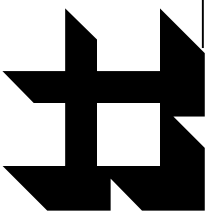

ГЛАВА 6

Проектирование баз данных

Свободные таблицы

В Visual FoxPro каждая таблица может существовать в одном из двух состояний: либо в виде свободной таблицы, представляющей собой файл DBF, не ассоциированный ни с одной из баз данных, либо в виде таблицы базы данных, включенной в какую-либо базу данных. При этом, если таблица включена в базу данных, то она не может быть включена ни в какую другую базу данных. Для таблиц, ассоциированных с базой данных, можно установить ряд свойств, которыми не могут обладать свободные таблицы. Это такие свойства, как триггеры, правила на уровне полей и записей и постоянные отношения между таблицами.

В связи с тем, что в ранних версиях FoxPro таблицы назывались базами данных, существует некоторая путаница в терминах. В Visual FoxPro таблицей называют файл с расширением dbf и связанные с ним файлы, имеющие то же самое имя, что и сам файл, но с расширением fpt, для хранения полей типа `Memo` и `General`, и с расширением cdx — для хранения индексов (так называемый структурный индексный файл).

То есть, увидев в любом файловом менеджере файлы типа

```
MyTable.DBF, MyTable.CDX, MyTable.FPT,
```

вы поймете, что это — таблица. Но это еще не все. Чтобы окончательно вас запутать, заметим, что таблица — это еще и некий *образ файла* с расширением dbf, открытый в указанной *сессии данных* и в указанной *рабочей области* (рис. 6.1).

Имейте в виду, что в абсолютном большинстве случаев под термином *"таблица"* подразумевается именно *"образ файла"*, т. е. файл, уже открытый через команду `USE` (или каким-либо еще способом) в среде Visual FoxPro. Если же подразумевается именно *файл* DBF, то, как правило, это оговаривается особо. Таблицы могут быть свободными (т. е. создаваться и использоваться вне баз данных), а могут находиться в составе баз данных.

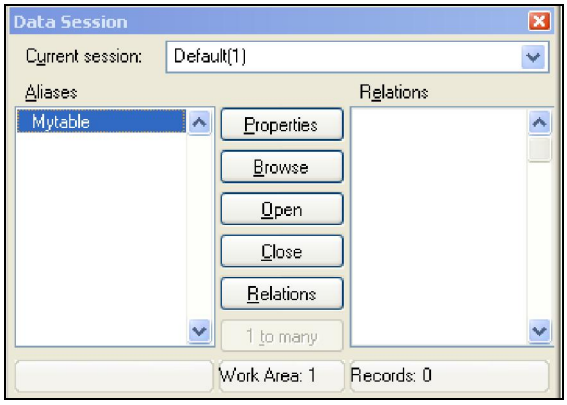

Рис. 6.1. Таблица, открытая в сессии данных по умолчанию в рабочей области

Контейнер базы данных

В Visual FoxPro под базой данных понимают "таблицу таблиц", т. е. файл на диске, имеющий расширение dbc, и сопутствующие ему файлы с расширением dcx (структурный индексный файл) и dct (для хранения memo-полей).

Если вы видите на диске файлы:

```
MyBase.DBC, MyBase.DCX, MyBase.DCT
```

это база данных.

Более правильно было бы применять к базам данных термин *"контейнер базы данных"*, поскольку база данных содержит не только информацию о составляющих ее таблицах, но и об индексах, отношениях между таблицами, процедурах и функциях, перемещаемых вместе с базой данных.

По своей сути, файл DBC — это обычный файл DBF, только с измененным расширением (как и большинство других файлов, используемых в Visual FoxPro). Отличие от простого файла DBF заключается только в содержимом 28 байта заголовка (считая, что первый байт имеет порядковый номер 0). В файле базы данных в этом байте заполнен 3 бит (чего нет в DBF-файлах). То есть ASCII код записанного там символа не менее 4 (обычно 7), в то время, как у DBF-файлов его содержимое наоборот не превышает 3.

Поскольку файл базы данных — это обычный файл DBF, вы можете открыть его как таблицу и просмотреть содержимое.

```
USE MyBase.dbc AGAIN
```

Указывать расширение файла, как и опцию `AGAIN`, в этом случае обязательно. Без указания расширения vfp посчитает, что его расширение dbf, и ошибется. А опция `AGAIN` нужна потому, что файл базы данных может быть уже открыт командой `OPEN DATABASE`, и без этой опции вы также получите сообщение об ошибке.

Название файла базы данных может содержать до 128 символов, включая пробелы, русские символы, цифры и некоторые специальные символы.

> ***Замечание***
>
> Как для файла базы данных, так и вообще для всех рабочих таблиц использующихся в проекте, следует выделять специальную папку (директорию). Эта рекомендация относится как к этапу разработки проекта, так и к поставке готового приложения клиентам.
>
> Желательно файл базы данных располагать в той же папке, где и включенные в него файлы DBF.

Таблица вовсе не обязательно включается в состав базы данных. И в этом случае у некоторых программистов возникает резонный вопрос: а зачем вообще включать таблицы в базы данных, если можно работать со свободными таблицами?

- Использование файла базы данных расширяет возможности таблиц DBF.

  Например, в файле DBF в принципе нельзя дать полю название, содержащее более 10 символов, но если он включен в базу данных, то название поля может уже содержать до 128 символов. Никакие правила (`RULE`), значения по умолчанию (`DEFAULT`), триггеры и кое-что другое попросту невозможны в файле DBF вне файла базы данных. Точнее так: невозможно их автоматическое выполнение.

- Использование файла базы данных позволяет выполнять операции, которые крайне сложно организовать другими способами.

Например, такая операция как *"транзакция"* может быть реализована только среди таблиц DBF, включенных в базу данных. В принципе этот процесс можно организовать и со свободными таблицами, но это потребует от программиста значительных усилий. А такой замечательный объект, как обновляемый Local View! Сколько усилий требовалось при программировании в FoxPro 2.x для реализации того, что этот объект делает автоматически!

В общем, использование файла базы данных серьезно облегчает жизнь программиста.

Кодовые страницы

А кодовые страницы порой, наоборот, осложняют жизнь (если, конечно, ничего о них не знать!). Чаще всего именно они являются причиной появления сообщений на различных форумах программистов: "Помогите, на экране — кракозябры!"

Кодовая страница — это средство поддержки наборов символов и раскладок клавиатуры для различных стран. Кодовая страница — это таблица, связывающая используемые программой коды символов с клавишами клавиатуры и знаками на экране.

Все записанные в таблицах символы на диске хранятся в виде чисел. Visual FoxPro при загрузке таблицы определяет его кодовую страницу, находит число, записанное в таблице, определяет соответствие между этим числом и определенным символом ко-

довой таблицы, и именно этот символ выводит на экран монитора. Таких таблиц соответствия в Visual FoxPro имеется много. Вот их список (табл. 6.1).

Таблица 6.1. *Список кодовых страниц, поддерживаемых Visual FoxPro*

Кодовая страница	Платформа	Идентификатор кодовой страницы
437	U.S. MS-DOS	x01
620 *	Mazovia (Polish) MS-DOS	x69
737 *	Greek MS-DOS (437G)	x6A
850	International MS-DOS	x02
852	Eastern European MS-DOS	x64
857	Turkish MS-DOS	x6B
861	Icelandic MS-DOS	x67
865	Nordic MS-DOS	x66
866	Russian MS-DOS	x65
874	Thai Windows	x7C
895 *	Kamenicky (Czech) MS-DOS	x68
932	Japanese Windows	x7B
936	Chinese Simplified (PRC, Singapore) Windows	x7A
949	Korean Windows	x79
950	Traditional Chinese (Hong Kong SAR, Taiwan) Windows	x78
1250	Eastern European Windows	xC8
1251	Russian Windows	xC9
1252	Windows ANSI	x03
1253	Greek Windows	xCB
1254	Turkish Windows	xCA
1255	Hebrew Windows	x7D
1256	Arabic Windows	x7E
10000	Standard Macintosh	x04
10006	Greek Macintosh	x98
10007 *	Russian Macintosh	x96
10029	Macintosh EE	x97

*Не определена, если вы включили `CODEPAGE=AUTO` в ваш файл конфигурации.

Используя другие кодовые страницы, приложения могут правильно отобразить символы из других алфавитов. Надо помнить, что не всегда символы некоторой кодовой страницы могут быть преобразованы к символам другой кодовой страницы.

Как видно из таблицы, русская кодовая страница определяется как 1251. Если кодовая страница файла отличается от 1251, на экране мы можем увидеть те самые "кракозябры", при виде которых некоторые программисты впадают в панику (рис. 6.2).

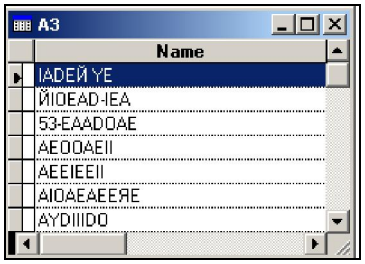

Рис. 6.2. Неверно установленная кодовая страница

Visual FoxPro отображает данные таблицы в соответствии с кодовой страницей. По умолчанию это текущая кодовая страница Windows. Однако вы можете принудительно указать кодовую страницу, используемую по умолчанию, в конфигурационном файле (CONFIG.FPW). Укажите в файле конфигурации строку

```
CODEPAGE=1251
```

Кодовая страница прописывается в заголовке таблицы при создании и проверяется Visual FoxPro при открытии. Но бывают случаи, когда кодовая страница нулевая. Нулевую кодовую страницу требуют и некоторые старые программы, которые по тем или иным причинам до сих пор используются.

Так вот, если кодовая страница нулевая, то при открытии таблицы Visual FoxPro предложит ее выбрать (рис. 6.3).

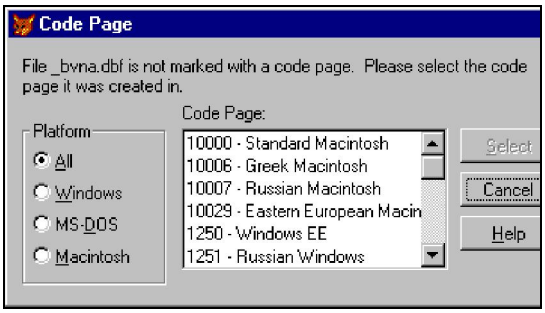

Рис. 6.3. Выбор кодовой страницы

Если диалоговое окно **Code Page** вам не нужно, можно его отменить, задав команду

```
SET CPDIALOG OFF
```

При выборе новой кодовой страницы (Select) она пропишется в заголовок, и в соответствии с ней будут отображаться данные. Если нажать кнопку **Cancel**, кодовая страница останется без изменения, но с отображением текста могут быть проблемы.

Существует программа CPZERO, которая меняет кодовую страницу (она находится в каталоге Visual FoxPro ..\Tools\Cpzero). В этом случае ее нужно добавить в проект и в качестве параметра передавать номер кодовой страницы, который надо установить:

```
do cpzero with main, 866
```

Кодовую страницу можно изменить вручную при наличии редактора View. Для этого нужно просто заменить соответствующий байт. Более подробно о структуре таблиц DBF рассказывается в *главе 15*. Пример смены кодовой страницы приведен в *главе 28*.

Нормализация данных

Создание правильной структуры базы данных имеет очень важное значение. Много проектов потерпели неудачу именно потому, что неверно была спроектирована сама структура данных. Для того чтобы структура была правильной, необходимо ознакомиться с теорией нормализации. Пренебрежение нормализацией делает структуру базы данных запутанной, а саму базу — ненадежной в работе. В качестве примера ненормализованной базы данных рассмотрим прайс — лист любой компьютерной компании. В прайс-листах очень часто можно увидеть записи такого вида (табл. 6.2).

Таблица 6.2. *Выписка из прайс-листа компьютерных компаний*

D01910S Сист. Блок cABinet Mini Pro P4-2.4/ 256Mb/ VIDEO/ 80Gb/ FDD/ COMBO/ LAN цена $459.37 гарантия 12 месяцев фирма "АБВ" тел. 777-77-77
Pentium 506-2660(533)/512-2ch/SATA 80Gb/ DVD-CDRW/ int i915 grafics цена $484 гарантия 2 года фирма "ГДЕ" тел. 888-88-88 icq 888888888

Visual FoxPro поддерживает четыре типа отношений между таблицами: "один-к-одному", "один-ко-многим", "много-к-одному", "много-ко-многим". Прежде чем перейти к основам проектирования таблиц и баз данных, остановимся подробнее на типах отношений между ними.

