**2. Практическая часть**

**2.1 Обзор графовых баз данных.**

Sones GraphDB.

Эта графовая БД разработана компанией Sones в 2009 г. и поддерживалась до конца 2011 г., после чего компания обанкротилась. За время поддержки была выпущена версия 2.1, доступная в двух вариантах: версия Community распространяется по лицензии AGPL v3, для коммерческих же проектов необходимо приобретать версию Enterprise. Кроме того, важное отличие платной версии от бесплатной состоит в наличии возможности устойчивого хранения базы на жестком диске. Бесплатная же версия позволяет хранить данные только в оперативной памяти (in-memory). Поскольку компания-разработчик прекратила свое существова- ние, единственная версия БД, которую можно протестировать, — бесплатная версия Сommunity, т. е. тестирование будет ограничено небольшими объемами данных, целиком помещающимися в оперативную память. Соответственно, работать с такими данными можно только в случае либо их маленького объема, либо постоянной очистки текущего хранилища от старых данных. Sones не имеет встроенной поддержки алгоритмов обхода графов, в связи с чем для тестирования задачи поиска соседей был реализо- ван поиск в ширину [9]. Кроме того, для тестирования поиска пере- сечений множеств соседних вершин была реализована работа с мно- жествами найденных вершин.

Neo4J GraphDB.

Эта графовая БД разработана компанией Neo Technology в 2009 г. Она доступна в трех вариантах: Community, Advanced и Enterprise. Версия Community распространяется по лицензии AGPL v3, для коммерческих же проектов необходимо приобретать версию Advanced, включающую дополнительные возможности мониторинга состояния базы. Версия Enterprise, кроме того, поддерживает резервное копирование и масштабируемость. На сегодняшний день Neo4J является наиболее популярной графовой БД, в первую очередь ввиду того, что бесплатная версия предлагает все необходимые инструмен- ты для полноценной работы с графами. Neo4J, в отличие от графовой БД Sones, может устойчиво хра- нить данные на жестком диске. Следовательно, объем хранящейся информации ограничен лишь объемами жесткого диска. Для достижения максимальной производительности в Neo4J существует два типа кэширования: файловый кэш (file buffer cache) и объектный кэш (object cache). Первый кэширует данные с жесткого диска, целью че- го является увеличение скорости чтения/записи на жесткий диск. Второй кэш хранит в себе различные объекты графа: вершины, ребра и свойства в специальном оптимизированном формате для увеличе- ния производительности обходов графа. Neo4J обладает режимом импорта большого объема данных BatchInserter [10]. В этом режиме происходит отключение транзакций в БД, следствием чего является резкое увеличение скорости импорта. Кроме того, в Neo4J поддерживаются алгоритмы обхода графов, в том числе можно сделать обход в ширину до заданного уровня, что как раз и необходимо для решения задачи поиска соседних вершин. Для пересечения двух множеств, как и в случае с базой данных Sones, была реализована работа с множествами.

DEX GraphDB.

Эта база разработана компанией Sparsity Technologies в 2008 г. Существует три типа лицензий, по которым предоставля- ется право пользования DEX GraphDB: Community, Commercial и Education. При наличии лицензии Community в базе суммарно может содержаться не более 1 млн вершин и разрешен доступ к чтению данных только одному потоку. По лицензии Commercial отсутствуют какие-либо ограничения, но годовая стоимость владения лицензией зависит от суммарного количества объектов в базе и количества по- токов, имеющих доступ к чтению данных. Лицензия Education предназначена для бесплатного предоставления исследовательским и учебным некоммерческим проектам. DEX GraphDB, как и Neo4J, имеет полноценную поддержку устой- чивого хранения данных. Но в отличие от Neo4J ядро DEX написано на C++, что очень хорошо сказывается на производительности. Графовая БД DEX имеет только один объектный кэш, который держит в оперативной памяти все часто используемые объекты хранилища [11]. БД DEX имеет широкие возможности по обходу графов. Она предоставляет множество встроенных алгоритмов обхода графов, та- ких как поиск в глубину, поиск в ширину, Дейкстры и поиска компо- нентов сильной связности. Кроме того, существует встроенная поддержка работы с множествами. Таким образом, обе поставленные задачи анализа биллинговой информации можно решить, используя только встроенные средства БД.

