Подходы к работе с базами данных

Д.А.Мацкявичюс, доцент по кафедре ВТ РХТУ

1998 г.

Содержание

Методическая разработка к циклу занятий «Подходы к работе с базами данных»

Цель занятий. Изучить назначение, построение, основы управления и использования баз данных на примере xBase-совместимых систем управления базами данных (СУБД). Освоить основные навыки программирования.

Используемое оборудование и материалы:

1. Компьютеры от PC AT/286.
2. Система управления базами данных семейства xBase (dBase, FoxBase, Foxpro).

Общая продолжительность: 24 академических часа (8 занятий по 3 часа).

На первом занятии (3 часа), совместно со студентами обсуждается важность и возможные области применения СУБД, основные используемые понятия, виды информации и соответствующие им типы полей в БД, выбирается тема для разработки БД на последующих занятиях. Для этой базы разрабатывается и обсуждается структура, вводится понятие справочников. В конце занятия раздается методическое пособия и приводится рекомендуемая литература.

Второе занятие (3 часа), проводится с использованием компьютера. Обсуждаются вопросы запуска и корректного выхода из СУБД, команды создания, модификации структуры БД и ввод информации. Создаются файлы БД, по разработанным на предыдущем занятии структурам и производится их заполнение. В конце занятия студентам предлагается выбрать индивидуальные темы для зачета, по которым они должны разработать структуру реляционной БД с 2—3 справочниками и виды запросов к ней с ориентировочными выходными формами.

На последующих 3 занятиях (9 часов), рассматриваются основные команды и функции (в т.ч. определенные пользователем), их синтаксис и применение, способы обращений с запросами к БД и варианты их вывода на внешние устройства и/или в файлы. На протяжении этих, а также последующих занятий студенты разрабатывают интерфейс для работы с БД, осваивают подходы к использованию генераторов форм и получают индивидуальные консультации по зачетному заданию.

На последнем занятии (3 часа) проводится зачет. Он состоит из обсуждения сферы применения и анализа разработанной структуры БД, правильность индексации по отдельным полям, соответствия индексов предполагаемым запросам. В случае необходимости возможен выборочный опрос по контрольным вопросам. Итоговая оценка выставляется на основании результатов текущего контроля и зачетной оценки.

Контроль знаний с выставлением оценок проводится в начале всех занятий по контрольным вопросам темы предыдущего занятия.

Преемственность с другими занятиями. Изучение данной темы желательно после освоения основ работы с персональным компьютером (MS-DOS, Norton Comander, утилиты). Подготовка к занятиям по электронным таблицам, изучению Windows (освоение клавиш управления курсором, способов работы с мышью, программирование макрокоманд).

Методическое пособие предназначено для самостоятельной подготовки студентов к занятиям по данной теме. Рассмотрены базисные понятия, концептуальные положения, а также ряд общих команд и функций, для которых приводится основной (сокращенный) синтаксис и наиболее важные случаи использования.

Данное пособие отображает лишь основы построения и управления СУБД, а также командного языка. Поэтому большая часть изложения максимально упрощена, синтаксис команд и функций дан в значительно усеченном виде, а сравнительно подробно рас-

сматриваются только наиболее важные из них. Большая часть информации ориентирована на СУБД FoxPro 2.x для DOS, но базисные элементы конкретного языка и идеология построения программ могут использоваться как в других совместимых СУБД, так и в качестве основ программирования на языках четвертого и пятого поколений.

Принятые обозначения и сокращения

БД	— база данных;
СУБД	— система управления базами данных
UDF	— функция, определенная пользователем (User Defined Function);
SQL	— структурированный язык запросов (Structured Query Language)
[...]	— в квадратных скобках приводяться необязательные элементы;
<...>	— в угловых скобках указывается условное обозначение элемента, который должен быть подставлен в команде. Сами скобки в команду не входят;
...|...	— указывает на необходимость наличия только одного из предлагаемых элементов;
col	— колонка на экране (column);
row	— строка на экране ;
Выр	— выражение любого типа;
ВырN	— выражение числового типа (Numeric);
ВырC	— выражение символьного типа (Character);
ВырD	— выражение типа дата (Date);
ВырL	— выражение логического типа (Logical). Имеет значения «Истина» (True) или «Ложь» (False), обозначается соответственно как .T. и .F.;
Массив	— имя массива;
Окно	— имя окна;
Поле	— имя поля базы данных;
Перем	— имя переменной, содержащейся в памяти;
Файл	— имя файла;
	—

Символом (■) помечены команды.

ВВЕДЕНИЕ

Данное пособие представляет собой краткое руководство, позволяющее упростить изучение основных аспектов разработки, создания и использования баз данных. Краткость изложения материала позволяет уменьшить время, необходимое для освоения темы, но дает лишь общее представление о конкретных механизмах работы и их использованию для разбираемой СУБД. Более подробная базовая информация может быть получена из приведенной в конце литературы.

Дополнительно можно отметить, что современные СУБД имеют мощнейшие возможности в области создания интерфейса на базе объектного визуального программирования, и форматирования результатов обработки отдельных полей и их совокупности.

Назначение баз данных, основные понятия, используемые в системах управления базами данных

Изначально, использование персональных компьютеров (ПК) было в основном связано с обработкой текстов, применением электронных таблиц и ведением баз данных. Характеризуя ситуацию в общем, можно сказать, что ПК применяется для накопления, хранения и обработки информации, а также автоматизации отдельных видов труда, откуда вытекает огромная значимость баз данных (БД), которые позволяют реализовать большую часть этих задач. Более того, специализация на быстром поиске информации позволяет реализовать большинство этих функций наиболее эффективно.

Самостоятельного значения БД, как правило, не имеют в связи с необходимостью проведения над ними вышеназванных операций. В связи с этим, обычно говорят о **системах управления базами данных** (СУБД). СУБД состоит из нескольких компонентов:

— среда пользователя (для работы с данными при помощи команд, вводимых с клавиатуры или через меню);

— алгоритмический язык программирования (для создания программ, работающих в режиме интерпретации);

— компилятор для создания независимых EXE-файлов;

— утилит для быстрого программирования рутинных операций.

Для разбора понятий, используемых в СУБД, целесообразно сначала рассмотреть с формальной точки зрения информацию, предназначенную для ввода в базу. Как правило, это перечень каких-либо явлений или предметов описанием их свойств. По сути — это картотека, где каждому явлению (предмету) соответствует карточка. По своему происхождению БД и есть электронная картотека, со всеми вытекающими отсюда преимуществами. Ее карточки имеют специфическое название — *записи*. Каждому явлению соответствует ряд общих свойств или характеристик, соответствующих содержанию бумажной карточки. Каждое свойство (характеристика) в БД называется *полем* и имеет имя тип и размер. Полный перечень полей с их типами и размерами составляет *структуру БД*.

Базы данных (БД) — файлы, содержащие информацию, разделенную на показатели. Базы данных используются для поиска, систематизации (сортировки и выделения) информации, проведения поиска по определенным критериям.

Современная коцепция предусматривает несколько иную трактовку понятий. Так, единичный файл, содержащий информацию, называется таблицей. Несколько таблиц, объединяются в информационную систему, которая и называется базой данных. В ней описываются все связи между таблицами, а также комплекс параметров, определяющих условия сохранности информации при добавлении, удалении и редактировании (условия целостности БД).

Структура БД — перечень полей, имеющихся в базе данных с их характеристиками.

Поле — минимальная единица структурированной информации в БД, характеризующая один показатель для одного явления. Каждое поле имеет тип и размер. Все поля для данного явления составляют запись. Поле — структурная единица записи. Обратите внимание, что с точки зрения всей БД, поле является совокупностью какого-либо значения во всех записях.

Типы полей зависят от представления информации, которую предполагается в них хранить, а также возможных способов обработки. Наиболее распространенные типы полей, используемые в различных СУБД, приведены в таблице 1. Для краткости записи тип поля или данных часто обозначается первой буквой английского названия (см. список сокращений).

Таблица 1. Характеристики полей БД

Тип поля		Ограничения (размер)	Содержание (виды информации)
Символьный	(Character)	254 символа	Любые символы
Числовой	(Numeric)	20 разрядов	Числовые значения. Размер поля состоит из суммы числа разрядов до десятичной точки, после нее +1 знак на точку
	(Float)		Числа с плавающей точкой
Дата	(Data)	8 символов	Дата
Время	(Time)	6 символов	Время
Логический	(Logical)	1 символ	Логические данные — истина (.T. или .Y.) или ложь (.T. или .N.)
Примечаний	(Memo)	нет*	Любые данные в т.ч. в двоичном виде
Общего назначения	(General)	нет*	Как правило графические или или мультимедийные (звук, видео) данные

Запись — совокупность всех полей, описывающих данное явление. Каждая запись имеет одинаковую структуру и размеры, а также свой порядковый номер, назначаемый

при ее создании. Движение по БД фактически является переходом от одной записи к другой. Запись — структурная единица базы данных.

С практической точки зрения базу данных можно представить в виде таблицы (рис. 1), где колонки являются полями, а строки — записями.

Поле 1	Поле 2	....	Поле *i*	Запись 1
				Запись 1
				.....
				Запись *j*

Рис. 1. Схематичное представление базы данных.

Создание и открытие БД

Создание базы данных осуществляется командой

- CREATE <имя файла>,

после выполнения которой СУБД выводит форму для задания структуры базы. При этом вводятся имена полей (до 10 символов), их тип и размер. Вновь созданная база данных открывается автоматически.

Для открытия базы данных используется команда

- USE <имя файла> [ORDER <имя индекса>]
  [IN <имя областиC|N>] [ALIAS <псевдонимC>].

Каждая база даных открывается в своей *области* (alias) количество которых зависит от вида и версии СУБД. Только одна из областей может быть текущей в данный момент времени. Обращение к определенной области производитса по букве, соответствующей ее номеру или по присвоенному при открытии псевдониму. В этом случае форма ссылки состоит из псевдонима и имени поля, разделенных точкой. Например, открытие командой «USE USERS in 4 ALIAS User», позволит обращаться к полю Name текущей записи базы Users (вне зависимости от того, какая область является текущей) следующими способами:

cCurName = User.Name или cCurName = d.Name.

В обоих случаях переменой памяти «cCurName» будет присвоено значение поля Name текущей записи базы Users.

Способы отображения на экране информации из базы данных

Информация на экран с позможностью ее просмотра (листания) может быть выведена тремя способами:

1. в режиме Browse, когда данные представлены в виде таблицы, как они храняться в БД с именами полей в качестве заголовков колонок;
2. в режиме Change, с выводом в виде карточки (списка), где каждое поле приводится на отдельной строке, а новая запись начинаетя после вывода всех полей предыдущей;
3. окно, описанное программой пользователя.

Первый способ обслуживается одной из наиболее мощных одноименных (Browse) команд и позволяет вывести на экран достаточно большое количество информации (однако ограниченной шириной экрана) и предоставляет возможность быстрого перемещения по БД. Второй — удобен для ввода или исправления информации в случае отсутсвия специально написанной программы. Последний способ позволяет наиболее удобно разместить на экране все данные, относящиеся к текущей записи, сгруппировать их различными способами по логическим блокам (расположением, рамками, цветом), снабдить необходимыми подписями и определить желаемые способы реагирования СУБД на стандартные действия пользователя.

Перемещения в базе данных

Положение в БД описывается с помощью понятия *указателя записи*, который находится либо на определенной записи, либо за пределами базы в случае неудачного поиска или принудительного перехода командой. Все перемещения рассматриваются как изменение положения указателя записи. Для этого существует несколько команд:

- GO TOP — переход к первой записи;
- GO BOTTOM — переход к последней записи;
- GO <выpN> — переход к записи с номером <выpN>;
- SKIP [<выpN>] — переход к записи, отстоящей от текущей на <выpN> (которое может быть отрицательным для движения назад) число записей (по умолчанию <выpN> = 1).

К перемещению по базе приводят также выполнение команд поиска, расчета и вообще любой обработки всей базы, а также листание клавишами управления курсором в режиме Browse и Change.

Контроль положения указателя записи, а также наличия записей в базе может осуществляться функциями:

- RECNO() — возвращает номер текущей записи;
- RECCOUNT() — возвращает число записей в БД;
- EOF() — возвращает истину (.T.) если указатель достиг конца файла;
- BOF() — возвращает ложь (.F.) если указатель достиг начала файла.

Способы поиска информации

1. Ручной, путем просмотра БД;
2. Установка фильтра;
3. Вывод в окно Browse с директивой FOR <условие>;
4. Индексация БД с директивой FOR <условие> (фильтрация с упорядочиванием);
5. Последовательный поиск командой LOCATE...CONTINUE;
6. Индексный поиск командой SEEK;
7. SQL запрос.

Все перечисленный способы реально используются при работе с БД и имеют определенные преимущества. Для конкретной ситуации подбирается наиболее эффективный и соответствующий по своим возможностям способ. Чаще всего такого результата позво-

ляет достичь комбинация нескольких подходов, например, индексация или установка фильтра с последующим ручным просмотром.

Принципиальным моментом при выборе того или иного механизма является то, куда и для чего осуществляется вывод результатов поиска (см. ниже).

Вывод результатов обработки БД

Принципиально, вывод может быть осуществлен тремя способами:

1. На экран
   а) в виде неструктурированного или частично структурированного листинга содержимого БД;
   б) в окне Browse;
   в) в пользовательское окно для последующего ручного просмотра;
   г) в виде отчета, предназначенного для просмотра;
   д) в другую БД для последующей работы с этой информацией
2. На принтер
   а) в виде неструктурированного или частично структурированного листинга содержимого БД;
   б) в виде отчета
3. В текстовый файл
   Данный способ наиболее рационален, так как позволяет использовать результаты многократно, передавать их другим лицам, распечатывать и, что наиболее важно, подвергать редактированию и обработке в текстовом редакторе.

Индексация БД

Индексация баз данных является основным средством для организации связей между отдельными таблицами (установления реляционных отношений), представления данных в упорядоенном виде и осуществления поиска информации с использованием ускоренных механизмов.

Реляционность в СУБД

Реляционность для грамотного построения баз данных в персональных СУБД играет ключевую роль и организуется посредством связывания отдельных таблиц с помощию встроенных в СУБД механизмов.

Нормализация данных. Справочники

Дублирующиеся данные как правило не должны содержаться в поле отдельных записей. Это обусловливается тем, что, во-первых, практически невозможен безошибочный ввод данных, а это в свою очередь приведет к неоднозначности поиска информации в данном поле. В случае изменения такого показателя, его придется корректировать во многих записях БД. Во-вторых, многие символьные поля с повторяющейся для отдельных записей информацией содержат данные значительного объема. Это дает пропорциональное разрастание объема файла БД, что снижает скорость всех операций и приводит к нерациональному расходованию дискового пространства. В качестве примера можно привести практически все имена собственные (названия периодических изданий, географические названия, перечни фирм, товаров или предметов и т.д.).

Процесс уменьшения избыточности информации в БД называется нормализацией. Ее использование в полной мере позволяет устранить все перечисленные проблемы и ускорить поиск за счет наиболее рационального использования индексации и связей между базами данных.

В упрощенном виде процессы нормализации можно описать в виде использования справочников. Так, в БД отдела кадров для обозначения подразделения в котором работает сотрудник, нет необходимости создавать поле с его названием. Достаточно создать поле с кодом (2—3 знака) и вторую базу (справочник), состоящую из двух полей: названия подразделения и его уникального кода. В основную базу (карточку сотрудника) заносится код соответствующего подразделения. Эти процедуры обычно описываются в виде программ, чтобы избежать ошибки ввода. В дальнейшем, использование этого справочника для других целей (расчет заработной платы, планирование научной работы, учет оборудования и т.п.) название подразделения будет только в одном месте.

В качестве кодовых полей более рационально использовать числа, так как в этом случае существенно проще контролировать наращение уникальных значений (добавление единицы), что, однако, может затруднить работу в особо сложных ситуациях, т.к. обработка символьных значений предполагает более гибкие возможности.

Подходы к разработке программ в СУБД

В качестве основных требований к программе можно предьявить следующее:
— компактность;
— высокая скорость выполнения;
— модульность (реализация типовых действий в виде многократно используемых модулей);
— реализация всех возможностей языка (что следует понимать как решение конкретных ситуаций наиболее эффективными способами).

Алгоритм разработки программы в СУБД

Аналитическая блок схема работы программы
Ориентировочная разработка структур БД
Выходные формы
Разработка меню

Структура команд и функций СУБД

Командный язык СУБД весьма разнообразен, но наибольшую значимость имеют команды, предназначенные для обработки базы данных, имеющие в общем виде следующий синтаксис:

■ КОМАНДА [<границы>] [<выражения специфичные для команды>] ;
[FOR<условие>] [WHILE<условие1>]

Здесь:

<границы>] — диапазон обработки записей при действии команды:
ALL — все записи БД;

NEXT <N> — следующие N записей;
REST — от текущей записи до конца файла;
RECORD <N> — запись с номером N.

FOR<условие> — ограничение действия команды только записями, отвечающими <условию>;

WHILE<условие1> — выполнение команды происходит до тех пор, пока выполняется <условие1>.

По умолчанию обрабатываются все записи (чаще) или только текущая (редко). Условие WHILE имеет смысл только в упорядоченной БД.

Для переноса командной строки на следующую строку, в конце строки ставится точка с запятой («;»).

Функции как правило состоят из имени и стоящих после него без пробела круглых скобок, в которых при необходимости могут перечисляться аргументы:

Функция	Возвращаемый результат
EOF()	.T. или .F.
STR(12)	строка "12"
SUBSTR("Окисляемость", 2, 4)	подстрока "кисл"

Знаки операций

Математические

«**|^» — возведение в степень;
«*» — умножение; «/» — деление; «%» — остаток от деления;
«+» — сложение; «-» — вычитание.

Логические

«NOT|!» — НЕ;
«AND» — И;
«OR» — ИЛИ.

Отношения

«<» — меньше; «>» — больше; «=» — равно;
«#» — не равно; «<=» — не больше; «>=» — не меньше.

Знаки отношения могут применяться к данным любого типа, но при этом сравниваемые пары должны быть одного типа. Сравнение символьных выражений проводится по длине строки, стоящей справа от знака отношения. (Следовательно, соотношение 'Иванов'='Иван' является истинным (.T.)) Отсюда вытекает возможность поиска строки по неполному соответствию.

<ВырС1>$<выр2> — специальная операция сравнения символьных выражений, возвращающая .T., если <выр1> входит в <выр2> и .F. в противном случае.

«==» — операция сравнения символьных строк на полное тождество по длине и содержанию.

Сцепления (конкатенации) для символьных строк

«+» — соединение в одну строку;
«-» — то же, но концевые пробелы, предшествующие этому знаку перемещаются в конец итоговой строки.

Для явного определения приоритетов выполнения операторов могут применяться круглые скобки.

Генераторы, используемые в СУБД

Для ускорения разработки отдельных элементов в пакеты СУБД включаются дополнительные модули (программы-утилиты или мастера (Wizards)), автоматически генерирующие программный код. Обычно это относится к экранным формам, отчетам, этикеткам и запросам к БД.

Отличие СУБД от программ калькуляции электронных таблиц

У СУБД и программ калькуляции электронных таблиц существует много общих или схожих возможностей, а в последнее время наблюдается четко выраженная тенденция к сближению функциональных возможностей этих двух групп программ. Однако существует и ряд существенных отличий:

— наличие в БД механизмов ускоренного доступа к данным;
— практически неограниченное число записей;
— наличие возможности устанавливать сложные связи между таблицами (реляционность);
— возможность создания программ для обслуживания БД;
— отсутствие встроенных средств графической обработки.

В целом, можно сказать, что для обработки небольшого объема статичных данных, в случае использования сложных математических методов или важности графического анализа, более рационально использование электронных таблиц. Наоборот, при значительном объеме часто меняющихся данных, необходимости извлечения разнообразной информации и сравнительно постоянных подходах к математической обработке, лучшим средством будут БД. Кроме того, часто решение задач с использованием сложных стандартных статистических методов, существенно разумнее проводить с помощью специализированных программ. Для выборки данных и передачи данных в эти программы СУБД часто оказываются значительно удобнее.

Функциии, определенные пользователем

Хотя FoxPro и включает около 200 встроенных функций, иногда возникают ситуации, когда необходимо определить специализированные функции для конкретных

приложений, такие функции называются функциями, определенными пользователем (User-Defined Functions, UDFs).

UDF — это часть языка FoxPro, которая может включаться во многие команды FoxPro, разрешенные в данной версии. Например, в команду BROWSE, открывающую окно Browse, в котором можно просматривать и изменять данные. Можно включить UDF в BROWSE WALID, для проверки правильности ввода данных.

Что такое UDF?

UDF — Это программы FoxPro, которые возвращают значение в вызывающую программу (программа в которой стартуется UDF называется вызывающей). Значение может быть возвращено в вызывающую программу при помощи команды RETURN <Выр>. UDF может быть выполнена в виде отдельной программы, либо в виде процедуры или функции в программе.

Конструирование UDF

Если UDF является функцией или процедурой, первая ее строка должна содержать команду FUNCTION или PROCEDURE, которая присваивает ей имя. Затем следует любое количество операторов. Смотри описание команд FUNCTION и PROCEDURE, для получения дополнительной информации по созданию функций и процедур в FoxPro.

UDF нельзя присваивать тоже имя, что и встреной функции, так как эти имена являются зарезервированными и имеют преимущество, в такой ситуации будет выполняться встроенная функция FoxPro.

Возвращаемое значение

Значение может быть возвращено в вызывающую программу при помощи команды RETURN <Выр>. Эта команда является последней командой UDF.

Следите за типом значения, которое возвращается из UDF в вызывающую программу. Например, если вызывающая программа ожидает, что функция вернет числовое значение, а та в свою очередь возвращает символьное выражение, будет генерироваться сообщение об ошибке.

Если UDF не возвращает значение (команда RETURN не включена в UDF или RETURN не содержит значения), в вызывающую программу автоматически возвращается логическое значение "истина".

Передача параметров

Данные передаются UDF в форме параметров. Команда PARAMETERS идентифицирует данные, передаваемые UDF и назначает локальные имена для данных. Содержимое элементов массива можно передавать в UDF.

Когда параметры передаются UDF, и функция становится активной, первая выполняющаяся команда в UDF должна быть оператором PARAMETERS.

Если UDF процедура или функция, оператор PARAMETERS должен быть первой выполнимой командой, следующей непосредственно за обоявлением FUNCTION или PROCEDURE.

Переменное число параметров

Число параметров передаваемых UDF может быть меньше числа параметров, перечисленных в списке оператора PARAMETERS. В этом случае оставшиеся параметры инициализируются логическим значением "ложь". Переменное число параметров может передаваться UDF, число переданных параметров можно определить при помощи функции PARAMETERS().

Если число параметров передаваемых UDF больше числа параметров, перечисленных в списке оператора PARAMETERS, генерируется сообщение об ошибке "Неверное число параметров"(Wrong number of parameters).

Когда параметры (временные переменные или массивы элементов) передаются UDF, они могут передаваться по значению или по ссылке.

По умолчанию, переменные передаются пользовательской функции по значению. Когда переменная передается по значению, значение переменной может быть изменено процедурой или пользовательской функцией, но оригинальное значение в вызывающей программе при этом не изменится.

Когда переменная передается по ссылке и пользовательская функция изменяет ее значение, оригинальное значение этой переменной в вызывающей программе также изменяется. Для передачи параметров UDF по ссылке установите перед вызовом функции SET UDFPARMS TO REFERENCE.

См. также: PARAMETERS().

Вызов UDF

Подобно всем функциям FoxPro, UDF описываются своими именами и набором параметром в скобках, следующем за именем. В скобках заключены (необязательные) параметры, передаваемые UDF. Список команд, в которые можно включать UDF приводится ниже.

UDF в логических выражениях

Если используется одна или более UDF в логических выражениях, то они оцениваются слева направо.

Когда два операнда (возможно UDF) сцеплены оператором отношения AND, оценка выражения заканчивается, когда операнд принимает значение "ложь". Например, две UDF ниже, LOGICAL1() и LOGICAL2(), возвращают логические выражения связанные оператором AND:

LOGICAL1() AND LOGICAL2()

Если LOGICAL1() возвращает логическое выражение "истина", оценивается LOGICAL2(). Однако, если LOGICAL1() возвращает логическое выражение "ложь", LOGICAL2() уже не оценивается.

UDF в командах

Многие FoxPro команды допускают UDF в своих опциях. UDF позволяют выполнять процедуры, которые могут проверять правильность данных, высвечивать сообщения, разрешать/запрещать управление и так далее.

Где могут использоваться пользовательские функции (UDF)

Пользовательские функции могут использоваться только в некоторых опциях некоторых команд. Следующая таблица представляет частичный список наиболее важных команд (с опциями), где можно использовать UDF.

Команда Опции, поддерживающие UDF
— @ ... GET
— AVERAGE Список выражений <Выр list>
— BROWSE FIELDS <field list> - вычисляемые поля
— DISPLAY FIELDS Вычисляемые поля
— DO WITH <parm list>
— DO CASE CASE <ВыpL>,<ВыpL1>
— DO WHILE WHILE <ВыpL>
— FOR Начальное и конечное значение счетчика
— INDEX Индексное выражение <Выр>
— IF Условие "если" <ВыpL>
— REPLACE WITH <Выр1>,<Выр2>,...
— Отчет (Report) Выражения для полей
Переменные отчета
Выражения для групп
— STORE Сохраняемое выражение <Выр>
— SUM Список выражений <Выр list>

Установки цвета элементов интерфейса

Система FoxPro предлагает развитый набор команд для полного управления цветами. Вы можете определять цвета, используемые как в системном интерфейсе FoxPro (системные меню, окна, предупреждения, диалоги ит.д.), так и в обьектах, определенных пользователем (@...SAY/GET, меню, окна, и т.д.). Определенные команды, как например DEFINE WINDOW, поддерживают необязательное ключевые слова COLOR или COLOR SHEME, которые позволяют вам напрямую специфицировать цвета обьектов, определенных пользователем.

Вместе с тем, использование названных возможностей не играет ключевой роли в использовании БД, в связи с чем полностью опущена соответствующая информация в описании отдельных команд.

В общем, практически для любого отдельного объекта, выводимого на экран может быть определен список цветовых пар (Color Pair List), полностью описывающий правила цветовых комбинаций текст/фон для стандартных состояний объекта. Кроме того, могут использоваться заранее определенные цветовые схемы, и цветовые множества. Последние переопределяют цветовые схемы для всех объектов и могут переключаться одной командой.

Цветовая пара состоит из комбинации основного и фонового цвета. Отдельные цвета могут быть специфицированы с помощью символьных аббревиатур. Символ звездочка (*) используется для обозначения мигания (SET BLINK ON) или повышенной яркости (SET BLINK OFF), а символ плюс (+) используется для обозначения повышенной интерсивности. Далее приведена таблица доступных в FoxPro цветов и даны их коды:

ЦВЕТ	КОД
Black (черный)	N

Blank (пустой)	X
Blue (голубой)	B
Brown (коричневый)	GR
Cyan (светло-голубой)	BG
Green (зеленый)	G
Magenta (пурпурный)	RB
Red (красный)	R
White (белый)	W
Yellow (желтый)	GR+

С помощью команды SET COLOR TO вы можете варьировать экранные атрибуты при работе как с цветными, так и с монохромными мониторами.

Для монохромных мониторов символ U является атрибутным кодом подчеркивания, а символ I является атрибутным кодом инверсного видео-изображения.

Список рекомендуемой литературы

1. Каррабис Дж.-Д., *Программирование в dBase III Plus*. Пер. с англ. — М.: Финансы и статистика, 1991. — 240 с.
2. Бремер С., *FoxPro 2.5 для Windows*. Пер. с нем. — К.: Торгово-издательское бюро BHV, 1994. — 416 с.
3. *FoxPro 2.0 Applications Programming*. Пер. с англ. — М.: Изд-во Эдэль, 1994. — 384 с.
4. Попов А.А., *Программирование в среде СУБД FoxPro 2.0. Построение систем обработки данных*. — М.: Радио и связь, 1993. — 352 с.
5. Берещанский Д.Г., *Практическое программирование на dBASE*. — М.: Финансы и статистика, 1989. — 192 с.
6. *FoxPro-2.0 (2.5, 2.6). Interface Guide*. — Microsoft Corp., 1992 (1993).
7. *FoxPro-2.0 (2.5, 2.6). Commands & Functions*. — Microsoft Corp., 1992 (1993).
8. Крамм Р., *Системы управления базами данных dBASEII и dBASEIII для персональных компьютеров*. — М.: Финансы и статистика, 1988. — 283 с.
9. Гринберг Ф., Гринберг Р., *Самоучитель программирования на входном языке СУБД dBASEIII*. — М.: Мир, 1989. — 453 с.
10. Горев А., Макашарипов С., *Microsoft Visual FoxPro 3.0. Новые возможности для программиста*. — СПб.: Питер, 1995. — 336 с.
11. Каратыгин С., Тихонов А., Тихонова Л., *Работа в Visual FoxPro на примерах*. — М.: БИНОМ, 1996. —512 с.

Контрольные вопросы по теме «Работа с базами данных»

1. Понятие «Базы данных» (БД).
2. Понятие «Система управления базами данных» (СУБД).
3. Что такое поле в БД.
4. Что такое запись в БД.
5. Что такое структура таблицы БД и структура БД.
6. Существующие типы полей и их характеристика.
7. Существующие типы переменных памяти, особенности их использования.
8. Факторы определяющие структуру базы данных.
9. Стандартные расширения имен файлов в СУБД семейства xBASE.
10. Отличия команд и функций.
11. Понятие функций, определяемых пользователем (UDF).
12. Синтаксис и использование команды «@...SAY».
13. Синтаксис и использование команды «@...GET».
14. Синтаксис и использование команды «@...PROMPT».
15. Синтаксис и использование команд «?» «??», и «???».
16. Синтаксис и использование команд «» и «».
17. Синтаксис и использование команды «READ».
18. Синтаксис и использование структуры «IF — ELSE — ENDIF».
19. Синтаксис и использование функции «IIF()».
20. Синтаксис и использование структуры «DO CASE — ENDCASE».
21. Структуры, позволяющие выбрать альтернативное решение при выполнении программы.
22. Синтаксис и использование функций «LASTKEY()», «READKEY()» и «INKEY()».
23. Особенности значений, возвращаемых функцией «READKEY()».
24. Синтаксис и использование цикла «DO WHILE — ENDDO».
25. Задание «бесконечного» цикла и его назначение.
26. Индексация БД по отдельным полям и по нескольким полям совместно.
27. Задание цикла со счетчиком и его назначение.
28. Синтаксис и использование команды «WAIT».
29. Операторы задания диапазона для обработки БД.
30. Использование операторов «FOR...» и «WHILE...».
31. Различие операторов «FOR...» и «WHILE...».
32. Скорость обработки БД при использовании операторов «FOR...» и «WHILE...».
33. Содержание понятия реляционности.
34. Назначение индексов.
35. Подключение индексов к БД.
36. Переключение индексов БД.
37. Понятие фильтра базы данных.
38. Способы отбора данных для просмотра и обработки.
39. Использование индексов с фильтром.
40. Способы дополнения/удаления записей БД и их особенности.
41. Способы выборки данных из БД.
42. Способы отображения информации из БД.

43. Отличие системного и программного (пользовательского) вывода/редактирования записей БД.
44. Понятие событийно-ориентированного программирования.
45. Особенности индексации полей типа Date.
46. Синтаксис и использование команд «LOCATE» и «CONTINUE» при поиске информации.
47. Синтаксис и использование команд «SEEK» и «FIND».
48. Команды «SEEK» и «FIND». Их особенности и преимущества.
49. Синтаксис и использование функции «FOUND()».
50. События, происходящие при успешном и неуспешнм поиске.
51. Синтаксис и использование команд «ON KEY» и «ON KEY LABEL».
52. Открытие и закрытие баз данных.
53. Способы поиска в БД и их особенности.
54. Печать выборки из базы данных (отчета) в файл и на принтер.
55. Возможности вывода результатов обработки БД.
56. Импорт и экспорт данных в БД.
57. Синтаксис и использование функций «VAL()» и «STR()».
58. Синтаксис и использование функции «SUBSTR()».
59. Генерация отчетов.
60. Синтаксис и использование функции «DATE()».

Ориентировочный перечень тем для разработки системы БД на занятии

Структура учебных БД должна включать поля всех основных типов и строится на реляционных отношениях со справочниками (не более двух-трех).

