МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

ИНСТИТУТ ПОВЫШЕНИЯ КВАЛИФИКАЦИИ И ПЕРЕПОДГОТОВКИ КАДРОВ

КУРСОВАЯ РАБОТА

по дисциплине «Средства визуального программирования приложений»

РАЗРАБОТКА КЛИЕНТСКОЙ ЧАСТИ ПРИЛОЖЕНИЯ ИНФОРМАЦИОННОЙ СИСТЕМЫ ТОРГОВОГО ПРЕДПРИЯТИЯ

Баранок Артур Витальевич

Витебск, 2012

Реферат

Курсовая работа 32 с., 1 рис., 14 листингов, 7 источников.

ИНФОРМАЦИОННАЯ СИСТЕМА, ТОРГОВАЯ ОРГАНИЗАЦИЯ, БАЗЫ ДАННЫХ, СРЕДА РАЗРАБОТКИ, ПРОГРАММИРОВАНИЕ, QT, QT CREATOR, C++.

Объект исследования – информационная система торговой организации.

Предмет исследования – применение современных информационных технологий и средств визуального программирования для создания автоматизированных информационных систем.

Цель работы – проектирование и разработка информационной системы торговой организации.

Методы исследования: описательно-аналитический, экспериментальный, статистический.

Элементы новизны: кроссплатформенность при разработки клиентского приложения.

Теоретическая и практическая значимость: основные результаты работ могут найти применение при разработке учебных курсов вуза по программированию с использованием инструмента разработки QT.

Содержание

Введение

На протяжении всего периода применения компьютеров и компьютерных систем существует тенденция создания высоконадежных управляющих комплексов, ориентированных на получение и использование информационных ресурсов. Эта тенденция выразилась в процессе создания различных видов систем, как встроенных в уникальные объекты информационно-технологических комплексов. В последнее время информационные технологии стали неотъемлемой частью нашей жизни. Информационные системы, связанные с предоставлением и обработкой информации для всех уровней управления объектами, приобретают особую важность в общественной жизни.

Целью работы является разработка информационной системы торговой организации. Начальным этапом построения проекта является проектирование базы данных на основе свободной системы управления базами данных MYSQL, составление запросов и взаимосвязей между различными таблицами. Для построения клиентского приложения применяется библиотека Qt4 и среда разработки «Qt Creator».

1 Теоретическая основа разработки автоматизированной информационной системы

1.1 Qt - кросс-платформенный инструментарий разработчика прикладного программного обеспечения

Qt – кросс-платформенный инструментарий разработчика прикладного программного обеспечения, широко используемый для создания графических интерфейсов. Он написан на C++ и предоставляет мощные расширения этого языка. Также доступны интерфейсы для других языков программирования, таких как Python (PyQt), Ruby (Korundum/QtRuby) и Perl (PerlQt).

Отличительная особенность Qt от других библиотек — использование Meta Object Compiler (MOC) — предварительной системы обработки исходного кода. MOC позволяет во много раз увеличить мощь библиотек, вводя такие понятия, как слоты и сигналы. Кроме того, это позволяет сделать код более лаконичным. Утилита MOC ищет в заголовочных файлах на C++ описания классов, содержащие макрос Q_OBJECT, и создаёт дополнительный исходный файл на C++, содержащий метаобъектный код.

Qt позволяет создавать собственные плагины и размещать их непосредственно в панели визуального редактора. Также существует возможность расширения привычной функциональности виджетов, связанной с размещением их на экране, отображением, перерисовкой при изменении размеров окна.

Qt комплектуется визуальной средой разработки графического интерфейса «Qt Designer», позволяющей создавать диалоги и формы в режиме WYSIWYG. В поставке Qt есть «Qt Linguist» — графическая утилита, позволяющая упростить локализацию и перевод программы на многие языки; и «Qt Assistant» — справочная система Qt, упрощающая работу с документацией по библиотеке, а также позволяющая создавать кросс-платформенную справку для разрабатываемого на основе Qt ПО. Начиная с версии 4.5.0 в комплект Qt включена среда разработки «Qt

Creator», которая включает в себя редактор кода, справку, графические средства «Qt Designer» и возможность отладки приложений. «Qt Creator» может использовать GCC или Microsoft VC++ в качестве компилятора и GDB в качестве отладчика. Для Windows версий библиотека комплектуется компилятором, заголовочными и объектными файлами MinGW [1].

Библиотека разделена на несколько модулей, для четвёртой версии библиотеки это:

- QtCore — классы ядра библиотеки, используемые другими модулями;
- QtGui — компоненты графического интерфейса;
- QtNetwork — набор классов для сетевого программирования. Поддержка различных высокоуровневых протоколов может меняться от версии к версии. В версии 4.2.x присутствуют классы для работы с протоколами FTP и HTTP. Для работы с протоколами TCP/IP предназначены такие классы, как QTcpServer, QTcpSocket для TCP и QUdpSocket для UDP;
- QtOpenGL — набор классов для работы с OpenGL;
- QtSql — набор классов для работы с базами данных с использованием языка структурированных запросов SQL. Основные классы данного модуля в версии 4.2.x: QSqlDatabase — класс для предоставления соединения с базой, для работы с какой-нибудь конкретной базой данных требует объект, унаследованный от класса QSqlDriver — абстрактного класса, который реализуется для конкретной базы данных и может требовать для компиляции SDK базы данных. Например, для сборки драйвера под базу данных FireBird/InterBase требует .h файлы и библиотеки статической линковки, входящие в комплект поставки данной БД;
- QtScript — классы для работы с Qt Scripts;
- QtSvg — классы для отображения и работы с данными Scalable Vector Graphics (SVG);
- QtXml — модуль для работы с XML, поддерживается SAX и DOM модели работы;
- QtDesigner — классы создания расширений QtDesigner для своих собственных виджетов;
- QtUiTools — классы для обработки в приложении форм Qt Designer;
- QtAssistant — справочная система;

- Qt3Support — модуль с классами, необходимыми для совместимости с библиотекой Qt версии 3.x.x;
- QtTest — модуль для работы с UNIT тестами;
- QtWebKit — модуль WebKit, интегрированный в Qt и доступный через её классы;
- QtXmlPatterns — модуль для поддержки XQuery 1.0 и XPath 2.0;
- Phonon — модуль для поддержки воспроизведения и записи видео и аудио, как локально, так и с устройств и по сети;
- QtCLucene — модуль для поддержки полнотекстового поиска, применяется в новой версии Assistant в Qt 4.4;
- ActiveQt — модуль для работы с ActiveX и COM технологиями для Qt-разработчиков под Windows.
- QtDeclarative — модуль, предоставляющий декларативный фреймворк для создания динамичных, настраиваемых пользовательских интерфейсов.

