

# 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 v2.1.4](#).

## 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
<code>MATCH ()--()</code> Nœuds anonymes reliés par une relation.
<code>MATCH (n)--(m)</code> Nœuds identifiés par <code>n</code> et <code>m</code> reliés par une relation.
<code>MATCH (n)--&gt;(m)&lt;--(o), (m)&lt;--(p)</code> Nœud <code>m</code> ayant trois relations incidentes orientées.
<code>MATCH (n)-[r]-&gt;(m)</code> Relation orientée identifiée par <code>r</code> .
<code>MATCH (n)-[*]-&gt;(m)</code> Nœuds identifiés par <code>n</code> et <code>m</code> reliés quel que soit le nombre de relations traversées.
<code>MATCH (n)-[r*]-&gt;(m)</code> Nœuds identifiés par <code>n</code> et <code>m</code> reliés quel que soit le nombre de relations traversées identifiées par <code>r</code> .

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

<b>Expression : WHERE</b>	
<code>WHERE n.nom = "Alice"</code>	Dont le nom est <code>Alice</code>
<code>WHERE n.nom &lt;&gt; "Alice"</code>	Dont le nom n'est pas <code>Alice</code>
<code>WHERE n.nom =~ "Alic.*"</code>	Dont le nom commence par <code>Alic</code>
<code>WHERE NOT n&lt;--()</code>	Où <code>n</code> ne possède aucune relation entrante

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

<b>Expression : RETURN</b>	
<code>RETURN *</code>	Retourne toutes les valeurs de tous les identificateurs

Expression : RETURN	
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}	Passes un certain nombre de résultats
LIMIT {nombre_limite}	Limite le nombre de résultats
SKIP {nombre_a_passer} 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) WHERE quelqun.nom = {nom} WITH quelqun, count(ami) AS amis WHERE amis > 10 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) WITH quelqun, count(ami) AS amis ORDER BY amis DESC SKIP 1 LIMIT 3 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) RETURN b.nom UNION MATCH (a)-[:AIME]->(b) RETURN b.nom	Retourne une union distincte (pas de ligne dupliquée) des deux requêtes
MATCH (a)-[:CONNAIT]->(b) RETURN b.nom UNION ALL MATCH (a)-[:AIME]->(b) 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

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

#### Expression : SET

<code>SET nr.titre = "Alice au pays des merveilles", nr.auteur = "Lewis Carroll"</code>	
Crée ou met à jour des propriétés sur un nœud ou une relation.	
<code>SET nr=objet</code>	Pose un ensemble de propriétés (objet <code>{}</code> ) ou remplace l'ensemble des propriétés sur un nœud ou une relation

<b>Expression : SET</b>	
SET n:Personne	Ajoute le label <b>Personne</b> au nœud n

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

<b>Expression : DELETE</b>	
DELETE n, r	Supprime un nœud et une relation

<b>Expression : REMOVE</b>	
REMOVE n.nom	Supprime la propriété <b>nom</b> sur le nœud n
REMOVE n:Personne	Supprime le label <b>Personne</b> du nœud n

<b>Expression : INDEX</b>	
CREATE INDEX ON :Personne(nom)	Crée un index basé sur le couple du label <b>Personne</b> avec la propriété <b>nom</b>
<pre>MATCH (n:Personne) WHERE n.nom = "Sylvain"</pre>	Retrouve le nœud <b>Sylvain</b> en utilisant l'index <b>:Personne(nom)</b> (index retrouvé par l'opération

Expression : INDEX	
	d'égalité. À noter que l'utilisation d'une fonction telle que <code>WHERE lower(n.nom) = "sylvain"</code> ne permet pas de déterminer l'index à utiliser)
<code>MATCH (n:Personne)</code> <code>USING INDEX n:Personne(nom)</code> <code>WHERE n.nom = "Sylvain"</code>	Force l'utilisation de l'index <code>n:Personne(nom)</code> parmi ceux définis pour le nœud
<code>DROP INDEX ON :Personne(nom)</code>	Supprime l'index basé sur le couple du label <code>Personne</code> avec la propriété <code>nom</code>

Expression : CONSTRAINT	
<code>CREATE CONSTRAINT ON (p:Personne) ASSERT p.nom IS UNIQUE</code>	Crée une contrainte d'unicité (et l'index associé) basé sur le couple du label <code>Personne</code> avec la propriété <code>nom</code> .
<code>DROP CONSTRAINT ON (p:Personne) ASSERT p.nom IS UNIQUE</code>	Supprime la contrainte d'unicité.

