ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ»

(НИУ «БелГУ»)

ИНСТИТУТ ИНЖЕНЕРНЫХ ТЕХНОЛОГИЙ И ЕСТЕСТВЕННЫХ НАУК

КАФЕДРА МАТЕМАТИЧЕСКОГО И ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ИНФОРМАЦИОННЫХ СИСТЕМ

«Создание кросс-платформенного приложения для администрирования БД спортивных ставок с использованием технологии FireDAC»

Выпускная квалификационная работа
обучающегося по направлению подготовки02.03.02 Фундаментальная информатика и информационные технологии
заочной формы обучения, группы 07001250
Цыкозы Александра Николаевича

Научный руководитель:
к.т.н., доцент
П.В. Васильев

БЕЛГОРОД 2017

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

С тех пор, как человек стал познавать мир вокруг себя, ему перестало хватать собственной памяти для того, чтобы учитывать все, что происходит вокруг. Со времен появления письменности, задача упростилась.

Древнейшим физическим хранилищем данных можно считать «кипу» - образец узелковой письменности инков. «Кипу» - древняя мнемоническая и счётная система инков; представляет собой сложные верёвочные сплетения и узелки, изготовленные из шерсти верблюдов или хлопка. В кипу могло быть от нескольких свисающих нитей до 2000. Она использовалась как для передачи сообщений, так и в других аспектах общественной жизни - в качестве календаря, топографической системы, для фиксации налогов, законов и прочего. Один из испанских хронологов писал, что «вся империя инков управлялась посредством кипу».

Следующим этапом развития хранилищ данных, можно считать книги. С момента появления полноценной письменности, стали вестись амбарные книги. Их преимущества, по сравнению с теми же самыми «кипу» очевидны - занимают меньше места и не требуют особых, сверхсложных, навыков для ведения.

В 20 веке понятие хранилища данных существенно меняется. С 1955 г., когда появилось программируемое оборудование обработки записей, начинается история хранилищ данных в том смысле, который мы теперь применяем. Гражданский кодекс РФ (глава 70, статья 1260) гласит, что базой данных является представленная в объективной форме совокупность самостоятельных материалов (статей, расчетов,

нормативных актов, судебных решений и так далее), систематизированных таким образом, чтобы эти материалы могли быть найдены и обработаны с помощью электронной вычислительной машины (ЭВМ)[7].

В соответствии с другим определением, введенным Кристофером Дейтом в его классическом учебнике «Введение в системы баз данных», база данных - это организованная в соответствии с определёнными правилами и поддерживаемая в памяти компьютера совокупность данных, характеризующая актуальное состояние некоторой предметной области и используемая для удовлетворения информационных потребностей пользователей [6,10].

Отличительной особенностью современных баз данных является то, что они хранятся и обрабатываются в вычислительной системе. В связи с тем, что современные базы данных имеют огромные объемы, а системы управления ими довольно сложны, со временем появилось отдельное направление в информационных системах и технологиях - администрирование баз данных.

Букмекер — профессия профессионального спорщика, занимающегося приёмом денежных ставок на различные предстоящие события (в основном спортивные) с заранее оговоренными вероятностями, которые определяются коэффициентами, а также выплатой выигрышей. Для букмекера важно уметь хорошо прогнозировать (самостоятельно или на основании других прогнозов) предполагаемые вероятности событий, на которые принимаются ставки, и будущую популярность различных ставок среди потенциальных клиентов [15].

Целью данной дипломной работы является создание кросс-платформенного приложения для администрирования базы данных спортивных ставок с использованием технологии FireDAC.

Актуальность работы заключается в том, что в рамках работы букмекерской конторы возникла необходимость в администрировании базы данных с возможностью доступа к ней с различных устройств.

Разработка приложения, поддерживающего работу с системой ставок, обеспечит кроссплатформенность и удобство сбора статистики и аналитики по ставкам. Таким образом, программа поможет систематизировать полученную о клиентах информацию и коэффициенты ставок.

За счет кроссплатформенности программа станет доступна и на устройствах с малой вычислительной мощностью. А клиентский интерфейс позволит администрировать базу данных как специалисту, так и обычному работнику.

Исходя из поставленной цели, в выпускной квалификационной работе рассмотрена техническая платформа для реализации приложения, приведены поставленные к системе требования. Рассмотрен полный цикл разработки программы. Задачи дипломной работы:

- анализ существующих видов ставок;
- анализ и расчеты исходов ставок;
- обзор технологий для работы с базами данных;
- постановка задачи для создаваемого приложения;

- настройка соединения с базой данных с использованием технологии FireDAC;
- разработка кросс-платформенного приложения для администрирования базы данных.

Дипломная работа состоит из введения, трех глав, заключения и списка использованных источников. Объем работы – 50 страниц, в работе содержится 14 рисунков и 7 таблиц и приложения.

В первой главе рассматривается механизм принятия и обработки ставок, их виды, также дается постановка задачи,

Во второй главе произведено проектирование архитектуры приложения, выделены требования, выдвигаемые к системе и выбраны средства реализации приложения.

В третьей главе описаны процессы реализации и тестирования разработанного приложения.

ГЛАВА 1. ТЕОРЕТИЧЕСКИЕ ОСНОВЫ РАБОТЫ СИСТЕМ ПРИНЯТИЯ СТАВОК

В данной главе даны основные понятия и определения, используемые при работе со ставками и установлены требования, которым должно соответствовать разрабатываемое приложение для администрирования ставок на спорт.

1.1 Общие понятия работы системы принятия ставок

Букмекер — профессиональный спорщик, занимающегося приёмом денежных ставок на различные предстоящие события (чаще всего спортивные) с заранее оговоренными вероятностями, которые определяются коэффициентами, а также выплатой выигрышей (рис. 1.1). Для букмекера важно уметь хорошо прогнозировать (самостоятельно или на основании других прогнозов) предполагаемые вероятности событий, на которые принимаются ставки, и будущую популярность различных ставок среди потенциальных клиентов [14, 16, 32].

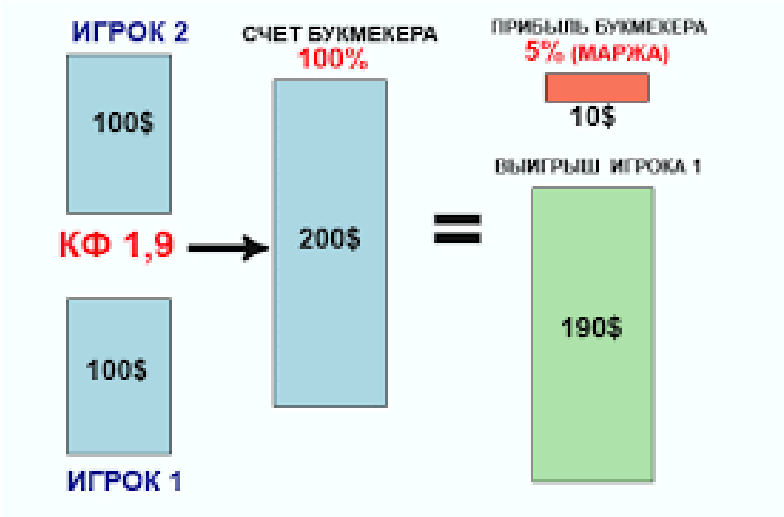

Рис.1.1 Выплата выигрыша и доля букмекера

Основные понятия, используемые в системе принятия ставок, это:

Линия - список событий и их исходов с коэффициентами выигрыша, предлагаемых букмекерской конторой для ставок.

Ставка – те деньги, которые игрок ставит на игру и теряет при проигрыше, а также совокупность выборов, которые игрок объединил, поставив на них определенную сумму.

Исход - результат спортивного события, которому букмекерская контора присвоила коэффициент выигрыша.

Также можно выделить несколько видов ставок:

- Одиночная - ставка на 1 исход события (таблица 1.1). Выигрыш по одиночной ставке равен произведению суммы ставки на коэффициент.

Таблица 1.1. Одиночная ставка

Команда 1	Команда 2	Победа 1	Ничья X	Победа 2
Спартак	Динамо	2.0	3.2	4.3

Коэффициент выигрыша по ставкам на победу Спартака - 2.0. Выплата при победе Спартака составит 100 х 2.0 = 200. Чистый выигрыш: 200 (выплата) - 100 (ставка)=100 рублей.

- Экспресс - ставка на несколько событий (таблица 1.2). Для выигрыша по экспрессу нужно, чтобы все события в экспрессе не проиграли (то есть либо выигрыш, либо возврат). Если хотя бы одно событие проиграло, то это влечет за собой и проигрыш всей ставки. Выигрыш по экспрессу равен произведению суммы ставки на коэффициенты всех исходов, входящих в экспресс.

Таблица 1.2. Экспресс ставка

Команда 1	Команда 2	Победа 1	Ничья X	Победа 2
Спартак	Динамо	2.0	3.2	4.3
Шахтер	Динамо К	1.8	3.2	5.0
Челси	Арсенал	2.3	2.9	3.3

Если, делается ставка 100 рублей на экспресс, состоящий из трех исходов: победа Спартака, победа Шахтера, победа Арсенала. Коэффициенты выигрышей по исходам, входящим в экспресс, перемножаются: 2.0 х 1.8 х 3.3=11.88. Выплата, если все выбранные исходы состоятся, составит: 100 х 11.88=1188 рубля, в том числе чистый выигрыш (за вычетом суммы ставки): 1088 рубля.

- Система - совокупность экспрессов, представляющая собой полный перебор вариантов экспрессов одного размера из фиксированного набора исходов. Сумма ставки на каждый экспресс (вариант системы) одинакова и количество исходов в каждом экспрессе одинаково. Выигрыш по системе равен сумме выигрышей по экспрессам, входящим в систему [21].

Основные выборы исходов для ставок и их обозначение в линии букмекерских контор

- 1 - победа первой команды.
- X – этим символом у букмекеров обозначается ничья.
- 2 - победа второй команды.
- 1X - победа первой команды или ничья. Для выигрыша по такой ставке надо, чтобы победила команда 1 или произошла ничья.

- 12 - победа первой команды или победа второй команды. Чтобы эта ставка сыграла, в матче не должно быть ничьей.
- X2 – смотрите 1X, применяется ко 2й команде.

Рассмотрим ставки на исходы, которые актуальны и популярны во всех видах спорта. Ставя на эти выборы, игрок получает возможность разнообразить круг событий для игры и варьировать свои тактики в ставках.

Победа участника соревнования с учетом форы. В линии это выглядит следующим образом: "фора 1 => кф 1". Итак, вы открываете линию и видите матч Спартак – Амкар. Форы на этот матч будут выставлены таким образом: фора 1 (-1,5) 1,8 и фора 2 (+1,5) 1,9. Фора 1 будет касаться только мячей 1 команды (соответственно, Спартака), а фора 2 относится только к голам Амкара. Если выбрать фору Амкара +1,5, то ставка идет на то, что эта команда проиграет максимум в 1 мяч, сыграет вничью или победит. К конечному результату голов Амкара вы прибавляете значение форы. Допустим, счет матча 2:1. Прибавив фору, мы получаем 2:2,5, то есть победу Амкара. В случае ставки на минусовую фору Спартака вас устроил бы счет 2-0, 3-0, 3-1 и так далее, разница минимум в 2 мяча.

Тотал в футболе – количество голов в матче. Если конторой установлен тотал футбольного матча (сумма забитых мячей), равный "3", и предлагается заключение ставок на тотал по двум исходам: больше или меньше данного тотала.

- если в матче будет забито меньше 3-х мячей, ставки на меньше "3" выигрывают, на больше проигрывают;

- если в матче будет забито больше 3-х мячей, ставки на меньше "3" проигрывают, на больше "3" выигрывают;
- если в матче будет забито ровно 3 мяча, всем ставкам на больше "3" и на меньше "3" будет присвоен коэффициент выигрыша "1".

