10■Day JDBC + SQLite Sprint — Linux Edition (8+ hrs/day)

Goal: Become production■capable on SQLite + Java (JDBC) for an inventory app (StockSync).
Scope: SQL & schema, indexes & plans, transactions, concurrency (WAL, FK, busy timeout), Flyway migrations, performance (ANALYZE, covering/partial indexes), backups & integrity, and a clean Java JDBC layer (no JPA).

Daily rhythm

Learn 1.5–2.5h → Mini■tasks 1–2h → Big project 3–4h → Review notes 0.5h.

Linux Quickstart (Debian/Ubuntu■like)

Run these commands:

```
sudo apt-get update
sudo apt-get install -y sqlite3 sqlite3-doc sqlitebrowser
sudo apt-get install -y openjdk-21-jdk maven || sudo apt-get install -y openjdk-17-jdk
maven
java -version
sqlite3 --version
mvn -v
```

Tip: If OpenJDK 21 isn't in your repos, 17 is fine for this sprint.

Day 1 — Java OOP (only what you need) + Tooling

Key ideas
Class / object / constructor / fields / methods. Use SQLite CLI + DB Browser to see and test your DB quickly.

Resources
• Java OOP (Oracle): docs.oracle.com/.../concepts
• Classes & Objects: docs.oracle.com/.../javaOO
• SQLite download: sqlite.org/download.html · CLI doc: sqlite.org/cli.html
• DB Browser: sqlitebrowser.org

Do

Install tools above. In CLI do:

```
sqlite3 ./playground.db
CREATE TABLE notes(id INTEGER PRIMARY KEY, text TEXT);
INSERT INTO notes(text) VALUES('hello');
SELECT * FROM notes; .schema
.quit
```

Do

Java mini program: make a Product class (sku, name, priceCents), constructor, and a method (priceAfterDiscount).

Mini■tasks
Explain class vs object in one sentence. Add a second class SaleLine with lineTotal().

Big project
"From Class to Rows": Java app that creates hello.db, a product table, inserts 10 products via PreparedStatement, and prints them.

Day 2 — SQL Essentials (SELECT, WHERE, JOIN, GROUP BY)

Key ideas
Master filters + joins + aggregates — 80% of real queries.

Resources
• SQL practice: sqlbolt.com
• SQLite SQL reference: sqlite.org/lang.html

Do

Create week.db with product(id, sku UNIQUE, name, price_cents, qty). Insert ~30 rows. Try:

```
SELECT * FROM product WHERE price_cents BETWEEN 1000 AND 5000;
SELECT sku, name FROM product WHERE name LIKE 'Bl%';
SELECT COUNT(*), AVG(price_cents) FROM product;
```

Mini■tasks
Add a category table and write one JOIN. Write one GROUP BY with SUM(qty).

Big project
“Catalog Queries Pack”: design 10 inventory queries (by SKU, name prefix, price ranges, low■stock list, etc.). Save to docs/day2_queries.sql with short use■case notes.

Day 3 — Query Plans & Indexing (make queries fast)

Key ideas
Use EXPLAIN QUERY PLAN to spot SCAN vs SEARCH (index). Build indexes that match WHERE/ORDER BY/JOIN patterns.

Resources
• EXPLAIN QUERY PLAN: sqlite.org/eqp.html
• Optimizer overview: sqlite.org/optoverview.html

Do

Create index and inspect plan:

```
CREATE INDEX idx_product_sku ON product(sku);
EXPLAIN QUERY PLAN SELECT * FROM product WHERE sku='SKU-123';
```

Mini■tasks
Create a name prefix index; test LIKE 'Blu%'. Try a covering index including selected columns.

Big project
“Index Lab”: For each Day■2 query, add the best index, paste the one■line EQP result, and note why the index helps. Keep before/after timings.

Day 4 — Transactions, WAL & Foreign Keys (SQLite realities)

Key ideas
Transactions group changes safely. Enable WAL for better read/write overlap. Enable foreign_keys per connection for integrity.

Resources
• Transactions: sqlite.org/lang_transaction.html
• WAL: sqlite.org/wal.html
• Foreign keys: sqlite.org/foreignkeys.html

Do

Turn on WAL (persists on DB file) and enable FKs (per connection), then build FK tables and test:

```
PRAGMA journal_mode=WAL;
PRAGMA foreign_keys=ON;
```

Mini■tasks
Wrap 3 inserts in a transaction and ROLLBACK; confirm they're gone. Repeat with COMMIT; confirm they persist.

Big project
"Integrity First": SQL script that inserts a sale + sale_lines and decrements product.qty inside one transaction; abort if qty would drop below 0.

Day 5 — JDBC Deep Dive: Connections, PreparedStatement, Errors

Key ideas
Use PreparedStatement (safe/fast), batching, and transactions from Java. Always close with try■with■resources; handle SQLException well.

