УДК 004.421

**СОЗДАНИЕ МОДИФИКАЦИИ ПАТТЕРНА REPOSITORY ДЛЯ РАБОТЫ С ГРАФОВЫМИ БАЗАМИ ДАННЫХ**

**Жильцов Владимир Александрович**

Студент, техник программист

ГОУ ВПО «Донецкий национальный технический университет»

ГУ «Институт проблем искусственного интеллекта»

***Научный руководитель: Дорохина Галина Владимировна***

*Заведующий отделом системного анализа и интеллектуальных интерфейсов*

*ГУ «Институт проблем искусственного интеллекта»*

**Аннотация:** известным паттернами проектирования для взаимодействия с базами данных является ObjectRelationMapping (ORM). Данный паттерн предназначен для работы с данными из базы данных как с объектами. Для работы с графовыми базами данных стандартных паттернов на основе ORM нет. В работе предложено модифицировать паттерн Repository, что обеспечит высокую скорость разработки конечного продукта и производительность при работе с графовой базой данных.

**Ключевые слова:** паттерн, Repository, графовая база данных, Neo4j, ObjectRelationMapping

**MODIFICATION OF THE REPOSITORY PATTERN FOR WORK WITH GRAPH DATABASES**

**Zhiltsov Vladimir Alexandrovich**

**Scientific adviser: Dorokhina Galina Vladimirovna**

Abstract: A well-known design pattern for interacting with databases is Object Relation Mapping (ORM). This pattern is designed to work with data from the database as with objects. There are no standard ORM-based patterns for dealing with graph databases. The paper proposes to modify the Repository pattern, which will ensure high speed of development of the final product and performance when working with a graph database.

Keywords: pattern, Repository, database, Neo4j, Object Relation Mapping

Сегодня все бизнес-приложения работают с использованием базы данных (БД) и чем эффективней будет взаимодействие с ней на прикладном уровне, тем меньше издержек и вышеприбыль будет у бизнеса. По этой причине, при разработке программного обеспечения разработчики уделяют большое количество времени на изучение предметной области проекта, базой данных, если она уже существует, а также на разработку модуля взаимодействия между БД и клиентской частью приложения. Верно выбранный архитектурный паттерн взаимодействия с БД позволит обеспечить легко поддерживаемый и масштабируемый код, а, следовательно, приложение, не требующее больших человеческих ресурсов, для его сопровождения после завершения разработки.

В течении длительного периода времени создавались и оттачивались подходы для взаимодействия с базой данных. Один из таких подходов - ObjectRelationMapping.

**ObjectRelationMapping.** ObjectRelationMapping – архитектурный паттерн, который связывает БД с принципами объектно-ориентированных языков программирования. Суть данной технологии заключается в том, что каждая строка реляционной БД сопоставляется с объектом класса, который задает разработчик. В свойство объекта помещаются данные из соответствующего столбца, сопоставление происходит чаще всего по имени свойства и столбца. Благодаря использованию данной технологии мы можем взаимодействовать с каждой строкой БД как с объектом. Если речь идет о графовой БД, то свойства каждого узла и связи между узлами сопоставляются со свойствами объекта.

Многие паттерны способны дополнять друг друга, ORM не является исключением. Данный паттерн является основой для более масштабных шаблонов, самые популярные из них это ActiveRecord, DataMapper, Repository. В современной разработке паттерн Repository стал наиболее популярным, так как онявляется наиболее гибким и масштабируемым.

**Repository.** Repository – паттерн, представляющий собой уровень доступа к данным, который выполняет двунаправленную передачу данных между постоянным хранилищем данных (зачастую базой данных) и уровнем отображения данных (веб страница, окно приложения и т.д.). Паттерн выполняет функцию изолирования уровня БД и отображения. Слой состоит из одного или нескольких репозиториев (классов манипулирования информации указанного типа), выполняющих передачу данных. Реализации репозиториев могут различаться по объему и предоставляемому функционалу, такуниверсальные репозитории будут использовать обобщенные типы и тем самым обрабатывать множество различных типов, а специализированные репозитории будут обрабатывать указанные типы.

В самом простом виде, репозиторий будет иметь связь, показанную на рисунке 1.

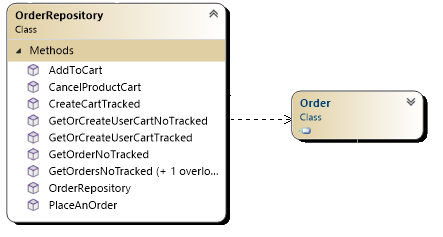


Рис. 1 – Самая примитивная реализация паттерна Repository

В таком виде паттерн хорошо подходит для реляционных БД, однако графовые БД требуют иной подход, так как они могут хранить данные не только в узлах, но и в связях.

**Создание модификации паттерна Repository для применения в графовых базах данных**. Предложенное развитие паттерна Repository состоит с следующем. Для начала необходимо определить интерфейс IModel, который будут расширять все остальные интерфейсы и реализовывать классы узлом и связей базы данных. Все классы, реализующие данный интерфейс, будут обязаны определить свойства и методы, находящиеся в данном интерфейсе. Одним из таких свойств является свойство Id, которое служит уникальным идентификатором.

Интерфейс IModel расширяют интерфейсы INode и IRelation, которые предназначены для узлов и связей соответственно.