Это и есть все основные виды исходов для ставок. Они имеют свои вариации, но основной принцип основан именно на этих понятиях. Для работы системы ставок важно следить за линией, резкий обвал или взлет коэффициентов обычно неслучаен.

1.2 Теория игр в ставках на спорт

Одна из особенностей ставок на тотализаторе, которая интересна пользователям критерия Кэлли, - возможность исследования системы ставок для нескольких игр одновременно.

Пример 1. Предположим, мы делаем ставки одновременно в двух независимых играх с положительным ожиданием, представляющих собой подбрасывание монеты, с долями f_1 и f_2, а также с вероятностями успеха p_1 и p_2 соответственно. Тогда ожидаемый уровень роста определяется как (1.1) и (1.2):

$$\begin{aligned} g(f_1, f_2) &= p_1 p_2 ln(1 + f_1 + f_2) + p_1 q_2 ln(1 + f_1 - f_2) \\ &+ q_1 p_2 ln(1 - f_1 + f_2) + q_1 q_2 ln(1 - f_1 - f_2) \end{aligned} \tag{1.1}$$

$$\partial g / \partial f_1 = 0 \tag{1.2}$$

Для нахождения оптимальных f_1^* и f_2^* мы решаем одновременные уравнение (3):

$$f_1 + f_2 = \frac{p_1 p_2 - q_1 q_2}{p_1 p_2 + q_1 q_2} \equiv c \tag{1.3}$$

Результат будет следующим (4):

$$f_1 - f_2 = \frac{p_1 q_2 - q_1 p_2}{p_1 q_2 + q_1 p_2} \equiv d$$
$$f_1^* = (c + d)/2 \qquad f_2^* = (c - d)/2 \tag{1.4}$$

Эти уравнения проходят проверку на симметричность: перестановка 1.1 и 1.2 на протяжении всех преобразований переводит уравнение в себя.

Альтернативная форма поучительна. Возьмем $m_i=p_i-q_i$, $I=1,2$; отсюда $p_i=(1+m_i)/2$ и $q_i=(1-m_i)/2$. Замена в (1.4) и упрощения приводят к (1.5):

$$c = \frac{m_1 + m_2}{1 + m_1 m_2} \quad d = \frac{m_1 - m_2}{1 - m_1 m_2}$$
$$f_1^* = \frac{m_1 (1 - {m_2}^2)}{1 - {m_1}^2 {m_2}^2} \quad f_2^* = \frac{m_2 (1 - {m_1}^2)}{1 - {m_1}^2 {m_2}^2} \tag{1.5}$$

что явно дает множители, на величину которых f_i^* уменьшаются в зависимости от m_i^*.

Так как m_i обычно несколько процентов, уменьшающие множители обычно очень близки к 1.1.

Рассмотрим частный случай, когда $p_1 = p_2 = p$, $d=0$ и $f^* = f_1^* = f_2^* = c/2 = (p-q)/(2(p^2+q^2))$. Полагая $m = p-q$ это может быть записано $f^* =$

$m/(1+m^2)$ для оптимальной доли ставок на обе монеты одновременно, по сравнению с $f^* = m$ для ставки на каждую монету последовательно.

Следующий пример – простая иллюстрация важного эффекта ковариации для оптимальной доли ставки.

Пример 2. Имеем две игры с подбрасыванием монеты как и в прошлом примере, но теперь на них не накладывается ограничение на независимость их исходов. Для простоты возьмем частный случай, когда две ставки имеют одинаковое распределение выплат, но с объединенным распределением, показанным в *Таблице 6.1*.

Таблица 1.3 Объединенное распределение двух идентичных игр с положительным ожиданием с коррелированными исходами

X_1 :	X_2 : 1	−1
1	c+m	b
-1	b	c

Тогда $c+m+b = (1+m)/2$, откуда $b=(1-m)2-c$ и, следовательно, $0 \le c \le (1-m)/2$.

Вычисления дают $Var(X_i)=1-m^2$, $Cor(X_1,X_2) = 4c-(1-m)^2$ и $Cor(X_1,X_2) = [4c-(1-m)^2]/(1-m^2)$.

Результат будет

$f^* = m/(2(2c+m))$. Мы видим, что для фиксированного m, по мере того как c уменьшается с $(1-m)/2$ при $Cor(X_1,X_2) =1$ до 0 при

Cor(X_1,X_2) = -(1-m)/(1+m), f^* для каждой ставки увеличивается с m/2 до ½, как показано в таблице 1.4.

Таблица 1.4 Увеличение f^* при уменьшении Cor(X_1,X_2)

$\mathrm{Cor}(X_1, X_2)$	c	f^*
1	$(1-m)/2$	$m/2$
0	$(1-m^2)/4$	$m/(1+m^2)$
$-(1-m)/(1+m)$	0	$1/2$

Важно отметить, что для точного решения или произвольной численной аппроксимации задачи одновременных ставок недостаточно знать только ковариации или корреляции [18,19,25]. Для построения функции *g* нам необходимо полное объединенное распределение.

Фундаментальной проблемой в играх является поиск возможностей ставок с положительным ожиданием. Аналогичная проблема в инвестировании – поиск возможностей инвестирования с «избыточной», с учетом поправок на риск, доходностью. Как только такие благоприятные возможности идентифицированы, игрок или инвестор должен решить, какую часть своего капитала поставить на кон (вложить).

1.3 Стратегии ставок на спорт

Игровых и финансовых стратегий ставок на спорт в можно найти очень много. Рассмотрим самые распространенные из них:

1.3.1 Игровые стратегии

- Ставка на недооцененное букмекером событие. Игроки стараются спрогнозировать исход события, не задумываясь о прибыльности своей игры на длительном отрезке времени. Но есть и другая категория, ориентирующихся в первую очередь на величину коэффициентов и прибыль в перспективе длинной игры. Игроки такой категории делают ставки, исключительно выгодные в длительной перспективе. Если делать ставки только на завышенные коэффициенты, то прибыль в перспективе обеспечена. Подобный подход и называют ставками на недооцененные букмекером события [22,23].

- Букмекерские вилки – это несколько ставок на определенное событие в различных букмекерских конторах, при которых обеспечен выигрыш независимо от результата избранного события. Такая ситуация возможна из-за разной оценки вероятности одного и того же события различными букмекерами. Букмекеры, как правило, не любят такую стратегию и стараются всячески пресечь.

- Догоном называют стратегию ставок на спорт, где величина каждой очередной ставки определяется в зависимости от результата предыдущей. Главной целью является возвращение ранее проигранных денег и приобретение некоторого дохода. Например, вы «догоняете» ничью в футболе и ставите постоянно на нее, пока это не случится. При этом с каждым разом вы увеличиваете ставку таким образом, чтобы вернуть предыдущие проигрыши и получить прибыль. Это событие обязательно случится, но вполне возможно, что к тому времени у вас просто не хватит денег на ставку.

- Коридоры. Стратегия схожая с вилками. В данном случае у разных букмекеров ищется «коридор» в тоталах или форах. В случае удачи (результат матча вписывается в выбранный «коридор») играют обе ставки и получается приличный выигрыш. В противном случае – расход и игрок ничего не теряет. Последствия такой игры аналогичны игре на вилках.

При администрировании базы данных важно будет учесть и предугадать использование той или иной стратегии клиентом.

1.3.2 Финансовые стратегии

- Флэт — самая простая и в тоже время самая безопасная финансовая стратегия, согласно которой все ставки должны быть одинаковой величины. Суть стратегии заключается в том, что если игрок решил делать ставки по 3 (5, 10, …) единиц, то принятый размер ставки не меняется в течение продолжительного периода. Увеличение или уменьшение банкролла не являются причиной для изменения величины ставки [13,24,26].
- Фиксированная прибыль – более прогрессивная стратегия по сравнению с флэтом. Здесь ставка «B» прямо пропорциональная предвидимому возврату «W» и обратно пропорциональная величине коэффициента «K» рассматриваемого события. Размер ставки определяется по формуле B = W / (K — 1).

Стратегия предусматривает искусственное задание величины прибыли (фиксированной прибыли). Безопасной величину прибыли считают в пределах 1/20 – 1/30 от банкролла, а рекомендуемые коэффициенты – от 1,3 до 5.

- Стратегия Мартингейла – суть стратегии в том, что игрок задается размером первой ставки и в случае проигрыша увеличивает размер каждой последующей ставки в два раза – пока не выиграет. После выигрыша начинается новая серия с начальной ставки. При этом коэффициенты на события должны быть не менее 2.
- Критерий Келли. Стратегия заключается в вычислении приемлемого размера ставки на выбранное событие. Оптимальная ставка вычисляется по следующей формуле: C = (K x V — 1) / (K — 1), где K — коэффициент, V — вероятность события, C — величина ставки.

1.4 Постановка задачи

На основании анализа работы букмекерской конторы к создаваемому приложению выделены следующие требования:

- приложение должно наглядно отделять прошедшие и будущие матчи
- приложение должно разделять ставки по их видам
- в приложении должна быть учтена статистика изменения ставок по отдельным матчам
- приложение должно иметь интуитивно понятый интерфейс
- приложение должно быть кроссплатформенным

Перечисленные требования помогут следить за статистикой, аудиторией проекта и повысят удобство использования базы данных.

ГЛАВА 2. ОБЗОР ПРОГРАММЫХ СРЕДСТВ ДЛЯ ПРОЕКТИРОВАНИЕ ПРОГРАММЫ АДМИНИСТРИРОВАНИЯ БАЗЫ ДАННЫХ

В данной главе произведен обзор программах средств для разработки требуемого приложения и рассмотрена структура базы данных.

2.1 Выбор средств разработки

Прежде чем приступить непосредственно к разработке, определимся с инструментами для разработки. Данные, которые будут использоваться в приложении, целесообразнее всего хранить в базе данных.

В качестве СУБД используем SQLite. Эта СУБД обладает рядом преимуществ, среди которых:

- скорость работы,
- простота использования,
- экономичность в отношении ресурсов,
- удобство использования в мобильных приложениях;

Вместе с приложением мы будем работать с уже готовой СУБД. Такой подход обусловлен спецификой задачи.

Для работы с SQLite в Windows понадобится скачать с официального сайта библиотеку. И поместить данную в одну из системных папок [8,28,33].

Доступ к базе данных из приложения будет осуществлен с помощью библиотеки FireDAC.

FireDAC — это универсальная библиотека доступа к данным, предназначенная для разработки приложений, подключаемых к различным базам данных. Библиотека поддерживает следующие СУБД: InterBase, SQLite, MySQL, SQL Server, Oracle, PostgreSQL, DB2, SQL Anywhere, Advantage DB, Firebird, Access, Informix. Однако, на данном этапе, в мобильных приложениях «напрямую» (без использования технологии DataSnap) с помощью FireDAC можно подключиться только к SQLite и InterBase [28,34].

2.2 Обзор основных возможностей RAD Studio

RAD Studio представляет собой набор средств разработки, включающих в себя Delphi, C++ Builder, HTML5 Builder и ряд сопутствующих продуктов. Основное новшество последний версии RAD Studio состоит в том, что с помощью Delphi стало возможным вести кроссплатформенную разработку [1,3,29].

Для создания кроссплатформенных приложений используется платформа приложений FM, ранее известная как FireMonkey. С точки зрения IDE FireMonkey — это прежде всего, новая библиотека визуальных классов (элементов управления). С ее помощью можно создавать качественные пользовательские интерфейсы практически для любых видов программ [2,20].

В то время, как другие библиотеки абстрагируют пользовательский интерфейс, FireMonkey привязывается непосредственно к нативной графической библиотеке, предлагая наилучшее решение с точки зрения использования GPU на целевой платформе.