В Qt используется CamelCasing: имена классов выглядят как MyClassName, а имена методов – как myMethodName. При этом имена всех классов Qt начинаются с Q, например QObject, QList или QFont. Большинству классов соответствуют заголовочные файлы с тем же именем (без расширения .h).

Для эффективной работы с классами на стадии выполнения в Qt используется специальная объектная модель, расширяющая модель C++. В частности, добавляются следующие возможности:

– древовидные иерархии объектов;

– аналог dynamic_cast для библиотеки, не использующий RTTI;

– взаимодействие объектов через сигналы и слоты;

– свойства объектов.

Многие объекты определяются значением сразу нескольких свойств, внутренними состояниями и связями с другими объектами. Они представляют собой индивидуальные сущности, и для них не имеет смысла операция буквального копирования, а также разделение данных в памяти. В Qt эти объекты наследуют свойства QObject [2].

В тех случаях, когда объект требовалось бы рассматривать не как сущность, а как значение (например, при хранении в контейнере) – используются указатели. Иногда указатель на объект, наследуемый от QObject, называют просто объектом.

При создании графических пользовательских интерфейсов взаимодействие объектов часто осуществляется через обратные вызовы, т.е. передачу кода для последующего выполнения (в виде указателей на функции, функторов, и т.д.). Также популярна концепция событий и обработчиков, в которой обработчик действует как перехватчик события определенного объекта.

В Qt вводится концепция сигналов и слотов.

Сигнал отправляется при вызове соответствующего ему метода. Программисту при этом нужно только указать прототип метода в разделе signals.

Слот является методом, исполняемым при получении сигнала. Слоты могут объявляться в разделе pulic slots, protected slots или private slots. При этом уровень защиты влияет лишь на возможность вызова слотов в качестве обычных методов, но не на возможность подключения сигналов к слотам.

Модель сигналов и слотов отличается от модели событий и обработчиков тем, что слот может подключаться к любому числу сигналов, а сигнал может подключаться к любому числу слотов. При отправке сигнала будут вызваны все подключенные к нему слоты [3].

Таким образом, для эффективной работы с классами на стадии выполнения Qt использует специальную объектную модель, в которой при помощи наследования от QObject и генерирования кода компилятором метаобъектов реализованы:

– иерархии объектов;

– специальный аналог dynamic_cast, не зависящий от RTTI;

– система сигналов и слотов;

– система свойств объектов;

– динамическая работа с классами.

1.2 Система управления базами данных MySQL

Под базами данных (БД) понимаются системы хранения и обработки данных, для доступа к которым используется язык SQL (Structured Query Language). База данных представляет собой структурированную совокупность данных. Поскольку компьютеры замечательно справляются с обработкой больших объемов данных, управление базами данных играет центральную роль в вычислениях. Реализовано такое управление может быть по-разному - как в виде отдельных утилит, так и в виде кода, входящего в состав других приложений.

MySQL - это система управления реляционными базами данных. В реляционной базе данных данные хранятся в отдельных таблицах, благодаря чему достигается высокая скорость и гибкость работы. Таблицы связываются между собой при помощи отношений, благодаря чему обеспечивается возможность объединять при выполнении запроса данные из нескольких таблиц. SQL как часть системы MySQL можно охарактеризовать как язык структурированных запросов плюс наиболее распространенный стандартный язык, используемый для доступа к базам данных.

СУБД MySQL использует традиционную архитектуру клиент/сервер, поэтому, работая с СУБД MySQL, пользователь реально работает с двумя программами.

- Программой сервера базы данных, расположенной на компьютере, где хранится база данных. Она «прослушивает» запросы клиентов, поступающие по сети, и осуществляет доступ к содержимому базы данных для предоставления информации, которую запрашивают клиенты.
- Клиентской программой, которая является программой, осуществляющей подключение к серверу и передающей запросы на сервер.

Дистрибуция СУБД MySQL включает в себя несколько клиентских программ и сервер. Клиентские программы используются в соответствии с поставленными целями. Одной из наиболее популярных и наиболее привлекательных клиентских программ является mysql. Это интерактивная программа, позволяющая создавать запросы и просматривать полученные результаты. Другими клиентскими

программами являются утилита mysqldump и mysqlimport, которые осуществляют дамп базы данных в файл и восстановление его обратно, и утилита mysqladmin, позволяющая производить проверку состояния сервера и осуществлять административные задачи, такие как выключение сервера (shut-down) [4].

По степени их универсальности различаются два вида СУБД - системы общего назначения и специализированные системы. СУБД общего назначения не ориентированны на какую - либо конкретную предметную область или на информационные потребности конкретной группы пользователей. Каждая система такого рода реализуется как программный продукт, способный функционировать на некоторой модели ЭВМ в определенной обстановки, и поставляется многим пользователям как коммерческое изделие. СУБД общего назначения обладают средствами настройки на работу с конкретной БД в условиях конкретного применения.

Использование СУБД общего назначения в качестве инструментального средства для создания автоматизированных информационных систем, основанных на технологии БД, позволяет существенно сокращать сроки разработки, экономить трудовые ресурсы. Развитые функциональные возможности таких СУБД, присущая им, как правило, функциональная избыточность позволяют иметь значительный «запас мощности», необходимый для безболезненного эволюционного развития построенных на их основе информационных систем в рамках их жизненного цикла. Вместе с тем средства настройки дают возможность достигнуть приемлемого уровня производительности информационной системы в процессе ее эксплуатации.

Однако в некоторых случаях доступные СУБД общего назначения не позволяют добиться требуемых характеристик производительности и/или удовлетворить заданные ограничения по объему памяти, предоставляемой для хранения БД.

