Федеральное государственное бюджетное образовательное учреждение высшего образования
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ МОРДОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМ. Н. П. ОГАРЁВА»

Факультет математики и информационных технологий

Кафедра фундаментальной информатики

УТВЕРЖДАЮ

Зав. кафедрой
канд. физ.-мат. наук, доцент
_____ А. Г. Смольянов
(подпись)
«20» июня 2016 г.

БАКАЛАВРСКАЯ РАБОТА

ПРАКТИЧЕСКОЕ ПРИМЕНЕНИЕ NoSQL БАЗ ДАННЫХ

Автор бакалаврской работы (подпись) 27.05.2016 (дата) К. В. Чиняев

Обозначение бакалаврской работы БР–02069964–02.03.02–16–16

Направление 02.03.02 Фундаментальная информатика и информационные технологии

Руководитель работы
канд. физ.-мат. наук, доцент (подпись) 17.06.2016 (дата) А. В. Попов

Нормоконтролер
канд. пед. наук, доцент (подпись) 17.06.2016 (дата) И. А. Карпюк

Рецензент
канд. физ.-мат. наук, доцент (подпись) 17.06.2016 (дата) Л. А. Сухарев

Саранск
2016

Федеральное государственное бюджетное образовательное учреждение высшего образования
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ МОРДОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМ. Н. П. ОГАРЁВА»

Факультет математики и информационных технологий

Кафедра фундаментальной информатики

Зав. кафедрой
канд. физ.-мат. наук, доцент
_____ А. Г. Смольянов
(подпись)
«20» июня 2016 г.

ЗАДАНИЕ НА ВЫПУСНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ

Студент Чиняев Константин Валерьевич

1 Тема Практическое применение NoSQL баз данных

Утверждена приказом № 934-С от 12.02.2016 г.

2 Срок представления работы к защите 27.05.2016

3 Исходные данные для научного исследования: литература по теме исследования

4 Содержание выпускной квалификационной работы

4.1 Эволюция баз данных

4.2 NoSQL базы данных

4.3 Немного о графовых базах данных. Neo4j

5 Приложения Код на языках Cypher и C/C++

Руководитель работы _____ 12.02.2016 _____ А. В. Попов
подпись, дата

Задание принял к исполнению _____ 12.02.2016 _____ К. В. Чиняев
подпись, дата

РЕФЕРАТ

Бакалаврская работа содержит 77 страниц, 27 рисунков, 4 таблицы, 15 использованных источников, 2 приложения.

NOSQL, НЕРЕЛЯЦИОННЫЕ БАЗЫ ДАННЫХ, РЕЛЯЦИОННЫЕ БАЗЫ ДАННЫХ, SQL, БАЗЫ ДАННЫХ НА ГРАФАХ, NEO4J, CYPHER, ЭВОЛЮЦИЯ БАЗ ДАННЫХ, МОДЕЛИ ДАННЫХ.

Объектом исследования являются NoSQL базы данных и их применение на практике, в частности базы данных на графах и СУБД Neo4j.

Цель работы – провести сравнительный анализ моделей данных и попытаться определить критерии оптимальности при выборе СУБД для разрабатываемых приложений.

Задачи:

– Изучить эволюцию баз данных;
– Рассмотреть различные модели NoSQL баз данных, сравнить их с традиционной реляционной моделью;
– Определить границы применимости различных моделей;
– Изучить графовую модель данных на примере СУБД Neo4j и рассмотреть практические примеры ее применения.

Степень внедрения – частичная.

Область применения – в практической деятельности разработчика приложений для работы с базами данных.

Эффективность – от правильного выбора при создании базы данных модели данных зависят многие метрики приложений, которые с ней работают: скорость обработки запросов, сохранность данных, отказоустойчивость и т. д. Поэтому для корректной и эффективной работы проекта важно оптимально выбрать хранилище данных и СУБД определенной архитектуры.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ 6
1 Эволюция баз данных 8
1.1 От первых ЭВМ и информационных носителей к способу хранения и доступа к файлам 9
1.2 Базы данных на «плоских» файлах. Инвертированные списки 9
1.3 Иерархические базы данных 12
1.4 Сетевые базы данных 16
1.5 Реляционные базы данных 17
2 NoSQL базы данных 23
2.1 Общие характеристики NoSQL баз данных 25
2.1.1 Неструктурированность (schemaless) 25
2.1.2 Представление данных в виде агрегатов (aggregates) 26
2.1.3 Распределенные системы без совместно используемых ресурсов (share nothing) 28
2.2 Модели согласованности 32
2.3 Теорема CAP 33
2.4 Модели данных и их классификация 33
2.4.1 Системы «ключ-значение» 34
2.4.2 Документные СУБД 35
2.4.3 Системы типа Google BigTable 37
2.4.4 Базы данных на основе графов 38
2.5 Реляционная база данных или NoSQL? Преимущества NoSQL 39
3 Немного о графовых базах данных. Neo4j 42
3.1 Основные элементы графовой базы данных Neo4j 44
3.2 Установка Neo4j 46
3.3 Язык запросов Cypher 48
3.4 В каких случаях база данных Neo4j применима на практике и когда дает преимущества перед реляционной моделью? 56

ЗАКЛЮЧЕНИЕ 60
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 61
ПРИЛОЖЕНИЕ А (обязательное) Код на языке Cypher, добавляющий данные о фильмах, актерах, режиссерах и продюсерах в базу Neo4j 63
ПРИЛОЖЕНИЕ Б (рекомендуемое) Код программы на C/C++, переводящей кириллические символы в латинские 76

ВВЕДЕНИЕ

С самых древних времен люди стремились сохранить важные для них знания. Очевидно, что первым способом хранения данных был язык, сочетающий в себе как звуки, так и жесты. Информация, передаваемая таким способом, не могла существовать без непосредственного общения людей и в любой момент могла быть утеряна навсегда. Начиная с Каменного века и по сегодняшний день человечество постоянно заботилось о «материализации» и сохранении информации. Для кодирования информации были придуманы самые разные алфавиты – наскальные рисунки, узелки, клинья, буквы, цифры, символы, цвета и др. Информация записывалась на камнях, табличках, папирусе, бумаге, хранилось это все в огромных библиотеках. Сейчас 21 век, с тех времен изменилось почти все: носители данных, способы записи и хранения данных, объемы данных достигли небывалых высот, возникла необходимость постоянной, непрерывной работы с различными данными, их обработке и анализе. Изменились и сами хранилища данных.

Именно первые примитивные хранилища данных являются прародителями наших современных баз данных, поэтому вначале мы кратко рассмотрим историю развития баз данных – это позволит лучше понять современные решения для хранения данных. Главной целью моей работы является изучение нереляционных баз данных. Это сравнительно новое направление в хранении данных, появившееся 10-15 лет назад, бурно развивающееся и ставшее очень популярным. Мы сравним новые решения и использующиеся в них модели данных с традиционной реляционной моделью, применяемой уже более 40 лет.

Более подробно будет рассмотрен один тип нереляционных моделей: модель данных на основе графов. Данную модель поддерживает СУБД Neo4j, имеющая встроенный язык запросов Cypher. Проанализировав практические примеры, мы определим, в каких случаях предпочтительнее поль-

зоваться данной СУБД, и какие преимущества при этом имеют NoSQL-решения.

На современном рынке существует большое разнообразие решений для хранения данных, поэтому очень важно иметь представление о них – это поможет сделать правильный выбор в пользу системы, которая будет эффективно справляться с возложенными на нее задачами.

1 Эволюция баз данных

Что такое «база данных»? В литературе предлагается множество формулировок этого понятия, но если не вдаваться в тонкости, то все определения можно свести к следующему: База данных – это «организованное хранилище данных». Несмотря на то, что термин базы данных вошел в наш словарный запас только в конце 1960-х годов, сбор и систематизация данных еще с древних времен являлись неотъемлемыми факторами развития человеческой цивилизации. Книги, особенно со строгой структурой, такие как словари, энциклопедии – все это наборы данных в физической форме. А библиотеки и другие индексированные архивы информации представляют собой доиндустриальные эквиваленты современных систем информации.

Изучая литературу, я встречал несколько описаний истории или эволюции баз данных. Кто-то пытался написать максимально просто и понятно, и начинал рассказ с 60-х годов XX века, с того момента как определение вошло в нашу жизнь и продолжал свою хронологию, повествуя про дальнейшее развитие электронной техники и баз данных. Кто-то пытался систематизировать её как комплекс революций – прорывов в технологиях баз данных. Так же встречалась точка зрения на их историю, как на эволюцию четырех составляющих: 1) носителей информации; 2) методов структурирования и организации данных; 3) архитектур баз данных; 4) программного инструментария для создания и управления базами данных. В англоязычной Википедии развитие технологии баз данных разделяют, ссылаясь на модель данных или структуру, на три эры: эра навигационных баз, реляционных/SQL и постреляционных.

В этой главе я постараюсь, не слишком отдаляясь от темы моей работы, совместить эти взгляды на историю баз данных и выстроить путь через все этапы эволюции до, ставшего традиционным, – реляционного подхода реализации хранилищ баз данных, и рассмотрю его поподробнее.

1.1 От первых ЭВМ и информационных носителей к способу хранения и доступа к файлам

Первые электронные вычислительные машины (ЭВМ) появились после Второй мировой войны, и создавались они тогда еще с целью выполнения математических вычислений. Появление первых ЭВМ – представляло собой первую революцию в базах данных. Мало кто тогда думал, что компьютер будет использоваться так, как мы его используем сейчас. Программы того времени хранили данные для обработки в себе или подгружали их с внешних носителей, которыми в то время являлись перфокарты и перфоленты, позднее магнитные ленты.

Данные постепенно отделялись от программ, с ними работающих, и, в конце концов, обособились в файлы. Но из-за отсутствия единых стандартов по созданию и организации файлов, перенос данных с компьютера одного типа на компьютер другого типа был невозможен. В 1950-х магнитные диски с прямым доступом к данным пришли на смену лентам – это обеспечило прямой и быстрый доступ к файлам, раньше требовалось перематывать ленту, чтобы добраться до нужной записи на ней (метод последовательного доступа). Революцией в развитии способов хранения и структурирования информации в ЭВМ стала разработка компанией IBM в 1963 году технологии индексно-последовательного доступа (ISAM – Indexed Sequential Access Method, англ.) – способа хранения данных для быстрого доступа к файлам. Эта технология нашла применение в мейнфреймах компании и дала дорогу первым электронным базам данных. В настоящее время это основной способ представления данных почти во всех базах данных, так или иначе модифицированный.

1.2 Базы данных на «плоских» файлах. Инвертированные списки

Пусть отделу кадров предприятия необходимо хранить информацию о своих сотрудниках, состоящую из табельного номера сотрудника, фамилии, имени, отчества, даты его рождения и других сведений. Все эти данные легко сгруппировать в виде таблицы, заголовками столбцов которой будут названия перечисленных элементов. Ширина каждого столбца подбирается, в зависимости от самого длинного элемента. Строка таблицы описывает одного конкретного человека.

Физически: таблицей будет являться – *файл,* а строкой таблицы – *запись* файла. Содержимое строки может заноситься в запись двумя способами:

1) в виде простого списка, элементы которого отделяются друг от друга каким-либо специальным символом, например, точкой с запятой (рисунок 1);

2) запись разбивается на *поля* фиксированной длины, равной ширине соответствующих столбцов таблицы; после чего элементы информации заносятся в соответствующие поля (рисунок 1).



Рисунок 1 – Представление данных в виде плоских файлов с разделителями и полями фиксированный длины

Полученный файл не содержит только записи. В нем нет никаких заголовков, нет информации о названиях и длинах полей, о записях, их формате и др. Структура записей файла была «известна» только программе, которая должна была абсолютно точно повторять формат записи. Если формат менялся, то программа должна была корректироваться. Такие файлы стали ис-

торически первым прообразом баз данных и получили название простых или *«плоских» файлов* (flat file, англ.).

Однако было бы слишком большим упрощением считать, что в один прекрасный день обычные структурированные файлы стали называть базами данных. В обычной файловой системе не были решены вопросы многопользовательского доступа к одному файлу, управления транзакциями и др. Поэтому перед первыми СУБД – системами управлениями базами данных ставились не только задачи достижения эффективных структур и механизмов поиска данных, но и задачи управления сложными внутренними процессами, происходящими на физическом уровне.

Несмотря на то, что данное представление устарело, идея плоских файлов эксплуатируется до сих пор, благодаря своему безусловному преимуществу – простоте физической организации (таблица 1).

Таблица 1 – СУБД, основанные на плоских файлах

СУБД	**Производитель**	**Характеристика**
GNU Recutils	Фонд свободно распространяемого ПО (Free Software Foundation)	Запись содержит произвольное количество именованных полей
BDB (Berkeley DB)	University of California, Berkeley. В 2006 г. права на производство перешли к Oracle	Начиная с 1994 г. вышли четыре версии, в основном, для открытых ОС, включая Linux. Первые версии (до перехода в Oracle) не поддерживали SQL
Borland Reflex	Analytica, купленная затем компанией Borland	Разработана в 1980-х гг. для DOS. Один из разработчиков Reflex перешел в Microsoft и основал там проект СУБД ACCESS

Одной из самых сложных задач в работе с плоскими файлами стал поиск и доступ к нужной записи. Наиболее эффективным решением этой задачи являлось и до сих пор является индексирование записей по *первичному ключу.*

Однако в ряде случаев требуется выполнить более гибкий поиск, тогда используется вторичный ключ. *Вторичный ключ* – это поле, одному значению которого соответствуют несколько записей. Поиск по вторичным ключам упрощается при их индексации. Эти индексы получили название *инвертированных списков* (inverted list, англ.). На инвертированных списках и файлах были построены одни из первых промышленных СУБД, получивших в начале 1970-х гг. широкое распространение.

1.3 Иерархические базы данных

Каждая предметная область содержит в себе несколько объектов, о которых нужно хранить информацию. Очевидно, что для каждого из таких объектов может быть создан свой файл. Можно ли совокупность этих файлов считать базой данных? Такой же вопрос можно задать себе, пользуясь бумажной картотекой в библиотеке. Да, безусловно, это база данных, к тому же состоящая из двух файлов. В библиотеках имеются две картотеки: в первой, называемой тематическим каталогом, книги упорядочены по отраслям знаний, а во второй - алфавитном каталоге - по авторам. Но всегда ли удобно работать с такой базой данных?

Пусть, проектируя программу, мы решили найти книгу, где описаны основные алгоритмы поиска и сортировки данных. Тогда нам следует пойти к каталогу «Программирование» и начать поиск нужной книги. Наткнувшись, скажем, на «Искусство программирования» Д. Кнута, вспоминаем о фразе, брошенной преподавателем на лекции, что Кнут – великий автор для тех, кто хочет научиться алгоритмически мыслить. Естественно, книгу нужно брать. А еще лучше посмотреть и другие книги Д. Кнута, вдруг в них есть

более новая или более интересная информация. Идем к авторскому каталогу, находим ящик с буквой «К» и перебираем карточки, добравшись до фамилии Кнут. Однако, если его фамилия указана не первой в списке авторов, то мы можем и не найти его книги, несмотря на то, что соответствующие карточки физически существуют, но лежат в каком-то другом ящике картотеки.

Получается, что даже если у нас есть хорошо структурированная и упорядоченная информация, некоторые проблемы остаются: нам приходится переходить от каталога к каталогу (от файла к файлу), держа в уме фамилию автора, к тому же мы не всегда можем быстро найти нужную карточку (запись). Очевидно, что современное поколение пользователей скажет: «Все просто, нужно установить связи, прописать гиперссылки и готово...». Связь - ключевое слово, отличающее базу данных от простого файла или набора файлов. Но как представить эту связь в базе данных?

Впервые эта задача была решена в конце 1960-х гг. с появлением *иерархических* СУБД, позволявших устанавливать между элементами информации отношение «один-ко-многим». Минимальной структурной единицей в иерархической модели является *поле,* считающееся неделимым элементом данных. Поименованная совокупность полей называется сегментом (рисунок 2).

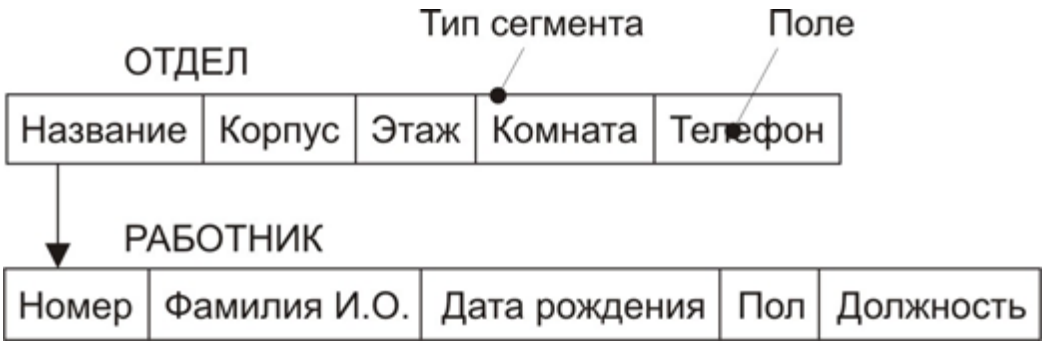


Рисунок 2 – Схема иерархической базы данных

Точнее, в иерархической базе данных различаются два понятия: тип сегмента и экземпляр сегмента. *Экземпляр сегмента* – это некоторая *логиче-*

ская запись, описывающая конкретный объект предметной области, например, конкретного человека – работника предприятия (рисунок 3).

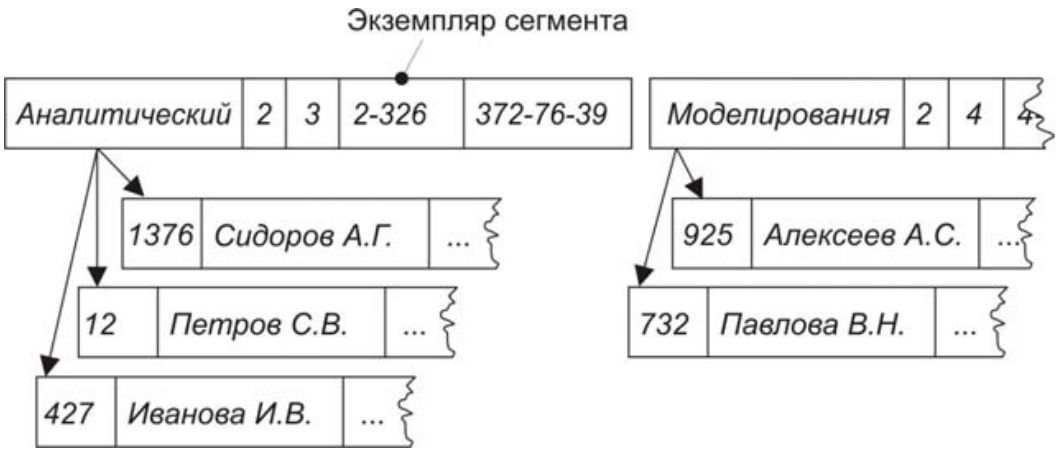


Рисунок 3 – Экземпляры сегментов иерархической базы данных

Структурная схема этой записи, в которой указаны имена составляющих ее полей, называется *типом сегмента* или просто сегментом. Таким образом, в иерархической базе данных впервые были разделены понятия схемы данных и собственно самих данных.

Схема иерархической базы данных представляет собой дерево, вершинами которого являются типы сегментов (рисунок 4).



Рисунок 4 – Пример дерева иерархической базы данных

Между типами сегментов установлены отношения «исходный - порожденный». Тип сегмента, у которого нет предка, называется корневым. На

уровне экземпляров каждое такое отношение реализуется в виде связей между одним экземпляром сегмента-родителя и несколькими экземплярами сегмента-потомка. Таким образом, иерархическая база данных – это совокупность деревьев, корнями которых являются экземпляры корневого типа сегмента. Каждое такое дерево называется *набором* и образует *физическую запись* иерархической базы данных.

В иерархических базах данных используется целое семейство методов доступа к данным: иерархический последовательный (HSAM, Hierarchical Sequential Access Method, англ.), иерархический прямой (HDAM, Hierarchical Direct, англ.), иерархический индексно-последовательный (HISAM, Hierarchical Indexed Sequential англ.) и иерархический индексно-произвольный (Hierarchical Indexed Direct англ.).

