**АНОТАЦИЯ**

СОДЕРЖАНИЕ

[ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ 7](#_Toc481441693)

[ВВЕДЕНИЕ 8](#_Toc481441694)

[1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ 9](#_Toc481441695)

[1.1. Определение проекта 9](#_Toc481441696)

[1.2. Жизненный цикл проекта 11](#_Toc481441697)

[1.3. Введение в графовые базы данных 13](#_Toc481441698)

[1.3.1. Что такое граф? 14](#_Toc481441699)

[1.3.2. Графовая СУБД 15](#_Toc481441700)

[1.3.3. Механизмы вычисления графов 17](#_Toc481441701)

[1.3.4. Преимущества графовых баз данных 19](#_Toc481441702)

[1.4. Сравнение реляционных и графовых БД. SQL и NoSQL 20](#_Toc481441703)

[1.4.1. SQL и NoSQL решения 20](#_Toc481441704)

[1.4.2. Сравнение реляционной и графовой БД на эффективность 25](#_Toc481441705)

[2. ИСПОЛЬЗУЕМЫЕ СТАНДАРТЫ, ТЕХНОЛОГИИ, ИНТСРУМЕНТЫ 30](#_Toc481441706)

[2.1. Графовая СУБД Neo4j 30](#_Toc481441707)

[2.2. Язык запросов Cypher 33](#_Toc481441708)

[3. ПРОЕКТИРОВАНИЕ 37](#_Toc481441709)

# 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

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

# ВВЕДЕНИЕ

# АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

## Определение проекта

В современной литературе по управлению проектами можно выделить два основополагающих подхода к определению проекта: системный подход и деятельностный.

Системный подход характеризует проект как систему временных действий, направленных на достижение неповторимого, но в то же время конкретного результата.

Системный подход к определению проекта предопределяет основные его характеристики:

* разовость – все проекты представляют собой разовое явление. Они приходят и уходят, появляются и исчезают, оставляя после себя конкретные результаты, существенно отличаясь от наших повседневных обязанностей и деятельности,
* уникальность – нет двух идентичных проектов. Каждый из них, независимо от его результатов, в своей основе имеет что-то уникальное, характерное только для него,
* инновационность – в процессе реализации проекта всегда создается нечто новое. Изменения могут быть колоссальными или менее значительными,
* результативность – все проекты имеют вполне очевидные результаты. Это может быть новый дом, напечатанная книга, модифицированная структура предприятия, рабочее приложение. Все проекты нацелены на получение конкретных результатов, другими словами, они направлены на достижение целей,
* временная локализация – все проекты ограничены временными рамками. Проект – это создание чего-либо к назначенному сроку, он имеет планируемую дату завершения, после которой команда проектантов распускается.

Все перечисленные характеристики взаимосвязаны и задают определенные границы проекта, три его измерения, критерия, по которым можно оценить проект (рис.1).

