**SQL LEARNING ROADMAP**

**0) Setup & tools (first thing)**

**What:** Install / run DBs locally and get the tools you’ll use.  
**Why:** You must practice on real servers and be comfortable with CLI and GUIs.  
**Do:**

* Install Docker (recommended) or local installs for MySQL, PostgreSQL, SQLite.
* Install a DB GUI: DBeaver / pgAdmin / MySQL Workbench / DB Browser for SQLite.
* Install psql, mysql CLI, and sqlite3. Learn basic shell usage.  
  **Quick exercises:** Start each DB in Docker, connect via CLI and GUI, create a demo database.  
  **Checkpoint:** You can spin up and connect to SQLite, MySQL and Postgres locally and run a SELECT 1;.

**1) Core SQL (CRUD) — foundation**

**What:** SELECT, FROM, WHERE, ORDER BY, LIMIT, INSERT, UPDATE, DELETE. Basic datatypes.  
**Why:** Every database task uses these.  
**Hands-on:** Create users, posts, comments tables and run queries: filter, sort, pagination.  
**Checkpoint:** You can write queries that return filtered lists, insert/update/delete safely, and use transactions to rollback mistakes.

**2) Data modeling & schema design basics**

**What:** CREATE TABLE, datatypes, PRIMARY KEY, FOREIGN KEY, UNIQUE, NOT NULL, normalization (1NF–3NF), ER diagrams.  
**Why:** Good schema = fast queries, fewer bugs, easier scaling.  
**Hands-on:** Design and create a normalized blog or e-commerce schema; draw ER diagram; denormalize one part for performance and justify it.  
**Checkpoint:** You can design a normalized schema from requirements and explain choices.

**3) Joins & set operations**

**What:** INNER, LEFT/RIGHT, FULL joins, CROSS JOIN, UNION, INTERSECT, EXCEPT. Join performance basics.  
**Why:** Real queries join many tables; poor joins cause bugs and slowness.  
**Hands-on:** Write queries for “posts with author info and latest comment”, “authors with zero posts”, use UNION for combined reports.  
**Checkpoint:** Confidently write all join types and know when to use each.

**4) Aggregation & GROUP BY**

**What:** GROUP BY, aggregate functions (COUNT, SUM, AVG, MIN, MAX), HAVING, rollups and grouping sets (advanced).  
**Why:** Reporting and analytics use grouped aggregates.  
**Hands-on:** Monthly active users, top 10 posts by views, groups with conditions using HAVING.  
**Checkpoint:** Build correct aggregate reports and avoid common mistakes (non-aggregated columns in select).

**5) Subqueries & Common Table Expressions (CTE)**

**What:** scalar / correlated subqueries, WITH (CTE), recursive CTEs.  
**Why:** Clean complex queries, hierarchical data, stepwise solutions.  
**Hands-on:** Use recursive CTE to output category trees or organizational chart; rewrite subqueries into CTEs for clarity.  
**Checkpoint:** You can write and optimize CTEs and understand correlated vs non-correlated subqueries.

**6) Window functions (advanced querying)**

**What:** ROW\_NUMBER(), RANK(), DENSE\_RANK(), LAG(), LEAD(), SUM() OVER (PARTITION BY ...).  
**Why:** Powerful for “top N per group”, running totals, sessionization — common in analytics.  
**Hands-on:** Top 3 salespeople per region, running revenue per day, de-dup by latest timestamp.  
**Checkpoint:** Solve complex ranking and cumulative problems elegantly with window functions.

**7) Advanced SQL features (DB specific)**

**What:**

* PostgreSQL: JSON / JSONB, arrays, PERFORM, RETURNING, FOREIGN DATA WRAPPER, full-text search, materialized views, UPSERT (ON CONFLICT).
* MySQL: JSON, EXPLAIN, INNODB specifics, REPLACE and ON DUPLICATE KEY UPDATE.
* SQLite: file-based specifics, pragmas, full-text search (FTS).  
  **Why:** Production systems often leverage DB-specific features for performance and flexibility.  
  **Hands-on:** Store JSON user metadata and query it; create materialized view for monthly aggregates.  
  **Checkpoint:** Use at least one advanced feature of each database and know tradeoffs.

**8) Indexing & query planning**

**What:** Index types (B-tree, hash, GiST, GIN), composite indexes, covering indexes, partial indexes, how the query planner uses indexes, EXPLAIN/EXPLAIN ANALYZE.  
**Why:** Indexes are the most important tool to speed queries. Misused indexes can hurt.  
**Hands-on:** Use EXPLAIN ANALYZE to compare query plans, add/remove indexes and measure effect. Create a GIN index for full-text or JSONB queries in Postgres.  
**Checkpoint:** Read query plans, add the right index to improve slow queries and know when not to index.

**9) Transactions & concurrency control**

**What:** Transactions, BEGIN/COMMIT/ROLLBACK, isolation levels, MVCC, locks, deadlocks, optimistic vs pessimistic locking.  
**Why:** Correctness under concurrent access is critical.  
**Hands-on:** Simulate concurrent updates in two sessions to observe locking and isolation phenomena; resolve deadlocks.  
**Checkpoint:** You can explain ACID, set appropriate isolation, and debug concurrency issues.

