

"Lab 8: JDBC Database Connectivity"

Persist items, members and loans in SQLite using DAOs

Lab 8 of 9

Branch: week11-jdbc

About 2 to 3 hours

Guide: LMS Guide, Step 8

Objective

Save the LMS in a **database** so that it survives restarts. You will design three tables, write three **DAO** classes with `PreparedStatement`, and connect them to the controller without touching the window.

Before you start

- Lab 7 merged. Lecture 11 notes read.
- Download `sqlite-jdbc-3.46.0.0.jar` (group `org.xerial`, from Maven Central) into a `lib/` folder. Add it to your IDE's libraries. **Do not commit the jar.**
- Run command: `java -cp "out:lib/*" lms.GuiMain` (use `;` instead of `:` on Windows).

Git: start and submit

```
git checkout main
git pull origin main
git checkout -b week11-jdbc
```

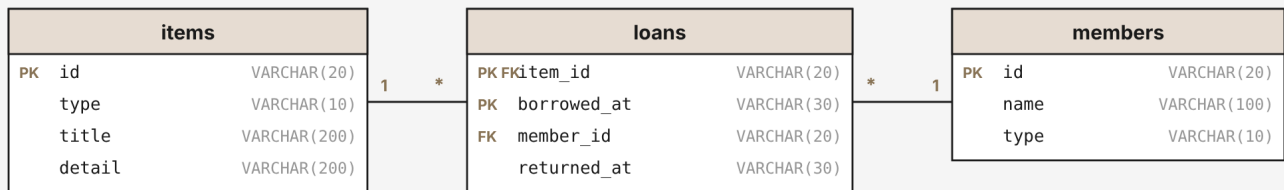
Commit small and often (see the *Git Support Guide*). When you finish:

```
git push -u origin week11-jdbc
```

Open a Pull Request titled **Week 11: JDBC Database Connectivity**, fill in the template, and request a review from your lecturer.

Design first (10 minutes)

Draw the ER diagram (`docs/er-week11.png`):



Reading the diagram

One item can appear in MANY loan rows over time (borrowed, returned, borrowed again).

One member can have MANY loans. FK = foreign key: the value must exist in the other table.

A loan with returned_at = NULL is still on loan - no separate 'is borrowed' column to keep in sync.

items.detail holds the author (Book), minutes (DVD) or issue number (Magazine); items.type says which.

PK = primary key (loans uses a composite key: item_id + borrowed_at)

Question to answer in your PR: *what does a loan row with `returned_at = NULL` mean, and why is that better than an `isBorrowed` column?*

Tasks

Task 1: `lms.exception.DataAccessException`

Unchecked, extends `RuntimeException`, constructor (String message, Throwable cause).

Task 2: `lms.dao.Database`

`getConnection()` (URL `jdbc:sqlite:lms.db`, overridable with `-Dlms.db.url=...`; for SQLite run `PRAGMA foreign_keys = ON` on each connection) and `init()` that runs three `CREATE TABLE IF NOT EXISTS` statements: `items(id, type, title, detail)`, `members(id, name, type)`, `loans(item_id, member_id, borrowed_at, returned_at)` with the primary and foreign keys from the diagram.

Task 3: three DAOs

- `ItemDAO`: `insert(LibraryItem)` and `findAll()`. `type` is the class name; `detail` is author / minutes / issue. Rebuild the right subclass from the `type` column with a `switch`.
- `MemberDAO`: `insert(Member)` and `findAll()`; `type` is `STUDENT` or `STAFF`.
- `LoanDAO`: `recordLoan(itemId, memberId)`, `recordReturn(itemId)` (`UPDATE ... WHERE item_id = ? AND returned_at IS NULL`), `findOpenLoans()`, `history()` as a **three-table JOIN** returning rows of borrowed time, item title, member name, return time or (on loan).

Use **only** `PreparedStatement` and try-with-resources. Catch `SQLException` and throw `DataAccessException`.

Task 4: test the DAOs from the console first

Write `lms.DaoDemo` that inserts an item and a member, reads them back, records a loan and a return, prints the history, and tries a loan for a member that does not exist (it must be rejected by the foreign key).

Task 5: connect to the controller

- Constructor now (Library, ItemDAO, MemberDAO, LoanDAO).

- `loadFromDatabase()` : add all items and members to the library, then **replay** open loans through `library.borrowItem(...)`.
- `addItem` : *validate by constructing, save to DB, then add to memory.*
- `borrow` : library first, then `recordLoan` ; if the DB call fails, undo the in-memory loan and rethrow. `giveBack` : library, then `recordReturn` .
- New `addMember(type, id, name)` and `loanHistory()` .

Task 6: GUI additions

A **New Member...** button (a form in `JOptionPane.showConfirmDialog`) and a **Loan History** button (a `JTable` in a dialog). Catch `DataAccessException` in handlers and show it.