На сегодняшний день платформа FM поддерживает следующие операционные системы: Windows (Win32 и Win64), OSX, iOS и Android.

В контексте разработки бизнес-приложений так же следует упомянуть и о механизмах доступа к базам данных. Действительно, работа с БД всегда была сильной стороной Delphi, и вполне логично было бы ожидать, что мобильные Delphi приложения будут работать с базами так же хорошо, как и настольные [4,5,30].

Комплект поставки редакций Delphi/RAD Studio пополнился содержит библиотеку доступа к данным – FireDAC, созданную на базе хорошо известного решения AnyDAC, долгое время развиваемого.

FireDAC является универсальным набором компонентов, поддерживающим доступ к весьма внушительному списку СУБД. Платформа FireMonkey с помощью FireDAC поддерживает практически все популярные СУБД. Что же касается мобильных приложений, то здесь существуют определенные ограничения, связанные, прежде всего, с отсутствием библиотек доступа к большинству СУБД.

FireDAC обеспечивает разработчикам поддержку широкого спектра платформ баз данных, включая Oracle, Microsoft SQL Server, IBM DB2, SAP, DataSnap, Sybase SQL Anywhere, InterBase, Advantage Database, PostgreSQL, SQLite, MySQL, Firebird и Microsoft Access.

FireDAC включен в продукты для средств разработки для различных устройств редакции Enterprise и старше, а также будет поставляться как отдельный продукт. Кроме этого, FireDAC работает с REST-серверами DataSnap, обеспечивая возможность создания ультратонких настольных и мобильных клиентских приложений для баз данных, веб-сервисов на основе REST, а также облачных сервисов [9,27].

2.3 Бизнес-требования

Для придания оригинальности разрабатываемому приложению, попытаемся расширить обычный для подобных программ функционал несколькими дополнительными функциями:

Пересчет количества ставок на определенных исход. Обычно при резком изменении преобладающего положительного исхода необходимы внесения корректировок, поэтому будет удобно автоматизировать этот процесс.

Поиск по базе. База насчитывает огромное количество записей, и от пользователя требуются такие данные как: количество поставивших, суммы, данные пользователей, данные победителей и так далее.

Статистика. Статистика позволит корректировать прогнозы ставок и наблюдать за тенденциями рынка. Данное приложение мы реализуем для Windows и для Android. Затем на основе единой базы исходных кодов мы сможем выполнить портирование приложения на MacOS и iOS.

2.4 Схема базы данных

Процесс разработки мы начнем с рассмотрения структуры базы данных. Логическая модель ее приведена на диаграмме (рис.2.1).

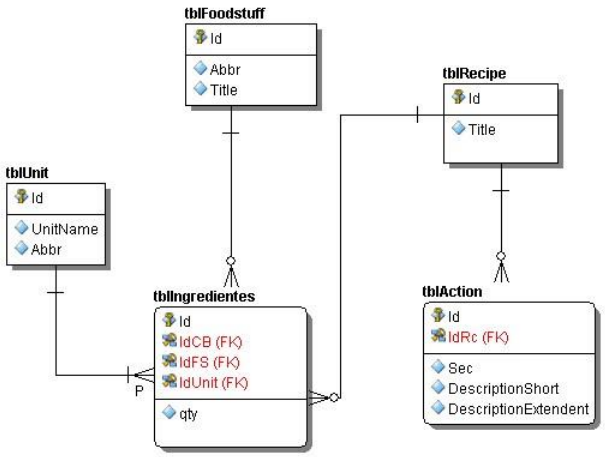

Рис.2.1. Логическая модель базы данных

Данная диаграмма получена с помощью еще одного инструмента от компании Embarcadero Technologies — ER/Studio. Developer редакция этого продукта доступна пользователям RAD Studio Architect.

При создании структуры базы данных каждой сущности логической модели будет соответствовать физическая таблица. Рассмотрим назначение полученных таблиц (рисунки 2.2-2.6).

Таблица событий содержит описания проводимых матчей или других событий для ставок.

Primary	Auto Increment	Field	Data Type	Size	Allow Null	Default
✓	✓	id	integer			
		event_id	integer			
		bet_type_id	integer			
		koef	float			
		time	datetime		✓	CURRENT_TIMESTAMP
		enabled	bool		✓	1

Рис.2.2. Таблица событий

Таблица типов ставок позволяет определить, к какому виду ставок относится данное событие:

Primary	Auto Increment	Field	Data Type	Size	Allow Null	Default
✓	✓	id	integer			
		title	varchar			

Рис.2.3. Таблица типов ставок

Таблица ставок клиентов содержит размеры ставок и номера клиентов.

Primary	Auto Increment	Field	Data Type	Size	Allow Null	Default
✓	✓	id	integer			
		bet_id	integer			
		user_id	integer			
		sum	float			
		time	datetime		✓	CURRENT_TIMESTAMP
		enabled	bool		✓	1

Рис.2.4. Таблица ставок клиентов

Таблица описания событий более детально описывает события.

Primary	Auto Increment	Field	Data Type	Size	Allow Null	Default
✓	✓	id	integer			
		title	varchar			
		time_start	datetime			
		time_end	datetime			
		sport_type_id	integer			

Рис.2.5. Таблица описания события

Таблица клиентов содержит личные данные, которые клиент должен предоставить для ставки.

Primary	Auto Increment	Field	Data Type	Size	Allow Null	Default
✓		id	integer			
		firstname	varchar			
		lastname	varchar			
		email	varchar			
		sum	float			

Рис.2.6. Таблица клиентов

ГЛАВА 3. РЕАЛИЗАЦИЯ СОЗДАНИЕ ДЛЯ АДМИНИСТРИРОВАНИЯ БД СПОРТИВНЫХ СТАВОК

В данной главе будет описан процесс создания приложений для СУБД SQLLite с использованием компонентов доступа FireDac и среды Delphi. FireDac является стандартным набором компонентов доступа к различным базам данных начиная с Delphi.

3.1 Создание нового проекта

Создадим новый проект File->New->VCL Forms Application — Delphi. В новый проект добавим новый дата модуль File->New->Other, в появившемся мастере выберите Delphi Projects->Delphi Files->Data Module. Этот дата модуль будет главным в нашем проекте. Он будет содержать некоторые экземпляры глобальных компонентов доступа, которые должны быть доступны всем формам, которые должны работать с данными. Например, таким компонентом является TFDConnection.

Компонент TFDConnection обеспечивает подключение к различным типам баз данных. Будем указывать экземпляр этого компонента в свойствах Connection остальных компонентов FireDac. К какому именно типу баз данных будет происходить подключение, зависит от значения свойства DriverName. Для того чтобы подключение знало, с какой именно библиотекой доступа необходимо работать, разместим в главном дата модуле компонент TFDPhysDriverLink. Его свойство VendorLib позволяет указывать путь до клиентской библиотеки [11,17,31].

Размещаем необходимую библиотеку доступа в папке dbclient, которая расположена в папке приложения. Для этого для события OnCreate дата модуля пропишем следующий код (листинг 1).

Листинг 1. Пусть к клиентской библиотеке

```
// указываем путь до клиентской библиотеки

xAppPath := ExtractFileDir(Application.ExeName) + PathDelim;

FDPhysFBDriverLink.VendorLib := xAppPath + 'dbclient' + PathDelim + 'dbclient.dll';
```

3.1.1 Параметры подключения

Компонент TFDConnection параметры подключения к базе данных содержатся в свойстве Params (имя пользователя, пароль, набор символов соединения и другие). Если воспользоваться редактором свойств TFDConnection (двойной клик на компоненте), то упомянутые свойства будут заполнены автоматически. Набор этих свойств (таблица 3.1 и рис.3.1) зависит от типа базы данных [11,12].

Таблица 3.1 Параметры TFDConnection

Параметр	Назначение
Pooled	Используется ли пул соединений
Database	Путь к базе данных или её псевдоним, определённый в файле конфигурации aliases.conf (или databases.conf) сервера
User_Name	Имя пользователя
Password	Пароль
OSAuthent	Используется ли аутентификация средствами операционной системы.
Protocol	Протокол соединения. Допускаются следующие значения: Local – локальный протокол; NetBEUI – именованные каналы; SPX – не поддерживается в современных версиях; TCPIP – TCP/IP.
Server	Имя сервера или его IP адрес. Если сервер работает на нестандартном порту, то необходимо также указать порт через слэш, например localhost/3051.
SQLDialect	Диалект. Должен совпадать с диалектом базы данных.
RoleName	Имя роли
CharacterSet	Имя набора символов соединения

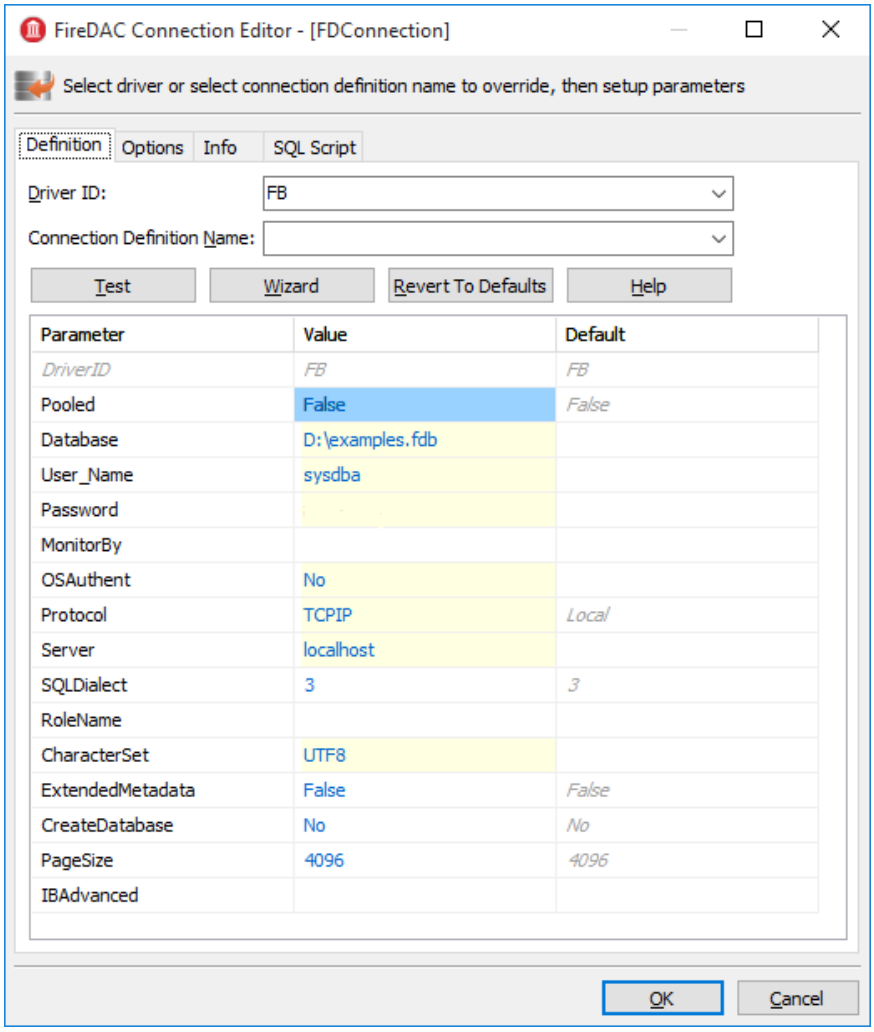

Рис.3.1. Свойства подключения базы данных

Поскольку параметры подключения, за исключением имени пользователя и пароля, обычно не изменяются в процессе эксплуатации приложения, будем считывать их из файла конфигурации (листинг 2).

