How to Create SQL Database: A Practical Workflow

Goal: build a working relational database with tables, keys, constraints, sample data, and a backup

End-to-end workflow: plan → install and connect → CREATE DATABASE → design tables → load data → verify → secure → back up

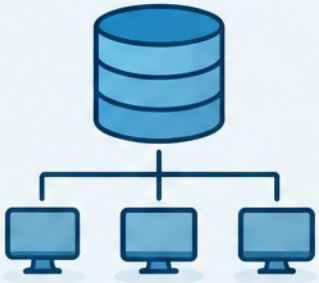
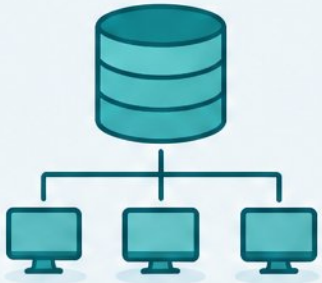
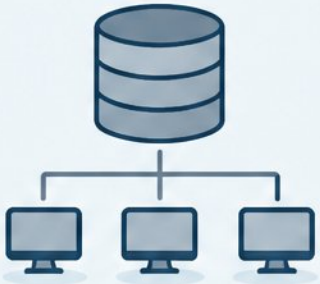
Two parallel hands-on tracks: PostgreSQL 18 (server-based) and SQLite (zero-install file database); MySQL and SQL Server differences flagged at each step

Audience: SQL beginners, developers, data analysts, and trainers who need a repeatable setup procedure

Reusable outcome: version-controlled schema.sql, seed.sql, and reset.sql that rebuilds the database from scratch



Choosing Your Engine Before You Create Anything

SQLite	PostgreSQL	MySQL	SQL Server
			
Embedded	Client-server	Client-server	Client-server

- **SQLite:** embedded, single .db file, zero configuration — perfect for first-week learning
- **PostgreSQL:** client-server, highest SQL-standard compliance — the safe 2026 server default
- **MySQL:** huge beginner ecosystem, common with WordPress/PHP hosting; SQL Server uses T-SQL
- **SQLite** serializes writes to one file — never use it behind a multi-user web app
- **Growth path:** start on SQLite, avoid its quirks, then migrate to PostgreSQL