1. Кадровые сведения о студентах, сотрудниках.
2. Географический справочник (по странам или городам).
3. Картотека библиографического описания документов.
4. Система учета материальных ценностей.
5. Телефонный справочник.
6. Адресный справочник.
7. Система для регистрации результатов научных исследований.
8. Справочник физико-химических свойств соединений.
9. Справочник компьютерных программ.

Приложения

Тематический список основных команд и функций

Как правило, все функции производят исследование информации, а в ряде случаев и ее изменение и возвращают результат своей работы.

Функции могут иметь или не иметь аргументов, передаваемых им для обработки, а также иметь необязательные аргументы. Передача неверных по типу или значению аргументов порождает ошибку с прерыванием работы программы, но в ряде случаев возможно возникновение логических ошибок при использовании бессмысленных (выделение строки нулевой длины) или расчитанных в ходе работы программы значений.

Функции, определяющие выполнение какого-либо условия, возвращают .T., если оно выполняются и .F., если нет.

Обработка символьных данных

- <ВырС> $ <ВырС1> — возвращает "истинно", если <ВырС> содержится в <ВырС1>.

ALLTRIM(<ВырС>) — возвращает <ВырС> с удаленными ведущими и завершающими пробелами.

ASC(<ВырС>) — возвращает код ASCII, эквивалентный первому символу в <ВырС>.

AT(<ВырС>, <ВырС1> [, <ВырN>]) — возвращает начальную позицию <ВырС> в <ВырС1>. В случае неудачного поиска возвращается 0. Поиск производится слева направо с учетом регистра (заглавные и строчные буквы различаются). Для проведения поиска без учета регистра используется функция ATC(). Использование выражения <ВырN>, позволяет определить позицию N-ного появления <ВырС> в <ВырС1>.

ISALPHA(<ВырС>) — определяет, является ли первый символ <ВырС> буквой.

ISDIGIT(<ВырС>) — определяет, является ли первый символ <ВырС> цифрой.

LEFT(<ВырС>, <ВырN>) — возвращает <ВырN> число символов слева из символьного выражения <ВырС>.

LEN(<ВырС>) — возвращает длину выражения <ВырС>.

LIKE(<ВырС>, <ВырС1>) — определяет, содержится ли в <ВырС1> фрагмент, аналогичный шаблону <ВырС>, в который могут включаться метасимволы «*» и «?» в любом месте и в любом количестве. Знак вопроса заменяется одним символом, а звездочка заменяется любым числом символов. Функция LIKE() чувствительна к регистру букв.

LOWER(<ВырС>) — переводит все символы <ВырС> в нижний регистр.

LTRIM(<ВырС>) — возвращает <ВырС> с удаленными ведущими пробелами.

OCCURS(<ВырС>, <ВырС1>) — возвращает число, соответствующее тому, сколько раз <ВырС> встретилось в <ВырС1>.

PROPER(<ВырС>) — возвращает <ВырС> в формате, когда первый символ каждого стова является заглавной буквой, а остальные — строчными.

RAT(<ВырС>, <ВырС1> [, <ВырС2>]) — тоже, что и AT(), но поиск производится справа налево.

REPLICATE(<ВырС>, <ВырN>) — возвращает символьную строку, получившуюся повторением <ВырС> заданное число <ВырN> раз.

RIGHT(<ВырС>, <ВырN>) — возвращает <ВырN> число символов слева из символьного выражения <ВырС>.

RTRIM(<ВырС>) — возвращает <ВырС> с удаленными завершающими пробелами.

SPACE(<ВырN>) — возвращает строку, состоящую из <ВырN> пробелов.

STRTRAN(<ВырС>, <ВырС1> [, <ВырС2>] [, <ВырN1>] [, <ВырN2>]) — заменяет в <ВырС> вхождения <ВырС1> на <ВырС2>. Если <ВырС2> не указано, вхождения <ВырС1> удаляются из <ВырС>. Параметры <ВырN1> и <ВырN2> определяют соответственно с какого вхождения и сколько замен должно производиться.

Поиск является зависимым от верхнего и нижнего регистров.

STUFF(<ВырС>, <ВырN1>, <ВырN2>, <ВырС1>) — заменяет в <ВырС> с позиции <ВырN1> <ВырN2> символов на <ВырС1>.

SUBSTR(<ВырС>, <ВырN1> [, <ВырN2>]) — возвращает из <ВырС> символы с позиции <ВырN1>. Параметр <ВырN1> может определять количество извлекаемых символов (без него извлекаются символы до конца).

TRIM(<ВырС>) — тоже, что и RTRIM().

UPPER(<ВырС>) — переводит все символы <ВырС> в верхний регистр.

VAL(<ВырС>) — преобразует <ВырС> в число.

Обработка числовых данных

ABS(<ВырN>) — возвращает абсолютное значение <ВырN>.

ASIN(<ВырN>) — возвращает арксинус <ВырN>.

ATAN(<ВырN>) — возвращает арктангенс <ВырN>.

ATN2(<ВырN>, <ВырN1>) — возвращает арктангенс отношения <ВырN>/<ВырN1>.

CEILING(<ВырN>) — возвращает ближайшее целое больше или равное <ВырN>.

COS(<ВырN>) — возвращает косинус <ВырN>.

DTOR(<ВырN>), RTOD(<ВырN>) — переводят значение угла <ВырN> из градусов в радианы и наоборот.

EXP(<ВырN>) — возвращает экспоненту <ВырN>.

FV(<ВырN>, <ВырN1>, <ВырN2>) — финансовая функция, возвращающая будущее значение капиталовложения, где <ВырN> — равный периодический платеж; <ВырN1> — доля периодической выгоды; <ВырN2> — число периодов произведенных платежей.

INT(<ВырN>) — возвращает целую часть <ВырN>.

LOG(<ВырN>) — возвращает натуральный логарифм <ВырN>.

LOG10(<ВырN>) — возвращает десятичный логарифм <ВырN>.

PAYMENT(<ВырN>, <ВырN1>, <ВырN2>) — финансовая функция, возвращающая величину каждой выплаты по заему, где <ВырN> — начальная величина заема; <ВырN1> — фиксированная периодическая процентная ставка; <ВырN2> — общее число платежей.

PI() — возвращает число Пи.

PV(<ВырN>, <ВырN1>, <ВырN2>) — финансовая функция, возвращающая настоящее значение вклада, где <ВырN> — размер периодического вклада; <ВырN1> — периодическая процентная ставка; <ВырN2> — общее число вкладов.

ROUND(<ВырN>, <ВырN1>) — возвращает округленное до <ВырN1> числа знаков после десятичной точки значение <ВырN>.

RAND([<ВырN>]) — возвращает случайное число в диапазоне от 0 до 1. <ВырN> позволят задать значение начального числа, используемого функцией. Если <ВырN> меньше 0, то значение функции вычисляется при помощи часов системы.

SIGN(<ВырN>) — возвращает 1, если <ВырN> положительное число, –1, если оно отрицательное и 0 для нуля.

SIN(<ВырN>) — возвращает синус <ВырN>.

STR(<ВырN> [, <ВырN1>] [, <ВырN2>]) — преобразует <ВырN> в символьную строку. Параметры <ВырN1> и <ВырN2> определяют длину строки и число разрядов соответственно. Если эти величины не указываются, принимаются значения по умолчанию (10 и 0).

SQRT(<ВырN>) — возвращает квадратный корень из <ВырN>.

TAN(<ВырN>) — возвращает тангенс <ВырN>.

Точность всех математических расчетов по умолчанию соответствует разрядности операндов. Выполнение команды SET FIXED ON (по умолчанию OFF) приводит к использованию значения десятичных разрядов определенных командой SET DECIMALS TO <ВырN> (по умолчанию — два знака после точки).

Обработка данных типа дата

CDOW(<ВырD>) — возвращает название дня недели <ВырD>.

CMONTH(<ВырD>) — возвращает название месяца <ВырD>.

CTOD(<ВырC>) — преобразует <ВырC> в формат даты.

DATE() — возвращает системную дату.

DMY(<ВырD>) — преобразует <ВырD> в символьный формат в виде "dd month yy".

DOW(<ВырD>) — возвращает числовое значение дня недели <ВырD>.

DTOC(<ВырD> [, 1]) — преобразует <ВырD> в символьное выражение. При использовании параметра «1» действует аналогично DTOS().

DTOS(<ВырD>) — возвращает восьмисимвольную строку в виде "yyyymmdd".

DAY(<ВырD>) — возвращает числовое значение дня недели <ВырD>.

MONTH(<ВырD>) — возвращает числовое значение месяца <ВырD>.

TIME() — возвращает системное время в виде строки символов в формате (HH:MM:SS).

YEAR(<ВырD>) — возвращает числовое значение года <ВырD>.

Над датами можно производить математические операции прибавления и вычитания дней. Результатом является соответствующая дата.

Универсальные функции

INLIST(<Выр>, <Выр1> [, <Выр2>...]) — определяет, содержится ли <Выр> в последовательности выражений <Выр1>, <Выр2> и т.д. Все выражения должны быть выражениями одинакового типа (символьные, численные, логические или даты).

BETWEEN(<Выр>, <Выр1>, <Выр2>) — определяет, лежит ли <Выр> между <Выр1> и <Выр2>.

EMPTY(<Выр>) — определяет, является ли выражение пустым (для логических данных Ложно (.F.)).

EVALUATE(<ВырС>) — вычисляет символьное выражение и возвращает результат. Символьное выражение <ВырС> может быть литеральной символьной строкой, любым допустимым выражением FoxPro или переменной в памяти, элементом массива или полем базы данных, содержащими допустимое выражение FoxPro. Везде, где это возможно, EVALUATE() должна использоваться для замены макроподстановок, использующих медленно работающий оператор &.

MAX(<Выр>, <Выр1> [, <Выр2>...]) — возвращает наибольшее значение из списка символьных, численных выражений или выражений с датой.

MIN(<Выр>, <Выр1> [, <Выр2>...]) — возвращает наименьшее значение из списка символьных, численных выражений или выражений с датой.

TYPE(<Выр>) — возвращает в виде прописной буквы тип данных для <Выр>: C — символьный, L — логический, N — числовой, M — мемо-поле, D — дата, U — неопределенный.

Обработка файлов и дисковые операции

- AVERAGE [<Выр list>] [TO <memvar list> | TO ARRAY <array>] — вычисляет среднеарифметическое значений числовых выражений или полей базы данных.

BOF([<ВырN> | <ВырС>]) — определяет, размещается ли указатель записи на начале файла базы данных в текущей или определенной по номеру (<ВырN>) или псевдониму (<ВырС>) БД.

CURDIR([<ВырС>]) — возвращает в виде строки текущую директорию DOS, на накопителе <ВырС>.

DBF([<ВырN> | <ВырС>]) — возвращает имя базы данных данных в текущей или определенной по номеру (<ВырN>) или псевдониму (<ВырС>) БД.

EOF([<ВырN> | <ВырС>]) — определяет, размещается ли указатель записи на конце файла базы данных в текущей или определенной по номеру (<ВырN>) или псевдониму (<ВырС>) БД.

FCHSIZE(<ВырN>, <ВырN1>) — изменяет размер файла, открытого низкоуровневой функцией работы с файлами и описываемого дескриптором <ВырN>. <ВырN1> — новый размер файла. Возвращается новый размер файла в байтах.

FCLOSE(<ВырN>) — подавляет и закрывает файл или коммуникационный вход, открытый низкоуровневой функции работы с файлами и описываемого дескриптором <ВырN>. Возвращается результат закрытия файла.

FCREATE(<ВырС> [, <ВырN>]) — представляет собой низкоуровневую функцию работы с файлами, которая создает новый файл с именем <ВырС> и открывает его для использования. Когда файл создается, FCREATE() возвращает числовой дескриптор файла. Если файл не может быть создан, возвращается значение -1. Включив необязательный численный аргумент <ВырN> можно указать атрибуты DOS файла (0 — Read/Write (по умолчанию); 1 — ReadOnly; 2 — Hidden; 3 — ReadOnly/Hidden; 4 — System; 5 — ReadOnly/System; 6 — Система/Hidden; 7 — ReadOnly/Hidden/System)

FEOF(<ВырN>) — низкоуровневая функция определяющая, расположен ли указатель файла, описываемого дескриптором <ВырN> на конце этого файла.

FGETS(<ВырN> [, <ВырN1>]) — возвращает серию байтов из файла или коммуникационного порта открытого функциями низкого уровня, описываемого дескриптором <ВырN>. <ВырN1> можно задать число возвращаемых байт (по умолчанию — пока не встретится символ возврата каретки, но не более 254 байт).

FILE(<ВырС>) — определяет наличие на диске файла. <ВырС> может быть именем файла с расширением или включать полное путевое имя DOS.

FOPEN(<ВырС> [, <ВырN>]) — открывает файл или коммуникационный порт (<ВырС>) для использования низкоуровневыми функциями работы с файлами. Если файл или коммуникационный порт успешно открыт, возвращается числовой дескриптор файла или порта. FOPEN() возвращает -1, если файл или порт невозможно открыть. <ВырN> определяет атрибуты чтения/записи файла или схему буферизации (0 — Только чтение (по умолчанию), буферизованный; 1 — Только запись, буферизованный; 2 — Чтение и запись , буферизованный; 10 — Только чтение, небуферизованный; 11 — Только запись, небуферизованный; 12 — Чтение и запись, небуферизованный). Если <ВырN> не включается в FOPEN(), файл открывается только для чтения и не является буферизованным (<ВырN> = 0). Коммуникационный порт должен всегда открываться небуферизованным.

FPUTS(<ВырN>, <ВырС> [, <ВырN1>]) — записывает <ВырС>, возврат каретки и подачу строки в файл или коммуникационный порт, описываемый дескриптором <ВырN>. <ВырN1> может задавать число записываемых символов.

FREAD(<ВырN>, <ВырN1>) — возвращает серию байтов из файла или коммуникационного порта открытого функциями низкого уровня, описываемого дескриптором <ВырN>. <ВырN1> задает число возвращаемых байт, начиная с текущей позиции указателя.

FSEEK(<ВырN>, <ВырN1> [, <ВырN2>]) — перемещает указатель файла в файле, открытом низкоуровневой функцией работы с файлами и описываемого дескриптором <ВырN> на <ВырN1> число байтов. Если <ВырN2> 0, то указатель файла перемещается относительно начала файла (по умолчанию). Если 1, то указатель файла перемещается относительно текущей позиции указателя. Если 2, то указатель файла перемещается относительно конца файла.

FULLPATH(<Файл1>[, <ВырN> | <Файл2>]) — возвращает полное путевое имя DOS для файла <Файл1> или полный путь, относительно <Файл2>. Если включается <ВырN> (любое значение), то исследуется маршрут DOS (а не маршрут FoxPro) для заданного файла.

- SEEK <Выр> — используется для поиска в индексированной базе данных первого вхождения некоторой записи, у которой индексное ключевое выражение подходит к специфицированному в команде выражению <Выр>. Может работать только с индексированными БД и только по текущему индексу. Соответсвие должно быть точным, если только не специфицировано SET EXACT OFF.

SEEK(<Выр> [, <ВырN> | <ВырС>]) — ищет в индексированной БД до первого обнаружения записи, индексный ключ которой соответствует значению определенному аргументом <Выр>, и затем возвращает логическое значение. Заменяет комбинацию команды SEEK и функции FOUND().

- SELECT <ВырN> | <ВырС> — активизирует рабочую область.
- SUM [<Выр list>] [TO <memvar list> | TO ARRAY <array>] — суммирует значения числовых полей в активной базе данных.
- WAIT [<ВырС>] — приостанавливает работу до ввода с клавиатуры символа с выводом системного сообщения "Press any key to continue..." или текста <ВырС>.
- ZAP — удаление всех записей из базы данных

Массивы и переменные

Ссылка на элементы массива производится по их индексам. Каждый элемент в массиве имеет уникальный численный индекс, который идентифицирует его. Если массив одномерный, индекс элемента является его номером строки. Например, индекс элемента в третьей строке массива равен 3. Элементы в двумерных массивах идентифицируются по двум индексам. Первый индекс указывает положение элемента в строках массива, а второй индекс указывает положение элемента в колонках массива. Например, индексы элемента в третьей строке и четвертой колонке массива равны 3, 4. Если массив двумерный, он также может идентифицироваться по одному индексу. Для определения того, какой индекс соответствует паре индексов, элементы сохраняются построчно. Функция AELEMENT() возвращает соответствующий индекс для пары индексов. Обратно, функция ASUBSCRIPT() возвращает пару индексов для одного индекса.

Можно изменить размерность массива повторным использованием команды DIMENSION. Размер массива может быть увеличен или уменьшен, одномерный массив может быть преобразован в двумерный и наоборот. Для изменения размерности массива используйте его имя. Если число элементов в массиве увеличивается, содержание всех элементов в исходном массиве копируется в массив с новой размерностью. Дополнительные элементы массива получают значение "ложно" (.F.). Если число элементов в массиве уменьшается, дополнительные элементы отсекаются, а существующие данные в этих элементах теряются. Если одномерный массив преобразуется в двумерный, содержание исходного одноэлементного массива копируется в новый массив построчно.

ASCAN(<Массив>, <Выр> [, <ВырN>[, <ВырN1>]]) — осуществляет поиск <Выр> в массиве переменных памяти <Массив> и возвращает номер совпадающего элемента или 0. Массив просматривается весь или с элемента <ВырN>. Число просматриваемых элементов может задаваться <ВырN1>.

ASORT(<Массив> [, <ВырN> [, <ВырN1>[, <ВырN2>]]]) — сортирует массив переменных памяти <Массив> в возрастающем и убывающем порядке. <ВырN> задает номер элемента, с которого начинается сортировка. Если задано <ВырN1>, то оно означает число элементов или строк для сортировки, а <ВырN> — номер строки, с которой начинается сортировка. Если <ВырN2> ноль или не задано, то массив сортируется в возрастающем порядке, в противном случае — в порядке убывания.

ASUBSCRIPT(<Массив>, <ВырN>, <ВырN1>) — возвращает индекс строки или столбца элемента массива <Массив>по его номеру <ВырN>. Если <ВырN1> = 1, то возвращается индекс строки если 2, — индекс столбца.

ACOPY(<Массив1>, <Массив2>[, <ВырN> [, <ВырN1>[, <ВырN2>]]]) — копирует заданное число элементов массива <Массив1> в <Массив2>. Возвращается число скопированных элементов. <ВырN> задает элемент в исходном массиве с которого начинается копирование, <ВырN1> число копируемых элементов, <ВырN2> элемент в целевом массиве в который начинается копирование

ADEL(<Массив>, <ВырN> [,2]) — удаляет из <Массива> элемент, строку или столбец с номером <ВырN>. 2 — индикатор того, что <ВырN> задает столбец. Удаление элемента, строки или столбца из массива не изменяет его размера — хвостовые элементы, строки или столбцы массива принимают значение логической лжи (.F.). При успешном удалении элемента, строки или столбца возвращается 1.

ADIR(<Массив> [, <ВырС1>[, <ВырС2>]]) — помещает информацию о подходящих файлах в <Массив>. <ВырС1> определяет шаблон файла, <ВырС2> — дополнительные атрибуты (D для каталогов, H для скрытых файлов, S для систем-

ных файлов, V для метки диска).В следующей таблице описаны все столбцы массива и тип хранимых в них данных:

Столбец	Информация	Тип данных
1	Имена файлов	Символьный
2	Размер файлов	Числовой
3	Даты файлов	Даты
4	Время файлов	Символьный
5	Атрибуты файлов	Символьный (список первых букв)

AELEMENT(<Массив>, <ВырN> [, <ВырN1>]) — возвращает номер элемента массива, соответствующий индексам его строки <ВырN> и столбца <ВырN1>.

AINS(<Массив>, <ВырN> [,2]) — вставляет элемента, строки или столбца в <Массив>. <ВырN> — для одномерного массива — номер вставляемого элемента, для двумерного массива — номер строки или столбца; 2 — индикатор того, что <ВырN> задает столбец. Вставка элемента, строки или столбца не изменяет размера массива — элементы, строки или столбцы, стоящие после вставляемых, перемещаются к концу массива, а последний элемент, строка или столбец удаляются из него. Вновь вставленный элемент, строка или столбец инициализируются логической фальшью (.F.). При успешной вставке возвращается 1.

ALEN(<Массив> [, <ВырN>]) — возвращает число элементов, строк или столбцов в <Массиве>. Если <ВырN> 0, то ALEN возвращает число элементов, Если 1, то ALEN возвращает число строк, Если 2, то ALEN возвращает число столбцов.

DIMENSION <Массив1> (<ВырN> [, <ВырN1>]) [, <Массив2> (<ВырN2> [, <ВырN4>])] ... Команда DIMENSION позволяет создавать одно- или двумерные массивы переменных памяти (<Массив1>, <Массив2>...). По действию и синтаксису команда DIMENSION идентична команде DECLARE. Численные выражения <ВырN> и <ВырN1> определяют количество строк и колонок массива соответственно.

Команды поиска и отображения информации

ALIAS([<ВырN>]) — возвращает псевдоним для базы данных в текущей рабочей области или заданной <ВырN>.

DELETED([<ВырC> | <ВырN>]) — определяет, помечена ли текущая запись на удаление текущей рабочей области или заданной номером <ВырN> или псевдонимом <ВырC>

FILTER([<ВырN> | <ВырC>]) — возвращает выражение фильтра, заданное командой SET FILTER для рабочей области заданной номером <ВырN> или псевдонимом <ВырC>.

FOUND([<ВырN> | <ВырC>]) — определяет успешность выполнения поиска командами CONTINUE, FIND, LOCATE или SEEK для рабочей области заданной номером <ВырN> или псевдонимом <ВырC>.

Команды и функции для работы с окнами

WCOLS([<Окно>]) — возвращает число колонок в окне <Окно>.

WEXIST(<Окно>) — определяет существование <Окна>.

WROWS([<Окно>]) — возвращает число строк в окне <Окно>.

ACTIVATE WINDOW [<Окно>] — выводит на дисплей и активирует ранее определенное пользователем окно.

SHOW WINDOW [<Окно>] — выводит на дисплей ранее определенное пользователем окно, но не активирует его.

DEFINE WINDOW <Окно1>
FROM <row1>, <col1> TO <row2>, <col2>
[TITLE <ВырС1>]

Команда DEFINE WINDOW создает окно пользователя и задает его атрибуты. После определения окон они могут быть выведены на дисплей командами ACTIVATE WINDOW или SHOW WINDOW. Число определяемых окон ограничивается только объемом доступной памяти.

HIDE WINDOW [<Окно>] — удаляет названное окно или окна с экрана. Оно остается резидентным в памяти и активным во всех отношениях.

Коды шаблонов PICTURE и FUNCTION

Коды FUNCTION

Код	Действие
A	Разрешен ввод только алфавитных символов.
B	Выравнивание числовых данных при выводе по левому краю.
C	После положительного числа выводится CR (кредит).
D	Использует текущий формат SET DATE для редактирования данных типа даты.
E	Редактирование данных, рассматривая их как Европейские даты.
I	Выводимый текст центрируется относительно поля.
J	Выводимый текст выравнивается по правому краю поля.
K	Осуществляет выделение поля при указании на него курсором.
L	В числовом поле на дисплее отображаются ведущие нули.
M<list>	Задает несколько предопределенных значений на выбор. <list> представляет собой набор элементов данных, разделенных запятыми. Для скроллинга по <list> достаточно нажать пробел или первую букву соответствующего элемента.
R	При использовании со строкой <format>, которая содержит символы, отличные от кодов шаблона PICTURE, не совпадающие с шаблоном символы отображаются на дисплее, но не помещаются в <var>.
S<n>	Ограничивает ширину отображения на дисплее <n> символами.
T	Отсекает ведущие и хвостовые пробелы в поле.
X	После отрицательных чисел выводится символ DB (дебит).
Z	При выводе поля, если числовое значение равно нулю, выводятся все пробелы.
!	Могут вводиться любые символы; однако, буквы алфавита преобразуются в заглавные.
^	Выводит числовые данные в научной записи.
$	Выводит данные в формате денежной записи.

Коды шаблонов PICTURE

Выражение PICTURE может включать любые необходимые символы; однако, в редактировании и вводе данных активно участвуют только приведенные ниже символы.

Если в формат включить какие-либо другие символы, они будут включены в отображение при выводе, а в случае операции ввода появятся в поле в качестве комментария, и курсор будет перескакивать через них при редактировании. Символы, которые могут быть использованы в выражении шаблона, описаны ниже.

Код	Действие
A	Допускает ввод только алфавитных символов.
L	Допускает только логические данные.
N	Допускает только буквы и цифры.
X	Допускает любые символы.
Y	Допускает только логические значения Y, y, N, n
9	Позволяет вводить только цифры.
#	Позволяет вводить цифры, пробелы и знак.
!	Выполняет преобразование строчных букв в заглавные.
$	Выводит на дисплей текущий денежный символ.
*	Звездочки выводятся перед числовыми значениями.
.	Точка задает позицию десятичной точки.
,	Запятая может использоваться для отделения разрядов цифр.

Системные функции SYS()

Возвращают системную информацию FoxPro.
SYS(0) Номер машины в локальной сети
SYS(1) Юлианскую системную дату
SYS(2) Секунды прошедшие с полуночи
SYS(3) Уникальное имя файла
SYS(5) Драйвер по умолчанию
SYS(6) Текущее устройство принтера
SYS(7 [, <ВырN>]) Текущий файл формата в рабочей области <ВырN>
SYS(9) FoxPro серийный номер
SYS(10, <ВырN>) Строку из номера дня <ВырN>
SYS(11, <ВырD> | <ВырC>) Юлианский номер дня
SYS(12) Свободную память
SYS(13) Состояние принтера
SYS(14,<ВырN>[,<ВырN1>|<ВырC>]) возвращает индексное выражение для индексного тега по его номеру в области <ВырN1>.
SYS(15) Перевод символов
SYS(16) Имя выполняемой программы
SYS(17) Используемый процессор
SYS(18) Текущее поле или объект
SYS(21) Номер главного индекса
SYS(22) Имя главного индекса/тега
SYS(23) FoxPro EMS используемая память
SYS(24) EMS предел памяти
SYS(100) Текущая установка CONSOLE
SYS(101) Текущая установка DEVICE
SYS(102) Текущая установка PRINTER
SYS(103) Текущая установка TALK

SYS(1001) Память FoxPro
SYS(1016) Использование памяти объектного пользователя
SYS(2000) Имя подходящего файла
SYS(2001) Состояние команды SET
SYS(2002) Включение или выключение курсора
SYS(2003) Текущий директорий
SYS(2004) Стартовый директорий FoxPro
SYS(2005) Текущий файл ресурсов
SYS(2006) Текущий графический режим
SYS(2007, <ВырС>) Контрольная сумма символьной строки
SYS(2008) Курсор вставки и замены
SYS(2009) Обмен курсора вставки и замены
SYS(2010) Установка файла CONFIG.SYS
SYS(2011) Текущее состояние блокировки
SYS(2012) Размер блока поля примечаний
SYS(2013) Имя строки системного меню
SYS(2014, <ВырС> [, <ВырС1>]) Возвращает минимальный путь между между файлом и текущей директорией или между файлом и директорией <ВырС1>.
SYS(2015) Уникальное имя процедуры
SYS(2016) Имя окна в SHOW GETS WINDOW
SYS(2017) Высветить экран входа в систему
SYS(2018) Параметры сообщения об ошибке
SYS(2019) Имя и расположение файла CONFIG.FP
SYS(2020) Размер диска принятого по умолчанию
SYS(2021,<ВырN>[,<ВырN><ВырС>]) Выражение фильтра индексации
SYS(2022) Размер кластера диска
SYS(2023) Драйвер для временных файлов

****

&

Назначение
Макроподстановка
&<memvar>[.<ВырС>]

Когда функции & предшествует имя переменной памяти символьного типа, программа FoxPro инструктируется о необходимости выполнения макроподстановки — значение <memvar> заменяет функцию & и имя переменной, которое следует за ним. После этого команда выполняется точно также как если бы она первоначально включала символы из переменной памяти.

Вы можете использовать макроподстановку в любой команде или функции, которая допускает использование цепочки символов.

При использовании функции макроподстановки вы можете предоставить пользователю подсказку, касающуюся информации, а затем применить эту информацию как часть команды, которая требует цепочки литералов.

[.<ВырС>]

Для сцепления дополнительных символов в макрорасширении применяется дополнительный ограничитель "точка" (.). Строка символов, заменяемая функцией &,

сама может содержать функции &, и если существует переменная памяти, имя которой сопровождается подобной вложенной функцией &, содержимое такой переменной также заменяется в команде.

Несмотря на это, нижеследующие команды разрешены для использования:

```
STORE "Customer" TO dbf_file
STORE "Cust_id" TO tagname
USE &dbf_file INDEX &tagname
```

Это использование макроподстановки "расточительно" и должно заменяться непосредственным обращением, например:

```
USE (dbf_file) TAG (tagname)
```

Появляющиеся в управляющих структурах программы операторы макроподстановки (например, DO WHILE) оцениваются только в начале цикла и повторно при последующих итерациях не оцениваются. Любые изменения <memvar> для оценки оператора макроподстановки, имеющие место в цикле, не распознаются.

@ ... SAY/GET

Выполнение ввода/вывода в заданной строке и столбце

```
@ <row, column>
    [SAY <Выр1>
    [PICTURE <ВырС>]
    [FUNCTION <fcodes1>]
    [GET <var>
    [PICTURE <ВырС1>]
    [FUNCTION <fcodes2>]
    [DEFAULT <Выр2>]
    [ENABLE | DISABLE]
    [MESSAGE <ВырС2>]
    [[OPEN]
    [WINDOW <Окно>]
    [RANGE [<Выр3>]
    [, <Выр4>]]
    [SIZE <ВырN1>, <ВырN2>]
    [VALID <ВырL> | <ВырN3>
    [ERROR <ВырС3>]]
    WHEN <ВырL1>]
```

Данный вариант команды @ используется для экранного или оконного форматированного вывода, для создания экранов ввода, либо для форматирования вывода на принтер.

Дополнительные опции

<row,column> (<строка,столбец>)

<row> и <column> представляют собой численные выражения, определяющие местоположение вывода. Для ввода/вывода на экран первая строка имеет номер 0, а номер последней строки равен высоте экрана минус единица. Обычно последняя строка имеет номер 24, если только вы не работаете в режиме расширенного экрана дисплея. Нумерация строк идет сверху-вниз. Для принтера максимальный номер строки ограничивается физическими размерами листа.

Для ввода/вывода на экран первый столбец имеет номер 0, а номер последнего столбца равен ширине экрана дисплея минус единица. Обычно последний столбец имеет

номер 79, если только вы не работаете в режиме расширенного экрана дисплея. Нумерация столбцов идет слева-направо. Для принтера максимальный номер столбца ограничивается физическими размерами листа.

Вывод от оператора @ .. SAY/GET будет направляться на экран, если не активировано окно, определяемое пользователем. При выводе в окно, координаты <row> и <column> являются относительными координатами данного окна, а не координатами самого физического экрана.

Опции оператора SAY

SAY <Выр1>

Если имеется предложение SAY, то <Выр1> оценивается и выводится на дисплей, начиная с позиции <row,column>(<строка,столбец>). <Выр1> может представлять собой UDF (функцию, определяемую пользователем).

Если выло определено SET DEVICE TO PRINT, то вывод направляется на принтер. Если выло определено SET DEVICE TO SCREEN, то вывод направляется на экран. Предложение @...SAY может быть также использовано в форматных файлах.

PICTURE <ВырС>|*FUNCTION* <fcodes1>

При создании поля SAY вы можете включить опцию PICTURE, FUNCTION или сразу обе. Эти опции содержат специальные коды, управляющие выводом и редактированием выражения <Выр1>. Коды FUNCTION могут включаться в предложение PICTURE. В этом случае предложение PICTURE должно начинаться с символа @. Кроме этого, предложение PICTURE может содержать в себе коды FUNCTION, коды PICTURE или их комбинации, но предложение FUNCTION может содержать только коды FUNCTION.

Опции оператора GET <var>

GET <var> выводит поле на экран или в окно для редактирования. При наличии обоих предложений, SAY и GET, после конца вывода SAY и перед началом вывода GET автоматически вставляется пробел. Для инициирования редактирования поля GET в сочетании с SAY должна использоваться команда READ.

PICTURE <ВырС1>

FUNCTION <fcodes2>

Эти предложения имеют здесь то же назначение, что и в предложении SAY, за исключением того, что они используются для редактирования данных, вводимых в ответ на GET. Смотри замечания по PICTURE и FUNCTION выше для более подробной информации.

DEFAULT <Выр2>

Если GET <var> задает переменную памяти и эта переменная не существует, то при включении опции DEFAULT она автоматически создается и инициализируется. Если <var> является элементом массива, то DEFAULT не создает <var>. DEFAULT игнорируется, если <var> является полем базы данных.