Сегодня иерархическая структура используется для хранения географической информации и файловых систем. Они по-прежнему широко используются, особенно в приложениях, требующих очень высокой производительности и доступности, такие как банковское дело и телекоммуникации. Одним из наиболее широко используемых коммерческих иерархических баз данных является IMS (Information Management System, англ.), созданная корпорацией IBM в 1969-1970 гг. (таблица 2). Еще один пример использования иерархических баз данных – реестр в семействе операционных систем Microsoft Windows.

Таблица 2 – Иерархические СУБД

СУБД	Производитель	История и характеристика
1	2	3
IMS	IBM (США)	1968 г. – Первая версия IMS для IBM/360. Главный архитектор – Верн Уоттс. Поддерживает доступ на языке Java, обработку XML, с 2000-х – поддерживаются веб-службы.

Окончание таблицы 2

1	2	3
ИНЕС	ВНИИСИ и ИПУ АН СССР (СССР)	1976 г. – первая версия ИНЕС (информационная единая система) для ЕС ЭВМ. Создана под руководством ученого В. Л. Арлазарова.
ОКА	Институт кибернетики АН УССР (СССР)	Советский аналог IMS

1.4 Сетевые базы данных

Существенную роль в эволюции и стандартизации баз данных сыграла группа CODASYL (Conference on Data Systems Languages – конференция по языкам систем обработки данных, англ.), которая была создана в 1959 г. для разработки стандартов по языкам программирования. В 1965 г. в рамках этой проекта была сформирована рабочая группа по базам данных (DBTG, Data Base Task Group, англ.). Через два года ею была опубликована концепция *сетевой* базы данных, широко известной также под названием «модель данных CODASYL».

Идеологом сетевой модели был Чарлз Бахман (Charles Bachman) – один из выдающихся исследователей и практиков в области компьютерных наук, удостоенный в 1973 г. премии Тьюринга за вклад в развитие баз данных. Архитектура базы данных CODASYL стала прообразом современных архитектур и включала в себя *схему*, отражающую полную логическую структуру данных; *подсхемы* – различные «взгляды» на базу данных с точки зрения разных прикладных программ; описание физической структуры данных, т.е. физическую организацию полей, записей и связей между ними.

Для работы с такой базой данных были предложены два языка: язык определения данных (ЯОД) (DDL, Data Definition Language, англ.), позволяющий задавать структуру данных, и язык манипулирования данными

(ЯМД) (DML, Data Manipulation Language, англ.), обеспечивающий доступ и изменение данных. Операторы этих языков были «встроены» в два языка программирования того времени – COBOL (Common Business-Oriented Language, англ.) и PL/1 (Programming Language One, англ.).

Элементарными единицами данных в модели CODASYL являются *элемент* и *агрегат* данных, совокупность которых образует *запись*. Записи, имеющие одинаковую внутреннюю структуру и описывающие однородные объекты, объединяются в один *тип записи.* Между двумя типами записей можно устанавливать связь «один ко многим», называемую *типом набора,* при этом один тип записи объявляется владельцем типа набора, а другой - его членом. Каждая запись первого типа может быть владельцем своего *набора,* членами которого являются записи второго типа. Физически члены одного набора ссылаются друг на друга с помощью *указателей* трех видов: указатель на следующего члена набора, указатель на предыдущего члена набора и прямой указатель на владельца набора. По сути, такая модель напоминает современные навигационные структуры информации в интернете.

С точки зрения структуры данных, наиболее существенным различием между сетевой и более простой иерархической моделью является отсутствие ограничений на установление связей – один и тот же тип записи может быть не только владельцем любого числа типов наборов (т.е. иметь несколько потомков), но и сам быть членом нескольких типов наборов (т.е. иметь несколько предков); вторым важным отличием сетевой модели является то, что в общем случае в наборе могут участвовать не два, а несколько типов записей. Благодаря своей высокой выразительности, сетевая модель стала очень популярной среди отечественных разработчиков, создавших целый ряд сетевых СУБД.

1.5 Реляционные базы данных

По существу, реляционная модель – это возврат к плоским файлам. Появлению этой модели предшествовали работы британского ученого Эдгара (Теда) Кодда (Edgar Codd). В конце 1960-х годов, Кодд работал на лабораторию IBM в Сан-Хосе, штат Калифорния, был хорошо знаком с базами того времени, и не был согласен с их конструкциями. В частности, он считал, что:

1) Существующие базы данных было слишком трудно использовать. С ними могли работать только люди, обладающие специальными навыками программирования.

2) Существующим базам данных не хватало теоретического фундамента. Кодд предложил подумать о данных в терминах формальных математических структур и логических операций; он рассматривал существующие базы данных, как использование произвольных представлений, которые не обеспечивают логической последовательности.

3) Существующие базы данных смешивают логические и физические реализации. Представление данных в существующих базах данных соответствует формату физического хранения в базе данных, а не логическому представлению данных. Что опять же делало их сложными для изучения людьми нетехнических специальностей.

Тогда Кодд публикует документ внутри компании IBM с изложением его идей для более формализованной модели базы данных. В 1970 г. он опубликовал статью, ставшую легендой, «Реляционная модель данных для больших, совместно используемых банков данных». По мнению коллег математика, ему удалось в своей работе виртуозно решить проблему оптимального поиска и управления массивом данных посредством изящества математики и логики предикатов. А ведь именно «красивые» (не в ущерб практичности) математические выкладки в написании программ были несбыточной мечтой многих программистов того времени. Так Кодд стал прародителем современной технологии хранилищ данных.

Во вводной части своей статьи Э. Кодд подвергает критике иерархическую и сетевую модели, которые не обеспечивают однозначное пред-

ставление связей на физическом уровне. На простом примере он показывает, что одну и ту же схему данных можно представить пятью разными способами физической организации записей. Это усугубляется обилием указателей и необходимостью упорядочивать записи на физическом уровне, что в свою очередь усиливает зависимость между приложениями и данными.

Стремясь избавиться от этой зависимости, Кодд предлагает все данные, включая и связи между объектами предметной области, хранить в виде *отношений* – двумерных таблиц. Схема данных складывается из схем отдельных таблиц. Каждый столбец таблицы соответствует понятию поля записи, а каждый *элемент* – понятию элемента данных в сетевой модели. Количество столбцов определяет степень отношения. Строка таблицы или *кортеж* отношения – это аналог экземпляра сегмента в иерархической модели или записи в сетевой модели. Порядок следования строк может быть произвольным.

Появление реляционной модели данных явило вторую революцию баз данных. Первой реляционной СУБД стала System R, созданная в середине 1970-х гг. в рамках исследовательского проекта IBM. Несмотря на то, что в основу ее разработки легли идеи Кодда, он не участвовал в этом проекте. Для работы с данными был создан специальный язык SEQUEL (Structured English Query Language – структурированный английский язык запросов, англ.). Согласно непроверенным сведениям, слово «SEQUEL» в то время являлось торговой маркой английской авиакомпании, вследствие чего язык был переименован в SQL (Structured Query Language, англ.).

Кодд, как автор реляционной модели, считал язык SQL недостаточным для полной реализации своей теории. Будучи в оппозиции, Кодд покинул IBM и, вместе с программистом Крисом Дейтом (Christopher J. Date), основал независимую консалтинговую компанию, где они продолжили работу над реляционной моделью. По сегодняшний день, в книгах Дейта, можно наблюдать его критическую позицию по отношению ко многим допущениям SQL (например, о допустимости значения NULL). В 1985 году Кодд предлагает

свои знаменитые «12 правил», в которых подробно определяет собственное видение оптимального содержания реляционной СУБД.

Тем не менее, SQL стал первым языком, реализующим реляционный подход к структурированию данных. Первый официальный стандарт языка SQL – SQL-86 был принят в 1986 г. Американским национальным институтом стандартов (ANSI, American National Standards Institute) и одобрен Международной организацией по стандартизации (ISO, International Standardization Organization). Затем появились его модификации SQL-89, SQL-92 (или SQL2), SQL-1999 (SQL3), SQL-2003, SQL-2006, SQL-2008. Полное официальное описание SQL содержится в девяти стандартах серии ISO/IEC 9075:2008. Information technology - Database languages - SQL.

System R была исследовательским проектом, а коммерциализация реляционной технологии произошла благодаря двум другим разработкам – Ingres и Oracle. Разработка СУБД Ingres велась в Калифорнийском университете параллельно System R и поддерживалась военными ведомствами США. Ingres стала «полигоном» для отработки технологий, реализованных позднее в Sybase, Informix и MS SQL Server. После коммерциализации Ingres группа разработчиков из Калифорнийского университета приступила к созданию на ее основе новой СУБД Postgres (Post Ingres), вышедшей в 1995 г.

Наибольшее влияние на коммерческое распространение реляционной модели оказала СУБД Oracle, выпущенная компанией Relational Software, Inc. (ныне Oracle Corporation) в 1979 г. IBM также участвовала в конкурентной борьбе, выпустив в 1982 г. реляционную СУБД DB2.

Следующий исторический шаг в развитии реляционных баз данных сделала компания Sybase, выпустив в 1987 г. Sybase SQL Server, предназначенный для построения высокопроизводительной клиент-серверной архитектуры. С 1988 по 1996 гг. Sybase сотрудничала с Microsoft, при этом практически один и тот же продукт выпускался под двумя названиями Sybase SQL Server и Microsoft SQL Server. К середине 1990-х гг. формируется «большая тройка» производителей СУБД: Sybase – Oracle – Informix.

К концу 1980-х годов, реляционная модель выиграла битву между базами и стала традиционной.

Развитие технологий программирования, а, в первую очередь, появление объектно-ориентированного подхода, его популяризация и переход языков на эту методологию в 80-х и начале 90-х, выявили серьезную проблему в реляционных базах данных. Разработчиками были замечены несоответствия между объектно-ориентированным представлением данных в программах и реляционном представлением этих данных в базе. Это вынуждало разрабатывать что-то новое, ведь для того чтобы сохранить или извлечь объект из реляционный базы данных требовалось выполнение нескольких SQL операций преобразования из объектно-ориентированного представления в реляционное и наоборот. И на все это затрачивались ресурсы, как мозговые – программиста, так и вычислительные, но опять же это приводило к проблемам производительности и надежности. Сторонники объектно-ориентированного программирования начали видеть реляционную базу данных как пережиток процедурного прошлого. В то время довольно известно было такое высказывание: «реляционная база данных, как гараж, который заставляет вас купить машину по частям и хранить эти части в маленьких ящиках» (A relational database is like a garage that forces you to take your car apart and store the pieces in little drawers., англ.).

Быстрый успех объектно-ориентированного программирования привел почти неизбежно к утверждению о том, что система объектно-ориентированного управления базами данных (ООСУБД) лучше подходит для удовлетворения потребностей современных приложений. Она позволяла хранить объекты программы непосредственно без нормализации, и приложения легко могли загружать и хранить свои объекты.

Пропаганда модели ООСУБД выросла в середине 90-х годов, и многим казалось естественным, что ООСУБД будет логическим преемником РСУБД. Нынешние поставщики реляционных баз данных, в первую очередь Oracle, Informix, Sybase и IBM, – быстро взялись за реализацию функций ООСУБД в

своих РСУБД. В то же время, были разработаны и получили первоначальное сцепление с дорогой некоторые чистые системы ООСУБД. Тем не менее, к концу десятилетия, ООСУБД системам полностью не удалось получить долю на рынке.

Производители баз данных, такие как Oracle и Informix успешно реализовали множество функций ООСУБД, даже редко используемые, сохранив долю на рынке и дав возможность программистам не уходить от своих любимых РСУБД.

После того, как шумиха вокруг объектно-ориентированных баз данных поутихла, реляционные базы данных властвовали до второй половины 2000-х годов. На самом деле, в течение примерно 10 лет (1995-2005), не было создано кардинально новых баз данных: было достаточно реляционных систем баз данных, чтобы насытить рынок и удовлетворить любым потребностям, модель РСУБД мертвой хваткой удерживала рынок, не позволяя возникнуть нереляционным альтернативам, что свидетельствует о силе реляционной модели.

2 NoSQL базы данных

Интернет бурно развивался. На передний план выходили такие крупные компании как Google. К 2005 году Google.com стал самым крупным сайтом в мире. Когда Google начинала, реляционная база данных хорошо зарекомендовала себя, всех устраивала и удовлетворяла необходимым потребностям, но постепенно ее стало недостаточно, чтобы справиться с объемами и скоростью данных, с которыми сталкивалась компания. Проблемы, с которыми сталкиваются предприятия, с «большими данными», сегодня являются проблемы, с которыми Google впервые столкнулись почти 20 лет назад. Очень рано, можно даже сказать самой первой, Google пришлось изобретать новые аппаратные и программные архитектуры, хранить и обрабатывать экспоненциально растущее количество веб-сайтов.

В то время как Google имела общий масштаб работы и объем данных далеко превышающий пределы любой другой веб-компании, другие веб-сайты тоже сталкивались с собственными проблемами. Сайт, посвященный онлайн электронной коммерции – Amazon, например, имел потребность в транзакционной возможности обработки, которая могла бы работать в массовом масштабе. Сайты социальных сетей, таких как MySpace и Facebook в конце концов сталкиваются с аналогичными проблемами, в расширении их инфраструктуры от нескольких тысяч до миллионов пользователей.

Даже самые дорогие коммерческие СУБД, такие как Oracle, не могли обеспечить достаточную масштабируемость для удовлетворения потребностей этих сайтов. Многие сайты пытались масштабировать базы данных с открытым исходным кодом с помощью различных методов. Это включало случаи использования распределенных объектов, такие как Memcached, чтобы разгрузить базы данных от нагрузки, также применялась техника шардинга (Sharding, англ.), суть которой заключалась в разделении данных на несколько физических серверов на основе ключевого атрибута, например, идентификатора клиента. Например, в Twitter и Facebook, данные о клиентах разбива-

ется на очень большое количество баз данных MySQL. Большинство данных для каждого конкретного пользователя находились в одной базе данных, так что операции для конкретного клиента производились довольно быстро.

Но эксплуатационные расходы на сегментирование, вместе с потерей реляционных функций, делали свое дело, что опять же вынуждало искать альтернативы РСУБД.

И альтернативы стали появляться.

Термин «NoSQL» впервые был применен в 1998 году Карло Строцци (Carlo Strozzi) в качестве названия для своей нереляционной системы управления базами данных (Strozzi NoSQL), не использующей язык SQL для манипулирования данными. Название произошло от того факта, что база данных не использовала SQL в качестве языка запросов, вместо этого, она манипулировала скриптами. Второе рождение термин получил в 2009 году, когда Эрик Эванс (Eric Evans) на конференции в Сан-Франциско, посвященной свободным распределенным базам данных и организованной Йоханом Оскарсоном (Johan Oskarsson), использовал его для обозначения нереляционных СУБД и хранилищ данных. Встреча этой конференции так же называлась хештегом «NoSQL». Именно эта конференция дала имя всей технологии и положила начало буму NoSQL-решений. С 2009 года термин «NoSQL» стал использоваться уже для обозначения растущего числа распределенных систем управления данными, которые отказывались от поддержки ACID-транзакций (Atomicity, Consistency, Isolation, Durability – Атомарность, Согласованность, Изолированность, Постоянство хранения, англ.) – одного из ключевых принципов работы с реляционными базами данных. По мнению Строцци, вместо «NoSQL» правильнее было бы называть эту технологию «NoREL» (не реляционные). Термин «NoSQL» обычно расшифровывается как «Not Only SQL», то есть «Не только SQL» и он не совсем подходит в качестве обобщенного названия нереляционных баз данных.

Главной предпосылкой к появлению NoSQL-систем послужила растущая потребность в горизонтальной масштабируемости приложений, другими

словами в необходимости наращивать производительность путём добавления новых вычислительных узлов к уже работающим. Таким образом, большинство NoSQL-систем изначально проектировались и создавались для работы в распределенной среде – кластере или облаке, где применение традиционных SQL-ориентированных систем связано с определёнными трудностями. Основной причиной отказа от поддержки транзакционной семантики послужила сложность эффективной реализации транзакций в распределённой среде: в общем случае приходилось использовать двухфазный протокол фиксации транзакций, который требовал пересылки большого количества сообщений по сети. Хотя NoSQL-системы обычно не поддерживают ACID-транзакции в полном объеме, в ряде случаев поддерживаются, например, атомарные операции для чтения и модификации, оптимистические блокировки и другие инструменты, помогающие упростить разработку приложений в условиях параллельного доступа к данным.

В этой главе я кратко расскажу про некоторые NoSQL решения.

2.1 Общие характеристики NoSQL баз данных

Общих характеристик для всех NoSQL баз данных не так много, несмотря на все текущее разнообразие систем.

2.1.1 Неструктурированность (schemaless)

В NoSQL базах данных структура данных не регламентирована (слабо типизирована), если сравнивать с реляционными. В отдельной строке или документе можно добавить произвольное поле без предварительного декларативного изменения структуры всей таблицы. Это позволяет менять модель данных, всего лишь отразив в исходном коде вашего приложения.

Если мы меняем логику приложения, значит, мы ожидаем новое поле также и при чтении. Но так как схемы данных нет, то поле отсутствует у дру-

гих уже существующих объектов. В этой ситуации есть два варианта дальнейших действий. Первый – это обход всех документов и обновление этого поля во всех существующих документах. В силу объемов данных этот процесс происходит без каких-либо проблем, а во время обновления уже существующие данные могут считываться другими процессами. Поэтому второй вариант – проверка в коде приложения – неизбежен.

Также отсутствие схемы дает нам возможность эффективной работы с разреженными (sparse, англ.) данными. Если в одном документе есть поле, а во втором – нет, значит, никакого пустого поля для второго создано не будет.

У неструктурированной схемы данных тоже есть свои недостатки – это дополнительный код в приложении. Второй недостаток – отсутствие всевозможных ограничений со стороны базы. Также возникают дополнительные сложности в понимании и контроле структуры данных при параллельной работе с базой разных проектов (отсутствуют какие-либо словари на стороне базы). Но в условиях быстро меняющегося современного мира такая гибкость является скорее преимуществом, нежели недостатком. Примером может служить Твиттер, несколько лет назад в его базе помимо текста твитта хранилось еще лишь немного дополнительной информации (время, Twitter handle и еще несколько байтов метаинформации), однако сейчас в дополнение к самому сообщению в базе сохраняется еще несколько килобайт метаданных.

2.1.2 Представление данных в виде агрегатов (aggregates)

Если сравнивать с реляционной моделью, которая сохраняет логическую бизнес-сущность приложения в различных физических таблицах с целью нормализации, NoSQL хранилища оперируют с этими сущностями как с неделимыми объектами.

В примере (рисунок 5) продемонстрированы агрегаты для стандартной концептуальной реляционной модели электронной коммерции (e-commerce, англ.) “заказ – позиции заказа – платежи – продукт”.

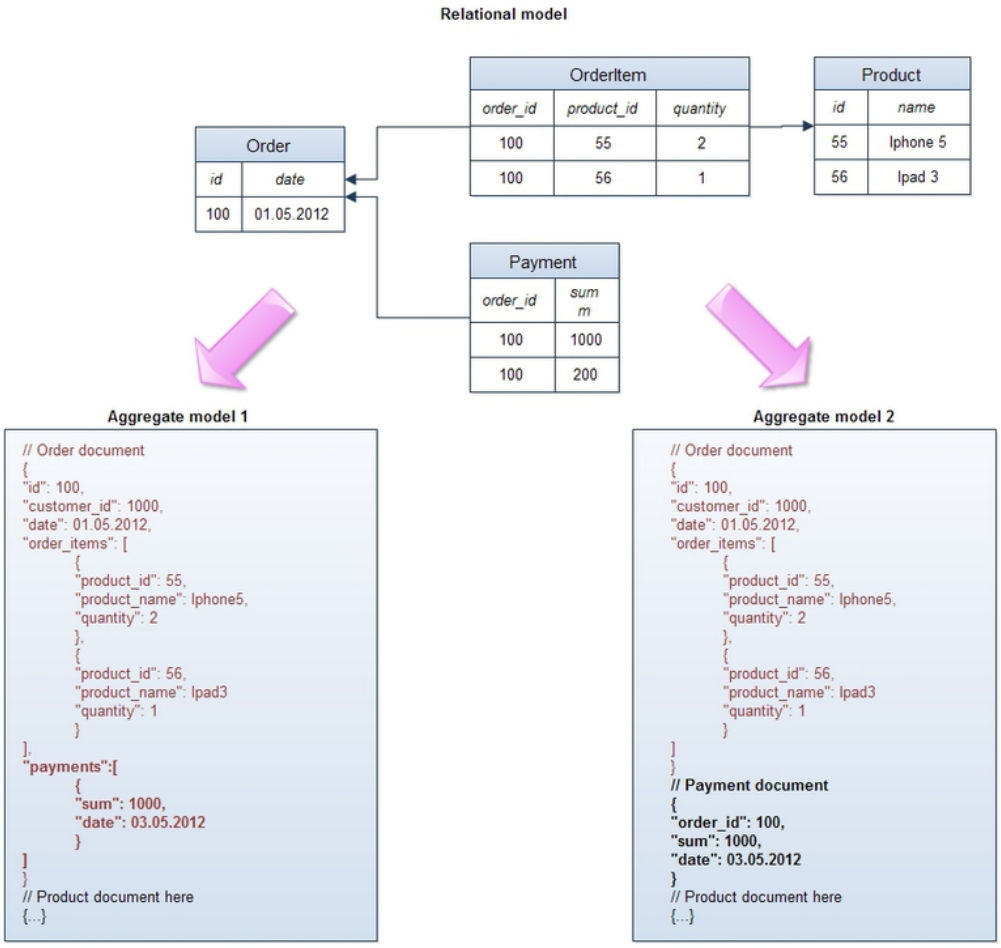


Рисунок 5 – Реляционная модель электронной коммерции в NoSQL

В обоих случаях заказ объединяется в один логический объект, каждая позиция хранит в себе ссылку на продукт и некоторые его атрибуты, например, название (такая денормализация необходима, чтобы не запрашивать объект продукта при извлечении заказа – главное правило распределенных систем – минимум “джоинов” между объектами). В одном агрегате платежи объединены с заказом и являются составной частью объекта, в другом – они вынесены и представляют собой отдельный объект. Это демонстрирует главное правило проектирования структуры данных в NoSQL базах – она должна

подчиняться требованиям приложения и быть максимально оптимизированной под наиболее частые запросы. Если платежи регулярно извлекаются вместе с заказом, тогда лучше их включать в общий объект, если же многие запросы работают только с платежами – значит, лучше их вынести в отдельную сущность.