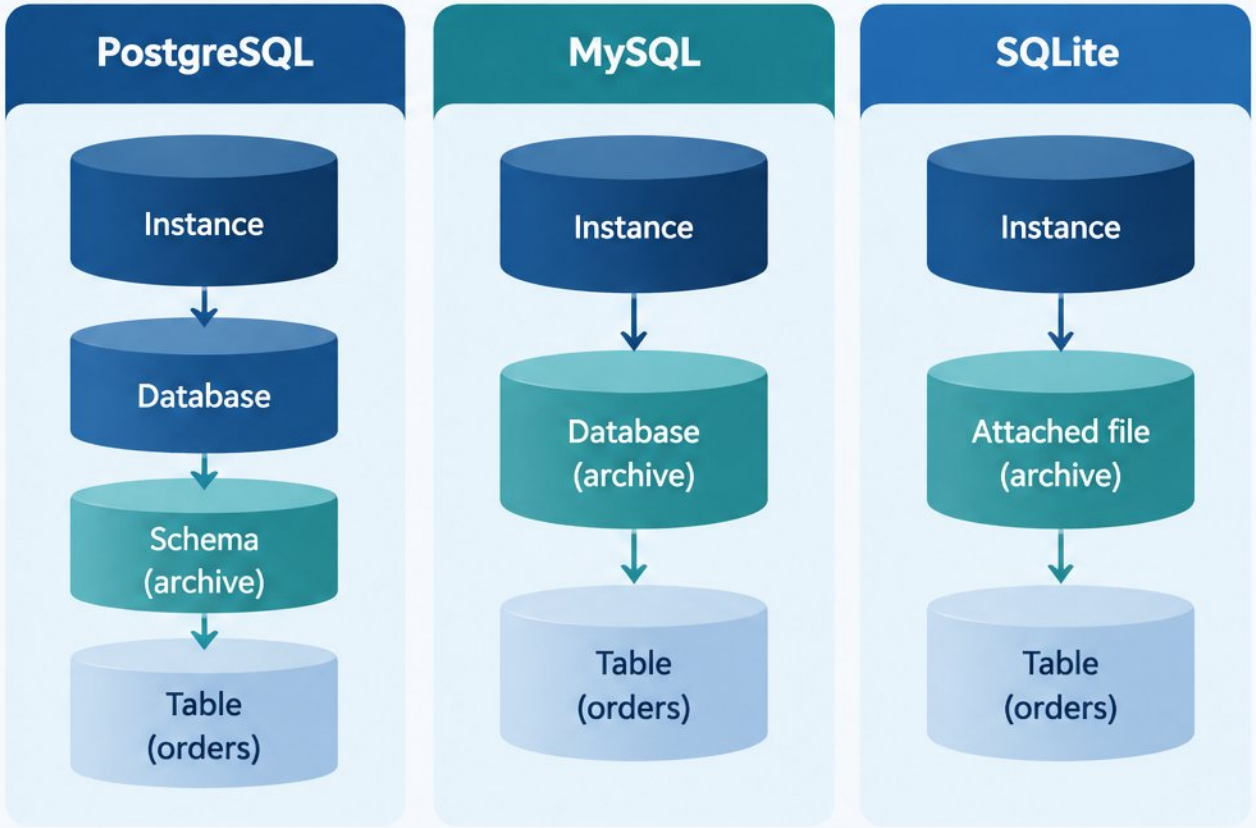


Core Concepts: Database, Schema, Table, and Instance

Same idea. Different structure. Know how each system organizes your data.

1. PostgreSQL: database → schema → table; every new database gets a 'public' schema
2. Unqualified names resolve via search_path; MySQL has no schemas — CREATE SCHEMA = CREATE DATABASE
3. MySQL cross-database joins work; PostgreSQL refuses cross-database references
4. SQLite: one database per file — 'main' plus each ATTACHed file named like a schema
5. Relational essentials: primary key, foreign key, UNIQUE, NULL semantics, referential integrity

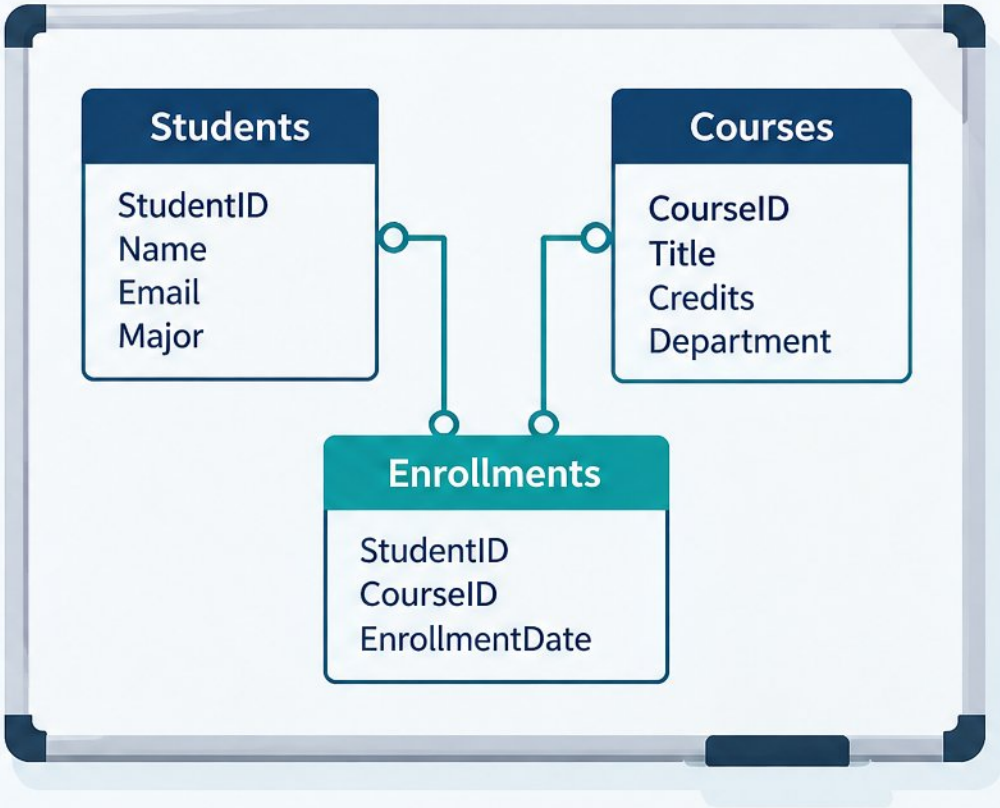
The same qualified name **"archive.orders"** means different things in each system.