Класс Node реализует интерфейс INode. Данный класс выступает базовым для узлов, хранящихся в БД. Интерфейс IRelation реализует обобщенный класс Relation <TNodeFrom, TNodeTo>, они являются базовыми для связей в БД. Помимо уникального идентификатора, интерфейс будет задавать ссылки на узлы, между которыми будет проложена связь, а также сами узлы. Не всегда возникает необходимость указывать из какого в какой узел идет связь, однако в текущем проекте использовать база данных Neo4j, которая создает исключительно направленные связи.

Реализованные интерфейсы и классы изображены на рисунке 2.

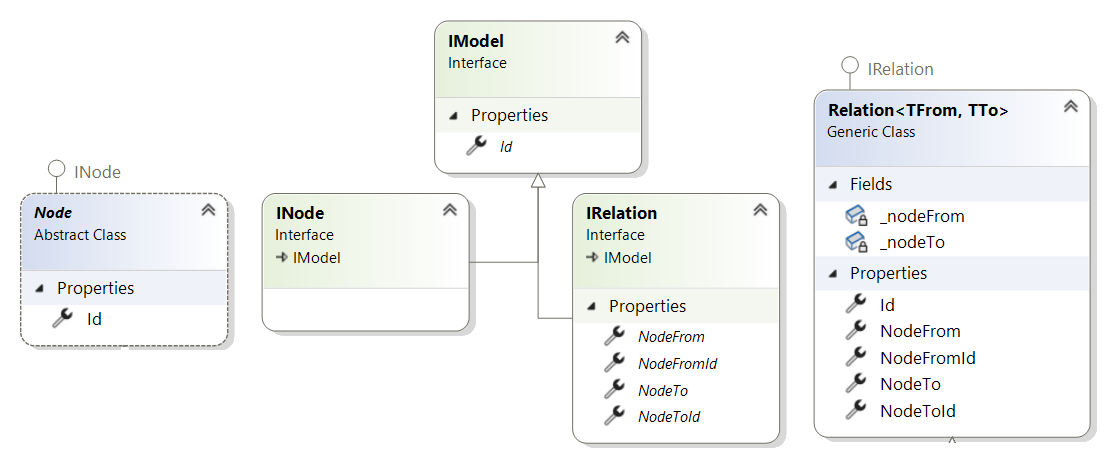


Рис. 2 – Диаграмма базовых классов для данных, хранимых в БД

Следующим шагом будет создание непосредственно репозиториев и необходимой инфраструктуры подходящих для графовых БД.

Интерфейс IGeneralRepository<TNode> отвечает за определение методов, которые должны быть в каждой реализации универсального репозитория, а класс GeneralRepository<TNode> реализует данный интерфейс. В универсальном классе реализованы основные методы для работы с узлами, но наибольшего внимания требуют методы для работы со связями. Так, если нам нужны связанные узлы с неким определенным типом связи, то нам необходимо указать тип связи, для этого вновь используются обобщения.

На рисунке 3 для примера изображена сигнатура метода GetRelatedNodesAsync из интерфейса IGeneralRepository<TNode>.

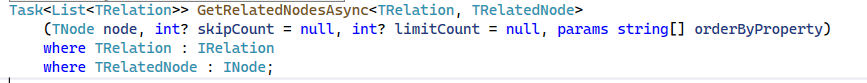
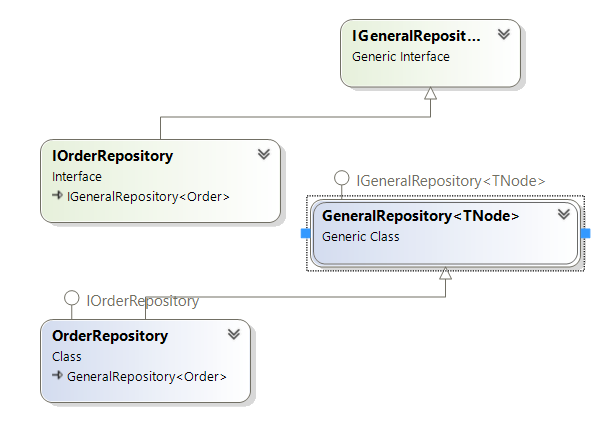


Рис. 3 – Диаграмма базовых классов для данных, хранимых в БД

Данный метод возвращает список связей указанного типа, количество связей будет равняться количеству связанных элементов.

Для получения результата для данного запроса, необходимо определить к какому узлу относится переданный нам объект, он является начальной или финальной точкой связи, для этого необходимо использовать рефлексию. Однако пользователю данного класса нет необходимости об этом беспокоиться.

Разработчику могут понадобиться методы, которые выходят за рамки реализованного функционала, для этого он может создать специализированные репозитории, которые должны наследоваться от универсального. В диаграмме классов на рисунке 4 можно увидеть пример реализации специализированного репозитория.

  
Рис. 4 – Диаграмма классов паттерна Repository

**Заключение.** Предложена модификация паттерна Repository, которая обеспечивает работу не только с реляционной, но и с графовой БД. Реализованная на базе предложенного паттерна инфраструктура является слабосвязанной, гибкой, а, следовательно, легко масштабируемой.

**Список источников**

1. Data Access Patterns: the Features of the Main Data Access Patterns Applied in Software Industry [Электронный ресурс] – Режим доступа: https://medium.com/mastering-software-engineering/data-access-patterns-the-features-of-the-main-data-access-patterns-applied-in-software-industry-6eff86906b4e
2. Repositorypattern[Электронный ресурс]– Режим доступа: https://ducmanhphan.github.io/2019-04-28-Repository-pattern/
3. Neo4jGraphDataPlatform[Электронный ресурс] – Режим доступа: https://neo4j.com/