Тогда приходится разрабатывать специализированную СУБД для данного конкретного применения. Решение указанных проблем при этом может оказаться возможным благодаря знанию специфических особенностей данного применения, к которым оказываются нечувствительными средства настройки доступных СУБД

общего назначения, либо за счет ущемления каких либо функций системы, не имеющих жизненно важного значения. Как правило, в этой роли оказываются, прежде всего функции, обеспечивающие комфортную работу пользователя.

СУБД общего назначения - это сложные программные комплексы, предназначенные для выполнения всей совокупности функций, связанных с созданием и эксплуатацией БД информационной системы. Они позволяют определить структуру создаваемой БД, инициализировать ее и произвести начальную загрузку данных. Системные механизмы выполняют также функции управления ресурсами среды хранения, обеспечения логической и физической независимости данных, предоставления доступа пользователям к БД, защиты логической целостности БД, обеспечения ее физической целостности - защиты от разрушений. Другая важная группа функций - управления полномочиями пользователей на доступ к БД, настройка на конкретные условия применения, организация параллельного доступа пользователей к базе данных в социальной пользовательской среде, поддержка деятельности системного персонала, ответственного за эксплуатацию БД.

Для создания БД разработчик описывает ее логическую структуру, организацию в среде хранения, а также способы видения базы данных пользователями. При этом используются предоставляемые СУБД языковые средства определения данных, и система настраивается на работу с конкретной БД. Такие описания БД называются соответственно схемой (или логической схемой, или концептуальной схемой) БД, схемой хранения (или внутренней схемой) и внешними схемами.

Обрабатывая схемы БД, СУБД создает пустую БД требуемой структуры - хранилище, которое можно далее наполнить данными о предметной области начать эксплуатировать для удовлетворения информационных потребностей пользователей.

Принципиально важное свойство СУБД заключается в том, что она позволяет различать и поддерживать два независимых взгляда на БД - взгляд пользователя, воплощаемой в «логическом» представлении данных, и «взгляд» системы -

«физическое» представление, характеризующее организацию хранимых данных. Пользователя не интересует при его работе с БД байты и биты, представляющие данные в среде хранения, их размещения в памяти, указателя, поддерживающие связи между структурными различными компонентами хранимых данных, выбранные методы доступа. В то же время эти факторы важны для выполнения функций управления данными самой СУБД.

Обеспечение логической независимости данных - одна из важнейших функций СУБД, предоставляющая определенную степень свободы вариации «логического» представления БД без необходимости соответствующей модификации «физического» представления. Благодаря этому достигается возможность адаптации взгляда пользователя на БД к его реальным потребностям, конструирования различных «логических» взглядов на одну и ту же «физическую» БД, что весьма важно в социальной пользовательской среде.

Под «физической» независимостью данных понимается способность СУБД предоставлять некоторую свободу модификации способов организации БД в среде хранения, не вызывая необходимости внесения соответствующих изменений в «логическое» представление. Благодаря этому вносить изменения в организацию хранимых данных, производить настройку системы с целью повышения ее эффективности, не затрагивая созданных прикладных программ, использующих базу данных. «Физическая» независимость данных реализуется в СУБД за счет тех же самых трансформационных механизмов архитектуры системы, которые обеспечивают «логическую» независимость данных.

Поддержка логической целостности (непротиворечивости) базы данных - другая важная функция СУБД. В развитых системах ограничения целостности базы данных объявляются в схеме базы данных, и их проверка осуществляется при каждом обновлении объектов данных или связей между ними, являющихся аргументами таких ограничений.

Реляционные базы данных

Реляционная модель требует от СУБД гораздо более высокого уровня сложности. В ней делается попытка избавить программиста от выполнения

рутинных операций по управлению данными, столь характерных для иерархической и сетевой моделей.

В реляционной модели база данных представляет собой централизованное хранилище таблиц, обеспечивающее безопасный одновременный доступ к информации со стороны многих пользователей. В строках таблиц часть полей содержит данные, относящиеся непосредственно к записи, а часть — ссылки на записи других таблиц. Таким образом, связи между записями являются неотъемлемым свойством реляционной модели.

В реляционной модели достигается информационная и структурная независимость. Записи не связаны между собой настолько, чтобы изменение одной из них затронуло остальные, а изменение структуры базы данных не обязательно приводит к перекомпиляции работающих с ней приложений.

В реляционных СУБД применяется язык SQL, позволяющий формулировать произвольные, нерегламентированные запросы. Это язык четвертого поколения, поэтому любой пользователь может быстро научиться составлять запросы. К тому же, существует множество приложений, позволяющих строить логические схемы запросов в графическом виде. Все это происходит за счет ужесточения требований к производительности компьютеров. К счастью, современные вычислительные мощности более чем адекватны.

Реляционные базы данных страдают от различий в реализации языка SQL, хотя это и не проблема реляционной модели. Каждая реляционная СУБД реализует какое-то подмножество стандарта SQL плюс набор уникальных команд, что усложняет задачу программистам, пытающимся перейти от одной СУБД к другой. Приходится делать нелегкий выбор между максимальной переносимостью и максимальной производительностью. В первом случае нужно придерживаться минимального общего набора команд, поддерживаемых в каждой СУБД. Во втором случае программист просто сосредоточивается на работе в данной конкретной СУБД, используя преимущества ее уникальных команд и функций.

Объектно-ориентированные базы данных

Объектно-ориентированная база данных (ООБД) позволяет программистам, которые работают с языками третьего поколения, интерпретировать все свои информационные сущности как объекты, хранящиеся в оперативной памяти. Дополнительный интерфейсный уровень абстракции обеспечивает перехват запросов, обращающихся к тем частям базы данных, которые находятся в постоянном хранилище на диске. Изменения, вносимые в объекты, оптимальным образом переносятся из памяти на диск.

Преимуществом ООБД является упрощенный код. Приложения получают возможность интерпретировать данные в контексте того языка программирования, на котором они написаны. Реляционная база данных возвращает значения всех полей в текстовом виде, а затем они приводятся к локальным типам данных. В ООБД этот этап ликвидирован. Методы манипулирования данными всегда остаются одинаковыми независимо от того, находятся данные на диске или в памяти.

