# Оглавление

[Оглавление 1](#_Toc238191943)

[Введение 2](#_Toc238191944)

[Анализ предметной области 3](#_Toc238191945)

[Выбор средств реализации 15](#_Toc238191946)

[Постановка задачи 17](#_Toc238191947)

[Реализация проекта 19](#_Toc238191948)

[Список используемой литературы 30](#_Toc238191949)

# Введение

Целью данной курсовой работы является создание информационной системы «Учебные планы. Вычитка часов». Проектируемая информационная система состоит из трех составляющих: базы данных, системы управления базой данных и пользовательского приложения. Информационная система соответствует архитектуре клиент-сервер.

Пояснительная записка к курсовой работе состоит из введения, обзора состояния современных СУБД, проектирования базы данных, разработки интерфейса пользователя, руководства пользователя и приложений, содержащих код программной части и метаданные базы данных.

# Анализ предметной области

##

Основная цель АРМ заключается в том, чтобы предложить пользователю абстрактное представление данных, скрыв конкретные особенности хранения и управления ими. Следовательно, отправной точкой при проектировании базы данных должно стать абстрактное и общее описание информационных потребностей организации, которые должны найти свое отражение в создаваемой базе данных.

Комитетом ANSI-SPARC был предложен трехуровневый подход. Суть его в идентификации трех уровней абстракции, т.е. трех различных уровней описания элементов данных: внешнего, концептуального и внутреннего. Цель трехуровневой архитектуры заключается в отделении пользовательского представления данных от их физического представления. Ниже причислены причины, по которым желательно выполнять такое разделение:

* каждый пользователь должен иметь возможность обращаться к одним и тем же данным, используя свое собственное представление о них;
* каждый пользователь должен иметь возможность изменять свое представление о данных, причем это изменение не должно оказывать влияния на других пользователей;
* взаимодействие пользователей с данными не должно зависеть от особенностей их хранения;
* администратор базы данных должен иметь возможность изменить структуру хранения данных в базе, не оказывая влияния на пользовательские представления;
* внутренняя структура базы данных не должна зависеть от физических аспектов хранения информации.

Внешний уровень описывает ту часть базы данных, которая относится к каждому пользователю. Это представления базы данных с точки зрения пользователей.

Внутренний уровень это физическое представление базы данных в компьютере. Этот уровень описывает, как информация хранится в базе данных.

Внешний уровень

Внутренний уровень

Физический уровень

Представление 1

Пользователь 1

Представление 2

Представление N

**…**

Внутренняя схема

База данных

Пользователь 2

Пользователь N

В соответствии с принципами трехуровневого подхода было построено внешнее представление: в базе данных должна быть представлена информация о обучаемых, учебных группах, преподавателях.

Основы реляционной модели данных были впервые изложены в статье Е. Кодда в 1970 г. Эта работа послужила стимулом для большого количества статей и книг, в которых реляционная модель получила дальнейшее развитие. Наиболее распространенная трактовка реляционной модели данных принадлежит К. Дейту [5]. Согласно Дейту, реляционная модель состоит из трех частей:

• Структурной части.

• Целостной части.

• Манипуляционной части.

Структурная часть описывает, какие объекты рассматриваются реляционной моделью. Постулируется, что единственной структурой данных, используемой в реляционной модели, являются нормализованные n-арные отношения.

Целостная часть описывает ограничения специального вида, которые должны выполняться для любых отношений в любых реляционных базах данных. Это целостность сущностей и целостность внешних ключей.

Манипуляционная часть описывает два эквивалентных способа манипулирования реляционными данными – реляционную алгебру и реляционное исчисление.

В классической реляционной модели используются только простые (атомарные) типы данных. Простые типы данных не обладают внутренней структурой.

Домены – это типы данных, имеющие некоторый смысл (семантику). Домены ограничивают сравнения – некорректно, хотя и возможно, сравнивать значения из различных доменов.

