q_region.prg

```
SELECT cu.region, COUNT(*) AS n ;
   FROM orders o ;
   JOIN customers cu ;
     ON cu.cust_id = o.cust_id ;
  WHERE YEAR(o.order_date) = 2025 ;
  GROUP BY 1 INTO CURSOR curRegion
```

curRegion

ISLANDS	596
NORTH	918
SOUTH	1115
WEST	701

_TALLY = 4

```
=SYS(3054, 1)
Using index tag Order_date to rushmore optimize table orders
Using index tag Isdel to rushmore optimize table orders
Rushmore optimization level for table orders: full
```

orders.cdx

```
  ├ order_id    PRIMARY
  ├ cust_id     REGULAR
  └ isdel       BINARY
```

```
BEGIN TRANSACTION
  TABLEUPDATE(1, .F., "orders")
  TABLEUPDATE(1, .F., "stockmoves")
END TRANSACTION
```

VFP 9 · THE DATA ENGINE IN DEPTH

Visual FoxPro SQL and Data Mastery

Queries, Cursors, Indexes, Rushmore, and Data Integrity in VFP 9

From CDX tag design and ENGINEBEHAVIOR to transactions, repair, import/export, and a reporting summary layer

SELECT-SQL · CDX · DBC · BUFFERING · XML

THENEXBOOKS

Visual FoxPro SQL and Data Mastery

Queries, Cursors, Indexes, Rushmore, and Data Integrity in VFP 9

A deep dive into the data engine — every example run and verified against one realistic database

First Edition

THENEXBOOKS

Visual FoxPro SQL and Data Mastery
Queries, Cursors, Indexes, Rushmore, and Data Integrity in VFP 9

First Edition, 2026

Copyright © 2026 thenexbooks (thenexbooks.gumroad.com). All rights reserved.

No part of this publication may be reproduced, distributed, or transmitted in any form or by any means without the prior written permission of the publisher, except for brief quotations in reviews and certain other non-commercial uses permitted by copyright law. Code examples and SQL in this book may be used freely in your own projects.

The information in this book is provided "as is" without warranty of any kind. While every precaution has been taken in its preparation, the publisher assumes no responsibility for errors or omissions, or for damages resulting from the use of the information contained herein. Several examples change, pack, re-index, or deliberately damage data: run them only on copies, and always back up real data before you change or repair it.

Microsoft, Visual FoxPro, Visual Studio, SQL Server, Excel, and Windows are trademarks of the Microsoft group of companies. Other product names are trademarks of their respective owners. Their use in this book is for identification only and does not imply endorsement. This book is independent and is not affiliated with, sponsored by, or endorsed by Microsoft Corporation. Visual FoxPro is no longer sold or developed by Microsoft; this book does not include the software, and you need your own licensed copy to run the examples.

Marrowgate Marine Supply, the marine-chandlery distributor used as the running example in this book, is fictional, as are its staff, customers, suppliers, products, orders, towns, and all sample data. Email addresses use reserved example domains.

Preface

Visual FoxPro was always, first and foremost, a data tool. Long before it had classes and forms, FoxPro could open a table with millions of rows, find a record in an instant through an index, and answer a question with a single command. Visual FoxPro 9 added a SQL engine that grew more capable with every release: outer joins, subqueries in almost every position, derived tables, correlated updates, and a cost-free optimizer called Rushmore that turns ordinary index tags into fast filters. Behind thousands of business applications that still run today, it is this data engine — not the screens — that does the real work.

It is also where most of the trouble lives. A query that silently returns one row per group under one setting and fails under another. A report that is fast on the developer's laptop and slow on the network. A total that is off by a cent. A table that no longer opens after a power cut. A "simple" import that turns a header row into a customer. These are data problems, and they reward a deep, precise understanding of how VFP stores, indexes, queries, changes, and protects data.

This book is that deep dive. It is not a general Visual FoxPro tutorial — forms, classes, menus, and report layouts appear only in passing. Instead, we spend twenty-eight chapters on tables and types, CDX indexes and Rushmore, SELECT-SQL in depth, data-changing SQL, cursors, the database container, buffering and transactions, integrity checks and repair, performance and size limits, dates and numbers, data cleaning, reporting queries, and import and export. Every example runs against one realistic database, and every result printed in this book was produced by running the code in Visual FoxPro 9.

Who This Book Is For

- **Visual FoxPro developers** who know the basics — the Command window, `USE`, `BROWSE`, a simple program — and want to master the data engine their applications depend on.
- **SQL developers** coming from SQL Server, PostgreSQL, MySQL, or Access who must query, fix, report on, or migrate DBF data and want to know exactly where VFP's SQL differs from what they know.
- **Maintainers of long-lived VFP systems** who need reliable techniques for integrity checks, repairs, performance tuning, and data exchange.
- **Analysts and report writers** who pull numbers out of VFP data and need crosstabs, running totals, period comparisons, and clean exports they can trust.

