# Aide-mémoire CYPHER

Voici un récapitulatif des syntaxes et des fonctions prévues dans l'utilisation de CYPHER. Cet aide-mémoire a été rédigé à partir de la feuille de référence Neo4j v2.3.

## 1. Lire des données et la structure du graphe

### Syntaxe générale

```
[MATCH WHERE]
[OPTIONAL MATCH WHERE]
[WITH [ORDER BY] [SKIP] [LIMIT]]
RETURN [ORDER BY] [SKIP] [LIMIT]
```

### Identifier des données

| **Expression : MATCH** |
|---|
| `MATCH ()--()`<br>Nœuds anonymes reliés par une relation. |
| `MATCH (n)--(m)`<br>Nœuds identifiés par `n` et `m` reliés par une relation. |
| `MATCH (n)-->(m)<--(o), (m)<--(p)`<br>Nœud `m` ayant trois relations incidentes orientées. |
| `MATCH (n)-[r]->(m)`<br>Relation orientée identifiée par `r`. |
| `MATCH (n)-[*]->(m)`<br>Nœuds identifiés par `n` et `m` reliés quel que soit le nombre de relations traversées. |
| `MATCH (n)-[r*]->(m)`<br>Nœuds identifiés par `n` et `m` reliés quel que soit le nombre de relations traversées identifiées par `r`. |

| **Expression : MATCH** |
|---|
| `MATCH (n)-[*1..5]->(m)`<br><br>Nœuds identifiés par `n` et `m` reliés par un nombre de relations traversées allant de 1 à 5. |
| `MATCH (n)-[r*1..5]->(m)`<br><br>Nœuds identifiés par `n` et `m` reliés par un nombre de relations traversées identifiées par r allant de 1 à 5. |
| `MATCH (n)-[r:CONNAIT*1..5]->(m)`<br><br>Où `n` connaît `m` directement (1 relation traversée) ou indirectement (jusqu'à 5 relations traversées). |
| `MATCH (n)-[r:CONNAIT\|AIME*1..5]->(m)`<br><br>Où `n` connaît OU (inclusif) aime `m` , directement (1 relation traversée) ou indirectement (jusqu'à 5 relations traversées). |
| `MATCH (n:Personne)-[:CONNAIT]->(m:Personne:Salarie)`<br><br>Où une personne `n` connaît une personne salariée `m` (labels). |
| `MATCH (n {nom:'Alice'})-->(m)`<br><br>Où le nœud `n` dont la propriété `nom` a pour valeur `Alice` est relié à un autre nœud. |
| `MATCH p = (n)-->(m)`<br><br>Assigne un chemin à l'identificateur `p`. |

| **Expression : WHERE** | |
|---|---|
| `WHERE n.nom = "Alice"` | Dont le nom est `Alice` |
| `WHERE n.nom <> "Alice"` | Dont le nom n'est pas `Alice` |
| `WHERE n.nom =~ "Alic.*"` | Dont le nom commence par `Alic` |
| `WHERE NOT n<--()` | Où `n` ne possède aucune relation entrante |
| `WHERE EXISTS ((n)-->(m))` | Où il existe une relation sortante entre `n` et `m` |

La clause `WHERE` est un prédicat retournant soit faux, soit vrai. À noter que `WHERE` fait toujours partie d'une clause `MATCH`, `OPTIONAL MATCH` ou `WITH`.

## Collecter des données

| Expression : RETURN | |
|---|---|
| `RETURN *` | Retourne toutes les valeurs de tous les identificateurs |
| `RETURN n AS nomColonne` | Utilise un alias pour nom de colonne |
| `RETURN DISTINCT n` | Retourne uniquement les lignes uniques pour l'identificateur `n` |
| `ORDER BY n.nom` | Tri ascendant sur la colonne `n.nom` |
| `ORDER BY n.nom DESC` | Tri descendant sur la colonne `n.nom` |
| `SKIP {nombre_a_passer}` | Passe un certain nombre de résultats |
| `LIMIT {nombre_limite}` | Limite le nombre de résultats |
| `SKIP {nombre_a_passer}`<br>`LIMIT {nombre_limite}` | Pagination (de *`nombre_a_passer`* à *`nombre_limite`*) |
| `RETURN count(*)` | Retourne le nombre de résultats obtenus pour la requête |