Отношение состоит из двух частей – заголовка отношения и тела отношения. Заголовок отношения – это аналог заголовка таблицы. Заголовок отношения состоит из атрибутов. Количество атрибутов называется степенью отношения. Тело отношения – это аналог тела таблицы. Тело отношения состоит из кортежей. Кортеж отношения является аналогом строки таблицы. Количество кортежей отношения называется мощностью отношения.

Отношение обладает следующими свойствами:

• В отношении нет одинаковых кортежей.

• Кортежи не упорядочены.

• Атрибуты не упорядочены.

• Все значения атрибутов атомарны.

Реляционной базой данных называется набор отношений.

Схемой реляционной базы данных называется набор заголовков отношений, входящих в базу данных.

Современные СУБД допускают использование null-значений, т. к. данные часто бывают неполными или неизвестными. Споры о допустимости использования null-значений ведутся до сих пор. Средством, позволяющим однозначно идентифицировать кортежи отношения, являются потенциальные ключи отношения.

Потенциальный ключ отношения – это набор атрибутов отношения, обладающий свойствами уникальности и неизбыточности. Доступ к конкретному кортежу можно получить, лишь зная значение потенциального ключа для этого кортежа.

Традиционно один из потенциальных ключей объявляется первичным ключом, остальные – альтернативными ключами.

Потенциальный ключ, состоящий из одного атрибута, называется простым. Потенциальный ключ, состоящий из нескольких атрибутов, называется составным.

Отношения связываются друг с другом при помощи внешних ключей.

Внешний ключ отношения – это набор атрибутов отношения, содержащий ссылки на потенциальный ключ другого (или того же самого) отношения. Отношение, содержащее потенциальный ключ, на который ссылается некоторый внешний ключ, называется родительским отношением. Отношение, содержащее внешний ключ, называется дочерним отношением.

Внешний ключ не обязан обладать свойством уникальности. Поэтому, одному кортежу родительского отношения может соответствовать несколько кортежей дочернего отношения. Такой тип связи между отношениями называется «один-ко-многим».

Связи типа «много-ко-многим» реализуются использованием нескольких отношений типа «один-ко-многим».

В любой реляционной базе данных должны выполняться два ограничения:

* Целостность сущностей.
* Целостность внешних ключей.

Правило целостности сущностей состоит в том, что атрибуты, входящие в состав некоторого потенциального ключа не могут принимать null-значений.

Правило целостности внешних ключей состоит в том, что внешние ключи не должны ссылаться на отсутствующие в родительском отношении кортежи, т.е. внешние ключи должны быть корректны.

Ссылочную целостность могут нарушить операции, изменяющие состояние базы данных. Такими операциями являются операции вставки, обновления и удаления кортежей.

Для поддержания ссылочной целостности обычно используются две основные стратегии:

RESTRICT (ОГРАНИЧИТЬ) – не разрешать выполнение операции, приводящей к нарушению ссылочной целостности.

CASCADE (КАСКАДИРОВАТЬ) – разрешить выполнение требуемой операции, но внести каскадные изменения в другие отношения так, чтобы не допустить нарушения ссылочной целостности.

Дополнительными стратегиями поддержания ссылочной целостности являются:

SET NULL (УСТАНОВИТЬ В NULL) – все некорректные значения внешних ключей изменять на null-значения.

SET DEFAULT (УСТАНОВИТЬ ПО УМОЛЧАНИЮ) – все некорректные значения внешних ключей изменять на некоторое значение, принятое по умолчанию.

В реальных СУБД можно также отказаться от использования какой-либо стратегии поддержания ссылочной целостности:

IGNORE (ИГНОРИРОВАТЬ) – выполнять операции, не обращая внимания на нарушения ссылочной целостности.

При проектировании базы данных основной целью является создание наиболее точного представления данных, связей между ними и требуемых ограничений. И прежде всего, определить отношения. Метод, который используется для решения этой задачи называется нормализацией [13].

Не любое отношение имеет право на существование. Существует ряд требований. Цель этих требований – уменьшение избыточности и вероятности непреднамеренной потери данных. Требования эти могут быть Реляционная база данных должна быть нормализована. Т.е. отношения, образующие БД должны отвечать ряду требований. Процесс нормализации имеет своей целью устранение избыточности данных и предотвращение возможных нарушений целостности.