Листинг 2. Добавление параметров подключения к базе данных в файл конфигурации

```
// считываем параметры подключения
xIniFile := TIniFile.Create(xAppPath + 'config.ini');
try
  xIniFile.ReadSectionValues('connection', FDConnection.Params);
finally
  xIniFile.Free;
end;
```

Содержимое файла config.ini представлено в листинге 3:

Листинг 3. Файл config.ini

```
[connection]
DriverID=SGLite
Protocol=TCPIP
Server=localhost/3051
Database=examples
OSAuthent=No
RoleName=
CharacterSet=UTF8
```

Содержимое секции connection можно получить, скопировав содержимое свойства Params компонента TFDConnection после работы мастера.

3.2 Создание запросов и подключения к базе данных

Для подключения к базе данных необходимо изменить свойство Connected компонента TFDConnection в значение True или вызвать метод Open. В последний метод можно передать имя пользователя и пароль в качестве параметров. Заменим стандартный диалог соединения с базой данных. Дадим возможность ошибиться при вводе регистрационной информации не более трёх раз, после чего

приложение будет закрыто. Для этого напишем код (листинг 4) в обработчике события OnCreate главного датамодуля.

Листинг 4. Обработчик события onCreate

```
// делаем максимум 3 попытки входа в систему, потом закрываем приложение
xLoginCount := 0;
xLoginPromptDlg := TLoginPromptForm.Create(Self);
while (xLoginCount < MAX_LOGIN_COUNT) and
    (not FDConnection.Connected) do
begin
 try
  if xLoginPromptDlg.ShowModal = mrOK then
   FDConnection.Open(
    xLoginPromptDlg.UserName, xLoginPromptDlg.Password)
  else
   xLoginCount := MAX_LOGIN_COUNT;
 except
  on E: Exception do
  begin
   Inc(xLoginCount);
   Application.ShowException(E);
  end
 end;
end;
xLoginPromptDlg.Free;

if not FDConnection.Connected then
 Halt;
```

3.3 Организация работы с данными в FireDAC

Работать с данными в FireDac можно при помощи компонент TFDQuery, TFDTable и TFDStoredProcm, которые унаследованы от TFDRdbmsDataSet.

Помимо наборов данных для работы непосредственно с базой данных, в FireDac существует также компонент TFDMemTable, который предназначен для работы с набором данных в памяти, является аналогом TClientDataSet.

Основным компонентом для работы с наборами данных является TFDQuery. Возможностей этого компонента хватает практически для любых целей. Компоненты TFDTable и TFDStoredProc всего лишь модификации, либо чуть расширенные, либо усеченные.

Назначение компонента — буферизация записей, выбираемых оператором SELECT, для представления этих данных в Grid, а также для обеспечения «редактируемости» записи (текущей в буфере). В отличие от компонента IBX.IBDataSet компонент Query не содержит свойств RefreshSQL, InsertSQL, UpdateSQL и DeleteSQL. Вместо этого «редактируемость» обеспечивается компонентом UpdateSQL (таблица 3.2), который устанавливается в свойство UpdateObject.

Таблица 3.2 Параметры UpdateObject

Параметр	Назначение
Connection	Связь с компонентом FDConnection.
MasterSource	Ссылка на Master-источник данных (TDataSource) для FDQuery, используемого в качестве Detail.
Transaction	Транзакция, в рамках которой будет выполняться запрос, прописанный в свойстве SQL
UpdateObject	Связь с компонентом FDUpdateSQL, который обеспечивает «редактируемость» набора данных.
UpdateTransaction	Транзакция, в рамках которой будут выполняться модифицирующие запросы.
UpdateOptions.CheckRequired	Если свойство CheckRequired установлено в True, то FireDac контролирует свойство Required соответствующих полей.
UpdateOptions.EnableDelete	Определяет, позволяется ли удаление записи из набора данных.
UpdateOptions.EnableInsert	Определяет, позволяется ли вставка записи в набор данных.
UpdateOptions.EnableUpdate	Определяет, позволяется ли изменение записи в наборе данных.
UpdateOptions.FetchGeneratorsPoint	Управляет моментом получения следующего значения генератора указанного в свойстве UpdateOptions.GeneratorName.
UpdateOptions.GeneratorName	Имя генератора для извлечения следующего значения автоинкрементного поля.
UpdateOptions.ReadOnly	Указывает, является ли набор данных только для чтения. По умолчанию False.
UpdateOptions.RequestLive	Установка RequestLive в True делает запрос редактируемым, если это возможно.
UpdateOptions.UpdateMode	Отвечает за проверку модификации записи:
CachedUpdates	Определяет, будет ли набор данных кэшировать изменения без немедленного внесения их в базу данных.
SQL	Содержит SQL запрос. Если это свойство содержит SELECT запрос, то его необходимо выполнять методом

	Open.

Для того чтобы указать SQL команды на этапе проектирования, используем редактор TFDUpdateSQL времени проектирования, который вызывается двойным щелчком по компоненту и имеет вид, показанный на рисунке 3.2.

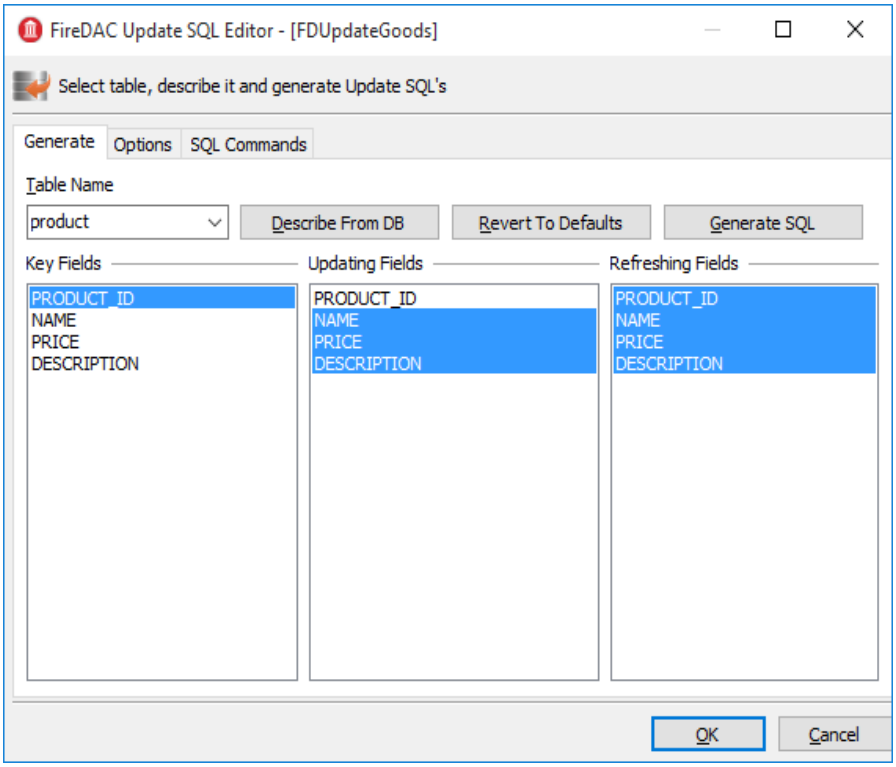

Рис.3.2. Задание SQL команд

После чего запросы будут сгенерированы автоматически, и на вкладке «SQL Commands», можно будет поправить каждый из запросов (рисунок 3.3).

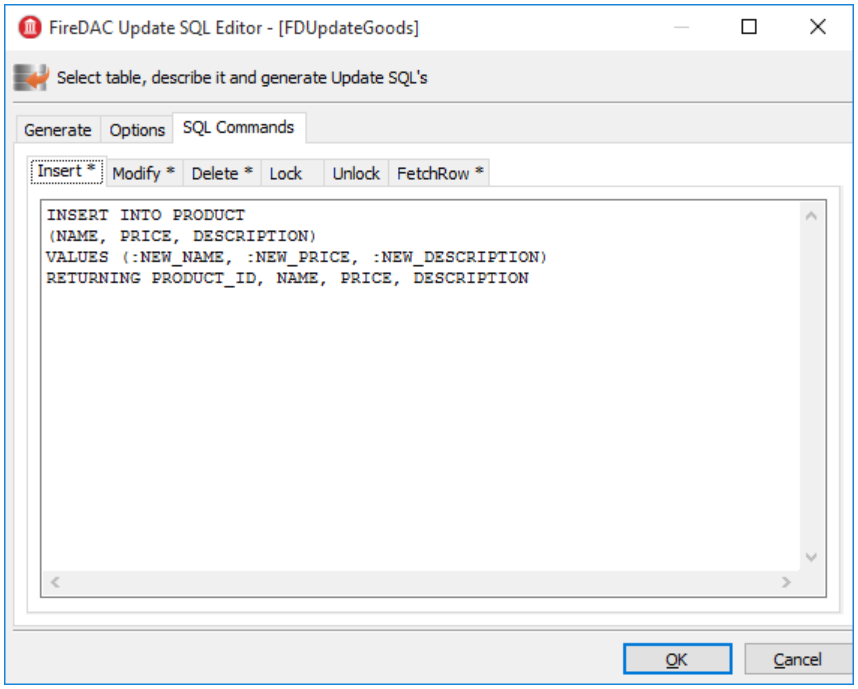

Рис.3.3. Ручная правка запроса

На закладке Options находятся некоторые свойства, которые могут повлиять на генерацию запросов. Эти свойства не относятся к самому компоненту TFDUpdateSQL, а являются ссылками на свойства UpdateOptions набора данных, у которого указан текущий TFDUpdateSQL в свойстве UpdateObject (таблица 3.3).

Таблица 3.3. Свойства закладки Options

Параметр	Назначение
Connection	Связь с компонентом FDConnection.
DeleteSQL	SQL запрос для удаления записи.
FetchRowSQL	SQL запрос для возврата одной текущей (обновлённой, вставленной) записи.
InsertSQL	SQL запрос для вставки записи.
LockSQL	SQL запрос для блокировки одной текущей записи. (FOR UPDATE WITH LOCK).
ModifySQL	SQL запрос для модификации записи.
UnlockSQL	SQL запрос для разблокировки текущей записи. В Firebird не применяется.

У компонента TFDUpdateSQL нет свойства Transaction. Это потому, что компонент не выполняет модифицирующие запросы непосредственно, а лишь заменяет автоматически сгенерированные запросы в наборе данных, который является предком TFDRdbmsDataSet.

В результате в браузере базы данных (рисунок 3.4) можно увидеть создаваемые таблицы:

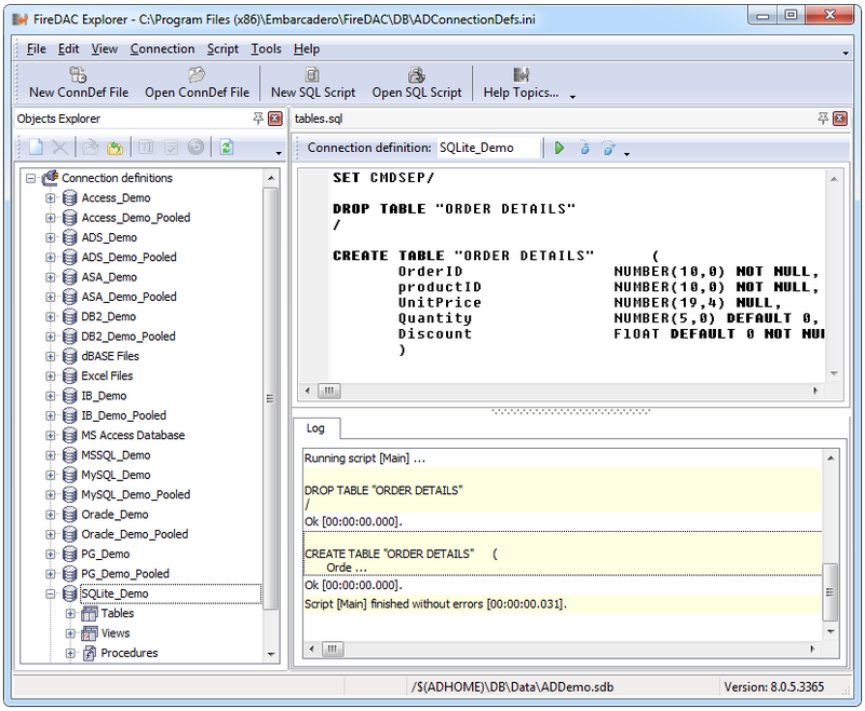

Рис.3.4. Браузер базы данных

Также в данном окне можно просмотреть созданные процедуры и представления.