| Expression : WITH | |
|---|---|
| `MATCH (quelqun)-[:EST_AMI_AVEC]-(ami)`<br>`WHERE quelqun.nom = {nom}`<br>`WITH quelqun, count(ami) AS amis`<br>`WHERE amis > 10`<br>`RETURN quelqun` | La syntaxe de l'instruction `WITH` est similaire à celle de `RETURN`. Cependant elle isole les différentes parties de la requête |
| `MATCH (quelqun)-[:EST_AMI_AVEC]-(ami)`<br>`WITH quelqun, count(ami) AS amis`<br>`ORDER BY amis DESC`<br>`SKIP 1 LIMIT 3`<br>`RETURN quelqun` | Comme avec la clause `RETURN`, il est possible d'utiliser les mots clés `ORDER BY`, `SKIP` et `LIMIT` |

| Expression : UNION | |
|---|---|
| `MATCH (a)-[:CONNAIT]->(b)`<br>`RETURN b.nom`<br>`UNION`<br>`MATCH (a)-[:AIME]->(b)`<br>`RETURN b.nom` | Retourne une union distincte (pas de ligne dupliquée) des deux requêtes |

| Expression : UNION | |
|---|---|
| `MATCH (a)-[:CONNAIT]->(b)`<br>`RETURN b.nom`<br>`UNION ALL`<br>`MATCH (a)-[:AIME]->(b)`<br>`RETURN b.nom` | Retourne une union complète (duplication de lignes possible) des deux requêtes |

# 2. Modifier les données et la structure d'un graphe

## Syntaxe générale

Modification uniquement :

```
(CREATE [UNIQUE] | MERGE)*
[SET|DELETE|REMOVE|FOREACH]*
[RETURN [ORDER BY] [SKIP] [LIMIT]]
```

Lecture et modification :

```
[MATCH WHERE]
[OPTIONAL MATCH WHERE]
[WITH [ORDER BY] [SKIP] [LIMIT]]
(CREATE [UNIQUE] | MERGE)*
[SET|DELETE|REMOVE|FOREACH]*
[RETURN [ORDER BY] [SKIP] [LIMIT]]
```

| Expression : CREATE | |
|---|---|
| `CREATE n` | Crée un nœud identifié `n` dans la requête |
| `CREATE (n {nom: "Alice"})` | Crée un nœud identifié `n` avec une propriété `nom` ayant pour valeur `Alice` |
| `CREATE (n objet)` | Crée un nœud identifié `n` avec une suite de propriétés (sous forme d'objet `{}`) |
| `CREATE (n)-[r:CONNAIT]->(m)` | Crée une relation `CONNAIT` entre le nœud `n` et le nœud `m` |
| `CREATE (n)-[:CONNAIT {depuis: "Lycée"}]->(m)`<br>Crée une relation `CONNAIT` pourvue de la propriété `depuis`. | |

| **Expression : CREATE** |
|---|
| `MATCH (n{nom:"Sylvain""}) CREATE UNIQUE (n)-[:CONNAIT {depuis: "Lycée"}]->(m{nom:"Christophe"})`<br><br>Crée une relation `CONNAIT` si elle n'existe pas, crée le nœud `m` s'il n'existe pas. |

| **Expression : SET** | |
|---|---|
| `SET nr.titre = "Alice au pays des merveilles", nr.auteur = "Lewis Caroll"`<br><br>Crée ou met à jour des propriétés sur un nœud ou une relation. | |
| `SET nr=objet` | Pose un ensemble de propriétés (objet `{}`) ou remplace l'ensemble des propriétés sur un nœud ou une relation |
| `SET n:Personne` | Ajoute le label `Personne` au nœud `n` |