Сначала (Кодд, 1970) были предложены первые три нормальных формы: 1НФ, 2НФ, 3НФ. Затем было сформулированное более строгое определение третьей нормальной формы, которое получило название нормальная форма Бойса-Кодда. Затем появились 4НФ и 5НФ, на практике используемые крайне редко.

На практике процесс разработки БД сводится к достижению 3НФ. При достаточном опыте проектировщика БД нормализация происходит интуитивно. Либо ее можно использовать в качестве тестов, т.е. повергнуть уже разработанную БД проверке: удовлетворяет ли она требованиям нормализации.

Процесс преобразования базы данных к виду, отвечающему нормальным формам, называется нормализацией. Нормализация позволяет обезопасить базу данных от логических и структурных проблем, называемых аномалиями данных. К примеру, когда существует несколько одинаковых записей в таблице, существует риск нарушения целостности данных при обновлении таблицы. Таблица, прошедшая нормализацию, менее подвержена таким проблемам, т. к. ее структура предполагает определение связей между данными, что исключает необходимость в существовании записей с повторяющейся информацией.

Понятие нормальной формы было введено Эдгаром Коддом при создании реляционной модели БД. Основное назначение нормальных форм – приведение структуры базы данных к виду, обеспечивающему минимальную избыточность. Устранение избыточности производится за счёт декомпозиции отношений (таблиц) таким образом, чтобы в каждом отношении хранились только первичные факты (то есть факты, не выводимые из других хранимых фактов). Таким образом, нормализация не имеет целью уменьшение или увеличение производительности работы или же уменьшение или увеличение объёма БД. Конечной целью нормализации является уменьшение потенциальной противоречивости хранимой в БД информации.

Нормализация может применяться к таблице, которая представляет собой правильное отношение.

Таблица находится в **первой нормальной форме**, если каждый её атрибут атомарен. Под выражением «атрибут атомарен» понимается, что атрибут может содержать только одно значение. Таким образом, не существует 1NF таблицы, в полях которых могут храниться списки значений. Для приведения таблицы к 1NF обычно требуется разбить таблицу на несколько отдельных таблиц.

В реляционной модели отношение всегда находится в 1 (или более высокой) нормальной форме в том смысле, что иные отношения не рассматриваются в реляционной модели. То есть само определение понятия отношение заведомо подразумевает наличие 1NF.

Таблица находится во **второй нормальной форме**, если она находится в первой нормальной форме, и при этом любой её атрибут, не входящий в состав первичного ключа, функционально полно зависит от первичного ключа. Функционально полная зависимость означает, что атрибут функционально зависит от всего первичного составного ключа, но при этом не находится в функциональной зависимости от какой-либо из входящих в него атрибутов(частей). Или другими словами: в 2NF нет неключевых атрибутов, зависящих от части составного ключа (+ выполняются условия 1NF).

Таблица находится в **третьей нормальной форме** (3NF), если она находится во второй нормальной форме (2NF) и при этом любой ее не-ключевой атрибут зависит только от первичного ключа (Primary key, PK) (иначе говоря, один факт хранится в одном месте).

Таким образом, отношение находится в 3NF тогда и только тогда, когда оно находится во 2NF и отсутствуют транзитивные зависимости неключевых атрибутов от ключевых. Транзитивной зависимостью неключевых атрибутов от ключевых называется следующая: A → B и B → C, где A – набор ключевых атрибутов (ключ), B и С – различные множества неключевых атрибутов.

При решении практических задач в большинстве случаев третья нормальная форма является достаточной. Процесс проектирования реляционной базы данных, как правило, заканчивается приведением к 3NF.

## Инфологическая модель

Инфологический подход не предоставляет формальных способов моделирования реальности, однако он закладывает основы методологии проектирования БД.

Первой задачей инфологического проектирования является определение ПО системы, позволяющее изучить информационные потребности будущих пользователей. Другая задача этого этапа – анализ ПО, который призван сформировать взгляд на ПО с позиций сообщества будущих пользователей БД, т.е. инфологической модели ПО. Анализ ПО выполняется разработчиком логической базы данных – специалистом в данной ПО.