В дальнейшам для анализа будет использоваться графовая БД Neo4J.

**2.2 Анализ использования графовой БД Neo4J в геолокационных приложениях.**

Язык запросов Cypher.

Для работы с базой данной neo4j используется специальный язык Cypher. Cypher является декларативным языком запросов для графов, который позволяет выразительно и эффективно выполненить запросы и обновленить базу графов. Cypher является относительно простым, но также мощным языком. Сложные запросы к базе данных легко могут быть выражены через Cypher.

Cypher разработан как человекочитаемый язык запросов, подходящий как для разработчиков, так и специалистов аналитиков. Задачей языка состоит в том, чтобы сделать простые вещи легко, и сложные вещи возможными. Его конструкции основаны на английской языке и иконографии, которые помогают сделать запросы более понятными. Язык старается быть удобным как для чтения, так и для записи.

Cypher заимствует свою структуру из SQL - запросы строятся с использованием различных операций.

Операции могут быть соединены друг с другом, и они распределяют промежуточные результаты между собой. Например, совпадающие переменные из одного пункта MATCH имеют один контекст между операциями.

Язык запросов состоит из нескольких различных операций:

* MATCH: Шаблон графа для поиска. Наиболее распространенный способ получения данных из графа.
* WHERE: Используется как часть часть MATCH, OPTIONAL MATCH и WITH. Добавляет ограничения на шаблон, или фильтрет промежуточные результаты проходящий через WITH.
* RETURN: Что будет возвращено.

Использование neo4j в приложении.

Для текущего проекта будет использоваться следующая модель данных, представленная на рисунке 3.1

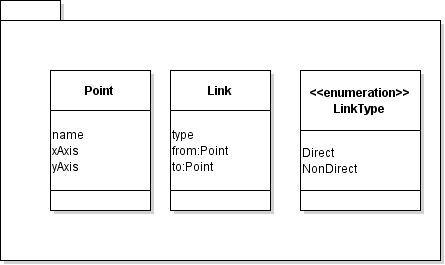


Рис.3.1 Модель данных приложения

Для создания в базе данных вершин графа используется следующий запрос:

CREATE (p:Point {name:'First', xAsis:15.15, yAxis:15.15})

CREATE (p:Point {name:'Second', xAsis:20.20, yAxis:20.20})

Для создания ребер графа использется следующий запрос:

MATCH (f:Point {name:'First'}) MATCH (s:Point{name:'Second'}) CREATE (f)-[d:DIRECT]->(s)

Для анализа работы с алгоритмами будет использоваться следующий граф.