| **Expression : MERGE** |
|---|
| `MERGE (n:Personne {nom: "Sylvain"})`<br><br>Trouve le nœud `n`, le crée s'il n'existe pas. |
| `MERGE (n:Personne {nom: "Sylvain"})`<br>`ON CREATE SET n.dateCreation=timestamp()`<br>`ON MATCH SET n.compteur= coalesce(n.compteur, 0) + 1, n.dateModification = timestamp()`<br><br>Mise à jour conditionnelle : si le nœud `n` n'existe pas, il est créé et une propriété `dateCreation` est ajoutée à ce nœud ; si le nœud `n` existe déjà, la propriété `compteur` du nœud `n` est augmentée et sa propriété `dateModification` est mise à jour. |
| `MATCH (a:Personne {nom: "Sylvain"}), (b:Personne {nom: "Christophe"})`<br>`MERGE (a)-[r:CONNAIT]->(b)`<br><br>Trouve la relation `r`, la crée si elle n'existe pas. |
| `MATCH (a:Personne {nom: "Sylvain"})`<br>`MERGE (a)-[r:CONNAIT]->(b:Personne {nom: "Christophe"})`<br><br>Trouve ou crée le sous-graphe attaché au nœud. |

| **Expression : DELETE** | |
|---|---|
| `DELETE n, r` | Supprime un nœud et une relation |

| Expression : DELETE | |
|---|---|
| `DETACH DELETE n` | Supprime un nœud et les relations qui lui sont attenantes |

| Expression : REMOVE | |
|---|---|
| `REMOVE n.nom` | Supprime la propriété `nom` sur le nœud `n` |
| `REMOVE n:Personne` | Supprime le label `Personne` du nœud `n` |

## 3. Indexation, emploi du planificateur de requêtes et contraintes

| Indexation et emploi du planificateur de requêtes | |
|---|---|
| `CREATE INDEX ON :Personne(nom)` | Crée un index basé sur le couple du label `Personne` avec la propriété `nom` |
| `MATCH (n:Personne)`<br>`WHERE n.nom = "Sylvain"` | Retrouve le nœud `Sylvain` en utilisant l'index `:Personne(nom)` (index retrouvé par l'opération d'égalité. À noter que l'utilisation d'une fonction telle que `WHERE lower(n.nom) = "sylvain"` ne permet pas de déterminer l'index à utiliser) |
| `DROP INDEX ON :Personne(nom)` | Supprime l'index basé sur le couple du label `Personne` avec la propriété `nom` |
| `MATCH (n:Personne)`<br>`USING INDEX n:Personne(nom)`<br>`WHERE n.nom = "Sylvain"` | Force l'utilisation de l'index `n:Personne(nom)` parmi ceux définis pour les nœuds |
| `MATCH (n:Personne)`<br>`USING SCAN n:Personne`<br>`WHERE n.nom = "Sylvain"` | Force l'utilisation du balayage sur le label `Personne` parmi ceux définis pour les nœuds |
| `MATCH (n:Personne {nom:"Sylvain"})-->(x)<--MATCH (m:Personne {nom:"Philippe"})`<br>`USING JOIN ON x`<br>`RETURN x`<br><br>Force l'utilisation d'un plan de jointure sur `m` et `n`. | |

**Indexation et emploi du planificateur de requêtes**

```
EXPLAIN
MATCH (n:Personne {nom:"Sylvain"})-[r]->(m:Personne {nom:"Philippe"})
RETURN r
```

Retourne un plan d'exécution théorique de la requête sans la lancer.

```
PROFILE
MATCH (n:Personne {nom:"Sylvain"})-[r]->(m:Personne {nom:"Philippe"})
RETURN r
```

Retourne le plan d'exécution réel de la requête à la fin de son exécution.

```
CYPHER PLANNER=rule
MATCH (n:Personne {nom:"Sylvain"})-[r]->(m:Personne {nom:"Philippe"})
RETURN r
```

Force l'utilisation du planificateur de type `rule`.

```
CYPHER PLANNER=cost
MATCH (n:Personne {nom:"Sylvain"})-[r]->(m:Personne {nom:"Philippe"})
RETURN r
```

Force l'utilisation du planificateur de type `cost` qui utilise les services de statistiques de Neo4j afin de déterminer le meilleur plan à exécuter.

```
CYPHER 2.2
MATCH (n:Personne {nom:"Sylvain"})-[r]->(m:Personne {nom:"Philippe"})
RETURN r
```

Force l'utilisation d'une version particulière de CYPHER à des fins de compatibilité descendante, les valeurs possibles sont : `1.9`, `2.2` et `2.3` (diffère selon les versions de Neo4j, si une valeur n'est plus admissible, un message affiche celles disponibles).

