**АНОТАЦИЯ**

СОДЕРЖАНИЕ

[ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ 8](#_Toc481790980)

[ВВЕДЕНИЕ 9](#_Toc481790981)

[1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ 10](#_Toc481790982)

[1.1. Определение проекта 10](#_Toc481790983)

[1.2. Жизненный цикл проекта 11](#_Toc481790984)

[1.3. Системы управления проектами 13](#_Toc481790985)

[1.4. SQL и NoSQL решения 14](#_Toc481790986)

[1.5. Графовые СУБД 19](#_Toc481790987)

[1.5.1. Механизмы вычисления графов 21](#_Toc481790988)

[1.5.2. Преимущества графовых баз данных 22](#_Toc481790989)

[1.5.3. Графовая СУБД Neo4j 24](#_Toc481790990)

[2. ИСПОЛЬЗУЕМЫЕ СТАНДАРТЫ, ТЕХНОЛОГИИ, ИНТСРУМЕНТЫ 28](#_Toc481790991)

[2.1. Сравнение MySQL и Neo4j 28](#_Toc481790992)

[2.2. Язык запросов Cypher 30](#_Toc481790993)

[3. РЕАЛИЗАЦИЯ 35](#_Toc481790994)

[3.1. Разработка графовой базы данных на СУБД Neo4j 35](#_Toc481790995)

[3.2. Создание скриптов для заполнения графовой БД 38](#_Toc481790996)

[3.3. Создание запросов на языке Cypher 38](#_Toc481790997)

[4. ОХРАНА ТРУДА 39](#_Toc481790998)

[5. ОБОСНОВАНИЕ ЭКОНОМИЧЕСКОЙ ЭФФЕКТИВНОСТИ 40](#_Toc481790999)

[ЗАКЛЮЧЕНИЕ 41](#_Toc481791000)

[БИБЛИОГРАФИЧЕСКИЙ СПИСОК 42](#_Toc481791001)

[ПРИЛОЖЕНИЕ А 45](#_Toc481791002)

[ПРИЛОЖЕНИЕ Б 46](#_Toc481791003)

# ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ

ВКР – выпускная квалификационная работа

ЖЦП – жизненный цикл проекта

ПО – программное обеспечение

ЖЦ – жизненный цикл

ТЭО – технико-экономическое обоснование

БД – база данных

СУБД – система управления базами данных

ЭВМ – электронно-вычислительная машина

ОС – операционная система

СУП – система управления проектами

OLTP – Online Transaction Processing – обработка транзакций в реальном времени

RDF – Resource Description Framework – среда описания ресурса

MapReduce – модель распределенных вычислений

NoSQL – Not only SQL – не только SQL

SQL – Structured Query Language – язык структурированных запросов

OLAP – Online Analytical Processing – интерактивная аналитическая обработка

SOR – System of Record – система записи

IoT – Internet of Things – методология вычислительной сети физических предметов

ACID – Atomicity, Consistency, Isolation, Durability – набор свойств гарантирующих надежную работу транзакций: атомарность, согласованность, изолированность, долговечность

URL – Uniform Resource Locator – единообразный локатор ресурса

# ВВЕДЕНИЕ

# АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

## Определение проекта

Существует ряд определений понятия “проект”, каждое из которых имеет право на существование.

В США, Институте управления проектами, проект трактуется как “некоторое предприятие с изначально установленными целями, достижение которых определяет завершение проекта”. В английской ассоциации проект менеджеров: “проект – это отдельное предприятие с определенными целями, часто включающими требования по времени, стоимости и качеству достигаемых результатов”. По “Оперативному руководству” №2.20 мирового банка, проекта определяется как “комплекс взаимосвязанных мероприятий, предназначенных для достижения, в течение заданного периода времени и при установленном бюджете, поставленных задач с четко определенными целями”.

На основании определений приведенных выше, можно выделить общие признаки проекта:

* Изменения – основное содержание проекта,
* Каждая цель или задача имеет ограниченное время,
* Проект не может длиться бесконечно, то есть, ограничен во времени,
* Бюджет проекта,
* Ограниченность в необходимых ресурсах,
* Новизна,
* Комплексность,
* Правовое и организационное обеспечение проекта,
* Границы с другими видами деятельности и намерениями. [1]

В общем понимании, проект – нечто уникальное, новое, имеющее определенный набор задач, ограничен во времени и ресурсах, имеет четкие границы и установленный бюджет.

## Жизненный цикл проекта

Для того что бы понять природу работы над проектом, необходимо описать его жизненный цикл, смысл которого вполне понятен. Проект имеет свое начало и свой конец, а также периоды роста, стабильности и спада. В простейшем случае ЖЦП включает в себя начальный, промежуточный и заключительный этапы, фазы. [2]

Фазы, этапы жизненного цикла проекта включают в себя выполнение основных мероприятий по проекту:

* разработка ТЭО и рабочего проекта,
* контрактная деятельность,
* организация и финансирование работ,
* создание новых технологий,
* планирование ресурсов и хода работ,
* закупка материалов и оборудования,
* выполнение работ и сдача готовых объектов.

