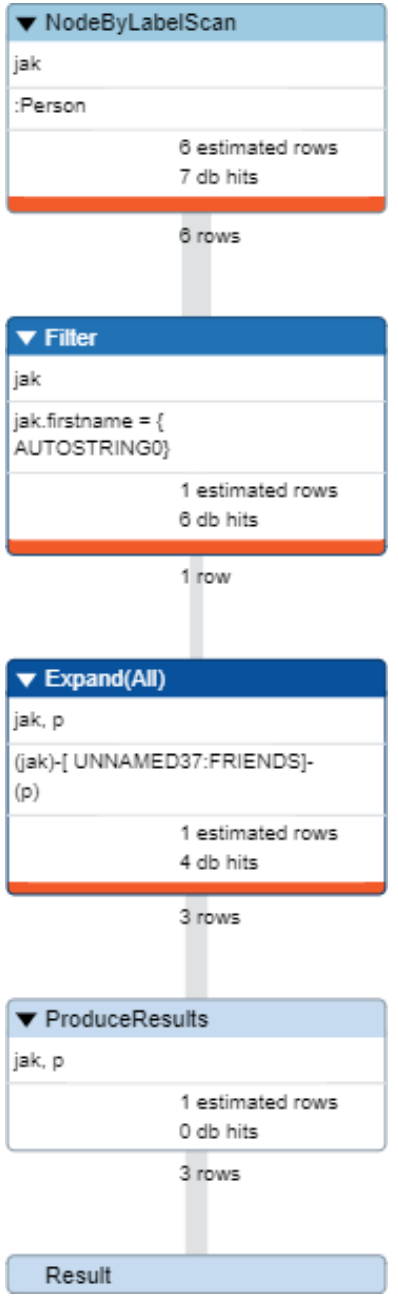
Neo4j : plan d'exécution de l'union en SQL



Neo4j : la même requête *Cypher*

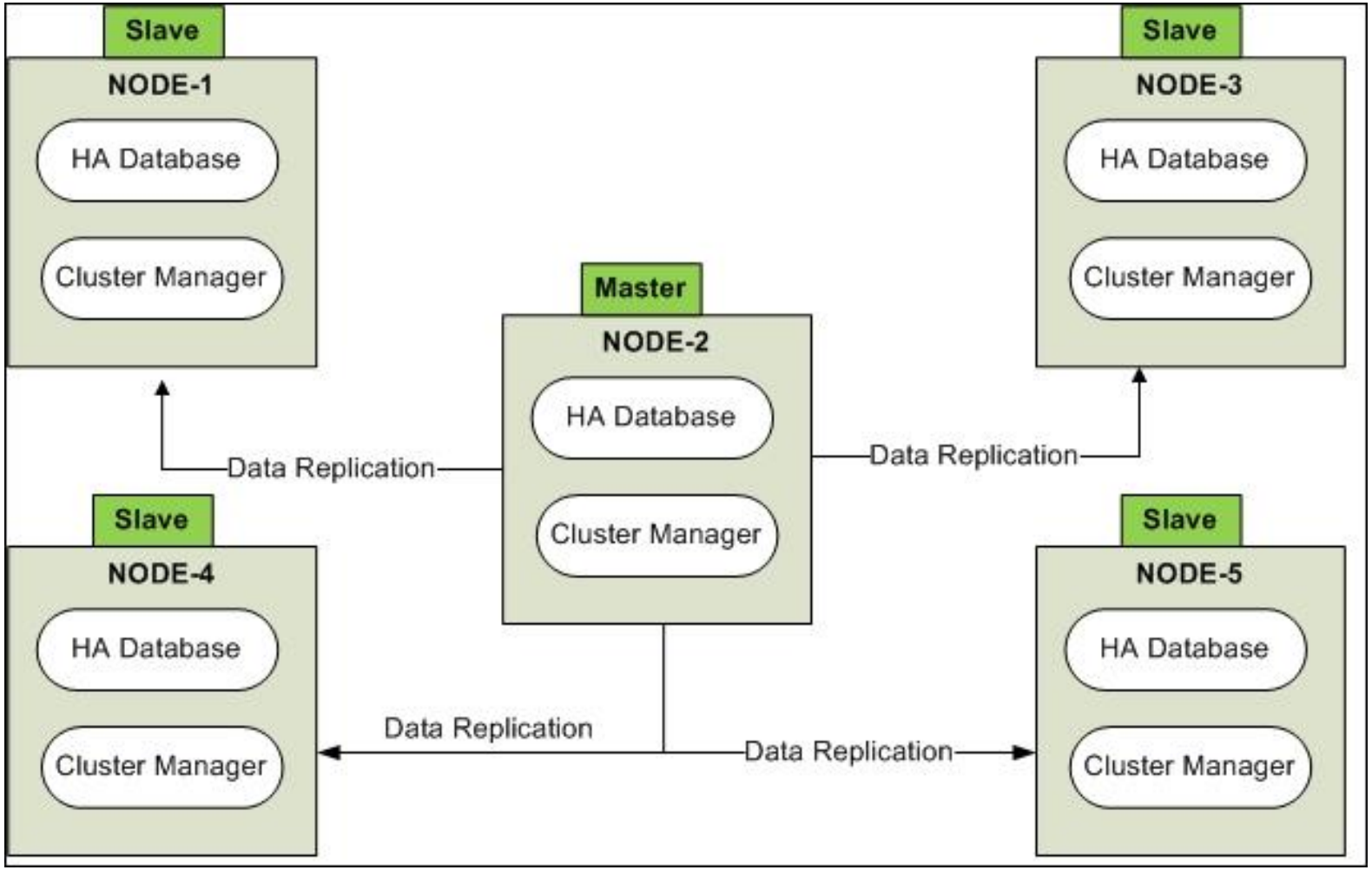
```
MATCH (jak:Person {firstname:"jak"})-[:FRIENDS]-(p)
RETURN p
```

- Recherche par *label*
- Filtrage de recherche de la personne ayant pour prénom "`jak`" appliqué directement (filtrage au plus tôt)
- Extension avec les nœuds associés par la relation `FRIENDS`

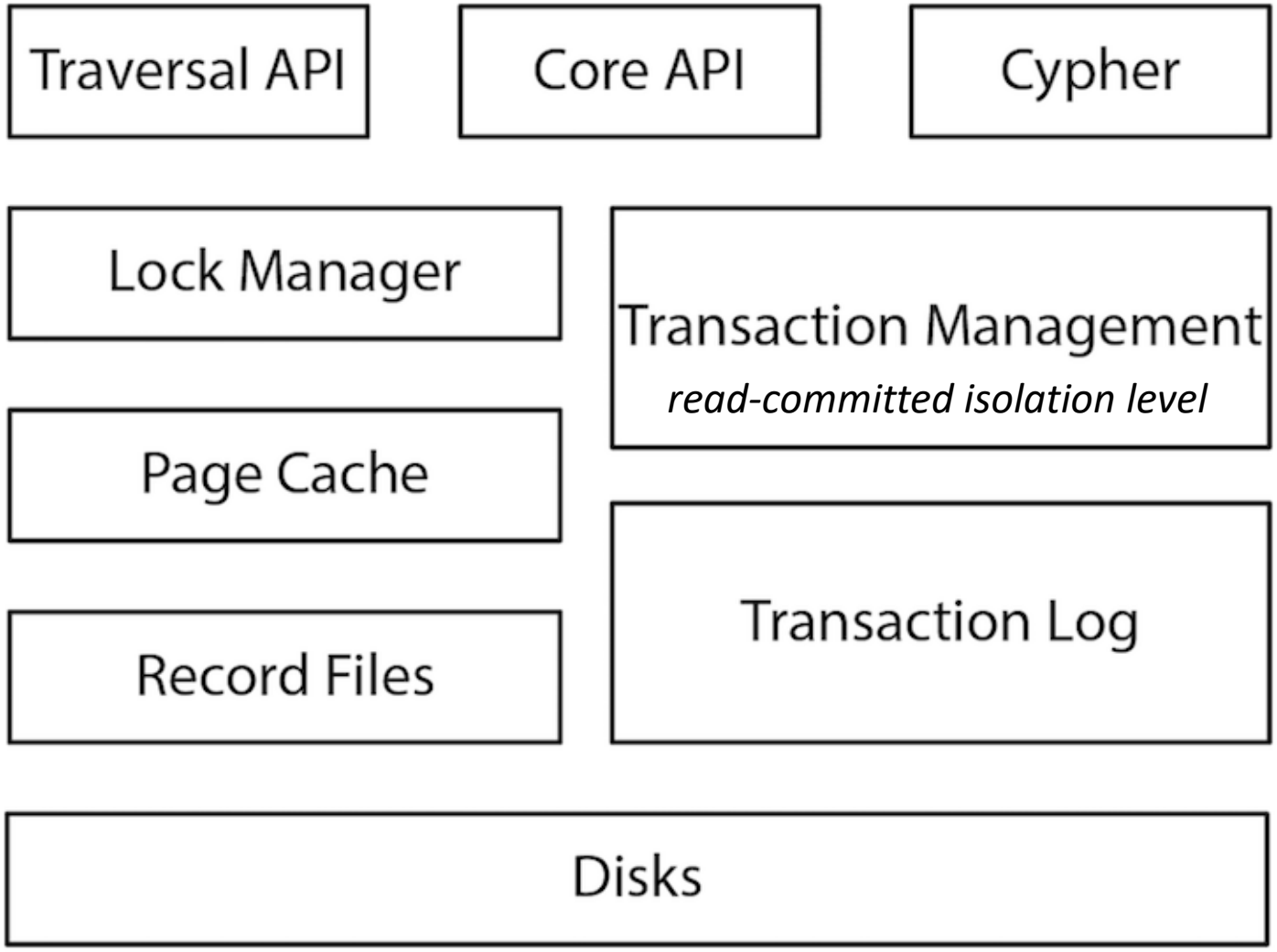


Neo4j : architecture Maitre-esclave à 2 composants

- *HA Database*: pour le stockage et la recherche des données
- *Cluster Manager* : pour la tolérance aux pannes