Инфологическая модель ПО представляет собой описание структуры и динамики ПО, характера информационных потребностей пользователей системы в терминах, понятных пользователю и независимых от реализации системы. Более того, инфологическая модель ПО не должна зависеть от модели данных, которая будет использована при создании БД.

Обычно описание ПО выражается в терминах не отдельных объектов и связей между ними, а их типов, связанных с ними ограничений целостности и тех процессов ПО, которые приводят к переходу ПО из одного состояния в другое. Такое описание может быть представлено любым способом, допускающим однозначную интерпретацию.

В простых случаях описание ПО представляется на естественном языке, в более сложных используется также математический аппарат: таблицы, диаграммы, графы и т.п. Если анализ ПО выполняется несколькими специалистами, то они должны принять соглашения, касающиеся:

* используемых методов анализа предметной области;
* правил именования и обозначения объектов ПО, атрибутов и связей;
* содержания и формата создаваемых ими документов.

База данных содержит четыре таблицы: Учебные планы, Группы, Предметы и Занятия. Таблицы связаны отношениями «один-ко-многим» как показано на ER-диаграмме:


## Даталогическая модель

На этапе даталогического проектирования разрабатывается даталогическая структура БД, соответствующая инфологической модели ПО. Решение этой задачи существенно зависит от модели данных, поддерживаемой выбранной СУБД. Результатом выполнения этого этапа являются схемы БД концептуального и внешнего уровней архитектуры, составленные на языках определения данных (DDL) выбранной СУБД.

На этапе даталогического проектирования строится логическая структура БД. При этом происходит преобразование исходной инфологической модели в модель данных, которая поддерживается конкретной СУБД. После этого производится проверка адекватности даталогической модели, отображаемой предметной области. Конечным результатом даталогического проектирования является описание структуры БД на языке описания данных конкретных СУБД.

Связи между классами, показанные в инфологической модели, в даталогической модели могут отображаться либо за счет совместного расположения связанных элементов, либо путем объявления связей между ними.

Не все виды связи существующие в предметной области можно отобразить даталогической моделью. Так большинство СУБД не обеспечивают поддержание связи типа М:М. В этом случае в даталогической модели вводится вспомогательный элемент, т.е. M:N разбивается на два отношения между исходными элементами и вспомогательными (1:M, 1:N).

Ниже приведен DDL-скрипт для создания базы данных:

SET SQL DIALECT 3;

SET NAMES WIN1251;

CREATE DATABASE 'LOCALHOST:C:\MySoft\UP\up.gdb'

USER 'SYSDBA' PASSWORD 'masterkey'

PAGE\_SIZE 16384

DEFAULT CHARACTER SET WIN1251;

CREATE TABLE GROUPS (

ID\_G INTEGER NOT NULL,

NAZV VARCHAR(4),

GOD INTEGER,

CHISL INTEGER

);

CREATE TABLE PREDM (

ID\_P INTEGER NOT NULL,

NAZV VARCHAR(25)

);

CREATE TABLE ZAN (

ID\_Z INTEGER NOT NULL,

ID\_UP INTEGER,

D DATE,

T TIME,

VID\_Z CHAR(1),

H SMALLINT DEFAULT 2

);

CREATE TABLE UP (

ID\_UP INTEGER NOT NULL,

ID\_G INTEGER,

ID\_P INTEGER,

SEM INTEGER,

H\_L INTEGER,

H\_P INTEGER,

H\_L\_CALC COMPUTED BY ((select sum(H) from zan where zan.id\_up=up.id\_up and vid\_z='Л')),

H\_P\_CALC COMPUTED BY ((select sum(H) from zan where zan.id\_up=up.id\_up and vid\_z='П'))

);

CREATE TABLE PREPODS (

ID\_P INTEGER NOT NULL,

FIO VARCHAR(30),

DR DATE

);