### 3. Opérateurs, fonctions mathématiques et NULL

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

Fonctions mathématiques	
<code>abs({expr})</code>	Valeur absolue
<code>rand()</code>	Génère une valeur aléatoire (ex : 0.3606221413404884)
<code>round({expr})</code>	Arrondit une valeur en décimale en valeur entière la plus proche (note : <code>round(10.5)</code> retourne 11)
<code>ceil({expr})</code>	Arrondit une valeur en décimale en valeur entière haute

Fonctions mathématiques	
<code>floor({expr})</code>	Arrondit une valeur en décimale en valeur entière basse
<code>sqrt({expr})</code>	Racine carrée
<code>sign({expr})</code>	0 pour la valeur zéro, -1 pour une valeur négative, 1 pour une valeur positive
<code>sin({expr})</code>	Fonctions trigonométriques : <code>cos</code> , <code>tan</code> , <code>cot</code> , <code>asin</code> , <code>acos</code> , <code>atan</code> , <code>atan2</code> , <code>haversin</code>
<code>degrees({expr})</code> , <code>radians({expr})</code> , <code>pi()</code>	Convertit les radians en degrés, la fonction <code>radians()</code> fait l'inverse, <code>pi()</code> retourne $\pi$
<code>log10({expr})</code> , <code>log({expr})</code> , <code>exp({expr})</code> , <code>e()</code>	Logarithme base 10, logarithme népérien, puissance de la valeur, valeur de e

NULL, IS NULL
<ul style="list-style-type: none"> <li>• <code>NULL</code> est utilisé pour définir les valeurs manquantes ou non définies.</li> <li>• <code>NULL</code> n'est pas égal à <code>NULL</code>. Ne pas connaître deux valeurs ne signifie pas qu'elles sont égales. L'expression <code>NULL = NULL</code> retourne <code>NULL</code> et non pas la valeur booléenne <code>true</code>. Pour vérifier la nullité d'une expression utiliser <code>IS NULL</code>.</li> <li>• Les expressions arithmétiques, de comparaison et les appels de fonctions (hormis <code>coalesce</code>) retournent <code>NULL</code> si l'un des arguments est <code>null</code>.</li> <li>• Les éléments manquants comme une propriété inexistante ou un élément qui n'existe pas dans une collection ont pour valeur <code>NULL</code>.</li> <li>• Dans une clause <code>OPTIONAL MATCH</code>, <code>NULL</code> peut-être utilisé pour les parties manquantes d'un modèle relationnel.</li> </ul>

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

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

Chemins	
<code>MATCH chemin=(n)--&gt;(m)</code>	Assigne les chemins du nœud <code>n</code> vers le nœud <code>m</code> dans l'identificateur <code>chemin</code>
<code>length(chemin)</code>	Retourne la longueur du chemin (nombre de relations traversées)
<code>chemin=shortestPath((n)--&gt;(m))</code>	Retourne un des chemins les plus courts
<code>chemin=allShortestPath((n)--&gt;(m))</code>	Retourne tous les chemins les plus courts
<code>nodes(chemin)</code>	Retourne la collection de nœuds qui composent le chemin
<code>relationships(chemin)</code> ou <code>rels(chemin)</code>	Retourne la collection de relations qui composent le chemin
<pre>MATCH chemin=(n)--&gt;(m) RETURN extract( noeud IN nodes(chemin)   noeud.propriete)</pre> <p>Assigne l'identificateur <code>chemin</code> et retourne une collection de valeurs pour une propriété de chacun des nœuds qui composent le chemin.</p>	
<pre>MATCH chemin = (debut) -[*]-&gt; (fin) FOREACH ( r IN rels(path)   SET r.marque = TRUE)</pre> <p>Exécute une opération de modification de propriété sur chacune des relations qui composent le chemin.</p>	

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

Fonctions applicables aux nœuds	
<code>id(un_noeud)</code>	Retourne l'identifiant interne, généré par Neo4j, du nœud
<code>START n=node({id})</code>	Commence l'interrogation à partir du nœud retrouvé par son identifiant interne
<code>START n=node({id1}), m=node({id2})</code>	Commence l'interrogation à partir des nœuds retrouvés par leur identifiant interne
<pre>START n=node:nodeIndexName(propriete={valeur})</pre> <p>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.</p>	