Neo4j : Composition d'un noeud



Neo4j : conclusion


Langage *Cypher*


Gestion de transactions ACID

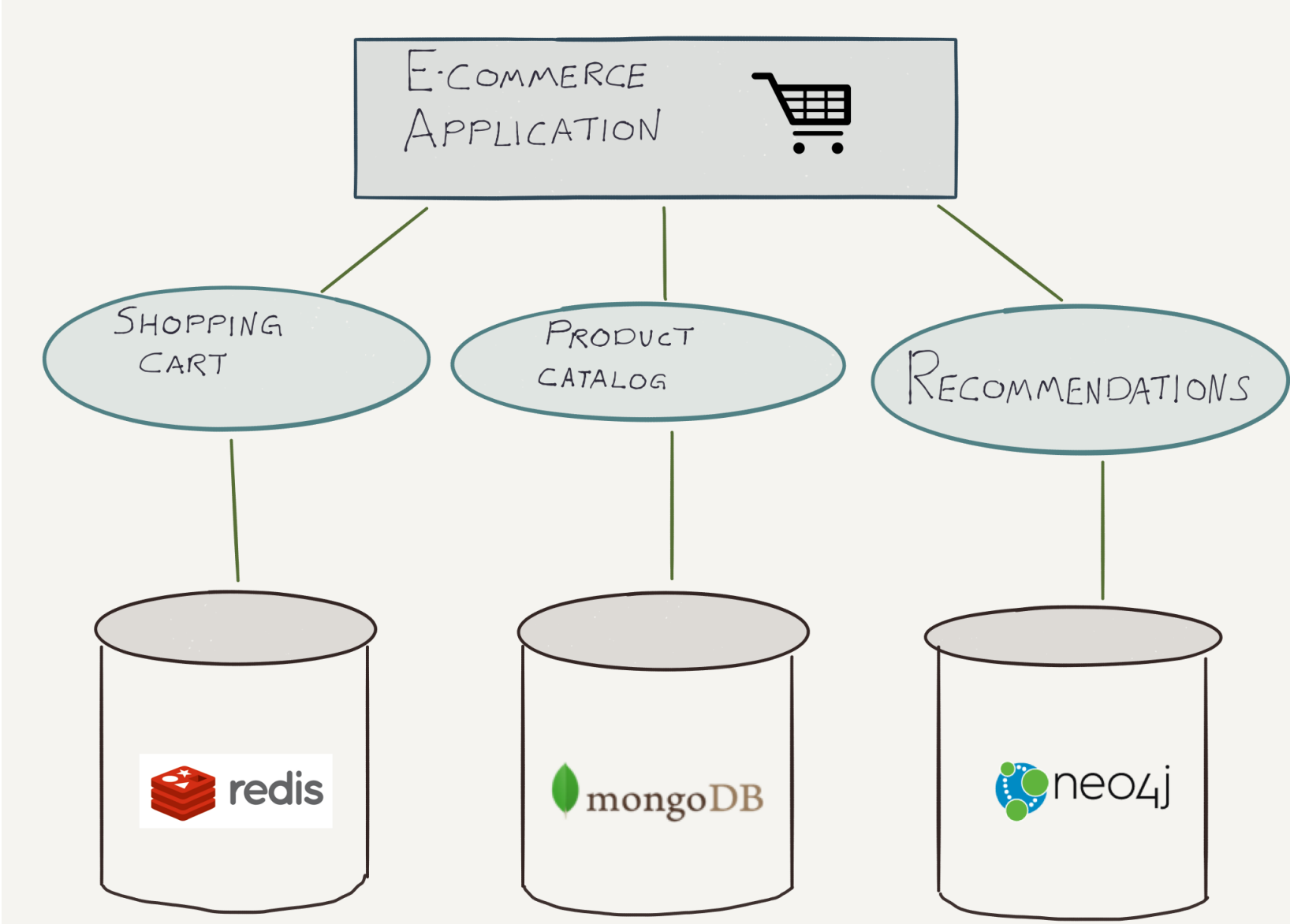

Bien adapté aux requêtes de type « parcours de graphe »


Perte en performance en raison de l'implémentation sous-jacente


Modélisation pensée pour les graphes


Non adapté aux requêtes de type « opération sur ensembles »

Exemple d'applications polyglottes utilisant Redis, MondoDB et Neo4j



OrientDB

- Base de données multi-modèles : graphes, documents, clé-valeur et objets
- Créé en 2010 par *CallidusCloud*
- Développé en Java
- Association des points forts tirés des bases de données graphes et des bases de données orientée document
- Langage de requête « à la SQL »

OrientDB : liens utiles

- Site officiel : https://orientdb.org/
- Documentation : : https://orientdb.org/docs/3.0.x/
- Tutoriels :
  - https://stph.scenari-community.org/contribs/nos/orient1/co/OrientDB-1.html
  - https://www.tutorialspoint.com/orientdb

OrientDB : modèle de données

- Notion de classes pour représenter des enregistrements
- Partition des classes en clusters
- Relations entre les classes
  - `LINK` pointe vers un objet
  - `LINKSET` pointe vers plusieurs objets (ensemble)
  - `LINKLIST` pointe vers plusieurs objets (liste)
  - `LINKMAP` pointe vers plusieurs objets (dictionnaire)
- Possibilité de stocker des graphes avec des classes particulières

OrientDB : classe Document et classe Graphe

Relational Model	Document Model	OrientDB Document Model
Table	Collection	Class or Cluster
Row	Document	Document
Column	Key/value pair	Document field
Relationship	Not available	Link

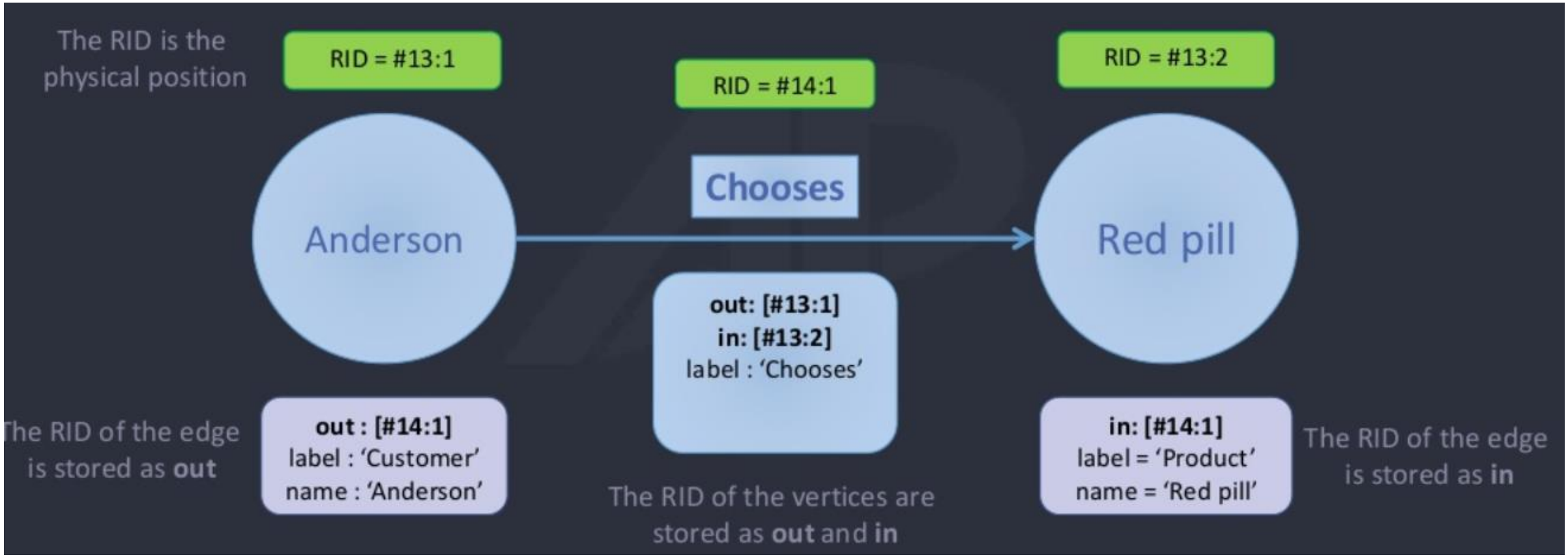
Relational Model	Graph Model	OrientDB Graph Model
Table	Vertex and Edge Class	Class that extends "V" (for Vertex) and "E" (for Edges)
Row	Vertex	Vertex
Column	Vertex and Edge property	Vertex and Edge property
Relationship	Edge	Edge