What You Will Learn

- How VFP stores data: table structures, every data type, Currency for money, NULLs, defaults, code pages, and collation.
- How to design CDX index tags that make lookups and queries fast — and how to prove it with Rushmore's `SYS(3054)` reports.

- SELECT-SQL in depth: joins of every kind, grouping and HAVING, subqueries, derived tables, UNION, TOP N, and the real, verified differences between `SET ENGINEBEHAVIOR` 70, 80, and 90.
- How to change data safely: INSERT, UPDATE, and DELETE-SQL including VFP 9's correlated `UPDATE ... FROM`, Xbase and SQL side by side, buffering, update conflicts, and transactions.
- How to use the database container well: long names, rules, defaults, triggers, referential integrity, stored procedures, and local views.
- How to keep data healthy: integrity checks, corruption prevention, `VALIDATE DATABASE`, `PACK` and `REINDEX` done safely, data dictionary queries, and the 2 GB limit.
- How to work with real-world data: date and number pitfalls, cleaning and de-duplication, reporting queries, CSV, Excel, fixed-width and XML import and export — and finally a data warehouse–style summary layer.

How This Book Is Organized

Part I — Tables, Types, and Indexes (Chapters 1–5) introduces the data engine, table structures and data types, work areas and Xbase commands, CDX index design, and NULLs, code pages, and collation. **Part II — SELECT-SQL in Depth** (Chapters 6–10) covers queries from the fundamentals through joins, grouping, ENGINEBEHAVIOR, subqueries, UNION, TOP N, and cursors. **Part III — Changing Data Safely** (Chapters 11–14) covers data-changing SQL, Xbase and SQL equivalence, buffering, and transactions. **Part IV — The Database Container** (Chapters 15–18) covers rules, defaults, long names, relations, triggers, referential integrity, stored procedures, and views. **Part V — Performance and Integrity** (Chapters 19–23) covers Rushmore, performance tuning and size limits, integrity checks, corruption prevention and repair, and data dictionary queries. **Part VI — Working with Real Data** (Chapters 24–28) covers dates, times, and numbers, data cleaning, reporting queries, import and export, and a capstone summary layer. The appendices provide a SQL and Xbase cheat sheet, templates and reference tables, a data integrity and maintenance checklist, and a glossary.

Each chapter ends with **Try It Yourself** exercises and a **Key Takeaways** box for quick review.

The Running Example: Marrowgate Marine Supply

Throughout the book we work with the data of **Marrowgate Marine Supply**, a fictional parts distributor on the quay of the harbor town of Marrowgate. It sells rope, anchors, paints, marine electrics, safety gear, and engine parts to boatyards, marinas, and retail chandleries, and it buys from six suppliers. Its database has eight tables — customers, suppliers, products, orders, order lines, purchase orders, purchase-order lines, and stock movements — holding six years of history: 20,010 orders, almost 60,000 order lines, and nearly 65,000 stock movements. Nadia Cole, Marrowgate's data developer, owns the database; Hugh Tremayne in finance wants trustworthy reports; and Isla Penrose in the warehouse wants to know what to reorder and which deliveries are late.

Chapter 1 shows you how to build the complete database with a setup program that is printed in full in Appendix B. It is deterministic: the same program produces exactly the same rows on every computer, so every count, total, and query result in this book can be reproduced on yours. The book's "today" is Tuesday, March 31, 2026, the end of Marrowgate's first quarter.

Versions and Conventions

The book targets **Visual FoxPro 9.0 SP2**, the final release of the product. Almost everything also applies to earlier VFP 9 builds, and much of it to VFP 8; where a feature first appeared in VFP 9 we say so. Microsoft no longer sells or develops Visual FoxPro, so you need an existing licensed copy to run the examples. The examples assume US settings — `SET DATE AMERICAN`, `SET CENTURY ON`, and the dollar sign — so dates appear as `03/31/2026`. Unless a chapter says otherwise, examples assume `SET DELETED ON`, `SET EXCLUSIVE OFF`, `SET MULTILOCKS ON`, and `SET ENGINEBEHAVIOR 90`.

> **TIP**
>
> A recommended practice or a technique that makes your work easier or your data safer.

> **NOTE**
>
> Background information, version differences, or a useful aside.

> **WARNING**
>
> A common mistake or something that can lose data, corrupt a table, or return wrong results.

VFP keywords appear in uppercase and field names in lowercase. Code and output appear in `monospace`; long commands are split with VFP's semicolon continuation. Output blocks show exactly what Visual FoxPro printed, with trailing spaces removed. Money in prose is written as `$1,469.25`; in VFP output, Currency values print with four decimal places, such as `1469.2500`.