Таким образом, исходя из основных мероприятий по проекту, строятся фазы ЖЦП (рис. 1).



Рисунок 1. Основные фазы жизненного цикла проекта

Фазы проекта – это отдельные части в рамках проекта, требующие дополнительного контроля для эффективного получения основного результата проекта, они обычно выполняются последовательно, но в некоторых проектных ситуациях могут перекрываться. Высокоуровневый характер фаз проекта превращает их в элемент жизненного цикла проекта. Фаза проекта не является группой процесса управления проектом.

Структура фаз позволяет разделить проект на логические подгруппы для более легкого управления, планирования и контроля. Количество фаз, необходимость в них и степень налагаемого контроля зависит от размера фаз, сложности и потенциального влияния на проект. [3]

Один из важных моментов, характеризующий ЖЦП, является нарастание трудозатрат по фазам жизненного цикла. Типичный вид трудозатрат показан на рисунке 2, будем считать, что на каждую фазу дается одинаковое количество времени. [4]

Рисунок 2. Диаграмма распределение трудозатрат по фазам ЖЦ

Проект часто начинается с идеи, которая появляется у одного человека. Постепенно, по мере формулирования, анализа и оценки этой идеи, привлекаются дополнительные специалисты. Еще больше участников требуется на фазе разработки проекта. Пик трудозатрат приходится на фазу реализации проекта.

На последнем этапе происходит постепенное высвобождение участников проектной команды. Следует помнить, что проект должен иметь четкое окончание во времени, после которого все работы по проекту закрываются, и на проект перестают тратиться ресурсы. [5]

## Системы управления проектами

Система управления проектами – это совокупность инструментов или методов, которые используются для управления проектами. Система может быть как формальной, так и нет, и помогает менеджеру проекта эффективно управлять и завершать его. СУП – это ряд процессов и связанных с ними функций контроля, объединённых в целостную структуру.

СУП строится на основе плана управления проектом, который описывает то, как будет использоваться система. Содержание СУП изменяется в зависимости от области приложения, особенностей организации, сложности проекта и доступности необходимых ресурсов. Система строиться так, чтобы максимально соответствовать стратегическим целям и производственным ресурсам клиентской организации. [6]

Согласно исходному перечню систем управления проектами, в текущей работе рассматриваются такие системы как:

* GitHub,
* Bitbucket,
* Jira.

Система GitHub – это веб сервис, созданный для разработки программного обеспечения. Позиционирует себя как огромную социальную сеть для разработчиков с открытым исходным кодом проектов. Сервис GitHub является бесплатным для публичных проектов, и насчитывает более 12 миллионов человек участвующих в 31 миллионе проектов. [7]

Сервис Bitbucket – распределенная система контроля версий, созданная для профессиональных команд. Она упрощает совместную работу команды над проектом благодаря гибким моделям развертывания. Единственное Git-решение, которое хорошо масштабируется и подходил для команд любых размеров. Также как и GitHub, является бесплатным для публичных проектов с открытым исходным кодом. [8]

Система Jira – это коммерческая, настраиваемая система отслеживания ошибок для управления проектами. Может использоваться для поддержки клиентов, поддерживает командную работу в режиме реального времени, предоставляет высокий уровень безопасности, API и множества плагинов. Поддерживает email-интеграцию и русский интерфейс. [9]

## SQL и NoSQL решения

Приведем некоторые ключевые концепции реляционных и нереляционных баз данных. На рисунке 3 показана база данных, содержащая сведения о взаимоотношениях людей. Первый вариант – это бессхемная структура, построенная в виде графа, характерная для NoSQL-решений, второй вариант – представление тех же данных в структурированном виде, типичном для SQL.



Рисунок 3. Два варианта структур представления данных

Бессхемность означает, что двум документам в структуре данных NoSQL не обязательно иметь одинаковые поля, и они могут хранить данные разных типов. Например, массив объектов, набор полей которых не совпадает:

|  |
| --- |
| “var cars = [{Model: “BMW”, Color: “Red”, Manufactured: 2016}, {Model: “Mercedes”, Type: “Coupe”, Color: “Black”, Manufactured: “1-2-2017”}];” |

При реляционном подходе данные надо хранить в заранее спроектированной структуре, из которой эти данные потом можно будет извлечь. [10]

Категория баз данных NoSQL заметно отличается от SQL БД. Базы данных NoSQL часто используется для описания систем управления данными, которые не относятся к SQL, или подхода к управлению данными, который предусматривает использование не только SQL. Существует ряд технологий категории NoSQL, включая базы данных документов, хранилища пар “ключ – значение”, хранилища семейств-столбцов, а также графовые базы данных, которые часто используются в играх, приложениях для работы с социальными сетями и приложениях IoT.

