

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО ОБРАЗОВАНИЮ

Государственное образовательное учреждение
высшего профессионального образования
«Оренбургский государственный университет»

Колледж электроники и бизнеса

Кафедра вычислительной техники и математики

Т.И. ИШТЕРЯКОВА

БАЗЫ ДАННЫХ

КУРС ЛЕКЦИЙ

Рекомендовано к изданию Редакционно-издательским советом
государственного образовательного учреждения
высшего профессионального образования
«Оренбургский государственный университет»

Оренбург 2009

УДК 621.38(075.32)

ББК 32.81я73

И- 97

Рецензент

Заместитель директора по НМР С.А. Кузюшин

Иштерякова, Т.И.

Базы данных: Курс лекций /Т.И.Иштерякова

И-97 – Оренбург: ГОУ ВПО ОГУ,

2009. – 133 с.

Курс лекций предназначен для проведения теоретических занятий, обеспечивающих учебный процесс по дисциплине “Базы данных” в колледже Электроники и бизнеса ОГУ для студентов 3 курса в 6 семестре специальности 230105 Программное обеспечение вычислительной техники и автоматизированных систем очной и заочной формы обучения.

Курс лекций составлен с учетом Государственного образовательного стандарта среднего профессионального образования по направлению подготовки дипломированных специалистов - утвержденного 30.12.2003 Министерством образования Российской Федерации.

ББК 32.81я73

©Иштерякова Т.И.

©ГОУ ВПО ОГУ, 2009

Содержание

Введение	7
1 Теория проектирования баз данных	8
1.1 Основные понятия	8
1.2 Информационные модели	10
1.2.1 Виды информационных моделей	10
1.2.2 Типы логических моделей	12
1.2.2.1 Иерархическая модель данных	12
1.2.2.2 Сетевая модель данных	13
1.2.2.3 Реляционная модель данных	14
1.3 Основы реляционной алгебры	15
1.4 Взаимосвязи в моделях и реляционный подход к построению моделей	20
1.4.1 Взаимосвязь «один-к-одному»	20
1.4.2 Взаимосвязь «один-ко-многим»	20
1.4.3 Взаимосвязь «многие-ко-многим»	21
1.5 Нормализация отношений	22
1.6 Проектирование баз данных	23
1.6.1 Требования, предъявляемые к базе данных	23
1.6.2 Этапы проектирования баз данных	24
1.6.2.1 Построение информационной модели и определение сущностей	24
1.6.2.2 Определение взаимосвязей между сущностями	24
1.6.2.3 Задание первичных и альтернативных ключей	25
1.6.2.4 Приведение модели к требуемому уровню нормальной формы	25
1.6.2.5 Физическое описание модели	25
1.7 Системы управления базами данных	26
1.7.1 Классификация СУБД	26
1.7.2 Основные характеристики СУБД	26
1.7.2.1 Производительность СУБД	27
1.7.2.2 Обеспечение целостности данных на уровне БД	27
1.7.2.3 Обеспечение безопасности	28
1.7.2.4 Работа в многопользовательских средах	28
1.7.2.5 Импорт, экспорт	28
1.7.2.6 Доступ к данным посредством языка SQL	28
1.7.2.7 Возможности запросов инструментальные средства разработки прикладных программ	29
1.7.3 Обзор современных систем управления базами данных	30
1.8 Постреляционные базы данных	30
1.8.1 Основные направления совершенствования баз данных	30
1.8.2 Общие понятия объектно-ориентированного подхода	32
1.8.3 Объектно-ориентированные модели данных	35
2 Организация баз данных в Visual Fox Pro	36

2.1 Работа с таблицами	36
2.1.1 Создание таблицы	36
2.1.2 Открытие таблицы.....	38
2.1.3 Модификация структуры таблицы	39
2.2 Работа с записями в таблице	40
2.2.1 Команда редактирования Browse	40
2.2.2 Добавление новой записи в таблицу	41
2.2.3 Удаление записи из таблицы	41
2.2.3.1 Пометка записи к удалению	42
2.2.3.2 Физическое удаление записи.....	43
2.3 Работа с базами данных	43
2.3.1 Создание и открытие базы данных.....	43
2.3.2 Добавление таблиц в базу данных.....	44
2.3.3 Освобождение таблиц	45
2.4 Индексирование базы данных.....	45
2.4.1 Виды индексов и индексных файлов	46
2.4.2 Создание индекса	47
2.4.3 Команды для работы с индексными файлами.....	48
2.4.3.1 Замена текущего индекса	48
2.4.3.2 Перестройка индексных файлов.....	49
2.4.3.3 Преобразование одноиндексного файла в тег	49
2.4.3.4 Удаление тега из мультииндексного файла	50
2.4.3.5 Вывод на экран имен индексных файлов и имен тегов	50
2.5 Создание взаимосвязей между таблицами в Visual Fox Pro	50
2.5.1 Ограничение взаимосвязей.....	50
2.5.2 Понятие рабочей области	52
2.5.3 Установление взаимосвязи «один-к-одному».....	53
2.5.4 Установление взаимосвязи «один-ко-многим»	53
2.5.5 Установление взаимосвязи с помощью главного меню	54
2.5.6 Команды для работы с таблицами в БД.....	57
2.5.6.1 Объединение двух табличных файлов в один файл	57
2.5.6.2 Корректировка данных в связанных таблицах.....	57
2.5.6.3 Создание итогового табличного файла.....	58
2.6 Управление базами данных.....	58
2.6.1 Сортировка данных.....	58
2.6.2 Поиск данных	60
2.6.2.1 Поиск методом полного перебора	60
2.6.2.2 Поиск по полю текущего индекса	61
2.6.3 Фильтрация данных	62
2.6.3.1 Ограничение на количество строк.....	62
2.6.3.2 Ограничение на количество полей	63
3 Программирование в Visual FoxPro.....	63
3.1 Работа с программными файлами	63
3.1.1 Создание программного файла	63
3.1.2 Запуск программного файла.....	65

3.1.3 Структура программы.....	65
3.1.4 Модульность программы.....	67
3.1.4.1 Внешние процедуры.....	67
3.1.4.2 Внутренние процедуры.....	68
3.1.4.3 Процедура-функция.....	68
3.2 Отладка программы в Visual FoxPro.....	69
3.2.1 Диалоговые окна Program Error и Data Session.....	69
3.2.2 Программа отладчик.....	70
3.2.3 Собственный обработчик ошибок.....	71
3.2.4 Анализ работы программы.....	72
3.3 Основы языка программирования.....	73
3.3.1 Переменные, константы и массивы.....	74
3.3.2 Присвоение значений.....	75
3.3.3 Команды для работы с переменными.....	76
3.3.4 Команды для работы с массивами.....	77
3.3.5 Команды ввода-вывода.....	79
3.4 Условная и циклическая обработка данных.....	82
3.4.1 Команды циклов.....	82
3.4.2 Команды выполнения и ветвления алгоритмов.....	83
3.5 Объектно-ориентированное построение Fox Pro.....	84
4 Организация интерфейса в Visual Fox Pro.....	85
4.1 Создание меню в Visual Fox Pro.....	86
4.1.1 Подготовка к созданию меню.....	86
4.1.2 Виды меню.....	87
4.1.3 Технологии построения меню.....	88
4.1.4 Структура меню.....	88
4.1.5 Определение составляющих меню.....	89
4.1.6 Управление доступом к пунктам меню.....	91
4.1.7 Активизация составляющих меню и удаление меню.....	92
4.2 Использование окон в Visual Fox Pro.....	93
4.2.1 Работа с окнами.....	93
4.2.2 Описание окна.....	94
4.2.3 Активизация окна.....	96
4.3 Элементы управления.....	96
4.4 Создание экранной формы.....	97
4.4.1 Создание экранной формы с помощью Мастера форм – Form Wizard.....	97
4.4.2 Создание экранной формы с помощью Конструктора форм – Form Designer.....	99
4.5 Создание отчетов.....	101
4.5.1 Виды отчетов.....	101
4.5.2 Создание отчетов с помощью Мастера.....	102
4.5.3 Создание отчетов с помощью Конструктора.....	108
5 Хранимые процедуры. Триггеры. Язык запросов SQL.....	110
5.1 Условия достоверности, хранимые процедуры, триггеры, представление данных.....	109

5.1.1 Условия достоверности ввода данных на уровне записей.....	109
5.1.2 Триггеры.....	111
5.1.3 Хранимые процедуры	112
5.2 Классификация реляционных языков	112
5.2.1 Типы реляционных языков.....	112
5.2.2 dBASE-подобные реляционные языки	113
5.2.3 Графические (схематичные) реляционные языки.....	115
5.3 Основные характеристики языка SQL	117
5.3.1 Краткая характеристика языка SQL	117
5.3.2 Операторы языка SQL для работы с реляционной базой данных.....	118
5.3.2.1 Создание реляционных таблиц	118
5.3.2.2 Изменение структуры таблиц	119
5.3.2.3 Удаление таблицы	119
5.3.2.4 Ввод данных в таблицу	120
5.3.2.5 Операции соединения таблиц	121
5.3.2.6 Удаление записей в таблице.....	122
5.3.2.7 Обновление (замена) значений полей записи	122
5.4 Организация запросов к базе данных на языке SQL	123
5.4.1 Синтаксис оператора SELECT	123
5.4.2 Задание условий выборки.....	125
5.4.3 Групповые функции SQL	126
5.4.4 Подчиненный запрос.....	127
Список использованных источников	129

Введение

Предмет «Базы данных» рассчитан на курс лекций в объеме 60 часов.

На лекционных занятиях будет раскрыт материал по теории баз данных, дан сравнительный анализ различных СУБД и изложены приемы создания и работы с базами данных в среде Visual Fox Pro.

СУБД Visual Fox Pro является одной из популярных программ в этой области. В данной СУБД можно работать как с мастерами создания простых таблиц, форм, отчетов, так и с конструкторами для создания крупных проектов. А так же можно применять объектно-ориентированный язык программирования Visual Fox Pro для осуществления сложных проектов и пользовательских классов.

Кроме того, в данном курсе лекций отражены основные характеристики и операторы языка SQL.

Курс лекций «Базы данных» состоит из следующих разделов:

1) Раздел «Теория баз данных» посвящен изложению основных понятий и общепринятых терминов, используемых при проектировании баз данных. Также в данном разделе рассматриваются виды информационных моделей, взаимосвязей в таблицах, нормализация отношений и этапы проектирования баз данных. Даны основные характеристики СУБД, обзор современных СУБД и основные направления развития реляционных баз данных (постреляционные базы данных);

2) В разделе «Организация баз данных в Visual Fox Pro» рассмотрены вопросы создания, открытия, просмотра, редактирования и модификации табличных файлов. Также описаны способы создания и работы с файлами баз данных, индексирования и управления базами данных;

3) Раздел «Программирование в Visual Fox Pro» знакомит с приемами создания и отладки программных файлов, основами языка программирования, а также элементами объектно-ориентированного программирования;

4) Раздел «Организация интерфейса в Visual Fox Pro» описывает способы создания пользовательских меню, применение каждого вида меню, их преимущества и недостатки. Также в данном разделе рассмотрены вопросы построения пользовательских окон различной степени сложности, описаны методы создания экранных форм и отчетов с помощью мастеров и конструкторов;

5) В разделе «Хранимые процедуры и триггеры. Основы языка запросов SQL» рассмотрены вопросы задания условий достоверности ввода данных с помощью триггеров и хранимых процедур. Данный раздел также содержит краткую характеристику языка SQL и описание организации запросов к базе данных на языке SQL.

1 Теория проектирования баз данных

1.1 Основные понятия

Обработка больших объемов информации становится не под силу человеку, поэтому для быстрой и достоверной обработки данных используются персональные компьютеры. Информация в компьютере структурируется и хранится, как правило, в виде таблиц. В свою очередь, отдельные таблицы объединяются в базы данных.

Базы данных (БД) — это один или несколько файлов данных, предназначенных для хранения, изменения и обработки больших объемов взаимосвязанной информации. Примерами баз данных могут быть телефонная книга, каталог товаров, штатное расписание и т. д.

Мало создать базу данных, надо разработать механизмы извлечения информации из базы данных и способы ее представления на экране и бумаге. Для этих целей предназначена система управления базами данных (СУБД).

СУБД — это система программного обеспечения, предоставляющая доступ к данным многих пользователей. СУБД обеспечивает правильность, полноту и непротиворечивость данных, а также простой и понятный интерфейс.

Основой баз данных является информация (данные) о конкретной группе предметов (объектов). Как правило, базы данных создаются для какого-либо предприятия (организации) в целом; информацией (данными) пользуются различные структурные подразделения предприятия, причем каждое подразделение использует одни и те же данные в различных форматах. Особое место занимает создание терминологии, то есть однозначного понимания назначения конкретных данных, их типа и структуры.

Прежде чем приступить к проектированию приложения баз данных, надо уточнить некоторые понятия и определения.

Объект — это нечто существующее и различимое, обладающее набором свойств. Отличие одного объекта от другого определяется конкретными значениями свойств. Объекты бывают материальные и идеальные. К материальным объектам относятся предметы материального мира: автомобиль, здания, предметы мебели и т. д. К идеальным (абстрактным) объектам можно отнести спектакль, содержание книги и т. д.

Сущность — отображение объекта в памяти человека или компьютера.

Параметр — конкретное значение любого из свойств объекта.

Атрибут — конкретное значение любого из свойств сущности.

Таблица — некоторая регулярная структура, состоящая из конечного числа записей (строк). Как правило, в базах данных используются двумерные массивы (матрицы).

Запись — это одна строка таблицы (или нескольких таблиц), полностью описывающая одну сущность. Каждая запись состоит из конечного числа полей.

Поле — это один элемент записи, в котором хранится конкретное значение атрибута.

Ключевым элементом данных (ключом) называется такой атрибут, по значению которого можно определить значения других атрибутов.

Первичный ключ — это атрибут или группа атрибутов, которые однозначно определяют каждую запись в таблице. Первичный ключ всегда должен быть уникальным, то есть его значения не должны повторяться.

Альтернативный ключ — это отличные от первичного ключа атрибут или группа атрибутов, которые также однозначно определяют каждую запись в таблице. Например: сущность «Служащий» имеет атрибуты: идентификатор служащего (табельный номер), фамилия, имя, отчество, должность, оклад. Первичным ключом назначим поле «Идентификатор служащего». Альтернативным ключом назначим группу полей «Фамилия», «Имя», «Отчество» (только в том случае, если нет тройных тезок).

Связь — это функциональная зависимость между сущностями. Если между некоторыми сущностями существует связь, то атрибуты из одной сущности ссылаются или некоторым образом связаны с атрибутами другой сущности. Связи описываются пятью основными характеристиками:

1) тип связи — это идентифицирующая характеристика, когда дочерняя сущность однозначно определяется через ее связь с родительской сущностью. Атрибуты, составляющие первичный ключ родительской сущности, обязательно входят в первичный ключ дочерней сущности;

2) не идентифицирующая характеристика, когда дочерняя сущность определяется иначе, чем через связь с родительской сущностью. Атрибуты первичного ключа родительской сущности входят как неключевые атрибуты в дочернюю сущность;

3) родительская сущность;

4) дочерняя (зависимая) сущность;

5) мощность связи — это отношение количества родительских сущностей к соответствующему количеству дочерних сущностей.

Хранимая процедура — это приложение (программа), объединяющая запросы пользователя и процедурную логику и хранящаяся в базе данных.

Правило — это логическое условие, определяющее значение одного атрибута в зависимости от значения другого атрибута (или группы атрибутов). С помощью правила контролируется достоверность вводимой информации.

Ограничение — это логическое условие, накладывающее ограничение (интервал допустимых значений) на значение атрибута.

Триггер — это предварительно определенное действие или последовательность действий, автоматически осуществляемых при выполнении операций обновления, добавления или удаления данных. Триггер выполняется автоматически при выполнении указанных действий и не может быть отключен. Триггер включает в себя:

– правила или ограничения;

– событие, которое требует проверки правил и ограничений;

– предусмотренные действия, которые выполняются с помощью процедуры или последовательности процедур.

Ссылочная целостность — это обеспечение непротиворечивости функциональных взаимосвязей между сущностями. Непротиворечивость выполняется путем соответствия значений первичного ключа родительской сущности значениям внешнего (любого не первичного) ключа дочерней сущности. Ссылочная целостность может контролироваться при всех операциях, изменяющих информацию в базе данных, при этом возможны следующие варианты обработки событий:

- отсутствие проверки;
- проверка допустимости;
- запрет операции;
- каскадное выполнение операций обновления или удаления данных одновременно в нескольких связанных таблицах;
- установка пустого значения по умолчанию.

Нормализация отношений — это процесс построения оптимальной структуры таблиц и связей в реляционной БД. В процессе нормализации данные группируются в таблицы, представляющие объекты и их взаимосвязи.

Словарь данных — это централизованное хранилище сведений о сущностях, взаимосвязях между сущностями, их источниках, значениях, использовании и форматах представления.

1.2 Информационные модели

1.2.1 Виды информационных моделей

При создании баз данных рассматривают два вида информационных моделей: информационная модель предприятия и информационная модель данных.

Информационная модель предприятия строится на втором этапе проектирования базы данных. Здесь определяются структурные подразделения фирмы, которые используют информацию из базы данных, и направление движения потоков информации между структурными подразделениями фирмы.

Информационная модель данных имеет более сложную структуру. Здесь отображаются:

- источники возникновения информации;
- структурные подразделения фирмы, которые создают или используют информацию;
- переходы от одного типа модели к другому;
- подразделения потребителей информации.

Подразделение 1,... Подразделение N — структурные подразделения фирмы. В левой части рисунка эти подразделения выступают как источники концептуальных требований, то есть предоставляют сведения для создания базы данных (рисунок 1). В верхней части рисунка эти же подразделения выступают как потребители информации и источники данных для базы данных.

Концептуальная модель данных — это совокупность концептуальных требований, выдвинутых сотрудниками структурных подразделений фирмы.

В результате отображения концептуальной модели на СУБД будет получена *логическая модель данных*.

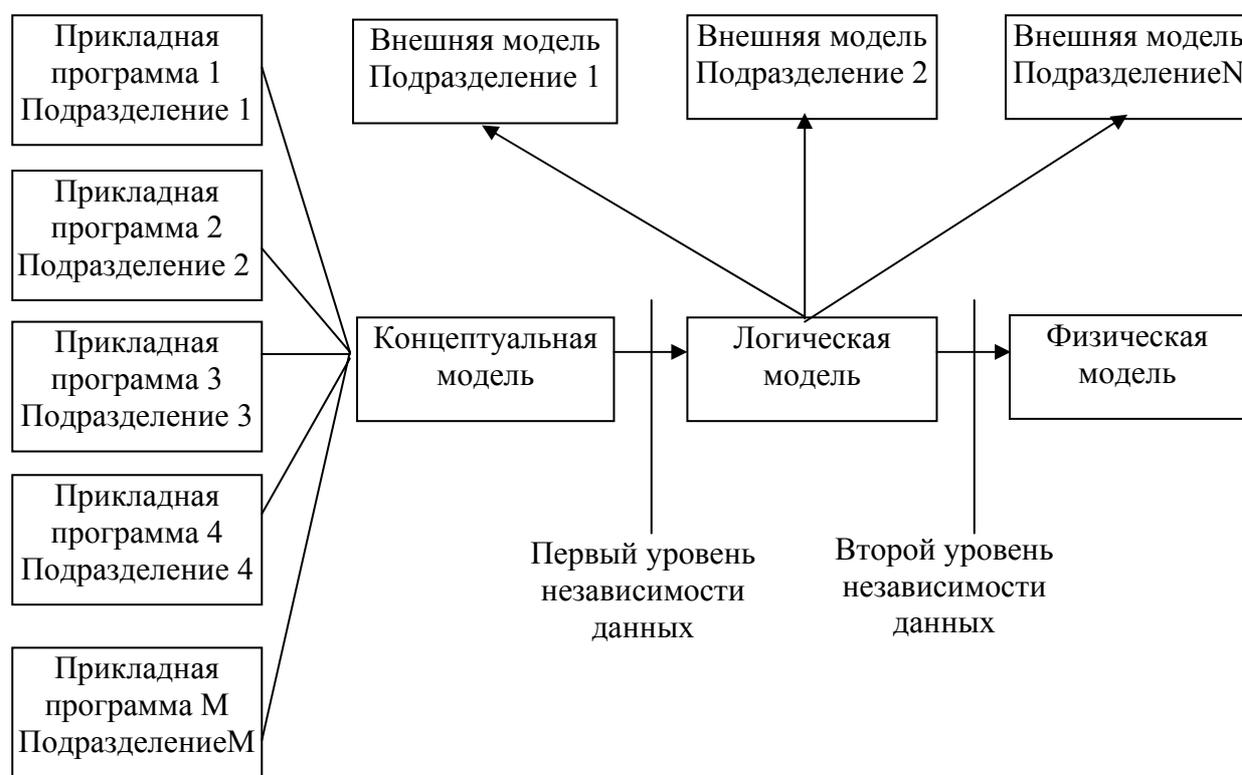


Рисунок 1- Информационная модель данных

В процессе отображения концептуальной модели подбирается такая СУБД, которая в полной мере может удовлетворить требования заказчика. Если по каким-либо причинам выполнить требования заказчика не удастся, то разработчик должен предоставить заказчику убедительные аргументы и убедить заказчика снизить концептуальные требования. Процесс согласования концептуальных требований трудоемкая и длительная процедура. После построения логической модели необходимо составить письменный протокол, в котором перечислить все концептуальные требования и операции по обработке информации в базе данных.

При переходе от концептуальной к логической модели необходимо обеспечить следующее условие: внешние модели никак не связаны с типом физической памяти, где хранятся данные базы данных, и методами обработки этих данных. Это условие называется *первым уровнем независимости данных*. Далее общая логическая модель данных разделяется на некоторые составные части, которые называют внешними моделями. *Внешняя модель*— это та часть общей логической модели данных, с которой работает конкретное структурное подразделение фирмы. Границы разделения внешних моделей не четкие, то есть одни и те же данные могут получать различные структурные подразделения фирмы, но пополнять и редактировать данные может только

одно конкретное подразделение. Форма отображения одних и тех же данных в разных подразделениях может быть различной. В зависимости от поставленных целей логическая модель данных может быть либо *иерархической*, либо *сетевой*, либо *реляционной*.

Отображение логической модели на конкретные технические средства называется *физической моделью данных*. Физическая модель строится на пятом этапе проектирования базы данных. При построении физической модели определяются технические характеристики персонального компьютера: объем оперативной памяти, необходимый объем памяти на жестком диске и т. д. Кроме того, на этом этапе определяют количество и типы индексов, методы доступа к данным. При переходе от логической модели к физической модели необходимо обеспечить выполнение условия: концептуальная модель допускает некоторое расширение требований к базе данных без переделки самой базы данных. Это условие называется *вторым уровнем независимости данных*. Второй уровень независимости данных достигается, как правило, за счет хорошей техники и дисциплины программирования (например, константа задается только один раз, а ее значение передается всем подпрограммам через параметры).

1.2.2 Типы логических моделей

Существует три типа логических моделей: иерархическая, сетевая и реляционная.

1.2.2.1 Иерархическая модель данных

Иерархическая модель данных, как следует из названия, имеет иерархическую структуру, т.е. каждый из элементов связан только с одним стоящим выше элементом, но в то же время на него могут ссылаться один или несколько стоящих ниже элементов

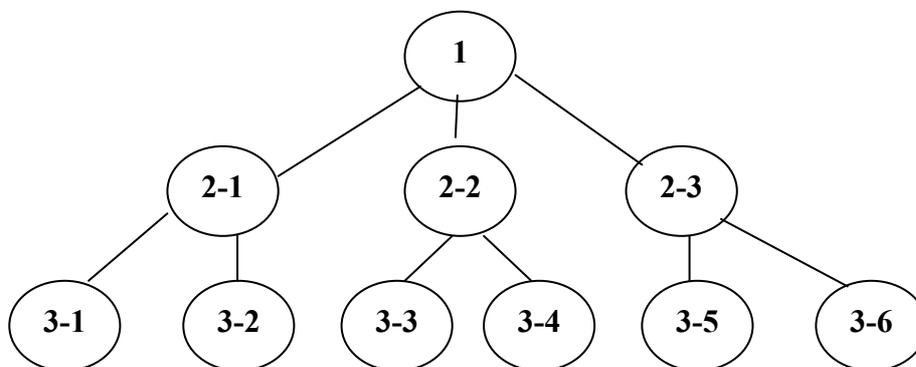


Рисунок 2 - Логическая иерархическая модель

В терминологии иерархической модели используются более конкретные

понятия: «элемент» (узел); «уровень» и «связь». Узел чаще всего представляет собой атрибут (признак), описывающий некоторый объект.

Иерархически модель схематически изображается в виде графа, в котором каждый узел является вершиной .

Эта модель представляет собой совокупность элементов, расположенных в порядке их подчинения от общего к частному и образующих граф – дерево с иерархической структурой (рисунок 2,3).

Такой граф имеет единственную вершину, не подчиненную никакой другой вершине и находящуюся на самом верхнем (первом) уровне. Число вершин первого уровня определяет число деревьев в базе данных. Запрещены взаимосвязи на одном уровне.

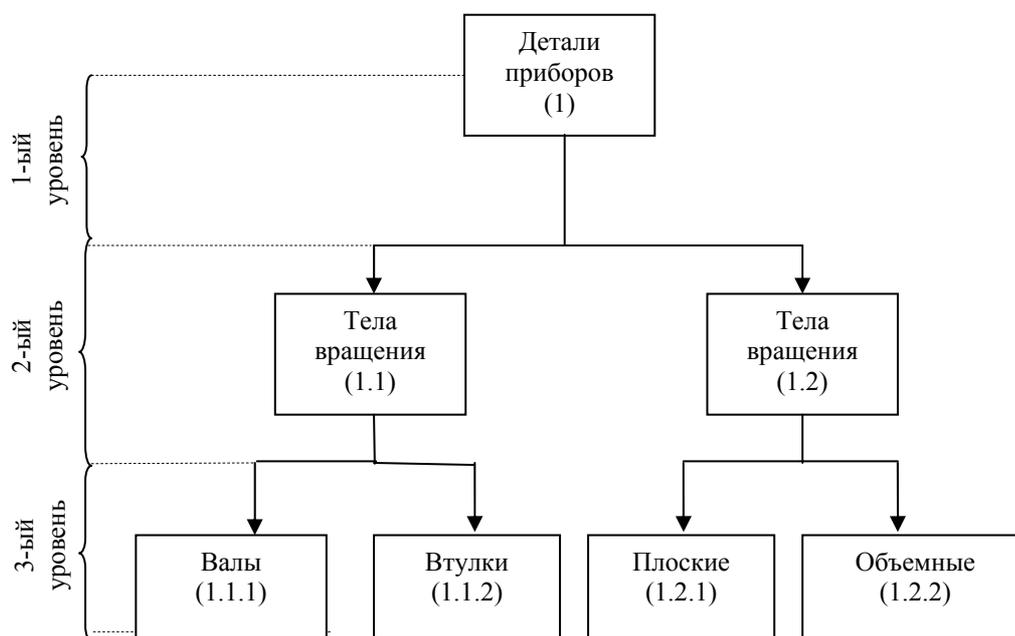


Рисунок 3 – Пример иерархической модели данных

1.2.2.2 Сетевая модель данных

Сетевая модель более демократична. В сетевой модели отсутствует понятие главного и подчиненного объекта (рисунок 4,5). Один и тот же объект может выступать как главный и как подчиненный, то есть иметь любое количество взаимосвязей. Здесь допустимы связи на одном уровне. Эта модель использует ту же терминологию, что и иерархическая модель: «узел», «уровень» и «связь».

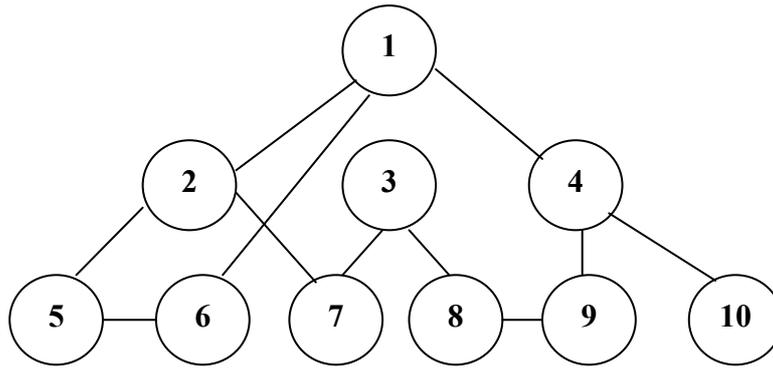


Рисунок 4 - Логическая сетевая модель

Как известно из теории графов, сетевой граф может быть преобразован в граф-дерево.

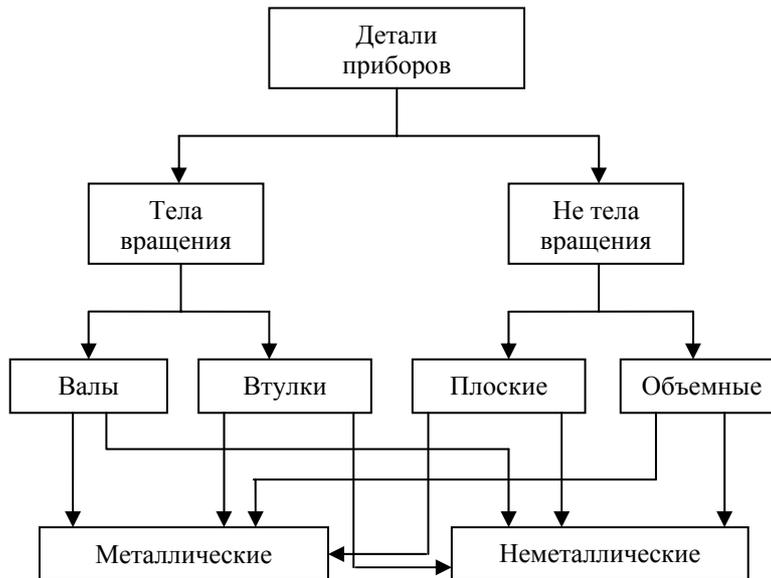


Рисунок 5 – Пример сетевой модели данных

1.2.2.3 Реляционная модель данных

Основная идея реляционной модели данных заключается в том, чтобы представить любой набор данных в виде двумерного массива – таблицы. В простейшем случае реляционная модель описывает единственную двумерную таблицу (таблица 1), но чаще всего эта модель описывает структуру и взаимоотношения между несколькими различными таблицами.

Обязательным условием построения реляционной модели является наличие в каждой таблице первичного ключа. Этот вид модели имеет

наибольшее распространение при построении баз данных.

Имя файла						
Поле		Признак ключа	Формат поля			
Имя (обозначение)	Полное наименование		Тип	Длина	Точность (для чисел)	N/NN
имя1						
...						
имя n						

Таблица 1 – Структура реляционной таблицы

Рассмотрим пример реляционной модели данных (таблица 2).

Таблица 2 - Детали приборов

Код	Расположение поверхностей	Дополнительная характеристика
1	Тела вращения	Валы
2	Тела вращения	Втулки
3	Не тела вращения	Плоские
4	Не тела вращения	Объемные

На рисунке 6 показано

Код	Расположение поверхностей
1	Тела вращения
2	Не тела вращения

Код	Дополнительная характеристика
1.1	Валы
1.2	Втулки
2.1	Плоские
2.2	Объёмные

разделение таблицы 2 на две связанные таблицы.

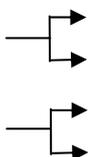


Рисунок 6 - Две связанные таблицы реляционной модели данных

Реляционные модели данных, или реляционные базы данных, являются в настоящее время основным способом в проектировании и организации информационных систем в производстве и бизнесе, поэтому в дальнейшем мы рассмотрим теоретические основы и практические методы разработки

реляционных баз данных.

1.3 Основы реляционной алгебры

Методы и алгоритмы обработки информации в реляционных базах данных основываются на теории реляционной алгебры, которую ранее называли алгеброй логики, или булевой алгеброй.

Алгебра логики представляет собой прежде всего алгебру высказываний. Под высказыванием в алгебре логики понимают всякое предложение, которое либо истинно, либо ложно; при этом может иметь место только одно из двух указанных значений (например: «Москва – столица России» – истинное высказывание; «снег черен» – ложное высказывание). Каждое конкретное высказывание имеет вполне определенное значение истинности – это постоянная, равная 0 или 1. От конкретных (постоянных) высказываний следует отличать так называемые переменные высказывания.

Переменное высказывание (предикат) – это не высказывание в подлинном смысле, а переменная для высказываний (пропозициональная переменная), т.е. символ, вместо которого можно подставить постоянные высказывания (или их наименования) и который может принимать лишь два значения: «истинно» или «ложно», или соответственно 1 и 0 (двоичная переменная). Переменные высказывания (т.е. пропозициональные переменные) обозначаются буквами, отличными от тех букв, которыми обозначаются постоянные высказывания. Применение переменных высказываний в алгебре логики служит для выражения всеобщности. Оно позволяет формулировать законы алгебры логики для любых высказываний.

Предметом изучения в алгебре логики являются бинарные (или двузначные) функции, т.е. функции, которые принимают лишь два значения («истинно» или «ложно»; 0 или 1) и зависят от одной или нескольких бинарных переменных. Это так называемые *булевы функции*.

Из одного или нескольких высказываний, принимаемых за простые, можно составлять сложные высказывания, которые будут бинарными функциями простых высказываний. Объединение простых высказываний в сложные в алгебре логики производится без учета внутреннего содержания (смысла) этих высказываний. Используются определенные логические операции (или логические связи), позволяющие объединять некоторые данные высказывания (постоянные или переменные) в более сложные (постоянные или переменные).

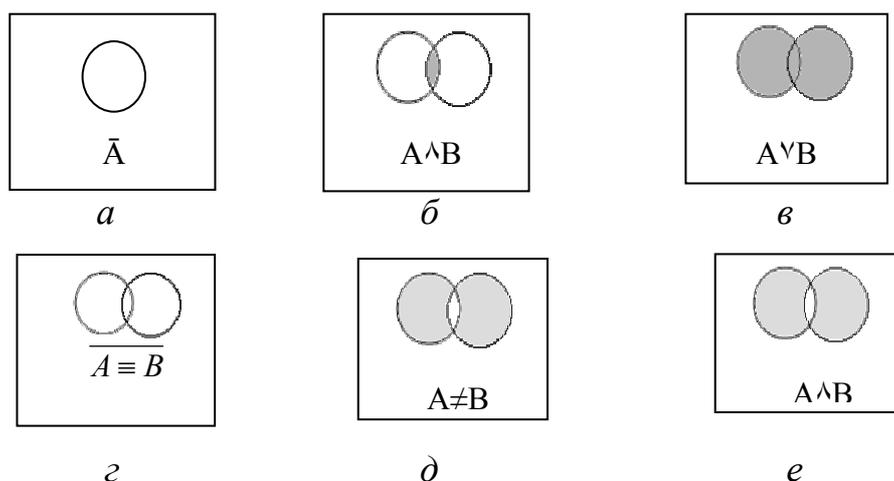
К числу основных логических операций относятся следующие: отрицание, конъюнкция, дизъюнкция, эквивалентность, импликация. Логические операции задаются таблично, как функции простых высказываний.

