Министерство образования и науки Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»
Институт естественных и точных наук
Факультет математики, механики и компьютерных технологий
Кафедра прикладной математики и программирования

РАБОТА ПРОВЕРЕНА
Рецензент,_____
_____
_____
«____»_____2018г.

ДОПУСТИТЬ К ЗАЩИТЕ
Заведующий кафедрой ПМиП,
д.ф.-м.н.
_____А.А. Замышляева
«____»_____2018 г.

Разработка программного комплекса для проведения практических занятий по теме «Реляционная алгебра» в курсе «Базы данных»

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА К ВЫПУСКНОЙ
КВАЛИФИКАЦИОННОЙ РАБОТЕ МАГИСТРА
ЮУрГУ–01.04.02.2018.181.ПЗ ВКР

Руководитель работы,
доцент кафедры ПМиП, к. т. н.
_____/М.Ю. Катаргин
«____»_____2018 г.

Автор работы
студент группы ЕТ-222
_____/ А.В. Горбунова
«____»_____2018 г.

Нормоконтролер,
доцент кафедры ПМиП, к.э.н.
_____/Д.А. Дрозин
«____»_____2018 г.

Челябинск 2018

АННОТАЦИЯ

Горбунова А.В. Разработка программного комплекса для проведения практических занятий по теме «Реляционная алгебра» в курсе «Базы данных». – Челябинск: ЮУрГУ (НИУ), ЕТ-222, 68 с., 46 ил., 29 табл., библиогр. список – 51 наим., 1 прил.

Работа посвящена созданию программного комплекса, предназначенного для проведения практических занятий по теме «Реляционная алгебра».

Программный комплекс состоит из основной базы данных и ряда тестовых баз данных, а также двух приложений-клиентов:

− приложение преподавателя
− приложение студента

Приложение преподавателя позволяет:

− вводить и редактировать информацию о контингенте студента;
− вносить в БД задания для студентов и их эталонные решения;
− просматривать студенческие решения;
− давать рекомендации студентам по решению задач;
− получать вопросы студентов и отвечать на них;
− проверять студенческие решения, анализировать правильность решения и выдавать сообщения об ошибках.

Приложение студента позволяет:

− отображать предложенные задачи;
− вводить и редактировать решения;
− выполнять проверку решения на правильность;
− показывать сообщения об ошибках;
− сохранять решения в БД.

Основная база данных содержит необходимую информацию о контингенте студентов, сведения о тестовых базах данных, формулировки задач, студенческие решения задач, эталонные решения. Студент и преподаватель имеют возможность обмениваться сообщениями по поводу каждой задачи.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Целью данной работы является разработка программного комплекса, который предназначен для проведения практических занятий по теме «Реляционная алгебра». Для достижения поставленной цели необходимо решить следующие *задачи*:

− выполнить анализ требований к программному обеспечению;

− выбрать платформу, средства и инструменты для создания программного обеспечения;

− спроектировать архитектуру программного комплекса;

− разработать алгоритмы работы программы.

− разработать интерфейс приложений;

Актуальность темы.

Реляционная алгебра − это один из разделов курса базы данных. Изучение курса невозможно без понимания, что такое реляционная алгебра, какие операнды и операции она использует. Большинство SQL-серверов преобразует тексты, написанные на языке Structured Query Language в планы выполнения запросов. План выполнения запроса представляет собой дерево, узлами которой являются алгебраические операции. Анализ этих планов позволяет оптимизировать эффективность запросов. Незнание алгебры приводит к невозможности чтения планов запросов.

1. ОСНОВНЫЕ ПОНЯТИЯ РЕЛЯЦИОННОЙ АЛГЕБРЫ

1.1 Основная терминология

Реляционная алгебра – одно из средств манипулирования отношениями [1]. Реляционная алгебра представляет собой набор таких операций над отношениями, что результат каждой из операций также является отношением. Это свойство алгебры называется замкнутостью.

Число столбцов отношения называется его степенью или арностью.

В реляционных выражениях можно использовать вложенные выражения сколь угодно сложной структуры.

Столбец отношения называется атрибутом. Множество значений, допустимых для атрибута, называют доменом. Например, для атрибута «Возраст сотрудника» в качестве домена можно принять значения из диапазона 14-100; для атрибута «Цвет светофора» – значения желтый, красный, зеленый.

Некоторые реляционные операции, в частности, операции объединения, пересечения и вычитания, требуют, чтобы отношения имели совпадающие (одинаковые) заголовки (схемы). Это означает, что совпадают количество атрибутов, названия атрибутов и тип (домен) одноимённых атрибутов. Некоторые отношения формально не являются совместимыми из-за различия в названиях атрибутов, но становятся таковыми после применения операции переименования атрибутов.

1.2 Логические выражения реляционной алгебры

Логические выражения могут принимать только два значения: «ИСТИНА» и «ЛОЖЬ». Например, значение выражения 2>3 равно «ЛОЖЬ» [2].

В логических выражениях могут быть использованы операции сравнения и логические операции.

Операции сравнения:

– «=» − сравнение на равенство
− «<» − меньше
− «>» – больше
− «>=» – больше или равно
– «<=» – меньше или равно
− «<>» – неравно

Логические операции

− «OR» – логическое ИЛИ
− «AND» – логическое И
− «NOT» – отрицание

Результат операции «OR» имеет значение «ИСТИНА», когда значение «ИСТИНА» имеет хотя бы один из операндов. Например: 2>3 «OR» 3<5 равно «ИСТИНА».

Результат операции «AND» имеет значение «ИСТИНА» только тогда, когда значение «ИСТИНА» имеют оба операнда. Например, 2<3 «AND» 3<7 имеет значение «ЛОЖЬ».

Результат операции «NOT» имеет значение обратное тому, которое имеет выражение, перед которым он стоит.

Старшинство логических операций в порядке убывания: «NOT», «AND», «OR». Если требуется нарушить порядок выполнения, предписанный старшинством, то можно использовать круглые скобки, например:

(a>2or b='привет') and x<=8

1.3 Операции реляционной алгебры

1.3.1 Объединение

Объединение отношений A и B, представляет собой множество кортежей, принадлежащих либо A, либо B, либо им обоим. Операция объединения применима только к отношениям с одинаковыми схемами.

Синтаксис: $A \cup B$

1.3.2 Пересечение

Пересечение отношений с одинаковыми схемами A и B, есть множество кортежей, принадлежащих одновременно A и B.

Синтаксис: $A \cap B$

1.3.3 Вычитание

Разностью отношений A и B, называют множество кортежей, принадлежащих A, но не принадлежащих B. Здесь также требуется, чтобы отношения имели одинаковую схему.

Синтаксис: $A - B$

1.3.4 Декартово произведение

Отношение, заголовок (A_1, A_2, …, A_n, B_1, B_2, …, B_m) которого является сцеплением заголовков отношений A(A_1, A_2,…, A_n) и B(B_1, B_2, …, B_m), а тело состоит из кортежей, являющихся всеми вариантами сцеплений кортежей отношений A и B: (a_1, a_2, …, a_n, b_1, b_2, …,b_m), таких, что (a_1, a_2, …, a_n) $\in$ A, (b_1, b_2, …, b_m) $\in$ B.

Синтаксис: $A \times B$

1.3.5 Выборка (селекция)

Отношение с тем же заголовком, что и у отношения A, и телом, состоящим из кортежей, значения атрибутов которых при подстановке в условие f дают значение ИСТИНА, где f представляет собой логическое выражение, в которое могут входить атрибуты отношения A и/или скалярные выражения.

Синтаксис: $\sigma_f(A)$

Например: выбрать записи, где улица=Пушкина. Результат приведен в таблице (Таблица 1.1).

Таблица 1.1

Результат

Улица	Дом	Квартира	Жилец
Пушкина	70	23	Иванов
Пушкина	70	54	Петров
Пушкина	70	23	Сидоров

1.3.6 Проекция

Существо этой операции состоит в удалении некоторых атрибутов из схемы отношения. Пусть A=(a,b,c,d). Отношение B=(a,b) может быть получено вычеркиванием столбцов из A, которые не входят в схему отношения B и удалением дубликатных кортежей.

Синтаксис: $\pi_{a,b}(A)$

Например, пусть имеется отношение Адреса жильцов (Улица, Дом, Квартира, Жилец). Результат приведен в таблице (Таблица 1.2).

Таблица 1.2

Адреса жильцов

Улица	Дом	Квартира	Жилец
Пушкина	70	23	Иванов
Пушкина	70	54	Петров
Пушкина	70	23	Сидоров
Цвиллинга	31	12	Бармалей
Цвиллинга	31	17	Крокодил

Проекция этого отношения на атрибуты (Улица, Дом) имеет вид, приведенный в таблице 1.3.

Таблица 1.3

Результат

Улица	Дом
Пушкина	70
Цвиллинга	31

1.3.7 Соединение

Соединение отношений A и B по логическому выражению f. Соединение есть множество кортежей в декартовом произведении A × B, удовлетворяющих логическому выражению f. Особенный интерес для нас будет представлять так называемое эквисоединение, когда выражение f представляет собой требование совпадения значений атрибутов из некоторых столбцов.

Синтаксис:

Например, пусть имеются отношения:

Расписание групп (Группа, Аудитория, Дата, Время, Предмет)

Расписание преподавателей (Преподаватель, Дата, Время, Аудитория).

Применяя выражение к таблице 1.4 получим результат приведенный в таблице 1.5.

Таблица 1.4

Расписание групп

Группа	Аудитория	Дата	Время	Предмет
ЕТ-222	511	12.09.18	10:15	Введение в теорию оптимального управления
ЕТ-222	507	12.09.18	12:30	Моделирование стохастических систем
ЕТ-223	322	15.09.18	18:00	Введение в теорию оптимального управления
ЕТ-223	613	16.09.18	19:35	Моделирование стохастических систем
ЕТ-224	611	17.09.18	12:30	Современные технологии программного моделирования

Таблица 1.5

Расписание преподавателей

Преподаватель	Дата	Время	Аудитория
Иванов	12.09.18	10:15	511
Кононов	12.09.18	12:30	507
Иванов	15.09.18	18:00	322
Кононов	16.09.18	19:35	613
Петров	17.09.18	12:30	611

Выполним соединение этих отношений по полям Дата, Время, Аудитория, то есть составим результирующие кортежи из таких пар кортежей исходных отношений, в которых значения названных полей совпадают. Результатом будет отношение, приведенное в таблице 1.6.

Таблица 1.6

Расписание

Группа	Аудитория	Дата	Время	Предмет	Преподаватель
ЕТ-222	511	12.09.18	10:15	Введение в теорию оптимального управления	Иванов
ЕТ-222	507	12.09.18	12:30	Моделирование стохастических систем	Кононов
ЕТ-223	322	15.09.18	18:00	Введение в теорию оптимального управления	Иванов
ЕТ-223	613	16.09.18	19:35	Моделирование стохастических систем	Кононов
ЕТ-224	611	17.09.18	12:30	Современные технологии программного моделирования	Петров

Логическое выражение, по которому выполнялось соединение, в данном случае имело вид:

Расписание группы.Аудитория=Расписание преподавателя.Аудитория
and Расписание группы.Дата = Расписание преподавателя.Время
and Расписание группы.Время = Расписание преподавателя.Время

1.4 Обзор существующих решений

В связи с развитием информационных технологий появилась направление в образовании, которое бы автоматизировало процесс проверки самостоятельных и лабораторных работ. Широкое распространение получила автоматическая проверка заданий по программированию [3-5]. Также разрабатываются проекты по автоматизации учебного процесса и в других дисциплинах. В том числе это касается дисциплины «Базы данных», разделом которой является реляционная алгебра.

Стоит отметить, что часто такие системы остаются в рамках ВУЗа, а в открытом доступе появляются только отчеты об их внедрении в виде публикаций в различных издательствах. В связи с этим анализ существующих решений по автоматизации проверки работ по реляционной алгебре проводился по следующим пунктам:

1) степень открытости автоматизированной системы: можно ли получить копию программного продукта и по необходимости адаптировать для своих целей; есть ли возможность изменить/расширить набор задач.

2) принцип работы системы: в случае, когда системы нет в открытом доступе или нет возможности ее изменить, возможно ли применить для своих целей методики или подходы, использованные в этих системах.

На сайте проекта Lagunita [6], принадлежащему Стенфордскому университету и содержащему онлайн-курсы этого университета, представлен мини-курс, где можно пройти тест на знание реляционной алгебры [7]. Обучающимся предлагается написать реляционные выражения для получения отношений, описанных в задаче. Каждое решение проверяется отдельно, проверка происходит в режиме онлайн. Отправленное на сервер выражение транслируется в SQL-запрос при помощи разработанного в Duke University интерпретатора RA [8]. В момент трансляции проверяется синтаксис выражения. Оттранслированный SQL-запрос исполняется на небольшой тестовой базе данных, находящейся в открытом доступе. Результат запроса сверяется с заранее подготовленным правильным результатом и на основе сравнения выдается сообщение о корректности или некорректности реляционного выражения.

В Российском государственном профессионально-педагогическом университете в 2016 году был разработан тренажер «Операции реляционной алгебры» [9], включающий в себя модуль для самостоятельной работы. Модуль состоит из 30 заданий. Разработанные алгоритмы модуля позволяют провести оценку написанного студентом реляционного выражения для выбранного задания. Оценка корректности выражения происходит в два этапа:

1) Составленное выражение сравнивается с заранее подготовленным списком верных вариантов написания выражения. Выражение считается корректным, если оно найдено в списке. Иначе производится анализ возможных ошибок на основе различий с предварительно подготовленными корректными выражениями.

2) Реляционное выражение исполняется на подготовленной тестовой базе данных. Результат работы выражения сравнивается с ожидаемым результатом, и на основе сравнения выводится сообщение о корректности или некорректности реляционного выражения.

По завершению этапов в случае ошибок пользователю отображаются специальные индикаторы с подсказками по исправлению.

В системе Support of Database Skill Testing, разработанной в Technical University of Kosice [10], для каждой задачи на сервере хранится эталонный SQL-запрос, написанный преподавателем. В момент тестирования реляционное выражение, написанное студентом, посылается на сервер и транслируется в SQL-запрос модулем компиляции. Данный модуль так же проводит синтаксический анализ реляционного выражения. Если в процессе разбора выражения были обнаружены ошибки, студент видит соответствующее сообщение. В случае отсутствия синтаксических ошибок два запроса - студента и преподавателя - исполняются на подготовленных данных. Результаты двух запросов сравниваются: если они равны, решение помечается как верное; если результирующие отношения не равны, определяется возможная ошибка и отсылается соответствующее сообщение.

Разработанный в University of Girona инструмент An Automatic Correction Tool for Relational Algebra Queries [11] предоставляет среду для выполнения и

проверки реляционных выражений. Задания и ожидаемые для них результаты подготавливаются преподавателями и сохраняются в репозитории системы. Данный инструмент позволяет группировать задания по тематикам, составлять индивидуальные наборы упражнений для студентов. Во время тестирования знаний студент получает случайный набор заданий.

Каждое задание проверяется отдельно в режиме онлайн. Инструмент анализирует решение, введенное студентом, и помечает его как верное в случае, если результат выражения совпадает с заранее сохраненным правильным ответом для соответствующего упражнения. Порядок следования кортежей не учитывается. Если решение содержит ошибки, система выдает рекомендации по их исправлению. Система распознает следующие виды ошибок:

1) неверный синтаксис при формулировке выражения;

2) конечный результат не совпадает с ожидаемым.

Для получения результата реляционного выражения разработчиками инструмента были реализованы алгоритмы, отвечающие за выполнение каждой отдельной операции реляционной алгебры. Реляционное выражение выполняется поэтапно, одна операция за один этап.

В таблице 1.7 представлена сводная информация по рассмотренным аналогам.

Таблица 1.7

Рассмотренные аналоги

ВУЗ и название системы	Степень открытости системы	Подход
Stanford University, Lagunita -Relational Algebra Exersises	Открытая часть системы (интерпретатор RA)	Сравнение результата выражения с подготовленным ответом
Российский государственный профессионально-педагогический университет -Тренажер «Операции реляционной алгебры»	Закрытая	Сравнение выражения с подготовленным набором корректных выражений; сравнение результата выражения с подготовленным ответом
Technical University of Kosice -Support of Database Skills Testing	Закрытая	Сравнение результата выражения с результатом подготовленного корректного SQL-запроса.
University of Girona – An Automatic Correction Tool for Relational Algebra Queries	Закрытая	Сравнение результата выражения с подготовленным ответом

Проведенный анализ показал, что в открытом доступе находится только использованный в решении, принадлежащему Стенфордскому университету,

инструмент RA. Ra − это простой интерпретатор команд реляционной алгебры в SQL-запрос. Интерпретатор разработан на языке программирования Java и использует базу данных SQLite.RA поддерживает все операторы реляционной алгебры, кроме деления:

− выборка: \select_{cond};
− проекция: \project_{attr_list};
− соединение: \join_{cond};
− декартово произведение:\cross;
− объединение:\union;
− разность:\diff;
− пересечение:\intersect;

Несмотря достаточно полную функциональность, интерпретатор обладает двумя существенными недостатками:

1) сложный в восприятии синтаксис - студентам потребуется дополнительное время на его освоение;

2) отсутствие возможности записывать реляционное выражение пошагово, сохраняя промежуточные результаты в переменные- отношения, что приводит к нечитабельности реляционного выражения.

Так же анализ показал, что процесс проверки задач по реляционной алгебре практически во всех решениях происходит аналогичным образом: реляционное выражение выполняется на заранее подготовленных данных, для которых известен результат; если полученный результат совпадает с ожидаемым результатом, решение считается верным.

Исключение составляет подход, разработанный в Российском государственном профессионально-педагогическом университете, где полученное в качестве решения задачи реляционное выражение сравнивается с решениями из заранее подготовленного списка корректных решений. Данный подход не оптимален: для некоторых задач сложно составить полный список возможных решений. Именно поэтому в этом тренажере используется дополнительная проверка, когда полученное на вход выражение не найдено в списке верных решений. В таких случаях оценка его корректности также производится на основе возвращаемого им результата на подготовленных данных.

Из-за закрытости остальных систем, провести оценку качества проверки реляционных выражений представляется возможным только на решении Стэндфордского университета.

Все продукты имеют закрытую часть системы, кроме решения принадлежащего Стенфордскому университету. Система «Relational Algebra Exersises» не имеет возможности пополнить базу заданий, вести статистику по решенным задачам для каждого студента. Используется база данных SQLite, которая не предназначена для работы с большими нагрузками. Данный программный имеет значительные недостатки и непригоден для использования.

Выводы по главе

В данной главе рассмотрены основные определения и операции реляционной алгебры, а именно:

− Объединение;
− Пересечение;
− Вычитание;
− Декартово произведение;
− Выборка;
− Проекция;
− Соединение.

Был проведен обзор существующих решений, описаны их достоинства и недостатки. На основе этого был выполнен сравнительный анализ рассмотренных средств и разрабатываемого ПО и сформулированы требования позволяющие устранить рассмотренные недостатки существующих программных продуктов.

В связи с развитием информационных технологий появилась направление в образовании, которое бы автоматизировало процесс проверки самостоятельных и лабораторных работ.

Таким образом, разработка программного комплекса для решения задач по теме «Реляционная алгебра» является актуальной задачей, которая поможет эффективно обучать студентов, что и является целью данной работы.