3.4 Создание справочников

В разрабатываемом приложении мы создадим два справочника: справочник ставок и справочник клиентов (рисунок 3.5). Каждый из справочников представляет собой форму с сеткой TDBGrid, источником данных TDataSource, набором данных TFDQuery, пишущей транзакции TFDTransaction.

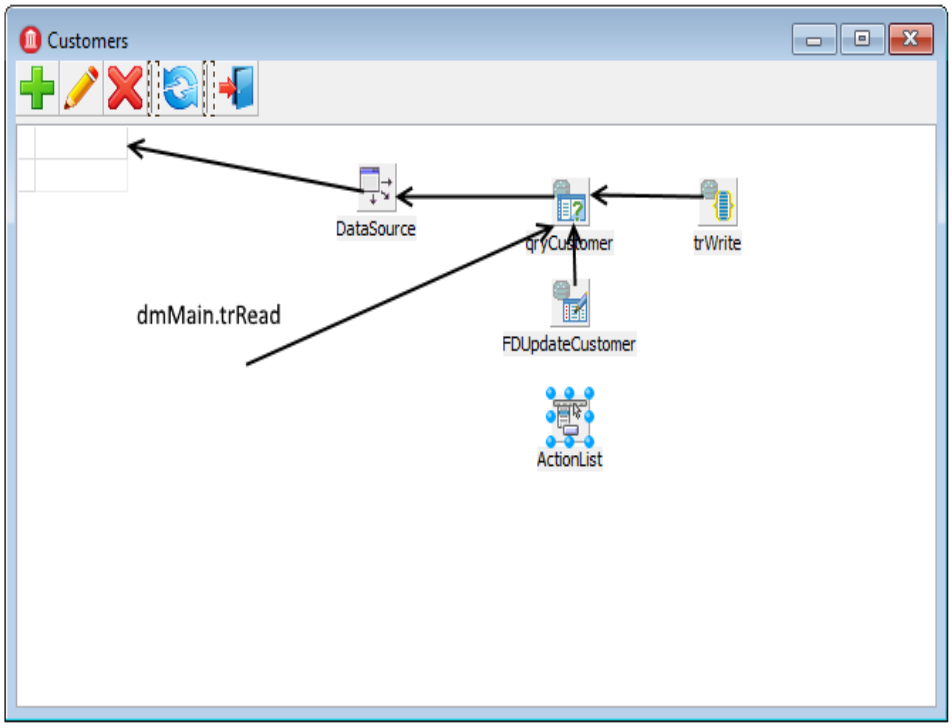

Рис.3.5. Создание справочника клиентов

Рассмотрим создание справочников на примере справочника клиентов. Разместим компонент TFDQuery на форме с именем qryCustomers. Этот набор данных будет указан в свойстве DataSet источника данных DataSource. В свойстве Transaction укажем ReadOnly транзакцию trRead, которая была создана в главном датамодуле проекта. В свойстве UpdateTransaction указываем транзакцию trWrite, в свойстве Connection — соединение расположенное в главном датамодуле. В свойстве SQL напишем следующий запрос:

Листинг 5. Запрос вывода всех полей таблицы

```
SELECT
  customer_id,
  name,
  address,
  zipcode,
  phone
FROM
  customer
ORDER BY name
```

Пишущая транзакция trWrite должна быть максимально короткой, и иметь режим изолированности SNAPSHOT. Мы не будем полагаться на автоматический старт и завершение транзакции, а будем стартовать и завершать транзакцию явно. Таким образом, наша транзакция должна иметь следующие свойства:

- Options.AutoStart = False
- Options.AutoCommit = False
- Options.AutoStop = False
- Options.DisconnectAction = xdRollback
- Options.Isolations = xiSnapshot
- Options.ReadOnly = False

Для возможности редактирования набора данных необходимо заполнить свойства InsertSQL, ModifySQL, DeleteSQL и FetchRowSQL. Эти свойства могут быть сгенерированы мастером, но после этого может потребоваться некоторая правка. Например вы можете дописать предложение RETURNING, удалить модификацию

некоторых столбцов, или же вовсе заменить автоматически сгенерированный запрос на вызов хранимой процедуры.

Листинг 6. Запрос вывода процедуры InsertSQL

```
INSERT INTO customer (customer_id,
            name,
            address,
            zipcode,
            phone)
VALUES (:new_customer_id,
    :new_name,
    :new_address,
    :new_zipcode,
    :new_phone)
```

Листинг 7. Запрос вывода процедуры ModifySQL

```
UPDATE customer
SET name = :new_name,
  address = :new_address,
  zipcode = :new_zipcode,
  phone = :new_phone
WHERE (customer_id = :old_customer_id)
```

Листинг 8. Запрос вывода процедуры DeleteSQL

```
DELETE FROM customer
WHERE (customer_id = :old_customer_id)
```

Листинг 9. Запрос вывода процедуры FetchRowSQL

```
SELECT
  customer_id,
  name,
  address,
  zipcode,
  phone
FROM
  customer
WHERE customer_id = :old_customer_id
```

В этом справочнике будем получать значение генератора перед вставкой записи в таблицу. Для этого необходимо установить

значение свойств компонента TFDQuery в следующие значения UpdateOptions.GeneratorName = GEN_CUSTOMER_ID и UpdateOptions.AutoIncFields = CUSTOMER_ID. Есть другой способ, когда значение генератора (автоинкрементного поля) возвращается после выполнения INSERT запроса с помощью предложения RETURNING.

Для добавления новой записи и редактирования существующей принято использовать модальные формы, по закрытию которых с результатом mrOK изменения вносятся в базу данных. Обычно для создания таких форм используются DBAware компоненты, которые позволяют отображать значения некоторого поля в текущей записи и немедленно вносить изменения в текущую запись набора данных в режимах Insert/Edit, т.е. до Post. Но перевести набор данных в режим Insert/Edit можно только стартовав пишущую транзакцию. Таким образом, если кто-то откроет форму для внесения новой записи и уйдёт на обед, не закрыв эту форму, у нас будет висеть активная транзакция до тех пор, пока сотрудник не вернётся с обеда и не закроет форму. Это в свою очередь приведёт к тому, что активная транзакция будет удерживать сборку мусора, что позже приведёт к снижению производительности. Эту проблему можно решить одним из двух способов:

- Использовать режим CachedUpdates, что позволяет держать транзакцию активной только на очень короткий промежуток времени, а именно на время внесения изменений.

- Отказаться от применения DBAware компонентов. Однако этот путь потребует от вас дополнительных усилий.

Для справочников гораздо удобнее использовать первый способ. Рассмотрим код редактирования записи клиента (листинг 10):

Листинг 10. Код редактирования записи клиента

```
procedure TCustomerForm.actEditRecordExecute(Sender: TObject);
var
  xEditor: TEditCustomerForm;
begin
  xEditor := TEditCustomerForm.Create(Self);
  try
    xEditor.OnClose := CustomerEditorClose;
    xEditorForm.DataSource := DataSource;
    xEditor.Caption := 'Edit customer';
    qryCustomer.CachedUpdates := True;
    qryCustomer.Edit;
    xEditor.ShowModal;
  finally
    xEditor.Free;
  end;
end;
```

Перед переводом набора данных в режим редактирования мы устанавливаем ему режим CachedUpdates, а вся логика обработки редактирования происходит в модальной форме (листинг 11).

Листинг 11. Обработчик события CustomerEditorClose

```
procedure TCustomerForm.CustomerEditorClose (Sender: TObject;
  var Action: TCloseAction);
begin
  if TForm(Sender).ModalResult <> mrOK then
  begin
    // отменяем все изменения
    qryCustomer.Cancel;
    qryCustomer.CancelUpdates;
    // возвращаем набор данных в обычный режим обновления
    qryCustomer.CachedUpdates := False;
    // и позволяем закрыть форму
    Action := caFree;
    Exit;
  end;

  try
    // подтверждаем изменения на уровне набора данных
    qryCustomer.Post;
    // стартуем транзакцию
    trWrite.StartTransaction;
    // если в наборе данных есть изменения
    if (qryCustomer.ApplyUpdates = 0) then
    begin
      // записываем их в БД
      qryCustomer.CommitUpdates;
      // и подтверждаем транзакцию
      trWrite.Commit;
    end
    else begin
      raise Exception.Create(qryCustomer.RowError.Message);
    end;
    qryCustomer.CachedUpdates := False;
    Action := caFree;
  except
    on E: Exception do
    begin
      // откатываем транзакцию
      if trWrite.Active then
        trWrite.Rollback;
      Application.ShowException(E);
      // Не закрываем окно, даём возможность исправить ошибку
      Action := caNone;
    end;
  end;
end;
```

Из кода видно, что до тех пор, пока кнопка OK не нажата, пишущая транзакция не стартует вовсе. Таким образом, пишущая транзакция активна только на время переноса данных из буфера набора данных в базу данных. Поскольку мы копим в буфере не более одной

записи, транзакция будет активна очень короткое время, что и требовалось.

Справочник ставок делается аналогично справочнику клиентов. Однако в нём мы продемонстрируем другой способ получения автоинкрементных значений.

Основной запрос будет выглядеть следующим образом (листинг 12):

Листинг 12. Запрос ставок

```
SELECT
   product_id,
   name,
   price,
   description
FROM product
ORDER BY name
```

Свойство компонента TFDUpdateSQL.InsertSQL будет содержать запрос, описанный в листинге 13:

Листинг 13. Запрос ставок

```
INSERT INTO PRODUCT
(NAME, PRICE, DESCRIPTION)
VALUES (:NEW_NAME, :NEW_PRICE, :NEW_DESCRIPTION)
RETURNING PRODUCT_ID
```

В этом запросе появилось предложение RETURNING, которое вернёт значение поля PRODUCT_ID после изменения его в BEFORE INSERT триггере. В этом случае не имеет смысла выставлять значение свойства UpdateOptions.GeneratorName. Кроме того, полю PRODUCT_ID необходимо выставить свойства Required = False и

ReadOnly = True, поскольку значение этого свойства не вносится напрямую. В остальном всё примерно также как это организовано для справочника производителей.

3.5 Создание журнала ставок

В приложении будет один журнал «Ставки». В отличие от справочников журналы содержат довольно большое количество записей и являются часто пополняемыми.

Ставка – состоит из заголовка, где описываются общие атрибуты (номер, дата, …), и строк ставок, номинал, и так далее. Для таких документов удобно иметь два вида: в главном отображаются данные о шапке документа, а в детализирующем — список. Таким образом, на форму документа потребуется поместить два компонента TDBGrid, к каждому из которых привязать свой TDataSource, которые в свою очередь будут привязаны к своим TFDQuery. В нашем случае набор данных с шапками документы будет называться qryInvoice, а со строками документа qryInvoiceLine.

В свойстве Transaction обоих наборов данных укажем ReadOnly транзакцию trRead, которая была создана в главном датамодуле проекта. В свойстве UpdateTransaction указываем транзакцию trWrite, в свойстве Connection — соединение, расположенное в главном датамодуле (рисунок 3.6).