OrientDB : classe clé-valeur et classe objet

Relational Model	Key/Value Model	OrientDB Key/Value Model
Table	Bucket	Class or Cluster
Row	Key/Value pair	Document
Column	not available	Document field or Vertex/Edge property
Relationship	not available	Link

Relational Model	Object Model	OrientDB Object Model
Table	Class	Class or Cluster
Row	Object	Document or Vertex
Column	Object property	Document field or Vertex/Edge property
Relationship	Pointer	Link

OrientDB : exemple de graphe



OrientDB : création de classe et d'attribut

- Création d'une nouvelle classe : `CREATE CLASS`
- Ajout éventuel de propriété : `CREATE PROPERTY`

```
orientdb {db=myschool}> CREATE CLASS student

Class created successfully. Total classes in database now: 12.

orientdb {db=myschool}> CREATE PROPERTY student.firstname STRING

Property created successfully with id=1.
```

OrientDB : insertion

- Insertion d'un nouvel enregistrement : `INSERT INTO`

```
orientdb {db=myschool}> INSERT INTO student SET firstname='Alexis
', age=22

Inserted record 'student#21:0{firstname:Alexis,age:22} v1' in
0,090000 sec(s).

orientdb {db=myschool}>
```

OrientDB : rechercher des informations

```
orientdb {db=myschool}> BROWSE CLASS student

+----+-----+-------+---------+----+
|#   |@RID |@CLASS |firstname|age |
+----+-----+-------+---------+----+
|0   |#21:0|student|Alexis   |22  |
+----+-----+-------+---------+----+

orientdb {db=myschool}> DISPLAY RECORD 0

DOCUMENT @class:student @rid:#21:0 @version:1
+----+---------+------+
|#   |NAME     |VALUE |
+----+---------+------+
|0   |firstname|Alexis|
|1   |age      |22    |
+----+---------+------+

orientdb {db=myschool}>
```

OrientDB : créer un graphe (1/2)

```
orientdb {db=social}> CREATE CLASS person EXTENDS V

Class created successfully. Total classes in database now: 12.

orientdb {db=social}> CREATE CLASS food EXTENDS V

Class created successfully. Total classes in database now: 13.

orientdb {db=social}> CREATE CLASS likes EXTENDS E

Class created successfully. Total classes in database now: 14.

orientdb {db=social}>
```

OrientDB : créer un graphe (2/2)

```
orientdb {db=social}> CREATE VERTEX person SET firstname="Alexis"
Created vertex 'person#21:0{firstname:Alexis} v1' in 0,002000 sec
(s).

orientdb {db=social}> CREATE VERTEX food SET name="Tomato"
Created vertex 'food#25:0{name:Tomato} v1' in 0,003000 sec(s).

orientdb {db=social}> CREATE EDGE likes FROM (SELECT FROM person
WHERE firstname="Alexis") TO (SELECT FROM food WHERE name="Tomato
")

Created edge '[likes#29:0{out:#21:0,in:#25:0} v1]' in 0,148000
sec(s).

orientdb {db=social}>
```

OrientDB : interrogation en SQL ou parcours de graphe

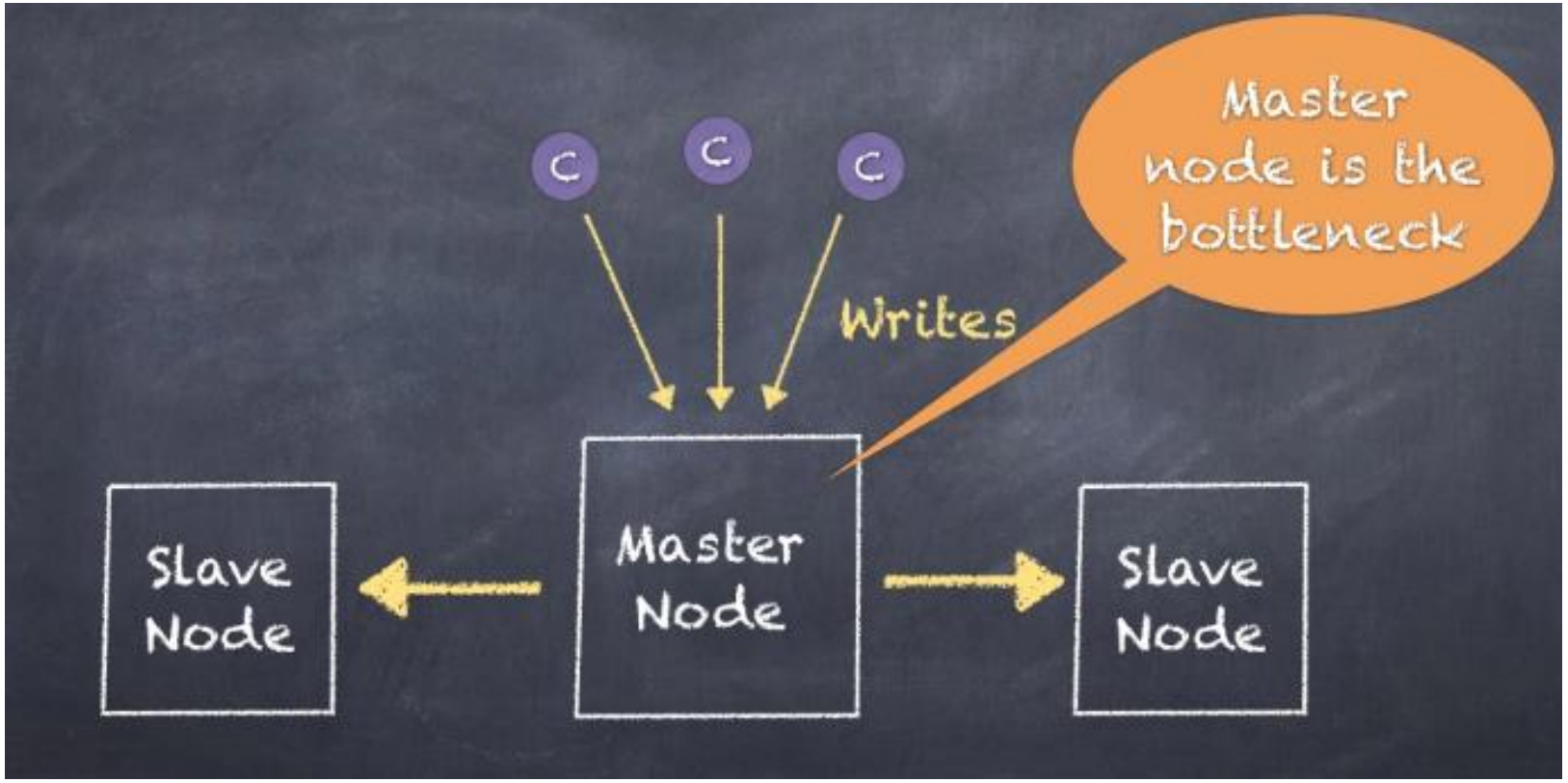
```
SELECT
  count(*) as NumberOfProfiles,
  Birthday.format('yyyy') AS YearOfBirth
FROM Profiles
GROUP BY YearOfBirth
ORDER BY NumberOfProfiles DESC
```

```
MATCH
    {class: Person, as: people, where: (name = 'John')}
RETURN people
```

```
SELECT
  customer.OrderedId as customerOrderedId,
  SUM(order.Amount) as totalAmount
FROM (
  MATCH {Class: Customers, as: customer}<-HasCustomer-{class: Orders, as: order}
  RETURN customer, order
)
GROUP BY customerOrderedId
ORDER BY totalAmount DESC
LIMIT 3
```