Contents

Preface

PART I — TABLES, TYPES, AND INDEXES

1 The Data Side of Visual FoxPro: Engine, Files, and the Marrowgate Database

2 Table Structures and Data Types: Choosing Fields That Last

3 Work Areas and Xbase Data Commands: USE, LOCATE, SCAN, and REPLACE

4 CDX Indexes and Tag Design

5 NULLs, Defaults, Code Pages, and Collation

PART II — SELECT-SQL IN DEPTH

6 SELECT-SQL Fundamentals: Columns, Filters, Sorting, and Destinations

7 Joins in Depth: Inner, Outer, Self, and Multi-Table Queries

8 Grouping and Aggregates: GROUP BY, HAVING, and ENGINEBEHAVIOR 70/80/90

9 Subqueries, UNION, TOP N, and Derived Tables

10 Cursors in Depth: READWRITE, NOFILTER, and Cursor Techniques

PART III — CHANGING DATA SAFELY

11 INSERT, UPDATE, and DELETE-SQL, Including Correlated UPDATE FROM

12 Xbase and SQL Side by Side: Equivalence, Semantics, and Choosing the Right Tool

13 Buffering: Row and Table Buffering, TABLEUPDATE, and Update Conflicts

14 Transactions: BEGIN TRANSACTION, ROLLBACK, and Error-Safe Data Changes

PART IV — THE DATABASE CONTAINER

15 Building the DBC: Long Names, Defaults, Validation Rules, and Captions

16 Persistent Relations, Triggers, and Referential Integrity

17 Stored Procedures: Data-Layer Logic Inside the DBC

18 Local Views: Parameterized, Updatable, and Multi-Table Views

PART V — PERFORMANCE AND INTEGRITY

19 Rushmore Optimization: SYS(3054), SET OPTIMIZE, and Optimizable Expressions

20 Performance Tuning: Large Tables, Network Data, and the 2 GB Limit

21 Data Integrity Checks: Finding Orphans, Duplicates, and Drift

22 Corruption Prevention and Repair: VALIDATE DATABASE, PACK, and REINDEX Safely

23 Data Dictionary Queries: ADBOBJECTS, AFIELDS, DBGETPROP, and Self-Documenting Databases

PART VI — WORKING WITH REAL DATA

24 Dates, Times, and Numbers: Calculations Without Surprises

25 Data Cleaning and De-Duplication

26 Reporting Queries: Crosstabs, Running Totals, Rankings, and Period Comparisons

27 Import and Export: APPEND FROM, COPY TO, CSV, Excel, XML, and CURSORTOXML

28 Capstone: A Data Warehouse–Style Summary Layer for Marrowgate

APPENDICES

A SQL and Xbase Cheat Sheet

B Templates and Reference Tables

C Data Integrity and Maintenance Checklist

D Glossary

PART I

Tables, Types, and Indexes

How the Visual FoxPro data engine stores and finds data: the file family, table structures and every data type, work areas and Xbase commands, CDX index tags designed for real queries, and the quiet details — NULLs, defaults, code pages, and collation — that decide whether results are right.

1 · The Data Side of Visual FoxPro

2 · Table Structures and Data Types

3 · Work Areas and Xbase Data Commands

4 · CDX Indexes and Tag Design

5 · NULLs, Defaults, Code Pages, and Collation

CHAPTER 1 · PART I — TABLES, TYPES, AND INDEXES

The Data Side of Visual FoxPro: Engine, Files, and the Marrowgate Database

Before we write a single serious query, we need a clear picture of what the Visual FoxPro data engine is, which files it reads and writes, and what database we will use for the rest of the book. By the end of this chapter you will have the Marrowgate Marine Supply database on your own disk, you will have opened it, looked inside it, and run the first query that the business actually needs.

What the Visual FoxPro Data Engine Is

Visual FoxPro is often remembered for its form designer and its object-oriented language, but underneath all of that sits a data engine that many teams still depend on every working day. That engine is the subject of this book. It is **local** and **file-based**: there is no server process to start, no service to install, and no network protocol between your program and your data. When your code runs `USE orders`, the engine opens a file on a disk, reads its header, and from that moment your program is the database engine for that table. If ten users open the same table from a shared folder, there are ten copies of the engine, one inside each running program, and they coordinate with each other through operating-system file and record locks.

This design has consequences that run through every chapter. The engine can do things that a server database makes awkward, such as walking a table record by record while you decide what to do with each one. On the other hand, everything that a server would guarantee centrally — integrity rules, transactions, crash safety — has to be expressed in the files themselves or in the code that opens them. Much of Parts III, IV, and V is about doing exactly that.