Данные в ООБД способны принять вид любой структуры, которую можно выразить на используемом языке программирования. Отношения между сущностями также могут быть произвольно сложными. ООБД управляет кэш-буфером объектов, перемещая объекты между буфером и дисковым хранилищем по мере необходимости.

С помощью ООБД решаются две проблемы. Во-первых, сложные информационные структуры выражаются в них лучше, чем в реляционных базах данных, а во-вторых, устраняется необходимость транслировать данные из того формата, который поддерживается в СУБД. Например, в реляционной СУБД размерность целых чисел может составлять 11 цифр, а в используемом языке программирования — 16. Программисту придется учитывать эту ситуацию.

Объектно-ориентированные СУБД выполняют много дополнительных функций. Это окупается сполна, если отношения между данными очень сложны. В таком случае производительность ООБД оказывается выше, чем у реляционных СУБД. Если же данные менее сложны, дополнительные функции оказываются избыточными. В объектной модели данных поддерживаются нерегламентированные

запросы, но языком их составления не обязательно является SQL. Логическое представление данных может не соответствовать реляционной модели, поэтому применение языка SQL станет бессмысленным. Зачастую удобнее обрабатывать объекты в памяти, выполняя соответствующие виды поиска.

Большим недостатком объектно-ориентированных баз данных является их тесная связь с применяемым языком программирования. К данным, хранящимся в реляционной СУБД, могут обращаться любые приложения, тогда как, к примеру, Java-объект, помещенный в ООБД, будет представлять интерес лишь для приложений, написанных на Java.

Объектно-реляционные базы данных

Объектно-реляционные СУБД объединяют в себе черты реляционной и объектной моделей. Их возникновение объясняется тем, что реляционные базы данных хорошо работают со встроенными типами данных и гораздо хуже — с пользовательскими, нестандартными. Когда появляется новый важный тип данных, приходится либо включать его поддержку в СУБД, либо заставлять программиста самостоятельно управлять данными в приложении.

Перестройка СУБД с целью включения в нее поддержки нового типа данных — не лучший выход из положения. Вместо этого объектно-реляционная СУБД позволяет загружать код, предназначенный для обработки "нетипичных" данных. Таким образом, база данных сохраняет свою табличную структуру, но способ обработки некоторых полей таблиц определяется извне, т.е. программистом.

2 Проектирование и разработка клиентского приложения информационной системы торговой организации

2.1 Создание структуры базы данных MySQL торговой организации

Процесс разработки (проектирования) базы данных включает два этапа: разработку логической организации базы данных и создание ее на носителе. Логическая организация базы данных - это предоставление пользователя о предметной области, информация о которой должна храниться в базе данных.

Под физической организацией базы данных понимается совокупность средств и методов размещения данных во внешней памяти и на их основе внутренняя модель данных. Внутренняя модель является средством отображения логической модели данных, показывает, каким образом записи размещаются в базе данных, как они упорядочиваются, как организуются связи, каким путем можно осуществить выборку и так далее.

Проектирование базы данных торговой организации начиналось с создания всех нужных таблиц в базе, всех полей входящих в каждую таблицу, взаимодействия таблиц между собой с помощью специальных отношений (один ко многим, многие ко многим) и создание в соответствии с этими параметрами первичных и вторичных ключей.

Структура базы данных информационной системы торговой организации:

I. Торговая точка (trade)

1. Номер точки (trade_id)
2. Тип торговой точки (универмаг, магазин, киоск) (trade_type)
3. Число торговых залов (trade_number)
4. Платежи за аренду (стоимость) (trade_rent)
5. Коммунальные услуги (стоимость) (trade_utilities)

II. Продавец (seller)

1. Номер (seller_id)

2. Фамилия (seller_family)

3. Имя (seller_name)

4. Оклад (seller_oklad)

5. Код торговой точки (trade_id)

III. Заявка (application)

1. Номер заявки (application_id)

2. Наименование товара (application_description)

3. Количество (application_sum)

4. Стоимость (application_cost)

5. Дата заявки (application_date)

5. Код товара (catalog_id)

6. Код поставщика (distributor_id)

IV. Поставщик (distributor)

1. Номер (distributor_id)

2. Категория (фирма, дилер) (distributor_category)

3. Адрес (distributor_adress)

4. Название (distributor_name)

V. Покупатель (buyer)

1. Номер покупателя (buyer_id)

2. Имя покупателя (buyer_name)

3. Фамилия покупателя (buyer_family)

4. Тип (частное или физическое лицо) (buyer_type)

VI. Продажи (sales)

1. Номер покупки (sales_id)

2. Дата покупки (sales_date)

3. Количество единиц (sales_sum)

4. Цена (sales_price)

5. Код товара (catalog_id)

6. Код продавца (seller_id)

7. Код покупателя (buyer_id)

VII. Справочник товаров (номенклатура, склад) (catalog)

1. Номер товара (catalog_id) - первичный

2. Название товара (catalog_name)

3. Количество (catalog_sum)

4. Цена (catalog_price)

VIII. Распределение (raspredelenie)

1. Код товара (catalog_id)

2. Код точки (trade_id)

2.2 Разработка клиентского приложения в Qt Creator

В библиотеке Qt есть отдельный модуль, предоставляющий удобный функционал использования базы данных — QtSql.

К уровню драйверов относятся классы для получения данных на физическом уровне, такие, как:

- QSqlDriver;
- QsqlDriverCreator <T*>;
- QSqlDriverCreatorBase;
- QSqlDriverPlugin;
- QSqlResult.

QSqlDriver является абстрактным базовым классом, предназначенный для доступа к специфичным БД. Класс не должен быть использован «прямо», взамен нужно воспользоваться QsqlDatabase. Для создания собственного драйвера SQL можно наследовать функции от QSqlDriver и реализовать необходимые виртуальные функции.

QSqlDriverCreator — шаблонный класс, предоставляющий фабрику SQL драйвера для специфичного типа драйвера. Шаблонный параметр должен быть подклассом QSqlDriver.

QSqlCreatorBase — базовый класс для фабрик SQL драйверов, чтобы возвращать экземпляр специфичного подкласса класса QSqlDriver.

