рафовые базы данных

|  |
| --- |
| Введение и основные термины:  **NoSQL –**ряд подходов, направленных на реализацию хранилищ баз данных, имеющих значительные отличия от моделей, используемых в реляционных СУБД, где в качестве средства допуска данных используется язык SQL.  **Graph** **Database** — разновидность баз данных с реализацией сетевой модели в виде графа использующая семантические запросы для работы.  [http://2014.ucoz.org/_pu/0/s94106255.jpg](http://2014.ucoz.org/_pu/0/94106255.png)  **Граф –** абстрактное представление множества объектов, где пары объектов соединены между собой.  **Узел** (node) — объект базы данных, представляющий собой ячейку с хранимой информацией.  **Метка** (node label) — условное обозначение типа данных узла. Например, узлы типа «кинофильм» могут быть связаны с узлами типа «актер». Метки регистрирозависимы.  **Ребро** (связь) — связь между двумя узлами. Количество связей ограничено.  **Тип связи**. Максимальное количество типов связей ограничено.  **Свойства узла** — набор данных, которые можно назначить узлу. Например, если узел обозначает фирму, то он может хранить название полное наименование фирмы.  Так же у узлов может быть **уникальный идентификатор**. По умолчанию используется при выводе результатов запроса.    Графовые базы часто используются в качестве базы данных социальных сетей или сайта интернет магазина. Например, так удобно будет узнать по списку знакомых в профиле и вывести список друзей задав в запросе вывести все узлы, с которыми имеют связи узлы имеющие связи с тем узлом, который обозначает нашу учетную запись.     * Плюсы графовых баз данных:   + Простая и понятная модель данных   + Нет необходимости составлять схему данных   + Простой и понятный язык запросов, а также быстрое получение результатов по сравнению с реляционными базами данных. * Минусы:   + Нет необходимости составлять схему данных   + База данных занимает много места на диске по сравнению с реляционными базами данных.   + Специфичная модель представления данных – подходит не для каждой предметной области       Модель данных Graph Database  [http://2014.ucoz.org/_pu/0/s02432563.jpg](http://2014.ucoz.org/_pu/0/02432563.png)  Модель данных состоит из узлов и отношений (ребер) между ними. Ниже представлены виды графов, на основе которых создается та или иная модель данных графовой базы данных.  Виды графов:  http://2014.ucoz.org/_pu/0/63544063.gif  Неориентированный граф (Undirected Graph) – граф в котором ребра не имеют направления (ориентации). Ребро (а,б) совпадает с ребром (б,а).    http://2014.ucoz.org/_pu/0/98570905.gif  Ориентированный граф (Directed Graph). Орграф представлен упорядоченным парами объектов где связь идет в определенно направлении.    http://2014.ucoz.org/_pu/0/14114074.gif  Псевдограф (Pseudo Graph). Представляет собой граф с петлями.    http://2014.ucoz.org/_pu/0/92526825.gif  Мультиграф (Multi Graph). Граф, в котором присутствуют кратные ребра. Кратные ребра – ребра, имеющие одинаковые конечные и начальные вершины. Соединять два узла двумя вершинами может быть нужна в системах социальных сетей, где в списках пользователей друг может являться как другом, так и отцом.    http://2014.ucoz.org/_pu/0/96372467.gif  Гипер граф (Hyper Graph). Один узел может иметь связь с двумя и более узлами. Количество таких связь опять же ограниченно.    Основные отличия Graph Database от реляционных баз данных  В отличии от традиционных (реляционных баз данных). Графовые базы данных позволяют создавать базы с огромным количеством связей между различными элементами. Конечно же такие базы данных ориентированы под конкретные предметные области, в основном — социальные сети.  