


```
SELECT w.Code, COUNT(*) AS Shipments,
       SUM(s.Price) AS Revenue
FROM Ops.Shipments AS s
JOIN Ops.Warehouses AS w
  ON w.WarehouseID = s.OriginWarehouseID
WHERE s.CreatedAt >= @From
GROUP BY w.Code
OPTION (RECOMPILE);
```

THE COMPLETE PRACTICAL GUIDE

Microsoft SQL Server for Developers and DBAs

T-SQL, Performance Tuning, and Administration in Practice

T-SQL · Window Functions · Execution Plans · Indexing · Query Store · Security · Backup & Restore · HA/DR

28 CHAPTERS · TESTED ON SQL SERVER 2022

THENEXBOOKS

Microsoft SQL Server for Developers and DBAs

T-SQL, Performance Tuning, and Administration in Practice

Write, Tune, Secure, and Run SQL Server Databases with Confidence

First Edition

THENEXBOOKS

Microsoft SQL Server for Developers and DBAs
T-SQL, Performance Tuning, and Administration in Practice

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 the sample database script 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. SQL Server features, editions, and licensing change between versions; always consult the official documentation for the version you use. Execution plans, I/O figures, and timings vary by machine, data, version, and configuration. Test every administrative procedure, especially backup and restore, in a non-production environment first.

Microsoft, SQL Server, Azure, Windows, and Visual Studio Code 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 not affiliated with or endorsed by Microsoft.

Bluewater Freight, the sample company in this book, is fictional, and all of its data is synthetic. Any resemblance to real companies or people is coincidental.

Preface

SQL Server runs a vast amount of the world's business software: order systems, finance platforms, hospital and school records, line-of-business applications, and the reporting that sits on top of them. Many developers meet it on their first job, and many of them, sooner or later, become the person everyone asks when the database is slow, a deployment needs a schema change, or a backup has to be restored at an inconvenient hour.

This book is written for that journey. It starts with the essentials of T-SQL and grows into the skills that separate someone who can write queries from someone who can be trusted with a production database: designing schemas that protect data, writing set-based code, handling errors and transactions correctly, reading execution plans, choosing indexes, diagnosing blocking and waits, securing access, and planning backups that actually restore.

Every example runs against the same realistic sample database, **Bluewater Freight**, a fictional European parcel and freight carrier with warehouses, customers, shipments, tracking events, drivers, and invoices. The T-SQL in this book was run on SQL Server 2022, and the results you see were copied from that output. You can follow along on the free Express or Developer edition, change a condition, and see what happens.

Who This Book Is For

- **Application developers** who build on SQL Server and want to go beyond ORMs and simple CRUD.
- **New and "accidental" DBAs** who have inherited responsibility for SQL Server instances.
- **Developers coming from other databases** such as PostgreSQL, MySQL, or Oracle who need to work in T-SQL.
- **Students and career changers** preparing for database, backend, or data roles.

No prior SQL Server experience is required. Basic programming knowledge helps, and readers who already know SQL can move quickly through Part I.

What You Will Learn

- Installing SQL Server and the tools, and creating databases, schemas, tables, and the right data types.
- Querying with joins, subqueries, grouping, window functions, common table expressions, and `APPLY`.
- Changing data safely with `MERGE`, `OUTPUT`, transactions, and `TRY...CATCH`.
- Views, stored procedures, functions, triggers, JSON, temporal tables, and safe dynamic SQL.
- Execution plans, indexing, statistics, parameter sniffing, Query Store, and intelligent query processing.

- Locking, isolation levels, row versioning, deadlocks, DMVs, wait statistics, and Extended Events.
- Security, backup and restore, maintenance, high availability, and connecting applications.

How This Book Is Organized

Part I — Getting Started with SQL Server (Chapters 1–5) sets up your environment and teaches the core of T-SQL. **Part II — T-SQL in Depth** (Chapters 6–11) covers aggregation, window functions, CTEs, data modification, programmable objects, and transactions. **Part III — Design and Advanced Features** (Chapters 12–16) is about integrity, dates, strings and JSON, temporal tables, auditing, and dynamic SQL. **Part IV — Performance Tuning** (Chapters 17–22) explains how SQL Server runs queries and how to make them fast and predictable. **Part V — Administration** (Chapters 23–26) covers security, backup and restore, maintenance, and high availability. **Part VI — SQL Server in Practice** (Chapters 27–28) connects applications and finishes with a complete case study. The appendices provide a T-SQL cheat sheet, a DMV reference, a production readiness checklist, and a glossary. Each chapter ends with a **Key Takeaways** box.

