**Язык Cypher. Некоторые функции. Индексы. Ограничения целостности.**

**Некоторые советы по выполнению запросов.**

По возможности старайтесь использовать параметры вместо литералов, когда это возможно. Это позволяет Cypher повторно использовать ваши запросы вместо нового поиска и построения нового плана выполнения.

Всегда устанавливайте верхний лимит для путей переменной длины. Без установки данного лимита возможен обход всех вершин в графе по ошибке.

Возвращайте данные, которые вам нужны. Старайтесь избегать возвращения всех вершин и отношений в графе.

Используйте PROFILE / EXPLAIN для анализа исполнения своих запросов.

**Некоторые функции**

**coalesce(n.property, $defaultValue)**

Функция возвращает первое не нулевое значение.

**timestamp()**

Функция возвращает количество миллисекунд, прошедших с 1 января 1970 по Гринвичу.

**id(nodeOrRelationship)**

Внутреннее id некоторого отношения(ребра) или вершины.

**toInteger($expr)**

**toFloat($expr)**

**toBoolean($expr)**

Функция конвертирует параметр в указанный тип данных, если это невозможно, то возвращает null.

**keys($expr)**

Возвращает список, содержащий ключи словарей, ассоциированных с вершинами, отношениями или просто некоторого словаря.

**properties($expr)**

Возвращает словарь, который содержит все свойства вершины или отношения.

**Функции по работе со временем**

**date("2018-04-05")**

Возвращает дату, полученную из строки.

**localtime("12:45:30.25")**

Возвращает время без часового пояса.

**time("12:45:30.25+01:00")**

Возвращает время в указанном часовом поясе.

**localdatetime("2018-04-05T12:34:00")**

Возвращает дату и время без часового пояса.

**datetime("2018-04-05T12:34:00[Europe/Berlin]")**

Возвращает дату и время в указанном часовом поясе.

**date({year: $year, month: $month, day: $day})**

В качестве параметра для всех временных функций могут быть поданы словари. В данном примере будет возвращена дата, составленная из месяца и дня. Каждая функция поддерживает различные наборы свойств в словаре.

**datetime({date: $date, time: $time})**

Временной тип может быть создан путем комбинирования других типов. В данном примере будет возвращена переменная из даты и времени.

**date({date: $datetime, day: 5})**

Временные типы могут быть созданы исходя из более сложных типов, а также путем перезаписи отдельных компонент. В данном примере была создана дата из даты-времени и при этом день был перезаписан.

**WITH date("2018-04-05") AS d**

**RETURN d.year, d.month, d.day, d.week, d.dayOfWeek**

Возможно обращения к отдельным компонентам даты.

**Функции для работы с временными отрезками.**

**duration("P1Y2M10DT12H45M30.25S")**

Возвращает 1 год, 2 месяца, 10 дней, 12 часов, 45 минут и 30.25 секунд.

**duration.between($date1,$date2)**

Возвращает разницу между двумя датами.

**WITH duration("P1Y2M10DT12H45M") AS d**

**RETURN d.years, d.months, d.days, d.hours, d.minutes**

Возвращает 1 год, 14 месяцев, 10 дней, 12 часов и 765 минут.

**WITH duration("P1Y2M10DT12H45M") AS d**

**RETURN d.years, d.monthsOfYear, d.days, d.hours, d.minutesOfHour**

Возвращает 1 год, 2 месяца, 10 дней, 12 часов и 45 минут.

**date("2015-01-01") + duration("P1Y1M1D")**

В качестве результата будет возвращена дата 2016-02-02. Вычитание также доступно.

**duration("PT30S") \* 10**

Возвращает продолжительность 5 минут. Деление также доступно.

**Индексы**

**CREATE INDEX FOR (p:Person) ON (p.name)**

Создание индекса для метки Person со свойством имя.

**CREATE INDEX index\_name FOR (p:Person) ON (p.age)**

Создание индекса для метки Person со свойством возраст и именем index\_name.

**CREATE INDEX FOR (p:Person) ON (p.name, p.age)**

Создание сложного индекса для метки Person со свойствами имя и возраст.

**MATCH (n:Person) WHERE n.name = $value**

Индексация свойств используются для операций сравнений. Следует отметить, что для примера toLower(n.name) = $value индексация не используется.

**MATCH (n:Person)**

**WHERE n.name IN [$value]**

Проиндексированные свойства могут быть использованы для проверки в рамках предложения IN.

**MATCH (n:Person)**

**WHERE n.name = $value and n.age = $value2**

Проиндексированные свойства автоматически используются для обоих свойств, однако требуется наличие предикатов на все проиндексированные свойства для использования индекса.

**MATCH (n:Person)**

**USING INDEX n:Person(name)**

**WHERE n.name = $value**

Использование индекса может быть принудительным в том случае, когда Cypher использует неоптимальный индекс или если необходимо использовать более одного индекса.

**DROP INDEX index\_name**

Удаление индекса с именем index\_name.