Плюсы и минусы обоих подходов можно вынести в таблицу (таблица 3):

Таблица 3. Достоинства и недостатки представления данных в виде агрегатов, сравнивая с нормализацией данных

Нормализация данных	Данные в виде агрегатов
+ Целостность информации при обновлении (меняем запись в одной таблице, а не в нескольких) + Ориентированность на широкий спектр запросов к данным	- Оптимизация только под определенный вид запросов - Сложности при обновлении денормализованных данных
- Неэффективна в распределенной среде - Низкая скорость чтения при использовании объединений (joins) - Несоответствие объектной модели приложения физической структуре данных	+ Лучший способ добиться большой скорости при чтении в распределенной среде + Возможность хранить физически объекты в том виде, в каком с ними работает приложение (легче кодировать и меньше ошибок при преобразовании) + Родная (native) поддержка атомарности на уровне записей

2.1.3 Распределенные системы без совместно используемых ресурсов (share nothing)

Большинство NoSQL-систем являются распределенными. Распределённая архитектура позволяет достичь не только горизонтальной масштабируемости (в идеале – линейного роста производительности при добавлении новых узлов), но и увеличить надежность системы с помощью поддержания нескольких копий данных. В распределённых системах управления данными используются два основных приёма (практически всегда применяемых совместно):

Разделение данных (шардинг, sharding) – подход, при котором каждый узел системы содержит свою часть данных и выполняет операции над ними (рисунок 6). Этот метод является основным средством обеспечения горизонтальной масштабируемости, однако теперь операции над несколькими объектами могут вовлечь в работу несколько узлов, что требует активной передачи данных по сети. Кроме того, при увеличении числа узлов, хранящих данные, увеличивается вероятность сбоев, что требует наличия избыточности – применения репликации (см. далее). Шардинг также порождает такие задачи, как распределение данных по узлам, балансировка нагрузки, оптимизация сетевых взаимодействий и т.д.

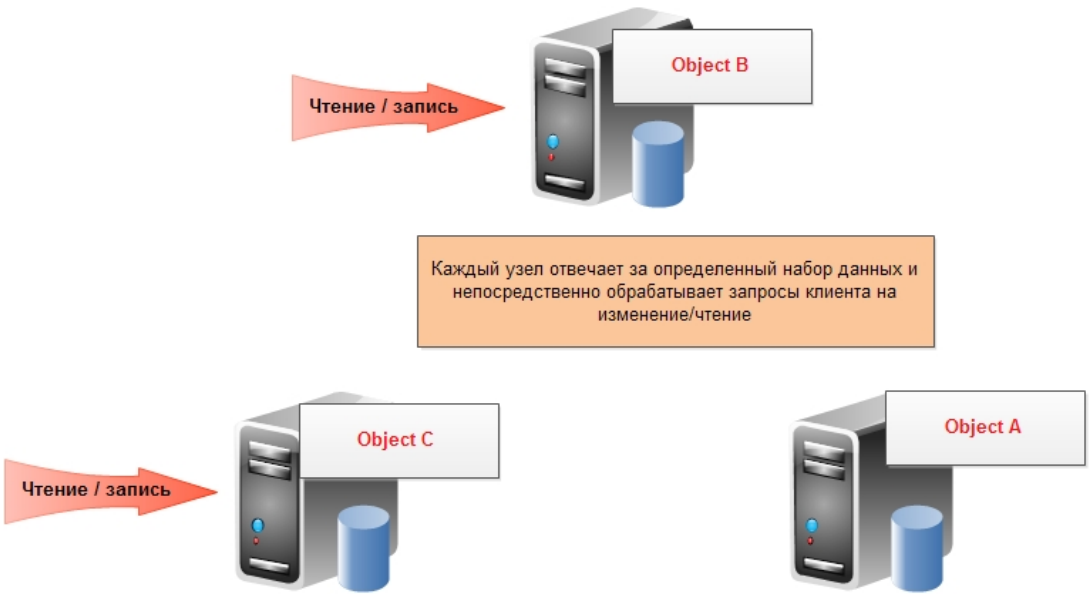


Рисунок 6 – Шардинг

Репликация (replication) – подход, при котором одни и те же данные хранятся на нескольких узлах в сети. Этот метод помогает повысить надежность системы и справляться как со сбоями отдельных узлов, так и с потерей целого кластера. Кроме того, репликация позволяет масштабировать операции чтения (несколько реже - записи). Серьезной задачей является поддержка согласованного состояния копий данных (реплик): при синхронном обновлении реплик увеличивается время ответа системы, а при асинхронном - возникает промежуток времени, когда реплики находятся в несогласованном состоянии.

Существуют две основные схемы репликации (поддерживающие как синхронные, так и асинхронные варианты):

Ведущий-ведомый (master-slave) – при такой схеме репликации операции модификации данных обрабатывают только ведущий узел (master), а сделанные изменения синхронно или асинхронно передаются на ведомого (slave) (рисунок 7). Чтения могут осуществляться как с ведущего узла (гарантированно содержит последнюю версию данных), так и с ведомого (данные могут быть несколько устаревшими при асинхронной репликации и отставать от ведущего).

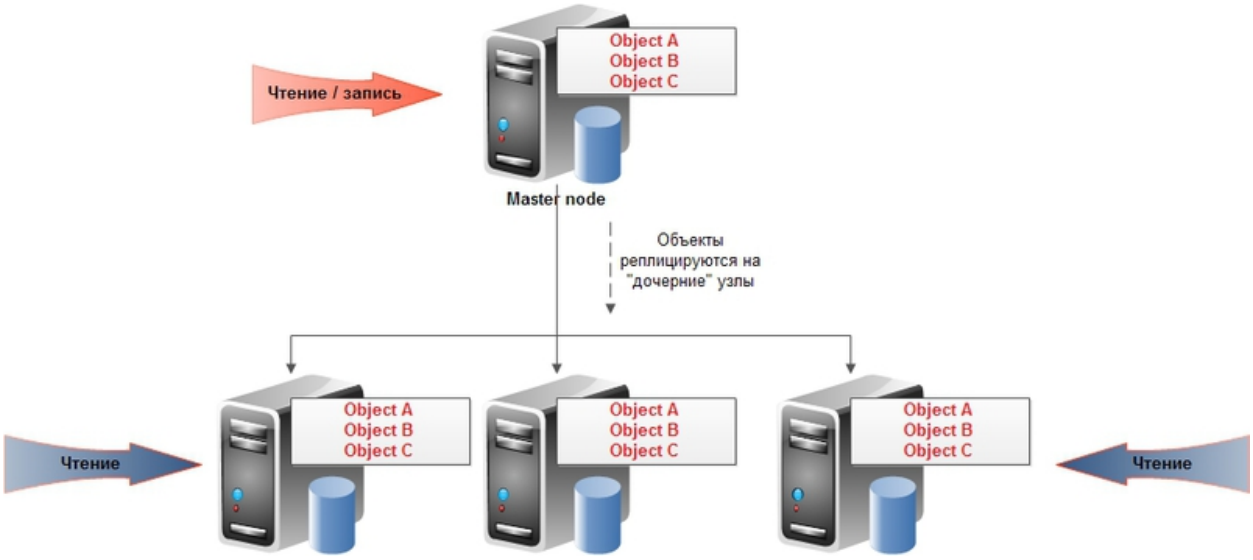


Рисунок 7 – Репликация master-slave

Ведущий-ведущий (master-master, multi-master, peer-to-peer) – при этой схеме все узлы могут обрабатывать операции записи и передавать обновления остальным (рисунок 8). В этом случае реализовать синхронную репликацию достаточно сложно, к тому же резко возрастают задержки, связанные с сетевыми взаимодействиями. При асинхронном обновлении возникает другая проблема – могут появиться конфликтующие версии данных, которые требуют наличия механизма для определения и разрешения конфликтов (автоматически или на уровне приложения).

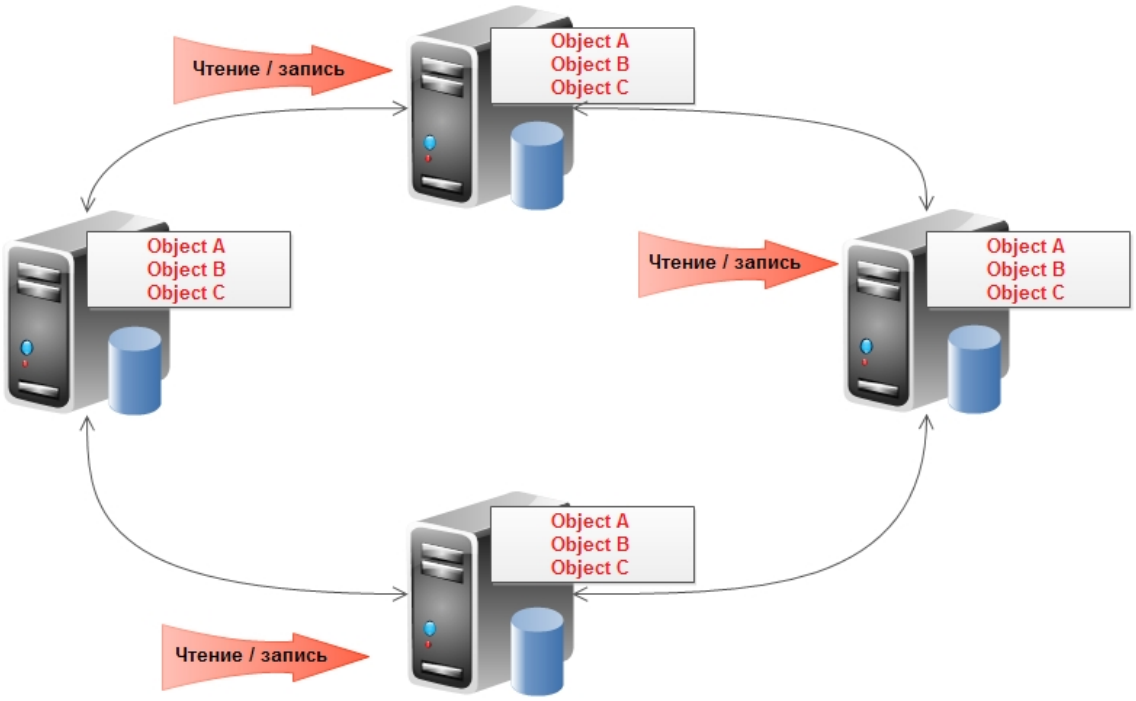


Рисунок 8 – Репликация peer-to-peer

Первый тип репликации предполагает хорошую масштабируемость на чтение (может происходить с любого узла), но не масштабируемую запись (только в мастер узел). Также существуют тонкости с обеспечением постоянной доступности (в случае падения мастера либо вручную, либо автоматически на его место назначается один из оставшихся узлов). Для второго типа репликации предполагается, что все узлы равны и могут обслуживать как запросы на чтение, так и на запись.

2.2 Модели согласованности

Применение репликации в распределённой системе порождает задачу поддержания идентичного состояния копий данных на разных узлах (и, соответственно, видимых разными клиентами). Для обозначения гарантий согласованности данных, которые предоставляются системой, используют термин «модель согласованности» (consistency model). Следует отметить, что слово «согласованность» в этом контексте отличается от свойства согласованности из определения ACID-транзакций. Модель согласованности определяет физическую согласованность состояния данных на разных узлах системы, в то время как в случае ACID имеется в виду логическая согласованность (в рамках ограничений целостности, определенных для базы данных). Ниже приводятся примеры моделей, часто применяемых в NoSQL-системах:

Согласованность в «конечном счете» (eventual consistency) гарантирует, что при отсутствии новых обновлений в течение некоторого времени все копии данных (реплики) станут согласованными.

Монотонные чтения (monotonic reads) являются усилением согласованности «в конечном счёте», гарантируя, что если какое-либо значение было прочитано клиентом, то последующие чтения никогда не вернут предыдущие значения.

Чтение своих записей (read your writes) также усиливает согласованность «в конечном счёте», давая гарантии того, что клиент всегда увидит данные, которые он до этого записал. Эта модель может также комбинироваться с монотонными чтениями.

Мгновенная согласованность (immediate consistency) означает, что как только операция модификации данных успешно завершена, все клиенты мгновенно увидят это изменение. Такую модель согласованности поддерживают, например, системы с синхронной репликацией.

Наиболее часто NoSQL-системы поддерживают согласованность «в конечном счёте» или её вариации.

2.3 Теорема CAP

Для обоснования компромиссов, выбираемых NoSQL-системами, часто приводится утверждение, известное как «теорема CAP», или теорема Брюэра (Eric Brewer). Хотя в названии и присутствует слово «теорема», к математике оно отношения не имеет. Утверждение было сделано Эриком Брюэром в докладе на конференции Principles of Distributed Computing в 2000 году. И оно означает, что распределенная система не может гарантировать одновременного выполнения следующих трёх свойств:

1) *Согласованность (Consistency)* – все узлы в каждый момент времени имеют согласованные данные (все пользователи в любой момент времени видят одинаковые данные).

2) *Доступность (Availability)* – при выходе из строя каких-либо узлов оставшиеся узлы должны продолжать функционировать.

3) *Устойчивость к разделению (Partition Tolerance)* – если из-за сбоя сети система распадается на группы узлов, не связанные между собой, то каждая группа должна продолжать функционировать.

Спустя два года, в 2002, Сет Гилберт и Ненси Линч (Seth Gilbert and Nancy Lynch) из Массачусетского технологического института опубликовали исследование, в котором дали определние понятию CAP и формализовали "*отложенную согласованность*" (*Delayed Consistency*), которую потом прозвали "*согласованность в конечном счете*" (*Eventual Consistency*). В дальнейшем многие практики ссылались на данную работу, как на доказательство теоремы CAP.

2.4 Модели данных и их классификация

Модель данных – это совокупность структур данных и операций их обработки. Еще одним важным отличием систем категории NoSQL от реляци-

онных баз данных являются их нереляционные модели данных и способы осуществления запросов. В целом модели данных, лежащие в основе NoSQL, значительно проще, чем классическая реляционная модель, что в ряде случаев облегчает работу с ними. Обычно (хотя и с некоторыми вариациями) NoSQL-системы классифицируют, исходя из модели данных, на следующие основные классы:

1) Системы «ключ-значение» (Key-Value Stores);

2) Документные СУБД (Document Stores);

3) Системы типа Google BigTable (Extensible Record Stores / Wide Column Stores / Column Families);

4) Базы данных на основе графов (Graph databases);

5) Мультимодельные базы (Multimodel databases).

Иногда под термином «NoSQL» понимают также вообще все системы, не являющиеся реляционными (SQL-ориентированными), однако чаще всего имеются в виду именно представители приведённых пяти классов систем. Существуют также более подробные классификации, скажем на сайте nosql-database.org выделены и иные классы, такие как Object Databases, Grid & Cloud Databases Solutions и другие.

Кроме того, существуют системы, которым присущи черты более чем одного класса, и иногда их трудно классифицировать по такому принципу. Некоторые выделяют такие базы в отдельную группу - *Мультимодельные базы (Multi-model databases)*, поддерживающие несколько моделей данных.

2.4.1 Системы «ключ-значение»

NoSQL-системы типа «ключ-значение» хранят данные (неструктурированные или структурированные) и позволяют получать доступ к ним при помощи единственного уникального ключа. Работа с данными обычно осуществляется с помощью простых операций вставки, удаления и поиска по ключу. Вторичные ключи и индексы в таких системах не поддерживаются. При

этом может поддерживаться некоторая структура данных, позволяющая менять отдельные поля объекта, но не позволяющая строить по ним запросы (в этом заключается основное отличие систем типа «ключ-значение» от документных СУБД).

Список NoSQL-систем класса «ключ-значение» насчитывает более 30 систем. Наиболее популярные из которых: Berkeley DB, MemcacheDB, Redis, Riak, Amazon DynamoDB. Все эти системы обладают своими особенностями и нюансами и подходят для различных задач. При выборе конкретной системы приходится принимать во внимание множество факторов, таких как желаемый уровень согласованности данных, наличие атомарных операций, легкость масштабирования и администрирования, надежность, наличие клиентских библиотек для используемого языка программирования и т.д. В целом, достоинствами систем типа «ключ-значение» являются хорошая горизонтальная масштабируемость, простота и производительность, однако часто их моделей данных оказывается недостаточно для построения серьезных приложений, где возникает потребность, например, в поиске по сочетанию атрибутов, поддержка вложенных объектов, индексы на полях объектов и т.д. В этом случае нужно обратить внимание на системы с более сложной моделью данных, например, документные СУБД.

2.4.2 Документные СУБД

Документные СУБД предоставляют больше возможностей, чем системы типа «ключ-значение». Единицей хранения данных в таких системах является документ – некоторый объект, обладающий произвольным набором атрибутов (полей), который может быть представлен, например, в JSON. Документные системы поддерживают поиск по полям документов, индексы, часто допускаются вложенные документы и массивы, а заранее предопределённой схемы данных, как правило, нет. В отличие от систем типа «ключ-значение», документные СУБД позволяют запрашивать коллекции докумен-

тов на основании нескольких ограничений на атрибуты, могут осуществлять агрегатные запросы, сортировку результатов, поддерживают индексы на полях документов и т.д. Документные системы обычно не поддерживают семантику ACID, однако между собой они значительно различаются в плане поддержки согласованности данных, наличии атомарных операций, а также в способах контроля параллельного доступа к документам и во многих других аспектах.

Документные СУБД имеют гибкую модель данных, которая в ряде случаев является более удобной, чем фиксированная схема, и лучше сочетается с объектно-ориентированным программированием, сокращая прослойку между языком программирования и СУБД. Системы этого класса могут достаточно легко масштабироваться, хотя делают это немного по-разному. Способы построения запросов также различаются, но в целом поддерживаются достаточно сложные выборки, включающие ограничения на значения полей, агрегацию, сортировку и т.д. К вопросу согласованности данных документные системы также подходят по-разному, обычно позволяя в определенной степени варьировать их в зависимости от конфигурации и потребностей приложения. Кроме того, могут быть реализованы дополнительные механизмы, например, оптимистические блокировки, атомарные операции и т.д., что позволяет усилить гарантии согласованности данных для тех приложений, где это необходимо. Документные СУБД обычно поддерживают постоянство хранения данных, используя запись на жесткие диски или SSD-накопители, а надежность обеспечивается с помощью журналирования и репликации; индексы и часто используемые документы при этом обычно хранятся в оперативной памяти для быстрого доступа. Транзакционная семантика на уровне нескольких документов в таких системах обычно не поддерживается, единственной возможностью является реализация на уровне приложения. Тем не менее, документные СУБД по функциональности постепенно приближаются к традиционным SQL-ориентированным СУБД. Примеры СУБД данного типа – CouchDB, Couchbase, MarkLogic, MongoDB.