The Sample Database: Bluewater Freight

The Bluewater Freight database contains 8 warehouses, 600 customers, 64 employees in a management hierarchy, a vehicle fleet, 24,000 shipments with prices and service levels, over 118,000 tracking events with JSON details, and monthly invoices with their lines. It is small enough for any laptop and the free Express edition, and rich enough to raise real questions: late deliveries, unpaid invoices, missing values, skewed customers, and tables that need better indexes. The script `bluewater.sql` is supplied with this book, and Chapter 2 shows you how to create the database.

Conventions Used in This Book

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

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

> **WARNING**
> A common mistake that produces wrong results, slow queries, data loss, or security problems.

T-SQL keywords appear in uppercase, and every object is schema-qualified. Results are shown in the style of SSMS "Results to Text" and may be shortened with "...".

Contents

Preface

PART I — GETTING STARTED WITH SQL SERVER

1 SQL Server Today: Editions, Versions, Tools, and What We Will Build

2 Installing SQL Server and the Tools, and the Bluewater Freight Database

3 Databases, Schemas, Tables, and Data Types

4 Querying Basics: SELECT, WHERE, ORDER BY, TOP, and Built-In Functions

5 Joins, Subqueries, and Set Operators

PART II — T-SQL IN DEPTH

6 Grouping, Aggregation, GROUPING SETS, and PIVOT

7 Window Functions: Ranking, Running Totals, LAG/LEAD, and Analytics

8 Common Table Expressions, Recursive Queries, and APPLY

9 Modifying Data: INSERT, UPDATE, DELETE, MERGE, and OUTPUT

10 Views, Stored Procedures, and User-Defined Functions

11 Transactions and Error Handling: TRY...CATCH, THROW, and XACT_ABORT

PART III — DESIGN AND ADVANCED FEATURES

12 Database Design: Keys, Constraints, Normalization, and Data Integrity

13 Dates, Strings, and JSON in T-SQL

14 Temporal Tables, Sequences, Computed Columns, and Ledger Basics

15 Triggers, Change Tracking, Change Data Capture, and Auditing

16 Dynamic SQL Done Safely: sp_executesql, Parameters, and SQL Injection

PART IV — PERFORMANCE TUNING

17 How SQL Server Runs a Query: Execution Plans and Their Operators

18 Indexing: Clustered, Nonclustered, Covering, Filtered, and Columnstore

19 Statistics, Cardinality Estimation, and Parameter Sniffing

20 Query Store and Intelligent Query Processing

21 Concurrency: Locking, Blocking, Isolation Levels, and Deadlocks

22 Troubleshooting with DMVs, Wait Statistics, and Extended Events

PART V — ADMINISTRATION

23 Security: Logins, Users, Roles, Permissions, RLS, and Encryption

24 Backup, Restore, and Recovery Models

25 Maintenance and Automation

26 High Availability and Disaster Recovery

PART VI — SQL SERVER IN PRACTICE

27 Connecting Applications: Drivers, Pooling, ORMs, and Pagination

28 Case Study: Taking Bluewater Freight from Schema to Production

APPENDICES

A T-SQL Cheat Sheet

B DMV and System View Reference

C Production Readiness Checklist

D Glossary

PART I

Getting Started with SQL Server

Editions, tools, and your own SQL Server with the Bluewater Freight database, then the core of T-SQL: tables and data types, querying, filtering, joins, and subqueries.

1 · SQL Server Today

2 · Installing SQL Server and the Tools

3 · Databases, Schemas, Tables, and Data Types

4 · Querying Basics

5 · Joins, Subqueries, and Set Operators

CHAPTER 1 · PART I — GETTING STARTED WITH SQL SERVER

SQL Server Today: Editions, Versions, Tools, and What We Will Build

Before we write a single query, we need a map. This chapter explains what Microsoft SQL Server is, how its versions and editions differ, which tools you will use every day, and introduces Bluewater Freight, the fictional parcel carrier whose database we will query, tune and administer throughout the book.

What SQL Server Is

Microsoft SQL Server is a relational database management system. At its heart is the **Database Engine**, a service that stores data in tables, protects it with transactions and locks, answers queries written in **Transact-SQL** (T-SQL), and writes every change to a transaction log so that the data survives crashes and can be restored from backups. When people say "SQL Server" in everyday conversation, they almost always mean the Database Engine, and that is the focus of this book.

The product ships with several other components that you may meet in the installer or in job descriptions:

- **SQL Server Agent** is a scheduler that runs jobs, such as nightly backups and index maintenance, and sends alerts. It is part of the paid editions and Developer edition, but not Express.
- **Integration Services (SSIS)** moves and transforms data between systems (extract, transform, load).
- **Reporting Services (SSRS)** publishes paginated reports.
- **Analysis Services (SSAS)** hosts analytical models for business intelligence tools.