DIFFERENT SYSTEMS. A SHARED FOUNDATION.

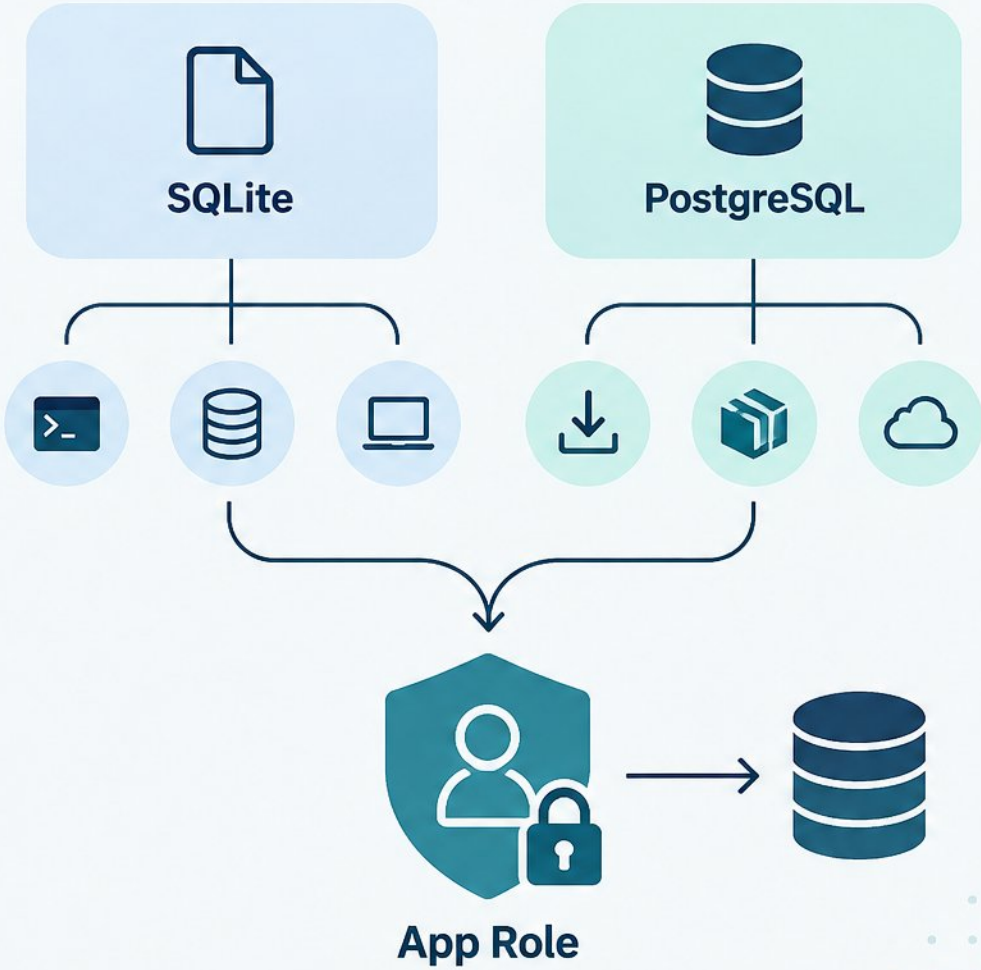
Plan the Database: From Requirements to Schema Sketch