2.4.3 Системы типа Google BigTable

Разработка Google BigTable была начата в 2004 году для поддержки различных сервисов Google, таких как Google Earth, Google Maps, Google Analytic и др. BigTable базируется на Google File System (GFS, используется для хранения данных и журнала), Chubby (используется для координации и хранения некоторых метаданных) и других разработках компании и не распространяется за пределами Google, но возможность её использования предоставляется в рамках Google App Engine. BigTable проектировалась таким образом, чтобы легко масштабироваться на сотни и тысячи узлов и работать с петабайтами данных. Примерами СУБД данного типа являются: Apache HBase, Apache Cassandra, Apache Accumulo, Hypertable, SimpleDB (Amazon.com).

Таблица BigTable представляет собой отображение ключа ряда (row key, англ.), ключа столбца (column key, англ.) и временной метки (timestamp, англ.) в значение в виде строки. Ключ ряда и ключ столбца также являются обычными строками. Ключи рядов упорядочены в лексикографическом порядке, а столбцы объединены в семейства столбцов (column family, англ.), которые должны быть определены до использования, после чего в каждое семейство столбцы могут быть добавлены динамически. Семейства столбцов обычно хранят однотипные данные, и их число невелико (не более сотни), в то время как столбцов в семействе может быть неограниченное количество. Каждая ячейка таблицы может содержать несколько версий данных, помеченных временными метками и упорядоченными по ним, текущее значение имеет наибольшую временную метку, поддерживается автоматическое удаление старых версий; ячейки также могут вообще не содержать данных. Таким образом, каждая строка таблицы (ряд) может содержать произвольное число атрибутов (столбцов), входящих в заранее определённые семейства.

Ряды таблицы разделяются по диапазонам ключей, формируя относительно небольшие по размеру сегменты («tablets» в терминологии BigTable), являющиеся единицами распределения при балансировке нагрузки. Кластер BigTable содержит один главный сервер (master) и сервера, непосредственно хранящие сегменты. Главный сервер отвечает за распределение сегментов по узлам, балансировку нагрузки, операции со схемой и т.д. Семейства столбцов являются единицами контроля прав доступа и параметров хранения. Данные хранятся по столбцам, а семейства столбцов, доступ к которым обычно осуществляется вместе, могут быть выделены в группы локальности (locality groups, англ.), что позволяет оптимизировать чтения. На уровне группы можно указать, например, чтобы данные ее семейств столбцов постоянно находились в оперативной памяти и читались из нее, а также настроить сжатие данных. BigTable поддерживает асинхронную репликацию между кластерами, гарантируя при этом согласованность «в конечном счёте».

В BigTable предусмотрены операции для создания и удаления таблиц и семейств столбцов, изменения метаданных (например, прав доступа), записи и удаления значений, чтения определенных строк, просмотра подмножеств данных (например, столбцов из определённого семейства). Также поддерживаются атомарные операции над строками таблицы и исполнение скриптов для обработки данных. Клиентские библиотеки кэшируют метаданные о расположении сегментов и большую часть времени обращаются непосредственно к узлам, хранящим данные.

2.4.4 Базы данных на основе графов

Графовую модель данных обычно рассматривают как обобщение RDF-модели или сетевой модели данных. Основными элементами модели являются *узлы* и *связи*. В зависимости от реализации узлов и ребер граф-модель данных разделяют на несколько подтипов. Данный вид баз данных применяется для моделирования социальных графов (социальных сетей), биоинформати-

ке, а также для семантического веб. Примеры графовых баз данных: Neo4j, OrientDB, AllegroGraph, InfiniteGraph, FlockDB.

По мнению некоторых авторов, для задач с естественной графовой структурой данных графовые СУБД могут существенно превосходить реляционные по производительности, а также иметь преимущества в наглядности представления и внесения изменений в схему БД. Более подробно о графовых базах данных будет рассказано в третьей главе на примере БД Neo4j.

2.5 Реляционная база данных или NoSQL? Преимущества NoSQL

Так что же лучше использовать в своем проекте, классическую реляционную модель или присмотреть NoSQL-решение? При окончательном выборе того или иного типа базы данных в первую очередь необходимо понимать специфику решаемых задач и требования к выбираемым технологиям. Понимание же существенных различий между структурированными и неструктурированными базами данных может облегчить процедуру выбора наиболее приемлемого решения.

Базы данных, созданные с помощью языка SQL (Structured Query Language), активно развиваются почти четыре десятилетия. В конце 90-х годов прошлого века развитие веб-технологий и программ с открытым исходным кодом привело к тому, что господство на рынке захватили несколько вендоров. Наиболее популярными СУБД стали MySQL, PostgreSQL и SQLite.

Что же касается баз данных NoSQL, то, не смотря на их почти полувековое существование, взлёт популярности данного типа БД пришелся только на последние пять лет. На рынке появились такие известные продукты, как MongoDB, CouchDB, Redis и Apache Cassandra. В последнее время количество вендоров и названий систем управления неструктурированными базами данных растет невиданными темпами. В таблице 4 приведены основные отличия реляционных СУБД и NoSQL.

Таблица 4 Основные отличия двух типов СУБД: SQL и NoSQL

	SQL	NoSQL
Хранилище данных	Хранится в реляционной модели в строках и столбцах. Строки содержат всю информацию об одной специфической характеристике/сущности, а столбцы - все отдельные значения данных.	Термин "NoSQL" включает в себя множество баз данных, каждая из которых с различными моделями хранения данных. Основными из них являются: документ, граф, ключ-значение и столбчатые данные.
Схемы и гибкость	Каждая запись соответствует определенной схеме, а это означает, что столбцы данных должны быть спроектированы до внесения данных, а каждая строка должна соответствовать столбцу. Схема может быть изменена, но ее изменение потребует изменение всей базы данных.	Схема данных является динамической и может меняться в любой момент времени. Запись может содержать не все столбцы.
Масштабируемость	Масштабирование происходит по вертикали. Это значит, что чем больше данных, тем больше сервер, что приводит в высокой стоимости хранения. Можно масштабировать РСУБД на нескольких серверах, но это сложный и трудоемкий процесс.	Масштабирование носит горизонтальный характер, то есть на нескольких серверах. Это можно сделать с затратой небольших ресурсов на оборудование и ПО, включая облачные технологии. Данный подход более экономичен по сравнению с вертикальным масштабированием. Кроме того технологии NoSQL могут автоматически распределять данные по серверам.
Соответствие ACID	Подавляющее большинство реляционных баз данных соответствует ACID.	Все зависит от технологии. Однако многие решения в области NoSQL приносят в жертву ACID ради производительности и масштабируемости.

Некоторые уверены, что понятие «неструктурированные базы данных» подразумевает просто некую модель данных. Однако, по существу, речь может идти о целом наборе технологий, которые подразумевают не только хранение, но и обработку данных разными методами.

Хранение документов предусмотрено и в реляционных базах данных, однако принцип сохранения документов сильно отличается. В неструктурированных документно-ориентированных БД данные хранятся в документах, а не в таблицах. Документы в NoSQL сгруппированы в коллекции, причём каждый документ может обладать собственной структурой.

В базах типа «ключ-значение», данные хранятся в ассоциативных массивах. Ключом является наименование атрибута, который связан со значением.

Графовые БД обычно используются для таких данных, которые лучше всего представить именно в графическом представлении (например, чтобы проследить взаимосвязи). В таких случаях данные хранятся в графических структурах, включающих узлы (сущности), характеристики (информация о сущностях) и линии (связи между сущностями). Графовые базы обладают тем преимуществом, что лучше всего показывают взаимоотношения (связи) сущностей.

Столбчатые базы данных образуют вместо таблиц семейство столбцов, которые выступают в качестве контейнеров для строк. В отличие от реляционных баз данных пользователю совсем не обязательно предварительно знать все столбцы. Кроме того, в таких БД не нужно, чтобы каждой строке соответствовало определенное количество столбцов. Столбчатые базы данных лучше всего подходит для анализа больших объемов данных.

У каждой NoSQL модели данных есть свои и достоинства и недостатки, потому что каждая модель создана для работы с определенными данными, реляционная модель универсальна, вопрос уже стоит в сложности структуры и эффективности работы. И выбор зависит только от задач и данных, с которыми вы будете иметь дело.

3 Немного о графовых базах данных. Neo4j

Что все-таки такое «графовая база данных»? Если говорить максимально просто, то это база данных, которая максимально приспособлена для хранения некоторых сущностей (*узлов*), способных иметь различные *свойства*, и *отношений* между ними. На языке ООП узел будет экземпляром некоторого класса в приложении, на языке графов – вершиной. Отношения между узлами – направленные ребра. Графовая модель данных позволяет организовать данные в виде графа один раз и в последствии работать с ним, интерпретируя различными способами по требуемому шаблону. Создав граф из узлов и ребер, мы можем посылать к нему разнообразные запросы. Запрос к графу называют *обходом графа*. Преимущество графовых баз в том, что мы можем менять запрос (требования обхода), мы просто обходим граф любым возможным способом и получаем искомую выборку данных.

Рассматривая различные NoSQL-решения, меня заинтересовала такая база данных как Neo4j. Почему я выбрал именно ее? – До этого я имел дело только с реляционными базами данных и немного с NoSQL базой данных – MongoDB. Поэтому документно-ориентированные базы данных я не стал рассматривать. Из курса «Теории графов» я узнал, что такое граф, вершины (узлы), ребра графа, рассматривал некоторые задачи из теории графов и многое другое. Зная о графах, мне стало любопытно, что же представляют из себя графовые базы данных? Так выбор был сделан в пользу именно этой модели данных. А Neo4j – потому что это самая известная и популярная графовая база данных.

Neo4j – это NoSQL база данных, разработанная компанией Neo Technology в 2003 году. Разработчики описывают свой продукт как «встраиваемая база данных, полнотранзакционный Java-движок, который хранит данные в упорядоченных графах, а не в таблицах». Но упор в ней делается не на данные, которые она хранит, а скорее на связи, которыми эти данные связаны. Благодаря этому совершенно разнородные данные можно хранить про-

сто и естественно, а операции над требуют меньше времени. Neo4j способна содержать десятки миллиардов узлов и столько же связей. И при наличии соответствующего оборудования она в силах справиться с задачей любого размера.

Согласно данным сайта db-engines.com, Neo4j является самой популярной графовой базой данных (рисунок 9).

20 systems in ranking, May 2016

Rank			DBMS	Database Model	Score		
May 2016	Apr 2016	May 2015			May 2016	Apr 2016	May 2015
1.	1.	1.	Neo4j	Graph DBMS	32.61	+0.70	+3.97
2.	2.	2.	OrientDB	Multi-model	6.13	-0.18	+2.14
3.	3.	3.	Titan	Graph DBMS	4.80	-0.31	+1.29
4.	4.	4.	Virtuoso	Multi-model	2.15	-0.24	-0.91
5.	5.	5.	ArangoDB	Multi-model	1.56	+0.05	+0.47
6.	6.	↑ 7.	Giraph	Graph DBMS	0.89	+0.03	+0.11
7.	7.	↑ 11.	Stardog	Multi-model	0.54	-0.03	+0.30
8.	8.	↓ 6.	AllegroGraph	Multi-model	0.50	-0.08	-0.28
9.	9.	↑ 10.	Sqrrl	Multi-model	0.31	+0.00	+0.01
10.	10.	↓ 9.	InfiniteGraph	Graph DBMS	0.23	+0.00	-0.13
11.	11.	↓ 8.	Sparksee	Graph DBMS	0.16	-0.02	-0.38
12.	↑ 14.	↑ 14.	FlockDB	Graph DBMS	0.15	+0.02	+0.04
13.	↓ 12.	↑ 15.	HyperGraphDB	Graph DBMS	0.13	-0.02	+0.02
14.	↓ 13.	↑ 16.	GlobalsDB	Multi-model	0.13	-0.02	+0.13
15.	↑ 16.	↓ 12.	InfoGrid	Graph DBMS	0.12	-0.01	-0.12
16.	↑ 17.	16.	Blazegraph	Multi-model	0.11	+0.04	+0.11
17.	↓ 15.	↓ 16.	VelocityGraph	Graph DBMS	0.10	-0.03	+0.10
18.	18.	↓ 13.	GraphDB	Multi-model	0.08	+0.02	-0.04
19.	19.	↓ 16.	GraphBase	Graph DBMS	0.02	-0.01	+0.02
20.	20.	↓ 16.	Amisa Server	Multi-model	0.00	±0.00	±0.00

Рисунок 9 – Топ 20 графовых баз данных сайта db-engines.com на май 2016

Для работы с любой базой необходим язык запросов. В Neo4j имеется свой собственный декларативный язык запросов – Cypher. Это относительно простой, но очень мощный язык. В Cypher включены такие ключевые слова

как WHERE, ORDER BY из SQL, синтаксис взят из таких языков как Python, Haskell, SPARQL. Скомбинировав все это и добавив что-то свое, получили язык, который позволяет делать запросы к графам в простой, привычной форме.

3.1 Основные элементы графовой базы данных Neo4j

В своей простейшей форме, базу графовую базу данных Neo4j можно рассматривать как совокупность объектов, связанных одним или несколькими типами отношений. Neo4j включают в себя четыре основных элемента: узлы (nodes), отношения (relationships), параметры (properties) и метки (labels).

Узлы/вершины: обычно каждый узел – это набор, состоящий из пар «ключ-значение». Как показано в примере на рисунке 10, приведенном ниже, два узла представляют собой объект *Person*, который имеет ключ *Name* со значением «Greg» и объект – *Business*, у которого есть ключ *Name* со значением «GraphStory».



Рисунок 10 – Пример простейшего графа. Узлы

Отношения/связи: в графах Neo4j каждое отношение имеет имя и направление, а также начальный и конечный узел. Также узел может иметь свойства. Направленные отношения являются важным аспектом графовых баз данных и понимания того, как они должны быть смоделированы, чтобы эффективно управлять базой данных. Узел может иметь отношения к себе. В приведенном ниже примере (рисунок 11), отношение [:WORKS_AT] утверждает, что «Greg работает на GraphStory».



Рисунок 11 – Пример простейшего графа. Отношение

Свойства/параметры: представляют собой пары ключ-значение и используется для представления атрибутов или метаданных для узлов и связей в графе. Узлы одного и того же типа могут иметь различные свойства, что является существенным преимуществом при управлении и анализе неструктурированных данных. На рисунке 12 можно увидеть ключи и значения рассмотренных ранее узлов. Узлы и отношения также несут *идентификатор* значения по умолчанию. Свойства могут иметь следующие значения:

- Численные значения,
- Строковые значения,
- Логические значения,
- Списки значений любого другого типа.



Рисунок 12 – Пример простейшего графа. Параметры узла

Метки: начиная с версии 2.0 Neo4j, была введена концепция меток как способ группировки узлов (рисунок 13). Группировка узлов дает объединение в наборы/множества. Маркировка узлов также предлагает способ, обеспечивающий соблюдение моделирования уровня ограничений, а также увеличивает скорость, с которой данные могут быть доступны посредством улучшенной индексации.



Рисунок 13 – Пример простейшего графа. Метки

3.2 Установка Neo4j

Я использовал дистрибутив Neo4J 3.0.1 x64, скаченный с официального сайта (neo4j.com) и установленный на Windows 7 x64. После установки, для начала работы достаточно указать путь к папке, где будет располагаться база данных или оставить папку по умолчанию и нажать «Start». Появиться ссыл-

ка (рисунок 14), нажав на которую, мы попадаем в веб-интерфейс базы (рисунок 15).

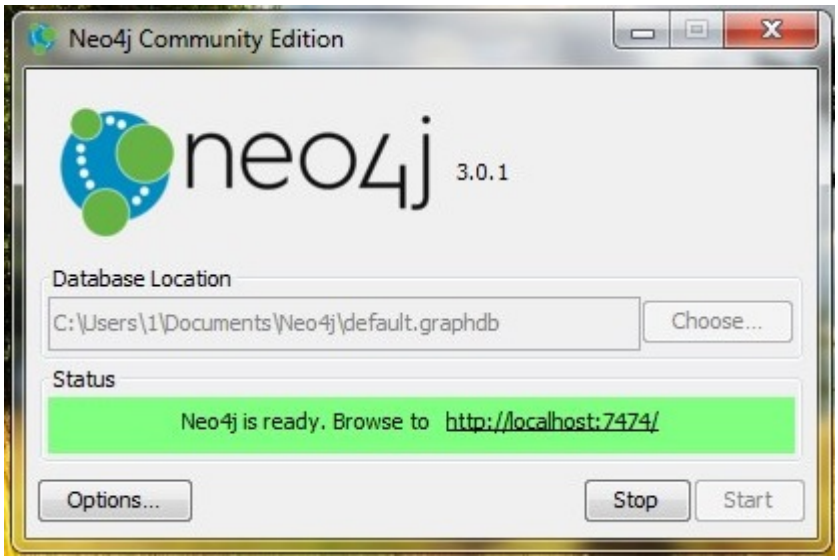


Рисунок 14 – Запуск Neo4j

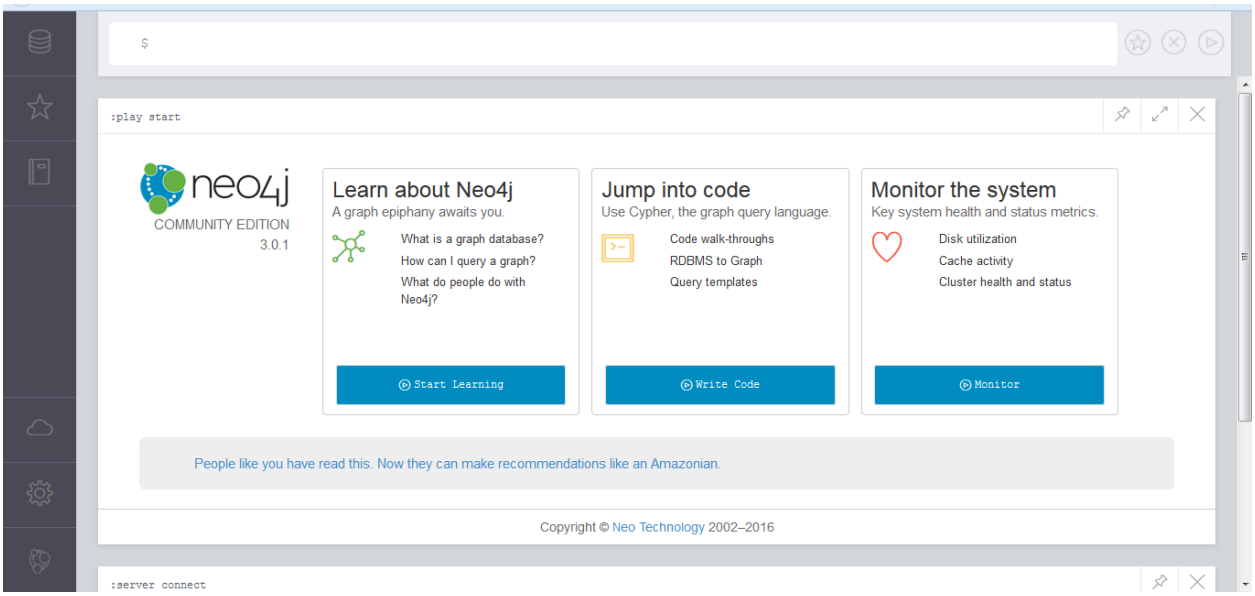


Рисунок 15 – Веб-интерфейс Neo4j

Слева расположено меню, сверху – консоль, ниже область, в которой можно посмотреть ссылки на ресурсы, которые помогут во всем разобраться, ссылки на примеры и многое другое. Также там будут появляться результаты действий, выполненных в консоли.

3.3 Язык запросов Cypher

Cypher – декларативный язык запросов, используемый для манипулирования данными в Neo4j. Он во многом схож с языком реляционных баз данных – Structured Query Language (SQL).

Cypher был создан, чтобы быть оптимально доступным и простым в использовании для максимально широкого спектра пользователей: разработчиков программного обеспечения, бизнес-аналитиков и технических архитекторов. Наиболее распространенные операции запросов в Cypher предназначены для того, чтобы сосредоточиться на том, что необходимо извлечь, а не на том, как это будет извлекаться.

Помимо превосходной скорости и способностей масштабирования, еще одним важным преимуществом использования Neo4j для операций с данными является транзакционной способность. Любой запрос Cypher, который изменяет граф данных представляет собой транзакцию и всегда либо выполнится полностью, либо не выполнится вовсе.