QSqlDatabase несет ответственность за загрузку и управление плагинов драйверов баз данных. Когда база данных добавлена с помощью функции QSqlDatabase::addDatabase()), необходимый плагин драйвера загружается, используя QSqlDriverPlugin. QSqlDriverPlugin предоставляет собой абстрактный базовый класс для пользовательских QSqlDriver плагинов. [5]

QSqlResult предоставляет абстрактный интерфейс для доступа к данным специфичных БД. В курсовом проекте мы используем QSqlQuery вместо QSqlResult, поскольку QSqlQuery предоставляет обертку для БД-специфичных реализации QSqlResult.

В проекте файла ipk2012.pro записываются все основные подключения и название проекта. Для того чтобы, проект мог работать с базой банных и обрабатывать запросы на sql в первой строке указывается код: sql: QT += core gui sql.

В файле mainwindow.h указывается подключение нужных библиотек, описание выбранного класса, описание конструктора, описание записанных слотов (записываются автоматически при создании), описание соответствующих указателей и функций (листинг 3.1).

Листинг 3.1 – Подключение библиотек в QT

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H
#include <QMainWindow>                окно Qmainwindow
#include <QTextCodec>                 кодеки для использования русских симовлов
#include <QtSql/QSqlDatabase>         создание или подключение базы
#include <QtSql/QSqlRelationalTableModel> создание реляционной таблицы модели
#include <QApplication>               создание аппликации
#include <QDataWidgetMapper>          создание определённых виджетов
```

```
namespace Ui {
class MainWindow;
}
class MainWindow : public QMainWindow   описание класса и наследование
{
   Q_OBJECT
public:
   explicit MainWindow(QApplication *a, QWidget *parent = 0);   описание
конструктора
   ~MainWindow();                          описание деструктора
private slots:                             описание слотов
   void on_actionExit_triggered();
   void on_action_triggered();
   void on_action_2_triggered();
   void on_action_3_triggered();
   void on_action_4_triggered();
   void on_action_5_triggered();
   void on_action_6_triggered();
   void on_action_7_triggered();
   void on_action_8_triggered();
   void on_action_9_triggered();
   void on_action_10_triggered();
   void on_action_11_triggered();
   void on_action_12_triggered();
   void on_action_13_triggered();
   void on_action_14_triggered();
   void on_action_15_triggered();
private:
   Ui::MainWindow *ui;                     указатель на класс
   QTextCodec *codec;                      указатель на кодеки
   QSqlDatabase *db;                       указатель на подключение или создание
   QAbstractTableModel *model;             указатель на абстрактную модель таблицы
   bool createConnection();                функция создания подключения
   void createRelationalTables();          функция создания реляционной таблицы
   void initializeModel();                 функция инициализации модели
};
        #endif // MAINWINDOW_H
```

В файле mainwindow.cpp записывается подключения нужных библиотек, файлов, а также весь процесс реализации данного класса mainwindow (листинг 3.2).

Листинг 3.2 – Реализация класса mainwindow

```
#include "mainwindow.h"
#include "ui_mainwindow.h"              подключения файла данной формы
#include <QtSql/QSqlDatabase>           создания или подключение базы
```

```
#include <QtSql/QSqlError>            ошибок подключения
#include <QtSql/QSqlQuery>            ошибок запросов
#include <QMessageBox>                текстовых сообщений
#include <QtSql/QSqlTableModel>       создания реляционной таблицы
#include <QDataWidgetMapper>          создания определённых виджетов
#include <QFile>                      создания для работы с файлами
#include <QSqlRelationalDelegate>     создания реляционного делегата типа sql
#include <QDate>                      создания функции для работы с датами в Qt4
#include <QApplication>               создания аппликации
#include <QPushButton.h>              создания виджета типа QPushButton
#include <QPrinter>                   создания класса Qprinter
#include <QTextDocument>              создания класса QTextDocument
#include <QPrintDialog>               создания класса QPrintDialog
#include <QTextCodec>                 создания кодеков для отображения  текста
#include "dialog.h"                   подключение файлов формы диалог
#include "dialog2.h"

MainWindow::MainWindow(QApplication *a, QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)        процесс реализации класса и конструктора с
параметрами и выделения памяти
{
    ui->setupUi(this);                указатель на setupUi
    QTextCodec *codec = QTextCodec::codecForName("CP1251");
    codec = QTextCodec::codecForName("CP1251"); установка кодировки cp1251 для
отображения русских символов среде Qt
    QTextCodec::setCodecForTr(codec);
    QTextCodec::setCodecForCStrings(codec);
    QTextCodec::setCodecForLocale(codec);
    ui->tw->horizontalHeader()->setResizeMode(QHeaderView::Stretch);    установка
горизонтальной шапки таблицы
    ui->tw->setItemDelegate(new QSqlRelationalDelegate(ui->tw)); создание
реляционного делегата типа sql и выделения для него памяти
    ui->tw->setWindowTitle("Информационная система торговой организации");
    QSqlDatabase db=QSqlDatabase::addDatabase("QMYSQL"); подключение к MYSQL
    db.setHostName("127.0.0.1"); Имя хоста
    db.setPort(3306); Порт
    db.setDatabaseName("ipk2012"); Название базы данных MYSQL
    db.setUserName("root"); Имя пользователя базы данных MYSQL
    db.setPassword("12345"); Пароль
    if (!db.open())
    {
        QMessageBox::critical(parent,QObject::tr("Database Error"),db.lastError().text());
    }     Процесс вывода ошибки при невозможности подключиться к базе данных
```

```
    QSqlTableModel *model=new QSqlTableModel;    создание реляционной модели и выделение для неё памяти
    model->setTable("trade");
    if (model->lastError().isValid() ) процедура вывода ошибки
    {
      QMessageBox::critical(parent,QObject::tr("Query Error"),model->lastError().text());
    }
    model->setHeaderData(0, Qt::Horizontal, QObject::tr("Номер точки"));
    model->setHeaderData(1, Qt::Horizontal, QObject::tr("Тип торговой точки"));
    model->setHeaderData(2, Qt::Horizontal, QObject::tr("Число торговых залов"));
    model->setHeaderData(3, Qt::Horizontal, QObject::tr("Платежи за аренду, руб."));
    model->setHeaderData(4, Qt::Horizontal, QObject::tr("Коммунальные услуги, руб."));                Описание шапки таблицы с помощью функции setHeaderData
    model->select();    Функция выборки
    ui->tw->setModel(model); Указатель на модель
    this->model = model;
    ui->tw->show();  Функция show-вывод на экран
    model->submitAll(); Функция submitall-представляет запись всех изменений
}
MainWindow::~MainWindow()  Деструктор класса
{
    delete ui;
}
```

Процедура выполнения готовых всех классов выполняется в файле main.cpp (листинг 3.3).

Листинг 3.3 – Выполнение классов в файле main.cpp