Внимание !!!

Если опция DEFAULT не включена и <var> не существует, то выдается предупреждение об ошибке. Переменная <var> не найдена'('Variable <var> not found'). Если опция DEFAULT присутствует и <var> существует, то DEFAULT игнорируется. Выражение <Выр2> в опции DEFAULT задает тип и начальное значение инициализируемой переменной памяти. <Выр2> должна иметь числовой или символьный тип.

ENABLE|DISABLE

*Включ*ение опции DISABLE запрещает доступ и изменение к полю в GET. Поле выводится цветами блокировки и оно не выбирается. По умолчанию поля GET имеют

статус ENABLE (доступные). Вы можете включать опцию ENABLE для напоминания, что поле может выбираться и модифицироваться.

MESSAGE <ВырС2>

Предложение MESSAGE позволяет при помещении курсора в соответствующее поле GET выводить на дисплей заданное символьное выражение <ВырС2>. Сообщение выводится в последней строке экрана или окна и временно отменяет любое выражение SET MESSAGE.

[OPEN] WINDOW <Окно>

Предложение WINDOW позволяет редактирование поля памяти типа memo в окне, определяемом пользователем. Перед этим окно <Окно> должно быть определено с помощью DEFINE WINDOW. В этом случае на экране появится слово Memo. Для того, чтобы открыть окно редактирования memo поля, надо нажать дважды кнопку "мыши" не слове Memo, или установить курсор на слове Memo и нажать Ctrl+Home, Ctrl+PgUp или Ctrl+PgDn. Для удаления окна нажмите кнопку "мыши" на рамке закрытия окна (если возможно) или вне окна. Вы также можете выйти из окна нажатием Ctrl+W, Ctrl+End, Ctrl+Q или Escape.

Если включить в предложение необязательное слово OPEN, то окно памяти открывается по умолчанию, но для входа в него тем не менее требуется нажатие Ctrl-Home, а для выхода — Ctrl-End.

RANGE [<Выр3>][,<Выр4>]

Опция RANGE может использоваться с символьными переменными, датами и числовыми переменными для задания диапазона значений, в пределах которых должны находиться вводимые величины. При задании значения, выходящего за заданные границы, выдается сообщение и указываются допустимые границы.

Значения <Выр3> и <Выр4> должны являться символьными, числовыми выражениями или выражениями типа даты, в зависимости от того, является ли <var> переменной символьного, числового типа или датой.

Если вводимое вами значение не укладывается в заданный диапазон, раздается предупреждающий звуковой сигнал и появляется сообщение, несущее информацию о допустимых значениях диапазона. Можно опустить либо верхнее, либо нижнее граничное значение, но не оба одновременно. Если одно из значений опущено, то проверка его не производится. Если нажать Enter, не изменив значения <var>, то проверка диапазона также не выполняется.

Для переопределения выдаваемого по умолчанию сообщения о границах диапазона служит команда ON READERROR.

SIZE <ВырN1>, <ВырN2>

Команда GET редактирует область высотой в одну строку. Длина поля GET задается редактируемой переменной памяти или полем базы данных, или опцией PICTURE, если она присутствует. Опция SIZE позволяет управлять размером поля GET — высотой и длиной. При редактировании memo поля вы можете задавать размер области редактирования.

Высота поля GET задается в строках параметром <ВырN1>, ширина — в столбцах параметром <ВырN2>. Размер может быть больше или меньше размера редактируемой переменной <var>.

Если вы задали размеры без опции PICTURE:

- если <var> является полем базы данных и размер области редактирования, заданного вами в опции SIZE, больше длины поля, то редактирование происходит в области, совпадающей с размером этого поля базыданных.

- если <var> является переменной памяти и размер области редактирования, заданного вами в опции SIZE, больше длины этой переменной, то редактирование происходит во всем GET поле.

- если <var> является переменной памяти или полем базы данных и размер области редактирования, заданного вами в опции SIZE, меньше длины <var>, то по полю GET осуществляется скроллинг.

Если вы задали размеры с опцией PICTURE:

- если PICTURE задает размер, меньший, чем поле GET, то опция PICTURE имеет приоритет. Например, если опция PICTURE имеет вид "AAA" (три алфавитных символа" и опция SIZE 1,10 (одна строка в высоту и десять столбцов в ширину), то в поле GET вы сможете редактировать первые три символа переменной памяти или поля базы данных.

- если PICTURE задает меньший размер, чем размер поля GET, то в поле GET осуществляется скроллинг.

VALID <ВырL> | <ВырN3>

Предложение VALID позволяет сделать контроль достоверности вводимых с экрана данных частью команды @...GET. Предложение VALID позволяет существенно упростить контроль достоверности экранного ввода при использовании его в сочетании с функциями, определяемыми пользователем. Если функция, определяемая пользователем (UDF), вызывается из предложения VALID, то UDF возвращает данные логического или числового типа. При попытке выхода из поля GET выражение VALID будет оцениваться с использованием значения выражения.

Внимание !!!

В отличие от контроля данных в предложении RANGE, контроль в предложении VALID выполняется всегда, независимо от того, каким способом был произведен выход из поля, за исключением выхода при помощи клавиши Escape. Предложение RANGE осуществляет контроль только при изменении <var>.

VALID с логическим выражением

Если <ВырL> истинно (.T.), ввод считается достоверным, и READ переходит к вводу следующего поля. Если <ВырL> ложно (.F.), то введенное значение считается неверным, и FoxPro выдает сообщение и предупредительный сигнал, сообщающие вам о необходимости повторить после нажатия пробела ввод данных.

VALID с числовым выражением

Если вместо логического выражения в предложении VALID будет задано числовое выражение, то выполняемые действия будут зависеть от значения, возвращаемого <ВырN3>. В целом, предложение VALID, возвращающее числовое значение, может быть использовано для обозначения того, какое поле будет следующим полем GET, к которому обратится READ.

* Ноль указывает на то, что контроль достоверности обнаружил ошибку, и курсор остается в том же поле GET (так же, как при логическом .F.). Как стандартное, так и определяемое при помощи ERROR <ВырC3> сообщения об ошибке в случае использования числового выражения подавляются. Отдельная подпрограмма с сообщением об ошибке может быть написана специально, как часть определяемой пользователем функции, вызываемой предложением VALID.

* Положительное значение указывает относительное число полей, на которое команда READ должна продвинуться вперед, прежде чем начать ввод следующего поля GET. Например, если вы находились на пятом поле GET, а предложение VALID возвратило

число 2 (два), то следующим полем ввода станет седьмое поле GET. Если <ВырN3> выходит за пределы последнего поля GET, то операция READ завершается.

* Отрицательное значение указывает относительное число полей, на которое команда READ должна вернуться назад, прежде чем начать ввод следующего поля GET. Например, если вы находитесь на пятом поле GET, а предложение GET возвратило отрицательное число два (-2), то следующим полем ввода станет третье поле GET. Если <ВырN3> выходит за пределы первого поля GET, то операция READ завершается.

ERROR <ВырC3>

ERROR <ВырC3> позволяет вам задавать собственное сообщение об ошибке в случае оценки VALID <ВырL> как ложно (.F.). Выражение <ВырC3> появляется на месте стандартного сообщения об ошибке.

WHEN <ВырL1>

Предложение WHEN разрешает или запрещает редактирование поля GET в зависимости от результата оценки заданного условия <ВырL1>. Прежде чем курсор сможет попасть в данное поле для начала редактирования, оценка <ВырL1> должна дать логическое .T. Если задано предложение WHEN, а оценка <ВырL1> дает .F., ввод не разрешен, а доступ к полю невозможен, и курсор в таком случае перемещается к следующему полю GET.

Органы управления

В следующих семи параграфах обсуждаются органы управления FoxPro. Органы управления представляют собой области экрана, используемые для выбора, подтверждения или отмены действий. Они могут появляться в окнах, диалоговых блоках или на основном экране. Блок проверки, селективная кнопка и всплывающие меню являются обычными примерами органов управления.

FoxPro очень активно использует управляющий интерфейс. Для примера достаточно взглянуть на диалоговые блоки Открыть Файл (Open File) или Установить параметры печати (Printer Setup) и убедиться, насколько разнообразные управляющие средства там используются. Дополнительная информация о порядке использования органов управления приводиться в книгах документации по FoxPro "Руководство по Интерфейсу" (Interface Guide) и "Начало Работы" (Getting Started).

Краткое описание основных органов управления

Ниже приводится краткое описание различных органов управления и порядка их использования:

1. Блоки Проверки

На экране блок проверки представляет собой пару квадратных скобок со строкой текста справа от них. Например:

[X] All Files

Блок проверки используется для маркировки двух состояний — истина (.T.) и ложь (.F.) (да или нет).

2. Списки

На экране список представляет собой вертикальный список возможностей внутри некоторой рамки, часто с полосами прокрутки с правой стороны. Списки обычно используются для показа списка директориев, файлов или полей с возможностью их выбора.

3. Всплывающие меню

На экране всплывающее меню представляет собой прямоугольник, ограниченный двойными линиями справа и внизу. При выборе всплывающего меню разворачивается

список возможных опций. Всплывающие меню используются для показы списка опций, только одна из которых может быть выбрана.

4. Текстовые кнопки

На экране текстовая кнопка представляет собой строку текста заключенную в угловые скобки. Например:

< OK > <Cancel>

Обычно текстовые кнопки используются для инициализации действий.

5. Селективные кнопки

Селективные кнопки представляют собой следующее:

(●) Apples(Яблоки)
( ) Apricots (Абрикосы)
( ) Lemons(Лимоны)
( ) Oranges (Апельсины)

Селективные кнопки напоминают кнопки выбора диапазонов автомобильного приемника — выбирая одну кнопку вы делаете ее текущей и тем самым отпускаете нажатую до этого. Знак ● (точка) указывает на выбранную в настоящий момент кнопку. Селективные кнопки обычно используются для показа нескольких опций, только одна из которых может быть выбрана.

6. Области модификации текста

Область модификации текста представляет собой прямоугольную область экрана, в пределах которой вы можете модифицировать и вводить текст. Вводимый текст хранится в переменной памяти или в поле базы данных. В этой области доступны все стандартные для FoxPro возможности редактирования: вырезание, копирование и вставка. В отличии от остальных органов управления FoxPro (блоков проверки, селективных и текстовых кнопок, ...), эта команда в своем синтаксисе использует слово EDIT, вместо GET. Области редактирования текста обычно используются для обновления и модификации memo полей и длинных символьных полей.

7. Невидимые кнопки

Невидимые кнопки представляют собой прямоугольные области экрана или окна которые могут выбираться. Для размещения поверх этих областей текста могут использоваться команды @ ... SAY. Невидимые кнопки могут использоваться для создания псевдо иконок или инициализации действий при нажатии кнопок "мыши" в поле данных.

Как они работают

Большинство органов управления создаются командой @ ... GET (только область редактирования текста создается командой @ ... EDIT). Опции PICTURE и FUNCTION команды @ ... GET определяют тип органа управления. Эти опции также содержат подсказки, связанные с большинством органов управления. Многие органы управления инициализируют выполнение некоторых действий. Это действие задается опцией VALID. Органы управления активизируются командой READ. В зависимости от заданных опций, при нажатии кнопки или выборе пункта выполнение команды READ завершается или не завершается.

"Горячие" клавиши

Для большинства органов управления могут быть определены "горячие" клавиши. "Горячие" клавиши обеспечивают быстрый способ нажатия кнопок или выбора опций. "Горячая" клавиши соответствует подсвеченной букве (обычно это первая буква в приглашении кнопки или пункте меню), при нажатии которой заданное действие выполняется немедленно.

Заблокированные органы управления

Для обеспечения недоступности органа управления (или его части) вы можете заблокировать его. Заблокированные управляющие средства, кнопки и пункты меню изображаются блокировочным цветом (обычно блеклым) и не поддаются выбору.

Краткое описание синтаксиса

В следующей таблице приводятся различия в синтаксисе, опциях и значениях по умолчанию различных управляющих средств. Более подробная информация дается при описании соответствующих команд.

Синтаксическая таблица органов управления

Управляющее средство	Код PICTURE	Дополнительные коды опций PICTURE/ FUNCTION	По умолчанию	Специальные символы PICTURE/ FUNCTION
1. Блоки проверки	*C	N — не завершать T — завершать	Не завершать	\< —"Горячая" клавиша \\ — Заблокировать
2. Невидимые кнопки	*I	N — не завершать T — завершать H — горизонтальная V — вертикальная	Не завершать Вертикальная	\\ — Заблокировать
3. Списки проверки	&	N — не завершать T — завершать	Завершать	Нет
4. Всплывающие меню	^	N — не завершать T — завершать	Не завершать	\< —"Горячая" клавиша \\ — Заблокировать
5. Текстовые кнопки	*	N — не завершать T — завершать H — горизонтальная V — вертикальная	Завершать Вертикальная	\< —"Горячая" клавиша \\ — Заблокировать \! — по умолчанию \? — Escape
6. Селективные кнопки	*R	N — не завершать T — завершать H — горизонтальная V — вертикальная	Не завершать Вертикальная	\< —"Горячая" клавиша \\ — Заблокировать
7. Области редактирования текста	Нет	I — центровка J — выравнивание по правой границе	по левой границе	Выравнивание Нет

@ ... GET — Блоки проверки

@ <row, column> GET <var>
FUNCTION <ВырС>
< PICTURE <ВырС1>
[DEFAULT <Выр>]
[SIZE <ВырN>, <ВырN1>]
[ENABLE | DISABLE]
[MESSAGE <ВырС2>]
[VALID <ВырL>]
[WHEN <ВырL1>]

[Эта разновидность команды @ ... GET предназначена для создания блока проверки. Блок проверки используется для переключения между двумя состояниями — истина (.T.) или ложь (.F.). На экране блок проверки представляет собой пару квадратных скобок со строкой текста справа от них.

Текст приглашения задается опциями PICTURE или FUNCTION. Когда блок проверки проверяется (условие истинно) между скобками устанавливается знак X. Одной командой @ ... GET создается только один блок проверки. Для активации блоков используется команда READ.

<row,column> (<строка,столбец>)

Расположение блока проверки на экране или в активном окне задается парой <row,column>. Координаты <row,column> задаются численными выражениями. Row может принимать значения от 0 до максимального числа строк на экране или в активном окне. Column может принимать значения от 0 до максимального числа столбцов на экране или в активном окне. Строки номеруются сверху-вниз, а столбцы слева-направо.

GET <var>

При задании проверки блока или наоборот, ваш выбор хранится в <var>. Это может быть переменная памяти, элемент массива или поле базы данных. <var> должна иметь числовой или логический тип. При первом появлении блока проверки на экране или в окне, его проверка осуществляется, если <var> содержит не нулевое числовое значение или логическую истину .T.. Блок не проверяется, если <var> равно нулю или ложно (.F.).

Команда READ активизирует блок проверки. Состояние блока проверки после завершения работы команды READ определяется хранимым в <var> значением: 0 или .F. для проверяемого, 1 или .T. для не проверяемого.

FUNCTION <ВырС> | *PICTURE* <ВырС1>

При создании блока проверки вы обязательно должны включать опцию FUNCTION, PICTURE или обе. FUNCTION или PICTURE содержат код спецификации блока проверки (он указывает на тип определяемой кнопки) и текст его приглашения. Код *C является кодом спецификации всех блоков проверки.

Символьное выражение <ВырС> в опции FUNCTION должно всегда начинаться с *C. Для создания приглашения после *C ставится пробел и его текст. Например, это предложение создает блок проверки Tax Exempt:

.... FUNCTION '*C Tax Exempt' ...

Выражение <ВырС1> в опции PICTURE имеет тот же синтаксис, что и выражение в FUNCTION, за исключением того, что выражение в PICTURE должно обязательно начинаться с AT символа (@) со следующим за ним кодом *C. Например, это предложение создает блок проверки Tax Exempt:

.... PICTURE '@*C Tax Exempt' ...

Вы также можете включать обе опции FUNCTION и PICTURE для создания блока проверки. Если включаются обе опции, символьное выражение <ВырС> в FUNCTION должно содержать *С для создания блока проверки. Символьное выражение <ВырС1> в PICTURE должно включать текст приглашения.

Следующие примеры демонстрируют различные способы создания блока проверки. Блок проверки располагается на экране во второй строке и втором столбце. Состояние блока (проверяемый или не проверяемый) хранится в переменной памяти CHOICE. Этот блок проверки создается в каждом примере:

Опции N и T в FUNCTION и PICTURE

Непосредственно после кода С в предложениях FUNCTION и PICTURE монут присутствовать две опции, определяющие должна ли команда READ завершаться после выбора блока проверки:

Опция	Описание
N	Не завершает выполнение READ после выбора блока проверки. Устанавливается по умолчанию.
T	Завершает выполнение READ после выбора блока проверки.

Блоки проверки со специальными возможностями

Вы можете присваивать специальные характеристики блоку проверки. Принудительно вы можете назначать "горячие" клавиши или ставить блокировки путем включения специальных символов при определении приглашения.

"Горячие" клавиши

"Горячая" клавиша соответствует подсвеченной букве, при нажатии которой немедленно осуществляется выбор (изменение состояния) блока проверки. Для создания "горячей" клавиши поместите обратный слэш и знак меньше (/<) перед требуемым символом приглашения.

Внимание !!!

"Горячая клавиша" не выбирает блок проверки, если текущее поле является областью редактирования текста или полем ввода. Вместо этого символ "горячей" клавиши вводится в текущее поле.

Блокирование блоков проверки

Заблокированный блок проверки не может бать выбран. Он выводится в цвете блокировки. Для блокирования блока проверки поместите два обратных слэша \\ до приглашения блока. Следующий пример блокирует блок проверки Tax Exempt:

```
STORE 1 TO choice
@ 2,2 GET choice FUNCTION '*C \\Tax Exempt'
READ
DEFAULT <Выр>
```

При задании проверки блока или наоборот, состояние блока (проверяемый или не проверяемый) хранится в переменной памяти, элементе массива или поле базы данных. Если вы задаете переменную памяти и она не существует, то при включении предложения DEFAULT эта переменная памяти автоматически создается и инициализируется. Если <var> является элементом массива или полем, то DEFAULT не создает <var>.

Внимание !!!

Если опция DEFAULT не включена и <var> не существует, то выдается предупреждение об ошибке 'Переменная <var> не найдена'('Variable <var> not found').

Если опция DEFAULT присутствует и <var> существует, то DEFAULT игнорируется.

Выражение <Выр> в опции DEFAULT задает тип и начальное значение инициализируемой переменной памяти. <Выр> должна иметь числовой или логический тип.

SIZE <ВырN>,<ВырN1>

Числовое выражение <ВырN> задает высоту органа управления. Блок проверки всегда имеет высоту в одну строку, поэтому значение <ВырN> всегда игнорируется. Но выражение <ВырN> должно всегда присутствовать при задании <ВырN1>.

По умолчанию, ширина блока проверки определяется длиной приглашения. Числовое выражение <ВырN1> изменяет ширину блока проверки (в столбцах). Если ширина, задаваемая <ВырN1>, меньше ширины приглашения, то приглашение не усекается.

ENABLE|DISABLE

По умолчанию, блок диалога является доступным при выполнении команды READ. Вы можете захотеть задержать возможность выбора блока проверки до выполнения определенных условий. Включение опции DISABLE предотвращает активацию блока проверки при выполнении команды READ. Заблокированный блок проверки не может быть выбран и отображается в цвете блокировки. Для получения доступа к заблокированным командам GET используется команда SHOW GET ENABLED.

MESSAGE <ВырC2>

Символьное выражение <ВырC2> необязательного предложения MESSAGE выдается при появлении блока проверки. По умолчанию, сообщение выдается отцентрированным в последней строке "стола". Местоположение сообщения может изменяться командой SET MESSAGE.

VALID <ВырL>

Вы можете включить необязательное предложение VALID. Выражение <ВырL> проверяется только после осуществления выбора блока проверки (проверяемый или не проверяемый). Это означает, что VALID не проверяется при установке на блок, а только после того как вы реально выбрали его состояние посредством "мыши", клавиш Enter или Escape.

Обычно <ВырL> является функцией, определяемой пользователем. Эти функции позволяют делать очень многое, включая: выбор, обеспечение доступности или блокировки других полей GET или объектов, инициализацию секций Browse, вызов других экранов ввода данных, позиционирование на новую запись. Для завершения выполнения команды READ в функции, определяемые пользователем, может включаться команда CLEAR READ.

WHEN <ВырL1>

В зависимости от логического значения выражения <ВырL1> необязательное предложение WHEN разрешает или наоборот запрещает выбор блока проверки. До выбора блока выражение <ВырL1> должно быть оценено как логическая истина (.T.). Если <ВырL1> оценивается как ложь (.F.), то блок проверки становится невыбираемым и пропускается, если располагается между другими полями GET или объектами.

@ ... GET — Списки

@ <row, column> GET <var> | <field>
FUNCTION <ВырC>]
[PICTURE <ВырC1>]
ROM <array>
RANGE <ВырN> [, <ВырN1>]]
POPUP <popup name>
DEFAULT <Выр>]

SIZE <ВырN2>, <ВырN3>]
ENABLE | DISABLE]
MESSAGE <ВырC2>]
VALID <ВырL>]
WHEN <ВырL1>]

Эта разновидность команды @ ... GET предназначена для создания списка. Список содержит выбираемые вами элементы. Для выбора элемента надо перейти на него и нажать в это время Enter или два раза кнопку "мыши".

На экране список представляет собой вертикальный список возможностей внутри некоторой рамки, часто с полосами прокрутки с правой стороны. Полосы прокрутки позволяют более быстро перемещаться по элементам списка с помощью "мыши" и визуально отображать текущее местоположение. Другой способ быстрого перемещения по списку, это использование клавиши Home для перехода к первому элементу и клавиши End для перехода к последнему элементу. Второй способ работает вне зависимости от наличия в списке полос прокрутки.

Элементы списка могут браться или из массива, или из всплывающего меню. Для построения списка из массива используется опция FROM <array> (<массив>). При использовании опции POPUP <popup name> (<имя меню>) список строится из всплывающего меню, которое предварительно создано командой DEFINE POPUP. Разрешается использование только одной опции FROM или POPUP, но не обеих сразу. Список активизируется командой READ.

Опции

<row,column> (<строка,столбец>)

Правый левый угол списка располагается на экране или в активном окне по адресу, задаваемому парой <row,column>. Координаты <row,column> задаются численными выражениями. Row может принимать значения от 0 до максимального числа строк на экране или в активном окне. Column может принимать значения от 0 до максимального числа столбцов на экране или в активном окне. Строки номеруются сверху-вниз, а столбцы слева-направо.

GET <var>|<field>

При выборе элемента списка, значение, соответствующее вашему выбору, заносится в <var> — элемент массива или переменную памяти, или в <field> — поле базы данных. <var> и <field> должны иметь числовой или символьный тип. Если <var> или <field> имеют числовой тип, то заносится порядковый номер выбранного элемента. Если они имеют символьный тип, то заносится приглашение элемента списка.

Установка начального состояния. При появлении списка значение <var> или <field> определяет, на какой элемент списка (если он вообще есть) устанавливается указатель. Например, если <var> или <field> равно 4, то при активации списка командой READ указатель будет установлен на 4-й элемент. Если <var> или <field> не соответствуют ни какому элементу списка (значение меньше 1 или больше числа элементов в списке), то указатель вообще не устанавливается.

FROM <array>

Опция FROM <array> строит список из массива. Массив может быть 1 и 2-х мерным. Если используется массив с размерностью 1, то первый элемент массива будет первым элементом списка, второй элемент массива вторым элементом списка и т.д.

При использовании двумерного массива для построения списка используются элементы в первом столбце массива. Первый элемент массива является первым элементом списка, второй элемент того же столбца вторым элементом списка и т.д.

RANGE <ВырN>[,<ВырN1>]

Элементы списка начинаются с первого элемента массива. Вы можете задавать различные начальные элементы в массиве путем включения опции RANGE <ВырN>. Например, если массив одномерный и <ВырN> равно 3, то третий элемент массива будет первым элементом списка, четвертый элемент массива — вторым элементом списка и т.д. Позиции элементов в двумерных массивах нумеруются по строкам. Например, предположим, что вы создали массив 3x3:

a b c
d e f
g h i

Элементы a, b и c располагаются в позициях 1, 2 и 3. Элементы d, e, f — в позициях 4, 5, 6 и т.д. При использовании двумерных массивов только элементы столбца в котором находится элемент с номером <ВырN> становятся элементами списка. Поэтому, например, если <ВырN> равно 2, то элементами списка станут элементы b, e и h. Если <ВырN> равно 5, то ими станут только элементы e и h. Если вы установили начальный элемент параметром <ВырN>, то пареметром <ВырN1> вы можете также установить общее число элементов в списке. Если <ВырN1> не включается, то элементами списка становятся все элементы массива начиная с элемента <ВырN> и до конца массива.

Кроме того, элементы списка всегда принадлежат одному столбцу. При выполнении команды SHOW GETS опция RANGE всегда перепроверяется. И если значения <ВырN> или <ВырN1> изменились, то в соответствии с ними происходит обновление списка.

Содержимое списка может динамически изменяться — модификацией массива вы можете вставлять и удалять его элементы. Манипуляции с массивами обеспечиваются функциями ACOPY(), ADEL(), ADIR(), AINS(), ALEN(), ASCAN() и ASORT().

POPUP <popup name> (<имя меню>)

Элементы списка могут быть созданы и из всплывающего меню, предварительно созданного командой DEFINE POPUP. Каждый появляющийся в меню пункт используется для создания элементов списка. Для создания списка с элементами из меню вы должны вначале определить соответствующее меню командой DEFINE POPUP. Команда DEFINE POPUP детально обсуждается ниже в этом руководстве. В этой опции используется тоже имя <popup menu>, что и в команде DEFINE POPUP.

Вы можете включить в DEFINE POPUP опцию PROMPT и тем самым создать меню (а следовательно и список), содержащее значения поля базы данных, имена доступных на диске файлов или названия полей базы данных. Для создания списка со значениями поля базы данных при создании меню включите опцию PROMPT FIELD. Для создания списка, содержащего имена доступных в настоящий момент файлов на диске, включите PROMPT FILES. Для создания списка с названиями полей базы данных включите PROMPT STRUCTURE.

Некоторые доступные при использовании команды DEFINE POPUP опции влияют на создаваемый из соответствующего меню список. Символ маркировки, по умолчанию ромбик, располагается сразу за выбранным из списка элементом. Если опция MARK включается в DEFINE POPUP, то будет использоваться символ маркировки, отличный от установленного по умолчанию. Использованием в DEFINE POPUP ключевого слова MARGIN вы можете выделить дополнительное место в списке для символа маркировки. Если в DEFINE POPUP включается ключевое слово SCROLL, то справа от списка будут располагаться полосы прокрутки.

FUNCTION <ВырC> | PICTURE <ВырC1>

Включайте FUNCTION '#T' или PICTURE '@&T' для завершения выполнения команды READ после выбора элемента. Использование FUNCTION '&N' или PICTURE '@&N' изменит поведение по умолчанию и не завершит работу команды READ после выбора элемента. Например:

... FUNCTION '#T' ...

... PICTURE '@&T' ...

DEFAULT <Выр>

При выборе элемента списка результат заносится в <var>. Если вы задаете <var> как переменную памяти и она не существует, то при включении предложения DEFAULT эта переменная памяти автоматически создается и инициализируется. Если <var> является элементом массива или полем, то DEFAULT не создает <var>.

Внимание !!!

Если опция DEFAULT не включена и <var> не существует, то выдается предупреждение об ошибке 'Переменная <var> не найдена'('Variable <var> not found').

Если опция DEFAULT присутствует и <var> существует, то DEFAULT игнорируется.

Выражение <Выр> в опции DEFAULT задает тип и начальное значение инициализируемой переменной памяти. Оно должно иметь числовой или символьный тип.

SIZE <ВырN2>,<ВырN3>

Ширина списка определяется шириной наиболее длинного его элемента. Число пунктом меню или элементов массива задает число отображаемых элементов списка. Включая опцию SIZE вы можете сами задавать длину и ширину списка. Длина списка измеряется в строках и задается выражением <ВырN2>, ширина списка измеряется в столбцах и задается выражением <ВырN3>.

Если все элементы списка одновременно не умещаются на экране, то справа автоматически появляются полосы прокрутки. Если список создается из DEFINE POPUP в которой используется ключевое слово SCROLL, то справа от списка всегда появляются полосы прокрутки.

ENABLE|DISABLE

Вы можете захотеть задержать возможность выбора из списка до выполнения определенных условий. Включение опции DISABLE предотвращает активацию списка при выполнении команды READ. Заблокированный список не может быть выбран и отображаются в цвете блокировки. Для получения доступа к заблокированным командам GET используется команда SHOW GET ENABLED.

MESSAGE <ВырC2>

Символьное выражение <ВырC2> выдается при установке на элемент списка. По умолчанию, сообщение выдается отцентрированным в последней строке "стола". Местоположение сообщения может изменяться командой SET MESSAGE.

VALID <ВырL>

Выражение <ВырL> проверяется только после осуществления выбора элемента списка. VALID не проверяется при перемещении по списку, но при нажатии Enter или кнопки "мыши" сразу осуществляется проверка. Обычно <ВырL> является функцией, определяемой пользователем. Эти функции позволяют делать очень многое, включая: выбор, обеспечение доступности или блокировки других полей GET или объектов, инициализацию секций Browse, вызов других экранов ввода данных, позиционирование на новую запись. Для завершения выполнения команды READ в функции, определяемые пользователем, может включаться команда CLEAR READ.

WHEN <ВырL1>

WHEN выполняется при первом перемещении (установке) на элемент. Подобно VALID выражение WHEN обычно является функцией, определяемой пользователем.

@ ... GET — Вертикальные меню

@ <row, column> GET <var>
UNCTION <ВырС>
PICTURE <ВырС1>
DEFAULT <Выр>]
FROM <array>]
RANGE <ВырN> [, <ВырN1>]]
[SIZE <ВырN2>, <ВырN3>]
[ENABLE | DISABLE]
[MESSAGE <ВырС2>]
[VALID <ВырL>]
[WHEN <ВырL1>]

Эта разновидность команды @ ... GET предназначена для создания всплывающих меню. На экране всплывающее меню представляет собой прямоугольник, ограниченный двойными линиями справа и внизу. При выборе всплывающего меню разворачивается список возможных опций. Список опций задается в FUNCTION и PICTURE.

Опции

<row,column> (<строка,столбец>)

Правый левый угол всплывающего меню располагается на экране или в активном окне по адресу, задаваемому парой <row,column>. Координаты <row,column> задаются численными выражениями. Row может принимать значения от 0 до максимального числа строк на экране или в активном окне. Column может принимать значения от 0 до максимального числа столбцов на экране или в активном окне. Строки номеруются сверху-вниз, а столбцы слева-направо.

GET <var>

Пункт меню, на который первоначально устанавливается определяется значением <var>. <var> может быть переменной памяти, элементом массива или полем базы данных. При выборе пункта меню ваш выбор заносится в <var>. Если для выхода из меню нажата клавиша Escape, то значение <var> остается неизменным.

<var> должна иметь числовой или символьный тип. Если <var> символьного типа, то в нее заносится приглашение выбранного пункта. Если <var> числового типа, то заносится номер выбранного пункта меню.

Установка начального состояния

При появлении всплывающего меню значение <var> определяет, на какой пункт меню (если он вообще есть) устанавливается указатель. Если <var> числовая, то первоначально указатель устанавливается на пункт меню с номером, указываемым <var>. Например, если элемент или поле равно 1, то указатель будет установлен на первый пункт меню. Если значение <var> не соответствует никакому пункту меню (значение меньше 1 или больше числа пунктов) то первоначально появляется пустое меню. Если <var> имеет символьный тип, то осуществляется сравнение с учетом регистра между <var> и пунктами меню. При сравнении игнорируются все управляющие символы, лидирующие и хвостовые пробелы. Если найден подходящий пункт, то при инициализации на него устанавливается указатель. Если ни один из пунктов меню не соответствует значению <var>, то это значение появляется на лицевой стороне меню и добавляется как временный пункт в конец меню.

FUNCTION <ВырС> | *PICTURE* <ВырС1>

При создании всплывающих меню вы обязательно должны включать опцию FUNCTION, PICTURE или обе. FUNCTION или PICTURE содержат код спецификации всплывающего меню и список опций. Код ^ является кодом спецификации всплывающих меню.

Внимание !!!