Я буду работать только с чистым Cypher, но Neo4j также поддерживает программный интерфейс для таких языков как Java, Python, Clojure, Ruby и других, имеет REST-интерфейс и возможность расширения программного интерфейса (API) с помощью серверных плагинов и неуправляемых расширений. Запросы можно делать и другими способами, например, напрямую через Java API или на языке Gremlin.

Теперь немного о синтаксисе языка. Начнем сразу с добавления данных. Для добавления данных используется команда CREATE. Эта команда просто создает шаблон данных, которые вы укажете. Например, в результате выполнения команды указанной ниже, будет создан один узел TheMatrix с меткой Movie и параметрами, указанными в фигурных скобках. Cypher возвратит количество изменений, в данном случае: добавление 1 узла, 1 метки и 2 свойств. Для удаления узлов и связей служит команда DELETE.

«CREATE (TheMatrix:Movie {title:"The Matrix", released:1997})»

Добавив после этой строки команду «RETURN TheMatrix», получим граф с нашими данными, состоящий из одного узла (рисунок 16). Чтобы добавить несколько узлов или вывести несколько элементов, необходимо перечислить их через запятую.

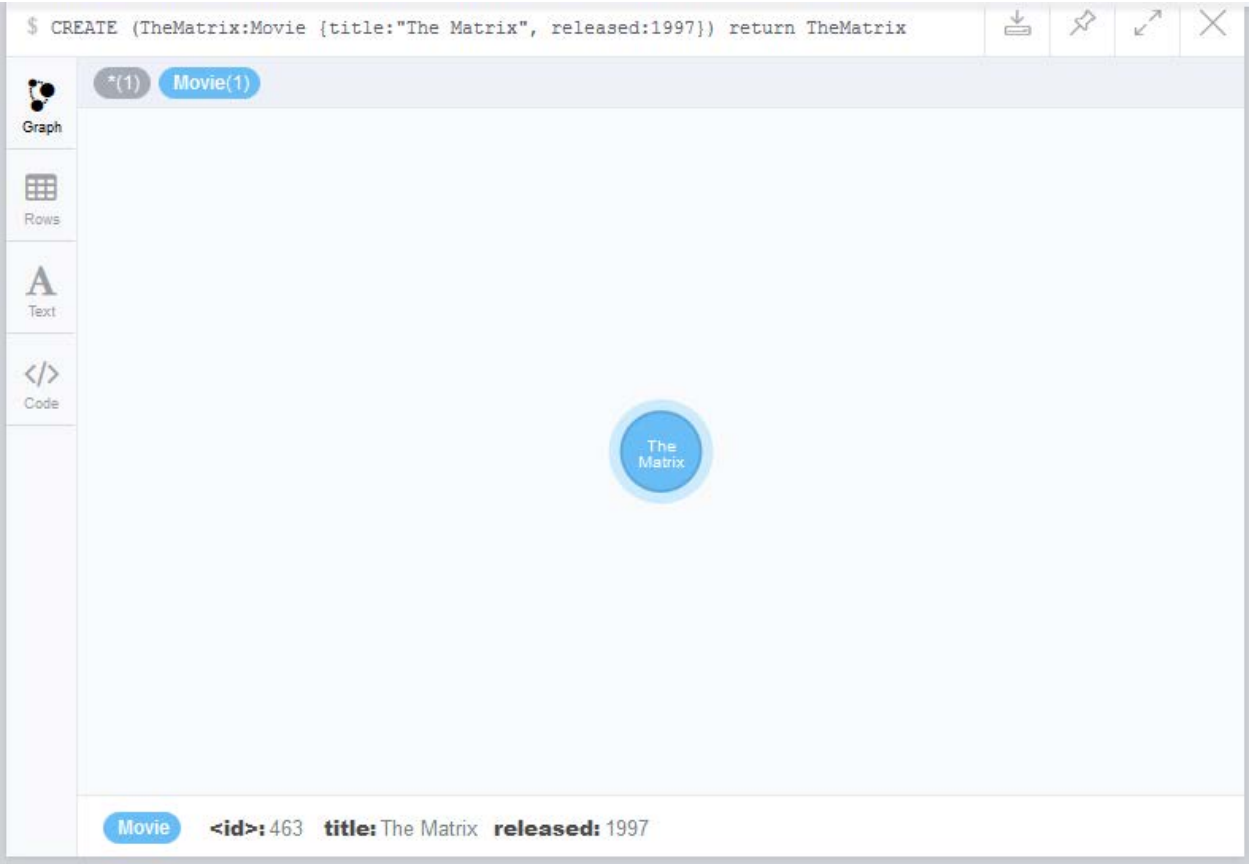


Рисунок 16 – Добавление узла TheMatrix

Перейду к работе с данными из приложения А. Данный фрагмент кода, добавляет в базу данные о фильмах, актерах, кинорежиссерах с некоторыми дополнительными параметрами (рисунок 17).



Рисунок 17 – Результат выполнения кода из приложения

Для поиска (поиска по образцу) в Cypher есть команда MATCH. Она работает так: мы как можем описываем то, что хотим найти, на выше упомянутом языке, и возвращаем результаты, похожие по описанию. Выполнив код:

```
MATCH (p:Person { name:"Keanu Reeves" })-[r:ACTED_IN]->(m:Movie)
RETURN m.title, r.roles
```

Получим в результате список фильмов, в которых снимался Keanu Reeves в связке с его ролью в этих фильмах (рисунок 18).

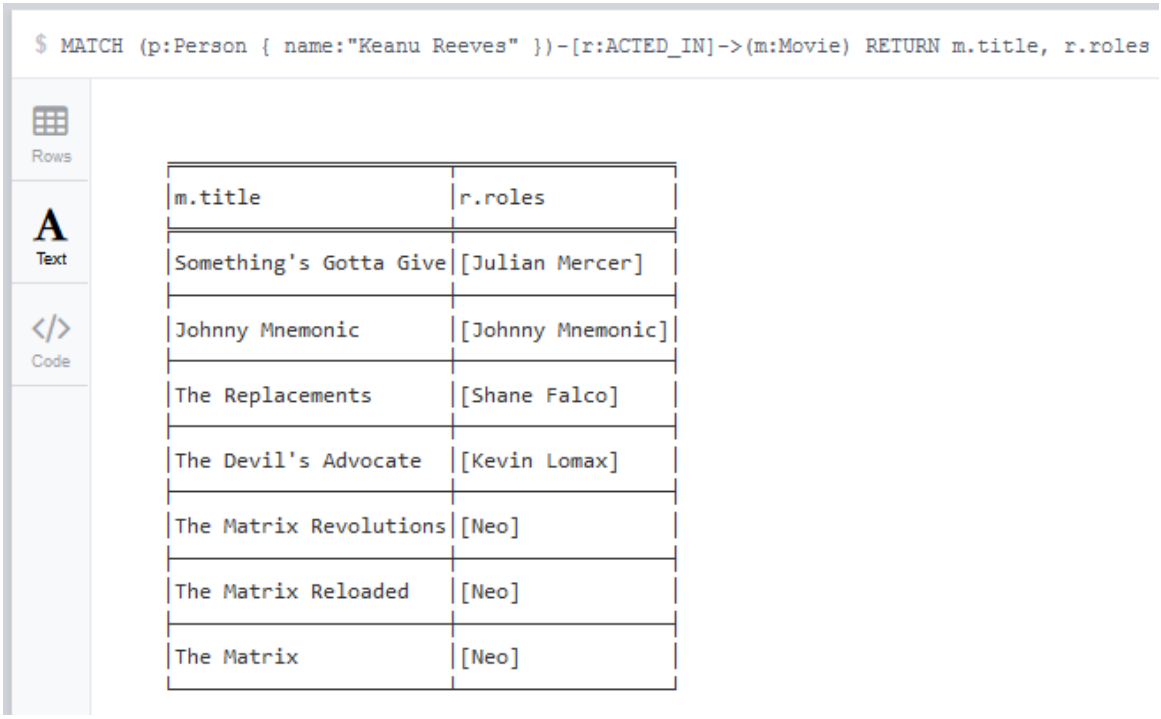


Рисунок 18 – Результат выполнения команды MATCH

Когда искать необходимо по каким-то условиям или критериям, на помощь приходит ключевое слово WHERE. После него можно расположить дополнительные условия, которые задаются набором операторов (рисунок 19) и регулярных выражений.

Operators	
Mathematical	+, -, *, /, %, ^
Comparison	=, <>, <, >, <=, >=
Boolean	AND, OR, XOR, NOT
String	+
Collection	+, IN, [x], [x .. y]
Regular Expression	=~

Рисунок 19 – Операторы языка Cypher

Из SQL перекочевали такие команды как «ORDER BY expression [ASC|DESC]» и «SKIP {offset} LIMIT {count}». На рисунке 20 результат выполнения кода (под рисунком 20), возвращающего топ 5 актеров, у которых больше всего ролей в фильмах.

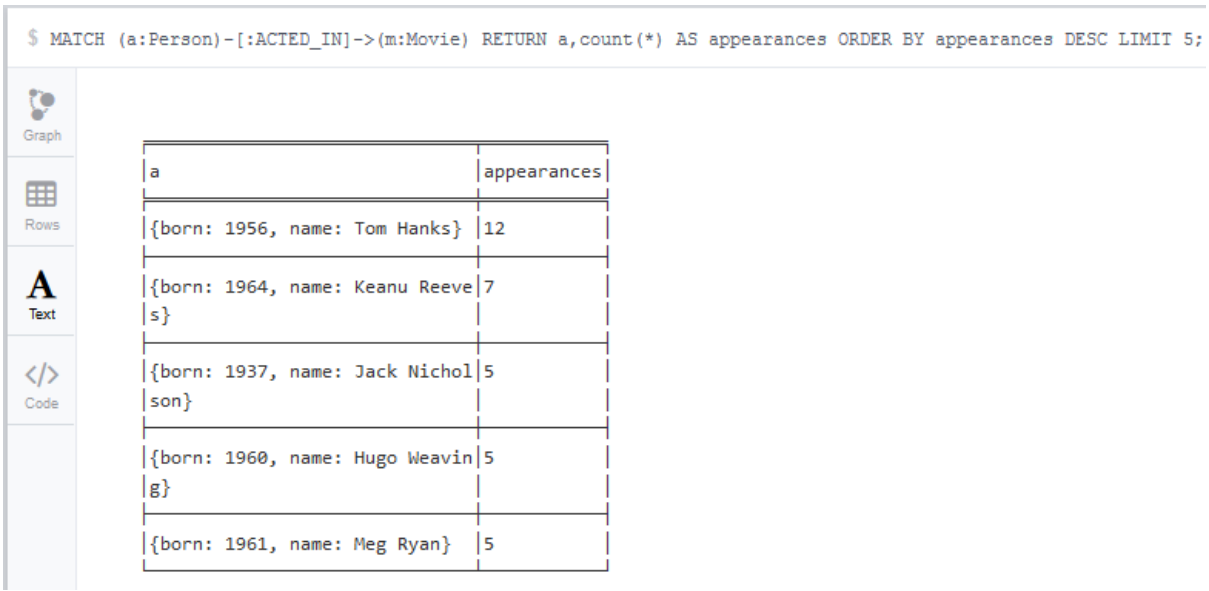


Рисунок 20 – Работа с примером. Команда ORDER BY

MATCH (a:Person)-[:ACTED_IN]->(m:Movie)

RETURN a, count(*) AS appearances

ORDER BY appearances DESC LIMIT 5;

Также в Neo4j есть возможность импорта данных из файлов формата CSV. Скачав базу данных в трех файлах, содержащую страны, регионы и города на русском языке, я решил занести все данные в Neo4j. Данные в файлах хранятся следующим образом (рисунок 21).

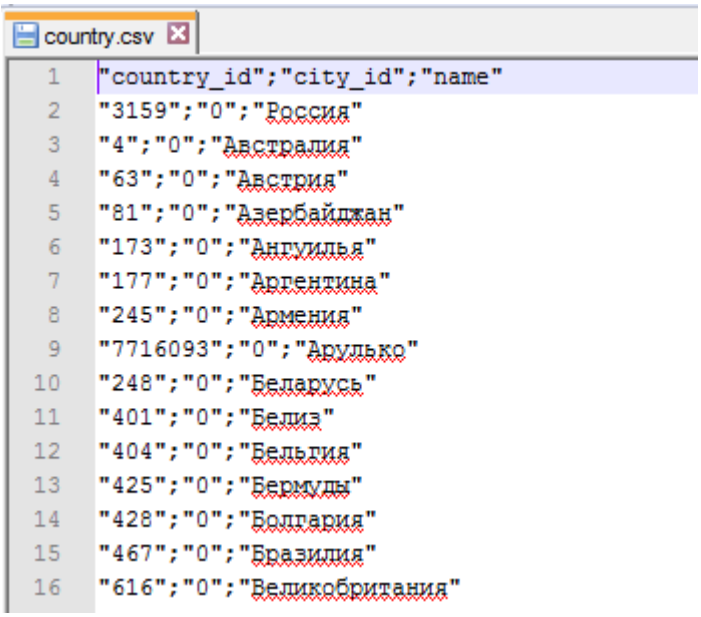


Рисунок 21 – Данные файла country.csv

Выполнив команды, находящиеся ниже, я внес данные в базу.

LOAD CSV WITH HEADERS FROM 'file:///country1.csv' AS line FIELDTERMINATOR ';'

CREATE (a:Country { name: line.name, country_id: toInt(line.country_id)})

RETURN a.name AS name, a.country_id AS country_id;

LOAD CSV WITH HEADERS FROM 'file:///region1.csv' AS line FIELDTERMINATOR ';'

CREATE (a:Region { name: line.name, region_id: toInt(line.region_id), country_id: toInt(line.country_id)})

RETURN a.name AS name, a.region_id AS region_id, a.country_id AS country_id;

LOAD CSV WITH HEADERS FROM 'file:///city1.csv' AS line FIELD-TERMINATOR ';'

CREATE (a:City { name: line.name, city_id: toInt(line.city_id), region_id: toInt(line.region_id), country_id: toInt(line.country_id)})

RETURN a.name AS name, a.city_id AS city_id, a.region_id AS region_id, a.country_id AS country_id;

Но кириллические символы отображались некорректно. Желая видеть свой граф данных в базе, я написал небольшую вспомогательную программу на C/C++, переводящую кириллические символы в латинские (смотрите приложение Б) и выполнил тот же самый код. На рисунке 22 представлен результат добавления городов в базу.



Рисунок 22 – Добавление данных из файла city1.csv

Создав узлы данных, я связал их связями, код представлен ниже.

```
MATCH (a:Region), (b:Country)
WHERE a.country_id = b.country_id
CREATE (a)-[r:MEMBER]->(b)
RETURN a.name AS Region, r AS relationship, b.name AS Country
```

```
MATCH (a:City), (b:Region)
WHERE a.region_id = b.region_id
CREATE (a)-[r:PARTOF]->(b)
RETURN a.name AS City, r AS relationship, b.name AS Region
```

Результат связывания регионов и городов представлен на рисунке 23.

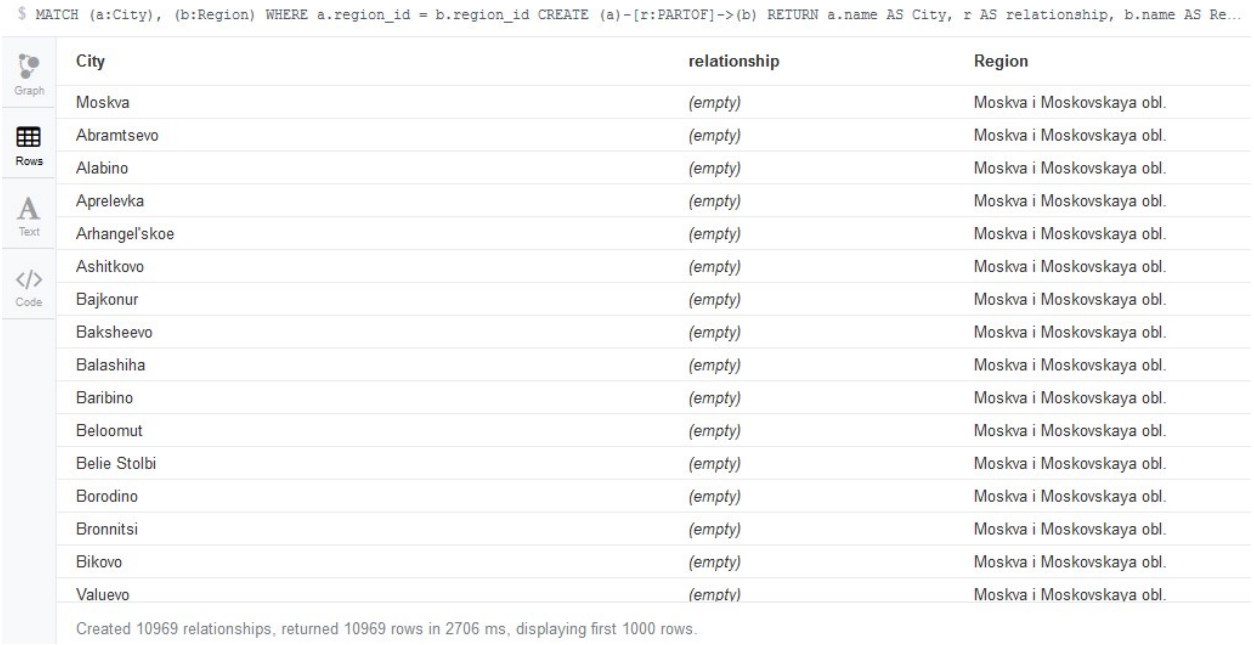


Рисунок 23 – Связывание регионов и городов

На рисунке 24 приведена часть полученного графа данных, отражающая связь страны – России и регионов, а на рисунке 25 – часть графа Московской области и населенных пунктов, расположенных в ней.