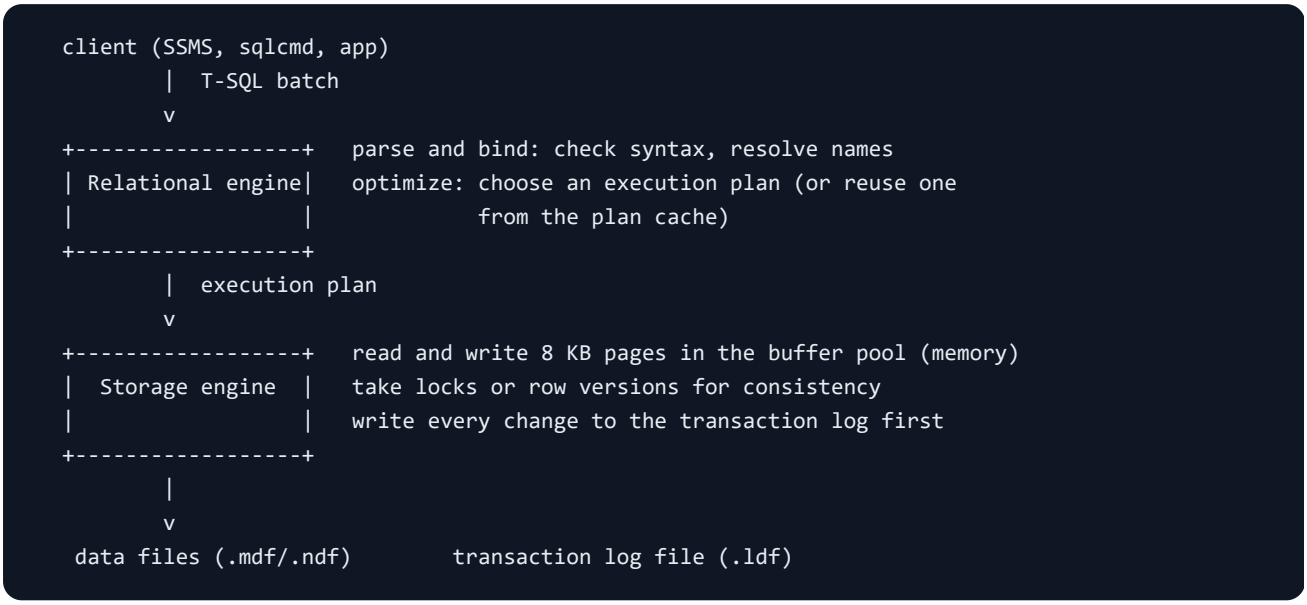
These components are useful, but each deserves its own book. We stay with the engine: how to model data, how to write correct and fast T-SQL, how the engine executes that T-SQL, and how to keep a database secure, backed up and healthy.

A single installation of the Database Engine is called an **instance**. One computer can host several instances side by side, for example a default instance and a named instance called `SQLEXPRESS`. Each instance owns its own set of **databases**. Some databases are system databases that the engine needs (`master`, `model`, `msdb` and `tempdb`); the others are user databases, such as the `BluewaterFreight` database we will create in Chapter 2. Inside a database, objects such as tables and views live in **schemas**, which act like folders. That gives every table a name with up to four parts, although in daily work you will use the two-part form, `Ops.Shipments`.

What Happens When You Run a Query

It helps to have a rough mental model of the engine from the first day, because almost every performance and administration topic later in the book refers back to it. When a client sends a batch of T-SQL, the engine works through it in a few stages:

The life of a query (simplified)



The **relational engine** parses your text, checks that the tables and columns exist, and hands the query to the **query optimizer**. The optimizer does not run your query the way you wrote it; it considers many equivalent strategies (which index to read, which join algorithm to use, in which order to join) and picks the one it estimates to be cheapest, using statistics about the data. The result is an **execution plan**, which is cached so that the next execution of the same query can skip the optimization step. Chapters 17 to 20 are about reading, influencing and stabilizing those plans.

The **storage engine** carries out the plan. Data lives on disk in 8 KB pages, but the engine reads and changes pages in memory, in an area called the **buffer pool**. Before a changed page is ever written back to the data file, a description of the change is written to the **transaction log**. That write-ahead rule is what makes transactions durable and what makes point-in-time restores possible, as Chapters 11 and 24 explain. Finally, locks and row versions keep concurrent users from seeing each other's half-finished work, which is the subject of Chapter 21.

You do not need to remember all of this now. Just keep in mind that the text of a query is a request, not a recipe: SQL Server decides *how* to answer it, and much of the craft of working with SQL Server is helping it make good decisions.