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

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

Les objets ( <i>maps</i> )	
<code>{}</code>	Un objet
<code>{nom:'Alice', age:38}</code>	Un objet avec deux propriétés
<code>{nom:'Alice', adresse:{ville:'Lyon', codePostal:'69003'}}</code>	Un objet porteur d'une propriété <code>adresse</code> ayant un autre objet pour valeur
<code>objet.nom, objet.age, objet.maCollection[0]</code>	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
<code>MATCH (noeud:Personne) RETURN noeud</code>	Les nœuds comme les relations sont retournés sous forme d'objet
<code>MERGE (p:Personne { nom: {param_objet}.nom}) ON CREATE SET p={param_objetmap}</code>	Les objets peuvent être passés en paramètres et il est possible d'accéder directement aux propriétés de ces paramètres

Les collections	
<code>[]</code>	Une collection vide
<code>RETURN ['a','b','c']</code>	Retourne une collection littérale
<code>RETURN length (['a','b','c'])</code>	Retourne la taille de la collection
<code>WITH ['a','b','c'] as col</code>	Retourne le premier élément de la collection

Les collections	
<code>RETURN col[0]</code>	
<code>WITH ['a','b','c'] as col RETURN col[0..2]</code>	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 <b>NULL</b> , le dépassement de portée est ignoré
<code>MATCH (noeud:Personne) RETURN noeud</code>	Les nœuds comme les relations sont retournés sous forme d'objet
<code>RETURN range(1,5,2) AS coll</code>	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 <b>[1, 3, 5]</b> ). La valeur de pas est optionnelle
<code>MATCH n UNWIND labels(n) as labs RETURN distinct labs</code>	<b>UNWIND</b> éclate une collection en liste de résultats individuels

Fonctions applicables aux collections	
<code>RETURN length (['a','b','c'])</code>	Retourne la taille de la collection
<code>head({coll}), last({coll}), tail({coll})</code>	<b>head</b> retourne le premier élément d'une collection, <b>last</b> le dernier élément, <b>tail</b> le reste de la collection
<code>extract(x IN maCollection   x.propriete)</code>	Crée une nouvelle collection remplie à partir de la propriété <b>x.propriete</b>
<code>filter(x IN maCollection WHERE x.propriete &lt;&gt; {valeur})</code>	Crée une nouvelle collection pour les objets qui répondent à la contrainte <b>x.propriete &lt;&gt; {valeur}</b>
<code>MATCH n WITH labels(n) as labs, n RETURN reduce(s = "", x IN labs   s + " "+ x)</code>	<b>reduce</b> est un accumulateur de résultats extraits d'une collection. Retourne une valeur littérale (numérique comme chaîne de caractères)
<code>FOREACH (valeur IN maCollection   CREATE (:Personne {nom:valeur}))</code>	Ajoute un élément dans le graphe en fonction de valeurs se trouvant dans une collection.
<code>[x IN maCollection WHERE x.propriete &lt;&gt; {valeur}   x.autrePropriete]</code>	Écriture simplifiée issue de la combinaison des fonctions <b>FILTER</b> et <b>EXTRACT</b> .

## 7. Prédicats

Prédicats généraux	
<code>n.nom &lt;&gt; 'Christophe'</code>	Utilisation d'opérateur de comparaison
<code>has(n.nom)</code>	Utilisation d'une fonction
<code>n.age &gt;= 30 AND n.age &lt;= 40</code>	Utilisation d'opérateurs booléens pour combiner les prédicats
<code>n:Personne</code>	Vérifie la présence d'un label
<code>n IS NULL</code>	Vérifie si quelque chose est nul
<code>NOT has(n.nom) OR n.nom = 'Philippe'</code>	Vérifie que la propriété n'existe pas OU que le prédicat suivant est vrai
<code>n.nom = 'Patricia'</code>	Retourne <code>NULL</code> si la propriété n'existe pas
<code>n.property =~ 'Syl.*'</code>	Utilisation d'une expression régulière
<code>(n)-[:CONNAIT]-&gt;(m)</code>	Vérifie que le pattern retourne au moins un élément
<code>NOT (n)-[:CONNAIT]-&gt;(m)</code>	Exclut le pattern du résultat
<code>n.nom IN ['Christophe', 'Philippe']</code>	Vérifie la présence de la valeur d'une propriété dans une collection

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

## 8. Agrégation

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

## 9. CASE et FOREACH

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

**Expression : FOREACH**

`FOREACH (n IN collection | SET n.couleur = 'Bleu')`

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}))`

Exécute une opération de création pour chacun des éléments qui composent la collection.

