СИСТЕМА ЛОГИЧЕСКОГО ПРОЕКТИРОВАНИЯ РЕЛЯЦИОННЫХ БД С ВОЗМОЖНОСТЬЮ ИСПОЛЬЗОВАНИЯ В ОБУЧАЮЩИХ ЦЕЛЯХ

ОЛЕЙНИК Сергей, ЧЕБОТАРЬСергей
Coordonator: САРАНЧУК Дориан

Технический Университет Молдовы

Аннотация: *В работе представлен программный продукт для логического проектирования БД. Для реализации приложения был применен обьектн-ориентированный подход. Описаны основные концепции разработанной системы, реализованные алгоритмы, приведены примеры работы приложения, а также инструкции по взаимодействию с программным интерфейсом. Показана необходимость в создании систем логического проектирования реляционных БД.*

Ключевые слова: *реляционная база данных, логическое проектирование, избыточность, объектно-ориентированный подход, обучающая система.*

1. Введение

Базы данных (БД) являются незаменимой составляющей современных информационных систем, обеспечивающей надёжный и удобный способ структурированного хранения данных в больших объёмах, в течение большого периода времени. Наибольшее распространение на сегодняшний день получили реляционные БД, предложенные Е. Коддом [1]. С каждой новой версией систем управления базами данных (СУБД) мировые производители привносят новые методики улучшения работы с БД, оптимизируя запросы пользователей или адаптируя структуру БД. Однако, основная работа, направленная на обеспечение эффективной работы БД, ложится на плечи её проектировщика. Логическое проектирование БД может сопровождаться рядом трудностей, связанных с экспоненциальной сложностью используемых для нормализации алгоритмов, а значит, автоматизация процесса логического проектирования БД на сегодняшний день является актуальной задачей.

Ненормализованная БД обладает рядом недостатков. Во первых, это повышенная избыточность данных. Со временем, избыточность затрудняет нормальную работу системы, снижая её производительность и повышая требования к минимальным системным ресурсам, необходимым для её функционирования. Другой отрицательной стороной ненормализованной базы данных является появление аномалий обновления в работе с БД, в результате которых удаление или изменение одних данных может повлечь за собой потерю другой важной информации. Нормализация БД призвана уменьшить избыточность и исключить аномалии, возникающие при работе с БД. Таким образом, нормализация баз данных является одним из важнейших этапов в создании современных информационных систем.

Для нормализации реляционных БД используется специальный вид ограничений целостности - функциональные зависимости. Они задаются при проектировании БД и должны поддерживаться на всем протяжении срока эксплуатации БД. Избыточные функциональные зависимости могут спровоцировать появление избыточных реляционных схем. Построение покрытий множеств функциональных зависимостей способствует устранению избыточности в реляционных БД [2].

Цель разработанной информационной системы – предложить пользователю функционально богатый инструментарий для логического проектирования реляционных БД. Для этого были реализованы алгоритмы построения покрытий функциональных зависимостей [3] и нормализации БД [4], предоставив пользователю возможность оперировать всеми типами зависимостей в реляционных базах данных: функциональными, многозначными и зависимостями соединения. Приложение сочетает в себе универсальность, простоту в использовании и должно предоставлять возможность применения в обучающих целях.

Исходя из отсутствия полных аналогов системы, удобства при взаимодействии с приложением, а также возможности получения пользователем всех промежуточных результатов проектирования, можно предположить, что система будет весьма полезна как для проектировщиков реляционных баз данных, так и для тех, кто желает получить опыт в нормализации БД.

2. Реализация системы

Для реализации системы логического проектирования БД с возможностью использования в обучающих целях было разработано программное приложение „Алгоритмы БД ”. При разработке был использован язык программирования C# и программная платформа .Net [5,6], а объектно-ориентированный подход сделал приложение гибким и расширяемым, обеспечивая добавление новых реализованных алгоритмов простым и не влияющим на надёжность и функционирование системы в целом способом.

Чтобы воспользоваться всеми преимуществами объектно-ориентированного подхода, были созданы иерархии классов зависимостей, множеств зависимостей (рисунок 1) и алгоритмов (рисунок 2). Особенно важно обратить внимание на иерархию классов алгоритмов. Ниже будут представлены и описаны только некоторые из созданных классов.