**Expression : CONSTRAINT**

```
CREATE CONSTRAINT ON (p:Personne) ASSERT p.nom IS UNIQUE
```

Crée une contrainte d'unicité (et l'index associé) basé sur le couple du label `Personne` avec la propriété `nom`.

```
DROP CONSTRAINT ON (p:Personne) ASSERT p.nom IS UNIQUE
```

Supprime la contrainte d'unicité.

```
CREATE CONSTRAINT ON (p:Personne) ASSERT EXISTS(p.nom)
```

Crée une contrainte forçant le contrôle de l'existence d'une proriété d'un nœud lors de sa création (Neo4j Enterprise uniquement).

| **Expression : CONSTRAINT** |
|---|
| `DROP CONSTRAINT ON (p:Personne) ASSERT EXISTS(p.nom)`<br><br>Supprime la contrainte de contrôle de l'existence d'une propriété sur un nœud (Neo4j Enterprise uniquement). |
| `CREATE CONSTRAINT ON ()-[ami:EST_AMI_AVEC]-() ASSERT EXISTS(p.date)`<br><br>Crée une contrainte forçant le contrôle de l'existence d'une propriété sur la relation lors de sa création (Neo4j Enterprise uniquement). |
| `DROP CONSTRAINT ON ()-[ami:EST_AMI_AVEC]-() ASSERT EXISTS(p.nom)`<br><br>Supprime la contrainte de contrôle de l'existence d'une propriété sur la relation (Neo4j Enterprise uniquement). |

## 4. Opérateurs, fonctions mathématiques et NULL

| **Opérateurs** | |
|---|---|
| Mathématiques | +, -, *, /, %, ^ |
| Comparaison | =, <>, <, >, <=, >= |
| Booléen | AND, OR, XOR, NOT |
| Chaîne de caractères | + |
| Collection | []+[], IN [], [x], [x .. y] |
| Expression régulière | =~ |

| **Fonctions mathématiques** | |
|---|---|
| `abs({expr})` | Valeur absolue |
| `rand()` | Génère une valeur aléatoire (ex : 0.3606221413404884) |
| `round({expr})` | Arrondit une valeur en décimale en valeur entière la plus proche (note : `round(10.5)` retourne 11) |
| `ceil({expr})` | Arrondit une valeur en décimale en valeur entière haute |
| `floor({expr})` | Arrondit une valeur en décimale en valeur entière basse |
| `sqrt({expr})` | Racine carrée |
| `sign({expr})` | 0 pour la valeur zéro, −1 pour une valeur négative, 1 pour une valeur positive |

| Fonctions mathématiques | |
|---|---|
| `sin({expr})` | Fonctions trigonométriques : cos, tan, cot, asin, acos, atan, atan2, haversin |
| `degrees({expr}), radians({expr}), pi()` | Convertit les radians en degrés, la fonction `radians()` fait l'inverse, `pi()` retourne π |
| `log10({expr}), log({expr}), exp({expr}), e()` | Logarithme base 10, logarithme népérien, puissance de la valeur, valeur de `e` |

**NULL, IS NULL**

- `NULL` est utilisé pour définir les valeurs manquantes ou non définies.
- `NULL` n'est pas égal à `NULL`. Ne pas connaître deux valeurs ne signifient pas qu'elles sont égales. L'expression `NULL = NULL` retourne `NULL` et non pas la valeur booléenne `true`. Pour vérifier la nullité d'une expression utiliser `IS NULL`.
- Les expressions arithmétiques, de comparaison et les appels de fonctions (hormis `coalesce`) retournent `NULL` si l'un des arguments est `null`.
- Les éléments manquants comme une propriété inexistante ou un élément qui n'existe pas dans une collection ont pour valeur `NULL`.
- Dans une clause `OPTIONAL MATCH`, `NULL` peut-être utilisé pour les parties manquantes d'un modèle relationnel.