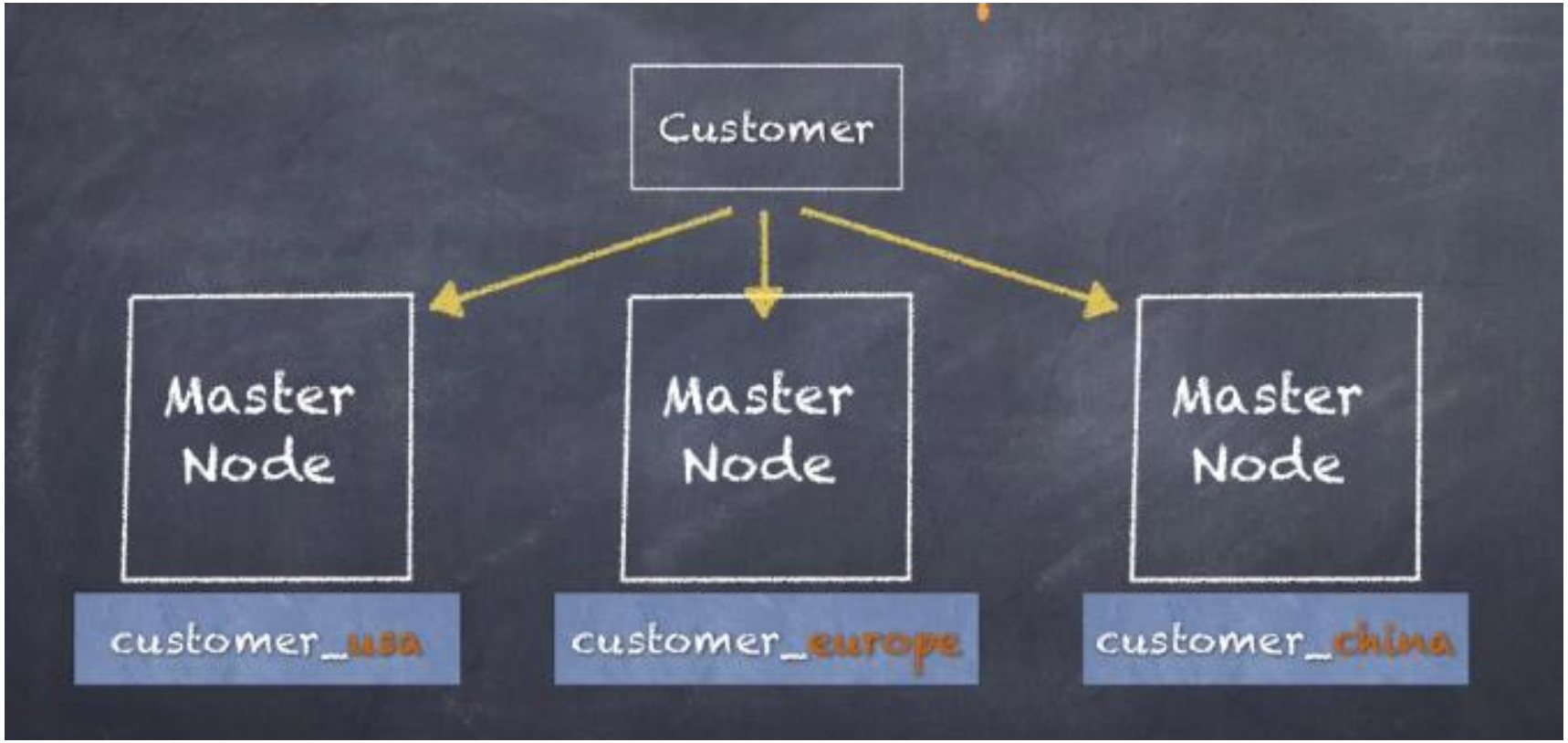
OrientDB : architecture (1/2)

Avant 2012 : architecture Maître-Esclave



OrientDB : architecture (2/2)

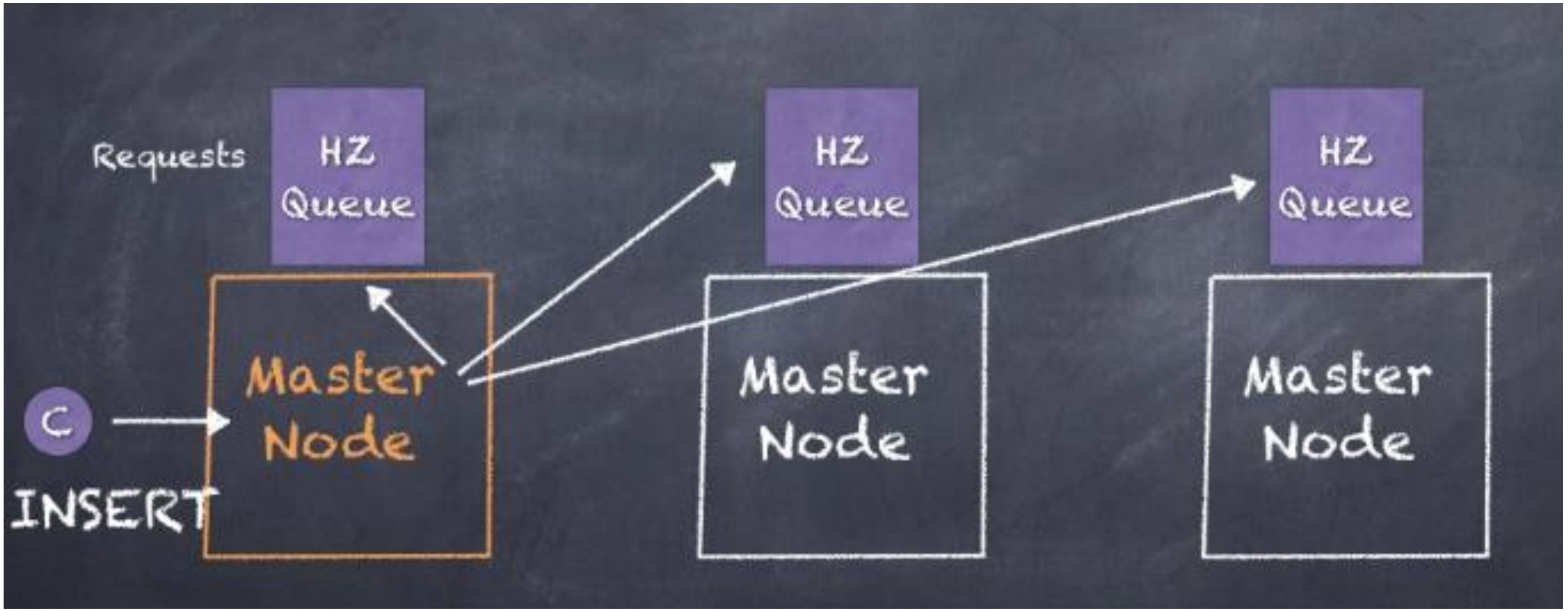
Depuis 2012 : architecture Multi-Maîtres ("*master-less*") avec partitionnement des données en cluster



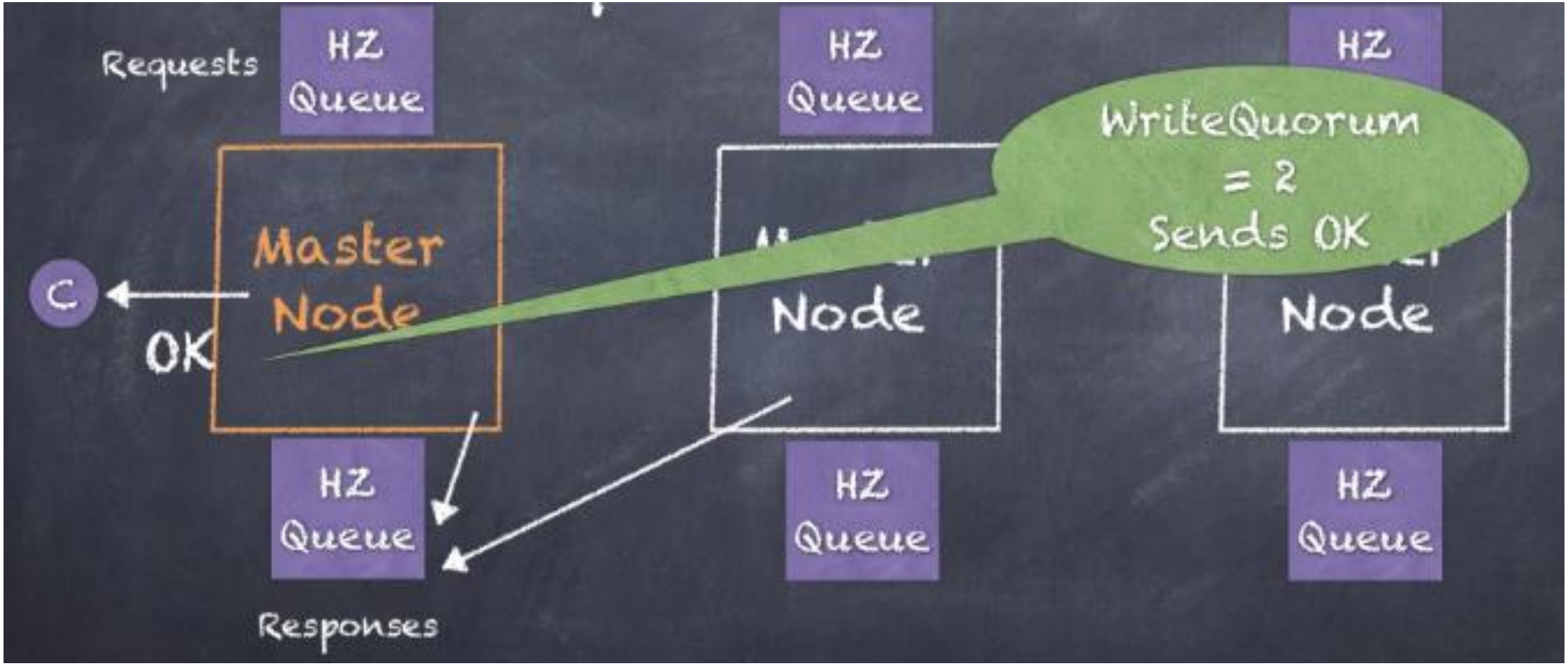
OrientDB : réplication

Possibilité de réplication :

