ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ»
(Н И У « Б е л Г У »)

ИНСТИТУТ ИНЖЕНЕРНЫХ ТЕХНОЛОГИЙ И ЕСТЕСТВЕННЫХ НАУК

КАФЕДРА ОБЩЕЙ МАТЕМАТИКИ

Разработка программных методов поиска неявных связей в темпоральных базах данных

Выпускная квалификационная работа
обучающегося по направлению подготовки
01.03.02, Прикладная математика и информатика
очной формы обучения,
группы 07001305
Сидоренко Владимира Владимировича

Научный руководитель
д.т.н., профессор
Аверин А.Г.

БЕЛГОРОД 2017

Оглавление

ВВЕДЕНИЕ

В последнее время рост информатизации и компьютеризации различных коммерческих, производственных, государственных и других структур значительно усилился. В основу их деятельности легло быстрое принятие взвешенных решений. Однако попытки улучшить процессы принятия решений нередко встречаются с огромным объемом данных предоставленных в сложной структуре и содержащихся в разнообразных информационных системах.

Базы данных выступают лишь в роли памяти, пользователь может извлечь из хранилища небольшую часть хранимой информации в ответ на точно заданный запрос. И когда перед нами огромный поток данных, приходится, как можно более эффективно применять эту информацию, чтобы найти спрятанное в ней знание, которое впоследствии может помочь улучшить управление некоторыми процессами, и привести деятельность организации к более качественному виду.

Методы поиска знаний (data mining) помогают нам снизить остроту проблемы. Многие организации увеличивают доход и повышают эффективность своей деятельности, за счёт использования современных методов анализа и поиска знаний в исходных данных. Нередко это так же позволяет уменьшить затраты и увеличить количество клиентов. Данные методы уже давно активно используются в маркетинге, анализе рынка, прогнозе котировок и во многих других бизнес-приложениях.

Быстродействие любого современного компьютера, даже мобильного, во много раз превосходит вычислительный потенциал мозга человека. Но ни одна программа до сих пор не в состоянии учесть всего многообразия различных факторов, так как это делает ассоциативное мышление человека. Поэтому как постановщик задачи Человек пока, что превосходит возможности компьютера.

Однако если попытаться направить вычислительную мощность современной техники в нужное русло, заданное человеком-аналитиком, огромное преимущество компьютеров в быстродействии обязано привести к качественному прорыву в поисках новых знаний.

Основной целью выпускной квалификационной работы является разработка программных методов поиска скрытых зависимостей в реляционной базе данных, за основу которой взяты сведения о заказах коммерческой фирмы, являющейся представителем ресторанного бизнеса.

Разрабатываемые методы позволят выявить новые детали продаж, и лучше понять спрос.

Задачи, решаемые в работе:

- изучение и анализ состояния вопроса;
- разработка программного обеспечения реализующего поиск скрытых закономерностей в базе данных;
- тестирование и отладка программного обеспечения.

Объектом исследования являются данные о клиентских заказах предприятия общественного питания.

Предметом исследования являются методы поиска скрытых закономерностей, а так же современные средства разработки десктопных приложений.

Актуальность данной разработки обусловлена её эффективностью, простотой применения и экономичностью при внедрении на малых предприятиях.

1. Анализ состояния вопроса

1.1 Интеллектуальный анализ данных

Под поиском знаний в базах данных подразумевают процесс анализа больших объемов информации с привлечением программных средств автоматизированного перебора данных в попытке обнаружить скрытые в данных связи или закономерности. Выделим несколько важных этапов процесса анализа данных с помощью методов data mining:

В первую очередь необходимо привести данные к форме, пригодной для обработки системами KDD.

Для этого выбираем набор количественных параметров, описывающих данную область. Выбор рассматриваемых параметров всегда остается за человеком, а значения параметров вычисляет прикладное программное обеспечение. Как только мы выбрали описывающие параметры, данные стоит отобразить в виде прямоугольной таблицы, в которой где каждая строка отображает конкретный объект или состояние объекта, а каждый столбец – свойства или значения всех исследуемых объектов. Как правило, все существующие системы KDD работают именно с такими прямоугольными таблицами. Если же данные размещены в реляционной базе данных, то все равно стоит привести их в прямоугольную форму

Переходим к работе с таблицами. Прямоугольная форма, которую мы получили, все еще является слишком сырым материалом для применения методов KDD, все входящие, в нее данные перед анализом нужно предварительно обработать. Они могут быть в разных форматах, неполными или наоборот избыточными. Если данных слишком много стоит ограничить количество полей. Некоторые строки скорее всего неинформативны: многие из них имеют одинаковое значение поля, или же, количество строк приблизительно равно количеству столбцов. В конце концов, столбцов может быть очень много, и если мы будем пытаться анализировать всё сразу, то это не даст никакого результата, так как для большинства методов KDD свойственна сильная зависимость времени счета от количества аргументов,

так что необходимо выбрать самые значимые параметры для исследования. Но существует не только избыточность полей, но и избыточность записей. Часто из базы данных при крайне большом количестве строк их выбирают случайным образом или берут каждую n-ю запись таблицы, или же выбирают определенный промежуток. Количество записей сильно зависит от метода анализа, но в основном, записей должно быть не менее 300 и не более нескольких сотен тысяч. В большинстве случаев к данным предъявляется одно строгое требование: для строки должно быть известно значение каждого столбца. В таком случае приходится восполнять недостающие значения. Самым очевидным способом является заполнение отсутствующих показаний средним значением. Не стоит забывать, что любая реальная база данных как правило содержит ошибки, неточно определенные значения, записи, подходящие к каким-то редким, исключительным ситуациям, и прочие дефекты, которые могут резко изменить всю картину работы. Такие записи необходимо отбросить.

Лучшим случаем является тот, когда показатели никак не коррелируют друг с другом. Но на практике подобное встречается редко. Если же поля сильно коррелируют друг с другом, стоит взять одно из них. Вот наглядный пример прямоугольной формы для неких данных:

таблицы для ценных бумаг:

Цена при открытии торгов	Цена при закрытии	Миним. цена за день	Максим. цена за день

Здесь видна сильная корреляция полей. Поэтому вместо такой таблицы можно построить следующую:

Среднее значение цены за день	Изменение цены за день

Следующим этапом подготовки является нормализация данных. Нужно стараться приближать данные равномерному распределению. Чаще всего используют следующий подход нормализации: пусть есть a - среднее и σ - дисперсия. Тогда $x \to \frac{x-a}{\sigma}$. Такое преобразование проводят перед подачей данных на вход нейросетей. Если в таблице есть качественная переменная которая, например, может принимать три разных значения, то лучше всего включить вместо нее 3 других переменных, которые будут принимать значение от 0 до 1.

Основной этап – применение непосредственно алгоритмов KDD. Сценарии этого применения могут быть абсолютно разными или включать комбинирование разных методов, особенно если применяемые методы позволяют изучать данные с разных точек зрения. Именно данный этап анализа и принято называть data mining.

Последний - этап - тестирование полученных результатов. Самый простой и в то же время, часто встречающийся способ состоит в том, что поток всех имеющихся данных, которые предстоит изучить, разбивают на две контрольные группы. Одна из них должна иметь намного больший объем, чем вторая. На первой группе применяются различные алгоритмы анализа, так, мы создаём модели зависимости и все, что требуется в условиях нашей задачи. Далее на второй группе мы проверяем точность построенной модели. По совпадениям между первой и второй группой, мы можем судить, насколько адекватна и статистически значима построенная модель.

Остаётся интерпретировать знания, полученные в процессе исследования в формат удобный для восприятия, в целях их дальнейшего применения при принятии решений, добавление найденных связей и закономерностей в базы знаний и т.д.

Стоит учитывать, что работать с данными намного удобнее, когда имеется возможность интегрировать такие компоненты как – визуальное представление, графический интерфейс, инструменты формирования запросов, оперативный анализ и другие вспомогательные средства, которые позволят лучше понимать данные и отображать результаты, и, конечно, сами алгоритмы, на основе которых строятся модели.

1.2 Классификация систем интеллектуального анализа данных.

Поиск скрытых знаний как область включает в себя множество дисциплин, она возникла и развивалась под влиянием достижений таких

направлений как: прикладная статистика, разработка искусственного интеллекта, наука распознавания образов, изучение реляционных баз данных и других. Отсюда происходит огромное количество подходов и алгоритмов, задействованных в разных существующих пакетах Data Mining. Многие из этих систем включают в себя несколько методологий за раз. И всё же, обычно, в каждом пакете есть какой-то главный компонент, на котором основана ее функциональность. Далее приведена классификация упомянутых главных моментов. Классам, которые удалось выделить, дается краткое описание, а так же приводятся примеры.

1.2.1 Статистические пакеты.

Прикладное программное обеспечение для статистики почти во всех современных версиях включает в себя не только привычные методы статистики, но и инструменты поиска закономерностей. Однако в них большее внимание уделяется именно классическим подходам — корреляционному, регрессионному, факторному анализу и другим

Недостатком систем данного класса считается повышенный входной порог подготовки пользователя. Стоит так же отметить, "тяжеловесность" мощных современных программ статистических исследований, что сильно мешает массовому применению в сферах финансов и бизнеса, особенно малого. Так же эти системы очень часто весьма дороги, их стоимость варьируется от $1000 до $15000.

К еще более серьезному и главному недостатку статистических пакетов, ограничивающему их применение в поиске знаний можно отнести тот факт, что большая часть методов, входящих в состав пакетов используют парадигму статистики, в которой сделана ставка на усредненные значения выборки. А подобные характеристики, как упоминалось ранее, при работе с реальными сложными физическими феноменами часто оказываются фиктивными величинами.

В качестве ярких примеров самых мощных и часто применяемых статистических пакетов стоит привести SAS (компания SAS Institute), SPSS (SPSS), STATGRAPICS (Manugistics), STATISTICA, STADIA и другие.

1.2.2 Нейронные сети.

Громоздкий класс систем, построение которых подобно построению нервной ткани из нейронов. В одной из особенно распространенных архитектур, многослойном перцептроне с обратным распределением ошибок, воссоздается взаимодействие нейронов в составе иерархической сети, где весь нейрон больше высокого яруса объединен своими входами с выходами нейронов нижележащего слоя. На нейроны самого нижнего слоя подаются значения входных параметров, на основе которых надобно принимать какие-то решения, предсказывать становление обстановки и так далее. Эти значения рассматриваются как сигналы, передающиеся в дальнейший слой, ослабляясь или усиливаясь в зависимости от числовых значений (весов), приписываемых межнейронным связям. В итоге на выходе нейрона самого верхнего слоя вырабатывается некоторое значение, которое рассматривается как результат - реакция каждой сети на введенные значения входных параметров. Для того что бы сеть можно было использовать в будущем, ее нужно "натренировать" на полученных ранее данных, для которых понятны и значения входных параметров, и верные ответы на них. Тренировка состоит в подборе весов межнейронных связей, обеспечивающих наибольшую близость ответов сети к известным положительным ответам.

Основным недостатком нейросетевой парадигмы является необходимость иметь слишком огромный объем обучающей выборки. Другой значительный недочет заключается в том, что даже натренированная нейронная сеть представляет из себя черный ящик. Знания, зафиксированные как веса нескольких сотен межнейронных связей, идеально не поддаются анализу и интерпретации человеком (знаменитые попытки дать

интерпретацию структуре настроенной нейросети выглядят неубедительными - система “KINOsuite-PR”)

Примеры нейросетевых систем - BrainMaker (CSS), NeuroShell (Ward Systems Group), OWL (HyperLogic). Стоимость их достаточно значительна: $1500-8000.

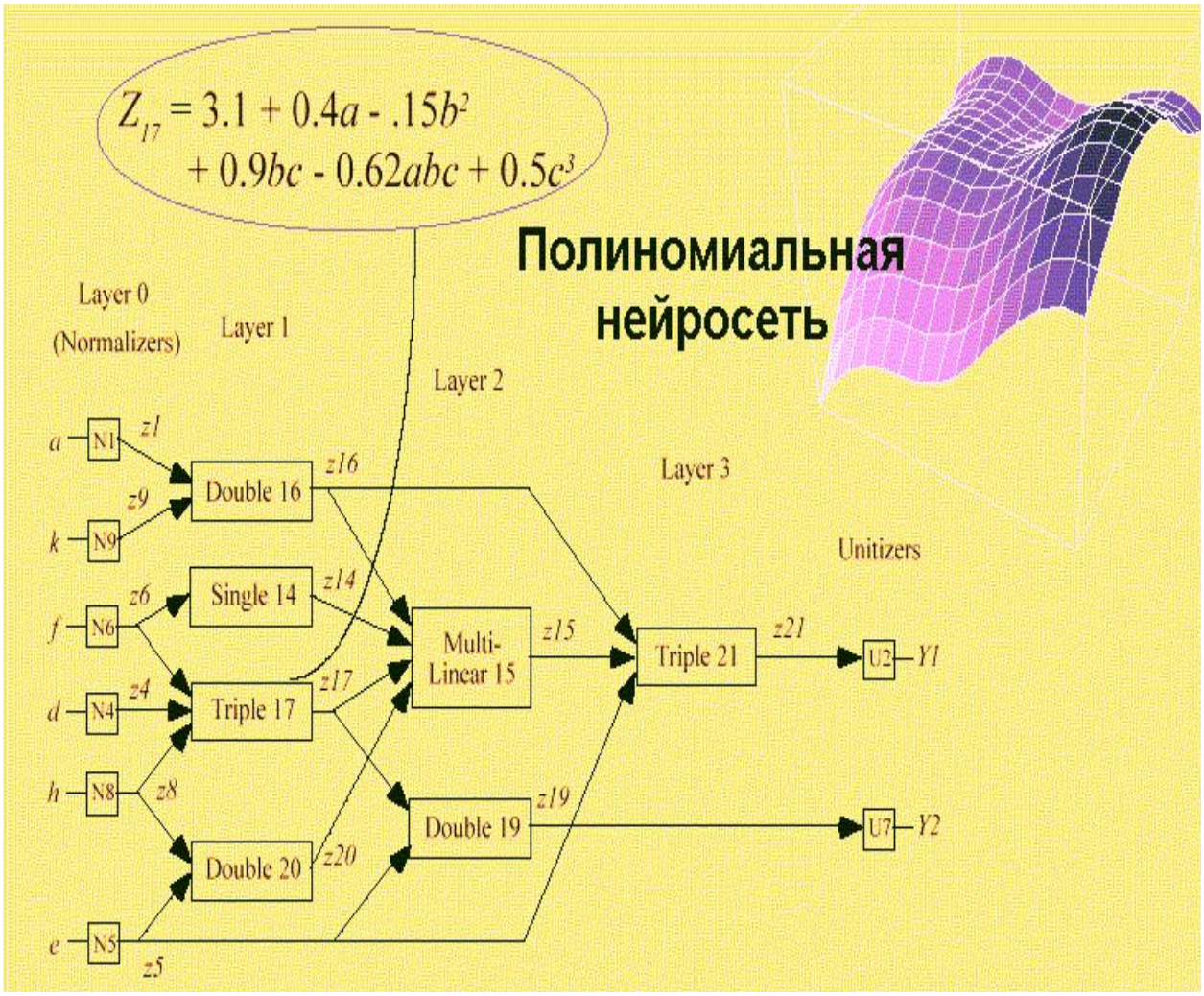


Рис 1.1. Полиномиальная нейросеть

1.2.3 Системы рассуждений на основе аналогичных случаев.