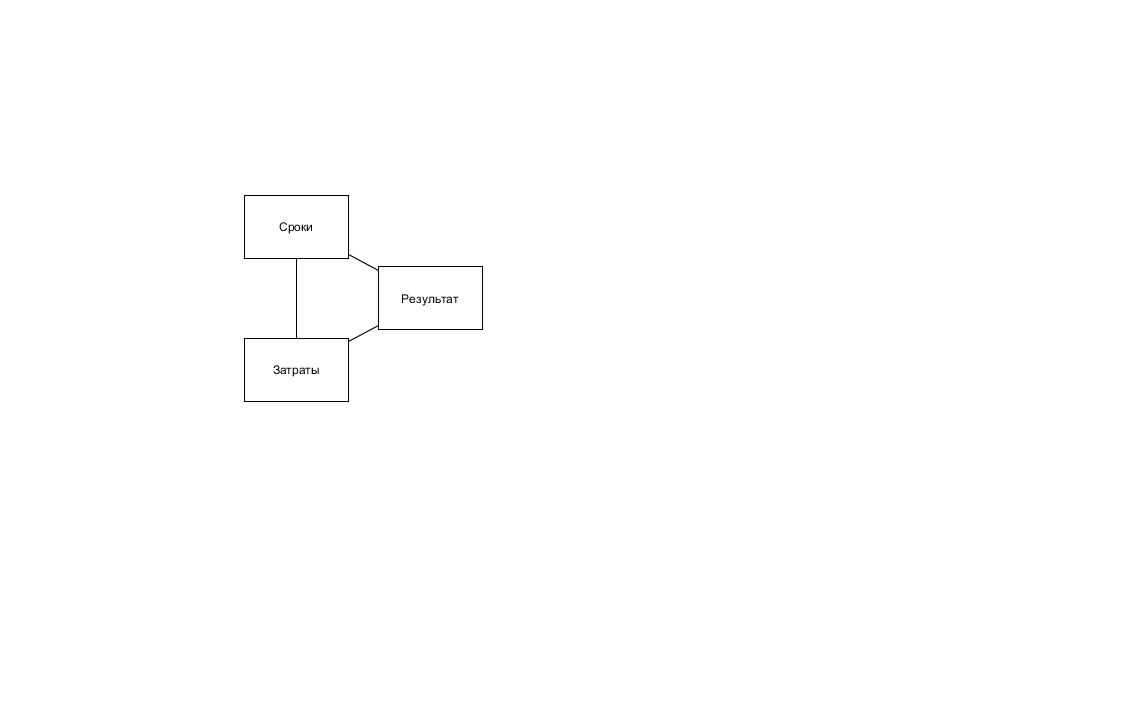


Рисунок 1. Схема измерения проекта

Планирование и реализация проекта всегда связаны с тремя главными вопросами:

* сколько времени это займет,
* сколько это будет стоить,
* совпадет ли конечный результат с тем, который планировался вначале.

Первый вопрос выводит на первый план проблему временных ограничений, установленных для реализации всего проекта и отдельных его этапов. Второй вопрос привлекает наше внимание к стоимости проекта, третий касается вопроса о результативности проектной деятельности.

Универсальность и многоаспектность проектной технологии определены разноуровневыми многослойными взаимодействиями и измерениями проекта. Измерения проекта – цели, время, стоимость – являются одновременно ограничениями проекта, задающими систему координат, в которой вынужден работать проект-менеджер. Сверхзадача проект-менеджера – найти оптимальное соотношение этих трех ограничений проекта, с которыми неразрывно связаны интересы участников проекта.

Второй подход – деятельностный – определяет проект как деятельность субъекта по переводу объекта из наличного состояния в состояние желаемого будущего, которое наиболее полно отвечает его представлениям, ожиданиям. [1]

Таким образом, проект в самом широком смысле может пониматься как творческая, разумная, целеполагающая деятельность субъекта. Проект - это то, что задумывается и после этого планируется, имеет определенные системы подходов с четко установленными временными рамками, капиталом и необходимым результатом. Каждый проект сам по себе уникален во всех, или практически во всех его аспектах.

## 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. 2).



Рисунок 2. 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 3, будем считать, что на каждую фазу дается одинаковое количество времени.[4]

Рисунок 3. 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]

## Введение в графовые базы данных

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.[6]

### Что такое граф?

Граф – совокупность точек, соединенных линиями. Точки называются вершинами, или узлами, а линии – ребрами, или дугами.

Степенью входа вершины является количество входящих в нее ребер, степенью выхода – количество исходящих ребер.

Граф, содержащий ребра между всеми парами вершин, является полным. Встречаются такие графы, ребрам которых поставлено в соответствие конкретное числовое значение, они называются взвешенными графами, а эти значения – весом ребра. Когда у ребра оба конца совпадают, то есть оно выходит из вершины и входит в нее, то такое ребро называется петлей (рис.4).[7]