OrientDB : exécution d'une requête



OrientDB : *WriteQuorum*



OrientDB : conclusion


Langage de requête et traversée de graphe


Transactions ACID


Architecture multi-master


Multi-Modèles

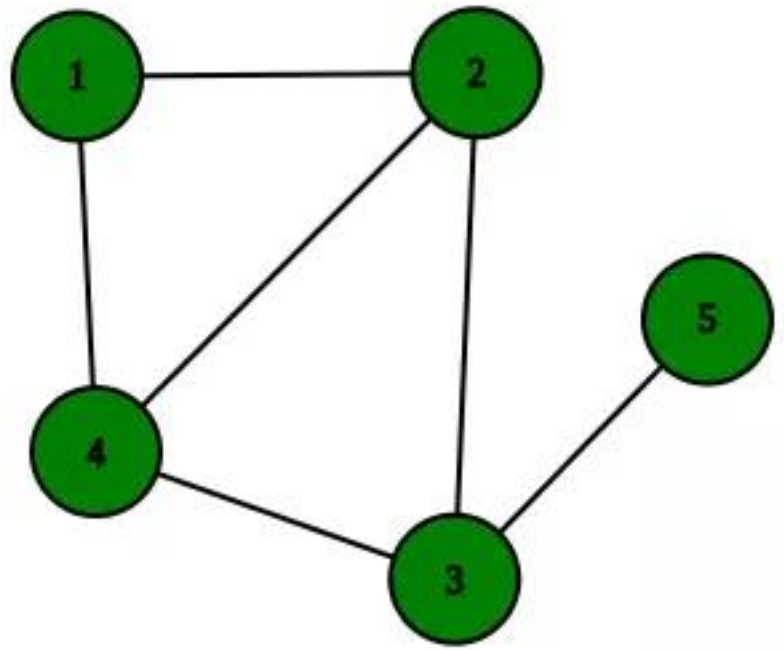

Monitoring, gestion des sauvegardes sous licence commerciale


Multi-Modèles

Graphes et Base de Données Relationnelles

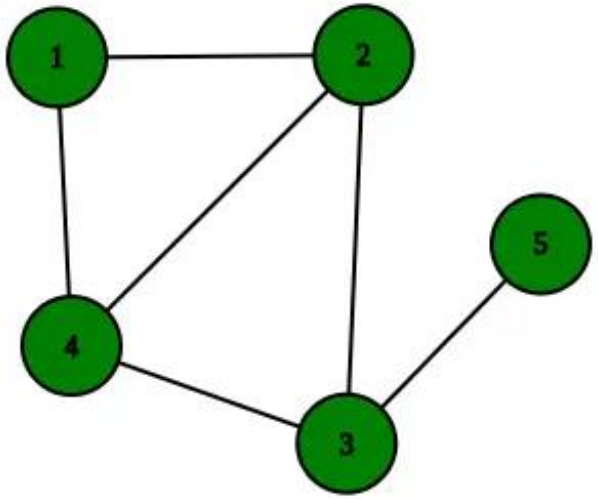
Possibilité de simuler un graphe via des requêtes récursives :

```sql
CREATE TABLE edges (
    src integer,
    dest integer
);

INSERT INTO edges (
     src,
     dest
)
VALUES
     (1, 2),
     (2, 3),
     (2, 4),
     (3, 4),
     (4, 1),
     (3, 5);
```



Requête SQL récursive

```
WITH RECURSIVE paths (src, dest, path) AS (
        SELECT e.src, e.dest, ARRAY[e.src, e.dest]
        FROM edges e
    UNION
        SELECT p.src, e.dest, p.path || ARRAY[e.dest]
        FROM paths p
        JOIN edges e
        ON p.dest = e.src AND e.dest != ALL(p.path)
)
SELECT * FROM paths;
```



src	dest	path
1	2	[1,2]
2	3	[2,3]
2	4	[2,4]
3	4	[3,4]
4	1	[4,1]
3	5	[3,5]
4	2	[4,1,2]
1	3	[1,2,3]
1	4	[1,2,4]
2	4	[2,3,4]

src	dest	path
2	5	[2,3,5]
2	1	[2,4,1]
3	1	[3,4,1]
3	2	[3,4,1,2]
4	3	[4,1,2,3]
1	4	[1,2,3,4]
1	5	[1,2,3,5]
2	1	[2,3,4,1]
4	5	[4,1,2,3,5]

Clause SQL `WITH RECURSIVE`

Pour faire des itérations

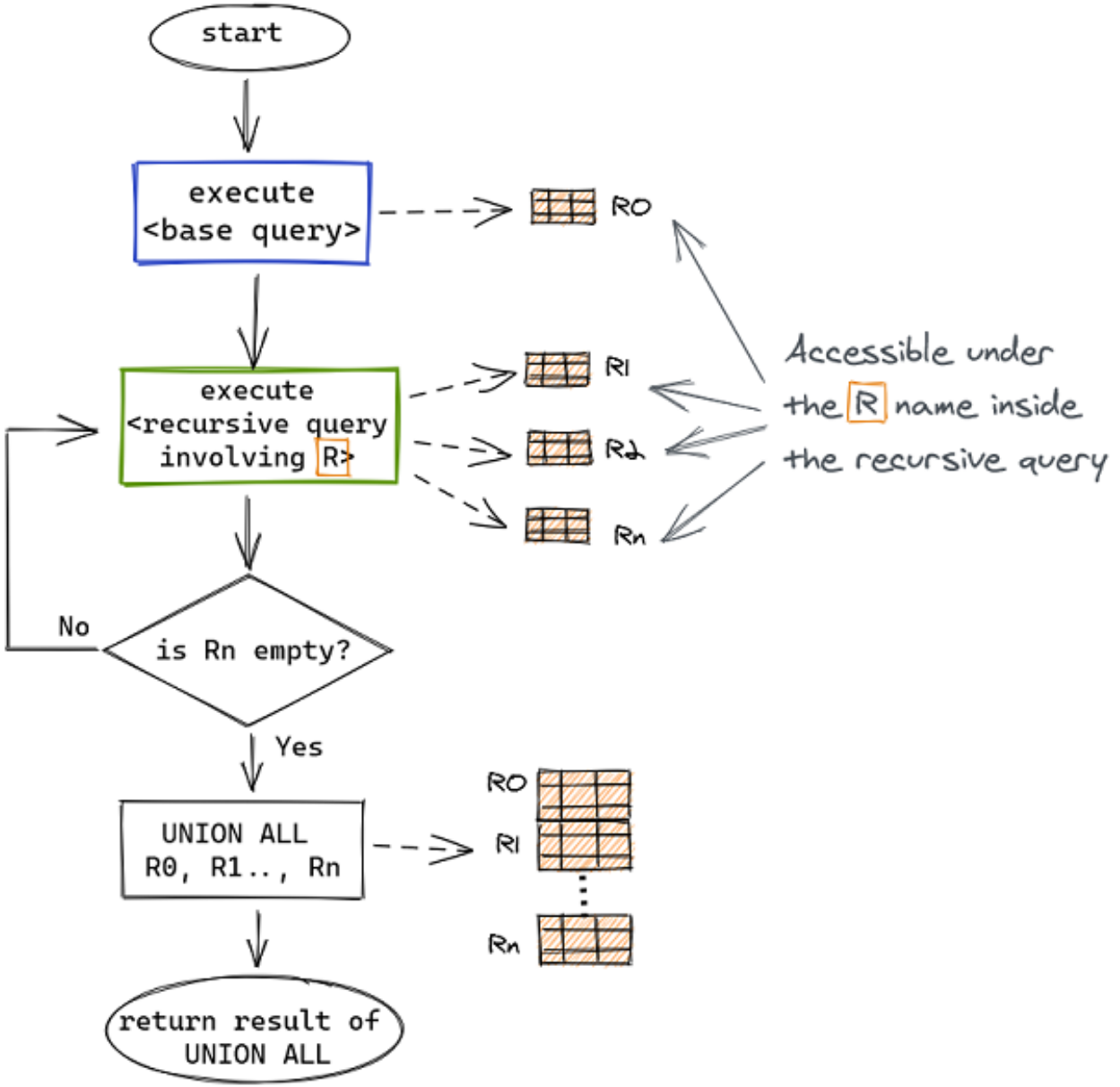
```
WITH RECURSIVE R AS (
      requeteDeBase
    UNION ALL
      requeteFaisantReferenceàR
    )
SELECT ... FROM R
```

La requête faisant référence à R doit finir par ne retourner aucun nuplet pour arrêter la boucle !