Идея систем case based reasoning - CBR – на первый взгляд весьма примитивна. Делая прогноз на грядущее либо выбирая лучшее положительное решение, эти системы подбирают в прошлом близкие аналоги

текущей обстановки и выбирают тот же результат, тот, что был для них верным. Следственно данный способ еще называют способом "ближайшего соседа" (nearest neighbour). В последнее время распространение получил также термин memory based reasoning, что акцентирует внимание на том, что решение принимается на основе информации, собранной в памяти.

Системы CBR показывают хорошие результаты в самых разных задачах. Главным их минусом считают то, что они в принципе не создают никаких моделей либо правил, обобщающих предшествующий опыт, - в выборе решения они базируются только на массиве доступных исторических данных, следственно не получится сказать, на основе каких определенно факторов CBR системы строят свои заключения.

Иной минус заключается в произволе, тот, что допускают системы CBR при выборе меры "близости". От этой меры самым решительным образом зависит объем множества прецедентов, которые надобно хранить в памяти для достижения удовлетворительной систематизации либо прогноза.

Примеры систем, использующих CBR, - KATE tools (Acknosoft, Франция), Pattern Recognition Workbench (Unica, США).

1.2.4 Деревья решений (decision trees).

Деревья решения наверное один из наиболее предпочтительных подходов к решению задач Data Mining. Они производят наглядную иерархическую структуру определяющих правил типа "ЕСЛИ... ТО..." (if-then), имеющую вид древа. Для определения, к какому классу отнести некоторый объект или ситуацию, нужно дать ответы на вопросы, находящиеся в узлах этой ветви, начиная с корня. Вопросы представлены в виде "значение параметра А больше х?". При положительном ответе, происходит переход к следующему узлу следующего уровня, если же

отрицательный — то к альтернативному узлу; дальше снова появляется вопрос, привязанный к соответствующему узлу.

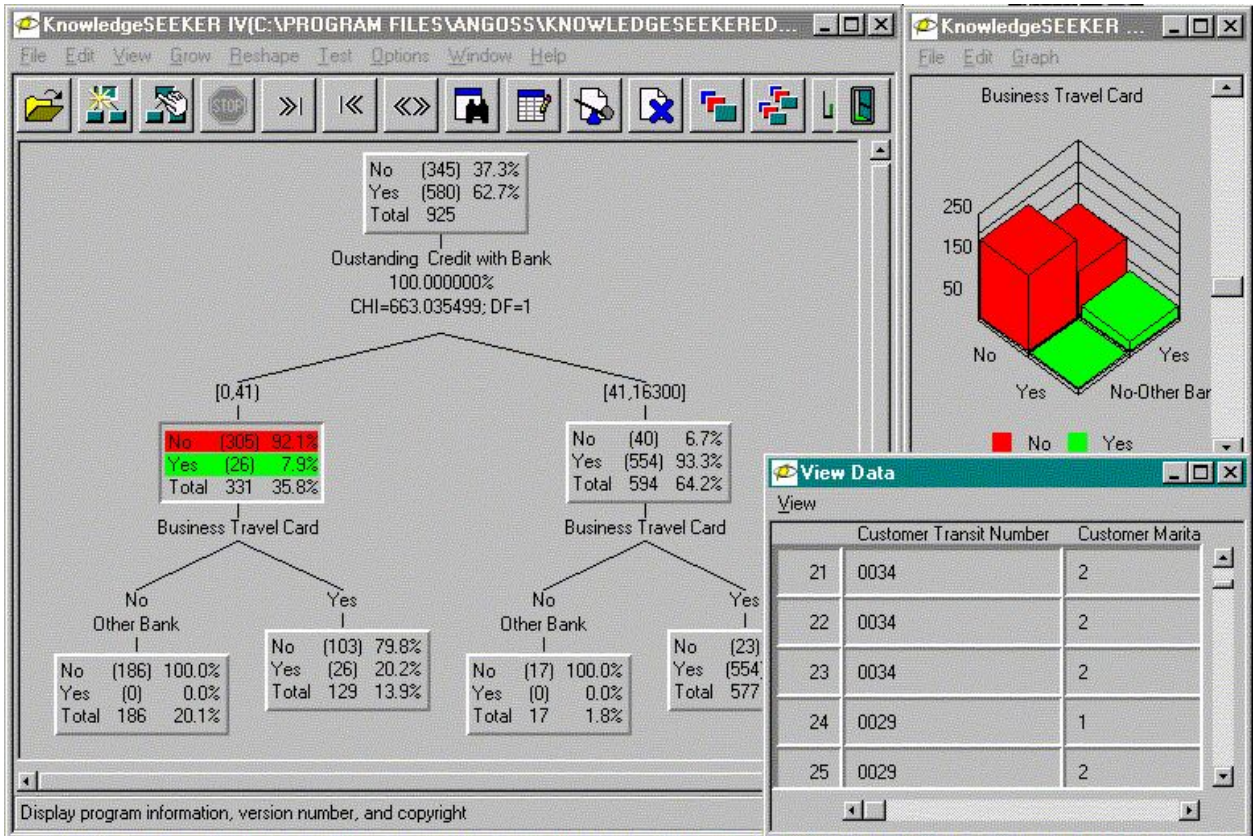


Рис 1.2. Система KnowledgeSeeker для обработки банковской информации

Популярность данного пакета связана с простотой, удобством, наглядностью и понятностью. Но, к сожалению, деревья решений не умеют находить “лучшие” (максимально полные и точные) закономерности в данных. Они отрабатывают наивный принцип последовательного изучения

параметров и приводят лишь фрагменты настоящих связей, создавая таким образом иллюзию логического вывода.

И всё же, чаще всего аналитические системы используют именно данный метод. Самыми известными являются See5/C5.0 (RuleQuest, Австралия), Clementine (Integral Solutions, Великобритания), SIPINA (University of Lyon, Франция), IDIS (Information Discovery, США), KnowledgeSeeker (ANGOSS, Канада). Стоимость этих систем варьируется от 1 до 10 тыс. долл.

1.2.5 Эволюционное программирование.

Проиллюстрируем нынешнее состояние данного подхода на примере системы PolyAnalyst - Российской разработке, имеющей сегодня всеобщее признание на рынке Data Mining. В данной системе догадки о виде зависимости целевой переменной от других переменных формулируются в виде программ на некотором внутреннем языке программирования. Процесс построения программ строится как эволюция в мире программ (этим подход немножко схож на генетические алгоритмы). Когда система находит программу, больше либо менее удовлетворительно выражающую желанную связанность, она начинает вносить в нее небольшие модификации и отбирает среди построенных дочерних программ те, которые повышают точность. Таким образом система "выращивает" несколько генетических линий программ, которые соперничают между собой в точности выражения желанной зависимости. Особый модуль системы PolyAnalyst переводит обнаруженные зависимости с внутреннего языка системы на внятный пользователю язык (математические формулы, таблицы и пр.).

Другое направление эволюционного программирования связано с поиском зависимости целевых переменных от остальных в форме функций

какого-то определенного вида. Скажем, в одном из особенно благополучных алгоритмов этого типа - методе группового учета доводов (МГУА) зависимость ищут в форме полиномов. В текущее время из продающихся в России систем МГУА реализован в системе NeuroShell компании Ward Systems Group. Стоимость систем до $ 5000.

1.2.6 Генетические алгоритмы.

Data Mining не основная область использования генетических алгорифмов. Их надобно рассматривать скорее как сильное средство решения многообразных комбинаторных задач и задач оптимизации. Тем не менее генетические алгорифмы вошли теперь в типовой инструментарий методов Data Mining, поэтому они и включены в данный обзор.

Первый шаг при построении генетических алгорифмов - это кодировка начальных логических обоснованностей в базе данных, которые именуют хромосомами, а каждый комплект таких обоснованностей называют популяцией хромосом. Дальше для реализации доктрины отбора вводится метод сравнения разных хромосом. Популяция обрабатывается с подмогой процедур репродукции, изменяемости (мутаций), генетической композиции. Эти процедуры имитируют биологические процессы. Особенно главные среди них: случайные мутации данных в индивидуальных хромосомах, переходы (кроссинговер) и рекомбинация генетического материала, содержащегося в индивидуальных родительских хромосомах (подобно гетеросексуальной репродукции), и миграции генов. В ходе работы процедур на всякой стадии эволюции получаются популяции со все больше совершенными индивидуумами.

Генетические алгоритмы комфортны тем, что их легко распараллеливать. Скажем, дозволено разбить поколение на несколько групп

и работать с всей из них самостоятельно, обмениваясь время от времени несколькими хромосомами. Существуют также и другие способы распараллеливания генетических алгоритмов.

Генетические алгоритмы имеют ряд недостатков. Критерий отбора хромосом и используемые процедуры являются эвристическими и далеко не гарантируют нахождения “лучшего” решения. Как и в реальной жизни, эволюцию может “заклинить” на какой-либо непроизводительной ветви. И, напротив, дозволено привести примеры, как два неперспективных родителя, которые будут исключены из эволюции генетическим алгоритмом, оказываются способными произвести высокоэффективного потомка. Это исключительно становится приметно при решении высокоразмерных задач с трудными внутренними связями.

Примером может служить система GeneHunter фирмы Ward Systems Group. Его стоимость - около $1000.

1.2.7 Алгоритмы ограниченного перебора.

Алгоритмы ограниченного перебора были предложены в середине 60-х годов М.М. Бонгардом для поиска логических обоснованностей в данных. С тех пор они продемонстрировали свою результативность при решении множества задач из самых разных областей.

Эти алгоритмы вычисляют частоты комбинаций примитивных логических событий в подгруппах данных. Примеры примитивных логических событий: X = a; X< a; X > a; a< X < b и др., где X - какой либо параметр, “a” и “b” - константы. Лимитацией служит длина комбинации примитивных логических событий (у М. Бонгарда она была равна 3). На основании обзора вычисленных частот делается завершение о полезности той

либо другой комбинации для установления ассоциации в данных, для систематизации, прогнозирования и пр.

Особенно ярким современным примером этого подхода является система WizWhy предприятия WizSoft.

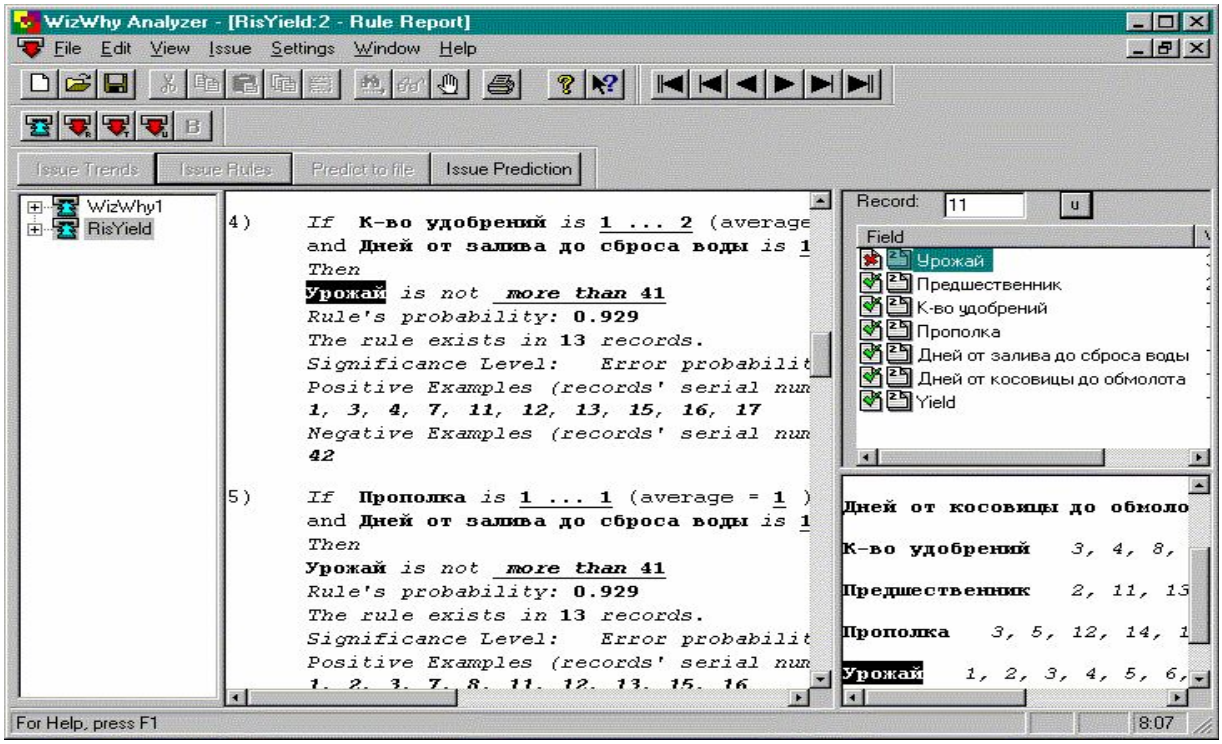


Рис 1.3. Система WizWhy нашла правила, выявляющие слабую урожайность неких сельских участков

Автор WizWhy заявляет, что его система обнаруживает ВСЕ логические if-then правила в данных. На самом деле это, финально, не так. Во-первых, максимальная длина комбинации в if-then правиле в системе WizWhy равна 6, и, во-вторых, с самого начала работы алгорифма производится эвристический поиск примитивных логических событий, на которых потом строится каждый последующий обзор. Осознав эти особенности WizWhy, несложно было предложить простейшую тестовую задачу, которую система не сумела вообще решить. Иной момент - система выдает решение за приемлемое время только для относительно маленький размерности данных.

Тем не менее, система WizWhy является на сегодняшний день одним из лидеров на рынке продуктов Data Mining. Это не лишено оснований. Система непрерывно показывает больше высокие показатели при решении утилитарных задач, чем все остальные алгоритмы. Стоимость системы около $ 4000, число продаж - 30000.

1.2.8 Системы для визуализации многомерных данных.

В той или другой мере средства для графического отображения данных поддерживаются всеми системами Data Mining. Совместно с тем, крайне значительную долю рынка занимают системы, специализирующиеся экстраординарно на этой функции. Примером тут может служить программаDataMiner 3D словацкой фирмы Dimension5 (5-е измерение).

В схожих системах главенствующее внимание сконцентрировано на дружелюбности пользовательского интерфейса, разрешающего ассоциировать с анализируемыми показателями разные параметры диаграммы рассеивания объектов (записей) базы данных. К таким параметрам относятся цвет, форма, ориентация касательно собственной оси, размеры и другие свойства графических элементов изображения. Помимо того, системы визуализации данных снабжены комфортными средствами для масштабирования и вращения изображений. Стоимость систем визуализации может доходить до нескольких сотен евро.