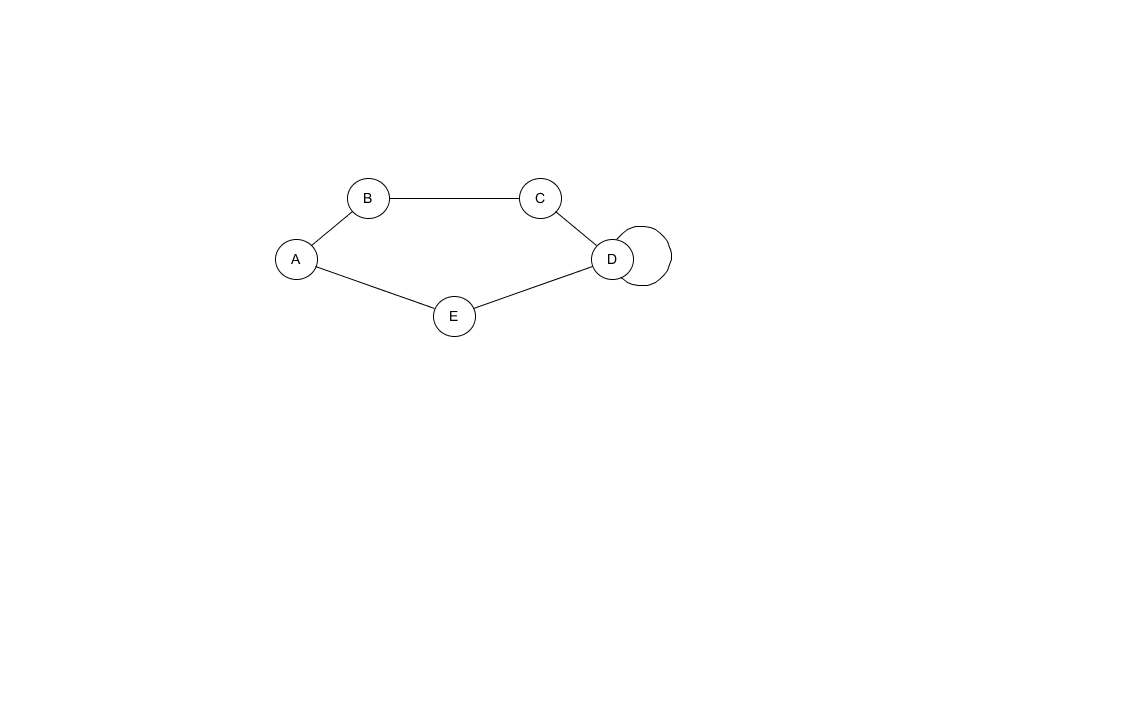


Рисунок 4. Пример простейшего графа с петлей

Также, граф имеют свою классификацию:

* связный – между любой парой вершин существует как минимум один путь,
* несвязный – существует хотя бы одна вершина, не связанная с другими,
* ориентированный – ребра графа являются направленными, то есть существует только одно доступное направление между двумя связанными вершинами,
* неориентированный – по каждому из ребер можно осуществлять переход в обоих направлениях,
* смешанный – характерен наличием как ориентированных, так и неориентированный ребер.[8]

Граф – это универсальная и выразительная структура, которая позволяет моделировать всевозможные сценарии, от постройки атомной станции до строительства системы дорог, от поставок продуктов питания до историй болезни населения, он чрезвычайно полезен при анализе самых разнообразных наборов данных в таких областях, как наука, государственное управление и бизнес.

### Графовая СУБД

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.[9]

Самые популярные на сегодняшний день графовые СУБД:

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

Фундаментальная модель данных графовых баз очень простая: 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 5 показан пример модели графовой базы данных с маленькими узлами и многочисленными связями между ними.



Рисунок 5. 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”.[10]

### 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.[11]

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 6 демонстрирует типовую архитектуру развертывания механизмов вычисления графов.



Рисунок 6. 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. Периодический задания на извлечения, преобразования и загрузку данных перемещают данные из системы записи базы данных в механизм вычисления графов для выполнения анализа и автономных запросов.[12]

### 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.

По мимо роста в производительности, графовые базы данных предоставляют очень 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 приятное – 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.[9, 10]

## Сравнение реляционных и графовых БД. SQL и NoSQL

Текущее положение дел таково, что мы живем в мире высоких технологий с большими объемами данных (к примеру, структурированные, взаимосвязанные данные) и мало того, эти данные продолжают расти.

Для того что бы эти данные как то хранить и работать с ними, существуют технологии баз данных. Эти технологии разделились на два типа: SQL и NoSQL – реляционные и нереляционные базы данных.

### SQL и NoSQL решения

Приведем некоторые ключевые концепции реляционных и нереляционных баз данных. На рисунке 7 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 7. 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.[13]

Категория баз данных NoSQL заметно отличается от SQL БД. 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.[14]

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 |
| Хорошо подходит для полу структурированных, сложных или вложенных данных | Хорошо подходит для структурированных данных |
| Схема | Динамические или гибкие схемы | Строгая схема |
| База данных является схемой – агностиком, и схема диктуется приложением. Это обеспечивает гибкость и высокую итеративную разработку | Схема должна поддерживаться и храниться в синхронизации между приложением и базой данных |
| Транзакции | Поддержка транзакции 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 |
| 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 |
| Масштабируемость | Масштабирование обычно выполняется горизонтально, при этом данные разделяются на диапазоны серверов | Масштабирование обычно осуществляется вертикально с использованием большого количества ресурсов сервера |

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.

### Сравнение реляционной и графовой БД на эффективность

Для сравнения этих двух баз данных, в качестве примера были выбраны такие СУБД как MySQL – реляционная база данных и Neo4j – графовая БД.