2. ФОРМУЛИРОВАНИЯ ТРЕБОВАНИЙ. ВЫБОР СРЕДСТВ ДЛЯ СОЗДАНИЯ ПРОГРАМНОГО КОМПЛЕКСА

2.1 Модель процесса решения задачи студентом

На Рисунке 2.1 представлена модель процесса решения задачи студентом.

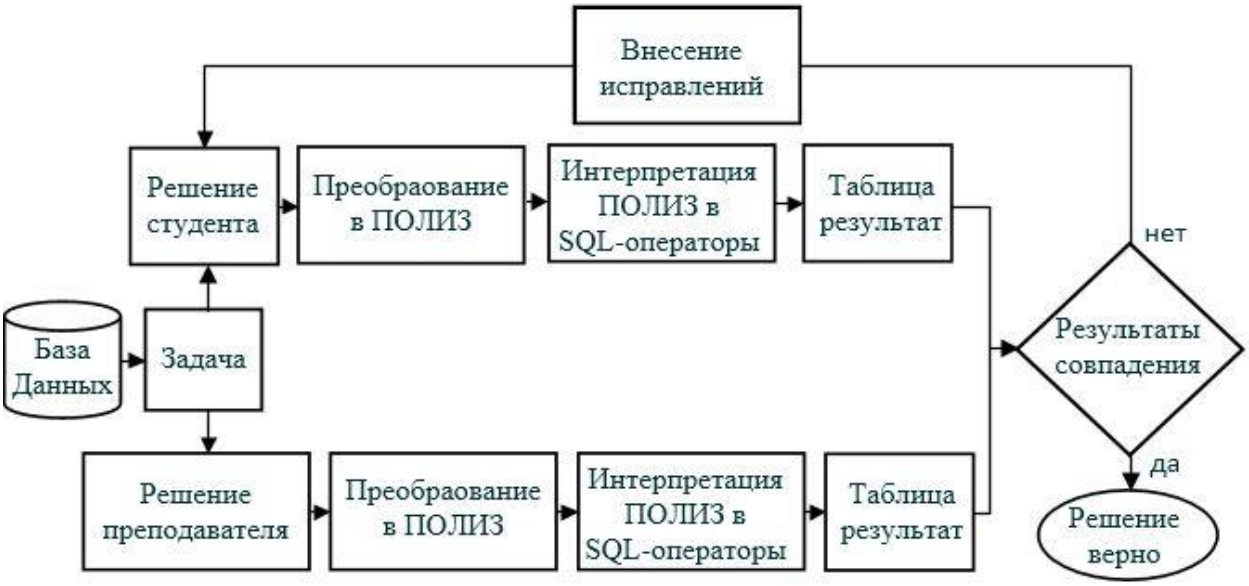

Рисунок 2.1 – Модель процесса решения задачи студентом

База данных содержит текст решения и эталонное решение. Задача загружается из БД. После ввода студентом ответа, текст решения преобразуется в обратную польскую запись (ПОЛИЗ) и интерпретируется в последовательность SQL операторов, выполняемую SQL-сервером. Результатом решения является некоторая таблица. С другой стороны аналогично выполняется эталонное решение. Далее происходит сравнение таблиц, порождаемых эталонным решением преподавателя и решением студента. Решение задачи может являться не единственным, но результирующая таблица для правильного решения всегда одна и та же. Диагностика ошибок выполняется частично программой и частично SQL-сервером. Нарушение синтаксиса реляционных выражений обнаруживается программой, а ошибки в сгенерированных программой SQL операторах обнаруживаются SQL-сервером. При наличии ошибок, студент получает соответствующее сообщение.

2.2 Модель взаимодействия с основной БД

Упрощенная схема взаимодействия приложений с базой данных изображена на рисунке 2.2:

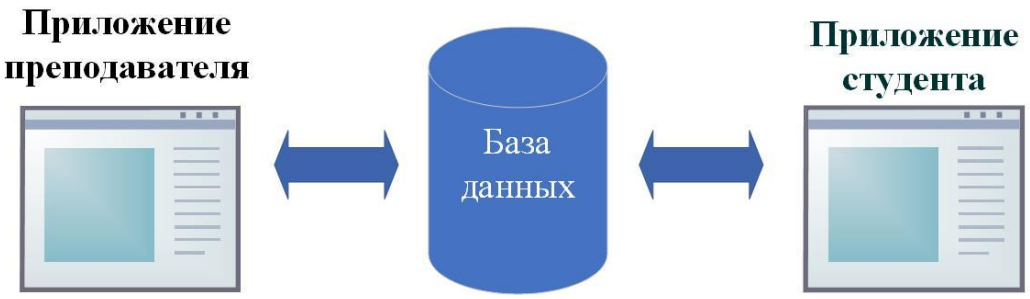

Рисунок 2.2 – Схема взаимодействия с базой данных

Приложения преподавателя и студента посылают SQL-запросы серверу баз данных, который обращается к базе и получает определенные данные [12-15].

2.2.1 Тестовая база данных «Продуктовый кооператив»

В качестве примера тестовой БД, к которой будут обращены запросы в форме выражений реляционной алгебры рассмотрим БД «Продуктовый кооператив».

На рисунке 2.3 изображена схема базы данных.

Продуктовый кооператив снабжает своих членов продуктами. База данных состоит из отношений:

(Пр) Продукт (название продукта (нпр))

(По) Поставщик (название поставщика(нпо))

(Пс) Поставка (нпр, нпо, цена) – поставщик поставляет этот продукт по цене

(Ч) Член (ФИО, Баланс(б))

(З) Заказ (ФИО, нпр, кол-во)

В каждый данный момент в БД имеется не более 1 заказа для каждого члена кооператива. Члены кооператива асинхронно вносят некоторые суммы, которые поступают на его счет и пополняют баланс. При выполнении заказа с баланса списывается его стоимость.

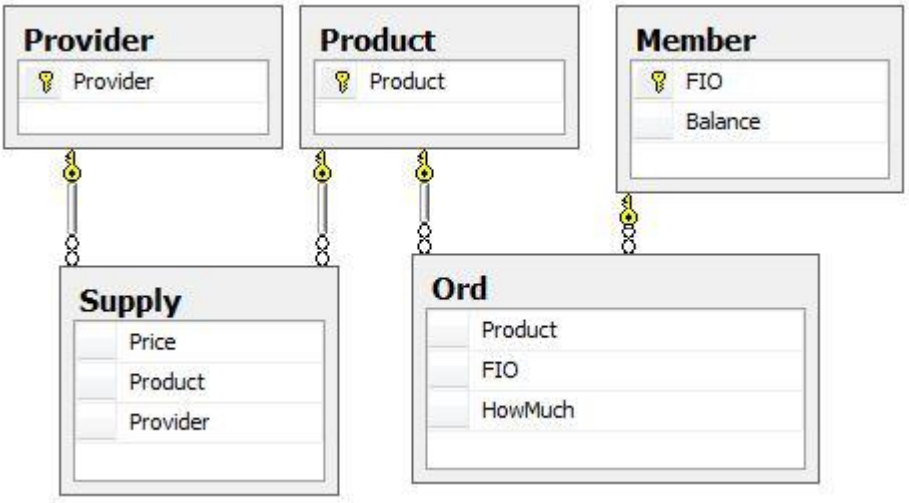

Рисунок 2.3 – Схема тестовой базы данных

Ниже в таблицах приведено описание таблиц входящих в БД.

Таблица «Provider» (Поставщик) содержит для каждого поставщика его название (Таблица 2.1).

Таблица 2.1

Поставщик

ИМЯ	ТИП	ПОЯСНЕНИЕ
Provider	varchar(50)	Название поставщика, например: ООО"Ферма"

Таблица «Product» (Продукт) содержит для каждого продукта его название (Таблица 2.2).

Таблица 2.2

Продукт

ИМЯ	ТИП	ПОЯСНЕНИЕ
Product	varchar(50)	Название продукта, например: хлеб.

Таблица «Member» (Член) содержит фамилию, имя, отчество и баланс (Таблица 2.3).

Таблица 2.3

Член

ИМЯ	ТИП	ПОЯСНЕНИЕ
FIO	varchar(50)	ФИО, например: Иванов И.
Balance	money	Баланс, например: 100

Таблица «Supply» (Поставка) содержит цену на продукт (Таблица 2.4).

Таблица 2.4

Поставка

ИМЯ	ТИП	ПОЯСНЕНИЕ
Price	money	Цена продукта
Product	varchar(50)	Продукт
Provider	varchar(50)	Поставщик

Таблица «Ord» (Заказ) содержит количество продуктов (Таблица 2.5).

Таблица 2.5

Заказ

ИМЯ	ТИП	ПОЯСНЕНИЕ
Product	varchar(50)	Продукта
FIO	varchar(50)	ФИО
HowMuch	float	Количество

Ниже в таблицах приведены данные таблиц входящих в БД.

Данные таблицы «Поставщик» (Таблица 2.6).

Таблица 2.6

Поставщик

	Поставщик
1	ООО "Моя земля"
2	ООО "Ферма"

Данные таблицы «Продукт» (Таблица 2.7).

Таблица 2.7

Продукт

	Продукт
1	Картошка
2	Курица
3	Мясо
4	Огурец
5	Помидор
6	Рыба
7	Хлеб
8	Черная икра

Данные таблицы «Член» (Таблица 2.8).

Таблица 2.8

Член

	ФИО	Баланс
1	Иванов И.И.	100,00
2	Петров П.П.	200,00
3	Сидоров С.С.	250,00

Данные таблицы «Поставка» (Таблица 2.9).

Таблица 2.9

Поставка

	Цена	Продукт	Поставщик
1	100,00	Курица	ООО «Ферма»
2	200,00	Рыба	ООО «Ферма»
3	50,00	Помидор	ООО «Ферма»
4	11,00	Картошка	ООО «Ферма»
5	250,00	Мясо	ООО «Моя земля»
6	30,00	Хлеб	ООО «Моя земля»
7	40,00	Огурец	ООО «Моя земля»
8	50,00	Помидор	ООО «Моя земля»
9	400,00	Черная икра	ООО «Ферма»

Данные таблицы «Заказ» (Таблица 2.10).

Таблица 2.10

Заказ

	Продукт	ФИО	Количество
1	Картошка	Иванов И.И.	3
2	Мясо	Иванов И.И.	4
3	Хлеб	Иванов И.И.	1
4	Картошка	Петров П.П.	2
5	Мясо	Петров П.П.	2
6	Курица	Петров П.П.	2
7	Огурец	Сидоров С.С.	4
8	Помидор	Сидоров С.С.	3
8	Черная икра	Сидоров С.С.	2

Ниже приведена формулировка задачи:

Список членов кооператива с отрицательным балансом, заказавшим черную икру.

Решение данной задачи:

$\pi_{FIO}(\sigma_{balance<0}(Member)$ Ord)

Процесс вычисления реляционного выражения иллюстрируется таблицей 2.11.

Таблица 2.11

Вычисление выражения

Операция	Описание
$A=\sigma_{balance<0}(Member)$	выборка из таблицы «Member» записей для членов кооператива с отрицательным балансом
B=A Ord	декартово произведение «A» и таблицы «Ord» по условию «Ord.FIO=Memb.FIO and Product='черная икра'»
$\pi_{FIO}(B)$	проекция результата на атрибут «FIO»

2.3Анализ требований к программе

2.3.1 Общие требования к программе

Программный комплекс состоит из основной базы данных и ряда тестовых, а также двух приложений-клиентов:

− приложение преподавателя;

− приложение студента.

Целью комплекса является создание комфортных условий при изучении раздела «Реляционная алгебра», как для студента, так и для преподавателя.

Студент приобретает возможности:

− решать задачи как в аудитории, так и в любом месте, где есть доступ к сети кафедры;

− обмениваться вопросами, ответами и замечаниями с преподавателем вне личного контакта;

− получать диагностику ошибок;

− видеть весь набор задач и выбирать личную последовательность их решения.

Преподаватель приобретает возможность:

− готовить упражнения по курсу;

− просматривать и анализировать студенческие решения в любое время и в любом месте;

− оказать помощь студенту в решении задач вне личного контакта;

− в любой данный момент видеть успехи группы.

2.3.1.1 Требования к функциональным характеристикам

Разрабатываемый программный комплекс должен обладать следующими функциями:

− работать под управлением ОС Windows;

− состоять из двух приложений, которые взаимодействуют с базами данных;

− разделение прав доступа управления базами данных;

− при запуске приложения обеспечивать авторизацию пользователя: запрашивать сервер, логин и пароль;

− главная форма приложения должна отображать перечень всех баз данных, хранящихся на сервере. При выборе из списка необходимой базы, пользователю должен предоставляться информация по ней;

− предоставлять возможность просмотра хранящейся в базе информации в табличной форме;

− иметь возможность удаления, редактирования и добавления записей таблицы.

2.3.1.2 Полномочия

Полномочия преподавателя определяются ролью БД «Преподаватель».

Роль «Преподаватель»:

− имеет право выполнять операции insert, update, delete над любыми таблиц как основной БД, так и тестовых БД;

− не имеет прав на какие бы то ни было изменения структур БД, их таблиц, программных текстов на стороне сервера.

Все студенты выполняют соединение с БД через один и тот же логин и пароль известный приложению. Право входа в приложение студента контролируется таблицей, где для каждого студента указан его зашифрованный пароль.

Абсолютными правами по отношению ко всем БД обладает администратор БД.

2.3.1.3 Требования к режиму функционирования ПО

Программный комплекс должен уметь обеспечить доступ к серверу кафедры, который содержит необходимые базы данных, с рабочих мест пользователей. На

стороне клиента должна быть реализована основная бизнес-логика (авторизация, обработка ошибок, операции с данными). Сервер же исполняет запросы на манипулирование данными (чтение, запись, удаление и модификацию).

2.3.1.4 Требования к внешнему оформлению и диалогу с пользователем

Взаимодействие пользователей с приложениями должно осуществляться с помощью визуального графического интерфейса (graphical user interface, GUI), функционирующего под управлением графической многозадачной операционной системы Windows, предоставляющего доступ ко всем предусмотренным функциональным возможностям. Интерфейс не должен быть перегружен графическими элементами и избыточной информацией. Компоненты интерфейса должны быть приближены к традиционным формам представления с учетом их смыслового значения. Приложение преподавателя и студента должны быть выполнены в едином стиле графического оформления интерфейса. Размер и положение компонентов форм должно автоматически рассчитываться в зависимости от разрешения монитора.

Необходимо обеспечить своевременную обратную связь для действий пользователя посредством визуального подтверждения того, что приложение восприняло и выполнило (или не выполнило) команду. Должны быть предоставлены сведения о состоянии текущего процесса, с возможностью остановить его в случае необходимости.

Система не должна допускать возможности внесение противоречий в БД. При наличии ошибок должно выводиться сообщение с наименованием ошибки и способами её устранения; все сообщения и названия экранных форм должны быть на русском языке.

Контроль над приложением должен осуществляться с помощью меню, значков и других элементов, рассчитанных на применение мыши и клавиатуры; при заполнении и редактировании числовых или текстовых полей должна использоваться клавиатура. Обязателен полный контроль над вводимыми пользователем данных. Например, при вводе числа программа не должна реагировать на ввод символов, которые не могут присутствовать в числе.

2.3.2 Требования к надежности

2.3.2.1 Требования к информационной безопасности

Защита информации от несанкционированного доступа обязана удовлетворять следующим требованиям:

− вход в систему разрешается только зарегистрированным пользователям;

− программные средства защиты не должны реализовываться в ущерб основным функциональным характеристикам ПО, таким как быстродействие и надежность.

2.3.2.2 Обработка отказов программы и аварийных ситуаций

Отказы программы вследствие некорректных действий пользователя при взаимодействии с приложением недопустимы. Должна обеспечиваться обработка ошибок, которые могут быть вызваны вводом неверных значений или

неправильным форматом данных. Все аварийные ситуации (из-за недопустимых или непредусмотренных действий пользователей) должны обрабатываться на программном уровне. В этих случаях нужно выдавать соответствующее сообщение, а приложение возвращать в состояние, предшествовавшее ошибке или аварии без потери обрабатываемой информации.

2.3.3 Условия эксплуатации

1.3.3.1 Требования к квалификации персонала

Пользователь, работающий с приложением преподавателя, должен являться администратором базы данных, иметь доступ ко всем базам и таблицам, а также обладать правами наполнения, редактирования и удаления записей.

Пользователь, работающий с приложением студента, должен иметь доступ ко всем базам и таблицам, а также обладать правами наполнения, редактирования и удаления записей в таблицах, к которым ему разрешен доступ.

Предполагается, что пользователь обладает базовыми знаниями в области администрирования баз данных, имеет опыт работы с windows-приложениями: умеет вводить информацию в экранные формы, работать с контекстным и главным меню и т.д.

1.3.3.2 Требования к составу и параметрам технических средств

Для функционирования приложения требуется:

− компьютер на котором расположен SQL-сервер должен иметь около 300Мб свободной дисковой памяти и не менее 4Гб оперативной памяти

− для компьютера пользователя достаточно самых скромных возможностей

− операционная система WindowsXP и выше;

− доступ к серверу баз данных;

2.4 Выбор инструментов для создания ПО и языка программирования

При разработке ПО всегда стоит задача о выборе типа интерфейса-desktop или web-интерфейс. Большинство разработчиков останавливают свой выбор на web-приложениях. Это объясняется тем, что web-интерфейсы более доступны, т.к. они взаимодействуют через браузер, который установлен на многих компьютерах. Однако стремление достичь единого решение приводит к усложнению системы, что негативно влияет на производительность. Код обрастает множественными модулями, вводятся большое количество абстракций, что усложняет проектирование и работу системы.

Альтернативой является desktop приложения, которые считаются более выигрышным вариантом. Абстракции программной архитектуры и сложные интерфейсы выносятся на уровень клиента, способы управления функциональностью становятся доступнее и проще. А именно:

− изменения, внесенные в данные, обычно ограничены от серверной базы данных и не оказывают на нее никакого воздействия, пока пользователь не подтвердит результат;

− существуют стандартные элементы управления, позволяющие отображать данные базы в табличном формате, в настраиваемой сетке. К преимуществам представленного подхода можно отнести: легкость реализации таких базовых функций, как правка содержимого ячеек; одновременный просмотр значительных объемов табличных данных;

Проанализировав вышеизложенную информацию можно сделать следующие выводы:

− рекомендуется делать выбор в пользу web-приложений, если основной упор делается на доступность и оперативность, а также минимальным требованиям к навыкам пользователей;

− специализированный клиент будет лучшим решением в том случае, если в приоритете гибкость системы, а одной из целей является обеспечение доступа к большим объемам структурированной информации и взаимосвязям между объектами.

Язык C++ имеет большое применение в разработки программного обеспечения и тем самым являясь одним из популярных языков программирования. Особенности языка C++:

− поддерживает объектно-ориентированный подход, который позволяет проектировать масштабируемые, гибкие и расширяемые приложения;

− имеет большую стандартную библиотеку.

В рамках учебной программы изучается среда разработки Borland Developer Studio 2006 [16]. В целях сокращения трудозатрат на изучения другой среды программирования за основу была взята Borland Developer Studio 2006. Также на кафедре имеется лицензия данного программного продукта.

Для взаимодействия MS SQL Server R2 используется Microsoft SQL Server Management Studio [17-18]. SSMS – интегрированная среда для управления любой инфраструктурой SQL. Данная среда является наиболее мощной и до сих пор поддерживаемой разработчиками. Имеет возможности настройки, администрирования и наблюдения экземпляров SQL,также можно обновлять, развертывать, отслеживать компоненты уровня данных, создавать скрипты и запросы.