## 5. Fonctions applicables aux relations et opérations sur les chemins

| Fonctions relationnelles | |
|---|---|
| `type(une_relation)` | Retourne le type de la relation sous forme de chaîne de caractères |
| `startNode(une_relation)` | Retourne le nœud de départ de la relation |
| `endNode(une_relation)` | Retourne le nœud d'arrivée de la relation |
| `id(une_relation)` | Retourne l'identifiant interne, généré par Neo4j, de la relation |

| Chemins | |
|---|---|
| `MATCH chemin=(n)-->(m)` | Assigne les chemins du nœud `n` vers le nœud `m` dans l'identificateur chemin |

| Chemins | |
|---|---|
| `length(chemin)` | Retourne la longueur du chemin (nombre de relations traversées) |
| `chemin=shortestPath((n)-->(m))` | Retourne un des chemins les plus courts |
| `chemin=allShortestPath((n)-->(m))` | Retourne tous les chemins les plus courts |
| `nodes(chemin)` | Retourne la collection de nœuds qui composent le chemin |
| `relationships(chemin)` ou `rels(chemin)` | Retourne la collection de relations qui composent le chemin |

```
MATCH chemin=(n)-->(m)
RETURN extract(noeud IN nodes(chemin) | noeud.propriete)
```

Assigne l'identificateur `chemin` et retourne une collection de valeurs pour une propriété de chacun des nœuds qui composent le chemin.

```
MATCH chemin = (debut) -[*]-> (fin)
FOREACH (r IN rels(path) | SET r.marque = TRUE)
```

Exécute une opération de modification de propriété sur chacune des relations qui composent le chemin.

## 6. Fonctions applicables aux nœuds et opérations sur les labels

| Fonctions applicables aux nœuds | |
|---|---|
| `id(un_noeud)` | Retourne l'identifiant interne, généré par Neo4j, du nœud |
| `size(n-->())` | Retourne le degré des relations sortantes du nœud `n` |
| `START n=node({id})` | Commence l'interrogation à partir du nœud retrouvé par son identifiant interne (indisponible depuis la version 2.2) |
| `START n=node({id1}), m=node({id2})` | Commence l'interrogation à partir des nœuds retrouvés par leur identifiant interne (indisponible depuis la version 2.2) |

```
START n=node:nodeIndexName(propriete={valeur})
```

Commence l'interrogation à partir du ou des nœuds retrouvés à partir d'une valeur de propriété. Cette propriété doit être auto-indexée par paramétrage.

| Opérations sur les labels | |
|---|---|
| `labels(un_noeud)` | Retourne une collection de labels pour un nœud donné |
| `MATCH (n:Personne)` | Retrouve les nœuds labellisés `Personne` |
| `WHERE (n:Personne)` | Vérifie l'existence du label `Personne` sur le nœud `n` |
| `CREATE (n:Personne {nom:"Sylvain"})` | Crée un nœud ayant un label `Personne`. |
| `SET n:Personne:Salarie` | Ajoute les labels `Personne` et `Salarie` au nœud `n` |
| `REMOVE n:Personne` | Retire le label `Personne` du nœud `n` |