The second defining trait is that Visual FoxPro gives you **two languages over the same tables**. The first is the record-oriented Xbase language inherited from dBASE and FoxBase: `USE`, `GO`, `SKIP`, `LOCATE`, `SCAN`, `REPLACE`, `SEEK`. These commands work on one table at a time, through a record pointer, in the current work area. The second is SELECT-SQL and the data-changing SQL commands `INSERT`, `UPDATE`, and `DELETE`, which are set-oriented: you describe the rows you want and the engine decides how to find them. Both languages read and write the very same DBF files, use the very same indexes, and honor the same settings such as `SET DELETED`. You do not choose one language when you create a table; you choose, statement by statement, whichever expresses the task more clearly. Chapter 3 covers the Xbase side, Part II covers SELECT-SQL, and Chapter 12 puts the two side by side.

The third trait is **Rushmore**, the query optimizer that uses index tags to build bitmaps of matching records before reading any data. Rushmore works for SELECT-SQL and for many Xbase commands with a `FOR` clause. Whether a query is optimized depends on whether its conditions match an index

tag expression exactly, which is why index design (Chapter 4) and Rushmore (Chapter 19) receive so much attention later on.

Everything in this book targets **Visual FoxPro 9 (SP2)**, the final release. Microsoft ended extended support in January 2015 and no longer sells the product, so you need an existing licensed copy. The data engine behavior we describe is the same across the VFP 9 service packs; where a feature was added in an earlier version we say so, because many shops still maintain data that was designed in older versions.

> **NOTE**
> This book is about the data side of Visual FoxPro. Forms, classes, menus, and reports appear only in passing. If you can open the Command window, type `USE customers`, browse a table, and run a small program, you know enough to follow every example.

The File Family

A Visual FoxPro database is a set of ordinary files in a folder. Understanding which file holds what is the first step to working with the data safely, because backups, copies, repairs, and network problems all happen at the file level.

Extension	What it holds	Opened by
`.DBF`	The table: a header describing the fields, then fixed-length records	`USE`, SELECT-SQL
`.FPT`	Memo, Blob, and General field contents for one table	Automatically, with its DBF
`.CDX`	The structural compound index: all index tags of one table	Automatically, with its DBF
`.DBC`	The database container: a table describing tables, fields, relations, views, and rules	`OPEN DATABASE`
`.DCT`	The memo file of the database container (stored procedures, rule text, properties)	With the DBC
`.DCX`	The index of the database container	With the DBC
`.IDX`	A legacy standalone single-key index	`SET INDEX TO`

Three points about this family matter from day one. First, the **DBF, its FPT, and its CDX travel together**. If you copy `customers.dbf` to another folder without `customers.fpt`, the table will not open, because its header says a memo file exists. If you copy it without `customers.cdx`, VFP will complain that the structural index is missing. Copy the three as a set, or better, use `COPY TO ... WITH CDX` or a backup routine that knows the rules (Chapter 22).

Second, the **structural CDX opens automatically**, and every tag in it is kept up to date whenever any program changes the table. The old standalone `.IDX` files were only updated when a program remembered to open them.

Third, **every file has a 2 GB limit**, and a DBF, its FPT, and its CDX are each limited separately. Chapter 20 shows how to estimate growth toward that ceiling. Other limits: 255 fields per table, records up to 65,500 bytes, and Character fields up to 254 characters.

Looking at the files on disk

Once you have built the Marrowgate database (we do that in a moment), a small program shows the whole family. `ADIR()` fills an array with file names and sizes; we keep only the data-related extensions.

q_files.prg

```
* q_files.prg - the file family behind the database
SET TALK OFF
LOCAL laF[1], lnI, lnN
lnN = ADIR(laF, "*.*")
=ASORT(laF)
FOR lnI = 1 TO lnN
   IF INLIST(UPPER(JUSTEXT(laF[lnI, 1])), "DBF", "FPT", "CDX", "DBC", "DCT", "DCX")
      ? PADR(LOWER(laF[lnI, 1]), 16), TRANSFORM(laF[lnI, 2], "999,999,999")
   ENDIF
ENDFOR
```

Output

```
customers.cdx         13,824
customers.dbf         33,249
customers.fpt            512
marrowgate.dbc        16,228
marrowgate.dct         3,712
marrowgate.dcx         6,144
orderlines.cdx       901,120
orderlines.dbf     1,970,160
orders.cdx           334,336
orders.dbf           740,891
polines.cdx            6,144
polines.dbf              625
products.cdx           6,144
products.dbf           5,837
purchorders.cdx        4,608
purchorders.dbf          665
rawcust.dbf            2,517
stockmoves.cdx       640,512
stockmoves.dbf     2,009,599
suppliers.cdx          3,072
suppliers.dbf            845
```

Notice what is and is not there. Only `customers` has an FPT file, because it is the only table with a Memo field (`notes`); its FPT is just 512 bytes because every note is still NULL and the file holds only its header. Every table in the database has a CDX, because `setup.prg` gave every one of them a primary key. The free table `rawcust.dbf` has neither an FPT nor a CDX. And there is exactly one DBC/DCT/DCX trio, which describes all eight database tables.