Выводы по главе

В данном разделе приведено подробное описание структуры тестовой базы данных «Продуктовый кооператив». Рассмотрен пример задачи и ее решение в операциях реляционной алгебры. Проанализированы требования к разрабатываемому ПО, в том числе требования к функциональным характеристикам, режиму функционирования, интерфейсу, надёжности и условиям эксплуатации.

Было приведено обоснование выбора инструментов для разработки приложения, в частности:

– выбор типа приложения был сделан в пользу разработки desktop-приложения;

– при написании кода используется язык программирования C++ и среда программирования Borland Developer Studio 2006;
– для реализации взаимодействия с MS SQL Server R2 используется Microsoft SQL Server Management Studio.

3. МАТЕМАТИЧЕСКАЯ МОДЕЛЬ И РАЗРАБОТКА ИНТЕРПРЕТАТОРА ВЫРАЖЕНИЙ РЕЛЯЦИОННОЙ АЛГЕБРЫ

3.1 База данных Relation

База Relation является основной и содержит информацию о тестовых базах данных для задач, вопросы по задачам, и эталонные ответы [19]. Задачи формулируются как некоторые запросы, которые нужно построить в алгебраической форме.

В каждом году обучается некоторое множество групп. В каждой группе содержатся некоторое количество студентов, состав групп определяется деканатом. Таблица «Base» содержит описание тестовых баз данных, к которым адресуются запросы. Таблица «Query» содержит запросы к текстовым базам данных, на которые студент дает ответ в виде совокупностей выражений реляционной алгебры. Таблица «Answer» содержит решения студентов в виде совокупностей выражений реляционной алгебры. Таблица «Status» отражает состояние решенной студентом задачи, и имеет значения: «Не проверено», «Не зачтено», «Зачтено», «Списано», «Студент не проверял», «Зачтено см. комментарий». Таблица «Comment» содержит комментарии студентов и преподавателя.

Студенту предоставлена возможность решать задачи и просматривать свои результаты. Обмениваться вопросами, ответами и замечаниями с преподавателем вне личного контакта.

Преподаватель может просматривать результаты решения задач, редактировать задачи и добавлять новые задачи. Отвечать на вопросы студента вне личного контакта.

База данных состоит из отношений:

Год (Учебный год)

Группа (Название группы)

Студент (ФИО, Пароль)

База (Название базы, Логин, Пароль, Название сервера, Картинка, Описание)

Запрос (Текст, Картинка, Номер задачи, Оценка, Эталонный ответ)

3.1.1 Реализация БД в MS SQL Server

Ниже приведена основная схема БД (Рисунок 3.1).

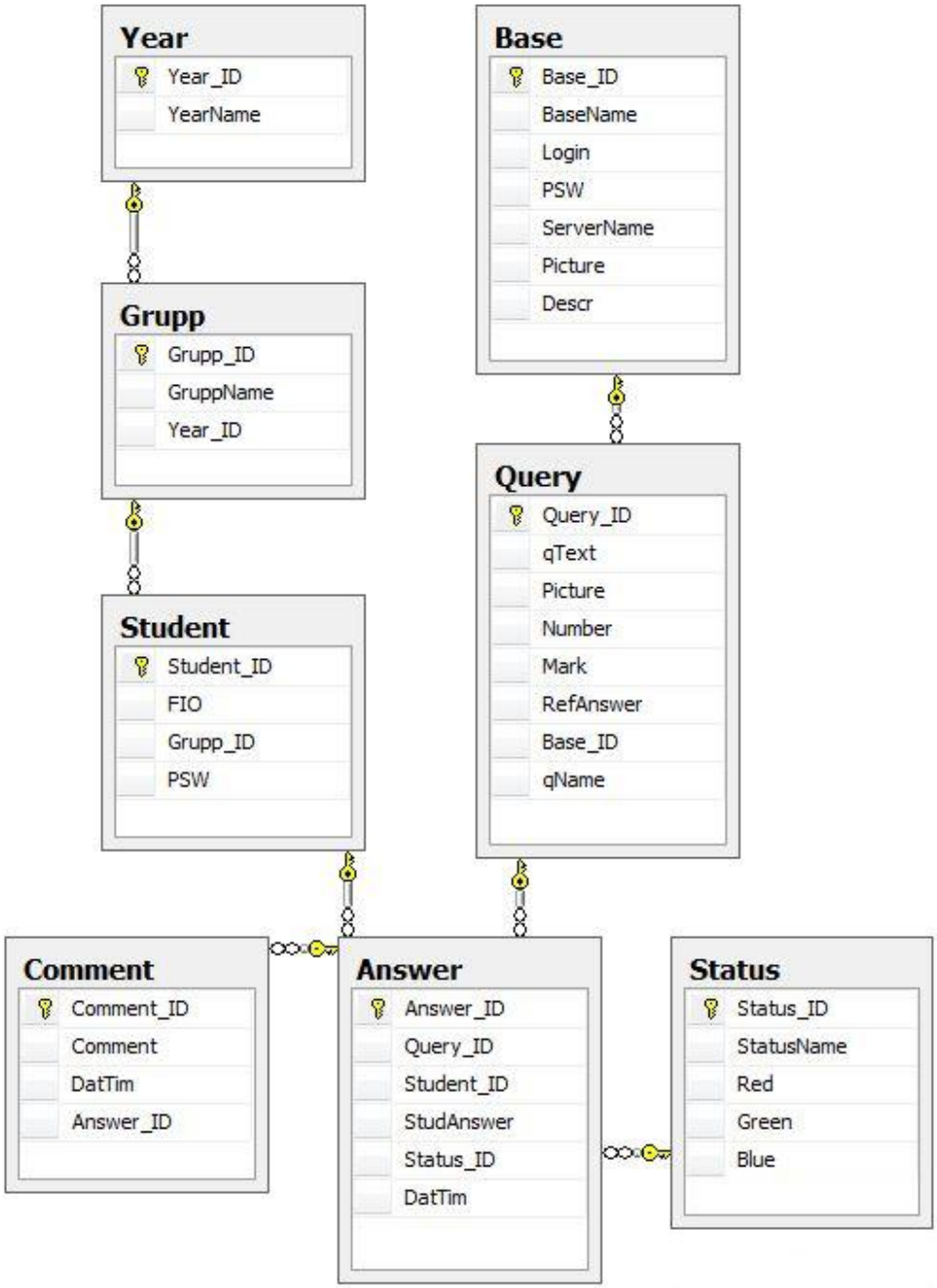

Рисунок 3.1 – Основная схема БД

3.1.2 Описание структуры БД

Для реализации БД используется MS SQL Server 2008 R2.

Ниже в таблицах приведено описание таблиц входящих в БД. Каждая таблица имеет искусственный первичный ключ – поле, имя которого совпадает с именем таблицы, имеет суффикс ID и обладает свойством identity.

3.1.2.1 Таблица «Year» (Учебный год)

Таблица содержит для каждого учебного года его наименование (Таблица 3.1).

Таблица 3.1

Учебный год

Имя	Тип	Пояснение
Year_ID	identity	Первичный ключ
YearName	char(9)	Наименование учебного года, например: 2017/2018

3.1.2.2 Таблица «Grupp» (Группа)

Таблица содержит для каждой академической группы ее наименование (Таблица 3.2).

Таблица 3.2

Группа

Имя	Тип	Пояснение
Grupp_ID	identity	Первичный ключ
GruppName	varchar(10)	Наименование академической группы, например: 121
Year_ID	int	Идентификаторучебного года

3.1.2.3 Таблица «Student» (Студент)

Таблица содержит для каждого студента его фамилию, имя, отчество и пароль (Таблица 3.3).

Таблица 3.3

Студент

Имя	Тип	Пояснение
Student_ID	identity	Первичный ключ
FIO	varchar(50)	ФИО студента, например: Иванов И.
Grupp_ID	int	Идентификатор
PSW	varchar(128)	Пароль, например: 123

3.1.2.4 Таблица «Base» (База)

Таблица содержит имя БД, логин, пароль и имя сервера (Таблица 3.4).

Таблица 3.4

База

ИМЯ	ТИП	ПОЯСНЕНИЕ
Base_ID	identity	Первичный ключ
BaseName	varchar(50)	Имя базы, например: TEST1
Login	varchar(50)	Логин, например: Petrov
PSW	varchar(255)	Пароль, например: 123
ServerName	varchar(50)	Имя сервера, например: MSSQL-2K8\KAT
Picture	varbinary(MAX)	Схема базы данных
Descr	varbinary(MAX)	Описание БД

3.1.2.5 Таблица «Query» (Запрос)

Таблица содержит текст запроса, ответ, сложность задачи и картинку со схемой БД (Таблица 3.5).

Таблица 3.5

Запрос

ИМЯ	ТИП	ПОЯСНЕНИЕ
Query_ID	identity	Первичный ключ
qText	varbinary(MAX)	Текст запроса, например: SELECT DISTINCT x,y FROM B
Picture	varbinary(MAX)	Картинка
Number	int	Номер задачи, например 1
Mark	tinyint	Сложность задачи, например 2
RefAnswer	varbinary(MAX)	Ответ, например: proj(B,{x,y})
Base_ID	int	Идентификатор
qName	varchar(50)	Название задачи

3.1.2.6 Таблица «Answer» (Ответ)

Таблица содержит для каждого студента его ответ на задачу (таблица 3.6).

Таблица 3.6

Ответ

ИМЯ	ТИП	ПОЯСНЕНИЕ
Answer_ID	identity	Первичный ключ
Query_ID	int	Идентификатор
Student_ID	int	Идентификатор
StudAnswer	varchar(MAX)	Ответ студента, например: proj(B,{x,y})
Status_ID	int	Идентификатор
DatTim	smalldatetime	Дата-время ответа

3.1.2.7 Таблица «Status» (Статус)

Таблица содержит статус для каждого ответа на задачу (таблица 3.7).

Таблица 3.7

Статус

ИМЯ	ТИП	ПОЯСНЕНИЕ
Status_ID	identity	Первичный ключ
StatusName	varchar(30)	Статус
Red	tinyint	Цвет-красный
Green	tinyint	Цвет-зеленый
Blue	tinyint	Цвет-синий

3.1.2.8 Таблица «Comment» (Комментарий)

Таблица содержит комментарий для каждой задачи (таблица 3.8).

Таблица 3.8

Комментарий

ИМЯ	ТИП	ПОЯСНЕНИЕ
Comment_ID	identity	Первичный ключ
Comment	varchar(MAX)	Статус
DateTim	smalldatetime	Дата-время ответа
Answer_ID	int	Идентификатор

3.2 Форма записи ответа

Решение формируется, как некоторое выражение, использующее операции реляционной алгебры, которое записывается в функциональной форме. В таблице 3.9 приведены формы записи реляционной алгебры и их соответствующая функциональная форма записи. Нет возможности осуществить отображения более чем двумерных выражений, поэтому выражения буду представлены в функциональной форме записи.

Таблица 3.9

Форма записи

Операция	Реляционная алгебра	Функциональная запись
Объединение	$A \cup B$	Union(A,B)
Пересечение	$A \cap B$	Intersect(A,B)
Вычитание	$A - B$	Except(A,B)
Декартово произведение	$A \times B$	Cross(A,B)
Выборка	$\sigma_f(A)$	Where(A,{f})
Проекция	$\pi_{a,b}(A)$	Project(A,{a,b})
Соединение		Join(A,B,{f})

Пример ответа: Minus(Join(A,B,{A.x=B.x}), Project(C,{x,y,z}))

Фрагменты выражения в фигурных скобках рассматриваются как единые лексемы, которые будут подставлены в SQL-операторы без каких-либо преобразований [20-21].

После ввода ответа строка преобразуется в обратную польскую запись и интерпретируется в последовательность SQL-операторов выполняемую SQL Server-ом [22]. Далее происходит сопоставление эталонного решения с результатом выполнения SQL-операторов и выдачей решения о правильности или неправильности решения.

Процесс выполнения текста решения состоит из 3 фаз:

1) Лексический анализ-выделение операций, отношений, списков полей, логических выражений и скобок.

2) Преобразование в ПОЛИЗ.

3) Преобразование ПОЛИЗ в последовательность SQL-операторов.

4) Выполнение полученных SQL-операторов.

3.3 Интерпретатор

Обратная польская запись (ПОЛИЗ) – способ бесскобочного представления выражений (не только арифметических), в которых операнды предшествуют операции. Рассмотрим процесс преобразования в ПОЛИЗ на примере арифметического выражения. Так, выражение (в обратной польской записи буде выглядеть следующим образом:

Выражение, представленное в ПОЛИЗ, замечательно тем, что оно может быть вычислено за один проход слева направо без возвратов и забеганий вперед. Вычисление выражения в ПОЛИЗ выполняется следующим образом. Двигаясь слева направо по строке, операнды помещаем в стек. Когда встретится операция, то она выполняется над операндами, находящимися на вершине стека. Результат операции помещается в вершину стека вместо использованных операндов. Таблица 3.10, приведенная ниже, иллюстрирует процесс вычисления. В столбце "Входная строка" подчеркнута обработанная часть строки. Промежуточные результаты обозначены R0, R1, R2. Окончательный результат вычисления –

единственный элемент на вершине стека операндов. Ниже рассмотрим вычисление значения выражения, заданного в ПОЛИЗ.

Таблица 3.10

Вычисление значения выражения, заданного в ПОЛИЗ

Входная строка	Стек	Пояснение
4 A * 2 X / − 3 B * 2 Y * + *	4	
4 A * 2 X / − 3 B * 2 Y * + *	A4	
4 A * 2 X / − 3 B * 2 Y * + *	R_0	R_0=4*A
4 A * 2 X / − 3 B * 2 Y * + *	2R_0	
4 A * 2 X / − 3 B * 2 Y * + *	X2R_0	
4 A * 2 X /− 3 B * 2 Y * + *	R_1R_0	R_1=2/X
4 A * 2 X / − 3 B * 2 Y * + *	R_0	R_0=R_0−R_1
4 A * 2 X / − 3 B * 2 Y * + *	3R_0	
4 A * 2 X / − 3 B * 2 Y * + *	B3R_0	
4 A * 2 X / − 3 B * 2 Y * + *	R_1R_0	R=3*B
4 A * 2 X / − 3 B * 2 Y * + *	2R_1R_0	
4 A * 2 X / − 3 B * 2 Y * + *	Y2R_1R_0	
4 A * 2 X / − 3 B * 2 Y * + *	$R_2R_1R_0$	$R_{2=}$2*Y
4 A * 2 X / − 3 B * 2 Y * + *	R_1R_0	R_2=R_1+R_2
4 A * 2 X / − 3 B * 2 Y * + *	R_0	R_0=R_0*R_1

Рассмотрим алгоритм преобразования скобочного выражения в ПОЛИЗ. Для простоты ограничимся арифметическими выражениями, использующими четыре действия арифметики. Предположим также, что операндами являются только переменные с однобуквенными идентификаторами. Преобразование выполняется за один проход. Строка просматривается слева направо.

Операнды сразу помещаются в выходную строку.

Знаки операций сначала помещаются в стек. Прежде чем быть помещенным в стек, знак операции выталкивает из стека все операции, которые имеют приоритет больше или равный приоритета входной операции.

Открывающая скобка просто помещается в стек как операция с самим низким приоритетом.

Закрывающая скобка выталкивает из стека в выходную строку все операции вплоть до ближайшей открывающей скобки, которая удаляется из стека, но в выходную строку не помещается. Закрывающая скобка в стек не помещается.

После того как входная строка закончилась, остаток стека выталкивается в выходную строку.

3.4 Трансляция операций в операторы SQL

Для реализации процесса использованы такие структуры, как конечный автомат, стеки и линейные списки и здесь подробно не рассматриваются. Описание процесса можно понять их текста программы, которое приведено в приложении (Приложение 1).

Таблица содержит приоритеты функций реляционной алгебры (Таблица 3.11) [23].

Таблица 3.11

Приоритеты функций

Union - объединение	1
Intersect - пересечение	2
Except - вычитание	1
Cross - декартово произведение	2
Where - выборка	3
Project- проекция	3
Join- соединение	2

Для выполнения преобразований используются временные таблицы. Все временные таблицы имеют имя #RR(номер), где номер представляет из себя нумерацию временной таблицы. Номер формируется из временной базы данных с помощью функции, которая содержит массив имен таблиц [24-25].

Рассмотрим подробно преобразование операций реляционной алгебры в SQL запросы.

3.4.1 Объединение

Операция объединения представляется в виде Union(A, B), где A и B отношения с атрибутами. Каждое имя отношения помешается последовательно в стек. С помощью функции проверяем наличия знака «#» перед именем таблицы, если его нет, то добавляем. Далее с использованием временной таблицы формируем запрос, который представляет из себя объединение двух отношений.

3.4.2 Пересечение

Операция пересечение представляется в виде Intersect(A,B). Каждое имя отношения помешается последовательно в стек. С помощью функции проверяем наличия знака «#» перед именем таблицы, если его нет, то добавляем. Далее с использованием временной таблицы формируем запрос, который представляет из себя пересечение двух отношений.

3.4.3 Вычитание

Операция вычитание представляется в виде Except(A,B). Каждое имя отношения помешается последовательно в стек. С помощью функции проверяем наличия знака «#» перед именем таблицы, если его нет, то добавляем. Далее с

использованием временной таблицы формируем запрос, который представляет из себя вычитание двух отношений.

3.4.4 Декартово произведение

Операция декартово произведение представляется в виде Cross(A,B). Каждое имя отношения помешается последовательно в стек. С помощью функции проверяем наличия знака «#» перед именем таблицы, если его нет, то добавляем. Далее формируем список из всех полей таблиц A и B, если в A есть поле с таким же именем, как в B, то не включаем его в результат. Добавляем знак «#» к именам таблиц в строках, содержащих элементы вида <имя таблицы>.<имя поля>. С помощью функции получаем для каждого отношения список полей. Далее с использованием временной таблицы формируем запрос, который представляет из себя декартово произведение двух отношений.

3.4.5 Выборка

Операция выборка представляется в виде Where(A,{f}). Каждое имя отношения помешается последовательно в стек. С помощью функции проверяем наличия знака «#» перед именем таблицы, если его нет, то добавляем. Далее формируем список из всех полей таблицы A. Добавляем знак «#» к именам таблиц в строках, содержащих элементы вида <имя таблицы>.<имя поля>. С помощью функции получаем для каждого отношения список полей. Далее с использованием временной таблицы формируем запрос, который представляет из себя выборку по условию.

3.4.6 Проекция

Операция проекция представляется в виде Project(A,{a,b}). Каждое имя отношения помешается последовательно в стек. С помощью функции проверяем наличия знака «#» перед именем таблицы, если его нет, то добавляем. Далее формируем список из всех полей таблицы A. Добавляем знак «#» к именам таблиц в строках, содержащих элементы вида <имя таблицы>.<имя поля>. С помощью функции получаем для каждого отношения список полей. Далее с использованием временной таблицы формируем запрос, который представляет из себя проекцию по условию.