Простым примером такой базы данных может быть база данных каталога интернет магазина, в данном случае это будет удобно потому, что мы не знаем точное количество характеристик свойств у товаров, кроме того у всех товаров количество свойство разное, а у каждого свойства могут быть подсвойства и так далее. Так же преимуществует здесь будет скорость запросов ведь при загрузке данных из реляционных таблиц мы курсором проходим по всей таблице данных, чтобы найти нужные нам данные, а в графовых базах данных мы идем по ребрам от узла к узлу таким образом скорость выполнения запросов и выдача результатов ускоряется.  Основные отличия:   * Способ представления информации;   + Графовые базы данных – графы;   + Реляционные базы данных – таблицы; * Скорость обработки запросов и выдача результатов. Это одновременно плюс и минус графовых баз данных. При емких запросах, когда нужно получить большое количество информации графовые обрабатывают запрос и выдают результаты быстрее реляционных баз данных, но при очень простых запросах графовые базы данных проиграют в скорости. * Занимаемое место на диске. Графовые базы данных занимают куда больше места на жестком диске, чем реляционные базы данных, но основное их преимущества — это скорость работы, поэтому место на диске будет не главным. Например, реляционная база данных занимает 500 мегабайт, после экспорта в графовую базу данных ее объем может достигать 2 гигабайт. * Другой язык запросов.     Небольшой пример[[1]](http://u.to/6x7_DA" \o "https://d.docs.live.net/c095266a254c4818/%D0%A3%D1%87%D0%B5%D0%B1%D0%B0/%D0%9C%D0%B0%D0%B3%D0%B8%D1%81%D1%82%D1%80%D0%B0%D1%82%D1%83%D1%80%D0%B0%20%D1%83%D1%87%D0%B5%D0%B1%D0%B0/%D0%90%D1%80%D1%85%D0%B8%D1%82%D0%B5%D0%BA%D1%82%D1%83%D1%80%D0%B0%20%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D1%8B%D1%85%20%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC/%D0%94%D0%BE%D0%BA%D0%BB%D0%B0%D0%B4%20%D0%BF%D0%BE%20%D0%B3%D1%80%D0%B0%D1%84%D0%BE%D0%B2%D1%8B%D0%BC%20%D0%91%D0%94/%D0%A0%D0%B5%D1%84%D0%B5%D1%80%D0%B0%D1%82%20Graph%20database.docx#_ftn1) демонстрирующий разницу между графовыми и реляционными базами данных.  http://2014.ucoz.org/_pu/0/95512336.png  Допустим в нашей реляционной базе данных используется связь многие-ко-многим в классическом варианте необходимо будет создавать третью (связующую) таблицу, в которой строки будут представлять пары ключей из обоих таблиц. Таким образом наши запросы к подобным базам данных будут содержать в себе вложенные select или join, таким образом время выполнения запроса будет увеличиться с каждым дополнительным select и join.  К этой ситуации можно привести следующий пример: реляционная база данных социальной сети, мы хотим получить общий список друзей тех людей, которые находятся у нас в друзьях. Несомненно, такой запрос может обрабатываться долго в реляционной базе данных.  http://2014.ucoz.org/_pu/0/54204145.png  На этом рисунке представлен фрагмент ранее продемонстрированной реляционных базы данных в виде графовой базы данных. Тут мы сразу видим, что у нас больше нет связующих таблиц, поэтому данных из обоих таблиц соединяются сразу напрямую через направление ребра, соответственно, при запросе, если мы ходим узнать к каким департаментам относится Алиса, то запрос пройдет намного быстрее.    