# 7. Objets (maps), collections et fonctions applicables sur les collections

| Les objets (*maps*) | |
|---|---|
| `{}` | Un objet |
| `{nom:'Alice', age:38}` | Un objet avec deux propriétés |
| `{nom:'Alice', adresse:{ville:'Lyon', codePostal:'69003'}}` | Un objet porteur d'une propriété `adresse` ayant un autre objet pour valeur |
| `objet.nom, objet.age, objet.maCollection[0]` | Il est possible d'accéder directement aux valeurs de propriétés d'un objet au travers de la clé de la propriété (son nom). Les clés non existantes retournent une erreur |
| `MATCH (noeud:Personne) RETURN noeud` | Les nœuds comme les relations sont retournés sous forme d'objet |
| `KEYS (objet)` | Retourne une collection contenant les clés de toutes les propriétés présentes dans l'objet passé en paramètre. Les objets peuvent également être des nœuds ou des relations |
| `MERGE (p:Personne { nom: {param_objet}.nom}) ON CREATE SET p={param_objetmap}` | Les objets peuvent être passés en paramètres et il est possible d'accéder directement aux propriétés de ces paramètres |

| Les collections | |
|---|---|
| `[]` | Une collection vide |
| `RETURN ['a','b','c']` | Retourne une collection littérale |
| `RETURN length(['a','b','c'])` | Retourne la taille de la collection |
| `RETURN size(['a','b','c'])` | Retourne la taille de la collection |
| `WITH ['a','b','c'] as col`<br>`RETURN col[0]` | Retourne le premier élément de la collection |
| `WITH ['a','b','c'] as col`<br>`RETURN col[0..2]` | Retourne les éléments de la collection allant de la case 0 à la case 1 (la seconde valeur de la portée est exclusive). Les cases sans valeurs retournent `NULL`, le dépassement de portée est ignoré |
| `MATCH (noeud:Personne)`<br>`RETURN noeud` | Les nœuds comme les relations sont retournés sous forme d'objet |
| `RETURN range(1,5,2) AS coll` | Génère une collection de valeurs numériques allant de 1 à 5 en augmentant la valeur de 2 (le pas) à chaque fois (soit le résultat `[1,3,5]`). La valeur de pas est optionnelle |
| `MATCH n`<br>`UNWIND labels(n) as labs`<br>`RETURN distinct labs` | `UNWIND` éclate une collection en liste de résultats individuels |

| Fonctions applicables aux collections | |
|---|---|
| `RETURN length (['a','b','c'])` | Retourne la taille de la collection |
| `head({coll}), last({coll}),`<br>`tail({coll})` | `head` retourne le premier élément d'une collection, `last` le dernier élément, `tail` le reste de la collection |
| `extract(x IN maCollection \|`<br>`x.propriete)` | Crée une nouvelle collection remplie à partir de la propriété `x.propriete` |
| `filter(x IN maCollection`<br>`WHERE x.propriete <> {valeur})` | Crée une nouvelle collection pour les objets qui répondent à la contrainte `x.propriete <> {valeur}` |
| `MATCH n`<br>`WITH labels(n) as labs, n`<br>`RETURN reduce(s = "", x IN labs \| s +" "+ x)` | `reduce` est un accumulateur de résultats extraits d'une collection. Retourne une valeur littérale (numérique comme chaîne de caractères) |

| Fonctions applicables aux collections |
|---|
| `FOREACH (valeur IN maCollection \| CREATE (:Personne {nom:valeur}))`<br>Ajoute un élément dans le graphe en fonction de valeurs se trouvant dans une collection. |
| `[x IN maCollection WHERE x.propriete <> {valeur} \| x.autrePropriete]`<br>Écriture simplifiée issue de la combinaison des fonctions `FILTER` et `EXTRACT`. |