The DBF sizes are not mysterious. A DBF is a header followed by fixed-length records and a single end-of-file byte, so its size is `HEADER() + RECCOUNT() * RECSIZE() + 1`. We check that formula against `orders.dbf` later in this chapter.

Free Tables and Database Tables

A DBF can live on its own as a **free table**, or it can belong to a **database container**, in which case we call it a database table or DBC table. The file format is the same, but a database table gains a long list of capabilities from its container:

- Long table names and field names of up to 128 characters (free tables: 10-character field names).
- A primary key (free tables can only have candidate keys).
- Default values, field validation rules, record validation rules, captions, and comments stored in the DBC.
- Insert, update, and delete triggers, and stored procedures that rules and triggers can call.
- Persistent relations between tables and referential integrity code generated from them.
- Participation in transactions — `BEGIN TRANSACTION` covers database tables only.
- Local views defined in SQL and stored with the data.

Each database table carries a **backlink** in its header: the path of the DBC it belongs to. When you open such a table, VFP reads the backlink and opens the database too, so that rules and triggers can fire. This is why moving a single database table to another folder causes trouble — it still points back at its container. Chapter 15 explains the backlink, how to take a table out of a database properly with `REMOVE TABLE`, and when the repair command `FREE TABLE` is needed; Chapter 22 covers the repair side in more detail.

Free tables suit import staging and data exchange. In Marrowgate, the only free table is `rawcust.dbf`, a dozen messy contact rows from a trade show, a web form, and phone notes that we clean up in Chapter 25.

> **TIP**
>
> Ask a table where it belongs with `CURSORGETPROP("Database", "orders")`. It returns the full path of the DBC for a database table and an empty string for a free table. In a program that must handle both, this is more reliable than guessing from the folder.

Data Sessions in One Paragraph

Every table you open lives in a **data session**: a private set of work areas, open tables, record pointers, and many `SET` values. A program that never thinks about data sessions runs in session 1, the default. Forms can be given a private data session, and code can create one with `CREATEOBJECT("Session")` and switch to it with `SET DATASESSION TO`. Two data sessions can open the same table independently, each with its own record pointer and buffered changes — the easiest way to simulate a second user, which Chapter 13 does. Until then, all our examples run in the default session, and `SET("DATASESSION")` returns 1.

Meet Marrowgate Marine Supply

Every example in this book runs against the same database. **Marrowgate Marine Supply** ("Marrowgate") is a fictional marine-chandlery parts distributor on the quay of the fictional harbor town of Marrowgate. It sells rope, anchors and ground tackle, paints and coatings, marine electrics, safety gear, and engine parts to boatyards, marinas, and retail chandleries, and it buys from six suppliers. Its order and stock system has run on Visual FoxPro data since the early days, and the history has grown to twenty thousand orders.

Three people bring the requests that drive our queries:

- **Nadia Cole** is Marrowgate's data developer. She owns the database and its queries, and she is the role model for the reader: careful, practical, and fond of checking results before trusting them.
- **Hugh Tremayne** is the finance controller. He asks for sales, margin, and month-end figures, and he wants them to reconcile to the cent.
- **Isla Penrose** runs the warehouse and purchasing. She asks what is low on stock, what must be reordered, and which supplier deliveries are late.

The book's "today" is **Tuesday 03/31/2026**, the end of the first quarter of 2026. Whenever a query depends on the current date, we use the literal `{^2026-03-31}`, usually stored in a variable named `ldAsOf`, rather than `DATE()`. That way your output matches ours whatever day you run the examples.