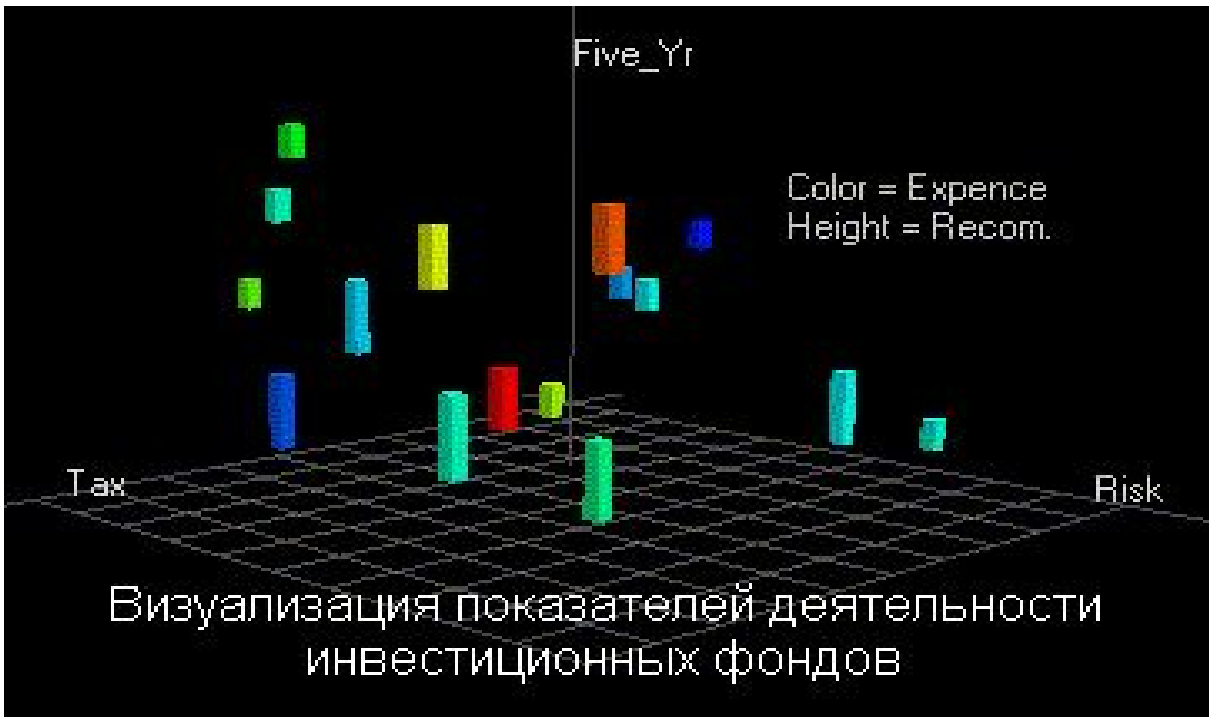


Рис 1.4. Визуализация данных системой DataMiner 3D

1.3 Недостатки существующих систем ИАД

Система обязана уметь решать несколько различных задач, от подбора и проверки правильности информации, вносимой в БД, до классического и/или интеллектуального исследования больших данных (степень влияния различных факторов на состояние банка, прогнозирование) и оптимизационного анализа. При этом от пользователя не должно требоваться специализированных знаний в области построения баз данных, факторного анализа или методов оптимизации.

На данный момент, имеющиеся на рынке средства ИАД крайне сложны и дороги, именно поэтому их сложно широко применять в качестве интегрированных в бизнес систем, направленных на конечного пользователя. В основу технологии ИАД заложен не один, а сразу несколько принципиально разных методов, причем применение некоторых из них проблематично без специальной подготовки. Выбор подхода часто требует постоянного привлечения специалистов.

2. Среды и средства, используемые в разработке.

2.1 Платформа .NET Framework

.NET Framework — это технология, которая поддерживает создание и выполнение нового поколения приложений и веб-служб XML. При разработке платформы .NET Framework учитывались следующие цели.

- Обеспечение согласованной объектно-ориентированной среды программирования для локального сохранения и выполнения объектного кода, для локального выполнения кода, распределенного в Интернете, либо для удаленного выполнения.
- Обеспечение среды выполнения кода, минимизирующей конфликты при развертывании программного обеспечения и управлении версиями.
- Обеспечение среды выполнения кода, гарантирующей безопасное выполнение кода, включая код, созданный неизвестным или не полностью доверенным сторонним изготовителем.
- Обеспечение среды выполнения кода, исключающей проблемы с производительностью сред выполнения сценариев или интерпретируемого кода.
- Обеспечение единых принципов работы разработчиков для разных типов приложений, таких как приложения Windows и веб-приложения.
- Разработка взаимодействия на основе промышленных стандартов, которое обеспечит интеграцию кода платформы .NET Framework с любым другим кодом.

Компания Microsoft выпустила .net framework в 2002 году. Основой платформы является общеязыковая среда исполнения Common Language Runtime (CLR), она подходит для разных языков программирования.

Программа для .NET Framework, написанная на любом поддерживаемом языке программирования, вначале переводится компилятором в цельный для .NET промежуточный байт-код Common Intermediate Language (CIL) (ранее именовался Microsoft Intermediate Language, MSIL). В терминах .NET получается сборка, англ. assembly. После этого код либо исполняется виртуальной машиной Common Language Runtime (CLR), либо транслируется утилитой NGen.exe в исполняемый код для определенного целевого процессора. Применение виртуальной машины предпочтительно, потому что освобождает разработчиков от необходимости заботиться об особенностях аппаратной части. В случае применения виртуальной машины CLR встроенный в неё JIT-компилятор «на лету» (just in time) преобразует промежуточный байт-код в машинные коды необходимого процессора. Современная спецтехнология динамической компиляции дозволяет добиться высокого яруса быстродействия. Виртуальная машина CLR также сама заботится о базовой безопасности, управлении памятью и системе исключений, освобождая разработчика от части работы.

2.2 Microsoft Visual Studio

Visual studio линейка продуктов компании Microsoft, включающих интегрированную среду разработки программного обеспечения и ряд других инструментальных средств. Данные продукты дозволяют разрабатывать как консольные приложения, так и приложения с графическим интерфейсом, в том числе с помощью спецтехнологии Windows Forms, а также веб-сайты, веб-приложения, веб-службы как в родном, так и в управляемом кодах для всех платформ, поддерживаемых Windows, Windows Mobile, Windows CE, .NET Framework, Xbox, Windows Phone .NET Compact Framework и Silverlight.

Visual Studio включает в себя редактор начального кода с помощью спецтехнологии IntelliSense и вероятностью простейшего рефакторинга кода. Встроенный отладчик может трудиться как отладчик яруса начального кода, так и отладчик машинного яруса. Остальные встраиваемые инструменты включают в себя редактор форм для облегчения создания графического интерфейса приложения, веб-редактор, дизайнер классов и дизайнер схемы базы данных. Visual Studio дозволяет создавать и подключать сторонние дополнения (плагины) для растяжения функциональности фактически на всяком ярусе, включая добавление поддержки систем контроля версий начального кода, добавление новых комплектов инструментов (скажем, для редактирования и визуального проектирования кода на предметно-ориентированных языках программирования) либо инструментов для прочих аспектов процесса разработки программного обеспечения (скажем, заказчик Team Explorer для работы с Team Foundation Server).

2.3 Язык программирования C#

Язык программирования C# (произносится си шарп) - объектно-ориентированный язык программирования. Разработан в 1998-2001 годах группой инженеров под начальством Андерса Хейлсберга в компании Microsoft как язык разработки приложений для платформы Microsoft .NET Framework и позднее был стандартизирован как ECMA-334 и ISO/IEC 23270.

C# относится к семье языков с C-сходственным синтаксисом, из них его синтаксис особенно близок к C++ и Java. Язык имеет статическую типизацию, поддерживает полиморфизм, перегрузку операторов (в том числе операторов очевидного и неявного приведения типа), делегаты, признаки, события, свойства, обобщённые типы и способы, итераторы, неизвестные

функции с помощью замыканий, LINQ, исключения, комментарии в формате XML.

Переняв многое от своих предшественников - языков C++, Pascal, Модула, Smalltalk и, в особенности, Java - C#, опираясь на практику их применения, исключает некоторые модели, зарекомендовавшие себя как проблематические при разработке программных систем, скажем, C# в различие от C++ не поддерживает множественное наследование классов (между тем допускается множественное наследование интерфейсов).

2.4 Технология ORM

Технология ORM либо Object-relational mapping (рус. объектно-реляционное отображение) - это спецтехнология программирования, которая разрешает преобразовывать несовместимые типы моделей в ООП, в частности, между хранилищем данных и объектами программирования. ORM применяется для облегчения процесса сохранения объектов в реляционную базу данных и их извлечения, при этом ORM сама заботится о реформировании данных между двумя несовместимыми состояниями. Множество ORM-инструментов в существенной мере полагаются на метаданные базы данных и объектов, так что объектам ничего не необходимо знать о структуре базы данных, а базе данных - ничего о том, как данные организованы в приложении. ORM обеспечивает полное распределение задач в отлично спроектированных приложениях, при котором и база данных, и приложение могут трудиться с данными всякий в своей начальной форме.

Ключевой спецификой ORM является отображение, которое применяется для привязки объекта к его данным в БД. ORM как бы создает «виртуальную» схему базы данных в памяти и разрешает манипулировать данными теснее на ярусе объектов. Отображение показывает как объект и его

свойства связанны с одной либо несколькими таблицами и их полями в базе данных. ORM использует информацию этого отображения для управления процессом реформирования данных между базой и формами объектов, а также для создания SQL-запросов для вставки, обновления и удаления данных в результат на метаморфозы, которые приложение вносит в эти объекты.

Применение ORM в плане освобождает разработчика от необходимости работы с SQL и написания большого числа кода, зачастую однообразного и подверженного ошибкам. Каждый генерируемый ORM код ориентировочно классно проверен и оптимизирован, следственно не необходимо в совокупности задумывается о его тестировании.

2.5 Entity Framework (EF)

EF - объектно-ориентированная спецтехнология доступа к данным, является object-relational mapping (ORM) решением для .NET Framework от Microsoft. Entity Framework представляет особую объектно-ориентированную спецтехнологию на базе фреймворка .NET для работы с данными. Если обычные средства ADO.NET разрешают создавать подключения, команды и прочие объекты для взаимодействия с базами данных, то Entity Framework представляет собой больше высокий ярус абстракции, тот, что разрешает отвлекаться от самой базы данных и работать с данными самостоятельно от типа хранилища. Если на физическом ярусе мы оперируем таблицами, индексами, первичными и внешними ключами, но на концептуальном ярусе, тот, что нам предлагает Entity Framework, мы теснее работает с объектами.

Центральной доктриной Entity Framework является представление сущности либо entity. Сущность представляет комплект данных, ассоциированных с определенным объектом. Следственно данная

спецтехнология полагает работу не с таблицами, а с объектами и их комплектами.

Любая сущность, как и всякий объект из реального мира, владеет рядом свойств. Скажем, если сущность описывает человека, то мы можем выделить такие свойства, как имя, фамилия, рост, возраст, вес. Свойства необязательно представляют примитивные данные типа int, но и могут представлять больше комплексные конструкции данных. И у всей сущности может быть одно либо несколько свойств, которые будут отличать эту сущность от других и будут уникально определять эту сущность. Сходственные свойства называют ключами. При этом сущности могут быть связаны ассоциативной связью один-ко-многим, один-ко-одному и многие-ко-многим, аналогично тому, как в реальной базе данных происходит связь через внешние ключи.

Отличительной чертой Entity Framework является применение запросов LINQ для выборки данных из БД. С подмогой LINQ мы можем не только извлекать определенные строки, хранящие объекты, из бд, но и получать объекты, связанные разными ассоциативными связями. Иным ключевым представлением является Entity Data Model. Эта модель сравнивает классы сущностей с реальными таблицами в БД.

Entity Data Model состоит из 3 ярусов: концептуального, ярус хранилища и ярус сравнения (маппинга). На концептуальном ярусе происходит определение классов сущностей, используемых в приложении.

Ярус хранилища определяет таблицы, столбцы, отношения между таблицами и типы данных, с которыми сопоставляется применяемая база данных. Ярус сравнения (маппинга) служит посредником между предыдущими двумя, определяя сравнение между свойствами класса сущности и столбцами таблиц.

Таким образом, мы можем через классы, определенные в приложении, взаимодействовать с таблицами из базы данных.

2.6 SQLite

Sqlite - суперкомпактная встраиваемая реляционная база данных. Начальный код библиотеки передан в социальное наследие. Слово «встраиваемый» (embedded) обозначает, что SQLite не использует парадигму заказчик-сервер, то есть движок SQLite не является отдельно работающим процессом, с которым взаимодействует программа, а предоставляет библиотеку, с которой программа компонуется, и движок становится комбинированный частью программы. Таким образом, в качестве протокола обмена применяются вызовы функций (API) библиотеки SQLite. Такой подход сокращает убыточные расходы, время отклика и упрощает программу. SQLite хранит всю базу данных (включая определения, таблицы, индексы и данные) в исключительном стандартном файле на том компьютере, на котором исполняется программа.

2.7. Технология WPF

Графическая среда WPF (Windows Presentation Foundation) является частью экосистемы платформы .NET и представляет собой подсистему для построения графических интерфейсов.

Если при создании традиционных приложений на основе WinForms за отрисовку элементов управления и графики отвечали такие части ОС Windows, как User32 и GDI+, то приложения WPF основаны на **DirectX**. В этом состоит ключевая особенность рендеринга графики в WPF: используя WPF, значительная часть работы по отрисовке графики, как простейших

кнопочек, так и сложных 3D-моделей, ложиться на графический процессор на видеокарте, что также позволяет воспользоваться аппаратным ускорением графики.

Одной из важных особенностей является использование языка декларативной разметки интерфейса XAML, основанного на XML: мы можем создавать насыщенный графический интерфейс, используя или декларативное объявление интерфейса, или код на управляемых языках C# и VB.NET, либо совмещать и то, и другое.

Преимущества WPF

- Использование традиционных языков .NET-платформы - C# и VB.NET для создания логики приложения
- Возможность декларативного определения графического интерфейса с помощью специального языка разметки XAML, основанном на xml и представляющем альтернативу программному созданию графики и элементов управления, а также возможность комбинировать XAML и C#/VB.NET
- Независимость от разрешения экрана: поскольку в WPF все элементы измеряются в независимых от устройства единицах, приложения на WPF легко масштабируются под разные экраны с разным разрешением.
- Новые возможности, которых сложно было достичь в WinForms, например, создание трехмерных моделей, привязка данных, использование таких элементов, как стили, шаблоны, темы и др.
- Хорошее взаимодействие с WinForms, благодаря чему, например, в приложениях WPF можно использовать традиционные элементы управления из WinForms.

- Богатые возможности по созданию различных приложений: это и мультимедиа, и двухмерная и трехмерная графика, и богатый набор встроенных элементов управления, а также возможность самим создавать новые элементы, создание анимаций, привязка данных, стили, шаблоны, темы и многое другое
- Аппаратное ускорение графики - вне зависимости от того, работаете ли вы с 2D или 3D, графикой или текстом, все компоненты приложения транслируются в объекты, понятные Direct3D, и затем визуализируются с помощью процессора на видеокарте, что повышает производительность, делает графику более плавной.

3. Разработка приложения реализующего поиск закономерностей в базах данных.

3.1 Описание исходных данных

В рамках проведения исследовательской работы предприятие общественного питания предоставило данные о клиентских заказах доставки еды на дом в неструктурированном виде, которые для удобства сразу можно разделить на две условные группы – сведения о заказе и сведения о клиенте.

Сведения о заказе включают в себя следующую информацию:

- Дата и время заказа;
- Наименования и стоимость блюд;
- Итоговая стоимость;
- Способ оплаты.

Сведения о клиенте:

- ФИО;
- Адрес проживания;
- Контактный номер телефона.

3.2 Создание базы данных

Так как данные предоставлены в текстовом формате, необходимо вручную создать базу данных и заполнить её. Для начала нарисуем ER (entity-relationship) модель, данные модели

стандартизированы и подойдут к любой СУБД которую мы выберем в дальнейшем.