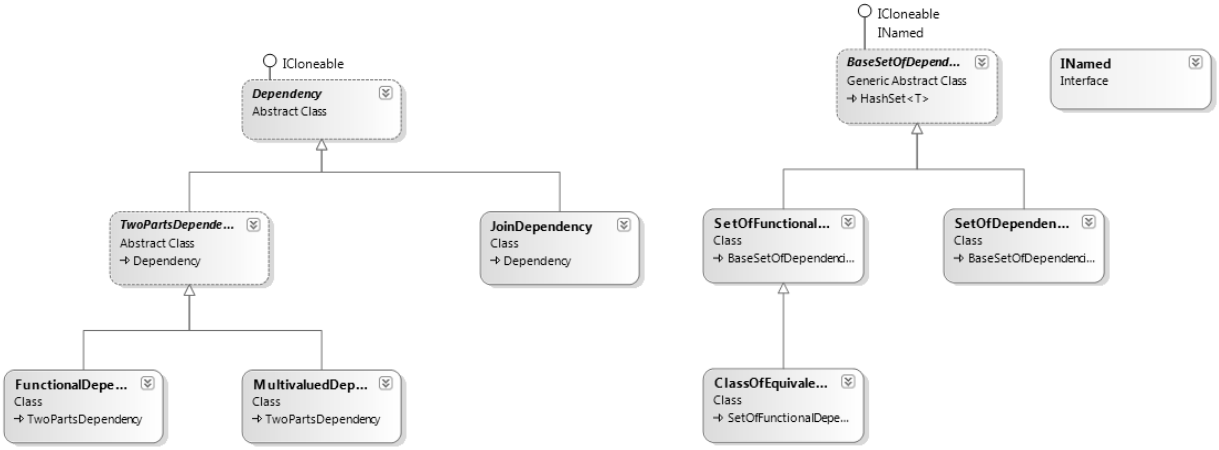

Рисунок 1 – Иерархии классов зависимостей и множеств зависимостей

Каждый алгоритм нашёл своё отображение в одноимённом классе. В классах алгоритмов определён открытый метод “Solve”, вызов которого с входным параметром множества зависимостей начинает выполнение возложенной на него задачи. Описанные действия происходят асинхронно, делая возможным обновление пользовательского интерфейса, что особенно актуально при больших диапазонах входных данных. Помимо возвращённого значения, предоставляется возможность получения всех промежуточных результатов (удаление, добавление и изменение зависимостей). Эта „прозрачность” алгоритма реализуется за счёт наследования класса „TransparentFunctionClass”. В данном классе инкапсулирована вся базовая логика оповещения об изменениях, проходящих в объектах классов алгоритмов. Для получения уведомлений пользователь класса (в нашем случае это класс окна пользовательского интерфейса) должен подписаться на событие „StateChangedEvent” типа „TransparencyEventHandler”, которое наследуется всеми „прозрачными” классами от базового класса „TransparentFunctionClass”. В случае использования одним „прозрачным” алгоритмов другого „прозрачного” алгоритма выполняется композиция, и объемлющий класс подписывается на события объектов используемых алгоритмов. Класс „TransparentFunctionClass” содержит определение метода „AddSubscrierOnStateChangedEvent”, который инкапсулирует в себе подпись на событие изменения состояний, происхлдящих в дочерних классах.

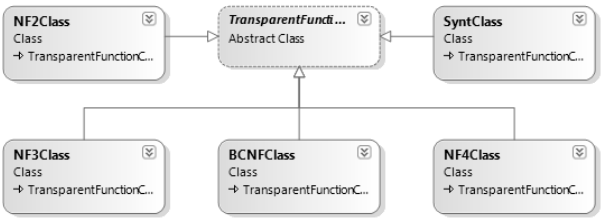

Рисунок 2 – Основные элементы иерархии классов алгоритмов

„TransparencyEventArgs” – это класс аргумента события. Он записывает в стек имена всех объектов, пропустивших через себя данное событие. Любой подписчик может проследить ход события от одного объекта к другому. На данной концепции базируется построение дерева вывода промежуточных результатов, в котором каждый выполняемый алгоритм представляется в виде узла.