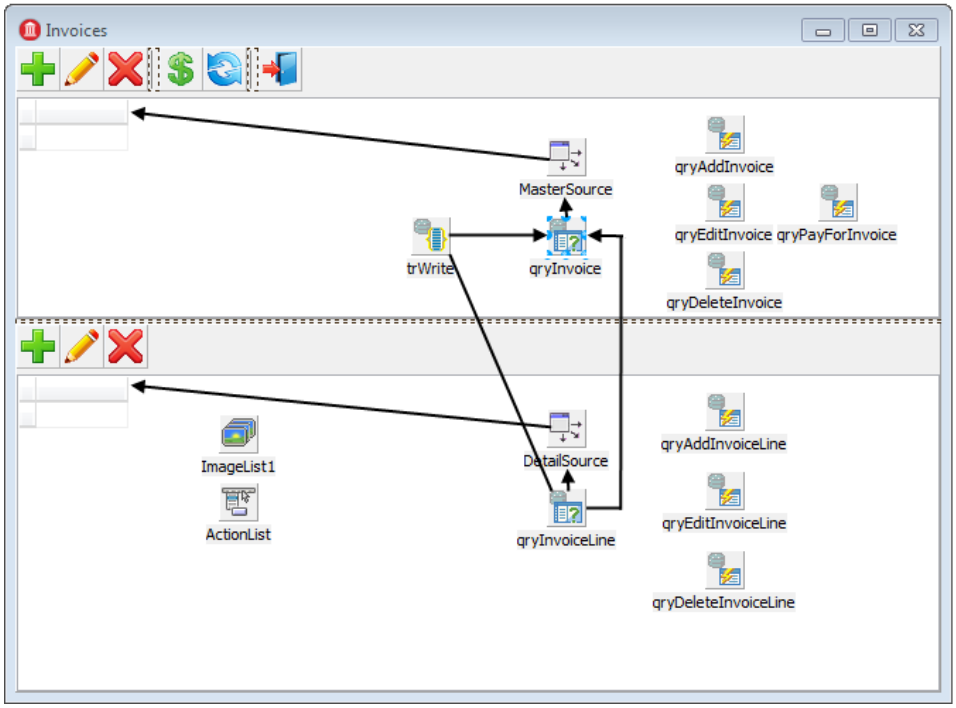

Рис.3.6. Добавление компонентов на форму

Поскольку чаще всего требуются именно последние введённые ставки, то имеет смысл сортировать их по дате в обратном порядке. С учётом вышесказанного, в свойстве SQL набора данных qryInvoice запрос будет выглядеть следующим образом (листинг 14):

Листинг 14. Сортировка ставок

```
SELECT
  invoice.invoice_id AS invoice_id,
  invoice.customer_id AS customer_id,
  customer.NAME AS customer_name,
  invoice.invoice_date AS invoice_date,
  invoice.total_sale AS total_sale,
  IIF(invoice.payed=1, 'Yes', 'No') AS payed
FROM
  invoice
  JOIN customer ON customer.customer_id = invoice.customer_id
WHERE invoice.invoice_date BETWEEN :date_begin AND :date_end
ORDER BY invoice.invoice_date DESC
```

При открытии этого набора данных необходимо будет инициализировать параметры запроса (листинг 15):

Листинг 15. Инициализация параметров запроса

```
qryInvoice.ParamByName('date_begin').AsSqlTimeStamp := dmMain.BeginDateSt;
qryInvoice.ParamByName('date_end').AsSqlTimeStamp := dmMain.EndDateSt;
qryInvoice.Open;
```

Все операции над счёт-фактурой будем производить с помощью хранимых процедур, хотя в более простых случаях это можно делать и с помощью обычных запросов INSERT/UPDATE/DELETE.

Каждую хранимую процедуру будем выполнять как отдельный запрос в компонентах TFDCommand. Этот компонент не является предком TFDRdbmsDataSet, не буферизирует данные и возвращает максимум одну строку результата, поэтому его использование несёт меньше накладных расходов для запросов, не возвращающих данные. Поскольку наши хранимые процедуры выполняют модификацию данных, то свойство Transaction компонентов TFDCommand необходимо установить транзакцию trWrite.

Для работы с шапкой счёт-фактуры предусмотрено четыре операции: добавление, редактирование, удаление и установка признака «оплачено». Как только счёт-фактура оплачена, мы запрещаем любые её модификации, как в шапке, так и в строках. Это сделано на уровне хранимых процедур. Приведём тексты запросов для вызова хранимых процедур (листинги 16-19).

Листинг 16. Инициализация хранимых процедуры qryAddInvoice.CommandText

```
EXECUTE PROCEDURE sp_add_invoice(
  NEXT VALUE FOR gen_invoice_id,
  :CUSTOMER_ID,
  :INVOICE_DATE
)
```

Листинг 17. Инициализация хранимых процедуры qryEditInvoice.CommandText

```
EXECUTE PROCEDURE sp_edit_invoice(
  :INVOICE_ID,
  :CUSTOMER_ID,
  :INVOICE_DATE
)
```

Листинг 18. Инициализация хранимых процедуры qryDeleteInvoice.CommandText

```
EXECUTE PROCEDURE sp_delete_invoice(:INVOICE_ID)
```

Листинг 19. Инициализация хранимых процедуры qryPayForInvoice.CommandText

```
EXECUTE PROCEDURE sp_pay_for_inovice(:invoice_id)
```

Поскольку хранимые процедуры вызываются не из компонента TFDUpdateSQL, то после их выполнения необходимо вызвать qryInvoice.Refresh для обновления данных в гриде. Обработчики событий аналогичны журналу заказчиков и вынесены в приложение к дипломной работе.

Для отображения всех табличных данных и работы с ними используется компонент TDBGrid (рис.3.7)

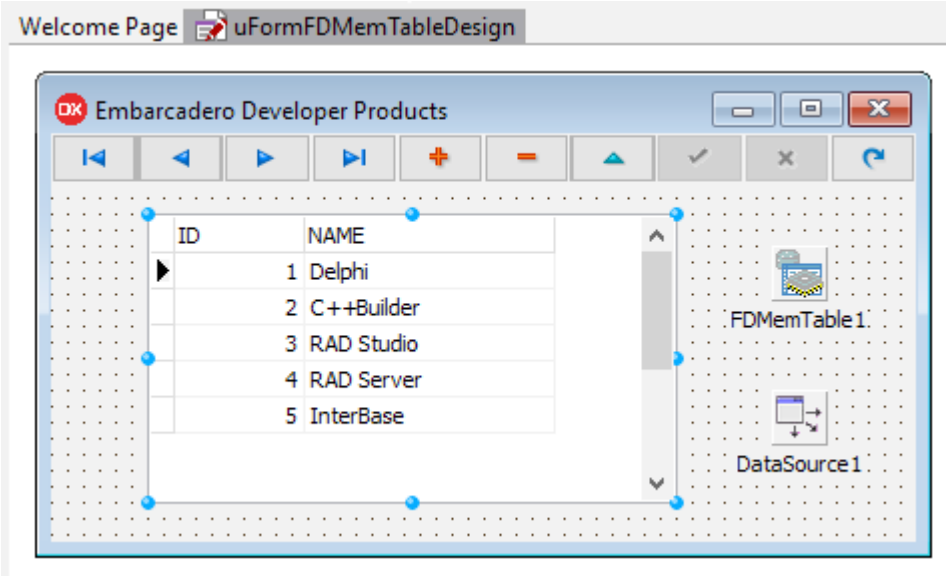

Рис.3.7. Компонент TDBGrid, отображение таблицы

Этот компонент инкапсулирует двумерную таблицу, в которой строки представляют собой записи, а столбцы — поля набора данных. Компонент TDBGrid является потомком классов TDBCustomGrid И TCustomGrid.

От класса TCustomGrid наследуются все функции отображения и управления работой двумерной структуры данных. Класс TDBCustomGrid обеспечивает визуализацию и редактирование полей из набора данных, причем TDBGrid только публикует свойства и методы класса TDBCustomGrid, не добавляя собственных.

3.6 Апробация приложения

Основной код программы может быть использован для другой платформы, в рамках работы создано приложение основной разрабатываемый интерфейс создавался для платформы Windows (рисунок 3.8):

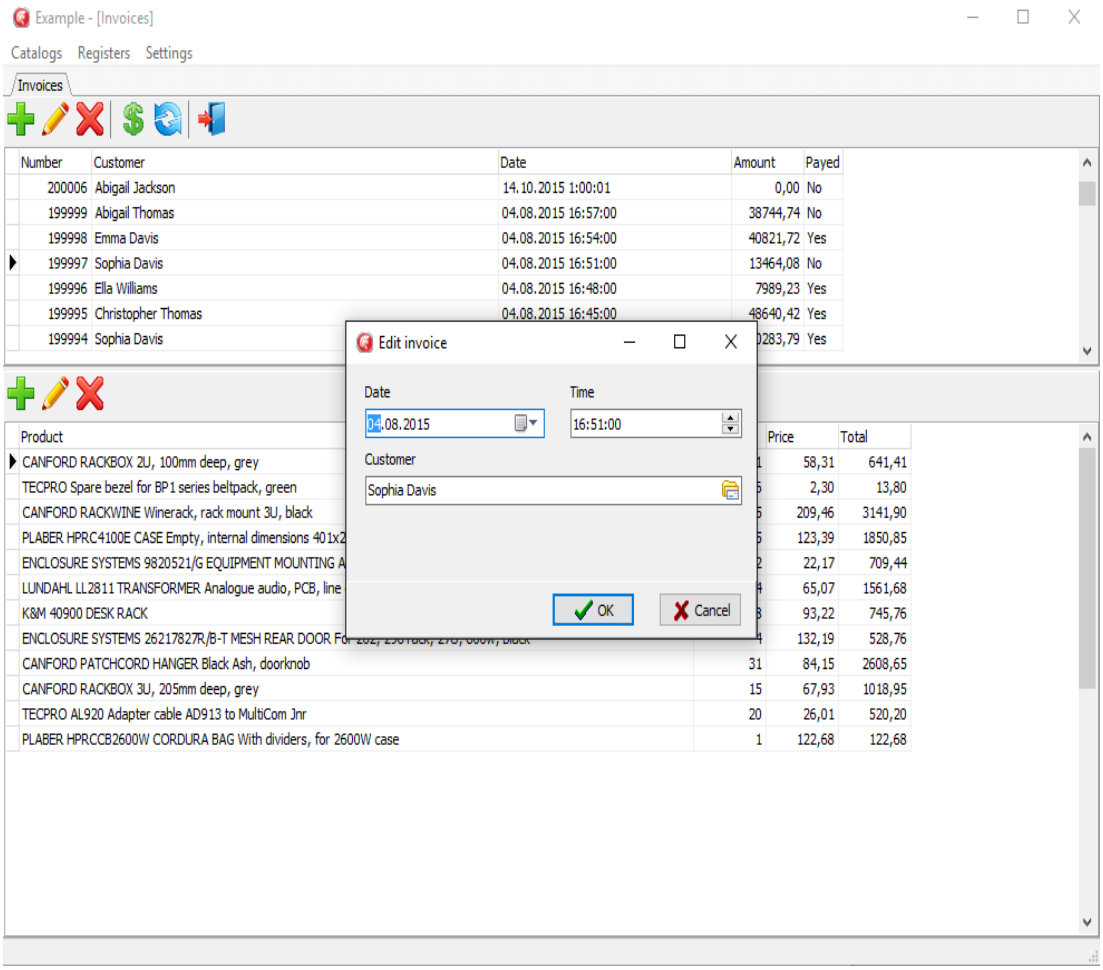

Рис.3.8. Окно приложения администрирования базы ставок на спорт

Выбор заказчиков и сортировка дат для удобства осуществляется в модальной форме (рисунок 3.9):