```
#include <QtGui/QApplication>          подключение библиотеки QApplication
#include "mainwindow.h"                подключение файла mainwindow.h
#include <QTextCodec>                  подключение библиотеки QTextCodec
int main(int argc, char *argv[])
{
    QApplication a(argc, argv);   выполнение класса   QApplication a
    MainWindow w(&a);             выполнение класса    MainWindow w(&a)
    w.show();                     вывод на экран класс    w.show();
    return a.exec();
}
```

Для вывода таблиц используется виджет Qtableview (модель таблицы для отображения таблиц базы данных и запросов). Описывая слот и процедуру, применяется обработка события мыши типа щелчок. Все данные процедуры

события автоматически записываются в файл mainwindow.cpp. Здесь указывается алгоритм вывода таблицы или определенного запроса (листинг 3.4).

Листинг 3.4 – Вывод таблиц с помощью функции QSqlTableModel

```
void MainWindow::on_actionExit_triggered()
{
   QMessageBox::information(0,"Info","exit pressed.");
}
void MainWindow::on_action_triggered()
{
   QSqlTableModel *model=new QSqlTableModel;
   model->setTable("trade");
   model->setHeaderData(0, Qt::Horizontal, QObject::tr("Номер точки"));
   model->setHeaderData(1, Qt::Horizontal, QObject::tr("Тип торговой точки"));
   model->setHeaderData(2, Qt::Horizontal, QObject::tr("Число торговых
залов"));
   model->setHeaderData(3, Qt::Horizontal, QObject::tr("Платежи за аренду,
руб."));
   model->setHeaderData(4, Qt::Horizontal, QObject::tr("Коммунальные услуги,
руб."));
   model->select();
   ui->tw->setModel(model);
   delete this->model;
   this->model = model;
ui->tw->show();
}
```

Текст обработки события для добавления строки имеет такой код (листинг 3.5):

Листинг 3.5 – Добавление строки

```
void MainWindow::on_action_8_triggered()
   {
  model->insertRow(model->rowCount());
   }
```

При удалении строки необходимо ввести параметр currentIndex для того, чтобы происходил процесс удаления текущей строки (удаление строки):

Листинг 3.6 – Удаление строки

```
void MainWindow::on_action_9_triggered()
  {
```

```
        model->removeRow(ui->tw->currentIndex().row());
    }
```

Для выполнения однотабличных и многотабличных запросов в клиентском приложении использовался простой вывод таблицы в окне mainwindow по выбранным запросам и диалоговые окна с функцией выбора параметра.

Для реализации диалогового окна в визуальном файле dialog.ui указываются нужные виджеты: label (текст описания поля) и lineedit (поле для введения данных пользователем) (рис.1).

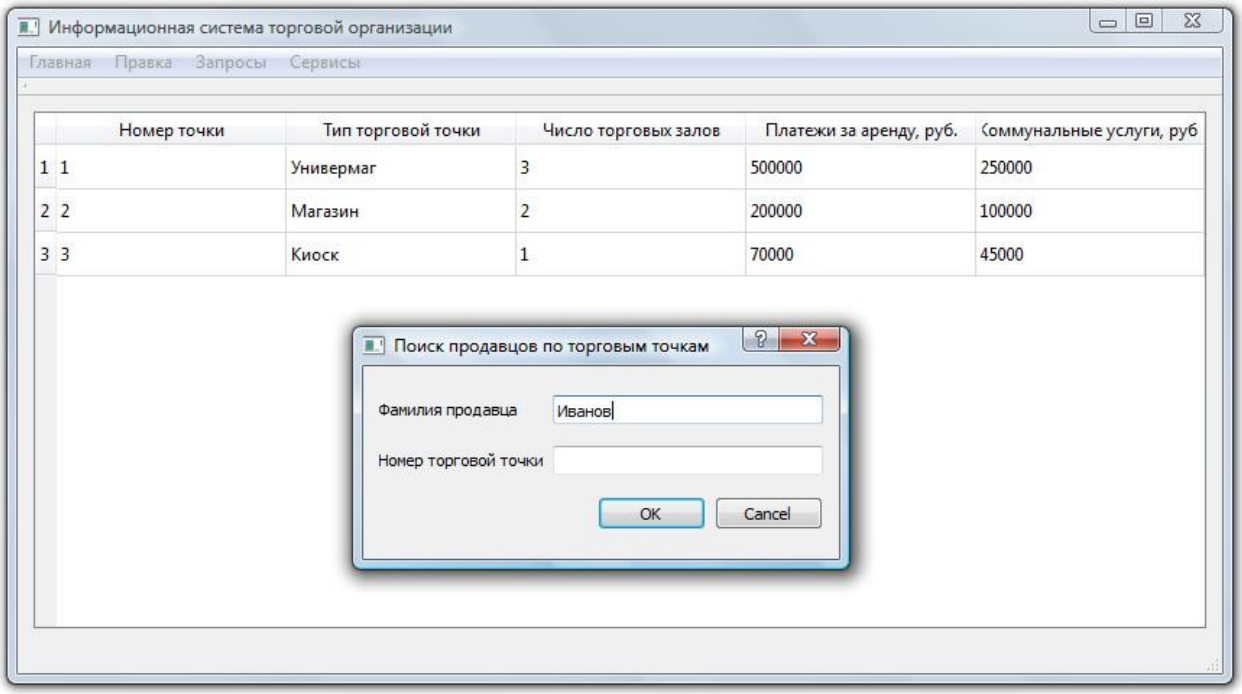

Рисунок 1. Диалоговое окно

Листинг 3.7 – Описание файла dialog.h

```
#ifndef DIALOG_H
#define DIALOG_H
#include <QDialog>                подключение библиотеки Qdialog
namespace Ui {
class Dialog;
}

class Dialog : public QDialog     описание класса и наследование Qdialog
{
   Q_OBJECT
public:
```