Exemple : requête, qui calcule la somme des nombres de 1 à 100

```
WITH RECURSIVE t(n) AS (
    VALUES (1)
  UNION ALL
    SELECT n+1 FROM t WHERE n < 100
)
SELECT sum(n) FROM t;
```

Fonctionnement de la clause `WITH RECURSIVE`

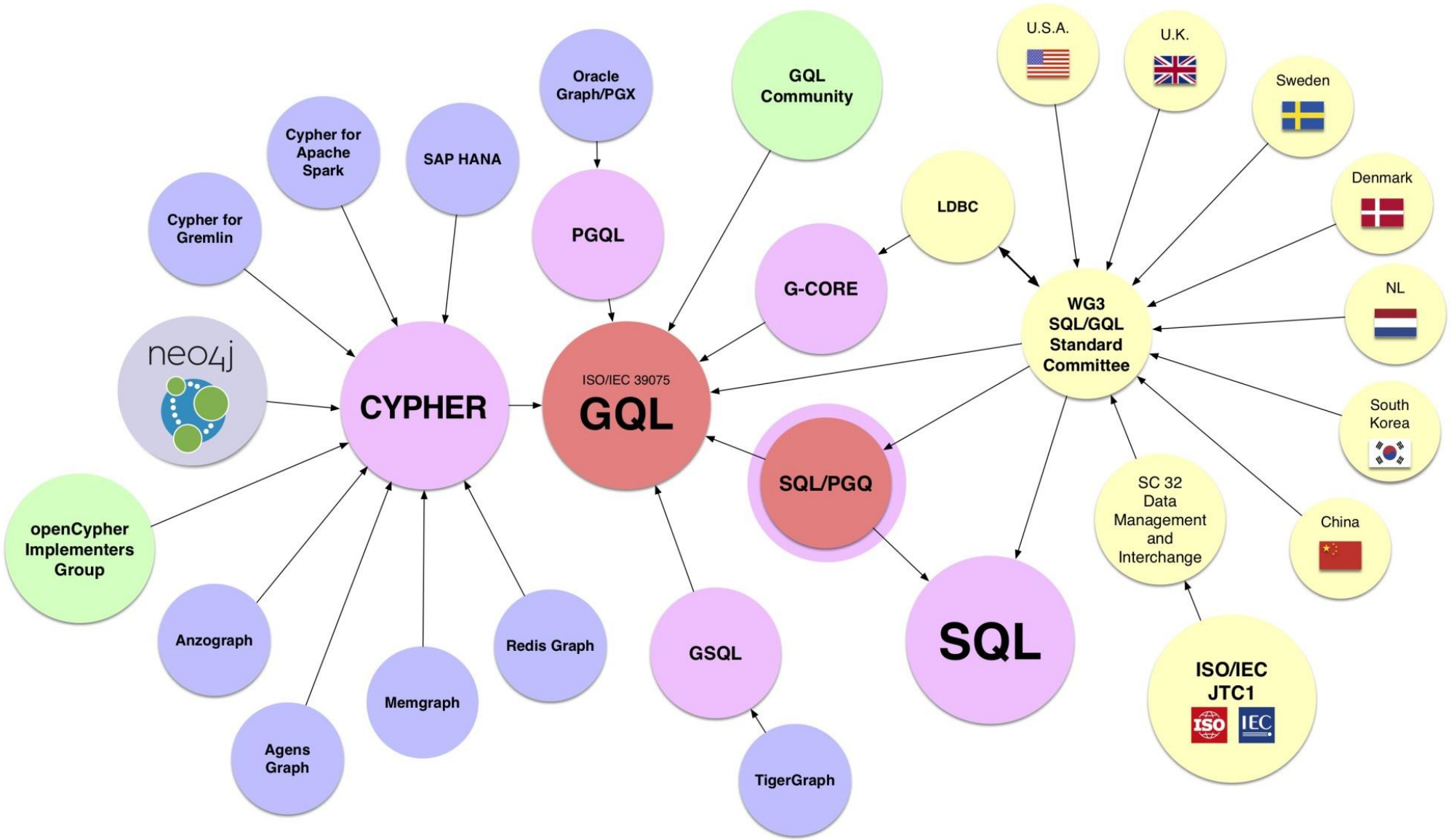


Simuler un Graphe dans une Base de Données Relationnelles

Exemples :

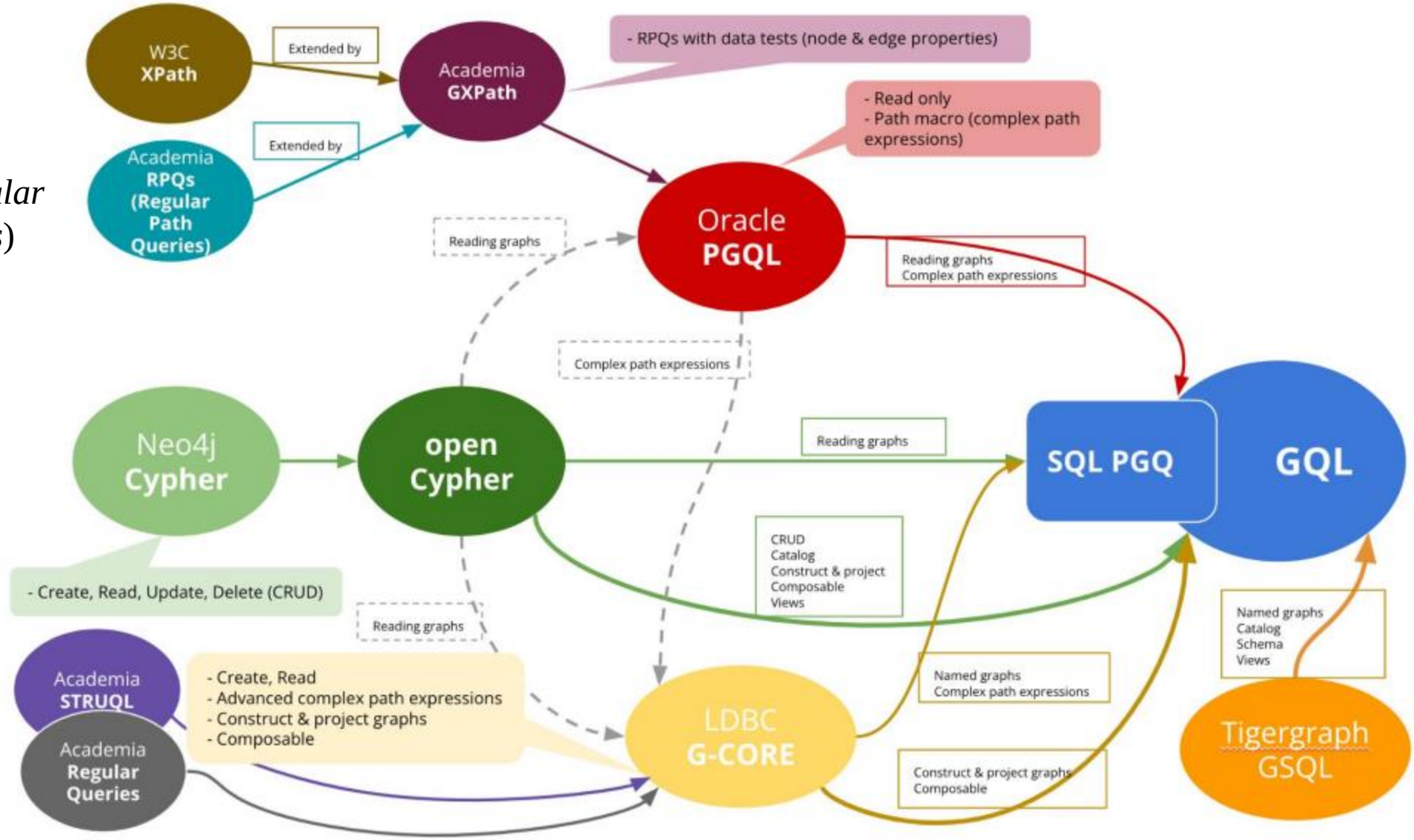
- https://www.dylanpaulus.com/posts/postgres-is-a-graph-database/
- https://blog.whiteprompt.com/implementing-graph-queries-in-a-relational-database-7842b8075ca8
- https://www.fusionbox.com/blog/detail/graph-algorithms-in-a-database-recursive-ctes-and-topological-sort-with-postgres/620/
- https://martinheinz.dev/blog/18
- https://www.alibabacloud.com/blog/postgresql-graph-search-practices---10-billion-scale-graph-with-millisecond-response_595039