Любое сложное выражение, полученное из простых высказываний посредством указанных выше логических операций, называется формулой алгебры логики. Важнейшую роль в алгебре логики играют равносильные

соотношения, выражающие основные законы алгебры логики:

Другим примером алгебры Буля является алгебра классов, которая дает наглядное геометрическое истолкование для основных логических операций.

Рассмотрим высказывание A , в котором идет речь о принадлежности некоторого свойства a предметам какой-то области. Представим себе, что предметы этой области изображаются точками части плоскости, ограниченной некоторым квадратом (рисунок 7), которая обозначается через Q .



a – множество A ; $б$ – пересечение множеств; $в$ – объединение множеств; $г$ – эквивалентности двух множеств; $д$ – отрицание эквивалентности; $е$ – отрицание дизъюнкции

Рисунок 7 - Графическое изображение булевых функций

Ясно, что эти точки плоскости Q разбиваются на два класса (множества): на класс точек, имеющих свойство a , т.е. таких, для которых $A = 1$, и на класс точек, не имеющих этого свойства, т. е. таких, для которых $A = 0$, причем каждая точка плоскости Q обязательно принадлежит только одному из этих классов.

Первый класс можно считать геометрическим изображением доказывания A , или множеством A . При этом может получиться, например, картина, приведенная на рисунок 7, а, на котором высказывание A изображено в виде некоторой области, ограниченной замкнутым (залитым) контуром. Очевидно, что высказывание A (не A) будет тогда изображаться множеством всех остальных точек квадрата Q . Рассмотрим, как при такой интерпретации можно представить основные логические операции.

Конъюнкция двух высказываний будет представляться переселением двух множеств (рисунок 7, б). Действительно, $A \wedge B = 1$ только тогда, когда $A=1$ и $B=1$, а это имеет место лишь для точек, одновременно принадлежащих

множеству А и множеству В (их пересечению).

Дизъюнкция двух высказываний $A \vee B$ будет изображаться множеством, которое получается путем объединения множеств А и В (рисунок 7, в).

Эквивалентность двух высказываний $A \equiv B$ изобразится так, показано на рисунке 7, г, так как истинность $A \equiv B$ равна 1 либо при $A = 1$ и $B = 1$, либо при $A = 0$ и $B = 0$.

Отрицание эквивалентности $A \equiv B$, показанное на рисунке 7, д, получается, если учесть, что $\overline{A \equiv B}$ равно $A \equiv B$

Отрицание дизъюнкции $\overline{A \vee B}$ показано на рисунке 7, е.

Подобные диаграммы, называемые *диаграммами Венна*, могут быть использованы для наглядного представления логических формул с целью их анализа и упрощения.

Рассмотренные логические операции – конъюнкция, дизъюнкция, отрицание – являются независимыми и могут быть выражены друг через друга. В частности, из них можно выделить системы логических операций и с их помощью можно представить все функции алгебры логики. Такие системы логических операций (иногда вместе с константами 1 или 0) называются функционально полными.

В таблице 3 представлены различные варианты обозначения логических операций.

Таблица 3 – Варианты обозначения логических операций

Конъюнкция (логическое)	Дизъюнкция (логическое)	Отрицание (логическое)	Эквивалентность	Импликация
И	или	НЕ		Если...
AND	OR	NOT		If... then
&		\neg	\rightarrow	\equiv
\wedge	\vee	\overline{A}	\supset	\leftrightarrow

Кроме перечисленных выше основных операций алгебры логики существуют еще две операции: отрицание конъюнкции и отрицание дизъюнкции, утверждающие несовместимость соответствующих высказываний.

Отрицание конъюнкции $\overline{A \wedge B}$ обозначается: A / B и называется операцией Шеффера.

Отрицание дизъюнкции $\overline{A \vee B}$, выраженное штрихом Шеффера, имеет смысл $A = A / B$.

Операция Шеффера играет важную роль в теории проектирования логических схем процессоров ЭВМ, поскольку электронная схема, реализующая операцию Шеффера, является универсальным функциональным элементом, при помощи которого могут быть построены любые функциональные логические устройства. Графическое представление операции

Шеффера приведено на рисунке 7, е.

Операции конъюнкции и дизъюнкции называются двойственными, и формулы алгебры логики, получаемые одна из другой заменой \wedge на \vee и \vee на \wedge , также называются *двойственными*. Для двойственных формул F и F^* справедлива равносильность высказываний

$$F(A_1, A_2, \dots, A_n) = \overline{F^*(\overline{A_1}, \overline{A_2}, \dots, \overline{A_n})}$$

В алгебре логики устанавливается следующий принцип двойственности: если формулы F и Φ равносильны, то двойственные формулы F^* и Φ^* также равносильны.

Наиболее наглядно структура формул алгебры логики видна, когда они приведены к одной из двух так называемых нормальных форм.

Первая нормальная форма – конъюнктивная (КНФ) представляет собой некоторую конъюнкцию дизъюнкций, причем в каждой дизъюнкции отдельные члены представляют собой либо простые высказывания (т. е. высказывания, которые не включают в себя других высказываний), либо отрицания : высказываний.

Вторая нормальная форма – дизъюнктивная (ДНФ) представляет собой некоторую дизъюнкцию конъюнкций, где в каждой конъюнкции отдельные члены являются простыми высказываниями, либо их отрицаниями.

Нормальные формы удобны для выделения двух важных классов формул: постоянно-истинных и постоянно-ложных. Постоянно-истинные формулы всегда равны 1 (совпадают с 1). Постоянно-ложные формулы всегда равны 0 (совпадают с 0). Эти классы формул играют существенную роль при упрощении логических выражений.

Существует также *дизъюнктивная совершенная нормальная форма (ДСНФ)*, которая обладает следующими свойствами:

- не имеет одинаковых слагаемых;
- каждое слагаемое содержит в качестве множителей либо основные переменные, либо их отрицания;
- ни в одном слагаемом нет двух одинаковых множителей и не содержится переменная вместе с ее отрицанием.

Подобным же образом определяется *конъюнктивная совершенная нормальная форма (КСНФ)*, представляющая собой конъюнкцию дизъюнкций, удовлетворяющих аналогичным условиям.

Совершенные нормальные формы могут использоваться при решении вопросов о равносильности сложных формул алгебры логики: две формулы алгебры логики являются равносильными, если они приводятся к одинаковым совершенным нормальным формам. Практическое применение этих форм затрудняется в связи с их громоздкостью, поэтому на практике часто используются так называемые минимальные нормальные формы.

Минимальная дизъюнктивная нормальная форма (МДНФ) представляет собой дизъюнкцию конъюнкций, в которой:

- нет повторяющихся множителей ни в одном слагаемом;
- нет таких пар слагаемых, в которых были бы одинаковые множители;

– для всяких двух слагаемых, в которых имеется одна общая переменная, входящая в одно слагаемое в прямом виде, а в другое слагаемое в виде отрицания, имеется третье слагаемое, представляющее собой конъюнкцию остальных множителей первых двух слагаемых.

Важной областью применения алгебры логики является алгоритмизация процессов переработки информации и управления в реляционных базах данных.

1.4 Взаимосвязи в моделях и реляционный подход к построению моделей

Взаимосвязь выражает связь между двумя множествами данных (таблицами). Существует три типа взаимосвязи: «один-к-одному», «один-ко-многим» и «многие-ко-многим».

1.4.1 Взаимосвязь «один-к-одному»

Одной записи в одной таблице соответствует одна запись в другой таблице. Например: каждый гражданин имеет только один паспорт. С другой стороны, паспорт выписывается только на одно лицо (рисунок 8).

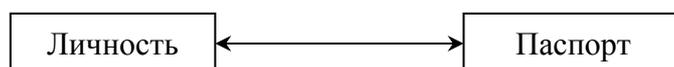


Рисунок 8 - Взаимосвязь «один-к-одному» для сущностей

Взаимосвязь жесткая, то есть ни в той, ни в другой таблице не может быть несвязанной записи. Если наложить ограничение, что один клиент может сделать только один заказ, то будем также иметь взаимосвязь «один-к-одному».

Для реляционной БД (рисунок 9) получим:

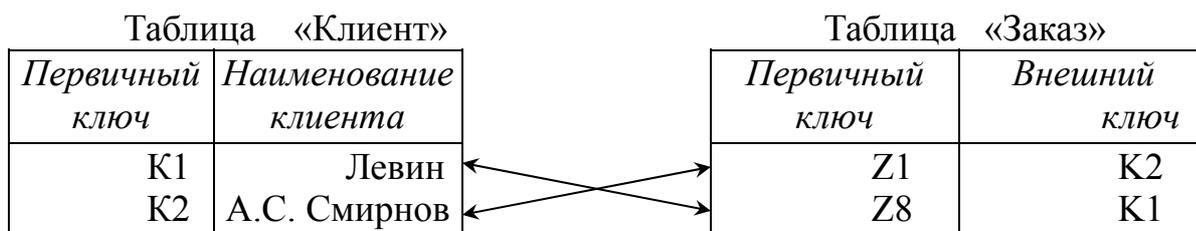


Рисунок 9 - Взаимосвязь «один-к-одному» для реляционной базы данных

1.4.2 Взаимосвязь «один-ко-многим»

Одной записи в одной таблице соответствует несколько записей в другой таблице. Наиболее распространенная взаимосвязь при создании реляционных баз данных (рисунок 10).



Рисунок 10 - Взаимосвязь «один-ко-многим» для сущностей

Каждый клиент может купить несколько автомобилей, но каждый автомобиль принадлежит только одному человеку (владелец автомобиля указывается в техническом паспорте). Если в дочерней таблице (таблица «Автомобиль») имеются записи, не содержащие внешнего ключа (ключ клиента), то эти записи будут потеряны, что недопустимо (рисунок 11).



Рисунок 11 - Взаимосвязь «один-ко-многим» для реляционной базы данных

1.4.3 Взаимосвязь «многие-ко-многим»

Нескольким записям в одной таблице соответствует несколько записей в другой таблице (рисунок 12).



Рисунок 12 - Взаимосвязь «многие-ко-многим» для сущностей

Продавцы-тезки обслуживают покупателей тоже тезок. Для реляционной базы данных будем иметь таблицы «Клиент» и «Продавец» (рисунок 13).

Таблица «Клиент»

Таблица «Продавец»

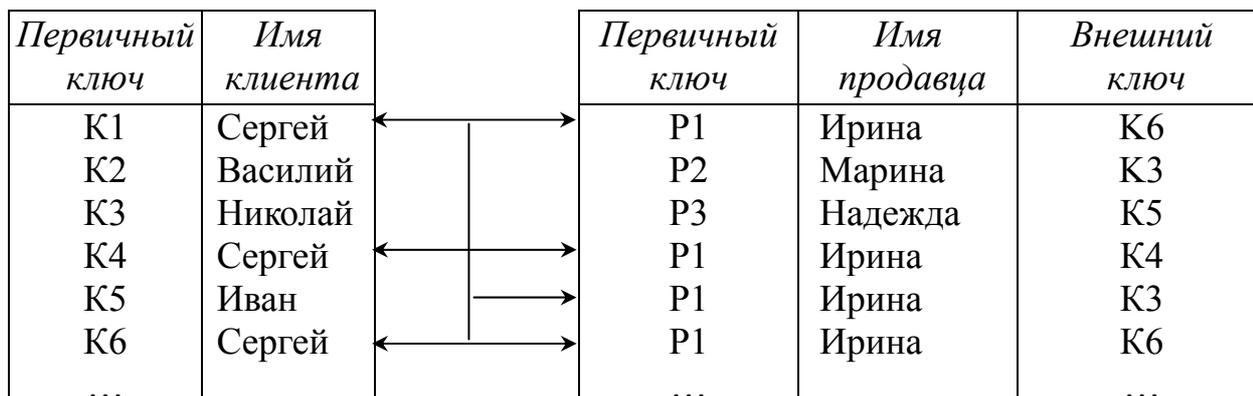


Рисунок 13- Взаимосвязь «многие-ко-многим» для реляционной базы данных

Таблица «Клиент» и таблица «Продавец» содержат повторяющиеся записи. При взаимосвязи «многие-ко-многим» одна из таблиц обязательно будет избыточной (не оптимальной). Для удаления избыточной информации взаимосвязь между таблицами «Клиент» и «Продавец» следует отобразить в виде таблицы перекрестных связей (рисунок 14). Таблица перекрестных связей содержит только ключи. Значения ключей в таблице перекрестных связей будут повторяться, но записи в таблице «Продавец» будут уникальными.

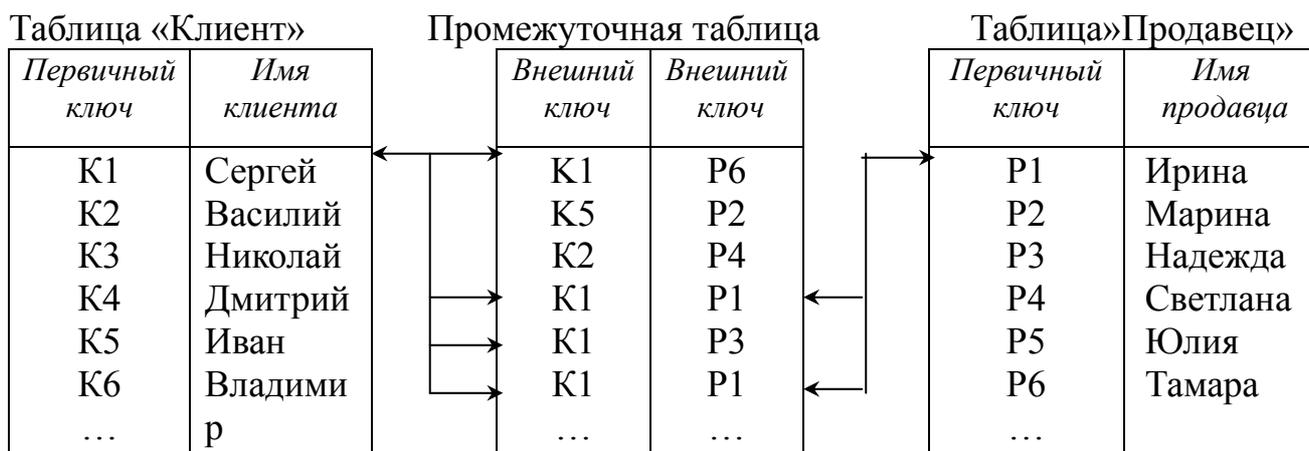


Рисунок 14 - Взаимосвязь «многие-ко-многим» для реляционной базы данных с использованием таблицы перекрестных связей

1.5 Нормализация отношений

Нормализация отношений — это процесс построения оптимальной структуры таблиц и связей в реляционной БД. В процессе нормализации данные группируются в таблицы, представляющие объекты и их взаимосвязи.

Всего существует пять нормальных форм таблицы. При создании приложений баз данных в объеме предприятия используют первые три нормальные формы.

Для таблицы будут выполнены условия первой нормальной формы, если:

- каждое поле (концептуальное требование) неделимо;
- отсутствуют повторяющиеся поля или группы полей.

Если перечисленные выше условия выполняются, то все концептуальные требования могут быть сведены либо в одну общую таблицу, либо можно создать по одной таблице для каждого структурного подразделения.

Условия второй нормальной формы:

- выполняются условия первой нормальной формы;
- первичный ключ однозначно определяет всю запись;
- все поля зависят от первичного ключа;
- первичный ключ не должен быть избыточным.

Сохраняя первичные и альтернативные ключи, назначенные на третьем этапе, назначаем, при необходимости, дополнительные первичные и внешние ключи, в результате чего выделяем из таблицы структурного подразделения одну или несколько таблиц. Таким образом, данные для одного структурного подразделения могут быть представлены как одной таблицей, так и несколькими таблицами. Переход между таблицами разных структурных подразделений осуществляется по первичным ключам, назначенным на третьем этапе, а переход между таблицами внутри одного структурного подразделения осуществляется по первичным ключам, назначенным при выполнении второй нормальной формы.

Условия третьей нормальной формы:

- выполняются условия второй нормальной формы;
- каждое не ключевое поле не должно зависеть от другого не ключевого поля.

При выполнении третьей нормальной формы должны быть разрушены транзитивные связи внутри каждой таблицы. При этом одно (или несколько) зависимых не ключевых полей выделяются в новую таблицу с обязательным добавлением первичных ключей для связи вновь выделенной таблицы с другими таблицами.

1.6 Проектирование баз данных

Создание приложения базы данных включает в себя строго определенную последовательность выполнения действий, называемых этапами проектирования. Выполнение указанных ниже действий приведет к созданию оптимальной структуры базы данных, в общем случае уменьшит время проектирования и обеспечит возможность уточнения структуры базы данных без ее полной переделки.

1.6.1 Требования, предъявляемые к базе данных

Проектирование баз данных начинается со сбора концептуальных требований. Концептуальное требование — это одно данное (одно свойство объекта), которое будет храниться в базе данных. Концептуальные требования получают как от руководства фирмы, так и в основном от конечных пользователей, то есть от сотрудников, непосредственно работающих с базой данных. Кроме того, на этом этапе решается вопрос — какие действия по обработке данных должны выполняться в базе данных. База данных должна:

- удовлетворять требованиям заказчика и содержать сведения только о тех объектах, которые интересуют заказчика;
- обладать приемлемым быстродействием, то есть пользователь должен получать необходимые ему сведения за короткое время;
- иметь возможность последующего расширения без существенной переделки, как самой базы данных, так и средств управления ею;
- не зависеть (или мало зависеть) от количества помещаемых в нее данных;
- легко перестраиваться при изменении программной и аппаратной среды;
- содержать только достоверные данные. Достоверность данных должна обеспечиваться как при вводе новых данных, так и при редактировании уже имеющихся данных;
- доступ к данным должны иметь определенные лица.

1.6.2 Этапы проектирования баз данных

1.6.2.1 Построение информационной модели и определение сущностей

На этом этапе проектирования базы данных решаются следующие вопросы:

- ставится задача на проектирование базы данных, то есть доказывается актуальность создания базы данных;
- собираются концептуальные требования и, на их основе, строится концептуальная модель данных.

Концептуальная модель данных составляется по результатам анализа поставленной заказчиком задачи и обработки концептуальных требований конечных пользователей. Результатом выполнения первого этапа проектирования является информационная модель данных и список основных сущностей — прообраз будущих таблиц. В данном случае под сущностью понимается структурное подразделение фирмы. Концептуальная модель данных будет состоять из совокупности групп концептуальных требований для каждого структурного подразделения фирмы, причем некоторые концептуальные требования могут повторяться в разных группах.

1.6.2.2 Определение взаимосвязей между сущностями

На этом этапе проектирования определяются направление движения потоков информации между структурными подразделениями фирмы-заказчика базы данных, источники возникновения информации, места ее модификации и потребления. Результатом выполнения этого этапа проектирования будет функциональная схема движения потоков информации, с указанием типов взаимосвязей, между структурными подразделениями фирмы.

1.6.2.3 Задание первичных и альтернативных ключей

Для каждой структурной единицы фирмы определяются атрибуты (данные), которые будут храниться в базе данных, а также первичный и альтернативный ключи. Добавление ключей в список концептуальных требований необходимо для обеспечения организации движения потоков информации между структурными подразделениями фирмы, в соответствии со вторым этапом проектирования базы данных. Кроме того, при анализе концептуальных требований определяется, какие алгоритмы и расчеты исходных величин (хранимые процедуры) будут храниться вместе с базой данных. При этом количество хранимых процедур должно быть минимальным.

1.6.2.4 Приведение модели к требуемому уровню нормальной формы

На этом этапе проектирования выполняется главная задача — нормализация отношений. В процессе нормализации концептуальные требования группируются в таблицы. На этом этапе проектирования концептуальные требования для каждого структурного подразделения могут быть сведены либо в одну таблицу, либо в несколько таблиц. Здесь также решается вопрос ликвидации избыточной информации, то есть концептуальные требования, используемые несколькими структурными подразделениями, сводятся в одну таблицу с одновременным добавлением ключей для перехода в другие таблицы (для других структурных подразделений). Таким образом, добиваются существенного сокращения объема памяти. На этом этапе также решается вопрос о том, какие таблицы будут справочниками, то есть информация в этих таблицах не изменяется или изменяется очень медленно. Следует иметь в виду, что чрезмерное увеличение количества таблиц приводит к потере общей идеи создания базы данных, и сама база данных становится трудной для понимания и управления. Для базы данных объема предприятия оптимальное количество таблиц должно быть не более сорока или пятидесяти.

Условия 1, 2 и 3 нормальной формы описаны в пункте 1.5.

После выполнения четвертого этапа проектирования должна быть получена структура базы данных: количество таблиц, список атрибутов (концептуальных требований), которые хранятся в каждой таблице, первичные и внешние ключи для перехода между таблицами, виды взаимосвязей между таблицами и список хранимых процедур.

1.6.2.5 Физическое описание модели

На этом этапе каждая таблица, созданная на четвертом этапе:

- получает свое имя, под которым она будет храниться в базе данных;
- каждый атрибут (концептуальное требование) таблицы получает свое имя, тип и размер;
- для каждого ключа, как первичного, так и внешнего, определяются его характеристики

На пятом этапе также предусматриваются меры по обеспечению ссылочной целостности, то есть установление между таблицами не противоречивых взаимосвязей. Установление не противоречивых взаимосвязей и обеспечение достоверности в данных в любой момент времени является главной и самой трудоемкой задачей.

В результате выполнения работ по пятому этапу можно определить технические характеристики персонального компьютера: объем оперативной памяти, объем памяти на жестком диске и т.д.

1.7 Системы управления базами данных

1.7.1 Классификация СУБД

Системой управления БД называют программную систему предназначенную для создания на ЭВМ общей БД, используемой для решения множества задач.

Подобные системы служат для поддержания баз данных в актуальном состоянии и обеспечивают эффективный доступ пользователей к содержащимся в ней данным в рамках предоставленных пользователем полномочий. СУБД предназначена для централизованного управления базой данных в интересах всех работающих в этой системе.

По степени универсальности различают две степени СУБД:

- системы общего назначения;
- специализированные системы.

СУБД общего назначения не ориентированные на какую-либо предметную область или на информационную потребности какой-либо группы пользователей. СУБД общего назначения это сложные программные комплексы предназначены для выполнения всей совокупности функций, связанных с созданием и эксплуатацией БД.

Специализированные СУБД создаются редких случаях при невозможности или нецелесообразности использования СУБД общего назначения.

1.7.2 Основные характеристики СУБД

Основными характеристиками СУБД являются:

- 1) Производительность;
- 2) Обеспечение целостности данных на уровне БД;
- 3) Обеспечение безопасности;
- 4) Работа в многопользовательских средах;
- 5) Импорт, экспорт;
- 6) Доступ к данным посредством языка SQL;
- 7) Возможности запросов и инструментальные средства разработки прикладных программ.

1.7.2.1 Производительность СУБД

Производительность СУБД оценивается:

- временем выполнения запроса;
- скоростью поиска информации в неиндексированных полях;
- временем выполнения операций импортирования базы данных из других форматов;
- скоростью создания индексов и выполнения таких массовых операций как обновление, вставка, удаление данных;
- максимальным числом параллельных обращений к данным в многопользовательском режиме;
- временем генерации отчетов.

На производительность СУБД оказывают влияние два фактора:

- 1) СУБД, которые следят за соблюдением целостности данных, несут дополнительную нагрузку, которую не испытывают другие программы;
- 2) производительность приложений сильно зависит от правильного проектирования и построения базы данных.

1.7.2.2 Обеспечение целостности данных на уровне БД

Эта характеристика подразумевает наличие средств позволяющих удостовериться, что информация в БД всегда остается корректной и полной. Должны быть установлены правила целостности, и они должны храниться вместе с базой данных и соблюдаться на глобальном уровне. Целостность данных должна обеспечивать независимо от того, каким образом данные заполняются в память. К средствам обеспечения целостности данных на уровне СУБД относятся :

- встроенные средства для назначения первичного ключа, в том чис-

ле средства для работы с типом полей с автоматическими приращением, когда СУБД самостоятельно присваивает новые уникальные значения;

- средства поддержания ссылочной целостности, которые обеспечивают запись информации о связях таблиц и автоматически пресекают любую операцию приводящую к нарушению ссылочной целостности.

1.7.2.3 Обеспечение безопасности

Некоторые СУБД предусматривают средства обеспечения безопасности данных.

Такие средства обеспечивают выполнение следующих операций:

- шифрование прикладных программ;
- шифрование данных;
- защиту паролем;
- ограничение уровня доступа.

1.7.2.4 Работа в многопользовательских средах

Практически все рассматриваемые СУБД предназначены для работы в многопользовательских средах, но обладают для этого различными возможностями. Обработка данных в многопользовательских средах предполагает выполнение программным продуктом следующих функций:

- блокировку БД, файла, записи, поля;
- идентификацию станции, установившей блокировку;
- обновление информации после модификации;
- контроль за временем и повторение обращений;
- обработка транзакций;
- работу с сетевыми системами.

1.7.2.5 Импорт, экспорт

Эта характеристика отражает:

- возможность обработки СУБД информации, подготовленной другими программными средствами;
- возможность использования другими программами данных сформулированные средствами рассматриваемые СУБД.

1.7.2.6 Доступ к данным посредством языка SQL

Язык запросов SQL реализован в целом ряде популярных СУБД для различных типов ЭВМ типа как базовой, либо как альтернативной. В силу своего широкого использования является международным стандартом языка запроса. Язык SQL предоставляет развитые возможности как конечным пользователям, так и специалистам в области обработки данных.

СУБД имеет доступ к данным SQL в следующих случаях:

- БД совместима с ODBC (Open Database Connectivity – открытое соединение БД);
- реализована естественной поддержкой SQL-баз данных;
- возможна реализация SQL запросов локальных данных.

Многие СУБД могут «прозрачно» подключаться к входным SQL-подсистемам с помощью ODBC или драйверов, являющихся их частью, поэтому существует возможность создания прикладных программ для них. Некоторые программные продукты совместимы также с SQL при обработке интерактивных запросов на получение данных, находящихся на сервере или на рабочем месте.

Access 2.0 и Paradox for Windows работают с источниками SQL-данных, совместимых с системой ODBC.

Fox Pro (for Dos и for Windows) поставляются с дополнительными библиотеками, которые обеспечивают доступ к SQL-базам данных, способным работать совместно с системой ODBC, но эта возможность менее интегрирована, чем средства первичного ввода информации в Access и Paradox for Windows.

Можно напрямую управлять базами данных Access с помощью языка SQL и передавать сквозные SQL-запросы совместимым со спецификацией ODBC SQL-базам данных, таким, как MS SQL Server и Oracle, так что Access способна служить средством разработки масштабируемых систем клиент-сервер.

1.7.2.7 Возможности запросов и инструментальные средства разработки прикладных программ

СУБД ориентированные на разработчика обладает развитыми средствами для создания приложения. К элементам инструментария разработки приложения можно отнести :

- мощные языки программирования;
 - средства реализации меню, экранных форм, ввода вывода данных
- и генерации отчетов;
- средства генерации приложений (прикладных программ);

- генерацию исполнимых файлов;

Функциональные возможности модели данных доступны пользователям СУБД, благодаря ее языковым средствам.

Языковые средства используются для выполнения двух основных функций:

- описание представления базы данных;
- выполнение операций манипулирования данными.

Первая из этих функций используется языком описания данных (ЯОД). Описание БД средствами языка описания данных называется схемой базы данных. Оно включает описание структуры БД и налагаемых на нее ограничений целостности в рамках тех правил, которые регламентированы моделью данных используемой СУБД. ЯОД не всегда синтаксически оформляется в виде самостоятельного языка. Он может быть составной частью единого языка данных, сочетающего возможности определения данных и манипулирования данными.

Язык манипулирования данными (ЯМД) позволяет запрашивать предусмотренные в системе операции над данными из базы данных.

1.7.3 Обзор современных систем управления базами данных

СУБД *Access* проста в изучении и эксплуатации и поэтому доступна для пользователей с низкой квалификацией, снабжена обширными средствами по созданию отчетов различной степени сложности, создаваемых на основе таблиц различных форматов. Как правило, *Access* используется для создания личных баз данных (справочники, записные книжки и т. д.), не имеющих коммерческого распространения.

СУБД *SQL-Server* обеспечивает высокую степень защиты данных, как от случайных потерь, так и от несанкционированного доступа, обладает развитыми средствами обработки данных и хорошим быстродействием. *SQL-Server* предназначен для хранения большого объема данных.

Visual Basic не требовательна к техническим характеристикам персонального компьютера. Так как *Visual Basic* является продуктом фирмы *Microsoft*, то легко интегрируется со всеми приложениями *Microsoft Office* и многими приложениями, интегрированными в *WINDOWS*. Предназначен *Visual Basic* для создания небольших приложений, в которых не требуются большие вычисления и серьезная обработка данных.

Visual C++ самая скоростная среда программирования, обеспечивающая выполнение расчетов и обработку данных любой сложности, совместима практически со всеми известными приложениями.

СУБД *Visual Fox Pro* предназначена для создания приложений баз данных объема предприятия, обладает хорошим быстродействием и устанавливается на различные платформы.

1.8 Постреляционные базы данных

1.8.1 Основные направления совершенствования реляционных баз данных

Рассмотрим основные направления в области исследований и разработок систем управления – так называемых постреляционных баз данных.

Первое направление связано с максимальным использованием существующих технологий управления и организации реляционных СУБД с дальнейшим совершенствованием систем управления внешней памятью.

Второе направление связано с созданием генераторов системы управления в виде наборов модулей со стандартизованными интерфейсами.

Третье направление развития СУБД по сути является синтезом первых двух направлений. СУБД проектируется как некоторый интерпретатор системы правил и набор модулей-действий, вызываемых в соответствии с этими правилами. Для таких систем разрабатываются специальные языки формирования правил.

Можно сказать, что СУБД следующего поколения – это прямые наследники реляционных систем.

Однако реляционные СУБД стали применять не только в сфере бизнеса для управленческих задач, но и в сфере промышленного производства (CALS-технологии). Применение реляционных баз данных оказалось весьма эффективным для разработки систематизированного проектирования технологических процессов (САПРТП), разработке экспертных систем и других задачах управления и технической подготовки производства.

Такие системы обычно оперируют сложно структурированными объектами – некоторым комплексом логически связанных таблиц, для анализа информации которых, а тем более для выбора рациональных технических решений приходится выполнять сложные запросы.

В соответствии с такими задачами применения реляционных БД появилось новое направление их развития. Суть этого направления сводится к тому, что в системах управления базами данных формируются сложные объекты, объединяющие в себе не только расходные таблицы БД, но и соответствующие запросы.

Это очень обширная область исследований, в которой затрагиваются вопросы разработки моделей данных, структур данных, языков запросов, управления транзакциями, журнализации и т.д. Это новое направление в разработке СУБД хотя и основывается на реляционной модели, но в ней не обязательно поддерживается требование полной нормализации отношений.

С расширением области применения реляционного подхода для решения задач, особенно в направлении CALS-технологий, стало очевидным, что применение принципа нормализации таблиц БД и создание отдельных

транзакций и процедур при выполнении различных запросов «сводит на нет» все преимущества нормализованной схемы организации базы данных.

В ненормализованных реляционных моделях данных допускается создание и хранение в качестве объекта (исходного элемента системы) кортежей (записей), массивов (регулярных индексированных множеств данных), регулярных множеств элементарных данных, а также отношений. При этом такая вложенность может быть неограниченной.

Системы управления базами данных, в которых формируются такие сложные объекты, называют *объектно-ориентированными базами данных (ООБД)*.

Генерация систем баз данных, ориентированных на приложения – это направление в развитии СУБД определяется тем, что невозможно создать универсальную систему управления базами данных, которая будет достаточна и не избыточна для применения в любом приложении. Например, если посмотреть на использование существующих СУБД для решения практических задач в производстве и бизнесе, то можно утверждать, что в большинстве случаев применяется не более чем 30 % возможностей системы. Тем не менее приложение несет всю тяжесть поддерживающей его СУБД, рассчитанной на использование в наиболее общих случаях.

Поэтому очень заманчиво производить не законченные универсальные СУБД, а нечто вроде компиляторов (compiler compiler), позволяющих собрать систему баз данных, ориентированную на конкретное приложение (или класс приложений).

Поэтому желательно уметь генерировать систему баз данных, возможности которой в достаточной степени соответствуют потребностям приложения. На сегодняшний день на коммерческом рынке такие генерационные системы отсутствуют (например, при выборе сервера системы Oracle при разработке конкретного приложения нельзя отказаться от каких-либо свойств системы).

Под оптимизацией запросов в реляционных СУБД обычно подразумевают такой способ обработки запросов, при котором по начальному представлению запроса путем его преобразований вырабатывается оптимальный процедурный план его выполнения.

Соответствующие преобразования начального представления запроса выполняются специальным компонентом СУБД – оптимизатором, и оптимальность производимого им плана выполнения запроса носит субъективный характер, поскольку критерий оптимальности заложен разработчиком в оптимизатор.

Такие инструментальные средства, обеспечивающие автоматизацию построения компиляторов имеются, например, в системах DB2, Oracle, Informix.

1.8.2 Общие понятия объектно-ориентированного подхода

Направление объектно-ориентированных баз данных появлялось в середине 1980-х гг. Наиболее активно это направление развивается в последние годы.

Возникновение направления ООБД определяется прежде всего потребностями практики – необходимостью разработки сложных информационных прикладных систем, для которых технология предшествующих систем БД не была вполне удовлетворительной.

Конечно, ООБД возникли не на пустом месте. Соответствующий базис обеспечивают как предыдущие работы в области БД, так и давно развивающиеся направления языков программирования с абстрактными типами данных и объектно-ориентированных языков программирования.

Что касается связи с предыдущими работами в области БД, то, на наш взгляд, наиболее сильное влияние на работы в области ООБД оказывают проработки реляционных СУБД и следующее (хронологически) за ними семейство БД, в которых поддерживается управление сложными объектами. Кроме того, исключительное влияние на идеи и концепции ООБД и всего объектно-ориентированного подхода оказал подход к семантическому моделированию данных. Достаточное влияние оказывают также развивающиеся параллельно с ООБД направления дедуктивных и активных БД.