- Turn the use case into entities, attributes, and relationships
- Model many-to-many links (students↔courses) with a junction table
- Normalize to 3NF: the key, the whole key, nothing but the key
- Draft an ER sketch and a column-level data dictionary before DDL
- Avoid comma-separated lists, missing foreign keys, mutable natural keys



Set Up the Environment and Connect Securely

- Fastest start: SQLite file via sqlite3 CLI, DB Browser, or DBeaver
- PostgreSQL: native installer, Postgres.app/EDB, Docker, or managed services
- Safe Docker start: bind port to 127.0.0.1, named volume for data
- Create a dedicated app role that owns its database
- Keep credentials in environment variables, never in scripts



CREATE DATABASE: Canonical Syntax Across Engines

Same goal. Different syntax. Know the key differences.

PostgreSQL

```
CREATE DATABASE
mydb
TEMPLATE template0
;
```

MySQL

```
CREATE {DATABASE |
SCHEMA} [IF NOT EXISTS]
mydb
CHARACTER SET utf8mb4
;
```

SQL Server

```
CREATE DATABASE mydb
ON
filegroups
LOG ON
log_files
;
```

SQLite

```
-- Database file is
-- created on open
-- Implicitly creates
-- the database file
```

- PostgreSQL: CREATE DATABASE needs superuser or CREATEDB; never inside a transaction
- Different encoding or locale? Add TEMPLATE template0 for a pristine clone
- MySQL: CREATE {DATABASE | SCHEMA} [IF NOT EXISTS] with CHARACTER SET utf8mb4
- SQL Server: filegroups and LOG ON; SQLite creates the file on open
- Write idempotent scripts with IF NOT EXISTS and a documented reset



Schemas, Namespaces, and Verification Queries

- **PostgreSQL:** CREATE SCHEMA catalog; set search_path or fully qualify each object name
- **MySQL:** USE course_catalog; a schema and a database are the same thing
- **SQLite:** only main and temp; ATTACH DATABASE adds a file under a chosen alias
- **Verify containers from the catalog:** \l and \dn, SHOW DATABASES, sys.databases
- **Check encoding and collation right** after creation: UTF-8/utf8mb4; case sensitivity varies



PostgreSQL

```
CREATE SCHEMA catalog;
set search_path or fully
qualify each object name
```

Verify from the catalog

```
\l and \dn
```

Check encoding and collation

```
UTF-8/utf8mb4; case
sensitivity varies
```

MySQL

```
USE course_catalog;
a schema and a database
are the same thing
```

Verify from the catalog

```
SHOW DATABASES
```

Check encoding and collation

```
UTF-8/utf8mb4; case
sensitivity varies
```

SQLite

```
only main and temp;
ATTACH DATABASE adds a file
under a chosen alias
```