3.4.7 Соединение

Операция проекция представляется в виде Join(A,B,{f}). Каждое имя отношения помешается последовательно в стек. С помощью функции проверяем наличия знака «#» перед именем таблицы, если его нет, то добавляем. Далее формируем список из всех полей таблицы A и B, если в A есть поле с таким же именем, как в B, то не включаем его в результат. Добавляем знак «#» к именам таблиц в строках, содержащих элементы вида <имя таблицы>.<имя поля>. С помощью функции получаем для каждого отношения список полей. Далее с использованием временной таблицы формируем запрос, который представляет из себя соединение по условию.

3.5 Представление операций в приложении

Операторы реляционной алгебры представляются в функциональной форме записи. На рисунке 3.2 представлен пример функциональной записи.

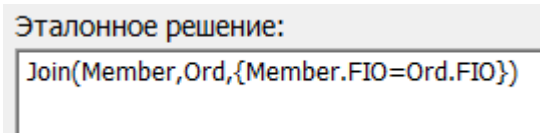

Рисунок 3.2 – Функциональная форма записи

Приведенное выше решение преобразуется в последовательность SQL-операторов, представленных на рисунке 3.3.

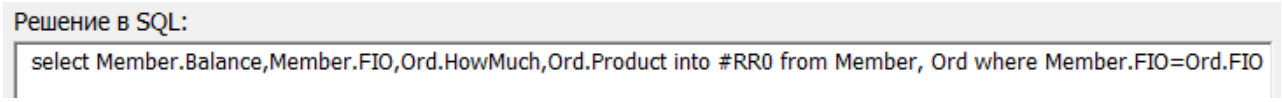

Рисунок 3.3 – SQL запрос

Полученный результат представляется в табличном виде (Рисунок 3.4).
Окончательный результат-это вычисление последнего SQL-оператора.

Ответ:

Balance	FIO	HowMuch	Product
100	Иванов И.И.	3	Картошка
100	Иванов И.И.	4	Мясо
100	Иванов И.И.	1	Хлеб
200	Петров П.П.	2	Картошка
200	Петров П.П.	2	Мясо
200	Петров П.П.	2	Курица
250	Сидоров С.С.	4	Огурец

Рисунок 3.4 –Таблица-результат

3.6 Обработка запросов с ошибкой

Разного рода ошибки в решении выявляются как приложением, так и SQL-сервером. Синтаксические ошибки в реляционных выражениях выявляются приложениями, а синтаксические ошибки SQL-операторов выявляются SQL- сервером и сообщаются пользователю.

Пример ошибки, допущенной в запросе. В функциональной форме записи допущена ошибка, пропущена точка в условии «Member.FIO=Ord.FIO». После выполнения запроса, SQL-сервер не сможет обработать данный запрос и вернет ошибку, приложение обработает данную ошибку и отобразиться следующее сообщение рисунок 3.5.

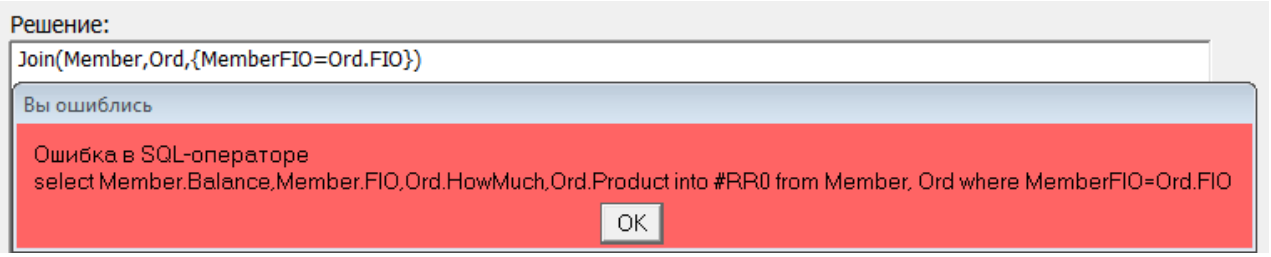

Рисунок 3.5 – Ошибка запроса

Пример ошибки, допущенной в запросе. В функциональной форме записи допущена ошибка, пропущена закрывающая фигурная скобка в условии «{Member.FIO=Ord.FIO}». Интерпретатор не сможет преобразовать данную запись в ПОЛИЗ и приложение выдаст сообщение с ошибкой рисунок 3.6.

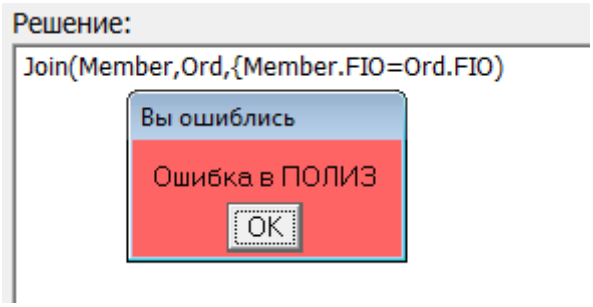

Рисунок 3.6 – Ошибка в ПОЛИЗ

Выводы по главе

В данном разделе приведено подробное описание структуры базы данных «Relation». Также представлена схема БД «Relation».

Для каждого оператора задано функциональное представление и приоритеты. На примере рассмотрено преобразование выражение в обратную польскую запись. Подробно описаны последовательность преобразования функциональной формы записи в SQL запрос.

На конкретном примере представлена форма записи выражения, его SQL представления и результат, которые отображаются в приложении.

Разного рода ошибки в решении выявляются как приложением, так и SQL-сервером. Синтаксические ошибки в реляционных выражениях выявляются приложениями, а синтаксические ошибки SQL-операторов выявляются SQL- сервером и сообщаются пользователю. Приведены основные ошибки и их обработка.

3. РАЗРАБОТКА ИНТЕРФЕЙСОВ ПРИЛОЖЕНИЙ

4.1 Приложение преподавателя

4.1.1 Регистрация

Фома регистрации появляется в момент старта программы (Рисунок 4.1).

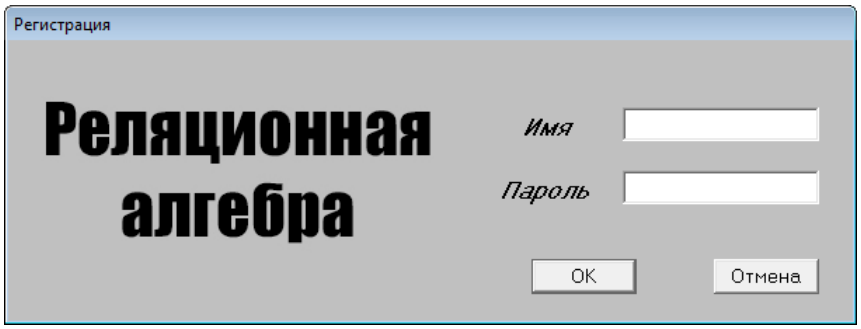

Рисунок 4.1 – Форма регистрации

Пользователь заполняет поля «Имя» и «Пароль». В случае успешной регистрации раскрывается окно головной формы (Рисунок 4.4). Если пользователь ввел неправильные данные, то появится окно с сообщением ошибки (Рисунок 4.2).

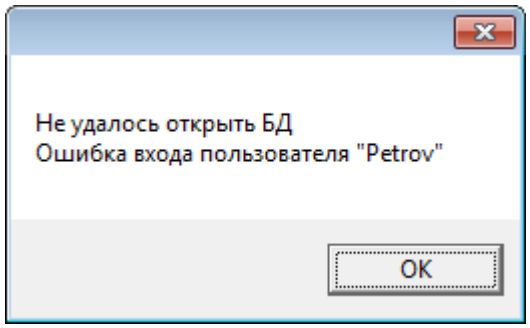

Рисунок 4.2 – Сообщение ошибка входа

При нажатии кнопки «Отмена» появится следующее окно с сообщением (Рисунок 4.3).

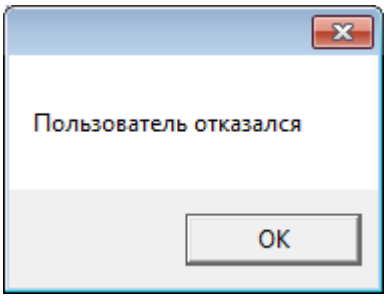

Рисунок 4.3 – Сообщение отказа входа

4.1.2 Главное окно

Рассмотрим подробно основные элементы главного окна (Рисунок 4.4).

1. Выпадающее меню содержит список учебных годов. По умолчанию выбирается текущий учебный год.

2. Главное меню программы. Содержит два пункта «Группы и студенты» и «Результат».

3. Таблица, которая отображает список тестовых баз данных, имя сервера, на котором хранится выбранная база, логин и пароль для получения доступа к базе данных.

4. Схематичное представление выбранной базы данных.

5. Описание структуры и особенностей выбранной базы данных.

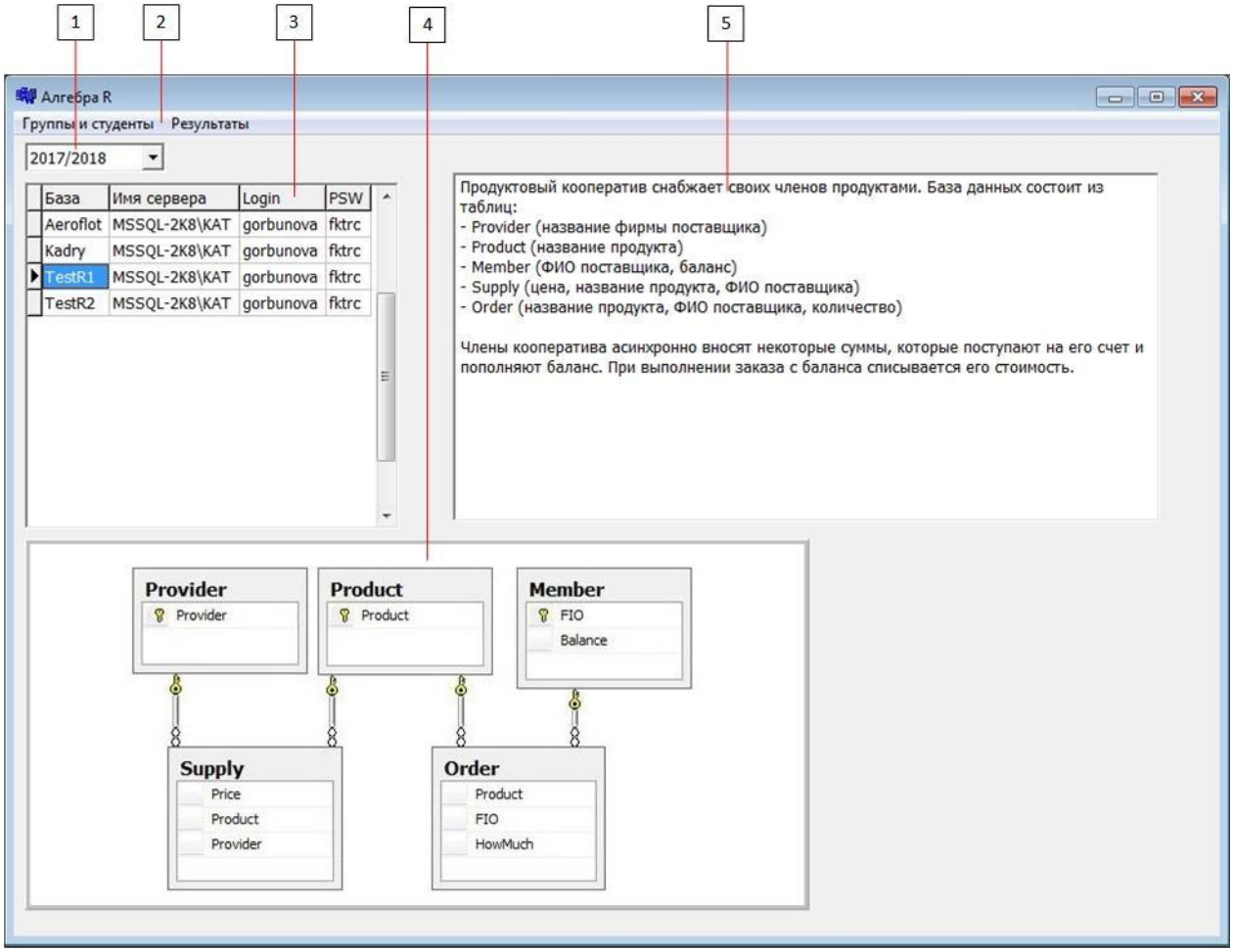

Рисунок 4.4 – Главное окно

4.1.3 Пункт меню «Группы и студенты»

При выборе пункта меню «Группы и студенты» отобразится форма (Рисунок 4.5).

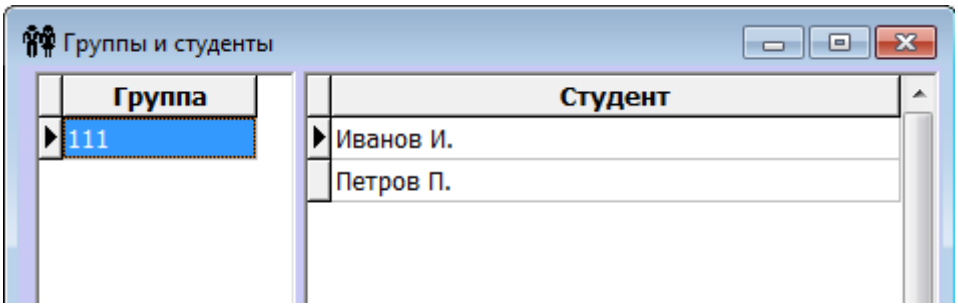

Рисунок 4.5 – Группы и студенты

В данном окне можно просмотреть названия групп и студентов, которые учатся в соответствующей группе.

Основным элементом интерфейса пользователя является таблица (Grid). Текущая запись в таблице выделяется значком  и подсвечивается в левой части таблицы. Как правило, каждая таблица имеет контекстное меню, вызываемое нажатием и отпусканием правой кнопки мыши на поверхности таблицы. Операции контекстного меню, относятся, к текущей записи. Почти каждая таблица имеет пункты контекстного меню, позволяющие редактировать таблицу – это команды: «Добавить», «Изменить», «Удалить». Кроме них, в каждом конкретном случае могут быть и другие команды. На рисунке 4.6 изображено контекстное меню таблицы «Группа».

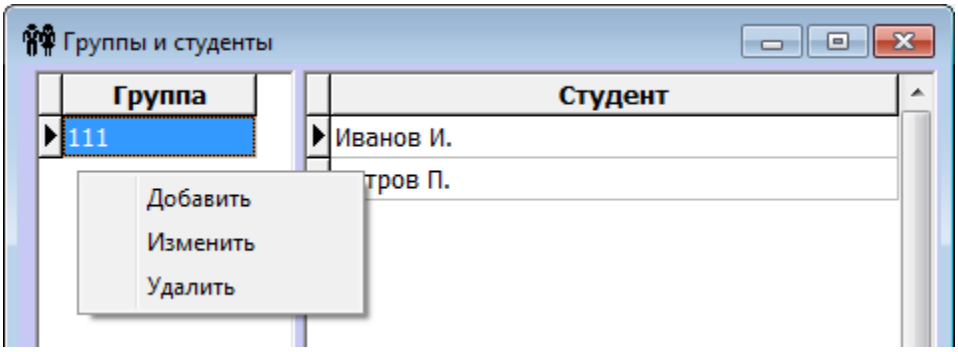

Рисунок 4.6 – Контекстное меню таблицы группа

При выборе пункта меню «Добавить», появится форма, изображенная на рисунке 4.7. Нажатие кнопки «ОК» означает, что пользователь удовлетворен результатами редактирования и желает поместить их в базу данных, нажатие кнопки «Отмена» означает отказ от операции.

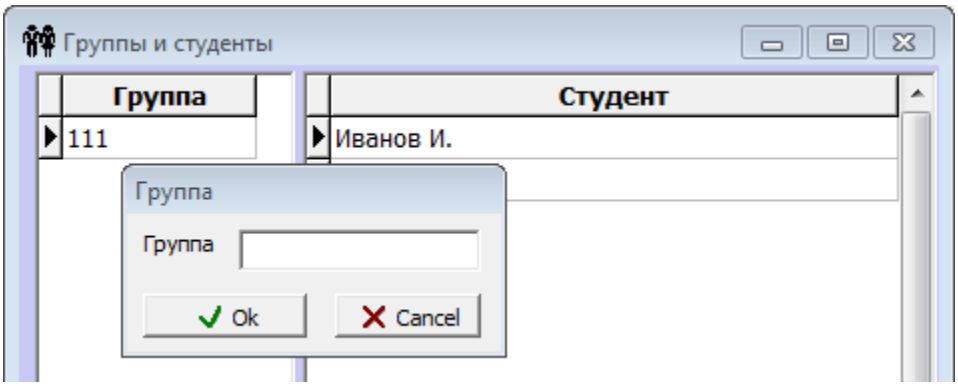

Рисунок 4.7 – Добавить группу

При выборе пункта меню «Изменить» появится форма, где в поле «Группа» отобразиться редактируемое поле (Рисунок 4.8).

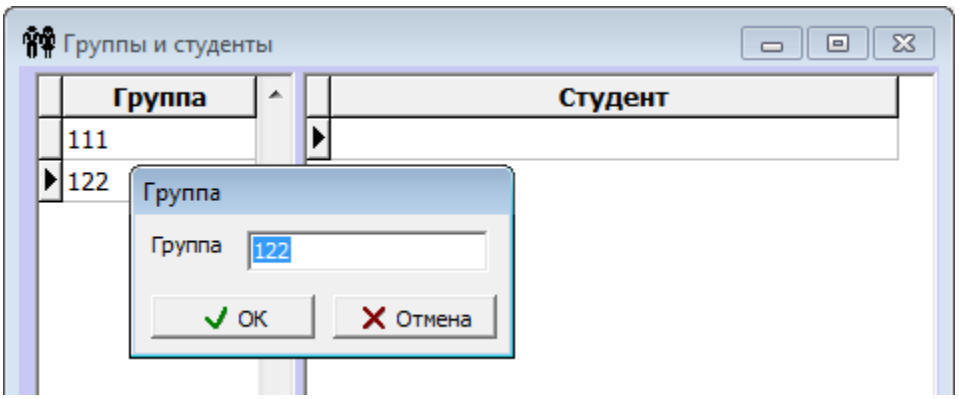

Рисунок 4.8 –Редактировать группу

При выборе пункта меню «Удалить» появится сообщение с подтверждением действия (Рисунок 4.9). По нажатию кнопки «Да» действие подтвердиться и удалиться выбранная группа. Кнопка «Отмена» отменяет действие.

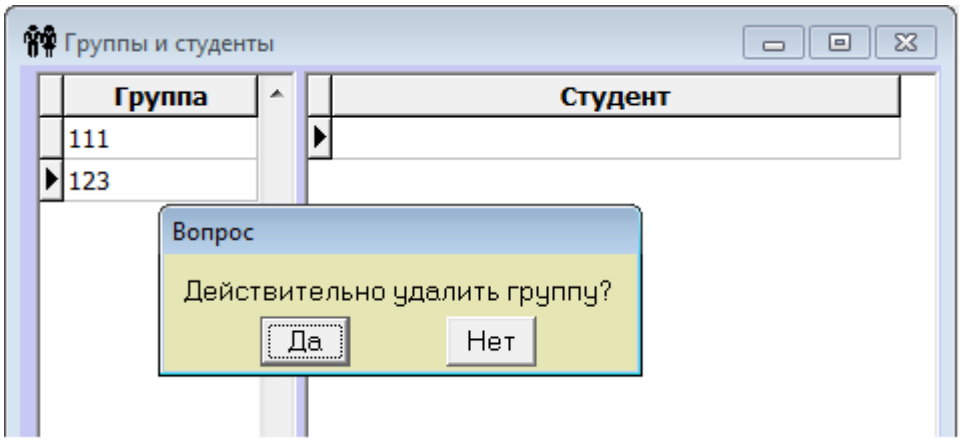

Рисунок 4.9 – Удаление группы

Таблица «Студент» имеет пункты «Добавить», «Изменить», «Удалить» (Рисунок 4.10).