В наиболее общей постановке объектно-ориентированный подход базируется на концепциях:

- объекта и идентификатора объекта;
- атрибутов и методов;
- классов;
- иерархии и наследования классов.

Любая сущность реального мира в объектно-ориентированных языках и системах моделируется в виде объекта. Любой объект при своем создании получает генерируемый системой уникальный идентификатор, который связан с объектом все время его существования и не меняется при изменении состояния объекта.

Каждый объект имеет состояние и поведение. Состояние объекта – набор значений его атрибутов. Поведение объекта – набор методов (программный код), оперирующих над состоянием объекта. Значение атрибута объекта – это тоже некоторый объект или множество объектов. Состояние и поведение объекта инкапсулированы в объекте; взаимодействуют объекты на основе передачи сообщений и выполнения соответствующих методов.

Множество объектов с одним и тем же набором атрибутов и методов образует класс объектов. Объект должен принадлежать только одному классу (если не учитывать возможности наследования). Допускается наличие примитивных предопределенных классов, объекты-экземпляры которых не

имеют атрибутов: целые, строки и т.д. Класс, объекты которого могут служить значениями атрибута объектов другого класса, называется *доменом* этого атрибута.

Допускается порождение нового класса на основе уже существующего класса – наследование. В этом случае новый класс, называемый подклассом существующего класса (суперкласса), наследует все атрибуты и методы суперкласса. В подклассе, кроме того, могут быть определены дополнительные атрибуты и методы. Различаются случаи простого и множественного наследования. В первом случае подкласс может определяться только на основе одного суперкласса, во втором случае суперклассов может быть несколько. Если в языке или системе поддерживается единичное наследование классов, то набор классов образует древовидную иерархию. При поддержании множественного наследования классы связаны в ориентированный граф с корнем, называемый решеткой классов. Объект подкласса считается принадлежащим любому суперклассу этого класса.

Одной из более поздних идей объектно-ориентированного подхода является идея возможного переопределения атрибутов и методов суперкласса в подклассе (перегрузки методов). Эта возможность увеличивает гибкость, но порождает дополнительную проблему: при компиляции объектно-ориентированной программы могут быть неизвестны структура и программный код методов объекта, хотя его класс (в общем случае – суперкласс) известен. Для разрешения этой проблемы применяется так называемый метод позднего связывания, означающий интерпретационный режим выполнения программы с распознаванием деталей реализации объекта во время выполнения посылки сообщения к нему. Введение некоторых ограничений на способ определения подклассов позволяет добиться эффективной реализации без потребностей в интерпретации.

При таком наборе базовых понятий, если не принимать во внимание возможности наследования классов и соответствующие проблемы, объектно-ориентированный подход очень близок к подходу языков программирования с абстрактными (или произвольными) типами данных.

С другой стороны, если абстрагироваться от поведенческого аспекта объектов, объектно-ориентированный подход весьма близок к подходу семантического моделирования данных (даже по терминологии). Фундаментальные абстракции, лежащие в основе семантических моделей, неявно используются и в объектно-ориентированном подходе. На абстракции агрегации основывается построение сложных объектов, значениями атрибутов которых могут быть другие объекты. Абстракция группирования – основа формирования классов объектов. На абстракции специализации (обобщения) основано построение иерархии или решетки классов.

Наиболее важным новым качеством ООБД, которое позволяет достичь объектно-ориентированного подхода, является поведенческий аспект объектов. В прикладных информационных системах, основывавшихся на БД с традиционной организацией (вплоть до тех, которые базировались на семантических моделях данных), существовал принципиальный разрыв между

структурной и поведенческой частями. Структурная часть системы поддерживалась всем аппаратом БД, ее можно было моделировать, верифицировать, а поведенческая часть создавалась изолированно. В частности, отсутствовали формальный аппарат и системная поддержка совместного моделирования и гарантирования согласованности этих структурной (статической) и поведенческой (динамической) частей. В среде ООБД проектирование, разработка и сопровождение прикладной системы становятся процессом, в котором интегрируются структурный и поведенческий аспекты. Конечно, для этого нужны специальные языки, позволяющие определять объекты и создавать на их основе прикладную систему.

Специфика применения объектно-ориентированного подхода для организации и управления БД потребовала уточненного толкования классических концепций и некоторого их расширения. Это определилось потребностями долговременного хранения объектов во внешней памяти, ассоциативного доступа к объектам, обеспечения согласованного состояния ООБД в условиях мультидоступа и тому подобных возможностей, свойственных базам данных.

Выделяют три аспекта, отсутствующих в традиционной парадигме, но требующихся в ООБД.

Первый аспект касается потребности в средствах спецификации знаний при определении класса (ограничений целостности, правил дедукции и т.п.).

Второй аспект – потребность в механизме определения разного рода семантических связей между объектами разных классов. Фактически это означает требование полного распространения на ООБД средств семантического моделирования данных. Потребность в использовании абстракции ассоциирования отмечается и в связи с использованием ООБД в сфере автоматизированного проектирования и инженерии.

Третий аспект связан с пересмотром понятия класса. В контексте ООБД оказывается более удобным рассматривать класс как множество объектов данного типа, т.е. одновременно поддерживать понятия и типа, и класса объектов.

1.8.3 Объектно-ориентированные модели данных

Первой формализованной и общепризнанной моделью данных была реляционная модель Кодда. В этой модели, как и во всех следующих, выделялись три аспекта: структурный, целостный и манипуляционный. Структуры данных в реляционной модели основываются на плоских нормализованных отношениях, ограничения целостности выражаются с помощью средств логики первого порядка, манипулирование данными осуществляется на основе реляционной алгебры или равносильного ей реляционного исчисления. Своим успехом реляционная модель данных во многом обязана тому, что она опирается на строгий математический аппарат

реляционной алгебры и теории множеств

Основные трудности объектно-ориентированного моделирования данных связаны с тем, что не существует конкретного математического аппарата, на который могла бы опираться общая объектно-ориентированная модель данных. Разработка методов управления данными внутри объектов, как и любой процесс программирования нетрадиционных задач, остается «искусством программирования».

Методы могут быть *публичными* (доступными из объектов других классов) или *приватными* (доступными только внутри данного класса).

Итак, объектно-ориентированная система управления базами данных представляет собой объединение системы программирования и СУБД и основана на объектно-ориентированной модели данных.

Основное назначение ООБД связано с потребностью создания единого информационного пространства.

В этой среде должны отсутствовать противоречия между структурной и поведенческой частями проекта и должно поддерживаться эффективное управление сложными структурами данных во внешней памяти.

В отличие от традиционных реляционных систем, в которых при создании приложения приходится одновременно использовать процедурный язык программирования, ориентированный на работу со скалярными значениями, и декларативный язык запросов, ориентированный на работу с множествами, языковая среда ООБД – это объектно-ориентированная система программирования, естественно включающая в себя средства работы с долговременными объектами. Естественность включения средств работы с БД в язык программирования означает, что работа с долговременными (храняемыми во внешней БД) объектами должна происходить на основе тех же синтаксических конструкций (и с той же семантикой), что и работа с временными объектами, существующими только во время работы программы.

Эта сторона ООБД наиболее близка родственному направлению языков программирования баз данных. Языки программирования ООБД и БД во многих своих чертах различаются только терминологически; существенным отличием является лишь поддержание в языках ООБД подхода к наследованию классов.

Другим аспектом языкового окружения ООБД является потребность в языках запросов, которые можно было бы использовать в интерактивном режиме. Если доступ к объектам внешней БД в языках программирования ООБД носит в основном навигационный характер, то для языков запросов более удобен декларативный стиль. Как известно, декларативные языки запросов менее развиты, чем языки программирования.

2 Организация баз данных в Visual Fox Pro

2.1 Работа с таблицами

2.1.1 Создание таблицы

На этапе создания табличного файла надо описать структуру таблицы, а уже потом в созданную структуру заносить информацию. Табличный файл можно создать следующими способами:

А) С помощью команд

В этом случае в окне Command надо подать команду *CREATE* и в появившейся на экране диалоговой панели Create задать в поле ввода Enter table имя создаваемому табличному файлу и указать место его хранения в раскрывающемся списке Папка, если это необходимо.

После нажатия кнопки *Сохранить* на экран выводится диалоговая панель Table Designer для описания структуры таблицы, которая содержит две вкладки: Table (для создания таблицы) и Index (для создания индексного файла). После выбора вкладки Table можно приступить к описанию структуры табличного файла.

Каждое поле структуры таблицы описывается одной строкой и для него заполняются четыре параметра (характеристики). Назначение параметров поля:

- Name
e – имя поля. Пишется буквами латинского алфавита, первый символ – обязательно буква, далее можно использовать цифры и знак подчеркивания.

- Type
– тип поля. Указывается один из допустимых типов поля либо с клавиатуры, либо открывается раскрывающийся список и на экран выводится список допустимых типов полей, из которого выбирается нужный тип.

- Width
h – указывается желаемый размер поля, но не больше допустимого.

- Decimal
mal – количество разрядов дробной части числа. Поле доступно только при использовании типа Numeric.

- Null
– запрет пустого (нулевого) значения поля.

Когда все поля структуры описаны, надо сохранить созданную структуру таблицы (пустую) на диске нажатием кнопки Ok. При этом на экран выводится сообщение «Input data record now?».

Если нажать кнопку No, то на диске, по указанному адресу, будет сохранена пустая структура таблицы. Если нажать кнопку Yes, то на экран будет выведен пустой шаблон записи для занесения данных в созданной таблице.

Данные в шаблон заносятся по правилам текстовых редакторов. По окончании ввода всех данных (всех записей), их надо сохранить на диске с помощью одной из команд: *Ctrl + End* или *Ctrl + W*. Если данные сохранять не

надо, то нажимают комбинацию клавиш *Ctrl + Q* или клавишу Esc.

Б) С помощью Главного меню

Из Главного меню подать команду *File > New* и в появившейся на экране диалоговой панели *New* в радиогруппе *File Type* выбрать кнопку *Table*, а затем нажать кнопку *New File* (рисунок 15).

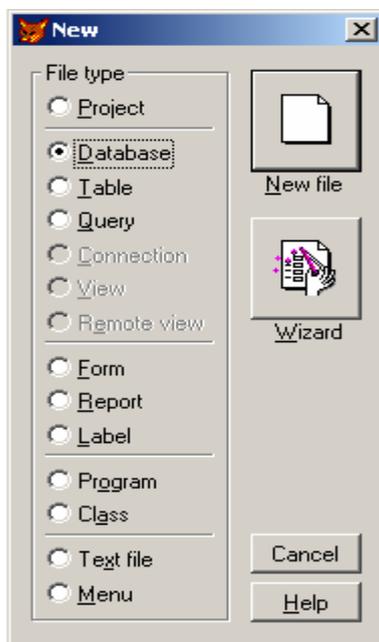


Рисунок 15 – Диалоговая панель *New*

Затем на экран выводится диалоговая панель *Create*, описанная выше. Описание структуры таблицы производится аналогично.

В) С помощью SQL- оператора

Сокращенный формат SQL-оператора для создания таблицы имеет вид:

CREATE TABLE <имя таблице> (<имя поля1> <тип поля1> (<размер по ля1>) [, <имя поляN> <тип поляN> (<размер поляN>),...])

2.1.2 Открытие таблицы

В Fox Pro следует различать два понятия: открытие таблицы и предъявление таблицы на экран. В каждый момент времени может быть открыто много таблиц (их количество определяется версией FoxPro). На экран предъявляется, как правило, одна таблица.

Открыть ранее созданную таблицу можно двумя способами: с помощью команд или с помощью главного меню.

А) С помощью команд

В окне *Command* надо подать команду *USE* <имя_таблицы>. Это краткий вариант команды открытия таблицы. При этом содержимое таблицы на экран не предъявляется. Для того чтобы определить, открыта таблица или нет, надо на экран вывести статус-строку, в которой будет указано имя открытой таблицы

Полный формат команды:

```
USE [<имя табличного_файла>] [IN <рабочая область>]
  [AGAIN]
  [INDEX <список_имен_индексных_файлов>]
  [ORDER[<Выр.N>|<IDX-файл>|TAG<имя_тега>]][OF<CDX-
файл>]]
  [ASCENDING | DESCENDING]
  [ALIAS <псевдоним>]
  [NOUPDATE]
  [EXCLUSIVE]
  [SHARED]
```

Назначение опций:

IN – указывается номер рабочей области, в которой будет открыта таблица. В Visual FoxPro допускается одновременное открытие до 25 таблиц в разных рабочих областях. Рабочие области имеют порядковый номер от 1 до 25, но первые 10 рабочих областей дополнительно имеют однобуквенные имена от А до J.

AGAIN – разрешает повторное открытие таблицы в другой рабочей области, но только в режиме просмотра (Read Only).

INDEX – открывает одновременно с таблицей один или несколько индексных файлов, принадлежащих данной таблице. Причем первый индексный файл в списке считается главным (текущим), если другой файл не определен в опции ORDER.

ORDER – переназначает главный (текущий) индексный файл, путем указания его номера (<Выр.N>) в <списке_имен_индексных_файлов>. При необходимости можно указать тип индексного файла (IDX или CDX), а также имя тега (TAG).

ASCENDING или *DESCENDING* - задает порядок чтения индекса, соответственно по возрастанию и по убыванию.

NOUPDATE – в открываемой таблице запрещены любые изменения (устанавливается режим Read Only).

EXCLUSIVE — открываемая таблица предназначена для личного использования при работе в локальной сети.

SHARED — открываемая таблица предназначена для общего использования при работе в локальной сети.

Если команда *USE* используется без опций, то она закрывает все таблицы.

Б) С помощью Главного меню

Из Главного меню подают команду File > Open и на экран выводится диалоговая панель Open, где указывается имя таблицы и место ее хранения.

Часто при написании программных кодов надо знать алиас конкретной таблицы. Функция *ALIAS()* возвращает в символьном виде значение алиаса указанной таблицы.

```
ALIAS (<имя рабочей области> | <имя таблицы>)
```

Если указанная рабочая область пуста, то функция *ALIAS()* возвращает

пустое значение.

2.1.3 Модификация структуры таблицы

Модифицировать можно текущую таблицу, при этом из существующей структуры можно удалить ненужное поле (поля), добавить в структуру новое поле (поля), изменить характеристики существующих полей. При изменении характеристик существующих полей возможна потеря информации в некоторых случаях. Например, если тип поля изменяется с Character на Numeric, или при уменьшении размера поля и т. д.

Из окна Command надо подать команду *MODIFY STRUCTURE*. Все команды в Visual FoxPro можно подавать в сокращенном варианте написания, указывая не менее четырех букв от каждого слова имени команды, например MODI STRU. В результате на экран выводится диалоговая панель Table Designer. Изменения структуры проводятся по описанным выше правилам с использованием приемов текстовых редакторов. С целью избежания потерь информации рекомендуется перед применением команды *MODIFY STRUCTURE* сделать резервную копию таблицы.

2.2 Работа с записями в таблице

2.2.1 Команда редактирования Browse

Команда *BROWSE* выводит на экран окно BROWSE , куда помещает содержимое таблицы. Эта команда предоставляет широкие возможности для доступа к данным: просмотр, редактирование, добавление и удаление. Если позволяет оперативная память, то одновременно может быть открыто несколько окон BROWSE, в каждое из которых будет помещена своя таблица. Количество одновременно открытых окон BROWSE также зависит от версии FoxPro.

При работе с базой данных часто возникает необходимость исправить данные в таблицах, то есть выполнить редактирование данных. Предварительно открывается нужный табличный файл. Вывести на экран таблицу можно в двух режимах: режим Browse (рисунок 16) и режим Edit (рисунок 17) . В режиме Browse в окно BROWSE выводится несколько записей. Количество записей определяется размером окна BROWSE, при этом возможна прокрутка как по горизонтали, так и по вертикали. В режиме Edit в окно BROWSE выводится одна запись, при этом записи можно пролистывать вперед (клавиша PgDn) и назад (клавиша PgUp).

Установить режимы Browse и Edit можно несколькими способами.

А) С помощью команд

для задания режима Browse надо в окне Command подать команду *BROWSE*;



Num_zach	Surname	Name	Class
4541	Иванов	Сергей	10П
1230	Петров	Андрей	41П
1321	Сидорова	Марина	35М
1414	Подрядова	Ксения	44П
1215	Алексеев	Николай	153

Рисунок 16 - Режим Browse

для задания режима Edit надо в окне Command подать команду *EDIT* или *CHANGE*.

Б) С помощью Главного меню

для задания режима Browse надо подать команду View -> Browse;

для задания режима Edit надо подать команду View -> Edit.



Num_zach	4541
Surname	Иванов
Name	Сергей
Klass	10П
Num_zach	1230
Surname	Петров
Name	Андрей
Klass	41П
Num_zach	1321
Surname	Сидорова
Name	Марина
Klass	35М
Num_zach	1414
Surname	Подрядова
Name	Ксения
Klass	44П
Num_zach	1215
Surname	Алексеев
Name	Николай
Klass	153
Num_zach	
Surname	
Name	
Klass	

Рисунок 17 - Режим Edit

2.2.2 Добавление новой записи в таблицу

В FoxPro новая запись добавляется в конец таблицы. Можно добавлять как одну, так и несколько записей. Предварительно открывают табличный файл, а затем одним из способов подают команду добавления записи.

А) с помощью команд

в окне Command подают команду *APPEND*.

Б) с помощью Главного меню

Б) Из Главного меню подают одну из команд:

Table -> Append New Record — для добавления одной записи.

Table -> Append Mode — для добавления нескольких записей.

При подаче любой из команд на экран выводится окно BROWSE в режиме Edit, которое содержит пустой шаблон записи. Если надо установить режим Browse, то из Главного меню следует подать команду View -> Browse.

2.2.3 Удаление записи из таблицы

В FoxPro удаление записи из таблицы производится за два шага: пометка к удалению и физическое удаление записи. На первом шаге производится пометка записи специальным символом — прямоугольник черного цвета перед первым полем записи. Помеченные к удалению записи остаются в таблице и значения полей этих записей участвуют во всех вычислениях. Если в алгоритме приложения предусмотреть проверку на пометку записи к удалению, то можно либо полностью, либо частично исключить помеченные записи из процесса вычислений, что очень удобно в некоторых случаях (например, при начислении заработной платы и т. д.). На втором шаге все помеченные к удалению записи удаляются одной командой. При этом оставшиеся в таблице записи перемещаются вверх, занимая освободившееся место, и, таким образом, размер таблицы уменьшается.

2.2.3

.1 Пометка записи к удалению

Пометить запись к удалению можно следующими способами:

А) С помощью мыши

Щелкнуть левой кнопкой мыши в поле выделения. Поле выделения находится слева от первого поля записи и представляет собой прямоугольник серого цвета. Если запись помечена к удалению, то поле выделения окрашивается черным цветом (рисунок 18).

Если запись помечена к удалению ошибочно, то для снятия пометки надо еще раз щелкнуть левой кнопкой мыши на поле выделения, при этом поле

выделения будет закрашено серым цветом.



Num_zach	Surname	Name	Klass
4541	Иванов	Сергей	10П
1230	Петров	Андрей	41П
1321	Сидорова	Марина	35М
1414	Подрядова	Ксения	44П
1215	Алексеев	Николай	15З

Рисунок 18 - Записи, помеченные к удалению

Б) С помощью клавиатуры

Предварительно курсор устанавливают на любое поле помечаемой к удалению записи и нажимают клавиши Ctrl + T. После выполнения команды поле выделения окрашивается черным цветом.

Для снятия пометки к удалению надо установить курсор на любое поле помеченной записи и нажать клавиши Ctrl + T.

В) С помощью главного меню

Из Главного меню надо подать команду Table -> Toggle Deletion Mark и на экран выводится диалоговая панель Delete.

Если запись или группа записей помечена к удалению ошибочно, то для отмены пометки надо подать команду Table -> Recall Records и на экран выводится диалоговая панель Recall, которая полностью аналогична диалоговой панели «Delete».

Г) Программно.

Для пометки записей к удалению надо дать команду *DELETE*. Формат команды:

```
DELETE [ FOR <L> ] [ WHILE <L> ] [ NOOPTIMIZE ]
```

Команда *DELETE* без опций помечает к удалению одну текущую запись.

2.2.3.2 Физическое удаление записи

Физическое удаление помеченных записей можно выполнить двумя способами.

А) С помощью команд

В окне Command надо подать команду *PACK*.

Б) С помощью Главного меню

Из Главного меню надо подать команду Table -> Remove Deleted Records.

В том и другом случае помеченные записи будут удалены, а оставшиеся в таблице записи автоматически перемешаются вверх, занимая освободившееся место.

2.3 Работа с базами данных

2.3.1 Создание и открытие базы данных

Создавать и работать с табличным файлом можно как с независимым объектом, хранящимся внутри каталога, так и как с частью базы данных. Для того чтобы таблица была частью базы данных, надо либо сначала создать файл базы данных (или открыть файл базы данных), а затем создавать таблицы (описанными выше командами), либо специальными командами в открытый файл базы данных поместить (добавить) ранее созданные таблицы. Если файл базы данных создан, то создаваемая база данных будет реляционной. В противном случае база данных не реляционная. Создать файл базы данных можно несколькими способами:

А) С помощью команд.

В окне Command надо подать команду:

```
CREATE DATABASE [ <имя базы данных> | ? ]
```

Команда создает базу данных и делает ее текущей (открытой). Если указана опция «?» или опции не указаны вообще, то на экран выводится диалоговая панель Create, где в специальном окне представлены имена имеющихся баз данных, и в поле ввода Enter database можно задать имя создаваемой базы данных.

Б) С помощью Главного меню.

Из Главного меню надо подать команду: File → New, тогда на экран выводится диалоговая панель New, где в радиогруппе File Type надо включить кнопку Database и нажать кнопку New File, тогда на экран выведется диалоговая панель Create, работа с которой описана в пункте *а*.

В) С помощью графического меню.

В графическом меню нажать кнопку New File и на экран выведется диалоговая панель New. База данных, созданная любым способом, становится текущей (открытой). Файл базы данных имеет расширение .DBC.

Для открытия базы данных надо из Главного меню подать команду File → Open или в графическом меню нажать кнопку Open, на экран выведется диалоговая панель Open, где надо указать имя базы данных и место ее хранения. В результате работы команды Open на экран выведется диалоговая панель Database Designer (рисунок 19).

2.3.2 Добавление таблиц в базу данных

Для того чтобы поместить в открытую базу данных ранее созданную свободную таблицу, надо выполнить одно из действий:

- либо из Главного меню подать команду Database → Add Table;
- либо установить курсор мыши на рабочее поле диалоговой панели

Database

Designer и правой кнопкой мыши вызвать на экран контекстное меню, из которого выбрать команду Add Table.

В том и другом случае на экран выводится диалоговая панель Open в окне которой представлены имена табличных файлов из текущего каталога. Надо выбрать имя нужного табличного файла и нажать кнопку Ok. После выполнения операции добавления таблицы, изображение самой таблицы появляется в диалоговой панели Database Designer (рисунок 19).

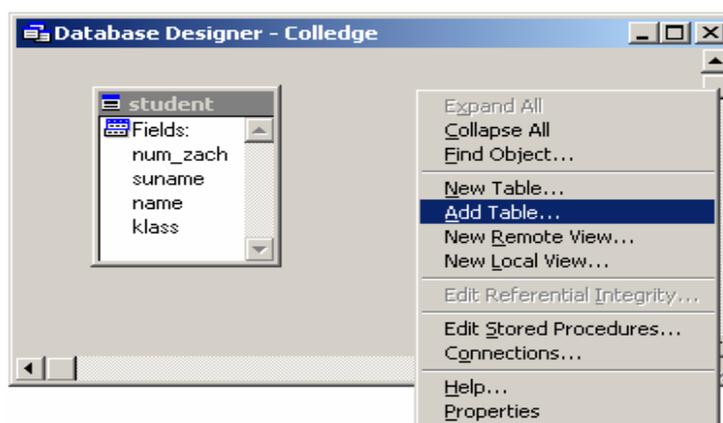


Рисунок 19 - Добавление таблицы в базу данных

Добавить *свободную* таблицу в открытую базу данных можно также с помощью команды `ADD TABLE <имя таблицы> | ?`

2.3.3 Освобождение таблиц

Для освобождения таблицы существуют две команды.

Команда `REMOVE TABLE <имя таблицы> ? [DELETE]` удаляет таблицу из базы данных и делает указанную таблицу свободной, оставляя ее в текущем каталоге. Если указана опция `DELETE`, то таблица удаляется не только из базы данных, но и с диска.

Команда `FREE TABLE <имя таблицы>` используется при закрытой базе данных для доступа к указанной таблице. Если опция `<имя таблицы>` опущена, то на экран выводится диалоговая панель Free Table, которая аналогична диалоговой панели Open, где можно указать имя освобождаемой таблицы.

Освободить таблицу можно через Главное меню командой Database -> Remove.

После выбора команды `Remove` надо уточнить свое действие: освободить таблицу (Remove) или удалить таблицу (Delete) нажатием соответствующей кнопки (рисунок 20).

После нажатия одной из кнопок (Remove или Delete) надо подтвердить

(или отменить) выбранное действие нажатием кнопки Yes (или No) в появившемся на экране новом запросе.

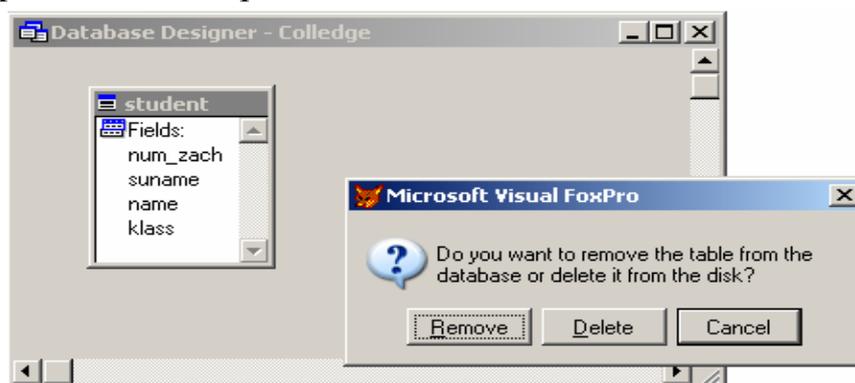


Рисунок 20 - Освобождение (или удаление) таблицы

2.4 Индексирование базы данных

Если записей в таблице много, то найти нужную запись бывает очень трудно. Поиск данных производится методом перебора, то есть просматриваются все записи таблицы от первой записи до последней записи, что приводит к большим затратам времени. Чтобы облегчить поиск данных в таблице, используют индексы.

2.4.1 Виды индексов и индексных файлов

Индекс, иногда его называют указатель, представляет собой порядковый номер записи в таблице. Индекс строится по значениям одного поля или по значениям нескольких полей.

Индекс, построенный по значениям одного поля, называется *простым*, а по значениям двух и более полей — *сложным*. Во время построения индекса записи в таблице сортируются по значениям поля (или полей) будущего индекса. Затем первой строке таблицы присваивается индекс номер один, второй строке — индекс номер два и т. д. до конца таблицы.

Как простой так и сложный индекс имеют свой тип (Type). *Первичный (Primary) индекс (ключ)* - это поле или группа полей, однозначно определяющих запись, т.е. значения первичного индекса уникальны (не повторяются). В реляционной БД каждая таблица может иметь только один первичный ключ. *Внешних ключей* у таблицы может быть много и они могут иметь один из типов:

- Candidate — кандидат в первичный ключ или альтернативный ключ. Он обладает всеми свойствами первичного ключа.
- Unique (уникальный) — допускает повторяющиеся значения в поле, по которому он построен, но на экран будет выводится только одна первая запись из группы записей с одинаковым значением индексного поля.
- Regular (регулярный) — не накладывает никаких ограничений на

значения индексного поля и на вывод записей на экран. Индекс только управляет порядком отображения записей. Это наиболее популярный тип индекса.

Взаимосвязь между таблицами осуществляется по индексам, которые называют ключами.

Построенный индекс хранится в специальном индексном файле. Если индексный файл хранит только один индекс, то он называется *одноиндексным* и имеет расширение .idx. Индексные файлы, которые хранят много индексов, называются *мультииндексными* и имеют расширение .cdx. Каждый индекс, который хранится в мультииндексном файле, называется *тегом*. Каждый тег имеет свое уникальное имя.

Мультииндексные файлы бывают двух типов: просто мультииндексные файлы (о них рассказано выше) и структурные мультииндексные файлы. *Структурный мультииндексный файл* имеет одинаковое имя с таблицей, которой он принадлежит (отличие только в расширении файла), и обладает следующими свойствами:

- автоматически открывается со своей таблицей;
- его нельзя закрыть, но можно сделать не главным.

Одна таблица может иметь много индексных файлов как одноиндексных, так и мультииндексных. В старших версиях FoxPro используются мультииндексные файлы.

2.4.2 Создание индекса

Создать индекс можно двумя способами.

А) С помощью команды:

```
INDEX ON <индексное выражение> TO <idx-файл> | TAG<имя тега>  
[OF <cdx-файл>]
```

```
[FOR <условие>]
```

```
[COMPACT]
```

```
[DESCENDING]
```

```
[UNIQUE]
```

```
[ADDITIVE]
```

```
[NOOPTIMIZE]
```

Назначение опций:

<индексное выражение> — имя поля (или полей), по значениям которого надо построить индекс. При построении сложного индекса имена полей перечисляются через знак + (плюс). Если сложный индекс построен по:

- числовым полям, то индекс строится по сумме значений полей;
- символьным полям, то индекс строится сначала по значению первого поля, а при повторяющихся значениях первого поля — по значениям второго поля; при повторяющихся значениях первого и второго полей — по значениям третьего поля и т. д.;
- по полям разных типов, то сначала значения полей приводят к

одному типу, как правило символному, а затем строят индекс.

Длина индексного выражения не должна превышать 254 символа.
TO <idx-файл> — указывается имя одноиндексного файла.

TAG <имя тега> [*OF* < cdx-файл>] — указывается имя тега в мультииндексном файле. Если используется опция [*OF*], то создаваемый тег помещается в указанный мультииндексный файл, а если требуемый мультииндексный файл отсутствует, то будет построен структурный мультииндексный файл. Если опция [*OF* < cdx-файл>] опущена, то созданный тег будет помещен в текущий мультииндексный файл.

FOR <условие> — устанавливает режим отбора в индекс тех записей таблицы, которые удовлетворяют <условию>.

COMPACT — управляет созданием компактного одноиндексного файла. В старших версиях FoxPro не используется.

DESCENDING — строит индекс по убыванию. По умолчанию используется построение индекса по возрастанию (*ASCENDING*). Для одноиндексных файлов можно построить индекс только по возрастанию. Если перед использованием команды *INDEX ON...* подать команду *SET COLLATE*, то можно построить одноиндексный файл по убыванию.

UNIQUE — строит уникальный индекс. Если индексное поле (поля) содержит повторяющиеся значения, то в индекс попадает только одна первая запись и остальные записи будут не доступны.

ADDITIVE — вновь создаваемый индексный файл не закрывает уже открытые к этому моменту времени индексные файлы. Если опция опущена, то вновь создаваемый индексный файл закрывает все ранее открытые индексные файлы,

NOOPTIMIZE — отключает использование технологии *Rashmore* для ускоренного доступа к данным.

Б) С помощью Главного меню

В этом случае индекс создается либо при создании таблицы, либо при модификации структуры таблицы. Для этого в диалоговой панели *Table Designer* надо выбрать вкладку *Index*.

Каждый индекс описывается одной строкой в окне диалоговой панели *Table Designer*.

В графе *Name* указывается имя тега мультииндексного файла. Если ранее открыт один из мультииндексных файлов, то вновь построенный индекс помещается в открытый мультииндексный файл. Если индекс строится одновременно с созданием табличного файла или табличный файл не имеет мультииндексных файлов, то вновь построенный индекс помещается в автоматически создаваемый структурный мультииндексный файл.

2.4.3 Команды для работы с индексными файлами

2.4.3.1 Замена текущего индекса

Для каждой таблицы одновременно может быть открыто несколько индексных файлов, но текущим (активным) будет только один индекс. По умолчанию принято, что текущим будет первый по порядку индекс в том индексном файле, имя которого указано первым в списке имен индексных файлов команды *USE* или команды *SET INDEX TO*.

Текущим можно сделать любой индекс из текущего индексного файла с помощью команды

```
SET ORDER TO  
[<ВырN1> | <idx-файл> | [TAG <имя тега> [OF <cdx - файл>] ]  
[IN <ВырN2> | <вырC>]  
[ASCENDING | DESCENDING]
```

Назначение опций:

<вырN> - задает текущий индекс по его порядковому номеру в мультииндексном файле.

<idx - файл> - делает текущим одноиндексный файл.

TAG <имя тега> [*OF* <cdx - файл>] - задает текущий индекс по имени тега из указанного мультииндексного файла. Если опция [*OF* <cdx-файл>] опущена, то тег выбирается из текущего мультииндексного файла.

IN <ВырN2> - указывает номер рабочей области, в которой находится индексный файл. Опция используется в том случае, если табличный файл открыт в одной рабочей области, а индексный файл - в другой рабочей области.

Текущим индекс также можно сделать с помощью диалоговой панели *Table Designer*, переместив строку описания нужного индекса на первое место.

2.4.3.2 Перестройка индексных файлов

При внесении изменений в большие таблицы тратится много времени, так как при внесении каждого изменения заново перестраиваются все открытые индексные файлы. Для экономии времени индексные файлы закрывают и вносят изменения в таблицу. Однако в этом случае возникает несоответствие между обновленной таблицей и индексными файлами. Для устранения указанного несоответствия надо заново перестроить индексные файлы. После открытия всех индексных файлов, принадлежащих измененному табличному файлу, надо подать команду *REINDEX*. Команда действует на все индексные файлы, открытые в текущей рабочей области. Переиндексирование можно также выполнить, подав из Главного меню команду *Table -> Rebuild Indexes*.

2.4.3.3 Преобразование одноиндексного файла в тег

Если табличному файлу принадлежит один или несколько одноиндексных файлов, то их можно скопировать как теги в мультииндексный файл. Для этих целей используют команду

```
COPY INDEXES <имена idx - файлов> | ALL [TO <cdx - файл>]
```

Опция *ALL* указывается в том случае, если надо скопировать все одноиндексные файлы. При этом список имен <имена idx-файлов> не указывается. Тегам присваиваются имена одноиндексных файлов. При копировании нескольких одноиндексных файлов их имена перечисляются через запятую. Если опция *TO* опущена, то одноиндексные файлы копируются в текущий мультииндексный файл. Если опция *TO* содержит имя несуществующего мультииндексного файла, то он создается.