Plusieurs implémentation et extensions du langage *Cypher (1/2)*



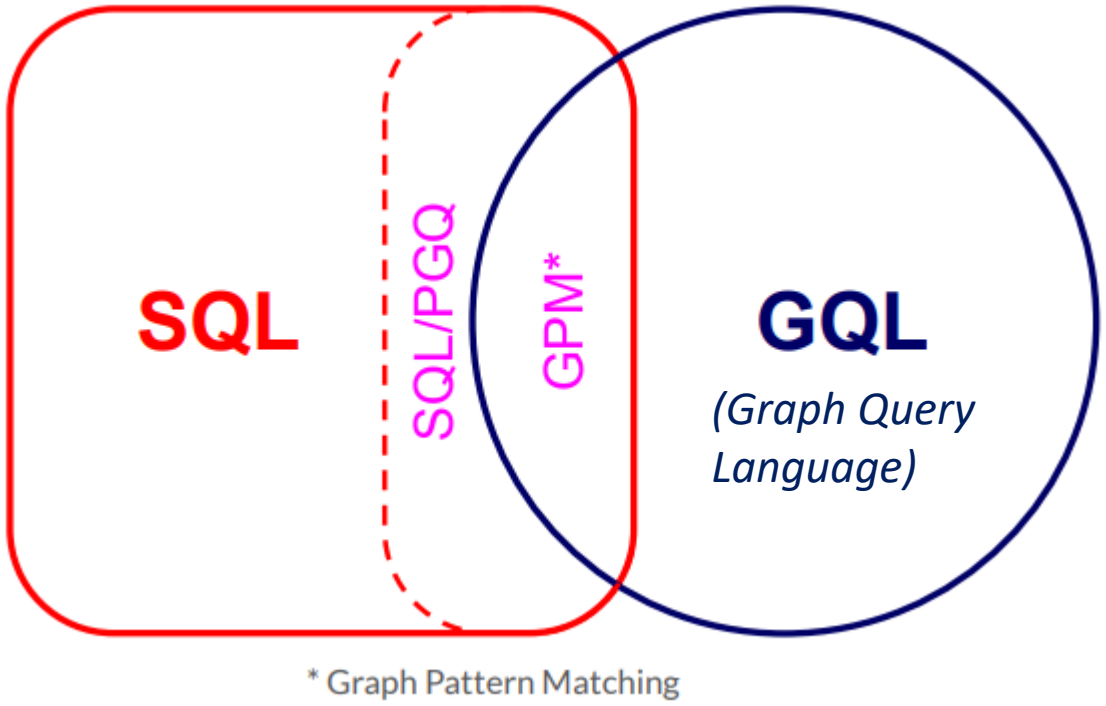
Plusieurs implémentation et extensions du langage *Cypher (2/2)*

RPQs (*Regular path queries*)



LDBC (*Linked Data Benchmark Council*)

Extension du standard SQL pour les graphes



Timeline to Standards

Date	SQL/PGQ	GQL
2017	Work started	
2018		Work started
2021-02-07	CD Ballot End	
2022-02-20		CD Ballot End
2022-12-04	DIS Ballot End	
2023-01-30	Final Text to ISO	
2023-03-13	**SQL/PGQ IS Published**	
2023-05-21		DIS Ballot End
2023-07-30		Final Text to ISO
2023-09-10		**GQL IS Published**

- SQL/PGQ published June 2023 (*SQL Property Graph Querying*)
- GQL published March 2024
- Graph Pattern Matching (GQL and SQL/PGQ) stable August 2022

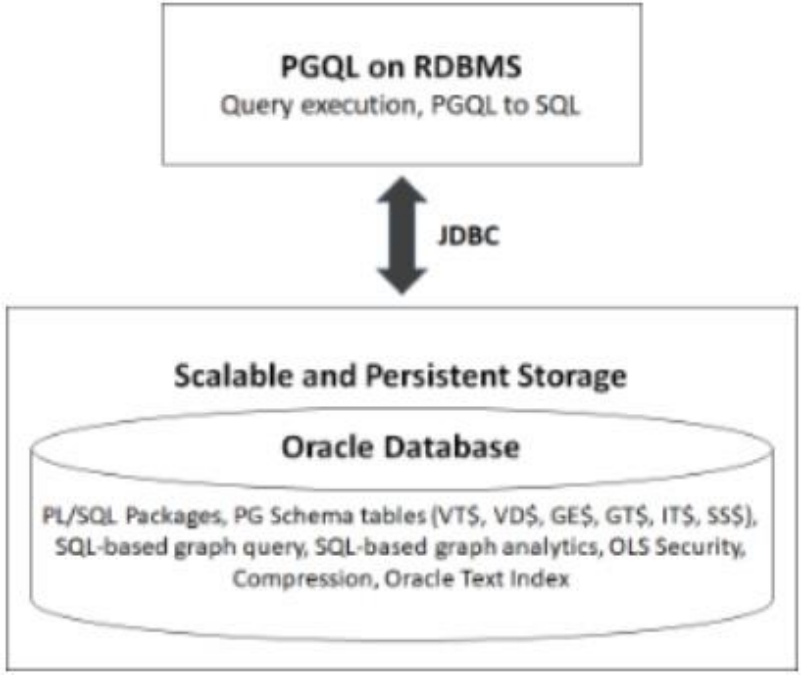
Property Graph Queries (SQL/PGQ)

- Extension du standard SQL aux graphes – SQL:2023 : https://www.iso.org/obp/ui/#iso:std:iso-iec:9075:-16:dis:ed-1:v1:en

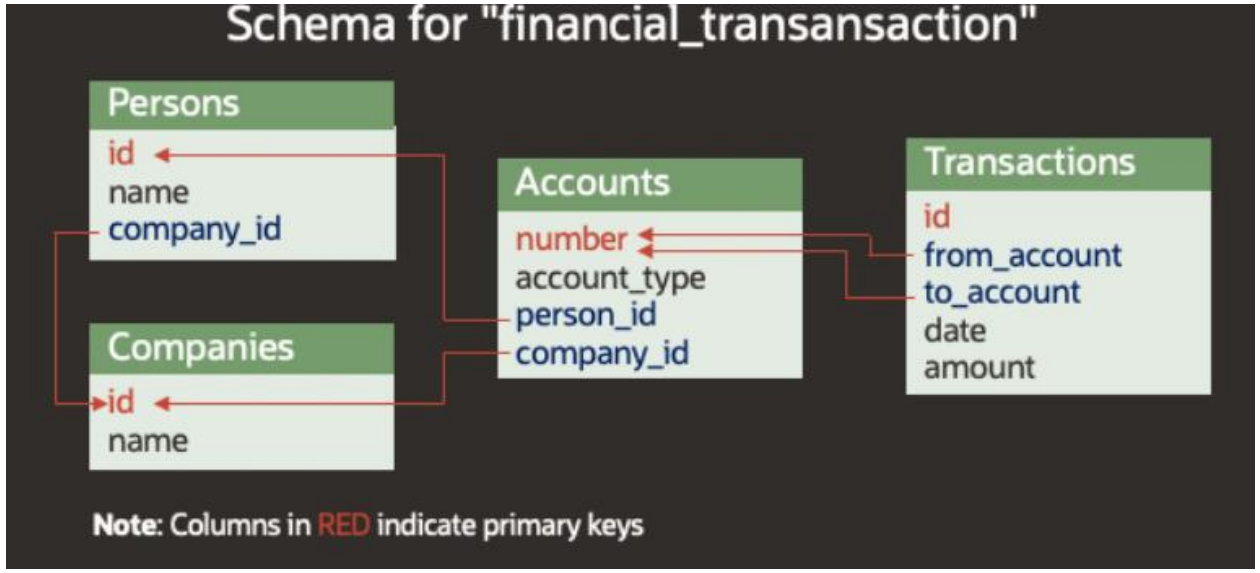
```
CREATE TABLE person (...);
CREATE TABLE company (...);
CREATE TABLE ownerof (...);
CREATE TABLE transaction (...);
CREATE TABLE account (...);

CREATE PROPERTY GRAPH financial_transactions
    VERTEX TABLES (person, company, account)
    EDGE TABLES (ownerof, transaction);

SELECT owner_name,
       SUM(amount) AS total_transacted
FROM financial_transactions GRAPH_TABLE (
  MATCH (p:person WHERE p.name = 'Alice')
        -[:ownerof]-> (:account)
        -[t:transaction]- (:account)
        <-[:ownerof]- (owner:person|company)
  COLUMNS (owner.name AS owner_name, t.amount AS amount)
) AS ft
GROUP BY owner_name;
```