`FOREACH (n IN collection | REMOVE n.couleur)`

Exécute une opération de suppression de propriété pour chacun des éléments qui composent la collection.

`FOREACH (n IN collection | DELETE n)`

Exécute une opération de suppression des éléments de graphe qui composent la collection.

## 10. Fonctions générales et chaînes de caractères

### Fonctions générales

<code>coalesce(n.couleur, 'bleu')</code>	Retourne <code>bleu</code> si <code>n.couleur</code> est nul, <code>n.couleur</code> sinon
<code>timestamp()</code>	Retourne une valeur de temps exprimée en millisecondes comptée à partir du 1 <sup>er</sup> janvier 1970 à minuit (UTC)
<code>id(noeud_ou_relation)</code>	Retourne l'identifiant interne d'un nœud ou d'une relation
<code>toInt({expression})</code>	Convertit le résultat de l'expression en nombre entier si cela est possible, en <code>NULL</code> sinon
<code>toFloat({expression})</code>	Convertit le résultat de l'expression en nombre décimal si cela est possible, en <code>NULL</code> sinon

### Fonctions applicables aux chaînes de caractères

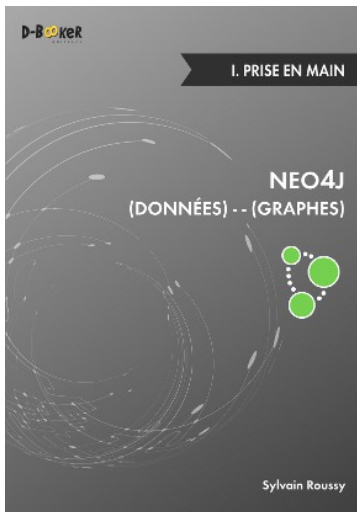
<code>toString({expression})</code>	Retourne sous forme de chaîne de caractères le résultat de l'expression
<code>replace({original}, {recherché}, {remplacement})</code>	Retourne une chaîne de caractères où la chaîne <code>recherche</code> est substitué à la chaîne <code>original</code>

Fonctions applicables aux chaînes de caractères	
<code>substring({original}, {début}, {longueur})</code>	Retourne une chaîne de caractères extraite de la chaîne <b>original</b> à partir de l'emplacement <b>début</b> et éventuellement pour un nombre de caractères équivalent à <b>longueur</b> (optionnel)
<code>left({original}, {longueur})</code> et <code>right({original}, {longueur})</code>	Retourne une chaîne composée du nombre de caractères équivalent à <b>longueur</b> en partant de la gauche de la chaîne <b>original</b> ( <b>left</b> ) ou bien de sa droite ( <b>right</b> )
<code>trim({original})</code> , <code>ltrim({original})</code> et <code>rtrim({original})</code>	Retire tout les espaces superflus ( <b>trim</b> ), à gauche de la chaîne ( <b>ltrim</b> ) ou à droite de la chaîne ( <b>rtrim</b> )
<code>upper({original})</code> , <code>lower({original})</code>	Retourne la chaîne de caractères <b>original</b> en majuscules ( <b>upper</b> ) ou en minuscules ( <b>lower</b> )
<code>split({original}, {séparateur})</code>	Découpe la chaîne de caractères <b>original</b> en une collection de chaînes extraites à partir de chaque chaîne <b>séparateur</b>

Cet aide-mémoire  
a été établi par Sylvain Roussy  
pour le livre

## Neo4j : Des données et des graphes

### 1. Prise en main



- Découvrir Neo4j
  - Comprendre Neo4j
  - Évaluer Neo4j : CYPHER
  - Évaluer Neo4j : l'API REST
- Annexes  
Aide-mémoire CYPHER

240 pages

Ouvrage disponible en version  
numérique et imprimée

Plus d'informations  
sur le site des  
[éditions D-Booker](http://www.d-booker.com)