Отношение "один-к-одному"

Отношение "один-к-одному" характерно для таблиц, в которых одной записи первой таблицы соответствует одна запись второй таблицы.

Например, если одна таблица будет содержать данные о фирме-продавце, а вторая — о ее телефоне (если он один), то мы получим отношение "один-к-одному". Чаще всего такие данные располагают в одной таблице. И правда, для чего иметь две?

Отношение "один-ко-многим"

Такое отношение характерно для таблиц, в которых одной записи первой таблицы соответствуют несколько записей второй таблицы.

Например, если первая таблица будет содержать наименование компьютера, а вторая — его комплектующие, то мы получим типичный пример отношения "один-ко-многим".

Отношение "много-к-одному"

Это отношение, обратное предыдущему. Если рассматривать один отдельно взятый компьютер, то совокупность его компонентов будет "многим", а сам компьютер — "один".

Отношение "много-ко-многим"

Такое отношение применимо к таблицам, в которых одна запись первой таблицы может соответствовать нескольким записям второй таблицы, и наоборот, одна запись второй таблицы может соответствовать нескольким записям в первой таблице. Примером такого отношения может служить выдача со склада потребителю дискет. Тут мы видим две группы товаров:

- на складе могут храниться дискеты разных производителей;
- потребитель может получить со склада дискеты разных производителей.

Ключевое поле

Ключевое поле — это такое поле (или набор полей), по содержимому которого можно однозначно идентифицировать запись таблицы. То есть по значению ключевого поля всегда однозначно можно сказать о какой записи идет речь и не может существовать двух записей с одинаковыми значениями ключевого поля.

Внешний ключ — это поле таблицы, содержащее в себе значение ключевого поля другой таблицы. То есть просто ссылка на запись другой таблицы.

Естественный ключ — это поле или набор полей, которые имеют некий физический смысл. Например, табельный номер, номер паспорта и т. п.

Суррогатный ключ — это поле, которое не имеет никакого физического смысла и введено исключительно с целью однозначной идентификации записи. Информация, которая в нем содержится, никак логически не связана с информацией из прочих полей той же таблицы.

Фактически самый главный и определяющий критерий, по которому можно судить о том, пригодно ли данное поле для использования его в качестве ключевого или нет — это *однозначность идентификации записи*. Здесь имеется в виду, действительно ли значение выбранного поля для каждой записи уникально?

С суррогатным ключом все понятно — мы сами формируем его значение без участия со стороны пользователя, поэтому можем проследить за уникальностью, а вот как тут дело обстоит у естественного ключа?

На первый взгляд кажется, что тоже все в порядке. Разве может номер паспорта быть не уникальным? Или табельный номер? Оказывается, еще как может!

Во-первых, любая информация, вводимая пользователем, заведомо может содержать ошибки. Люди могут ошибаться, это аксиома. Кто-то ошибается чаще, кто-то реже, но ошибаются все. Часть ошибок, безусловно, можно отловить программно, но далеко не все.

Например, если речь идет о некотором цифробуквенном номере (номер паспорта) и пользователь ввел "`123 АБВ`", а надо было "`423 АБВ`", то синтаксически это правильно, но по содержанию — это ошибка. Если впоследствии окажется, что теперь надо ввести новый номер паспорта, но уже "`123 АБВ`", то программа откажется это сделать, поскольку такой номер уже есть. А на что его исправить — неизвестно, поскольку документов, с которых он был введен, уже нет.

Во-вторых (и это, пожалуй, более существенно), информация имеет тенденцию меняться. Например, не так давно была произведена замена паспортов в Российской Федерации, и если бы программа опиралась в качестве идентификатора на его номер, то теперь все пришло бы в большой беспорядок в связи с этой заменой.

Опять же, не так давно в Российской Федерации были введены так называемые "Идентификационные Номера Налогоплательщика", т. е. "ИНН". Казалось бы, ну вот он, уникальный идентификатор контрагента. Но наша родная бюрократия и здесь сумела все испортить. Оказалось, что ИНН нарушает все критерии уникальности, в связи с некоторыми особенностями его формирования.

Таким образом, в общем случае, естественный ключ не обеспечивает главного — однозначности идентификации записи по значению ключевого поля. А если все-таки обеспечивает, то он перестает быть естественным в первоначальном смысле этого слова, поскольку вынуждает пользователя приспосабливаться и изворачиваться.

Так почему же, несмотря на столь явные аргументы против, все равно есть достаточно большое количество приверженцев использования естественного ключа?

Здесь существует явное непонимание того, что суррогатный ключ — это информация, которая нигде, никоим образом не предоставляется пользователю. Суррогатный ключ — это некоторая скрытая, невидимая пользователем "внутренняя" механика.

Пользователь не должен знать, как устроена программа изнутри.

Вывод однозначен — *суррогатные ключи предпочтительнее по всем критериям перед естественными ключами.*

Используйте собственное ключевое поле во всех без исключения таблицах. Даже если вам кажется, что без него можно обойтись.

Называйте внешние ключи так же, как и соответствующие ключевые поля — такой способ наименования внешних ключей очень облегчает понимание, о чем, собственно, идет речь при написании программы.

Ограничивайте название ключевого поля 10 символами — причина здесь в том, что количество символов в названии индексного тега не может быть больше 10. А по ключевому полю обязательно следует построить индекс. Если количество символов в ключевом поле будет больше 10, то название индексного тега будет отличаться от названия ключевого поля. А это не очень хорошо в том смысле, что серьезно затруднит программирование, поскольку использовать этот индекс вы будете сплошь и рядом.

Значение ключевого поля должно присваиваться один раз в момент создания записи и не меняться все время существования этой записи. Крайне нежелательно модифицировать значение ключевого поля. Лучше строить программу таким образом, чтобы не возникало необходимости в подобной модификации.

Ни в коем случае не давайте пользователям возможность самостоятельно изменять значение ключевых полей. Более того, пользователь вообще не должен подозревать об их существовании. Это должен быть исключительно внутренний механизм обеспечения целостности базы данных.

Не пытайтесь навесить на ключевые поля какие-либо еще функции, кроме однозначной идентификации записи.

Проектирование нормализованных баз данных

При проектировании структур баз данных нужно решить три взаимосвязанные задачи:

- быстрый доступ к данным;
- исключение избыточности данных, поскольку это ведет не только к нерациональному использованию дискового пространства, но и к ошибкам в работе программы;
- взаимосвязь данных, при которой при изменении данных в одной таблице изменяются и данные связанных с ней таблиц.

Процесс уменьшения избыточности данных называется *нормализацией*. Теория нормализации предлагает в целях уменьшения избыточности при большом количестве данных разделять данные на несколько взаимосвязанных таблиц. Согласно той же теории существуют пять нормальных форм, причем каждая последующая нормальная форма должна удовлетворять требованиям предыдущей нормальной формы. Мы рассмотрим первые три, поскольку четвертая и пятая формы при практическом проектировании не применяются.

В качестве примера приведем таблицу Pricelist (табл. 6.3).

Таблица 6.3. *Выписка из прайс-листа компьютерных компаний*

D01910S Сист. блок cABinet Mini Pro P4-2.4/ 256Mb/ VIDEO/ 80Gb/ FDD/ COMBO/ LAN
цена $459.37 гарантия 12 месяцев фирма "X1" тел. 555-55-55
Pentium 506-2660(533)/512-2ch/SATA 80Gb/ DVD-CDRW/ int i915 grafics
цена $484 гарантия 2 года фирма "X2" тел. 666-66-66
Pentium 506-2660(533)/512-2ch/SATA 80Gb/ DVD-CDRW/ int i915 grafics
цена $488 гарантия 1 год фирма "X3" тел. 777-77-77
D01646S Сист. блок cABinet Mini Pro C4-2.53/ 256Mb/ VIDEO/ 80Gb/ FDD/ Combo/ LAN
цена $286.15 гарантия 12 месяцев фирма "X4" тел. 888-88-88

Итак, есть прайс-лист. Вроде бы все в порядке. Однако, с точки зрения Visual FoxPro, с такими данными практически невозможно работать. Первая проблема, с которой столкнется компьютерная программа, — это идентификация записей. То есть как определить, о какой записи вообще идет речь. С точки зрения человека — нет проблем: вторая запись сверху первого листа. Это для человека понятно. Но для компьютера — это полная бессмыслица. Он не знает, что такое "лист", что такое "сверху", что такое "строка листа". Для решения этой проблемы создают специальное служебное поле, в которое записывают некий уникальный идентификатор. То есть некое значение, которое однозначно идентифицирует именно данную запись и значение которого не может никак, ни при каких обстоятельствах, повториться в какой-либо другой записи. Такое поле называют ключом. По-другому — ключевым полем. Итак, для идентификации строки запишем в это поле порядковый номер строки. Вторая проблема данного прайс-листа — это анализ его содержимого. Человек "понимает", что слово "Pentium" — это название марки процессора. Следующие за ним числа — это тактовая частота. При этом для человека все равно, в каком именно месте строки будет располагаться это слово. Человек прочитает всю строку и по некоторым своим собственным критериям произведет ее разбор и вычленит нужный фрагмент. Компьютерные программы пока еще не настолько умные. Им все еще надо явно указывать, что символы, например, с первого по 20 — это название марки процессора. На практике это означает, что для компьютера всю эту длинную символьную строку надо разбить на фрагменты (поля), где каждый фрагмент будет хранить только одну "сущность". Одно поле — для названия марки процессора, другое — для значения его тактовой частоты, третье — для цены и т. п. В результате, вместо линейного списка получим таблицу. Однако эта таблица — это, по сути, все еще "человеческий" способ отображения информации. Ну, добавили некое служебное поле. Ну, разбили строку на фрагменты. По сути, ничего ведь не изменилось. Только ради этого не стоило все это затевать. Очередные проблемы связаны с вводом и модификацией данных. В прайс-листе встречается масса повторяющейся информации. Например, слово "Pentium" встречается несколько раз. Если вводить это слово в каждой строке, то, скорее всего, будут встречаться постоянные ошибки. Кто-то введет слово "Pentim", кто-то "Petium", а кто-то вообще перепутает марку. Это значит, что всю такую повторяющуюся информацию надо вынести в отдельные таблицы — справочники. Пользователь один

раз введет это название в справочнике, а при заполнении прайс-листа будет просто выбирать нужное значение из справочника.

Однако не все так просто. Если мы будем вводить в прайс-лист собственно слово "Pentium" (копировать слово из справочника), то есть риск повторить ту же ошибку. В самом справочнике ввели название с ошибкой, и эта ошибка многократно размножилась во всех прайс-листах. Это значит, что вводить (изменять) информацию надо только в одном месте, но отображать эту введенную информацию там, где необходимо. Для решения этой проблемы в таблице-справочнике также вводится ключевое поле. А в таблице прайс-листа создается поле, содержащее нужное (выбранное) значение. Такие поля носят название "внешний ключ". То есть это, безусловно, "ключ", но не ключ данной записи, а ключ записи другой (внешней) таблицы.

Вот здесь и кроется принципиальное отличие "человеческого" и "компьютерного" восприятия. Человек видит слово "Pentium" в таблице прайс-листов и воспринимает собственно прайс-лист просто как набор строк. А с точки зрения компьютера, прайс-лист — это результат объединения информации из разных таблиц.

Это самое "объединение таблиц" в компьютерных системах носит название "связи" или "отношения". Буквальный перевод английского слова "Relation". Отсюда и термин "реляционные базы". Собственно, описанный процесс в общих чертах и есть процесс нормализации базы данных. Однако на практике никто не доводит нормализацию до ее логического завершения. База данных всегда остается в той или иной степени денормализована.

Опять же, из-за разницы человеческого и компьютерного "восприятия" существует два типа ключевых полей, разделяемых по способу формирования их значений. Это суррогатные ключи и естественные ключи.

Суррогатный ключ — содержимое ключевого поля формируется самой программой и никак, никоим образом не связано с содержимым записи.

