SQL & DATABASE MANAGEMENT

A practical guide to storing, querying, protecting, and maintaining data

LEARNSAPPEAL | ORIGINAL COURSE

Ask clear questions of well-organized data.

Learn how relational databases represent information, how SQL retrieves and changes it, and how good design supports accuracy, safety, and reliable applications.

What you need

Basic computer use is enough. Examples use familiar course and learner records. SQL dialects vary slightly; the examples use widely shared syntax, and notes call out places to check your database documentation. You can practice with a local SQLite database without creating a paid account.

Course map: Database foundations (Modules 1-4), then querying, design, reliability, and operations (Modules 5-12). Includes SQL examples, exercises with answers, a project, and glossary.

How to use this course

Read each example and predict the result before running it. Start with SELECT queries that only read data. Use a practice database for changes. When a query behaves unexpectedly, inspect the rows, keys, and relationships before adding more SQL.

By the end: describe relational tables and keys; query and combine data; summarize results; insert and update records safely; design constraints and relationships; explain normalization, transactions, and indexes; and plan basic access, backup, and maintenance.

Course map

• 1. Database basics and relational tables
• 2. Keys, relationships, and a small schema
• 3. Read data with SELECT
• 4. Filter, sort, and limit results
• 5. Combine tables with JOIN
• 6. Summaries with GROUP BY
• 7. Add, change, and remove rows
• 8. Constraints and data quality
• 9. Design and normalization
• 10. Transactions and concurrent work
• 11. Indexes and query performance
• 12. Security, backups, and capstone

DATABASE FOUNDATIONS

1. Database basics and relational tables

A database stores information so it can be found and changed consistently. A relational database organizes information in tables. Each row is a record; each column describes one attribute. A schema defines tables, columns, types, and rules. SQL is used to define, read, and modify relational data. A database management system (DBMS) manages storage, access, queries, and recovery.

Example

```
Table: courses
course_id | title          | topic
1         | Python Basics  | coding
2         | Web Foundations| web
```

Check your understanding

- What does a row represent?
- What does a schema describe?

Suggested answers

1. One record or entity instance.
2. The tables, columns, types, and rules of a database.

2. Keys, relationships, and a small schema

A primary key uniquely identifies a row. A foreign key links a row to a related table. A one-to-many relationship lets one course have many enrollments. A composite key can use more than one column. Keys prevent ambiguity and help the database enforce relationships.

Example

```
CREATE TABLE courses (
  course_id INTEGER PRIMARY KEY,
  title TEXT NOT NULL UNIQUE,
  topic TEXT NOT NULL
);
CREATE TABLE enrollments (
  learner_id INTEGER NOT NULL,
  course_id INTEGER NOT NULL,
  enrolled_on DATE NOT NULL,
  PRIMARY KEY (learner_id, course_id),
  FOREIGN KEY (course_id) REFERENCES courses(course_id)
);
```

Check your understanding

- Which key uniquely identifies a course row?
- What does the foreign key connect?

Suggested answers

1. course_id, the primary key.
2. An enrollment course_id to a course row.

3. Read data with SELECT

SELECT names the columns to return; FROM names the table. Return only needed columns instead of using * in a saved application query. Aliases make computed columns easier to read. Without ORDER BY, row order is not guaranteed. SELECT reads data and does not change stored rows.

Example

```
SELECT course_id, title, topic
FROM courses
ORDER BY title;
```

Check your understanding

- Which clause chooses the table?
- Can you assume results are sorted without ORDER BY?

Suggested answers

1. FROM.
2. No. Specify ORDER BY when the order matters.

4. Filter, sort, and limit results

WHERE filters individual rows before grouping. Use AND, OR, comparison operators, IN, BETWEEN, and LIKE to express conditions. ORDER BY sorts output; LIMIT can cap how many rows are returned, though syntax differs among databases. NULL means missing or unknown, so compare it with IS NULL rather than = NULL.

Example

```
SELECT title, topic
FROM courses
WHERE topic = 'coding'
ORDER BY title
LIMIT 10;
```

Check your understanding

- What does WHERE filter?
- How do you test for missing values?