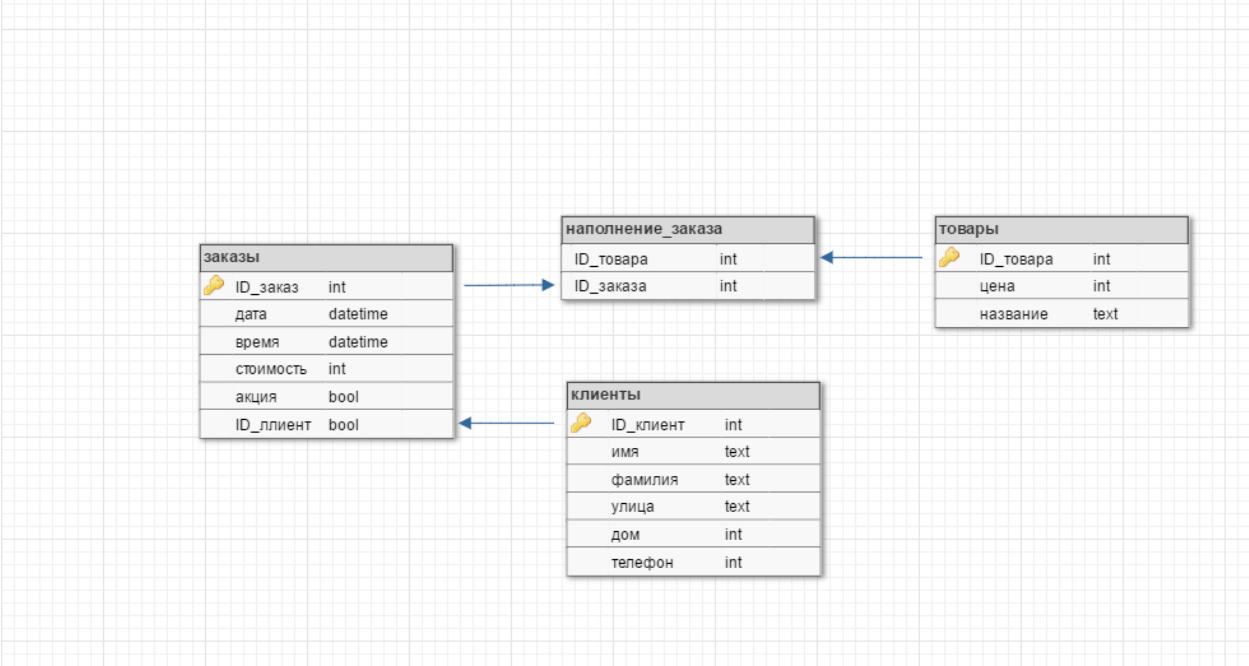


Рис 3.1. ER модель базы данных

Приложение задумано как десктопное, а значит достаточно локальной встраиваемой базы данных, для этих целей идеально подойдёт СУБД SQlite, она удовлетворяет нашим требованиям и распространяется бесплатно. Что бы приступить к созданию базы данных воспользуемся фирменным приложением DB Browser for Sqlite. Данное приложение позволит нам создать таблицы при помощи конструктора, а так же предоставит полный доступ к работе с ними на языке SQL.

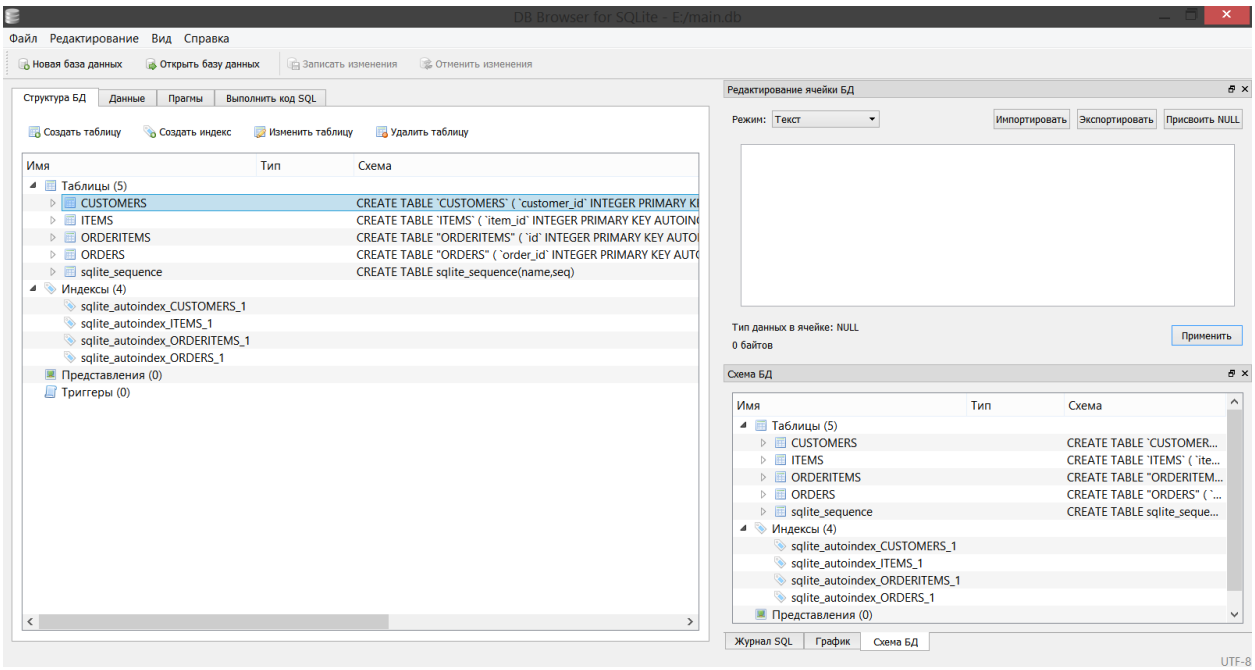


Рис 3.2. Обозреватель баз данных DB Browser for SQlite

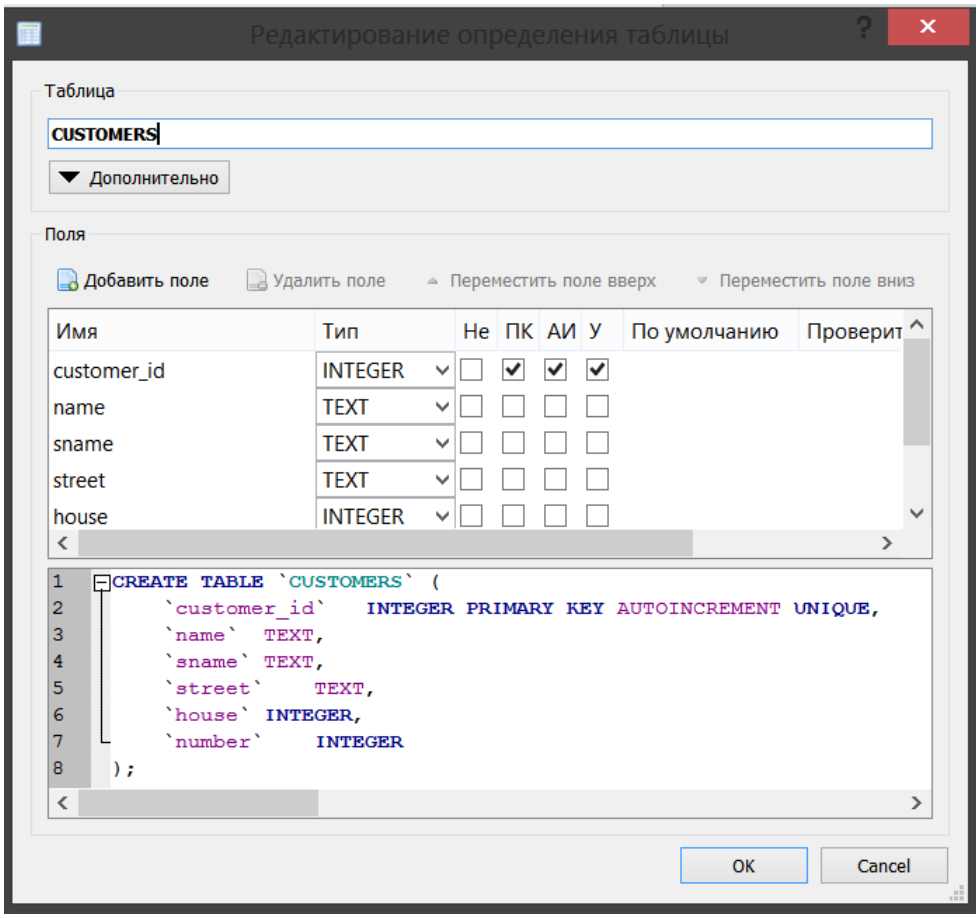


Рис 3.3. Окно редактора таблиц

Редактор транслирует все наши действия в SQL код:

```
CREATE TABLE `CUSTOMERS` (
	`customer_id`	INTEGER PRIMARY KEY AUTOINCREMENT UNIQUE,
	`name`	TEXT,
	`sname`	TEXT,
	`street`	TEXT,
	`house`	INTEGER,
	`number`	INTEGER
);
```

И в результате мы получаем готовую таблицу

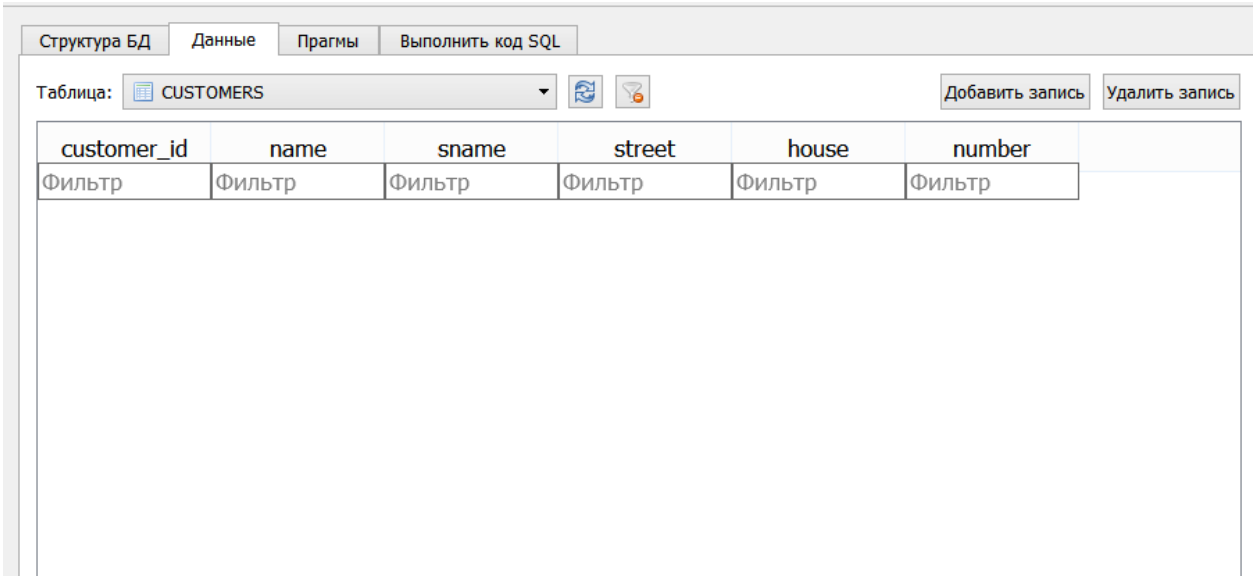


Рис 3.4. Готовая таблица

3.3 Создание проекта.

В качестве среды разработки была выбрана Microsoft Visual Studio, данная платформа предоставляет широкий диапазон возможностей, она поддерживает множество языков программирования, имеет огромное количество самых различных встраиваемых пакетов и может похвастаться хорошей поддержкой со стороны разработчиков. VS распространяется на платной основе, однако доступ к Professional версии можно получить по лицензии университета. Из предлагаемых языков программирования мы остановимся на C#, данный язык идеально подходит для прикладных задач, он не перегружен тяжеловесным синтаксисом и имеет много уровней абстракции, а так же предоставляет хороший функционал. Так как C# так же является представителем среды .Net Framework встроенных библиотек будет более чем достаточно.

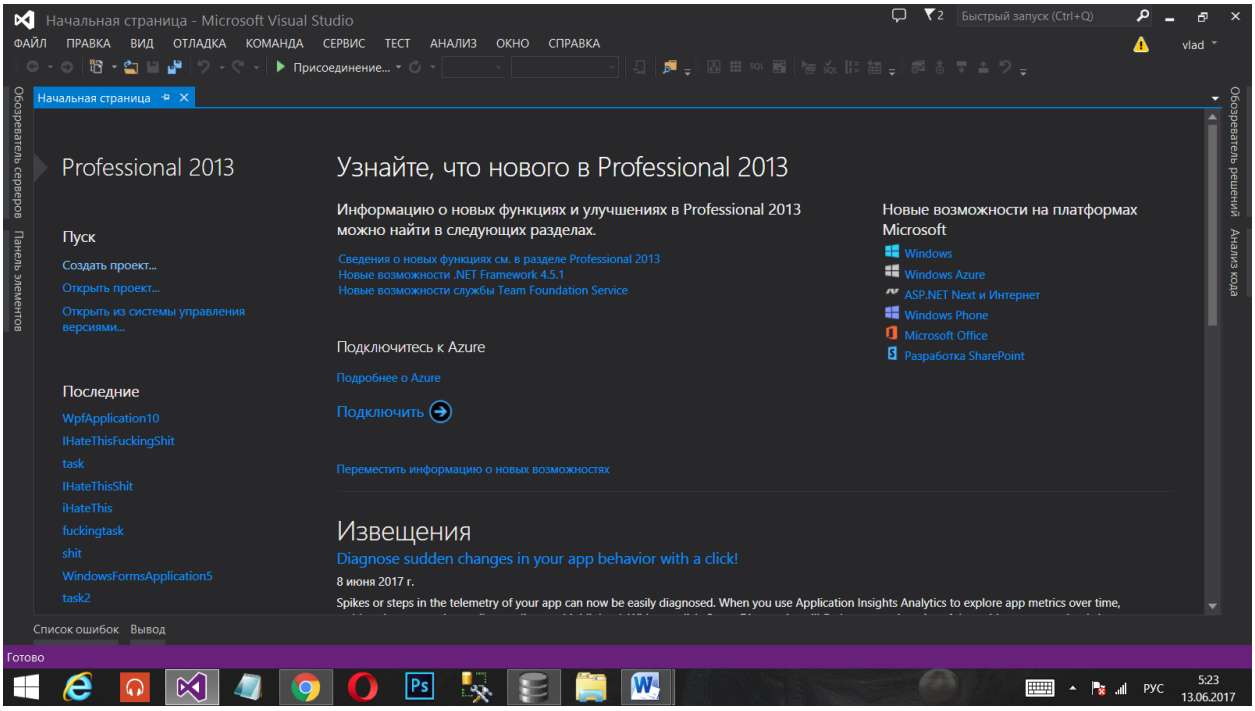


Рис 3.5. Стартовая страница Visual Studio

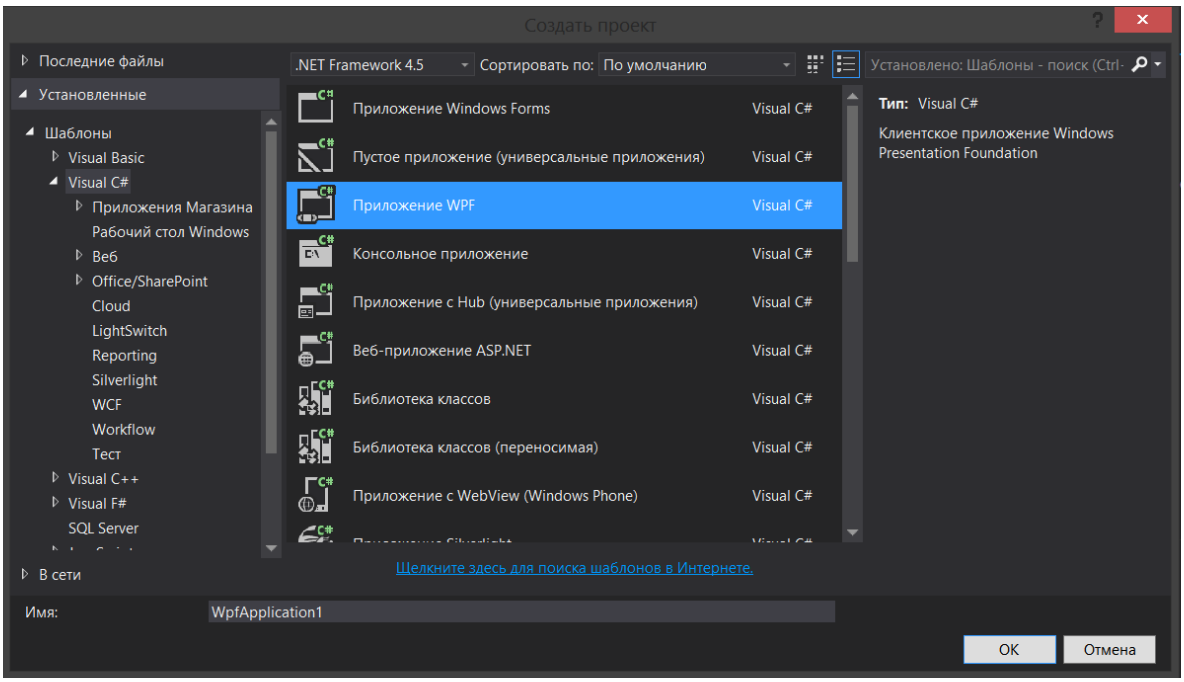


Рис 3.6. Создание нового проекта

Так как мы создаем оконное приложение с графическим интерфейсом, рабочее пространство представлено в виде двух разделов: проектирование окна и введение кода.