Естественный ключ — содержимое ключевого поля — это некая реальная сущность внешнего мира. Например, название марки процессора — Pentium.

На самом деле — это не взаимоисключающие, а взаимодополняющие понятия. В большинстве таблиц так или иначе будут присутствовать оба этих ключа.

Суррогатный ключ — для обеспечения ссылочной целостности и непротиворечивости данных для компьютера.

Естественный ключ — для связи с пользователем. Именно по значению естественного ключа пользователь будет анализировать информацию.

Если суррогатный ключ уникален в пределах таблицы без каких-либо условий, то естественный ключ уникален при определенных ограничениях. Например, только среди не пустых значений. Или только среди не удаленных записей.

Давайте применим на практике теорию нормализации и создадим нормализованную многотабличную структуру.

Первая нормальная форма

Требования первой нормальной формы таковы:

- таблица не должна иметь повторяющихся записей;
- каждый атрибут отношения должен хранить одно-единственное значение и не являться ни списком, ни множеством значений;
- таблица не должна иметь повторяющихся групп полей;
- строки не должны быть упорядочены;
- столбцы не должны быть упорядочены.

Для удовлетворения первому условию таблица должна иметь уникальный индекс. В качестве такого индекса предусмотрим поле `Id`. Для уникальности индекс вовсе не обязательно должен быть числом. Возможно использование функции `SYS(2015)` или других способов. Важно одно — значения этого поля не должны повторяться. В этом случае выполняется первое условие создания первой нормальной формы (табл. 6.4).

Таблица 6.4. *Приведение таблицы к первой нормальной форме*

ID	Содержимое таблицы
1	D01910S Сист. блок cABinet Mini Pro P4-2.4/ 256Mb/ VIDEO/ 80Gb/ FDD/ COMBO/ LAN цена $459.37 гарантия 12 месяцев фирма "X1" тел. 555-55-55
2	Pentium 506-2660(533)/512-2ch/SATA 80Gb/ DVD-CDRW/ int i915 grafics цена $484 гарантия 2 года фирма "X2" тел. 666-66-66
3	Pentium 506-2660(533)/512-2ch/SATA 80Gb/ DVD-CDRW/ int i915 grafics цена $488 гарантия 1 год фирма "X3" тел. 777-77-77
4	D01646S Сист. блок cABinet Mini Pro C4-2.53/ 256Mb/ VIDEO/ 80Gb/ FDD/ Combo/ LAN цена $286.15 гарантия 12 месяцев фирма "X4" тел. 888-88-88

Такая структура не может называться нормализованной, поскольку не соответствует второму требованию первой нормальной формы. Каждый атрибут отношения должен хранить одно-единственное значение и не являться ни списком, ни множеством значений. Следовательно, мы должны просмотреть содержимое поля таблицы и разделить составные атрибуты на отдельные строки и столбцы (табл. 6.5).

Таблица 6.5. *Структура таблицы, приведенная к первой нормальной форме*

Поля таблицы	Запись1	Запись2	Запись3	Запись4
Уникальный индекс — ID	1	2	3	4

Наименование фирмы	X1	X2	X1	X4
Телефон	555-55-55	666-66-66	777-77-77	888-88-88
Процессор	cABinet Mini Pro P4-2.4	Pentium 506-2660(533)/	Pentium 506-2660(533)	cABinet Mini Pro C4-2.53
Оперативная память	256Mb	512-2ch	512-2ch	256Mb
Жесткий диск	80Gb	SATA 80Gb	SATA 80Gb	80Gb
Видеокарта	VIDEO	int i915 grafics	int i915 grafics	VIDEO
CD-ROM/ DVD-ROM	COMBO	DVD-CDRW	DVD-CDRW	COMBO
Дисковод	FDD			FDD
Цена	$459.37	$484	$488	$286.15
Срок гарантии	12	24	12	12

Теперь наша информация уже похожа на настоящую таблицу, однако она все еще далека от совершенства. Очевидно, что повторяющиеся значения являются источником потенциальных проблем. Достаточно ошибиться при вводе, например, названия организации, и формально это будет уже совсем другая организация. А найти подобные ошибки в объемной таблице — проблема из проблем. Во-вторых, может измениться телефон организации, что бывает хоть и нечасто, но пренебрегать такой возможностью не стоит. В-третьих, в этой таблице мы дважды видим наименование модели, хотя нам хватило бы и одного. Дальнейшая нормализация базы данных направлена именно на борьбу с избыточностью данных. Борьба с избыточностью данных выгодна со всех сторон — как со стороны повышения удобства пользования данными и поддержания их в целостном виде, так и со стороны эффективности обработки и хранения данных аппаратными средствами сервера баз данных.

Вторая нормальная форма

Требования второй нормальной формы:

- она соответствует первой нормальной форме;
- любое неключевое поле однозначно определяется набором ключевых полей.

Из этих требований следует, что таблицы должны иметь составной индекс.

Другими словами, таблица находится во 2НФ, если оно находится в 1НФ, и при этом все неключевые атрибуты зависят только от ключа целиком, а не от какой-то его части.

Чтобы устранить избыточность данных, необходимо разделить нашу таблицу на несколько таблиц (в данном случае — 2) .

Выделим отдельную таблицу для организаций с их атрибутами — "Наименование" и "Телефон" (табл. 6.6 и 6.7).

Таблица 6.6. *Структура таблицы № 1*

ID	Идентификатор записи
COD_ORG	Код организации
PROC	Модель процессора
OZU	Модель ОЗУ
GD	Модель жесткого диска
VCARD	Модель видеокарты
CDR	Модель CD-ROM
FDD	Модель дисковода
PRICE	Цена
GAR	Срок гарантии

Таблица 6.7. *Структура таблицы "Организация"*

COD_ORG	Код организации
NAIM	Наименование организации
TEL	Телефон

Третья нормальная форма

База данных удовлетворяет требованиям третьей нормальной формы, если она:

- удовлетворяет требованиям второй нормальной формы;
- ни одно из неключевых полей таблицы не идентифицируется с помощью другого неключевого поля.

Чтобы привести базу данных к 3НФ, необходимо устранить функциональные зависимости между неключевыми атрибутами отношения. Другими словами, факты, хранимые в таблице, должны зависеть только от ключа. Посмотрим на уже созданные нами две таблицы более внимательно. В нашем случае присутствует явное излишество в виде полного описания модели компьютера. Если вынести эти модели в отдельные таблицы, а в основную таблицу записывать только их коды, это существенно упростит работу.

Создадим таблицы-справочники для хранения моделей процессоров, жестких дисков, оперативной памяти, видеокарт, CD-ROM, дисководов FDD. Тогда наши таблицы приобретут следующий вид (табл. 6.8—6.16).

Таблица 6.8. *Основная таблица*

ID	Идентификатор записи
COD_ORG	Код организации
COD_MOD	Код модели компьютера
PRICE	Цена
GAR	Срок гарантии

Таблица 6.9. *Структура таблицы "Организация"*

COD_ORG	Код организации
NAIM	Наименование организации
TEL	Телефон

Таблица 6.10. *Структура таблицы "Модель компьютера"*

COD_MOD	Код модели компьютера
COD_PROC	Код модели процессора
COD_OZU	Код модели оперативной памяти
COD_GD	Код модели жесткого диска
COD_VCARD	Код модели видеокарты
COD_CDR	Код модели CD-ROM
COD_FDD	Код модели FDD

Таблица 6.11. *Справочник процессоров*

COD_PROC	Код процессора
NAME	Наименование

Таблица 6.12. *Справочник ОЗУ*

COD_OZU	Код ОЗУ
NAME	Наименование

Таблица 6.13. *Справочник моделей жестких дисков*

COD_GD	Код НЖД
NAME	Наименование

Таблица 6.14. *Справочник*

COD_VCARD	Код модели видеокарты
NAME	Наименование

Таблица 6.15. *Справочник процессоров*

COD_CDR	Код модели CD-ROM
NAME	Наименование

Таблица 6.16. *Справочник процессоров*

COD_FDD	Код дисковода FDD
NAME	Наименование

Денормализация

Нормализация — это фактически исправление огрехов, допущенных при проектировании БД. Как известно, часто оказывается проще не допускать этих ошибок с самого начала, нежели спроектировать БД кое-как, а потом оптимизировать (чем мы, собственно, и занимались). Хотя наиболее удачные базы данных всегда в той или иной степени нормализованы, у нормализованной базы данных есть один существенный недостаток, связанный со снижением ее производительности. Чтобы понять, почему это происходит, необходимо знать, что при выполнении базой данных запросов или транзакций существенную роль начинают играть такие факторы, как использование центрального процессора и памяти, а также система ввода-вывода. Короче говоря, для обработки транзакций и запросов в случае нормализованной базы данных к центральному процессору, памяти и системе ввода-вывода предъявляются существенно большие требования, чем когда эта база данных ненормализована. Ведь для получения требуемой информации или обработки существующих данных нормализованная база данных должна сначала найти все необходимые таблицы, а затем объединить содержащуюся в них информацию.

Тем не менее иногда представляется логичным несколько отступить от полной нормализации базы данных. Дело в том, что программировать приложения, работающие с хорошо нормализованной базой данных, довольно трудно. Разработчики приложе-

ний, которым приходится создавать программы, предназначенные для конечных пользователей, часто в качестве источников данных используют подзапросы, представления или соединения нескольких таблиц. Это может привести к тому, что приложение станет работать настолько медленно, что им просто не будут пользоваться. Кроме того, базы данных с высокой степенью нормализации отличаются очень сложной структурой и повышенной фрагментарностью групп данных. Поэтому иногда в целях повышения производительности прибегают к процессу денормализации. *Денормализация* — это процесс, который разрешает управляемое дублирование информации, введение производных столбцов, а также, в редких случаях, повторение данных в целях достижения лучшей производительности системы. Процесс денормализации данных может включать в себя перепроектирование отдельных таблиц, а также дублирование информации с целью уменьшения числа таблиц, подлежащих объединению для поиска необходимых данных; все эти меры приводят к снижению требований к системе ввода-вывода, а также уменьшают загрузку центрального процессора. Как при нормализации данных, так и при их денормализации существует золотая середина; однако в любом случае от вас требуется глубокое понимание сущности реальных данных, а также особых требований, предъявляемых к деятельности вашей организации.

Создание и модификация свободных таблиц

После определения структуры таблиц можно приступать к их созданию. В Visual FoxPro вы можете создавать как таблицы, входящие в базу данных, так и отдельные таблицы. Файл таблицы, как и любой другой файл в системе Windows, может содержать до 128 символов, пробелы, русские символы, цифры и некоторые специальные символы. Существует ряд рекомендаций по именованию таблиц, связанных с упрощением работы с ними:

- **не используйте в названии русские символы** — причина этой рекомендации в том, что VFP разрабатывался прежде всего для англоязычных пользователей, и использование в нем символов другого языка — это уже последующее дополнение;
- **не используйте в названии пробелы** — в принципе, ошибок использование пробелов не вызовет, но несколько усложнит сам процесс программирования, поскольку имена и пути доступа, содержащие пробелы, необходимо заключать в кавычки. Просто добавит лишней заботы — не забывать кавычки. А зачем усложнять себе жизнь, когда без этого легко можно обойтись;
- **ограничивайте длину названия 8 символами и не используйте в названии цифр и спецсимволов** — в отличие от аналогичной рекомендации в отношении наименования файла базы данных этому есть причина. Причина не настолько явная, чтобы ее описать в двух словах. Но цепочка рассуждений, приведшая к этой рекомендации, основана на рекомендации по наименованию ключевых полей таблиц и наименований индексных тегов;

- **не называйте таблицу так же, как и одно из его полей** — разумеется, ошибки это не вызовет, но усложнит понимание написанного кода самим программистом. Не всегда с ходу можно однозначно определить, что речь идет именно о таблице, а не о поле таблицы. А если еще и их названия совпадают, то понимание еще более осложняется;
- **не используйте для названия одно из зарезервированных в VFP слов** — опять же, ошибки это не вызовет, но резко снизит "читабельность" кода. Ведь зарезервированные слова автоматически подсвечиваются определенным цветом (если вы используете стандартный текстовый редактор VFP), и станет проблемой отличить опцию или команду от имени таблицы;
- **не используйте псевдонимы таблицы внутри базы данных** — в данном случае речь идет о том, что внутри базы данных можно присвоить таблице псевдоним, отличный от имени файла DBF. Речь не идет об опции `ALIAS` в команде `USE`. Это несколько другое. Если вы войдете в режим модификации структуры таблицы и перейдете на закладку **Table**, то увидите, что в опции `Name` стоит имя, совпадающее с именем файла DBF. Вот это-то имя и можно изменить, присвоив таблице некоторый "псевдоним", по которому и будут обращаться к указанной таблице. Это может внести путаницу в сам процесс программирования.