**Ограничения целостности**

**CREATE CONSTRAINT ON (p:Person)**

**ASSERT p.name IS UNIQUE**

В результате выполнения операции будет задано ограничение целостности **UNIQUE** (т.е. значение свойства должно быть уникально) для метки «**Person**» и со свойством **name**. Если какой-либо другой узел с этой меткой обновлен или создан с существующим значением свойства **name**, операция записи завершится ошибкой. Это ограничение создаст сопутствующий индекс.

**CREATE CONSTRAINT uniqueness ON (p:Person)**

**ASSERT p.age IS UNIQUE**

В результате выполнения операции будет задано ограничение целостности **UNIQUE** с именем **uniqueness** для метки «**Person**» и со свойством **name**. Если какой-либо другой узел с этой меткой обновлен или создан с существующим значением свойства **name**, операция записи завершится ошибкой. Это ограничение создаст сопутствующий индекс.

**CREATE CONSTRAINT ON (p:Person)**

**ASSERT exists(p.name)**

(★) В результате выполнения операции будет задано ограничение целостности для метки **Person** и свойства **name**. Если узел с этой меткой создается без **name** или если свойство **name** удаляется из существующего узла с меткой **Person**, то операция записи завершится ошибкой.

(★) – здесь и далее таким символом помечены функции доступные в enterprise – версии и недоступные в community – версии.

**CREATE CONSTRAINT node\_exists ON (p:Person)**

**ASSERT exists(p.surname)**

(★) В результате выполнения операции будет задано ограничение целостности **node\_exists** для метки **Person** и свойства **surname**. Если узел с этой меткой создается без **name** или если свойство **name** удаляется из существующего узла с меткой **Person**, то операция записи завершится ошибкой.

**CREATE CONSTRAINT ON ()-[l:LIKED]-()**

**ASSERT exists(l.when)**

(★) В результате выполнения операции будет задано ограничение целостности на отношение типа **LIKED** и свойством **when**. Если связь с этим типом создается без указания **when** или если свойство **when** удаляется из существующей связи с типом **LIKED**, операция записи завершится ошибкой.

**CREATE CONSTRAINT relationship\_exists ON ()-[l:LIKED]-()**

**ASSERT exists(l.since)**

(★) В результате выполнения операции будет задано ограничение целостности **relationship\_exists** на отношение типа **LIKED** со свойством **since**. Если связь с этим типом создается без свойства **since** или если **since** удаляется из существующей связи с типом **LIKED**, операция записи завершится ошибкой.

**CREATE CONSTRAINT ON (p:Person)**

**ASSERT (p.firstname, p.surname) IS NODE KEY**

(★) В результате выполнения операции будет задано ограничение целостности для метки **Person** со свойствами **firstname** и **surname**. Если узел с этой меткой создается без **firstname** и **surname** или если их комбинация не уникальна, или если **firstname** и/или **surname** на существующем узле с меткой **Person** изменяются с нарушением этих ограничений, то будет сформулировано сообщение об ошибке.

**CREATE CONSTRAINT node\_key ON (p:Person)**

**ASSERT (p.name, p.surname) IS NODE KEY**

(★) В результате выполнения операции будет задано ограничение целостности **node\_key** для метки **Person** со свойствами **firstname** и **surname**. Если узел с этой меткой создается без **firstname** и **surname** или если их комбинация не уникальна, или если **firstname** и/или **surname** на существующем узле с меткой **Person** изменяются с нарушением этих ограничений, то будет сформулировано сообщение об ошибке.

**DROP CONSTRAINT uniqueness**

Удалит ограничение целостности с именем **uniqueness**.

**Управление базами данных**

**CREATE OR REPLACE DATABASE myDatabase**

(★) Создает базу данных с именем **myDatabase**. Если база данных с таким именем существует, то существующая база данных удаляется и создается новая.

**STOP DATABASE myDatabase**

(★) Остановить базу данных **myDatabase**.

**START DATABASE myDatabase**

(★) Запустить базу данных **myDatabase**.

**SHOW DATABASES**

Выводит список всех баз данных в системе и информацию о них.

**SHOW DATABASE myDatabase**

Вывести информацию о базе данных **myDatabase**.

**SHOW DEFAULT DATABASE**

Список информации о базе данных по умолчанию.

**DROP DATABASE myDatabase IF EXISTS**

(★) Удалите базу данных **myDatabase**, если она существует.

**Управления пользователями**

**CREATE USER alice SET PASSWORD $password**

Создайте нового пользователя (**alice**) и пароль. Этот пароль необходимо изменить при первом входе в систему.

**ALTER USER alice SET PASSWORD $password**

**CHANGE NOT REQUIRED**

Установите новый пароль для пользователя **alice**. Этот пользователь не должен будет менять этот пароль при следующем входе в систему.

**ALTER USER alice SET STATUS SUSPENDED**

(★) Измените статус пользователя на приостановленный. Используйте **SET STATUS ACTIVE** для возобновления пользователя.

**ALTER CURRENT USER SET PASSWORD FROM $old TO $new**

Изменяет пароль для вошедшего в систему текущего пользователя. Пользователь не должен будет менять этот пароль при следующем входе в систему.

**SHOW USERS**

Перечислите всех пользователей в системе, их статус, роли и необходимость изменения пароля.

(★) Статус и роли доступны только для Enterprise Edition.

**DROP USER alice**

Удалить пользователя **alice**.