Допустима и обратная операция, то есть один тег преобразуется (копируется) в одноиндексный файл с помощью команды:

```
COPY TAG <список имен тегов> {OF <cdx-файл> } TO <idx-файл>]
```

Предварительно мультииндексный файл должен быть открыт. Можно скопировать отдельные теги, указав <список имен тегов>, либо все теги, используя опцию *ALL*.

2.4.3.4 Удаление тега из мультииндексного файла

Удалить один тег или все теги из мультииндексного файла, открытого в любой рабочей области, можно с помощью команды

```
DELETE TAG <имя тега 1> [OF <имя cdx-файла> ]  
[, <имя тега 2> [OF <имя cdx-файла> ] ] ... |  
ALL [OF <имя cdx-файла>]
```

Одной командой можно удалить теги, находящиеся в разных мультииндексных файлах.

2.4.3.5 Вывод на экран имен индексных файлов и имен тегов

Во время работы с приложением часто возникает необходимость получить справку, имеет ли таблица индексные файлы, а также узнать имена тегов мультииндексных файлов. Для этих целей предназначены следующие функции.

Функция возвращает имена открытых одноиндексных файлов:

```
NDX(<выр.N> [, <номер рабочей области | псевдоним рабочей области> ])
```

Функция возвращает имена открытых мультииндексных файлов:

```
CDX(<выр.N> [, <номер рабочей области | псевдоним рабочей области> ])
```

Функция возвращает имя структурного мультииндексного файла:

MDX(<выр.N> [, <номер рабочей области | псевдоним рабочей области>])

Во всех функциях <выр.N> - порядковый номер индексного файла, имя которого должна вернуть функция.

TAG([<имя sdx-файла> , <выр.N> [, <номер рабочей области | псевдоним рабочей области>]])

Функция возвращает для указанного <имя sdx-файла> мультииндексного файла имя тега <выр.N >, заданного порядковым номером. Если <имя sdx-файла> опущено, то по умолчанию подставляется имя текущего мультииндексного файла.

2.5 Создание взаимосвязей между таблицами в Visual Fox Pro

2.5.1 Организация взаимосвязей

В FoxPro допускается одновременная работа с несколькими таблицами. Версия Visual FoxPro 3.0 имеет возможность работать одновременно с 25 таблицами. При этом каждая таблица помещается в свою рабочую область, и между таблицами могут быть установлены взаимосвязи. Указатели записей во взаимосвязанных таблицах будут двигаться синхронно. Таблица, в которой указатель перемещается произвольно, называется старшей. Таблица, в которой указатель перемещается в соответствии с перемещением указателя в старшей таблице, называется подчиненной или младшей. Если таблица имеет первичный индекс (ключ), то она может быть родительской. Если таблица имеет внешний индекс (ключ), то она может быть дочерней. В один и тот же момент времени одна таблица может быть родительской по отношению к одной таблице, и дочерней — по отношению к другой таблице. Если несколько записей из первой таблице по одинаковым значениям внешнего ключа ссылаются по единственному значению первичного ключа на одну запись во второй таблице, то первая таблица будет дочерней, а вторая таблица — родительской. Другими словами: одна запись в родительской таблице порождает несколько записей в дочерней таблице.

Для установления взаимосвязи и родительская и дочерняя таблицы должны иметь хотя бы одно общее поле. Поле будет общим для старшей и младшей таблицы, если оно имеет:

- одинаковое имя
- одинаковый тип
- одинаковый размер в той и другой таблице.

Допускается подключать к одной старшей таблице несколько младших таблиц. При этом для каждой пары таблиц должно быть свое общее поле. Возможен и частный случай, когда несколько таблиц объединяются по одному

общему полю. На практике встречаются три типа объединения таблиц: параллельное, последовательное и смешанное (рисунок 21).

Перед установлением взаимосвязей между таблицами надо выполнить следующие условия:

- все таблицы должны быть открыты, причем каждая в своей рабочей области;
- все таблицы попарно должны иметь общее поле (хотя бы одно);
- желательно, чтобы для общего поля и в старшей и в младшей таблице был построен индекс, но в одной таблице индекс должен быть построен обязательно.

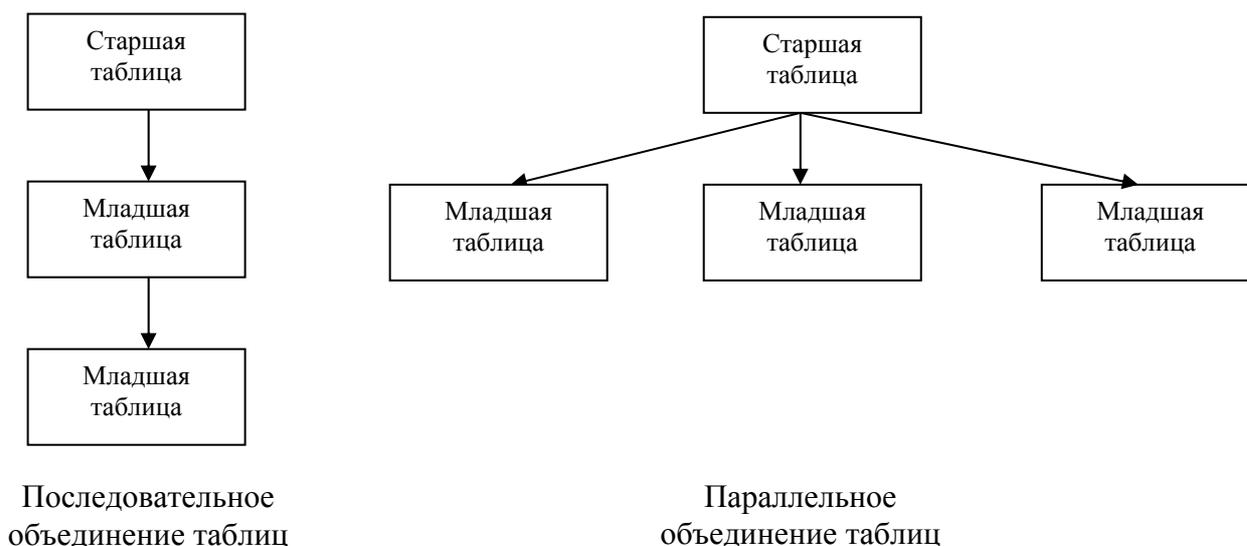


Рисунок 21 - Примеры объединения таблиц

Существует несколько подходов для установления взаимосвязи между таблицами:

- конструкция SET RELATION TO;
- конструкция SET SKIP TO;
- команда JOIN;
- команда UPDATE;
- команда TOTAL;
- переменные памяти.

В каждом конкретном случае, с целью повышения эффективности работы программы, используют тот или иной подход при установлении взаимосвязей между таблицами. Переменные памяти позволяют сохранить значения полей таблиц даже после их закрытия. Переходя от одной таблицы к другой и сохраняя необходимые значения полей, можно обеспечить обработку любого количества таблиц.

2.5.2 Понятие рабочей области

Каждая таблица вместе со своими индексными файлами открывается в отдельной рабочей области. Количество рабочих областей зависит от версии FoxPro. Все рабочие области имеют порядковые номера. Например, от 1 до 25 для версии Visual FoxPro 3.0. Кроме того, первые десять рабочих областей имеет однобуквенные имена от A до J. Не все команды FoxPro могут обрабатывать порядковые номера рабочих областей, поэтому при открытии табличного файла рабочей области можно присвоить псевдоним (ALIAS). Псевдоним пишется буквами латинского алфавита, причем первый символ обязательно буква, а последующие символы могут быть и буквами, и цифрами, допускается и символ подчеркивания. Пробелы при составлении псевдонима не допустимы. Желательно в качестве псевдонима использовать осмысленное сочетание из трех-четырех букв. Более длинные псевдонимы желательно не использовать, так как при написании программных кодов обращение к таблицам производится по псевдонимам рабочих областей, в которых они открыты. При открытии таблиц не обязательно соблюдать строгую нумерацию рабочих областей. В любой момент времени может быть открыто много рабочих областей, но только одна рабочая область будет текущей или активной. Для назначения текущей рабочей области используют команду:

```
SELECT <номер рабочей области | псевдоним рабочей области>
```

Для доступа к полям таблицы используются префиксы. Для текущей рабочей области префиксом является точка. Для пассивных рабочих областей префиксом является конструкция из двух символов -> минуса и знака больше. Для старших версий FoxPro, начиная с 3.0, допускается использование точки как одного префикса для всех рабочих областей. При обращении к какому-либо полю следует указать псевдоним рабочей области, в которой открыта таблица, затем префикс и потом имя нужного поля.

2.5.3 Установление взаимосвязи «один-к-одному»

При установлении взаимосвязи, как правило, в родительской таблице берут первичный ключ, а в дочерней таблице — внешний ключ. Причем эти оба ключа являются общим полем. Текущей делают ту рабочую область, где находится дочерняя таблица, и с помощью команды *SET RELATION TO* к дочерней таблице подключают одну или несколько родительских таблиц, открытых в пассивных рабочих областях.

Формат команды:

```
SET RELATION TO [<выр.1> INTO <псевдоним рабочей области 1> ] [ ,  
<выр.2> INTO <псевдоним рабочей области 2> ] [...] [ADDITIVE]
```

Назначение опций:

<выр.1> <выр.2>... — имена общих полей между дочерней таблицей и несколькими родительскими таблицами.

INTO <псевдоним> — псевдонимы пассивных рабочих областей, где открыты соответствующие родительские таблицы.

ADDITIVE — добавляет вновь созданные взаимосвязи к уже имеющимся взаимосвязям.

По умолчанию ранее установленные взаимосвязи разрываются. Даже если в дочерней таблице общим полем является внешний ключ типа Regular и имеется несколько записей с одинаковым значением внешнего ключа, на экран будет выводиться только одна запись из дочерней таблицы, так как установлен тип взаимосвязи «один-к-одному».

Для разрыва всех взаимосвязей «один-к-одному» активной таблицы надо подать команду *SET RELATION TO* без опций.

Если надо разорвать одну конкретную взаимосвязь между активной (дочерней) таблицей и пассивной (родительской) таблицей, то следует подать команду

SET RELATIONS OFF INTO <псевдоним рабочей области> | <номер рабочей области>

2.5.4 Установление взаимосвязи «один-ко-многим»

Взаимосвязь «один-ко-многим» устанавливается «поверх» взаимосвязи «один-к-одному», то есть сначала устанавливают между таблицами взаимосвязь «один-к-одному», а затем подают команду:

SET SKIP TO [<псевдоним рабочей области 1> [, <псевдоним рабочей области 2>] ...]

После выполнения этой команды на экран будет выводиться для одной записи из родительской таблицы несколько записей из дочерней таблицы.

Для удаления всех взаимосвязей «один-ко-многим» надо подать команду *SET SKIP TO* без опций.

2.5.5 Установление взаимосвязи с помощью главного меню

Если имеется реляционная база данных, то установить взаимосвязи между таблицами можно с помощью команд главного меню. Для этого предварительно в каждой таблице строится первичный ключ и внешние ключи. Затем выводят на экран диалоговую панель Table Designer. Потом курсор мыши размещают на имени первичного ключа родительской таблицы и буксируют его внутрь дочерней таблицы, устанавливая на имя соответствующего внешнего ключа. Во время буксировки курсор мыши дважды меняет свою форму. По окончании буксировки на экран выводится диалоговая панель Edit Relationship, где надо проверить, а при необходимости уточнить параметры взаимосвязи.

После нажатия кнопки **Ok** на экран выводится диалоговая панель **Database Designer**, между именами соответствующих индексов автоматически прорисовывается прямая линия. Со стороны «один» линия начинается с символа «+», а со стороны «много» — с символа «->» (рисунок 22). Обратная буксировка (от дочерней таблицы к родительской) не допустима.

Тип взаимосвязи между таблицами устанавливается автоматически, в зависимости от типа индексов. Если оба индекса (в родительской таблице и дочерней таблице) уникальные, то возникает тип взаимосвязи «один-к-одному». Если в родительской таблице индекс уникальный, а в дочерней таблице индекс регулярный, то получается тип взаимосвязи «один-ко-многим».

Чтобы редактировать существующую взаимосвязь, надо вызвать на экран диалоговую панель **Edit Relationship**. Для этого надо курсор мыши установить на линию взаимосвязи и щелкнуть левой кнопкой мыши, при этом линия взаимосвязи утолщается.

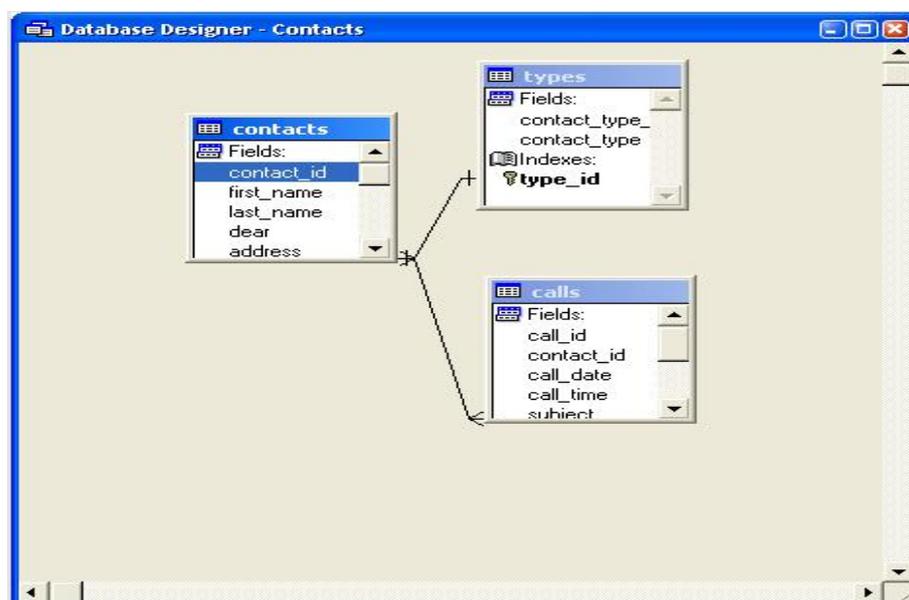


Рисунок 22 - Установление взаимосвязи между таблицами

Затем правой кнопкой мыши на экран вызывают контекстное меню, из которого выбирают команду **Edit Relationship**.

Для удаления взаимосвязи ее надо предварительно выделить, а затем нажать клавишу **Del**.

Одним из главных условий создания базы данных является обеспечение условий ссылочной целостности. Выполнение этих условий гарантирует помещение в базу данных только достоверной информации. Ссылочная целостность обеспечивается механизмом каскадных воздействий: каскадное удаление и каскадное редактирование ключевых полей. Каскадное удаление — это одновременное удаление записи из родительской таблицы с удалением соответствующих записей из всех дочерних таблиц. Удаление записи из дочерней таблицы не влечет за собой удаления соответствующей записи из родительской таблицы. Аналогично, при изменении значения поля (полей)

первичного ключа, необходимо изменить значение соответствующего поля (полей) внешнего ключа дочерней таблицы (или таблиц), то есть каскадное редактирование.

Для установления каскадных воздействий надо выделить линию взаимосвязи щелчком левой кнопки мыши и вызвать на экран контекстное меню, щелкнув правой кнопки мыши, когда курсор удерживается на линии связи. В появившемся на экране контекстном меню надо выбрать команду Referential Integrity. На экран выводится диалоговая панель Referential Integrity Builder (рисунок 23).

В верхней части диалоговой панели Referential Integrity имеются графы:

- Parent Table — содержит имена родительских таблиц. Указателем черным треугольником) отмечена текущая родительская таблица.
- Child Table — содержит имена дочерних таблиц, которые могут быть соединены с соответствующими родительскими таблицами.
- Update — содержит один из возможных типов каскадных воздействий при редактировании поля (полей) первичного ключа у соответствующей пары таблиц (имена таблиц указаны в левых графах).
- Delete — содержит один из возможных типов каскадных воз действий при удалении записи из родительской таблицы.
- Insert — содержит один из возможных типов каскадных воз действий при вставке записи в родительскую таблицу.
- Parent Tag — содержит имя тега в родительской таблице. В графе справа (Child Tag) будет указано имя соответствующего тега дочерней таблицы
- Child Tag — содержит имя тега в дочерней таблице. В графе слева (Parent Tag), будет указано имя соответствующего тега родительской таблицы.

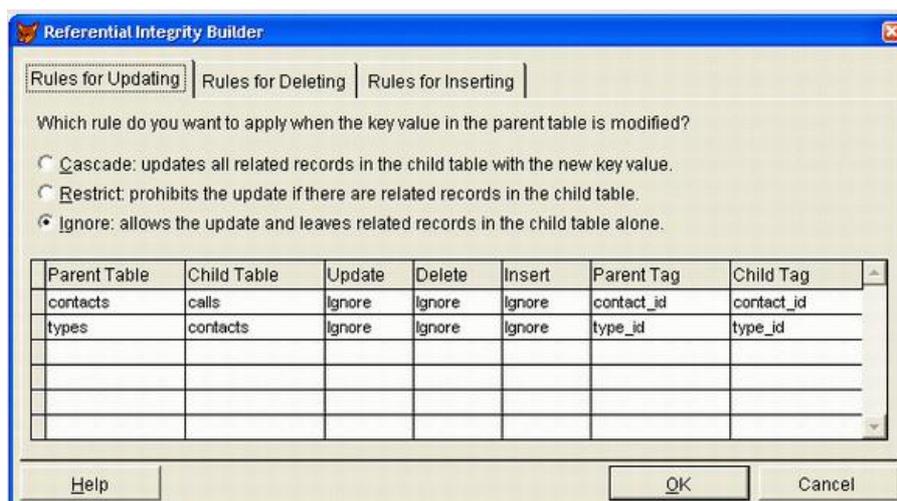


Рисунок 23 - Задание условий ссылочной целостности

Установить соответствующий тип каскадных воздействий можно либо в верхней части диалоговой панели, либо в нижней части диалоговой панели.

В верхней части диалоговой панели в одной из граф Update, Delete или Insert курсором мыши выделяют нужное поле (клетку). В поле появляется кнопка раскрывающегося списка, в котором можно выбрать одно из значений:

Cascade — при изменении значения поля (полей) первичного ключа или альтернативного ключа (Candidate) в родительской таблице автоматически выполняется каскадное изменение значения соответствующего поля (полей) внешнего ключа дочерней таблицы (таблиц).

Restrict — запрещает изменение значения поля (полей) первичного или альтернативного ключа в родительской таблице, если в дочерней таблице (таблицах) имеются соответствующие значения во внешнем ключе.

Ignore — допускаются любые изменения в значениях первичного и альтернативного ключей в родительской таблице. Условия ссылочной целостности не контролируются.

Аналогично работают с графами Delete и Insert, но для графы Insert возможны только два воздействия: Restrict и Ignore. В нижней части диалоговой панели имеются три вкладки:

Rules for Updating — управление каскадным редактированием записей;

Rules for Deleting — управление каскадным удалением записей;

Rules for Inserting — управление каскадным добавлением записей.

Тип воздействия Cascade, Restrict и Ignore устанавливается с помощью соответствующей радиокнопки.

2.5.6 Команды для работы с таблицами в БД

2.5.6.1 Объединение двух табличных файлов в один файл

В некоторых случаях требуется информацию из разных таблиц свести в одну таблицу. Одна таблица размещается в активной рабочей области, а вторая таблица — в пассивной области. В результате объединения будет получен новый (третий) табличный файл, который будет записан по указанному адресу. В состав нового (третьего) табличного файла могут быть включены любые поля из двух объединяемых таблиц. Для этих целей используют команду:

JOIN WITH <псевдоним пассивной рабочей области> *TO* <имя создаваемого файла>

[*FOR*]

[*FIELDS* <список имен полей>] [*NOOPTIMIZE*]

В опции *FIELDS* указываются имена полей как из активной рабочей области, так и из пассивной рабочей области. Если опция *FIELDS* опущена, то в создаваемый файл записываются все поля обеих таблиц.

Отметим особенности выполнения команды:

- Если в пассивной таблице нет записи с каким-либо значением общего поля, а в активной таблице такое значение общего поля имеется, то

запись из активной рабочей таблицы не помещается в результирующую таблицу.

- Если в активной или пассивной таблице имеются записи с дублирующим значением общего поля, то в результирующий файл помещаются все найденные записи.

2.5.6.2 Корректировка данных в связанных таблицах

Иногда бывает нужно внести большое количество изменений в одну таблицу используя данные из другой таблицы. Для этого корректируемую таблицу надо поместить в активную рабочую область, а таблицу-источник для исправления данных — в пассивную рабочую область. Для этих целей предусмотрена команда:

```
UPDATE ON <имя общего поля>,  
FROM <псевдоним пассивной рабочей области>  
REPLACE <имя поля1 в активной рабочей области> WITH  
<выражение 1>  
[, <имя поля 2 в активной рабочей области> WITH < выражение 2> ... ]  
[RANDOM]
```

Назначение опций.

ON — указывается имя общего поля. Причем обе таблицы должны быть проиндексированы по этому полю в порядке возрастания.

FROM — указывается псевдоним пассивной рабочей области, в которую помещается таблица-источник данных корректировки.

REPLACE — указывается имя поля в активной таблицы, куда надо внести изменения в соответствии с <выражением>, указанным в опции *WITH* и составленным по значениям полей пассивной таблицы.

К недостаткам этой команды следует отнести отсутствие контроля на достоверность данных после внесения изменений. Для обеспечения достоверности должны быть предприняты дополнительные меры.

2.5.6.3 Создание итогового табличного файла

При создании отчетов необходимо иметь итоговые суммы по некоторым числовым столбцам (полям). Команда *TOTAL* создает итоговый табличный файл, содержащий суммы по указанным полям. Предварительно табличный файл должен быть отсортирован по ключевому полю. При работе команды подсчитывается сумма по указанным числовым полям, поля других типов копируются в итоговый файл.

Формат команды:

```
TOTAL ON <имя ключевого поля> TO <имя итогового файла>
```

[SCOPE]
[FIELDS <список имен полей >]
[FOR <выр. L1>]
[WHILE <выр. L2>]
[NOOPTIMIZE]

2.6 Управление базами данных

Как правило, база данных состоит из нескольких таблиц. Чтобы получить полную информацию об одной сущности, надо прочитать запись во всех таблицах базы данных. Между таблицами могут быть установлены взаимосвязи, что существенно облегчает работу с такой базой данных. Если таблицы разрознены, то возникают дополнительные проблемы по управлению такими таблицами.

Базы данных могут содержать очень большой объем информации - до нескольких миллионов записей. На поиск нужных данных в таких таблицах уходит много времени. Поэтому предусмотрены специальные методы и команды для работы с базами данных.

2.6.1 Сортировка данных

Очень удобно организовать поиск и просмотр данных, если они отсортированы.

Сортировка данных — это их упорядочение по значениям одного или нескольких полей. Упорядочение может быть выполнено как по возрастанию, так и по убыванию.

Если данные сортируют по возрастанию, то на первое место ставится запись, имеющая наименьшее значение по указанному полю (полям). На второе место — запись, имеющая следующее значение поля (полей) и так далее. На последнем месте будет запись, имеющая наибольшее значение указанного поля (или полей). Числовые поля сортируются от наименьшего числа до наибольшего, символьные поля — от первой буквы алфавита до последней буквы алфавита, поля дат — от наименьшей (наиболее ранней) даты до наибольшей (самой поздней) даты.

При сортировке данных по убыванию размещение записей производится в обратном порядке, по отношению к предыдущему (по возрастанию). В FoxPro сортировка производится следующими способами:

А) В соответствии с индексом.

Сначала индекс делают текущим, а затем выводят таблицу на экран или обновляют содержимое экрана. Если построен сложный индекс, то и сортировка будет выполнена по полям, входящим в индекс. Это наиболее

удобный вид сортировки, так как для своего выполнения требует мало времени, не требует дополнительных затрат памяти и исключена потеря информации.

Б) С помощью команды SORT

В этом случае сортировку таблицы можно произвести по любому полю (или полям), даже если по указанному полю (или полям) нет индекса.

```
SORT TO <имя файла> ON <имя поля 1> [/A] [/D] [/C]  
    [, <имя поля 2> [/A] [/D] [/C]... ]  
    [ASCENDING | DESCENDING]  
    [SCOPE]  
    [FOR <выр.L1>]  
    [WHILE <выр.L2>]  
    [FIELDS <список имен полей>]  
    [NOOPTIMIZE]
```

Назначение опций:

<имя файла> — имя табличного файла (таблицы), который надо отсортировать.

ON ... — указывается одно или несколько полей, по значениям которых надо выполнить сортировку таблицы. Каждое поле может иметь уточняющие параметры:

/A — сортировка по возрастанию. Этот параметр принят по умолчанию. */D* — сортировка по убыванию.

/C — игнорирует значение регистра для символьных полей (строчные и прописные буквы считаются одинаковыми).

ASCENDING — задает сортировку по возрастанию только для полей, не имеющих параметров */A* и */D*.

DESCENDING — задает сортировку по убыванию только для полей, не имеющих параметров */A* и */D*.

FIELDS — в списке перечисляются имена полей, которые надо поместить во вновь создаваемый табличный файл. Если опция опущена, то в создаваемый файл помещаются все поля из исходного табличного файла.

При использовании этой команды на диске дополнительно сохраняется отсортированный файл, на построение которого требуется дополнительное время. Эта команда используется редко.

2.6.2 Поиск данных

Поиск данных часто используемая функция при работе с базами данных.

В FoxPro предусмотрены две методики поиска данных: метод последовательного перебора и метод деления пополам.

2.6.2.1 Поиск методом полного перебора

При поиске методом полного перебора просматривается вся таблица от первой записи до последней.

Поиск производится по любому полю таблицы.

LOCATE FOR [SCOPE] <условие поиска>

[WHILE <выр.L>]

[NOOPTIMIZE]

Назначение опций:

<условие поиска> — пишется имя поля для поиска значения, далее указывается один из логических знаков *<, >, <=, >=, =* или *<>* и само искомое значение. Если искомое значение имеет сим вольный тип, то оно указывается в двойных кавычках. Если искомое значение имеет тип дата, то оно указывается в апострофах. Если искомое значение имеет числовой тип, то выделять его знаками не требуется.

Если поиск закончился успешно, то курсор устанавливается на найденную запись. Для того чтобы увидеть найденную запись, надо подать команду *DISPLAY* или *BROWSE*. Для нахождения следующей записи, по тому же искомому значению, надо подать команду *CONTINUE*. Если поиск закончился неудачно, то курсор устанавливается на последнюю запись и на экран выводится сообщение: *End of Locate Scope*.

Этот поиск можно выполнить с помощью команды Главного меню

Table Go to Record Locate.

На экран выводится диалоговая панель *Locate Record* (рисунок 24).

Назначение кнопок:

For — выводит на экран диалоговую панель для задания условия поиска.

Score и *While* — выводят на экран диалоговые панели для задания ограничений количества просматриваемых записей.

В случае успешного и неуспешного поиска курсор принимает соответствующее положение. Для нахождения следующей записи, соответствующей условию поиска, надо повторить эту процедуру еще раз. Если по полю поиска построен индекс, то автоматически будет использоваться команда *SEEK*.

2.6.2.2 Поиск по полю текущего индекса

Таблица делится на две части пополам, берется средняя запись и значение поля текущего индекса сравнивается с искомым значением. Если значение поля индекса меньше, чем искомое значение, тогда поиск продолжается в нижней половине таблицы. (Если значение поля текущего индекса больше, чем искомое значение, то поиск продолжается в верхней половине таблицы.) Нижнюю половину таблицы делят пополам, сравнивают среднее значение поля индекса с искомым значением и так далее, до тех пор

пока не найдут нужное значение или не просмотрят всю таблицу. Для каждой методики поиска предусмотрена своя команда.

А) Поиск с помощью окна Command

В этом случае надо воспользоваться одной из команд:

SEEK <искомое значение>

FIND <искомое значение>

Для написания <искомого значения> используются те же символы выделения, что и в команде *LOCATE*. В случае удачного и не удачного поиска команда *SEEK* управляет курсором так же, как и команда *LOCATE*. Так как курсор устанавливается на первую найденную запись и записи отсортированы по полю поиска, то на экран желательно выводить несколько записей, начиная от текущей.

Б) Поиск с помощью команды Главного меню

Предварительно таблицу, в которой надо найти какое-либо значение, делают текущей. Затем из Главного меню подают команду Edit -> Find. На экран выводится диалоговая панель Find (рисунок 24).

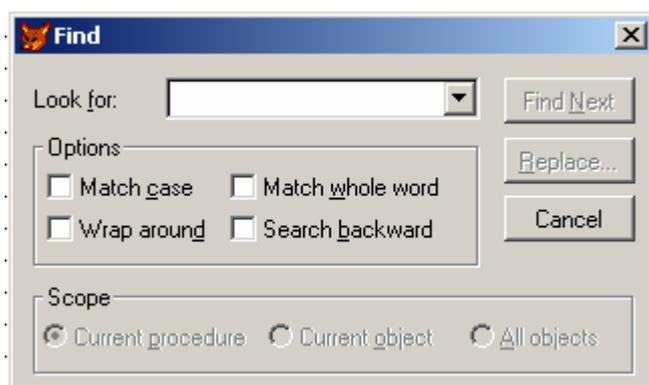


Рисунок 24 - диалоговая панель Find

В поле ввода Look For вводится значение для поиска. Для поиска заданного значения просматриваются все поля всех записей подряд. При обнаружении первого значения запись делается текущей, то есть на нее устанавливается указатель (черный треугольник слева от первого поля).

Флажки в группе Options задают следующие условия поиска:

- Match Case — различие регистров. Прописные и строчные буквы считаются разными.

- Wrap Around — круговой поиск. При достижении конца таблицы поиск начинается заново с первой строки.

- Match Whole Word — точное соответствие полному значению поиска.

При выключенном флажке значение поиска может входить как часть в найденное слово.

- Search Backward — поиск в обратном направлении (снизу вверх).

Кнопка Find Next предназначена для нахождения следующего значения, удовлетворяющего условию поиска.

2.6.3 Фильтрация данных

В FoxPro определены два вида фильтров данных: фильтр для строк, когда ограничивается количество строк, предъявляемых на экран, и фильтр для полей, когда ограничивается количество полей, отображаемых на экране и, следовательно, доступных для редактирования.

2.6.3.1 Ограничение на количество строк

Если таблица большая (несколько сотен записей), то работать с ней не удобно. Поэтому производят фильтрацию данных, то есть выборку и предъявление на экран группы записей, отвечающих определенным требованиям. С выборкой можно проводить все операции редактирования. По окончании работы с выборкой исходную таблицу восстанавливают в полном (первоначальном) виде.

Для установки фильтра данных используют команду:

```
SET FILTER TO <выр. L>
```

В опции <выр. L> указывают имя поля и его значение, по которым надо выполнить фильтрацию. Для снятия фильтра и восстановления первоначального вида таблицы используют ту же команду, но без опции.

2.6.3.2 Ограничение на количество полей

Фильтрация полей применяется при работе с длинными записями, которые имеют длину более одного экрана. При редактировании таких записей на экран выводят те поля, в которые надо внести изменения, и одно-два поля, идентифицирующих запись. Фильтрация полей выполняется в два этапа: на первом этапе определяется список полей, а на втором — фактическая установка фильтра.

Для определения списка полей используют команду:

```
SET FIELDS TO [<список полей> | ALL [LIKE <маска> | EXCEPT <маска>]]
```

После выполнения этой команды, из текущей таблицы для установки фильтра будут отобраны либо поля, имена которых указаны в <списке полей>, либо все поля. По умолчанию принята опция ALL — все поля. Опция LIKE включает в список полей все поля, имена которых удовлетворяют маске (шаблону). Опция EXCEPT включает в список полей все поля, имена которых не удовлетворяют маске (шаблону).

Если повторно подать эту же команду с другим <списком полей>, то вновь определенный список полей будет добавлен к существующему списку полей. Команда *SET FIELDS TO* без опции закрывает все поля, открытые предыдущими командами *SET FIELDS TO*.

Для установки фильтра полей используют команду *SET FIELDS ON | OFF | LOCAL | GLOBAL*.

Назначение опций:

ON — устанавливает фильтр для ранее определенных полей.

OFF — отменяет список полей и разрешает отображение всех полей таблицы. По умолчанию установлена опция *OFF*.

LOCAL — определяет, что для фильтра доступны только поля текущей таблицы.

GLOBAL — разрешает отображать поля всех таблиц, между которыми установлены реляционные отношения (установлены взаимосвязи).

После выполнения команд, описанных в п. 2.6.3.1 и п. 2.6.3.2, команды *DISPLAY* и *BROWSE* будут выдавать на экран записи и поля в соответствии с установленными фильтрами.

3 Программирование в Visual FoxPro

3.1 Работа с программными файлами

3.1.1 Создание программного файла

При работе с базой данных часто бывает необходимо при каждом новом запуске представлять на экране одну и ту же базу данных в различных вариантах. Процесс конфигурирования базы данных достаточно длительный. Для облегчения этого процесса используют программные файлы. В каждом программном файле записывают и сохраняют на диске один из вариантов конфигурации базы данных. Впоследствии, запуская программный файл на выполнение, получают желаемую конфигурацию базы данных.

Программный файл создается с помощью встроенного текстового редактора FoxPro.

Текстовый редактор можно запустить как из окна Command, так и из Главного меню.

А) Чтобы запустить текстовый редактор из окна Command, в этом окне надо задать команду:

```
MODIFY COMMAND | FILE [<имя файла> | ?] [NOEDIT] [NOMENU] [NOWAIT] [RANGE, ] [[WINDOW <имя окна 1>] [IN [WINDOW <имя окна 2>] | IN SCREEN] [SAME] [SAVE]
```

Ключевое слово *COMMAND* предназначено как для создания нового программного файла (имя файла не указывается) с расширением *.prg*, так и для открытия уже существующего файла (имя файла указывается либо полностью, либо частично — по шаблону).

Ключевое слово *FILE* предназначено только для открытия существующего программного файла, причем имя файла надо указывать полностью.

Если имя файла не известно, то вместо имени файла можно указать символ «?» и на экран будут выведен список имен программных файлов (с расширением .prg), где можно выбрать имя нужного файла.

Назначение опций:

NOEDIT — разрешает просмотр текста, но редактирование текста запрещено;

NOMENU — запрещает вызов системного меню окна редактирования, то есть блокирует клавишу F10 и кнопку в левом верхнем углу окна редактирования;

NOWAIT — после открытия окна редактирования разрешает работу (без пауз) программы, вызвавшей открытие окна редактирования. В противном случае работа программы прерывается до закрытия окна редактирования.