Resources
• JDBC Basics: docs.oracle.com/.../jdbc/basics
• PreparedStatement Javadoc: docs.oracle.com/.../PreparedStatement.html
• Xerial SQLite JDBC: github.com/xerial/sqlite-jdbc

Do

Write a small DAO: findBySku(String), insert(Product), updateQty(long id, int delta) in a transaction. Add batch insert of 500 products.

Mini■tasks
Catch SQLException and log SQL + parameters; add timing for each operation.

Big project
"JDBC CRUD & Txn Sim": Seed 500 products; simulate 200 random sales (1–5 lines each) in transactions; log time per operation; print slowest 5 queries.

Day 6 — Data Modeling for StockSync (schema you can trust)

Key ideas
Start normalized, protect with constraints (NOT NULL, UNIQUE, CHECK). Use INTEGER cents for money. Add partial indexes when helpful.

Resources
• SQLite SQL reference (constraints): sqlite.org/lang.html
• Partial indexes: sqlite.org/partialindex.html

Do

Create core tables and indexes:

```
CREATE TABLE product(...);
CREATE TABLE sale(...);
CREATE TABLE sale_line(... FKs ...);
CREATE INDEX idx_product_sku ON product(sku);
CREATE INDEX idx_sale_line_saleid ON sale_line(sale_id);
```

Mini■tasks
Add at least one CHECK to prevent nonsense (e.g., qty >= 0). Consider a partial index (e.g., WHERE qty>0).

Big project
"Schema Review + Seed": Seed 1k products + 10k sale lines. Write 8–10 business queries (top sellers, low

stock). Save plans & timings.

Day 7 — Migrations with Flyway (version your DB)

Key ideas
Keep schema changes versioned and repeatable with Flyway (V1__init.sql, V2__indexes.sql, ...).

Resources
• Flyway + SQLite: documentation.red-gate.com/fd/sqlite

Do

Install Maven if not already. Add Flyway Maven plugin or use CLI; create V1__init.sql, V2__indexes.sql, V3__constraints.sql; run migrate and verify schema history.

Mini■tasks
Break a migration on purpose → fix with order/repair. Add a small data backfill migration.

Big project
“Migration Drill”: Add column is_active to product, create a partial index, migrate, and show before/after EQP + timing for name■prefix search.

Day 8 — Concurrency & Locking (robust under load)

Key ideas
SQLite is single■writer/many■readers. Keep writes short. Use WAL and set busy_timeout. Ensure foreign_keys ON per connection.

Resources
• Locking & concurrency: sqlite.org/lockingv3.html
• PRAGMA busy_timeout: sqlite.org/pragma.html#pragma_busy_timeout

Do

In JDBC connection init, execute these PRAGMAs after connecting:

```
PRAGMA foreign_keys=ON;
PRAGMA busy_timeout=3000;
```

Mini■tasks
Two threads: one writes small transactions; the other reads — compare failures with vs without busy_timeout.

Big project
“Resilience Harness”: Bounded retry with jitter on SQLITE_BUSY. Stress 10k small writes in bursts; confirm failures drop under WAL + busy_timeout; no lost updates.

Day 9 — Performance Lab (ANALYZE, covering/partial, VACUUM)

Key ideas
ANALYZE refreshes planner stats. Covering/partial indexes reduce latency. VACUUM INTO creates compact backups.

Resources
• ANALYZE: sqlite.org/lang_analyze.html
• Partial indexes: sqlite.org/partialindex.html
• VACUUM INTO: sqlite.org/lang_vacuum.html

Do

Maintenance commands:

```
ANALYZE;
VACUUM INTO 'backup/stocksync_YYYYMMDD.db';
```

Mini■tasks
Make one slow query 10× faster with a covering index (prove with EQP). Record before/after times.

Big project
“Perf Gate”: On 1k products + 10k sale lines, choose 3 hot queries, add indexes (incl. one partial), show >2× speedup with plans & timings in docs/day9_report.md.

Day 10 — Operations & Final Assembly (ship!)

Key ideas
Ship with a runbook: integrity, backups, restore, maintenance. Provide a tidy Java CLI for init/seed/sale/report/backup/health.

Resources
• PRAGMA integrity_check: sqlite.org/pragma.html
• VACUUM / Backup: sqlite.org/lang_vacuum.html · Backup API: sqlite.org/backup.html

Do

Create /docs/runbook.md with core health & backup steps:

```
PRAGMA integrity_check; -- expect 'ok'
ANALYZE;
VACUUM INTO 'backup/stocksync_YYYYMMDD.db';
```

Mini■tasks
Add a startup check in your app that logs if PRAGMA foreign_keys != 1.

Big project
“StockSync DB Engine — CLI”: Java CLI with commands: init (enable WAL, ensure FKs, run Flyway), seed (CSV import), sale (one transaction), find-sku, report (top-sellers/low-stock), backup, health. Deliver /db/migration, /src (JDBC), /docs (runbook, perf-results).

Notes

• Keep transactions short to avoid writer contention. Use WAL and a small busy_timeout on each connection.

• Store money in cents (INTEGER). Add CHECK constraints to guard against bad data.

• Each day, save your query plans and timing tables; these are your performance receipts.