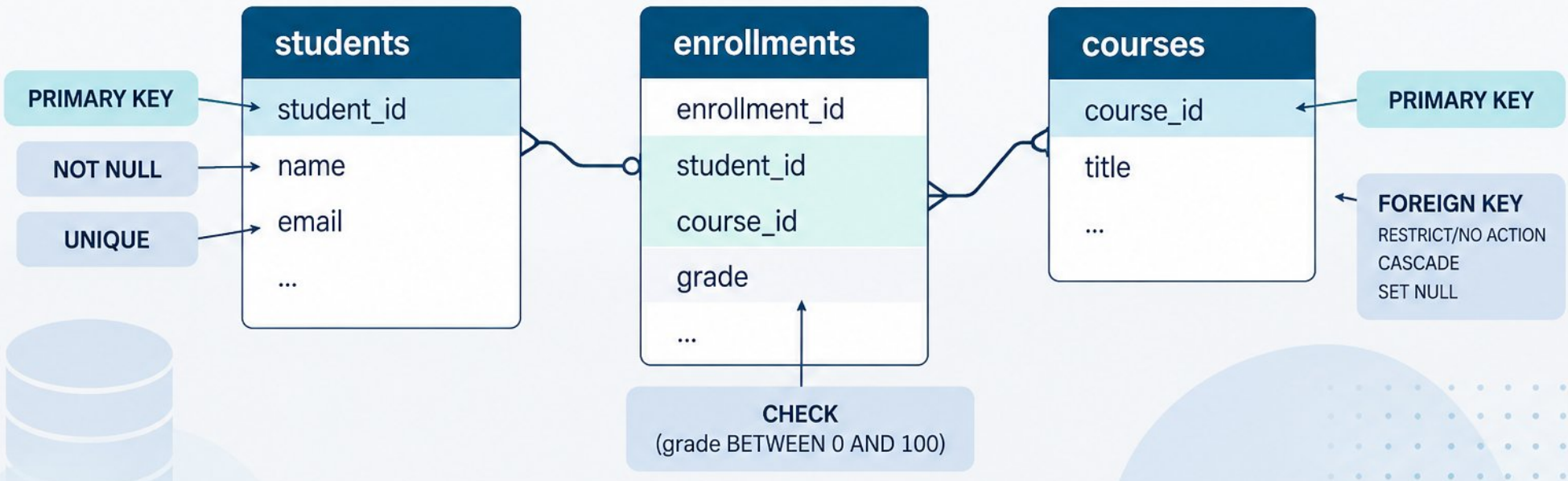
Verify from the catalog

```
sys.databases
```

Check encoding and collation

```
UTF-8/utf8mb4; case
sensitivity varies
```

CREATETABLE: Keys, Constraints, and Data Types



- Build parent tables first (students, courses), then enrollments with foreign keys
- Primary keys: PostgreSQL IDENTITY, MySQL AUTO_INCREMENT, SQLite INTEGER PRIMARY KEY
- Catch bugs with NOT NULL, UNIQUE, CHECK (grade BETWEEN 0 AND 100), named FOREIGN KEY
- Referential actions: RESTRICT/NO ACTION default, CASCADE, SET NULL
- Never CASCADE financial or audit tables; index FK columns manually in PostgreSQL
- SQLite needs PRAGMA foreign_keys = ON per connection; MySQL ignored CHECK before 8.0.16

Load and Query Sample Data

1 Prepare CSV



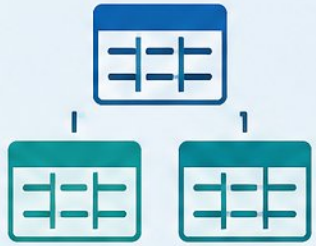
Clean CSV first:
UTF-8 without BOM,
YYYY-MM-DD dates,
one lowercase header row

2 Load into Database



Fastest CSV path:
PostgreSQL \\copy or COPY,
MySQL LOAD DATA,
SQLite .import,
SQL Server BULK INSERT

3 Tables



Batch INSERTs in one transaction or ~1,000 rows per statement; avoid a commit per row

4 Verify with SELECT



- Row counts
- Orphan rows via LEFT JOIN
- Duplicates via GROUP BY
- Prove links with a three-table JOIN: students, enrollments, courses