Что бы сравнить эффективность выбранных баз данных необходимо наполнить их одними и теми же данными, при этом объём данных должен быть существенный, так как при малом объеме разницы мы не увидим. Исходя из требований, была выбрана тестовая предметная область – социальная сеть и составлена ER – диаграмма (рис.8), на основе которой были созданы реляционная и графовая модели.



Рисунок 8. 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 оказался довольно ощутим, это связано с тем, что было использовано много полей с текстовой информацией. Далее было проведено несколько тестов на эффективность. Тесты проводились 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.

Первый тест заключался в измерении времени поиска пользователя по его идентификатору определенное количество раз. После проведения теста были получены следующие результаты, которые отражены в таблице 2.

Таблица 2. Результаты проведения первого теста

|  |  |  |
| --- | --- | --- |
| Количество запросов | Время для MySQL, мс | Время для Neo4j, мс |
| 10 | 4 | 9 |
| 100 | 34 | 76 |
| 1000 | 286 | 510 |
| 5000 | 1034 | 1103 |
| 10000 | 1340 | 1187 |
| 30000 | 3390 | 1384 |
| 60000 | 6876 | 2102 |
| 90000 | 10537 | 3175 |
| 120000 | 14033 | 3858 |

По результатам таблицы 2 была построена диаграмма зависимостей (рис.9), для более наглядного представления результатов.

Рисунок 9. Диаграмма поиска пользователя по его идентификатору

Второй тест заключается в измерении времени поиска общего количества фотографии у пользователей, которые администрируют хотя бы одну группу, в зависимости от диапазона значений идентификаторов пользователя. Данный тест является сложнее, так как поиск осуществляется по общим взаимосвязям. После проведения теста были получены результаты, которые отражены в таблице 3.

Таблица 3. Результаты проведения второго теста

|  |  |  |
| --- | --- | --- |
| Диапазон идентификатора < | Время для MySQL, мс | Время для Neo4j, мс |
| 10 | 164 | 56575 |
| 100 | 674 | 2400 |
| 1000 | 5433 | 2481 |
| 5000 | 23747 | 2408 |
| 10000 | 44917 | 2650 |
| 30000 | 136592 | 3108 |
| 60000 | 280672 | 5004 |
| 90000 | 438875 | 5102 |

По результатам таблицы 3 была построена еще одна диаграмма зависимостей (рис. 10) для наглядного изображения результатов второго теста.

Рисунок 10. Диаграмма поиска общего количества фотографий

Для получения более точных результатов второго теста, проведем его еще раз, но при условии, что базы данных MySQL и Neo4j не будут работать одновременно на одной ЭВМ, как это было до этого. После проведения повторного теста были получены результаты, которые отражены в таблице 4.

Таблица 4. Результаты повторного теста

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

По результатам таблицы 4 была построена диаграмма зависимостей (рис. 11) для повторного второго теста.

Рисунок 11. Повторная диаграмма поиска общего количества фотографий

Исходя из диаграммы, можно заметить, что график для MySQL стал немного стабильнее, но при этом Neo4j стала сильно проигрывать при поиске на небольших объемах данных.[15]

Проведенные тесты показали, что графовая база данных эффективней 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.

# ИСПОЛЬЗУЕМЫЕ СТАНДАРТЫ, ТЕХНОЛОГИИ, ИНТСРУМЕНТЫ

## 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. Neo4j используется сегодня сотнями тысяч организаций, практический во всех отраслях, таких как управления сетью, 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 12 изображен пример графовой модели данных Neo4j.



Рисунок 12. Пример графовой модели данных Neo4j

Neo4j эффективно реализует графовую модель вплоть до уровня хранилища. В отличие от обработки графики или библиотек в памяти, Neo4j предоставляет полные характеристики 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.13) 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 13. 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 14.



Рисунок 14. 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. [16]

## 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 он 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, вдохновлены SQL. Совмещение с образцом заимствует подходы к выражению из SPARQL. Некоторые из выражений были заимствованы из таких языков, как Haskell и Python. [17]

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 15 изображен шаблон, который описывает трех друзей.



Рисунок 15. 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 5 приведены фразы (не все), и их описание которые используется в запросах на Cypher.

Таблица 5. 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 |
| 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 |
| ORDER BY | Фраза, которая следует за фразами RETURN или WITH, указывающая, что вывод должен быть отсортирован следующим образом |

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. [9]

Объединяя и комбинируя фразы, описанные в таблице 5 можно составлять более сложные и специфические запросы для получения необходимых нам результатов.

# ПРОЕКТИРОВАНИЕ