**Язык Cypher. Запись, изменение значений.**

Общая структура запроса для чтения-записи:

**[MATCH WHERE]**

**[OPTIONAL MATCH WHERE]**

**[WITH [ORDER BY] [SKIP] [LIMIT]]**

**(CREATE | MERGE)\***

**[SET|DELETE|REMOVE|FOREACH]\***

**[RETURN [ORDER BY] [SKIP] [LIMIT]]**

Структура запроса для записи.

**(CREATE | MERGE)\***

**[SET|DELETE|REMOVE|FOREACH]\***

**[RETURN [ORDER BY] [SKIP] [LIMIT]]**

**SET**

**SET n.property1 = $value1,**

**n.property2 = $value2**

Изменение или создание свойства.

**SET n = $map**

Задание всех свойств хранимых в $map. При задании будут удалены все имеющиеся свойства ранее.

**SET n += $map**

Добавление и изменение свойств переменной n из $map. Неповторяющиеся свойства будут оставлены.

**SET n:Person:Actor**

Добавляет метки Person и Actor вершинам n.

**UNION**

**MATCH (a)-[:KNOWS]->(b)**

**RETURN b.name**

**UNION**

**MATCH (a)-[:LOVES]->(b)**

**RETURN b.name**

В качестве результата запроса будет возвращено объединение результатов (без повторений) двух запросов, при этом типы возвращаемых результатов должны совпадать.

**MATCH (a)-[:KNOWS]->(b)**

**RETURN b.name**

**UNION ALL**

**MATCH (a)-[:LOVES]->(b)**

**RETURN b.name**

Тоже самое что и выше, однако результаты могут повторятся.

**CREATE**

**CREATE (n {name: $value})**

Создание вершины с заданным свойствам.

**CREATE (n $map)**

Создание вершины с заданными свойствами в переменной **$map**

**UNWIND $listOfMaps AS properties**

**CREATE (n) SET n = properties**

Создается узлы с заданными свойствами.

**CREATE (n)-[r:KNOWS]->(m)**

Создается ребра с заданным типом и направлением, а также осуществляется привязка переменной **r**.

**CREATE (n)-[:LOVES {since: $value}]->(m)**

Создается ребро с заданным типом, направлением и свойством.

**MERGE**

**MERGE (n:Person {name: $value})**

**ON CREATE SET n.created = timestamp()**

**ON MATCH SET**

**n.counter = coalesce(n.counter, 0) + 1,**

**n.accessTime = timestamp()**

Проверяет, существует ли указанный шаблон в графе. Если нет, он создает соответствующую структуру. Используйте ON CREATE и ON MATCH для условных обновлений.

**MATCH (a:Person {name: $value1}),**

**(b:Person {name: $value2})**

**MERGE (a)-[r:LOVES]->(b)**

MERGE находит или создает отношение r между вершинами найденными в запросе.

**MATCH (a:Person {name: $value1})**

**MERGE**

**(a)-[r:KNOWS]->(b:Person {name: $value3})**

MERGE находит или создает подграфы, прикрепленные к узлу.

**DELETE**

**DELETE n, r**

Удалить вершины и ребра связанные с переменными n и r.

**DETACH DELETE n**

Удалить вершины, связанные с переменной n. Кроме того автоматически будут удалены ребра, ведущие в данную вершину.

**MATCH (n)**

**DETACH DELETE n**

Полное удаление базы данных. Будут удалены все вершины и ребра с ними связанные.

**CALL subquery**

**CALL {**

**MATCH (p:Person)-[:FRIEND\_OF]->(other:Person)**

**RETURN p, other**

**UNION**

**MATCH (p:Child)-[:CHILD\_OF]->(other:Parent)**

**RETURN p, other**

**}**

Данная конструкция позволяет формировать подзапрос, который в дальнейшем может быть обработан.

**REMOVE, FOREACH**

**REMOVE n:Person**

Удалить метку у вершины n.

**REMOVE n.property**

Удалить свойство property у вершины.

**FOREACH (r IN relationships(path) |**

**SET r.marked = true)**

Для каждого ребра из пути path будет добавлено свойство marked и использовано значение true.

**FOREACH (value IN coll |**

**CREATE (:Person {name: value}))**

Для каждого значения из списка coll будет создана вершина и в нее будет добавлено свойство value.

**Labels**

**MERGE (n:Person {name: $value})**

Находит или создает уникальные вершины с меткой и свойствами.

**WHERE (n:Person)**

Проверка существование метки **Person** у вершины **n**.

**labels(n)**

Все метки вершины в виде списка

**Lists**

**['a', 'b', 'c'] AS list**

Элементы списка помещаются в квадратные скобки.

**size($list) AS len, $list[0] AS value**

Списки можно передавать как параметры.

