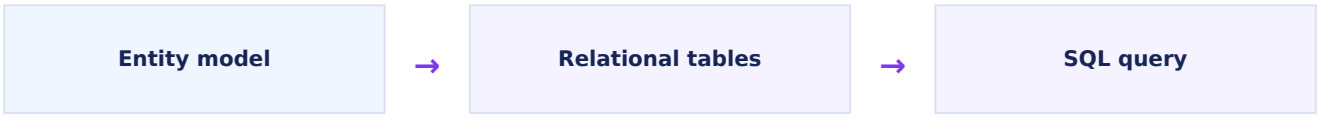
LEARNHUB / TECHNICAL COURSE LIBRARY

Database Management Systems

Practical examples

Design reliable relational data and query it with SQL. Move from tables and keys to normalization, joins, transactions, and indexes.



Level: Beginner · Category: Computer Science

Original, beginner-friendly study content designed to accompany the course syllabus and its linked video lesson.

Contents

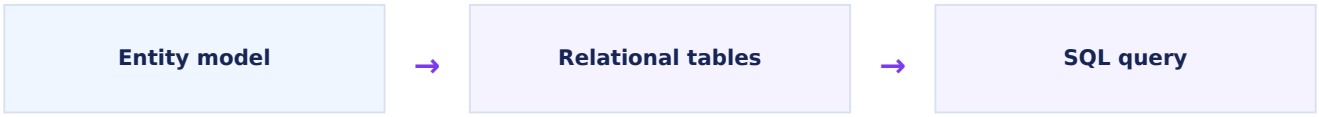
- How to use this resource
- Learning map
- Chapter 01 · Relational tables and SQL
- Chapter 02 · ER diagrams and relationships
- Chapter 03 · Keys and normalization
- Chapter 04 · Queries, joins, and aggregation
- Chapter 05 · Transactions and indexes
- Summary and next steps

Course purpose: Design reliable relational data and query it with SQL. Move from tables and keys to normalization, joins, transactions, and indexes.

How to use this resource

Read the chapter titles first, then study the explanations and examples. Pause at each practice prompt before reading its suggested answer. Write down questions that remain and revisit them during revision.

Learning map



Chapter 01 · Relational tables and SQL

Problem context

Design a minimal table for courses and identify a primary key and two useful columns.

Concept to apply

A relational database stores related data in tables of rows and columns. SQL is a language for defining, querying, and changing that data.

Worked example

An `Orders` table stores order ID, customer ID, order date, and status; a query selects open orders after a specified date.

Walk-through

Give each table a clear purpose. Use primary keys to identify rows and foreign keys to represent references. Start queries with `SELECT` and `FROM`, then filter with `WHERE`; be explicit about columns rather than relying on every column.

Check your result

> Use a stable course ID as the primary key and include a course name and category. Keep one course per row and make the ID unique and non-null.

Chapter 02 · ER diagrams and relationships

Problem context

Model authors and books where a book may have multiple authors. Name the tables and key fields.

Concept to apply

An entity-relationship model names entities, attributes, and relationships before implementation. Cardinality describes how many records may participate in a relationship.

Worked example

Students and courses are many-to-many: one student joins many courses and one course includes many students, so an Enrollment table joins their IDs.

Walk-through

Identify entities from the domain, then determine whether relationships are one-to-one, one-to-many, or many-to-many. A junction table represents a many-to-many relation while preserving each side's identity.

Check your result

> Use Authors and Books plus an AuthorBook junction table with foreign keys to each. Add a composite or surrogate primary key according to the product's requirements.

Chapter 03 · Keys and normalization

Problem context

A table stores comma-separated course IDs in one student row. Identify the design issue and suggest a relational repair.

Concept to apply

A functional dependency says one attribute determines another. Normalization organizes data to reduce update anomalies while maintaining meaningful dependencies.

Worked example

If customer address is copied into every order row, an address update can leave old copies behind; store it once in a customer table when that matches the business rule.

Walk-through

First normal form requires atomic values rather than repeating groups. Second and third normal forms remove dependencies on part of a composite key and on non-key attributes, respectively. Normalization improves consistency; it is not a goal to split tables without reason.

Check your result

> The repeated values are not atomic. Create an Enrollment table with one student-course pair per row, referencing Student and Course.

Chapter 04 · Queries, joins, and aggregation

Problem context

Write the steps for listing each course and its enrollment count, including courses with zero students.

Concept to apply

A join combines rows related by a key. An aggregate computes a summary such as count, sum, or average over a group of rows.

Worked example

A sales report joins orders to customers, groups by customer, and sums order totals to show each customer's spending.

Walk-through

Use explicit join conditions to avoid accidental row multiplication. Filter rows with `WHERE` before grouping and filter groups with `HAVING`. Understand whether a left join should preserve unmatched rows.

Check your result

> Start from Course, left join Enrollment on course ID, group by the course ID and name, and count enrollment identifiers so unmatched courses show zero.

Chapter 05 · Transactions and indexes

Problem context

Choose a likely index for a frequent query filtering enrollments by course ID and explain a trade-off.

Concept to apply

A transaction groups operations into a unit that should satisfy ACID properties: atomicity, consistency, isolation, and durability. An index is an auxiliary structure that speeds selected lookups at storage and write cost.

Worked example

A bank transfer debits one account and credits another in one transaction so a failure cannot leave only one side updated.

Walk-through

Use transactions when multiple changes must succeed or fail together. Add indexes for frequent filters, joins, or ordering after measuring the workload; unnecessary indexes consume storage and slow writes.

Check your result

> An index beginning with course ID can help that filter. It consumes space and adds work on insert, update, and delete, so confirm it matches real query patterns.

Summary and next steps

Database Management Systems brings together 5 connected topics. Review each definition, explain one example without looking, and then try the practice prompt again.

Key ideas to carry forward

Relational tables and SQL: A relational database stores related data in tables of rows and columns. SQL is a language for defining, querying, and changing that data.

ER diagrams and relationships: An entity-relationship model names entities, attributes, and relationships before implementation. Cardinality describes how many records may participate in a relationship.

Keys and normalization: A functional dependency says one attribute determines another. Normalization organizes data to reduce update anomalies while maintaining meaningful dependencies.

Queries, joins, and aggregation: A join combines rows related by a key. An aggregate computes a summary such as count, sum, or average over a group of rows.

Transactions and indexes: A transaction groups operations into a unit that should satisfy ACID properties: atomicity, consistency, isolation, and durability. An index is an auxiliary structure that speeds selected lookups at storage and write cost.

A short review routine

1. Recall the main idea from each chapter before rereading. 2. Work one example by hand. 3. Explain a common mistake and how you would catch it. 4. Revisit a difficult topic after a short break.

Course outcomes

- Model entities and relationships in relational tables
- Write queries that filter and summarize data
- Use keys and normalization to reduce inconsistency
- Explain how transactions and indexes support reliable access

This practical examples is an original LearnHub learning aid. Use the referenced ideas for study, then consult the course video and the rest of the library for additional practice.