Suggested answers

1. Individual rows before aggregation.
2. Use IS NULL or IS NOT NULL.

QUERY, DESIGN, AND MAINTAIN

5. Combine tables with JOIN

A join combines rows using a relationship between tables. INNER JOIN returns matching pairs. LEFT JOIN keeps every row from the left table and adds matching data when available. Be precise about the join condition or you may accidentally create many unrelated row combinations.

Example

```
SELECT c.title, e.learner_id, e.enrolled_on
FROM courses AS c
JOIN enrollments AS e
  ON e.course_id = c.course_id
ORDER BY c.title, e.learner_id;
```

Check your understanding

- What does INNER JOIN return?
- What extra behavior does LEFT JOIN provide?

Suggested answers

1. Rows whose join keys match in both tables.
2. It retains unmatched rows from the left table.

6. Summaries with GROUP BY

Aggregate functions such as COUNT, SUM, AVG, MIN, and MAX summarize rows. GROUP BY forms groups; HAVING filters groups after aggregation. WHERE filters rows before grouping. COUNT(*) counts rows; COUNT(column) counts non-NULL values in that column. Include useful labels and choose the right unit for each metric.

Example

```
SELECT c.topic, COUNT(*) AS enrollment_count
FROM courses AS c
JOIN enrollments AS e
  ON e.course_id = c.course_id
GROUP BY c.topic
HAVING COUNT(*) >= 5
ORDER BY enrollment_count DESC;
```

Check your understanding

- When do you use HAVING instead of WHERE?
- What can COUNT(column) omit?

Suggested answers

1. To filter groups after aggregation.
2. Rows where that column is NULL.

7. Add, change, and remove rows

INSERT adds rows, UPDATE changes selected rows, and DELETE removes selected rows. Always inspect the WHERE condition before an UPDATE or DELETE. Without a restrictive condition, an operation may affect every row. Use parameters from application code instead of building SQL by concatenating user input.

Example

```
INSERT INTO courses (course_id, title, topic)
VALUES (3, 'Data Skills', 'data');

UPDATE courses
SET topic = 'programming'
WHERE course_id = 3;
```

Check your understanding

- What can happen if UPDATE has no WHERE clause?
- Why use parameterized queries?

Suggested answers

1. Every row may be changed.
2. They separate data from SQL instructions and help prevent injection.

8. Constraints and data quality

Constraints let the database reject invalid states. NOT NULL requires a value; UNIQUE prevents duplicate values; CHECK enforces a row-level condition; PRIMARY KEY identifies rows; FOREIGN KEY enforces a reference. Constraints are valuable because multiple programs can write to the same database, and application checks alone may be bypassed.

Example

```
CREATE TABLE reviews (
  review_id INTEGER PRIMARY KEY,
  course_id INTEGER NOT NULL,
  rating INTEGER NOT NULL CHECK (rating BETWEEN 1 AND 5),
  FOREIGN KEY (course_id) REFERENCES courses(course_id)
);
```

Check your understanding

- Where is a CHECK constraint enforced?
- Why do constraints help when applications already validate input?

Suggested answers

1. By the database when a row is inserted or updated.
2. They protect data from every writer, not only one application.

9. Design and normalization

Good design stores each fact in an appropriate place and connects tables by keys. Normalization reduces avoidable duplication and update anomalies. For example, store course details once in courses and reference course_id from enrollments rather than repeating the title in every enrollment. Avoid splitting tables without a real need; design should support the questions the system asks.

Example

```
courses(course_id, title, topic)
learners(learner_id, display_name)
enrollments(learner_id, course_id, enrolled_on)

One course row; many enrollment rows refer to it.
```

Check your understanding

- What problem can repeated facts cause?
- Where should course title live?

Suggested answers

1. Conflicting updates and inconsistent values.
2. In the courses table, referenced by its key.

10. Transactions and concurrent work

A transaction groups related operations into a unit. COMMIT makes the transaction’s changes permanent; ROLLBACK discards them. ACID summarizes atomicity, consistency, isolation, and durability. Transactions help keep related updates together, such as recording a payment and its ledger entry. Databases coordinate concurrent work according to their isolation rules.