Расположение таблицы

Как для файла базы данных, так и вообще для всех рабочих таблиц, использующихся в проекте, следует выделять специальную папку (директорию). Эта рекомендация относится как к этапу разработки проекта, так и к поставке готового приложения клиентам.

Желательно файл базы данных располагать в той же папке, где и включенные в него файлы DBF.

По большому счету, конечно, можно держать файл базы данных в одной директории, а включенные в него файлы DBF, — в другой. Но это усложняет сам процесс разработки проекта. Например, создание резервной копии в простейшем случае заключается в простом копировании директории с рабочими файлами в другое место. Если же рабочие файлы отделены от файла базы данных, то потребуется уже копирование двух директорий. Соответственно усложнится код.

Создание таблицы

Создать таблицу можно:

- с помощью мастера;
- с помощью конструктора;
- программным способом.

Создание таблиц с помощью мастера

Для создания базы данных с помощью мастера необходимо пройти 5 шагов.

1. На первом шаге нужно выбрать одну из перечисленных в списке баз данных, а если вы хотите добавить в список свою базу данных — нажать кнопку **Add…** (рис. 6.4).

Рис. 6.4. Шаг первый

2. На втором шаге мы должны указать, будет ли таблица создана свободной, либо будет входить в состав базы данных (рис. 6.5).
3. На третьем шаге определим поля таблиц (рис. 6.6).

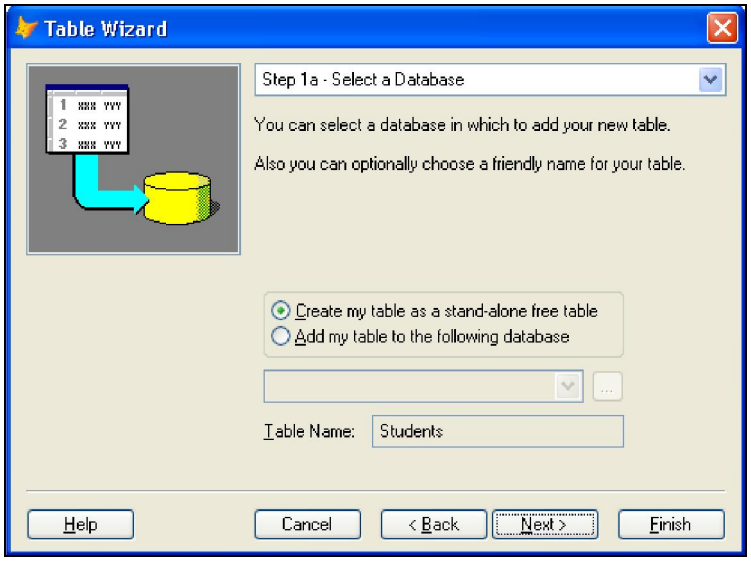

Рис. 6.5. Шаг второй

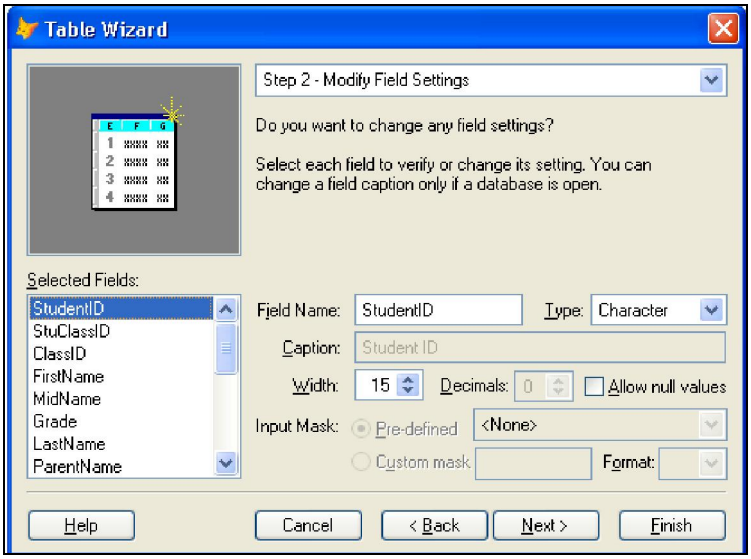

Рис. 6.6. Шаг третий

4. На четвертом шаге определим индексы для таблиц, один из них будет **Primary Key** — главный индекс таблицы (рис. 6.7).
5. И на последнем, пятом, шаге предлагается либо сохранить базу данных для дальнейшего использования, либо сохранить и модифицировать ее, используя конструктор базы данных.

- `Save database for later use`
- `Save database and modify it in the Database Designer`

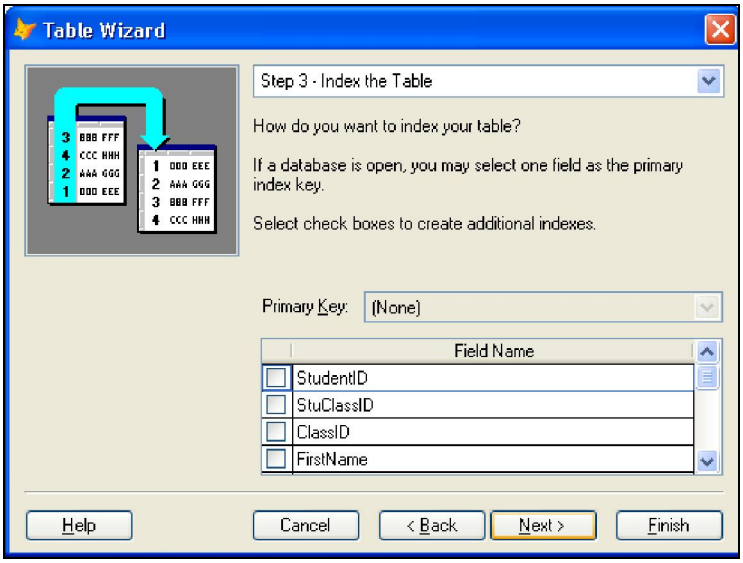

Рис. 6.7. Шаг четвертый

Создание таблиц с помощью Конструктора таблиц

Несмотря на простоту использования мастеров, основным инструментом разработчика все же являются Конструкторы. Конструкторы отличаются от мастеров свойством "повторной входимости": на любом этапе проектирования можно воспользоваться Конструктором и продолжить процесс с того шага, на котором вы остановились ранее. Так, например, в таблицах можно изменять структуру таблицы или ее индексы. Кроме того, конструкторы обладают большими функциональными возможностями по контролю данных. Рассмотрим порядок работы с конструктором таблиц.

В главном меню Visual FoxPro необходимо выбрать: **File | New | Table**. Если выбрать **New File**, то появится диалоговое окно с именем таблицы, по умолчанию это имя table1.dbf, его можно изменить. После сохранения имени файла появляется конструктор таблиц (рис. 6.8).

Как видно на рис. 6.8, конструктор имеет три вкладки. Первая вкладка служит для ввода полей таблицы, вторая — для ввода индексов, в третьей — отображается статус таблицы. По умолчанию всегда открывается первая вкладка. Чтобы структура таблицы была определена, нужно ввести информацию о полях таблицы (табл. 6.17).

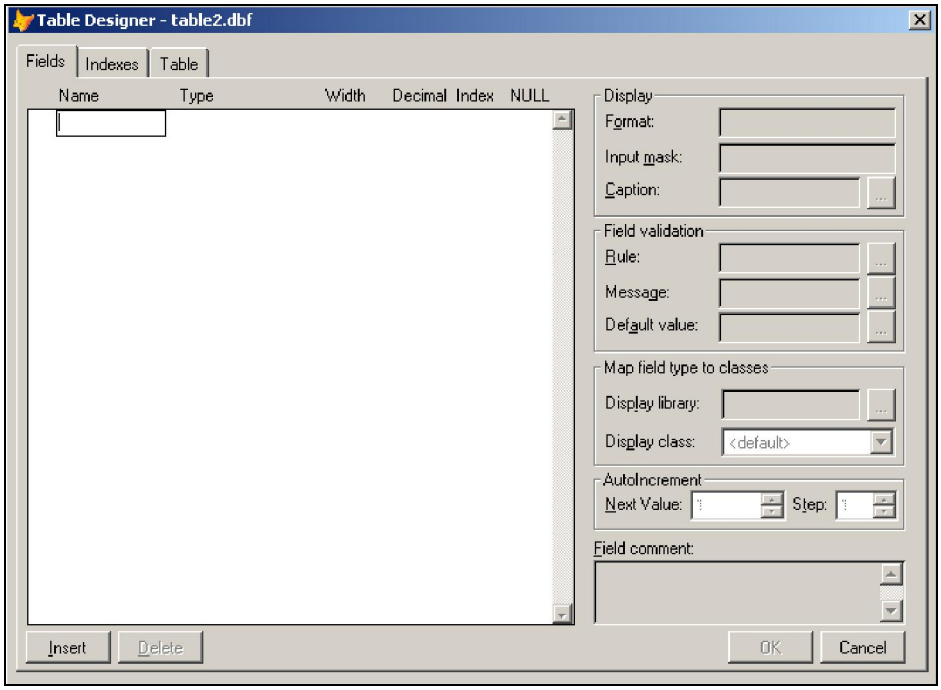

Рис. 6.8. Конструктор таблиц

Таблица 6.17. *Типы данных полей таблиц*

Тип данных	**Описание**	**Размерность**	**Диапазон**
`Blob`	Двоичный литерал неопределенной длины. Начинается обязательно с *`0h`*, например *`0hABCD`*. Хранится в memo-поле (FPT-файл). Не имеет установленной кодовой страницы	Размер 4 байта в таблице	Размер ограничивается либо размером памяти, либо действует ограничение на размер таблицы в 2 Гбайт
`Character`	Символьный тип данных	Длина от одного до 254 символов	Любые символы
`Character (Binary)`	Символьный тип данных, для которых вы не хотите изменять кодовую страницу	Длина — от 1 до 254 символов	Любые символы
`Currency`	Денежный тип, для него перед числом устанавливается знак $	Представляет собой 8-байтное значение валюты	Максимальный размер составляет от –$922337203685477.5807 до $922337203685477.5807

***Таблица 6.17** (продолжение)*

Тип данных	Описание	Размерность	Диапазон
`Data`	Хронологический тип данных в виде даты, содержит дни, месяцы, годы	Размерность 8 байт	Может содержать даты от {^0001-01-01} до {^9999-12-31}
`DataTime`	Данные хронологического типа, содержит дату (дни, месяцы, годы) и время (часы, минуты, секунды)	Размерность 8 байт	Диапазон значений может быть от {^0001-01-01} до {^9999-12-31} для даты и от 00:00:00 до 11:59:59 для времени (для сокращенного формата)
`Double`	Числовые данные с плавающей точкой двойной точности. Применяется для значений, содержащих большое количество значащих цифр, хотя точность расчета от этого не меняется	Длина 8 байт	Диапазон значений может включать от +/-4.94065645841247E-324 до +/-8.9884656743115E307
`Float`	То же самое, что и `Numeric`, т. е. числовой тип данных	Длина от 8 байт в памяти и от 1 до 20 байт в таблице	Диапазон значений от - .9999999999E+19 до .9999999999E+20
`General`	Это содержимое файла и ссылка на приложение, при помощи которого этот файл надо просматривать, хотя может быть еще ссылка на файл-источник. Ссылка на OLE-объект, например, страницу в Microsoft Excel	Длина 4 байта в таблице	Размер ограничивается либо размером памяти, либо действует ограничение на размер таблицы в 2 Гбайт
`Integer`	Целое число без дробной части	Занимает 4 байта	Диапазон разрешенных значений от -2147483647 до 2147483647
`Integer (Autoinc)`	То же самое, что и `Integer`, только с автоматически добавляемой единицей к каждому следующему значению. Тип только для чтения	Занимает 4 байта	Значение контролируется автоматически
`Logical`	Логический тип данных	1 байт	Может принимать только два значения — `.T.` (True) и `.F.` (False) – "Истина" и "Ложь"
`Memo`	Символьный текст неопределенной длины или ссылка на блок данных	Занимает 4 байта	Размер ограничивается либо размером памяти, либо действует ограничение на размер таблицы в 2 Гбайт

Memo (Binary)	То же самое, что и Memo, но с неизменяемой кодовой страницей	Занимает в таблице 4 байта	Размер ограничивается либо размером памяти, либо действует ограничение на размер таблицы в 2 Гбайт

Таблица 6.17 *(окончание)*

Тип данных	Описание	Размерность	Диапазон
Numeric	Числовой (целый или дробный) тип данных	8 байтов в памяти; от 1 до 20 байтов в таблице	От .9999999999E+19 до .9999999999E+20
Varbinary	Двоичный тип данных		
Varchar	Это то же, что и Character, но отображает только реально введенные символы, без автоматического добавления концевых пробелов в момент записи. То есть VarChar(10) будет занимать в таблице 10 байт (точнее 10 байт и 1 бит), даже если в него не введено ни одного символа	Занимает в таблице 1 байт на каждый символ	От 1 до 254 символов
Varchar (Binary)	То же самое, что и Varchar, но для случая, когда вы не хотите менять кодовую страницу	Занимает 1 байт на каждый символ	От 1 до 254 символов

Правила именования таблиц и их полей:

- длина имени свободной таблицы — до 128 байт, поля свободной таблицы — до 10 байт;
- длина имени таблицы базы данных — до 128 байт, поля таблицы в БД — также до 128 байт;
- имена таблиц в базе могут содержать пробелы. В этом случае они записываются в кавычках, однако использовать пробелы в именах не рекомендуется.