The eight tables

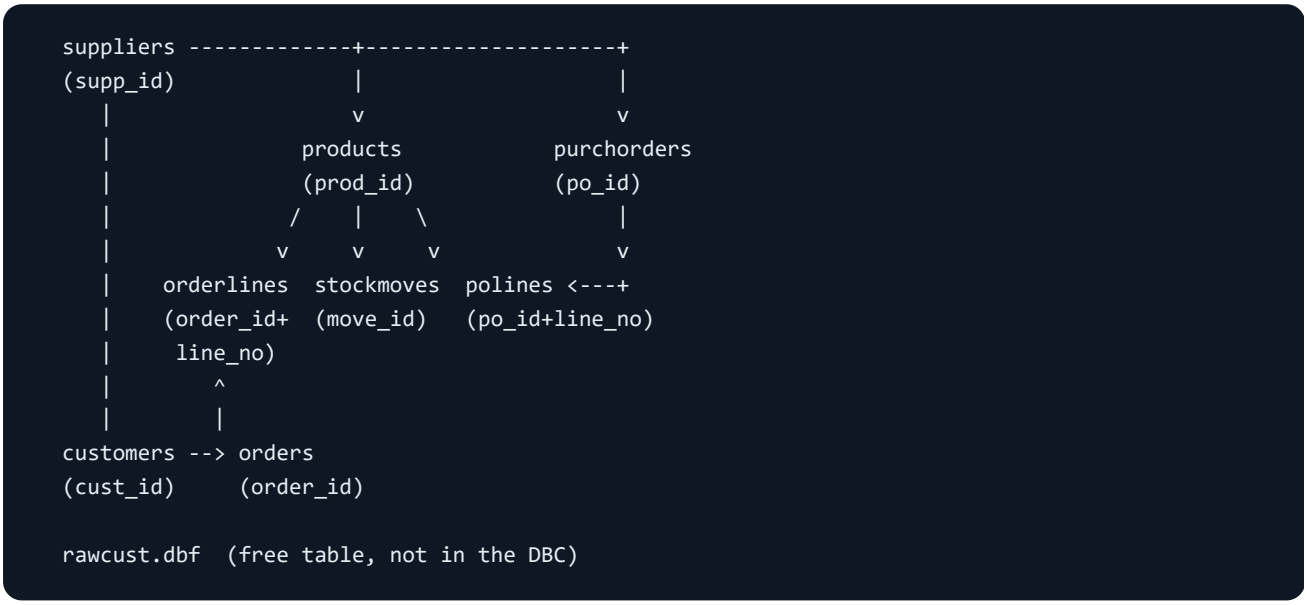
The database `marrowgate.dbc` contains eight tables. Field names are all ten characters or fewer, so the same structures could also exist as free tables.

Table	One row per	Primary key	Records
`suppliers`	supplier	`supp_id`	6
`customers`	boatyard, marina, or chandlery	`cust_id`	200
`products`	stocked item	`prod_id`	60
`orders`	customer order (header)	`order_id` (AUTOINC)	20,010

`orderlines`	line on an order	`order_id` + `line_no`	59,687
`purchorders`	purchase order to a supplier	`po_id` (AUTOINC)	4
`polines`	line on a purchase order	`po_id` + `line_no`	6
`stockmoves`	stock movement (in or out)	`move_id` (AUTOINC)	64,810

The relationships form two trees that meet at `products`. Every product comes from one supplier; every order belongs to one customer; order lines point both at their order and at a product; purchase-order lines point at their purchase order and at a product; and every stock movement belongs to one product.

The Marrowgate schema



A few domain facts help you read the data. Customer types are B (boatyard), M (marina), and R (retail chandlery); regions are NORTH, SOUTH, WEST, and ISLANDS. Order status is O (open), S (shipped), or X (cancelled). Purchase-order status is O (open), R (received), or X (cancelled). Stock movements have a type: OPN for the opening balance, REC for a receipt (positive quantity), SHP for a shipment (negative quantity), and ADJ for a stock-count adjustment (either sign). Product categories are ROPE, ANCHOR, PAINT, ELECTR, SAFETY, and ENGINE. All money fields use the Currency type.

The business rules are just as concrete. A line amount is `ROUND(qty * unit_price * (100 - disc_pct) / 100, 2)`. A line receives a 5 percent volume discount when its gross value is $500.00 or more. Order goods are the sum of the line amounts, and the order total is goods plus freight; there is no tax anywhere in the book. Sales reports always exclude cancelled orders. A product must be reordered when `qty_onhand <= reorder_at` and it is not discontinued. And one invariant holds for every product: its `qty_onhand` equals the sum of its stock movements. Integrity checks in Chapter 21 lean on that rule.

Building the Database with setup.prg

You do not type the data in. The program `setup.prg` builds the entire database from scratch in whatever folder is current: it creates the DBC, the eight tables with their index tags and persistent relations, inserts the canonical rows that the examples talk about (eight named customers, twelve named products, ten orders from 2026, four purchase orders), and generates six years of order and stock history. The history is produced by a small deterministic random-number generator with a fixed seed, so every reader who runs the program gets byte-for-byte the same tables. The complete listing is printed in Appendix B, "The Marrowgate Setup Program"; here we look at a short excerpt to see the style.

setup.prg (excerpt)

```
CREATE TABLE orders ( ;
   order_id   I     NOT NULL AUTOINC NEXTVALUE 10001 STEP 1 PRIMARY KEY, ;
   cust_id    C(6)  NOT NULL REFERENCES customers TAG cust_id, ;
   order_date D     NOT NULL, ;
   ship_date  D     NULL, ;
   status     C(1)  NOT NULL DEFAULT "O", ;
   freight    Y     NOT NULL DEFAULT 0)
INDEX ON order_date TAG order_date
INDEX ON status TAG status
...
FUNCTION mgRnd(tnN)
   * Park-Miller minimal standard generator: deterministic integers 1..tnN
   gnSeed = MOD(gnSeed * 16807, 2147483647)
   RETURN INT(gnSeed / 2147483647 * tnN) + 1
ENDFUNC
```