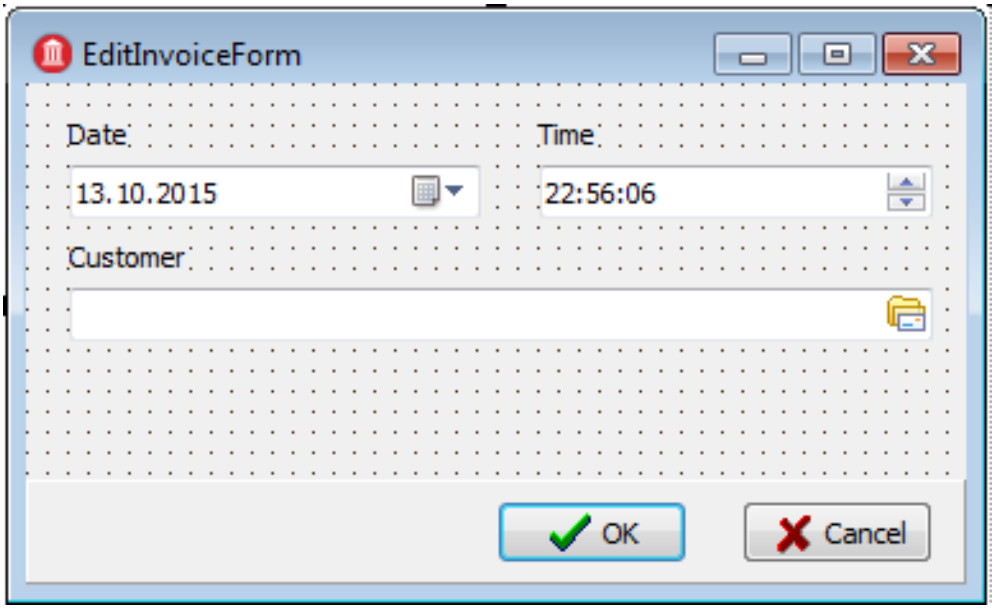

Рис.3.9. Окно выбора клиента

В качестве модальной окна для выбора клиента используем ту же форму, что была создана для ввода клиентов.

Приложение проверено на проведение стандартных сценариев поиск, отображения, добавления и удаление записей. Работа корректна. Таким образом можно сказать, что приложение готово к эксплуатации.

ЗАКЛЮЧЕНИЕ

Биржа ставок на спорт – это своего рода платформа для заключения пари между игроками на спортивные и другие события. В ней Вы можете сами предлагать пари — выбрав определённое событие, предложить свой коэффициент и сумму ставки на определенный исход (в данном случае Вы продаёте ставку, так сказать, выступаете в роли букмекера). Или же можете согласится с условиями пари, предложенными другими игроками, то есть купить ставку. Таким образом, Вы становитесь участником процесса купли-продажи на большом финансовом онлайн-рынке, где в качестве товара выступают ставки на спортивные события. А биржа ставок на спорт в свою очередь выступает в роли посредника, предоставляя игрокам сервис для участия в подобного рода торгах. За это биржи берут до 5 % от выигрыша.

В век информационных технологий все данные о ставках и участниках пари можно записать в базу и при помощи автоматизации процессов облегчить работу фирмы.

Целью данной выпускной работы - создание кросс-платформенного приложения для администрирования БД спортивных ставок с использованием технологии FireDAC.

В процессе разработки изучен инструментарий, разработана архитектура и создано приложение, отвечающее заявленным требованиям.

Исходя из поставленной цели, в дипломной работе рассмотрены современные технологические и технические платформы

для реализации подобных приложений, приведены поставленные к системе требования.

Решены такие задачи выпускной работы как:

- анализ существующих видов ставок;
- анализ и расчеты исходов ставок;
- обзор технологий для работы с базами данных;
- постановка задачи для создаваемого приложения;
- настройка соединения с базой данных с использованием технологии FireDAC;
- разработка кросс-платформенного приложения для администрирования базы данных.

Программа успешно протестирована и внедрена в работу компании.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Буч Гради. Объектно-ориентированный анализ и проектирование. - Бином, Невский диалект, 1998. - 560 с.

2. Вирт Н. Алгоритмы + структуры данных = программы, - 1976.

3. Вирт Н. Алгоритмы и структуры данных. - Спб.: Невский диалект, 2001.

4. Гарсиа- М.Г. и др. Системы баз данных = Database Systems: Полный курс / Г. Гарсиа-Молина, Г. Ульман, Д. Уидом; [Пер с англ. и ред. А. С. Варакина].- М.: Вильямс, 2003. - 1083 с.

5. Глушаков С.В., Ломотько Д.В. Базы данных: Учебный курс. - Харьков: Фолио; Ростов: Феникс; Киев: Абрис, 2000. - 504 с.

6. Горчаков Л.В., Стась А.Н. Основы искусственного интеллекта. Учебное пособие. - в печати.

7. Гражданский кодекс РФ. Омега-Л, 2003. 416 с.

8. Гришин М.П. Информатика: методическое пособие по выполнению практических работ в компьютерном классе (Access). Моск. гос. индустриальный университет. Институт дистанционного образования.

9. Дейт К. Введение в системы баз данных. - М.: Наука, 1980. - 464 с.

10. Диго С.М. Проектирование и использование баз данных. - М.: Финансы и статистика, 1995. - 208 с.

11. Дюбуа П. MySQL: Полное и исчерпывающее руководство по применению и администрированию баз данных MySQL 4, а также программированию приложений. Пер. с англ. и ред. Н. В. Воронина. - М.: Вильямс, 2004. - 1051с.

12. Жуков А. Изучаем Delphi. - СПб.: Питер, 2004. - 346 с.

13. Зандстра М. Освой самостоятельно PHP4 за 24 часа. - М.: Вильямс, 2004,- 384 с.

14. Карпова Т.С.. Базы данных: Модели, разработка, реализация: Учебное пособие. - СПб.: Питер, 2002. - 303 с.

15. Кватрани Терри. Rational Rose 2000 и UML. Визуальное моделирование.- ДМК, 2001.

16. Кириллов В.В. Основы проектирования реляционных баз данных. Санкт- Петербургский Государственный институт точной механики и оптики (технический университет) Кафедра вычислительной техники. Электронное учебное пособие. Copyright © CIT. http://es.tspu.edu.ru/parfenov/bd.chm.

17. Климов Ю. С., Касаткин А. И., Мороз С. М. Программирование в среде Turbo Pascal 6.0. - Минск: Высшая школа. - 1992.

18. Коннолли Т., Бегг К., Страчан А. Базы данных = Database Systems: Проектирование, реализация и сопровождение: Теория и практика. - М.: Вильямс, 2001.

19. Кренке Д. Теория и практика построения баз данных = Database processing / Пер. с англ. А. Вахитова. - СПб.: Питер, 2003. - 799 с.

20. Кристофидес Н. Теория графов. Алгоритмический подход. М.: Мир, 1978.

21. Культин Н.Б. Программирование в Turbo Pascal 7.0 и Delphi. - СПб.: BHV. - С-Петербург, 1998. - 240 с.

22. Марков А.С., Лисовский К.Ю. Базы данных. Введение в теорию и методологию: Учебник. - М.: Финансы и статистика, 2004. - 512 с.

23. Новиков Ф.А. Дискретная математика для программистов. М - 2002.

24. Петров В.Н. Информационные системы: Учебник для вузов. - СПб.: Питер, 2002. - 687 с.

25. Райордан Р. Основы реляционных баз данных = Designing Relational Database Systems:Пер. с англ. - М.: Русская Редакция, 2001. - 352 с.

26. Роб П., Коронел К. Системы баз данных: проектирование, реализация и управление = Database systems /

Пер. с англ. А. Никифорова. - СПб.: БХВ-Пе- тербург, 2004. - 1024 с.

27. Страуструп Бьерн. Язык программирования C++. СПб.: Бином, Невский диалект, 1999. - 991 с.

28. Тиори Т., Фрай Дж. Проектирование структур баз данных (в двух книгах). - М.: Мир, 1985.

29. Ульман Дж. Основы систем баз данных. - М.: Финансы и статистика, 1983. - 334 с.

30. Харрингон Дж. Л. Проектирование реляционных баз данных = Relational Database Design. Clearly Explained: Просто и доступно: Учебное пособие. Науч. ред. А. Головко. - М.: Лори, 2000. - 230 с.

31. Хомоненко А.Д., Цыганков В.М., Мальцев М. Г. Базы данных: Учебник для вузов. - СПб.: КОРОНА принт, 2003. - 665 с.

32. Хопкрофт Дж, Мотвание Р., Ульман Дж. Введение в теорию автоматов, языков и вычислений. - М.: Вильямс, 2003.

33. Цикритзис Д., Лоховски Ф. Модели данных. - М.:Финансы и статистика, 1985. - 344 с.

34. Чен П. Модель «сущность-связь» - шаг к единому представлению о данных. // СУБД, 1995. - № 3. - с. 137-158.

ПРИЛОЖЕНИЕ А. ЛИСТИНГ ПРОГРАММЫ

Удаление ставки

```
if MessageDlg('Вы действительно хотите удалить ставку?', mtConfirmation,
  [mbYes, mbNo], 0) = mrYes then
begin
  // Стартуем транзакцию
  trWrite.StartTransaction;
  try
    qryDeleteInvoice.ParamByName('INVOICE_ID').AsInteger :=
      qryInvoice.FieldByName('INVOICE_ID').AsInteger;
    // выполнение хранимой процедуры
    qryDeleteInvoice.Execute;
    // подтверждение транзакции
    trWrite.Commit;
    // обновление данных в гриде
    qryInvoice.Refresh;
  except
    on E: Exception do
    begin
      if trWrite.Active then
        trWrite.Rollback;
      Application.ShowException(E);
    end;
  end;
end;
```

Обработчик TButtonedEdit

```
procedure TEditInvoiceForm.edtCustomerRightButtonClick(Sender: TObject);
var
  xSelectForm: TCustomerForm;
begin
  xSelectForm := TCustomerForm.Create(Self);
  try
    xSelectForm.Visible := False;
    if xSelectForm.ShowModal = mrOK then
    begin
      FCustomerId := xSelectForm.qryCustomer.FieldByName('CUSTOMER_ID')
        .AsInteger;
      edtCustomer.Text := xSelectForm.qryCustomer.FieldByName('NAME').AsString;
    end;
  finally
    xSelectForm.Free;
  end;
end;
```

Инициализация клиента

```
procedure TInvoiceForm.actEditInvoiceExecute(Sender: TObject);
```

```
var
  xEditorForm: TEditInvoiceForm;
begin
  xEditorForm:= TEditInvoiceForm.Create(Self);
  try
    xEditorForm.OnClose := EditInvoiceEditorClose;
    xEditor.Caption := 'Редактирование счёт-фактуры';

    xEditorForm.InvoiceId := qryInvoice.FieldByName('INVOICE_ID').AsInteger;
    xEditorForm.SetCustomer(qryInvoice.FieldByName('CUSTOMER_ID').AsInteger,
                    qryInvoice.FieldByName('CUSTOMER_NAME').AsString);
    xEditorForm.InvoiceDate := qryInvoice.FieldByName('INVOICE_DATE').AsDateTime;

    xEditorForm.ShowModal;
  finally
    xEditorForm.Free;
  end;
end;


procedure TEditInvoiceForm.SetCustomer(ACustomerId: Integer;
  const ACustomerName: string);
begin
  FCustomerId := ACustomerId;
  edtCustomer.Text := ACustomerName;
end;
```