Число записей в файле до 1 миллиарда, длина записи — до 65 500 байт, число полей в записи таблицы — до 255, количество одновременно открытых таблиц — до 65 535.

Индексация

Ваши пользователи будут вносить записи в созданные вами таблицы. Трудно представить, что они будут вносить данные в каком-то порядке, скорее всего, это будет происходить по мере необходимости, и записи в таблице будут не упорядочены. Можно отсортировать их командой SORT, но она создает копию указанной таблицы, и уже в ней производит сортировку. Если таблица большая, происходит все это мед-

ленно, да и места на диске занимает много. Для поиска в неупорядоченной таблице можно использовать команду LOCATE, имеющую следующий синтаксис:

```
LOCATE  [FOR lExpression1]  [Scope]  [WHILE lExpression2]  [NOOPTIMIZE]
```

После выполнения команды LOCATE указатель будет установлен на первую запись, для которой выполняется условие Expression. Перемещаться по неупорядоченной таблице и получать нужные записи можно с помощью оператора CONTINUE. Если записей с указанным условием больше нет, указатель будет перемещен в конец таблицы. Проверить, действительно ли найдена запись, можно с помощью функции FOUND(). Если запись обнаружена, функция FOUND() возвратит .T.

Но в большинстве случаев оказывается проще проиндексировать таблицу и вести поиск, подключив индексы. Что же это такое — индексы?

Индексы — это двоичные файлы, в которых номера записей определяются значением индекса. Сначала выполняется поиск в индексном файле, затем по найденной ссылке происходит прямой переход к нужной записи в самой таблице.

Именно это делает поиск скоростным. Принцип технологии оптимизации запросов Rushmore, применяемый в современных системах управления базами данных, позволяет увеличить быстродействие в 100—1000 раз за счет использования оперативной памяти при работе с индексами. С давних времен, еще в первых версиях FoxPro, индексы могли быть только автономными. Они имели расширение IDX и индекс строился по одному полю таблицы. Теперь чаще всего используются так называемые мультииндексные файлы, которые содержат несколько индексных определений.

Создание индексного файла производится командой:

```
INDEX ON eExpression TO IDXFileName | TAG TagName [BINARY]
   [COLLATE cCollateSequence] [OF CDXFileName] [FOR lExpression] [COMPACT]
[ASCENDING | DESCENDING] [UNIQUE | CANDIDATE] [ADDITIVE]
```

Индекс можно создавать по полю или по некоторой комбинации полей. Если, например, мы хотим упорядочить таблицу SOTR по номеру отдела и по фамилиям сотрудников внутри отдела, то мы должны создать такой индекс:

```
INDEX ON otdel+fio to OTDF
```

При использовании предложения ASCENDING записи упорядочиваются по возрастанию, а в случае использования DESCENDING — по убыванию. По умолчанию используется ASCENDING. Если использовать фразу FOR — индексирование будет выполнено по условию. Например, можно включить в индекс не все отделы, а только те, номер которых больше 5. В этом случае команда INDEX будет выглядеть так:

```
INDEX ON otdel+fio to OTDF FOR val(otdel)>5
```

Такие индексные файлы можно объединить в мультииндексный файл, который может содержать несколько индексов. Мультииндексный файл имеет расширение cdx. Мультииндексный файл всегда является компактным, потому что в память может считываться более одного индекса, что обеспечивает наибольшую эффективность

технологии Rushmore. Мультииндексный файл содержит количество тегов (имен индексов), равное количеству созданных индексов, и хранит их в одном едином файле с расширением cdx. В таком файле индексы изменяются одновременно при любых изменениях в таблице, поиск по ним осуществляется наиболее оптимальным методом.

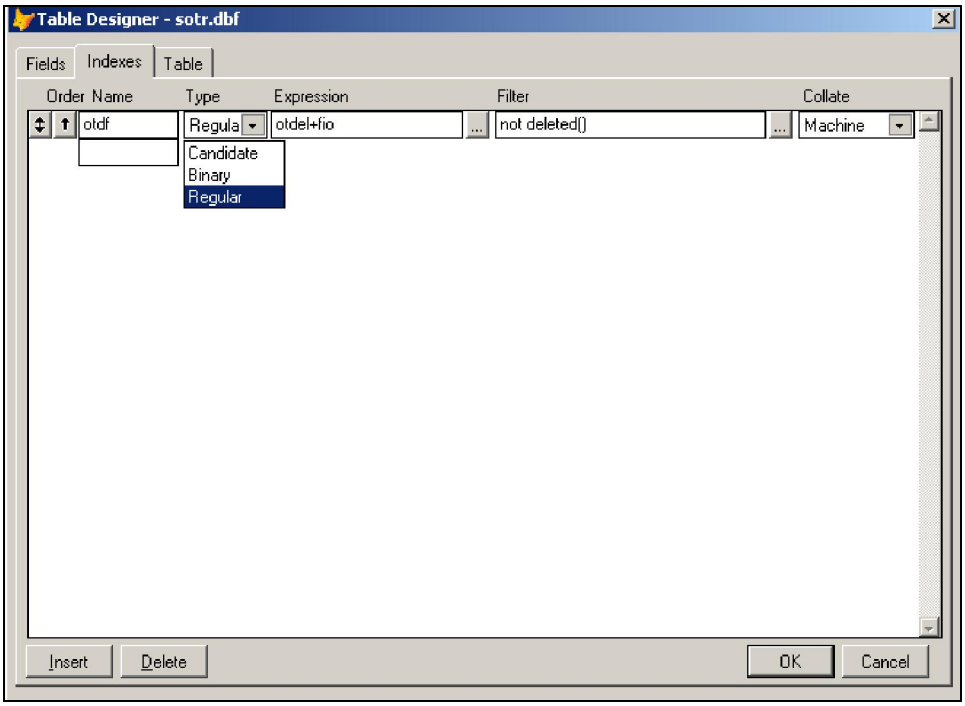

Рис. 6.9. Вкладка **Indexes** Конструктора таблиц

Индексы создаются на вкладке **Indexes** Конструктора таблиц (рис. 6.9).

Order, т. е. порядок индекса (Ascending или Descending), может быть изменен при активации кнопки с двунаправленной стрелкой, либо левой кнопкой мыши, либо клавишей <Spacebar>. Если стрелка направлена вверх, это возрастающий порядок индекса, если вниз — убывающий.

Name представляет собой имя индекса, или тега. Имя не может быть длиннее 10 символов и не может включать спецсимволы и пробелы.

Expression представляет собой имя поля или сложное выражение. Для формирования выражения можно воспользоваться кнопкой, находящейся справа от окна **Expression**.

Filter позволяет включать в индексное выражение некоторое логическое условие, при этом в индекс включаются не все записи, а только удовлетворяющие указанному условию.

Регулярные и уникальные индексы

Индексы не обязательно должны быть уникальными, ведь цель их построения — наиболее быстрый доступ к записи. Список **Type** содержит все типы индексов, которые могут быть построены (табл. 6.18).

Таблица 6.18. *Типы индексов*

Типы индексов	Описание
`Primary` (первичный, уникальный)	Уникальный ключ, используется для организации отношений между таблицами и определения условий целостности данных. Не может иметь пустых и повторяющихся значений. Первичный ключ в таблице может быть только один
`Candidate` (потенциальный)	Это тоже уникальный ключ, который не может иметь пустых и повторяющихся значений. Фактически это тоже `Primary`, а называется по-другому только потому, что таблица не может иметь несколько `Primary`
`Regular` (стандартный)	Обычный индекс, который может иметь повторяющиеся значения, содержит сам индекс и ссылку на запись
`Binary` (двоичный)	Это двоичный ключ, создающийся на основе логических выражений, например, при индексировании удаленных записей. Занимает мало места, не имеет нулевых значений

Индексация по сложным выражениям и использование хранимых процедур

В виде индекса может выступать не только поле, но и любая комбинация полей, а также, например, часть одного поля и часть другого. Для построения сложного выражения вызовите **Expression Builder**, нажав кнопку справа от окна **Expression** (рис. 6.10).

Разумеется, никто не может вам запретить заполнение поля **Expression** вручную, но можно воспользоваться и диалоговым окном. Диалоговое окно содержит множество функций для построения сложных выражений: символьные, математические, логические, функции для работы с датами. Сначала нужно выбрать из раскрывающегося списка нужную функцию, а потом имя поля. В этом случае сразу произойдет замена вставки функции на нужное наименование поля. Поле можно выбрать не только из текущей таблицы. Нужную таблицу можно выбрать в окне **From table**. Правда, создавать индексы из нескольких таблиц — несколько рискованно. Иногда при индексировании таблицы применяют не стандартные, а разработанные самостоятельно функции. Ничего особенного в этом нет, пишут UDF-функцию (т. е. функцию, определенную пользователем), а в окне **Expression** пишут строку (UDF(поле)). Единственный недостаток этого способа — почему-то при индексации таблицы пользовательская функция всегда оказывается "не там", что вызывает ошибку. Теперь для со-

хранения пользовательских функций у нас имеется другой вариант: хранимые процедуры баз данных. Применительно к Visual FoxPro смысл хранимых процедур в том, что это некие процедуры, которые связаны собственно с данными, а не с приложением,
их обрабатывающим. Это значит, что при доступе к тем же данным из другого приложения целостность данных нарушена не будет. За этим проследят хранимые процедуры.

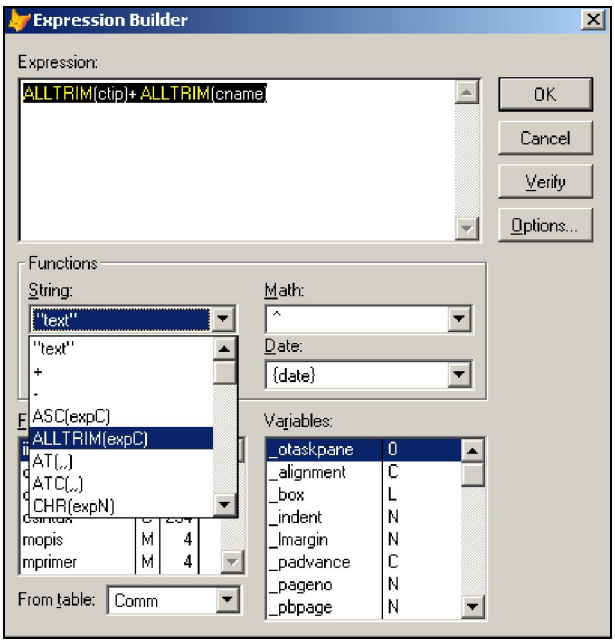

Рис. 6.10. Построение сложных индексных выражений с помощью **Expression Builder**

Поиск в индексированной таблице

После создания индексов поиск в таблице происходит очень быстро, значительно быстрее, чем в неупорядоченной таблице, если, конечно, созданный индекс подключен.