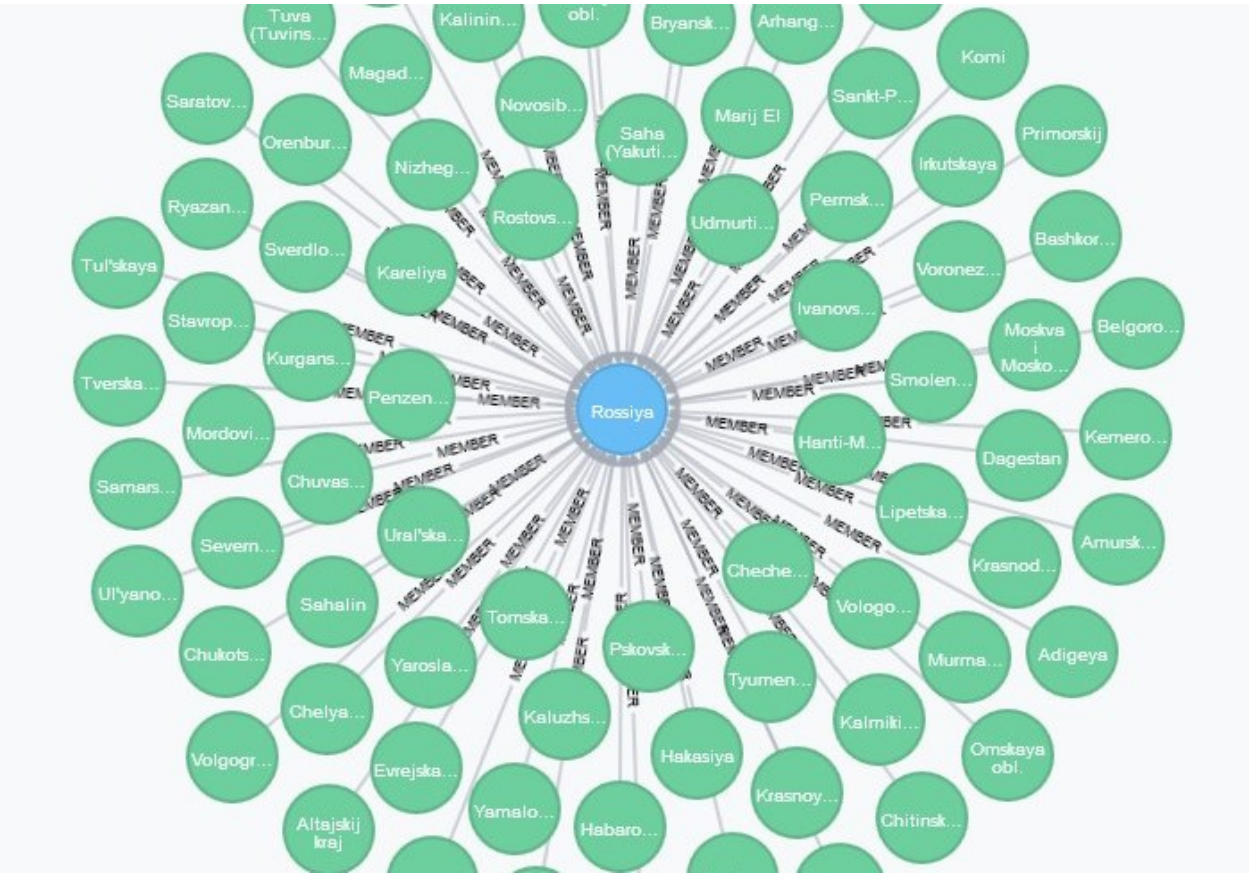


Рисунок 24 – Граф Россия и регионы

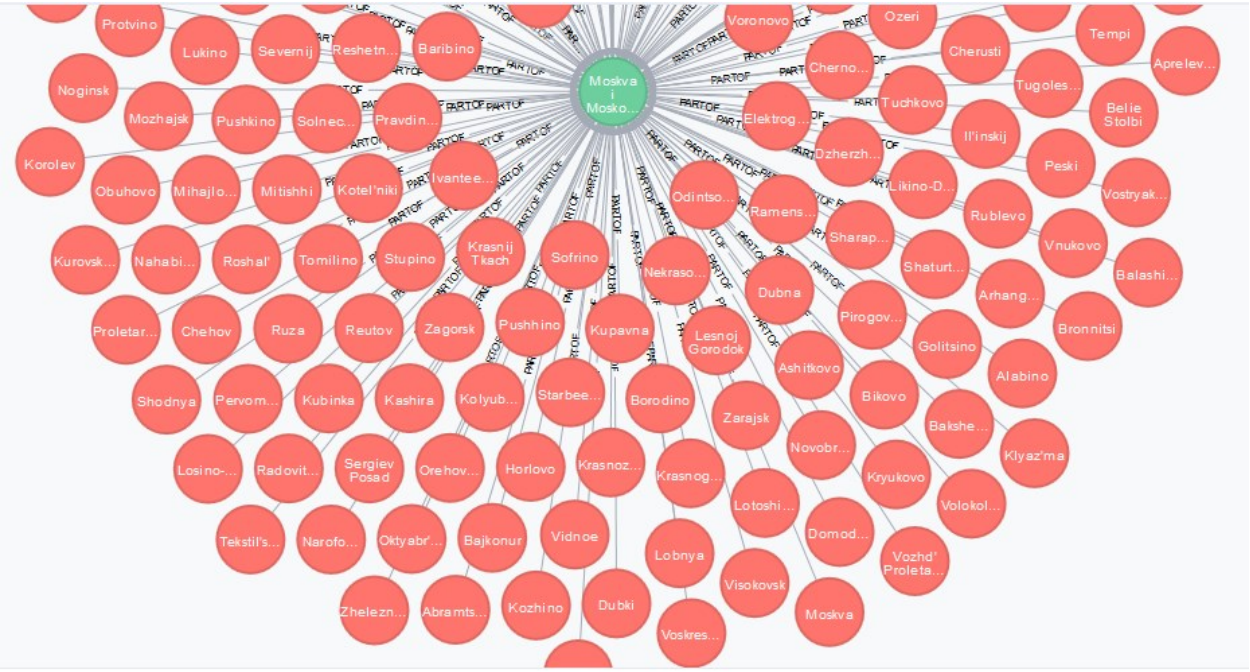


Рисунок 25 – Граф Московская область и населенные пункты

Это были лишь некоторые примеры применения основных команд языка Cypher базы Neo4j. Конечно же, есть множество других команд, структур и различных тонкостей их использования. Но даже этого минимума достаточно для начала работы с базой данных. Данные примеры показывают, что Cypher по своим возможностям ничуть не уступает SQL, а в некоторых аспектах даже превосходит его в работе со связями данных.

3.4 В каких случаях база данных Neo4j применима на практике и когда дает преимущества перед реляционной моделью?

Все, что так или иначе можно представить в виде графов, можно внести в Neo4j и с этим можно будет работать. Но так же этими данными можно заполнить и любую другую базу, реляционную или нереляционную. Но нам ведь нужно получить преимущества, показать что Neo4j будет работать лучше, эффективнее, нежели что-то другое. Как утверждается в описании, эта база должна иметь преимущества в работе со сложными запутанными информационными моделями с большим набором данных и множеством связей. Какие системы подходят под это описание?

1) Социальные сети;
2) Графы предложений и рекомендаций;
3) Дорожные, гео данные;
4) Транспортные данные;
5) Сложные информационные системы;

Пример из приложения А или рассмотренный выше пример с данными стран, городов и регионов не дают возможности сравнить эффективность графовой СУБД Neo4j с реляционной СУБД при большом количестве запросов. Данные слишком малы, их должно быть больше и со значительно более сложной структурой. Поэтому я приведу результаты работы исследователя, опубликовавшего свою статью в коллективном блоге Хабрахабр

(https://habrahabr.ru/post/258179/). Он заполнил базу MySQL и Neo4j одинаковыми данными:

addUsers(100 000); – количество пользователей.

addGroups(200 000); – количество групп.

addPhotos(300 000); – количество фотографий.

addAudio(250 000); – количество аудиозаписей.

addFriendships(1 000 000); – количество друзей.

addMessages(4 000 000); – количество сообщений.

addUserAudio(350 000); – количество аудиозаписей.

addUserGroups(400 000); – количество групп пользователей.

addUserPhoto(400 000); – количество фотографий пользователей.

В MySQL эти данные занимали порядка 350 Мб, а в Neo4j 3.5 Гб. – разница довольно существенная. После этого были проведены несколько экспериментов, в которых измерялось время выполнения запросов к базам. Автор определял время, затрачиваемое на поиск пользователя по его идентификатору определенное количество раз (рисунок 26). Эффективность работы Neo4j с большим числом запросов видна из этого графика.

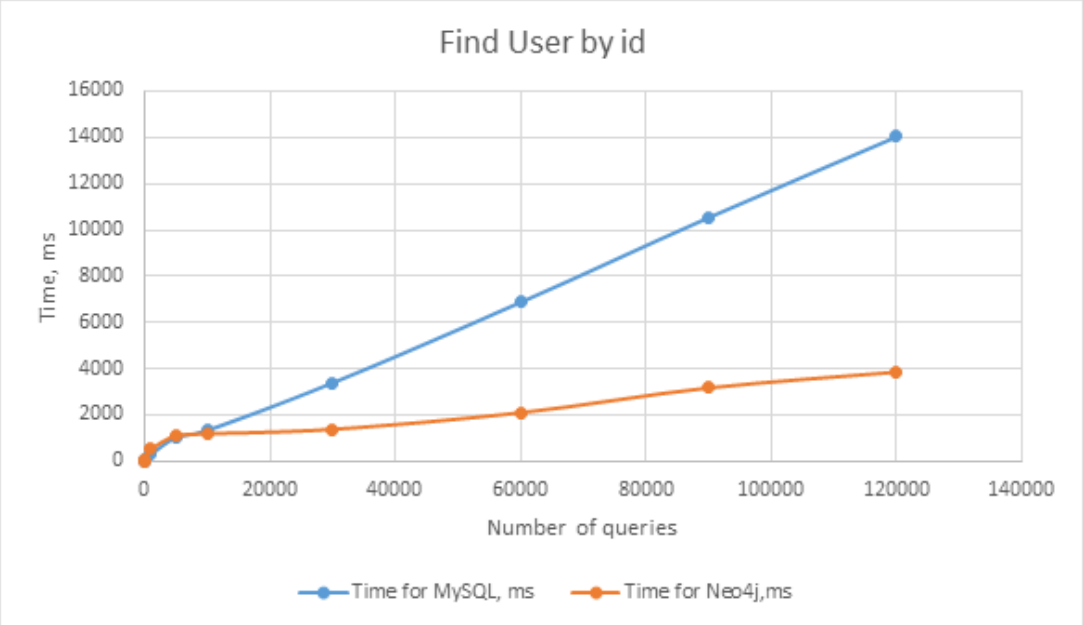


Рисунок 26 – График запросов к базам. Число запросов и время выполнения

В ходе следующего эксперимента определялось время поиска общего количества фотографий у пользователей, которые администрируют хотя бы одну группу, в зависимости от диапазона значений идентификаторов пользователя. Этот эксперимент более сложный, чем предыдущий (рисунок 27). На данном графике по горизонтали отмечается изменение интервала вхождения идентификаторов для поиска, а по вертикали – затрачиваемое время в миллисекундах. Из графика видно, насколько эффективней графовая база данных Neo4j по сравнению с MySQL в случае обработки сложных запросов в системе с большим объемом данных.

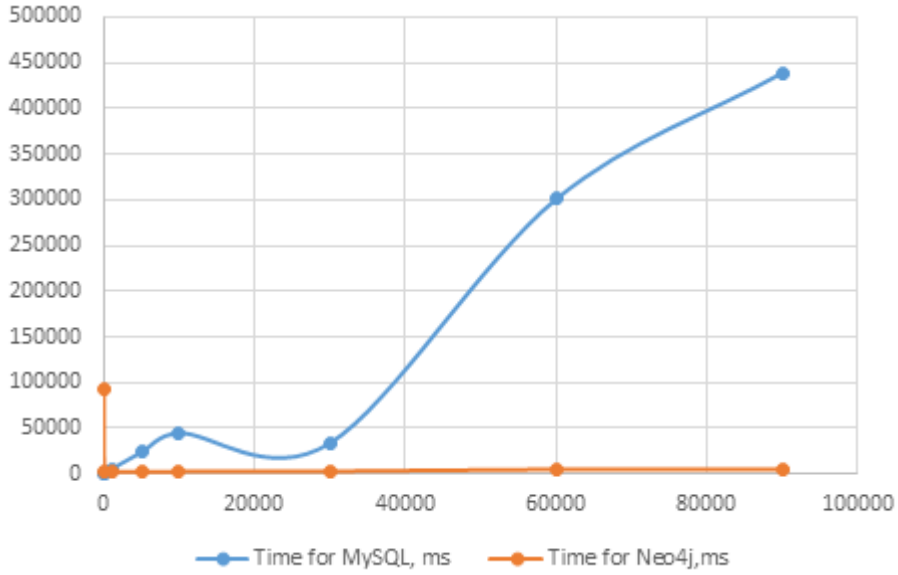


Рисунок 27 – Сравнение эффективности MySQL и Neo4j при работе со сложными запросами

Таким образом, можно сделать вывод: при больших объемах данных графовая СУБД Neo4j намного быстрее осуществляет поиск и эффективнее справляется с большим числом запросов, чем реляционная СУБД MySQL. Однако при этом для данных на жестком диске требуется значительно больше свободного места. Поэтому для небольших систем предпочтительней использовать реляционную БД или искать альтернативу среди других NoSQL баз данных.

В завершение выделим основные достоинства графовой базы данных Neo4j:

- Эффективность при работе с сильно связанными сущностями;
- Поддержка индексов на значениях атрибутов;
- Поддержка ACID;
- Простой синтаксис языка запросов Cypher;
- Наличие REST API;
- Доступность различных плагинов и расширений.

Необходимо отметить, что в настоящее время СУБД Neo4j успешно применяются во многих крупных компаниях: E-bay, Walmart, National Geographic, HP, CISCO и т. д.

ЗАКЛЮЧЕНИЕ

В рамках проделанной работы был рассмотрен путь эволюции хранилищ данных, от самых примитивных до современных баз данных. Представлены основные модели данных, относящиеся к категории NoSQL, изучены их достоинства и недостатки в сравнении с классической реляционной моделью. Для каждой модели данных были отмечены области их практического применения, в которых будут проявляться их сильные стороны.

На примере СУБД Neo4j была более подробно рассмотрена графовая модель данных. Управление данными в этой СУБД осуществляется с помощью запросов на языке Cypher. Результаты тестов подтверждают – если необходимо работать со сложными, содержащими множество взаимосвязей, данными, то их размещение в этой СУБД позволяет получать более быстрые и эффективные решения, чем при использовании традиционных реляционных СУБД.

Выбор подходящего для конкретного проекта хранилища данных, учитывая все их разнообразие на сегодняшний день, – нетривиальная задача. С задачами классического характера, допускающими создание строгой структуры данных, успешно справится и реляционная модель, а вот в более специфических случаях, когда требуется гибкая структура хранилища, лучше могут подойти нереляционные модели.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 Анохин А. Н. Эволюция баз данных. Учебное пособие по курсу «Базы данных». – Обнинск: ИАТЭ НИЯУ МИФИ, 2011. – 84 с.

2 Изучаем граф-ориентированную СУБД Neo4j на примере лексической базы Wordnet [Электронный ресурс] // Habrahabr : Habrahabr.ru. – [Б. м. : б. и.], [2015]. – Режим доступа: https://habrahabr.ru/post/273241/.

3 Кузнецов С. Д. Системы управления данными категории NoSQL. Научная статья. / С. Д. Кузнецов, А. В. Посконин – М. : МГУ им. М. В. Ломоносова. 2014. – 19 с.

4 Начинаем работать с графовой базой данных Neo4j [Электронный ресурс] // Habrahabr : Habrahabr.ru. – [Б. м. : б. и.], [2014]. – Режим доступа: https://habrahabr.ru/post/219441/.

5 Страны, регионы, города [Электронный ресурс] // Habrahabr : Habrahabr.ru. – [Б. м. : б. и.], [2008]. – Режим доступа: https://habrahabr.ru/post/21949/.

6 Мартин Фаулер NoSQL: новая методология разработки нереляционных баз данных : пер. с англ. / Мартин Фаулер, Прамодкумар Дж. Садалаж – М.: Вильямс, 2013. - 192 с.

7 Эрик Редмонд Семь баз данных за семь недель. Введение в современные базы данных и идеологию NoSQL : пер. с англ. / Эрик Редмонд, Джим Р. Уилсон – М.: ДМК Пресс, 2013. – 384 с.

8 Elements of a Graph Database [Электронный ресурс] // Graph Story : Graphstory.com. – [Б. м. : б. и.], [2016]. – Режим доступа: http://www.graphstory.com/elements-of-a-graph-database.

9 Gregory Jordan Practical Neo4j. – New York : Apress Media, 2014. – 393 p.

10 Guy Harrison Next Generation Databases NoSQL, NewSQL, and Big Data. – New York : Apress Media, 2015. – 235 p.

11 Knowledge Base of Relational and NoSQL Database Management Systems [Электронный ресурс] // DB-Engines : Db-engines.com. – [Б. м. : б. и.], [2016]. – Режим доступа: http://db-engines.com/en/ranking/graph+dbms.

12 Neo4j [Электронный ресурс] // Википедия : Wikipedia.org. – [Б. м. : б. и.], [2016]. – Режим доступа: https://ru.wikipedia.org/wiki/Neo4j.

13 NoSQL базы данных: понимаем суть [Электронный ресурс] // Habrahabr : Habrahabr.ru. – [Б. м. : б. и.], [2012]. – Режим доступа: https://habrahabr.ru/post/152477/.

14 NoSQL [Электронный ресурс] // Википедия : Wikipedia.org. – [Б. м. : б. и.], [2015]. – Режим доступа: https://ru.wikipedia.org/wiki/NoSQL.

15 The Neo4j Developer Manual v3.0 [Электронный ресурс] // Neo Technology : Neo4j.com. – [Б. м. : б. и.], [2016]. – Режим доступа: http://neo4j.com/docs/developer-manual/current/#cypher-query-lang.

ПРИЛОЖЕНИЕ А

(обязательное)