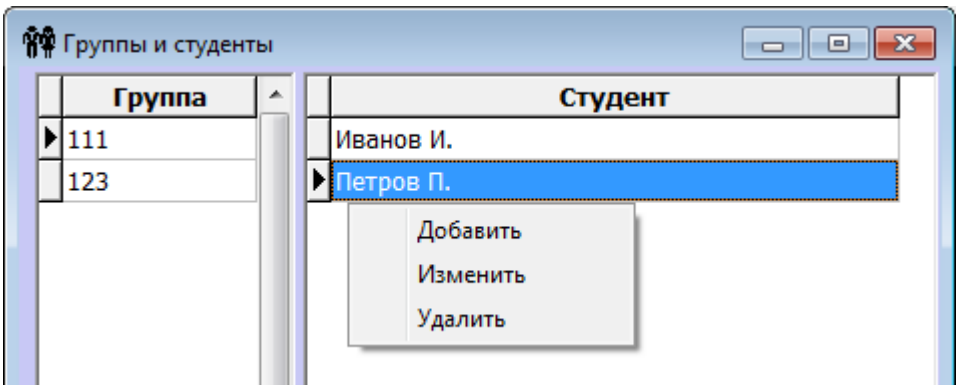

Рисунок 4.10 – Контекстное меню таблицы студент

При выборе пункта меню «Добавить» появиться форма добавления студента (Рисунок 4.11). Форма содержит два поля для ввода информации. В поле «Ф.И.О.» добавляется фамилия, имя и отчество студента. Поле «Пароль»

используется для назначения пароля. Если в поле «Пароль» оставить значение «0», то студент сможет сам назначить себе пароль.

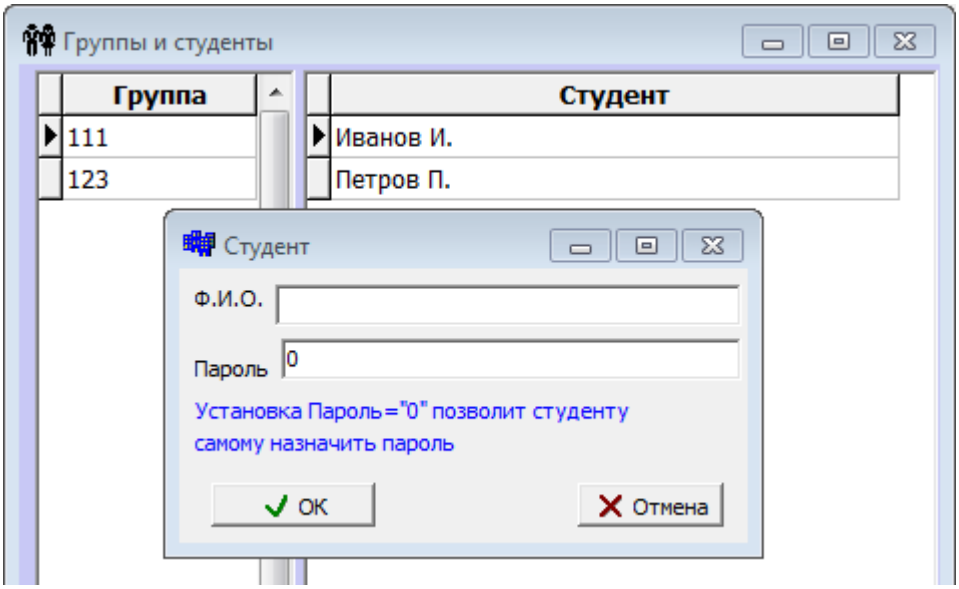

Рисунок 4.11 – Добавление студента

Команды «Изменить», «Удалить» выполняют аналогичные функции, которые были изложены выше.

4.1.4 Информация о тестовых базах данных

Слева на главной форме расположена таблица с данными о тестовых базах данных (Рисунок 4.12). Здесь можно просмотреть все базы, которые используются для составления и решения задач.

База	Имя сервера	Login	PSW
Aeroflot	MSSQL-2K8\KAT	gorbunova	fkt
Kadry	MSSQL-2K8\KAT	gorbunova	fkt
TestR1	MSSQL-2K8\KAT	gorbunova	fkt
TestR2	MSSQL-2K8\KAT	gorbunova	fkt

Рисунок 4.12 – Таблица тестовых БД

При выборе базы данных из таблицы схема и описание соответственно изменяется (Рисунок 4.13, Рисунок 4.14).

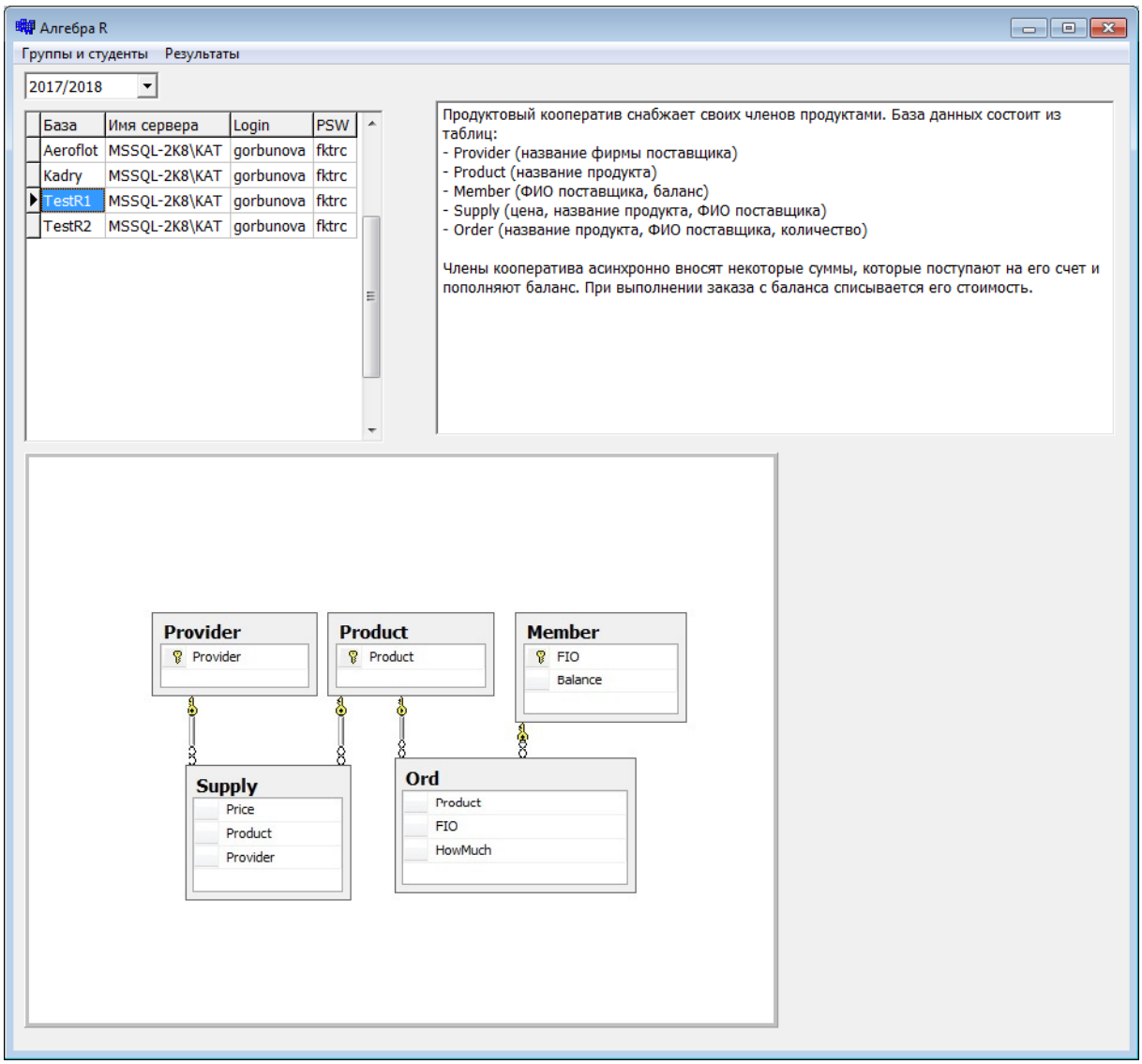

Рисунок 4.13 – БД продуктовый кооператив

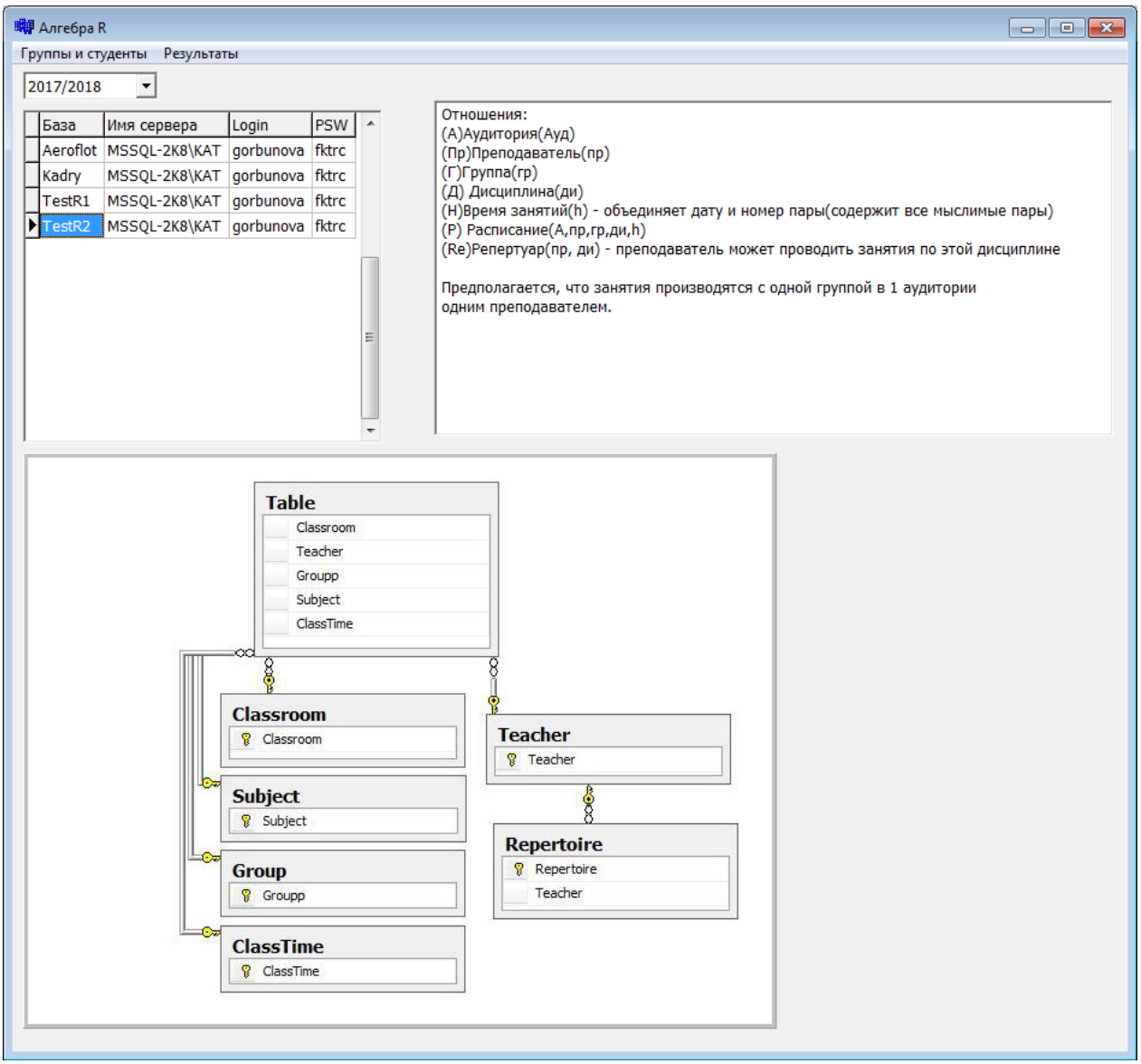

Рисунок 4.14– БД расписание занятий

Таблица содержит контекстное меню с пунктами «Добавить», «Изменить», «Удалить», «Загрузить схему БД», «Вопросы» (Рисунок 4.15).

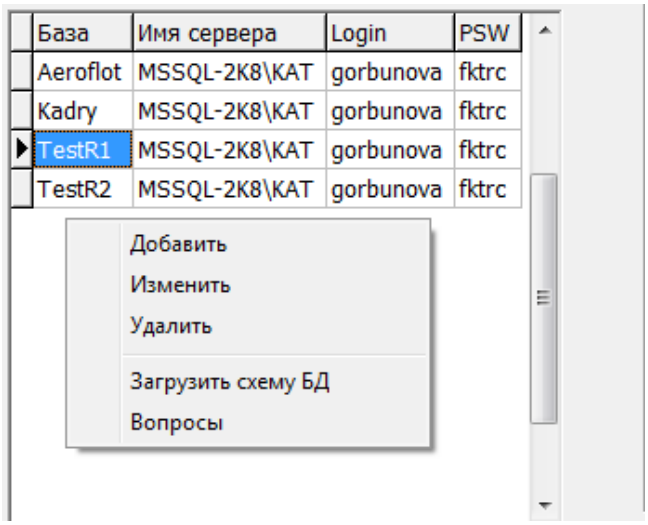

Рисунок 4.15 – Тестовые БД

Выбрав пункт меню «Добавить» появится форма добавления информации о базе данных (Рисунок 4.16). Форма содержит поля «База данных», «Сервер», «Login», «Пароль».

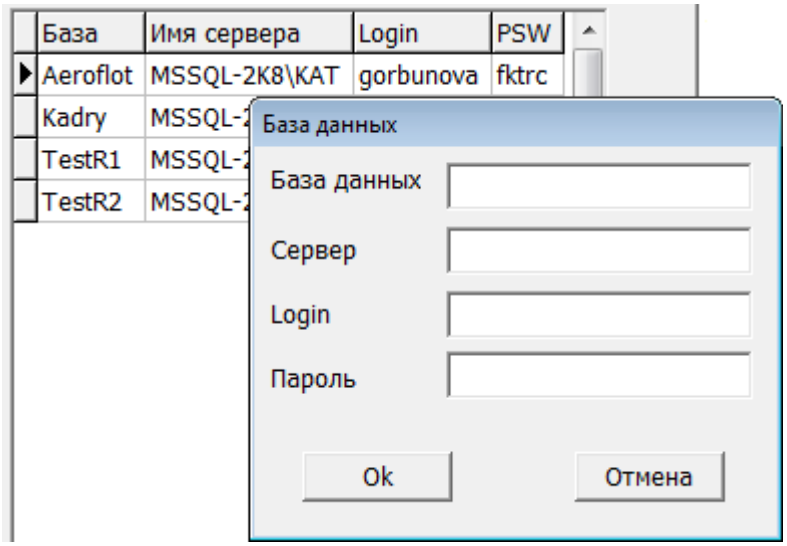

Рисунок 4.16 – Добавление базы данных

Для изменения информации в выбранной базе необходимо выбрать пункт меню «Изменить» (Рисунок 4.17). Появится форма с информацией о выбранной базе данных, в которой можно редактировать необходимые поля.

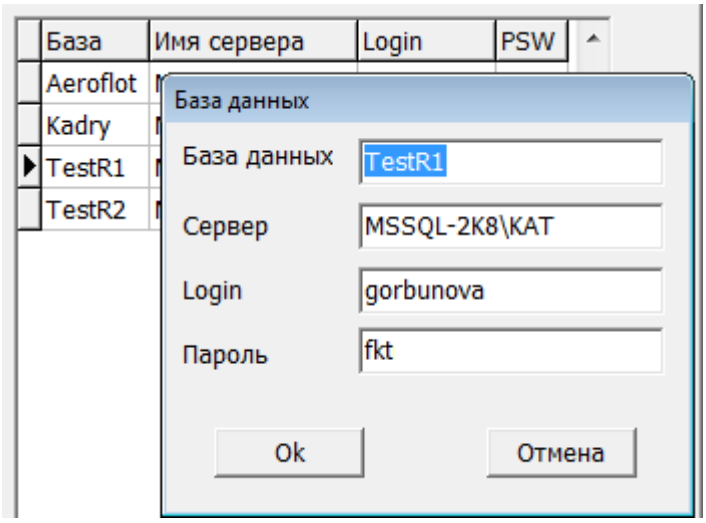

Рисунок 4.17 – Изменение базы данных

При выборе пункта меню «Загрузить схему БД» вызовется окно Windows (Рисунок 4.18). В данном окне необходимо указать путь к схеме базы данных. Формат файла имеет тип *.jpg. После подтверждения действий новая схема загрузиться в основную базу данных и будет готова для дальнейшего использования.

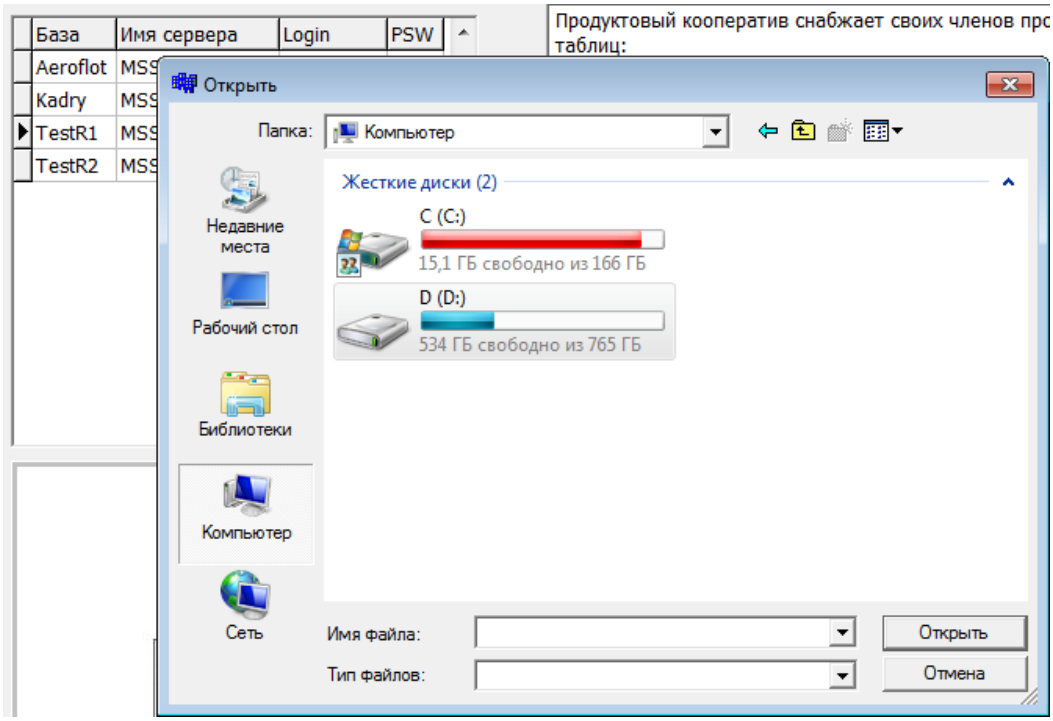

Рисунок 4.18 – Загрузка схемы БД

4.1.5 Редактирование задач

Выбрав пункт меню «Вопросы» откроется форма для редактирования задач (Рисунок 4.19).Рассмотрим элементы формы «Вопросы».