```
    explicit Dialog(QWidget *parent = 0);  конструктор типа dialog
    ~Dialog();                              деструктор типа dialog
    QString search();QString search_2();QString search_3();QString search_4();
```
описание функций типа qstring для записи текста из виджета типа lineedit
```
  private slots:
private:
    Ui::Dialog *ui;                  указатель ui в форме dialog
};
#endif // DIALOG_H
```

В файле dialog.cpp подключается заголовочный файл dialog.h и ui_dialog.h. (листинг 3.8). Здесь происходит процесс реализации класса и конструктора dialog, выделение записи для конструктора и реализация функции QString search().

Листинг 3.8 – Реализации класса, конструктора dialog и функции QString search()

```
#include "dialog.h"
#include "ui_dialog.h"
Dialog::Dialog(QWidget *parent) :
    QDialog(parent),
    ui(new Ui::Dialog)
{
    ui->setupUi(this);
}
Dialog::~Dialog()
{
    delete ui;
}
QString Dialog::search()
{
    return ui->lineEdit->text();
}
QString Dialog::search_2()
{
    return ui->lineEdit_2->text();
}
```

В ходе разработки приложения были разработаны следующие типы запросов:

1. Поиск продавцов по критерию «Фамилия» и номеру торговой точки (листинг 3.9)

Листинг 3.9 – Тип запроса «Поиск продавцов по критерию «Фамилия» и номеру торговой точки»

```
void MainWindow::on_action_10_triggered()
{
    Dialog dialog;
    if (dialog.exec())
    {
    QString p=dialog.search();QString d=dialog.search_2();
    QSqlError sqlError;
    QSqlQueryModel *model = new QSqlQueryModel;
    if (d=="") {    model->setQuery("select seller_id, seller_family, seller_name,
seller_oklad, trade_id from seller WHERE seller_family = \'"+p+"\'  ");
        model->setHeaderData(0, Qt::Horizontal, QObject::tr("Номер"));
        model->setHeaderData(1, Qt::Horizontal, QObject::tr("Фамилия
продавца"));
        model->setHeaderData(2, Qt::Horizontal, QObject::tr("Имя продавца"));
        model->setHeaderData(3, Qt::Horizontal, QObject::tr("Оклад"));
        model->setHeaderData(4, Qt::Horizontal, QObject::tr("Номер торговой
точки"));
    }
    if(d!="") {
    model->setQuery("select seller_id, seller_family, seller_name, seller_oklad,
trade_id from seller WHERE seller_family = \'"+p+"\' and trade_id =\'"+d+"\' ");
    model->setHeaderData(0, Qt::Horizontal, QObject::tr("Номер"));
    model->setHeaderData(1, Qt::Horizontal, QObject::tr("Фамилия продавца"));
    model->setHeaderData(2, Qt::Horizontal, QObject::tr("Имя продавца"));
    model->setHeaderData(3, Qt::Horizontal, QObject::tr("Оклад"));
    model->setHeaderData(4, Qt::Horizontal, QObject::tr("Номер торговой
точки"));
    }
    sqlError = model->lastError();
    if (sqlError.type()!=QSqlError::NoError)
    {
        QMessageBox::critical(this,"SQL Error", sqlError.text());
    }
//! [3]
    ui->tw->setModel(model);
    delete this->model;
    this->model = model;
    }
}
```

2. Общее количество товаров на складе (листинг 3.10)

Листинг 3.10 – Тип запроса «Количество товаров на складе»

```
void MainWindow::on_action_11_triggered()
```

```
{
    QSqlQueryModel *model=new QSqlQueryModel;
    model->setQuery("SELECT catalog_id, catalog_name, catalog_sum FROM
catalog");
    model->setHeaderData(0, Qt::Horizontal, QObject::tr("Номер товара"));
    model->setHeaderData(1, Qt::Horizontal, QObject::tr("Название товара"));
    model->setHeaderData(2, Qt::Horizontal, QObject::tr("Количество"));
    ui->tw->setModel(model);
    delete this->model;
    this->model = model;
    ui->tw->show();
}
```

3. Количество оформленных покупателями товаров с сортировкой по полю дата (листинг 3.11)

Листинг 3.11 – Тип запроса «Оформленные заказы с сортировкой по дате»

```
void MainWindow::on_action_12_triggered()
{
    QSqlQueryModel *model=new QSqlQueryModel;
    model->setQuery("SELECT catalog_id, sales_sum, sales_price, sales_date
from sales ORDER BY sales_date DESC");
    model->setHeaderData(0, Qt::Horizontal, QObject::tr("Номер товара"));
    model->setHeaderData(1, Qt::Horizontal, QObject::tr("Количество
товаров"));
    model->setHeaderData(2, Qt::Horizontal, QObject::tr("Стоимость
товара"));
    model->setHeaderData(3, Qt::Horizontal, QObject::tr("Дата оформления
заказа"));
    ui->tw->setModel(model);
    delete this->model;
    this->model = model;
    ui->tw->show();
}
```

4. Оформленные товары с фамилией и именем покупателей с сортировкой по дате (многотабличный запрос) (листинг 3.12)

Листинг 3.12 – Тип запроса «Оформленные товары покупателей с сортировкой по дате»

```
void MainWindow::on_action_13_triggered() // многотабличный запрос
{
    QSqlQueryModel *model=new QSqlQueryModel;
```

```
            model->setQuery("SELECT buyer.buyer_name, buyer.buyer_family,
catalog.catalog_name, sales.sales_date from buyer, catalog, sales WHERE buyer.buyer_id
= catalog.catalog_id AND catalog.catalog_id = sales.sales_id ORDER BY sales_date
DESC");
            model->setHeaderData(0, Qt::Horizontal, QObject::tr("Имя"));
            model->setHeaderData(1, Qt::Horizontal, QObject::tr("Фамилия"));
            model->setHeaderData(2, Qt::Horizontal, QObject::tr("Название товара"));
            model->setHeaderData(3, Qt::Horizontal, QObject::tr("Дата оформления
заказа"));
            ui->tw->setModel(model);
            delete this->model;
            this->model = model;
            ui->tw->show();
        }
```

5. Сумма и количество проданных товаров в торговых точках (многотабличный запрос с реализацией диалогового окна) (листинг 3.13)

Листинг 3.13 – Тип запроса «Сумма и количество проданных товаров»