Класс функциональной зависимости („FunctionalDependency”) наследует от базового класса „TwoPartsDependency”, который в свою очередь уже наследует от абстрактного класса „Dependency”.

Класс „SetOfFunctionalDependency” (множество функциональных зависимостей) наследует от класса „BaseSetOfDependencies”, для которого базовым классом является параметризированный класс, предоставляемый фреймворком – „HashSet<T>”. Здесь T – это тип зависимости, т.е. для функциональной зависимости это будет „FunctionalDependency”.

3. Использование системы в обучающих целях

Пользователю предлагается удобный способ ввода всех видов зависимостей, ускоряется процесс ввода зависимостей путём сохранения атрибутов, которые были ранее введены, и их быстрым выбором установкой напротив них флага. Помимо быстрого добавления атрибутов была реализована возможность отмены выбранных атрибутов снятием флажка и удаления их из строки редактирования.

Кроме стандартного ввода атрибутов был организован упрощённый ввод. Упрощённый ввод может быть использован в обучающих целях, ускоряя процесс ввода названий атрибутов, зависимостей и их множеств. В случае использования длинных названий атрибутов пользователю предлагается возможность использования для данных атрибутов автоматически генерируемых псевдонимов. При установке флага использования псевдонимов для данного множества зависимостей все названия атрибутов заменяются на псевдонимы. Панель ввода представлена на рисунке 3.

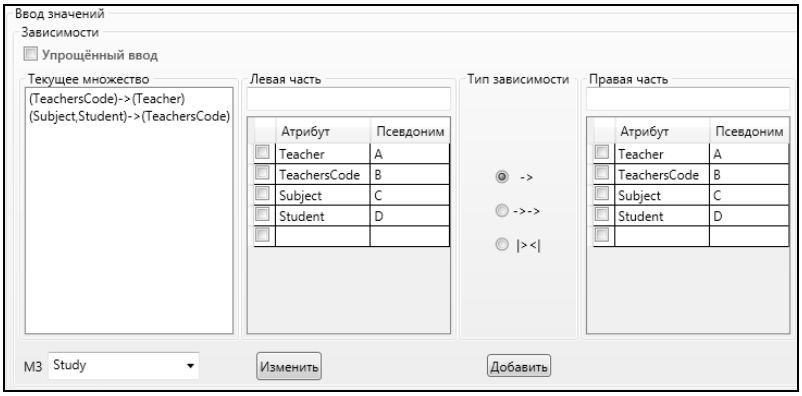

Рисунок 3 – Управление зависимостями

Сформированные множества зависимостей могут быть использованы в алгоритмах путём выбора названия множества из выпадающего списка, с последующим применением одного из алгоритмов к выбранному множеству. Выбор алгоритма на выполнение и необходимые для заполнения поля представлены на рисунке 4.

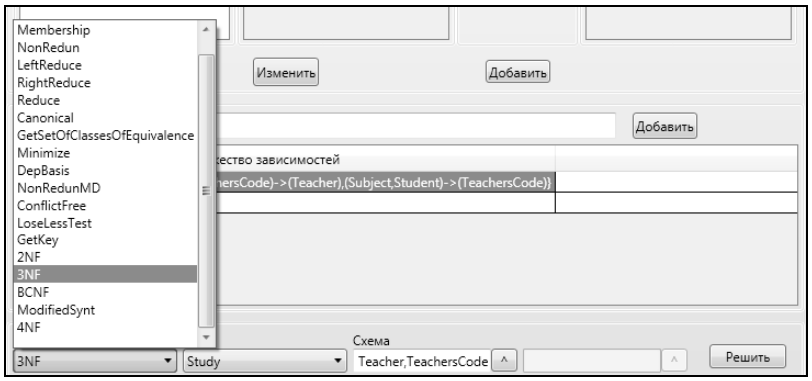

Рисунок 4 – Выбор алгоритма

Приложение предлагает интуитивный и понятный набор операций и своевременно уведомляет пользователя о неверных действиях.

Созданное приложение предусматривает два способа представления информации, полученной в результате работы алгоритма, а именно: древовидное представление (дерево состоит из узлов - выполнение алгоритмов) и представление в виде сплошного текста, доступного для копирования целиком. Помимо результатов работы алгоритмов, предусмотрено получение их детального описания, представленного в виде комментариев [3,4], что может стать полезным при использовании приложения в обучающих целях. Древовидный вывод представлен на рисунке 5.