1. Выпадающий список для выбора тестовых баз данных. При выборе базы происходит изменение схемы БД и подгружаются задачи.
2. Схематичное представление выбранной базы данных.
3. Навигатор для перемещения между задачами.
4. Поля для отображения вопроса задачи.
5. Поле для отображения эталонного решения.
6. Поле для отображения SQLзапроса.
7. Таблица для отображения SQLзапроса.

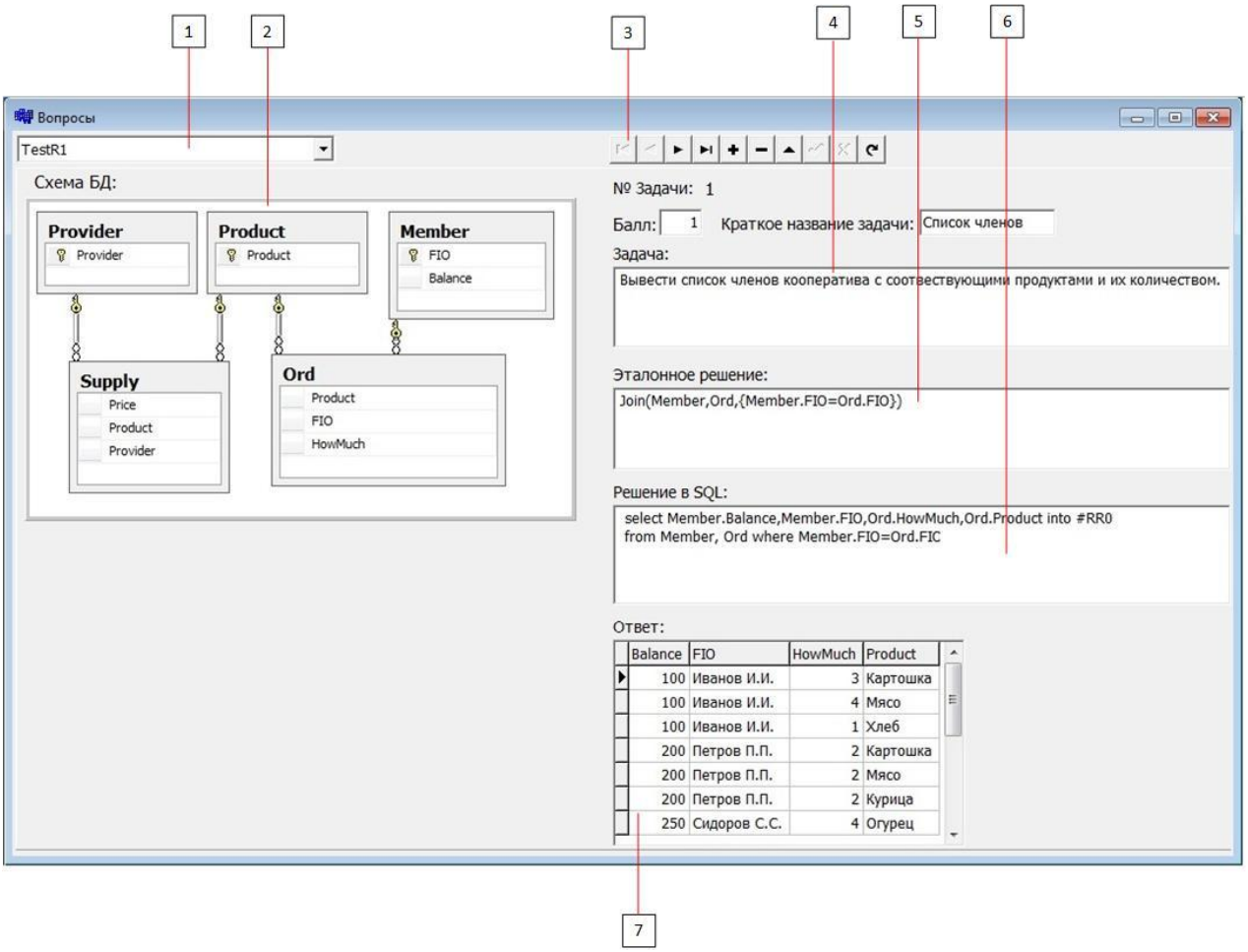

Рисунок 4.19 – Форма вопросы

4.1.6 Перемещение между задачами

Для перемещения между задачами существует навигатор (Рисунок 4.20). Функции кнопок навигатора пояснены выносками.

Рисунок 4.20 –Навигатор

4.1.7 Редактирование и просмотр задач

Номер, балл и краткое название задачи отображается ниже навигатора (Рисунок 4.21). При перемещении меду задачами информация автоматически меняется, в соответствии выбранной задачи. Для изменения, удаления и добавления задачи необходимо выбрать соответствующую команду навигатора. Для подтверждения изменения существует кнопка ✓, для отмены кнопка ✕.

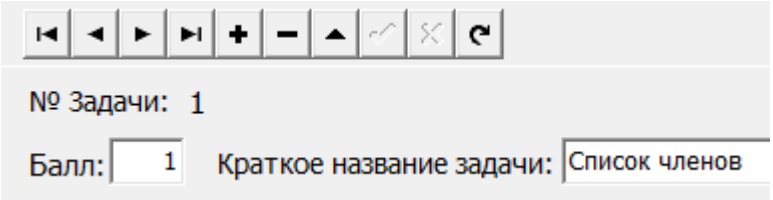

Рисунок 4.21 – Номер и балл задачи

Поле «Задача» отображает текст вопроса задачи. Поле «Эталонное решение» предназначено для ввода эталонного решения. Данное поле содержит контекстное меню (Рисунок 4.22). При выборе пункта меню «Выполнить» происходит выполнения запроса.

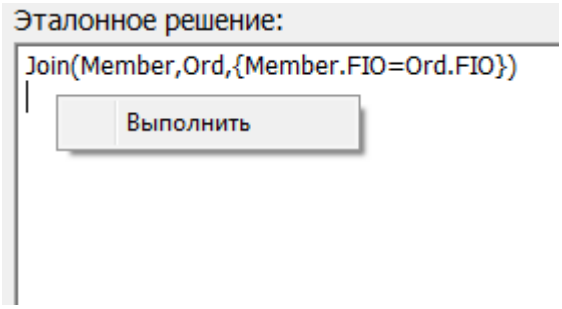

Рисунок 4.22 – Контекстное меню эталонного решения

В поле «Решение в SQL» отображается запрос на языке SQL(Рисунок 4.23).

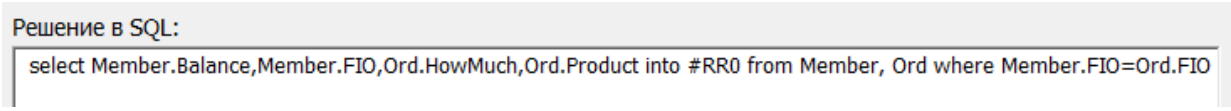

Рисунок 4.23 –SQLзапрос

В поле «Ответ» отображается в табличном виде полученные данные в результате выполнения запроса (Рисунок 4.24).

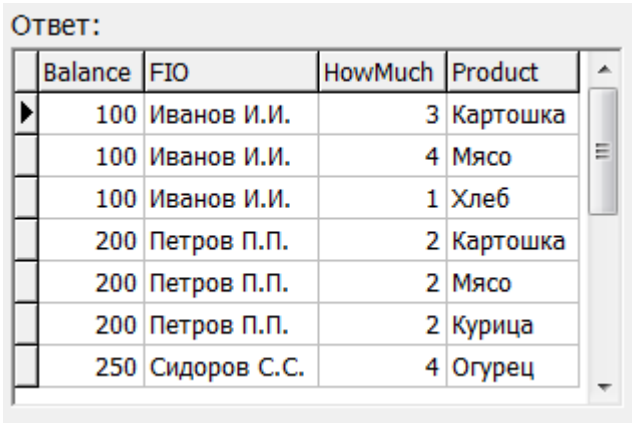

Рисунок 4.24 – Таблица ответ

4.1.8 Пункт меню «Результаты»

Главное окно приложения содержит пункт меню «Результаты» (Рисунок 4.25).

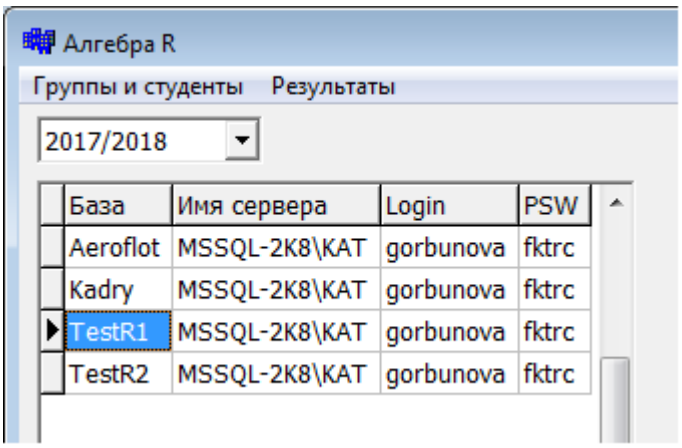

Рисунок 4.25 – Таблица ответ

При выборе пункта меню «Результаты» откроется форма (Рисунок 4.26). Рассмотрим элементы формы «Результаты».

1.Выпадающий список для выбора учебного года.

2.Выпадающий список для выбор учебной группы.

3.Таблица со списком студентов, которые учатся в выбранной группе.

4.Таблица с информацией о задачах для выбранного студента.

5.Поле отображения текста вопроса задачи.

6.Поле отображения решения задачи студентом. Содержит контекстное меню с пунктом «Выполнить».

7.Поле для отображения SQLзапроса.

8.Поле для комментария по задачи.

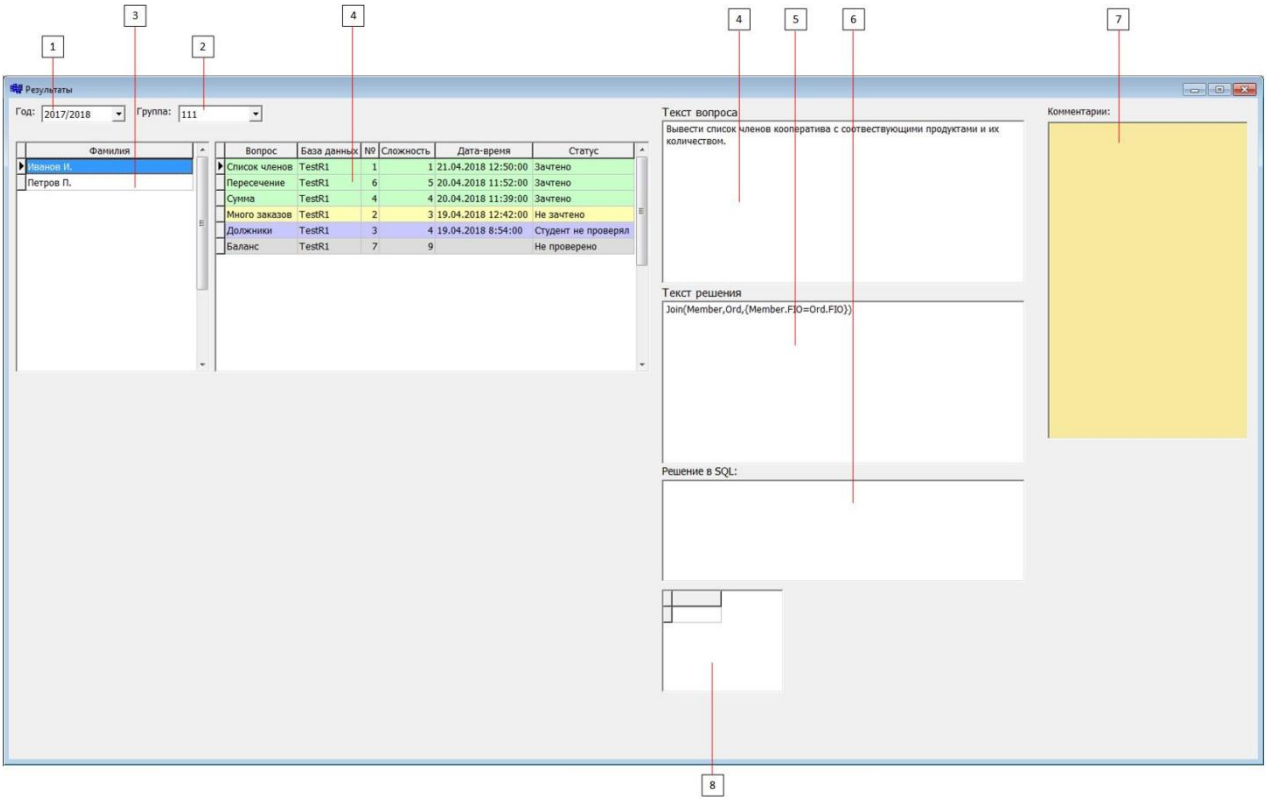

Рисунок 4.26 – Форма «Результаты»

Таблица с информацией по задачам содержит следующие столбцы:
− Вопрос - краткое описание названия задачи;
− База данных - название тестовой базы данных;
− Номер - номер вопроса;
− Сложность - сложность задачи;
− Дата и время - время отправки решения;
− Статус - информация о статусе задачи.

При нажатии правой кнопки по таблице появится контекстное меню (Рисунок 4.27). Меню имеет два пункта «Изменить статус» и «Добавить комментарий».

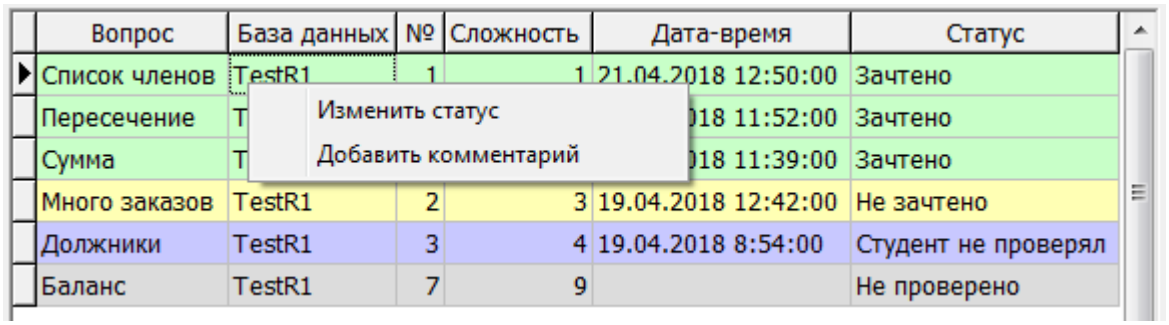

Рисунок 4.27 – Контекстное меню задач

При выборе пункта меню «Изменить статус» в верхней части формы появиться выпадающее меню. Меню содержит различные статусы (Рисунок 4.28). Статусы предназначены для пометки решения задачи студентом. Исходя из выбранного статуса студент может просмотреть правильность своего решения.

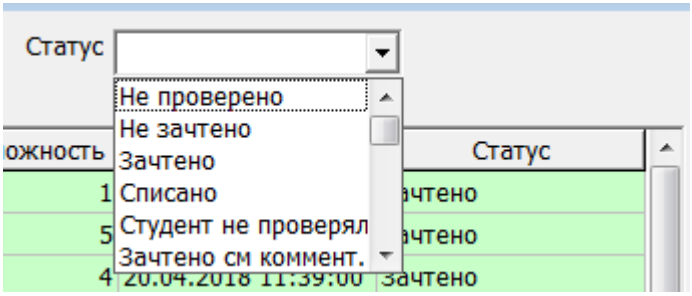

Рисунок 4.28 – Статус задачи

4.1.9 Комментарии

На форме справа расположено поле для комментариев (Рисунок 4.29). При перемещении по задачам текст сообщения изменяется, в соответствии задачи.

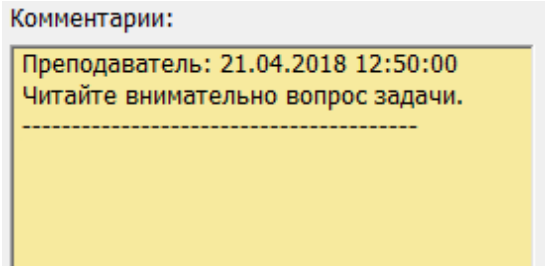

Рисунок 4.29 – Комментарии

Таблице задачи при выборе пункта контекстного меню «Добавить комментарий» появиться окно «Комментарий» (Рисунок 4.30).

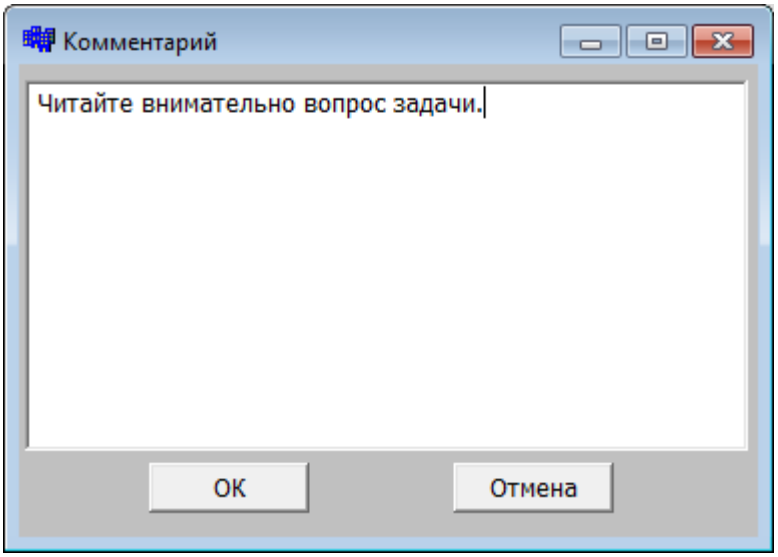

Рисунок 4.30 – Окно комментарий

При нажатии кнопки «ОК» текст комментария добавиться в основную БД и отобразиться в поле «Комментарий» для выбранной задачи. Кнопка «Отмена» отменяет действие.

4.2 Приложение студента

При запуске программы появится окно (Рисунок 4.31). Рассмотрим элементы формы «Результаты».

1.Выпадающий список с текущим учебным годом.
2.Выпадающий список с группами.
3.Выпадающий список с ФИО студентов.
4.Выпадающий список с тестовыми БД.
5.Таблица со всеми заданиями, которые необходимо решить.
6. Схема БД.
7.Поле с текстом вопроса задачи.
8.Поле с решением.
9.Таблица с результатами запроса.