Включить нужный индекс можно командой

```
SET INDEX TO <имя индексного файла>
```

или

```
SET ORDER TO <имя тега>.
```

При поиске имеет значение установка SET EXACT ON. Если SET EXACT установлен в значение ON, то поиск подходящей записи происходит по всей искомой строке, если в OFF, то поиск производится по начальным символам строки.

Кроме того, важна установка SET NEAR. Если перед поисковой командой установлено SET NEAR ON и точно такая запись не найдена, то указатель устанавливается на первую подходящую запись, если же SET NEAR OFF, то в конец файла Для самого поиска используется команда SEEK:

```
SEEK  eExpression   [ORDER nIndexNumber | IDXIndexFileName
    | [TAG] TagName [OF CDXFileName] [ASCENDING | DESCENDING]] [IN nWorkArea |
cTableAlias]
```

Кроме самой команды SEEK, можно использовать и функцию SEEK().

```
 SEEK (eExpression [, nWorkArea | cTableAlias
   [, nIndexNumber | cIDXIndexFileName | cTagName]])
```

Особенно удобна функция SEEK в условных операторах, например:

```
IIF(SEEK(m.fio,"SOTR"), sotr.otdel,"")
```

Если фамилия найдена в таблице SOTR, то в данном поле вывести номер отдела, в противном случае ничего не показывать. Особенно удобно это использовать в отчетах и экранных формах, но об этом чуть позже.

Выбор активного индекса в процессе выполнения

Бывает так, что в процессе выполнения программы индекс нужно изменить. Например, нужно организовать поиск как по наименованию книги, так и по издательству, автору, дате издания. В этом случае таблица открывается с тем индексом, поиск по которому планируется первым.

```
USE knigi TAG naim
```

Изменяется поиск (например, сейчас будем искать по издательству), значит, изменяется и индекс:

```
SELECT knigi
SET ORDER TO izd
```

Индекс можно и вовсе отключить, если дать команду SET ORDER TO без параметров.

Замечания

- Учтите, что максимальная длина индекса, создаваемого по длинному полю, составляет 120 символов.
- Постарайтесь не использовать команду PACK в сетевом режиме. При использовании этой команды *в сетевом режиме* программа автоматически перестает быть сетевой.
- Введите в таблицу и используйте специальные поля: "дата ввода" и "дата удаления". Не показывайте на экране записи с непустой датой удаления.

- Что делать, если повреждены индексы? Интересный вопрос... переиндексировать, конечно. Только как это лучше сделать? Удалять и строить заново? Что ж, это тоже выход. А можно просто взять резервную копию, скопировать "здоровые" CDX-файлы на место поврежденных, а потом открыть таблицы эксклюзивно и переиндексировать, запустив команду `REINDEX`.

Отношения (связи) между таблицами

Вы можете установить отношения между таблицами (рис. 6.11), которые будут использованы в ваших проектах. Для этого одна из таблиц должна являться родительской, а другая — дочерней. Возможны отношения — одна родительская таблица и несколько дочерних.

Для родительской таблицы нужно определить первичный ключ или ключ-кандидат, для дочерних — индекс для связи с родительской таблицей.

Порядок ваших действий при создании отношения:

1. Открыть **Data Environment**.
2. Установить курсор на первичный ключ родительской таблицы.
3. Не отпуская кнопки мыши, переместите курсор на индекс дочерней таблицы, по которому устанавливается связь.
4. Теперь можно отпустить кнопку мыши.
5. Между таблицами появляется линия, которая отображает созданное соединение.

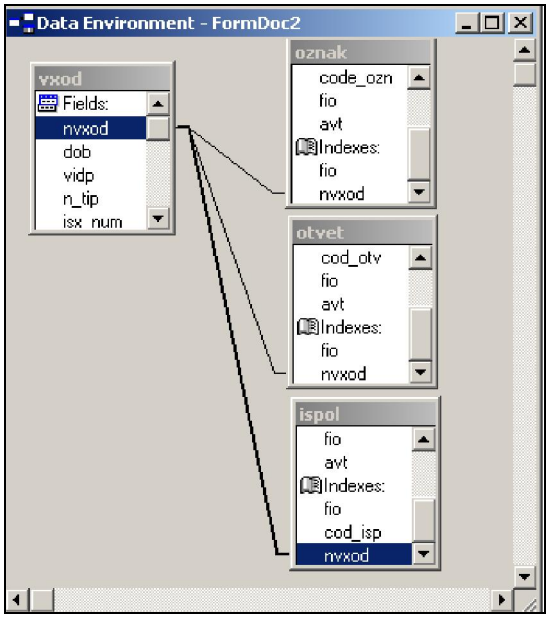

Рис. 6.11. Отношения между свободными таблицами

Те же самые отношения могут быть созданы программным способом, например:

```
USE vxod order ID shared in 0
USE ispol ORDER ID shared in 0
USE otvet ORDER ID shared in 0
USE oznak ORDER ID shared in 0
Sele vxod
SET RELATION TO vxod.ID INTO ispol
SET RELATION TO vxod.ID INTO otvet
SET RELATION TO vxod.ID INTO oznak
```

Создание таблицы программным способом

Если необходимо создать таблицу программно, то можно использовать ряд команд Visual FoxPro.

Команда

```
CREATE [FileName | ?]
```

открывает Конструктор таблицы для ее создания.

Параметр `FileName` определяет имя или полное имя создаваемой таблицы. Если таблица создается в момент, когда существует текущая открытая база данных, тогда таблица будет добавлена в эту базу данных.

Команда

```
CREATE TABLE | DBF TableName1 [NAME LongTableName] [FREE]      [CODEPAGE =
nCodePage]
    ( FieldName1 FieldType [( nFieldWidth [, nPrecision] )] [NULL | NOT NULL]
    [CHECK lExpression1 [ERROR cMessageText1]]
    [AUTOINC [NEXTVALUE NextValue [STEP StepValue]]] [DEFAULT eExpression1]
    [PRIMARY KEY | UNIQUE [COLLATE cCollateSequence]]
    [REFERENCES TableName2 [TAG TagName1]] [NOCPTRANS]
    [, FieldName2 ... ]
    [, PRIMARY KEY eExpression2 TAG TagName2 |, UNIQUE eExpression3 TAG TagName3
    [COLLATE cCollateSequence]]
    [, FOREIGN KEY eExpression4 TAG TagName4 [NODUP]
    [COLLATE cCollateSequence]
    REFERENCES TableName3 [TAG TagName5]] [, CHECK lExpression2 [ERROR
cMessageText2]] )
    | FROM ARRAY ArrayName
```

Команда

```
COPY STRUCTURE TO TableName [Fields FieldList]
  [[WITH] CDX | [WITH] PRODUCTION]
  [DATABASE DatabaseName [NAME LongTableName]]
```

Позволяет создать новую пустую таблицу с той же структурой, что и текущая таблица.

Команда

```
COPY STRUCTURE EXTENDED TO TableName [Fields FieldList]
[DATABASE DatabaseName [NAME LongTableName]]
```

Создает новую таблицу `TableName` с записями, содержащими структуру текущей таблицы.

В записи новой таблицы содержатся 18 полей, которые приведены в табл. 6.19.

Данные таблицы TableName могут быть использованы командой `CREATE FROM` для создания новой таблицы. Таблица TableName может быть отредактирована как вручную, так и программно.

Таблица 6.19. *Поля таблицы, созданной командой* `Copy Structure Extended`

Наименование поля	Тип	Описание
FIELD_NAME	Character	Имя поля таблицы-оригинала

FIELD_TYPE	Character	Символ, характеризующий тип поля: W — Blob C — Character Y — Currency N — Numeric F — Float I — Integer B — Double D — Date T — DateTime L — Logical M — Memo G — General Q — Varbinary V — Varchar и Varchar (Binary)
FIELD_LEN	Numeric	Длина поля
FIELD_DEC	Numeric	Число знаков после запятой (для числовых полей)
FIELD_NULL	Logical	Поддержка NULL
FIELD_NOCP	Logical	Флаг запрета изменения кодовой страницы поля
FIELD_DEFA	Memo	Значение поля по умолчанию
FIELD_RULE	Memo	Правило проверки поля
FIELD_ERR	Memo	Error-текст правила проверки поля
TABLE_RULE	Memo	Правило проверки таблицы (только первая запись)
TABLE_ERR	Memo	Error-текст правила проверки таблицы (только первая запись)
TABLE_NAME	Character	Длинное имя таблицы (только первая запись)
INS_TRIG	Memo	Выражение для триггера вставки (только первая запись)
UPD_TRIG	Memo	Выражение для триггера изменения (только первая запись)
DEL_TRIG	Memo	Выражение для триггера удаления (только первая запись)
TABLE_CMT	Memo	Комментарий таблицы (только первая запись)
FIELD_NEXT	Numeric	Следующее значение для поля с AutoInc
FIELD_STEP	Numeric	Шаг для AutoInc-поля. Нуль означает, что поле не AutoInc

Команда

```
CREATE [FileName1 [ Database DatabaseName {NAME LongTableName}]] FROM
[FileName2]
```

создает таблицу по данным таблицы, построенной командой `COPY STRUCTURE EXTENDED`. Созданная таблица открывается и становится текущей.

При создании таблицы необходимо определить, где именно она создается. Не исключена ситуация, что каталогом по умолчанию является CD-диск, на котором вам не удастся создать таблицу. В этом случае появится сообщение об ошибке.

Создание и модификация баз данных

Аналогично созданию таблиц, существует несколько способов создать базу данных: с помощью мастера, с помощью конструктора, с помощью диспетчера проектов, программным способом. Для создания баз данных с помощью мастера нужно обратиться к меню Visual FoxPro и вызвать мастер баз данных: **Tools | Wizards | Database**. В этом случае база данных создается на основе уже имеющихся баз данных Visual FoxPro.

Создание базы данных с помощью конструктора

Надо признаться, что программисты гораздо чаще используют конструкторы, чем мастера.

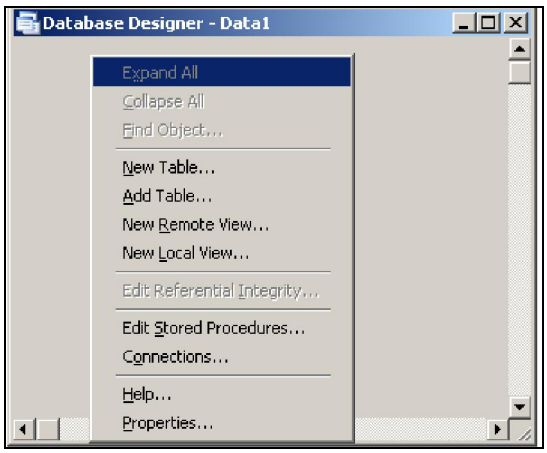

Рис. 6.12. Окно **Database Designer** — конструктор баз данных

Конструктор баз данных (рис. 6.12) можно вызвать несколькими способами:

- в командном окне ввести команду `CREATE DATABASE [DatabaseName | ?]`;
- выбрать в главном меню Visual FoxPro команду **File | New | Database | New File**;
- в командном окне ввести команду `MODIFY DATABASE`.

Затем нужно ввести в появившемся диалоговом окне имя создаваемой базы данных, по умолчанию будет присвоено имя `Data1`, но вы можете присвоить базе данных лю-

бое другое имя. Если путь для создаваемой базы данных не указан, она создается в каталоге по умолчанию. Созданная база данных всегда открывается в режиме единоличного доступа независимо от установки SET EXCLUSIVE. Созданная база данных становится текущей. Команды создания таблиц (CREATE TABLE) без опции FREE будут создавать таблицы в составе текущей базы данных.

При работе с базой данных в главное меню добавляется новый пункт **Database**.

В табл. 6.20 приведен список команд меню **Database**.