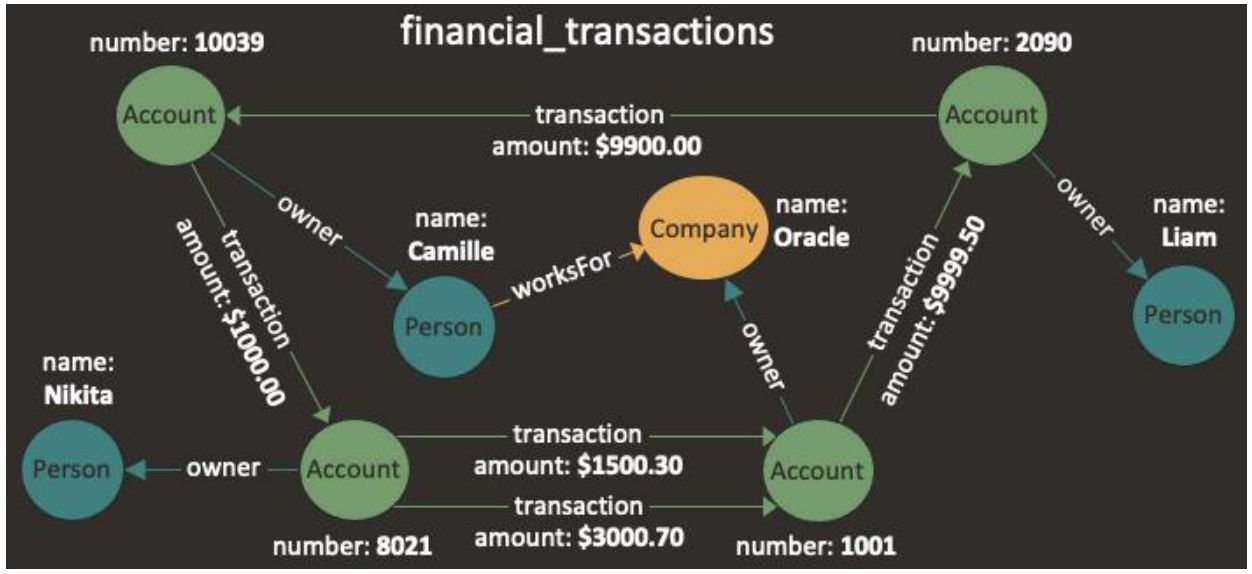
Property Graph Query Language (PGQL) d'Oracle

- Description : https://pgql-lang.org/
- Projet Open-source d'Oracle : https://github.com/oracle/pgql-lang
- Langage de la base de données Graphe d'Oracle, *Oracle's Graph Database* : https://www.oracle.com/database/graph/
- Dernière spécification en août 2022 : https://pgql-lang.org/spec/1.5/



Exemple sous PGQL d'Oracle

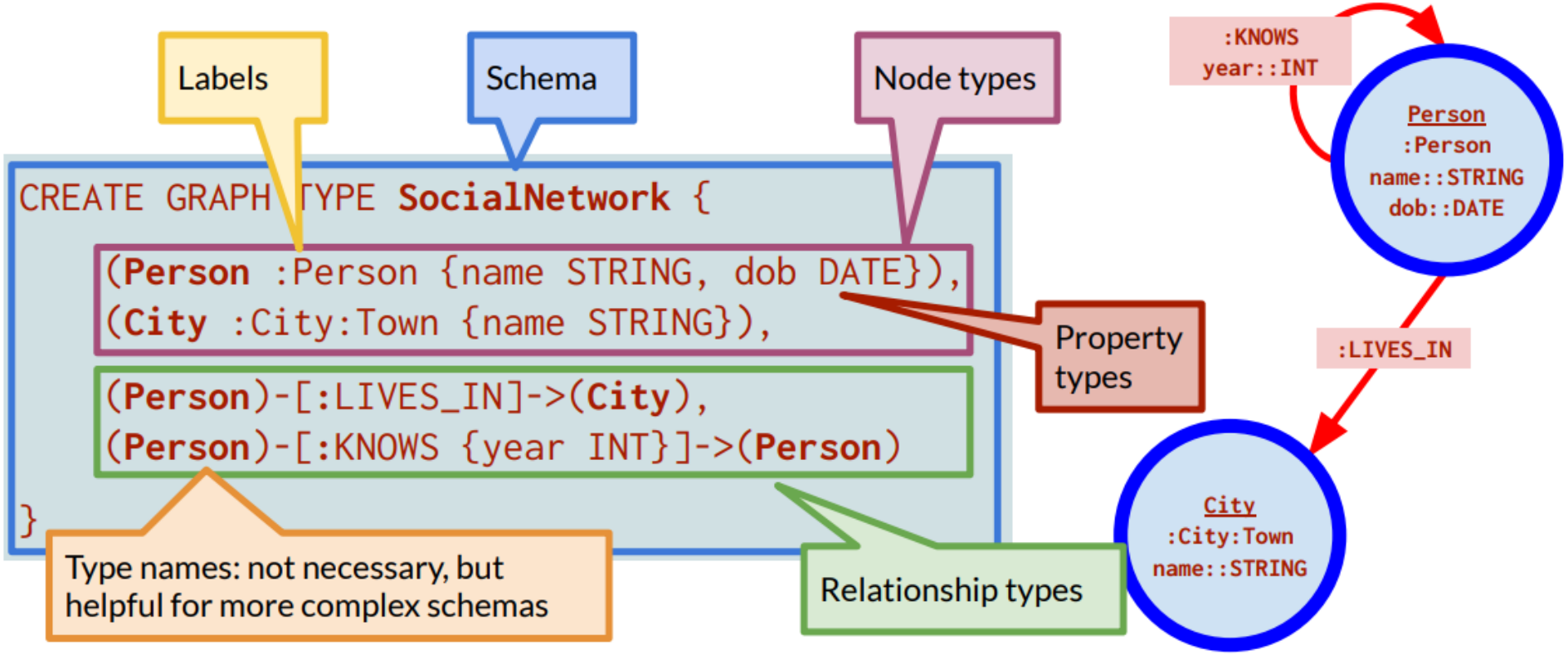





```
CREATE PROPERTY GRAPH financial_transactions
  VERTEX TABLES (
    Persons LABEL Person PROPERTIES ( name ),
    Companies LABEL Company PROPERTIES ( name ),
    Accounts LABEL Account PROPERTIES ( number )
  )
  EDGE TABLES (
    Transactions
      SOURCE KEY ( from_account ) REFERENCES Accounts ( number )
      DESTINATION KEY ( to_account ) REFERENCES Accounts ( number )
      LABEL transaction PROPERTIES ( amount ),
    Accounts AS PersonOwner
      SOURCE KEY ( number ) REFERENCES Accounts ( number )
      DESTINATION Persons
      LABEL owner NO PROPERTIES,
    Accounts AS CompanyOwner
      SOURCE KEY ( number ) REFERENCES Accounts ( number )
      DESTINATION Companies
      LABEL owner NO PROPERTIES,
    Persons AS worksFor
      SOURCE KEY ( id ) REFERENCES Persons ( id )
      DESTINATION Companies
      NO PROPERTIES
  )
```

Graph Query Language (GQL)

- Page web officielle : https://www.gqlstandards.org/
- Page web avec des liens : https://github.com/szarnyasg/gql-sql-pgq-pointers



Exemples de requêtes en GQL

```
INSERT ()-[r:S]->()
SET r = { a: 20, b: "West", c: 0.937 }
RETURN r.a, r.b, r.c // 20, "West", 0.937
```

```
MATCH ()-[r { a: 20 }]->()
SET r.b = "West"
RETURN r.a, r.b // 20, "West"
```

```
SELECT t.name AS team, avg(p.age) AS avgAge, count(p) AS numPlayers
FROM SportsGraph
MATCH (t:BasketballTeam)->(p:Player) WHERE t.level = 'pro'
GROUP BY t HAVING numPlayers > 5
ORDER BY avgAge DESC
LIMIT 5
```

Principaux moteurs orientés graphes

		Deployment	Graph Model	Graph OLTP			Graph OLAP	Scale-Out
				Query Language	Visualization tools	Transaction		
Graph Only Companies	**TigerGraph**	On-prem / AWS, Azure, GCP	PG	GSQL	Graph Studio	ACID	GSQL, 23 built-in algorithms	Yes
	Neo4J	On-prem / AWS, Azure, GCP	PG	Cypher	Studio	Non-repeatable reads may occur	Pregel API, 48 built-in algorithms (including Graph ML)	Yes
Data Companies	**DataStax** Enterprise Graph	On-prem / AWS, Azure, GCP	PG	Gremlin	Studio	Row-level (Cassandra)	SparkGraphComputer API	Yes
	Databricks GraphX & GraphFrames	On-prem / AWS, Azure, GCP	PG	Motif Finding DSL	-	-	Pregel API, 7 built-in algorithms	Yes
Enterprise Cloud Companies	**Amazon** Neptune	AWS	PG, RDF	Gremlin, SPARQL	Neptune Workbench	ACID	-	Yes
	Microsoft SQL Graph	On-prem / Azure	PG	SQL Extension	Power BI plugin, 3rd party tools	ACID	Python/R scripts via Machine Learning Services	Yes (Read-Only Queries)
	Microsoft Cosmos DB Graph	Azure	PG	Gremlin	Azure Portal, 3rd party tools	-	-	Yes
	Oracle Spatial and Graph	On-prem / OCI AWS, Azure, GCP	PG, RDF	PGQL, SPARQL	Graph Studio	ACID	Green Marl DSL, 50+ built-in algorithms (including Graph ML)	Yes
	IBM Db2 Graph	On-prem / CP4D	PG	Gremlin	Graph UI	ACID	-	Yes