Код на языке Cypher, добавляющий данные о фильмах, актерах, режиссерах и продюсерах в базу Neo4j

```
CREATE (TheMatrix:Movie {title:'The Matrix', released:1999, tagline:'Welcome to the Real World'})
CREATE (Keanu:Person {name:'Keanu Reeves', born:1964})
CREATE (Carrie:Person {name:'Carrie-Anne Moss', born:1967})
CREATE (Laurence:Person {name:'Laurence Fishburne', born:1961})
CREATE (Hugo:Person {name:'Hugo Weaving', born:1960})
CREATE (AndyW:Person {name:'Andy Wachowski', born:1967})
CREATE (LanaW:Person {name:'Lana Wachowski', born:1965})
CREATE (JoelS:Person {name:'Joel Silver', born:1952})
CREATE
  (Keanu)-[:ACTED_IN {roles:['Neo']}]->(TheMatrix),
  (Carrie)-[:ACTED_IN {roles:['Trinity']}]->(TheMatrix),
  (Laurence)-[:ACTED_IN {roles:['Morpheus']}]->(TheMatrix),
  (Hugo)-[:ACTED_IN {roles:['Agent Smith']}]->(TheMatrix),
  (AndyW)-[:DIRECTED]->(TheMatrix),
  (LanaW)-[:DIRECTED]->(TheMatrix),
  (JoelS)-[:PRODUCED]->(TheMatrix)
CREATE (Emil:Person {name:"Emil Eifrem", born:1978})
CREATE (Emil)-[:ACTED_IN {roles:["Emil"]}]->(TheMatrix)

CREATE (TheMatrixReloaded:Movie {title:'The Matrix Reloaded', released:2003, tagline:'Free your mind'})
CREATE
  (Keanu)-[:ACTED_IN {roles:['Neo']}]->(TheMatrixReloaded),
  (Carrie)-[:ACTED_IN {roles:['Trinity']}]->(TheMatrixReloaded),
  (Laurence)-[:ACTED_IN {roles:['Morpheus']}]->(TheMatrixReloaded),
  (Hugo)-[:ACTED_IN {roles:['Agent Smith']}]->(TheMatrixReloaded),
  (AndyW)-[:DIRECTED]->(TheMatrixReloaded),
  (LanaW)-[:DIRECTED]->(TheMatrixReloaded),
  (JoelS)-[:PRODUCED]->(TheMatrixReloaded)

CREATE (TheMatrixRevolutions:Movie {title:'The Matrix Revolutions', released:2003, tagline:'Everything that has a beginning has an end'})
CREATE
  (Keanu)-[:ACTED_IN {roles:['Neo']}]->(TheMatrixRevolutions),
```

```
  (Carrie)-[:ACTED_IN {roles:['Trinity']}]->(TheMatrixRevolutions),
  (Laurence)-[:ACTED_IN {roles:['Morpheus']}]->(TheMatrixRevolutions),
  (Hugo)-[:ACTED_IN {roles:['Agent Smith']}]->(TheMatrixRevolutions),
  (AndyW)-[:DIRECTED]->(TheMatrixRevolutions),
  (LanaW)-[:DIRECTED]->(TheMatrixRevolutions),
  (JoelS)-[:PRODUCED]->(TheMatrixRevolutions)

CREATE (TheDevilsAdvocate:Movie {title:"The Devil's Advocate", re-
leased:1997, tagline:'Evil has its winning ways'})
CREATE (Charlize:Person {name:'Charlize Theron', born:1975})
CREATE (Al:Person {name:'Al Pacino', born:1940})
CREATE (Taylor:Person {name:'Taylor Hackford', born:1944})
CREATE
  (Keanu)-[:ACTED_IN {roles:['Kevin Lomax']}]->(TheDevilsAdvocate),
  (Charlize)-[:ACTED_IN {roles:['Mary Ann Lomax']}]->(TheDevilsAdvocate),
  (Al)-[:ACTED_IN {roles:['John Milton']}]->(TheDevilsAdvocate),
  (Taylor)-[:DIRECTED]->(TheDevilsAdvocate)
CREATE (AFewGoodMen:Movie {title:"A Few Good Men", released:1992, tag-
line:"In the heart of the nation's capital, in a courthouse of the U.S. government,
one man will stop at nothing to keep his honor, and one will stop at nothing to find
the truth."})
CREATE (TomC:Person {name:'Tom Cruise', born:1962})
CREATE (JackN:Person {name:'Jack Nicholson', born:1937})
CREATE (DemiM:Person {name:'Demi Moore', born:1962})
CREATE (KevinB:Person {name:'Kevin Bacon', born:1958})
CREATE (KieferS:Person {name:'Kiefer Sutherland', born:1966})
CREATE (NoahW:Person {name:'Noah Wyle', born:1971})
CREATE (CubaG:Person {name:'Cuba Gooding Jr.', born:1968})
CREATE (KevinP:Person {name:'Kevin Pollak', born:1957})
CREATE (JTW:Person {name:'J.T. Walsh', born:1943})
CREATE (JamesM:Person {name:'James Marshall', born:1967})
CREATE (ChristopherG:Person {name:'Christopher Guest', born:1948})
CREATE (RobR:Person {name:'Rob Reiner', born:1947})
CREATE (AaronS:Person {name:'Aaron Sorkin', born:1961})
CREATE
  (TomC)-[:ACTED_IN {roles:['Lt. Daniel Kaffee']}]->(AFewGoodMen),
  (JackN)-[:ACTED_IN {roles:['Col. Nathan R. Jessup']}]->(AFewGoodMen),
  (DemiM)-[:ACTED_IN {roles:['Lt. Cdr. JoAnne Galloway']}]-
>(AFewGoodMen),
  (KevinB)-[:ACTED_IN {roles:['Capt. Jack Ross']}]->(AFewGoodMen),
  (KieferS)-[:ACTED_IN {roles:['Lt. Jonathan Kendrick']}]->(AFewGoodMen),
  (NoahW)-[:ACTED_IN {roles:['Cpl. Jeffrey Barnes']}]->(AFewGoodMen),
```

```
  (CubaG)-[:ACTED_IN {roles:['Cpl. Carl Hammaker']}]->(AFewGoodMen),
  (KevinP)-[:ACTED_IN {roles:['Lt. Sam Weinberg']}]->(AFewGoodMen),
  (JTW)-[:ACTED_IN {roles:['Lt. Col. Matthew Andrew Markinson']}]-
>(AFewGoodMen),
  (JamesM)-[:ACTED_IN {roles:['Pfc. Louden Downey']}]->(AFewGoodMen),
  (ChristopherG)-[:ACTED_IN {roles:['Dr. Stone']}]->(AFewGoodMen),
  (AaronS)-[:ACTED_IN {roles:['Man in Bar']}]->(AFewGoodMen),
  (RobR)-[:DIRECTED]->(AFewGoodMen),
  (AaronS)-[:WROTE]->(AFewGoodMen)

CREATE (TopGun:Movie {title:"Top Gun", released:1986, tagline:'I feel the need,
the need for speed.'})
CREATE (KellyM:Person {name:'Kelly McGillis', born:1957})
CREATE (ValK:Person {name:'Val Kilmer', born:1959})
CREATE (AnthonyE:Person {name:'Anthony Edwards', born:1962})
CREATE (TomS:Person {name:'Tom Skerritt', born:1933})
CREATE (MegR:Person {name:'Meg Ryan', born:1961})
CREATE (TonyS:Person {name:'Tony Scott', born:1944})
CREATE (JimC:Person {name:'Jim Cash', born:1941})
CREATE
  (TomC)-[:ACTED_IN {roles:['Maverick']}]->(TopGun),
  (KellyM)-[:ACTED_IN {roles:['Charlie']}]->(TopGun),
  (ValK)-[:ACTED_IN {roles:['Iceman']}]->(TopGun),
  (AnthonyE)-[:ACTED_IN {roles:['Goose']}]->(TopGun),
  (TomS)-[:ACTED_IN {roles:['Viper']}]->(TopGun),
  (MegR)-[:ACTED_IN {roles:['Carole']}]->(TopGun),
  (TonyS)-[:DIRECTED]->(TopGun),
  (JimC)-[:WROTE]->(TopGun)

CREATE (JerryMaguire:Movie {title:'Jerry Maguire', released:2000, tagline:'The
rest of his life begins now.'})
CREATE (ReneeZ:Person {name:'Renee Zellweger', born:1969})
CREATE (KellyP:Person {name:'Kelly Preston', born:1962})
CREATE (JerryO:Person {name:"Jerry O'Connell", born:1974})
CREATE (JayM:Person {name:'Jay Mohr', born:1970})
CREATE (BonnieH:Person {name:'Bonnie Hunt', born:1961})
CREATE (ReginaK:Person {name:'Regina King', born:1971})
CREATE (JonathanL:Person {name:'Jonathan Lipnicki', born:1996})
CREATE (CameronC:Person {name:'Cameron Crowe', born:1957})
CREATE
  (TomC)-[:ACTED_IN {roles:['Jerry Maguire']}]->(JerryMaguire),
  (CubaG)-[:ACTED_IN {roles:['Rod Tidwell']}]->(JerryMaguire),
```

```
  (ReneeZ)-[:ACTED_IN {roles:['Dorothy Boyd']}]->(JerryMaguire),
  (KellyP)-[:ACTED_IN {roles:['Avery Bishop']}]->(JerryMaguire),
  (JerryO)-[:ACTED_IN {roles:['Frank Cushman']}]->(JerryMaguire),
  (JayM)-[:ACTED_IN {roles:['Bob Sugar']}]->(JerryMaguire),
  (BonnieH)-[:ACTED_IN {roles:['Laurel Boyd']}]->(JerryMaguire),
  (ReginaK)-[:ACTED_IN {roles:['Marcee Tidwell']}]->(JerryMaguire),
  (JonathanL)-[:ACTED_IN {roles:['Ray Boyd']}]->(JerryMaguire),
  (CameronC)-[:DIRECTED]->(JerryMaguire),
  (CameronC)-[:PRODUCED]->(JerryMaguire),
  (CameronC)-[:WROTE]->(JerryMaguire)
CREATE (StandByMe:Movie {title:"Stand By Me", released:1986, tagline:"For
some, it's the last real taste of innocence, and the first real taste of life. But for eve-
ryone, it's the time that memories are made of."})
CREATE (RiverP:Person {name:'River Phoenix', born:1970})
CREATE (CoreyF:Person {name:'Corey Feldman', born:1971})
CREATE (WilW:Person {name:'Wil Wheaton', born:1972})
CREATE (JohnC:Person {name:'John Cusack', born:1966})
CREATE (MarshallB:Person {name:'Marshall Bell', born:1942})
CREATE
  (WilW)-[:ACTED_IN {roles:['Gordie Lachance']}]->(StandByMe),
  (RiverP)-[:ACTED_IN {roles:['Chris Chambers']}]->(StandByMe),
  (JerryO)-[:ACTED_IN {roles:['Vern Tessio']}]->(StandByMe),
  (CoreyF)-[:ACTED_IN {roles:['Teddy Duchamp']}]->(StandByMe),
  (JohnC)-[:ACTED_IN {roles:['Denny Lachance']}]->(StandByMe),
  (KieferS)-[:ACTED_IN {roles:['Ace Merrill']}]->(StandByMe),
  (MarshallB)-[:ACTED_IN {roles:['Mr. Lachance']}]->(StandByMe),
  (RobR)-[:DIRECTED]->(StandByMe)

CREATE (AsGoodAsItGets:Movie {title:'As Good as It Gets', released:1997, tag-
line:'A comedy from the heart that goes for the throat.'})
CREATE (HelenH:Person {name:'Helen Hunt', born:1963})
CREATE (GregK:Person {name:'Greg Kinnear', born:1963})
CREATE (JamesB:Person {name:'James L. Brooks', born:1940})
CREATE
  (JackN)-[:ACTED_IN {roles:['Melvin Udall']}]->(AsGoodAsItGets),
  (HelenH)-[:ACTED_IN {roles:['Carol Connelly']}]->(AsGoodAsItGets),
  (GregK)-[:ACTED_IN {roles:['Simon Bishop']}]->(AsGoodAsItGets),
  (CubaG)-[:ACTED_IN {roles:['Frank Sachs']}]->(AsGoodAsItGets),
  (JamesB)-[:DIRECTED]->(AsGoodAsItGets)

CREATE (WhatDreamsMayCome:Movie {title:'What Dreams May Come', re-
leased:1998, tagline:'After life there is more. The end is just the beginning.'})
```

```
CREATE (AnnabellaS:Person {name:'Annabella Sciorra', born:1960})
CREATE (MaxS:Person {name:'Max von Sydow', born:1929})
CREATE (WernerH:Person {name:'Werner Herzog', born:1942})
CREATE (Robin:Person {name:'Robin Williams', born:1951})
CREATE (VincentW:Person {name:'Vincent Ward', born:1956})
CREATE
  (Robin)-[:ACTED_IN {roles:['Chris Nielsen']}]->(WhatDreamsMayCome),
  (CubaG)-[:ACTED_IN {roles:['Albert Lewis']}]->(WhatDreamsMayCome),
  (AnnabellaS)-[:ACTED_IN {roles:['Annie Collins-Nielsen']}]-
>(WhatDreamsMayCome),
  (MaxS)-[:ACTED_IN {roles:['The Tracker']}]->(WhatDreamsMayCome),
  (WernerH)-[:ACTED_IN {roles:['The Face']}]->(WhatDreamsMayCome),
  (VincentW)-[:DIRECTED]->(WhatDreamsMayCome)

CREATE (SnowFallingonCedars:Movie {title:'Snow Falling on Cedars', re-
leased:1999, tagline:'First loves last. Forever.'})
CREATE (EthanH:Person {name:'Ethan Hawke', born:1970})
CREATE (RickY:Person {name:'Rick Yune', born:1971})
CREATE (JamesC:Person {name:'James Cromwell', born:1940})
CREATE (ScottH:Person {name:'Scott Hicks', born:1953})
CREATE
  (EthanH)-[:ACTED_IN {roles:['Ishmael Chambers']}]->(SnowFallingonCedars),
  (RickY)-[:ACTED_IN {roles:['Kazuo Miyamoto']}]->(SnowFallingonCedars),
  (MaxS)-[:ACTED_IN {roles:['Nels Gudmundsson']}]->(SnowFallingonCedars),
  (JamesC)-[:ACTED_IN {roles:['Judge Fielding']}]->(SnowFallingonCedars),
  (ScottH)-[:DIRECTED]->(SnowFallingonCedars)

CREATE (YouveGotMail:Movie {title:"You've Got Mail", released:1998, tag-
line:'At odds in life... in love on-line.'})
CREATE (ParkerP:Person {name:'Parker Posey', born:1968})
CREATE (DaveC:Person {name:'Dave Chappelle', born:1973})
CREATE (SteveZ:Person {name:'Steve Zahn', born:1967})
CREATE (TomH:Person {name:'Tom Hanks', born:1956})
CREATE (NoraE:Person {name:'Nora Ephron', born:1941})
CREATE
  (TomH)-[:ACTED_IN {roles:['Joe Fox']}]->(YouveGotMail),
  (MegR)-[:ACTED_IN {roles:['Kathleen Kelly']}]->(YouveGotMail),
  (GregK)-[:ACTED_IN {roles:['Frank Navasky']}]->(YouveGotMail),
  (ParkerP)-[:ACTED_IN {roles:['Patricia Eden']}]->(YouveGotMail),
  (DaveC)-[:ACTED_IN {roles:['Kevin Jackson']}]->(YouveGotMail),
  (SteveZ)-[:ACTED_IN {roles:['George Pappas']}]->(YouveGotMail),
  (NoraE)-[:DIRECTED]->(YouveGotMail)
```

```
CREATE (SleeplessInSeattle:Movie {title:'Sleepless in Seattle', released:1993, tagline:'What if someone you never met, someone you never saw, someone you never knew was the only someone for you?'})
CREATE (RitaW:Person {name:'Rita Wilson', born:1956})
CREATE (BillPull:Person {name:'Bill Pullman', born:1953})
CREATE (VictorG:Person {name:'Victor Garber', born:1949})
CREATE (RosieO:Person {name:"Rosie O'Donnell", born:1962})
CREATE
  (TomH)-[:ACTED_IN {roles:['Sam Baldwin']}]->(SleeplessInSeattle),
  (MegR)-[:ACTED_IN {roles:['Annie Reed']}]->(SleeplessInSeattle),
  (RitaW)-[:ACTED_IN {roles:['Suzy']}]->(SleeplessInSeattle),
  (BillPull)-[:ACTED_IN {roles:['Walter']}]->(SleeplessInSeattle),
  (VictorG)-[:ACTED_IN {roles:['Greg']}]->(SleeplessInSeattle),
  (RosieO)-[:ACTED_IN {roles:['Becky']}]->(SleeplessInSeattle),
  (NoraE)-[:DIRECTED]->(SleeplessInSeattle)

CREATE (JoeVersustheVolcano:Movie {title:'Joe Versus the Volcano', released:1990, tagline:'A story of love, lava and burning desire.'})
CREATE (JohnS:Person {name:'John Patrick Stanley', born:1950})
CREATE (Nathan:Person {name:'Nathan Lane', born:1956})
CREATE
  (TomH)-[:ACTED_IN {roles:['Joe Banks']}]->(JoeVersustheVolcano),
  (MegR)-[:ACTED_IN {roles:['DeDe', 'Angelica Graynamore', 'Patricia Graynamore']}]->(JoeVersustheVolcano),
  (Nathan)-[:ACTED_IN {roles:['Baw']}]->(JoeVersustheVolcano),
  (JohnS)-[:DIRECTED]->(JoeVersustheVolcano)

CREATE (WhenHarryMetSally:Movie {title:'When Harry Met Sally', released:1998, tagline:'At odds in life... in love on-line.'})
CREATE (BillyC:Person {name:'Billy Crystal', born:1948})
CREATE (CarrieF:Person {name:'Carrie Fisher', born:1956})
CREATE (BrunoK:Person {name:'Bruno Kirby', born:1949})
CREATE
  (BillyC)-[:ACTED_IN {roles:['Harry Burns']}]->(WhenHarryMetSally),
  (MegR)-[:ACTED_IN {roles:['Sally Albright']}]->(WhenHarryMetSally),
  (CarrieF)-[:ACTED_IN {roles:['Marie']}]->(WhenHarryMetSally),
  (BrunoK)-[:ACTED_IN {roles:['Jess']}]->(WhenHarryMetSally),
  (RobR)-[:DIRECTED]->(WhenHarryMetSally),
  (RobR)-[:PRODUCED]->(WhenHarryMetSally),
  (NoraE)-[:PRODUCED]->(WhenHarryMetSally),
  (NoraE)-[:WROTE]->(WhenHarryMetSally)
```

```
CREATE (ThatThingYouDo:Movie {title:'That Thing You Do', released:1996,
tagline:'In every life there comes a time when that thing you dream becomes that
thing you do'})
CREATE (LivT:Person {name:'Liv Tyler', born:1977})
CREATE
  (TomH)-[:ACTED_IN {roles:['Mr. White']}]->(ThatThingYouDo),
  (LivT)-[:ACTED_IN {roles:['Faye Dolan']}]->(ThatThingYouDo),
  (Charlize)-[:ACTED_IN {roles:['Tina']}]->(ThatThingYouDo),
  (TomH)-[:DIRECTED]->(ThatThingYouDo)

CREATE (TheReplacements:Movie {title:'The Replacements', released:2000, tag-
line:'Pain heals, Chicks dig scars... Glory lasts forever'})
CREATE (Brooke:Person {name:'Brooke Langton', born:1970})
CREATE (Gene:Person {name:'Gene Hackman', born:1930})
CREATE (Orlando:Person {name:'Orlando Jones', born:1968})
CREATE (Howard:Person {name:'Howard Deutch', born:1950})
CREATE
  (Keanu)-[:ACTED_IN {roles:['Shane Falco']}]->(TheReplacements),
  (Brooke)-[:ACTED_IN {roles:['Annabelle Farrell']}]->(TheReplacements),
  (Gene)-[:ACTED_IN {roles:['Jimmy McGinty']}]->(TheReplacements),
  (Orlando)-[:ACTED_IN {roles:['Clifford Franklin']}]->(TheReplacements),
  (Howard)-[:DIRECTED]->(TheReplacements)

CREATE (RescueDawn:Movie {title:'RescueDawn', released:2006, tagline:"Based
on the extraordinary true story of one man's fight for freedom"})
CREATE (ChristianB:Person {name:'Christian Bale', born:1974})
CREATE (ZachG:Person {name:'Zach Grenier', born:1954})
CREATE
  (MarshallB)-[:ACTED_IN {roles:['Admiral']}]->(RescueDawn),
  (ChristianB)-[:ACTED_IN {roles:['Dieter Dengler']}]->(RescueDawn),
  (ZachG)-[:ACTED_IN {roles:['Squad Leader']}]->(RescueDawn),
  (SteveZ)-[:ACTED_IN {roles:['Duane']}]->(RescueDawn),
  (WernerH)-[:DIRECTED]->(RescueDawn)

CREATE (TheBirdcage:Movie {title:'The Birdcage', released:1996, tagline:'Come
as you are'})
CREATE (MikeN:Person {name:'Mike Nichols', born:1931})
CREATE
  (Robin)-[:ACTED_IN {roles:['Armand Goldman']}]->(TheBirdcage),
  (Nathan)-[:ACTED_IN {roles:['Albert Goldman']}]->(TheBirdcage),
  (Gene)-[:ACTED_IN {roles:['Sen. Kevin Keeley']}]->(TheBirdcage),
  (MikeN)-[:DIRECTED]->(TheBirdcage)
```

```
CREATE (Unforgiven:Movie {title:'Unforgiven', released:1992, tagline:"It's a hell of a thing, killing a man"})
CREATE (RichardH:Person {name:'Richard Harris', born:1930})
CREATE (ClintE:Person {name:'Clint Eastwood', born:1930})
CREATE
  (RichardH)-[:ACTED_IN {roles:['English Bob']}]->(Unforgiven),
  (ClintE)-[:ACTED_IN {roles:['Bill Munny']}]->(Unforgiven),
  (Gene)-[:ACTED_IN {roles:['Little Bill Daggett']}]->(Unforgiven),
  (ClintE)-[:DIRECTED]->(Unforgiven)
CREATE (JohnnyMnemonic:Movie {title:'Johnny Mnemonic', released:1995, tagline:'The hottest data on earth. In the coolest head in town'})
CREATE (Takeshi:Person {name:'Takeshi Kitano', born:1947})
CREATE (Dina:Person {name:'Dina Meyer', born:1968})
CREATE (IceT:Person {name:'Ice-T', born:1958})
CREATE (RobertL:Person {name:'Robert Longo', born:1953})
CREATE
  (Keanu)-[:ACTED_IN {roles:['Johnny Mnemonic']}]->(JohnnyMnemonic),
  (Takeshi)-[:ACTED_IN {roles:['Takahashi']}]->(JohnnyMnemonic),
  (Dina)-[:ACTED_IN {roles:['Jane']}]->(JohnnyMnemonic),
  (IceT)-[:ACTED_IN {roles:['J-Bone']}]->(JohnnyMnemonic),
  (RobertL)-[:DIRECTED]->(JohnnyMnemonic)

CREATE (CloudAtlas:Movie {title:'Cloud Atlas', released:2012, tagline:'Everything is connected'})
CREATE (HalleB:Person {name:'Halle Berry', born:1966})
CREATE (JimB:Person {name:'Jim Broadbent', born:1949})
CREATE (TomT:Person {name:'Tom Tykwer', born:1965})
CREATE (DavidMitchell:Person {name:'David Mitchell', born:1969})
CREATE (StefanArndt:Person {name:'Stefan Arndt', born:1961})
CREATE
  (TomH)-[:ACTED_IN {roles:['Zachry', 'Dr. Henry Goose', 'Isaac Sachs', 'Dermot Hoggins']}]->(CloudAtlas),
  (Hugo)-[:ACTED_IN {roles:['Bill Smoke', 'Haskell Moore', 'Tadeusz Kesselring', 'Nurse Noakes', 'Boardman Mephi', 'Old Georgie']}]->(CloudAtlas),
  (HalleB)-[:ACTED_IN {roles:['Luisa Rey', 'Jocasta Ayrs', 'Ovid', 'Meronym']}]->(CloudAtlas),
  (JimB)-[:ACTED_IN {roles:['Vyvyan Ayrs', 'Captain Molyneux', 'Timothy Cavendish']}]->(CloudAtlas),
  (TomT)-[:DIRECTED]->(CloudAtlas),
  (AndyW)-[:DIRECTED]->(CloudAtlas),
  (LanaW)-[:DIRECTED]->(CloudAtlas),
  (DavidMitchell)-[:WROTE]->(CloudAtlas),
```

```
  (StefanArndt)-[:PRODUCED]->(CloudAtlas)

CREATE (TheDaVinciCode:Movie {title:'The Da Vinci Code', released:2006, tag-
line:'Break The Codes'})
CREATE (IanM:Person {name:'Ian McKellen', born:1939})
CREATE (AudreyT:Person {name:'Audrey Tautou', born:1976})
CREATE (PaulB:Person {name:'Paul Bettany', born:1971})
CREATE (RonH:Person {name:'Ron Howard', born:1954})
CREATE
  (TomH)-[:ACTED_IN {roles:['Dr. Robert Langdon']}]->(TheDaVinciCode),
  (IanM)-[:ACTED_IN {roles:['Sir Leight Teabing']}]->(TheDaVinciCode),
  (AudreyT)-[:ACTED_IN {roles:['Sophie Neveu']}]->(TheDaVinciCode),
  (PaulB)-[:ACTED_IN {roles:['Silas']}]->(TheDaVinciCode),
  (RonH)-[:DIRECTED]->(TheDaVinciCode)
CREATE (VforVendetta:Movie {title:'V for Vendetta', released:2006, tag-
line:'Freedom! Forever!'})
CREATE (NatalieP:Person {name:'Natalie Portman', born:1981})
CREATE (StephenR:Person {name:'Stephen Rea', born:1946})
CREATE (JohnH:Person {name:'John Hurt', born:1940})
CREATE (BenM:Person {name: 'Ben Miles', born:1967})
CREATE
  (Hugo)-[:ACTED_IN {roles:['V']}]->(VforVendetta),
  (NatalieP)-[:ACTED_IN {roles:['Evey Hammond']}]->(VforVendetta),
  (StephenR)-[:ACTED_IN {roles:['Eric Finch']}]->(VforVendetta),
  (JohnH)-[:ACTED_IN {roles:['High Chancellor Adam Sutler']}]-
>(VforVendetta),
  (BenM)-[:ACTED_IN {roles:['Dascomb']}]->(VforVendetta),
  (JamesM)-[:DIRECTED]->(VforVendetta),
  (AndyW)-[:PRODUCED]->(VforVendetta),
  (LanaW)-[:PRODUCED]->(VforVendetta),
  (JoelS)-[:PRODUCED]->(VforVendetta),
  (AndyW)-[:WROTE]->(VforVendetta),
  (LanaW)-[:WROTE]->(VforVendetta)

CREATE (SpeedRacer:Movie {title:'Speed Racer', released:2008, tagline:'Speed
has no limits'})
CREATE (EmileH:Person {name:'Emile Hirsch', born:1985})
CREATE (JohnG:Person {name:'John Goodman', born:1960})
CREATE (SusanS:Person {name:'Susan Sarandon', born:1946})
CREATE (MatthewF:Person {name:'Matthew Fox', born:1966})
CREATE (ChristinaR:Person {name:'Christina Ricci', born:1980})
CREATE (Rain:Person {name:'Rain', born:1982})
```

```
CREATE
  (EmileH)-[:ACTED_IN {roles:['Speed Racer']}]->(SpeedRacer),
  (JohnG)-[:ACTED_IN {roles:['Pops']}]->(SpeedRacer),
  (SusanS)-[:ACTED_IN {roles:['Mom']}]->(SpeedRacer),
  (MatthewF)-[:ACTED_IN {roles:['Racer X']}]->(SpeedRacer),
  (ChristinaR)-[:ACTED_IN {roles:['Trixie']}]->(SpeedRacer),
  (Rain)-[:ACTED_IN {roles:['Taejo Togokahn']}]->(SpeedRacer),
  (BenM)-[:ACTED_IN {roles:['Cass Jones']}]->(SpeedRacer),
  (AndyW)-[:DIRECTED]->(SpeedRacer),
  (LanaW)-[:DIRECTED]->(SpeedRacer),
  (AndyW)-[:WROTE]->(SpeedRacer),
  (LanaW)-[:WROTE]->(SpeedRacer),
  (JoelS)-[:PRODUCED]->(SpeedRacer)

CREATE (NinjaAssassin:Movie {title:'Ninja Assassin', released:2009, tag-
line:'Prepare to enter a secret world of assassins'})
CREATE (NaomieH:Person {name:'Naomie Harris'})
CREATE
  (Rain)-[:ACTED_IN {roles:['Raizo']}]->(NinjaAssassin),
  (NaomieH)-[:ACTED_IN {roles:['Mika Coretti']}]->(NinjaAssassin),
  (RickY)-[:ACTED_IN {roles:['Takeshi']}]->(NinjaAssassin),
  (BenM)-[:ACTED_IN {roles:['Ryan Maslow']}]->(NinjaAssassin),
  (JamesM)-[:DIRECTED]->(NinjaAssassin),
  (AndyW)-[:PRODUCED]->(NinjaAssassin),
  (LanaW)-[:PRODUCED]->(NinjaAssassin),
  (JoelS)-[:PRODUCED]->(NinjaAssassin)
CREATE (TheGreenMile:Movie {title:'The Green Mile', released:1999, tag-
line:"Walk a mile you'll never forget."})
CREATE (MichaelD:Person {name:'Michael Clarke Duncan', born:1957})
CREATE (DavidM:Person {name:'David Morse', born:1953})
CREATE (SamR:Person {name:'Sam Rockwell', born:1968})
CREATE (GaryS:Person {name:'Gary Sinise', born:1955})
CREATE (PatriciaC:Person {name:'Patricia Clarkson', born:1959})
CREATE (FrankD:Person {name:'Frank Darabont', born:1959})
CREATE
  (TomH)-[:ACTED_IN {roles:['Paul Edgecomb']}]->(TheGreenMile),
  (MichaelD)-[:ACTED_IN {roles:['John Coffey']}]->(TheGreenMile),
  (DavidM)-[:ACTED_IN {roles:['Brutus "Brutal" Howell']}]->(TheGreenMile),
  (BonnieH)-[:ACTED_IN {roles:['Jan Edgecomb']}]->(TheGreenMile),
  (JamesC)-[:ACTED_IN {roles:['Warden Hal Moores']}]->(TheGreenMile),
  (SamR)-[:ACTED_IN {roles:['"Wild Bill" Wharton']}]->(TheGreenMile),
  (GaryS)-[:ACTED_IN {roles:['Burt Hammersmith']}]->(TheGreenMile),
```

```
  (PatriciaC)-[:ACTED_IN {roles:['Melinda Moores']}]->(TheGreenMile),
  (FrankD)-[:DIRECTED]->(TheGreenMile)

CREATE (FrostNixon:Movie {title:'Frost/Nixon', released:2008, tagline:'400 mil-
lion people were waiting for the truth.'})
CREATE (FrankL:Person {name:'Frank Langella', born:1938})
CREATE (MichaelS:Person {name:'Michael Sheen', born:1969})
CREATE (OliverP:Person {name:'Oliver Platt', born:1960})
CREATE
  (FrankL)-[:ACTED_IN {roles:['Richard Nixon']}]->(FrostNixon),
  (MichaelS)-[:ACTED_IN {roles:['David Frost']}]->(FrostNixon),
  (KevinB)-[:ACTED_IN {roles:['Jack Brennan']}]->(FrostNixon),
  (OliverP)-[:ACTED_IN {roles:['Bob Zelnick']}]->(FrostNixon),
  (SamR)-[:ACTED_IN {roles:['James Reston, Jr.']}]->(FrostNixon),
  (RonH)-[:DIRECTED]->(FrostNixon)

CREATE (Hoffa:Movie {title:'Hoffa', released:1992, tagline:"He didn't want law.
He wanted justice."})
CREATE (DannyD:Person {name:'Danny DeVito', born:1944})
CREATE (JohnR:Person {name:'John C. Reilly', born:1965})
CREATE
  (JackN)-[:ACTED_IN {roles:['Hoffa']}]->(Hoffa),
  (DannyD)-[:ACTED_IN {roles:['Robert "Bobby" Ciaro']}]->(Hoffa),
  (JTW)-[:ACTED_IN {roles:['Frank Fitzsimmons']}]->(Hoffa),
  (JohnR)-[:ACTED_IN {roles:['Peter "Pete" Connelly']}]->(Hoffa),
  (DannyD)-[:DIRECTED]->(Hoffa)

CREATE (Apollo13:Movie {title:'Apollo 13', released:1995, tagline:'Houston, we
have a problem.'})
CREATE (EdH:Person {name:'Ed Harris', born:1950})
CREATE (BillPax:Person {name:'Bill Paxton', born:1955})
CREATE
  (TomH)-[:ACTED_IN {roles:['Jim Lovell']}]->(Apollo13),
  (KevinB)-[:ACTED_IN {roles:['Jack Swigert']}]->(Apollo13),
  (EdH)-[:ACTED_IN {roles:['Gene Kranz']}]->(Apollo13),
  (BillPax)-[:ACTED_IN {roles:['Fred Haise']}]->(Apollo13),
  (GaryS)-[:ACTED_IN {roles:['Ken Mattingly']}]->(Apollo13),
  (RonH)-[:DIRECTED]->(Apollo13)
CREATE (Twister:Movie {title:'Twister', released:1996, tagline:"Don't Breathe.
Don't Look Back."})
CREATE (PhilipH:Person {name:'Philip Seymour Hoffman', born:1967})
CREATE (JanB:Person {name:'Jan de Bont', born:1943})
```

```
CREATE
  (BillPax)-[:ACTED_IN {roles:['Bill Harding']}]->(Twister),
  (HelenH)-[:ACTED_IN {roles:['Dr. Jo Harding']}]->(Twister),
  (ZachG)-[:ACTED_IN {roles:['Eddie']}]->(Twister),
  (PhilipH)-[:ACTED_IN {roles:['Dustin "Dusty" Davis']}]->(Twister),
  (JanB)-[:DIRECTED]->(Twister)

CREATE (CastAway:Movie {title:'Cast Away', released:2000, tagline:'At the edge
of the world, his journey begins.'})
CREATE (RobertZ:Person {name:'Robert Zemeckis', born:1951})
CREATE
  (TomH)-[:ACTED_IN {roles:['Chuck Noland']}]->(CastAway),
  (HelenH)-[:ACTED_IN {roles:['Kelly Frears']}]->(CastAway),
  (RobertZ)-[:DIRECTED]->(CastAway)

CREATE (OneFlewOvertheCuckoosNest:Movie {title:"One Flew Over the Cuck-
oo's Nest", released:1975, tagline:"If he's crazy, what does that make you?"})
CREATE (MilosF:Person {name:'Milos Forman', born:1932})
CREATE
  (JackN)-[:ACTED_IN {roles:['Randle McMurphy']}]-
>(OneFlewOvertheCuckoosNest),
  (DannyD)-[:ACTED_IN {roles:['Martini']}]->(OneFlewOvertheCuckoosNest),
  (MilosF)-[:DIRECTED]->(OneFlewOvertheCuckoosNest)

CREATE (SomethingsGottaGive:Movie {title:"Something's Gotta Give", re-
leased:2003})
CREATE (DianeK:Person {name:'Diane Keaton', born:1946})
CREATE (NancyM:Person {name:'Nancy Meyers', born:1949})
CREATE
  (JackN)-[:ACTED_IN {roles:['Harry Sanborn']}]->(SomethingsGottaGive),
  (DianeK)-[:ACTED_IN {roles:['Erica Barry']}]->(SomethingsGottaGive),
  (Keanu)-[:ACTED_IN {roles:['Julian Mercer']}]->(SomethingsGottaGive),
  (NancyM)-[:DIRECTED]->(SomethingsGottaGive),
  (NancyM)-[:PRODUCED]->(SomethingsGottaGive),
  (NancyM)-[:WROTE]->(SomethingsGottaGive)
CREATE (BicentennialMan:Movie {title:'Bicentennial Man', released:1999, tag-
line:"One robot's 200 year journey to become an ordinary man."})
CREATE (ChrisC:Person {name:'Chris Columbus', born:1958})
CREATE
  (Robin)-[:ACTED_IN {roles:['Andrew Marin']}]->(BicentennialMan),
  (OliverP)-[:ACTED_IN {roles:['Rupert Burns']}]->(BicentennialMan),
  (ChrisC)-[:DIRECTED]->(BicentennialMan)
```

```
CREATE (CharlieWilsonsWar:Movie {title:"Charlie Wilson's War", released:2007, tagline:"A stiff drink. A little mascara. A lot of nerve. Who said they couldn't bring down the Soviet empire."})
CREATE (JuliaR:Person {name:'Julia Roberts', born:1967})
CREATE
  (TomH)-[:ACTED_IN {roles:['Rep. Charlie Wilson']}]->(CharlieWilsonsWar),
  (JuliaR)-[:ACTED_IN {roles:['Joanne Herring']}]->(CharlieWilsonsWar),
  (PhilipH)-[:ACTED_IN {roles:['Gust Avrakotos']}]->(CharlieWilsonsWar),
  (MikeN)-[:DIRECTED]->(CharlieWilsonsWar)

CREATE (ThePolarExpress:Movie {title:'The Polar Express', released:2004, tagline:'This Holiday Season… Believe'})
CREATE
  (TomH)-[:ACTED_IN {roles:['Hero Boy', 'Father', 'Conductor', 'Hobo', 'Scrooge', 'Santa Claus']}]->(ThePolarExpress),
  (RobertZ)-[:DIRECTED]->(ThePolarExpress)

CREATE (ALeagueofTheirOwn:Movie {title:'A League of Their Own', released:1992, tagline:'Once in a lifetime you get a chance to do something different.'})
CREATE (Madonna:Person {name:'Madonna', born:1954})
CREATE (GeenaD:Person {name:'Geena Davis', born:1956})
CREATE (LoriP:Person {name:'Lori Petty', born:1963})
CREATE (PennyM:Person {name:'Penny Marshall', born:1943})
CREATE
  (TomH)-[:ACTED_IN {roles:['Jimmy Dugan']}]->(ALeagueofTheirOwn),
  (GeenaD)-[:ACTED_IN {roles:['Dottie Hinson']}]->(ALeagueofTheirOwn),
  (LoriP)-[:ACTED_IN {roles:['Kit Keller']}]->(ALeagueofTheirOwn),
  (RosieO)-[:ACTED_IN {roles:['Doris Murphy']}]->(ALeagueofTheirOwn),
  (Madonna)-[:ACTED_IN {roles:['"All the Way" Mae Mordabito']}]->(ALeagueofTheirOwn),
  (BillPax)-[:ACTED_IN {roles:['Bob Hinson']}]->(ALeagueofTheirOwn),
  (PennyM)-[:DIRECTED]->(ALeagueofTheirOwn)
```

