**Графовые базы данных**

Графовые базы данных эффективны для реализации баз с простыми записями и сложными отношениями между объектами. Имеют стандартную модель, в которой записи хранятся в виде узлов, соединенных ребрами (дугами), которые обозначают связи между записями.

Производительность при поиске связей возрастает по сравнению с реляционными моделями, так как в отличие от последних, где связь требует времени обработки в момент запроса, здесь основная обработка переносится на момент создания связи (персистентные связи).

Легко производится добавление новых типов связей в базу данных, так как как таковой схемы базы данных нет.

Обычно реализуют ACID-транзации, за счет которых обеспечивается согласованность. В частности, при создании связи транзакции не допускают отсутствия начального или конечного узлов, узлы могут быть удалены только в том случае, если у него нет никаких связей.

Производительность в основном реализуется за счет репликации, например, Neo4J реализует репликацию типа «ведущий-ведомый» с возможностью изменения на «ведомых» узлах (сразу подается сигнал на «ведущий» узел и он уже управляет последующей репликацией изменений на остальные «ведомые» узлы). Фрагментация производится редко, обычно в случаях, когда граф имеет несколько слабо связанных между собой компонентов.

Типовые приложения использования графовых баз данных:

1. Связанные данные (социальные сети).

2. Сервисы маршрутизации, диспетчеризации и геолокации.

3. Справочные базы данных.

Главный ограничивающий фактов при выборе данной модели – частота обновления данных для узлов графа: если данные узлов графа часто требуют обновления, графовые базы данных нецелесообразно использовать, так как поиск требуется будет проводить по всему графу.

Для примера разберем основные принципы работы с графовой СУБД **Neo4J**.

1. Имеет браузерного клиента, который работает на основе Java Script и предоставляет возможности выполнения запросов на специальном языке Cypher и просмотра результатов в табличном и графическом виде.
2. Команда создания узла:

CREATE (<node-nam e>:<label-nam e>)

CREATE ( <node-nam e>:<label-nam e>

{

<Property1-nam e>:<Property1-Value>

........

<Propertyn-nam e>:<Propertyn-Value>

})

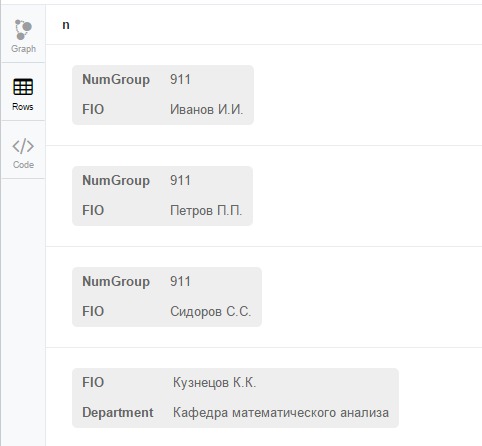
Например, создадим узлы студента, преподавателя:

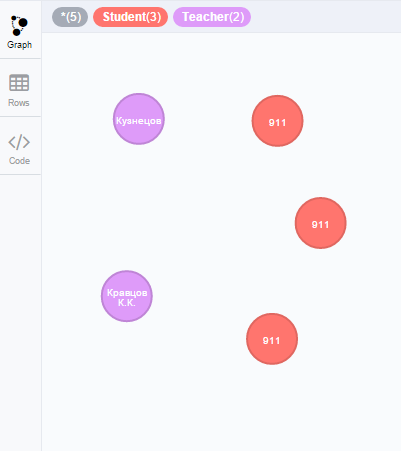
create (stud:Student{FIO:"Иванов И.И.", NumGroup:"911"})

create (tch:Teacher{FIO:"Кузнецов К.К.", Department:"Кафедра математического анализа"})

и т.д.

В результате получим:





1. Команда поиска узлов:

MATCH

(

<node-nam e>:<label-nam e>

)

Команда должна завершаться указанием, что сделать с найденными узлами, например, показать их (вернуть как результат запроса):

match (t:Teacher) return t

или показать только конкретные поля узлов:

match (t:Teacher) return t.FIO

или с учетом условия:

match (t:Teacher) where t.Department="Кафедра программирования" return t.FIO

В качестве указания, что делать с найденными узлами, может стоять DELETE, приводящая к удалению узлов, или команда SET, с помощью которой для найденных узлов возможно произвести изменение каких-то свойств.

1. Команда создания связи.