Task 7: start-up

`GuiMain`: `Database.init()`, seed demo data **only if the tables are empty**, create the DAOs and controller, `controller.loadFromDatabase()`, show the window.

Expected output

`DaoDemo` should end like this (timestamps differ):

```
--- open loans after return: 0
--- history (JOIN) ---
2026-10-13 00:33:25 | Head First Java | Amal Fernando | 2026-10-13 00:33:25
--- foreign key check: loan for a member that does not exist ---
Rejected: Could not record the loan of B100
```

Then borrow some items, press **Loan History**, close the app, reopen it. The loans must still be there:

The screenshot shows the 'Library Management System' GUI. A search results dialog is open, displaying a table of borrowed items. The dialog has an 'OK' button. The main window shows a search bar, a 'Search' button, and a 'Library' section with buttons for 'New Member...', 'Loan Report', and 'Loan History'. The status bar at the bottom shows 'G001 borrowed by M002' and a 'Member:' dropdown set to 'M002 - Mr. Kasun Silva' with 'Borrow' and 'Return' buttons.

ID	Title	Type	Detail	Status
B001	Clean Code	Book	Robert C. Martin	On loan: Nadeesha Perera
B002	Software Weekly	Book	Mr. Kasun Silva	available
B003	Clean Code	Book	Nadeesha Perera	available
B004	Refactoring	Book	Mr. Kasun Silva	on loan: Mr. Kasun Silva

Library Management System

New item

Type: Book

ID:

Title:

Detail:

Add Item

Search:
Search

ID	Title	Type	Detail	Status
B001	Clean Code	Book	Robert C. Martin	On loan: Nadeesha Perera
B002	Refactoring	Book	Martin Fowler	Available
D001	Introduction to Algorithms	DVD	120 min	Available
G001	Software Weekly	Magazine	Issue 42	On loan: Mr. Kasun Silva

Library

New Member...
Loan Report
Loan History

Ready

Member: M001 - Nadeesha Perera
Borrow
Return

Checkpoints

Tick each before moving on. If a box stays empty, ask for help *now*; later steps build on it.

- ☐ `lms.db` appears after the first run and is **not** in `git status` (check `.gitignore`)
- ☐ Items, members and open loans survive a restart
- ☐ Every query uses `?` placeholders, with no string concatenation of values
- ☐ The foreign key really rejects a bad loan
- ☐ MainFrame still has **no** SQL and no `java.sql` import
- ☐ ER diagram committed

Experiments: break it on purpose

Learning happens when you predict, run, and compare. Write your **prediction** first, then paste the real compiler message or output into your PR description.

- Delete `lms.db` and restart. What do you see, and why is seeding "only if empty" important?
- Replace one `?` query by string concatenation. Then enter the name `' OR '1'='1` and describe what could go wrong (do **not** keep this code).
- Remove the jar from the classpath. Paste the exact error.
- Call `loanDAO.recordLoan("B001", "M001")` twice within the same second. Which constraint fires and why is that sensible?

Stretch tasks (optional, no marks lost if skipped)

- Add `ItemDAO.delete(String id)` and a Delete button; what should happen to an item that has loan rows?
- Add `LoanDAO.overdue(int days)`.
- Write `MemberDAO.findById`, then use it in `loadFromDatabase`.

- Export history to CSV.

Common pitfalls

Symptom	Fix
No suitable driver found for <code>jdbc:sqlite:lms.db</code>	The jar is not on the classpath
<code>no such table: items</code>	<code>Database.init()</code> not called before the first query
FOREIGN KEY constraint failed	Insert the item and member before the loan
<code>[SQLITE_BUSY] database is locked</code>	Another process (a DB viewer or a second copy) holds the file
Seeds are inserted on every start	The emptiness check ran after inserting
<code>ClassNotFoundException</code> on Windows	Classpath separator is <code>;</code> not <code>:</code>

What to submit

- ☐ PR **Week 11: JDBC Database Connectivity** from branch `week11-jdbc`
- ☐ `docs/er-week11.png`, screenshots of history and of the restart
- ☐ PR description: answer on `NULL` in `returned_at`; experiment results
- ☐ **No** `lms.db` and **no** jar in the commit

Marking guide

Your PR is marked on four criteria. Each is scored 4 (excellent) to 1 (needs work) and weighted as shown.

Criterion (weight)	4	3	2	1
Works (40%)	All tasks run, expected output matches	Minor differences	Partly works	Does not compile or run
OOP design (30%)	Concepts of this week used correctly and deliberately	Mostly correct	Concepts present but misused	Concepts missing
Code quality (15%)	Clear names, small methods, no dead code, comments explain <i>why</i>	Mostly clean	Messy in places	Hard to read
Git and PR (15%)	Exact branch/PR name, small clear commits, full PR description, UML in <code>docs/</code>	One slip	Several slips	One giant commit or direct to <code>main</code>

Check Before you open the PR, read your own diff on GitHub as if you were the reviewer. Most deductions are things you would have spotted yourself.