# 8. Prédicats

| Prédicats généraux | |
|---|---|
| `n.nom <> 'Christophe'` | Utilisation d'opérateur de comparaison |
| `has(n.nom)` | Utilisation d'une fonction |
| `n.age >= 30 AND n.age <= 40` | Utilisation d'opérateurs booléens pour combiner les prédicats |
| `n.age > 30 < m.age` | Inéquations combinées |
| `n:Personne` | Vérifie la présence d'un label |
| `n IS NULL` | Vérifie si quelque chose est nul |
| `NOT has(n.nom) OR n.nom = 'Philippe'` | Vérifie que la propriété n'existe pas OU que le prédicat suivant est vrai |
| `n.nom = 'Patricia'` | Retourne `NULL` si la propriété n'existe pas |
| `n.property =~ 'Syl.*'` | Utilisation d'une expression régulière |
| `n.property STARTSWITH ('Syl')` | Retourne vrai si la valeur de la propriété commence par 'Syl' |
| `n.property ENDSWITH ('vain')` | Retourne vrai si la valeur de la propriété se termine par 'vain' |
| `n.property CONTAINS ('ylv')` | Retourne vrai si la valeur de la propriété contient 'ylv' |
| `(n)-[:CONNAIT]->(m)` | Vérifie que le pattern retourne au moins un élément |
| `NOT (n)-[:CONNAIT]->(m)` | Exclut le pattern du résultat |
| `EXISTS ((n)-[:CONNAIT]->(m))` | Retourne `true` si le chemin existe lorsque `EXISTS` est exprimé sur un motif relationnel, si une propriété existe sur un nœud, une relation ou un objet (map), `false` sinon |

| Prédicats généraux | |
|---|---|
| `n.nom IN ['Christophe', 'Philippe']` | Vérifie la présence de la valeur d'une propriété dans une collection |

| Prédicats applicables aux collections | |
|---|---|
| `all(x IN collection WHERE has(x.propriete))` | Retourne vrai si le prédicat `has(x.propriete)` est vrai pour tous les éléments de la collection |
| `any(x IN collection WHERE has(x.propriete))` | Retourne vrai si le prédicat `has(x.propriete)` est vrai pour au moins un des éléments de la collection |
| `none(x IN collection WHERE has(x.propriete))` | Retourne vrai si le prédicat `has(x.propriete)` est faux pour tous les éléments de la collection |
| `single(x IN collection WHERE has(x.propriete))` | Retourne vrai si le prédicat `has(x.propriete)` est vrai pour un et un seul élément de la collection |

# 9. Agrégation

| Agrégation | |
|---|---|
| `count(*)` | Compte le nombre de lignes retrouvées |
| `count(identificateur)` | Compte le nombre de valeurs non nulles |
| `count(DISTINCT identificateur)` | Toutes les fonctions d'agrégation supportent le modificateur `DISTINCT`, celui-ci omet les doublons |
| `collect(n.nom)` | Collecte les valeurs non nulles (retourne une collection) |
| `sum(n.nombre)` | Fait la somme des valeurs numériques. Les autres fonctions similaires sont `avg` (moyenne), `min`, `max` |
| `percentileDisc(n.age, 0.5)` | Retourne les centiles discrets (paramètre ayant une valeur allant de 0.0 à 1.0) |
| `percentileCont(n.age, 0.5)` | Retourne les centiles continus (paramètre ayant une valeur allant de 0.0 à 1.0) |
| `stdev(n.age)` | Écart-type pour un échantillon de population |

| Agrégation | |
|---|---|
| `stdevp(n.age)` | Écart-type pour une population complète |

# 10. CASE et FOREACH

| Expression : CASE | |
|---|---|
| `CASE n.couleur`<br>`WHEN 'bleu' THEN 1`<br>`WHEN 'marron' THEN 2`<br>`ELSE 3`<br>`END` | Retourne la valeur de `THEN` lorsque la valeur de `CASE` correspond à la valeur de `WHEN`. `ELSE` est optionnel et représente toutes les autres valeurs |
| `CASE`<br>`WHEN n.couleur = 'bleu' THEN 1`<br>`WHEN n.age < 10 THEN 2`<br>`ELSE 3`<br>`END` | Retourne la valeur de `THEN` lorsque la valeur du prédicat `WHEN` est vrai. Les prédicats sont évalués dans l'ordre |

| Expression : FOREACH |
|---|
| `FOREACH (n IN collection \| SET n.couleur = 'Bleu')`<br>Exécute une opération de modification de propriété sur chacun des éléments de graphe qui composent la collection. |
| `FOREACH (n IN collection \| CREATE (v:Voiture {peinture:n.couleur}))`<br>Exécute une opération de création pour chacun des éléments qui composent la collection. |
| `FOREACH (n IN collection \| REMOVE n.couleur)`<br>Exécute une opération de suppression de propriété pour chacun des éléments qui composent la collection. |
| `FOREACH (n IN collection \| DELETE n)`<br>Exécute une opération de suppression des éléments de graphe qui composent la collection. |