Опции могут быть заданы и в предложении FROM. Если опции задаются там, то вам необязательно включать их в FUNCTION или PICTURE. Однако, если вы включаете FROM, символ ^ обязательно должен быть задан в FUNCTION или PICTURE.

Символьное выражение <ВырС> в опции FUNCTION должно всегда начинаться с ^. Для создании пунктов, появляющихся в меню, включается через пробел после ^ список соответствующих им опций, разделенных точкой с запятой. Следующий пример создает пункты меню Cash, Charge, Net 30, Net 60 :

... FUNCTION '^ Cash;Charge;Net 30;Net 60'...

Выражение <ВырС1> в опции PICTURE имеет тот же синтаксис, что и выражение в FUNCTION, за исключением того, что выражение в PICTURE должно обязательно начинаться с AT символа (@) со следующим за ним кодом ^. Например, это предложение создает пункты меню — Charge, Net 30, Net 60 :

... PICTURE '@^ Cash;Charge;Net 30;Net 60'...

Вы также можете включать обе опции FUNCTION и PICTURE для создания всплывающих меню. Если включаются обе опции, символьное выражение <ВырС> в FUNCTION должно содержать ^ для создания всплывающего меню, и следующие за ней через пробел опции его пунктов. Символьное выражение <ВырС1> в PICTURE может включать приглашения для создания дополнительных пунктов меню.

Опции N и T в FUNCTION и PICTURE

Непосредственно после кода ^ в предложениях FUNCTION и PICTURE могут присутствовать две опции, определяющие должна ли команда READ завершаться после выбора пункта меню:

Опция	Описание
N	Не завершает выполнение READ после выбора пункта меню. Устанавливается по умолчанию.
T	Завершает выполнение READ после выбора пункта меню.

Пункты всплывающих меню со специальными свойствами

Вы можете устанавливать специальные характеристики пунктов всплывающих меню. Например, заданием специальных символов при описании пунктов меню вы можете задать "горячие" клавиши или заблокировать опции.

"Горячие" клавиши

"Горячая" клавиша представляет собой подсвеченную букве, нажатие которой вызывает немедленный выбор соответствующего пункта. "Горячие" клавиши работают только после разворачивания меню. Для присвоения горячей клавиши перед требуемым символом в пункте установите обратный слеш и знак меньше (\<).

Заметим, однако, что если в приглашении символ появляется более одного раза, то только первое его появление может стать "горячей" клавишей . Например рассмотрим следующее предложение FUNCTION:

... FUNCTION '^ Cu\<cumbers; ...'

Хотя в приглашении Cucumbers как "горячая" клавиша задается вторая буква "c", при активации только первая "c" будет подсвечена.

Блокировка пунктов меню

Заблокированные опции невозможно выбрать. Они выводятся цветом блокировки. Для блокирования пункта меню достаточно поставить перед ним обратный слеш \. Символ \ блокирует только один пункт. Для блокировки всего меню смотри описание опции DISABLE ниже. В этом примере блокируется пункт Charge:

```
STORE 1 TO choice
@ 4,2 GET choice FUNCTION '^ Cash;\Charge;Net 30;Net 60'
READ
```

DEFAULT <Выр>

При выборе пункта меню ваш выбор записывается в переменную памяти, элемент массива или поле базы данных. Если вы задаете <var> как переменную памяти и она не существует, то при включении предложения DEFAULT эта переменная памяти автоматически создается и инициализируется. Если <var> является элементом массива или полем, то DEFAULT не создает <var>.

Внимание !!!

Если опция DEFAULT не включена и <var> не существует, то выдается предупреждение об ошибке 'Переменная <var> не найдена'('Variable <var> not found').

Если опция DEFAULT присутствует и <var> существует, то DEFAULT игнорируется.

Выражение <Выр> в опции DEFAULT задает тип и начальное значение инициализируемой переменной памяти. <Выр> должна иметь числовой или символьный тип.

FROM <array>

Опция FROM <array> может использоваться для создания всплывающего меню из массива. Если вы создаете пункты меню с помощью FROM <array>, то вы все равно должны задать @^ в PICTURE или ^ в FUNCTION.

При задании FROM <array> все другие опции, заданные в FUNCTION или PICTURE игнорируются. Массив может иметь размерность 1 или 2. Пункты, появляющееся в таком меню, берутся из элементов заданного массива <array>. Если используется массив с размерностью 1, то первый элемент массива будет первым пунктом меню, второй элемент массива вторым пунктом меню и т.д.

При использовании массива размерности 2 для построения меню используются элементы в первом столбце массива. Первый элемент массива является первым пунктом меню, второй элемент того же столбца вторым пунктом меню и т.д.

RANGE <ВырN>[,<ВырN1>]

По умолчанию пункты меню начинаются с первого элемента массива. Вы можете задавать различные начальные элементы в массиве путем включения опции RANGE <ВырN>. Например, если массив одномерный и <ВырN> равно 3, то третий элемент массива создает первый пункт меню, четвертый элемент массива — второй пункт меню и т.д.

Позиции элементов в двумерных массивах нумеруются по строкам. Например предположим, что вы создали массив 3x3:

```
a b c
d e f
g h i
```

Элементы a, b и c располагаются в позициях 1, 2 и 3. Элементы d, e, f — в позициях 4, 5, 6 и т.д. При использовании двумерных массивов только элементы столбца в котором находится элемент с номером <ВырN> создают пункты меню. Поэтому, например, если <ВырN> равно 2, то для создания пунктов меню будут использоваться элементы b, e и h. Если <ВырN> равно 5, то будут использоваться только элементы e и h. Если вы установили начальный элемент параметром <ВырN>, то пареметром <ВырN1> вы можете так-

же установить общее число элементов, используемых для создания пунктов меню. Число появляющихся пунктов меню задается параметром <ВырN1>. Если <ВырN1> не включается, то в создании пунктов меню участвуют все элементы массива начиная с элемента <ВырN> и до конца массива.

В случае двумерного массива <ВырN1> задает число используемых элементов массива, содержащихся в одном столбце с начальным элементом с номером <ВырN>. Например, если <ВырN> равно 2 и <ВырN1> равно 3, то меню будет создаваться из второго элемента массива и следующих двух элементов этого же столбца.

Содержимое всплывающего меню может динамически изменяться — модификацией массива вы можете вставлять и удалять его пункты, а также существует возможность изменения числа появляющихся пунктов. При выполнении команды SHOW GETS опция RANGE всегда перепроверяется. И если значения <ВырN> или <ВырN1> изменились, то в соответствии с ними происходит обновление меню. Манипуляции с массивами обеспечиваются функциями ACOPY(), ADEL(), ADIR(), AINS(), ALEN(), ASCAN() и ASORT().

SIZE <ВырN2>[,<ВырN3>]

По умолчанию ширина всплывающего меню определяется шириной его самого широкого пункта. Вы можете принудительно изменить ширину меню, путем задания <ВырN3>. Другой аргумент SIZE — <ВырN2>, игнорируется. Для него вы можете задавать любое числовое значение.

ENABLE|DISABLE

По умолчанию всплывающее меню становится доступным при использовании команды READ. Вы можете захотеть задержать возможность выбора из меню до выполнения определенных условий. Включение опции DISABLE предотвращает активацию меню при выполнении команды READ. Заблокированное меню не может быть выбрано и отображается в цвете блокировки. Блокировка одного пункта меню описывается в параграфе "Блокировка пунктов меню" выше. Для получения доступа к заблокированным командам GET используется команда SHOW GET ENABLED.

MESSAGE <ВырC2>

Символьное выражение <ВырC2> выдается при активации меню. По умолчанию, сообщение выдается отцентрированным в последней строке "стола". Местоположение сообщения может изменяться командой SET MESSAGE.

VALID <ВырL>

Вы можете включить необязательное предложение VALID. Выражение <ВырL> проверяется только после осуществления выбора пункта меню. Это значит, что VALID не проверяется при перемещении по меню, но при нажатии клавиш Enter, Spase или кнопки "мыши" сразу осуществляется проверка.

Обычно <ВырL> является функцией, определяемой пользователем. Эти функции позволяют делать очень многое, включая: выбор, обеспечение доступности или блокировки других полей GET или объектов, инициализацию секций Browse, вызов других экранов ввода данных, позиционирование на новую запись. Для завершения выполнения команды READ в функции, определяемые пользователем, может включаться команда CLEAR READ.

WHEN <ВырL1>

В зависимости от логического значения выражения <ВырL1> необязательное предложение WHEN разрешает или наоборот запрещает выбор пунктов меню. До начала выбора из меню выражение <ВырL1> должно быть оценено как логическая истина

(.T.). Если <ВырL1> оценивается как ложь (.F.), то всплывающее меню не может быть активировано и оно пропускается с переходом на другую команду GET.

@ ... GET — Текстовые кнопки

Назначение

Создание текстовой кнопки.

Синтаксис

@ <row, column> GET <var>
FUNCTION <ВырC>
< PICTURE <ВырC1>
[DEFAULT <Выр>]
[SIZE <ВырN>, <ВырN1>
[, <ВырN2>]]
[ENABLE | DISABLE]
[MESSAGE <ВырC2>]
[VALID <ВырL>]
[WHEN <ВырL1>]

Описание

На экране текстовая кнопка представляет собой строку текста заключенную в угловые скобки. Например:

< OK > <Cancel>

Строка текста, часто называемая приглашением, задается в предложениях FUNCTION или PICTURE. Обычно текстовые кнопки используются для инициализации действий. Производимое действие задается опцией VALID. Текстовая кнопка инициализируется командой READ.

Опции

<row,column> (<строка,столбец>)

Расположение первой текстовой кнопки на экране или в активном окне задается парой <row,column>. Координаты <row,column> задаются численными выражениями. Row может принимать значения от 0 до максимального числа строк на экране или в активном окне. Column может принимать значения от 0 до максимального числа столбцов на экране или в активном окне. Строки номеруются сверху-вниз, а столбцы слева-направо.

GET <var>

При нажатии текстовой кнопки ваш выбор заносится в <var>. <var> может быть переменной памяти, элементом массива или полем базы данных. <var> должна иметь числовой или символьный тип. Если <var> имеет числовой тип, то в <var> заносится порядковый номер нажатой вами текстовой кнопки. Например, если вы создали 4 текстовые кнопки и нажали третью, то в <var> будет 3. Если <var> имеет символьный тип, то в <var> заносится приглашение нажатой кнопки.

FUNCTION <ВырC> | *PICTURE* <ВырC1>

При создании текстовых кнопок вы обязательно должны включать опцию FUNCTION, PICTURE или обе. FUNCTION или PICTURE содержат код спецификации текстовых кнопок (он указывает на тип определяемой кнопки) и текст приглашения для каждой кнопки. Код * (звездочка) является кодом спецификации текстовых кнопок.

Символьное выражение <ВырC> в опции FUNCTION должно всегда начинаться со *. Текст приглашения каждой текстовой кнопки задается в списке, следующим через пробел за кодом звездочка. Приглашения отделяются друг от друга точкой с запятой.

Для каждого приглашения создается одна кнопка. Например в этом предложении создаются текстовые кнопки OK и Cancel:

... FUNCTION '* OK;Cancel' SIZE 1, 8 ...

Выражение <ВырС1> в опции PICTURE имеет тот же синтаксис, что и выражение в FUNCTION, за исключением того, что выражение в PICTURE должно обязательно начинаться с АТ символа (@) со следующим за ним кодом *. Например в этом предложении создаются текстовые кнопки OK и Cancel:

... PICTURE '@* OK;Cancel' SIZE 1, 8 ...

Вы также можете включать обе опции FUNCTION и PICTURE для создания текстовых кнопок. Если включаются обе опции, символьное выражение <ВырС> в FUNCTION должно содержать * для создания текстовой кнопки, и может включать через пробел от звездочки сами тексты приглашений кнопок. Символьное выражение <ВырС1> в PICTURE может включать приглашения для создания дополнительных текстовых кнопок.

Опции N, T, H и V в FUNCTION и PICTURE

Непосредственно после кода *I в предложениях FUNCTION и PICTURE монут присутствовать опции, определяющие поведение (опции N и T) и внешний вид (опции H и V) текстовых кнопок. Эти опции приведены ниже:

Опция	Описание
N	Не завершает выполнение READ после выбора текстовой кнопки.
T	Завершает выполнение READ после выбора текстовой кнопки. Устанавливается по умолчанию.
H	Располагать текстовые кнопки по горизонтали.
V	Располагать текстовых кнопки по вертикали. Устанавливается по умолчанию.

Вы можете комбинировать опции H и V с опциями N и T, как NH, NV, YH или TV. Например, это предложение определяет горизонтальный набор кнопок, который не завершает работу команды READ после выбора:

... FUNCTION '*NH ...'

Текстовые кнопки со специальными возможностями

Вы можете присваивать специальные характеристики текстовым кнопкам. Принудительно вы можете назначать "горячие" клавиши, заблокировать некоторые кнопки, определить кнопку выбора по Escape и кнопку выбора по умолчанию. Для этого вам необходимо лишь включить специальные символы при определении приглашения. При записи приглашения в <var> все специальные символы из него удаляются.

"Горячие" клавиши

"Горячая" клавиша представляет собой подсвеченную букву, нажатие которой вызывает немедленный выбор соответствующей кнопки. Для присвоения "горячей" клавиши перед требуемым символом в приглашении соответствующей кнопки установите обратный слеш и знак меньше (\<).

Внимание !!!

"Горячей" клавишей не выбирается требуемая кнопка, если текущим полем является область модификации текста или поле GET. В этом случае нажатие "горячей" клавиши вызовет ввод соответствующего ей символа в текущее поле.

В следующем примере создаются кнопки OK и Cancel с горячими клавишами O и C соответственно:

```
STORE 1 TO choice
@ 4,2 GET choice FUNCTION '* \<OK;\<Cancel'
```

READ

Блокировка кнопок

Заблокированные текстовые кнопки не могут быть нажаты. Они выводятся цветом блокировки. Для блокирования текстовой кнопки достаточно поставить перед ее приглашением два обратных слеша \\. Символ \\ блокируют только одну кнопку. Для блокировки всего набора текстовых кнопок смотри описание опции DISABLE ниже.

В этом примере блокируется текстовая кнопка ОК:

```
STORE 1 TO choice
@ 4,2 GET choice FUNCTION '* \\OK;Cancel'
READ
```

Кнопки по умолчанию

Кнопка по умолчанию автоматически выбирается при нажатии комбинации Ctrl+Enter. На экране кнопка по умолчанию отличается от всех остальных кнопок тем, что она заключается в двойные угловые скобки <<...>>. Для создания кнопки по умолчанию перед соответствующим приглашением ставится обратный слеш и восклицательный знак (\!). На одну команду READ может определяться не более одной текстовой кнопки по умолчанию.

В этом примере кнопка ОК определяется текстовой кнопкой по умолчанию.

```
STORE 1 TO choice
@ 4,2 GET choice FUNCTION '* \!OK;Cancel'
READ
```

Кнопки Escape

Кнопка Escape выбирается автоматически при нажатии клавиши Escape. Такой прием часто используется для обеспечения быстрого выхода из текущей программы. Для создания кнопки Escape перед соответсвующим ей приглашением ставится обратный слеш и знак вопроса (\?). На одну команду READ может определяться не более одной текстовой кнопки Escape.

В этом примере кнопка Cancel определяется текстовой кнопкой Escape.

```
STORE 1 TO choice
@ 4,2 GET choice FUNCTION '* OK;\?Cancel'
READ
```

Кнопки со смешанными характеристиками

Вы можете определять кнопки с более чем одной специальной характеристикой. В следующем примере кнопка ОК является кнопкой по умолчанию и имет "горячую" клавишу О. Кнопка Cancel является кнопкой Escape и имеет "горячую" клавишу С.

```
STORE 1 TO choice
@ 4,2 GET choice FUNCTION '* \!\<OK;\?\<Cancel'
READ
```

DEFAULT <Выр>

При нажатии текстовой кнопки ваш выбор записывается в переменную памяти, элемент массива или поле базы данных. Если вы задаете <var> как переменную памяти и она не существует, то при включении предложения DEFAULT эта переменная памяти автоматически создается и инициализируется. Если <var> является элементом массива или полем, то DEFAULT не создает <var>.

Внимание !!!

Если опция DEFAULT не включена и <var> не существует, то выдается предупреждение об ошибке 'Переменная <var> не найдена'('Variable <var> not found').

Если опция DEFAULT присутствует и <var> существует, то DEFAULT игнорируется.

Выражение <Выр> в опции DEFAULT задает тип и начальное значение инициализируемой переменной памяти. <Выр> должна иметь числовой или символьный тип.

SIZE <ВырN>,<ВырN1>[,<ВырN2>]

Выражения <ВырN> задает высоту объекта. Текстовая кнопка всегда имеет высоту в одну строку, поэтому числовое выражение <ВырN> игнорируется.

По умолчанию ширина каждой текстовой кнопки определяется длиной ее приглашения. Изменяя числовое выражение <ВырN1> вы можете изменять ширину (в столбцах) текстовой кнопки. Если ширина, задаваемая <ВырN1>, меньше ширины приглашения текстовой кнопки, то приглашения не усекается.

По умолчанию отсутствует промежуток между вертикальными кнопками, а горизонтальные кнопки располагаются через один столбец. Расстояние между текстовыми кнопками может задаваться выражением <ВырN2>. При создании вертикальных кнопок, <ВырN2> определяет число строк между ними. При создании горизонтальных кнопок, <ВырN2> определяет число столбцов между ними.

READ

ENABLE|DISABLE

По умолчанию текстовые кнопки становится доступным при использовании команды READ. Вы можете захотеть задержать возможность выбора из этого набора до выполнения определенных условий. Включение опции DISABLE предотвращает активацию набора текстовых кнопок при выполнении команды READ.

Заблокированные текстовые кнопки не могут быть выбраны и отображаются в цвете блокировки. Блокировка одной кнопки в наборе описывается в параграфе "Блокировка кнопок" выше. Для получения доступа к заблокированным командам GET используется команда SHOW GET ENABLED.

@ ... GET — Селективные кнопки

@ <row, column> GET <var>
FUNCTION <ВырC>
< PICTURE <ВырC1>
[DEFAULT <Выр>]
[SIZE <ВырN>, <ВырN1>
[, <ВырN2>]]
[ENABLE | DISABLE]
[MESSAGE <ВырC2>]
[VALID <ВырL>]
[WHEN <ВырL1>]

Эта разновидность команды @ ... GET используется для создания селективных кнопок. Селективные кнопки напоминают кнопки выбора диапазонов автомобильного приемника — выбирая одну кнопку вы делаете ее текущей и тем самым отпускаете нажатую до этого. Знак • (точка) указывает на выбранную в настоящий момент кнопку.

На экране селективные кнопки представляют собой следующее:

(•) Apples(Яблоки)
( ) Apricots (Абрикосы)
( ) Lemons(Лимоны)
( ) Oranges (Апельсины)

Строка текста справа от кнопки называется приглашением. Текст приглашения задается предложениями FUNCTION или PICTURE. Селективная кнопка инициализируется командой READ.

Опции

<row,column> (<строка,столбец>)

Расположение первой селективной кнопки набора на экране или в активном окне задается парой <row,column>. Координаты <row,column> задаются численными выражениями. Row может принимать значения от 0 до максимального числа строк на экране или в активном окне. Column может принимать значения от 0 до максимального числа столбцов на экране или в активном окне. Строки номеруются сверху-вниз, а столбцы слева-направо.

GET <var>

При нажатии селективной кнопки ваш выбор заносится в <var>. <var> может быть переменной памяти, элементом массива или полем базы данных. <var> должна иметь числовой или символьный тип. Если <var> имеет числовой тип, то в <var> заносится порядковый номер нажатой вами селективной кнопки. Например, если вы создали 4 селективные кнопки и нажали третью, то в <var> будет 3. Если <var> имеет символьный тип, то в <var> заносится приглашение нажатой кнопки.

FUNCTION <ВырС> | *PICTURE* <ВырС1>

При создании селективных кнопок вы обязательно должны включать опцию FUNCTION, PICTURE или обе. FUNCTION или PICTURE содержат код спецификации селективных кнопок (он указывает на тип определяемой кнопки) и текст приглашения для каждой кнопки. Код *R является кодом спецификации селективных кнопок.

Символьное выражение <ВырС> в опции FUNCTION должно всегда начинаться со *R. Текст приглашения каждой селективной кнопки задается в списке, следующим через пробел за кодом *R. Приглашения отделяются друг от друга точкой с запятой. Для каждого приглашения создается одна кнопка. Например в этом предложении создаются селективные кнопки Apples, Apricots, Lemons и Oranges:

... FUNCTION '*R Apples;Apricots;Lemons;Oranges' ...

Выражение <ВырС1> в опции PICTURE имеет тот же синтаксис, что и выражение в FUNCTION, за исключением того, что выражение в PICTURE должно обязательно начинаться с AT символа (@) со следующим за ним кодом *R. Например в этом предложении создаются селективные кнопки Apples, Apricots, Lemons и Oranges:

... PICTURE '@*R Apples;Apricots;Lemons;Oranges' ...

Вы также можете включать обе опции FUNCTION и PICTURE для создания селективных кнопок. Если включаются обе опции, символьное выражение <ВырС> в FUNCTION должно содержать *R для создания селективной кнопки, и может включать через пробел от звездочки сами тексты приглашений кнопок. Символьное выражение <ВырС1> в PICTURE может включать приглашения для создания дополнительных селективных кнопок.

Опции N, T, H и V в FUNCTION и PICTURE

Непосредственно после кода *I в предложениях FUNCTION и PICTURE могут присутствовать опции, определяющие поведение (опции N и T) и внешний вид (опции H и V) селективных кнопок. Эти опции приведены ниже:

Опция	Описание
N	Не завершает выполнение READ после выбора селективной кнопки. Устанавливается по умолчанию.
T	Завершает выполнение READ после выбора селективной кнопки.
H	Располагать селективные кнопки по горизонтали.
V	Располагать селективные кнопки по вертикали. Устанавливается по умолчанию.

Вы можете комбинировать опции H и V с опциями N и T, как NH, NV, YH или TV. Например, это предложение определяет вертикальный набор кнопок, который не завершает работу команды READ после выбора:

... FUNCTION '*RNV ...'

Селективные кнопки со специальными возможностями

Вы можете присваивать специальные характеристики селективным кнопкам. Принудительно вы можете назначать "горячие" клавиши или блокировать отдельные кнопки, путем включения специальных символов в текст приглашения соответсвующей кнопки. При записи приглашения в <var> все специальные символы из него удаляются.

"Горячие" клавиши

"Горячая" клавиша представляет собой подсвеченную букву, нажатие которой вызывает немедленный выбор соответствующей кнопки. Для присвоения "горячей" клавиши перед требуемым символом в приглашении соответствующей кнопки установите обратный слеш и знак меньше (\<).

Внимание !!!

"Горячей" клавишей не выбирается требуемая кнопка, если текущим полем является область модификации текста или поле GET. В этом случае нажатие "горячей" клавиши вызовет ввод соответствующего ей символа в текущее поле.

В следующем примере определяются селективные кнопки Apples, Apricots, Lemons и Oranges с "горячими" клавишами A для Apples и P для Apricots:

```
STORE 1 TO choice
@ 4,2 GET choice FUNCTION '*R' ;
     PICTURE ' \<Apples;A\<pricots;Lemons;Oranges'
READ
```

Блокировка кнопок

Заблокированные селективные кнопки не могут быть нажаты. Они выводятся цветом блокировки. Для блокирования селективной кнопки достаточно поставить перед ее приглашением два обратных слеша (\\). Символ \\ блокируют только одну кнопку. Для блокировки всего набора селективных кнопок смотри описание опции DISABLE ниже.

DEFAULT <Выр>

При нажатии селективной кнопки ваш выбор записывается в переменную памяти, элемент массива или поле базы данных. Если вы задаете <var> как переменную памяти и она не существует, то при включении предложения DEFAULT эта переменная памяти автоматически создается и инициализируется. Если <var> является элементом массива или полем, то DEFAULT не создает <var>.

Внимание !!!

Если опция DEFAULT не включена и <var> не существует, то выдается предупреждение об ошибке 'Переменная <var> не найдена'('Variable <var> not found').

Если опция DEFAULT присутствует и <var> существует, то DEFAULT игнорируется.

Выражение <Выр> в опции DEFAULT задает тип и начальное значение инициализируемой переменной памяти. Оно должно иметь числовой или символьный тип.

SIZE <ВырN>,<ВырN1>[,<ВырN2>]

Выражения <ВырN> задает высоту объекта. Селективная кнопка всегда имеет высоту в 1 строку, поэтому числовое выражение <ВырN> игнорируется. Но <ВырN> должна обязательно присутствовать, если вы хотите задать <ВырN1> и <ВырN2>.

По умолчанию ширина каждой селективной кнопки определяется длиной ее приглашения. Изменяя числовое выражение <ВырN1> вы можете изменять ширину (в

столбцах) селективной кнопки. Если ширина, задаваемая <ВырN1>, меньше ширины приглашения селективной кнопки, то приглашения не усекается.

По умолчанию отсутствует промежуток между вертикальными кнопками, а горизонтальные кнопки располагаются через один столбец. Расстояние между селективными кнопками может задаваться выражением <ВырN2>. При создании вертикальных кнопок, <ВырN2> определяет число строк между ними. При создании горизонтальных кнопок, <ВырN2> определяет число столбцов между ними.

ENABLE|DISABLE

По умолчанию селективные кнопки становится доступным при использовании команды READ. Вы можете задержать возможность выбора из этого набора кнопок до выполнения определенных условий. Включение опции DISABLE предотвращает активацию набора селективных кнопок при выполнении команды READ.

Заблокированные селективные кнопки не могут быть выбраны и отображаются в цвете блокировки. Блокировка одной кнопки в наборе описывается выше, в параграфе "Блокировка кнопок". Для получения доступа к заблокированным командам GET используется команда SHOW GET ENABLED.

@ ... EDIT — Области модификации текста

@ <row, column> EDIT <var>
[FUNCTION <ВырC1>]
[DEFAULT <Выр>]
SIZE <ВырN>, <ВырN1>
[, <ВырN2>]
[ENABLE | DISABLE]
[MESSAGE <ВырC2>]
[VALID <ВырL>
[ERROR <ВырC3>]]
[WHEN <ВырL1>]
[NOMODIFY]
[SCROLL]
[TAB]

Эта команда создает прямоугольную область модификации текста для редактирования <var>. <var> может быть переменной памяти, элементом массива переменных памяти, полем базы данных или memo полем. <var> должна иметь символьный тип. В этой области доступны все стандартные для FoxPro возможности редактирования: вырезание, копирование и вставка. Текст в области модификации может прокручиваться по вертикали. По горизонтали осуществляется автоматический переход на новую строку. В отличии от остальных органов управления FoxPro (блоков проверки, селективных и текстовых кнопок, ...), эта команда в своем синтаксисе использует слово EDIT, вместо GET.

При активации области модификации текста командой READ, содержимое <var> отображается в этой области для редактирования. При выходе из области модификации текста, его содержимое снова возвращается в <var>.

По умолчанию, для сохранения изменений достаточно нажать клавиши Tab или Ctrl+Tab. Выходя таким образом вы сохраняете все сделанные изменения и перемещаетесь к следующему полю GET или следующему объекту.

Если клавиша Tab уже определена, то вместо нее используется Ctrl+Tab. Более подробная информация о ключевом слове TAB приводиться ниже в этой главе.

Для выхода из области модификации текста без сохранения изменений используется клавиша Escape. Заметим, однако, что если вы редактируете memo поле и задана опция VALID, нажатие этой клавиши может сохранить или не сохранить изменения, в зависимости от значения, возвращаемого VALID. Более подробная информация о VALID приведена далее в этой части.

<row,column> (<строка,столбец>)

Верхний левый угол области модификации текста располагается на экране или в активном окне по адресу, определяемому парой <row,column>. Координаты <row,column> задаются численными выражениями. Row может принимать значения от 0 до максимального числа строк на экране или в активном окне. Column может принимать значения от 0 до максимального числа столбцов на экране или в активном окне. Строки номеруются сверху-вниз, а столбцы слева-направо.

EDIT <var>

При редактировании области текста сам текст читается и сохраняется в <var>. <var> может быть переменной памяти, элементом массива переменных памяти или полем базы данных. <var> должна иметь символьный тип.

FUNCTION <ВырС1>

В предложение FUNCTION может включаться одна из опций выравнивания текста в области редактирования. Любые другие символы в FUNCTION будут игнорироваться. Эти опции следующие:

I — центрирование текста в области модификации.

J — выравнивание текста по правому краю. По умолчанию текст в области модификации выравнивается по левому краю.

При включении обеих опций, опция J имеет приоритет.

DEFAULT <Выр>

При редактировании области текста сам текст читается и сохраняется в переменной памяти, элементе массива, поле базы данных или memo поле. Если вы задаете <var> как переменную памяти и она не существует, то при включении предложения DEFAULT эта переменная памяти автоматически создается и инициализируется. Если <var> является элементом массива или полем, то DEFAULT не создает <var>.

Внимание !!!

Если опция DEFAULT не включена и <var> не существует, то выдается предупреждение об ошибке 'Переменная <var> не найдена'('Variable <var> not found').

Если опция DEFAULT присутствует и <var> существует, то DEFAULT игнорируется.

Выражение <Выр> в опции DEFAULT задает тип и начальное значение инициализируемой переменной памяти. <Выр> должна иметь символьный или memo тип.

SIZE <ВырN>,<ВырN1>[,<ВырN2>]

Опция SIZE обязательно должна присутствовать; она задает высоту и ширину области модификации текста. Высота области модификации в строках задается в <ВырN>, ширина в столбцах — в <ВырN1>. Если <ВырN> равно 1, то создается специальная одностроковая область. В нем вы можете осуществлять горизонтальную прокрутку. Нажатием клавиши Enter вы переходите к следующему полю GET или следующему объекту. Необязательное числовое выражение <ВырN2> задает число символов, которые можно редактировать, начиная с первого символа в <var>. Если <ВырN2> опущено, то вся переменная <var> может редактироваться.

ENABLE|DISABLE

По умолчанию области модификации текста становятся доступными при использовании команды READ. Вы можете захотеть задержать возможность выбора области модификации текста до выполнения определенных условий. Включение опции DISABLE размещает область модификации текста на экране или в окне, но предотвращает ее активацию командой READ. Заблокированная область модификации текста не может быть выбрана и отображается в цвете блокировки. Включение опции ENABLE активизирует область модификации текста так, что она может быть выбрана. Для получения доступа к заблокированным командам GET используется команда SHOW GET ENABLED.

NOMODIFY

Включая необязательную опцию NOMODIFY вы показываете область модификации текста, но не разрешаете ее редактирование. Вы можете перемещаться по тексту, копировать текст из области, но вам запрещены любые его изменения.

SCROLL

Если переменная <var> слишком велика, чтобы уместится в области модификации текста, вы можете осуществлять ее прокрутку. Для этого используются клавиши управления курсором, PgUp, PgDown, Home или End и их комбинации с клавишей Ctrl. Для перемещения по тексту можно пользоваться и "мышью".

Если опция SCROLL присутствует, то справа от области модификации текста выводятся полосы прокрутки. Полосы прокрутки позволяют осуществлять более быстрое перемещение по тексту с помощью "мыши" и визуально указывают на ваше местоположение в области редактирования. Полосы прокрутки появляются только в случае, если текст является слишком большим, чтобы полностью поместится в области модификации текста.

TAB

По умолчанию, при нахождении в области модификации текста нажатие клавиши Tab не вставляет в текст символа табуляции. Нажатие Tab (или Ctrl+Tab) сохраняет все ваши изменения, деактивирует область модификации текста и переходит к следующему полю GET или следующему объекту. Однако, если вы включаете ключевое слово TAB, то нажатие этой клавиши вставляет символ табуляции по месту расположения курсора. Для сохранения все ваших изменений, деактивции области модификации текста и перехода к следующему полю GET или следующему объекту, в этом случае, вы должны нажать комбинацию клавиш Ctrl+Tab.

@ ... BOX

@ <row1, column1>, <row2, column2> BOX [<ВырС>]

Данная форма команды @ предназначается для рисования прямоугольника на экране или в окне.

<row1>,<column1>,<row2>,<column2>

<row1,column1> — это координаты верхнего левого угла прямоугольника, а <row2,column2> — координаты его правого нижнего угла. При одинаковых значениях <row1> и <row2> рисуется горизонтальная линия, а при одинаковых значениях <column1> и <column2> — вертикальная линия.