Отличия Graph Database от других NoSQL баз данных  NoSQL Key-Value сама по себе простая, и не сложная для реализации, но не эффективна, если вы заинтересованы только в запросе или обновлении части данных. Так же трудно реализовать сложные структуры поверх распределенных систем. Используются, например, для хранения изображений, в качестве небольшого кэша, состоящего из объектов.  BigTable. Данные хранятся в виде разряженной матрицы. Строки и столбцы данной матрице используются как ключи. Один из способов применения данной СУБД – веб-индексирование, именно для этой задачи Google и создала BigTable.  Документо-ориентированные базы данных – это базы данных специально предназначения для хранения иерархических структур данных. Основное предназначение – хранилища документов, имеющие структуру дерева.  Графовые базы данных обладают поистине уникальной моделью данных, храня объекты и связи в качестве узлов и связей между ними. Запросы, используемые для таких баз данных способны находить информацию в несколько раз быстрее, чем другие NoSQL базы данных. Однако, эта скорость показывает себя только при обработки большого объема информации одним запросом.    Примеры на основе СУБД Neo4j  **Neo4j** – графовая база данных с открытым исходным кодом. Разработана в Neo Technologies в 2003 году. Разработана на Java.  Данный продукт не нуждается в сложной установке. При загрузке с сайта архива вы получаете папку с уже готовым продуктов, который можно сразу запустить и приступить к работе.           Возможности Neo4j:   * Поддержка ACID[[2]](http://u.to/6B7_DA" \o "https://d.docs.live.net/c095266a254c4818/%D0%A3%D1%87%D0%B5%D0%B1%D0%B0/%D0%9C%D0%B0%D0%B3%D0%B8%D1%81%D1%82%D1%80%D0%B0%D1%82%D1%83%D1%80%D0%B0%20%D1%83%D1%87%D0%B5%D0%B1%D0%B0/%D0%90%D1%80%D1%85%D0%B8%D1%82%D0%B5%D0%BA%D1%82%D1%83%D1%80%D0%B0%20%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D1%8B%D1%85%20%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC/%D0%94%D0%BE%D0%BA%D0%BB%D0%B0%D0%B4%20%D0%BF%D0%BE%20%D0%B3%D1%80%D0%B0%D1%84%D0%BE%D0%B2%D1%8B%D0%BC%20%D0%91%D0%94/%D0%A0%D0%B5%D1%84%D0%B5%D1%80%D0%B0%D1%82%20Graph%20database.docx#_ftn2) * Соответствие JTA[[3]](http://u.to/5R7_DA" \o "https://d.docs.live.net/c095266a254c4818/%D0%A3%D1%87%D0%B5%D0%B1%D0%B0/%D0%9C%D0%B0%D0%B3%D0%B8%D1%81%D1%82%D1%80%D0%B0%D1%82%D1%83%D1%80%D0%B0%20%D1%83%D1%87%D0%B5%D0%B1%D0%B0/%D0%90%D1%80%D1%85%D0%B8%D1%82%D0%B5%D0%BA%D1%82%D1%83%D1%80%D0%B0%20%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D1%8B%D1%85%20%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC/%D0%94%D0%BE%D0%BA%D0%BB%D0%B0%D0%B4%20%D0%BF%D0%BE%20%D0%B3%D1%80%D0%B0%D1%84%D0%BE%D0%B2%D1%8B%D0%BC%20%D0%91%D0%94/%D0%A0%D0%B5%D1%84%D0%B5%D1%80%D0%B0%D1%82%20Graph%20database.docx#_ftn3), JTS, XA[[4]](http://u.to/7x7_DA" \o "https://d.docs.live.net/c095266a254c4818/%D0%A3%D1%87%D0%B5%D0%B1%D0%B0/%D0%9C%D0%B0%D0%B3%D0%B8%D1%81%D1%82%D1%80%D0%B0%D1%82%D1%83%D1%80%D0%B0%20%D1%83%D1%87%D0%B5%D0%B1%D0%B0/%D0%90%D1%80%D1%85%D0%B8%D1%82%D0%B5%D0%BA%D1%82%D1%83%D1%80%D0%B0%20%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D1%8B%D1%85%20%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC/%D0%94%D0%BE%D0%BA%D0%BB%D0%B0%D0%B4%20%D0%BF%D0%BE%20%D0%B3%D1%80%D0%B0%D1%84%D0%BE%D0%B2%D1%8B%D0%BC%20%D0%91%D0%94/%D0%A0%D0%B5%D1%84%D0%B5%D1%80%D0%B0%D1%82%20Graph%20database.docx#_ftn4) * Высокая доступности * Возможность создать базу данных с более чем миллиардом узлов и связей.   + 32 миллиарда узлов   + 32 миллиарда связей   + 64 миллиардов свойств * Высокая скорость выполнения запросов при обходе базы данных * Простой и понятный язык запросов (Cypher)[[5]](http://u.to/7R7_DA" \o "https://d.docs.live.net/c095266a254c4818/%D0%A3%D1%87%D0%B5%D0%B1%D0%B0/%D0%9C%D0%B0%D0%B3%D0%B8%D1%81%D1%82%D1%80%D0%B0%D1%82%D1%83%D1%80%D0%B0%20%D1%83%D1%87%D0%B5%D0%B1%D0%B0/%D0%90%D1%80%D1%85%D0%B8%D1%82%D0%B5%D0%BA%D1%82%D1%83%D1%80%D0%B0%20%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D1%8B%D1%85%20%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC/%D0%94%D0%BE%D0%BA%D0%BB%D0%B0%D0%B4%20%D0%BF%D0%BE%20%D0%B3%D1%80%D0%B0%D1%84%D0%BE%D0%B2%D1%8B%D0%BC%20%D0%91%D0%94/%D0%A0%D0%B5%D1%84%D0%B5%D1%80%D0%B0%D1%82%20Graph%20database.docx#_ftn5).       **Примеры запросов на языке (Cypher).**   * *CREATE (n: Person {name: 'Aleksey', type: 'Student’})*   + Данный запрос создаст узел с меткой «Person» содержащий в себе свойства: «Name» и «Student» * Если мы ходим создать узел с несколькими метками, то перечисляем их через « : », например, *CREATE (n:Person:Student … })*. Таким образом можно некоторые свойства использовать как метки тем самым сэкономив память. * Чтобы создать несколько узлов то просто перечисляем из через запятые *CREATE* *a, b, c, d, e* * Связи между узлами можно создать двумя способами:   + Создать связь при создании узлов *CREATE (a*: *Person {name: 'Aleksey'})-[b:knows]-> (c*: *Person {name: 'Nastya'})*   + Создать связь между уже существующими узлами     - *MATCH (a :Person { name: 'Aleksey'}), (c: Person { name: 'Nastya'}) MERGE (a)-[b:knows]->(c)*     Теперь коротко об основных операторах, которые используются в запросах:   * CREATE – используется для создания узлов и отношений. * MERGE – данный оператор используется либо уже имеющиеся данные в базе данных, либо создает их при отсутствии   + Пример: *MERGE (n:Person { name:'Aleksey'}) RETURN* *n –*данный запрос либо вернет уже имеющейся узел в базе данных, либо создаст новый узел. * MATCH – в отличии от MERGE просто возвращает узел по заданным характеристикам * SET – используется для обновления узлов или отношений.   + Например: MATCH (n { firstname: 'Aleksey' }) SET n.lastname = 'Frolov' RETURN n * DELETE – используется для удаления узлов, отношений или целых путей. * REMOVE – используется для удаления свойств или меток из узлов. * FOREACH – используется для обновления данных в пределах выбранного пути.   + Например: *MATCH* *p=(begin)-[\*]->(END) WHERE* *begin.name='A' AND* *END .name='D' FOREACH (n* *IN* *nodes(p)| SETn.marked = TRUE) –*установит на всей узлах в пределах заданных условий свойство *«marked»* содержание *«true»* * CREATE UNIQUE – некоторый аналог «MERGE», только если «MERGE» выбирает или создает, то этот оператор создает то чего нет или не создает, если это уже есть.   Кроме этих операторов «Neo4j» содержит в себе операторы для операций чтения и некоторые общие операторы, которые не были использованы в примерах выше. Со всеми этими операторами можно ознакомиться в методичке с официального сайта[[6]](http://u.to/9R7_DA" \o "https://d.docs.live.net/c095266a254c4818/%D0%A3%D1%87%D0%B5%D0%B1%D0%B0/%D0%9C%D0%B0%D0%B3%D0%B8%D1%81%D1%82%D1%80%D0%B0%D1%82%D1%83%D1%80%D0%B0%20%D1%83%D1%87%D0%B5%D0%B1%D0%B0/%D0%90%D1%80%D1%85%D0%B8%D1%82%D0%B5%D0%BA%D1%82%D1%83%D1%80%D0%B0%20%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D1%8B%D1%85%20%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC/%D0%94%D0%BE%D0%BA%D0%BB%D0%B0%D0%B4%20%D0%BF%D0%BE%20%D0%B3%D1%80%D0%B0%D1%84%D0%BE%D0%B2%D1%8B%D0%BC%20%D0%91%D0%94/%D0%A0%D0%B5%D1%84%D0%B5%D1%80%D0%B0%D1%82%20Graph%20database.docx#_ftn6). Методичка очень хорошо разделана по разделам и содержит в себе вполне понятные примеры, методичка будет понятна даже для людей со слабыми знаниями английского языка (методичка на английском языке). У меня при изучении методички претензий к ее содержанию не возникло.    