Example

```
BEGIN;
UPDATE accounts SET balance = balance - 25 WHERE id = 1;
UPDATE accounts SET balance = balance + 25 WHERE id = 2;
COMMIT;
-- Use ROLLBACK instead if a required step fails.
```

Check your understanding

- What does ROLLBACK do?
- Why group related changes in a transaction?

Suggested answers

1. Undoes uncommitted work in the current transaction.
2. To avoid leaving only part of a multi-step change applied.

11. Indexes and query performance

An index is an additional structure that can help find rows without scanning the whole table. Indexes can speed up selected reads, but consume storage and can slow writes because they also need updates. Index columns used frequently for filtering, joining, or ordering only after measuring. Use the database query plan tools to understand what the engine chose.

Example

```
CREATE INDEX idx_enrollments_course
ON enrollments(course_id);

-- Then inspect the query plan using
-- the syntax supported by your DBMS.
```

Check your understanding

- Why not index every column?
- What helps you inspect how a query runs?

Suggested answers

1. Indexes use storage and add write-maintenance work.
2. The database query plan or explain tool.

12. Security, backups, and capstone

Use individual database accounts and grant only required permissions. Keep credentials out of source code and use a secret manager or protected configuration. Encrypt connections and protect backups. Test restoring a backup; a file that has never been restored is an unverified recovery plan. Monitor failures, storage, and slow queries. Keep production changes reviewed and reversible.

Example

```
Capstone: LearnsAppeal catalog
1. Model courses, learners, and enrollments.
2. Add keys, constraints, and sample rows.
3. Query courses by topic and enrollment totals.
4. Add a safe update inside a transaction.
5. Plan permissions, backup, and restore checks.
```

Check your understanding

- Why test a backup restore?
- Name one useful capstone query.

Suggested answers

1. To verify the data can actually be recovered.
2. Courses by topic, learners per course, or recent enrollments.

Practice projects

1. Course catalog database

Design tables for courses, topics, and enrollments. Add primary and foreign keys, sample data, and queries for a topic list, recent enrollments, and counts per course.

2. Library checkout tracker

Model books, borrowers, and loans. Prevent invalid loan references, find overdue loans, and explain how you would record a checkout and return safely.

3. Data quality review

Given a table of duplicate or missing records, write SELECT queries to find issues. Propose constraints and a correction plan before making changes.

Project checklist
Are entities and relationships clear? Do keys prevent ambiguity? Are queries ordered when needed? Are writes scoped? Are user values parameterized? Can the database be backed up and restored?

SQL clause quick reference

Clause	Purpose	When it applies
SELECT / FROM	Choose output columns and source tables	Read query
WHERE	Filter rows	Before grouping
JOIN	Combine related tables	Match keys
GROUP BY	Form groups for aggregates	Summarize rows
HAVING	Filter groups	After grouping
ORDER BY	Sort returned rows	When sequence matters

Glossary

Term	Meaning
Constraint	A database rule that limits acceptable values or relationships.
DBMS	Software that manages database storage, queries, access, and recovery.
Foreign key	A column or set of columns referencing a key in another table.
Index	An auxiliary structure that can speed selected lookups.
Join	A query operation that combines rows across tables.
Normalization	Organizing data to reduce duplication and update anomalies.
Primary key	A column or group of columns that uniquely identifies a row.
Query	A request to retrieve or modify database information.
Relational database	A database organized into related tables.
SQL	A language for defining, querying, and changing relational data.
Transaction	A group of changes treated as a unit.
NULL	A marker for missing or unknown data, distinct from zero or empty text.

Further study

SQL dialects differ. Use the documentation for your database when checking types, identity columns, date handling, transaction behavior, and query-plan syntax.

• PostgreSQL. *Querying a Table.* https://www.postgresql.org/docs/current/tutorial-select.html

- PostgreSQL. *Constraints.* https://www.postgresql.org/docs/current/ddl-constraints.html
- PostgreSQL. *Indexes.* https://www.postgresql.org/docs/current/indexes.html
- SQLite. *SELECT statement.* https://www.sqlite.org/lang_select.html
- SQLite. *Transactions.* https://www.sqlite.org/lang_transaction.html

Course completion checklist

I can describe relational tables and keys; read, filter, join, and summarize data; write scoped changes; use constraints to protect data quality; explain normalization and transactions; reason about indexes; and plan access, backup, and recovery.