**range($firstNum, $lastNum, $step) AS list**

**range()** создает список чисел (шаг не является обязательным), другие функции, возвращающие списки: **labels()**, **nodes()**, **relationships()**.

**MATCH p = (a)-[:KNOWS\*]->()**

**RETURN relationships(p) AS r**

Список отношений, содержащих путь переменной длины, может быть возвращен с помощью именованных путей и relationships().

**RETURN matchedNode.list[0] AS value,**

**size(matchedNode.list) AS len**

Свойства могут быть списками строк, чисел или булевских значений.

**list[$idx] AS value,**

**list[$startIdx..$endIdx] AS slice**

Доступ к элементам списка можно получить с помощью индексов **$idx** в кадратных скобках. Недействительные индексы возвращают значение **null**. Срезы можно извлекать с интервалами от **start\_idx** до **end\_idx**, каждый из которых может быть опущен или отрицателен. Элементы вне диапазона игнорируются.

**UNWIND $names AS name**

**MATCH (n {name: name})**

**RETURN avg(n.age)**

С UNWIND любой список можно преобразовать обратно в отдельные строки. В примере выбираются все имена из списка имен.

**MATCH (a)**

**RETURN [(a)-->(b) WHERE b.name = 'Bob' | b.age]**

Шаблоны могут быть использованы для осуществления проекции из выборки прямо в список.

**MATCH (person)**

**RETURN person { .name, .age}**

Для пар ключ-значений могут быть построены проекции. В данном примере для всех вершин будут выделены две пары с ключами **name** и **age**.

**Карты, пары ключ-значения (Maps)**

**{name: 'Alice', age: 38,**

**address: {city: 'London', residential: true}}**

Карты объявляются в фигурных скобках, как и карты свойств. Списки в качестве значений поддерживаются.

**WITH {person: {name: 'Anne', age: 25}} AS p**

**RETURN p.person.name**

Доступ к свойству вложенной карты.

**MATCH (matchedNode:Person)**

**RETURN matchedNode**

Узлы и отношения возвращаются как карты их данных: map.name, map.age, map.children[0]. Доступ к записям карты можно получить по их ключам. Неверные ключи приводят к ошибке.

**Предикаты**

**n.property <> $value**

Использование операторов сравнения.

**exists(n.property)**

Применение функций.

**n.number >= 1 AND n.number <= 10**

Использование булевских операторов для комбинирования предикатов.

**1 <= n.number <= 10**

Доступно применение операторов в связке.

**variable IS NULL**

Проверка на равенство null чего-либо.

**NOT exists(n.property) OR n.property = $value**

Либо свойство не существует, либо предикат верен.

**n.property = $value**

Не существующие свойства вернут null, который не равен чему-либо.

**n["property"] = $value**

Доступ к свойствам также можно получить с помощью динамически вычисляемого имени свойства.

**n.property STARTS WITH 'Tim' OR**

**n.property ENDS WITH 'n' OR**

**n.property CONTAINS 'goodie'**

Выборка, основанная на проверки строки.

**n.property =~ 'Tim.\*'**

Проверка на соответствие сроки некоторому регулярному выражению (шаблону).

**NOT (n)-[:KNOWS]->(m)**

Исключения из результатов (n)-[:KNOWS]->(m) переменных, соответствующих шаблону.

**n.property IN [$value1, $value2]**

Проверка на вхождения элемента в списке.

**List predicates**

**all(x IN coll WHERE exists(x.property))**

Возвращает true, если выполняется условие для всех элементов в списке.

**any(x IN coll WHERE exists(x.property))**

Возвращает true, если выполняется условие хотя бы для одного элемента в списке.

**none(x IN coll WHERE exists(x.property))**

Возвращает true, если не выполняется условие для всех элементов в списке.

**single(x IN coll WHERE exists(x.property))**

Возвращает true, если условие выполняется точно для одного элемента в списке.

**List expressions**

**size($list)**

Количество элементов в списке.

**reverse($list)**

Инверсированное расположение элементов в списке.

**head($list), last($list), tail($list)**

head() – возвращает первые элемент в списке,

last() – последний элемент в списке. the last element of the list.

tail() – возвращает все элементы за исключением первого элемента.

Все функции вернут null для пустого списка.

**[x IN list | x.prop]**

Возвращает список значений выражения для каждого элемента в исходном списке.

**[x IN list WHERE x.prop <> $value]**

Осуществляется фильтрация элементов списка по предикату.

**[x IN list WHERE x.prop <> $value | x.prop]**

Комбинирование двух операций выше.

**reduce(s = "", x IN list | s + x.prop)**

Вычисляется выражение для каждого результата в списке и аккумулирует результат.