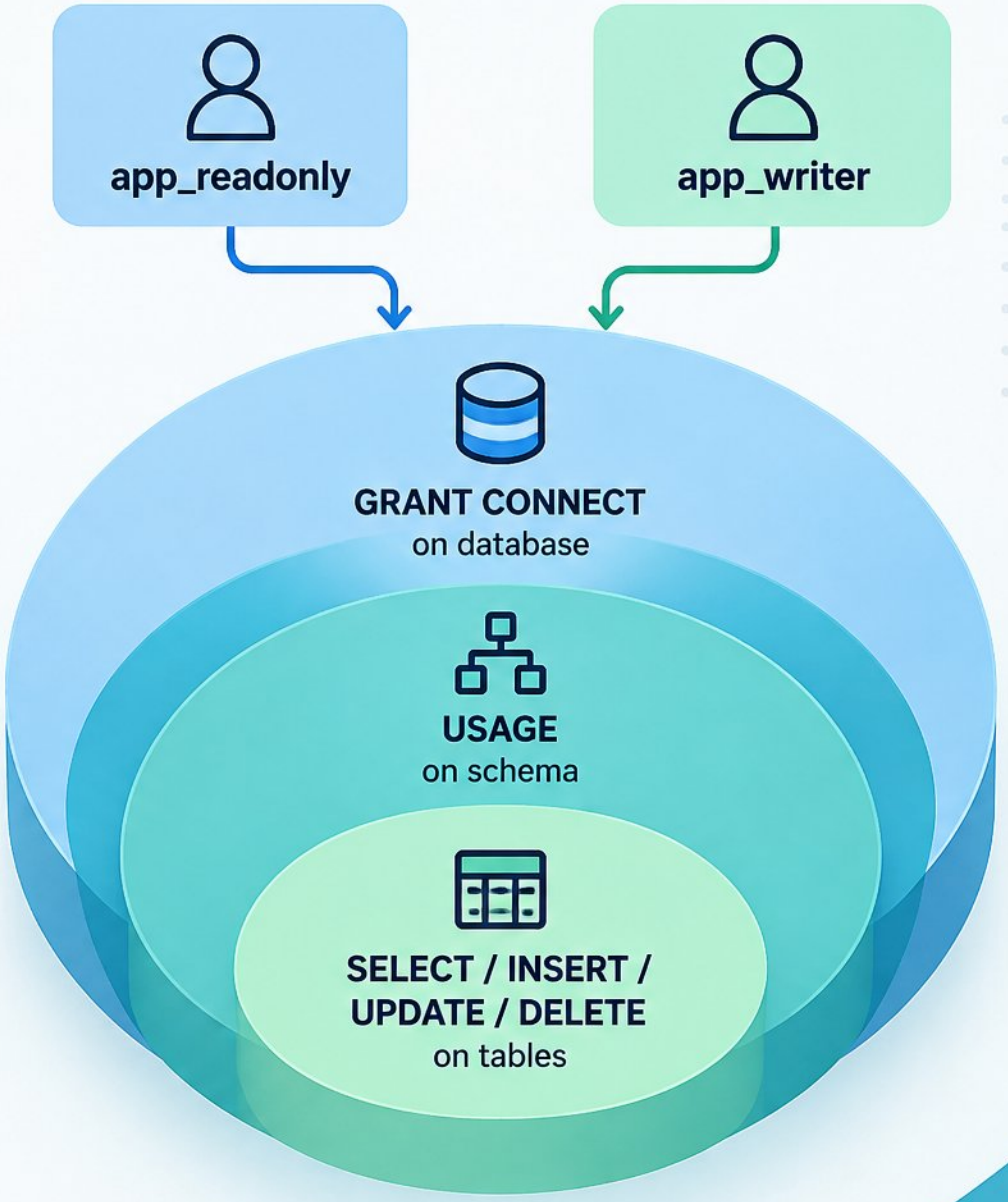
5



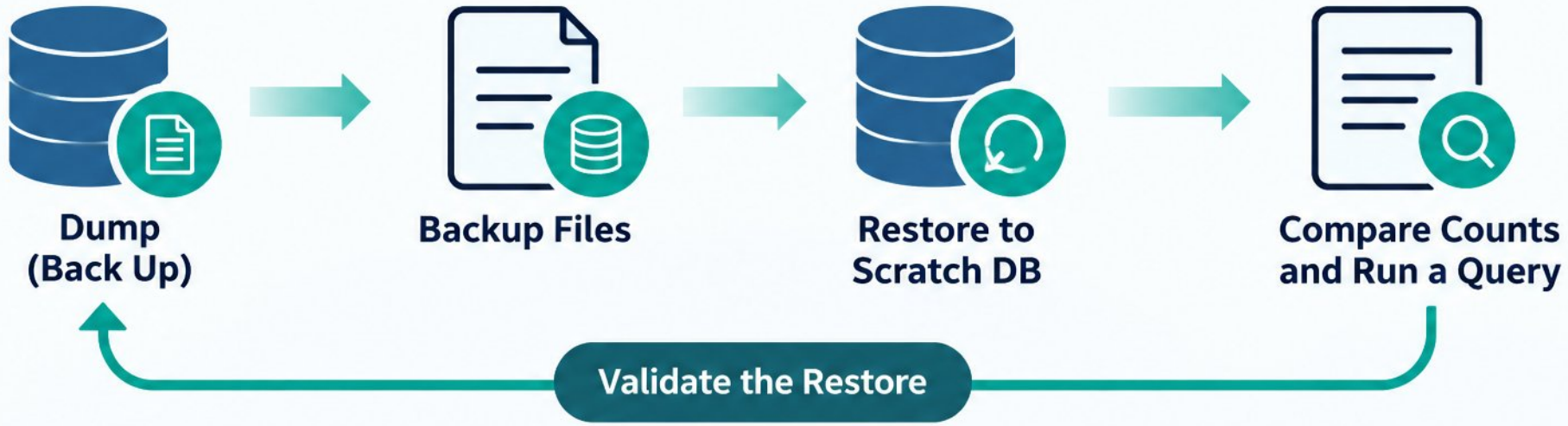
Keep schema.sql, seed.sql, reset.sql separate so the class can be replayed