/\* Check constraints definition \*/

ALTER TABLE ZAN ADD CHECK (VID\_Z IN ('Л', 'П'));

ALTER TABLE UP ADD CONSTRAINT UNQ1\_UP UNIQUE (ID\_G, ID\_P, SEM);

ALTER TABLE GROUPS ADD CONSTRAINT PK\_GROUPS PRIMARY KEY (ID\_G);

ALTER TABLE PREDM ADD CONSTRAINT PK\_PREDM PRIMARY KEY (ID\_P);

ALTER TABLE UP ADD CONSTRAINT PK\_UP PRIMARY KEY (ID\_UP);

ALTER TABLE ZAN ADD CONSTRAINT PK\_ZAN PRIMARY KEY (ID\_Z);

ALTER TABLE UP ADD CONSTRAINT FK\_UP\_1 FOREIGN KEY (ID\_G) REFERENCES GROUPS (ID\_G) ON UPDATE CASCADE;

ALTER TABLE UP ADD CONSTRAINT FK\_UP\_2 FOREIGN KEY (ID\_P) REFERENCES PREDM (ID\_P) ON UPDATE CASCADE;

ALTER TABLE ZAN ADD CONSTRAINT FK\_ZAN\_1 FOREIGN KEY (ID\_UP) REFERENCES UP (ID\_UP) ON UPDATE CASCADE;

ALTER PROCEDURE ADD\_ZAN (ID\_Z INTEGER, N INTEGER) AS

declare variable d date;

declare variable t time;

declare variable vid\_z char(1);

declare variable h smallint;

declare variable id\_up integer;

begin

select id\_up, d, t, vid\_z, h from zan where id\_z =:id\_z

into:id\_up,:d,:t,:vid\_z,:h;

while (n>0) do

begin

d = d+7;

insert into zan (id\_up, d, t, vid\_z, h)

values (:id\_up,:d,:t,:vid\_z,:h);

n=n-1;

end

end

# Выбор средств реализации

В современном обществе информационные технологии проникают практически во все сферы человеческой деятельности. Системы хранения и обработки информации стали нормой во всех отраслях производства и услуг, в том числе образовательных. Особенно эта тенденция затрагивает производственные процессы, традиционно связанные с большим количеством информации на бумажных носителях. Эти задачи стали решать с тех пор, как только компьютерные технологии стали доступны для широкого пользования.

Подавляющее большинство задач автоматизации управления решается путем использования систем управления базами данных. По этой причине на рынке программных средств СУБД представлены широким спектром: от бесплатных или условно бесплатных до продуктов, стоимостью в тысячи долларов. Выбор конкретной СУБД для реализации стоящих задач уже сам по себе является достаточной сложной проблемой.

Даже самая дорогая СУБД сама по себе не снимает стоящих проблем. Это лишь инструмент для их решения. Важным фактором является грамотное проектирование базы данных, не допускающее избыточности хранимой информации и нарушения ее целостности и обеспечивающее оптимальные структуры хранения данных.

Однако помимо технических средств и СУБД необходимо осуществить взаимодействие между ними и пользователем. Этот интерфейс должен быть интуитивно понятен и максимально эргономичен, т.е. предоставить пользователю возможность легкой и быстрой модификации данных, выборки данных по запросу, представление данных в определенной форме. Проектирование интерфейса выполняется при помощи современных систем программирования.

В данной курсовой работе в качестве СУБД выступает сервер баз данных Firebird. Firebird является клоном такого известного продукта фирмы Borland, как Interbase. InterBase – сервер баз данных, имеющий 20-летнюю историю (создан в 1985 году). Инновации, предложенные в этом сервере, не только остаются актуальными до сих пор, но и начинают широко внедряться в альтернативных СУБД. Основной особенностью функциональности InterBase является версионность.