Two Readers, One Engine

This book is written for two groups of people who often sit in the same meeting but look at SQL Server from different angles.

Developers build applications that read and write data. They want to know how to express a business question in T-SQL, how to design tables that do not fight the application, how to page through results, how to call stored procedures safely from C#, Python, Node.js or Java, and why a query that ran in a few milliseconds on a laptop suddenly takes seconds in production.

DBAs, including the many "accidental DBAs" who inherited a server because nobody else would, want to know how to install and configure an instance, how to back it up and prove the backups work, how to grant the right permissions and no more, how to find the query that is hurting everyone, and how to keep the system available when hardware fails.

The two roles overlap far more than they differ. A developer who understands execution plans writes better queries; a DBA who understands how the application uses the database tunes it better. Parts I to III lean towards development, Part IV (performance) is shared ground, and Part V leans towards administration. Part VI brings both views together.

Versions and Compatibility Levels

Microsoft releases a new major version of SQL Server every few years and then ships **cumulative updates** (CUs) that fix bugs and sometimes add small features. Each major version has a marketing name, such as SQL Server 2022, and an internal version number, which you will see in tools and in `SERVERPROPERTY` output.

Release	Internal version	Highest compatibility level
SQL Server 2016	13.x	130
SQL Server 2017	14.x	140
SQL Server 2019	15.x	150
SQL Server 2022	16.x	160

The **compatibility level** is a database setting, not a server setting. It controls which T-SQL behaviors and which query optimizer improvements apply to queries that run in that database. A SQL Server 2022 instance can host a database at level 160, 150, 140 and lower, which lets you upgrade the server first and move each database to the new level after testing. A few features we use in this book work only when the database is at level 160: `GENERATE_SERIES`, the `WINDOW` clause, the `LEADING`, `TRAILING` and `BOTH` options of `TRIM`, and Parameter Sensitive Plan optimization. Other SQL Server 2022 additions, such as `DATE_BUCKET`, `IS [NOT] DISTINCT FROM`, the ordinal argument of `STRING_SPLIT`, `JSON_OBJECT` and `JSON_ARRAY`, also work at lower levels.

Every example in this book was written for and tested on **SQL Server 2022 Express edition, with the database at compatibility level 160**. If you use SQL Server 2022 or a later release, and set the same level as Chapter 2 shows, your results will match ours. You can ask any instance what it is with `SERVERPROPERTY`:

Query

```
SELECT SERVERPROPERTY('ProductVersion') AS ProductVersion,
       SERVERPROPERTY('ProductLevel')   AS ProductLevel,
       SERVERPROPERTY('Edition')        AS Edition,
       SERVERPROPERTY('EngineEdition')  AS EngineEdition;
```

Result

```
ProductVersion ProductLevel Edition                 EngineEdition
-------------- ------------ ----------------------- -------------
16.0.1000.6    RTM          Express Edition (64-bit) 4
```

Our test instance reports version 16.0, the SQL Server 2022 release line, at the original release level (`RTM`). Your build number will probably be higher because you will install a newer cumulative update, which is exactly what you should do. `EngineEdition` 4 means Express; the documentation for `SERVERPROPERTY` lists the other values. The database compatibility level is stored in `sys.databases`:

Query

```
SELECT compatibility_level
FROM sys.databases
WHERE database_id = DB_ID();
```

Result

```
compatibility_level
-------------------
160
```

> **NOTE**
>
> Newer releases, such as SQL Server 2025, add features like a native `json` data type, vector support and regular-expression functions. We mention them where they are relevant, but we do not show their syntax as tested, because our test platform is SQL Server 2022. Check the current Microsoft documentation before relying on them.

Editions: Which One Do You Need?

The Database Engine is the same code in every edition. Editions differ in licensing, in limits on memory, CPU and database size, and in which advanced features are switched on.

Edition	Cost	Typical use	Key limits and notes
Express	Free	Learning, small apps, embedded use	10 GB maximum per database; capped memory and CPU; no SQL Server Agent

Developer	Free	Development and testing only	All Enterprise features; not licensed for production
Standard	Paid	Departmental and mid-sized production systems	Limits on memory and cores; some advanced features restricted
Enterprise	Paid	Large, mission-critical systems	Full feature set and scale

Express is what we used to test this book. It is free, quick to install and perfectly adequate for learning, and every T-SQL example in Parts I to IV runs on it. Its limits are real, though: the 10 GB cap applies to each database's data files, and the engine will use only a limited amount of memory and CPU no matter how large the machine is. The exact figures are in the edition comparison page of the documentation. Most importantly for DBAs, Express **does not include SQL Server Agent**, so the scheduling examples in Chapter 25 show standard syntax and explain alternatives such as Windows Task Scheduler with `sqlcmd`.