RANGE , *<выр.N2>* — при работе с Мето-полем позволяет выбрать часть текста в указанном интервале от до;

WINDOW *<имя окна 1>* — открывает окно редактирования с характеристиками другого предварительно открытого окна *<имя окна 1>*;

IN WINDOW *<имя окна 2>* — открывает окно редактирования внутри окна *<имя_окна2>*.

IN SCREEN — опция задана по умолчанию. Открывает окно редактирования внутри главного окна FoxPro.

SAME — если текст программы, текст документа или Мето-поле уже открыты, то запрещает новое редактирование в окне редактора.

SAVE — опция используется только в программах и сохраняет на экране образ окна редактирования после его закрытия.

Встроенный редактор FoxPro предназначен также для вывода на экран и редактирования Мето-полей, которые открываются с помощью команды Ctrl + Home, и текстовых файлов, имена которых указаны в тексте программного файла с целью их дальнейшего вывода на печать.

Б) Для создания нового программного файла из Главного меню надо подать команду:

File New выбрать расширение .prg (или Program).

Для открытия уже существующего программного файла надо подать команду:

File -> Open указать имя нужного программного файла.

В том и другом случае написание и редактирование текста программного файла производится по правилам текстовых редакторов

3.1.2 Запуск программного файла

Выполнить программный файл можно либо из окна Command с помощью команды *DO <имя программного файла>*, либо с помощью команды Главного меню:

PROGRAM -> DO -> указать в списке имя программного файла, либо нажав в графическом меню кнопку, на которой изображен восклицательный знак.

Остановить выполнение программного файла можно нажатием клавиши Esc, если в тексте программного файла указана команда *SET ESCAPE ON*.

Во многих версиях FoxPro эта команда указана по умолчанию.

В Visual FoxPro после нажатия клавиши Esc, на экран выводится диалоговая панель с тремя кнопками:

Cancel — выполнение программы прерывается.

Suspend — выполнение программы приостанавливается до подачи команды RESUME.

Ignore — продолжить выполнение программы.

В уже отлаженных и хорошо работающих программах можно заблокировать нажатие клавиши Esc с помощью команды *SET ESCAPE OFF*.

3.1.3 Структура программы

Программный файл обычно содержит следующие элементы:

1) *Заголовок программного файла* — позволяет легко ориентироваться в большом количестве программ. Поскольку это комментарий, строка заголовка начинается символом * и не компилируется.

Заголовок может выглядеть так:

.....
Наименование программы : Search.prg

* Назначение: поиск по введенному параметру

* Вызывается из программы main.prg

* Программа начата 25.03.2008

* Дата последней модификации 14.04.2008
.....

2) *Установочная часть программы* - содержит команды SET, определяющие окружение: озвучивание, цветовую гамму, форматы, подавление вывода ненужных сообщений. После компиляции выполненные в программе настройки изменить нельзя. Например, если файл, который вы хотите создать, уже существует, на экране появляется вопрос, перезаписать ли этот файл? При положительном ответе файл будет обновлен. Для того чтобы при выполнении программы такое сообщение не выводилось, в ней должна быть команда *SET SAFETY OFF*. Кроме того, в установочной части записываются команды очистки экрана, макроса, окон и других результатов, полученных ранее. Ниже показан пример установочной части программы.

CLEAR

CLEAR MEMORY

CLEAR ALL

CLOSE ALL

CLEAR MACROS

SET TALK OFF
SET COLOR OF SCHEME 1 TO 5
SET CONFIRM ON
SET BELL OFF
SET DATE GERMAN
SET ESCAPE OFF
SET COMPATIBLE OFF
SET SCOREBOARD OFF
SET HOURS TO 24
SET CLOCK ON
SET EXACT ON
SET SAFETY OFF
SET SYSMENU ON
SET PROCEDURE TO roz_pro
SET AUTOSAVE ON

.....
3) *Операционная часть программы* — содержит команды открытия файлов, выбора рабочих областей, обработки и отображения информации.

4) *Заключительная часть программы* — приводит рабочее пространство в исходное состояние, удаляет временные переменные и закрывает открытые файлы. Например:

POP KEY
CLEAR ALL
CLEAR
ON KEY
CLOSE ALL
RELEASE WINDOWS ALL

3.1.4 Модульность программ

Программный продукт включает в себя внешние и внутренние процедуры, которые оформляются как отдельные модули. Модульное построение программного продукта позволяет повысить его наглядность и унифицировать часто повторяющиеся операции, сокращает время написания и отладки программ. При выполнении программных кодов, если FoxPro встречает обращение к подпрограмме (модулю), то он ищет текст модуля в следующей последовательности:

- 1) в текущей процедуре;
- 2) в процедурном файле;
- 3) снизу вверх в старших процедурах;
- 4) на диске в виде отдельной программы.

3.1.4.1 Внешние процедуры

Внешней процедурой называется некоторая последовательность команд, которые выполняют определенное и законченное действие по обработке данных. Внешняя процедура (или несколько процедур) хранятся в отдельном файле на диске. Приложение может содержать несколько процедурных файлов, но подключен (активен) всегда только один процедурный файл. Подключение процедурного файла производится командой:

SET PROCEDURE TO [*<имя процедурного файла>*]

Команда без опции закрывает процедурный файл.

Вызов внешней процедуры. Формат команды:

DO *<имя программного файла внешней процедуры>*
[*WITH* *<список параметров>*] [*IN* *<имя файла>*]

Назначение опций:

WITH *<список параметров>* — содержит список входных и выходных параметров. В качестве входных и выходных параметров допускается использование переменных, констант и выражений.

IN *<имя файла>* — явно указывает место хранения программного файла и используется вместо предварительно подаваемой команды *SET PROCEDURE*.

Команда *DO* выполняет указанную программу. Если указанная программа не откомпилирована или в нее были внесены изменения, то она автоматически компилирует указанную программу, а затем ее выполняет.

Если в *<имени программного файла>* опущено расширение, то расширения будут просматриваться в следующем порядке: .exe, .app, .fcr, .prg.

Команда описания заголовка процедуры. Формат команды:
PROCEDURE *<имя процедуры>*

В качестве имени процедуры используется уникальный набор букв латинского алфавита. Допускается использование цифр, но первым символом должна быть буква.

Команда восприятия параметров. Формат команды:

PARAMETERS *<список параметров>*

Эта команда позволяет передать процедуре значения переменных из командной строки или из вызывающей программы. Она пишется сразу после заголовка процедуры и содержит такое же количество параметров и того же типа, что и в команде *DO*, вызывающей данную процедуру.

Команда окончания процедуры. Формат команды:

RETURN [*TO MASTER* | *TO* *<имя процедуры>* | *<выр>*]

Команда *RETURN* без опций завершает выполнение процедуры и передает управление следующему по порядку оператору.

Команда *RETURN TO MASTER* завершает выполнение процедуры и выполняет переход к самому верхнему уровню вызывающих процедур, передавая управление следующему по порядку оператору.

Команда *RETURN TO* *<имя процедуры>* завершает выполнение процедуры и выполняет переход к процедуре с указанным именем, передавая

управление следующему по порядку оператору.

<выр> содержит последний выполняемый оператор в теле процедуры. Такой способ окончания процедуры используется в процедурах-функциях для передачи последнего вычисленного значения. Команда возврата *REENTRY* позволяет вернуться на ту команду, которая вызвала текущую процедуру, т. е. циклическая ссылка. Иногда работой алгоритма предусмотрен многократный вызов одной и той же процедуры, но всегда надо предусмотреть принудительный выход из такой ситуации. Команда выхода на командный уровень *CANCEL* прекращает выполнение программы, освобождает из памяти переменные, созданные пользователем и возвращает управление окну Command. Команда выхода из среды Visual FoxPro *QUIT* осуществляет выход из среды Visual FoxPro в операционную систему. Рекомендуется использовать эту команду перед выключением компьютера, с целью предотвращения потери данных.

3.1.4.2 Внутренние процедуры

Внутренние процедуры хранятся вместе с текстом основной программы и размещаются в конце основной программы.

Для создания внутренних процедур используются те же команды, что и при создании внешних процедур.

3.1.4.3 Процедура-функция

Если результатом работы процедуры является единственное вычисленное значение, то удобнее использовать процедуру-функцию. Процедура-функция может быть оформлена как внешняя или как внутренняя процедура. Процедура-функция вызывается по своему имени, которое допускается указывать в командах. После имени процедуры-функции обязательно указываются или пустые круглые скобки, или круглые скобки с именами передаваемых аргументов.

Команда описания заголовка процедуры-функции:

FUNCTION(<имя процедуры-функции>)

Имя процедуры-функции пишется по правилам имен процедур Передача входных параметров (аргументов) и написание тела процедуры-функции производится аналогично процедурам, но запрещено использование команды RETURN в любом варианте. Единственный результат вычисления, определённый последним исполняемым оператором, передается под именем самой функции.

Вызов процедуры-функции производится командой:

DO <имя процедуры-функции>

3.2 Отладка программы в Visual FoxPro

3.2.1 Диалоговые окна Program Error и Data Session

Visual FoxPro 7.0 является языком интерпретирующего типа, поэтому программа при наличии ошибок выполняется до первой из них, а затем открывается окно с выделенной строкой, содержащей ошибку. Кроме того, выводится диалоговое окно *Program Error* кратким описанием причины ошибки и кнопками *Cancel*, *Suspend*, *Ignore*, *Help*. Щелчок на одной из этих кнопок обеспечивает соответственно переход в программный код для исправления ошибки и выхода из программы, перевод задачи в состояние ожидания с возможностью возврата к выполнению, игнорирование ошибки, получение контекстной справки.

Наиболее часто диагностируются такие синтаксические ошибки, как недостающие скобки в выражениях (*Missing*), несоответствие типов данных (*Mismatch*), использование необъявленных переменных, ошибочный ввод наименований таблиц, полей, переменных или зарезервированных слов (*Unrecognized*). Эти ошибки легко исправятся нажатием клавиши *Cancel*. Исправленный текст программы сохраняется, а ее выполнение повторяется. Но случаются ошибки, возникающие, например, при недостаточном объеме свободного дискового пространства, при выполнении недопустимой операции или использовании неправильного алгоритма. Многих ошибок можно избежать, если в начале программы выполнить тестирование используемых внешних свойств, текущего состояния системы. Например, при работе с данными значительного объема следует убедиться в том, что объем свободного дискового пространства достаточен. С этой целью используют функцию *DISKSPACE()*.

Выполнение программы можно приостановить и посмотреть состояние обрабатываемых данных с помощью окна *Data Session*. При использовании программы в обратном сеансе *Data Session*, следует выбрать нужный сеанс в ниспадающем списке *Current Session*. В окне *Data Session* можно увидеть, какие файлы открыла программа, структуру таблиц, порядок сортировки данных и выполнить необходимые изменения.

3.2.2 Программа отладчик

При работе со сложными программами можно использовать программу отладчика, которая позволяет трассировать программу, т.е. выполнять ее пошагово, устанавливать останова в сомнительных местах, просматривать и изменять значения используемых переменных.

Для работы с отладчиком выполните команду *Tools>Debugger* и в открывшемся окне *Visual FoxPro Debugger* - команду *Debug>Do*, чтоб определить выполняемую программу.

Для настройки отладчика используется вкладка *Debug* диалогового окна *Options*, предоставляющая возможность выбора одного из окон для отладки программы.

- *Trace*. Определяет программный блок, содержащий ошибку, устанавливает точки останова (*Breakpoint*) в предполагаемых местах возникновения ошибки и т.д.

- *Watch*. Наблюдает за значением выражения в процессе выполнения программы.

- *Locals*. Позволяет просматривать значения локальных переменных, элементов массива, объектов.

- *Call Stack*. Позволяет отлаживать вложенные программы.

- *Debug Output*. Сохраняет отладочную информацию в текстовом файле.

Любое из этих окон или все одновременно могут быть открыты при выборе соответствующих кнопок панели инструментов .

Кроме того, на панели инструментов имеются следующие кнопки.

- *Open*. Выбор отлаживаемой программы.

- *Resume*. Продолжение выполнения программы.

- *Step Into*. Пошаговое выполнение программы.

- *Step Over*. Процедурное выполнение программы.

- *Step Out*. Выполнение всей программы и выход из нее.

- *Run to cursor*. Выполнение программы до текущего положения курсора.

- *Toggle breakpoint*. Включение точек останова в отлаживаемую программу.

- *Clear all breakpoints*. Удаление всех точек останова.

- *Breakpoint dialog*. Работа с точками останова в программе.

- *Toggle event logging*. Выбор из списка событий, отслеживаемых при выполнении программы.

Для прекращения выполнения отлаживаемой программы выполните команду *Debug>Cancel* или *Debug>Fix*, и она выведет предупреждение о прекращении.

В версии *Visual FoxPro 7.0* точки останова можно устанавливать непосредственно в текстовом редакторе при вводе текста программы. Этот сервис делается доступным, если установить опцию *Select Margin* на вкладке *Editor* команды *Options...* из меню *Tools*. При установленной *Select Margin* окно текстового редактора дополняется слева узкой полоской, на которой в нужных вам местах при помощи команды *Toggle BreakPoint* (меню правой кнопки мыши) устанавливаются точки останова .

Отладку программы можно упростить применением в "критических местах" простейшей команды вывода ? и некоторых команд *SET*.

Команда *SET DEBUG ON | OFF* помещает или удаляет соответственно из системного меню пункты для работы с окнами отладчика и трассировки. Но при *SET DEBUG OFF* окно отладчика все же будет доступным после выполнения команды *SET ECHO ON* или *ACTIVATE WINDOW DEBUG*, а окно трассировки - после *SET STEP ON* или *ACTIVATE WINDOW TRACE*.

Кроме того, при отладке программы рекомендуется выполнить команды *SET TALK ON* — для вывода на экран результата выполняемых команд, и *SET STATUS ON* — для вывода строки статуса.

Результат отладки можно сохранить в файле, выполнив команду *SET DEBUGOUT TO [<имя файла> [ADDITIVE]]*

Для прекращения записи в файл и закрытия файла выполните команду *SET DEBUGOUT TO*.

3.2.3 Собственный обработчик ошибок

Можно организовать собственный обработчик ошибок, используя команду *ON ERROR <команда>*, обеспечивающую переход к ее выполнению при возникновении ошибки. Обычно эта команда вызывает процедуру - обработчик ошибки, определяющий программы, в которой произошла ошибка, вывод диагностических сообщений, определение номера ошибочной строки программы - и устанавливает причину ошибки. Для этого в обработчик ошибки вводятся соответственно функции: *PROGRAM()*, *MESSAGE()*, *LINENO()*, *ERROR()*.

- *ERROR()* возвращает номер системной ошибки, который используется обработчиком при выполнении команды *ON ERROR*.

- *PROGRAM(<ВырN>)* определяет уровень вложенности программы — <выр N> который может достигать 128. При значении <выр N> равном 0 функция возвращает имя главной программы. *PROGRAM()* без параметров также возвращает имя текущей программы. Обработчик ошибки может установить индикатор ошибки и вывести сообщение об ошибке

- *Если* и требуется вывести номер строки программы, в которой произошла ошибка используется функция *LINENO([1])*, которая при значении параметра 1 возвращает номер строки текущей программы или процедуры, а без параметра - возвращает номер строки относительно начала главной программы. При этом также подсчитываются строки-продолжения, строки-комментарии и пустые строки. Обработчик ошибки должен включать строку,

```
ON ERROR DO r_er WITH LINENO ( )
```

а в процедуре *r_er* сообщение об ошибочной строке обеспечивает команда

```
WAIT WINDOW 'Ошибка произошла в строке: '+ALLTRIM(STR(par))
```

Для вывода сообщения о происшедшей ошибке и ошибочной строке программы в обработчике ошибки используются, соответственно, функции *MESSAGE()* и *MESSAGE(1)*. Например:

```
ON ERROR DO r_er
```

...

PROCEDURE r_er

? ' Ошибочная строка программы: '+MESSAGE(1)

? ' Номер ошибки: '+STR(ERROR())

? ' Сообщение об ошибке; '+MESSAGE()

Можно ознакомиться с содержанием функции обработки ошибок *rierror()* создаваемой системой при определении условий целостности данных на уровне базы данных (Referential Integrity).

3.2.4 Анализ работы программы

Visual FoxPro 7.0 позволяет использовать *Coverage Profiler* (Профайлер охвата) для анализа работы программы.

При выполнении анализируемой программы *Coverage Profiler* записывает в файл регистрации с расширением .log информацию: сколько раз и как долго выполнялась каждая строка программы, какие строки не выполнялись ни разу.

Для работы с *Coverage Profiler* применяется команда Tools>Coverage Profiler, в результате чего открывается одноименное окно для определения существующего файла регистрации с расширением .log.

Для создания файла регистрации можно также использовать команду *SET COVERAGE TO <имя файла с расширением .log > [ADDITIVE]*

При наличии опции *ADDITIVE* файл регистрации дополняется новой информацией, а иначе его содержимое обновляется.

Например, создадим файл регистрации but.log:

SET COVERAGE TO but.log

Теперь выполним анализируемую программу but.prg.

DO but

Посмотрим на результаты анализа, используя команду

DO (_COVERAGE)

Системная переменная *_COVERAGE* содержит имя приложения для анализа программ.

В результате выполнения команды открывается окно Coverage Profiler с анализируемой программой. В дальнейшей работе с Coverage Profiler используем кнопки панели инструментов (таблица 4).

Таблица 4 - Кнопки панели инструментов

Кнопка	Назначение
Open	Открывает файл регистрации с расширением .log
Save	Сохраняет файл регистрации
Statistic	Открывает окно со статистическими данными о программе
Add-Ins	Открывает окно для выбора и выполнения дополнительных приложений и программ
Options	Открывает окно для настройки Coverage Profiler
Coverage	Устанавливает режим просмотра программы
Mode	Устанавливает режим просмотра результата анализа программы
Profile	Устанавливает режим разделения окна на две части
Mode	

При выборе кнопки Profile Mode выводится результат анализа использования программных строк.

При выборе кнопки **Source Text Log** открывается содержимое файла регистрации. Каждая строка этого файла является перечислением параметров: время выполнения, имя используемого класса, имя программы, порядковый номер строки в программе, полное наименование файла, уровень вложенности программы .

При выборе кнопки *Source Files Skipped* открывается таблица, содержащая информацию о файлах, не включенных в файл регистрации. Кнопка *Statistics by Project* выводит на экран результат анализа всего проекта.

3.3 Основы языка программирования

В FoxPro каждый оператор должен начинаться с новой строки. Переход на новую строку позволяет распознать начало и конец отдельной команды. Для облегчения чтения длинную команду можно разделить на две (и более) строки. Чтобы указать, что следующая строка является продолжением данной строки, в конце данной строки помещают символ «;». Запрещается использовать символ «;» внутри строковых констант. Если строковую константу надо разместить в двух строках, то каждую часть строковой константы помещают в кавычки и соединяют между собой знаком «+». Если по смыслу строковая константа должна содержать символ «;», то его размещают внутри кавычек.

FoxPro игнорирует лишние пробелы, поэтому их можно использовать для удобного расположения команд FoxPro.

При написании текста программ удобно вставлять комментарии для пояснения работы конкретного блока команд. В FoxPro предусмотрены два типа комментариев.

Если символ «*» или NOTE установлены в начале строки, то это означает, что комментарий занимает всю строку. Можно после символа «*» указать несколько символов «-» для лучшего вы

деления комментария.

Если в любом месте строки текста программы встречаются символы «&&», то после них следует комментарий. То есть слева от этих символов пишется исполняемая команда, а справа — комментарий.

3.3.1 Переменные, константы и массивы

В FoxPro разрешается использовать переменные и константы тех же типов, что и поля таблицы, за Исключением типа Мемо. Значение переменной изменяется по мере выполнения программы, а значение константы всегда остается постоянным. Имена переменных и констант составляются по правилам имен полей, то есть имена пишутся буквами латинского алфавита и могут содержать цифры, но первый символ в имени — всегда буква. Длина имени не должна превышать 12 символов.

В FoxPro предусмотрены два типа переменных: пользовательские и системные. Имена системных переменных начинаются с символа « _ » подчеркивание и предназначены для хранения параметров настройки среды FoxPro. Системные переменные являются резидентными и их нельзя удалить.

Таблица 5 – Типы и размеры пользовательских переменных

Тип данных	Описание	Использование разделителей	Пример
Character	Текст от 1 до 256 символов	«текст», 'текст', [текст], "текст"	Pr = «Hello»
Currency	Денежные суммы	\$число	Pr=\$500
Date	Даты	(дата)	Pr= (25/01/02)
DateTime	Даты и Время	(дата время)	Pr= (25/01/02 11:00am)
Logical	«Истина», «Ложь»	.T. или .F.	Pr=.T.
Numeric	Числа	Без разделителей	Pr= 1550

В таблице 5 приведены часто употребляющиеся типы и размеры пользовательских переменных.

В FoxPro отсутствуют описатели типов переменных. Тип переменной определяется значением, которое она содержит. Таким образом, при выполнении программы в каждый момент времени одна и та же переменная может иметь разные типы.

Массив — это последовательность переменных, имеющих одинаковое имя и отличающихся друг от друга порядковым номером (индексом). Любой элемент массива в каждый момент времени может иметь один из допустимых

типов. В FoxPro определены одномерные и двумерные массивы. Имена массивов составляются по правилам имен переменных.

Для объявления массивов используют одну из команд DECLARE или DIMENSION, формат которых одинаковый:

DECLARE | DIMENSION <имя массива 1> (<Выр.N1> [, <Выр.N2>])
[, <имя массива 2> (<выр.N3> [, <Выр.N4>])] [...]

Пример:

DIME abc(5,10), cd(18)

Одной командой *DIMENSION* объявлены два массива: двумерный массив с именем abc, содержащий пять строк и десять столбцов, и одномерный массив cd, содержащий восемнадцать элементов массива. Тип каждого элемента массива как одномерного, так и двумерного будет определен потом, тем значением (данным), которое будет в него помещено.

3.3.2 Присвоение значений

Присвоить значения переменным и элементам массива можно несколькими способами.

С помощью оператора присваивания:

<имя переменной> = <выражение>

<имя элемента массива> = <выражение>.

С помощью команды STORE

STORE <значение> *TO* <имя переменной>

STORE <значение> *TO* <имя элемента массива>

TOR <значение> *TO* <имя массива>

Эта команда наиболее эффективна при работе с массивом (третий вариант команды), она позволяет всем элементам массива задать одинаковые (начальные) значения.

Примеры:

st = «Москва» && создается переменная *st*

*
* символного типа,

*
* в которую помещается значение «Москва»,

dt = (25/01/08)&& создается переменная *dt*

*
* типа дата, в которую помещается

*
* значение 25 января 20 08 года.

STORE 145 TO nm && создается переменная

*
* числового типа, в которую

*
* помещается значение 145.

DECL mas(12) && объявляется одномерный

*
* массив с именем *mas* из 12 элементов.

*STORE * TO mas* && все элементы массива *mas*

*
* получают значение * и символный тип.

mas(4) = 0 && четвертый элемент массива
 * *mas* получает значение 0 и тип числовой
mas(7) = (07/02/02) && седьмой элемент массива *mas*
 * получает значение 7 февраля
 * 2002 года и тип дата.

3.3.3 Команды для работы с переменными

В FoxPro допустимы следующие действия над переменными: хранение в файле, загрузка их в оперативную память, просмотр и удаление.

С помощью команды *SAVE TO* можно сохранить либо все переменные, находящиеся в оперативной памяти, либо их произвольную часть, как в файле, так и в Мето-поле с заданным именем.

Формат команды:

SAVE TO <имя файла> | *TO MEMO* <имя поля> [*ALL LIKE* | *EXCEPT* <шаблон>]

В команде предусмотрено использование одного из двух шаблонов. В любом из шаблонов допустимо использование символов "*" и "?". По умолчанию к указанному в команде имени файла, добавляется расширение .mem. По умолчанию команда создает файл с указанным именем. Если файл с таким именем уже существует, то он заменяется на новый без предупреждения. Если вы хотите увидеть предупреждение на экране, что указанный файл уже существует, надо перед командой *SAVE TO* установить команду *SET SAFETY OFF*.

Если при работе с приложением возникает необходимость разместить в оперативной памяти переменные, значения которых были определены ранее, то надо воспользоваться командой:

RESTORE FROM <имя файла> | *FROM MEMO* <имя поля> [*ADDITIVE*]

В результате работы команды оперативная память очищается от имеющихся переменных, и в нее помещаются переменные из указанного файла с расширением .mem или из Мето-поля. Для того чтобы сохранить в оперативной памяти переменные, значения которых определены до использования команды *SAVE TO*, и добавить в оперативную память переменные с отличными (другими) именами из файла (или Мето-поля), надо указать опцию *ADDITIVE*.

Переменные, которые хранятся в оперативной памяти, можно просмотреть с помощью команды:

DISPLAY MEMORY [*LIKE* <шаблон>] [*TO PRINTER* | *FILE* <имя файла>]

На экране каждая переменная занимает одну строку, куда выводится имя, тип, значение и статус переменной. После заполнения экрана надо нажать либо клавишу *ENTER*, либо клавишу «пробел» для продолжения вывода переменных на экран. При желании можно направить вывод переменных либо на принтер (*TO PRINTER*), либо в файл (*TO FILE*).

Все переменные или часть переменных, хранящихся в оперативной памяти, можно удалить с помощью команды:

```
RELEASE <список имен переменных> RELEASE ALL [LIKE | EXCEPT  
<шаблон>]
```

Для удаления всех переменных также можно использовать команду:
CLEAR MEMORY

3.3.4 Команды для работы с массивами

В FoxPro для работы с массивами предусмотрены четыре команды: две для работы с одномерными массивами и две для работы с двумерными массивами.

Команда *SCATTER* переносит данные из текущей записи активной таблицы в одномерный массив. Если одномерный массив не был создан заранее или размерность созданного массива мала, то автоматически либо создается одномерный массив, либо увеличивается размерность ранее созданного массива. Эта команда переносит данные из текущей записи активной таблицы либо в одномерный массив, либо в переменные, которые при необходимости могут быть также автоматически созданы. При автоматическом создании массива ему присваивается имя табличного файла, но перед именем добавляется буква М и префикс «.» точка. При автоматическом создании переменных им присваиваются имена соответствующих полей с добавлением перед именем буквы М и префикса «.» точка. Тип переменных определяется типом соответствующих полей. Формат команды:

SCATTER

```
[FIELDS <список имен полей> | FIELDS LIKE <шаблон> | FIELDS EXCEPT  
<шаблон> ] [MEMO] TO <имя массива> | TO <имя_массива> [BLANK] |  
[MEMVAR] | [MEMVAR] [BLANK]
```

Назначение опций:

FIELDS <список имен полей> — задает список имен полей, значения которых будут копироваться в одномерный массив или переменные.

FIELDS LIKE <шаблон> — в одномерный массив или переменные будут копироваться значения только тех полей, имена которых соответствуют шаблону.

FIELDS EXCEPT <шаблон> — в одномерный массив или переменные будут копироваться значения только тех полей, имена которых не соответствуют шаблону.

MEMO — указывает на необходимость копирования Мемо-полей. По умолчанию Мемо-поля не копируются.

TO <имя массива> — задает имя массива, куда будут копироваться значения указанных полей.

TO <имя массива> *BLANK* — создает пустой массив с указанным именем. Элементы массива получают тип и размер соответствующих полей.

MEMVAR — указывает на то, что поля текущей записи будут копироваться в переменные.

MEMVAR BLANK — создает набор пустых переменных. Переменные получают тип и размер соответствующих полей.

Если опция *FIELDS* не задана, то копирование начинается с первого поля записи подряд. Если число элементов массива больше, чем количество полей записи, то последние элементы массива сохраняют свое прежнее значение.

Команда *GATHER* переносит данные из одномерного массива в текущую запись активной таблицы. Для правильной работы команды необходимо заранее описать одномерный массив и заполнить его данными. При этом программист сам должен следить за соответствием типов и размеров элементов массива типам и размерам соответствующих полей записи таблицы. Формат команды:

```
GATHER FROM <имя массива> | MEMVAR  
[FIELDS <список имен полей> | FIELDS LIKE <шаблон> |  
FIELDS EXCEPT <шаблон>]  
[MEMO]
```

Назначение опций:

FIELDS <список имен полей> — задает имена полей, в которые, в строго указанном порядке, будут заноситься значения из элементов одномерного массива.

FIELDS LIKE <шаблон> — задает имена полей по шаблону (аналогично команде *SCATTER*).

FIELDS EXCEPT <шаблон> — задает имена полей по шаблону (аналогично команде *SCATTER*).

MEMO — задает копирование полей типа Мемо.

Если опция *FIELDS* не задана, то копирование начинается в первое поле записи и далее подряд. Если число элементов массива больше, чем количество полей записи, то последние элементы массива не копируются. Если число элементов массива меньше количества полей записи, то последние поля записи сохраняют свои прежние значения.

Команда *COPY TO ARRAY* копирует записи табличного файла в заранее созданный двумерный массив. Копируется столько записей, сколько строк в двумерном массиве, начиная с текущей записи. Если количество полей записи больше, чем число элементов массива в одной строке, то лишние поля записи не копируются. Если количество элементов массива в одной строке больше, чем количество полей записи, то последние элементы массива сохраняют прежние значения. Формат команды:

```
COPY TO ARRAY <имя массива> [FIELDS <список имен полей>] [SCOPE]  
[FOR <выр.L1>] [WHILE <Выр.L2>] [NOOPTIMIZE]
```

Назначение опций:

FIELDS <список имен полей> — задает имена полей, значения которых будут копироваться в двумерный массив

SCOPE — задает запись, начиная с которой производится копирование, либо

диапазон записей.

FOR <выр.L1> — выделяет для копирования только те записи, которые удовлетворяют логическому условию <выр.L1>.

WHILE <Выр.L2> — выделяет для записи указанные поля.

NOOPTIMIZE — запрещает ускоренную технологию обработки (Rushmore).

Команда *APPEND FROM ARRAY* добавляет в конец табличного файла записи из двумерного массива. Причем каждая строка двумерного массива заносится в табличный файл как новая запись. Если количество элементов в строке двумерного массива больше, чем количество полей в записи, то последние элементы массива не копируются. Если количество элементов в строке двумерного массива меньше, чем количество полей в записи, то последние поля записи сохраняют прежние значения. Формат команды:

APPEND FROM ARRAY < имя двумерного массива >

[*FOR* <выр.L1>]

[*FIELDS* <список имен полей>]

Назначение опций:

[*FOR* <выр.L1>] — выделяет из массива те строки, которые удовлетворяют логическому условию <выр.L1>.

FIELDS <список имен полей> — задает имена полей, в которые будут заноситься данные из массива.

3.3.5 Команды ввода-вывода

При работе с базами данных постоянно требуется вводить какие-либо данные, задавать критерии поиска и фильтрации, получать из базы данных результаты произведенных действий. Для этих целей предусмотрены команды ввода-вывода, которые бывают как простыми, так и универсальными.

В FoxPro имеются команды для вывода на экран значений переменных и элементов массивов. Причем эти команды допускают управление шрифтом и простейшее форматирование.

Команда ? вычисляет и выводит на экран <выр.1>.

Формат команды вывода:

?|?? [<выр.1>]

[*PICTURE* <выр. C1>]

[*FUNCTION* <выр. C2>]

[*AT* <выр. N1>]

[*FONT* <выр. C3>[, <Выр.N2>]] [*STYLE* <выр. C4>]

[, <выр.2>] ...

Назначение опций:

? <выр.1> — вычисляет и выводит значение <выр.1> в следующей строке экрана.

?? <выр.1> — вычисляет и выводит на экран <выр.1> в текущей строке экрана.

Если перед этой командой указана команда *SET PRINTER ON*, то вывод будет производиться на принтер (печать).

PICTURE <выр. C1> — задает шаблон для вывода значения <выр.1> (подробнее см.команду @ ... SAY ... GET ...).

FUNCTION <выр. C2> — задает коды управления выводом (подробнее см.команду @ ... SAY ... GET ...).

AT <выр. N1> — используется для создания таблиц.

<Выр.N1> задает номер колонки на экране, с которой начинается вывод значения <выр.1>.

FONT <выр. C3> [, <Выр.N2>] [*STYLE* <выр.C4>] - опция задает тип шрифта (<выр.C3>), размер шрифта (<выр.N>) и стиль написания шрифта (*STYLE* <выр.C4>).

В Visual FoxPro предусмотрены следующие стили написания шрифта:

В — жирный I — курсив N — нормальный
O — контурный Q — непрозрачный S — с тенью
- — перечеркнутый T — прозрачный U — подчеркнутый

Для очистки экрана используются две команды:

1) Очистка всего экрана — *CLEAR*. Команда очищает либо весь экран, либо все рабочее окно и размещает курсор в левом верхнем углу.

2) Очистка прямоугольной области экрана:
@ <Y1,X1> [*CLEAR* | *CLEAR TO* <Y2,X2>]

Команда очищает прямоугольную область экрана с координатами левого верхнего угла (<Y1,X1>) и координатами правого нижнего угла (<Y2,X2>). Если координаты правого нижнего угла области не указаны, то принимаются координаты правого нижнего угла экрана.

Универсальная команда ввода-вывода предназначена для форматного ввода-вывода на экран и принтер.

Краткий формат команды:

@ <Y1>, <X1> [*SAY* <выр.1>] ... [*GET* <переменная>]

Команда размещает курсор на экране или в окне в позиции Y1 (номер строки от 0 до 35) X1 (номер столбца от 0 до 79) и, начиная с этой позиции, на экран выводится сообщение, указанное в опции SAY <выр.1>. Если в команде дополнительно используется опция GET <переменная>, то <переменная> выводится на экран сразу после SAY <выр.1>. Если перед командой @ ... SAY ... GET ... указана команда SET DEVICE TO PRINT, то вывод производится на принтер и тогда количество строк и столбцов определяется размером бумаги. Для переназначения вывода на экран следует использовать команду SET DEVICE TO SCREEN (команда используется по умолчанию). Значение <переменная> должно быть определено до использования команды @ ... SAY ... GET ... В общем случае, назначение опции SAY указать пользователю, где выполнить ввод (вывод) по пункту GET.

Команда *READ* может редактировать переменные всех активных команд @ ... SAY ... GET и разрешает ввод данных в спроектированный экран.

Формат команды:

READ

[CYCLE]

[ACTIVATE <выр.L1>]

[DEACTIVATE <выр.L2>]

[MODAL]

[WITH <список имен окон>]

[SHOW <выр.L3>]

[VALID <выр.L4> | <выр.N1>]

[WHEN <выр.L5>]

[OBJECT <выр.N2>]

[TIMEOUT <выр.N3>]

[SAVE]

[NOMOUSE]

[LOCK | NOLOCK]

[COLOR <список пар цветов> | COLOR SCHEME <Выр.N4>]

Назначение опций:

CYCLE — при переходе от одной команды @ ... SAY ... GET к другой команде, предотвращает окончание команды READ при обработке последней команды @ ... SAY ... GET и возвращает курсор на первую команду @ ... SAY ... GET. Для выхода из циклического режима надо либо нажать клавишу Esc, либо подать команду Ctrl + W, либо другие прерывающие команды.