Access Control and Least-Privilege Roles

- Never connect apps as postgres or root; create a dedicated login role and owning database
- PostgreSQL sequence: GRANT CONNECT on database, USAGE on schema, then table-level SELECT/INSERT/UPDATE/DELETE
- Group roles (app_readonly, app_writer) keep grants manageable and auditable
- Tighten defaults: REVOKE ALL ON SCHEMA public FROM PUBLIC; use ALTER DEFAULT PRIVILEGES for future tables
- MySQL/SQL Server: CREATE USER plus GRANT; SQLite has no user layer — protect with file permissions
- Justify every grant by workload; practice REVOKE in the lab to see its effect



Back Up, Restore, and Prove the Restore Works



- **PostgreSQL:** pg_dump -Fc, restore with pg_restore; snapshot-consistent, no writer blocking
- **MySQL:** mysqldump --single-transaction --quick --routines --triggers --events
- **SQLite:** .backup or VACUUM INTO; a live cp misses the -wal file
- **Cluster objects (roles,** tablespaces) skipped by pg_dump — use pg_dumpall --globals-only
- **Untested backup is a hope:** restore to scratch DB, compare counts, run a query
- **Apply 3-2-1:** three copies, two media, one off-site

Validate the Schema Against the Design



Structural

- Structural: every ER table, PK, FK index, and UNIQUE column exists



Behavioral

- Behavioral: break each constraint on purpose; confirm the expected error



Data

- Data: match row counts, hunt orphans and duplicates, rebuild a flat view with JOINs



Documentation

- Documentation: data dictionary matches live columns; rebuild steps written down



Fix findings in the scripts, then rebuild from an empty database.

Troubleshooting Common Failures

Permission and auth	'permission denied for schema public', peer auth failed, wrong port 5433 vs 5432, missing CREATEDB
Import errors	extra data after last column, invalid integer syntax, MySQL 'Incorrect date value', SQLite imports all TEXT
Silent integrity loss	SQLite foreign keys off after bulk load; MySQL CHECK ignored before 8.0.16
Environment issues	BOM in first CSV header, non-UTF-8 terminals, locked SQLite files, secure_file_priv (MySQL)
Recovery habit	keep reset.sql that drops and recreates everything — one command back to a clean lab



Workflow Recap and Hands-On Lab

Checklist: engine → role → database/schema → tables → CSV load → verify → grant → backup

Guided lab (~20 min): rebuild the course catalog from schema.sql, seed.sql, reset.sql

Extensions: instructors table, index on enrollments(course_id), new CHECK, read-only role

Assessment rubric scores the running database and the reasoning behind each step

Next: Flyway, Alembic, Prisma Migrate, ORMs, and production ops — after the manual workflow is fluent



PersonWise.

PersonWise • AI digital-human course platform

Scan the code or click anywhere to watch this course live, with voice Q&A

personwise.ai/t/56429add/public