Существует несколько сценариев создания связи:

* Создание связи между существующими узлами без свойств связей:

MATCH (<node1-label-nam e>:<node1-nam e>),(<node2-label-nam e>:<node2-nam e>)

CREATE

(<node1-label-nam e>)-[<relationship-label-nam e>:<relationship-nam e>]->(<node2-label-nam e>)

RETURN <relationship-label-nam e>

Например, создание связи между узлами «Студент» с названием «Староста»:

match (s1:Student), (s2:Student) where s1.FIO="Иванов И.И." and s2.FIO="Петров П.П." create (s1)-[r:Leader]->(s2)

* Создание связи с участием нового узла. Если одного или обоих узлов не существует, то вместо MATCH и указания метод для узлов, которые надо связать можно указать свойства нового узла:

CREATE

(<node1-label-nam e>:<node1-nam e>)-

[<relationship-label-nam e>:<relationship-nam e>]->

(<node1-label-nam e>:<node1-nam e>)

* Создание связи между существующими узлами со свойствами связи:

MATCH (<node1-label-nam e>:<node1-nam e>),(<node2-label-nam e>:<node2-nam e>)

CREATE

(<node1-label-nam e>)-[<relationship-label-nam e>:<relationship-nam e>

{<define-properties-list>}]->(<node2-label-nam e>)

RETURN <relationship-label-nam e>

* Создание связи с участием нового узла со свойствами связи.

Например, создадим связь между узлом «Студент» и новым узлом «Дисциплина» и свойствами, обозначающими дату сдачи экзамена и оценку.

match (s1:Student) where s1.FIO="Иванов И.И."

CREATE (s1)-[r:Study{date\_exam:"06.01.2016",mark:5}]->(s2:Subject{title:"Математический анализ"})

Имеются функции:

* Строковые (UPPER, LOWER, SUBSTRING, REPLACE)
* Агрегирующие (AVG, COUNT, MAX, MIN, SUM)
* Функции связи (STARTNODE, ENDNODE, ID, TYPE).

Обращение к Neo4J из клиентских приложений можно осуществить с помощью стандартного подхода (созданного аналогично SQL), основанного на применении провайдеров и драйверов. Приведем пример для языка программирования Java. Для этого требуется библиотечный архив

neo4j-jdbc-2.3.2-jar-with-dependencies.jar

**import** java.sql.\*;  
  
**public class** probaNeo {  
 **public static void** main(String [] args)  
 {  
 **try** {  
 Connection con= DriverManager.*getConnection*(**"jdbc:neo4j://localhost:7474/"**,**"neo4j"**,**" neo4j "**);  
 String q=**"match (s1:Student),(s2:Subject) where s1.FIO=\"Петров П.П.\"\n”+**

**"and s2.title=\"Математический анализ\"\n"** +  
 **"CREATE (s1)-[r:Study{date\_exam:\"06.01.2016\",mark:4}]->(s2)"**;  
 Statement com=con.createStatement();  
 com.executeUpdate(q);  
 q=**"MATCH (s1:Student),(s2:Subject),(s1)-[r:Study]->(s2) where \n"+**

**"s2.title=\"Математический анализ\" and r.mark>2 \n"+**

**"return s1.FIO as fio, r.mark as mark"**;  
 com=con.createStatement();  
 ResultSet rs=com.executeQuery(q);  
 **while**(rs.next())  
 System.***out***.println(rs.getString(**"fio"**)+**" "**+rs.getInt(**"mark"**));  
 con.close();  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
 }  
}

Для работы с .NET технологиями драйвер подключения к Neo4J подгружается через инструмент NuGet:

PM> Install-Package Neo4j.Driver -Pre

Фактически он представляет собой средства выполнения из клиентского приложения команд сервера Neo4j.

using System;

using System.Collections.Generic;

using Neo4j.Driver.V1;

namespace console1

{

class Program

{

public static void Main(string[] args)

{

Console.WriteLine("Hello World!");

using (var driver = GraphDatabase.Driver("bolt://localhost:7474", AuthTokens.Basic("neo4j", "root")))

using (var session = driver.Session())

{

session.Run("CREATE (a:Person {name: {name}, title: {title}})",

new Dictionary<string, object> { {"name", "Arthur"}, {"title", "King"} });

var result = session.Run("MATCH (a:Person) WHERE a.name = {name} " +

"RETURN a.name AS name, a.title AS title",

new Dictionary<string, object> { {"name", "Arthur"} });

foreach (var record in result)

{

Console.WriteLine(""+record["title"].As<string>()+" "+record["name"].As<string>());

}

}

Console.Write("Press any key to continue . . . ");

Console.ReadKey(true);

}

}

}