Developer edition is also free and includes every Enterprise feature, which makes it the better choice if you want to experiment with features that Express lacks. Its license allows development and testing only. If your laptop runs Developer edition and you deploy the same database to Standard edition in production, test on a Standard-like configuration before release, because Developer will happily let you use Enterprise-only features.

Standard and **Enterprise** are the paid editions used in production. Licensing details change over time and are outside the scope of this book; talk to your Microsoft licensing partner.

SQL Server in the Cloud

Microsoft also offers SQL Server as a service in Azure:

- **Azure SQL Database** is a fully managed database. Microsoft runs the instance, patches it and backs it up; you get a database, not a server. Some instance-level features are not available.
- **Azure SQL Managed Instance** is a managed instance that is very close to on-premises SQL Server, including SQL Server Agent and cross-database queries, which makes it a common target for migrations.
- **SQL Server on Azure Virtual Machines** is ordinary SQL Server on a virtual machine that you manage yourself.

The T-SQL in this book works on all of these with very few exceptions, and Chapter 26 discusses the cloud options from an availability point of view.

Windows, Linux and Containers

SQL Server runs on Windows and on several Linux distributions, and Microsoft publishes container images that run under Docker or Kubernetes. The Database Engine and T-SQL are the same on every platform, but a few components and features are Windows-only, so check the "editions and supported

features" pages for Linux if you plan to use it. Chapter 2 shows a Windows installation, which is what most readers will use for learning, and a container option for macOS and Linux users.

The Tools You Will Use

You talk to the Database Engine through client tools. All of them send T-SQL to the server and display the results; they differ in how comfortable they are for a particular job.

Tool	What it is	Best for
SQL Server Management Studio (SSMS)	Windows desktop application from Microsoft	Daily administration, Object Explorer, execution plans, Query Store reports
`sqlcmd`	Command-line client	Scripts, automation, servers without a desktop
VS Code with the MSSQL extension	Cross-platform editor plus Microsoft's SQL extension	Developers who live in VS Code; macOS and Linux users
SQL Server Configuration Manager	Windows management console	Services, network protocols, service accounts

SSMS remains the most complete tool for SQL Server. It has an Object Explorer tree for browsing every object on an instance, graphical execution plans, built-in reports, dialogs for backups and security, and a query editor with IntelliSense. Recent versions of SSMS also support connections to Azure SQL Database and Managed Instance. We show screen-independent steps for SSMS, because menus move a little from version to version.

sqlcmd is a small command-line program that runs T-SQL from a file or from the command line. It is the tool you will use in scripts, scheduled tasks and deployment pipelines, and it is often the only tool available on a locked-down server.

VS Code with the MSSQL extension gives you a lightweight, cross-platform query editor with IntelliSense, a results grid and connection profiles. It is a good choice if you already write application code in VS Code.

> **WARNING**
>
> Azure Data Studio, which you may see in older tutorials, has been retired. Microsoft points its users to the MSSQL extension for VS Code. We do not use Azure Data Studio in this book, and we recommend that you do not start new work with it.

T-SQL in a Nutshell

T-SQL is Microsoft's dialect of SQL. It follows the ISO SQL standard closely for the core statements (`SELECT`, `INSERT`, `UPDATE`, `DELETE`, `CREATE TABLE`) and adds procedural features: variables, `IF` and `WHILE`, error handling with `TRY...CATCH`, stored procedures and functions. Here is the first query we will run against the sample database:

Query

```
SELECT TOP (5) ShipmentID, TrackingNumber, Status, CreatedAt
FROM Ops.Shipments
ORDER BY ShipmentID;
```

Result

```
ShipmentID TrackingNumber Status    CreatedAt
---------- -------------- --------- -------------------
1          BW3000104729   Delivered 2024-01-01 11:17:04
2          BW3000209458   Delivered 2024-01-05 14:22:57
3          BW3000314187   Delivered 2024-01-07 07:37:57
4          BW3000418916   Delivered 2024-01-09 18:44:11
5          BW3000523645   Delivered 2024-01-10 12:58:03
```

A few rules of the road apply to every script in this book:

- **Keywords are not case-sensitive.** `select` and `SELECT` are the same. We write keywords in uppercase so that they stand out.
- **End statements with a semicolon.** T-SQL still accepts most statements without one, but Microsoft has documented that omitting it is deprecated, and some statements, such as those that start with a common table expression or `MERGE`, require it.
- **Schema-qualify every object.** We write `Ops.Shipments`, never just `Shipments`. The engine resolves names faster and with less ambiguity, and readers of your code know exactly which object you mean.
- **`GO` is not T-SQL.** It is a batch separator understood by SSMS, `sqlcmd` and VS Code. The tool splits the script at each `GO` and sends the pieces to the server one at a time. Some statements, such as `CREATE PROCEDURE` and `CREATE VIEW`, must be the first statement in their batch, which is why scripts are full of `GO` lines.
- **Comments** start with `--` for a single line, or are enclosed in `/* ... */` for several lines.