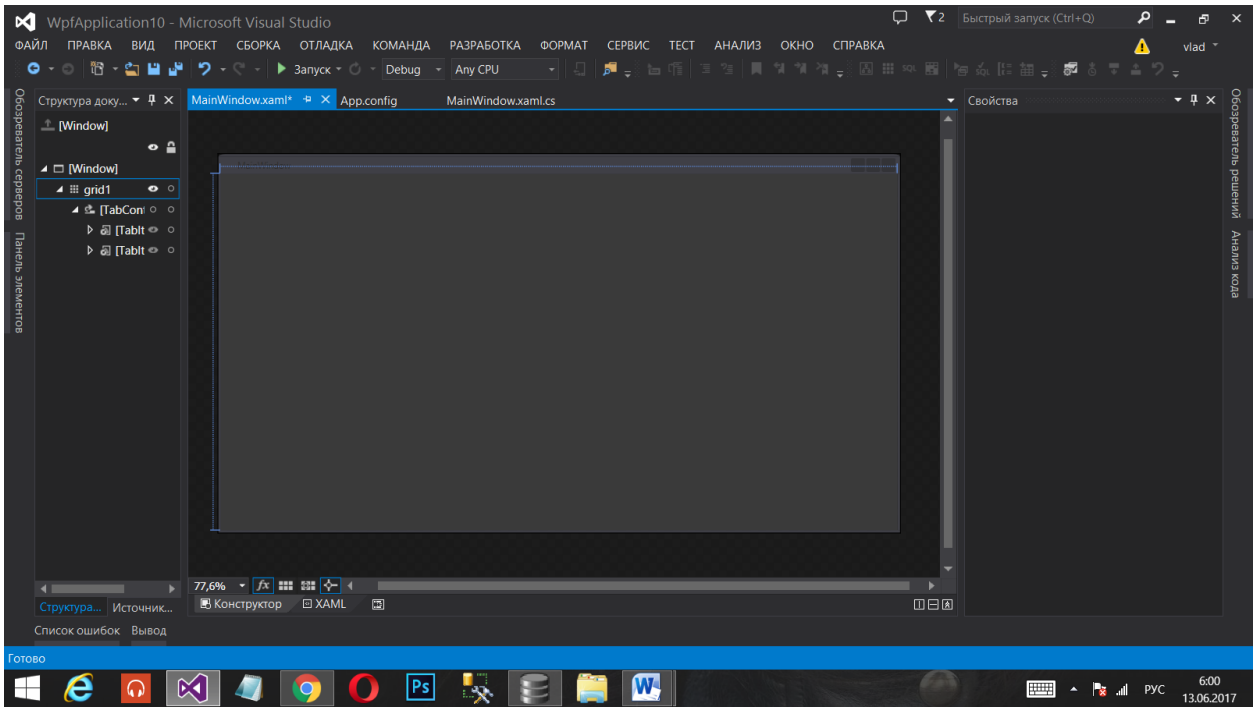


Рис 3.7. Разработка окна

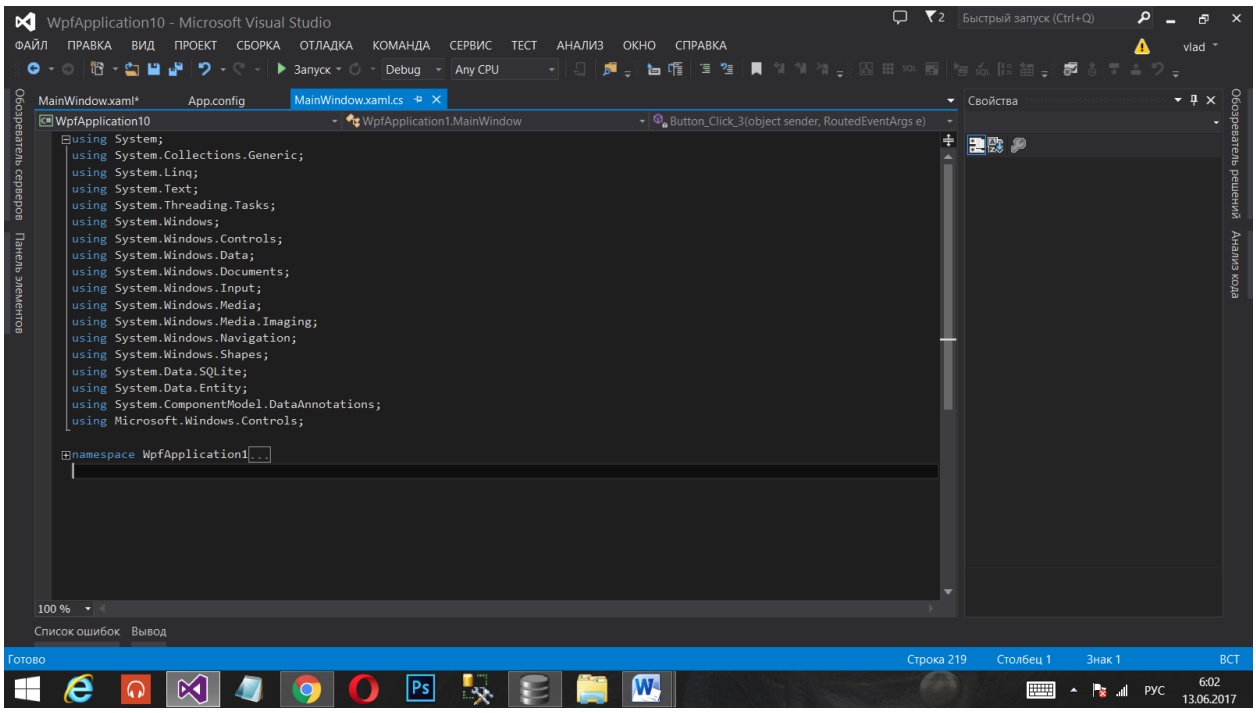


Рис 3.8. Введение кода

3.4 Включение базы данных в проект.

После того как проект и база данных созданы, необходимо настроить между ним подключение, Visual Studio для этого предлагает использовать встроенные инструменты, но к сожалению, они работают только с MS SQL SERVER, и не поддерживают SQlite, а значит подключаться к БД придется вручную. Для начала добавим в проект файл базы данных как и любой другой файл через команду Проект->Добавить существующий элемент.

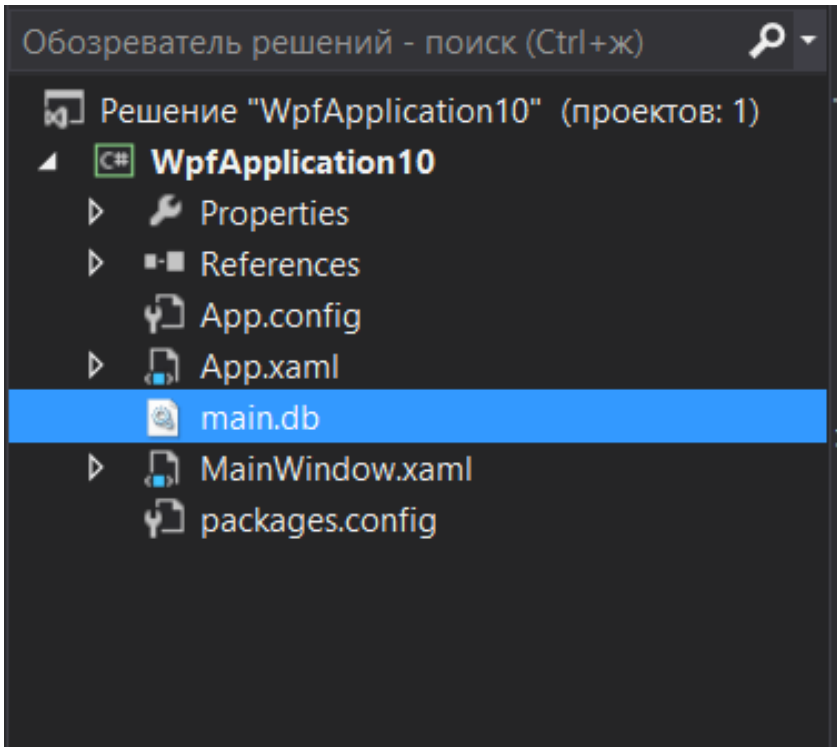


Рис 3.9. Добавление файла базы данных

Теперь ранее созданный файл main.db отображается в нашем решении, но интерфейс и подключение все еще не определены. Чтобы программа поняла, что она взаимодействует с базой данных нужно включить в проект библиотеки SQlite. Для этого вызовем управление встраиваемыми пакетами, найдём и выберем System.Data.SQlite.

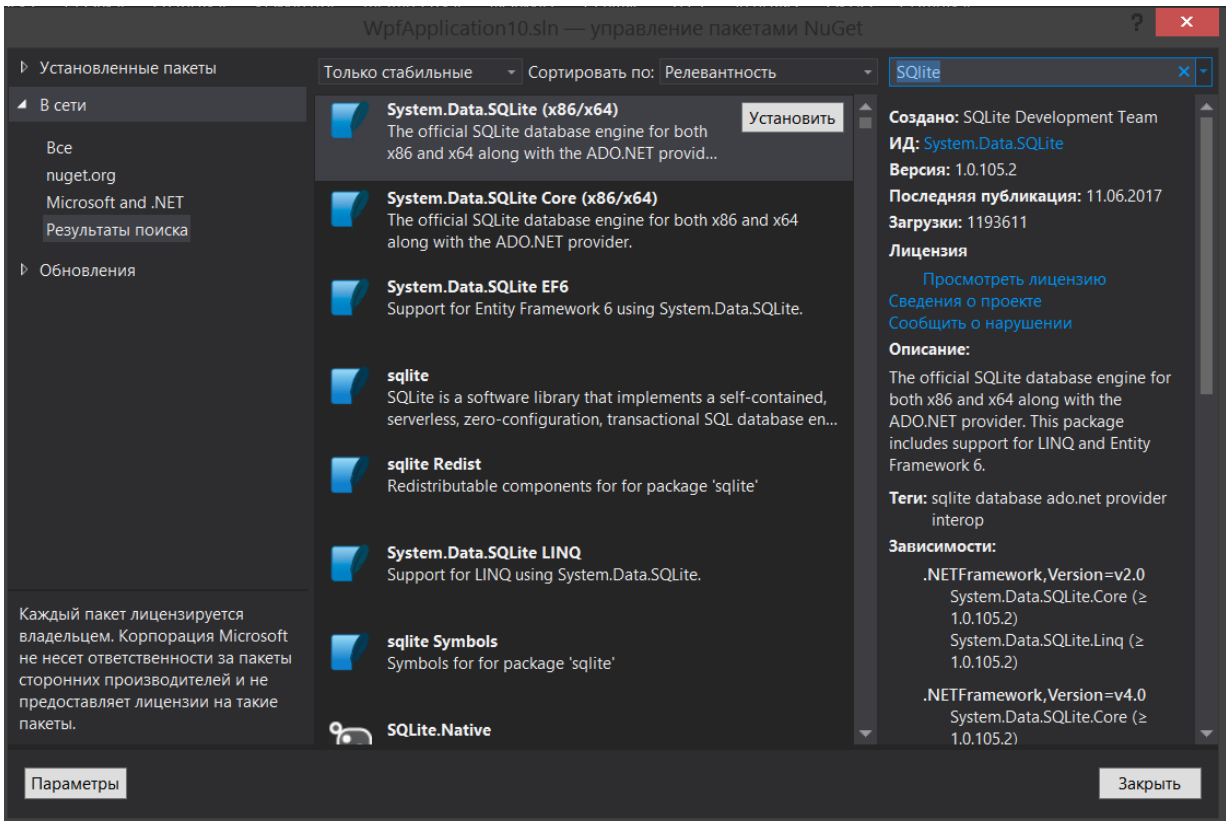


Рис 3.10. Управление подключаемыми пакетами

Так же требуется обозначить использование данной библиотеки в коде программы командой:

```
using System.Data.SQLite;
```

Для упрощения дальнейшей работы с базой данных и запросами к ней добавим слой абстракции в виде пакета Entity Framework, который является представителем технологии ORM и поставляется Microsoft. Для этого укажем в коде еще одну ссылку:

```
using System.Data.Entity;
```

Теперь когда интерфейс определён осталось задать строку подключения, это можно сделать в окне дополнительной конфигурации.