BOX <ВырС>

Включив опцию <ВырС>, можно задать символы, которыми будет изображен прямоугольник. <ВырС> может содержать до девяти различных символов — по одному для каждого угла, по одному для каждой стороны и один символ для заполнения им прямоугольника. Эти символы выводятся, начиная с верхнего левого угла, и далее по

часовой стрелке. Если девятый символ задан, он будет использован для заполнения внутренней области прямоугольника. Если задан лишь один символ, то он будет использован для рисования всего контура границы прямоугольника. Если же <ВырС> отсутствует, то прямоугольник будет нарисован одинарной линией.

@ ... CLEAR

@ <row1, column1> [CLEAR | CLEAR TO <row2, column2>]

Данная форма команды @ выполняет очистку области экрана или активного окна.

<row1, column1>

Координаты верхнего левого угла очищаемого экрана или активного для вывода окна.

[*CLEAR* | *CLEAR TO* <row2, column2>]

Если опцию CLEAR или CLEAR TO опустить, то произойдет очистка строки row1, начиная со столбца column1.

При задании опции CLEAR будет выполнена очистка прямоугольной области экрана, верхний левый угол которой определен координатой <row1,column1>, а правый нижний является правым нижним углом экрана или окна. При использовании опции CLEAR TO очищается прямоугольная область экрана, как было описано выше, за исключением того, что правый нижний угол прямоугольника будет определен второй парой координат <row2,column2>.

- APPEND BLANK — добавляет пустую запись в конец текущей базы данных.
- APPEND FROM <file>

[FIELDS <field list>]
[FOR <ВырL>]
[TYPE] [DELIMITED [WITH TAB | WITH <delimiter> | WITH BLANK]
| DIF | FW2 | MOD | PDOX | RPD | SDF | SYLK
| WK1 | WK3 | WKS | WR1 | WRK | XLS]

Добавление записей в конец базы данных из другого файла <file>. Если этот файл не является файлом базы данных FoxPro, то вы должны задать тип TYPE файла, из которого производится добавление.

FIELDS <field list>

Команда APPEND FROM поддерживает необязательно задаваемый список полей <field list>. Такая форма команды позволяет добавлять к базе данных только поля с указанными в нем именами.

FOR <ВырL>

Если в команду включено предложение FOR <ВырL>, то новые записи добавляются только из тех записей файла, указанного в предложении FROM <file>, для которых <ВырL> принимает значение логической "истины". Добавление продолжается то тех пор, пока не встретится конец FROM-файла. Если предложение FOR опущено, то будут добавлены все записи, пока FROM-файл не кончится.

TYPE

Это предложение должно использоваться, если FROM-файл не является базой данных FoxPro. Предложение TYPE служит для задания других типов файлов. Вы можете использовать команду APPEND FROM с широким набором различных типов файлов, включая разделенные специальными символами текстовые файлы ASCII.

При добавлении из файлов, не имеющих обычное расширение по умолчанию, вы должны задавать расширение файла явно. Например, для электронных таблиц Lotus 1-2-3 версий 2.x обычно используется файл с расширением .WK1. При добавлении из файла

электронной таблицы с расширением, отличным от .WK1, вы должны явно указывать это расширение.

DELIMITED [WITH TAB | WITH <delimiter> | WITH BLANK]

Файл типа DELIMITED представляет собой текстовый ASCII-файл, каждая запись которого заканчивается возвратом каретки и переводом строки. Поля обычно разделяются запятыми, а символьные поля дополнительно берутся в двойные кавычки. Например, "555", 9999999, "TELEPHONE". Однако, для задания файлов, которые должны содержать поля, разделяемые не запятыми, а одним пробелом или символом табуляции, можно использовать опции DELIMITED WITH BLANK или DELIMITED WITH TAB соответсвенно, а для указания на то, что символьные поля разделяются не двойными кавычками, а какими-либо другими символами, может использоваться опция DELIMITED WITH <delimiter> (<разделитель>). Для всех файлов DELIMITED предполагается использование расширения .TXT.

Вы можете импортировать даты из разделенных файлов, если только они в нем находятся в нужном формате. По умолчанию они имеют формат 'mm/dd/yy'. Позиция столетия не является обязательной — FoxPro при импорте автоматически включает номер столетия. Если столетие в дате не задано (например '12/25/91'), то предполагается двадцатое столетие.

Разделителями в дате могут быть любые нечисловые символы, за исключением разделителя, разделяющего поля в разделенном файле.

Даты других форматов могут импортироваться без всяких ограничений, если их формат соответствует доступному формату даты в SET DATE. Для импорта дат не в формате по умолчанию, перед использованием APPEND FROM задайте нужный формат командой SET DATE. Для проверки успешности импортирования дат, используйте функцию CTOD(). Если даты обрабатываются этой функцией, то они будет импортированы так как нужно.

DIF

В случае DIF файла (Data Interchange Format (Формат Обмена Данными) используемый VisiCalc) вектора (столбцы) становятся полями, а кортежи (строки) становятся записями. Предполагается, что DIF файлы имеют расширение .DIF.

FW2

FW2 файлы, создаваемые Framework II. Предполагается, что FW2 файлы имеют расширение .FW2.

MOD

MOD файлы, создаваемые Multiplan версии 4.01 фирмы Microsoft. Предполагается, что MOD файлы имеют расширение .MOD.

PDOX

Файлы базы данных Paradox версии 3.5 фирмы Borland могут добавлятся в файлы базы данных FoxPro при включении этой опции. Предполагается, что файлы Paradox имеют расширение .DB.

RPD

RPD файлы создаются RapidFile версии 1.2. Предполагается, что RPD файлы имеют расширение .RPD.

SDF

Файл SDF (System Data Format (Системный Формат Данных)) представляет собой текстовый ASCII-файл, в котором все записи имеют фиксированную длину и заканчиваются возвратом каретки и переводом строки. Поля не разделяются. Для файлов формата SDF предполагается расширение .TXT.

SYLK

SYLK файлы имеют формат обмена Symbolic Link (используются в MultiPlan фирмы Microsoft). При импорте из них столбцы становятся полями, а строки записями. SYLK файлы не имеют расширения по умолчанию.

WK1

Эта опция используется при импорте из электронных таблиц Lotus 1-2-3. В базе данных FoxPro столбцы электронной таблицы становятся полями, а ее строки становятся записями. Для электронных таблиц, созданных Lotus 1-2-3 версии 2.x предполагается расширение .WK1.

WK3

Эта опция используется при импорте из электронных таблиц Lotus 1-2-3. В базе данных FoxPro столбцы электронной таблицы становятся полями, а ее строки становятся записями. Для электронных таблиц, созданных Lotus 1-2-3 версии 3.x предполагается расширение .WK3.

WKS

Эта опция используется при импорте из электронных таблиц Lotus 1-2-3. В базе данных FoxPro столбцы электронной таблицы становятся полями, а ее строки становятся записями. Для электронных таблиц, созданных Lotus 1-2-3 версии 1-A предполагается расширение .WKS.

WR1

Эта опция используется при импорте из электронных таблиц Lotus Symphony. В базе данных FoxPro столбцы электронной таблицы становятся полями, а ее строки становятся записями. Для электронных таблиц, созданных Lotus Symphony версии 1.1 или 1.2 предполагается расширение .WR1.

WRK

Эта опция используется при импорте из электронных таблиц Lotus Symphony. В базе данных FoxPro столбцы электронной таблицы становятся полями, а ее строки становятся записями. Для электронных таблиц, созданных Lotus Symphony версии 1.0 предполагается расширение .WRK.

XLS

Эта опция используется при импорте из электронных таблиц Microsoft Excel версии 2.0. В базе данных FoxPro столбцы электронной таблицы становятся полями, а ее строки становятся записями. Для электронных таблиц, созданных Excel предполагается расширение .XLS.

APPEND FROM ARRAY

■ APPEND FROM ARRAY <array> — добавляет к выбранной на текущий момент базе данных запись из массива переменных памяти, игнорируя поля памяти memo.

Новые записи добавляются столько раз, сколько строк в массиве.

Одномерные массивы

В случае одномерного массива содержимое его первого элемента станет первым полем новой добавляемой записи, содержимое второго элемента станет вторым полем, и так далее.

В том случае, если одномерный массив имеет больше элементов, чем база данных полей, то все лишние элементы игнорируются. Если же база данных имеет больше полей, чем массив элементов, то все лишние поля инициализируются значениями по умолчанию. В таблице приводятся значения по умолчанию для различных типов полей:

Тип поля Значение по умолчанию

СимвольныйПробелы
Числовой 0
Даты Пустая дата
Логический ложь (.F.)

Двумерные массивы

При добавлении данных из двумерного массива новая запись добавляется для каждой его строки. Например, если массив имеет четыре строки, то в базу данных будут добавлены четыре новые записи. Содержимое первого столбца массива становится первым полем новой добавляемой записи, содержимое второго столбца становится вторым полем, и т.д. Например, если массив имеет четыре строки и три столбца, то элементы первого столбца массива будут первым полем четырех добавленных записей.

Если двумерный массив имеет больше элементов, чем база данных полей, то оставшиеся столбцы массива игнорируются. Если же база данных имеет больше полей, чем массив столбцов, то оставшиеся поля инициализируются так же, как это было описано выше для одномерных массивов.

Соответствие типов данных

FoxPro способен заполнять поля базы данных даже при несоответствии типа данных элемента массива и поля базы данных. В некоторых случаях поле заменяется элементом массива, в других, оно инициализируется значением по умолчанию. Смотри таблицу значений по умолчанию для поля каждого типа.

Следующая таблица иллюстрирует в каких случаях происходит замена при несоответствии типов данных. "Да" указывает на производимую замену, "Нет" указывает, что поле инициализируется значением по умолчанию.

Тип элемента	Тип поля			
	Символьный	Числовой	Даты	Логический
Символьный	Да	Нет	Нет	Нет
Числовой	Да	Да	Нет	Да
Даты	Да	Да	Да	Нет
Логический	Да	Нет	Нет	Да

Например, если элемент массива содержит значение логической истины (.T.) или лжи (.F.), и соответствующее ему поле в только что добавленной записи имеет символьный тип, FoxPro заменит это поле на T или F соответственно.

COPY TO ARRAY

Пересылка данных из записи базы данных в массив

■ COPY TO ARRAY <array>
[FIELDS <field list>]
[<scope>]
[FOR <ВырL>]
[WHILE <ВырL1>]

Команда COPY TO ARRAY пересылает данные из текущей выбранной базы данных в массив переменных памяти. Команды COPY TO ARRAY и SCATTER аналогичны. COPY TO ARRAY позволяет переслать в массив нескоько записей, в то время как SCATTER пересылает в массив или набор переменных памяти одну запись базы данных. COPY TO ARRAY требует, чтобы массив был сначала создан командами DIMENSION, DECLARE или PUBLIC. Команда же SCATTER в случае, если массив или набор переменных ранее определены не были, сама создает их.

Для копирования одной записи вы должны использовать одномерный массив. Массив должен иметь элементов, не менее, чем полей базы данных, отличных от полей памяти memo. Поля памяти игнорируются командой COPY TO ARRAY.

Для копирования нескольких записей или всей базы данных (если вы имеете достаточно памяти) используйте двумерные массивы. Первый индекс массива (строки) указывает на номер хранимой записи, второй (столбцы) указывает на номер поля.

В случае одномерного массива первое поле записи помещается в первый элемент массива, второе поле во второй элемент массива, и т.д. Если одномерный массив имеет больше элементов, чем база данных имеет полей, то оставшиеся элементы массива не изменяются. Если же массив имеет меньше элементов, чем база данных полей, то оставшиеся поля игнорируются.

Двумерные массивы создаются в построчно-столбцовом формате. В случае команды COPY TO ARRAY запись помещается в строку массива, а каждое поле записи при этом занимает отдельный столбец. Для каждой записи первое поле хранится в первом столбце массива, второе поле во втором столбце массива, и т.д. Если массив имеет больше столбцов, чем база данных полей, то оставшиеся элементы массива не изменяются. Если же массив имеет меньше столбцов, чем база данных полей, то оставшиеся поля в массив записны не будут.

Каждая последующая строка массива заполняется содержимым следующей записи в описанном выше построчно-столбцовом формате. Работа команды заканчивается тогда, когда либо в массиве не осталось свободных строк, либо в базе данных кончились записи.

Данные можно переслать назад из массива в записи базы данных при помощи команды APPEND FROM ARRAY. Команда GATHER позволяет пересылку данных как из массива, так и из переменных памяти.

<array>

Массив переменных памяти в который производится копирование.

FIELDS <field list> (<список полей>)

Включив предложение FIELDS <field list>, вы можете задать, какие именно поля будут скопированы в массив <array>. Если же предложение FIELDS опущено, то в массив будут скопированы все поля.

■ COUNT [TO <memvar>] — подсчитывает число записей в переменную памяти <memvar>.

CREATE LABEL

CREATE LABEL [<file> | ?]
[[WINDOW <Окно1>]
[IN [WINDOW] <Окно2> | IN SCREEN]]

Данная команда активирует утилиту FoxPro, позволяющую генерировать этикетки. Она позволяет легко создавать и распечатывать любые этикетки. Данная утилита позволяет создание этикеток в стандартном этикеточном формате, либо позволяет разработку собственных форматов пользователя. Вы сможете до печати посмотреть, как буддет вы-

глядеть созданная вами этикетка, воспользовавшись средством предварительного просмотра Page Preview.

Вызов команды CREATE LABEL без дополнительных аргументов открывает новое окно редактирования этикеток. Этикетке присваивается имя UNTITLED. При выходе из окна редактирования этикеток вы можете сохранить определение этикетки в файле под определенным именем.

Опции

<file> | ?

Если в команде задано имя <file>, то этикетка будет записана в файл определения метки с указанным вами именем. Если не задавать расширения имени этого файла, ему будет автоматически назначено расширение .LBX. При включении необязательного предложения ? появляется диалоговое окно Open File, в котором предлагаются для выбора существующие файлы этикеток. В этом окне можно задать и имя нового файла этикетки.

WINDOW <Окно1>

При включении опции WINDOW <Окно1> окно создания этикеток возьмет все свои характеристики из окна <Окно1>. Например, если окно <Окно1> было определено командой DEFINE WINDOW с опцией FLOAT, то и окно создания этикеток будет перемещаемым. Окно <Окно1> должно быть предварительно определено, но не обязательно должно быть активным или видимым.

Окно создания этикеток может быть, по умолчанию, больше окна <Окно1>. В этом случае оно все равно принимает характеристики окна в котором располагается. Верхний левый угол окна создания этикеток будет иметь те же координаты, что и верхний левый угол окна <Окно1>, но оно может выходить за границы окна <Окно1>.

CREATE QUERY

CREATE QUERY [<file> | ?]

Команда CREATE QUERY открывает окно RQBE для создания интерактивных запросов (команда SQL SELECT). Все что вам необходимо сделать, это установить нужные параметры в окне RQBE и FoxPro все остальное сделает сам.

Команда SELECT используется для осуществления поиска в базе данных. Это очень мощная команда, которая заменяет последовательность из нескольких команд FoxPro. Т.к. одна SELECT равносильна нескольким командам FoxPro, то ее использование оптимизирует выполнение программы.

Команда SELECT обеспечивает механизм задания запроса FoxPro на получение нужной информации из базы данных. Она позволяет задавать требуемую информацию без указания FoxPro как ее искать. FoxPro сам определяет наилучший способ поиска этой информации.

После создания запроса, он хранится как простой программный файл FoxPro с расширением .QPR. Программа запроса может быть выполнена с помощью команды DO.

Вызов команды CREATE QUERY без дополнительных параметров открываеет новое окно запросов RQBE. Запросу присваивается имя UNTITLED. При выходе из окна запросов вы можете сохранять запрос в файле с нужным именем.

Более подробная информация о создании запросов приводится в главе "Меню File" ("File Menu") в книге "Руководство по Интерфейсу FoxPro" ("Interface Guide") или в уроке по созданию запросов в книге "НачалоРаботы"(Getting Started").

Опции

<file>

Если в команде задано имя <file>, то информация о запросе будет записана в файл с указанным вами именем. Если не задавать расширения имени этого файла, ему будет автоматически назначено расширение .QPR.

?

При включении необязательного предложения ? появляется диалоговое окно Open File, в котором предлагаются для выбора существующие файлы запросов. В этом окне можно задать и имя нового файла запроса.

CREATE REPORT

CREATE REPORT [<file> | ?]
[[WINDOW <Окно1>]
[IN [WINDOW] <Окно2> | IN SCREEN]]

Команда CREATE REPORT открывает окно построения отчета.

Более подробная информация о создании отчетов приводится в главе "Построитель Отчетов" ("Report Writer") в книге "Руководство по Интерфейсу FoxPro" ("Interface Guide"). Команда CREATE REPORT может также использоваться для генерации быстрого отчета без открытия окна построения отчета. Информация о создании командой CREATE REPORT быстрых отчетов приводится в следующей секции.

<file>

Вызов команды CREATE REPORT без дополнительных параметров открываеет новое окно построения отчетов. Отчету присваивается имя UNTITLED.

При выходе из окна построения отчетов вам предложат сохранять отчет в файле с нужным именем.

Если в команде задано имя <file>, то информация об отчете будет записана в файл с указанным вами именем. Если не задавать расширения имени этого файла, ему будет автоматически назначено расширение .FRX.Если файл отчета с таким именем уже существует, то вы должны будете подтвердить его перезапись (если SAFETY установлена в SET ON).

?

При включении необязательного предложения ? появляется диалоговое окно Open File, в котором предлагаются для выбора существующие файлы отчетов. В этом окне можно задать и имя нового файла отчета.

CREATE SCREEN

■ CREATE SCREEN [<file> | ?]
[WINDOW <Окно1>]
[IN [WINDOW] <Окно2> | IN SCREEN]

Команда CREATE SCREEN открывает окно построения экрана.

Более подробная информация о создании экранов приводится в главе "Построитель Экранов" ("Screen Builder") в книге "Руководство по Интерфейсу FoxPro" ("Interface Guide"). Команда CREATE SCREEN может также использоваться для генерации быстрого экрана без открытия окна построения экрана. Информация о создании командой CREATE SCREEN быстрых экранов приводится в следующей секции.

<file>

Вызов команды CREATE SCREEN без дополнительных параметров открываеет новое окно построения экрана. Экрану присваивается имя UNTITLED. При выходе из окна построения экрана вам предложат сохранять экран в файле с нужным именем.

Если в команде задано имя <file>, то информация об экране будет записана в файл с указанным вами именем. Если не задавать расширения имени этого файла, ему будет автоматически назначено расширение .SCX. Если файл экрана с таким именем уже существует, то вы должны будете подтвердить его перезапись (если SAFETY установлена в SET ON).

?

При включении необязательного предложения ? появляется диалоговое окно Open File, в котором предлагаются для выбора существующие файлы экрана. В этом окне можно задать и имя нового файла экрана.

CREATE TABLE — SQL

■ CREATE TABLE | DBF <dbf_name>
(<fname1> <type> [(<precision> [, <scale>]) [, <fname2> ...]])
| FROM ARRAY <array>

Команда CREATE TABLE создает базу данных. Каждое новое поле базы данных определяется со своим именем, типом, точностью и числом десятичных знаков. Эти определения могут быть получены из самой команды или из массива. Новая база данных открывается исключительно, несмотря на текущую установку SET EXCLUSIVE.

CREATE TABLE | DBF <dbf_name>

В этом предложении <dbf_name> задает имя создаваемой базы данных.

<dbf_name> может включать в себя маршрут и может быть именным выражением.

(<fname1> <type> [(<precision> [, <scale>]) [, <fname2> ...]])

В этом предложении <fname1> и <fname2> имена полей в новой базе данных. Каждое <fname> может быть именным выражением.

<type> является буквой, указывающей на тип данных поля. Некоторый типы данных требуют задания <precision> (ширины поля) и <scale> (числа десятичных знаков). <type>, <precision> и <scale> могут быть следующими:

Тип	Ширина	Число десятичных знаков	Описание
C	n	—	Строка символов шириной n
D	—	—	Дата
F	n	d	С плавающей точкой ширины n с d десятичными разрядами
L	—	—	Логическое
M	—	—	Поле памяти (memo)
N	n	d	Числовое ширины n с d десятичными разрядами
P	—	—	Шаблон

FROM ARRAY <array>

В этом предложении <array> задает имя существующего массива, содержащего имя, тип, ширину и число десятичных знаков в каждом поле базы данных. Содержимое массива может быть задано функцией AFIELDS().

DELETE

■ DELETE — помечает записи текущей выбранной базы данных на удаление.

См. также: PACK, RECALL.

DO

■ DO <file>
[WITH <parm list>]

Команда DO выполняет программу. Файл программы сам может включать команды DO. Уровень вложенности команд DO ограничен глубиной 32. Когда вы используете команду DO для выполнения программы, будут выполняться команды, содержащиеся в выполняющемся файле программы.

Опция WITH позволяет вам передать параметры в программу. Список параметров <parm list> содержит выражения, переменные памяти, литералы, поля баз данных или функции, определенные пользователем. По умолчанию, параметры передаются в программу или процедуру по ссылке. Параметр может передаваться по значению размещением его в круглых скобках. Передача параметров по значениям или по ссылкам в функции, определенные пользователем, обсуждается в описании команды SET UDFPARMS, а более подробную информацию о передаче параметров можно найти в описании команды PARAMETER.

DO CASE

Выполняет набор команд в зависимости от логического условия

■ DO CASE
CASE <ВырL>
<statements>
[CASE <ВырL1>
<statements>
...
CASE <ВырLN>
<statements>]
[OTHERWISE
<statements>]
ENDCASE

Команда DO CASE используется для выполнения набора команд FoxPro (<statements>) в зависимости от результата логического условия. Когда выполняется команда DO CASE, рассматриваются последовательные логические условия (CASE); результат проверки определяет, какой набор команд (если он найдется) будет выполняться.

Когда первый раз встретится истинное (.T.) логическое условие CASE, выполняются команды, следующие за ним. Выполнение команд продолжается до тех пор, пока не встретится следующее ключевое слово CASE или ENDCASE. Затем выполнение продолжается с первой команды после ключевого слова ENDCASE.

Если логическое условие CASE ложно (.F.), команды после него до следующего CASE игнорируются.

Будет выполнены команды только после одного CASE — первого истинного логического условия CASE. Все последующие истинные CASE игнорируются.

Если все логические условия CASE оказались ложными и используется ключевое слово OTHERWISE, выполняются все команды после него, и выполнение затем переходит на команды после ключевого слова ENDCASE.

DO WHILE

Выполняет команды внутри цикла, пока логическое условие остается истинным

■ DO WHILE <ВырL>

<statements>
[LOOP]
[EXIT]
ENDDO

В этой команде структурногопрограммированиявычисляется логическое выражение <ВырL>. Пока <ВырL> имеет значение "истинно" (.T.), выполняется набор команд <statements>, расположенный между ключевыми словами DO WHILE и соответствующим ENDDO. Каждое ключевое слово DO WHILE должно иметь соответствующее ключевое слово ENDDO. Комментарии могут располагаться в той же строке, что DO WHILE и ENDDO; при компиляции и выполнении комментарии будут игнорироваться.

Логическое выражение. Пока логическое выражение <ВырL> остается истинным, выполняются команды между ключевыми словами DO WHILE и соответствующим ENDDO.

<statements>

Команды FoxPro, которые выполняются, пока логическое выражение <ВырL> остается истинным.

LOOP

Ключевое слово LOOP может располагаться в любом месте между DO WHILE и ENDDO. Ключевое слово LOOP возвращает управление назад прямо в DO WHILE.

EXIT

Ключевое слово EXIT передает управление из цикла DO WHILE команде, следующей первой после ключевого слова ENDDO. Ключевое слово EXIT может располагаться в любом месте между DO WHILE и ENDDO.

EXPORT

■ EXPORT TO <file>

Команда EXPORT копирует записи из выбранного в данный момент файла базы данных в новый файл. Это позволяет вам использовать данные из базы данных FoxPro в других пакетах программного обеспечения.

Если экспортируемый файл базы данных был индексирован, новый файл будет создаваться в индексном порядке.

<file>

<file> является новым файлом, в который FoxPro будет экспортировать данные. Если вы не задали в имени файла расширение, назначается стандартное расширение для файла данного типа.

FIELDS <field list>

Если вы включаете опцию FIELDS (Поля) и список полей, то вы должны задать поля, копируемые в новый файл. Если же опция FIELDS опущена, в файл копируются все поля. Поля memo не копируются в новый файл, даже если имя поля memo включается в список полей.

<scope>

Вы можете включить <scope> (объем) записей для копирования. В файл будут скопированы только записи, попадающие в границы заданного объема.

Стандартный объем для команды EXPORT это все записи.

TYPE

Вы должны задавать тип создаваемого файла. Дополнительная опция TYPE необязательна, но вы должны включить один из следующих типов создаваемого файла.

DIF

Поля базы данных FoxPro становятся векторами (колонками), а записи становятся кортежами (строками) в DIF (Формат Обмена Данными, используемый в VisiCalc) файле. Новый файл получит расширение .DIF, если расширение не задано явно.

MOD

Файлы MOD, используемые Microsoft's Multiplan версии 4.01.

SYLK

Файл SYLK является форматом обмена СимволическихСвязей (используемым в Multiplan), в котором поля базы данных FoxPro становятся крупноформатной таблицей, а записи становятся строками. Файлы SYLK не имеют стандартного расширения.

WK1

С помощью этой опции из базы данных FoxPro может быть создана крупноформатная таблица Lotus 1-2-3. Крупноформатной таблице назначается расширение .WK1 (для использования в Lotus 1-2-3 2.x). Поля базы данных становятся колонками в новой крупноформатной таблице; записи в базе данных становятся строками таблицы.

WKS

С помощью этой опции из базы данных FoxPro может быть создана крупноформатная таблица Lotus 1-2-3. Крупноформатной таблице назначается расширение .WKS (для использования в Lotus 1-2-3 1-A). Поля базы данных становятся колонками в новой крупноформатной таблице; записи в базе данных становятся строками таблицы.

WR1

С помощью этой опции из базы данных FoxPro может быть создана крупноформатная таблица Lotus Symphony. Крупноформатной таблице назначается расширение .WK1 (для использования в Symphony версии 1.10). Поля базы данных становятся колонками в новой крупноформатной таблице; записи в базе данных становятся строками таблицы.

WRK

С помощью этой опции из базы данных FoxPro может быть создана крупноформатная таблица Lotus Symphony. Крупноформатной таблице назначается расширение .WK1 (для использования в Symphony версии 1.01). Поля базы данных становятся колонками в новой крупноформатной таблице; записи в базе данных становятся строками таблицы.

XLS

С помощью включения опции XLS база данных FoxPro может быть скопирована в крупноформатную таблицу Microsoft Excel версии 2. Поля базы данных становятся колонками в крупноформатной таблице; записи в базе данных становятся строками таблицы. Расширение .XLS назначается новой крупноформатной таблице, если вы явно не задали иное расширение.

FOR ... ENDFOR

Выполняет операторы внутри цикла определенное количество раз

■ FOR <memvar> = <ВыpN> TO <ВыpN1> [STEP <ВыpN2>]
  <statements>
  [EXIT]
  [LOOP]
ENDFOR | NEXT

Оператор FOR ... ENDFOR выполняет операторы, заключенные внутри цикла, заданное количество раз. Началом цикла является утверждения FOR; концом — утверждение ENDFOR. Переменная в памяти <memvar> используется в качестве счетчика, определяющего, сколько раз будут выполнены операторы внутри цикла.

Первое численное выражение <ВырN> является начальным значением счетчика, а второе численное выражение <ВырN1> его конечным значением. Операторы программы после FOR выполняются, пока не будет достигнуто утверждение ENDFOR или NEXT. Тогда счетчик <memvar> увеличивается на шаг STEP <ВырN2>. Если опция STEP отсутствует, то значение счетчика увеличивается на единицу.

После этого значение счетчика сравнивается с конечным значением <ВырN1>. Если значение счетчика меньше или равно значению выражения <ВырN1>, то операторы в цикле FOR выполняются снова. Если значение счетчика больше выражение <ВырN1>, то выполнение операторов цикла заканчивается и программа продолжает выполняться с оператора, следующего за утверждением ENDFOR.

Выражения <ВырN>, <ВырN1> и <ВырN2> могут быть численными выражениями или численными переменными в памяти.

Внимание !!!

Значения <ВырN>, <ВырN1> и <ВырN2>считываютсятолько первоначально, и их значения могут быть изменены внутри цикла без изменения количества итераций. Однако если вы измените значение счетчика <memvar> внутри цикла, это изменит количество итераций цикла.

Если значение выражения <ВырN2> в опции STEP отрицательно, то значение счетчика при каждой итерации будет уменьшаться, а значение начального выражения <ВырN> должно быть больше, чем значение конечного выражения <ВырN>.

<memvar>

<memvar> является в переменной в памяти, которая рассматривается как счетчик. Она не нуждается в определении перед выполнением FOR ... ENDFOR.

<ВырN>

<ВырN> является начальным значением счетчика.

<ВырN1>

<ВырN1> является конечным значением счетчика.

STEP <ВырN2>

Счетчик <memvar> увеличивается после каждой итерации на шаг <ВырN2>. Если <ВырN2> является отрицательным выражением, счетчик уменьшается. Если STEP <ВырN2> отсутствует, счетчик увеличивается на 1.

<statements>

Команды FoxPro, предназначенные для выполнения. Вы можете разместить любое число команд между утверждениями FOR и ENDFOR.

EXIT

EXIT передает управление из цикла FOR ... ENDFOR оператору, непосредственно следующему за командой ENDFOR. EXITможет располагаться в любом месте между операторами FOR и ENDFOR.

LOOP

LOOP также может располагаться в любом месте цикла между операторами FOR и ENDFOR. Включение LOOP ENDFOR возвращает управление прямо на FOR без выполнения команд между LOOP и ENDFOR. Счетчик увеличивается/уменьшается также, как и в случае достижения ENDFOR.

FWRITE()

Записывает символьную строку в файл или коммуникационный порт

FWRITE(<ВырN>, <ВырC> [, <ВырN1>])

<ВырN> Дескриптор файла или порта

<ВырC> Символьная строка, записываемая в файл или порт

<ВырN1> Число символов, записываемых в файл или порт

Функция FWRITE() записывает строку символов в низкоуровневый файл или коммуникационный порт. В отличие от функции FPUTS(), функция FWRITE()не размещает возврат каретки и подачу строки в конце строки символов, которую она записывает в файл или коммуникационный порт. Функция FWRITE() возвращает число

байтов, записанных в файл или порт. Если FWRITE() не может осуществить запись в файл или порт, возвращается 0.

<ВырN>

FWRITE() осуществляет запись в файл или порт, заданный <ВырN>. Функции FOPEN() и FCREATE() используются для назначения дескриптора файла переменной в памяти. Обычно, <ВырN> задано этой переменной.

<ВырC>

Строка, которую функция FWRITE() записывает в файл или порт, задается с помощью <ВырC>.

<ВырN1>

Функция FWRITE() записывает в файл или порт целую строку, если не задано <ВырN1>. Когда включено <ВырN1>, <ВырN1> символов записывается в файл или порт. Вся строка <ВырC> будет записана в файл или порт в случае, если <ВырN1> больше или равно числу символов в <ВырC>. Если <ВырN1> меньше числа символов в <ВырC>, только <ВырN1> символов будут записаны в файл или порт.

GATHER

Перемещает содержание переменных памяти или элементов массивов в поля базы данных.

GATHER FROM <array> | MEMVAR
[FIELDS <field name list>]
[MEMO]

Команда GATHER перемещает данные из набора переменных памяти или массивов переменных памяти в текущую запись активной базы данных.

FROM <array>

Если данные перемещаются из массива, то элементы массива пересылаются начиная с первого в соответствующие поля записи. Содержание первого элемента массива копируется в первое поле, второго элемента массива во второе поле и т. д.

Если количество элементов массива меньше, чем количество полей в записи базы данных, то поля модифицируются до того момента, пока не исчерпаются все элементы массива. Если количество элементов массива больше, чем количество полей в записи базы данных, неуместившиеся элементы массива игнорируются.

MEMVAR

Если используется опция MEMVAR, то из списка переменных памяти пересылаются переменные, с такими же именами, что и поля в текущей записи базы данных.

Внимание !!!

Переменные памяти с теми же именами, что и поля в базе данных, легко могут быть созданы с помощью опций MEMVAR или BLANK команды SCATTER.

Содержимое каждой переменной пересылается в соответствующее поле с тем же именем. Если не может быть найдена переменная с тем же именем, что и поле базы данных, то содержимое поля не изменяется.

FIELDS <field name list>

Чтобы только заданные поля базы данных были заменены содержимым элементов массива или переменных памяти, вы можете задать опцию FIELDS со списком имен полей базы данных.