ACTIVATE <выр.L1> — опция работает с командами @ ... SAY ... GET в разных окнах. При перемещении пользователя от одного окна к другому окну команда READ активизируется для текущего окна только в случае истинности логического выражения <выр.X1>.

DEACTIVATE <Выр.L2> — при работе пользователя с несколькими окнами опция разрешает покинуть очередное окно, если логическое выражение <выр.L2> истинно.

MODAL — разрешает активацию только тех окон, имена которых указаны в <списке имен окон> опции WITH.

WITH <список имен окон> — задает имена окон, с которыми работает команда READ.

SHOW <выр.L3> — разрешает обновить информацию на экране, предварительно измененную командой READ. Эта опция работает совместно с командой SHOW GETS, которая фактически и обновляет содержимое экрана.

VALID <выр.L4> | <Выр.N1> — обеспечивает достоверность вводимых данных с помощью команды READ. Команда READ прекратит всю работу только после ввода достоверных данных, то есть если логическое выражение <выр.Ы> истинно.

WHEN <Выр.L5> — разрешает активацию команды READ только в случае истинности логического выражения <Выр.L5>.

OBJECT <Выр.N2> — указывает номер объекта (команды @ ... SAY ... GET), с которого начинается обход. Нумерация объектов начинается с единицы.

TIMEOUT <Выр.N3> — задает время активности команды READ в секундах для ожидания ввода пользователя

3.4 Условная и циклическая обработка данных

3.4.1 Команды циклов

Так же как и в языках программирования, при разработке приложений баз данных предусмотрены команды циклов. Назначение команд, а иногда и формат команд, аналогичные другим языкам программирования.

Формат команды цикла по условию:

DO WHILE <выр. L>

...

[*LOOP*]

[*EXIT*]

ENDDO

Цикл выполняется многократно, до тех пор пока истинно логическое условие <выр. L>. Для принудительного выхода из цикла, до нарушения логического условия <выр. L>, используют команду *EXIT*. Команда *LOOP* используется для прекращения вычислений, предусмотренных текущей итерацией, и принудительного перехода к следующей итерации внутри цикла.

Формат команды фиксированного цикла:

FOR <пер> = <выр. N1> *TO* <выр. N2> [*STEP* <выр. N3>]

...

[*EXIT*]

[*LOOP*]

ENDFOR | *NEXT*

Цикл выполняет фиксированное количество итераций (шагов).

<пер.> называется переменной цикла. В переменной цикла фиксируется количество выполненных итераций, то есть после выполнения очередной итерации значение переменной цикла увеличивается на значение шага цикла <выр. N3>.

<выр. N1> задает начальное (стартовое) значение переменной цикла.

<выр. N2> задает конечное значение цикла.

<выр. N3> задает шаг цикла.

<выр. N1>, <выр. N2> и <выр. N3> имеют целочисленный тип.

По умолчанию шаг цикла принят равным 1. Если <выр. N2> больше <пер.>, то цикл будет выполняться до тех пор, пока <пер.> не будет больше <выр. N2>. Это событие обязательно наступит, так как после выполнения каждой итерации, переменная цикла <пер.> будет увеличиваться на 1. В некоторых случаях требуется, чтобы переменная цикла после выполнения каждой итерации увеличивалась быстрее, тогда в <выр. N3> указывают целое число, отличное от 1, которое и будет прибавляться к <пер.> после выполнения очередной итерации. Мы рассмотрели цикл на увеличение (сложение), то есть переменная цикла с каждой новой итерацией увеличивала свое значение на величину шага цикла <выр. N3> и цикл прекращался, когда переменная цикла <пер.> становилась больше <выр. N2>.

Допускается задание цикла на уменьшение (вычитание). В этом случае начальное значение *<выр. N1>* переменной цикла *<пер.>* должно быть больше конечного значения *<выр. N2>*, а шаг цикла *<выр. N3>* должен быть целым и отрицательным. В этом случае после каждой выполненной итерации из переменной цикла *<пер.>* будет вычитаться значение шага цикла *<выр. N3>* и цикл закончит свою работу после того, как переменная цикла *<пер.>* станет меньше конечного значения цикла *<выр. N2>*.

Цикл сканирования организует просмотр записей текущей таблицы.

Формат команды:

SCAN

```
...  
[FOR <L1>]  
  [ WHILE <L2> ]  
  [ NOOPTIMIZE ]  
  [LOOP]  
  [EXIT]
```

ENDSCAN

Цикл сканирования работает аналогично циклу *DO WHILE*, но предназначен для работы с текущей таблицей. Назначение опций аналогичное. По умолчанию цикл сканирования выполняется для всех записей.

3.4.2 Команды выполнения и ветвления алгоритмов

Формат команды выполнения:

RUN <команда> | ! <команда>

Эта команда позволяет выполнить команду DOS или любую внешнюю команду, которая используется с приглашением DOS и подается в окне Command.

При реализации сложных алгоритмов обработки данных, выбора одного решения из многих возможных, используют различные команды ветвления алгоритма.

Формат команды ветвления алгоритма на два направления:

IF <выр. L> <оператор1> [ELSE <оператор2>] ENDIF

Если логическое выражение *<выр. L>* истинно, то выполняется *<оператор 1>*.

Если логическое выражение *<выр. L>* ложно, то выполняется *<оператор2>*.

Формат команды ветвления алгоритма на много направлений:

DO CASE

CASE <условие 1>

 ...

< оператор >

CASE <условие 2>

 ...

< оператор >

[*OTHERWISE*

...
< оператор >

...]

ENDCASE

Количество блоков CASE <условие N> не ограничено. При работе команды DO CASE последовательно проверяются условия CASE <условие N> и выполняются все блоки, условия которых истинны. Поэтому для однозначной работы алгоритма необходимо, чтобы условия CASE <условие N> взаимно не перекрывались. Если все указанные условия ложны, то выполняется блок OTHERWISE.

Варианты задания условий CASE <условие N>:

1) Точные условия.

Переменная условия точно равна какому-либо значению или точно не равна какому-либо значению.

CASE a = b или CASE a > b или CASE a < b

2) Диапазон значений.

Значение переменной условия задается диапазоном, включая границы диапазона. CASE BETWEEN(ms, 01, 09)

3) Список значений.

Значение переменной условия задается списком. Элементы списка отделяются друг от друга запятой. CASE INLIST(ms, '10', '19', '43', '44', '25')

Для условия CASE <условие N> могут использоваться значения любого типа, но надо следить за правильностью использования типов данных в функциях.

Формат команды безусловного перехода

GO | GOTO

[*RECORD*] | *TOP* | *BOTTOM*

[IN <выр. N2> | <псевдоним>]

Команда переводит курсор внутри таблицы.

Назначение опций:

[*RECORD*] <выр. N1> — переводит курсор на указанную запись.

TOP — переводит курсор на первую запись.

BOTTOM — переводит курсор на последнюю запись.

IN <выр. N2> | <псевдоним> — задает номер рабочей области или псевдоним рабочей области, в которой находится нужная таблица. По умолчанию имеется в виду активная таблица.

3.5 Объектно-ориентированное построение Fox Pro

Visual FoxPro – это платформа, поддерживающая гибридный язык программирования. Т.е. работая в FoxPro можно создавать программы, используя как модульное или процедурное программирование, так и объектно-ориентированное программирование.

Разработка объектно-ориентированных приложений начинается с размещения объектов на форме, каждый из которых выполняет конкретные функции. Создание каждого объекта предполагает использование Visual FoxPro, при этом допускается копирование объектов из одного приложения в другое. При работе с объектами под событием понимается щелчок кнопки мыши на объекте, выбор пункта меню и т. д.

Объектно-ориентированное построение Visual FoxPro предполагает построение приложения шаг за шагом, переходя от одного объекта к другому. Все объекты группируются в классы. Класс содержит информацию о том, как должен выглядеть объект и определяет выполняемые им действия. Объект является экземпляром класса и наследует характеристики класса. Примеры объектов: кнопка, радиогруппа, флажок, окно и т. д. При создании входных форм широко используются базовые классы. Классы могут быть вложены друг в друга, в этом случае старший класс называется контейнером. Доступ ко всем объектам возможен либо на этапе проектирования формы, либо программно.

Для программного доступа к объектам используются следующие ключевые слова:

THIS — ссылка на сам объект;

THISFORM — ссылка на форму, которая содержит объект;

THISFORMSET — ссылка на группу форм, которая содержит объект.

В Visual FoxPro отобразить данные можно двумя способами: в окне BROWSE и в экранной форме. В Visual FoxPro экранная форма существует как функциональный объект проектирования. На экранной форме объединяются визуальные компоненты, элементы управления и сервисные (вспомогательные) элементы.

4 Организация интерфейса в Visual FoxPro

Созданный программистом интерфейс представляет само приложение. Работая со сложными системами, когда обрабатывается большое количество взаимосвязанных баз данных, таблиц, представлений, необходимо иметь доступ к любым данным, не загромождая экран лишней информацией, контролировать обработку данных во избежание ошибок и иметь средства управления процессом этой обработки. При создании интерфейса используют такие объекты, как окна, страницы документов, различные элементы управления, графические объекты, OLE-объекты, экранные формы для наглядного представления и редактирования табличных данных.

4.1 Создание меню в Visual Fox Pro

В соответствии со стандартами Windows в любом приложении рекомендуется иметь строку меню, которая в Visual FoxPro содержит команды, предназначенные для вызова форм, формирования отчетов, запросов и т. д.

При разработке приложения вы можете создать все требуемые объекты (базу данных, входящие в нее таблицы, формы, отчеты, запросы). Затем объединить отдельные объекты с помощью меню. Можно поступить иначе. Сначала разработать и создать меню, а затем по мере создания форм и отчетов включать их запуск в меню. Второй способ более нагляден. Вы в любой момент можете запустить меню и продемонстрировать заказчику, как создаваемая система выглядит, как осуществляется вызов тех или иных программ, запустить уже созданные формы, напечатать подготовленные отчеты.

4.1.1 Подготовка к созданию меню

На начальном этапе разработки необходимо определить требования, предъявляемые к создаваемому приложению, состав информации, которая будет содержаться в проектируемой базе данных.

После этого определяется структура таблиц и совпадающие поля для их связывания. Затем создаются сами таблицы, входящие в базу данных, определяются отношения между ними.

Одновременно с составом информации вы должны определить те средства, которые получит в свое распоряжение пользователь при работе с вашим приложением.

Приложение должно содержать эффективную справочную систему, содержащую информацию о приложении, описание его основных функций и инструкцию по работе. В среде Windows предпочтительнее всего создавать справочную систему в принятом в Windows стандарте, чтобы облегчить пользователю поиск информации в знакомой ему среде.

После того как определена структура данных, спроектированы таблицы, входящие в базу данных, вы можете приступить к разработке структуры меню. Прежде чем описывать структуру меню в конструкторе, нарисуйте эскиз меню на бумаге, посоветуйтесь с пользователями приложения.

В базах данных меню является основным инструментом диалога. В FoxPro

предусмотрены возможности для создания меню различных типов. Управление работой меню осуществляется либо мышью, либо с клавиатуры.

Цель создания меню - обеспечить пользователю простой доступ ко всем компонентам приложения. С помощью меню организуется одновременная работа с такими заранее созданными объектами, как базы данных, таблицы, представления, запросы, формы, отчеты и т.д.

4.1.2 Виды меню

Световое меню — это набор пунктов (элементов) меню, один из которых является активным, то есть выделен цветом. Выбор пункта меню осуществляется либо щелчком мыши на пункте меню, либо с помощью клавиш-стрелок курсор устанавливается на нужный пункт меню и нажимается клавиша Enter. Отказ от выбора пункта меню — нажатие клавиши Esc.

Вызванное световое меню накладывается, как правило, поверх прежнего изображения на экране. После удаления меню с экрана происходит автоматическое восстановление первоначального вида экрана. При работе с меню допускается использование клавишных команд, дублирующих выбор пункта меню. Такие клавишные команды называются «горячими» клавишами. Если меню многоуровневое (три и более уровней), то использование «горячих» клавиш становится эффективным средством работы с меню.

При работе прикладных программ, содержащих меню, на экран могут выводиться системные сообщения. Текст системных сообщений накладывается на элементы меню и впоследствии не удаляется. Для запрета вывода на экран системных сообщений используется команда *SET TALK OFF*.

По умолчанию системные сообщения выводятся на экран, то есть установлена команда *SET TALK ON*.

При работе со световым меню можно задать режим, когда при нажатии клавиши с первой буквой имени пункта меню этот пункт (команда) меню становится активным. Для задания этого режима используется команда *SET CONFIRM OFF*.

Если установлена команда *SET CONFIRM ON*, то после нажатия клавиши буквы дополнительно надо нажать либо клавишу Enter, либо клавишу SpaceBar.

При работе конкретного приложения в разные моменты времени некоторые пункты меню могут быть не доступны для выбора и на экране выделены в полтона (бледно). Для того чтобы сделать пункт меню недоступным для выбора, надо перед именем пункта установить символ «\». В вертикальных (POPUP) меню иногда необходимо одну группу пунктов отделить разделительной чертой от других пунктов меню, причем курсор на разделительной черте не фиксируется. Для этих целей вместо имени пункта меню надо указать сочетание двух символов «\».

Клавишное меню представляет собой набор одно- или двухклавишных команд, и не имеет в своем составе каких-либо элементов управления расположенных на экране. Иногда в последней строке экрана размещают подсказку о размещении команд клавишного меню. Клавишное меню предназначено для реализации простых и часто повторяющихся операций по обработке данных.

Команды клавишного меню устанавливаются связь между программой и прерываниями от нажатия клавиш или для обработки возникших во время работы программы ошибок.

4.1.3 Технологии построения меню

В FoxPro предусмотрены две технологии построения меню: FOX-меню и dBASE-меню.

Меню типа FOX активны только во время работы программы и являются частью программы. При выборе пункта такие меню вырабатывают числовой код, который запоминается и анализируется с последующей выработкой реакции на выбор пункта меню. Как правило, анализ выбора пользователя осуществляется с помощью оператора DO CASE... ENDCASE. Из меню этого типа можно сделать единственный выбор. Чтобы организовать многократный выбор из меню, надо описание меню поместить внутрь цикла.

Меню типа dBASE после своего описания остается в оперативной памяти резидентно и может многократно вызываться на экран и удаляться с экрана. Меню может вызываться не только из конкретного приложения, но и из окна Command даже в том случае, если закрыта база данных и программный файл закончил свою работу. Для удаления меню из оперативной памяти предусмотрены специальные команды. При своей работе меню вырабатывает не только числовые коды, которые фиксируют выбор пользователя, но и непосредственно вызывает процедуры и команды на выполнение. В качестве пунктов меню здесь можно использовать имена файлов и структуру базы данных, а также организовать множественный выбор пунктов меню.

При создании и эксплуатации данного меню должны быть предусмотрены следующие элементы:

- определение меню – описывается содержание меню, клавиши быстрого доступа, формы и реакции меню. Меню типа dBase определяется один раз;
- активация меню – команды (клавиши быстрого доступа) активации выводят меню на экран и делают его чувствительным к выбору пользователя;
- деактивация меню – меню удаляется с экрана, но остается в оперативной памяти, и в последствии может быть предъявлено на экран;
- удаление меню – меню удаляется из оперативной памяти. Для нового использования меню его надо заново определять.

4.1.4 Структура меню

Меню любой сложности строится из меню двух типов: горизонтального и вертикального (всплывающего).

Горизонтальное меню, примером которого является основное меню Visual FoxPro, состоит из нескольких горизонтально расположенных пунктов,

которые называются PAD-пунктами (например, пункты File, Edit, View...).

Всплывающее меню — POPUP — состоит из нескольких вертикально расположенных пунктов, которые называются BAR и появляются только при активизации соответствующего PAD-пункта (например, пункты New, Open, Save... пункта File основного меню). POPUP-меню может использоваться как в составе меню более высокого уровня, как и самостоятельно. Главное или POPUP-меню определяется и активизируется следующим образом:

```
DEFINE MENU / POPUP <имя меню>  
<определение составляющих меню (PAD, POPUP, BAR)>  
<описание реакции составляющих меню на выбор>  
ACTIVATE MENU <имя меню>
```

Строка главного меню может дополнять или замещать основное меню Visual FoxPro, или отображаться в определенном пользователем окне. Синтаксис команды создания строки меню следующий:

```
DEFINE MENU <имя меню>  
[BAR[AT LINE <номер строки, в которой появится меню>]]  
[IN WINDOW <имя окна, определенного заранее>]  
[FONT <имя шрифта>]  
[STYLE <имя стиля>]  
[KEY <имя клавиши, используемой для вызова меню>]  
[MESSAGE <сообщение, которое выводится внизу экрана>]  
[COLOR SCHEME <список цветовых пар>]  
[SCROLL]
```

Меню с опцией BAR имеет свойства системного меню.

Если меню не умещается по ширине экрана/окна, то используйте опцию [SCROLL].

4.1.5 Определение составляющих меню

В зависимости от сложности меню проектируется на нескольких уровнях (Menu level). На верхнем уровне определяются независимые меню, к которым относится MENU и всплывающее меню POPUP.

Пункты PAD, составляющие MENU, а также пункты BAR, составляющие всплывающее меню POPUP, зависят от соответствующего независимого меню. Поэтому для определения в команде используется конструкция OF <имя независимого меню, включающего этот пункт>.

PAD-пункты меню задаются следующей командой.

```
DEFINE PAD <имя PAD-пункта> OF <имя BAR-меню, включающего этот PAD>  
PROMPT <название PAD-пункта>  
[AT <Y,X>]  
[BEFORE <имя пункта> / AFTER <имя пункта>]
```

*определяет положение данного пункта относительно уже включенных

```
[MESSAGE <текст сообщения>]
[MARK <символ, расположенный слева от пункта меню>]
[KEY <>]
[SKIP[FOR <връжL, определяющее условие блокировки данного пункта меню>]]
[COLOR SCHEME <>]
```

Следует различать имя пункта меню и его название (заголовок). Имя задает программный объект и записываться оно должно только латинскими буквами, в то время как заголовок может быть любым.

Например, в следующей команде имя пункта меню - `radname`, а его заголовок задается символьной строкой '`< Информация1`'. При этом пункт может быть активизирован нажатием клавиши "И", поскольку она назначена клавишей быстрого доступа.

```
DEFINE PAD radname OF mainmenu PROMPT '< Информация ' AT 1,8
```

Определение всплывающего меню POPUP

```
DEFINE POPUP <имя всплывающего меню>
[FROM <Y1,X1> TO <Y2,X2>]
[PROMPT <вырC> / PROMPT FIELD <выр>
/ PROMPT FILES [LIKE <шаблон>] / PROMPT STRUCTURE]
[IN WINDOW <>] [KEY <>] [FOOTER <>][TITLE <>]
```

*заголовки соответственно в нижней и верхней строках всплывающего меню

```
[MARK <>]
[MESSAGE <>]
[MULTISELECT]
[SHADOW] *для выделения более темным цветом
[SCROLL]
[COLOR SCHEME <>]
```

Опция `PROMPT` задает заголовки пунктов меню.

При использовании `PROMPT FIELD <Выр>` элементами меню станут значения поля крытой таблицы, если `<выр>` - имя поля. В общем случае `<выр>` содержит несколько полей, в том числе и из других открытых таблиц.

Например, пункты всплывающего меню `list` можно назвать именами студентов из поля `name` таблицы `stud.dbf`.

```
USE stud
DEFINE POPUP list FROM 1,7 TO 15,30;
PROMPT FIELD name
ACTIVATE POPUP list
```

При использовании `PROMPT FILES [LIKE <шаблон>]` пункты будут называться менами файлов, отображенных в соответствии с шаблоном, а использование опции `PROMPT STRUCTURE` обеспечит выбор имен полей открытой таблицы.

Для взаимного перемещения пунктов всплывающего меню в команде определения можно использовать опцию `MOVER`. Для создания, например, всплывающего меню из четырех пунктов с возможностью их перемещения используйте следующие ниже строки.

```
CLEAR
```

```

DEFINE POPUP popDemo MOVER FROM 2,2
DEFINE BAR 1 OF popDemo PROMPT 'Первый'
DEFINE BAR 2 OF popDemo PROMPT 'Второй'
DEFINE BAR 3 OF popDemo PROMPT 'Третий'
DEFINE BAR 4 OF popDemo PROMPT 'Четвертый'
ACTIVATE POPUP popDemo

```

Определение пунктов BAR всплывающего меню

```

DEFINE BAR <номер пункта> OF <имя POPUP-меню> PROMPT <вырС>
[BEFORE <имя пункта>/AFTER <имя пункта>]
[KEY <>]
[MARK o]
[MESSAGE <>]
[SKIP[FOR <вырL>]]
[COLOR SCHEME <>]

```

Например, при выборе всплывающего меню с названием "Студенты" открывается ниспадающий список студентов 23-й группы и дополнительно в статусной строке выводится сообщение "Студенты 23-й группы". Кроме того, блокируются строки с именами студентов из Китая.

```

USE stud
DEFINE POPUP popname FROM 5,8 TO 18,30;
TITLE "Студенты" PROMPT FIELD
If (group=23, name, SPACE(12));
MESSAGE "Студенты 23-й группы";
SKIP FOR country='Китай'
ACTIVATE' POPUP popname

```

4.1.6 Управление доступом к пунктам меню

Кроме управления доступом к некоторым пунктам меню с помощью опции SKIP рассмотренных команд определения, есть специальные команды разрешения или запрещения доступа, в зависимости от значения <вржL>, соответственно .F. / .T..

```

SETSKIP OF MENU <имя менк> <вржL>
SETSKIP OF PAD <имя PAЭ-меню> OF <имя BAR-Меню)> <вржL>
SETSKIP OF POPUP <имя POPUP-меню> <вржL>
SETSKIP OF BAR <номер BAR-меню> OF <имя POPUP-меню> <вржL>

```

Эти команды используются следующим образом.

```

IF <условие блокировки>
SET SKIP OF PAD PadName OF BarName .T.
* пункт недоступен
ELSE
SET SKIP OF PAD PadName OF BarName .F.
* пункт доступен

```

ENDIF

Запуск на выполнение выделенного пункта меню осуществляется нажатием клавиш Enter или Space, или щелчком на нем основной кнопкой мыши. Команды, определяющие реакцию на выбор элементов меню, следующие:

ON SELECTION MENU <имя> [<команда>]

ON SELECTION BAR <номер> OF <имя POPUP-меню> [<команда>]

ON SELECTION PAD <имя> OF <имя BAR-Меню> [<команда>]

ON SELECTION POPUP <имя>/ALL [<команда>]

Если параметр <команда> отсутствует, назначение отменяется.

Обычно в качестве параметра <команда> используется вызов процедуры. Например, для просмотра отчета при выборе всплывающего меню *rep* выполните следующие команды:

ON SELECTION POPUP rep DO repproc

...

PROCEDURE repproc

USE, scholar

REPORT FORM scholar.frx PREVIEW

RETURN

Но иногда при выборе пункта меню выполняется только одна команда. В частности, для выхода из программы при выборе пункта с именем *exit* меню *main* необходимо выполнить команду

ON SELECTION PAD exit OF main CANCEL

Для ветвления меню используют такие команды:

ON BAR <номер> OF <имя POPUP-меню> [ACTIVATE POPUP <имя POPUP-меню>]

ON BAR <номер> OF <имя POPUP-меню> [ACTIVATE MENU <имя BAR-меню>]

Теперь при выборе BAR-пункта меню будет отображаться заранее определенный пункт меню следующего уровня. Так же можно организовать и ветвление пункта PAD.

ON PAD <имя> OF <имя BAR-Меню> [ACTIVATE POPUP <имя POPUP-меню>]

или

ON PAD <имя> OF <имя BAR-меню> [ACTIVATE MENU <имя BAR-меню>]

Выполнение этих команд без опции [ACTIVATE...] ликвидирует ветвление.

4.1.7 Активизация составляющих меню и удаление меню

В любом месте программы заранее определенное меню может быть активизировано команды

ACTIVATE MENU <имя>

или

```
ACTIVATE POPUP <имя> [AT <Y,X>][BAR <номер>][REST][NOWAIT]
```

Опция *BAR* <> позволяет выбрать *BAR* с указанным номером по умолчанию. Опция *REST* используется для меню, создаваемых с применением варианта *PROMPT FIELD* команды *DEFINE POPUP*, чтобы выделить текущее поле таблицы. По умолчанию при выборе выделяется первый элемент всплывающего меню.

Ниже приведен пример создания всплывающего меню *base*. (Пункт "ПЕЧАТЬ" вызывает процедуру *print*.)

```
DEFINE POPUP base FROM 1,1 TO 5,35
DEFINE BAR 1 OF base PROMPT '\<ДОПОЛНЕНИЕ'
DEFINE BAR 2 OF base PROMPT '\<КОРРЕКЦИЯ'
DEFINE BAR 3 OF base PROMPT '\<УДАЛЕНИЕ';
MESSAGE 'Будьте осторожны!' COLOR, w+/r,,w+*/r
DEFINE BAR 4 OF base PROMPT '\<ПЕЧАТЬ'
DEFINE BAR 5 OF base PROMPT '\<ВЫХОД'
ON SELECTION BAR 1 OF base DO procl
ON SELECTION BAR 2 OF base DO proc2
ON SELECTION BAR 3 OF base DELETE
ON SELECTION BAR 4 OF base DO print
ON SELECTION BAR 5 OF base CANCEL
ACTIVATE POPUP base
```

```
.....
PROCEDURE print
SET PRINTER ON
? PRINTSTATUS()
IF NOT PRINTSTATUS()
WAIT 'Подготовьте принтер!'
ELSE
LIST TO PRINTER
ENDIF
SET PRINTER OFF
CLEAR
RETURN TO MENU
```

Команда *DEACTIVATE MENU/POPUP* удаляет указанные меню с экрана.

Команда *HIDE/SHOW MENU/POPUP* делает невидимыми/видимыми активные меню.

Удалить из памяти элементы меню можно командой *CLEAR MENUS/POPUP*. То же выполняет и команда *RELEASE*.

4.2 Использование окон в Visual FoxPro

4.2.1 Работа с окнами

В Fox Pro имеется возможность одновременной работы с несколькими окнами. С помощью специальных команд окна можно создавать, изменять их положение и размеры, закрывать как программными средствами, так и с помощью клавиатуры и мыши.

Различают три состояния окна:

1) нормальное - окно имеет размеры и занимает положение на экране, определенное командами при его создании;

2) свернутое - окно имеет минимальный размер (одна строка с заголовком окна) и размещается в нижней части экрана;

3) распахнутое - окно раскрыто на весь экран.

Для работы с окнами предусмотрены следующие клавишные команды:

Ctrl+F7 и клавиши стрелки - перемещение окна по экрану

Ctrl+F9 - сворачивает окно до минимального размера.

Shift+ctrl+F9 - сворачивает окно до минимального размера и перемещает его в правый нижний угол экрана.

Ctrl+F10 - переход к другому окну

4.2.2 Описание окна

Для описания окна используется следующая команда:

DEFINE WINDOW <ИМЯ ОКНА> FROM Y1,X1 TO Y2,X2

[AT Y1,X1]

[SICE Y2,X2]

[IN[WINDOW]<имя окна2> IN SCREEN IN DESKTOP]

[FONT <ВЫР.С1>[ВЫР.Н>]][STYLE<ВЫР.С2>]

[FILL FILE<ВЫР.С3>]

[ICON FILE<ВЫР.С4>]

[HALEHEIGHT]

[MDI/NOMDI]

[FOOTER<ВЫР.С5>]-описывает заголовок окна в центре нижней границы окна.

[TITLE<ВЫР.С6>]-описывает заголовок окна

[DOUBLE/PANEL/NONE/SYSTEM<строка бордюра>]

[CLOSE/NOCLOSE]

[FLOAT/NOFLOAT]

[GROW/NOGROW]

[MINIMIZE]

[SHADOW]

[ZOOM/NOZOOM]

[FILL<ВЫР.С7>]

[COLOR<список пар цветов>/COLOR SCHEME<ВЫР.С2>]

Рассмотрим основные опции команды.

FROM и AT. Определяют месторасположение окна на экране или в окне-родителе.

IN [WINDOW]. Создает дочернее окно внутри другого окна, уже существующего. Новое окно нельзя переместить за пределы родительского окна. При изменении положения родительского окна перемещаются и дочерние.

IN SCREEN. Вывод текущей информации на экран по умолчанию.

NAME <имя объекта>. Определено только для Visual FoxPro. Создает ссылку на объект окна.

TITLE <>. Задаёт любой заголовок в верхней части окна. При необходимости в процессе выполнения программы заголовки окна можно переназначить командой

MODIFY WINDOW <имя окна> TITLE "Новый заголовок окна"

FOOTER o. Выводит указанное сообщение в нижней части экрана. Определено только для операционной системы DOS.

DOUBLE | PANEL | NONE | SYSTEM | <строка бордюра>. Определяют оформление границ окна.

CLOSE | NOCLOSE. Позволяет/запрещает закрывать окно с помощью мыши. (По умолчанию - NOCLOSE.)

FLOAT | NOFLOAT. Позволяет/запрещает перемещать окно. (По умолчанию - NOFLOAT.)

GROW | NOGROW. Позволяет/запрещает изменять размер окна. (По умолчанию -NOGROW.)

ZOOM | NOZOOM. Позволяет/запрещает раскрывать окно на весь экран и возвращать его прежний размер. (По умолчанию - NOZOOM.)

MINIMIZE | NOMINIMIZE. Позволяет/запрещает сворачивать окно до минимального размера. (По умолчанию — NOMINIMIZE.)

В объектном программировании перечисленные опции определяются свойством WindowState. При значении свойства 0 - окно отображается стандартным размером, 1 -минимизированным, 2 - максимальным.

ICON FILE - задает значок, который будет использоваться для отображения минимизированного окна (соответствует свойству Icon).

MDI | NOMDI - определяет возможность использования многодокументного интерфейса (Multi Document Interface). В объектном программировании эти опции определяются значением свойства WindowType. При значении 0 (Modeless) - окно немодальное. При значении 1 (Modal) - работать с другим окном можно, но только после того, как закрыто предыдущее окно, что ограничивает возможность ошибки пользователя.

Значение Modeless используется для организации многооконного интерфейса ввода и редактирования данных. Может быть создано несколько экземпляров одного и того же немодального окна.

SHADOW - создает объемное изображение окна (с помощью тени). Определено

только для операционной системы DOS.

FILL - указывает символ заполнения окна. Определено только для операционной системы DOS.

FILL FILE - заполняет задний план окна содержимым BMP-файла. Этот рисунок выводится в натуральную величину и столько раз, сколько хватает места в окне. Рисунок остается в окне на всем протяжении существования последнего и не стирается командой CLEAR. Рисунок можно использовать не только как декоративный элемент дизайна экрана, он может нести информационно-смысловую нагрузку.

COLOR SCHEME | *COLOR*. Устанавливает цвет окна непосредственно или с помощью цветовой схемы. По умолчанию окнам присваиваются цвета из *COLOR SCHEME 1*.

4.2.3 Активизация окна

Для активизации окна из программы, следует выполнить команду
ACTIVATE WINDOW [*<имя окна1>* [, *<имя окна2>* ...]] |

ACTIVATE WINDOW ALL

[*IN* [*WINDOW*] *<имя окна-родителя>* | *SCREEN*]

[*BOTTOM* | *TOP* | *SAME*]

[*NOSHOW*]

Если не используется опция *NOSHOW*, активное окно становится видимым.

Опции *BOTTOM* | *TOP* указывают на то, что активизированные окна размещаются под/перед уже имеющимися на экране, а *SAME* — активизация не изменит положение окна. По умолчанию устанавливается *TOP*, т.е. активизированное окно располагается на переднем плане.

Окна могут использоваться и без предварительной активизации командой *ACTIVATE WINDOW*» если выполняется команда, имеющая опцию *WINDOW*.

4.3 Элементы управления

При создании пользовательского интерфейса удобно использовать элементы управления. Элементы управления можно как программировать с помощью команд FoxPro, так и, используя лист формы, располагать на форме готовые элементы управления. Для описания элементов управления, как правило, используется команда *@ GET* с опцией *FUNCTION* или *PICTURE*. Элементы управления можно располагать как на экране, так и на форме. Если элемент управления надо расположить на форме, то программный код этого элемента управления помещают на лист программного кода формы (двойной щелчок на листе формы выводит на экран лист программного кода формы).

При описании элемента управления с помощью команд создается либо терминальный элемент (TERMINATING), который при своем описании требует команду считывания *READ*, и при работе задает выполнение какой-либо команды (например печать); либо нетерминальный элемент (NOTERMINATING), который предназначен для определения особенностей выполнения действий терминального элемента (например определение режимов печати). При составлении шаблонов для опций FUNCTION и PICTURE первый символ шаблона определяет функциональный код создаваемого элемента управления:

* — кнопки различных типов;

A — списки различных типов.

Второй символ шаблона указывает на конкретный вид элемента управления:

*R — радиокнопка (Radio Button).

*C — флажок (CheckBox).

*пробел — невидимая кнопка (Invisible Button).

Если пробел есть, то кнопка будет видима. Если пробела нет, то кнопка — невидима.

^пробел — раскрывающийся список (ComboBox).

Для создания списка (List Box) вместо опции FUNCTION используют опцию FROM.

При описании некоторых элементов можно использовать третий символ, который замещает опцию по умолчанию:

N — нетерминальная кнопка.

T — терминальная кнопка.

H — расположение элементов по горизонтали (HORIZONTAL).

V — расположение элементов по вертикали (VERTICAL).

После установки двух (или трех) символов шаблона обязательно устанавливается пробел. Шаблоны для опций FUNCTION и PICTURE пишутся одинаково, но для опции PICTURE шаблон всегда начинается с символа @.

Стандартные элементы управления, взятые с панели инструментов, могут располагаться только на форме

4.4 Создание экранной формы

4.4.1 Создание экранной формы с помощью Мастера форм — Form Wizard

С помощью Мастера форм можно создавать экранные формы для просмотра и редактирования данных как одной таблицы, так и для нескольких взаимосвязанных таблиц. Экранная форма создается только для физически

существующей таблицы.

После открытия проекта (базы данных) в диалоговой панели Диспетчера проектов (Project Manager) надо выбрать вкладку Документы (Documents) и курсором выделить команду Формы (Forms). Затем нажать кнопку Новый (New).