В таблице 1 приведены основные различия между SQL и NoSQL базами данных. [11]

Таблица 1. Основные различия SQL и NoSQL

|  |  |  |
| --- | --- | --- |
| Характеристики | NoSQL | SQL |
| Модель | Хранит данные в документах JSON, парах ключ/значение, хранилищах семейств-столбцов и графах | Хранит данные в таблице |
| Данные | Предлагает гибкость, поскольку не каждой записи нужно хранить те же свойства | Отлично подходит для решений, где каждая запись имеет одинаковые свойства |
| Новые свойства могут быть добавлены на лету | Добавление нового свойства может потребовать изменение схем или данных обратной засылки |
| Отношения часто захватываются путем денормализации данных и представления всех данных для объекта в одной записи | Отношения часто захватываются в нормализованной модели с использованием объединений для разрешения ссылок между таблицами |

Продолжение таблицы 1

|  |  |  |
| --- | --- | --- |
|  | Хорошо подходит для полу структурированных, сложных или вложенных данных | Хорошо подходит для структурированных данных |
| Схема | Динамические или гибкие схемы | Строгая схема |
| БД является схемой-агностиком, и схема диктуется приложением. Это обеспечивает гибкость и высокую итеративную разработку | Схема должна поддерживаться и храниться в синхронизации между приложением и БД |
| Транзакции | Поддержка ACID зависит от решения | Поддерживает транзакции ACID |
| Последовательность и доступность | Возможна поддержка сильной согласованности в зависимости от решения | Обеспечение жесткой последовательности |
| Согласованность, доступность и производительность могут предоставляться в соответствии с потребностями приложения | Приоритетность согласованности над доступностью и производительностью |

Продолжение таблицы 1

|  |  |  |
| --- | --- | --- |
| Производительность | Производительность может быть увеличена за счет уменьшения согласованности, если это необходимо | Производительность вставки и обновления зависит от того, насколько быстро выполняется запись, так как обеспечивается сильная согласованность. Производительность может быть увеличена с помощью масштабирования доступных ресурсов и использования структур в памяти |
| Вся информация об объекте обычно находится в одной записи, поэтому обновление может произойти за одну операцию | Информация об объекте может быть распределена по многим таблицам или строкам, для чего требуется много соединений для завершения обновления или запроса |

В зависимости от того что необходимо разрабатывать либо исходя из требований к приложению или системе, можно выбрать одно из имеющихся решений. В некоторых случаях применяются оба подхода для еще более гибкой и производительной работы.

## Графовые СУБД

Графовая база данных – NoSQL решение, разновидность баз данных с реализацией сетевой модели в виде графа и его обобщений с использованием графовой СУБД, которая дает новые возможности для работы со связанными данными. Графовую модель данных обычно рассматривают как обобщение RDF – модели или сетевой модели данных. [12]

Система управления графовыми базами данных (графовые базы данных) поддерживает методы создания, чтения, изменения, и удаления, основанные на графовой модели данных. Графовые базы данных, как правило, поддерживают систему транзакций реального времени (OLTP). Соответственно, они оптимизированы для выполнения транзакций и спроектированы с учетом транзакционной целостности и оперативности.

Имеются две особенности графовых баз данных, которые необходимо учитывать при рассмотрении применяемой ими технологии:

1. принцип хранения. Некоторые графовые базы данных используют специализированные хранилища графов, предназначенные и оптимизированные для хранения и обработки именно графов. Но такую технологию хранения используют не все графовые базы данных. Некоторые сериализуют графы и размещают их в реляционной, объектно-ориентированной или какой-то другой базе данных или хранилище;
2. порядок обработки. Некоторые определения требуют, чтобы графовая база данных использовала смежность без индексов, то есть физическое соединение друг с другом.

Взаимосвязи в графовой модели данных являются гражданами первого сорта. Здесь к ним относятся не так, как в других системах управления базами данных, где для отображения взаимосвязей применяются такие механизмы, как внешние ключи или внешние операции, например MapReduce. Собирая абстракции узлов и взаимосвязей в связанные структуры, графовая база данных позволяет строить модели любой сложности, лучше всего отражающие предметную область. Полученные модели проще и в то же время нагляднее, чем те, что создаются с помощью традиционных реляционных баз данных или других NOSQL-хранилищ. [13]

Самые популярные на сегодняшний день графовые СУБД:

* Neo4j,
* HyperGraphDB,
* ArangoDB,
* FlockDB,
* Giraph,
* OrientDB,
* Infinite Graph.

Фундаментальная модель данных графовых баз очень проста: узлы, соединенные ребрами (дугами). Помимо этой существенной характеристики, существуют много вариаций в моделях данных – в частности, в том, какие механизмы используются для хранения вершин и ребер. Например, база FlockDB, представляет собой простую совокупность узлов и ребер без какого-либо механизма для дополнительных атрибутов, Neo4j позволяет присоединять Java – объекты в качестве свойств узлов и ребер в неструктурированном виде, а Infinite Graph хранит Java-объекты, являющиеся экземплярами подклассов таких встроенных типов, как узлы и ребра.

На рисунке 4 показан пример модели графовой базы данных с маленькими узлами и многочисленными связями между ними.



Рисунок 4. Пример модели графовой базы данных

Работая с этой моделью, мы имеем возможность задавать вопросы вроде “найти книгу в категории “Базы данных”, написанную кем-то, чей друг мне нравится”. [14]

### Механизмы вычисления графов

Такие механизмы позволяют выполнять глобальные графовые вычислительные алгоритмы для больших наборов данных. Они предназначены для решения таких задач, как идентификация кластеров данных или получение ответов на такие вопросы, как: “Сколько всего взаимосвязей, сколько их в среднем, полна ли социальная сеть?”