MEMO

Для включения в обработку командой GATHER memo полей, используйте ключевое слово MEMO. Если MEMO не включено, memo поля пропускаются, когда команда GATHER копирует данные из массива или переменных памяти в текущую запись.

GETFILE()

Отображает диалог Открыть файл (Open File)

GETFILE([<ВырС>] [, <ВырС1>])

<ВырС> Расширения файлов, представляемых в диалоге Открыть файл

<ВырС1> Строка-подсказка, которая будет отображаться в верхней части диалога Открыть файл

Функция GETFILE() отображает диалог Открыть файл (Open File) FoxPro в пределах которого может быть выбран файл на диске. Затем эта функция возвращает имя выбранного файла. Имя файла, выбранного пользователем, возвращается как значение функции. В том случае, если никакой файл не выбран (выбирается вариант Cancel или нажимается клавиша Escape), функция возвращает пустую строку.

<ВырС>

Необязательный аргумент <ВырС> указывает расширение файлов, кото рые должны быть отображены в прокручиваемом списке диалога Открыть файл, когда контрольный блок All Files (Все файлы) не выбран. Для по лучения информации, касающейся полного списка расширений файлов, используемых в системе FoxPro, обратитесь к таблице расширений файлов, имеющейся в "Руководстве разработчика" (Developer's Guide) по системе FoxPro.

<ВырС> может содержать список расширений файлов, которые разделяются точкой с запятой (например, 'PRG;FXP'). В этом случае GETFILE() отобразит все файлы с расширениями PRG и FXP. Однако, если файлы имеют одно и тоже имя, но разные расширения (например, CUSTOMER.PRG и CUSTOMER.FXP), будет представлен только файл с первым расширением, перечисленным в списке расширений. В этом примере, если список расширений файлов содержит 'PRG;FXP', будет представлен только файл CUSTOMER.PRG.

— <ВырС> может содержать список расширений файлов, которые разделяются вертикальной чертой (например, 'PRG|FXP'). В этом случае будут представлены все файлы с данными расширениями, даже если они имеют одинаковые имена.

— Если <ВырС> содержит только одну точку с запятой (';'), в прокручиваемом списке отображаются все файлы без расширения.

— Если <ВырС> содержит пустую строку (''), отображаются все файлы, имеющиеся в текущей директории.

— <ВырС> также может содержать метасимволы ДОС (* и ?). Будут отображаться все файлы с расширениями, удовлетворяющими условиям метасимволов. Например, если <ВырС> является '?X?', будут представлены все файлы с расширениями FXP, EXE, TXT и т. д.

Необязательный аргумент <ВырС1> представляет собой строку-подсказку, которая будет отображаться в верхней части диалога Открыть файл.

IF ... ENDIF

■ IF <ВырL>
    <statements1>
[ELSE
    <statements2>]
ENDIF

В этой программной структуре вычисляется выражение <ВырL>. Если выражение <ВырL> истинно, то будет выполнены операторы, следующие за оператором IF и предшествующие одному из операторов ELSE или ENDIF (тому, который встретиться раньше).

Если выражение <ВырL> ложно, и определен оператор ELSE будет выполнены операторы, находящиеся между ELSE и ENDIF.

Если выражение <ВырL> ложно, а оператор ELSE отсутствует, то все операторы, находящиеся между IF и ENDIF будут проигнорированы. В этом случае выполнение программы будет продолжено с оператора, следующего за ENDIF.

Операторы IF могут быть вложены один в другой, но каждому оператору IF должен соответствовать оператор ENDIF.

Комментарии могут располагаться в той же строке, что и ELSE и ENDIF. Они будут игнорироваться в процессе выполнения программы.

IIF()

Возвращает одно из двух выражений в зависимости от значения логического выражения

IIF(<ВырL>, <Выр1>, <Выр2>)

<ВырL> Вычисляемое логическое выражение

<Выр1> Выражение, возвращаемое, если <ВырL> истинно

<Выр2> Выражение, возвращаемое, если <ВырL> ложно

Функция IIF (Непосредственное Если) возвращает одно из двух выражений в зависимости от значения логического выражения. Если логическое выражение имеет значение "истинно" (.T.), возвращается первое выражение. Если логическое выражение имеет значение "ложно" (.F.), функция IIF() возвращает второе выражение.

<ВырL>

<ВырL> задает логическое выражение, которое вычисляет функция IIF().

<Выр1>, <Выр2>

Одно из двух выражений, <Выр1> или <Выр2>, возвращается функцией IIF(). Если логическое выражение <ВырL> "истинно" (.T.), возвращается первое выражение <Выр1>. Если <ВырL> "ложно" (.F.), возвращается второе выражение <Выр2>.

Выражения <Выр1> и <Выр2> должны быть символьного типа, типа дата, логического или численного типа. <Выр1> и <Выр2> не обязаны быть одного типа.

Помимо этого следует отметить, что функция IIF() выполняется значительно быстрее, чем эквивалентные ей операторы IF ... ENDIF.

IMPORT

Импортирует данные из файла с другим форматом в создаваемую новую базу данных FoxPro

■ IMPORT FROM <file>

TYPE

Вы должны задавать тип создаваемого файла. Дополнительная опция TYPE необязательна, но вы должны включить один из следующих типов создаваемого файла.

FW2

Файлы FW2, созданные Framework II.

MOD

Файлы MOD, созданные Microsoft's Multiplan версии 4.01.

PDOX

Файлы баз данных Borland's Paradox.

RPD

Файлы RPD, созданные RapidFile.

SYLK

Файл SYLKявляется форматом обмена Символических Связей (используемым в Multiplan), в котором колонки становятся полями, а строки становятся записями. Файлы SYLK не имеют стандартного расширения.

WK1

Данные из крупноформатной таблицы Lotus 1-2-3. Колонки в крупноформатной таблице становятся полями в базе данных; строки таблицы становятся записями в базе данных. Расширение .WK1 назначается крупномасштабной таблице, созданной Lotus 1-2-3 2.x.

WKS

Данные из крупноформатной таблицы Lotus 1-2-3 1-A.

WR1

Данные из крупноформатной таблицы Lotus Symphony. Колонки в крупноформатной таблице становятся полями в базе данных; строки таблицы становятся записями в базе данных. Расширение .WR1 назначается крупномасштабной таблице, созданной Symphony версии 1.10.

WRK

Данные из крупноформатной таблицы Lotus Symphony. Колонки в крупноформатной таблице становятся полями в базе данных; строки таблицы становятся записями в базе данных. Расширение .WRK назначается крупномасштабной таблице, созданной Symphony версии 1.01.

XLS

Данные из крупноформатной таблицы Exceel версии 2. Колонки в крупноформатной таблице становятся полями в базе данных; строки таблицы становятся записями в базе данных. Расширение .XLS назначается крупномасштабной таблице, созданной Exceel.

INDEX

Создание индексного файла для упорядочения базы данных.

■ INDEX ON <Выр> TO <.idx file> | TAG <tag name> [OF <.cdx file>]
[FOR <ВырL>]
[COMPACT]
[ASCENDING | DESCENDING]
[UNIQUE]
[ADDITIVE]

Команда INDEX используется для создания индексного файла или тега для текущей выбранной базы данных. База данных, в которой имеется индексный файл, доступна и отображается в порядке, определенном в индексном выражении <Выр>, пока установлено упорядочение по этому индексу. Физический порядок записей базы данных не изменяется. Если установлен режим SET TALK ON, FoxPro сообщает, сколько записей было проиндексировано во время процесса индексирования.

Промежуток, через который сообщается о количестве проиндексированных записей, может быть установлен с помощью команды SET ODOMETER.

Команда DISPLAY STATUS дает дополнительную информацию об открытых индексных файлах. Эта информация включает в себя имена всех открытых индексных файлов, их типы (структурный, .CDX, .IDX), их индексные выражения и имя главного индекса или главного тега.

Число индексных файлов (.IDX или .CDX), которые вы можете открыть, ограничено только объемом доступной памяти и общим числом файлов, которые вы можете открыть. Число файлов, которые вы можете открыть, определяется параметром FILES в

файле конфигурации DOS CONFIG.SYS. Более подробную информацию о параметре FILES для DOS можно найти в вашем руководстве по DOS.

Типы индексов

FoxPro позволяет вам создавать два типа индексных файлов:

- Составные индексные файлы .CDX, содержащие несколько индексных элементов, называемых тегами
- Индексные файлы .IDX, содержащие один индексный элемент.
- Может быть создан сложный структурный индексный файл. Сложный структурный индексный файл автоматически открывается каждый раз при открытии базы данных.

Внимание !!!

Так как сложные структурные индексные файлы .CDX открываются автоматически при открытии базы данных, они имеют предпочтительный индексный тип.

Включайте опцию COMPACT для создания компактного индексного файла .IDX. Индексные файлы .CDX всегда являются компактными.

Если вы используете файлы одновременно в FoxPro и FoxBASE+ или FoxBASE/Mac, вы не должны использовать компактные индексные файлы. С другой стороны, если вы создаете индексные файлы .IDX, всегда используйте COMPACT для использования технологии быстрого доступа к индексу FoxPro 2.0.

Индексный порядок и обновление

Только один индексный файл (главный индексный файл) или тег (главный тег) управляет порядком доступа и представления базы данных.

Некоторые команды (например, SEEK) используют главный индексный файл или тег для поиска записи. Однако, при изменениях в базе данных обновляются все открытые индексные файлы .IDX и .CDX. Вы можете указывать главный индексный файл или тег в опции INDEX команды USE или с помощью SET INDEX и SET ORDER. Более подробную информацию о назначении главного индексного файла или главного тега можно найти в описании этих команд в этой главе.

Дополнительные опции

<Выр>

Индексное выражение <Выр> включает поля текущей выбранной базы данных. Индексный ключ, основанный на индексном выражении, создается в индексном файле для каждой записи в базе данных. FoxPro использует эти ключи для доступа и представления записей в базе данных.

TAG <tag name> [OF <.cdx file>]

Составной индексный файл .CDX может быть создан с помощью опции TAG <tag name>. Составной индексный файл .CDX является одним индексным файлом, который содержит некоторое число различных тегов (индексных компонентов); каждый тег идентифицируется уникальным именем тега <tag name>. Имена тегов, подобно именам переменных в памяти, должны начинаться с буквы или символа подчеркивания и могут содержать любую комбинацию до 10 букв, цифр или символов подчеркивания. Число тегов в составном индексном файле .CDX ограничено только объемом доступной памяти и пространством на диске.

Многокомпонентные составные индексные файлы всегда компактны. Поэтому, не нужно включать опцию COMPACT при создании составного индексного файла. Составные индексы имеют расширения .CDX.

Могут быть созданы два типа составных индексных файлов .CDX.

Первый тип, сложный структурный индексный файл, создается, когда вы используете опцию TAG <tag name> без дополнительной опции OF <.cdx file>. Структурные файлы

.CDX всегда имеют то же базовое имя, что и база данных. Сложный структурный индекс автоматически открывается каждый раз, когда открывается база данных.

Если структурный индексный файл базы данных не может быть обнаружен или был удален или переименован, будет представлен диалог "Structural CDX file not found" ("Структурный файл CDX не найден"), когда вы попытаетесь открыть файл базы данных. Если вы выберите кнопку Cancel (Отменить) (по умолчанию), база данных не открывается. Выбор кнопки Ignore (Игнорировать) открывает базу данных и удаляет флаг в заголовке базы данных, который указывает, что представлен структурный индексный файл.

UNIQUE

Опция UINIQUE определяет то, что только первая запись встреченная с особенным ключевым значением будет включена в индексный файл .IDX или тег .CDX. Все последующие записи с двойными ключами будут исключены из индексного файла. Использование опции UNIQUE в команде INDEX эквивалентно выполнению команды SET UNIQUE перед командами INDEX или REINDEX.

INKEY()

Возвращает целое значение, соответствующее нажатию клавиши или щелчку мыши

INKEY([[<ВырN>] [, <ВырC>]])

<ВырN> Число секунд, в течение которых INKEY будет ожидать нажатие клавиши или щелчок мыши. Если с выражение <ВырN> равно 0, функция INKEY() будет неопределенно долго ожидать нажатия клавиши.

<ВырC> Если S, INKEY показывает курсор

Если H, INKEY скрывает курсор

Если M, INKEY определяет щелчок или нажатие клавиши

Значение, возвращаемое INKEY()

Клавиша		Одна	Shift	Ctrl	Alt
F1	28	84	94	104	
F2	-1	85	95	105	
F3	-2	86	96	106	
F4	-3	87	97	107	
F5	-4	88	98	108	
F6	-5	89	99	109	
F7	-6	90	100	110	
F8	-7	91	101	111	
F9	-8	92	102	112	
F10	-9	93	103	113	
F11	133	135	137	139	
F12	134	136	138	140	
1	49	33	—	120	
2	50	64	33	121	
3	51	35	—	122	
4	52	36	—	123	
5	53	37	—	124	
6	54	94	30	125	
7	55	38	—	126	
8	56	42	—	127	
9	57	40	—	128	
0	48	41	—	19	
a	97	65	1	30	
b	98	66	2	48	
c	99	67	3	46	
d	100	68	4	32	
e	101	69	5	18	
f	102	70	6	33	
g	103	71	7	34	
h	104	72	8	35	
i	105	73	9	23	
j	106	74	10	36	
k	107	75	11	37	
l	108	76	12	38	
m	109	77	13	50	
n	110	78	14	49	
o	111	79	15	24	
p	112	80	16	25	
q	113	81	17	16	

r	114	82	18	19	Del	7	46	—	—	
s	115	83	19	31	End	6	49	23	—	
t	116	84	20	20	PgUp	18	57	31	—	
u	117	85	21	22	PgDn	3	51	30	—	
v	118	86	22	47	Up	5	56	—	—	
w	119	87	23	17	Right	4	54	2	—	
x	120	88	24	45	Left	19	52	26	—	
y	121	89	25	21	Down	24	50	—	—	
z	122	90	26	44	Escape		2727	27	—	—
Ins	22	48	—	—	Enter	13	13	10	—	
Home	1	55	29	—	BackSpace		127	127	127	—

JOIN

Объединяет две базы данных

■ JOIN WITH <ВырN> | <ВырC>
TO <file>
FOR <ВырL>
[FIELDS <field list>]

Команда JOIN создает новый файл базы данных из двух других баз данных. Одна из них — это текущая выбранная база данных, а вторая определена с помощью номера ее рабочей области или псевдонима. Команда JOIN устанавливает указатель записи на первую запись текущей выбранной базы данных и просматривает все записи во второй базе данных.

<ВырN> | <ВырC>

Номер рабочей области <ВырN> или псевдоним базы данных <ВырC>.

TO <file>

Имя базы данных, создаваемой из сливаемых баз данных.

FOR <ВырL>

Команда JOIN вычисляет выражение <ВырL> в операторе FOR для каждой записи. Если выражение <ВырL> истинно, то запись переписывается в файл новой базы данных. Команда JOIN переходит к следующей записи текущей базы данных и повторяет процедуру.

Rushmore будет оптимизировать запрос JOIN FOR, если <ВырL> является оптимизируемым выражением. Для повышения эффективности используйте оптимизируемые выражения в опции FOR. Выражения, оптимизируемые Rushmore, рассматриваются в главе Оптимизация ваших прикладных программ в "Руководстве разработчика" (Developre's Guide) по системе FoxPro.

FIELDS <field list>

Вы можете включить список полей, включаемых в новый файл базы данных, определив его в опции FIELDS и списке полей <field list>.

Список полей в опции FIELDS может включать поля как из текущей базы данных, так и из базы данных с именем, определенным псевдонимом.

KEY()

Возвращает ключевое выражение для индексного файла

KEY([<.cdx file>,] <ВырN>[, <ВырN1> | <ВырC>])

<cdx file> Имя составного индексного файла .CDX.

<ВырN> Указывает, какое индексное выражение нужно получить из открытого индексного файла

<ВырN1> Номер рабочей области базы данных

<ВырС> Псевдоним базы данных

Функция KEY() возвращает индексное выражение для открытого индексного файла. Индексное выражение задается, когда индексный файл создается командой INDEX. Индексное выражение определяет, как база данных будет доступна и как она будет представляться, когда индексный файл будет открыт как главный управляющий индексный файл.

Более подробную информацию о создании индексных файлов и индексных выражений можно найти в описании команды INDEX в этом руководстве. Функция KEY() подобна функции SYS(14).

<cdx file>

Функция KEY() может использоваться для того, чтобы возвратить индексные ключевые выражения для тегов в многокомпонентных составных индексных файлах. Составной индексный файл может быть сложным структурным файлом, автоматически открывающимся с файлом базы данных, или независимым составным индексным файлом.

<ВырN>

Обе команды USE и SET INDEX поддерживают список индексных файлов, который позволяет вам открывать индексные файлы для базы данных. В список индексных файлов можно включать любые комбинации однокомпонентных индексных файлов .IDX, сложных структурных или независимых составных индексных файлов.

Численное выражение <ВырN> указывает, какое индексное выражение нужно возвратить из открытых индексных файлов. Функция KEY() возвращает индексные выражения из открытых индексных файлов в следующем порядке:

- Прежде всего возвращаются индексные выражения из однокомпонентных индексных файлов .IDX (если есть такие открытые файлы). Порядок, в котором однокомпонентные индексные файлы включены в USE или SET INDEX, определяет порядок возвращаемых индексных выражений.

- Следующими возвращаются индексные выражения для каждого тега в сложном структурном индексе (если он представлен). Индексные выражения возвращаются из тегов в том же порядке, что и порядок создания тегов в структурном индексе.

- Последними возвращаются индексные выражения для каждого тега во всех открытых независимых составных индексов. Индексные выражения возвращаются из тегов в том же порядке, что и порядок создания тегов в независимых составных индексах.

Если <ВырN> больше числа открытых однокомпонентных .IDX файлов и тегов сложного структурного и независимых составных индексов, возвращается пустая строка.

<ВырN1> | *<ВырС>*

Если вы не задали рабочую область или псевдоним, возвращается индексное выражение из индексного файла, открытого для базы данных в текущей рабочей области. Вы можете получить индексные выражения из индексного файла, открытого в другой рабочей области, задав номер рабочей области <ВырN1> или псевдоним базы данных <ВырС>.

Если нет базы данных с заданным псевдонимом, выдается сообщение "Alias not found" ("Псевдоним не найден").

LABEL

■ LABEL [FORM <file1> | ?]
[ENVIRONMENT]
[PREVIEW]
[SAMPLE]
[TO PRINTER | TO FILE <file2>]

Команда LABEL отображает или выводит на печать этикетки под управлением файла описания этикеток в <file1>. Файлы описания этикеток создаются с помощью команд MODIFY LABEL или CREATE LABEL.

Для файла описания этикеток по умолчанию принимается расширение .LBX. Если файл описания этикеток находится не в той поддиректории или устройстве, доступ к которому определен по умолчанию, то должны быть определены обозначения устройства или поддиректории.

Если команда LABEL появляется без каких-либо дополнительных аргументов, представляется диалог Открыть файл (Open File) со списком существующих файлов этикеток для выбора.

Более подробную информацию о этикетках можно найти в "Руководстве по интерфейсу" (Interface Guide) по системе FoxPro или в "Началах работы" (Getting Started) по системе FoxPro.

FORM <file1>

Вы можете указать, какие этикетки следует печатать, с помощью имени базы данных описания этикеток <file> после ключевого слова FORM.

?

Если вы используете опцию ?, представляется диалог Открыть файл (Open File) со списком существующих файлов этикеток для выбора.

PREVIEW

Включение опции PREVIEW посылает представление ваших этикеток на экран, что позволяет проверять их содержание и планирование перед печатью этикеток. Если включена опция PREVIEW, этикетки будут посылаться на экран, но не будут печататься. Для того, чтобы напечатать их, необходимо подать другую команду LABEL без опции PREVIEW.

SAMPLE

Опция SAMPLE может быть использована для проверки выравнивания этикеток. После отображения на экране или распечатки образца этикетки будет выдан вопрос "Вы хотите продолжить выборку?". Ответ "Y" позволит вам повторить проверку выравнивания этикеток.

TO PRINTER | TO FILE <file2>

Для вывода этикеток на принтер используйте команду TO PRINTER. Для вывода этикеток в текстовый файл используйте команду TO FILE. Файлу <file2> в команде TO FILE по умолчанию дается расширение .TXT.

LASTKEY()

Возвращает значение, соответствующее последней нажатой клавише

Функция LASTKEY() возвращает целочисленное значение, соответствующее последней нажатой клавише. Значение, возвращаемые функцией LASTKEY(), те же, что и значения, возвращаемые функцией INKEY(). Таблицу с клавишами и их значениями можно найти в описании функции INKEY().

LIST

Показывает записи базы данных или выражения

■ **LIST**
[FIELDS <Выр list>]
[<scope>] [FOR <ВырL>]
[WHILE<ВырL>]
[OFF]
[NOCONSOLE]
[TO PRINTER | TO FILE <file>]

Команды LIST эквивалентны командам DISPLAY за исключением трех отличий:

- Область действия <scope> для команды LIST по умолчанию ALL (все записи)
- Команда LIST не приглашает вас продвигаться вперед, когда экран или окно заполнятся
- Команда LIST не показывает записей, помеченных для удаления, когда DELETED является SET ON (включено).

Дополнительную информацию о командах LIST можно найти в соответствующих темах DISPLAY в этом руководстве.

LOCATE

Определяет местоположение записи в базе данных

■ LOCATE

Команда LOCATE последовательно ищет в текущей выбранной базе данных первую запись, соответствующую выражению <ВырL>. База данных не должна быть индексирована.

Если команда LOCATE находит соответствующую запись, функция RECNO() вернет номер этой записи, функция FOUND() вернет значение "истинно" (.T.), а функция EOF() вернет значение "ложно" (.F.). Номер записи также отобразится на экране, несмотря на то, что команда TALK установлена в SET OFF.

Если команда LOCATE нашла соответствующую запись, вы можете использовать команду CONTINUE для поиска соответствующих записей в оставшейся части файла. При выполнении команды CONTINUE процесс поиска продолжается, начиная с записи, следующей непосредственно за найденной. Команда CONTINUE выполняется до тех пор, пока не будет достигнут конец области видимости или конец файла, или до тех пор, пока не заработает новая команда LOCATE.

Если соответствующая запись не будет найдена, функция RECNO() вернет величину, равную количеству записей в базе данных плюс один, функция FOUND() вернет значение "ложно", а функция EOF() вернет значение "истинно".

Команда LOCATE также может быть запущена путем выбора в Меню записей (Record) пункта Locate...

Для получения более полной информации по интерактивному использованию команды LOCATE обратитесь к "Руководству по интерфейсу" (Interface Guide) системы FoxPro.

Итерационная процедура определения местоположения записей и осуществления серии операций над ними может быть создана в программном файле путем помещения ряда операторов FoxPro между командами LOCATE и CONTINUE.

Команды LOCATE и CONTINUE определены для текущей выбранной рабочей области. Если выбрана другая рабочая область, то процесс поиска приостанавливается до тех пор, пока не будет переопределена исходная рабочая область. Тогда процесс поиска будет продолжен.

LOCFILE()

Определяет местоположение файла на диске

LOCFILE(<ВырС>[, <ВырС1>] [, <ВырС2>])

<ВырС> Отыскиваемый файл

<ВырС1> Список расширений файлов

<ВырС2> Приглашение диалога Открыть файл

Возвращаемый тип

Символьный

Функция LOCFILE() определяет местоположение файла на диске и возвращает имя файла с полным путевым именем. Если файл не может быть найден, представляется диалог Открыть файл так, чтобы вы могли отыскать файл.

Параметры

<ВырС>

Параметр <ВырС> задает имя отыскиваемого файла. Если <ВырС> является простым именем файла, исследуется директория по умолчанию. Если файла нет в директории по умолчанию, исследуется маршрут, заданный для FoxPro. Если файл найден, возвращается полное путевое имя файла. Если <ВырС> не имеет расширения, рассматриваются расширения из <ВырС1> при попытке определить местоположение файла.

Если в <ВырС> вы зададите директорию вместе с именем файла, будет исследоваться заданная директория. Если файл не может быть найден в заданной директории, исследуется директория по умолчанию, а затем исследуется маршрут, заданный для FoxPro. Если файл найден, возвращается полное путевое имя файла.

Если файл не может быть найден в директории по умолчанию, также как и в маршруте FoxPro или заданной директории, представляется диалог Открыть файл. Диалог Открыть файл может быть использован для определения местоположения файла. Когда файл выбирается из диалога, возвращается полное путевое имя файла, и маршрут файла добавляется в маршрут FoxPro.

Если был осуществлен выход из диалога Открыть файл с помощью кнопки Cancel или нажатия клавиши Escape, выводится предупреждение "File does not exist" (Файл не существует) (сообщение об ошибке 1), и функция LOCFILE() не возвращает значения.

<ВырС1>

Параметр <ВырС1> определяет расширения файлов, представляемых в диалоге Открыть файл, если не выбран блок проверки All Files (Все файлы). Полный список расширений, используемых FoxPro, можно найти в таблице расширений файлов в "Руководстве разработчика" (Developer's Guide) по системе FoxPro.

<ВырС1> может быть представлен в различных формах:

- <ВырС1> может содержать единственное расширение (например, 'PRG'), и будут представлены только те файлы, которые имеют данное расширение.

- <ВырС1> может содержать список расширений файлов, разделенных точкой с запятой (например, 'PRG;FXP'). В данном примере функция LOCKFILE() представит все файлы с расширениями PRG и FXP. Однако, если файлы имеют одно и тоже имя, но различные расширения, (например, CUSTOMER.PRG и CUSTOMER.FXP), будет представ-

лен только один файл с расширением, идущим в списке расширений первым. В этом примере, если список расширений файлов содержит 'PRG;FXP', будет представлен CUSTOMER.PRG.

- <ВырС1> может содержать список расширений файлов, разделенных вертикальной чертой (например, 'PRG|FXP'). В этом случае будут представлены все файлы с данными расширениями, даже если файлы имеют одинаковые имена.

- Если <ВырС1> содержит только точку с запятой (';'), все файлы без расширения будут представлены в прокручивающемся списке.

- Если <ВырС1> является пустой строкой, будут представлены все файлы в текущей директории.

- <ВырС1> может также содержать метасимволы DOS (* и ?). Будут представлены все файлы, которые удовлетворяют критерию метасимволов. Например, если <ВырС1> является '?X?', будут представлены все файлы с расширениями FXP, EXE, TXT и т. д.

<ВырС2>

Необязательное символьное выражение <ВырС2> является приглашением, представляемым вверху диалога Открыть файл.

LUPDATE()

Возвращает дату последнего обновления заданной базы данных

LUPDATE([<ВырN> | <ВырС>])

Функция LUPDATE() возвращает дату последнего обновления базы данных. Эта функция полезна в процедурах обновления.

Параметр *<ВырN> | <ВырС>*

Если вы не задали рабочую область или псевдоним, возвращается дата последнего обновления файла базы данных в текущей рабочей области. Вы можете получить дату последнего обновления базы данных, открытой в другой рабочей области, задав номер рабочей области <ВырN> или псевдоним базы данных <ВырС>.

Если в заданной рабочей области нет открытой базы данных, функция LUPDATE() возвращает пустую дату (' / / '). Если нет базы данных с заданным псевдонимом, выдается сообщение "Alias not found" ("Псевдоним не найден").

MODIFY COMMAND | MODIFY FILE

■ MODIFY COMMAND <file> | MODIFY FILE [<file>]

Открывает окно текстового редактора для заданного файла.

MODIFY QUERY

Открывает окно RQBE

■ MODIFY QUERY [<file> | ?]

Команда MODIFY QUERY позволяет вам интерактивно модифицировать существующий запрос или создавать новый запрос в окне RQBE.

<file>

Существующий запрос вы можете модифицировать, задав имя запроса <file>. Если файл с таким именем не существует или не найден, может быть создан новый запрос.

Появление команды MODIFY QUERY без дополнительных аргументов вызывает диалог Открыть файл. Запросу назначается имя UNTITLED. Когда вы выйдете из окна RQBE, вы можете сохранить запрос, используя другое имя. После того, как вы создали

запрос, он сохраняется как файл программы FoxPro с расширением .QPR. Программа запроса может быть выполнена с помощью команды DO.

?

Если вы используете опцию ? или опустите имя файла, будет представлен диалог Открыть файл со списком существующих файлов запросов для выбора файла или ввода имени для создания нового файла.

MODIFY REPORT

Открывает окно компоновки отчета

■ MODIFY REPORT [<file> | ?]

Этот формат команды MODIFY используется для модификации существующих файлов описаний отчетов. Файлы описаний отчетов создаются командой CREATE REPORT или выбором кнопки New... (Новый...) из всплывающего меню File (Файл).

Более подробную информацию о создании и модификации отчетов можно найти в "Руководстве по интерфейсу" (Interface Guide) и "Началах работы" (Getting Started) системы FoxPro.

<file>

С помощью имени отчета <file> вы можете модифицировать существующие отчеты. Если файл с таким именем не существует или не найден, может быть создан новый отчет. Для модификации отчета, созданного в FoxBASE+, вы должны включить расширение файла .FRM. Появление команды MODIFY REPORT без дополнительных аргументов вызывает диалог Открыть файл. Отчету назначается имя UNTITLED. Когда вы выйдете из окна компоновки отчетов, вы можете сохранить файл отчетов с другим именем. Структура .FRX файлов описана в "Руководстве разработчика" (Developer's Guide) по системе FoxPro.

■ MODIFY STRUCTURE

Открывает диалог для модификации структуры базы данных, открытой в текущей рабочей области.

NOTE | * | &&

Размещает комментарии в файле программы

NOTE [<comments>]

* [<comments>]

&& [<comments>]

Эти команды позволяют включить комментарии в файл программы. Если вы хотите продолжить комментарий на другой строке, то после первой строки комментария должна стоять точка с запятой.

NOTE [<comments>]

Слово NOTE в начале строки программы указывает, что эта строка является комментарием. Например:

NOTE Это комментарий

** [<comments>]*

Звездочка (*) в начале строки программы указывает, что эта строка является комментарием. Например:

* Это комментарий

&& [<comments>]

Символы && в конце строки программы указывают, что далее следуют встроенные комментарии.

OBJNUM()

Возвращает номер объекта поля GET

OBJNUM(<var> [, <ВырN>])

<var> Имя поля GET

<ВырN> Номер уровня READ

Команды @ ... GET и @ ... EDIT позволяют вам создавать GET объекты (поля, блоки проверки, невидимые, селективные и текстовые кнопки, всплывающие меню, списки и области редактирования текста). Номер объекта определяется порядком его создания.

<var>

Вы можете возвратить номер объекта для органа управления, заданного параметром <var>, который является переменной в памяти, элементом массива или полем базы данных, указывающим какой орган управления был создан.

<ВырN>

Вложенные команды READ могут быть получены с помощью команд GET и команды READ в процедуре, вызванной с помощью READ. Команды READ могут иметь до пяти уровней вложения. Для получения номера обоекта, относящегося к другому уровню READ, используйте необязательное численное выражение <ВырN>, задающее номер уровня READ. Если <ВырN> не используется, функция OBJNUM() возвращает номер объекта GET для текущего уровня READ.

ON KEY

Осуществляет прерывание выполнения программы при нажатии клавиши

ON KEY [<command>]

Команда ON KEY устанавливает прерывание при нажатии клавиши. Если будет нажата любая клавиша, то инициируется прерывание и выполняется команда <command>.

В один и тот же момент времени может работать только одна подпрограмма ON KEY. Если выполняется множество подпрограмм ON KEY, то эффективно работает самая последняя запущенная подпрограмма. Если в рабочем состоянии находятся подпрограммы ON ESCAPE и ON KEY, и команда ESCAPE установлена в ON , то нажатие клавиши Escape инициирует подпрограмму ON ESCAPE а не ON KEY. Если команда ESCAPE установлена в OFF и нажата клавиша Escape, то будет выполнена команда ON KEY.

После завершения выполнения подпрограммы ON KEY программа продолжает выполняться с той строки, которая непосредственно следуют за строкой, в которой произошел переход. Однако, если <command> является DO <program>, и выход из файла осуществлен через оператор RETRY, то выполнение программы продолжается с той же строки, в которой произошел переход.