В появившейся на экране диалоговой панели New Form выбрать способ создания формы — пиктограмму Form Wizard. В диалоговой панели Wizard Selection надо указать на данных каких таблиц будет построена форма: Form Wizard — по данным одной таблицы; One-to-Many Form Wizard — по данным нескольких таблиц.

После запуска Мастера форм на экран будут выводиться диалоговые панели Мастера, где надо ответить на вопросы. Все диалоговые панели Мастера снабжены четырьмя кнопками управления: Cancel — отказ от построения экранной формы; Next — переход к следующему шагу Мастера; Back — возврат к предыдущему шагу Мастера, Finish — завершение построения формы. Кнопку Finish можно нажать на любом шаге Мастера, тогда недостающие сведения будут подставлены по умолчанию.

В первой диалоговой панели Form Wizard (Step 1 — Field Selection) в раскрывающемся списке Databases надо указать имя базы данных, а в раскрывающемся списке Tables указать имя таблицы, по данным которой будет построена экранная форма. В раскрывающемся списке Available Fields следует выбрать имена полей, значения которых будут размещены в экранной форме. Перенос выбранных полей в окно Selected Fields производится с помощью кнопок-стрелок.

Во второй диалоговой панели Form Wizard (Step 2 — Style) определяется стиль исполнения экранной формы. В списке Style выбирается один из допустимых стилей: Standard, Chiseled, Shadowed, Boxed или Embossed. Результат вашего выбора можно просмотреть здесь же в специальном окне. С помощью радио-группы Button Type определяется тип отображения кнопок управления:

- Text Buttons — текстовые надписи на кнопках управления;
- Picture Buttons — пиктограммы на кнопках управления;
- No Buttons — кнопки управления отсутствуют.

В третьей диалоговой панели Form Wizard (Step 3 — Sort Order) определяется порядок сортировки данных. В списке Available Fields указываются имена полей, по значениям которых надо выполнить сортировку данных. А с помощью кнопок Ascending (по возрастанию) и Descending (по убыванию) определяется порядок сортировки. С помощью кнопки Add поля сортировки заносятся в окно Selected Fields.

В четвертой диалоговой панели Form Wizard (Step 4 — Finish) определяется заголовок созданной экранной формы, который пишется в поле ввода Type a title for your form. Далее с помощью кнопок радио-группы выбирается вариант продолжения работы:

- Save form for later use — сохранить созданную форму;
- Save and run form — сохранить экранную форму и поставить ее на

выполнение;

- Save form and modify it in the Form Designer — сохранить экранную форму и открыть ее в Конструкторе для модификации.

Если при создании таблицы, на основе которой строится экранная форма, было задано свойство Caption, то содержимое этого свойства используется в качестве надписей к полям. Если свойство Caption не задано, то для надписей к полям используются имена полей.

Созданная экранная форма готова к использованию и не требует генерации программных кодов. Для запуска созданной экранной формы надо из главного меню подать команду Program -> Do установить курсор на имя файла экранной формы и нажать кнопку Ok.

4.4.2 Создание экранной формы с помощью Конструктора форм — Form Designer

Процесс создания формы состоит в размещении компонентов формы (элементы управления, визуальные компоненты и т. д.) на листе формы и определения для них свойств и событий при выполнении различных действий. Созданная экранная форма может находиться в одном из двух состояний:

- Режим проектирования – на форме размещаются компоненты, описываются их свойства и события

- Рабочий режим — файл формы поставлен на выполнение. После чего в экранную форму выводятся данные, которые можно просматривать, редактировать, удалять и производить некоторые действия (сортировка, поиск, фильтрация и т. д.) — активное состояние.

Для открытия окна Конструктора форм надо выполнить одно из действий:

1) из главного меню подать команду File -> New, в появившейся на экране диалоговой панели выбрать кнопку New File и указать тип создаваемого файла Form;

2) вызвать на экран окно проекта и выбрать вкладку Document, затем указать группу Form и нажать кнопку New;

3) вызвать на экран окно проекта и выбрать вкладку Document, затем указать группу Form и из главного меню подать команду File -> New, в появившейся на экране диалоговой панели выбрать кнопку New File и указать тип создаваемого файла Form.

При работе с Конструктором форм можно использовать четыре панели инструментов:

- Form Controls — предназначена для выбора и размещения компонентов на листе формы;

- Form Designer — предназначена для управления формой и вызовом на экран остальных трех панелей инструментов;

- Layout — предназначена для выравнивания компонентов на листе

формы;

- Color Palette — предназначена для задания цвета компоненту.

Чтобы разместить компонент на форме, надо на панели инструментов выбрать нужный компонент (установить курсор мыши на нужный компонент и щелкнуть левой кнопкой). Затем курсор мыши расположить на поле формы и либо щелкнуть левой кнопкой мыши, либо, зажав левую кнопку мыши, нарисовать прямоугольный контур, внутри которого будет размещен выбранный компонент.

В каждый момент времени на форме может быть активен только один компонент. Активный компонент на форме выделен по контуру черными маркерами. Для того чтобы сделать компонент активным, надо разместить на нем курсор мыши и щелкнуть левой кнопкой мыши. Для отмены активности компонента надо либо выделить другой компонент, либо щелкнуть левой кнопкой мыши на поле формы.

Для активного компонента на экран можно вывести окно свойств Properties (окно свойств может быть постоянно открыто) с помощью команды главного меню View -> Properties. При смене активного компонента в окно Properties автоматически выводится информация для выбранного компонента. Окно Properties содержит вкладки:

- All — содержит список (в алфавитном порядке) всех свойств и Методов выбранного компонента.

- Data — содержит свойства компонента, относящиеся к источнику данных.

- Methods — содержит список всех методов компонента.

- Layout — содержит свойства, определяющие внешний вид компонента.

- Other — содержит все свойства, которые не вошли в остальные вкладки.

При работе с окном Properties выбирают нужное свойство компонента и присваивают ему значение одним из способов:

- с клавиатуры вписывают требуемое (допустимое) значение;

- с помощью кнопки открывают список, из которого выбирают нужное значение;

- с помощью кнопки выводят на экран диалоговую панель,

- с помощью которой формируют нужное значение.

На каждой вкладке выше списка свойств расположены три кнопки:

- 1) вывод на экран построитель выражений (формул) для задания значения свойства;

- 2) подтверждение ввода значения свойства;

- 3) отказ от введенного значения свойства.

Кроме окна Properties свойства и методы компонента можно задать программным способом.

Изменить геометрические размеры активного компонента можно, буксируя один из маркеров выделения компонента или изменяя значения свойств Height и Width.

Изменить положение компонента на форме можно, буксируя компонент

или изменяя значения свойств Left и Top.

Активный компонент можно удалить либо нажав клавишу Backspace, либо нажав клавишу Del, либо из главного меню подать команду Edit -> Cut, либо с помощью кнопки графического меню

Для удобства размещения объектов на форме на листе формы выведена сетка. Установить или удалить сетку можно с помощью команды главного меню View -> Grid Line. Деления сетки (размер шага) можно задать в диалоговой панели Grid Properties, которая выводится на экран командой главного меню Format S Grid Scale.

Компоненты можно размещать строго по линиям сетки, если главного меню подать команду Format -> Snap to Grid. По умолчанию при нажатии клавиши Tab фокус управления (порядок обхода компонентов) передается в естественном порядке слева направо и сверху вниз. Изменить порядок обхода компонентов (передачу фокуса управления) можно в одном из режимов: Interactive — интерактивный, то есть в порядке расположения элементов на форме, и в режиме By List — в естественном порядке. Для установления одного из режимов надо, находясь в Конструкторе формы, из главного меню подать команду Tools -> Options — на экране диалоговая панель Options, на которой надо выбрать вкладку Forms и в раскрывающемся списке Tab Ordering выбрать одну из команд : Interactive или By List. Команда главного меню View -> Tab Order в режиме Interactive выводит на экран порядковый номер компонента при переходе от одного компонента к другому. Удерживая клавишу Shift в зажатом состоянии, можно с помощью щелчка мыши изменить порядок передачи фокуса управления. По окончании перестановок надо нажать кнопку Reorder. Команда главного меню View -> Tab Order в режиме By List будет выводить на экран диалоговую панель Tab Order, где в окне представлены имена расположенных на форме компонент в порядке получения ими фокуса управления. С помощью мыши или кнопок By Column и By Row можно переставить имена компонент в желаемом порядке .

4.5 Создание отчетов

4.5.1 Виды отчетов

При работе с базами данных информация хранится в таблицах. Просмотр табличной информации в явном виде неудобен. Поэтому информацию из таблицы (или таблиц) извлекают, группируют, сортируют и выводят на экран или бумагу. Под отчетом понимают отформатированную информацию, которая выводится на экран, принтер или в файл.

FoxPro позволяет создать несколько видов отчетов:

- одностраничный табличный отчет;
- многостраничный табличный отчет;
- отчет в свободной форме;

- почтовая этикетка.

Табличный отчет представляет собой регулярную структуру, которая состоит из произвольного количества однотипных записей. В качестве столбцов (полей) можно использовать физические поля таблицы (или нескольких таблиц) и вычисляемые поля.

По желанию пользователя строки могут быть сгруппированы и отсортированы.

Если отчет создан в свободной форме, то информация, принадлежащая одной строке таблицы, может быть размещена на экране или бумаге произвольным способом. В этом случае одна строка занимает весь экран или весь печатный лист.

Почтовая этикетка является разновидностью отчета в свободной форме. На части печатного листа формируют почтовый адрес адресата и адресанта. Затем созданный блок многократно размножают на листе. При этом в каждую почтовую этикетку будет помещен уникальный почтовый адрес адресата (клиента фирмы) и одинаковый почтовый адрес адресанта (самой фирмы).

В Visual FoxPro создать любой из отчетов можно одним из способов:

- Стандартный отчет (Quick Report) — отчет создается по данным одной или нескольких таблиц; в отчет помещаются все поля всех указанных таблиц;

- Мастер отчетов (Report Wizard) — отчет создается по данным одной или нескольких таблиц; допускается произвольный выбор полей из указанных таблиц, сортировка и группировка данных, изменение стиля отображения данных;

- Конструктор отчетов — при создании отчета любого вида предварительно открывается база данных (проект), для которой надо создать отчет.

4.5.2 Создание отчетов с помощью мастера

Для запуска Мастера отчетов надо выполнить следующее:

На вкладке окна проекта выбрать кнопку Document. Если вкладка Document содержит кнопки управления, то выбрать кнопку New. Если на вкладке Document кнопок управления нет, то на экран надо вызвать контекстное меню, из которого выбрать команду New.

Эту же процедуру можно выполнить через главное меню Visual FoxPro, подав команду File, а в подменю надо указать команду New. В том и другом случае на экран выводится диалоговая панель New Report (рисунок 25), где надо выбрать одну из кнопок:

- Report Wizard — запускает Мастер отчетов.
- New Report — запускает Конструктор отчетов.



Рисунок 25 - Задание способа создания отчета

После запуска Мастера отчетов надо определить, какие данные из одной или нескольких таблиц будут помещены в создаваемый отчет. Для этого в диалоговой панели Wizard Selection надо выбрать одну из команд:

- *One-to-Many Report Wizard* — создание отчета по данным из нескольких таблиц, тогда между таблицами будут установлены взаимосвязи «один-ко-многим».

- *Report Wizard* — создание отчета по данным одной таблицы.

В диалоговой панели Wizard Selection надо выбрать команду Report Wizard и нажать кнопку Ок.

Все диалоговые окна Мастера отчетов содержат по пять кнопок управления:

- Cancel — отказ от построения отчета.
- Back — возврат к предыдущему шагу построения отчета;
- Next — переход к следующему шагу построения отчета;
- Finish — построить отчет по указанным данным;
- Help — выводит на экран контекстную подсказку.

На первом шаге Мастера отчетов «Step I — Select fields» (рисунок 26) надо определить имя таблицы, по значениям которой будет создан отчет. В раскрывающемся списке Databases and tables надо выбрать имя базы данных, которой принадлежит нужная нам таблица. Ниже раскрывающегося списка расположено окно, в котором перечислены имена всех таблиц, принадлежащих выбранной базе данных. Справа от раскрывающегося списка Databases and tables расположена кнопка обзора, при помощи которой можно определить место хранения базы данных.

В окне Available fields указаны имена полей выбранной таблицы. Поля, значения которых надо поместить в отчет, по очереди выделяют курсором и нажимают одну из кнопок.

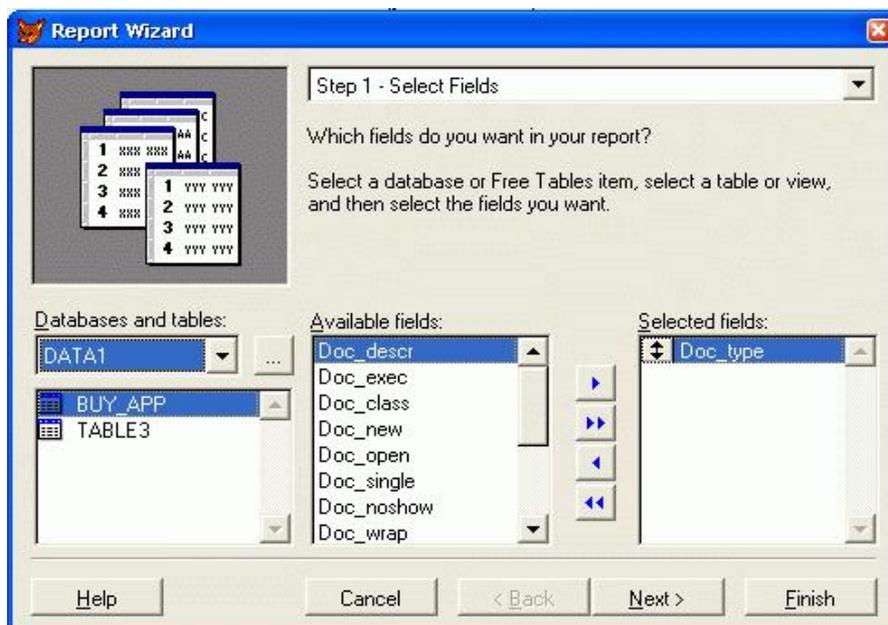


Рисунок 26 – Первый шаг мастера отчетов

На втором шаге «Step 2 — Group Records» (рисунок 27) создания отчета определяется необходимость группировки данных в отчете.

По умолчанию группировка данных не производится (None), то есть данные помещаются в отчет в том порядке, как они хранятся в таблице.

Для задания группировки по значению какого-либо поля надо в раскрывающемся списке «1» выбрать имя нужного поля. Допускается группировка данных внутри указанных групп данных по значениям других полей. Глубина вложения группировок — 3, то есть имеются раскрывающиеся списки с именами «2» и «3».

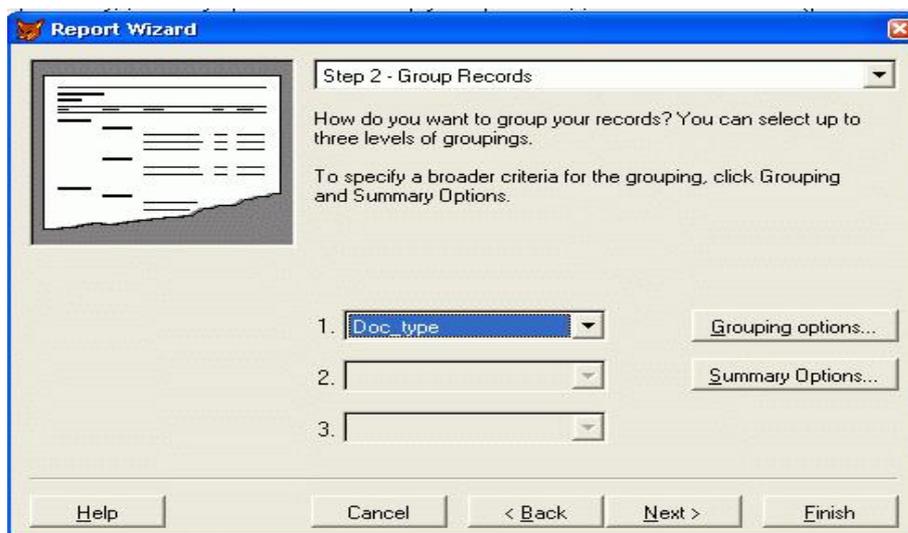


Рисунок 27 – Второй шаг мастера отчетов

Если задана группировка по значениям хотя бы одного поля, то становится доступной кнопка Grouping Options. Эта кнопка позволяет задать интервалы группировки значений поля и выводит на экран диалоговую панель Grouping Intervals (рисунок 28).

Если поле группировки символьное, то интервал группировки Entire Field задается либо по одной первой букве, либо по первым двум буквам и т. д. Если поле группировки числовое, то интервал группировки Exact Number задается либо по первому десятичному разряду, либо по первым двум десятичным разрядам и т. д. Интервал группировки может быть задан по каждому указанному полю группировки данных.

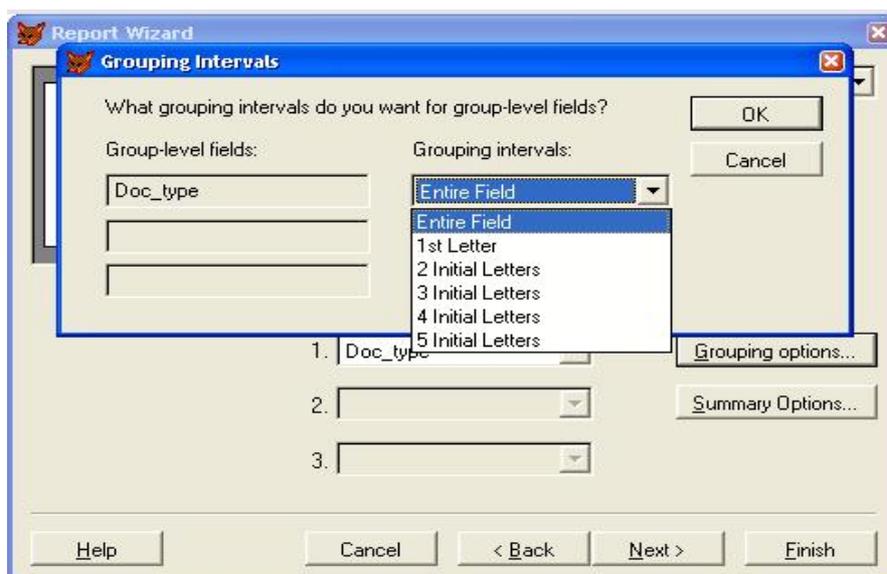


Рисунок 28 - Задание группировки данных

Для каждого числового столбца (поля) отчета можно задать вычисление итогового значения по столбцу. После нажатия кнопки Summary Options на экран выводится диалоговая панель Summary Options.

Для каждого числового столбца (поля) можно задать вычисление следующих итоговых значений:

- Sum — вычислить сумму по столбцу;
- Avg — вычислить среднее значение по столбцу;
- Count — определить количество значений в столбце;
- Min — определить минимальное значение столбца;
- Max — определить максимальное значение столбца.

С помощью кнопок радиогруппы определяется объем итоговых вычислений и область размещения данных.

Назначение кнопок:

- Detail and Summary — задает область размещения данных, вычисление промежуточных итоговых значений по группам и вычисление итогового значения по столбцу.

- Summary Only — задает область размещения данных и вычисление итогового значения по столбцу.

- No totals — задает область размещения данных.

Итоговые значения не вычисляются. Значения промежуточных итоговых значений по группам можно отображать в отчете в виде абсолютных значений, тогда флажок Calculate percent of total for sums должен быть выключен, либо в виде процента от общего итогового значения по столбцу, тогда флажок Calculate percent of total for sums должен быть включен.

На третьем шаге «Step 3 — Choose Report Style» Мастера отчетов определяется один из допустимых стилей выполнения отчета (рисунок 29).

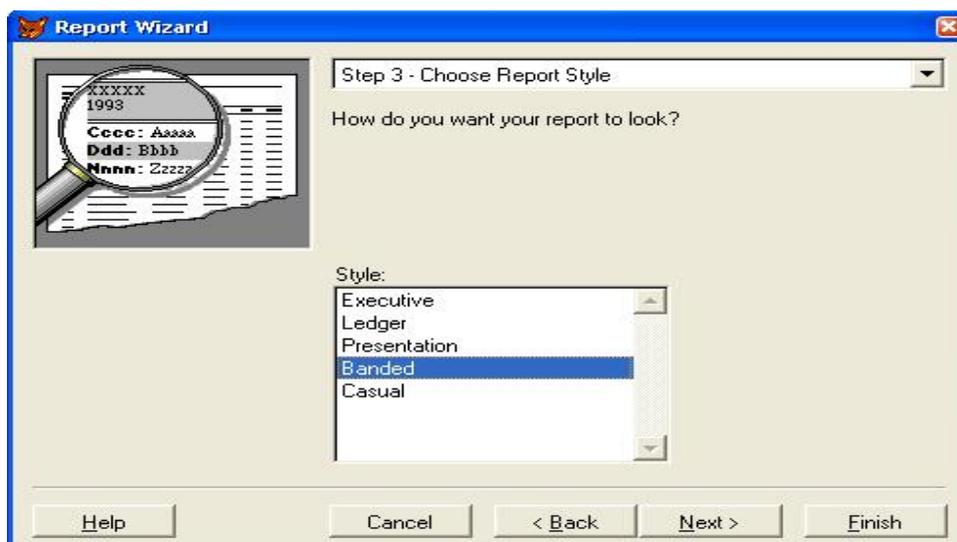


Рисунок 29 - Третий шаг мастера отчетов

Задание нужного стиля выполняется выделением соответствующей строки в окне Style.

В специальном окне, расположенном в верхнем левом углу диалоговой панели четвертого шага Мастера отчетов, можно просмотреть внешний вид выбранного стиля исполнения отчета.

На четвертом шаге «Step 4 — Define Report Layout» Мастера отчетов определяется порядок размещения объектов в отчете.

При создании отчета по значениям одной таблицы на этом шаге построения отчета можно определить только ориентацию листа бумаги с помощью радиогруппы Orientation, выбрав одну из кнопок:

- Portrait — вертикальное (книжное) расположение листа;
- Landscape — горизонтальное (альбомное) расположение листа.

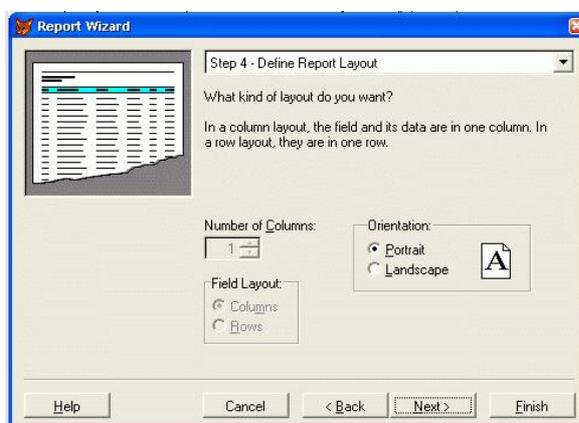


Рисунок 30 - Задание внешнего вида отчета

Остальные элементы управления, расположенные на диалоговой панели, не доступны для выбора.

На пятом шаге «Step 5 — Sort Records» Мастера отчетов определяется необходимость сортировки данных по одному или нескольким полям таблицы.

В списке Available fields of index tag указаны имена полей, по значениям которых можно выполнить сортировку данных. Если группировки полей (шаг 2) не было, то в этом списке будут представлены имена всех полей таблицы, помещаемых в отчет.

Если на втором шаге Мастера отчетов были заданы имена полей для группировки данных, то имена этих полей в список Available fields of index tag не будут включены.

Аналогично шагу 2 Мастера отчетов в списке Available fields of index tag курсором выделяется имя поля сортировки и нажатием кнопки «Add» переносится в список Selected fields.

Для удаления имени поля из списка Selected fields его надо выделить курсором и нажать кнопку Remove.

Для каждого поля сортировки можно задать направление сортировки, включив одну из кнопок радиогруппы:

- Ascending — сортировка по возрастанию значений;
- Descending — сортировка по убыванию значений.

На шестом шаге «Step 6 — Finish» Мастера отчетов задается заголовок созданного отчета и определяется дальнейший режим работы Visual FoxPro. Текст заголовка отчета пишется в поле ввода Type a title for your report.

Дальнейший режим работы с Visual FoxPro определяется включением одной из кнопок радиогруппы Select an option and click Finish:

- Save report for later use — сохранить созданный отчет.
- Save report and modify it in the Report Designer — сохранить созданный отчет и открыть его заново в Конструкторе отчетов для выполнения модификации отчета.
- Save and print report — сохранить и распечатать отчет.

На диалоговой панели размещены два флажка:

- Use display setting stored in the database — при включенном флажке для отображения значений полей используются установки, указанные в базе данных.

- Wrap fields that do not fit — при включенном флажке разрешить перенос на

следующую строку символов значения полей, если значение не умещается в отведенном размере строки.

Кнопка Preview предназначена для предварительного просмотра созданного отчета перед сохранением на диске. Если созданный отчет не устраивает пользователя, то с помощью кнопки Back можно вернуться назад на один или несколько шагов и исправить отчет.

Нажатие кнопки Finish завершает создание отчета.

4.5.3 Создание отчетов с помощью конструктора

Для запуска Конструктора отчетов в диалоговой панели New Report надо выбрать кнопку New Report — экране появится рабочее окно Конструктора отчетов (рисунок 31). Как можно было заметить ранее, отчет должен содержать обязательные реквизиты: заголовок отчета, данные в табличном или произвольном виде, а также дополнительные элементы: пояснительный текст, рамки и т. д., предназначенные для удобства восприятия основных данных.

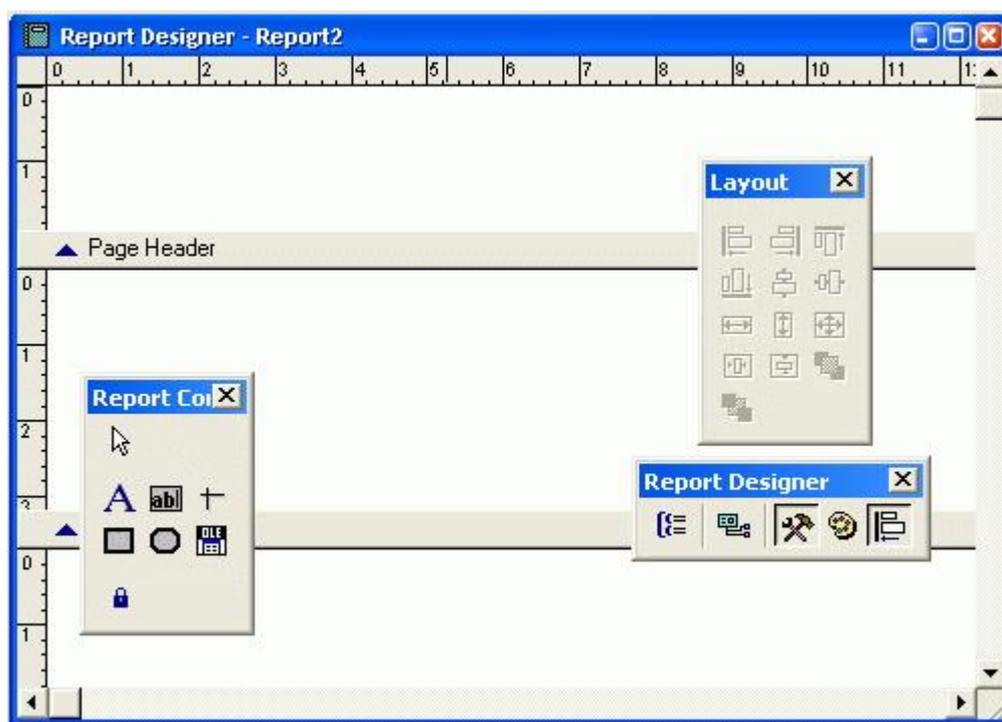


Рисунок 31 - Рабочее окно Конструктора отчетов и панель инструментов

Назначение кнопок панели инструментов:

-  Select Object — указатель выбора объекта отчета.
-  Label — размещает текст.
-  Field— размещает поле.
-  Line — рисует линию.
-  Rectangle — рисует прямоугольник.
-  Rounded Rectangle — рисует прямоугольник с закругленными краями.
-  Picture/OLE Bound Control — помещает рисунок в отчет.
-  Button Lock — закрепляет выбор кнопки.

Первоначально рабочее окно Конструктора отчетов содержит три области (полосы):

Page Header — информация, размещенная в этой полосе, распечатывается в

начале каждой страницы (верхний колонтитул).

Detail — размещается содержимое полей таблиц или результат вычислений над полями. Page Footer — информация, размещенная в этой полосе, распечатывается в конце каждой страницы (нижний колонтитул).

Дополнительно могут быть добавлены следующие области:

Title — информация перед основным отчетом: может быть имя отчета, сопроводительное письмо и т. д.

Group Header — информация, используемая при группировке. Группы помогают идентифицировать информацию, содержащуюся на каждом уровне группировки.

Group Footer — итоговая информация по группе.

Summary — информация размещается один раз после основного отчета и содержит общие суммы или текст, подводящий итог содержимого отчета.

При создании и редактировании отчета возможны следующие операции:

- добавление области;
- перемещение области;
- удаление области.

Процесс создания табличного отчета состоит из следующих этапов:

- 1) определение окружения;
- 2) размещение текста;
- 3) размещение полей;
- 4) размещение линий, прямоугольников и рисунков;
- 5) перемещение объектов;
- 6) сохранение отчета.

5 Хранимые процедуры. Триггеры. Язык запросов SQL

5.1 Условия достоверности, хранимые процедуры, триггеры, представления данных

В реляционных базах данных, к которым относится и Visual FoxPro, для управления данными могут использоваться не только прикладные программы, но и непосредственно сервер базы данных. Это реализуется с помощью условий достоверности ввода данных, триггеров и хранимых процедур, которые являются неотъемлемой частью базы данных.

Удобным средством просмотра информации, хранящейся в базе данных, являются *представления данных*, которые содержат результат выборки данных из одной или нескольких таблиц, удовлетворяющих указанному условию. Представления данных имеют много общего с запросами и таблицами. Так же, как и для запросов вы можете связывать несколько таблиц, указывать отображаемые поля, задавать условие выборки. Просмотр представления данных осуществляется аналогично просмотру таблицы Visual FoxPro.

5.1.1 Условия достоверности ввода данных на уровне записей

Условия достоверности позволяют контролировать ввод данных средствами сервера на уровне записей и полей таблицы. В первом случае условие определяется в окне ввода свойств таблицы, а во втором — в окне свойств поля таблицы. Проверка на уровне записи обычно используется, если необходима проверка при добавлении или удалении записи, а также, если условие проверки изменения записи требует анализа более одного поля.

При определении условий достоверности ввода данных используются триггеры и хранимые процедуры. *Триггеры* задают действия, выполняемые при добавлении, удалении или изменении записей таблицы. *Хранимые процедуры* содержат наиболее часто используемые процедуры, выполняемые сервером базы данных. Если вы определили условия достоверности ввода данных, их проверка осуществляется независимо от способа изменения данных в таблице.

Для определения свойств таблицы откройте окно конструктора для выбранной таблицы и перейдите на вкладку Table (рисунок 32).

Для определения достоверности ввода данных могут использоваться триггеры добавления и изменения, а также поля ввода Rule (Условие) и Message (Сообщение) области Record validation (Проверка правильности ввода записей). В пол Rule вводится логическое выражение, которое может содержать вызов хранимой процедуры. Если значение этого выражения равно True (Истина), то считается, что введены допустимые данные и разрешается переход на другую запись или закрытие таблицы. В противном случае выводится сообщение об ошибке, которое было задано в поле Message.

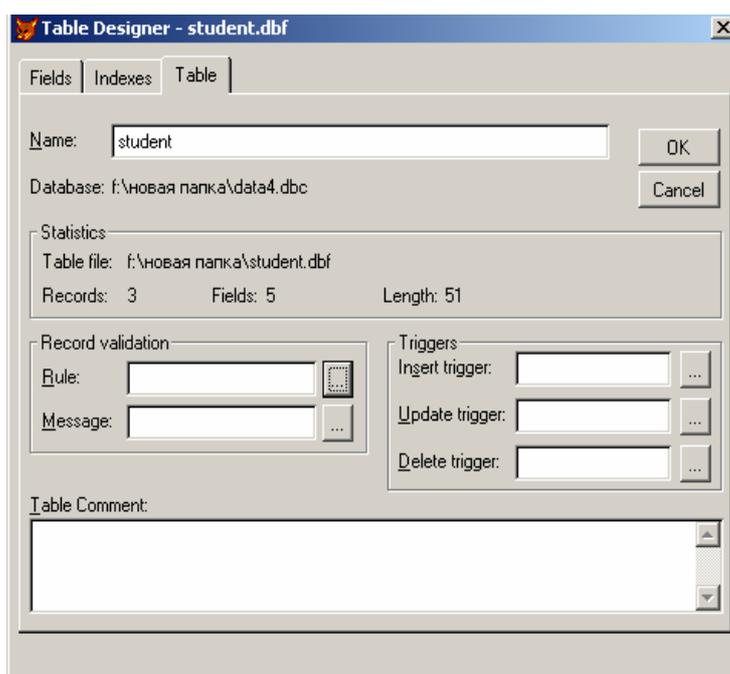


Рисунок 32 - Определение свойств таблицы

5.1.2 Триггеры

В Visual FoxPro для таблиц, входящих в состав базы данных, можно определить следующие триггеры:

Таблица 6 – Триггеры

Триггер	Описание
Insert	Определяет действия, которые будут выполняться после добавления новой записи в таблицу
Update	Определяет действия, которые будут выполняться после изменения записи таблицы
Delete	Определяет действия, которые будут выполняться после удаления записи из таблицы

Для определения триггеров введите в поля ввода Insert trigger, Update trigger или Delete trigger вкладки Table (Таблица) окна конструктора выбранной таблицы оператора сравнения, вызов хранимой процедуры или любое логическое выражение. Если результат вычисления выражения равен True то считается, что введены допустимые значения. В противном случае не происходит сохранения введенных данных и формируется сообщение об ошибке

При использовании хранимых процедур вы сможете не только проверить при условии достоверности ввода данных, но и задать действия, выполняемое при добавлении, удалении и изменении данных.

Вызов триггера Delete осуществляется:

- при выполнении команды DELETE;
- когда вы помечаете запись на удаление в режиме Browse или Edit

Вызов триггера insert осуществляется в следующих случаях:

– При выполнении команд APPEND FROM, APPEND FROM ARRAY, APPEND BLANK;

- при добавлении в таблицу новой записи в режиме Browse или Edit;
- при выполнении команд IMPORT, INSERT - SQL, RECALL;
- при снятии метки об удалении записи в режиме Browse или Edit.