Из-за своей направленности на глобальные запросы механизмы вычисления графов, как правило, оптимизированы для сканирования и пакетной обработки больших объёмов информации, и в этом отношении они похожи на другие технологии пакетного анализа, такие как интеллектуальный анализ данных или аналитическая обработка в реальном времени (OLAP), используемые в реляционном мире. [15]

Некоторые механизмы вычисления включают в себя и средства хранения графов, а другие (большинство) заботятся только об обработке данных, получаемых из внешнего источника, а затем возвращают результаты для сохранения в другом месте. Рисунок 5 демонстрирует типовую архитектуру развертывания механизмов вычисления графов.



Рисунок 5. Типовая архитектура развертывания механизмов вычисления графов

Схема включает в себя систему записи (SOR) базы данных со свойствами OLTP (к примеру, Oracle или Neo4j), которая обслуживает запросы и отвечает на запросы, поступающие от приложения и, в конечном счете, от пользователей. Задания на извлечения, преобразования и загрузку данных перемещают данные из системы записи базы данных, в механизм вычисления графов для выполнения анализа и автономных запросов. [16]

### Преимущества графовых баз данных

Практический любою деятельность, будь то план маршрута похода в магазин или же разработку атомной электростанции можно представить в виде графа но, к сожалению, мы живем в мире, где правят жесткие правила и стандарты, ограниченные бюджеты, установленные сроки для выполнения задач или реализацию проектов. Предоставляемый графовыми базами данных новый способ моделирования данных сам по себе не дает достаточного основания для замены давно устоявшихся и понятных платформ обработки данных. Этот способ должен давать незамедлительную и очень значительную практическую пользу. Мотивация перехода на графовые базы данных заключается в том, что при определенной модели данных такой переход будет давать существенное увеличение производительности на один и более порядков.

Ощутимый прирост производительности при использовании графовых баз данных достигается, если работа ведется с взаимосвязанными данными, по сравнению с теми же реляционными базами данных или NoSQL-хранилищами. В отличие от реляционных БД, где учет взаимосвязей большого объёма данных ощутимо ухудшает производительность запросов, производительность графовых БД при росте объёма данных остается неизменной. Это связана с тем фактом, что запросы в графовой БД локализуются в определенной части графа. В итоге время выполнения каждого запроса зависит только от размера части графа, в которой происходит поиск, а не от общего его размера.

Помимо роста в производительности, графовые базы данных предоставляют гибкую модель данных и способ развертывания, который соответствует современным способам развертывания ПО. Структура данных должна соответствовать изменяющимся потребностям, а не навязываться заранее и оставаться неизменной. В графовых БД эта задача легко решается, графовая модель данных учитывает потребности бизнеса, что и дает ей возможность изменяться со скоростью изменения самого бизнеса.

Возможность расширения означает, что можно добавлять и дополнять новые виды взаимосвязей, новые узлы, новые метки, а также новые подграфы в существующую структуру и что самое важное – это происходит без нарушения существующих запросов и функционала приложения. Такая возможность положительно влияет на производительность процесса разработки и снижает риски для проекта. Благодаря гибкости графовой БД, нет необходимости заранее моделировать задачу в мельчайших подробностях, что очень неудобно, поскольку в бизнесе требования очень часто меняются. Способность графов к расширению также позволяет уменьшить количество миграций, это снижает нагрузку при обслуживании данных и уменьшает риск потерь. [13, 14]

### Графовая СУБД Neo4j

Исходя из исходных данных, для разработки графовой базы данных, в этой работе используется графовая СУБД Neo4j.

СУБД Neo4j – это бесплатная база данных NoSQL с открытым исходным кодом, которая реализованная на языках Java и Scala. СУБД Neo4j используется сегодня сотнями тысяч организаций, практический во всех отраслях, таких как аналитика программного обеспечения, научные исследования, маршрутизация, организационное и проектное управление, социальные сети и во многих других.

В графовой базе данных Neo4j есть одно основное правило: “Нет потерянных ссылок”. Поскольку отношение всегда имеет начальный и конечный узлы, нельзя удалить узел, не удалив связанные с ним отношения. Также существующее отношение никогда не укажет на несуществующую конечную точку (узел). На рисунке 6 изображен пример графовой модели данных Neo4j.



Рисунок 6. Пример графовой модели данных Neo4j

СУБД Neo4j эффективно реализует графовую модель вплоть до уровня хранилища. В отличие от обработки графики или библиотек в памяти, Neo4j обладает полными характеристиками базы данных, включая соответствие транзакциям ACID, поддержку кластера и переключение при выполнении во время выполнения, что позволяет использовать данные графа в производственных сценариях.