The `CREATE TABLE` command does a remarkable amount of work in one statement. Because a database is open, the new table joins it. `AUTOINC NEXTVALUE 10001` makes VFP assign order numbers automatically, starting at 10001. `PRIMARY KEY` creates a primary index tag named after the field. `REFERENCES customers TAG cust_id` creates a regular tag on `cust_id` and records a persistent relation from orders to customers in the DBC. `DEFAULT "O"` stores a default value in the DBC, and `NULL` on `ship_date` allows that one field to hold NULL while an order is unshipped. Chapter 2 examines each type and clause in detail.

To run it, create an empty folder, put `setup.prg` in it, make that folder the default, and run the program from the Command window:

```
CD C:\MarrowgateLab
DO setup
```

It finishes in a few seconds and prints a summary. You should see exactly this:

Output

```
Marrowgate setup complete
customers          200
suppliers            6
products            60
orders           20010
orderlines       59687
purchorders          4
polines              6
stockmoves       64810
checksum qty    1779406  amount         19285608.0400
checksum on hand      15545
```

The checksum lines (total line quantity and amount, total quantity on hand) confirm that your copy matches the book.

> **WARNING**
>
> `setup.prg` deletes any existing Marrowgate database in the current folder before it builds a new one. Always run it in a folder of its own — never in a folder that holds real data. That same behavior is what makes it useful: whenever an experiment changes the data (an UPDATE, a DELETE, a PACK, an ALTER TABLE), run `DO setup` again and you are back to the canonical state.

A First Tour

With the database built, we can look around. The first program opens the database, asks it which tables it contains, opens each one, and reports how many records and fields it has.

q_tour.prg

```
* q_tour.prg - a first tour of the Marrowgate database
SET TALK OFF
SET DELETED ON
SET EXCLUSIVE OFF
SET MULTILOCKS ON
SET CENTURY ON
SET ENGINEBEHAVIOR 90
OPEN DATABASE marrowgate
? "Current database:", JUSTFNAME(DBC())
? "Is orders open yet?", USED("orders")
LOCAL laTab[1], lnI, lnN, lcTable
lnN = ADBOBJECTS(laTab, "TABLE")
? "Tables in the DBC:", lnN
FOR lnI = 1 TO lnN
   lcTable = LOWER(laTab[lnI])
   USE (lcTable) IN 0 SHARED
   ? PADR(lcTable, 12), STR(RECCOUNT(lcTable), 6), "records", ;
      STR(FCOUNT(lcTable), 3), "fields"
```

```
ENDFOR
? "Work area of orders:", SELECT("orders")
CLOSE DATABASES ALL
```

Output

```
Current database: MARROWGATE.DBC
Is orders open yet? .F.
Tables in the DBC:          8
suppliers         6 records   4 fields
customers       200 records  10 fields
products         60 records  10 fields
orders        20010 records   6 fields
orderlines    59687 records   6 fields
purchorders       4 records   6 fields
polines           6 records   5 fields
stockmoves    64810 records   6 fields
Work area of orders:                    4
```

Several things are worth noticing. `OPEN DATABASE` makes the database current, but it opens no tables: `USED("orders")` is still `.F.` afterwards. `ADBOBJECTS()` reads the table names from the DBC itself, in the order they were added. `USE (lcTable) IN 0 SHARED` uses a **name expression** — the parentheses tell VFP to evaluate the variable rather than look for a table literally named `lcTable` — and `IN 0` opens each table in the lowest free work area without changing the current one. Orders was the fourth table opened, so it landed in work area 4. `SHARED` lets other users open the table too. `CLOSE DATABASES ALL` closes everything.

Looking at a table's structure

In the Command window you would type `USE orders` and then `DISPLAY STRUCTURE`, which prints a table of the fields with their type, width, decimals, index flag, collation, and NULL setting, and for AUTOINC fields the next value and step. For orders it reports `Integer (AutoInc)` for `order_id` with next value 30011 and step 1 — the setup program inserted ten thousand plus twenty thousand orders, so 30011 is next. Its full listing is wider than a printed page, so in programs we often read the structure with `AFIELDS()` instead, which fills an array with one row per field and lets us print just the columns we want.