ON KEY

Код клавиши	Соответствующие клавиши
272—281	Alt+Q,W,E,R,T,Y,U,I,O,P
286—294	Alt+A,S,D,F,G,H,J,K,L
300—306	Alt+Z,X,C,V,B,N.M

315—324 F1 — F10 Функциональные
клавиши
327 HOME
328 Стрелка вверх
329 PgUp
331 Стрелка влево
333 Стрелка вправо
335 End
336 Стрелка вниз
337 PgDn
338 Ins
339 Del
340—349 Shift+F1 — Shift+F10
350—359 Ctrl+F1 — Ctrl+F10
360—369 Alt+F1 — Alt+F10
370 Ctrl+Print+Scrn
371 Ctrl+Стрелка влево
372 Ctrl+Стрелка вправо
373 Ctrl+End
374 Ctrl+PgDn
375 Ctrl+Home
376—387 Alt+1,2,3,4,5,6,7,8,9,0,-,=
388 Ctrl+PgUp

Формат ON KEY = <ВырN> позволяет вам обозначить клавишу как "горячую клавишу". Это позволяет вам реализовать контекстно-зависимую подсказку, например переопределив клавишу F1. Функции PROGRAM() и VARREAD() могут быть использованы для возврата активной процедуры и имени поля, которое было введено при нажатии "горячей" клавиши.

Здесь приведены несколько дополнительных советов по использованию команды ON KEY = <ВырN> для создания контекстно-зависимой подсказки.

* Подпрограмма подсказки не должна использовать команды @ ... GET или CLEAR GETS. При попытке их использования появится сообщение:

"A READ is currently in effect".

* Для упрощения работы подпрограммы подсказки используйте при входе команду SAVE SCREEN, а при выходе команду RESTORE SCREEN. Это позволит вам избежать необходимости переписывать область экрана, используемую в подсказке.

ON KEY LABEL

Осуществляет прерывание по нажатию специальных клавиш

ON KEY [LABEL <key label>] [<command>]

Команда ON KEY LABEL устанавливает прерывание по нажатию специальных клавиш или по щелчку мыши. Если нажата любая из специальных клавиш, или выполнен щелчок мыши, то инициируется прерывание и выполняется команда <command>.

В отличие от команды ON KEY, могут работать одновременно несколько ON KEY LABEL. Например, вы можете для каждой из клавиш стрелок и обеих клавиш мыши иметь свою подпрограмму ON KEY LABEL.

LABEL <key label>

Метка клавиши <key label> представляет собой букву или цифру на самой клавише или специальное имя, присвоенное клавише. Смотрите таблицу специальных имен меток клавиш, приведенную ниже.

<command>

Команда, выполняемая при нажатии заданной клавиши.

Если <command> является DO <program>, первая строка программы <program> после всех утверждений с параметрами должна заблокировать установки ON KEY. Последняя строка программы <program> должна восстановить установки ON KEY перед возвратом в вызвавшую программу. Это предохраняет от рекурсивного вызова процедур ON KEY.

Обозначение клавиш	<Имена клавиш>
Стрелка влево	LEFTARROW
Стрелка вправо	RIGHTARROW
Стрелка вверх	UPARROW
Стрелка вниз	DNARROW
Home	HOME
End	END
PgUp	PGUP
PgDn	PGDN
Del	DEL
Backspace	BACKSPACE
Пробел	SPACEBAR
Ins	INS
Tab	TAB
Shift Tab	BACKTAB
Enter	ENTER
F1 — F12	F1,F2,F3...
Ctrl+F1 — Ctrl+F12	Ctrl+F1 , Ctrl+F2...
Shift+F1 — Shift+F12	Shift+F1 , Shift+F2...
Alt+F1 — Alt+F12	Alt+F1, Alt+F2...
Alt+0 — Alt+9	Alt+0, Alt+1, Alt+2...
Alt+A — Alt+Z	Alt+A, Alt+B, Alt+C...
Ctrl+Стрелка влево	Ctrl+LEFTARROW
Ctrl+Стрелка вправо	Ctrl+RIGHTARROW
Ctrl+Home	Ctrl+HOME
Ctrl+End	Ctrl+END
Ctrl+PgUp	Ctrl+PGUP
Ctrl+PgDn	Ctrl+PGDN
Ctrl+A — Ctrl+ Z	Ctrl+A, Ctrl+B, Ctrl+C...
Правая клавиша мыши	RIGHTMOUSE
Левая клавиша мыши	LEFTMOUSE
Мышь	MOUSE
Escape	ESC

ORDER()

Возвращает имя главного индексного файла или главного тега

ORDER([<ВырN> | <ВырC>[, <ВырN1>]])

PACK

■ PACK [MEMO] [DBF]

Физически удаляет записи, помеченные для удаления, в файле базы данных и уменьшает размер memo файла

Появление PACK MEMO удаляет неиспользуемое пространство в memo файле, но не удаляет записи, помеченные для удаления, из базы данных.

Если вы используете опцию DBF, база данных будет упаковываться без упаковки memo файла.

PADC(), PADL(), PADR()

Назначение

Заполняет выражение

Синтаксис

PADL(<Выр>, <ВырN> [, <ВырC>])

PADR(<Выр>, <ВырN> [, <ВырC>])

PADC(<Выр>, <ВырN> [, <ВырC>])

Параметры

<Выр> Заполняемое выражение

<ВырN> Длина выражения после его заполнения

<ВырC> Символы, используемые для заполнения

Эти функции возвращают в виде символьной строки заданное выражение, заполненное до длины <ВырN>. Вы можете использовать символьное выражение для заполнения данного выражения.

Функция PADL() заполняет выражение слева, PADR() — справа, RADC() заполняет выражение с обеих сторон.

Параметры

Выражение <Выр> будет заполняться. Это выражение может быть символьным, численным выражением или выражением с датой.

Параметр <ВырN> задает длину выражения после его заполнения.

Параметр <ВырC> используется для заполнения. Если <ВырC> опущен, для заполнения используются пробелы (ASC(32)). Если параметр <ВырC> используется, то <ВырC> повторяется необходимое число раз для заполнения выражения <Выр> до длины <ВырN> символов.

PARAMETERS

Определяет переменные в памяти как параметры

PARAMETERS <parameter list>

Команда PARAMETERS позволяет ставить в соответствие данным, передаваемым из вызывающих программ, локальные переменные в памяти или массивы.

Переменные и массивы передаются в программы по ссылке. Если значение изменяется в вызванной программе, то в вызывающую программу возвращается новое значение. Если вы хотите передать переменную или массив по значению, заключите переменную или массив в круглые скобки. Любые изменения, произведенные над параметром, не передаются назад в вызывающую программу. Переменные и массивы передаются по значению в функции, определенные пользователем (UDF). Установка команды SET

UDFPARMS в REFERENCE позволяет передавать переменные и массивы в UDF по ссылке.

Параметры внутри списка параметров <parameter list> должны быть корректными именами переменных в памяти или массивов. Если значение переменной или элемента массива изменяется в вызванной программе, новое значение передается назад в вызывающую программу.

Параметры, содержащиеся в списке параметров <parameter list>, разделяются запятыми. Если в списке параметров оператора PARAMETERS перечислено больше переменных или массивов, чем передано вызывающей программой, то оставшиеся переменные или массивы инициализируются в значение "ложно" (.F.).

Функция PARAMETERS() возвращает число параметров, которые были переданы последний раз процедуре.

PARAMETERS()

Возвращает число параметров, которые были переданы программе, вызванной последней PARAMETERS()

■ PRIVATE

Определяет локальные переменные памяти или массивы.

PROCEDURE

Определяет начало подпрограммы

PROCEDURE <procedure name>

Во многих программах некоторые подпрограммы часто повторяются. Для уменьшения размера исложностипрограммопределитеэти общеупотребительные подпрограммы как процедуры. Здесь мы будем называть эти процедуры подпрограммами.

Команда PROCEDURE <procedure name> является утверждением внутри программы. Он определяет начало каждой подпрограммы в программе и идентифицирует подпрограмму по имени. Имя подпрограммы может включать в себя до 10 символов. Оно должно начинаться с буквы или символа подчеркивания и может состоять из любой комбинации букв, цифр и символов подчеркивания.

После утверждения PROCEDURE <procedure name> следует серия команд, которая составляет подпрограмму. В качестве последнего оператора подпрограммы необязательно использовать оператор RETURN, так как неявный оператор RETURN автоматически выполняется после последнего оператора подпрограммы.

Когда процедура выполняется в цикле DO <procedure name>, FoxPro ищет подпрограмму в определенном порядке. Сначала находится файл, содержащий команду DO <procedure name>. Если нужной процедуры там нет, FoxPro будет искать файл процедур, открытый с помощью команды SET PROCEDURE TO. Если подпрограмма не может быть найдена в процедурном файле, или если процедурный файл не может быть открыт, FoxPro просматривает программы в цепочке выполнения.

Файлы программ начинают просматриваться с последней выполненной программы и до первой выполненной программы.

Если подпрограмма по прежнему не найдена, FoxPro ищет ее в

автономном программном файле. Если программный файл найден, программа выполняется, иначе возвращается сообщение об ошибке "File does not exist" ("Файл не найден").

PUBLIC

■ PUBLIC <memvar list>

Определяет глобальные переменные памяти или массивы.

Все переменные в памяти или массивы, созданные в Командном окне, автоматически становятся глобальными.

PUTFILE()

Представляет диалог Save As (Сохранить как)

PUTFILE([<ВырС>] [, <ВырС1>][, <ВырС2>])

<ВырС> Строка приглашения, представляемая над списком имен файлов

<ВырС1> Имя файла по умолчанию, представляемое в текстовом блоке

<ВырС2> Расширения файлов, представляемых в диалоге Save As (Сохранить как).

■ QUIT

Выход из сеанса FoxPro.

READ

Активизирует объекты, созданные командами @ ... GET и @ ... EDIT

READ

[CYCLE]

[VALID <ВырL3 | ВырN>]

Выполнение команды READ активизирует GET объекты. GET объекты это GET поля, Области редактирования текста, Текстовые кнопки, Селективные кнопки, Всплывающие меню, Блоки проверки и Списки. Все GET объекты, созданные после последней команды READ или CLEAR GETS, являются активными.

Позиционирование

Когда выдана команда READ, вы можете нажать клавишу Enter, Tab или Стрелка вниз для перемещения вперед от объекта к объекту. Нажатие клавиш Shift+Tab или Стрелка вверх осуществляет перемещение назад от объекта к объекту. Перемещение от объекта к объекту происходит в порядке появления команд GET.

Если у вас есть мышь, вы можете выбрать объект щелчком на объекте. Если вы не знакомы с операциями с мышью, обратитесь к "Началам работы" (Getting Started) по системе FoxPro.

Ввод данных

Когда вы перемещаетесь в поле GET или области редактирования текста, вы можете вводить текст или редактировать существующий текст. Стандартные средства редактирования текста FoxPro доступны в поле GET или области редактирования текста — текст может быть вырезан, скопирован и вклеен.

Выход из чтения

Есть несколько способов выхода из чтения. Перемещение вперед после последнего объекта GET или назад после первого объекта осуществляет выход из чтения (если в команде READ была включена опция CYCLE). Нажатие клавиши Escape, Ctrl+W или выбор управляющего средства, которое определено для прекращения чтения, также заканчивает чтение.

Многооконное чтение

Чтение может охватывать несколько окон. GET объекты могут располагаться в различных окнах и активизироваться одной командой READ. Когда вы перемещаетесь от объекта к объекту, объекты проходятся в порядке появления команд GET независимо от окон объектов. Окно активизируется и становится текущим окном вывода, когда ткущий объект находится в окне. Когда вы нажимаете клавиши Tab, Enter или Стрелка вниз на последнем GET объекте в окне, вы переходите на первый GET объект в следующем окне. Когда вы нажимаете клавиши Shift+Tab или Стрелка вверх на первом GET объекте в окне, вы переходите на последний GET объект в предыдущем окне. Команда SHOW GETS является примером многооконного чтения.

Вложенное чтение

Вложенное чтение может создаваться появлением команд GET и READ в процедуре, вызванной в команде READ. Чтение может быть вложено на глубину до пяти уровней. Текущий уровень чтения может быть получен функцией RDLEVEL(). Описание функции RDLEVEL() содержит пример вложенного чтения.

CYCLE

Если используется опция CYCLE, команда READ не прекращается, когда вы перемещаетесь вперед после последнего объекта GET или назад после первого объекта GET. Когда вы нажимаете клавиши Tab, Enter или Стрелка вниз на последнем GET объекте, вы переходите на первый GET объект.

Когда вы нажимаете клавиши Shift+Tab или Стрелка вверх на первом GET объекте, вы переходите на последний GET объект. Если используется опция CYCLE, кнопка прекращения, Escape, Ctrl+W, CLEAR READ или опция TIMEOUT будут прекращать чтение.

* Для того, чтобы сделать доступным окно Browse (Правка), включите заголовок окна Browse (Правка) (по умолчанию, псевдоним базы данных) в список окон.

* Для того, чтобы сделать доступным memo окно, включите псевдоним базы данных в список окон.

Если заголовок окна содержит символ, не являющийся буквой, цифрой или символом подчеркивания, включите в список окон часть заголовка до первого символа,не являющегося буквой, цифрой и символом подчеркивания. Например, если окно имеет заголовок Client List, включите в список окон Client (пробел между Client и List не является буквой, цифрой и символом подчеркивания).

Если заголовок окна содержит символ, не являющийся буквой, цифрой или символом подчеркивания, вы можете включите в список окон заголовок в кавычках. Например, если окно имеет заголовок Client List, вы можете включить в список окон "Client List".

VALID <ВырL3> | <ВырN>

Когда вы пытаетесь выйти из чтения, вычисляется опция VALID. Опция VALID может быть логическим или численным выражением или функцией, определенной пользователем, возвращающей логическое или численное значение.

Если выражение <ВырL3> вычисляется со значением "истинно", чтение прекращается. Если <ВырL3> вычисляется со значением "ложно", курсор остается в том же поле или выбирается тот же GET объект, если это возможно.

Если курсор не может остаться в том же GET поле, или не может быть выбран тот же GET объект (процедура VALID может деактивизировать GET поле или объект), вы перемещаетесь на первый GET объект.

Если опция VALID возвращает число, вы перемещаетесь на соответствующий объект. Если не существует объекта, соответствующая числу, чтение прекращается. Возвращаемое значение, не являющееся логическим или численным, рассматривается как возвращение значения "истинно" (.Т.).

Базовая команда READ

FoxPro 2.0 позволяет вам легко создавать интерфейсы, управляемые событиями (подобныеFoxPro)для ваших прикладных программ.

Совершенствование команды READ и функции READKEY(), добавление двух новых функций, WLAST() и WREAD(), и изменения оконных функций дают возможность создавать ваши собственные интерфейсы, управляемые событиями. Этот параграф предлагает метод для создания интерфейса, управляемого событиями, в Foxpro 2.0.

Эти изменения не влияют на любые прикладные программы, написанные в более ранних версиях FoxPro — ваши старые прикладные программы будут выполняться точно также, как и раньше.

В предыдущих версиях FoxPro некоторые опытные пользователи использовали циклы, зависящие от событий, для управления выбором в меню и выбором окон при создании интерфейса, управляемого событиями. Обработчик состояний должен постоянно проверять состояние системы и выполнять действия в зависимости от того, какое окно находится сверху, и какая команда меню была выбрана.

В цикле, зависящем от событий, используется команда DO WHILE для создания этого цикла (часто довольно большого). Циклы, зависящие от событий, следует избегать по следующим соображениям:

* Циклы, зависящие от событий, выполняются постоянно, используя время работы процессора.

* Циклы, зависящие от событий, сложны для программирования.

* Циклы,зависящиеотсобытий,плохоподходят для мультипрограммных систем, подобных DESQview, Windows и Macintosh MultiFinder.

Одна команда READ с несколькими окнами

В FoxPro 2.0 сделаны улучшения, которые позволяют реже пользоваться циклами, зависящими от событий. Одна команда READ, поддерживающая несколько окон, может использоваться для того, чтобы избежать большинство циклов, зависящих от событий. Одна команда READ с несколькими окнами предоставляет многосторонность и гибкость без необходимости в сложном программировании.

Мы советуем вам охватывать одной командой READ все окна, с которыми вы хотите работать. FoxPro может координировать окна, содержащие GET объекты (команды @ ... GET, блоки проверки, селективные кнопки, всплывающие меню и т. п.), с окнами, не связанными обычным образом с командой READ — окнами Browse (Правка), memo окнами, окнами редактирования текста, системными окнами FoxPro и т. п. По умолчанию, все интерактивные окна (окна Browse (Правка), "настольная оргтехника" и окна, открытые командами MODIFY FILE, MODIFY REPORT и т. п.) могут участвовать в команде READ.

Координация окон одной командой READ

Для программной координации окон, содержащих GET объекты, одной командой READ:

1. Создайте окна, которые будут содержать GET объекты.

2. Активизируйте одно из окон и выдайте команды GET, создающие GET объекты для этого окна.

3. Активизируйте другое окно и выдайте команды GET, создающие GET объекты для этого окна.

4. Выполните эту операцию для оставшихся окон.

5. Выдайте команду READ.

Программный код установки экрана, сгенерированный GENSCRN, предоставляет хороший пример того, как это выполняется.

Теперь окна будут координироваться одной командой READ. Каждый из GET объектов будет возникать в соответствующем окне, и вы можете перемещаться по GET объектам в каждом из окон. Когда вы перемещаетесь от объекта к объекту, объекты проходятся в порядке появления GET утверждений, независимо от окон объектов. Окно активизируется и становится текущим окном вывода, когда текущий объект находится в этом окне.

Когда вы нажимаете клавишу Tab, Enter или D на последнем объекте в окне, вы перемещаетесь на первый GET объект в следующем окне. Когда вы нажимаете клавиши Shift+Tab или Стрелка вверх на первом GET объекте в окне, вы перемещаетесь на последний GET объект в предыдущем окне. Вы можете перемещаться по различным GET объектам в различных окнах щелчком мыши на GET объектах.

Доступ к другим окнам во время выполнения одной команды READ может управляться опцией MODAL команды READ. Использование ключевого слова MODAL в команде READ предотвращает взаимодействие с окнами, которые не охвачены набором окон в команде READ. Когда опция MODAL используется в команде READ, вы не можете закрывать, распахивать, минимизировать или перемещать окна, не охваченные командой READ. fелчок мыши на окне, не охваченном командой READ, будет вызывать звуковой сигнал, если установлена команда SET BELL ON.

Функция READKEY() была расширена в FoxPro 2.0 для удобства управления дочерними командами READ. Использованием численного аргумента в функции READKEY() вы можете определить, как была прекращена последняя команда READ. Например, значение может указывать, что последняя команда READ была прекращена закрытием окна, командой CLEAR READ, возвращением опцией DEACTIVATE или ACTIVATE команды READ значения .T. и т. д.

Далее приводится значения, возвращаемые функцией READKEY(), в зависимости от того, как было прекращено чтение:

Значение	Событие
1	Ни одно из следующих
2	Выдана команда CLEAR READ
3	Выбрано управляющее средство прекращения
4	Закрыто окно чтения
5	Опция DEACTIVATE возвратила значение .T.
6	Вышел лимит времени чтения

READKEY()

Возвращает значение, соответствующее клавише, нажатой для выхода из команд редактирования APPEND, BROWSE, CHANGE, CREATE, EDIT, INSERT, MODIFY и READ.

READKEY()

Если данные не изменялись во время выполнения команды, возвращаемое значение будет лежать между 0 и 36. Если же данные были изменены, возвращаемое значение будет лежать между 256 и 292.

Значения, возвращаемые функцией READKEY

Клавиша(и)	Код не обновлялся	Код обновлялся	Смысл
Backspace ← Ctrl+H	0	256	Назад на один символ
CTRL+S → Ctrl+D*	1	257	Вперед на один символ
Ctrl+L Home Ctrl+A*	2	258	Назад на одно слово
End Ctrl+F	3	259	Вперед на одно слово
Ctrl+E* Shift+Tab	4	260	Назад на одно поле
Ctrl+K* Стрелка вниз Ctrl+X*			
Tab Ctrl+Enter	5	261	Вперед на одно поле
Ctrl+I Ctrl+J PgUp Ctrl+R*	6	262	Назад на один экран
PgDn Ctrl+C	7	263	Вперед на один экран
Ctrl+Q Escape	12	268	Выход без сохранения
Ctrl+End Ctrl+W	—	270	Выход с сохранением
Enter Ctrl+>	15	271	Возврат или наполнение
Ctrl+M Превышение лимита времени	20	276	Превышение лимита времени
Ctrl+Home Ctrl+]	33	289	Переключение показа меню
Ctrl+PgUp Ctrl+-	34	290	Распахивание
Ctrl+PgDn Ctrl+^	35	291	Минимизация
F1	36	292	Функциональная клавиша подсказки

* Если установлен режим SET SYSMENU OFF, или загружен FOXPLUS.FKY, будут возвращаться эти значения. FOXPLUS.FKY находится в директории GOODIES.

Если используется необязательное численное выражение <ВырN>, функция READKEY() возвращает значение, указывающее, как была прекращена последняя команда READ. Численное выражение <ВырN> может иметь любое значение. Далее приведены значения, возвращаемые функцией READKEY(<ВырN>) в зависимости от того, как была прекращена последняя команда READ:

Значение	Событие
1	Ни одно из следующих
2	Появилась команда CLEAR READ
3	Выбрано управляющее средство прекращения
4	Закрыто окно чтения
5	Опция DEACTIVATE возвратила значение .T.
6	Превышен лимит времени чтения

■ RECALL

Команда RECALL снимает отметку с записей, помеченных для удаления в текущей базе данных. Любая команда DELETE может быть отменена командой RECALL.

RECCOUNT()

RECCOUNT([<ВырN> | <ВырC>])
Возвращает число записей в базе данных.

RECNO()

RECNO([<ВырN> | <ВырC>])
Возвращает номер текущей записи

■ REINDEX

Перестраивает открытые индексные файлы

RELATION()

Возвращает выражения отношений для базы данных
RELATION(<ВырN>[, <ВырN1> | <ВырC>])
Параметры
<ВырN> Указывает, какое отношение нужно возвратить
<ВырN1> Номер рабочей области для базы данных
<ВырC> Псевдоним базы данных
Эта функция возвращает выражение отношения для базы данных, открытой в заданной рабочей области. Если отношения отсутствуют, возвращается пустая строка. Более подробную информацию о создании отношений между базами данных можно найти в описании команды SET RELATION.

Команды DISPLAY STATUS и LIST STATUS также предоставляют выражения отношений.

Параметры

<ВырN>

Численное выражение <ВырN> задает возвращаемое отношение.

Например, если <ВырN> равно 3, функция RELATION возвращает выражение отношения для отношения, созданного третьим.

<ВырN1> | <ВырC>

Функция RELATION() возвращает выражения отношений для файла базы данных в текущей рабочей области.

Вы можете получить выражения отношений для базы данных, открытой в другой рабочей области, задав номер рабочей области <ВырN1> или псевдоним базы данных <ВырC>.

Если в заданной рабочей области нет открытой базы данных, функция RELATION() возвращает пустую строку. Если нет базы данных с заданным псевдонимом, выдается сообщение "Alias not found" ("Псевдоним не найден").

REPLACE

Обновляет записи базы данных

■ REPLACE

<field1> WITH <Выр1> [ADDITIVE]

[, <field2> WITH <Выр2> [ADDITIVE]] ...

[<scope>] [FOR <ВырL>] [WHILE <ВырL1>]

Команда REPLACE заменяет данные, ранее находившиеся в поле <field1>, на данные в выражении <Выр1>. Поля базы данных, находящейся не в текущей рабочей области, должны указываться с псевдонимом базы данных.

Если указатель записи находится на конце файла в текущей рабочей области, а вы задаете поле в другой рабочей области, замещения выполняться не будут.

<field1> WITH <Выр1>

[, <field2> WITH <Выр2> ...

Заменяет данные в поле <field1> на данные в выражении <Выр1>, заменяет данные в поле <field2> на данные в выражении <Выр2> и т. д.

В случае численных полей, когда значение выражения в предложении WITH больше, чем реальная ширина поля, команда REPLACE будет производить следующее:

— Сначала, дробные позиции будут обрезаны и оставшаяся часть будет округлена.

— Если полученное значение опять не будет соответствовать, то экспоненциальное представление числа заменит заданное содержимое поля (точность будет потеряна).

— Если опять неудача, то содержимое поля будет заполнено символом звездочка.

<scope>

Вы можете указывать область охвата <scope> заменяемых записей. Заменяться будут только те записи, которые попадают в границы заданной области.

Область охвата по умолчанию для команды REPLACE это текущая запись.

ADDITIVE

Опция ADDITIVE применяется при заменах только в memo полях. Если опция ADDITIVE указана, то замещающие данные добавляются в конец текста

соответствующего memo поля. Если опция ADDITIVE не задана, то содержимое соответствующего memo поля замещается на новые данные <Выр1>.

REPORT

■ REPORT [FORM <file> | ?]
[ENVIRONMENT]
[<scope>] [FOR <ВырL>] [WHILE <ВырL1>]
[HEADING <ВырC>]
[NOEJECT]
[NOCONSOLE]
[PDSETUP]
[PLAIN]
[PREVIEW]
[TO PRINTER | TO FILE <file>]
[SUMMARY]
Команда REPORT используется для генерации отчетов под управлением файлов описания отчета, которые должны быть сгенерированы командами CREATE REPORT и MODIFY REPORT. Полученные отчеты могут быть посланы на принтер, на экран, в окно или ве4 е0текстовый файл.
Если команда REPORT выдается без дополнительных аргументов, появляется диалог Открыть файл (Open File), который предоставляет список файлов отчетов для выбора.
Более подробную информацию о создании отчетов можно найти в "Руководстве по интерфейсу" (Interface Guide) системы FoxPro.
FORM <file>
Вы можете задать отчет для печати, использованием имени файла описания отчета <file> после ключевого слова FORM.
?
Если вы используете опцию ?, появится диалог Открыть файл (Open File) со списком существующих файлов отчетов для выбора.
ENVIRONMENT
Когда вы создаете или модифицируете отчет, вы можете при желании сохранить текущее состояние среды FoxPro в файле описания отчета. Сохранение состояния среды FoxPro располагает дополнительную запись в базе данных описания отчета. Эта запись содержит имена всех открытых баз данных, индексный порядок и все отношения между базами данных. Если выиспользуете опцию ENVIRONMENT, состояние среды, сохраненное в файле описания отчета, восстанавливается.
HEADING <ВырC>
Опция HEADING используется для задания дополнительной строки заголовка, располагающейся в начале каждой страницы отчета. Если в команде REPORT одновременно используются опции HEADING и PLAIN, приоритет имеет опция PLAIN.
NOEJECT
Опция NOEJECT подавляет подачу страницы в принтере перед печатью отчета.
NOCONSOLE
Опция NOCONSOLE подавляет представление отчета на экране или в

окне, когда отчет печатается или пересылается в текстовый файл.

PDSETUP

Когда вы создаете описание отчета с помощью Редактора отчетов (Report Writer) FoxPro, вы можете задать установку принтера, определяющую внешний вид отчета при печати. Если вы сохраняете состояние среды в описании отчета, также сохраняется установка принтера. Если вы используете необязательное ключевое слово PDSETUP при печати отчета командой REPORT, установка принтера загружается и используется во время печати.

PLAIN

Если используется опция PLAIN, заголовки страниц будут возникать только в начале отчета.

PREVIEW

Использование опции PREVIEW посылает представление отчета на экран, что позволяет вам проверить содержание и компоновку отчета перед его печатью. Если используется опция PREVIEW, отчет посылается на экран для предварительного исследования и не печатается. Для печати отчета вы должны выдать другую команду REPORT FORM без опции PREVIEW.

TO PRINTER | TO FILE <file>

Если используется опция TO PRINTER, отчет посылается и на указанное устройство печати, и на экран или в окно. Если эта опция опущена, отчет представляется только на экране. Опция TO FILE посылает отчет в заданный ASCII текстовый файл <file>.

SUMMARY

Опция SUMMARY подавляет печать всех строк отчета, кроме строк итогов и подытогов.

RETURN

■ RETURN [<Выр> | TO MASTER | TO <program name>] — Возвращение управления вызывающей программе.

ROW()

ROW() — возвращает номер текушей экранной строки

RUN | !

Выполнение внешней программы

■ RUN [/N [K]] <DOS command> | <program name>

■ ! [/N [K]] <DOS command> | <program name>

Команда RUN позволяет вам запускать внешние команды DOS или

DOS-программы и затем возвращаться в FoxPro. Команда RUN может быть выдана в командном (Command) окне или внутри программ.

ВНИМАНИЕ:

1. Для того, чтобы использовать команду RUN под управлением MS-DOS, файл операционной системы COMMAND.COM должен быть в текущем каталоге или указан с помощью параметра DOS COMSPEC.

2. Не запускайте с помощью команды RUN программы, которые реорганизуют диски (например, CHKDSK) из FoxPro. Они могут так модифицировать содержимое ваших дисков, что результат их работы вызовет сбой в работе FoxPro.

В состав FoxPro входит средство управления памятью FoxSwap. Средство FoxSwap обеспечивает больше доступной оперативной памяти для команды RUN.

Использование в команде RUN режима /N или /NK дает пользователю возможность задавать объем свободной памяти для внешних программ FoxPro. Числовое значение N определяет объем освобождаемой памяти в килобайтах. Буква N при указании освобождаемого объема памяти в команду не включается.

При N=0 для команды RUN выделяется максимально возможный объем памяти.

При задании ненулевого значения FoxPro выполняет следующее:

— записывает содержимое всех буферов на диск.

— если N килобайт свободной памяти имеется, выполняется команда

RUN, в противном случае вызывается FoxSwap и команде RUN предоставляется N килобайтов оперативной памяти.

SCAN

Перемещение по базе данных и условное выполнение команд

■ SCAN [<scope>] [FOR <ВырL>] [WHILE <ВырL1>]
[<операторы>]
[LOOP]
[EXIT]
ENDSCAN

Используется для перемещение по базе данных и условного выполнения <операторов> для каждой встреченной записи, которая отвечает специфицированным условиям. Подобно команде DO WHILE команда SCAN автоматическм продвигает указатель записей (record pointer) на следующую запись и затем проверяет выполнение специфицированных условий.

См. также: DO CASE, DO WHILE, FOR ... ENDFOR.

SELECT()

SELECT([0 | 1])

Функция SELECT() вернет номер текущей выбранной рабочей области, или номер верхней неиспользуемой рабочей области (с аргументом 1).

SELECT — SQL

Назначение

■ Возвращает данные из одной или нескольких баз данных.

Синтаксис

SELECT [ALL | DISTINCT]
[<alias>.]<select_item> [AS <column_name>]
[, [<alias>.]<select_item> [AS <column_name>] ...]
FROM <database>[<local_alias>] [,<database> [<local_alias>] ...]
[[INTO <destination>]
< [TO FILE <file>[ADDITIVE] | TO PRINTER]]
[NOCONSOLE]

[PLAIN]
[NOWAIT]
[WHERE <joincondition> [AND <joincondition> ...]
[AND | OR <filtercondition> [AND | OR <filtercondition> ...]]]
[GROUP BY <groupcolumn> [, <groupcolumn> ...]]
[HAVING <filtercondition>]
[UNION [ALL] <SELECT command>]
[ORDER BY <order_item>[ASC | DESC]
[, <order_item>[ASC | DESC]...]]

Описание

SELECT является командой SQL, и используется для нахождения и возврата данных из одной или нескольких баз данных. Когда Вы используете команду SELECT для посылки запроса, FoxPro интерпритирует запрос и возвращает данные из баз(ы) данных. Команда SELECT встроена в FoxPro подобно остальным командам FoxPro. Вы можете создать запрос команды SELECT:

— в командном окне

— в программе (подобно другим командам FoxPro)

— в окне RQBE

В документации по FoxPro термины "таблица" и "база данных" являются синонимами. Строка в таблице — это тоже самое, что и запись в базе данных. Термины база данных, запись и поле используются, когда описывается представление информации, содержащейся в базе данных. Термины таблица, строка и колонка используются при обсуждение выходных запросов.

Когда установлено SET TALK ON и выполняется команда SELECT, FoxPro выводит на экран время запроса и число записей в результате. _TALLY запоминает число записей в результате запроса.

Дополнительные опции

SELECT [ALL | DISTINCT]
[<alias>.]<select_item> [AS <column_name>]
[, [<alias>.]<select_item> [AS <column_name>] ...]

SELECT опция запрашивает и специфицирует поля базы данных, константы и выражения, появляющиеся в результате запроса. По умолчанию, все строки полученные в результате запроса будут выведены на экран. Для исключения появления одинаковых строк в качестве результата запроса включите опцию.

Внимание !!!

DISTINCT можно использовать только один раз на предложение SELECT.