Некоторые особенности (рис.7) делают Neo4j очень популярной среди пользователей, разработчиков и администраторов баз данных:

* Материализация отношений во время создания, не приводящая к увеличению времени выполнения сложного запроса,
* Константное время обхода связей графа, как в глубину, так и в ширину благодаря эффективному представлению узлов и отношений,
* Все отношения в Neo4j одинаково важны и быстры, что позволяет впоследствии материализовать и использовать новые отношения для “сокращения” и ускорить получение данных домена при возникновении новых потребностей,
* Компактное хранение и кэширование, которое приводит к эффективному масштабированию и миллиардам узлов в одной базе данных на среднем оборудовании,
* Написано поверх JVM.



Рисунок 7. Особенности Neo4j

Помимо этого, в графовой СУБД Neo4j присутствует реализация графического интерфейса, это дает возможность удобно просматривать графовые данные, которые хранятся в базе. Пример графического интерфейса для изображения данных в Neo4j представлен на рисунке 8.



Рисунок 8. Графический интерфейс изображения данных в Neo4j

Neo4j распространяется в двух версиях: версия для свободного пользования – высокопроизводительная, полностью ACID-транзакционная база данных и корпоративная версия – включает в себя (но не ограничивается) всю функциональность версии для свободного пользования в дополнение к масштабируемой кластеризации, отказоустойчивости, высокой готовности, оперативной архивации и всестороннему мониторингу. [17]

# ИСПОЛЬЗУЕМЫЕ СТАНДАРТЫ, ТЕХНОЛОГИИ, ИНТСРУМЕНТЫ

## Сравнение MySQL и Neo4j

Что бы сравнить производительность баз данных необходимо наполнить их одними и теми же данными, при этом объём данных должен быть существенный, так как при малом объеме разница будет неощутима. Исходя из требований, была выбрана тестовая предметная область – социальная сеть и составлена ER – диаграмма (рис.9), на основе которой были созданы реляционная и графовая модели.



Рисунок 9. ER – диаграмма реляционной и графовой модели

После того как обе БД были наполнены следующим количеством данных:

* 100 000 users,
* 200 000 groups,
* 300 000 photos,
* 250 000 audios,
* 1 000 000 friends,
* 4 000 000 messages,
* 350 000 user audios,
* 400 000 user groups,
* 400 000 user photos,

Размер базы данных MySQL составил 351.5 мегабайт, а размер БД Neo4j – 3.45 гигабайт. Размер в объеме между базами данных довольно ощутим, это связано с тем, что было использовано много полей с текстовой информацией. Эксперимент проводился на ЭВМ со следующими конфигурациями:

* Операционная система – Windows 8.1,
* Тип системы – 64-разрядная ОС,
* Процессор – Intel ® Core ™ i5 – 3230M CPU @ 2.60 GHz,
* Оперативная память (ОЗУ) – 6.00 Гигабайт,
* Жесткий диск – HGST SAS 3.0 900 Гигабайт.

Эксперимент заключался в измерении времени поиска общего количества фотографии у пользователей, которые администрируют хотя бы одну группу, в зависимости от диапазона значений идентификаторов пользователя. Поскольку такой поиск осуществляется по общим взаимосвязям, он является сложным и отражает более наглядное время поиска для сравнения.

Для того что бы результаты поиска были максимально точными, эксперимент проводился при условии что базы данных MySQL и Neo4j не работали одновременно на одной ЭВМ.

Результаты проведенного эксперимента отражены в таблице 2.

Таблица 2. Результаты измерения времени поиска

|  |  |  |
| --- | --- | --- |
| Диапазон идентификатора < | Время для MySQL, мс | Время для Neo4j, мс |
| 10 | 120 | 10767 |
| 100 | 690 | 10706 |
| 1000 | 5879 | 10884 |
| 5000 | 24668 | 12245 |
| 10000 | 52462 | 12280 |

Продолжение таблицы 2

|  |  |  |
| --- | --- | --- |
| 30000 | 154534 | 13352 |
| 60000 | 296369 | 14545 |
| 90000 | 489830 | 18058 |

По результатам таблицы 2 была построена диаграмма зависимостей (рис. 10) для проведенного эксперимента.

Рисунок 10. Диаграмма измерения времени поиска

Проведенный эксперимент показал, что графовая база данных производительнее по времени поиска, когда объём данных большой, но при этом занимает приличное место на жестком диске в отличие от реляционной БД. Также можно сказать, что для систем с небольшим объёмом данных лучше использовать реляционную базу данных, так как при таких условиях она будет справляться лучше. Эксперимент и его результаты были взяты из источника [18].

## Язык запросов Cypher

Cypher – это декларативный графовый язык запросов, который позволяет выразительно и эффективно строить запросы и обновлять графовые хранилища. Относительно простой, но все же очень мощный язык. Очень сложные запросы к базам данных могут быть легко выражены через Cypher. Его конструкции основаны на английской прозе и аккуратной иконографии, которая помогает сделать запросы более понятными.

Будучи декларативным языком, Cypher сосредоточен на ясности выражения того, что извлекать из графа, а не та том, как это извлекать. Он отличается от императивных языков, таких как Java, скриптовых языков, таких как Gremlin, и привязок JRuby. Такой подход делает оптимизацию запросов деталями реализации, а не обременяет пользователя ею и требует от нее обновления всех обходов только потому, что изменилась физическая структура базы данных (новые индексы и т. д.).