Coming from Another Database?

If you already know PostgreSQL, MySQL or Oracle, most of your SQL knowledge transfers directly. These are the differences that trip people up most often in their first week:

- **No `LIMIT`.** T-SQL uses `TOP (n)` or the standard `OFFSET ... FETCH` clause (Chapter 4).
- **Auto-numbering** uses the `IDENTITY` property or a `SEQUENCE` object, not `SERIAL` or `AUTO_INCREMENT` (Chapters 3 and 14).
- **Default schema is `dbo`**, and schemas are a namespace inside one database. Cross-database queries use three-part names such as `OtherDb.Sales.Customers`.
- **String comparisons are usually case-insensitive**, because the default collation on most installations is case-insensitive. `'retail' = 'Retail'` is true unless you choose a case-sensitive collation (Chapter 3).

- **No Boolean column type.** T-SQL uses `bit` (0 or 1), and a predicate cannot be selected directly as a value.
- **Quoted identifiers** use square brackets, `[Order Details]`, or double quotes when `QUOTED_IDENTIFIER` is on, which is the default for modern clients.
- **Empty string is not NULL.** Unlike Oracle, `''` is a real, zero-length value.
- **String concatenation** uses `+` or the `CONCAT` function, not `||`.

Meet Bluewater Freight

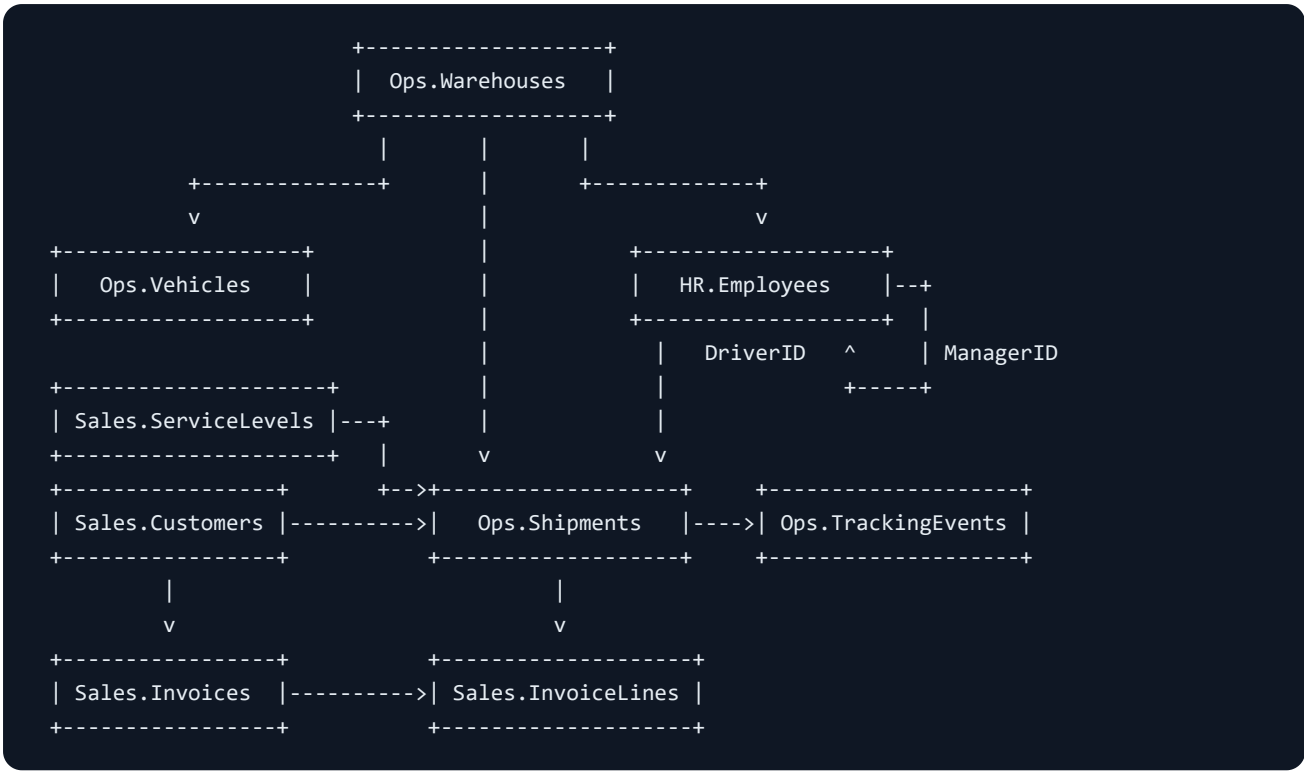
A book about databases needs a database, and a realistic one. **Bluewater Freight** is a fictional European parcel and freight carrier. It runs eight warehouses, in Rotterdam, Hamburg, Antwerp, Lyon, Milan, Gdańsk, Valencia and Oslo, the last of which opened in September 2024. Customers book shipments at one of four service levels, drivers pick them up, the parcels move through the network, and every scan creates a tracking event. At the end of each month, finance sends each customer one invoice that lists the shipments of that month.