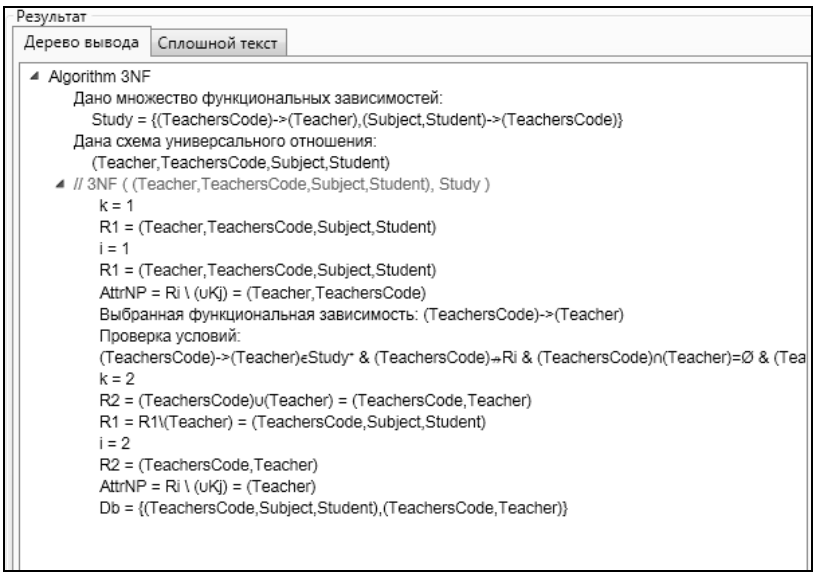

Рисунок 5 – Вывод результата в древовидной форме

Заключение

– Были реализованы алгоритмы, необходимые для нормализации реляционной базы данных, на объектно-ориентированном языке программирования с использованием событий для оповещения об изменении текущего состояния объектов.

– Объектно-ориентированный подход позволил реутилизацию кода классов, предоставляемых платформой, выделить базовую логику в отдельные классы, а также построить расширяемую систему для новых или оптимизированных алгоритмов с сохранением стабильной работы.

– Благодаря применению событийного программирования стал возможным асинхронный вызов алгоритмов с получением промежуточных результатов и поэтапного обновления интерфейса.

– Детализированное описание работы алгоритмов позволяет использовать приложение в обучающих целях.

– Удобный пользовательский интерфейс делает работу с программой непринуждённой, а древовидное представление промежуточных результатов работы алгоритмов с добавленными комментариями способствуют пониманию полученных результатов и принципов работы алгоритма.

– Были намечены пути дальнейшего совершенствования системы, а именно: использование параллельного программирования, актуального для многоядерных платформ, что позволит эффективнее использовать аппаратные ресурсы компьютера; а также попытаться уменьшить сложность экспоненциальных алгоритмов до полиномиальной, а значить, заметно сократить время работы алгоритмов, к которому предъявляются особенно высокие требования.

Библиография

1. Codd E.F. *A Relational Model of Data for large Shared Data Banks*. Comm. ACM, 1970.
2. Paredaens, J. *About Fuctional Dependencies in a Database Structure and their Coverings*. Phillips MBLE Lab. Report 342, 1977.
3. Саранчук Дориан, Чеботарь Сергей, Олейник Сергей. *Реализация алгоритмов построения функциональных зависимостей.* International Conference on Information Technologies, Systems and Networks ITSN-2012, 15 octombrie 2012, ULIM, Chişinău, Republica Moldova.
4. Саранчук Дориан, Чеботарь Сергей, Олейник Сергей. *Реализация алгоритмов нормализации баз данных.* International Conference on Information Technologies, Systems and Networks ITSN-2012, 15 octombrie 2012, ULIM, Chişinău, Republica Moldova.
5. Эндрю Троелсен. *Язык программирования C# и платформа .Net 4*.- ООО „И.Д. Вильямс”, 2011.
6. Joseph Albahari, Ben Albahari. *C# 4.0 in a nutshell,* - O’REILLY, 2010.