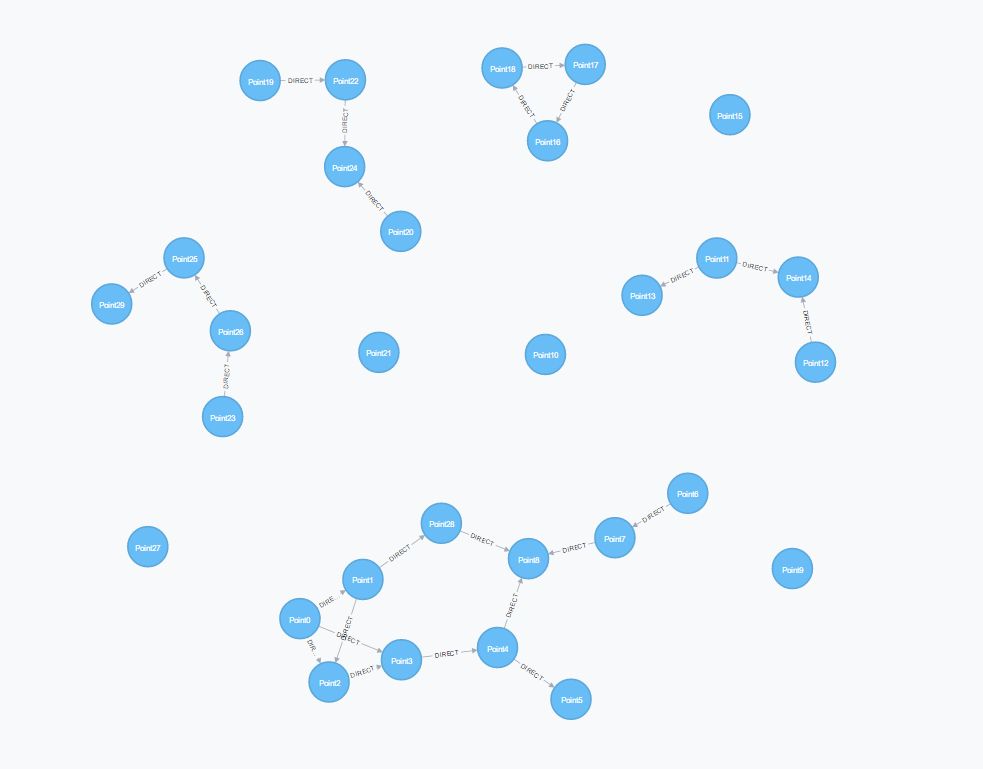


Рис. 3.2 – Тестовый граф

В базе данных neo4j также присутствуют реализации алгоритмов работы с графами, таких как поиск связанности вершин.

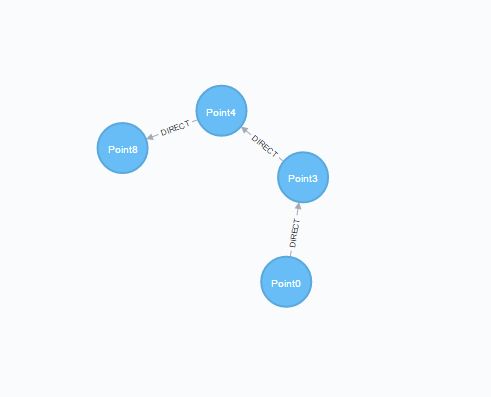
MATCH (f:Point{name:'Point0'})-[\*]->(s:Point{name:'Point8'}) RETURN count(\*)>0

Данный запрос вернет булево значение в зависимости связаны ли 2 вершины.

Другим встроенным алгоритмом является нахождение кратчайшего пути

MATCH (f:Point { name:"Point0" }),(s:Point { name:"Point8" }), p = shortestPath((f)-[\*]-(s)) RETURN p

Результатом выполнения будет являться подграф являющимся кратчайшим путем для данного графа.



3.3 – Результат нахождения кратчайшего пути для тетсового графа

Для примения пространственного анализа Neo4J в java-приложения имеется библиотека Neo4j Spatial. Данная библиотека позволяет выполнять следующие функции:

- Импорт данных из файлов ESRI Shapefile и Open Street Map

- Поддержка разных геометрических фигур

- Реализации структуры данных R-дерево для поиска пространственных данных

- Поддержка топологических операций поиска (содержит, в пределах, пересекается, охватывает, не пересекаются, и т.д.)

- Возможность применения пространственных операций на любом графе

Для выполнения поиска точек в пределах определенной геометрической фигуры может примениться следующий запрос, выполненный через библиотеку Neo4j Spatial:

GraphDatabaseService database = new GraphDatabaseFactory().newEmbeddedDatabase(storeDir);

try {

SpatialDatabaseService spatialService = new SpatialDatabaseService(database);

Layer layer = spatialService.getLayer("layer\_roads");

SpatialIndexReader spatialIndex = layer.getIndex();

Search searchQuery = new SearchIntersectWindow(new Envelope(xmin, xmax, ymin, ymax));

spatialIndex.executeSearch(searchQuery);

List<SpatialDatabaseRecord> results = searchQuery.getResults();

} finally {

database.shutdown();

}

Результатом выполнения будут вершины графа, координаты которого будут располагаться внутри параметров xmin, xmax, ymin, ymax.