DATA ENGINEERING AND ANALYTICS

# Relational Database Design

Level: Practitioner • 2 days (expandable to 3) • Virtual, In-person

## Overview

A bad database schema is the most expensive kind of technical debt: application code can be refactored in an afternoon, but a schema full of duplicated data and vague relationships gets built on top of for years, and every workaround becomes load-bearing. Most developers were never actually taught database design; they learned SQL, absorbed some table patterns from existing systems, and inherited the consequences. The design skill itself (turning a messy real-world domain into a clean, consistent, evolvable schema) is teachable, and it pays back on every system you touch afterward.

This is a hands-on, practitioner course. It follows the gradient of real design work: first modeling a domain as entities and relationships, then the keys and constraints that make the model trustworthy, then normalization as a reasoning tool rather than a ritual, and finally the practical judgment calls: physical design choices, deliberate denormalization, and evolving a schema that is already in production. In keeping with a less-but-deeper philosophy, the course works one realistic domain from napkin to production-shaped schema rather than skimming many toy examples. Every module ends with a design or SQL lab, and each module builds on the one before.

## Who Should Attend

- Developers who create or modify database schemas and want to do it with confidence
- Data engineers and analysts who inherit schemas and need to judge and improve them
- Technical leads who review data models and want a shared design vocabulary for their team

## Prerequisites

- Working SQL: SELECT with joins, plus basic INSERT and UPDATE (see *SQL Querying and T-SQL Fundamentals* if you need this foundation)
- Experience with any relational database (SQL Server, PostgreSQL, MySQL, or similar)
- No prior formal modeling experience required

## What You Will Learn

- Model a business domain as entities, attributes, and relationships before writing any DDL
- Choose sound primary keys and enforce relationships with foreign keys and constraints
- Apply normalization through third normal form, and explain the anomalies it prevents
- Translate a logical model into a physical schema with appropriate data types and integrity rules
- Judge when and how to denormalize deliberately, and document the tradeoff
- Evolve a live schema safely: migrations, refactoring patterns, and protecting data integrity

## Course Outline

### Day one: from domain to sound model

- Modeling the Domain
  - Entities, attributes, and relationships: seeing the domain before the tables

  - Entity-relationship diagrams that people actually read
  - Common modeling traps: modeling the UI, modeling the report, modeling the spreadsheet
  - Lab: model a realistic business domain as an ER diagram from a written brief
- Keys and Relationships
  - Primary keys: natural versus surrogate, and how to choose honestly
  - Foreign keys and cardinality: one-to-many, many-to-many, and junction tables
  - Optionality, self-references, and hierarchies
  - Lab: turn the ER diagram into tables with keys and relationships in SQL
- Normalization as a Thinking Tool
  - Redundancy and the update, insert, and delete anomalies it causes
  - First, second, and third normal form, taught through repairs rather than definitions
  - How far to go: what higher normal forms buy and when they matter
  - Lab: find and fix the normalization flaws in a deliberately broken schema

### Day two: from sound model to production schema

- Physical Design Choices
  - Data types that protect data: precision, length, dates, and money
  - Constraints as executable business rules: NOT NULL, UNIQUE, CHECK, and defaults
  - Naming conventions and schema organization a team can live with
  - Lab: harden the course schema with types, constraints, and consistent names
- Designing for How Data Is Used
  - Indexes and how schema design shapes query performance
  - Deliberate denormalization: when duplication is a fair trade, and how to contain it
  - Patterns you will meet: audit columns, soft deletes, lookup tables, and temporal data
  - Lab: tune the schema against a realistic query workload and justify each change
- Evolving a Live Schema
  - Migrations: changing a schema without losing data or uptime
  - Refactoring patterns: splitting tables, changing keys, and backfilling safely
  - Reviewing a schema: a practical checklist for judging designs you inherit
  - Lab: plan and execute a non-trivial migration on the course schema

## Extended Version

The three-day version keeps the same gradient and adds depth and a full design cycle:

- Deeper treatment of temporal data, history tracking, and versioned records
- Concurrency and integrity under load: transactions and isolation as design inputs
- A comparative look at how the same domain would be modeled in a document database, and what that teaches about relational strengths
- A capstone that designs, builds, and defends a complete schema for a new business domain, then evolves it against a requirements change