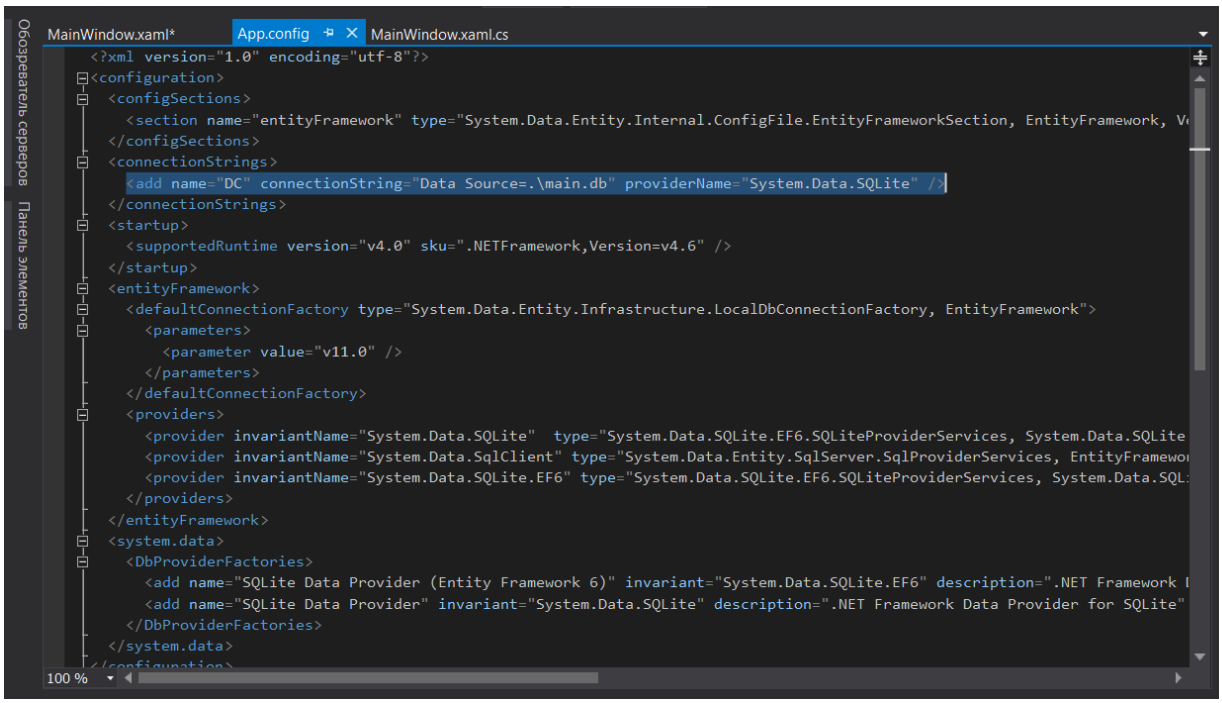


Рис 3.11. Окно дополнительной конфигурации

Строка подключение указывает путь к файлу базы данных, а так же определяет провайдера, которым для нас является библиотека SQlite.

<add name="DC" connectionString="Data Source=.\main.db" providerName="System.Data.SQLite" />

3.5 Заполнение базы данных.

Так как в проекте задействована технология ORM, первым делом стоит определить логическую модель базы данных внутри программы, для этого достаточно создать иерархию классов описывающих описывающую сущности и отношения между ними. Пример такого класса:

```
public class Customer
    {
        [Key] public int customer_id { get; set; }
        public string name { get; set; }
        public string sname { get; set; }
        public string street { get; set; }
        public int house { get; set; }
        public int number { get; set; }
        // Ссылка на заказы
        public virtual List<Order> Orders { get; set; }
    }
```

Важно соблюдать правила работы с Entity Framework, по традиции в реляционных базах данных сущности называют во множественном числе, название класса должно дублировать название сущности но в единственном числе, стоит так же указать первичный ключ с помощью ключевого слова [Key]. Далее создадим класс контекста который свяжет базу данных с её представлением.

```
public class maincontext : DbContext
```

```
{
        public maincontext()
       : base("DC") { }
    public DbSet<Customer> CUSTOMERS { get; set; }
    public DbSet<Order> ORDERS { get; set; }
    public DbSet<Item> ITEMS { get; set; }
    public DbSet<Orderitem> ORDERITEMS { get; set; }
}
```

Данный класс наследуется от системного DbContext, что позволяет вызывать конструктор базового класса, которому в качестве аргумента передается строка подключения.

Далее стоит позаботиться о графическом интерфейсе для занесения данных, учитывая множественность элементов которые предстоит отобразить в окне приложения, хорошим решением станет реализовать систему вкладок, для этого добавим на форму элемент управления TabControl и создадим первую вкладку. На самой вкладке разместим требуемые элементы кнопки поля ввода и список блюд.

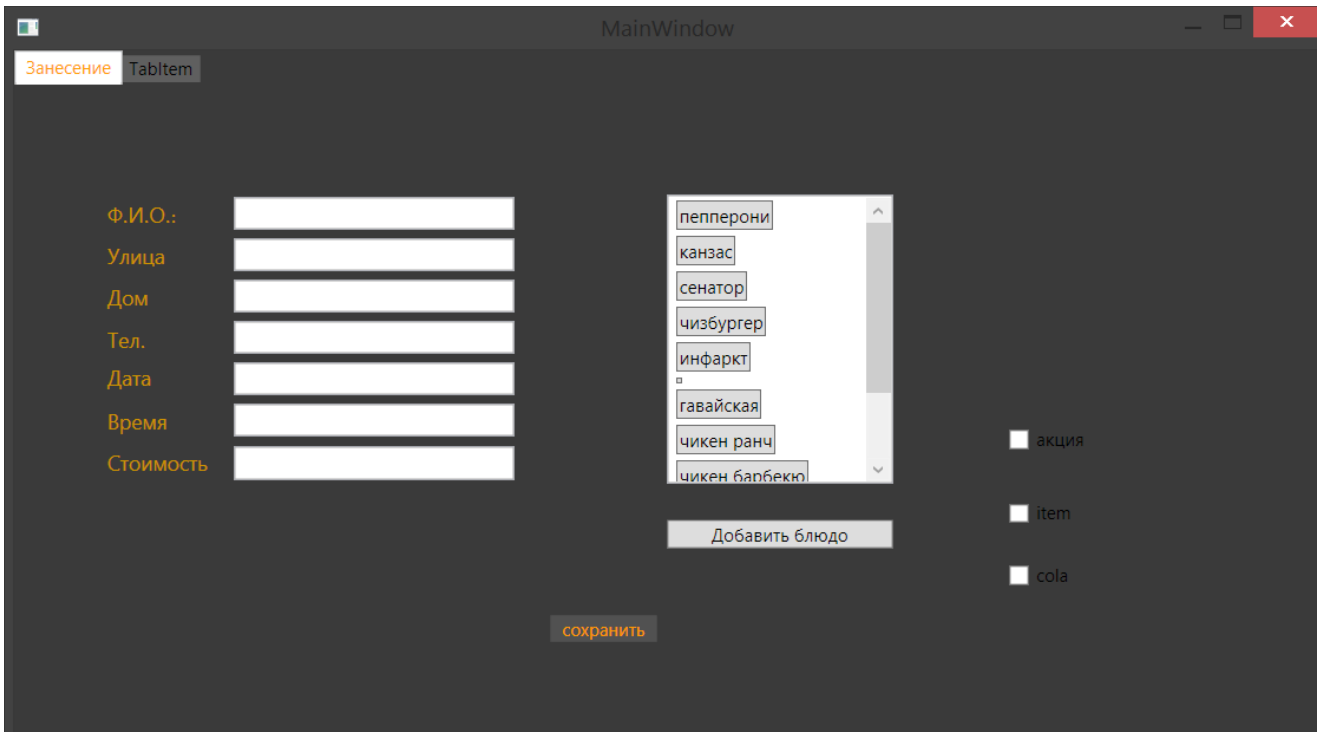


Рис 3.12. Вкладка внесения информации о заказе.

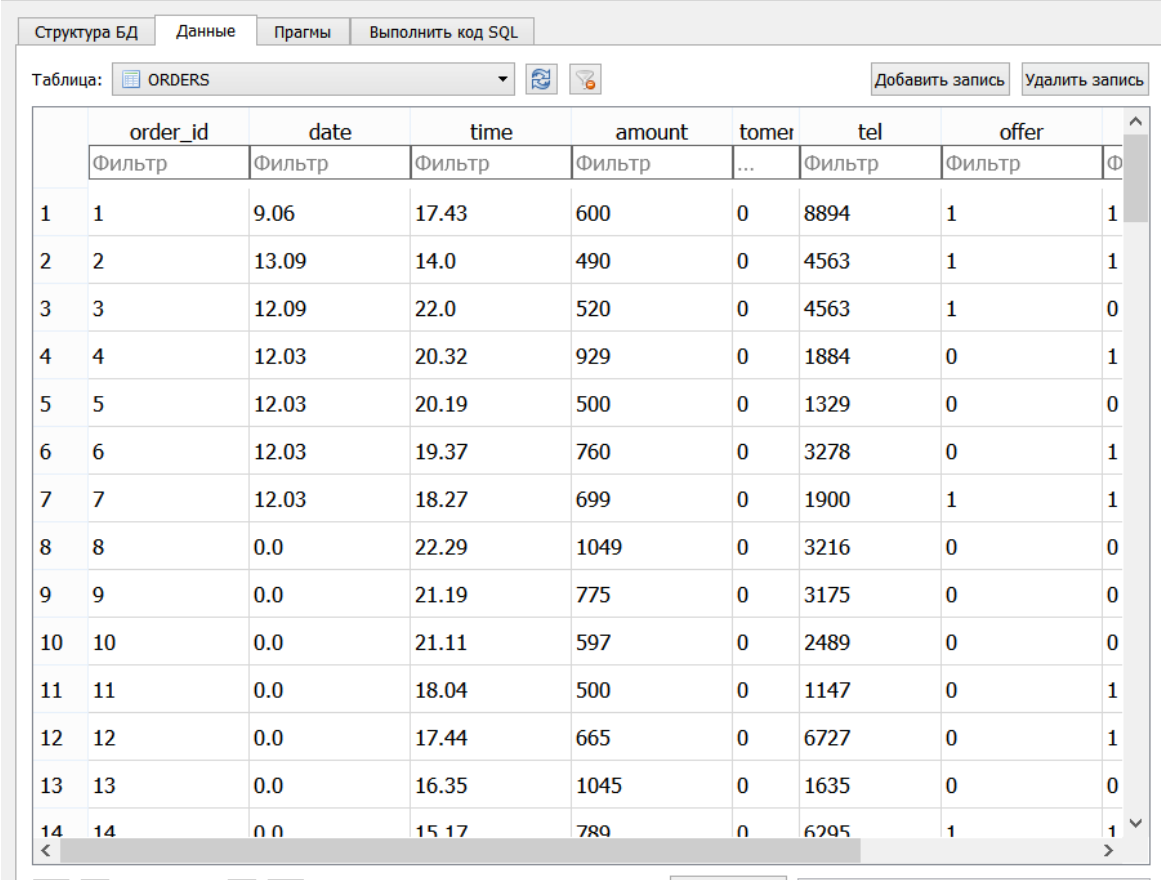
Современная графическая среда WPF позволяет создавать качественный, простой и приятный глазу интерфейс, а так как для работы он задействует Direct X, более сложные элементы на форме работают такие как вкладки, работают достаточно быстро.

Следующим шагом будет написание кода, который реализует внесение вводимых данных в форму, в базу. Как и говорилось ранее нам совсем не обязательно прибегать к запросам SQL, ведь у нас есть слой абстракции, нам достаточно вносить данные в поля экземпляров классов, и передавать экземпляры контексту, который за нас транслирует операции в SQL код. Пример подобного внесения:

```
// новый контекст
maincontext db = new maincontext();
// экземляр класса заказ
Order o = new Order();
```

```
            o.amount = Convert.ToInt32(t7.Text);
            o.date = Convert.ToDouble(t5.Text);
            o.time = Convert.ToDouble(t6.Text);
if (check.IsChecked == true) o.offer = 1;
            check.IsChecked = false;
db.ORDERS.Add(o);
            db.SaveChanges();
```

Откроем DB Browser for SQlite и проверим как отобразились данные



	order_id	date	time	amount	tomer	tel	offer	
	Фильтр	Фильтр	Фильтр	Фильтр	...	Фильтр	Фильтр	Ф
1	1	9.06	17.43	600	0	8894	1	1
2	2	13.09	14.0	490	0	4563	1	1
3	3	12.09	22.0	520	0	4563	1	0
4	4	12.03	20.32	929	0	1884	0	1
5	5	12.03	20.19	500	0	1329	0	0
6	6	12.03	19.37	760	0	3278	0	1
7	7	12.03	18.27	699	0	1900	1	1
8	8	0.0	22.29	1049	0	3216	0	0
9	9	0.0	21.19	775	0	3175	0	0
10	10	0.0	21.11	597	0	2489	0	0
11	11	0.0	18.04	500	0	1147	0	1
12	12	0.0	17.44	665	0	6727	0	1
13	13	0.0	16.35	1045	0	1635	0	0
14	14	0.0	15.17	789	0	6295	1	1

Рис 3.13. Заполненная таблица.

3.6 Работа с данными

Теперь, когда наши данные занесены в реляционную базу данных, можно приступать к непосредственной работе с ними. Как говорилось ранее, мы исследуем данные о клиентских заказах доставки еды на дом и ищем в них закономерности. Начнём с поиска зависимостей между количеством заказов и временем, а так же между временем и суммой заказа.

Первое о чем стоит подумать – удобное представление данных. Создадим подходящий интерфейс, который обеспечит нам наглядное отображение. Графическая среда WPF обладает широким набором инструментов для реализации задач такого рода.

С помощью ранее добавленного элемента управления TabControl создадим новую вкладку под названием «отображение 1», на ней разместим графическое поле Canvas. Использование Canvas является не лучшим подходом в создании графических интерфейсов так как, данный элемент не поддерживает масштабирование, но тем не менее он идеально подходит для рисования графиков. На canvas нанесём координатную сетку, сделать это можно напрямую через язык XAML, на котором описывается все содержимое окна, или же через исполняемый код, мы задействуем оба варианта.

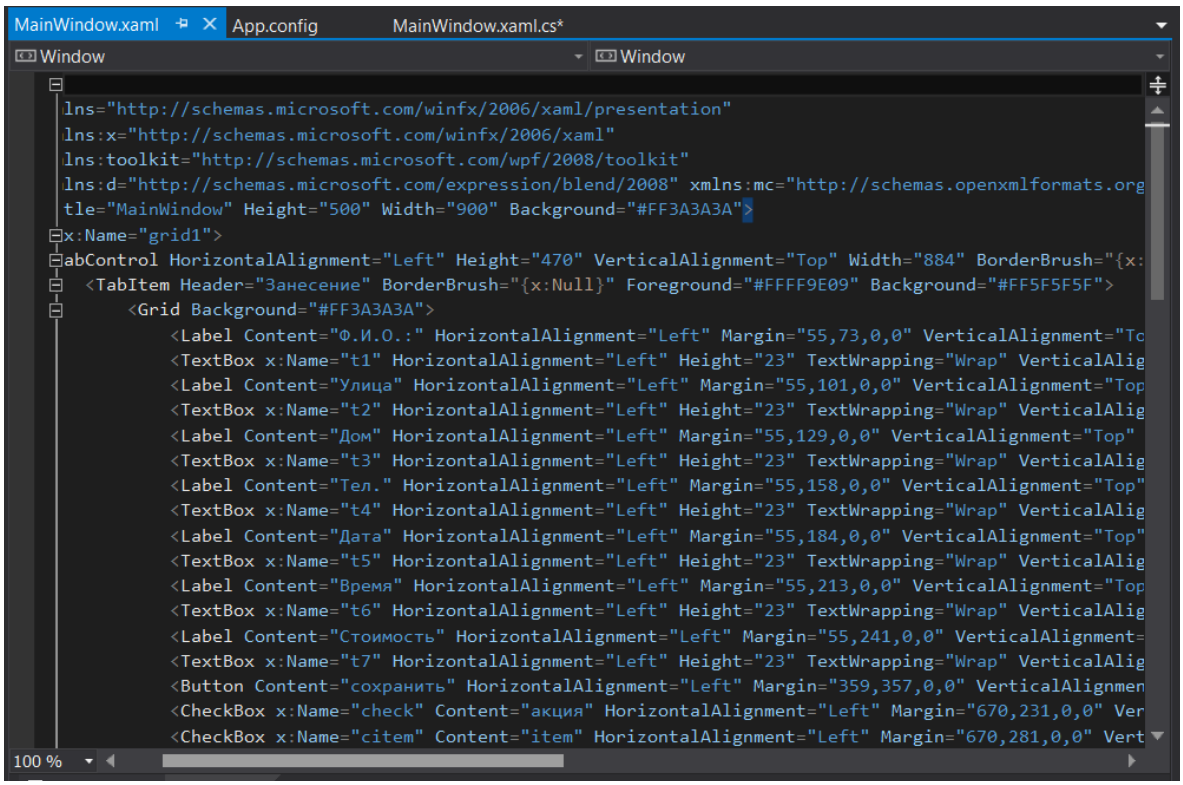


Рис 3.14. Редактор XAML кода.

```
<Grid>
 <Line X1="1" Y1 = "400" X2 = "1" Y2 = "1"Stroke="DarkOrange"/>
 <Line X1="1" Y1="400" X2="880" Y2="400" Stroke="DarkOrange"/>
 </Grid>
```

Так две строки описывают добавление двух линий: ось x и ось y, далее добавим деления, так как деления – это тоже линии и добавить их нужно достаточно много удобнее будет это сделать из исполняемого кода, для этого напишем следующий цикл:

```
for (byte i = 1; i < 88; i++)
        { Line a = new Line();
          a.X1 = i * 10;
```

```
    a.X2 = i * 10;
    a.Y1 = 395;
    a.Y2 = 400;
    a.Stroke = Brushes.DarkOrange;
    cnv.Children.Add(a) }
for (byte i = 0; i < 14; i++)
{ Line a = new Line();
    a.X1 = 1;
    a.X2 = 5;
    a.Y1 = i * 28.5;
    a.Y2 = i * 28.5;
    a.Stroke = Brushes.DarkOrange;
    cnv.Children.Add(a);    }
```

И наконец, добавим на форму несколько текстовых меток с помощью которых укажем обозначения. Результат проделанной работы на рисунке ниже.