Таблица 6.20. *Список команд меню* ***Database***

Пункт меню	Описание
New Table	Создает новую таблицу в базе данных
Add Table	Добавляет таблицу в базу данных
New Remote View	Создает новое удаленное представление
New Local View	Создает новое локальное представление
Modify	Модифицирует структуру текущей таблицы или представления
Browse	Позволяет просмотр и редактирование данных
Remove	Удаляет текущую таблицу или представление
Find Object	Поиск таблицы или представления
Rebuild Table Indexes	Переиндексация текущей таблицы
Remove Deleted Records	Упаковка помеченных на удаление записей
Edit Relationship	Изменение отношений между таблицами
Edit Referential Integrity	Редактирование правил для обеспечения целостности
Edit Stored Procedures	Редактирование хранимых процедур
Connections	Создание и редактирование соединений с удаленными данными
Arrange	Визуальное упорядочивание объектов базы данных
Clean Up Database	Упаковка базы данных
Properties	Работа с диалоговым окном "Свойства базы данных"

Меню можно вызвать, нажав правую кнопку мыши на свободном поле конструктора.

Меню позволяет создать новую таблицу в составе базы данных, включить в состав базы уже имеющуюся таблицу, создать удаленное или локальное представление, хранимую процедуру или соединение. Кроме того, используя пункт меню **Properties**, можно написать процедуры, выполняющиеся при определенных условиях: перед открытием или закрытием таблиц, представлений или соединений.

Кроме уже описанных способов, можно создать базу данных, используя диспетчер проектов**: File | New | Project | New File**. Появляется диалоговое окно, предлагающее ввести имя для создаваемого проекта, по умолчанию это proj1.pjx. В появившемся окне Менеджера проектов мы выберем вкладку **Data**, установим курсор на `Databases` и нажмем его. Появятся кнопки, позволяющие создать в проекте новую базу данных (рис. 6.13).

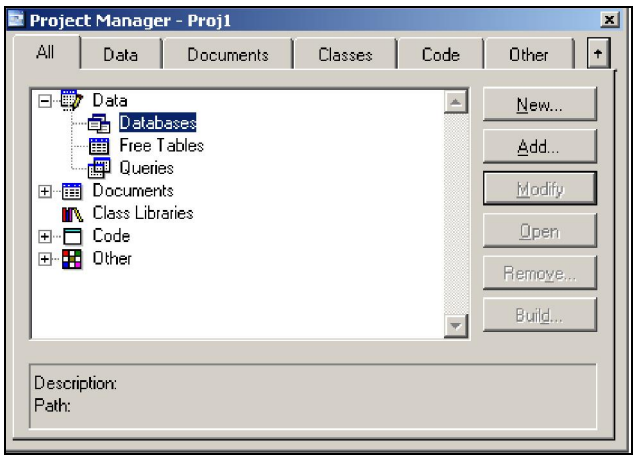

Рис. 6.13. Окно **Project Manager**

Если мы выберем кнопку **New…**, то появятся две возможности создать новую базу: воспользоваться мастером **Database Wizard** — уже имеющимися базами данных, чтобы на их основе создать свою, либо создать базу данных "с нуля" (рис. 6.14).

Рис. 6.14. Выбор варианта создания базы данных

Получим практически то же самое, что и в первом случае, но только вместе с созданным проектом.

Вновь созданная база данных всегда открывается монопольно, независимо от состояния настройки `SET EXCLUSIVE`. Кроме того, созданная вновь база данных становится

текущей, поэтому выполнение команды CREATE TABLE без опции FREE приводит к включению таблицы в состав базы данных.

Еще один способ создания баз данных — программный, для него используются либо командное окно Visual FoxPro, либо программа или функция.

Команда

```
CREATE DATABASE <BaseName> | ?
```

Указывает имя базы данных или открывает диалоговый блок **Create** с тем, чтобы вы могли указать имя для базы данных и место на диске, где вы желаете ее сохранить. Если вы опустите параметры, то откроется диалоговый блок **Create**. Файлы баз данных сохраняются с расширением имени файла dbc.

Диаграмма отношений между таблицами

Любая реляционная база данных потому и называется реляционной, что характеризуется отношениями (relation) между таблицами. При этом одна таблица является родительской (главной), а вторая — дочерней (подчиненной). Чтобы установить отношение, используются индексы, которые синхронизируют перемещение указателя записи в связанных таблицах. Для определения отношений откроем окно конструктора базы данных и произведем следующие действия:

1. Определяем родительскую таблицу.
2. Устанавливаем курсор на первичный ключ родительской таблицы.
3. Переместим курсор на индекс дочерней таблицы.
4. После того, как мы отпустим кнопку мыши, образуется линия (связь), показывающая вновь созданное отношение. На стороне родительской таблицы отношение отображается одной линией, выходящей из таблицы, а на стороне дочерней таблицы — тремя линиями.
5. Мы можем отредактировать отношения, установив курсор мыши на линию, связывающую две таблицы, и дважды щелкнув на ней мышью. Появится окно редактирования отношения (рис. 6.15).

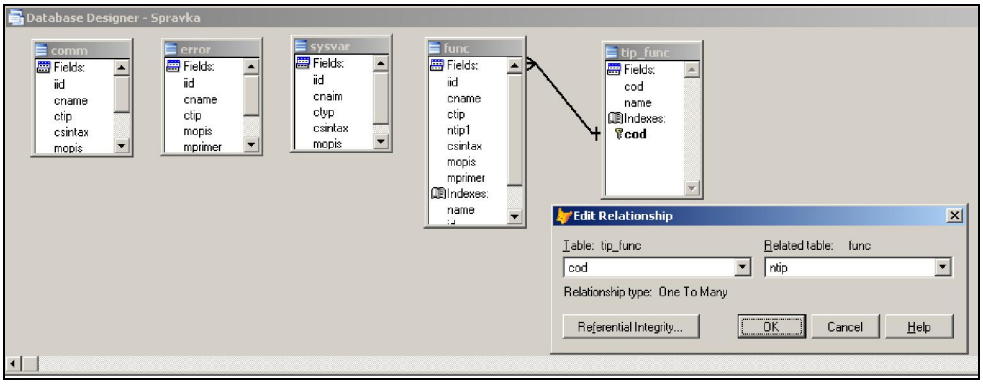

Рис. 6.15. Установка отношения (реляции)

Слева мы видим информацию о родительской таблице, справа — о дочерней. Для того чтобы сохранить отредактированное отношение, нужно нажать на кнопку **OK**, а чтобы отказаться — **Cancel**. Если отношение не нужно, его можно удалить, щелкнув по правой кнопке мыши. Линия связи при этом увеличится в толщине, и появится меню редактирования и удаления отношения, в котором достаточно выбрать пункт "`Remove Relationship`", чтобы избавиться от ненужного отношения.

Второй способ удаления — выбрать отношение и нажать клавишу <Delete> на клавиатуре.

Определение свойств полей

В правой части Конструктора таблиц имеется несколько окон, контролирующих достоверность ввода в каждое из полей (рис. 6.16).

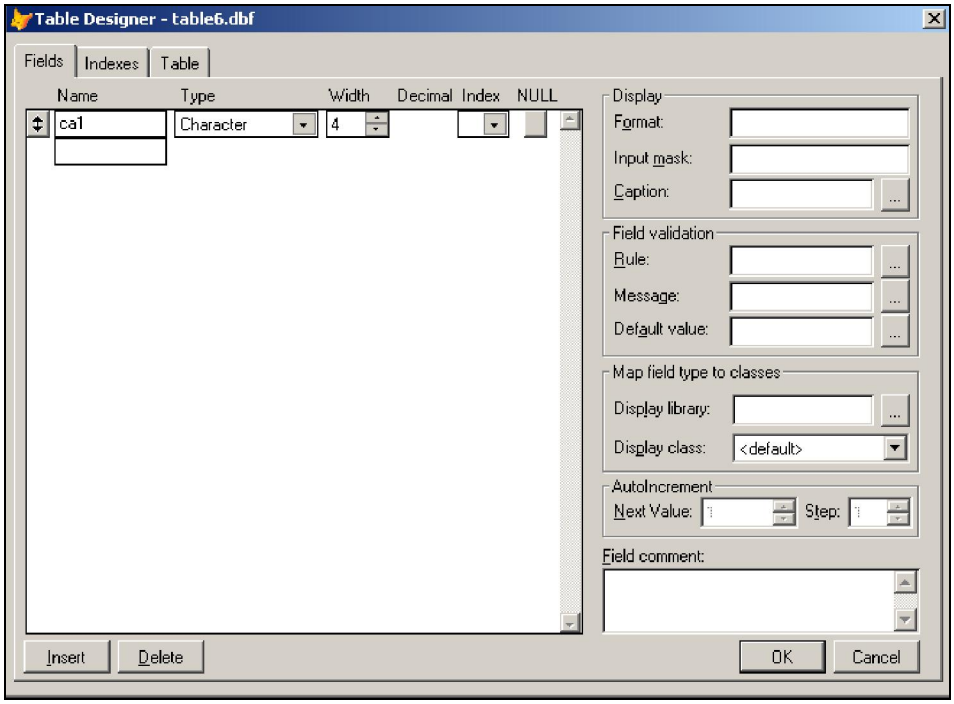

Рис. 6.16. Конструктор таблиц. Определение свойств полей

Группа **Display** содержит текстовое поле **Format**, в котором производится установка формата отображения поля таблицы, поле **Input mask** для ввода маски при вводе данных (например, можно определить маску для ввода страхового номера 999-999-999-99), а также раскрывающийся список **Caption** для смены заголовка поля.

Например, в **Caption** можно ввести русские заголовки полей таблицы, и именно они будут отображаться при выполнении команды **Browse**.

Группа **Field validation** содержит следующие поля:

- **Rule** — ввод логического выражения, используемого для проверки поля;
- **Message** — вывод сообщения о неправильном вводе значения поля;
- **Default value** — значение поля, которое вводится во все новые записи по умолчанию, например, в поле "код страны" можно ввести 643 — код России.

Группа **Map field type to classes** содержит текстовое поле **Display library**, предусмотренное для ввода библиотеки классов, и поле **Display class** — для ввода базового класса.

Группа **Autoincrement** позволяет ввести шаг, на который увеличивается значение поля при вводе каждой новой записи, и начальное значение для автоинкремента.

Определение свойств таблицы

При создании новой таблицы она открывается монопольно (т. е. для одного пользователя), при этом на вкладке **Table** Конструктора таблиц становится доступной для редактирования информация о таблице (рис. 6.17).

В области **Record validation** есть несколько раскрывающихся списков. Первый из них — **Rule** — может содержать логическое выражение для проверки правильности ввода данных на уровне записей.

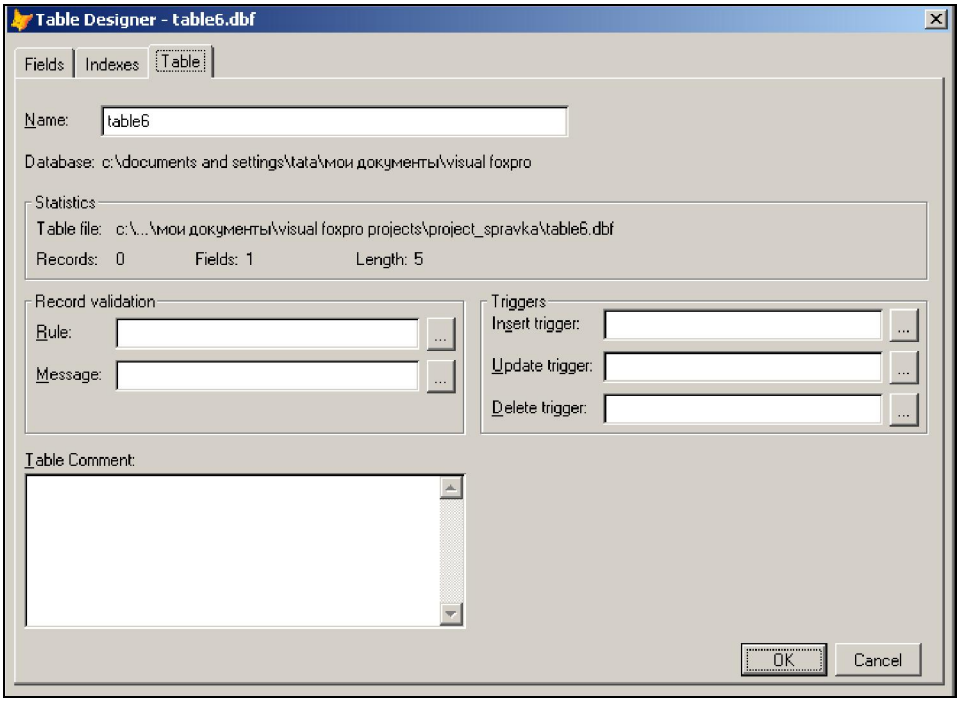

Рис. 6.17. Информация о таблице. Определение свойств

В поле **Message** вводится сообщение, которое будет выводиться при неверном вводе записи.