Cypher вдохновлен рядом различных подходов и опирается на сложившиеся практики выразительного запроса. Большинство ключевых слов, таких как WHERE и ORDER BY, заимствованы из SQL. Совмещение с образцом заимствует подходы к выражению из SPARQL. Некоторые из выражений были заимствованы из таких языков, как Haskell и Python. [19]

Cypher предоставляет пользователю (или приложению, действующему от имени пользователя) возможность задавать шаблон для поиска данных. Проще говоря, можно попросить базу данных “найти что-то похожее на это”. Для описания того, “как это должно выглядеть” используется ASCII-графика. На рисунке 11 изображен шаблон, который описывает трех друзей.



Рисунок 11. Модель графа, изображенная схемой

А вот его эквивалент на Cypher, представленный ASCII-графикой:

|  |
| --- |
| (emil: Person {name: ‘Emil’}) <- [:KNOWS] – (jim: Person {name: ‘Jim’})  – [:KNOWS] -> (ian: Person {name: ‘Ian’})  – [:KNOWS] -> (emil) |

Этот шаблон описывает маршрут, соединяющий узел с именем jim с двумя другими узлами ian и emil, которые также связаны соединением, идущим от узла ian к узлу emil. ian, jim и emil являются идентификаторами. Идентификаторы позволяют ссылаться на узлы в описании модели и обойти тот факт, что язык запросов имеет только одно измерение (его текст записывается слева направо), в то время как схема графа использует два измерения.

Шаблоны графов, представленные ASCII-графикой, являются основой Cypher. Запросы на Cypher прикрепляют одну или несколько частей шаблона к определенным местам графа с помощью предикатов, а затем перемещают незафиксированные части, пытаясь найти несоответствие.

Как и в большинстве языков запросов, Cypher состоит из фраз. В таблице 3 приведено описание фраз языка Cypher, наиболее часто используемых в запросах.

Таблица 3. Фразы языка запросов Cypher

|  |  |
| --- | --- |
| Фраза | Описание |
| MATCH | Основа большинства Cypher запросов. Шаблон для сопоставления и самый распространенный способ для получения данных из графа |
| RETURN | Определяет, какие узлы, взаимосвязи, и свойства в совпавших данных должны быть возвращены клиенту |
| WHERE | Определяет критерии совпадения результатов для фильтрации шаблона |
| CREATE и CREATE UNIQUE | Создает узлы и взаимосвязи |
| MERGE | Гарантирует, что заданный шаблон будет существовать в графе либо за счет использования уже найденных в графе узлов и взаимосвязей, соответствующих заданным предикатам, либо за счет создания новых узлов и взаимосвязей |
| DELETE | Удаляет узлы, взаимосвязи и свойства |
| SET | Устанавливает значение свойств |
| FOREACH | Вносит изменения в каждый из элементов списка |
| UNION | Объединяет результаты двух или более запросов |
| WITH | Объединяет части запроса в цепочку и передает результаты от одной части запроса к другой. Работает подобно именованным каналам в Unix |

Продолжение таблицы 3

|  |  |
| --- | --- |
| START | Явно указывает одну или несколько отправных точек, узлов или взаимосвязей в графе. (Фраза START признана устаревшей, и вместо нее рекомендуется явно указывать отправные точки во фразе MATCH) |
| LIMIT | Ограничивает количество строк в выводе |

Пример использования нескольких фраз в одном запросе языка Cypher:

|  |
| --- |
| MATCH (a: Person) – [:KNOWS] -> (b) – [:KNOWS] -> (c), (a) – [:KNOWS] -> (c)  WHERE a.name = ‘Jim’  RETURN b, c  LIMIT 25 |

В качестве результата, такой запрос вернет нам прямых друзей пользователя Jim в пределах 25-ти строк, проще говоря, запрос вернет 25 прямых друзей Jim-а. [13]

Объединяя и комбинируя фразы, описанные в таблице 3 можно составлять более сложные и специфические запросы для получения необходимых результатов.

# РЕАЛИЗАЦИЯ

В данном разделе описаны действия связанные с разработкой графовой базы данных на СУБД Neo4j для системы управления проектами, скриптов для заполнения графовой БД и создание запросов на языке Cypher.

## Разработка графовой базы данных на СУБД Neo4j

Исходя из разделов 1.1, 1.2, и 1.3, графовая база данных для системы управления проектами должна упрощать управление, контроль и отслеживание изменений в имеющихся или вновь созданных проектах. Учитывая эти особенности, разрабатываемая графовая БД должна хранить следующие данные:

* Наименования и описания проектов, их заказчиков и руководителей,
* Наименования и описания задач, подзадач проекта,
* Исполнителей (разработчиков) назначенных на задачи,
* Совершённые изменения и их описания на этапе выполнения задач,
* URL адреса репозиториев в которых были созданы задачи или совершены их изменения на этапе выполнения.

Как упоминалось в разделе 1.5.3, графовая СУБД Neo4j хранит данные в гибком виде, используя для этого вершины и связи, поэтому СУБД Neo4j не нуждается в проектировании логической схемы модели данных. Это приводит к упрощенной разработке базы данных. Необходимо лишь наименовать связи, свойства и метки для их дальнейшего использования и извлечения необходимых данных из базы.