Механизм версионности кроме InterBase практически нигде не использовался, и потихоньку начал внедряться в коммерческих серверах не более 10 лет назад. На текущий момент в той или иной степени версионный механизм поддерживают кроме InterBase и Firebird: Oracle, PostgreSQL, а также MS SQL 2005. В 2000 году компания Borland в силу специфических причин выпустила InterBase 6.0 в OpenSource. Когда исходные коды InterBase были опубликованы, группа пользователей скопировала исходные тексты и начала новый проект под названием Firebird. Далее, с версии InterBase 6.5 Borland вернулся к схеме платного сервера с закрытыми исходными текстами, а Firebird продолжил существование как бесплатный для частного и коммерческого использования сервер с открытыми исходными текстами.

Несмотря на расхождения между последними версиями InterBase и Firebird, они оба наследуют все те положительные черты исходной СУБД InterBase, которые обеспечили высокую популярность этого сервера. Более того, здоровая конкуренция между этими серверами по возможностям SQL, производительности и другим параметрам гарантируют пользователям той или иной СУБД интенсивное развитие выбранного ими сервера. Небольшой объем дистрибутива, простой и быстрый процесс установки с моментальной готовностью к работе, минимальные требования к оборудованию, широкий спектр компонент и драйверов для разнообразных сред разработки, возможность обслуживания больших баз данных и большого числа пользователей, а также архитектура многоверсионности, упрощающая логику приложений – все это востребовано как начинающими, так и опытными разработчиками.

# Постановка задачи

Приложение призвано обеспечить интерфейс пользователя и СУБД и должно удовлетворять следующим требованиям:

* содержательное название.
* ясные и понятные инструкции.
* логическая обоснованность группировки и последовательности полей.
* легко узнаваемые названия полей.
* согласованная терминология и сокращения.
* согласованное использование цветов.
* визуальное выделение пространств и границ полей ввода данных.
* удобные средства перемещения курсора.
* средства исправления отдельных ошибочных символов и целых полей.
* средства вывода сообщений об ошибках при вводе недопустимых значений.
* особое выделение необязательных для ввода полей.
* средства вывода пояснительных сообщений с описанием полей.
* средства вывода сообщения об окончании заполнения формы.

Для доступа к базе данных и реализации интерфейса пользователя использовалась среда программирования Delphi.

В основе любого приложения баз данных лежат наборы данных, которые представляют собой группы записей (их удобно представить в виде таблиц в памяти), переданных из базы данных в приложение для просмотра и редактирования. Каждый набор данных инкапсулирован в специальном компоненте доступа к данным. В Delphi реализован набор базовых классов, поддерживающих функциональность наборов данных, и практически идентичные по составу наборы дочерних компонентов для технологий доступа к данным. Их общий предок – класс TDataSet.

Для обеспечения связи набора данных с визуальными компонентами отображения данных используется специальный компонент DataSource. Его роль заключается в управлении потоками данных между набором данных и связанными с ним компонентами отображения данных. Этот компонент обеспечивает передачу данных в визуальные компоненты и возврат результатов редактирования в набор данных, отвечает за изменение состояния визуальных компонентов при изменении состояния набора данных, передает сигналы управления от пользователя (визуальных компонентов) в набор данных. Компонент DataSource расположен на странице Data Access палитры компонентов. С каждым компонентом доступа к данным может быть связан как минимум один компонент DataSource. В его обязанности входит соединение набора данных с визуальными компонентами отображения данных. Компонент DataSource обеспечивает передачу в эти компоненты текущих значений полей из набора данных и возврат в него сделанных изменений.

С одним компонентом DataSource могут быть связаны несколько визуальных компонентов отображения данных. Эти компоненты представляют собой модифицированные элементы управления, которые предназначены для показа информации из наборов данных.

При открытии набора данных компонент обеспечивает передачу в набор данных записей из требуемой таблицы БД. Курсор набора данных устанавливается на первую запись. Компонент DataSource организует передачу в компоненты отображения данных значений необходимых полей из текущей записи. При перемещении по записям набора данных текущие значения полей в компонентах отображения данных автоматически обновляются.

Пользователь при помощи компонентов отображения данных может просматривать и редактировать данные. Измененные значения сразу же передаются из элемента управления в набор данных при помощи компонента DataSource. Затем изменения могут быть переданы в базу данных или отменены.