That simple story gives us everything we need: reference data, customers with messy real-world attributes, a self-referencing staff hierarchy, a large and growing fact table, an even larger event table with JSON details, and money that must add up. The database uses three schemas: `Sales` for customers, service levels and invoices, `Ops` for warehouses, vehicles, shipments and tracking, and `HR` for employees.

Table	Rows	What it holds
`Ops.Warehouses`	8	Code (RTM, HAM, ANR, LYS, MIL, GDN, VLC, OSL), city, country, opening date, capacity
`Sales.ServiceLevels`	4	Economy, Standard, Express, Overnight, with target days and rates
`Sales.Customers`	600	Name, segment, city, country, email, credit limit, sign-up time
`HR.Employees`	64	Staff with job title, manager, warehouse, hire date and salary
`Ops.Vehicles`	40	Vans, box trucks and tractors assigned to warehouses
`Ops.Shipments`	24,000	One row per shipment, January 2024 to December 2025
`Ops.TrackingEvents`	118,640	Scans such as PickedUp, Exception and Delivered, with JSON details
`Sales.Invoices`	8,352	One invoice per customer per month, January 2024 to November 2025
`Sales.InvoiceLines`	21,630	One line per invoiced shipment

The tables are linked by foreign keys. In the diagram below an arrow points from the referenced (parent) table to the table that references it:

Bluewater Freight relationships



The data is generated by a deterministic script, `bluewater.sql`, which is supplied with the book. "Deterministic" means that every run produces exactly the same rows, so the results printed in this book are the results you will see. Once you have created the database in Chapter 2, this query lists every table with its row count, using the catalog views that we explore in Chapter 3:

Query

```
SELECT CONCAT(s.name, '.', t.name) AS TableName,
       SUM(p.rows)                 AS TotalRows
FROM sys.tables AS t
JOIN sys.schemas AS s ON s.schema_id = t.schema_id
JOIN sys.partitions AS p
  ON p.object_id = t.object_id AND p.index_id IN (0, 1)
GROUP BY s.name, t.name
ORDER BY s.name, t.name;
```

Result

```
TableName           TotalRows
------------------- ---------
HR.Employees        64
Ops.Shipments       24000
Ops.TrackingEvents  118640
Ops.Vehicles        40
Ops.Warehouses      8
Sales.Customers     600
Sales.InvoiceLines  21630
Sales.Invoices      8352
Sales.ServiceLevels 4
```

The Shape of the Data

Real business data is never uniform, and neither is ours. Bluewater Freight is growing, so shipment volume rises over the two years:

Query

```
SELECT YEAR(CreatedAt) AS ShipYear, COUNT(*) AS Shipments
FROM Ops.Shipments
GROUP BY YEAR(CreatedAt)
ORDER BY ShipYear;
```

Result

```
ShipYear Shipments
-------- ---------
2024     6033
2025     17967
```

Most shipments are delivered, but a few hundred are cancelled or returned, some are still in transit at the end of the data, and a handful created on the last day have not even been picked up yet. The status column will give us many chances to talk about skewed data in the performance chapters:

Query

```
SELECT Status, COUNT(*) AS Shipments
FROM Ops.Shipments
GROUP BY Status
ORDER BY Shipments DESC;
```

Result

```
Status    Shipments
--------- ---------
Delivered 22872
Cancelled 470
Returned  438
InTransit 203
Created   17
```

Each shipment leaves a trail in `Ops.TrackingEvents`. Here is the full journey of one shipment from Antwerp to Zürich, found by its tracking number. Notice the customs hold on the way and the JSON text in the `Details` column, which Chapter 13 teaches you to query:

Query

```
SELECT e.EventTime, e.EventType, e.Location, e.Details
FROM Ops.TrackingEvents AS e
JOIN Ops.Shipments AS s ON s.ShipmentID = e.ShipmentID
WHERE s.TrackingNumber = 'BW5118144025'
ORDER BY e.EventTime;
```

Result

```
EventTime           EventType      Location Details
------------------- -------------- -------- ----------------------------------------
2025-11-01 15:41:19 Created        Antwerp  NULL
2025-11-01 17:41:19 PickedUp       Antwerp  NULL
2025-11-02 11:41:19 InTransit      Antwerp  NULL
2025-11-02 21:41:19 Exception      Zürich   {"reason":"Customs hold","attempt":2}
2025-11-06 05:00:00 OutForDelivery Zürich   NULL
2025-11-06 09:00:00 Delivered      Zürich   {"signedBy":"Warehouse desk","photo":true}
```

The data also contains deliberate imperfections. Some customers have no city or email (`NULL`), some names contain characters outside plain ASCII, such as *Müller & Sohn* and *Société Lumière*, and forty customers who signed up in late 2025 have never shipped anything. These details are there on purpose, because they are exactly the cases where queries go wrong in real systems.

> **TIP**
>
> Because the script is deterministic and can be re-run, you can experiment freely. If you delete or change rows while practising, run `bluewater.sql` again and the database returns to the exact state the book expects. Chapter 2 shows how.

A Taste of What Is Coming

By the end of Part II you will be able to answer business questions like this one without a second thought: *for each warehouse, how many shipments were delivered in 2025, and what share arrived on or before the promised date?*

Query — 2025 deliveries and on-time rate by warehouse

```
SELECT w.Code, w.City,
       COUNT(*) AS Delivered2025,
       CAST(AVG(CASE WHEN CAST(s.DeliveredAt AS date) <= s.PromisedDate
                     THEN 100.0 ELSE 0 END) AS decimal(5,1)) AS OnTimePct
FROM Ops.Shipments AS s
JOIN Ops.Warehouses AS w ON w.WarehouseID = s.OriginWarehouseID
WHERE s.Status = 'Delivered'
  AND s.CreatedAt >= '2025-01-01' AND s.CreatedAt < '2026-01-01'
GROUP BY w.Code, w.City
ORDER BY Delivered2025 DESC;
```

Result

```
Code City      Delivered2025 OnTimePct
---- --------- ------------- ---------
VLC  Valencia  2199          79.8
RTM  Rotterdam 2173          78.0
GDN  Gdańsk    2169          78.8
LYS  Lyon      2151          78.9
OSL  Oslo      2116          79.7
```

```
MIL  Milan     2113          79.1
HAM  Hamburg   2096          78.0
ANR  Antwerp   2043          78.0
```

Do not worry if parts of this query look unfamiliar. Joins arrive in Chapter 5, grouping in Chapter 6, and the half-open date range in the `WHERE` clause, which is the correct way to filter a `datetime2` column by year, is explained in Chapter 4. Later, in Part IV, we will look at how SQL Server actually executes a query like this, why it has to read every row of `Ops.Shipments` today, and which index would change that.

How This Book Is Organized

The book has six parts. You can read it from cover to cover, but each part also stands on its own once you have worked through Part I.

- **Part I — Getting Started** (Chapters 1–5): installation, the sample database, tables and data types, basic queries, joins and set operators.
- **Part II — T-SQL in Depth** (Chapters 6–11): aggregation, window functions, common table expressions, data modification, views, procedures and functions, and transactions with error handling.
- **Part III — Design and Advanced Features** (Chapters 12–16): database design and integrity, dates, strings and JSON, temporal tables and sequences, triggers and change tracking, and safe dynamic SQL.
- **Part IV — Performance Tuning** (Chapters 17–22): execution plans, indexing, statistics and parameter sniffing, Query Store, concurrency, and troubleshooting with DMVs and Extended Events.
- **Part V — Administration** (Chapters 23–26): security, backup and restore, maintenance and automation, and high availability.
- **Part VI — SQL Server in Practice** (Chapters 27–28): connecting applications and a closing case study that takes Bluewater Freight from schema to production.

The appendices contain a T-SQL cheat sheet, a reference of the system views and DMVs used in the book, a production readiness checklist and a glossary.

Conventions Used in This Book

Code that you can type and run appears in blocks captioned **Query**, sometimes with a short description. The output that follows, captioned **Result**, was copied from a real run against the sample database and is formatted like the "Results to Text" mode of SSMS: column names, a dashed line, then the rows. Long results are shortened with `...`. Messages such as `(3 rows affected)` and error messages are shown verbatim in their own result blocks when they matter. Where numbers such as logical reads depend on your hardware, data or version, we say so.