Схема разработанной базы данных для системы управления проектами представлена на рисунке 12.

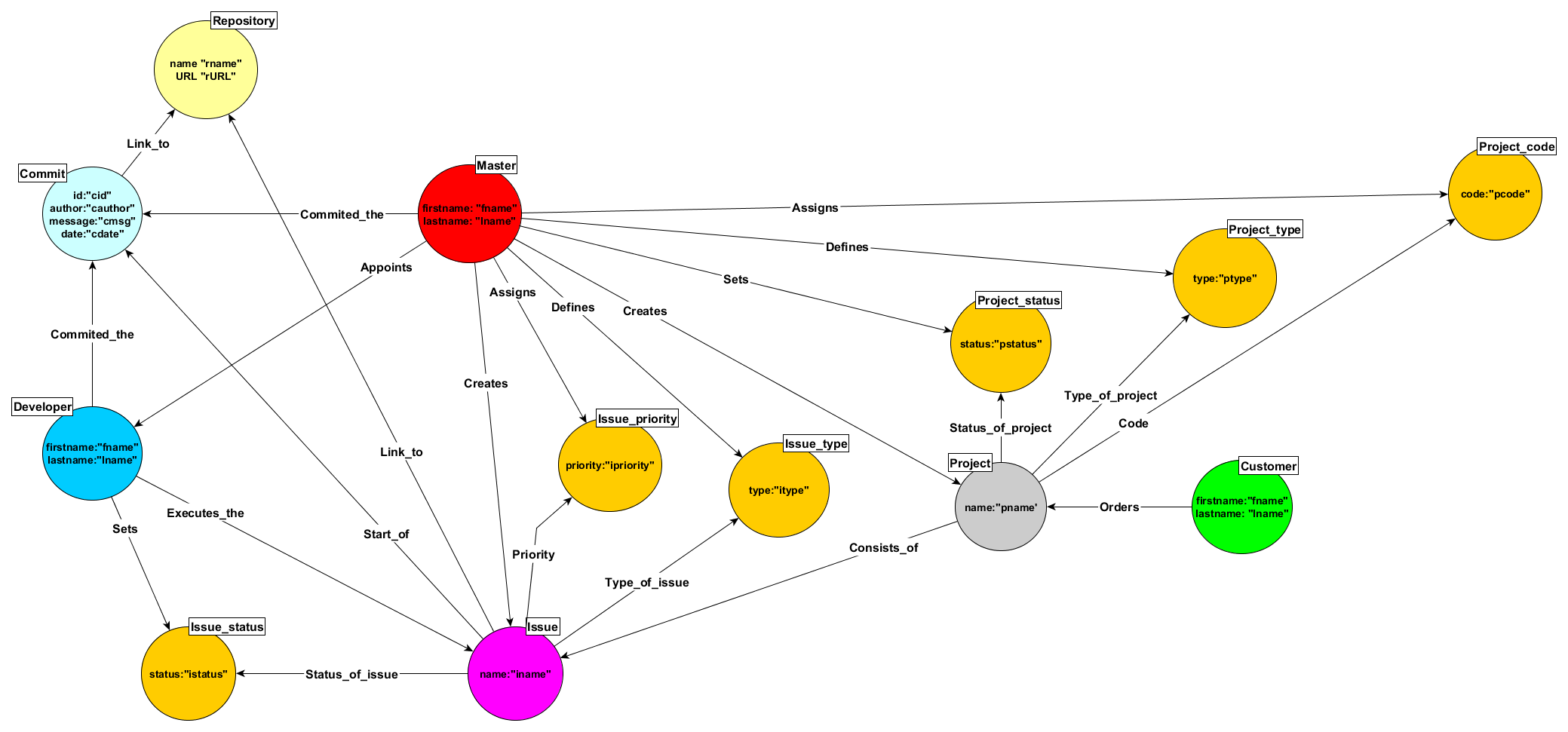


Рисунок 12. Схема графовой модели БД для системы управления проектами

Схема состоит из 13 вершин и 23 связей. За каждой вершиной закреплена метка, и каждая связь имеет свое свойство и направление.

Вершины графовой базы данных хранят такие данные как:

* ФИО заказчиков (Customer),
* Наименования проектов (Project),
* Статусы проектов (Project\_status),
* Типы проектов (Project\_type),
* Коды проектов (Project\_code),
* ФИО руководителей проектов (Master),
* Наименования задач (Issue),
* Типы задач (Issue\_type),
* Приоритеты задач (Issue\_priority),
* Статусы задач (Issue\_status),
* ФИО разработчиков (Developer),
* Совершённые изменения и их описания (Commit),
* Наименования и URL адреса репозиториев (Repository).

Свойства связей между вершинами, обеспечивают общее понимание разработанной графовой схемы системы управления проектами. К примеру, вершины Master (руководитель проекта), Project (проект), Issue (задача) и Developer (разработчик) связаны между собой такими свойствами как: Creates (создает), Consists\_of (состоит из), Appoints (назначает) и Executes\_the (выполняет). Следуя схеме (рис.12) нетрудно понять что, руководитель создает проект и задачи, назначает разработчиков, которые будут выполнять конкретную задачу или задачи, а сам проект состоит из задач.