```
void MainWindow::on_action_14_triggered()
{
   Dialog2  dialog2;
   if (dialog2.exec())
   {
      QString p=dialog2.search();QString d=dialog2.search_2();
   QSqlError sqlError;
   QSqlQueryModel *model = new QSqlQueryModel;
   if ( d=="" ) {model->setQuery("select trade.trade_id, trade.trade_type,
seller.seller_family, sum(sales.sales_price) as saleprice01, sum(sales.sales_sum) as
saleprice02 from trade, seller, sales where trade.trade_id=\'"+p+"\' and
trade.trade_id=seller.seller_id and seller.seller_id=sales.sales_id;") ;
      model->setHeaderData(0, Qt::Horizontal, QObject::tr("Номер торговой точки"));
      model->setHeaderData(1, Qt::Horizontal, QObject::tr("Название торговой
точки"));
      model->setHeaderData(2, Qt::Horizontal, QObject::tr("Продавец точки "));
      model->setHeaderData(3, Qt::Horizontal, QObject::tr("Общая сумма проданных
товаров"));
      model->setHeaderData(4, Qt::Horizontal, QObject::tr("Количество проданных
товаров"));}
   if(d!=""){ model->setQuery("select trade.trade_id, trade.trade_type, seller.seller_family,
sum(sales.sales_price) as saleprice01, sum(trade.trade_rent) as saleprice03 from trade,
seller, sales where trade.trade_id=\'"+p+"\' and trade.trade_type=\'"+d+"\' and
trade.trade_id=seller.seller_id and seller.seller_id=sales.sales_id;") ;
      model->setHeaderData(0, Qt::Horizontal, QObject::tr("Номер торговой точки"));
```

```
    model->setHeaderData(1, Qt::Horizontal, QObject::tr("Название торговой
точки"));
    model->setHeaderData(2, Qt::Horizontal, QObject::tr("Продавец точки "));
    model->setHeaderData(3, Qt::Horizontal, QObject::tr("Общая сумма проданных
товаров"));
    model->setHeaderData(4, Qt::Horizontal, QObject::tr("Оплата за аренду"));}
  sqlError = model->lastError();
  if (sqlError.type()!=QSqlError::NoError) {
    QMessageBox::critical(this,"SQL Error", sqlError.text());
  }
//! [3]
  ui->tw->setModel(model);
  delete this->model;
  this->model = model;
  }
      }
```

6. Дополнительный запрос с функцией вывода на печать с сохранением текста в формате .pdf (листинг 3.14)

Листинг 3.14 – Тип запроса «Вывод на печать»

```
      void MainWindow::on_action_41_triggered()
      {
        ui->tw->setModel(model);      указатель модель, использованную в данный
момент
        QPrinter printer;             создание класса типа Qprinter
        QTextDocument doc;            создание класса типа  QTextDocument
        QString html;                 создание переменной типа QString
        QPrintDialog printDialog(&printer,this);   создание класса QPrintDialog  с
параметрами  класса  QPrinter , с помощью которого  происходит вывод на печать
        if (printDialog.exec()) {
          html+="<table  bordercolor='green' border='2' width='100%'cols='10'>";
       html += "<tr>";                установка горизонтальной шапки таблицы
            for (int l=0,p=model->columnCount();l<p;++l) {
             html += "<td>" + model->headerData(l,Qt::Horizontal,0).toString() +
"</td>";
            }
             html += "</tr>";

          for (int i=0,n=model->rowCount();i<n;++i) {
      html += "<tr>";
           for (int j=0,m=model->columnCount();j<m;++j) {
            html += "<td>" + model->data(model->index(i,j)).toString() + "</td>";
           }
```

```
            html += "</tr>";
        }
        html+="</table>";           закрывающийся тег таблицы
        doc.setHtml(html);          выполнение функции переноса нашей
таблицы в текстовый документ .doc
        printer.setOutputFormat(QPrinter::PdfFormat);  перевод в формат .pdf
        printer.setOutputFileName("example.pdf");      сохранение текста в
формате pdf
        doc.print(&printer);        выполнение печати
    }
    }
```

Таким образом, основным преимуществом Qt является возможность запускать написанное приложение на большинстве современных операционных систем путём простой компиляции программы для каждой ОС без изменения исходного кода. Qt является полностью объектно-ориентированным, легко расширяемым и поддерживающим технику компонентного программирования.

Заключение

В процессе выполнения курсовой работы изучены взаимосвязи в информационной структуре торгового предприятия, проанализированы различные запросы к базе данных определенных таблиц. На основе полученных данных было спроектировано и разработано клиентское приложение с использованием кросс-платформенного инструментария Qt.

В теоретической части курсовой работы рассмотрены основы разработки автоматизированной информационной системы: процесс создания, проектирования и использования базы данных MYSQL, реализация различных классов при разработке приложений с использованием библиотеки Qt.

В практической части курсовой работы реализована структура базы данных торгового предприятия и описан процесс создания клиентского приложения с использованием программы Qt.

Основные результаты и практические примеры курсовой работы могут найти применение при планировании практических заданий курсов «Базы данных» и «Визуальное программирование» в высших учебных заведениях.

Список использованных источников

1. В. Гольцман. MYSQL5. Библиотека программиста. Издательство: П. Год: 2010. – 253 с.

2. Жасмин Бланшет. Программирование gui на c++.: «Кудиц-пресс-Москва» - 2008г.

3. Макс Шлее. Qt4. Профессиональное программирование на C++. —

СПб.: БХВ-Петербург, 2007. — 880 с.

4. М.Кузнецов. Самоучитель MYSQL5. : «БХВ-Петербург» - 2007 г.

5. Свободная энциклопедия Википедия [Электронный ресурс] / Режим доступа: http://ru.wikipedia.org/wiki/Qt. – Дата доступа: 10.08.2012.

6. Юрий Земсков. Программирование на C++ с использованием библиотеки Qt 4. г. Волгоград 2007 г.

7. developerWorks Россия [Электронный ресурс] / http://www.ibm.com/developerworks/ru/library/l-qt_1/ – Дата доступа: 01.09.2012.