# Реализация проекта

На следующем рисунке приведена главная форма приложения:

На форме размещаются визуальные компоненты DBGrid, которые обеспечивают непосредственное взаимодействие пользователя с данными и кнопки, позволяющие добавлять, изменять, удалять записи таблиц.

Все невизуальные компоненты: IBDatabase, IBTransaction, IBDataset, DataSource вынесены в специальный контейнер DataModule:

Для выборки данных, их изменения, удаления и вставки в TIBDataSet используется набор свойств, представляющих собой SQL-запросы для манипулирования данными, – это SelectSQL, DeleteSQL, InsertSQL и ModifySQL.

RefreshSQL не используется для модификации записи, но является очень полезным для получения значений полей, которые были изменены триггерами базы данных и конкурирующими транзакциями.

В свойстве SelectSQL указывается запрос на выборку данных (SELECT… FROM…), которые будут доступны для просмотра и, в зависимости от содержимого остальных запросов, для редактирования, удаления и т.д.

В свойствах DeleteSQL, InsertSQL и ModifySQL указываются соответствующие запросы, которые будут вызываться автоматически самим компонентом при вызове методов Delete, Insert и Edit для удаления, вставки и редактирования записей.

Фактически все, что нужно сделать программисту, – это написать нужные запросы, выполняющие нужные операции над записями.

При помощи IBDatabase приложение подключается к базе данных с использованием учетной записи администратора: SYSDBA.

На следующем рисунке приведен вид окна установки параметров подключения:

Компоненты IBDataSetGroups и IBDataSetUP, а также IBDataSetUP и IBDataSetZan находятся с отношении главный-подчиненный (master-detail). Этот режим обеспечивает отбор записей подчиненной таблицы в зависимости от значения ключевого поля текущей записи главной таблицы.

На следующем рисунке представлено окно редактирования текста запроса на выборку компонента IBDataSetUP. Для реализации master-detail используется свойство IBDataSetUP. DataSource:= DataSourceGroups и условие отбора по значению текущей группы:

Компонент DBGrid позволяет гибко настраивать вид таблицы. В частности скрывать столбцы, управлять заголовком (шрифт, цвет, размещение). В проектируемом приложении заголовки столбцов (свойство Title) выполнены полужирным по центру.

При добавлении или изменении текущей записи открывается форма с компонентами редактирования. При этом в первом случае соответствующий набор данных переводится в режим Insert, во втором – в режим Edit.

На формах расположены компоненты редактирования полей текущей записи, т.н. db-aware компоненты. Все они связаны с соответствующими наборами данных, расположенными на главной форме приложения. Внесение изменений при помощи этих компонентов автоматически переводит набор данных в состояние Edit. При добавлении новой записи необходимо перевести набор данных в состояние Insert, вызвав соответствующий метод.

## Руководство пользователя

Для работы информационной системы необходимо наличие файла приложения UP.EXE и файла базы данных UP.GDB. Кроме того, на машине пользователя должен быть установлен SQL-сервер Firebird.

На следующем рисунке приведен общий вид приложения:

Пользователь может просматривать списки групп, предметов, преподавателей, учебных планов, а также информацию о проведенных занятиях.

Приложение позволяет пользователю:

* добавлять, изменять, удалять учебные группы
* добавлять, изменять, удалять предметы
* добавлять, изменять, удалять учебные планы
* редактировать сведения о проведенных занятиях

В таблице УЧЕБНЫЕ ПЛАНЫ пользователь видит планы только той группы, которая является текущей в таблице ГРУППЫ, а в таблице ЗАНЯТИЯ – только те занятия, которые проведены по текущему плану в таблице УЧЕБНЫЕ ПЛАНЫ.

Кроме того, выполняется автоматический подсчет суммарного количества часов проведенных лекционных и практических занятий.

При модификации таблицы ЗАНЯТИЯ, изменения автоматически отражаются в таблице УЧЕБНЫЕ ПЛАНЫ.

При нажатии кнопки Удалить появляется окно, в котором необходимо подтвердить удаление записи. Это необходимо для защиты от непреднамеренной потери данных.