**10) Backup, restore & disaster recovery**

**What:** Logical vs physical backups, pg\_dump, pg\_basebackup, mysqldump, binary logs, WAL, point-in-time recovery (PITR), snapshot backups for SQLite.  
**Why:** You must be able to recover from data loss.  
**Hands-on:** Make a backup, wipe the DB, restore it; perform PITR (Postgres) in a controlled lab.  
**Checkpoint:** Have a tested backup & restore procedure and can recover to a point in time.

**11) Replication, high availability & scaling**

**What:** Streaming/logical replication (Postgres), MySQL replication & Group Replication, clustering, failover patterns, read replicas, partitioning/sharding basics, caching strategies (Redis).  
**Why:** Scale reads, provide redundancy and tolerate failures.  
**Hands-on:** Set up a primary + replica with Docker; test failover and read routing. Implement table partitioning and compare query performance.  
**Checkpoint:** You can configure basic replication and explain sharding vs partitioning.

**12) Security, access control & auditing**

**What:** Roles & privileges, GRANT/REVOKE, password hashing, SSL/TLS, encryption at rest, row-level security (Postgres), audit logging, least privilege.  
**Why:** Databases are high-value attack surfaces.  
**Hands-on:** Create roles for app vs analytics, enable SSL connections, implement RLS policy for multi-tenant data.  
**Checkpoint:** Enforce least privilege and set up basic auditing.

**13) Stored procedures, triggers & functions**

**What:** Writing server-side logic (PL/pgSQL, MySQL procedures), triggers for auditing/validation, pros & cons.  
**Why:** Business logic at DB level can simplify apps but can complicate maintenance.  
**Hands-on:** Build an audit trigger that logs updates; write a stored function that encapsulates a complex update.  
**Checkpoint:** Know when to use stored code vs app code and can implement safe triggers and functions.

**14) Integrations, ORMs & migrations**

**What:** ORMs (SQLAlchemy, Django ORM), migrations (Alembic, Django migrations), connection pooling, best practices to avoid N+1, parameterized queries to prevent SQL injection.  
**Why:** Most apps interact via ORMs — you must bridge SQL knowledge with app development.  
**Hands-on:** Build a simple CRUD app with an ORM, create migrations, show how to profile and fix an N+1 problem.  
**Checkpoint:** Integrate DB with an app and confidently apply migrations in staging & production.

**15) Monitoring, observability & performance tuning**

**What:** pg\_stat\_statements, MySQL performance\_schema, slow-query logs, metrics exporters, Prometheus + Grafana basics, alerting, tuning shared\_buffers/work\_mem/innodb\_buffer\_pool\_size, autovacuum.  
**Why:** Detect and fix production issues before they become outages.  
**Hands-on:** Enable slow-query logging, collect statistics, create a dashboard for query latency.  
**Checkpoint:** Have a monitoring setup that surfaces slow queries and resource hotspots.

**16) Internals & deep optimization (expert)**

**What:** Query planner internals, MVCC internals, vacuuming, buffer management, storage engines (InnoDB), write amplification, WAL internals, vectorized execution, columnar vs row stores.  
**Why:** To design high-performance systems and troubleshoot rare issues.  
**Hands-on:** Read and interpret planner costs, tweak planner statistics, simulate anti-patterns and fix them.  
**Checkpoint:** Explain the DB engine’s execution model and make changes that measurably improve throughput or latency.

**17) Data warehousing, analytics & ETL**

**What:** OLTP vs OLAP, star schema, slowly changing dimensions, partitioning for analytics, materialized views, columnar stores (e.g., ClickHouse / columnar extensions), ETL/ELT pipelines.  
**Why:** Many real systems need analytics and reporting at scale.  
**Hands-on:** Build a small data warehouse from OLTP data, implement incremental loads, create reports with precomputed aggregates.  
**Checkpoint:** Deliver an analytics solution with acceptable latency for common reports.

**18) Projects, portfolio & real-world practice**

**Project ideas:**

* Full blog or e-commerce app (transactions, inventory, ACID).
* Analytics pipeline: ingest CSVs, build star schema, create dashboards.
* Multi-tenant SaaS schema with row-level security and migrations.
* Performance challenge: optimize a slow query from a noisy dataset and document changes.  
  **Why:** Employers and real systems judge by projects.  
  **Checkpoint:** Have 2–3 polished projects in GitHub with schema, sample data, README, and benchmarking results.

**Extra: daily practice routine & study habits**

* Practice writing SQL daily (review and refactor queries).
* Keep a dev repo with schemas, seed data, and scripts.
* Use sample datasets: **Sakila / Pagila**, **Northwind**, **Chinook** for realistic practice.
* Regularly run EXPLAIN ANALYZE and document before/after performance.
* Post every interesting bug/solution to a study notebook (or GitHub Gists).

**Quick reference — what to master at each level**

* **Beginner:** CRUD, basic joins, simple schema.
* **Intermediate:** Aggregation, subqueries/CTEs, transactions, indexes basics.
* **Advanced:** Window functions, JSON/arrays, explain plans, backups, replication.
* **Expert:** Internals, tuning, clustering, sharding, monitoring, secure operations.