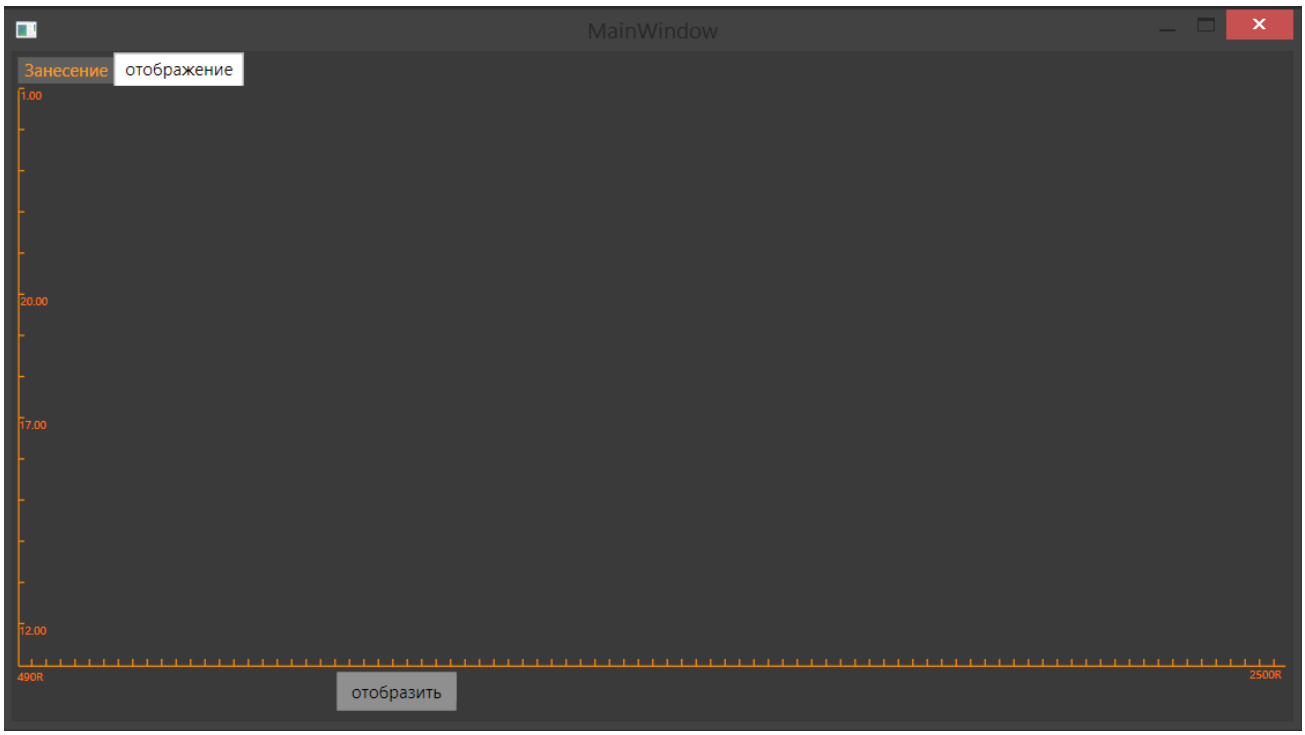


Рис 3.15. Вкладка «отображение».

По оси Y отложено время с 12.00 до 1.00 так как это часы приема заказов, по оси X отложены значения, которые принимает параметр – стоимость заказа, значения меняются от 490(минимальная сумма заказа) до 1450(обнаруженный максимум). Теперь отобразим заказы в виде точек. Как и прежде, Entity Framework избавляет нас от громоздких SQL запросов, вместо этого в обработчике события «нажатие кнопки» мы пишем LINQ запрос к экземпляру контекста:

```
maincontext db = new maincontext();
        var orders = from o in db.ORDERS
                     select o;
        foreach (Order o in orders)
        {
           int s = o.amount;
```

```
            double t = o.time;
            cnv.Children.Add(CreateEllipse(s, t));
        }
```

Возвращаемые запросом переменные хранятся в коллекции объектов, поэтому запускаем цикл, и в каждой итерации передаем значения времени и суммы заказа в функцию CreateEllipse, описанную ранее и отвечающую за нанесение точек на Canvas. Так же в функцию CreateEllipse добавим код отвечающий за создание тултипов – всплывающих подсказок, на которых будут отображаться точные значения точек по наведению курсора.

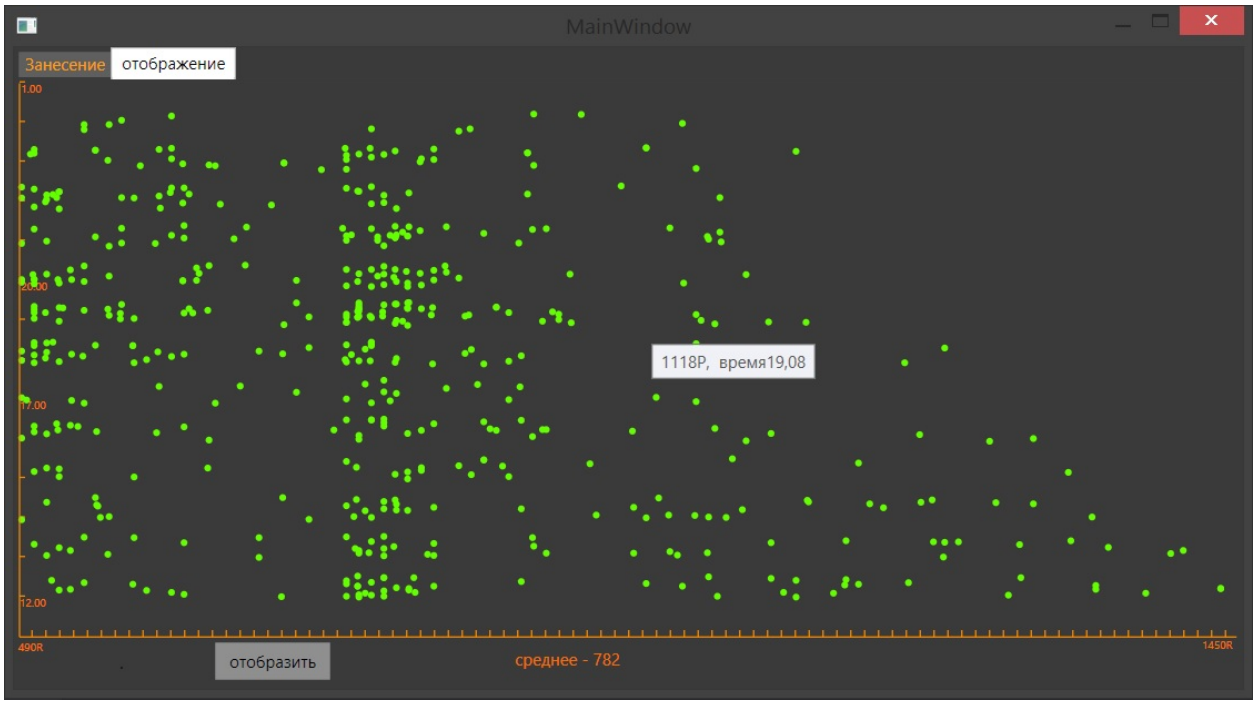


Рис 3.16. Отображение.

На графике отчетливо видна область аномальных значений: стоимость заказов достигает своих максимальных показателей в промежутке между 12.00 и 16.00, что, скорее всего, является зависимостью. Так же на графике можно заметить интервал с наибольшей частотой значений 750 – 800.

Теперь детальнее рассмотрим точки наивысших значений. В информации о заказе указано, воспользовался клиент акцией «1+1+1= 4» или нет, проверим, как данный параметр отразился на сумме заказов, для этого создадим новый запрос к базе данных, который вернет нам все заказы, выполненные по акции.

```
var orders = from o in db.ORDERS

where o.offer == 1

select o;
```

На графике закрасим точки отражающие данные заказы другим цветом.

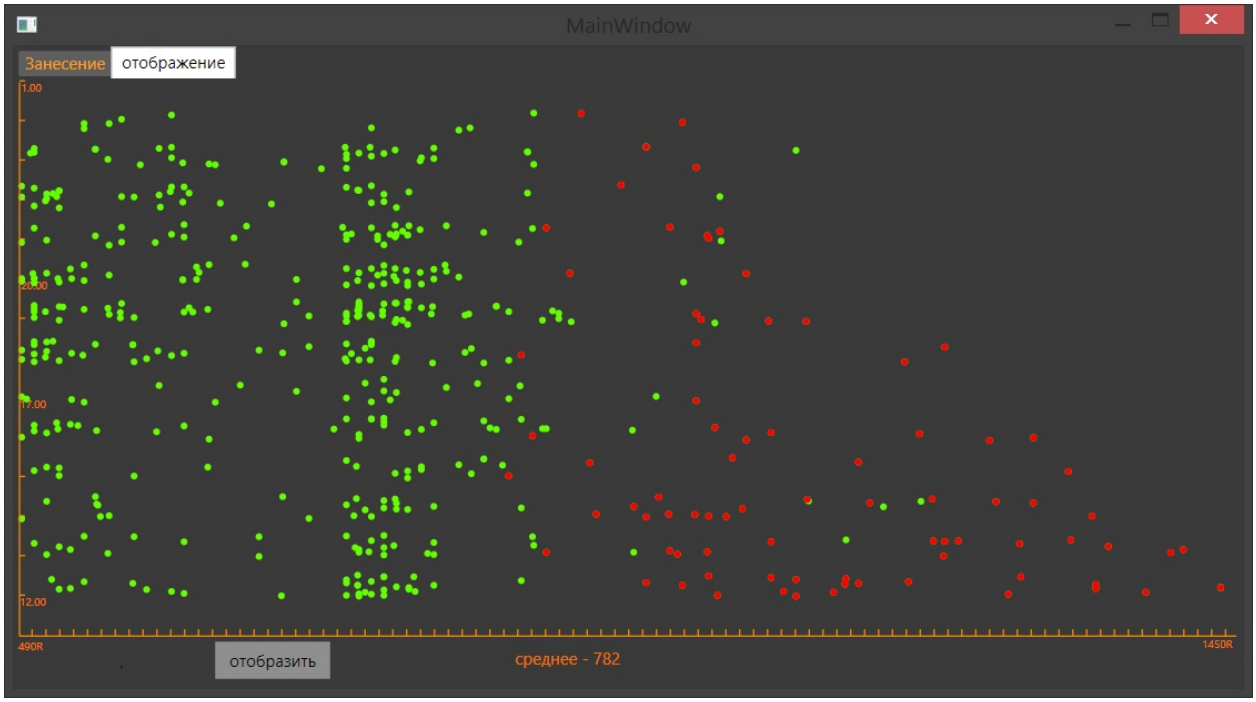


Рис 3.17. Выделение заказов сделанных по акции.

Изучим частоту параметра «акция» для заказов на сумму свыше 1000, для этого создадим 2 новых запроса к базе данных, а результат выведем на экран при помощи диалогового окна.

Рис 3.18.Вывод диалогового окна с результатами.

Среди всех заказов стоимостью свыше 1000 рублей по акции было сделано 70, без неё было сделано 11, что в процентном соотношении равно 86% и 14%, такие цифры демонстрируют нам явную закономерность, что позволяет говорить о чрезвычайной эффективности действующей акции, «1+1+1 = 4».

Для лица осуществляющего анализ продаж, подобное отображение будет крайне полезным, так как на его основе можно выстраивать более эффективную стратегию.

ЗАКЛЮЧЕНИЕ

В ходе выполнения выпускной квалификационной работы решена прикладная задача реализации программного обеспечения поиска скрытых закономерностей в реляционной базе данных.

Выводы и результаты работы сводятся к следующему.

Собраны и систематизированы данные о существующих системах интеллектуального анализа данных.

Разработано приложение осуществляющее поиск скрытых закономерностей в реляционной базе данных.

Приложение применено и протестировано.

Использование разработанного программного обеспечения позволит:

- проводить анализ продаж;
- выявлять закономерности между атрибутами заказов;
- выявлять закономерности в заказах клиента;

За счет низкой стоимости и удобства применения подобная программа способна окупить себя в течение года ее функционирования и обеспечить дальнейший рост.

На основании вышесказанного можно сделать вывод о том, что разработка данного программного обеспечения является целесообразной, и будет приносить реальную пользу при его использовании в предприятии.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Троелсен Э. C# и платформа .NET. - СПб.:Питер, 2004. - 796 с.-(Библеотека программиста).

2. Солсо Р. Объекто-ориентированное программирование Питер, 2007.-447с. - (Учебник для ВУЗов).

3. Орлов С.А. Технология разработки программного обеспечения - Питер,2003.-464с.-(Учебник для ВУЗов).

4. Тельнов Ю.Ф. Интеллектуальные информационные системы в экономике. Питер,2002.-306с.-(Учебник для ВУЗов).

5. Дюк В., Самойленко А. Data Mining. Издательский дом Питер,2001.-614с.-(Учебник для ВУЗов).

6. Васильев В.П. Информационно-аналитические системы. Питер,2001.-712с.-(Учебник для ВУЗов).

7. Энди Кармайкл, Дэн Хейвуд. Быстрая и качественная разработка программного обеспечения.-Вильямс,2003.-400 с.

8. Айвазян С.А., Бухштабер В.М., Юнюков И.С., Мешалкин Л.Д. Прикладная статистика. Питер,2001.-329с.-(Учебник для ВУЗов).

9. Knowledge Discovery Through Data Mining: What Is Knowledge Discovery? - Tandem Computers Inc., 1996.

10. Кречетов Н. Продукты для интеллектуального анализа данных. - Рынок программных средств, N14-15_97, с.32-39.

11. Boulding K.E. General Systems Theory - The Skeleton of Science // Management Science, 2, 1956.

12. Гик Дж., Прикладная общая теория систем. - М.: Мир, 1981. Питер,2002.-612с.-(Учебник для ВУЗов).