Попытка удалить группу или предмет, которые указаны хотя бы в одном учебном плане, а также учебный план, по которому проведено хотя бы одно занятие приведет к ошибке. Чтобы удалить группу или предмет, нужно предварительно удалить все учебные планы, в которых они задействованы. Аналогично, чтобы удалить учебный план, необходимо предварительно удалить все проведенные по нему занятия.

При нажатии кнопки Добавить или кнопки Изменить кнопочной панели, открывается отдельное окно, в котором можно указать необходимые данные.

Окно редактирования данных группы:

В данном окне доступны для редактирования данные о группе:

* Название группы
* Год поступления группы
* Численный состав группы

Окно редактирования предмета:

Окно редактирования данных о преподавателе:

В данном окне доступны для редактирования данные о преподавателе:

* Фамилия, имя, отчество.
* Дата рождения.

В данном окне доступно для редактирования только название предмета.

Окно редактирования данных об учебном плане:

В данном окне доступны для редактирования данные об учебном плане:

* Семестр.
* Изучаемый предмет.
* Количество лекционных часов.
* Количество практических часов.
* Преподаватель.

Сведения о группе носят справочный характер и не доступны для редактирования. При составлении нового учебного плана, он автоматически наследует ту группу, которая является текущей в таблице ГРУППЫ.

В окно редактирования данных о проведенном занятии доступны для редактирования данные о проведенном занятии:

* Дата и время проведения занятия.
* Вид занятия (лекционное или практическое).
* Количество часов (по умолчанию = 2).
* Преподаватель (по умолчанию = преподаватель по учебному плану).

Сведения об учебном плане носят справочный характер и не доступны для редактирования. При добавлении нового занятия ему присваивается номер учебного плана, который является текущим в таблице УЧЕБНЫЕ ПЛАНЫ.

После внесения изменений окно закрывается нажатием на одну из кнопок OK или Cancel. В первом случае все внесенные изменения сохраняются, во втором – отменяются.

При добавлении записи появляется аналогичное окно с пустыми полями для ввода новых значений. Кнопка OK закрывает окно с сохранением, кнопка Cancel – без сохранения.

**Заключение**

В результате выполнения курсовой работы были разработаны база данных под управлением сервера Firebird и приложение, обеспечивающее пользовательский интерфейс с базой данных.

Данный проект не является окончательным и будет значительно доработан в рамках предстоящей дипломной работы.

# Список используемой литературы

1. Избачков Ю., Петров В. Информационные системы: Учебник для вузов. 2-е издание, Изд.: Питер, 2006.
2. Аппак М.А., «Автоматизированные рабочие места на основе персональных ЭВМ», М.: 'Радио и связь', 1999 г.
3. Борри Х. Firebird: руководство разработчика баз данных: Пер. с англ. – СПб.: БХВ-Петербург, 2006.
4. Грофф Дж., Вайнберг П. Энциклопедия SQL. 3-е издание. Пер. с англ. – С-Пб.: Питер, 2003.
5. Дейт К. Введение в системы баз данных. – Киев: Диалектика, 1998.
6. Карпова Т.С. Базы данных. Модели, разработка, реализация. – СПб: Питер, 2001.
7. Конноли Т., Бегг К., Страчан А. Базы данных: проектирование, реализация, сопровождение. – М., Вильямс. 2003.
8. Ковязин А., Востриков С. Мир Interbase. Архитектура, администрирование и разработка приложений баз данных в Interbase. – М., Кудиц-Образ, 2002.
9. Ульман Д., Уидом Д. Введение в системы баз данных. – Лори. 2000.
10. Фаронов В.В. Delphi 5. Руководство программиста. Нолидж. 2001.
11. Фаронов В.В. Программирование баз данных в Delphi 7. – СПб: Питер, 2004.
12. Хансен Г., Хансен Д. Базы данных. Разработка и управление. – М., Бином. 2000.
13. Голицына О.Л., Максимов Н.В., Попов И.И. Базы данных: учеб. пособие. – М., Форум: Инфра-М, 2007.