триггер Update вызывается, когда:

– наступает любое событие, которое приводит к модификации записи (например, при изменении значения поля);

– выполняются команды GATHER, REPLACE, REPLACE FROM ARRAY UPDATE – SQL.

При использовании триггеров необходимо учитывать ограничения, имеющиеся в Visual FoxPro:

- при модификации записей, помеченных для удаления, и выполнении команды РАСК триггеры не вызываются
- выполнение команды ZAP не вызывает триггер Delete.
- при использовании буферизации ввода триггер update вызывается только при вызове функции TABLEUPDATE ()

При использовании триггера в окне конструктора перейдите на вкладку Table и очистите поле ввода выражения для триггера или используйте команду Delete TRIGGER.

5.1.3 Хранимые процедуры

Для создания хранимой процедуры нужно выполнить следующие действия:

- 1) в окне проекта выберите базу данных;
- 2) перейдите в группу **Stored Procedures** (Хранимые процедуры);
- 3) нажмите кнопку New;
- 4) открывается окно редактирования хранимых процедур, которое содержит все ранее созданные хранимые процедуры, включая процедуры, создаваемые автоматически при определении условий целостности базы данных.

Предупреждение:

Редактирование или удаление хранимых процедур, которые Visual FoxPro создал автоматически при определении условия целостности данных, может привести к непредсказуемым последствиям.

Для редактирования хранимых процедур в Visual FoxPro можно использовать команду MODIFY PROCEDURE, которая открывает окно редактирования хранимых процедур текущей базы данных.

Для удаления хранимой процедуры вам необходимо в окне редактирования „делить текст удаляемой процедуры и нажать клавишу <Delete> или в окне конструктора проекта установить курсор на имя процедуры и выполнить команду **Remove** (Удалить) окна проекта.

5.2 Классификация реляционных языков

5.2.1 Типы реляционных языков

Реляционные языки обеспечивают типовые операции по обработке реляционных таблиц, позволяют формулировать логические условия, используемые в операциях выборки, проверку целостности (непротиворечивости) данных взаимосвязанных таблиц.

Реляционные языки оперируют с данными как со множествами, применяя к ним основные операции теории множеств. На входе реляционного оператора — множество записей одной или нескольких реляционных таблиц, на выходе — множество записей новой реляционной таблицы. Реляционные языки имеют различный уровень процедурности — содержание и последовательность перехода от входных данных к выходным.

Выделяют следующие разновидности языков реляционной алгебры:

- dBASE-подобные языки приближены к языкам структурного программирования, обеспечивают создание интерфейса пользователя и типовые операции обработки;
- графические реляционные языки, которые ориентированы на конечных пользователей;
- SQL-подобные языки запросов, реализованные в большинстве многопользовательских и распределенных систем управления базами данных.

5.2.2 dBASE-подобные реляционные языки

СУБД реляционного типа, такие, как dBASE, Paradox, FoxPro, Clipper, Rbase и др., используют языки манипулирования данными, обеспечивающие основные операции обработки реляционных баз данных, образующих класс dBASE-подобных (X-Base).

Рассмотрим данный класс языков на примере СУБД класса dBASE.

С помощью полноэкранных команд, вызываемых через главное меню (Управляющий центр, режим Assist и т.п.), осуществляются создание и редактирование схемы реляционной таблицы (файла), ввод и редактирование данных. Для реализации тех же действий на программном уровне имеются соответствующие команды языка (CREATE, MODIFY, UPDATE, DELETE и др.).

Работа с реляционной таблицей (файлом базы данных) организуется в отдельной рабочей области, которой присваивается имя (алиасное имя или номер). После активизации файла к нему можно перейти, указав номер рабочей области. СУБД запоминает указатель на последней обрабатываемой записи (при первоначальном открытии файла текущий номер записи е- 1).

Позиционирование в файле на запись выполняется:

- непосредственно, указанием номера записи (начало или конец файла,

определенный номер записи);

- при поиске записи по заданному логическому условию.

Язык обеспечивает выполнение всех рассмотренных типовых операций над отдельным файлом; а именно:

- APPEND BLANK, BROWSE, CHANGE, EDIT, INSERT — добавление, редактирование записей (в режиме полноэкранного ввода);

- DELETE -удаление записей (в программном режиме);

- SEEK, FIND, LOCATE — поиск записи по условию;

- COPY — копирование всех или части записей активного файла в новый файл;

- CONTINUE - продолжение поиска записи по ранее сформулированному условию и т.п.

Границы области действия команды принимают значения:

- RECORD n — определенная запись с номером n;

- ALL — все записи файла;

- NEXT n — следующие n записей, начиная с текущей;

- REST - все: записи, начиная с текущей; идо конца файла.

Условия выполнения команд задаются с помощью формата ключевых слов FOR и WHILE.

Условие 1 действует в качестве фильтра (ВЫБОРКА) записей исходного файла: если записи соответствуют условию, они участвуют в операции. Условие формулируется применительно к полям записи, например:

FOR [№ зач.книжки]>=1000000 AND [№ зач.книжки] < 2000000

Условие2 позволяет прекратить операцию в случае его нарушения, например:

WHILE [Датарождения] < 1.1.80

Многие команды включают список полей, указываемых за ключевым словом FIELDS, на которые распространяется действие операции.

Пример:

1) DELETE границы FOR условие1 WHILE условие2

Операция логического удаления (пометки) записей активного файла, если они отвечают требованию условия. Операция выполняется до тех пор, пока истинно условие2.

2) COPY TO нов_файл границы FIELDS список_полей FOR условие 1 WHILE условие2

Записи активного файла, удовлетворяющие условию!, если истинно условие2, используются для формирования нового файла, схема которого задается как список_полей.

Таким образом, можно выполнить как горизонтальную, так и вертикальную выборку записей реляционной базы данных.

Совместная обработка файлов выполняется лишь для двух файлов: при этом файлы должны быть предварительно открыты в разных рабочих областях, иметь совпадающие внешние ключи.

Пример:

JOIN WITH псевдоним FOR условие FIELDS список_полей

Запись исходного файла объединяется с записью файла, открытого, под именем псевдоним, если выполняется условие. Формируется новый файл, схема которого задается списком полей.

Для одновременной работы более чем с двумя файлами используются переменные, в которых сохраняются значения полей в качестве внешнего ключа связи.

Кроме того, данный класс реляционных языков реализует типовые конструкции языков структурного программирования:

- циклы (DO WHILE END DO);
- условные операторы (IF ... ELSE ... ENDIF);
- альтернативные операторы (DO CASE ... OTHERWISE ... ENDCASE) и

др.

Реляционные dBASE-подобные языки занимают промежуточное положение между языками манипулирования данными СУБД и языками программирования, обладают выраженной процедурностью обработки, когда явно указывается последовательность действий, приводящих к конечному результату.

5.2.3 Графические (схематичные) реляционные языки

Типичным представителем является язык QBE (Query By Example), реализованный в среде электронных таблиц, в ряде СУБД, в пакете Microsoft Query.

Данный язык относится к языкам манипулирования данными. Работа выполняется со схемой реляционной таблицы с использованием простейших синтаксических конструкций.

Для вертикальной выборки (проекции) записей реляционной таблицы осуществляется пометка отбираемых полей с помощью символа V (помеченное поле выводится в выходную структуру новой реляционной таблицы).

Пример:

Получить список имен и фамилий студентов

СТУДЕНТ	Имя	Фамили	Дата	№зач.	
	√	√			

Для горизонтальной выборки (селекции) задаются логические условия (критерии) отбора записей в поисковых полях.

Условия могут задаваться как выражения, построенные с помощью операторов различного вида:

Арифметические операторы — используются для выполнения вычислений с числами в качестве констант выражения:

- * Умножения двух чисел;
- + Сложения двух чисел;
- Вычитания одного числа из другого;
- / Деления одного числа на другое;

Операторы сравнения — используются для сравнения двух значений:

- > (больше);
- >= (больше или равно);
- < (меньше);
- <= (меньше или равно);
- <> (не равно);
- = (равно).

Логические операторы — используются с выражениями, которые могут быть истинными или ложными:

И (AND) — должны выполняться оба критерия.

Или (OR) — должен выполняться один из критериев.

Не (NOT) — этот критерий не должен выполняться.

Могут использоваться *специальные операторы* типа:

BETWEEN — значение в заданном диапазоне.

IN — одно из значений списка.

IS — с ключевым словом Null определяет, является ли величина нулем (нет значения) или нет (есть значение).

LIKE — использует символы подстановки для сравнения двух значений.

Условия задаются в следующих вариантах:

- для одного поля;
- в одной строке для нескольких полей, считая все условия совместными;
- в разных строках для одного или разных полей, считая их альтернативными.

Язык QBE позволяет вычислять групповые функции (по группе выделенных строк) с помощью функций:

Avg — среднее арифметическое значение поля;

Count — число выбранных записей;

Max — максимальное значение поля;

Min — минимальное значение поля;

Sum — сумма значений поля.

Для использования подобных функций указываются поля, образующие группу записей.

Пример:

Для

подсчета количества студентов с именем Иван и датой рождения в диапазоне 1.1.79 — 1.1.80 создается запрос:

СТУДЕНТ	Имя	Фамилия	Дата рождения	№ зач. книжки	Дата рождения
	Иван	Calc Count	>1 1 79		<1 1 80

Ключевое слово Calc означает вычисление значений по данному полю.

Выражения используются в запросе и для формирования ,новых данных

Пример: Для каждого студента определить количество прожитых на

сегодняшнюю дату дней:

СТУДЕНТ	Имя	Фамилия	Дата рождения	№ зач. книжки
	√	√	⊕ Calc	

Символ ⊕ — идентификатор значения поля, today — встроенная функция вычисления сегодняшней даты.

Для совместной обработки реляционных таблиц строится многотабличный запрос, в котором указываются внешние ключи связи, помечаемые в бланках запроса.

Некоторые версии языка. QBE позволяют создать набор {множество} значений указанного поля одной таблицы, по отношению к которому проверяются значения поля другой таблицы. Операции сравнения выполняются на уровне множества* значений поля с помощью сравнения наборов:

ONLY — второй набор — подмножество первого,

NO — наборы не совпадают,

EVERY — первый набор — подмножество второго,

EXACTLY — наборы совпадают.

Первый набор формируется с помощью ключевого слова SET.

Пример:

Получить сведения о студентах, которые имеют такие же результаты, что и студент с зачетной книжкой 123456

ОЦЕНКА	№ зач. книжки	Код дисциплины	Результат
SET	123456		⊕
	√ ⊕ ⊕	⊕ ⊕	EXACTLY ⊕

СТУДЕНТ	Имя	Фамилия	Дата рождения	№ зач. книжки
	√	√		⊕ ⊕

В ряде СУБД кроме выборки записей возможны операции включения новых записей (INSERT), удаления записей (DELETE) или групповой корректировки выбранных записей (CHANGETO).

5.3 Основные характеристики языка SQL

5.3.1 Краткая характеристика языка SQL

SQL (Structured Query Language) — это язык программирования, который используется при работе с реляционными базами данных в современных СУБД (ORACLE, dBASE IV, dBASE V, Paradox, Access и др.).

Язык SQL стал *стандартом* языков запросов для работы с реляционными базами данных для архитектуры как файл-сервер, так и клиент-сервер, а также в условиях применения системы управления распределенными базами данных. SQL использует ограниченный набор команд, но в то же время — это реляционно полный язык, предназначенный для работы с базами данных, создания *запросов* выборки данных, выполнения вычислений, обеспечения их целостности. Синтаксис версий языка SQL может в определенной степени различаться для отдельных СУБД. Рассмотрим наиболее общие операторы языка SQL.

5.3.2 Операторы языка SQL для работы с реляционной базой данных

5.3.2.1 Создание реляционных таблиц

Создание реляционной базы данных означает спецификацию состава полей: указание имени, типа и длины каждого поля (если это необходимо). Каждая таблица имеет уникальное имя.

Синтаксис оператора создания новой таблицы:

```
CREATE TABLE таблица (поле1 тип [(размер)] [индекс 1]  
[, поле2 тип [(размер)] [индекс2] [...]] [, составной_индекс [...]])
```

где *таблица* - имя создаваемой таблицы;

поле1, поле2 -

имена полей таблицы;

тип - тип поля;

размер - размер текстового поля;

индекс1, индекс2 — директивы создания простых индексов (по

отдельному полю);

составной индекс — директива создания составного индекса.

Каждый индекс имеет уникальное в пределах данной таблицы имя.

Для создания *простого* индекса используется фраза (помещается за именем поля):

```
CONSTRAINT имя индекса {PRIMARY KEY | UNIQUE |  
REFERENCES внешняя_таблица [(внешнее поле)]}
```

Директива создания *составного* индекса (помещается в любом месте после определения его элементов):

```
CONSTRAINT имя {PRIMARY KEY (ключевое 1 [, ключевое2 [, ...]]) |  
UNIQUE (уникальное 1 [, уникальное [, ...]]) | FOREIGN KEY (ссылка1, ссылка2 [,  
...]) REFERENCES внешняя_таблица [(внешнее_поле1 [, внешнее_поле2  
[, ...]])}
```

Служебные слова:

UNIQUE — уникальный индекс (в таблице не может быть двух записей, имеющих одно и то же значение полей, входящих в индекс);

PRIMARY KEY — первичный ключ таблицы (может состоять из нескольких полей; упорядочивает записи таблицы);

FOREIGN KEY — внешний ключ для связи с другими таблицами (может состоять из нескольких полей);

REFERENCES - ссылка на внешнюю таблицу.

Пример:

```
CREATE TABLE Студент
```

```
([Имя] TEXT,
```

```
[Фамилия] TEXT,
```

```
[Дата рождения] DATETIME,
```

```
CONSTRAINT Адр UNIQUE ([Имя], [Фамилия], [Дата рождения]))
```

Будет создана таблица СТУДЕНТ, в составе которой: два текстовых поля: *Имя*, *Фамилия*, одно поле типа дата/время — *Дата рождения*.

Создан составной индекс с именем *Адр* по значениям указанных полей, индекс имеет уникальное значение, в таблице не может двух записей с одинаковыми значениями полей, образующих индекс.

5.3.2.2 Изменение структуры таблиц

При необходимости можно выполнить реструктуризацию таблицы:

- удалить существующие поля;
- добавить новые поля;
- создать или удалить индексы.

Все указанные действия затрагивают одновременно только одно поле или один индекс:

```
ALTER TABLE таблица
```

```
ADD {[COLUMN] поле тип[(размер)] [CONSTRAINT индекс]
```

```
CONSTRAINT составной индекс} |
```

```
DROP {[COLUMN] поле i CONSTRAINT имя индекса} }
```

Опция ADD обеспечивает добавление поля, опция DROP — удаление поля таблицы, добавление опции CONSTRAINT означает подобные действия для индексов таблицы.

Пример:

```
ALTER TABLE Студент ADD COLUMN [Группа] TEXT(5)
```

Для создания нового индекса для существующей таблицы можно использовать также команду:

```
CREATE [ UNIQUE ] INDEX индекс ON таблица (поле[,...])
```

```
[WITH { PRIMARY | DISALLOW NULL | IGNORE NULL }]
```

Фраза *WITH* обеспечивает наложение условий на значения полей, включенных в индекс:

DISALLOW NULL - запретить пустые значения в индексированных полях новых -записей;

IGNORE NULL - включать в индекс записи, имеющие пустые значения в индексированных полях.

Пример:

```
CREATE INDEX Гр ON Студент ([Группа]) WITH DISALLOW NULL
```

5.3.2.3 Удаление таблицы

Для удаления таблицы (одновременно и структуры, и данных) используется команда:

```
DROP TABLE имя таблицы
```

Для удаления только индекса таблицы (сами данные не разрушаются) выполняется команда:

```
DROP INDEX имя индекса ON имя таблицы
```

Пример:

```
DROP INDEX Адр ON Студент - удален только индекс Адр
```

```
DROP TABLE Студент-удалена вся таблица
```

5.3.2.4 Ввод данных в таблицу

Формирование новой записи в таблице выполняется командой:

```
INSERT INTO таблица_куда [(поле1[, поле2[,...]])]  
VALUES (значение1[, значение2[,...]]);
```

Указывается имя таблицы, в которую добавляют запись, состав полей, для которых вводятся значения.

Пример:

```
INSERT INTO Студент ([Фамилия], [Имя], [Дата рождения])  
VALUES ("Петров", "Иван", 2.3/3/80)
```

Возможен *групповой* ввод записей (пакетный режим), являющихся результатом выборки (запроса) из других таблиц:

```
INSERT INTO таблица_куда[внешняя_база данных]  
SELECT [источник.]поле 1[, поле2[,...]]  
FROM выражение  
WHERE условие
```

Перед загрузкой выполняется оператор подзапроса *SELECT*, который и формирует выборку для добавления. Фраза *SELECT* определяет структуру данных-источника передаваемых записей - имена таблицы и полей, содержащих исходные данные для загрузки в *таблицу_куда*; *FROM* позволяет указать имена исходных таблиц, участвующих в формировании выборки, а

фраза *WHERE* — задает условия выполнения подзапроса. Структура данных выборки должна соответствовать структуре данных таблицы, в которую производится добавление.

Добавление (перезагрузка) записей возможна и во *внешнюю* базу данных, для которой указывается полностью специфицированное имя {*диск, каталог, имя, расширение*).

Пример:

```
INSERT INTO Студент SELECT [Студент-заочник].* FROM [Студент-заочник]
```

Все записи таблицы [Студент-заочник] в полном составе полей будут добавлены в таблицу *Студент*.

Примечание. Структуры таблиц должны совпадать.

Пример:

```
INSERT INTO Студент SELECT [Студент-заочник].* FROM [Студент-заочник] WHERE [Дата рождения] >= #01/01/80#
```

Записи таблицы [Студент-заочник] добавляются в таблицу *Студент*, если дата рождения студента больше или равна указанной.

5.3.2.5 Операции соединения таблиц

Операцию *INNER JOIN* можно использовать в любом предложении *FROM*. Она создает симметричное объединение, наиболее частую разновидность внутреннего объединения: записи из двух таблиц объединяются, если связующие поля этих таблиц содержат одинаковые значения:

```
FROM таблица1 INNER JOIN таблица2 ON таблица1 .поле1 = таблица2.поле2
```

Данный оператор описывает симметричное соединение двух таблиц по ключам связи (*поле1; поле2*). Новая запись формируется в том случае, если в таблицах содержатся одинаковые значения ключей связи.

Возможные варианты операции:

- *LEFT JOIN* (левостороннее) соединение — выбираются все записи "левой" таблицы и только те записи "правой" таблицы, которые содержат соответствующие ключи связи;

- *RIGHT JOIN* (правостороннее) соединение — выбираются все записи "правой" таблицы и только те записи "левой" таблицы» которые содержат соответствующие ключи связи.

Пример:

```
SELECT Студент.*, Оценка.* FROM Студенты INNER JOIN Оценка ON Студент.[№ зач.книжки] = Оценка.[№ зач.книжки];
```

```
SELECT Студент.*, Оценка.* FROM Студенты LEFT JOIN Оценка ON Студент.[№ зач.книжки] = Оценка.[№ зач.книжки];
```

```
SELECT Студент.*, Оценка.* FROM Студенты RIGHT JOIN Оценка ON Студент.,[№ зач.книжки] = Оценка.[№ зач.книжки];
```

В первом случае создается симметричное соединение двух таблиц по полю [№ зач.книжки]; Не выводятся записи, если значение их ключей связи (указанное поле) не представлено в двух таблицах. Во втором случае будут выведены все записи таблицы СТУДЕНТ и соответствующие им записи таблицы ОЦЕНКА. В третьем случае — наоборот, все записи таблицы ОЦЕНКА и соответствующие им записи таблицы СТУДЕНТ.

Операции JOIN могут быть вложенными для последовательного соединения нескольких таблиц.

Пример:

```
SELECT Студент., .Оценка. Дисциплина.[Наименование дисциплины]
FROM (Студент INNER JOIN (Оценка INNER JOIN
(Дисциплина ON Оценка. [Код дисциплины] =
Дисциплина. [Код дисциплины])
ON Студент. [№ зач.книжки]=Оценка. [№ зач.книжки])
```

Сначала происходит соединение таблиц ОЦЕНКА и ДИСЦИПЛИНА по ключу связи [Код дисциплины]. Соединение симметричное, то есть если коды дисциплины не совпадают, записи этих таблиц не соединяются. Затем происходит соединение таблиц СТУДЕНТ и ОЦЕНКА по ключу связи [№ зач.книжки].

Таким образом, на выходе запроса — результат соединения трех таблиц, но при условии совпадения ключей связи.

5.3.2.6 Удаление записей в таблице

В исходной таблице можно удалять отдельные или все записи, сохраняя при этом структуру и индексы таблицы. При удалении записей в индексированной таблице автоматически корректируются ее индексы:

```
DELETE [таблица,*] FROM выражение WHERE условия_отбора
```

Полная чистка таблицы от записей и очистка индексов выполняется операцией: DELETE * FROM *таблица*

Пример:

```
DELETE * FROM Студент
```

Все ранее загруженные записи будут удалены.

```
DELETE*FROM Студент WHERE [Дата рождения]>#1.1.81#
```

Удаляются только те записи, в которых поле [Дата рождения] больше указанной даты.

Данная операция удаляет записи в таблице, связанные с другой таблицей: условия удаления записей могут относиться к полям связанных таблиц:

```
DELETE таблица.* FROM таблица INNER JOIN др._таблица
ON таблица.[полеN] = др._таблица.[полеM] WHERE условие
```

Пример:

DELETE Студент. From Студент INNER JOIN [Студент заочник]
ON Студент.[Группа]- [Студент заочник]. [Группа]*

Удаляются записи в таблице *Студент*, для которых имеются связанные записи в таблице *[Студент заочник]*.

Примечание. Средствами Microsoft ACCESS невозможно восстановить записи, удаленные с помощью запроса на удаление записей.

5.3.2.7 Обновление (замена) значений полей записи

Можно изменить значения нескольких полей одной или группы записей таблицы, удовлетворяющих условиям отбора:

*UPDATE таблица SET новое значение WHERE условия отбора
новое_значение* указывается как *имя_поля=новое значение*

Пример:

UPDATE Студент SET [Группа] = "1212"

WHERE [Фамилия] LIKE 'B' AND [Дата рождения] <= #01/01/81#*

Студентов, чьи фамилии начинаются на букву B и дата рождения не превышает указанной, перевести в группу 1212.

*UPDATE Студент INNER JOIN [Студент заочник] ON Студент.[Группа]=
[Студент заочник]. [Группа] SET [Группа]= [Группа]&"a"*

*В таблице Студент изменить номера групп, если они встречаются в
таблице*

[Студент заочник], добавив букву a.

5.4 Организация запросов к базе данных на языке SQL

5.4.1 Синтаксис оператора SELECT

Выборка с помощью оператора SELECT — наиболее частая команда при работе с реляционной базой данных. Этот оператор обладает большими возможностями по заданию структуры выходной информации, указанию источников входной информации, способа упорядочения выходной информации, формированию новых значений и т.п. (таблица 7).

При выполнении выборки могут формироваться и новые данные, так называемые вычисляемые поля, являющиеся результатом обработки исходных данных. Возможно упорядочение выводимых данных, формирование групп записей, подсчет групповых итогов, формирование подмножеств данных (записей), являющихся основой для формирования условий по обработке следующего этапа — вложенных запросов.

Универсальный оператор SELECT имеет следующую конструкцию:

SELECT [предикат] { [таблица* | [таблица.]поле1 [,*

[таблица.]поле2.[...]]}
 [AS псевдоним! [, псевдоним2 [...]]]
 FROM выражение [...] [IN внешняя_база_данных\
 [WHERE...]
 [GROUP BY...]
 [HAVING...]
 [ORDER BY...]
 [WITH OWNERACCESS OPTION]

Синтаксис оператора SELECT весьма лаконично реализует сложные алгоритмы запросов. Практическое освоение элементов постепенное — методом от простого к сложному, а отладка оператора сложной конструкции может идти по частям.

Таблица 7 - Аргументы оператора SELECT

Аргумент	Назначение
Предикат	Предикаты используются для ограничения числа возвращаемых записей: ALL - все записи; DISTINCT — записи, различающиеся в указанных для вывода полях; DISTINCTROW — полностью различающиеся записи по всем полям; TOP — возврат заданного числа или процента записей в диапазоне, соответствующем фразе ORDER BY
Таблица	Имя таблицы, поля которой формируют выходные данные
Поле1, поле2	Имена полей, используемых при отборе (порядок их следования определяет выходную структуру выборки данных)
Псевдоним1, Псевдоним2	Новые заголовки столбцов результата выборки данных
FROM	Определяет выражение, используемое для задания источника формирования выборки (обязательно присутствует в каждом операторе)
Внешняя база данных	Имя внешней базы данных — источника данных для выборки
[WHERE...]	Определяет условия отбора записей (необязательное)
[GROUP BY...]	Указание полей (максимум — 10) для формирования групп, по которым возможно вычисление групповых итогов; порядок их следования определяет виды итогов (старший, промежуточный и т.п.) — необязательное

Продолжение таблицы 7

Аргумент	Назначение
[HAVING...]	Определяет условия отбора записей для сгруппированных данных (задан способ группирования GROUP BY...) — необязательное
[ORDER BY...]	Определяет поля, по которым выполняется упорядочение выходных записей; порядок их следования соответствует старшинству ключей сортировки. Упорядочение возможно как по возрастанию (ASC), так и по убыванию (DESC) значения выбранного поля
[WITH OWNERACCESS OPTION]	При работе в сети в составе защищенной рабочей группы для указания пользователям, не обладающим достаточными правами, возможности просматривать результат запроса или выполнять запрос

Изучать оператор SELECT лучше всего на конкретных примерах. Слово SELECT определяет структуру выводимой информации, это могут быть поля таблиц, вычисляемые выражения.

Вычисляемое выражение состоит из:

- полей таблиц;
- констант;
- знаков операций;
- встроенных функций;
- групповых функций SQL.

Пример:

SELECT [Имя],[Фамилия] FROM Студент

SELECT TOP 5 [Фамилия] FROM Студент

SELECT TOP 5 [Фамилия] FROM Студент ORDER BY [Группа]

В первом случае выбираются все записи таблицы Студент в составе указанных полей. Если отбираются все поля в том же самом порядке, что и в структуре таблицы, можно указать символ точки. Во втором случае отбирается 5 первых фамилий студентов, в третьем случае — выбирается 5 первых фамилий студентов, упорядочение записей осуществлено по учебным группам.

Если используются одноименные поля из нескольких таблиц, включенных в предложение FROM, следует указать перед именем такого поля имя таблицы через (точку): [Студент заочник].[Группа] и [Студент].[Группа] — два одноименных поля из разных таблиц.

Для изменения заголовка столбца с результатами выборки используется служебное слово AS.

Пример:.

SELECT DISTINCT [Дата рождения] AS Юбилей FROM Студент

SELECT [Фамилия] & " "& [Имя] AS ФИО, [Дата рождения] AS Год FROM Студент

В первом случае будут выведены неповторяющиеся даты рождения студентов, которые имеют новое наименование — Юбилей. Во втором случае в результирующей таблице присутствуют все записи, но вместо [Дата рождения] указан Год и вместо Фамилия и Имя, соединенных вместе через пробел, — ФИО.

Наиболее часто слово AS применяется для именования вычисляемых полей.

5.4.2 Задание условий выборки

Предложение WHERE может содержать выражения, связанные логическими операторами, с помощью которых задаются условия выборки (таблица 8).

Кроме того, могут использоваться операторы для построения условий:

LIKE - выполняет сравнение строковых значений;

BETWEEN...AND— выполняет проверку на диапазон значений;

IN — выполняет проверку выражения на совпадение с любым из элементов списка;

IS — проверка значения на Null (пусто).

Условие обеспечивает "горизонтальную" выборку данных, т.е. на выход "пройдут" только те записи, которые удовлетворяют сформулированным условиям.

Таблица 8 - Логические операторы для построения условий выборки

Оператор	Назначение	Оператор	Назначение	Оператор	Назначение
AND	логическое И или конъюнкция (логическое умножение)	Imp	логическая импликация выражений	Or	логическое ИЛИ дизъюнкция (включающее Or)
Eqv	проверка логической эквивалентности выражений .	Not	отрицание	Xor	логическое ИЛИ (исключающее Or)

Пример:

- 1) SELECT Студент.* FROM Студент WHERE [Дата рождения] >=#01.01.79#
- 2) SELECT Студент.* FROM Студент WHERE [Дата рождения] >=#01.01.79#

AND [Группа] IN ("1212", "1213")

3) SELECT Студент,* FROM Студент WHERE [Дата рождения] BETWEEN#01.0i.79 AND #01.01.81# AND [Группа] IN ("1212", "1213")

4) SELECT Студент.* FROM Студент INner JpIN [Студент заочник] ON Студент. [Группа]=[Студент заочник]. [Группа] WHERE Студент.[Дата рождения] >=#01:01.79#

В первом случае выбираются студенты, дата рождения которых позже 1.1.79. Во втором случае будут отображены все студенты, обучающиеся в группах 1212 или 1213 и дата рождения которых позже 1.1.79. В третьем случае выбираются студенты, дата рождения которых находится в заданном диапазоне, и они обучаются в любой из указанных групп. В четвертом случае выбираются студенты, которые обучаются в тех же группах, что и студенты-заочники, дата рождения которых позже 1.1.79.

5.4.3 Групповые функции SQL

Групповые функции необходимы для определения статистических данных на основе наборов числовых значений:

- Avg — вычисляет арифметическое среднее набора чисел, содержащихся в указанном поле запроса;
- Count — вычисляет количество выделенных записей в запросе;
- Min, Max — возвращают минимальное и максимальное значения из набора в указан ном поле запроса;
- StDev, StDevPs — возвращают среднеквадратическое отклонение генеральной совокупности и выборки для указанного поля в запросе;
- Sum — возвращает сумму значений в заданном поле запроса;
- Var, VarPs — возвращают дисперсию распределения генеральной совокупности и выборки для указанного поля в запросе.

Для определения полей группирования указывается ключевое слово GROUP BY. Можно указать также слово HAVING для заданного условия по группе при вычислении групповых значений.

Пример:

1)SELECT Фамилия, Avg(Результат) AS Средний_балл FROM Результаты

GROUP BY [№ зач.книжки]

2) SELECT [Код дисциплины], Avg(Результат) AS Средний_балл FROM Результаты GROUP BY [Код дисциплины]

В первом случае создается список фамилий студентов с указанием среднего балла по каждому студенту, во втором случае — список кодов дисциплин и средний балл по дисциплине.

Пример:

1)SELECT Фамилия, Avg(Результат) AS Средний_балл FROM Результаты

GROUP BY [№ зач.книжки] HAVING Avg(Результат) > 4.5

2) *SELECT '{Код дисциплины}, Avg(Результат) AS Средний_балл
FROM Результаты GROUP BY [Код дисциплины] HAVING
Avg(Результат) < 4*

В первом случае создается список фамилий студентов с указанием среднего балла по каждому студенту, выводятся фамилии тех студентов, которые имеют средний балл выше 4.5. Во втором случае выводится список кодов дисциплин со средним баллом при условии, что он ниже 4.

5.4.4 Подчиненный запрос

В инструкцию *SELECT* может быть вложена другая инструкция *SELECT*, *SELECT...INTO*, *INSERT INTO*, *DELETE* или *UPDATE*. Различают основной и подчиненные запросы, которые являются вложенными в основной запрос.

Подчиненный запрос можно использовать вместо выражения в списке полей инструкции *SELECT* или в предложениях *WHERE* и *HAVING*.

Существуют три типа подчиненных запросов:

- 1) сравнение (*ANY | ALL | SOME*) (инструкция);
- 2) выражение [*NOT*] *IN* (инструкция);
- 3) [*NOT*] *EXISTS* (инструкция).

Первый тип — сравнение выражения с результатом подчиненного запроса.

Ключевые слова:

ANY — каждый (сравнение с каждым элементом подчиненной выборки).

ALL — все (сравнение со всеми элементами подчиненной выборки).

SOME — некоторые (сравнение с некоторыми элементами подчиненной выборки).

Пример:

SELECT FROM Оценка WHERE [Результат] > ANY
(SELECT [Результат] FROM Оценка
WHERE Результат.[№зач.книжки]="123124")*

Отбираются только те записи из таблицы Оценка, в которых значение результата больше (выше) каждой оценки студента с № зач.книжки 123124.

Второй тип — выражение, которое должно быть найдено в наборе записей, являющихся результатом выполнения подчиненного запроса.

Пример:

1) *SELECT * FROM Студент WHERE [№ зач.книжки] IN
(SELECT [№ зач.книжки] FROM Оценка WHERE [Результат] > =4)*
2) *SELECT * FROM Дисциплина WHERE [Код дисциплины] NOT IN
(SELECT [Код дисциплины] FROM Оценка)*

В первом случае отбираются студенты, которые в таблице Оценка имеют результат 4 или выше.

Во втором случае отбираются дисциплины, которые не встречаются в таблице Оценка.

Третий тип — инструкция *SELECT*, заключенная в круглые скобки, с предикатом *EXISTS* (с необязательным зарезервированным словом *NOT*) в логическом выражении для определения, должен ли подчиненный запрос возвращать какие-либо записи.

Пример:

```
SELECT* FROM Студент WHERE EXISTS  
(SELECT * FROM Оценка WHERE Сотрудник.[№ зач.книжки]=  
Оценка.[№ зач.книжки])
```

Отбираются студенты, которые имеют хотя бы одну оценку.

Список использованных источников

- 1 Агальцов, В.П. Базы данных. – М.: Мир, 2002. – 376 с., ил.
- 2 Попов, А.А. Fox Pro 2.5/2.6, 1997г.
- 3 Омельченко, Л.Н. Самоучитель Visual Fox Pro 6.0 – СПб.: БХВ – Санкт-Петербург, 1999. – 512 с., ил.
- 4 Мусина, Т.В. Visual Fox Pro 7.0. Учебный курс. / Т.В.Мусина, В.А.Пушенко – К.:ВЕК+; СПб.: КОРОНА принт; К.: НТИ; М.: Бином-Пресс, 2004. – 400 с.
- 5 Фуфаев, Э.В. Базы данных, 2005г.
- 6 Полякова, Л.Н. Основы SQL: Курс лекций [Текст]: учеб.пособие /Л.Н.Михеева – М.: ИНТУИТ.РУ, Интернет-Университет Информационных Технологий, 2004. – 368 с.
- 7 Макарова, Н.В. Информатика: Учебник. /Под ред. Н.В.Макаровой. – 3-е перераб. изд. – М.: Финансы и статистика, 2006. – 768 с.: ил.