Разработанная таким образом графовая модель, гарантирует извлечение необходимых связанных данных при корректно построенных запросах, позволяет легко отслеживать внесенные изменения при выполнении задач проекта и контролировать состояние проектов в целом.

.

## Создание скриптов для заполнения графовой БД

## Создание запросов на языке Cypher

# ОХРАНА ТРУДА

# ОБОСНОВАНИЕ ЭКОНОМИЧЕСКОЙ ЭФФЕКТИВНОСТИ

# ЗАКЛЮЧЕНИЕ

# БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Энциклопедия знаний [Электронный ресурс] – Режим доступа: <http://pandia.ru/365896/>, свободный. – Загл. с экрана (дата обращения: 04.05.2017).
2. Учебные материалы для студентов [Электронный ресурс] – Режим доступа: <http://studme.org/63927/logistika/zhiznennyy_tsikl_proekta#784>, свободный. – Загл. с экрана (дата обращения: 04.05.2017).
3. PMBOK: Руководство к Своду знаний по управлению проектами, 4-е изд., PMI, 2008. – 241 с.
4. ЭУП: Электронное учебное пособие [Электронный ресурс] – Режим доступа: <http://eos.ibi.spb.ru/umk/11_18/5/5_R0_T3.html>, свободный. – Загл. с экрана (дата обращения: 04.05.2017).
5. Библиотека Online. Лекции по управлению программными проектами [Электронный ресурс] – Режим доступа: <http://citforum.ru/SE/project/arkhipenkov_lectures/5.shtml>, свободный. – Загл. с экрана (дата обращения: 04.05.2017).
6. NOVOSOFT: Система управления проектами [Электронный ресурс] – Режим доступа: <http://www.novosoft.ru/consulting/project_management_system.shtml>, свободный. – Загл. с экрана (дата обращения: 04.05.2017).
7. GitHub [Электронный ресурс] – Режим доступа: <https://github.com/about>, свободный. – Загл. с экрана (дата обращения: 04.05.2017).
8. Bitbucket [Электронный ресурс] – Режим доступа: <https://ru.atlassian.com/software/bitbucket>, свободный. – Загл. с экрана (дата обращения: 04.05.2017).
9. Jira [Электронный ресурс] – Режим доступа: <https://ru.atlassian.com/software/jira>, свободный. – Загл. с экрана (дата обращения: 04.05.2017).
10. Ресурс для IT-специалистов. SQL или NoSQL [Электронный ресурс] – Режим доступа: <https://habrahabr.ru/company/ruvds/blog/324936/>, свободный. – Загл. с экрана (дата обращения: 04.05.2017).
11. Техническая документации Майкрософт [Электронный ресурс] – Режим доступа: <https://docs.microsoft.com/ru-ru/azure/documentdb/documentdb-nosql-vs-sql>, свободный. – Загл. с экрана (дата обращения: 04.05.2017).
12. Энциклопедия знаний Академик [Электронный ресурс] – Режим доступа: <http://dic.academic.ru/dic.nsf/ruwiki/1738292>, свободный. – Загл. с экрана (дата обращения: 04.05.2017).
13. Робинсон Ян, Вебер Джим, Эифрем Эмиль. Графовые базы данных: новые возможности для работы со связанными данными / пер. с англ. Р.Н.Рагимова; науч. ред. А.Н. Кисилев. – 2-е изд. – М.:ДМК Пресс,2016. – 256 с.
14. Издательский дом "Вильямс": Более подробно о моделях данных [Электронный ресурс] – Режим доступа: <http://www.williamspublishing.com/PDF/978-5-8459-1829-1/part.pdf>, свободный. – Загл. с экрана (дата обращения: 04.05.2017).
15. IBM: Processing large-scale graph data: A guide to current technology [Электронный ресурс] – Режим доступа: <https://www.ibm.com/developerworks/library/os-giraph/>, свободный. – Загл. с экрана (дата обращения: 04.05.2017).
16. Pregel: ACM Digital Library [Электронный ресурс] – Режим доступа: <http://dl.acm.org/citation.cfm?id=1807184>, свободный. – Загл. с экрана (дата обращения: 04.05.2017).
17. What is Neo4j? [Электронный ресурс] – Режим доступа: <https://neo4j.com/developer/graph-database/#_what_is_neo4j>, свободный. – Загл. с экрана (дата обращения: 04.05.2017).
18. Ресурс для IT-специалистов. MySQL vs Neo4j [Электронный ресурс] – Режим доступа: <https://habrahabr.ru/post/258179/>, свободный. – Загл. с экрана (дата обращения: 04.05.2017).
19. Chapter 3. Cypher [Электронный ресурс] – Режим доступа: <http://neo4j.com/docs/developer-manual/current/cypher/>, свободный. – Загл. с экрана (дата обращения: 04.05.2017).

# ПРИЛОЖЕНИЕ А

# ПРИЛОЖЕНИЕ Б