13. Киселев М., Соломатин Е. Средства добычи знаний в бизнесе и финансах. - Открытые системы, № 4, 1997, с.41-44.

ресурсы:

1. Официальная документация по Visual Studio [Электронный ресурс] - URL: http://visualstudio.com/support/documentation (дата обращения: 06.02.2017).

2. Официальное сообщество разработчиков на Visual Studio [Электронный ресурс] -URL: http://Visualstudio.com/support/community (дата обращения: 26.02.2017).

3. Русское сообщество разработчиков на Visual Studio [Электронный ресурс] - URL: http://Cyberforum.com (дата обращения: 06.04.2017).

4. Типовые примеры и решения при разработке приложений на C# [Электронный ресурс] - URL: http://metaint.ru (дата обращения: 01.03.2017).

5. Справочная информация по устройству операционной системы Windows 8 [Электронный ресурс] - URL: http://www.MSDN.com/support/iphone (дата обращения: 11.02.2017).

6. Справочная информация по устройству СУБД SQlite [Электронный ресурс] - URL: http://www.

7. Ахмедов А. Technical Design Document: что, зачем и как [Электронный ресурс]. Режим доступа:http://dtf.ru/articles/read.php?id=46398

SQlite.com/support (дата обращения: 26.03.2017).

8. Официальная документация по Entity Framework6 [Электронный ресурс] URL: http://Msdn.com/EF6/features/ publishing.html (дата обращения: 12.04.2017).

9. Орловский. С. Нас ждет ренессанс стратегий [Электронный ресурс]. Режим доступа:http://kanobu.ru/articles/nas-zhdet-renessans-strategij-300471/

ПРИЛОЖЕНИЯ

XAML разметка страницы

```
<Window

      xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

      xmlns:toolkit="http://schemas.microsoft.com/wpf/2008/toolkit"

        xmlns:d="http://schemas.microsoft.com/expression/blend/2008" xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006" mc:Ignorable="d" x:Class="WpfApplication1.MainWindow"

      Title="MainWindow" Height="500" Width="900" Background="#FF3A3A3A">

   <Grid x:Name="grid1">

       <TabControl HorizontalAlignment="Left" Height="470" VerticalAlignment="Top" Width="884" BorderBrush="{x:Null}" Background="#FF3A3A3A">

          <TabItem Header="Занесение" BorderBrush="{x:Null}" Foreground="#FFFF9E09" Background="#FF5F5F5F">

             <Grid Background="#FF3A3A3A">

                          <Label Content="Ф.И.О.:" HorizontalAlignment="Left" Margin="55,73,0,0" VerticalAlignment="Top" Foreground="#FFCE9300" FontSize="14" FontFamily="Century751 BT"/>

                                  <TextBox x:Name="t1" HorizontalAlignment="Left" Height="23" TextWrapping="Wrap" VerticalAlignment="Top" Width="190" Margin="146,74,0,0"/>

                          <Label Content="Улица" HorizontalAlignment="Left" Margin="55,101,0,0" VerticalAlignment="Top" Foreground="#FFCE9300" FontSize="14" FontFamily="Century751 BT"/>

                                  <TextBox x:Name="t2" HorizontalAlignment="Left" Height="23" TextWrapping="Wrap" VerticalAlignment="Top" Width="190" Margin="146,102,0,0"/>

                           <Label Content="Дом" HorizontalAlignment="Left" Margin="55,129,0,0" VerticalAlignment="Top" Foreground="#FFCE9300" FontSize="14" FontFamily="Century751 BT"/>

                                  <TextBox x:Name="t3" HorizontalAlignment="Left" Height="23" TextWrapping="Wrap" VerticalAlignment="Top" Width="190" Margin="146,130,0,0"/>

                           <Label Content="Тел." HorizontalAlignment="Left" Margin="55,158,0,0" VerticalAlignment="Top" Foreground="#FFCE9300" FontSize="14" FontFamily="Century751 BT"/>

                                  <TextBox x:Name="t4" HorizontalAlignment="Left" Height="23" TextWrapping="Wrap" VerticalAlignment="Top" Width="190" Margin="146,158,0,0"/>

                          <Label Content="Дата" HorizontalAlignment="Left" Margin="55,184,0,0" VerticalAlignment="Top" Foreground="#FFCE9300" FontSize="14" FontFamily="Century751 BT"/>

                                  <TextBox x:Name="t5" HorizontalAlignment="Left" Height="23" TextWrapping="Wrap" VerticalAlignment="Top" Width="190" Margin="146,186,0,0"/>

                         <Label Content="Время" HorizontalAlignment="Left" Margin="55,213,0,0" VerticalAlignment="Top" Foreground="#FFCE9300" FontSize="14" FontFamily="Century751 BT"/>

                                  <TextBox x:Name="t6" HorizontalAlignment="Left" Height="23" TextWrapping="Wrap" VerticalAlignment="Top" Width="190" Margin="146,214,0,0"/>

                           <Label Content="Стоимость" HorizontalAlignment="Left" Margin="55,241,0,0" VerticalAlignment="Top" Foreground="#FFCE9300" FontSize="14" FontFamily="Century751 BT"/>

                                  <TextBox x:Name="t7" HorizontalAlignment="Left" Height="23" TextWrapping="Wrap" VerticalAlignment="Top" Width="190" Margin="146,243,0,0"/>

                      <Button Content="сохранить" HorizontalAlignment="Left" Margin="359,357,0,0" VerticalAlignment="Top" Width="75" Click="Button_Click_1" BorderBrush="{x:Null}" Background="#FF515151" Foreground="#FFFF9A00"/>
```

```
                    <CheckBox x:Name="check" Content="акция" HorizontalAlignment="Left" Margin="670,231,0,0"
VerticalAlignment="Top"/>

                     <CheckBox x:Name="citem" Content="item" HorizontalAlignment="Left" Margin="670,281,0,0"
VerticalAlignment="Top" d:IsHidden="True"/>

                      <CheckBox x:Name="ccola" Content="cola" HorizontalAlignment="Left" Margin="670,323,0,0"
VerticalAlignment="Top" d:IsHidden="True"/>

                    <ListBox HorizontalAlignment="Left" Height="196" Margin="439,73,0,0" VerticalAlignment="Top"
Width="154">

            <Button Content="пепперони"/>

            <Button Content="канзас"/>

            <Button Content="сенатор"/>

            <Button Content="чизбургер"/>

            <Button Content="инфаркт"/>

            <Button/>

            <Button Content="гавайская"/>

            <Button Content="чикен ранч"/>

            <Button Content="чикен барбекю"/>

            <Button Content="цезарь"/>

            <Button Content="мясная"/>

          </ListBox>

                              <Button Content="Добавить блюдо" HorizontalAlignment="Left" Margin="439,293,0,0"
VerticalAlignment="Top" Width="154" Click="Button_Click_4"/>

        </Grid>

      </TabItem>

      <TabItem Header="отображение" BorderBrush="{x:Null}" Background="#FF5F5F5F">

        <Grid>

            <Canvas x:Name="cnv" HorizontalAlignment="Left" Height="400" VerticalAlignment="Top" Width="880"
Margin="0" Background="#FF3A3A3A">

            <Label Content="12.00" Canvas.Top="365" Foreground="#FFFF7300" FontSize="8" Canvas.Left="-4"/>

            <Label Content="1.00" Foreground="#FFFF7300" FontSize="8" Canvas.Left="-3" Canvas.Top="-6"/>

            <Label Content="17.00" Canvas.Top="222" Foreground="#FFFF7300" FontSize="8" Canvas.Left="-4"/>

            <Label Content="20.00" Canvas.Top="137" Foreground="#FFFF7300" FontSize="8" Canvas.Left="-3"/>

            <Label Content="490R" Canvas.Top="397" Foreground="#FFFF7300" FontSize="8" Canvas.Left="-5"/>

            <Label Content="1450R" Canvas.Top="395" Foreground="#FFFF7300" FontSize="8" Canvas.Left="848"/>

          </Canvas>

          <Line X1 = "1"  Y1 = "400" X2 = "1" Y2 = "1" Stroke="DarkOrange" />

          <Line X1="1" Y1="400" X2="880" Y2="400" Stroke="DarkOrange" />
```

```
                    <Button Content="отобразить" HorizontalAlignment="Left" VerticalAlignment="Top" Width="85" Margin="141,403,0,0" Click="Button_Click" BorderBrush="{x:Null}" Background="#FF8F8F8F" Height="29"/>

                <Button Content="." HorizontalAlignment="Left" VerticalAlignment="Top" Width="75" Margin="37,408,0,0" Click="Button_Click_3" BorderBrush="{x:Null}" Background="#FF3A3A3A"/>

            <Label x:Name="label1" Content="" HorizontalAlignment="Left" VerticalAlignment="Top" Margin="353,403,0,0" Background="{x:Null}" Foreground="#FFF87200"/>

        </Grid>

      </TabItem>

    </TabControl>


  </Grid>

</Window>
```

Файл APP.CONFIG

```
<?xml version="1.0" encoding="utf-8"?>

<configuration>

 <configSections>

    <section name="entityFramework" type="System.Data.Entity.Internal.ConfigFile.EntityFrameworkSection, EntityFramework, Version=6.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" requirePermission="false" />

 </configSections>

 <connectionStrings>

  <add name="DC" connectionString="Data Source=.\main.db" providerName="System.Data.SQLite" />

 </connectionStrings>

 <startup>

  <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.6" />

 </startup>

 <entityFramework>

            <defaultConnectionFactory type="System.Data.Entity.Infrastructure.LocalDbConnectionFactory, EntityFramework">
```

```
    <parameters>
      <parameter value="v11.0" />
    </parameters>
  </defaultConnectionFactory>
  <providers>
      <provider invariantName="System.Data.SQLite" type="System.Data.SQLite.EF6.SQLiteProviderServices, System.Data.SQLite.EF6"/>
      <provider invariantName="System.Data.SqlClient" type="System.Data.Entity.SqlServer.SqlProviderServices, EntityFramework.SqlServer" />
    <provider invariantName="System.Data.SQLite.EF6" type="System.Data.SQLite.EF6.SQLiteProviderServices, System.Data.SQLite.EF6" />
  </providers>
 </entityFramework>
 <system.data>
  <DbProviderFactories>
      <add name="SQLite Data Provider (Entity Framework 6)" invariant="System.Data.SQLite.EF6" description=".NET Framework Data Provider for SQLite (Entity Framework 6)" type="System.Data.SQLite.EF6.SQLiteProviderFactory, System.Data.SQLite.EF6" />
      <add name="SQLite Data Provider" invariant="System.Data.SQLite" description=".NET Framework Data Provider for SQLite" type="System.Data.SQLite.SQLiteFactory, System.Data.SQLite" />
  </DbProviderFactories>
 </system.data>
</configuration>
```

Листинг загрузки окна

```
using System;
using System.Collections.Generic;
using System.Linq;
```

```
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
using System.Data.SQLite;
using System.Data.Entity;
using System.ComponentModel.DataAnnotations;
using Microsoft.Windows.Controls;
public MainWindow()
      {
         InitializeComponent();


         for (byte i = 1; i < 88; i++)
         {
            Line a = new Line();
            a.X1 = i * 10;
            a.X2 = i * 10;
            a.Y1 = 395;
            a.Y2 = 400;
```

```
            a.Stroke = Brushes.DarkOrange;
            cnv.Children.Add(a);
        }
        for (byte i = 0; i < 14; i++)
        {
            Line a = new Line();
            a.X1 = 1;
            a.X2 = 5;
            a.Y1 = i * 28.5;
            a.Y2 = i * 28.5;
            a.Stroke = Brushes.DarkOrange;

            cnv.Children.Add(a);
        }

    }
```

Описанные классы отображения базы данных

```
public class Customer
    {
        [Key] public int customer_id { get; set; }
        public string name { get; set; }
        public string sname { get; set; }
        public string street { get; set; }
        public int house { get; set; }
```

```
    public int number { get; set; }


    // Ссылка на заказы
   // public virtual List<Order> Orders { get; set; }
}


public class Order
{
    [Key] public int order_id { get; set; }
    public double date { get; set; }
    public double time { get; set; }
    public int amount { get; set; }
    public int customer_id { get; set; }
    public int tel { get; set; }
    public int offer { get; set; }
    public int item { get; set; }
    public int cola { get; set; }
   // public DateTime PurchaseDate { get; set; }
    // Ссылка на покупателя
    //public Customer Customer { get; set; }
}
public class Item
{
    [Key]public int item_id { get; set; }
    public int price { get; set; }
    public string item_name { get; set; }
```

```
}
public class Orderitem
{
    public int id { get; set; }
    public int order_id { get; set; }
    public int item_id { get; set; }

}
public class maincontext : DbContext
{
    // Имя будущей базы данных можно указать через
    // вызов конструктора базового класса
    public maincontext()
        : base("DC") { }
    // Отражение таблиц базы данных на свойства с типом DbSet
    public DbSet<Customer> CUSTOMERS { get; set; }
    public DbSet<Order> ORDERS { get; set; }
    public DbSet<Item> ITEMS { get; set; }
    public DbSet<Orderitem> ORDERITEMS { get; set; }
}
```

Функции рисования

Ellipse CreateEllipse(double X, double Y)

```
{
    int width = 5;
    int height = 5;
    Ellipse ellipse = new Ellipse { Width = width, Height = height };
    SolidColorBrush mySolidColorBrush = new SolidColorBrush();
    mySolidColorBrush.Color = Color.FromArgb(255, 100, 255, 0);
    ellipse.Fill = mySolidColorBrush;
    Canvas.SetLeft(ellipse, (X -490)*0.9);
    Canvas.SetBottom(ellipse, (Y *28)-310);
    string s = Convert.ToString(X)+"P,  время"+Convert.ToString(Y);
    ellipse.ToolTip = s;
    return ellipse;
}
    Ellipse CreateEllipseRed(double X, double Y)
{
    int width = 5;
    int height = 5;
    Ellipse ellipse = new Ellipse { Width = width, Height = height };
    SolidColorBrush mySolidColorBrush = new SolidColorBrush();
    mySolidColorBrush.Color = Color.FromArgb(255, 255, 0, 0);
    ellipse.Fill = mySolidColorBrush;
    Canvas.SetLeft(ellipse, (X -490)*0.9);
    Canvas.SetBottom(ellipse, (Y *28)-310);
    string s = Convert.ToString(X)+"P,  время"+Convert.ToString(Y);
    ellipse.ToolTip = s;
    return ellipse;
}
```

Запросы к базе данных

```
int sum = 0;
            maincontext db = new maincontext();
          var orders = from o in db.ORDERS
                    select o;


          foreach (Order o in orders)
          {
             sum = sum + o.amount;
             int s = o.amount;
             double t = o.time;
             cnv.Children.Add(CreateEllipse(s, t));
          }
          sum = sum / orders.Count();
          label1.Content ="среднее - "+ sum;


maincontext db = new maincontext();
          var query1 = from o in db.ORDERS
                    where o.offer == 1
                    select o;
          foreach (Order o in query1)
          {
             int s = o.amount;
             double t = o.time;
             cnv.Children.Add(CreateEllipse(s, t));
          }
```

```
var query2 = from o in db.ORDERS
             where o.offer == 1 && o.amount > 999
             select o;
int s2 = query2.Count();



var query3 = from o in db.ORDERS
             where o.offer == 0 && o.amount > 999
             select o;
int s3 = query3.Count();
MessageBox.Show(Convert.ToString(s2) + Convert.ToString(s3));
```