В рамках изучения Neo4j я создал небольшую базу данных содержащую информацию о людях и городах, но в большей степени о людях.  http://2014.ucoz.org/_pu/0/65100541.png  «Neo4j» может очень хорошо графически отображать результаты запросов, если результат запроса может быть сопоставим с графическим отображением, например, если вы удаляете узлы из базы при помощи «DELETE», то «return» вернет просто информацию о количестве удаленных узлов и о времени, затраченном на выполнение запроса.  Все узлы и связи в моем примере создавались вручную и это дало мне возможность оценить время, затрачиваемое на выполнение таких простых запросов. Например, запросы на создание связи между двумя узлами у меня в последний 5-7 запросах на создание связей занимали около 700 миллисекунд, а первые запросы занимали не более 100. В этом один из минусов данных баз данных время на выполнение коротких запросов превышает время на выполнение подобных запросов в реляционных базах данных.  Теперь непосредственно к запросам на получение данных из базы данных.    http://2014.ucoz.org/_pu/0/69437850.png    Данный запрос выводит нам список людей, которые проживают в Москве. Это запрос выполнялся 94 миллисекунд.  Запрос ниже выводит туже самую информацию, но при этом выводит еще и город, с которым был связан наш запрос, данный запрос выполнялся 49 миллисекунд.    http://2014.ucoz.org/_pu/0/97809612.png    Следующей запрос демонстрирует запрос на нахождение короткого пути между двумя узлами  http://2014.ucoz.org/_pu/0/31883901.png  Помимо нахождения короткого пути в этом запросе еще установлено условие на тип связи между двумя узлами, без него результатом был бы самый возможный короткий маршрут, который представлен ниже.  http://2014.ucoz.org/_pu/0/57175476.png  То есть при работе с базой данных из приложения необходимо правильно составлять запросы, чтобы пользователю случайно не попались неправильные данные.  Есть замечание к запросам на поиск маршрутов, которое делают сами разработчики «Neo4j». Без серьезной необходимости не делать запросы на нахождение всевозможных путей между узлами, пример запроса: *match* *n=(begin)-[\*]-end* *wherebegin.Name=’Aleksey’ and* *end.Name=’Misha’ return* *n* – запросы такого перегрузят систему, использовав все ресурсы компьютера по максимуму. Я применил данный запрос ко своей небольшой базе данных и результат не был получен даже в течении часа. Я решил немного облегчить запрос добавив условие на тип связи между узлами и выбрал только связи «knows» получился следующий запрос:  http://2014.ucoz.org/_pu/0/80742606.png  Запрос выполнялся 239 миллисекунд.      Выводы  Графовые базы данных являются однозначно одними из самый быстрых по скорости работы с большими объемами данных, кроме того они отлично находят применение на сайтах типа «интернет-магазин», например, информацию сайта автозапчастей можно очень детально разбить, ведь, каждая деталь автомобиля состоит из других деталей и так до самых винтиков, поэтому, если автовладельцу до мелочей интересно узнать про содержимое своего автомобиля использование графовой базы данных на этом сайте даст ему такую возможность. Несомненно, что для такого сайта можно использовать и реляционную базу данных, но скорость выдачи результатов будет значительно ниже, а найти такого человека, который по 10-30 секунд будет ждать загрузки страницы с информацией тяжело. |