# 11. Fonctions générales et chaînes de caractères

| Fonctions générales | |
|---|---|
| `coalesce(n.couleur, 'bleu')` | Retourne bleu si `n.couleur` est nul, `n.couleur` sinon |

| Fonctions générales | |
|---|---|
| `timestamp()` | Retourne une valeur de temps exprimée en millisecondes comptée à partir du 1er janvier 1970 à minuit (UTC) |
| `id(noeud_ou_relation)` | Retourne l'identifiant interne d'un nœud ou d'une relation |
| `toInt({expression})` | Convertit le résultat de l'expression en nombre entier si cela est possible, en `NULL` sinon |
| `toFloat({expression})` | Convertit le résultat de l'expression en nombre décimal si cela est possible, en `NULL` sinon |
| `keys({expression})` | Retourne toutes les clés des propriétés d'un nœud, d'une relation ou d'un objet (map) sous forme d'une collection |

| Fonctions applicables aux chaînes de caractères | |
|---|---|
| `toString({expression})` | Retourne sous forme de chaîne de caractères le résultat de l'expression |
| `replace({original}, {recherché}, {remplacement})` | Retourne une chaîne de caractères où la chaîne `remplacement` est substitué à la chaîne `recherché` dans la chaîne `original` |
| `substring({original}, {début}, {longueur})` | Retourne une chaîne de caractères extraite de la chaîne `original` à partir de l'emplacement `début` et éventuellement pour un nombre de caractères équivalent à `longueur` (optionnel) |
| `left({original}, {longueur})` et `right({original}, {longueur})` | Retourne une chaîne composée du nombre de caractères équivalent à `longueur` en partant de la gauche de la chaîne `original` (`left`) ou bien de sa droite (`right`) |
| `trim({original}), ltrim({original})` et `rtrim({original})` | Retire tout les espaces superflus (`trim`), à gauche de la chaîne (`ltrim`) ou à droite de la chaîne (`rtrim`) |
| `upper({original}), lower({original})` | Retourne la chaîne de caractères `original` en majuscules (`upper`) ou en minuscules (`lower`) |
| `split({original}, {séparateur})` | Découpe la chaîne de caractères `original` en une collection de chaînes extraites à partir de chaque chaîne `séparateur` |

# 12. Import de données

**Import CSV**

```
LOAD CSV
FROM 'file:///Users/sroussy/Documents/neo4j.csv' AS ligneCSV
CREATE (p:Personne { nom: ligneCSV[0]})
```

Génère un nœud `Personne` affectant à la propriété `nom` la valeur de la première colonne du fichier, pour chaque ligne du fichier CSV.

```
LOAD CSV WITH HEADERS
FROM 'file:///Users/sroussy/Documents/neo4j.csv' AS ligneCSV
CREATE (p:Personne { nom: ligneCSV[0]})
```

Génère un nœud `Personne` affectant à la propriété `nom` la valeur de la première colonne du fichier, pour chaque ligne du fichier CSV, en prenant en compte que la première ligne du fichier décrit les en-têtes des colonnes.

```
LOAD CSV
FROM 'file:///Users/sroussy/Documents/neo4j.csv' AS ligneCSV
FIELDTERMINATOR ';'
CREATE (p:Personne { nom: ligneCSV[0]})
```

`FIELDTERMINATOR` permet de spécifier un séparateur de colonnes spécifique (par défaut le séparateur de colonnes est le caractère virgule).

```
USING PERIODIC COMMIT 2000
LOAD CSV
FROM 'file:///Users/sroussy/Documents/neo4j.csv' AS ligneCSV
CREATE (p:Personne { nom: ligneCSV[0]})
```

`USING PERIODIC COMMIT` permet de terminer une transaction (*Commit*) toutes les 2000 mises à jour du graphe.