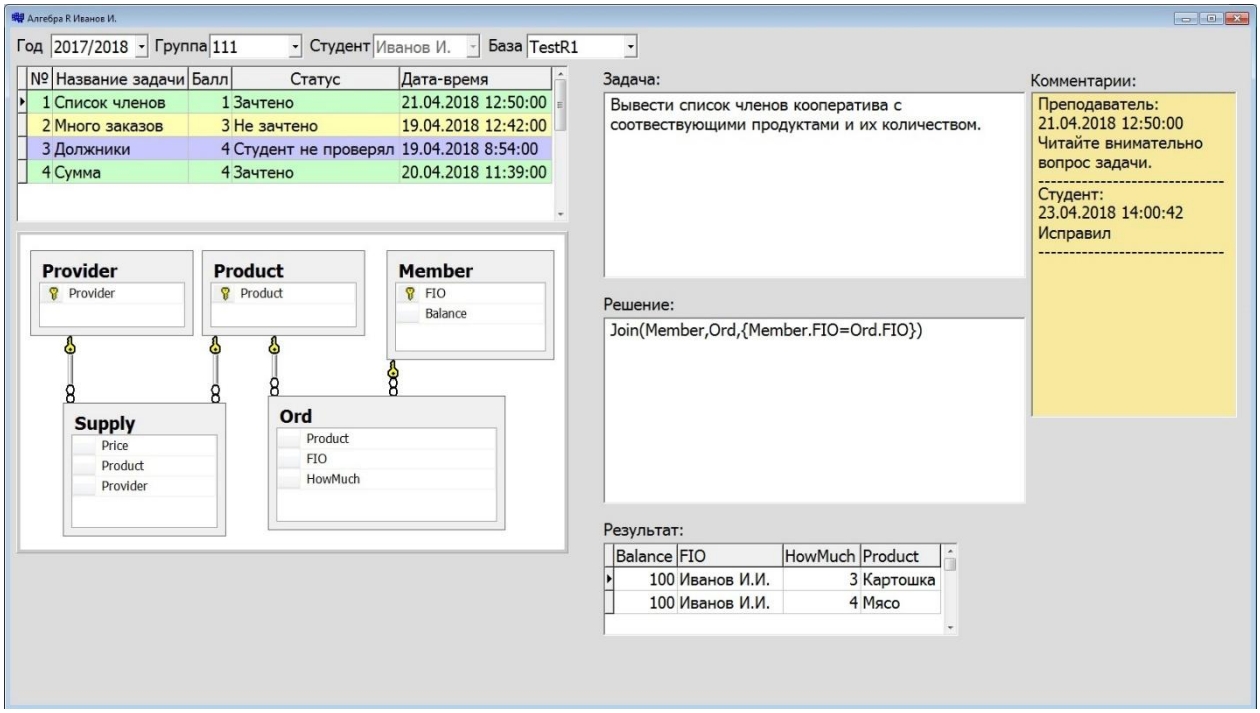

Рисунок 4.31 – Окно приложения студента

После выбора группы пользователь выбирает из выпадающего меню свои ФИО. После выбора ФИО появиться окно рисунок 4.32, в котором необходимо ввести пароль. При успешном входе станут активными таблица с информацией по задачам, схема БД, поле с комментариями.

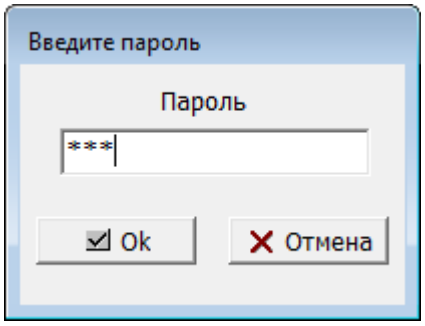

Рисунок 4.32 – Регистрация

После выбора необходимой БД в таблице с задачами отобразятся все задачи, которые содержатся в выбранной базе (Рисунок 4.33).

Рисунок 4.33 – Список задач

Таблица с информацией по задачам содержит следующие столбцы:

− Вопрос - краткое описание названия задачи;
− База данных - название тестовой базы данных;
− Номер - номер вопроса;
− Сложность - сложность задачи;
− Дата и время - время отправки решения;
− Статус - информация о статусе задачи.

Поле решение содержит контекстное меню с пунктами «Выполнить» и «Отправить решение» (Рисунок 4.34).

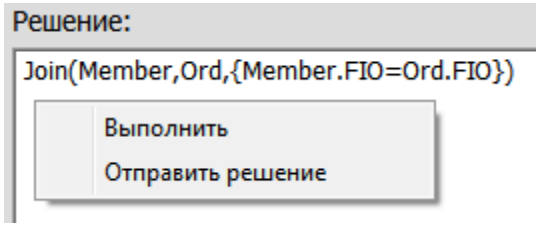

Рисунок 4.34 – Контекстное меню

При выборе пункта меню «Выполнить» произойдет обработка запроса и данные отобразятся в таблице «Результат» (Рисунок 4.35).

Результат:

Balance	FIO	HowMuch	Product
100	Иванов И.И.	3	Картошка
100	Иванов И.И.	4	Мясо
100	Иванов И.И.	1	Хлеб
200	Петров П.П.	2	Картошка

Рисунок 4.35 – Результат

При выборе пункта меню «Отправить решение» произойдет отправка решения на сервер и запись в БД.

На форме справа расположено поле для комментариев (Рисунок 4.36). При перемещении по задачам текст сообщения изменяется, в соответствии задачи.

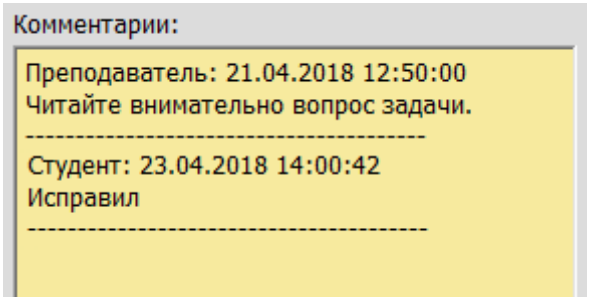

Рисунок 4.36 – Комментарии

При выборе пункта контекстного меню «Добавить комментарий» на таблице задачи появиться окно «Комментарий» (Рисунок 4.37).

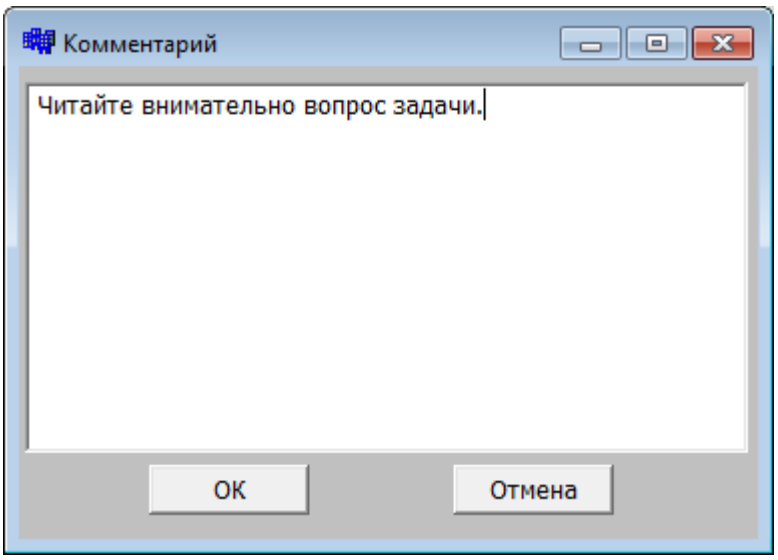

Рисунок 4.37 – Окно комментарий

При нажатии кнопки «ОК» текст комментария добавиться в основную БД и отобразиться в поле «Комментарий» для выбранной задачи. Кнопка «Отмена» отменяет действие.

Выводы по главе

В данной главе приведено подробное описание структуры приложения преподавателя и студента. Продемонстрированы на конкретных примерах функциональные возможности. Программный продукт имеет интуитивно понятный интерфейс.

ЗАКЛЮЧЕНИЕ

В данной работе были проанализированы требования к приложению, проведен обзор существующих решений, описаны их достоинства и недостатки. Был выполнен сравнительный анализ рассмотренных средств и разработанного приложения, на основе чего были выявлены его основные функциональные преимущества.

В результате работы над проектом:

Разработан программный комплекс, который состоит из основной базы данных и ряда тестовых, а также двух приложений-клиентов:

− приложение преподавателя;

− приложение студента.

Описана архитектура системы и ее интерфейс, приведены схемы основных алгоритмов, обеспечивающих корректное функционирование системы. Для тестирования и отладки были разработаны задачи и отладочная база данных.

В результате студент приобретает возможности:

− решать задачи как в аудитории, так и в любом месте, где есть доступ к сети кафедры;

− обмениваться вопросами, ответами и замечаниями с преподавателем вне личного контакта;

− получать диагностику ошибок;

− видеть весь набор задач и выбирать личную последовательность их решения.

Преподаватель приобретает возможность:

− готовить упражнения по курсу;

− просматривать и анализировать студенческие решения в любое время и в любом месте;

− оказать помощь студенту в решении задач вне личного контакта;

− в любой данный момент видеть успехи группы.

Для каждого оператора задано функциональное представление и приоритеты. На примере рассмотрено преобразование выражение в обратную польскую запись. Подробно описаны последовательность преобразования функциональной формы записи в SQL запрос. Процесс выполнения текста решения состоит из 3 фаз:

1) Лексический анализ-выделение операций, отношений, списков полей, логических выражений и скобок.

2) Преобразование в ПОЛИЗ.

3) Преобразование ПОЛИЗ в последовательность SQL операторов.

4) Выполнение полученных SQL операторов.

Разного рода ошибки в решении выявляются как приложением, так и SQL-сервером. Синтаксические ошибки в реляционных выражениях выявляются приложениями, а синтаксические ошибки SQL операторов выявляются SQL сервером и сообщаются пользователю. Приведены основные ошибки и их обработка.

Для обеспечения безопасности предусмотрены полномочия.

Роль «Преподаватель»:

− имеет право выполнять операции insert, update, delete над любыми таблиц как основной БД, так и тестовых БД.

− не имеет прав на какие бы то ни было изменения структур БД, их таблиц, программных текстов на стороне сервера

Все студенты выполняют соединение с БД через один и тот же логин и пароль известный приложению. Право входа в приложение студента контролируется таблицей, где для каждого студента указан его зашифрованный пароль.

Абсолютными правами по отношению ко всем БД обладает администратор БД.

В результате было разработано программное обеспечение для решения задач по теме «Реляционная алгебра», отвечающее всем указанным требованиям. Таким образом, все поставленные задачи были успешно выполнены.

Возможно расширение функциональности за счет изменений в структуре базы данных, повышение удобства работы с приложением за счет изменения разметки и визуального оформления. Также возможно усовершенствование контрольной части путем отображение сводных данных по задачам, по посещаемости, по процессу выполнения всех заданий, по количеству или суммарной сложности выполнения задач.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1 Бураков, П.В., Петров, В.Ю. Введение в системы баз данных: Учебно пособие / П.В.Бураков, В.Ю.Петров: [Электронный ресурс], URL: http://www.ict.edu.ru/ft/006173/itmo461.pdf (дата обращения 25.02.2018)

2 Relation Algebra Exercises | DB4 Courseware | Stanford Lagunita: [Электронный ресурс], URL:https://lagunita.stanford.edu/courses/DB/RA/SelfPaced/courseware/chrelational_algebra/seqexercisera/ (дата обращения 21.02.2018).

3 Минкин, И.В. Система автоматической проверки лабораторных работ по праграммированию: [Электронный ресурс], URL: http://conf.sfukras.ru/sites/mn2010/section6.html (дата обращения 23.02.2018).

4 Подлесных, Д.А., Булавас, В.В. Системадля лабораторных работ по информатике с автоматизацией проверки: [Электронный ресурс], URL: http://conf59.mipt.ru/static/reports_pdf/2922.pdf (дата обращения 23.02.2018)

5 Скоробогатов, С.Ю. Автоматизированная системадля проведения практических занятий по программированию [Электронный ресурс], URL: http://engjournal.ru/catalog/pedagogika/hidden/1330.html (дата обращения 23.02.2018).

6 URL: https://lagunita.stanford.edu/courses (дата обращения 23.02.2018)

7 Relation Algebra Exercises | DB4 Courseware | Stanford Lagunita: [Электронный ресурс], URL:https://lagunita.stanford.edu/courses/DB/RA/SelfPaced/courseware/chrelational_algebra/seq-exercise-ra/ (дата обращения 21.02.2018).

8 RA: A Relation Algebra Interpreter: [Электронный ресурс], URL: https://users.cs.duke.edu/~junyang/ra/ (дата обращения 21.04.2018).

9 Онищенко, А.С. Тренажер «Операции реляционной алгебры» с элементами электронного пособия / А.С. Онищенко. –Каф.инф.систем и технологий. – Екатеринбург, 2016. – 52 с.

10 Telepovska, H. Support of Relational Algebra Knowledge Assessment/ Emerging Trends in Computing, Informatics, Systems Sciences, and Engineering, Lecture Notes in Electrical Engineering: . –2013. –№151. –p.475-485.

11 Soler, J. An Automatic Correction Tool for Relational Algebra Queries / J.Soler, I.Boada, F.Prados, J.Poch, R.Fabregat: [Электронный ресурс], URL: https://pdfs.semanticsholar.org/605c/7d3dda2f39f56d1cd7dc31d53be8f88703c1.pdf (дата обращения 25.04.2018).

12 Дейт, К. Дж. Введение в системы баз данных / К. Дж. Дайт. – М.: Вильямс, 2005. – 1328 с.

13 Ben-Gan, I. Inside Microsoft SQL Server 2008 T-SQL Programming / I.Ben-Gan . –Microsoft Press, 2010. – 283 p.

14 Mistry, R. Introducing Microsoft SQL Server 2008 R2 / R.Mistry. – Microsoft Press, 2010. – 215 p.

15 Мейер, Д. Теория реляционных баз данных / Д. Мейер. – М. Мир, 1987. – 608 с.

16 Касаткин А.И., Вальвачев А.Н. От Turbo C к Borland C++/ Минск: Галерея Петрополь, 1992

17 Fritchey, G. SQL Server Execution Plans / G.Fritchey. – Simple Talk Publishing, 2012. – 321 p.

18 Виейра, Р. Программирование баз данных Microsoft SQL Server 2005 для профессионалов / Р.Виейра. – М.: Диалектика, 2008. – 1072 с.

19 Vieira, R. Professionsl SQL Server 2005 programming / R.Vieira. – Willey, 2008. – 1072 p.

20 Вирт, Н. Алгоритмы и структуры данных: Пер. с англ. / Н.Вирт. – М.: ДМК Пресс, 2010. – 272 с.

21 Ахо, А. В. Структуры данных и алгоритмы / А.В. Ахо. – М.: Вильям, 2000. – 382 с.

22 Cormen, T. Introduction to Algorithms / T.Cormen. – Massachusetts Instituts of Technology,2009. – 1292 p.

23 Кузнецов, С. Базы данных. Вводный курс. / С.Кузнецов, [Электронный ресурс], URL: http://citforum.ru/database/advanced_intro/13.shtml (дата обращения 21.02.2018)

24 Страуструп, Б. Язык программирования С++. Часть первая. / Б.Страуструп. – Киев: ДиаСофт, 1993. – 264 с.

25 Страуструп Б. Язык программирования С++. Часть вторая. / Б.Страуструп. – Киев: ДиаСофт, 1993. – 296 с.

26 Aho, A. V. Data structures and algorithms / A.V. Aho. – Addison-wesley publishing company, 2000. – 382 p.

27 Бакнелл, Д. Фундаментальные алгоритмы и структуры данных в Delphi /Д. Бакнелл. – ДифСофтЮП, 2003. – 556 с.

28 Vieira, R. Professionsl SQL Server 2005 programming / R.Vieira. – Willey, 2008. – 1072 p.

29 Hopcroft, J. Introduction to Automata Theory, Languages, and Computation / J.Hopcroft. – Addison-Wesley, 2002. – 528 с.

30 Карпов, В.Э. Теория компиляторов: Учебное пособие. 2-е изд. / В.Э. Карпов. – М., 2010. – 91 с.

31 Савихин, О.Г. Структуры данных: Учебное пособие. / О.Г. Савихин. –Н.Н. Нижегородский госуниверситет, 2007. – 45 с.

32 Эллис, М., Страуструп, Б. Справочное руководство по языку программированию С++ / Эллис М., Страуструп Б. – М: Изд.Мир, 1992

33 Ирэ П. Объектно-ориентированное программирование с использованием С++ / П.Ирэ . – Киев:НИПФ ДиаСофт Лтд.,1995. – 480 с.

34 Lewis II, P. Compiler Design Theory / P. Lewis II. – Addison-Wesley, 1976. – 655 p.

35 Льюис, Ф.Теоретические основы проектирования компиляторов / Ф. Льюис. – М. Мир, 1976. – 655 с.

36 Ишаков, Е.Н. Теория языков программирования и методов трансляции: Учебное пособие / Е.Н.Ишаков. – Оренбург: ГОУ ОГУ, 2007. – 137с.

37 Henderson, K. The Guru's Guide to Transact-SQL / K. Henderson. – Addison-Wesley, 2005. – 558p.

38 Ахо, А.В. Компиляторы: принципы, технологии и инструментарий, 2-е изд. / А.В. Ахо, Р. Сети, Дж. Д. Ульман. – М.: Вильямс, 2008. –1184 с.

39 Горохова-Алексеева, А.В. Подход к автоматизации проверки заданий в области компьютерной графики / А.В. Горохова-Алексеева, Д.А. Королев // Системный администратор. – 2016. –№07-08. – с.138–140.

40 Дейт, К. Дж. Введение в системы баз данных / К. Дж. Дайт. – М.: Вильямс, 2005. – 1328 с.

41 Карпова, И.П. Базы данных: Учебное пособие для ВУЗов / И.П. Карпова. – СПб.: Питер, 2013. –240 с.

42 Кнут, Д. Искусство программирования. Основные алгоритмы. / Д. Кнут. – М.: Вильямс, 2005. – 720 с.

43 Binning, C. Reverse query processing / C. Binning, D. Kossmann, E. Lo: [Электронный ресурс],URL:https://www1.ethz.ch/dbis/research/publications/rqpi-cde07.pdf (дата обращения 10.05.2018).

44 Databene Benerator: [Электронный ресурс], URL: https://databene.org/databene-benerator (дата обращения 03.02.2018).

45 Khalek, S.A. Query-aware Test Generation Using a Relational Constraint Solve / S.A. Khalek, B. Elkarablieh, Y. O. Laleye and Khurshid: [Электронный ресурс], URL: http://users.ece.utexas.edu/~khurshid/papers/2008/08ase-query-aware.pdf (дата обращения 13.03.2018).

46 Mockaroo – Random Data Generator | CSV/SQL/Exel: [Электронный ресурс], URL: https://www.mockaroo.com/ (дата обращения 21.04.2018).

47 Telepovska, H. Support of Database Skills Testing: [Электронный ресурс], URL: https://pdfs.semanticscholar.org/dbc7/a0273d410af05e38dbc403e4923b44df-9459.pdf (дата обращения 25.02.2018)

48 Test Data Generator for database population and testing purposes with optional SQL, text, XML output: [Электронный ресурс], URL: http://www.sqledit.com/dg/index.html (дата обращения 25.02.2018)

49 Bucknall, J. The Tomes of Delphi Algorithms and Data Structures / J.Bucknall. – Wordware Publishing, 2003. – 556 p.

50 Хопкрофт, Д. Введение в теорию автоматов, языков и вычислений / Д. Хопкрофт. – М.: Вильямс, 2002. – 528 с.