Каждый <select_item> генерирует одну колонку в качестве результата запроса. Если более одного <select_item> имеют одинаковое имя, Вы должны быть уверены, что для квалификации имени было включено имя псевдонима базы данных или период перед <select_item>. <select_item> может быть:

— Полем базы данных в опции FROM

— Константой, указывающей что одно и тоже значение появляется в каждой строке результата запроса

— Выражением, которое может включать функции, определенные пользователем (UDF)

UDF Предупреждения

Хотя имеются явные преимущества в разрешении использовать UDFs, есть в этом и некоторые ограничения.

1. Скорость выполнения SELECT операции ограничивается скоростью с которой выполняется функция определенная пользователем. Наивысшая скорость манипуляции с включением UDF достигается при использовании API или UDF, написанных на языке Си или ассемблере.

2. Вы можете не делать никаких предположений о вводе/выводе FoxPro или об окружении базы данных в функции определенной пользователем, вызываемой из SELECT. Как правило, вам нужно знать какая рабочая область выбрана, имя текущей базы данных или имена полей, участвующих в процессе обработки. Все это — переменные, которые зависят от того где в оптимизационном процессе вызывается функция, определенная пользователем.

3. Нет смысла изменять ввод/вывод FoxPro или окружение базы данных в UDF, вызываемой из SELECT. Как правило, результат будет непредсказуемым.

4.Есть только один надежный путь передачи значений UDF, вызываемой из SELECT — передача списка аргументов функции, когда она вызывается.

5. Если в результате экспериментов, пренебрегая рекомендациями, Вы все же добились корректной работы в данной версии FoxPro, нет гарантии, что Ваша программа будет работать правильно в следующих версиях.

В остальном можете использовать UDF свободно, по мере необходимости не забывая о пункте 1.

Следующие функции допустимы для использования с <select_item>, они являются полями или выражениями включающими поля:

* AVG(<select_item>) — Среднее по колонке числовых данных.

* COUNT(<select_item>) — Счетчик <select_items> в колонке. COUNT(*) счетчик числа строк в выходе запроса.

* MIN(<select_item>) — Определяет наименьшее значение <select_item> в колонке.

* MAX(<select_item>) — Определяет наибольшее значение <select_item> в колонке.

* SUM(<select_item>) — Сумма по колонке числовых данных.

Необязательная опция AS <column_name> указывает название колонки в выходе запроса. Это используется, когда <select_item> является выражением или содержит функцию и Вы хотите задать название, имеющее смысл.

<column_name> может быть выражением, но не должно содержать символов (например, пробелов) которые недопустимы в именах полей баз данных.

FROM <database> [<local_alias>] [, <database> [<local_alias>] ...]

FROM — обязательная опция, содержащая список баз данных, в которых находятся данные извлекаемые в результате запроса. Если база данных не открыта в рабочей области, но находится в текущем директории или в директории, описанном в пути FoxPro, появляется окно диалога "Открытие файла", через которое можно выбрать базу данных.

Если база данных не была открыта в рабочей области, она открывается и остается открытой пока не выполнится запрос.

<local_alias> — это временное имя для <database>. Если Вы указываете <local_alias>, Вы должны указывать <local_alias> вместо имени базы данных везде в команде SELECT. <local_alias> не имеет эффекта в окружении FoxPro.

Если несколько баз данных имеют поля с одинаковыми именами необходимо квалифицировать их, для чего может быть использован короткий <local_alias>.

INTO <destination>

INTO необязательная опция, указывающая где сохраняется результат запроса. Если INTO указано, то результат на экране не отражается. Это означает, что при включении опций INTO и TO в один запрос, TO будет игнорироваться. Если опция INTO не вклю-

чается, то результат запроса направляется на экран и кроме того может быть направлен на принтер или в файл включением опции TO.

<destination> для опции INTO:

* ARRAY <array> — Результат запроса запоминается в массиве временных переменных с именем <array>.

* CURSOR <cursor> — Результат запроса сохраняется в курсоре с именем <cursor>. Если Вы указываете имя открытой базы данных, FoxPro закрывает базу и создает курсор с этим именем без предупреждения, если SET SAFETY OFF. После выполнения команды SELECT временный курсор становится открытым и активизируется только для чтения. Вы можете осуществлять просмотр. После того как Вы закроете временный курсор, он удаляется. Курсор может существовать как временный файл в драйвере SORTWORK.

* DBF <database> | TABLE <database> — Результат запроса сохраняется в базе данных с именем <database>. Если Вы указываете имя открытой базы данных, FoxPro закрывает базу и снова открывает ее без предупреждения, если SET SAFETY OFF. Если Вы не указали расширения, то по умолчанию принимается расширение .DBF. После выполнения команды SELECT база данных становится открытой и активной.

[TO FILE <file> [ADDITIVE] | TO PRINTER]]

Если опция INTO не включается в запрос и включается необязательная опция TO, вы можете направить вывод результата запроса в текстовый файл или на принтер в дополнение к выводу на экран. Необязательный ключ ADDITIVE направляет вывод в конец существующего файла <file>. Если Вы не включаете обе опции TO и INTO, то результат запроса по умолчанию почвляется на экране пока неопределена опция NOCONSOLE.

При выводе результатов запроса, колонкам присваиваются названия в соответствии со следующими правилами:

* Если <select_item> — поле с уникальным именем, выходная колонка имеет то же имя.

* Если более чем один <select_item> имеет одинаковое имя, например CUST.ZIP и STATE.ZIP, выходные колонки будут иметь имена ZIP_A и ZIP_B. Для <select_item> с десятисимвольными именами, имя изменяется путем добавления символов подчеркивания и буквы.

* Если <select_item> — выражение, то выходная колонка именуется как EXP_A. Другие выражения именуются EXP_B, EXP_C, и так далее.

* Если <select_item> содержит функцию, такую как COUNT(), выходная колонка имеет имя COUNT_A. Другой <select_item> содержащий функцию SUM() на выходе имеет имя SUM_B.

NOCONSOLE

Необязательная опция предотвращающая вывод на экран или принтер результатов запроса. Может использоваться в независимости от использования опции TO. С опцией INTO ее значение игнорируется.

PLAIN

Необязательная опция предотвращающая вывод на экран названий колонок в результате запроса. Может использоваться в независимости от использования опции TO. С опцией INTO ее значение игнорируется.

NOWAIT

Необязательная опция вызывающая прокрутку экрана при заполнении.

Обычно, при заполнении экрана FoxPro требует нажатия клавиши для вывода на экран последующих порций результатов запроса. NOWAIT осуществляет прокрутку экрана без пауз. Может использоваться в независимости от использования опции TO. С опцией INTO ее значение игнорируется.

[WHERE <joincondition> [AND <joincondition> ...]

[AND | OR <filtercondition> [AND | OR <filtercondition>...]]]

Запрашивает данные из нескольких баз данных. Сообщает FoxPro на необходимость включения в результат запроса только некоторых записей.

<joincondition> указывает поля, которые связывают базы данных в опции FROM. Если Вы указываете более одной базы данных, то необходимо указать условие связи для каждой базы данных после первой.

Будьте внимательны при объединении баз данных с пустыми полями, так как FoxPro допускает пустые поля. Так например при связывании двух баз данных имеющих пустые поля в 100 и 400 записях соответственно, результат запроса будет содержать 40.000 записей из пустых полей. Для того, чтобы обойти это, используйте функцию EMPTY().

Множественный связи должны объеденяться с использованием оператора AND. Каждое <joinconditions> имеет форму:

<field1> <сравнение> <field2>

где <field1> — поле из одной базы данных, <field2> — поле из второй базы данных и <сравнение> принимает одно из следующих значений:

<сравнение>

---------- =

< >, !=, #

= =

>

> =

<

< =

При использовании = сравнения со строками, его действие зависит от установки команды SET ANSI. Если SET ANSI OFF, сравнение строк трактуется в терминах пользователей X-Base, в противном случае трактуется как стандарт ANSI.

<filtercondition> указывает условие, на основании которого записи включаются в результат запроса. Запрос может включать несколько условий фильтра, связанных операторами AND или OR. Можно использовать также оператор NOT или функцию EMPTY() для проверки пустых полей. Условие фильтра может иметь одну из следующих форм:

Форма: <field1> <comparison> <field2>

Пример: customer.cust_id = payments.cust_id

Форма: <field> <comparison> <Выpession>

Форма: <field> <comparison> ALL (<subquery>)

Когда фильтр включает ключевое слово ALL, <field> должно встретиться в условии сравнения для каждого значения сгенерированного подзапросом перед включением записи в результат запроса.

Форма: <field> <comparison> ANY | SOME (<subquery>)

Пример: taxrate < ANY (SELECT taxrate FROM customer WHERE state = 'MI')

Когда фильтр включает ключевое слово ANY или SOME, <field> должно встретиться в условии сравнения хотя бы для одного значения сгенерированного подзапросом перед включением записи в результат запроса.

Форма: <field> [NOT] BETWEEN <start_range> AND <end_range>

Проверка, лежит ли значение в диапазоне.

Форма: [NOT] EXISTS (<subquery>)

Пример: EXISTS (SELECT * FROM invoice WHERE customer.zip = invoice.zip)

Проверка, удовлетворяет по меньшей мере одна колонка условию. Когда фильтр включает EXISTS, условие истинно пока пожзапрос не станет пустым.

Форма: <field> [NOT] IN <value_set>

Пример: customer.zip NOT IN ('43411','43506','43667')

<field> — является частью набора значений, перед включением записи в результат запроса.

Форма: <field> [NOT] IN (<subquery>)

Пример: customer.cust_id IN (SELECT tranfile.cust_id FROM tranfile WHERE tranfile.item='Fo

<field> — является частью набора значений возвращаемого подзапросом, перед включением записи в результат запроса.

Форма: <field> [NOT] LIKE <ВырС>

Пример: customer.state NOT LIKE 'OH'

Этот фильтр ищет <field> соответствие указанному <ВырС>. Вы можете включить символы % и _ как часть <ВырС>. _ — указывает единичный неопределенный символ в строке а % — указывает последовательность неопределенных символов в строке.

Подзапрос — это SELECT внутри SELECT и должен быть ограничен скобками. Вы можете организовать множественные подзапросы одного уровня (не вложенные) в опции WHERE. Подзапросы могут иметь множественные условия связи.

[GROUP BY <groupcolumn> [, <groupcolumn> ...]]

Необязательная опция которая группирует строки в запросе на основании значения в одной или более колонках. <groupcolumn> может быть регулярным полем базы данных, полем базы данных которое включено в SQL функцию, или числовым выражением указывающим положение колонки в результирующей таблице. (Номер самой левой колонки 1).

HAVING <filtercondition>

Сообщает о необходимости включения групп в результат запроса на протяжении <filtercondition>. <filtercondition> не может содержать подзапрос.

HAVING <filtercondition> используется с GROUP BY для указания критерия, по которому группа включается в результат запроса. HAVING может включать так много условий фильтрации, как это необходимо, они связываются операторами AND или OR. Можно также использовать оператор NOT для обращения логических выражений.

HAVING без GROUP BY действует подобно опции WHERE. Вы можете использовать локальные псевдонимы и функции полей в опции HAVING.

[UNION [ALL] <SELECT command>]

Необязательная опция, комбинирует окончательный результат одной команды SELECT с окончательным результатом другой <SELECT command>.

По умолчанию UNION проверяет комбинацию результатов на предмет исключения повторения строк. Для избежания исключения повторов необходимо указать ключевое слово ALL.

Можно использовать скобки для комбинирования нескольких опций UNION.

UNION правила:

* Нельзя использовать UNION для комбинации подзапросов.

* Обе команды SELECT должны выводить одинаковое число колонок.

* Каждая колонка в результате запрос одной команды SELECT должна иметь тот же тип данных, что и соответствующая колонка в другой <SELECT command>.

* Только последняя <SELECT command> может иметь опцию ORDER BY и ORDER BY должна описывать выходную колонку по номеру. Если ORDER BY включена, эффект сказывается на всех результатах.

[ORDER BY <order_item>[ASC | DESC]
[, <order_item> [ASC | DESC]...]]

Сортирует результат запроса на основании одной или нескольких колонок. Каждый <order_item> соответствует колонке в результате запроса и может быть:

* Полем из FROM базы данных , которое также является <select_item> в главной опции SELECT (не в подзапросе)

* Числовым выражением указывающим положение колонки в в результирующей таблице. Можно указать необязательный ключ DESC , для сортировки результата в убывающем порядке в соответствии с <order_item(s)>. ASC указывает на порядок по возрастанию и принимается по умолчанию.

SET CENTURY

■ SET CENTURY ON | OFF

Включает или выключает четырехзначное описание года.

SET DATE

■ SET DATE [TO] AMERICAN | ANSI | BRITISH | FRENCH | GERMAN | ITALIAN | JAPAN | USA | MDY | DMY | YMD

Эта команда устанавливает формат даты для выражений с датами. Далее приведен список допустимых типов даты и их форматы.

AMERICAN mm/dd/yy
ANSI yy.mm.dd
BRITISH dd/mm/yy
FRENCH dd/mm/yy
GERMAN dd.mm.yy
ITALIAN dd-mm-yy
JAPAN yy/mm/dd
USA mm-dd-yy
MDY mm/dd/yy
DMY dd/mm/yy
YMD yy/mm/dd

По умолчанию установлено SET DATE TO AMERICAN.

SET DECIMALS

■ SET DECIMALS TO [<ВырN>] — определяет минимальное число десятичных позиций, которые должны быть отображены в качестве результата работы числовых функций и вычислений.

SET DEFAULT

Задание дисковода и/или каталога

■ SET DEFAULT TO [<ВырС>]

Команда SET DEFAULT заставляет FoxPro выполнять полную команду MS-DOS изменения каталога (CD) и осуществлять операции ввода/вывода данных на указанном с помощью выражения <ВырС> дисководе или на дисководе и в указанном каталоге.

Вы можете специфицировать в выражении <ВырС> либо имя дисковода, либо имя дисковода и имя каталога, имя дочернего каталога или использовать принятые в MS-DOS сокрашения (\ или ..).

SET DEVICE

Прямой вывод на экран или в окно, на принтер или в файл

■ SET DEVICE TO SCREEN | TO PRINTER | TO FILE <file>

Команда SET DEVICE контролирует отображение результатов работы команд @...SAY. Вывод данных может быть направлен в окно (SCREEN), на
принтер (PRINTER) или в файл (FILE) с именем <file>. Команда @...GET
ничего не посылает на устройство (DEVICE).

TO SCREEN

Вывод команды @ ... SAY направляется на экран.

TO PRINTER

Если использована команда SET DEVICE TO PRINTER, то выводные данные посылаются на принтер. Если значения экранных координат, отмеченных в команде @...SAY, меньше, чем те, которые были в предыдущей команде @...SAY, то результатом будет выталкивание страницы из принтера.

TO FILE <file>

Если использована команда SET DEVICE TO FILE <file>, то весь вывод
данных направляется в файл с именем <file>.

SET EXACT

Спецификация точности совпадения при подборе

■ SET EXACT ON | OFF

Если установлено SET EXACT OFF, то две строки могут сравниваться
даже если они имеют различную длину. Если строки совпадают до тех пор,
пока одна из них не закончилась, то они принимаются равными.

Если установлено SET EXACT ON, то две строки считаются равными, если каждый символ одной строки совпадает с соответствующим символом в другой строке и длина обоих строк одинакова.

По умолчанию установлено SET EXACT OFF.

Сравнения строк В FoxPro есть два оператора, которые проверяют эквивалентность.

Одиночный знак равенства сравнивает два значения одинакового типа.

Этот оператор применяется для сравнения чисел, дат и логических данных и может использоваться для сравнения символьных данных. Однако, когда сравниваются символьные выражения с помощью этого оператора, результат может отличаться от того, который Вы ожидаете. Символьные выражения сравниваются посимвольно слева направо, пока один из символов будет не равен другому, или пока выражение в правой

части не достигнет конца (SET EXACT OFF), или пока оба выражения не закончаться (SET EXACT ON).

Двойной знак равенства "==", также можно использовать для сравнения символьных строк. Если два символьных выражения сравниваются с использованием этого оператора, то они будут равны только в том случае, если все их символы совпадают и они имеют одинаковую длину, так как в этом случае заключительные пробелы участвуют в сравнении (независимо от установки SET EXACT).

В таблице, приведенной ниже, символ "_" используется в сравниваемых значениях вместо пробела.

Сравнение значений "=" "=" "=="
EXACT OFF EXACT OFF EXACT ON/OFF
"abc"="abc" .T. .T. .T.
"ab" ="abc" .F. .F. .F.
"abc"= "ab" .T. .F. .F.
"abc"="ab_" .F. .F. .F.
"ab" ="ab_" .F. .T. .F.
"ab_"="ab" .T. .T. .F.
"" ="ab" .F. .F. .F.
"ab" ="" .T. .F. .F.
"___"="" .T. .T. .F.
"" ="___" .F. .T. .F.
TRIM("___")="" .T. .T. .T.
""=TRIM("___") .T. .T. .T.

SET FILTER

- SET FILTER TO [<ВыpL>] — используется для определения групп записей внутри текущей базы данных, которые отвечают условию <Выр>. После выдачи этой команды в базе данных проявляются только те записи, которые отвечают указанному условию. Все команды, которые обращаются к базе данных, зависят от сформулированного условия.

SET NEAR

- SET NEAR ON | OFF — определяет положение указателя записей, которое он должен занять при после неудачного поиска записи. Для ON — на запись, которая расположена непосредственно за ближайшей подходящей записью. Для OFF — в конец файла.

Команды ускоряющиеся при использовании технологии Rushmore

AVERAGE DISPLAY REPLACE
BROWSE EDIT REPORT
CALCULATE EXPORT SCAN
CHANGE JOIN SORT
COPY TO LABEL SUM
COPY TO ARRAY LIST TOTAL
COUNT LOCATE

DELETE RECALL

SET ORDER

■ SET ORDER TO [<ВырN> | [TAG] <tag name>
[IN <ВырN1> | <ВырC>]

База данных может иметь несколько открытых индексных файлов одновременно. Однако, только один индексный файл (главный индексный файл) или тег из составного .CDX индексного файла (главный тег) определяет порядок доступа и просмотра записей. Некоторые команды (например SEEK) используют главный индексный файл или главный тег для поиска записи.

Данная команда позволяет определить главный индексный файл или главный тег. Главный индекс также может определять последовательность записей в процессе их обработки.

Индексные файлы могут быть открыты вместе с базой данных, включением опции INDEX в команду USE. Если база данных имеет связанный с ней структурный индексный файл, то он открывается автоматически вместе с базой данных. после открытия базы данных, Вы можете открывать/закрывать индексные файлы для базы данных, используя команду SET INDEX.

[TAG] <tag name> [OF <.cdx file>]

Для назначения тега .CDX индексного файла в качестве главного тега, включите <tag name>. <tag name> может быть из структурного индексного файла или любого открытого .CDX составного индексного файла.

Если существуют одинаковые имена тегов в открытых .CDX составных
индексных файлах включите опцию OF <.cdx file>.

IN <ВырN1> | <ВырC>

SET ORDER назначает главный индексный файл или тег для базы данных, открытой в текущей рабочей области. можно определять главный индексный файл или тег для базы данных, открытой в произвольной рабочей области, для чего надо указать псевдоним <ВырC> или номер рабочей области <ВырN1>.

SET RELATION

Спецификация отношения между двумя и более базами данных

■ SET RELATION TO
[<Выр1> INTO <ВырN> | <ВырC>
[, <Выр2> INTO <ВырN1> | <ВырC1> ...]
[ADDITIVE]]

С помощью команды SET RELATION TO вы можете связать два открытых файла баз данных. Перед установкой связи, одна база данных (родительская база данных) должна быть открыта в текущей рабочей области и другая база данных (дочерняя база данных) должна быть открыта в другой рабочей области. После этого Вы можете подать команду SET RELATION для установления связи.

В каждый момент времени, когда указатель записи в родительской базе данных перемещается, также перемещается на соответствующую запись указатель записей в связанном дочернем файле. Если подходящая запись не найдена в связанной дочерней базе данных, то указатель записи для этой базы данных перемещается в самый конец файла.

Связываемые базы данных обычно имеют общие поля. Например, предположим, что база данных (CUSTOMER.DBF) содержит информацию о покупателях. Она содержит

поля для имени, адреса и уникального номера покупателя. Вторая база данных (BILLING.DBF) содержит информацию о поставщиках. Она также содержит поле для номера покупателя, и поля различных платежных характеристик.

Вы можете использовать SET RELATION для связи двух баз данных по их общему полю — полю номера покупателя. Для установки связи, дочерняя база данных должна быть проиндексирована по общему полю. После установки связи, при перемещении указателя записи на запись с данным номером покупателя в родительской базе данных (CUSTOMER), указатель записи в дочерней базе данных (BILLING) переместится на запись с тем же номером покупателя.

<Выр1>

Основное выражение связи <Выр1> обеспечивает связь между двумя базами данных (родительской и дочерней). выражение связи обычно является индексным выражением для главного управляющего индексного выражения дочерней базы данных, хотя может быть и числовым выражением.

Дочерняя база данных обычно индексируется по символьным, числовым полям или полям дат. Индекс для дочерней базы данных может быть единичным элементом — .IDX индекс, или множественным структурным или независимым составным .CDX индексом. Если индекс является составным, указывается подходящий индексный тег или порядок дочерней базы данных.

SET ORDER команда может использоваться для указания индексного тега, который упорядочивает дочернюю базу данных.

Например, рассмотрим базы данных CUSTOMER и INVOICE, описанные выше. Предположим, что дочерняя база данных (INVOICE) будет индексироваться и упорядочиваться по номеру покупателя при помощи команды:

SET ORDER TO TAG cust_id

Для связи баз данных CUSTOMER и INVOICE по номеру покупателя

используется команда SET RELATION, которая указывает индексное выражение в качестве связующего выражения <Выр1>:

SET RELATION TO cust_id INTO invoice

Дочерняя база данных должна быть не индексирована, пока выражение

для связи <Выр1> будет числовым. Если Вы подаете команду SET RELATION

с нечисловым выражением связи и дочерняя база данных не является индексированной, высвечивается сообщение "Database is not ordered"(База

данных не упорядочена).

Если <Выр1> является числовым, <Выр1> действует при перемещении

указателя записи в родительской базе данных. Указатель записи в дочерней базе при этом перемещается на запись с номером <Выр1>.

Для устранения всех связей во всех рабочих областях, необходимо

подать команду SET RELATION TO.

INTO <ВыpN> | <ВыpC>

Дочерняя база данных идентифицируется рабочей областью, в которой

она открыта или ее псевдонимом. Включайте для дочерней базы данных номер рабочей области <ВыpN>, или псевдоним базы данных <ВыpC>.

<Выр2> INTO <ВыpN1> | <ВыpC1> ...

Вы можете создать множественные связи в текущей рабочей области

при помощи единичной команды SET RELATION, для этого включите туда список связей (<Выр1> INTO <ВыpN> | <ВыpC>, <Выр2> INTO <ВыpN1> | <ВыpC1>, ...) разделенных запятыми.

ADDITIVE

Необязательное ключевое слово ADDITIVE может быть задано в команде SET RELATION. Когда оно использовано, то все предыдущие отношения в текущей рабочей области остаются в силе, в противном случае все они разрушаются и создается новая связь на их месте.

SET RELATION OFF

- SET RELATION OFF INTO <ВырN> | <ВырC> — отменяет установленное отношение между текущей и другой БД.

SET SKIP

Создание отношений связи один ко многим между базами данных.

SET SKIP TO [<alias1> [, <alias2>] ...]

Вы можете установить отношение связи между базами открытыми в разных рабочих областях при помощи команды SET RELATION. Когда указатель записи перемещается по родительской базе данных, указатель записи в родительской базе данных перемещается на первую соответствующую запись. Запись, на которую перемещается указатель записи в дочерней базе данных определяется выражением отношения связи в SET RELATION. Если отношение связи установлено один к одному — для каждой записи в родительской базе данных указатель записи перемещается только на первую соответствующую запись в дочерней базе данных. Если соответствующая запись в дочерней базе данных не найдена, указатель записи перемещается на конец файла.

Часто дочерняя база данных содержит множество записей, которые соответствуют одной записи в родительской базе данных. SET SKIP позволяет установить отношение связи один ко многим между записью в родительской базе данных и множеством записей в дочерней базе данных. Когда Вы проходите через родительский файл, указатель записи показывает на одну и ту же запись, пока не будут пройдены все связанные записи в дочернем файле.

Отношение один ко многим осуществляется в начале также, как и отношение один к одному. Во первых, отношение связи между родительской и дочерней базой данных устанавливаются командой SET RELATION. Затем, SET SKIP создает множественные отношения связи.

[<alias1> [, <alias2>] ...]

Если родительская база данных связана с несколькими дочерними базами данных, Вы можете включить список псевдонимов (<alias1>, <alias2>, ...), разделенного запятыми для создания отношений связи один ко многим с каждой дочерней базой данных.

В командах поддерживающих диапазон (DISPLAY, LIST, и т.д.), записи в родительской базе данных будут повторяться для каждого значения соответствующей записи в дочерней базе данных.

Для удаления отношения связи один ко многим в текущей рабочей области используйте команду SET SKIP TO без дополнительных аргументов, при этом отношение связи один к одному остается в силе и разрушается только командой SET RELATION TO.

SET TALK

- SET TALK ON | OFF

Включает или выключает вывод на экран вывод результатов выполнения команд.

SET()

Синтаксис

SET(<ВырС> [,1])

Функция SET () возвращает статус различных SET ... TO и SET ON или

OFF команд. Не все SET команды могут использоваться в этой функции.

Таблица, приведенная ниже, перечисляет команды SET, которые могут быть включены в данную функцию. SET () с аргументом ([,1]) возвращает

дополнительную информацию.

Функция SET() идентична функции SYS(2001).

SET () понимает четырех-символьные аббревиатуры всех FOXPRO SET

ключевых слов (например RELA или PRIN).

<ВырС>

Возвращает информацию о SET команде, которую Вы указали в символьном выражении <ВырС>. Возвращаемое значение, которое принимает команда SET может быть числовой или символьной строкой.

1

Если включен данный необязательный аргумент, возвращается дополнительная информация о SET команде, однако эта информация возвращается не для всех команд. Команды, которые имеют возвращаемую дополнительную информацию, отмечены в таблице, приведенной ниже.

Команда SET Возвращаемое значение
Alternate — ON или OFF
Alternate,1 — Альтернативный файл
Ansi — ON или OFF
Autosave — ON или OFF
Bell — ON или OFF
Blink — ON или OFF
Bloksise — Числовое выражение
Brstatus — ON или OFF
Carry — ON или OFF
Century — ON или OFF
Clear — ON или OFF
Clock — ON или OFF
Color — Атрибуты цвета
Compatible — ON или OFF
Confirm — ON или OFF
Console — ON или OFF
Currency — LEFT или RIGHT
Currency,1 — Символьное выражение
Cursor — ON или OFF
Date — Символьное выражение
Debug — ON или OFF
Decimals — от 0 до 18
Default — Драйвер по умолчанию;SYS(2003) — для директория
Deleted — ON или OFF
Delimiters — ON или OFF
Delimiters,1 — Символьное выражение

Development — ON или OFF
Device — PRINTER или SCREEN
Display — Не поддерживается, используйте SYS(2006)
Dohistory — ON или OFF
Echo — ON или OFF
Escape — ON или OFF
Exact — ON или OFF
Exclusive — ON или OFF
Fields — ON или OFF
Fields,1 — Список полей
Filter — ON или OFF
Fixed — ON или OFF
Fullpath — ON или OFF
Heading — ON или OFF
Help — ON или OFF
Help,1 — Имя файла
History — ON или OFF
Hours — 12 или 24
Intensity — ON или OFF
Lock — ON или OFF
Logerrors — ON или OFF
Mackey — Символьное выражение
Margin — от 0 до 254
Mark — Символ маркера
Memowidth — от 8 до 256
Menu — ON или OFF
Mesage — Номер строки экрана
Mesage,1 — Символьное выражение
Mouse — ON или OFF
Multilocks — ON или OFF
Near — ON или OFF
Notify — ON или OFF
Odometer — от 1 до 32767
Optimize — ON или OFF
Order — Индексный файл или тег (с опциями ASCENDING/DESCENDING)
Path — Путь
Point — Символ указателя
Printer — ON или OFF
Printer,1 — Порт или файл
Procedure — Имя файла процедур
Refresh — от 0 до 32000
Reprocess — ON или OFF
Resourse — ON или OFF
Resourse,1 — Текущий файл ресурсов
Safety — ON или OFF
Scoreboard — ON или OFF
Separator — Символ разделитель
Shadows — ON или OFF

Space — ON или OFF
Status — ON или OFF
Step — ON или OFF
Sticky — ON или OFF
Sysmenu — ON или OFF или AUTOMATIC
Talk — ON или OFF
Talk,1 — WINDOW, NOWINDOW или имя окна
Textmerge — ON или OFF
Textmerge,1 — Ограничители поглощающие текст
Topc — Символьное или логическое выражение
Typeahead — от 0 до 32000
Udfparms — VALUE или REFERENCE
Unique — ON или OFF
View — Не поддерживается

■ SKIP [<ВырN>] [IN <ВырN1> | <ВырC>] — перемещает указатель записи с его текущего положения на <ВырN> записей.

■ SORT

SORT TO <file> ON <field1> [/A] [/D] [/C]
[, <field2> [/A] [/D] [/C] ...]
Команда SORT сортирует активную базу данных по указанным полям в файл <file>.
Команда SORT выполняется в возрастающем порядке: [/A], если не указан убывающий порядок сортировки [/D]. Необязательный параметр [/C] задает сортировку с игнорированием верхнего и нижнего регистра. Необязательный параметр /C может быть использован совместно с параметрами /A и /D. Если вводятся два параметра, используется только одна косая черта (например /DC или /AC).

■ STORE

STORE <Выр> TO <memvar list> | <array> <memvar> <array> = <Выр>
Запись данных в переменную памяти или в массив
Символ равенства также может использоваться для задания значения переменной памяти, элементу массива или для инициализации массива. Переменная памяти или массив должны располагаться слева от знака равенства; записываемое значение должно располагаться справа от знака равенства.

■ UPDATE

Обновление содержимого базы данных с помощью данных из другой базы данных
UPDATE ON <key field> FROM <ВырN> | <ВырC>
REPLACE <field1> WITH <Выр1> [, <field2> WITH <Выр2> ...]
[RANDOM]
Команда UPDATE обновляет файл базы данных, открытый в текущей рабочей области, данными из файла, который открыт в другой рабочей области. Вы можете обновлять поля в текущей базе данных выборочно.
Команда UPDATE позволяет откорректировать данные в активном файле базы данных с использованием файла FROM

<alias> (псевдоним). Файл <alias> должен быть активен в одной из неактивных рабочих областей.

Корректировка производится на основании значений ключевого поля <key>. Имя ключевого поля <key> должно совпадать в обоих базах данных.

Записи в файле <alias> должны быть индексированы или отсортированы по полю <key>.

<key field>

Активная база данных и база данных из которой берется информация для обновления должны иметь общее поле. Это общее поле, ON <key field>, управляет обновлением. Активная база данных обязательно должен быть индексирована по полю <key field> или должна быть отсортирована по полю <key field>. Осуществление команды UPDATE улучшается, если обновляющая база данных также отсортирована или проиндексирована.

FROM <ВырN> | <ВырC>

Активная база данных обновляется информацией FROM(из) , базы данных расположенной в другой рабочей области. Поэтому необходимо указать номер <ВырN> или псевдоним <ВырC> этой рабочей области.

REPLACE <field1> WITH <Выр1> ...

Команда UPDATE заменяет поле <field1> в активной базе данных на

обновляющее выражение <Выр1>. Можно обновить сразу несколько полей в активной базе данных, для чего необходимо включить списки полей

(<field2>,<field3> ...) и соответствующих выражений для обновления

этих полей (<Выр2>, <Выр3> ...).

Заметим, что для каждой записи в активной базе данных, может существовать несколько подходящих записей в обновляющей базе данных. В

этом случае содержимое записи активной базы данных обновляется информацией из каждой подходящей записи.

Кроме того, в активной базе данных несколько записей могут иметь

одинаковый ключ. В этом случае обновляется только первая запись.

RANDOM

Если обновляющая база данных не индексирована или не отсортирована

в порядке возрастания, необходимо включить опцию RANDOM. В противном случае некоторое число записей могут остаться необновленными.

UPDATED()

Функция UPDATE () возвращает логическое значение "истина" (.T.) если последняя операция READ изменила какие либо данные в соответствующих GET полях или объектах.

USED()

USED([<ВырN> | <ВырC>])

Функция USED () возвращает логическое значение "истина" (.T.) если база данных открыта в рабочей области.

Предметный указатель