В группе **Triggers** возможна установка трех логических выражений, которые контролируют ввод записи (**Insert trigger**), редактирование (**Update trigger**) и удаление (**Delete trigger**). При этом логические выражения могут содержать вызов хранимых процедур.

Триггеры

При определении условий достоверности ввода данных используются триггеры и хранимые процедуры. В триггерах определяются действия, которые запускаются при добавлении, редактировании и удалении записей. Код для триггеров необходимо либо определить в соответствующих окнах области **Triggers** конструктора таблиц, либо разработать и хранить в хранимых процедурах базы данных.

Для того чтобы определить триггер, достаточно ввести в поля **Insert trigger**, **Update trigger** и **Delete trigger** логические выражения, принимающие значение `.T.` или `.F.`. При выполнении этого условия запись считается прошедшей контроль, в противном случае данные не записываются, и выдается сообщение об ошибке.

Нужно учитывать, что:

- выполнение команды `ZAP` не запускает триггер удаления;
- триггеры не вызываются при редактировании удаленных записей и выполнении команды `PACK`;
- при использовании буферизации триггер **Update** вызывается при вызове функции `TableUpdate()`.

Хранимые процедуры

Хранимые процедуры являются неотъемлемой частью базы данных. Использование хранимых процедур позволяет часть прикладного обеспечения перенести на сервер (модель распределенного приложения). Процедуры хранятся в словаре базы данных, разделяются между несколькими клиентами и выполняются на том же компьютере, что и SQL-сервер. Преимущества такого подхода: возможно централизованное администрирование прикладных функций, значительно снижается сетевой трафик (т. к. передаются не SQL-запросы, а вызовы хранимых процедур). Недостаток — ограниченность средств разработки хранимых процедур. На практике сейчас обычно используется смешанный подход: простейшие прикладные функции выполняются хранимыми процедурами на сервере, а более сложные реализуются на клиенте непосредственно в прикладной программе.

Хранимые процедуры создаются в проекте приложения. В окне проекта выберите вкладку **Database**, затем **Stored Procedures** и нажмите кнопку **New**. На экране откроется окно редактирования хранимых процедур. При этом будут отображены все хранимые процедуры проекта. В этом окне вы можете ввести текст процедуры: как общей для приложения, так и отдельной, используемой в выражении для триггера. Для удаления ненужной более процедуры используйте кнопку **Remove**.

Дополнительные возможности контроля данных и их целостности

Visual FoxPro позволяет определить условие достоверности ввода данных как на уровне поля таблицы, так и на уровне записи. Контроль на уровне записи производится обычно в случае, если необходимо проконтролировать запись по нескольким полям. Для определения достоверности ввода используются поля ввода **Rule** и **Message** области **Record Validation** (**Table Designer | Table | Rule**). В поле **Rule** вводится логическое выражение, которое определяет правильность ввода поля. Например, это может быть алгоритм контроля правильности ИНН при вводе ИНН, или банальное условие число >0 для числового поля. В поле **Message** вводится сообщение, выводимое при ошибочном вводе, например: "`введен ошибочный ИНН`" или "`Это поле не может быть нулевым`". Если вычисленное логическое выражение, введенное в поле **Rule**, равно `.T.`, то данные считаются правильными.

Добавление таблицы к базе данных

Если таблица создавалась не в составе базы данных, то ее можно добавить в существующую базу данных одним из следующих способов:

- выполнить **File | Open | Database** — и в **Database Designer** правой кнопкой мыши вызвать контекстное меню, в нем выбрать **Add Table**;
- выполнить команду в командном окне **MODIFY DATABASE** и вызвать контекстное меню, либо воспользоваться появившимся в главном окне FoxPro пунктом меню **Database**;
- вызвать диспетчер проектов командой `MODIFY PROJECT`, выбрать пункт **Database**, тогда справа появится кнопка **Add Table**.

Исключение таблицы из базы данных

Удалить из базы данных ненужную таблицу можно следующим образом:

- в **Database Designer**, указав правой кнопкой мыши на таблицу, вызвать контекстное меню, и в нем выбрать **Delete**;
- воспользоваться появившимся в главном окне FoxPro пунктом меню **Database**, указать на ненужную таблицу и выбрать в меню **Database** пункт **Remove**. Будьте внимательны: воспользовавшись этим пунктом, вы можете не только исключить таблицу из базы данных, но и удалить ее с диска;
- вызвать диспетчер проектов командой `MODIFY PROJECT`, выбрать пункт **Database** и вкладку **Tables**, установить курсор на ненужную таблицу и нажать кнопку **Remove**.

Курсоры

Курсор — это временная таблица, являющаяся образом файла DBF. Курсор открывается в одной из рабочих областей, при этом возможны два способа его создания.

Первый способ — это использование команды `CREATE CURSOR`. Созданный таким способом курсор будет редактируемым, т. е. позволяет добавлять, изменять и удалять записи. Это будет именно временная таблица, т. е. она будет автоматически уничтожена в момент закрытия.

Второй способ — это использование опции `CURSOR` в команде `SELECT-SQL: SELECT * FROM MyTable INTO CURSOR TmpTable`.

Вот этот-то `TmpTable` и есть *курсор*.

В зависимости от различных условий этот курсор может иметь разное физическое "воплощение" и разные свойства.

Если SQL-запрос полностью оптимизируем, то вместо создания нового файла будет просто открыта та же самая таблица с наложенным на нее фильтром. Зачастую это очень неприятная неожиданность. Проверить, чем же физически является сформированный курсор, можно, используя функцию `DBF()`:

```
SELECT * FROM MyTable INTO CURSOR TmpTable
?DBF('TmpTable')
```

Если будет возвращено имя файла с расширением dbf, то данный курсор является той же самой исходной таблицей с наложенным на нее фильтром.

Если вы хотите при любых запросах быть уверенными, что курсор — это именно временная таблица, а не исходная таблица с наложенным фильтром, то вам следует добавить опцию `NOFILTER`

```
SELECT * FROM MyTable INTO CURSOR TmpTable NOFILTER
```

Однако если курсор — это временная таблица, то это еще не значит, что эта временная таблица будет непременно физически расположена на диске. Вполне возможно, что вся временная таблица целиком поместится в оперативную память. То есть функция `DBF('TmpTable')` будет исправно показывать некий временный файл, но попытка найти его физически на диске окончится неудачей и функция `FILE(DBF('TmpTable'))` вернет `.F.`.

Правда, расположение временной таблицы целиком в памяти ни в коем случае не помешает работе с этой временной таблицей. Собственно, в подавляющем большинстве случаев вас и не должно заботить, где физически расположена та или иная таблица.

Полученные таким образом курсоры можно редактировать с помощью специальной опции `ReadWrite`

```
SELECT * FROM MyTable INTO CURSOR TmpTable NOFILTER READWRITE
```

Использование этой опции позволяет создавать курсор, который можно редактировать. Для более ранних версий VFP для того, чтобы курсор можно было редактировать, его следует открыть снова:

```
SELECT * FROM MyTable INTO CURSOR TmpReadTable NOFILTER
USE (DBF('TmpReadTable')) IN 0 AGAIN ALIAS TmpWriteTable
USE IN TmpReadTable
```

Переоткрытый таким образом курсор `TmpWriteTable` уже можно будет редактировать.

Курсоры можно индексировать так же, как и обычные таблицы. Правда, если курсор открыт в режиме только для чтения, то вы сможете создать для него только один индексный тег.

Все созданные таким образом курсоры автоматически удаляются с диска (если временный файл физически был создан на диске) в момент их закрытия. Если вы создали для такого курсора структурный индексный файл, то этот файл также будет автоматически удален в момент закрытия курсора.

Курсор всегда создается на машине клиента, поэтому конфликтов с другими пользователями можно не опасаться. Более того, даже если запущен дважды один и тот же проект на одной машине, все равно не будет конфликта, связанного с одинаковыми именами курсоров, поскольку они открыты в разных *сеансах данных*. Конфликт возможен, если вы создаете несколько курсоров с одним и тем же именем в одном *сеансе данных*.

Например, вы открыли 2 формы, использующие `Default DataSession`, и в обоих создали курсор с одинаковым именем. В этом случае курсор, созданный позднее, затрет курсор, созданный ранее. При этом настройка `SET SAFETY` не играет никакой роли. Курсор будет пересоздан молча, без каких-либо дополнительных запросов.

Избежать подобных конфликтов можно несколькими способами:

- открывать формы и отчеты только в `Private DataSession`;
- самостоятельно следить за уникальностью имен курсоров;
- использовать функцию для генерации уникальных имен файлов.

Последний вариант кажется наиболее предпочтительным. Однако тут следует быть осторожным. Дело в том, что в описании к Visual FoxPro для генерации уникальных имен файлов предлагается использовать следующую функцию:

```
lcCursorName=SubStr(SYS(2015),3,10)
```

Проблема в том, что функция `SYS(2015)` может содержать в возвращаемом значении как буквы, так и цифры. Это значит, что при использовании выделения строки по `SubStr()` вы вполне можете получить первым символом цифру. А использование в качестве имени переменной цифры в синтаксисе Visual FoxPro недопустимо, и вы неожиданно получите сообщение о синтаксической ошибке. Чтобы этого избежать, следует либо принудительно подмешать букву

```
lcCursorName='t'+SubStr(SYS(2015),3,10)
```

либо вообще не выделять строку

```
lcCursorName=SYS(2015)
```

Соответственно, выполнение запроса станет выглядеть так:

```
LOCAL lcCursorName
lcCursorName=SYS(2015)
SELECT * FROM MyTable INTO CURSOR &lcCursorName NOFILTER
```

Такой способ создания уникальных имен курсоров действительно обеспечит уникальность, но это очень неудобный способ из-за необходимости при обращении к такому курсору постоянного использовать макроподстановки. Поэтому по возможности желательно его избегать.

Локальные и удаленные представления (назначение)

Представления — это обычные SQL-выражения, хранящиеся в базе данных. Они могут быть локальными и удаленными. Локальные представления работают только с таблицами самого Visual FoxPro, тогда как удаленные — с любым ODBC-совместимым источником данных.

Для чего нужны представления и в чем их преимущество?

Представление можно открыть в некоторой рабочей области и использовать его в качестве источника данных для объектов управления на форме или выполнять операции поиска и изменения данных с последующим обновлением таблиц, на которых основано представление. Кроме этого, для представлений и их полей имеется ряд настроек, которые позволяют повысить надежность разрабатываемого приложения, обеспечить корректность хранимых данных и увеличить скорость и удобство разработки приложения. К числу преимуществ представлений можно отнести следующие:

- универсальность. Один и тот же код позволяет работать как с таблицами Visual FoxPro, так и с таблицами сервера баз данных. Все что необходимо — это переключиться между локальными и удаленными представлениями;
- производительность. Во многих случаях представления работают быстрее по сравнению со сквозными запросами, т. к. низкоуровневая функция `TABLEUPDATE` автоматически создает вам SQL-предложение. Скорее всего, программное создание SQL-команды и ее исполнение будет медленнее. Если представить весь тот объем работы, который выполняет функция `TABLEUPDATE`, — сканирование буфера изменений, определение типа внесенных модификаций, чтение свойств представления, построение пакета SQL-команд, передача пакета ODBC, получение сообщения от сервера, очистка буфера изменений — мы уверены, вы согласитесь, что функция будет работать быстрее, чем сквозной запрос;
- все вышеперечисленные действия выполняются средствами Visual FoxPro и не имеют синтаксических ошибок;

- функциональность. Представления имеют более широкую функциональность, чем сквозные запросы. Так, для организации пакета из пяти модификаций вам придется склеить в одну строку пять команд, разделенных точкой с запятой, тогда как `TABLEUPDATE()` сделает это автоматически.