q_structure.prg

```
* q_structure.prg - the structure of orders, read with AFIELDS()
SET TALK OFF
SET DELETED ON
SET EXCLUSIVE OFF
OPEN DATABASE marrowgate
USE orders IN 0 SHARED
LOCAL laF[1], lnI
FOR lnI = 1 TO AFIELDS(laF, "orders")
   ? PADR(LOWER(laF[lnI, 1]), 11), laF[lnI, 2], STR(laF[lnI, 3], 3), ;
      STR(laF[lnI, 4], 2), IIF(laF[lnI, 5], "NULL", "NOT NULL")
ENDFOR
```

```
? "Record size:", RECSIZE("orders"), "Header size:", HEADER("orders")
? "Expected file size:", HEADER("orders") + RECCOUNT("orders") * RECSIZE("orders") + 1
CLOSE DATABASES ALL
```

Output

```
order_id    I   4  0 NOT NULL
cust_id     C   6  0 NOT NULL
order_date  D   8  0 NOT NULL
ship_date   D   8  0 NULL
status      C   1  0 NOT NULL
freight     Y   8  4 NOT NULL
Record size:         37 Header size:        520
Expected file size:                 740891
```

The expected size is exactly the 740,891 bytes that `q_files.prg` reported for `orders.dbf`. The record size is one more than the visible fields add up to (36 bytes including the one-byte deletion flag) because a table with at least one NULL-capable field carries a hidden system field that stores the NULL flags. That hidden field also adds an entry to the header, which is why orders has a 520-byte header while `stockmoves`, with the same number of visible fields and no NULL fields, has 488. Chapter 2 takes the structure apart byte by byte.

The first real query: the reorder list

Isla's first question every morning is the same: what do we need to reorder? The rule is that a product is due when its quantity on hand is at or below its reorder point and it is not discontinued. In SQL that is one statement.

q_reorder.prg

```
* q_reorder.prg - the first query: what must Isla reorder today?
SET TALK OFF
SET DELETED ON
SET EXCLUSIVE OFF
SET CENTURY ON
SET ENGINEBEHAVIOR 90
OPEN DATABASE marrowgate
SELECT p.prod_id, p.descr, p.qty_onhand, p.reorder_at, p.supp_id ;
   FROM products p ;
   WHERE p.qty_onhand <= p.reorder_at AND NOT p.discont ;
   ORDER BY p.prod_id ;
   INTO CURSOR curReorder
? "Products to reorder:", _TALLY
SELECT curReorder
SCAN
   ? curReorder.prod_id, curReorder.descr, ;
      STR(curReorder.qty_onhand, 5), STR(curReorder.reorder_at, 5), curReorder.supp_id
ENDSCAN
USE IN curReorder
CLOSE DATABASES ALL
```

Output

```
Products to reorder:          4
AN-2002 Stainless Anchor Swivel 10 mm                9    12 SP006
EL-4002 Marine Battery Switch                        0     8 SP003
PT-3002 Two-Part Epoxy Primer 1 L                   20    20 SP002
RP-1002 Nylon Anchor Rode 14 mm                     300   500 SP001
```

Four products, and one of them deserves a second look: PT-3002 has 20 on hand and a reorder point of 20. It is on the list because the comparison is `<=`. Whether "at the reorder point" should trigger a reorder is a business decision, and Marrowgate has made it: yes. A query that silently used `<` would drop the epoxy primer and nobody would notice until a boatyard was turned away.

Even this small query shows the conventions we follow everywhere. The table has a short lowercase alias (`p`); every column is qualified with it; the query has an explicit `INTO CURSOR` destination rather than relying on the default browse window; and `_TALLY` reports how many rows the query produced. After the query, the result cursor `curReorder` is the current work area, so the `SCAN` loop walks it directly. When we are done, `USE IN curReorder` closes it and releases its temporary file. Chapter 6 covers every clause of SELECT-SQL in detail, and Chapter 10 explains what a cursor really is.

The Settings This Book Assumes

Visual FoxPro has hundreds of `SET` commands, and a handful of them change what queries return. Rather than leave those to chance, every example program in this book starts with the settings it depends on. These are the ones we assume unless a section says otherwise:

Setting	Our value	Why it matters
`SET DELETED`	ON	Records marked for deletion are hidden from Xbase commands and from SQL
`SET EXCLUSIVE`	OFF	`USE` opens tables shared, as multi-user code must
`SET MULTILOCKS`	ON	Required for table buffering and several locks at once (Chapter 13)
`SET CENTURY`	ON	Dates print with four-digit years: 03/31/2026
`SET ENGINEBEHAVIOR`	90	The VFP 9 SQL rules (the default); 70 and 80 are compared in Chapter 8

The factory default for `SET DELETED` is OFF, and for `SET EXCLUSIVE` it is ON in the development environment, so a fresh VFP session behaves differently from the book until you change them. You can put the settings in a startup program, set them in the Options dialog, or simply begin each program with them, as our examples do. Dates in the book appear in the American format, MM/DD/YYYY, with `SET DATE AMERICAN`; money values print the way Visual FoxPro prints Currency, with four decimal places, unless a program formats them.