ПРИЛОЖЕНИЕ Б

(рекомендуемое)

Код программы на C/C++, переводящей кириллические символы в латинские

```
#include <fstream>
#include <iostream>
#include <string>
using namespace std;
int main() {
    setlocale(LC_CTYPE, "rus");
    bool find;
    string intext, outtext;
    string rus[66]={"А", "а", "Б", "б", "В", "в", "Г", "г", "Д", "д", "Е", "е", "Ж",
"ж", "З", "з", "И", "и", "Й", "й", "К", "к", "Л", "л", "М", "м", "Н", "н", "О", "о",
"П", "п",  "Р", "р", "С", "с", "Т", "т", "У", "у", "Ф", "ф",  "Х", "х", "Ц", "ц", "Ч",
"ч",  "Ш", "ш", "Щ", "щ", "Ь", "ь", "Ю", "ю",  "Я", "я", "Ы", "ы", "Ъ", "ъ", "Ё",
"ё", "Э", "э"};
    string eng[66]={"A", "a", "B", "b", "V", "v", "G", "g", "D", "d", "E", "e", "Zh",
"zh", "Z", "z", "I", "i", "J", "j", "K", "k",  "L", "l", "M", "m", "N", "n", "O", "o",
"P", "p",  "R", "r", "S", "s", "T", "t", "U", "u", "F", "f",  "H", "h", "Ts", "ts", "Ch",
"ch", "Sh", "sh",  "Shh", "shh", "", "", "Yu", "yu","Ya", "ya", "I", "i", "", "", "Yo",
"yo", "E", "e"};
    string files[6]={"country.csv","country1.csv","region.csv","region1.csv",
"city.csv", "city1.csv"};
    for (int k=0; k<3;k++) {
        ifstream filein (files[2*k].c_str());
        ofstream fileout (files[2*k+1].c_str());
        if(filein.is_open()) {
```

Окончание ПРИЛОЖЕНИЯ Б

```
        ifstream filein;
        filein.open(files[2*k].c_str());
        while (getline(filein, intext)) {
                outtext = "";
                for(int i=0; i<=intext.length(); i++) {
                        find=false;
                        for(int j=0;j<66;j++) {
                                if( intext.substr(i,1).compare(rus[j])==0 ) { out-
                                text+=eng[j]; find=true;  break;}
                        }
                        if( ! find ) outtext+=intext.substr(i,1);
                }
          }
          filein.close();
          fileout.close();
     }
     else {cout << "File not found" << endl;}
  }
  return 0;
}
```

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
"НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
МОРДОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМ. Н.П. ОГАРЁВА"

ОТЗЫВ РЕЦЕНЗЕНТА
О ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ

Студента Чикляев Константина Валерьевича
(Фамилия, имя, отчество)

Факультет математики и ИТ
Кафедра фундаментальной информатики Группа 402
Направление подготовки 02.03.02. Фунд. инф-ка и ИТ
Квалификация (степень) бакалавр

Наименование темы: Практическое применение NoSQL баз данных

Рецензент Сухарев А.А., канд. алгебры и геометрии, к.ф.-м.н.
(Фамилия И.О., место работы, ученое звание, степень)

ОЦЕНКА ВЫПУСКНОЙ РАБОТЫ

№ п/п	Показатели		Оценка				
			5	4	3	2	0*
1.	Актуальность тематики работы		✓				
2.	Степень полноты обзора состояния вопроса и корректность постановки задачи		✓				
3.	Уровень и корректность использования в работе методов исследования, математического моделирования			✓			
4.	Степень комплексности работы, применение в ней знаний естественнонаучных, социально-экономических, общепрофессиональных и специальных дисциплин		✓				
5.	Ясность, четкость, последовательность и обоснованность изложения		✓				
6.	Применение современного математического и программного обеспечения, компьютерных технологий в работе		✓				
7.	Качество оформления (общий уровень грамотности, стиль изложения, качество иллюстраций, соответствие требованиям стандарта)			✓			
8.	Оригинальность и новизна полученных результатов (научных, конструкторских и технологических решений)		✓				
9.	Тип работы	фундаментальная с оригинальными результатами					
		реферативная					
		прикладная	✓				
10.	Рекомендации	к опубликованию					
		к внедрению					
ИТОГОВАЯ ОЦЕНКА			отлично				

* – не оценивается (трудно оценить)

Отмеченные достоинства: 1) В работе проведен достаточно полный сравнительный анализ средств создания и управления NoSQL баз данных, которые наиболее широко распространены в настоящее время. Приведен характерные примеры построения запросов по различным параметрам нужных сведений из таких баз данных.
2) Работа изложена ясно и четко, прослеживается логическая связь между разделами. Необходимые пояснения и примеры баз данных вынесены в приложения, что делает работу доступной для понимания.
3) Автором приведен новые результаты в виде анализа и обработки баз данных Neo4j средствами декларативного языка запросов Cypher.

Отмеченные недостатки: Визуализация сведений, встроенная в рассматриваемую в разделе 3 базу данных, отражает не все связи между узлами (объектами) базы данных и отличается от общепринятых стандартов.

Заключение: Указанных недостатков не умаляет достоинств работы, которая удовлетворяет всем требованиям, предъявляемым к бакалаврским работам; работа заслуживает оценки "отлично", а её автор, Ушаев Константин Валерьевич, заслуживает присвоения квалификации (степени) Бакалавр по направлению подготовки "Управление в технических системах и информационные технологии"

14 июня 2016 г. Рецензент _____
(подпись)

Отзыв на бакалаврскую работу
«Практическое применение NoSQL баз данных»

студента 4 курса дневного отделения
факультета математики и информационных технологий
К.В. Чиняева

Хранилища информации в виде компьютерных баз данных в настоящее время используются практически во всех областях человеческой деятельности. Для разработчиков приложений баз данных важное значение имеет выбор подходящей СУБД, каждая из которых поддерживает определенную логическую и физическую модель данных. Традиционно в течение нескольких десятилетий для большинства задач использовалась реляционная модель и язык запросов SQL, однако в последнее время появилось много СУБД, работающих с другими моделями (так называемые NoSQL базы данных). В бакалаврской работе необходимо было теоретически изучить различные нереляционные модели и на основе практических примеров провести сравнительный анализ NoSQL СУБД с традиционными реляционными СУБД.

В целом бакалаврская работа соответствует заявленной теме и раскрывает исследуемые технологии. В ходе выполнения бакалаврской работы К.В. Чиняев показал себя грамотным специалистом, способным самостоятельно формулировать и решать поставленные задачи.

Считаю, что бакалаврская работа К.В. Чиняева заслуживает оценки "отлично".

Руководитель дипломной работы
к.ф. – м.н., доцент кафедры
фундаментальной информатики А.В. Попов