51 Хендерсон, К. Профессиональное руководство по Transact-SQL / К. Хендерсон. – К.Питер, 2005. – 558 с.

ПРИЛОЖЕНИЕ 1

Текст программы интерпретатора

Файл «Expression_R.cpp»

```
#include "Msg.h"
#include "tools.h"
#include "tools1.h"
#include "Apple.h"
#include "Expression_R.h"

extern int CurBase_ID;
static TADOConnection *Conn;
static EXPR_BASE *exb;
AnsiString rezSQL;
//---------------------------------------------
AnsiString Join(STACK *st, int v){
AnsiString s,FieldList,Table1,Table2;
s="select ";
Table1=exb->AddSharp(st[v-2].s);
Table2=exb->AddSharp(st[v-1].s);
FieldList=MakeFieldListForTwoTables(Conn, Table1, Table2);
FieldList=exb->AddSharpForFields(FieldList);
s=s+FieldList;
// имена всех промежуточных результатов начинаются с #RR
AnsiString Where=exb->AddSharpForFields(st[v].s);
s=s+" into #RR"+exb->rNum+" from "+Table1+", "+Table2+" where "+Where;
st[v].Tag=FIELDS;
st[v-1].Tag=TABLE;
return s;
}
//---------------------------------------------
AnsiString Where(STACK *st, int v){
// операция СЕЛЕКЦИЯ
AnsiString s,FieldList, TableName, where;
s="select * ";
TableName=exb->AddSharp(st[v-1].s);
where=exb->AddSharpForFields(st[v].s);
s=s+" into #RR"+exb->rNum+" from "+TableName+" where "+where;
st[v].Tag=FIELDS;
st[v-1].Tag=TABLE;
return s;
}
//---------------------------------------------
AnsiString Cross(STACK *st, int v){
AnsiString s,FieldList, Table1, Table2;
Table1=exb->AddSharp(st[v-1].s);
Table2=exb->AddSharp(st[v].s);
s="select ";
FieldList=MakeFieldListForTwoTables(Conn, Table1, Table2);
s=s+exb->AddSharpForFields(FieldList);
s=s+" into #RR"+exb->rNum+" from "+Table1+","+Table2;
st[v].Tag=FIELDS;
st[v-1].Tag=TABLE;
return s;
}
//---------------------------------------------
AnsiString Except(STACK *st, int v){
AnsiString s, Table1,Table2;
Table1=exb->AddSharp(st[v-1].s);
Table2=exb->AddSharp(st[v].s);
s="select * into #RR";
s=s+exb->rNum+" from (select * from ";
```

```
s=s+Table1+" except select * from "+Table2+") TT";
return s;
}
//---------------------------------------------
AnsiString Project(STACK *st, int v){
AnsiString s, Table, FieldList;
Table=exb->AddSharp(st[v-1].s);
FieldList=exb->AddSharpForFields(st[v].s);
s="select ";
s=s+FieldList+" into #RR"+exb->rNum+" from "+Table;
st[v].Tag=FIELDS;
st[v-1].Tag=TABLE;
return s;
}
//---------------------------------------------
AnsiString Union(STACK *st, int v){
AnsiString s,Table1,Table2;
Table1=exb->AddSharp(st[v-1].s);
Table2=exb->AddSharp(st[v].s);
s="select * into #RR";
s=s+exb->rNum+" from (select * from ";
s=s+Table1+" union select * from "+Table2+") TT";
return s;
}
//---------------------------------------------
AnsiString Equal(STACK *st, int v){
AnsiString s, TableFrom, TableTo;
TableTo=exb->AddSharp(st[v-1].s);
DropTableFromTempDb(exb->TestBaseConn,TableTo);  // в tools1.cpp
TableFrom=exb->AddSharp(st[v].s);
s="select * into ";
s=s+TableTo+" from "+TableFrom;
return s;
}
//---------------------------------------------
AnsiString Rename(STACK *st, int v){
}
//---------------------------------------------
AnsiString Intersect(STACK *st, int v){
AnsiString s, Table1, Table2;
Table1=exb->AddSharp(st[v-1].s);
Table2=exb->AddSharp(st[v].s);
s="select * into #RR";
s=s+exb->rNum+" from (select * from ";
s=s+Table1+" intersect select * from "+Table2+") TT";
return s;
}
//---------------------------------------------
static FF ffs[]={
   {"Join", Join,2,3},
   {"Project",Project,3, 2},
   {"Where",Where,3,2},
   {"Cross",Cross,2,2},
   {"Except",Except,1,2},
   {"Union",Union,1,2},
   {"Intersect",Intersect,2,2},
   {"Rename",Rename,4,2},
   {"=", Equal,0,2}
};
//---------------------------------------------
EXPRESSION_R::EXPRESSION_R(AnsiString s,EXPR_BASE *ExB){
this->ff=ffs;
```

```
    this->nff=sizeof(ffs)/sizeof(FF);
    this->InFix=s;
    this->PolizLen=0;
    this->npp=0;

    InBraces=false;
    BracketLevel=0;
    pp=NULL;
    rConn=ExB->TestBaseConn;
    this->ExBase=ExB;
    Conn=rConn; // Conn - глобально в файле
    exb=ExB; // exb глобально в файле
    }
    //---------------------------------------------------
    EXPRESSION_R::~EXPRESSION_R(){
    TERM *p;
    for(p=this->Head->Next; p!=NULL; p=Head->Next){
       Head->Next=p->Next;
       //(p->Tag!=FUNCTION){ // значит строка
       delete [] p->t;
       //
       delete p;
    }
    if(Head->Tag!=FUNCTION){ // значит строка
       delete [] Head->t;
    }
    if(PolizLen>0){
       delete [] pp;
    }

    if(rConn){
       rConn->Connected=false;
       delete rConn;
    }
    }
    //---------------------------------------------------
    AnsiString EXPRESSION_R::GetTerm(int &k){
    AnsiString r="";

    char *p=this->InFix.c_str()+k;
     // убираем ведущие пробелы и запятые вне фигурных скобок
    while(*p!='\0' && !isgraph(*p) || (!InBraces && *p==',')){
       k++;
       p++;
    }
    if(*p=='\0') return "";

    // односимвольные терминалы
    if(*p=='(' || *p==')' || *p=='{' || *p=='}' || *p=='\0' || *p=='='){
       r=*p;
       p++;
       k++;
    } else {
       // многобуквенный терм
       for(p=this->InFix.c_str()+k; isdigit(*p) || isalpha(*p) || (InBraces &&
isgraph(*p) && *p!='}'); p++){
             r=r+(*p);
             k++;
       }
    }
    return r;
    }
```

```
//---------------------------------------------------
void EXPRESSION_R::MakeInList(){
int k=0,i;
AnsiString Term;
TERM *Last, *New;
bool IsFunction;
char *p;
this->Head=new TERM;
Head->Tag=HEAD;
Head->t=new char[2];
strcpy(Head->t,"H");
Last=Head;
for(;;){
   Term=GetTerm(k);
   if(Term.IsEmpty()){
         break;
   }
   p=Term.c_str();
   if(*p=='{'){
         InBraces=true;
         continue;
   }
   if(*p=='}'){
         InBraces=false;
         continue;
   }
   New=new TERM;
   New->Next=NULL;
   Last->Next=New;
   Last=New;
   // однобуквенный?
   if(*p=='(' || *p==')' /*|| *p=='=' */) {
         New->Tag=*p;
         New->t=new char[2];
         New->t[0]=*p;
         New->t[1]='\0';
         continue;
   }
   // это функция?
   IsFunction=false;
   for(i=0;i<nff; i++){
         if(Term.AnsiCompareIC(ff[i].FuncName)==0){
               IsFunction=true;
               break;
         }
   }
   if(IsFunction){
         New->FunctionNumb=i;
         New->t=new char [strlen(ff[i].FuncName)];
         strcpy(New->t,ff[i].FuncName);
         New->Tag=FUNCTION;
         continue;
   }

   // осталось 3 варианта - RELATION, BOOLEAN, FIELDS
   // пока будет DONTKNOW
   New->Tag=DONTKNOW;
   New->t=new char[Term.Length()+1];
   strcpy(New->t,Term.c_str());
}
return;
}
```

```
//---------------------------------------------------
void EXPRESSION_R::PrintList(FILE *f){
TERM *p;
AnsiString s;
for(p=this->Head; p!=NULL; p=p->Next){
   s="";
   if(p->Tag==FUNCTION){
         s=s+"FUNCTION "+ff[p->FunctionNumb].FuncName;
   } else {
         s=s+p->Tag+"  "+p->t;
   }
   fprintf(f,"%s\n",s.c_str());
}
}
//---------------------------------------------------
int EXPRESSION_R::Poliz(){
// построение обратной польской записи
// Польская запись будет состоять из массива указателей
// на узлы построенного списка

// посчитаем число элементов в ПОЛИЗ
int i,StackPtr=-1, pr1,pr2;
bool found;
TERM *p,*q;
for(p=Head->Next; p!=NULL; p=p->Next){
   npp++; // длина списка
}
pp=new TERM*[npp]; // массив для польской записи
PolizLen=0;
for(p=Head->Next; p!=NULL; p=p->Next){
   switch(p->Tag){
         case '(':
         case '=':
               // в стек
               OperationStack[++StackPtr]=p;
               break;
         case FUNCTION:
               // всё с большим или равным приоритетом - в выходную строку
               pr1=ff[p->FunctionNumb].Priority; // приоритет входной операции
               // выталкиваем из стека все операции с приоритетом >= приоритета
входной операции
               for(;;){
                     if(StackPtr<0) break;
                     q=OperationStack[StackPtr];
                     pr2=q->Tag=='(' ? (-1) : ff[q->FunctionNumb].Priority; // у
'(' приоритет ниже всего
                     if(pr2 >= pr1){
                           pp[PolizLen++]=p;
                           StackPtr--;
                     } else {
                           break;
                     }
               }
               // очередную операцию - в стек
               OperationStack[++StackPtr]=p;
               break;
         case ')':
               // всё до '(' переписать из стека в выходную строку
               found=false;
               while(StackPtr>=0){
                     if(OperationStack[StackPtr]->Tag=='('){
                           found=true;
```

```
                        StackPtr--;
                        break;
                  } else {
                        pp[PolizLen++]=OperationStack[StackPtr--];
                  }
            }
            if(!found){ // непарная открывающая скобка
                  return EXTRA_OS;
            }
            break;
      default: // операнд
            pp[PolizLen++]=p;
            break;
   } //switch
} // for (p=
//остаток стека - на выход
while(StackPtr>=0){
   if(OperationStack[StackPtr]->Tag==')'){
      // непарная закрывающая скобка
      return EXTRA_ZS;
   }
   pp[PolizLen++]=OperationStack[StackPtr--];
}
this->PolizString=this->getPolizString();
return SUCCESS;
}
//---------------------------------------------
AnsiString EXPRESSION_R::getPolizString(){
AnsiString s="";
for(int i=0; i<this->PolizLen; i++){
   if(pp[i]->Tag==FUNCTION){
      s=s+"FUNCTION  "+pp[i]->FunctionNumb+"  "+pp[i]->t;
   } else {
      s=s+pp[i]->Tag+"  "+pp[i]->t;
   }
   s=s+"| ";
}
return s;
}
//---------------------------------------------
void EXPRESSION_R::PrintPoliz(FILE *f){
AnsiString s;
for(int i=0; i<this->PolizLen; i++){
   s="";
   if(pp[i]->Tag==FUNCTION){
      s=s+"FUNCTION  "+pp[i]->FunctionNumb+"  "+pp[i]->t;
   } else {
      s=s+pp[i]->Tag+"  "+pp[i]->t;
   }
   fprintf(f, "%s\n",s.c_str());
}
}
//-----------------------------------------
AnsiString EXPRESSION_R::PerformPoliz(){
int v=-1, i, fNumb;
AnsiString Sql="", s;
TADOQuery *q;
for(i=0; i<this->PolizLen; i++){
   switch(pp[i]->Tag){
      case FUNCTION:
            fNumb=pp[i]->FunctionNumb;
            Sql=(*(ff[fNumb].f))(this->Stack, v);
```

```
                q=CreateQuery(rConn,Sql,0);
                rezSQL=rezSQL+" "+Sql;
                try {
                      q->ExecSQL();
                }
                catch(Exception &e){
                      s="Ошибка в SQL-операторе\n";
                      s=s+Sql;
                      Msg(USER_ERROR,NULL,s.c_str());
                      return "";
                }
                delete q;
          // если это была операция присваивания (Equal)
          // то в стек операндов нужно поместить имя таблицы из левой части
          // в противном случае - имя промежуточного результата
                if(stricmp(ff[fNumb].FuncName,"=")==0){
                      s=this->ExBase->AddSharp(Stack[v-1].s);
                } else {
                      s="#RR";
                      s=s+ExBase->rNum;
                      ExBase->rNum++;
                }
                v-=ff[fNumb].nPlace-1;
                Stack[v].s=s;
                Stack[v].Tag=TABLE;
                break;
          case RELATION:
          case FIELDS:
          case BOOLEAN:
          case DONTKNOW:
                v++;
                Stack[v].s=pp[i]->t;
                Stack[v].Tag=pp[i]->Tag;
                break;
          default: // ошибка
                Msg(USER_ERROR,NULL,"Ошибка в ПОЛИЗ");
    }
}
Sql="select * from ";
Sql=Sql+Stack[0].s;
return Sql;
}
//-----------------------------------------------
AnsiString EXPRESSION_R::getResultQuery(){
MakeInList();
Poliz();
AnsiString s=PerformPoliz();
return s;
}
```

Файл «ExpBase.cpp»

```
#include "ExprBase.h"
#include "tools.h"
#include "apple.h"
#include "ctype.h"
#include "stdlib.h"
//-------------------------------------
EXPR_BASE::EXPR_BASE(int TestBase_ID){
TestBaseConn=MakeConnection(TestBase_ID);
TestBaseConn->Open();
// TableList - массив имён таблиц в БД
TableList=getTableNames(this->TestBaseConn, this->nTable);
```

```
    this->rNum=0;
    }
    //-------------------------------------
    EXPR_BASE::~EXPR_BASE(){
    delete TestBaseConn;
    delete [] TableList;
    }
    //-------------------------------------
    AnsiString EXPR_BASE::AddSharp(AnsiString TableName){
    /* функция добавляет ведущий '#' к TableName если:
    его ещё нет и TableName не содержится в TableList
    */
    if(TableName.IsEmpty()){
       return "";
    }
    AnsiString s=TableName;
    bool HaveSharp=s.c_str()[0]=='#';
    if(!HaveSharp){
       bool isInTableList=false;
       for(int i=0; i<this->nTable; i++){
             if(TableList[i].AnsiCompareIC(TableName)==0){
                   isInTableList=true;
                   break;
             }
       }
       if(!isInTableList){
             s=AnsiString("#")+s;
       }
    }
    return s;
    }
    //-----------------------------------------------------
    #define CAN 0  // символ может входить в имя таблицы
    #define NOT 1 // не может
    #define DOT 2 // символ=='.'
    #define EOL 3 // '\0' - конец строки
    static int SymbClass(char c){
    if(c=='.')return DOT;
    if(c=='\0') return EOL;
    if(c=='#' || c=='_' || c>='a' && c<='z' || c>='A' && c<='Z' || isdigit(c))
return CAN;
    return NOT;
    }
    AnsiString EXPR_BASE::AddSharpForFields(AnsiString ss){
    /* добавить # к именам таблиц (если требуется)
    в строках, содежащих элементы вида <имя таблицы>.<имя поля>
    */
    int State=1; // текущее состояние конечного автомата
    int Class; // класс очередного входного символа
    int j;
    AnsiString Name=""; // собираем здесь возможное имя таблицы
    AnsiString Out=""; // строка - результат
    int i=0; // индекс очередного символа из входной строки
    char *p=ss.c_str(); // входная строка в виде char *
    bool first=true; // не вижу другого способа обработать символ конца строки '\0'
    bool Good=true; // входная строка не имеет ошибок
    do{
       if(first){
             first=false;
       } else {
             i++;
       }
```

```
Class=SymbClass(p[i]);
switch(State){
      case 1:
            switch(Class){
                  case CAN:
                        // возможно это начало имени таблицы
                        Name=p[i];
                        State=2;
                        break;
                  case NOT:
                        // символ не может принадлежать имени таблицы
                        Out=Out+p[i];
                        Name="";
                        State=1;
                        break;
                  case DOT:
                        // такого быть не должно (не считая чисел 1.23)
                        Good=false;
                        State=3;
                        break;
                  case EOL:
                        // строка кончилась
                        Out=Out+Name;
                        break;
                  default:
                        Good=false;
                        State=3;
                        break;
            }
            break;
      case 2:
            switch(Class){
                  case CAN:
                        // очередной символ возможного имени таблицы
                        Name=Name+p[i];
                        State=2;
                        break;
                  case NOT:
                        // слово кончилось и это не имя таблицы
                        Out=Out+Name+p[i];
                        Name="";
                        State=1;
                        break;
                  case DOT:
                        // слово кончилось и это имя таблицы
                        Out=Out+AddSharp(Name)+p[i];
                        Name="";
                        State=1;
                        break;
                  case EOL:
                        // строка кончилась
                        Out=Out+Name;
                        State=3;
                        break;
                  default:
                        Good=false;
                        Out="";
                        State=3;
                        break;
            }
            break;
      case 3:
```

```
                break;
   } // switch(State)
} while(p[i] && State!=3);
return Good ? Out : AnsiString("");
}
//---------------------------------------------------------
AnsiString EXPR_BASE::TableListWithSharp(AnsiString TableList){
int State=1; // текущее состояние конечного автомата
int Class; // класс очередного входного символа
int j;
AnsiString Name=""; // собираем здесь возможное имя таблицы
AnsiString Out=""; // строка - результат
int i=0; // индекс очередного символа из входной строки
char *p=TableList.c_str(); // входная строка в виде char *
bool first=true; // обработка символ конца строки '\0'
bool Good=true; // входная строка не имеет ошибок
do{
   if(first){
         first=false;
   } else {
         i++;
   }
   Class=SymbClass(p[i]);
   switch(State){
         case 1:
               switch(Class){
                     case CAN:
                           // возможно это начало имени таблицы
                           Name=p[i];
                           State=2;
                           break;
                     case NOT:
                           // символ не может принадлежать имени таблицы
                           Out=Out+p[i];
                           Name="";
                           State=1;
                           break;
                     case EOL:
                           // строка кончилась
                           Out=Out+AddSharp(Name);
                           State=3;
                           break;
                     default:
                           Good=false;
                           State=3;
                           break;
               }
               break;
         case 2:
               switch(Class){
                     case CAN:
                           // очередной символ возможного имени таблицы
                           Name=Name+p[i];
                           State=2;
                           break;
                     case NOT:
                           // слово кончилось и это имя таблицы
                           Out=Out+AddSharp(Name)+p[i];
                           Name="";
                           State=1;
                           break;
                     case EOL:
```

```
                        // строка кончилась
                        Out=Out+AddSharp(Name);
                        State=3;
                        break;
                  default:
                  // Этого не может быть
                        Good=false;
                        Out="";
                        State=3;
                        break;
               }
               break;
         case 3:
               break;
   } // switch(State)
} while(p[i] && State!=3);
return Good ? Out : AnsiString("");
}
```