Обработка добавления новой ставки

```
procedure TInvoiceForm.EditInvoiceEditorClose(Sender: TObject; var Action: TCloseAction);
var
  xEditorForm: TEditInvoiceForm;
begin
  xEditorForm := TEditInvoiceForm(Sender);

  // если форма закрыта не по нажатию кнопки OK,
  // то вообще ничего не делаем. Транзакция не стартует.
  if xEditorForm.ModalResult <> mrOK then
  begin
    Action := caFree;
    Exit;
  end;

  // Выполняем всё в короткой транзакции
  trWrite.StartTransaction;
  try
    qryEditInvoice.ParamByName('INVOICE_ID').AsInteger := xEditorForm.InvoiceId;
    qryEditInvoice.ParamByName('CUSTOMER_ID').AsInteger :=
      xEditorForm.CustomerId;
    qryEditInvoice.ParamByName('INVOICE_DATE').AsSqlTimeStamp :=
      DateTimeToSQLTimeStamp(xEditorForm.InvoiceDate);

    qryEditInvoice.Execute();

    trWrite.Commit;
    qryInvoice.Refresh;
```

```
    Action := caFree;
  except
    on E: Exception do
    begin
      if trWrite.Active then
        trWrite.Rollback;
      Application.ShowException(E);
      // Не закрываем окно, даём возможность исправить ошибку
      Action := caNone;
    end;
  end;
end;
```

Запрос позиций ставок

```
SELECT
   invoice_line.invoice_line_id AS invoice_line_id,
   invoice_line.invoice_id AS invoice_id,
   invoice_line.product_id AS product_id,
   product.name AS productname,
   invoice_line.quantity AS quantity,
   invoice_line.sale_price AS sale_price,
   invoice_line.quantity * invoice_line.sale_price AS total
FROM
   invoice_line
JOIN product ON product.product_id = invoice_line.product_id
WHERE invoice_line.invoice_id = :invoice_id
```

Процедура qryAddInvoiceLine

```
EXECUTE PROCEDURE sp_add_invoice_line(
  :invoice_id,
  :product_id,
  :quantity
)
```

Процедура qryEditInvoiceLine

```
EXECUTE PROCEDURE sp_edit_invoice_line(
  :invoice_line_id,
  :quantity
)
```

Процедура qryDeleteInvoiceLine

```
EXECUTE PROCEDURE sp_delete_invoice_line(
  :invoice_line_id
```

```
)
```

Обработчик edtProductRightButtonClick

```
procedure TEditInvoiceLineForm.edtProductRightButtonClick(Sender: TObject);
var
  xSelectForm: TGoodsForm;
begin
  // не позволяем изменять товар в режиме редактирования
  // это можно сделать только при добавлении новой позиции
  if FEditMode = emInvoiceLineEdit then
    Exit;

  xSelectForm := TGoodsForm.Create(Self);
  try
    xSelectForm.Visible := False;
    if xSelectForm.ShowModal = mrOK then
    begin
      FProductId := xSelectForm.qryGoods.FieldByName('PRODUCT_ID')
        .AsInteger;
      edtProduct.Text := xSelectForm.qryGoods.FieldByName('NAME').AsString;
      // в данном случае мы копируем также цену по прайсу
      edtPrice.Text := xSelectForm.qryGoods.FieldByName('PRICE').AsString;
    end;
  finally
    xSelectForm.Free;
  end;
end;
```

Обработчик actEditInvoiceLineExecute

```
procedure TInvoiceForm.actEditInvoiceLineExecute(Sender: TObject);
var
  xEditorForm: TEditInvoiceLineForm;
begin
  xEditorForm:= TEditInvoiceLineForm.Create(Self);
  try
    xEditorForm.OnClose := EditInvoiceLineEditorClose;
    xEditorForm.EditMode := emInvoiceLineEdit;
    xEditorForm.Caption := 'Редактирование позиции';

    xEditorForm.InvoiceLineId := qryInvoiceLine.FieldByName('INVOICE_LINE_ID').AsInteger;
    xEditorForm.SetProduct(qryInvoiceLine.FieldByName('PRODUCT_ID').AsInteger,
                  qryInvoiceLine.FieldByName('PRODUCTNAME').AsString,
                  qryInvoiceLine.FieldByName('SALE_PRICE').AsCurrency);
    xEditorForm.Quantity := qryInvoiceLine.FieldByName('QUANTITY').AsInteger;

    xEditorForm.ShowModal;
  finally
    xEditorForm.Free;
  end;
```

```
end;
```

Обработчик EditInvoiceLineEditorClose

```
procedure TInvoiceForm.EditInvoiceLineEditorClose(Sender: TObject;
  var Action: TCloseAction);
var
  xCustomerId: Integer;
  xEditorForm: TEditInvoiceLineForm;
begin
  xEditorForm := TEditInvoiceLineForm(Sender);
  // если форма закрыта не по нажатию кнопки OK,
  // то вообще ничего не делаем. Транзакция не стартует.
  if xEditorForm.ModalResult <> mrOK then
  begin
    Action := caFree;
    Exit;
  end;

  // Всё делаем в короткой транзакции
  trWrite.StartTransaction;
  try
    qryEditInvoiceLine.ParamByName('INVOICE_LINE_ID').AsInteger :=
      xEditorForm.InvoiceLineId;
    qryEditInvoiceLine.ParamByName('QUANTITY').AsInteger :=
      xEditorForm.Quantity;

    qryEditInvoiceLine.Execute();

    trWrite.Commit;
    qryInvoice.Refresh;
    qryInvoiceLine.Refresh;

    Action := caFree;
  except
    on E: Exception do
    begin
      if trWrite.Active then
        trWrite.Rollback;
      Application.ShowException(E);
      // Не закрываем окно редактирования. Позволяем пользователю исправить ошибку
      Action := caNone;
    end;
  end;
end;
```

ПРИЛОЖЕНИЕ Б. БАЗА ДАННЫХ

```
CREATE TABLE `bet` (
 `id` integer NOT NULL AUTO_INCREMENT,
 `event_id` integer NOT NULL,
 `bet_type_id` integer NOT NULL,
 `koef` float NOT NULL,
 `time` datetime,
 `enabled` bool,
 PRIMARY KEY (`id`)
);

INSERT INTO `bet` (`id`,`event_id`,`bet_type_id`,`koef`,`time`,`enabled`) VALUES (1,1,13,1.4,'2017-05-08 18:12:31',1);
INSERT INTO `bet` (`id`,`event_id`,`bet_type_id`,`koef`,`time`,`enabled`) VALUES (2,1,14,1.6,'2017-05-08 18:12:31',1);
CREATE TABLE `bet_type` (
 `id` integer NOT NULL AUTO_INCREMENT,
 `title` varchar(256) NOT NULL,
 PRIMARY KEY (`id`)
);

INSERT INTO `bet_type` (`id`,`title`) VALUES (1,'Тотал (больше 0.5)');
INSERT INTO `bet_type` (`id`,`title`) VALUES (2,'Тотал (больше 1)');
INSERT INTO `bet_type` (`id`,`title`) VALUES (3,'Тотал (больше 1.5)');
INSERT INTO `bet_type` (`id`,`title`) VALUES (4,'Тотал (больше 2)');
INSERT INTO `bet_type` (`id`,`title`) VALUES (5,'Тотал (больше 2.5)');
INSERT INTO `bet_type` (`id`,`title`) VALUES (6,'Тотал (больше 3)');
INSERT INTO `bet_type` (`id`,`title`) VALUES (7,'Тотал (меньше 0.5)');
INSERT INTO `bet_type` (`id`,`title`) VALUES (8,'Тотал (меньше 1)');
INSERT INTO `bet_type` (`id`,`title`) VALUES (9,'Тотал (меньше 1.5)');
INSERT INTO `bet_type` (`id`,`title`) VALUES (10,'Тотал (меньше 2)');
INSERT INTO `bet_type` (`id`,`title`) VALUES (11,'Тотал (меньше 2.5)');
INSERT INTO `bet_type` (`id`,`title`) VALUES (12,'Тотал (меньше 3)');
```

```
INSERT INTO `bet_type` (`id`,`title`) VALUES (13,'Победа первой');
INSERT INTO `bet_type` (`id`,`title`) VALUES (14,'Победа второй');
INSERT INTO `bet_type` (`id`,`title`) VALUES (15,'Ничья');
INSERT INTO `bet_type` (`id`,`title`) VALUES (16,'1X (победа первой или ничья)');
INSERT INTO `bet_type` (`id`,`title`) VALUES (17,'2X (победа второй или ничья)');
INSERT INTO `bet_type` (`id`,`title`) VALUES (18,'Победа первой с форой 0');
INSERT INTO `bet_type` (`id`,`title`) VALUES (19,'Победа первой с форой 0.,5');
INSERT INTO `bet_type` (`id`,`title`) VALUES (20,'Победа первой с форой 1');
INSERT INTO `bet_type` (`id`,`title`) VALUES (21,'Победа первой с форой 1.5');
INSERT INTO `bet_type` (`id`,`title`) VALUES (22,'Победа второй с форой 0');
INSERT INTO `bet_type` (`id`,`title`) VALUES (23,'Победа второй с форой 0.,5');
INSERT INTO `bet_type` (`id`,`title`) VALUES (24,'Победа второй с форой 1');
INSERT INTO `bet_type` (`id`,`title`) VALUES (25,'Победа второй с форой 1.5');
INSERT INTO `bet_type` (`id`,`title`) VALUES (26,'Победитель ЧМ 2017 ');
INSERT INTO `bet_type` (`id`,`title`) VALUES (27,'Победитель - Джиро д’Италия - Победитель финальной классификации ');
INSERT INTO `bet_type` (`id`,`title`) VALUES (28,'Победитель - William Hill World Darts Championship 2018 - Победитель турнира ');
DROP TABLE IF EXISTS `current_bet`;
CREATE TABLE `current_bet` (
  `id` integer NOT NULL AUTO_INCREMENT,
  `bet_id` integer NOT NULL,
  `user_id` integer NOT NULL,
  `sum` float NOT NULL,
  `time` datetime,
  `enabled` bool,
  PRIMARY KEY (`id`)
);
```

```
INSERT INTO `current_bet` (`id`,`bet_id`,`user_id`,`sum`,`time`,`enabled`) VALUES (1,3,1,12,'2017-05-08 19:16:36',1);
CREATE TABLE `event` (
 `id` integer NOT NULL AUTO_INCREMENT,
 `title` varchar(256) NOT NULL,
 `time_start` datetime NOT NULL,
 `time_end` datetime NOT NULL,
 `sport_type_id` integer NOT NULL,
 PRIMARY KEY (`id`)
);

INSERT INTO `event` (`id`,`title`,`time_start`,`time_end`,`sport_type_id`) VALUES (1,'Анжи v Локомотив','2017-05-08 21:45:00','2017-05-08 23:00:00',1);
INSERT INTO `event` (`id`,`title`,`time_start`,`time_end`,`sport_type_id`) VALUES (2,'Славутич Ч. v Нефтяник-Укрнефть','2017-05-08 21:45:00','2017-05-08 23:00:00',1);
INSERT INTO `sport_type` (`id`,`title`,`time`,`enabled`) VALUES (8,'Дартс','2017-05-08 19:07:47',1);
CREATE TABLE `user` (
 `id` integer NOT NULL,
 `firstname` varchar(256) NOT NULL,
 `lastname` varchar(256) NOT NULL,
 `email` varchar(256) NOT NULL,
 `sum` float NOT NULL,
 PRIMARY KEY (`id`)
);

INSERT INTO `user` (`id`,`firstname`,`lastname`,`email`,`sum`) VALUES (1,'Дмитрий','Иванов','ivanov@mail.ru',120);
INSERT INTO `user` (`id`,`firstname`,`lastname`,`email`,`sum`) VALUES (2,'Василий','Терещенко','ter@gmail.com',11);
INSERT INTO `user` (`id`,`firstname`,`lastname`,`email`,`sum`) VALUES (3,'Николай','Валуев','valuev@list.ru',1050);
```