

JDBC: Database Connectivity

SQLite, tables and keys, PreparedStatement, the DAO pattern, saving the LMS

Lecture + Lab 8

Branch: week11-jdbc

Learning outcome 5

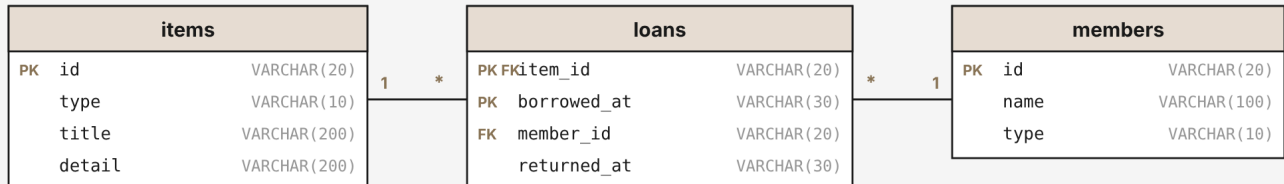
What you will be able to do after today

- Design three related tables with **primary and foreign keys**.
- Connect from Java with **JDBC** and run `INSERT`, `SELECT` and `UPDATE`.
- Use **PreparedStatement** (and say why).
- Organise database code with the **DAO pattern** and plug it into the MVC controller.

1. Why a database?

Close the LMS and everything vanishes. A database stores data **permanently** and answers questions fast. We use **SQLite**: the whole database is one file, `lms.db`, with no server to install.

2. Designing the tables



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`)

Table	Holds	Key
<code>items</code>	Every Book, DVD, Magazine. <code>type</code> says which; <code>detail</code> is author / minutes / issue	<code>id</code>
<code>members</code>	Students and staff	<code>id</code>
<code>loans</code>	One row per loan. <code>returned_at</code> is <code>NULL</code> while the item is still out	(<code>item_id</code> , <code>borrowed_at</code>)

Foreign keys make the database refuse nonsense: you cannot create a loan for a member that does not exist.

```
CREATE TABLE IF NOT EXISTS loans (
  item_id    VARCHAR(20) NOT NULL,
  member_id  VARCHAR(20) NOT NULL,
  borrowed_at VARCHAR(30) NOT NULL,
  returned_at VARCHAR(30),
  PRIMARY KEY (item_id, borrowed_at),
  FOREIGN KEY (item_id) REFERENCES items(id),
  FOREIGN KEY (member_id) REFERENCES members(id)
);
```

3. Your first JDBC program

Add the driver. Download `sqlite-jdbc-3.46.0.0.jar` from Maven Central (group `org.xerial`) into a `lib/` folder and add it to the classpath (IDE: *Add library/JAR*). With Maven it is a `<dependency>`.

```
try (Connection con = DriverManager.getConnection("jdbc:sqlite:lms.db");
     PreparedStatement ps = con.prepareStatement("SELECT id, title FROM items WHERE type = ?")) {
  ps.setString(1, "Book");
  try (ResultSet rs = ps.executeQuery()) {
    while (rs.next()) {
      System.out.println(rs.getString("id") + " " + rs.getString("title"));
    }
  }
} catch (SQLException e) {
  throw new DataAccessException("Could not read items", e);
}
```

Step	Class	Notes
Connect	<code>Connection</code>	URL <code>jdbc:sqlite:lms.db</code> ; creates the file if missing
Prepare	<code>PreparedStatement</code>	SQL with <code>?</code> placeholders
Fill	<code>ps.setString(1, ...)</code>	Placeholders are numbered from 1
Run	<code>executeUpdate()</code> / <code>executeQuery()</code>	Changes vs reads
Read	<code>ResultSet.next()</code> , <code>getString(...)</code>	Loop over rows
Close	<code>try-with-resources</code>	Closes automatically, even on errors

Why `PreparedStatement`?

```
// DANGEROUS: SQL injection
String sql = "SELECT * FROM members WHERE name = '" + name + "'";
// name = ' OR '1'='1 -> returns EVERY member
```

With `?` placeholders, the value is sent **separately from the SQL**, so it can never be interpreted as SQL.
Never build SQL by joining strings.

4. The DAO pattern

A **Data Access Object** hides all SQL for one kind of thing behind ordinary Java methods.

```

public class ItemDAO {
    public void insert(LibraryItem item) { ... }           // object -> row
    public List<LibraryItem> findAll() { ... }             // rows -> objects
}
public class MemberDAO { ... }
public class LoanDAO {
    public void recordLoan(String itemId, String memberId) { ... }
    public void recordReturn(String itemId) { ... }
    public List<OpenLoan> findOpenLoans() { ... }
    public List<String[]> history() { ... }               // a three-table JOIN
}

```

The `type` column decides which subclass a row becomes. This is polymorphism meeting persistence:

```

private LibraryItem toItem(String id, String type, String title, String detail) {
    return switch (type) {
        case "Book" -> new Book(id, title, detail);
        case "DVD" -> new DVD(id, title, Integer.parseInt(detail));
        case "Magazine" -> new Magazine(id, title, Integer.parseInt(detail));
        default -> throw new DataAccessException("Unknown item type in database: " + type, null);
    };
}

```

Exception translation. `SQLException` is checked and database-specific. The DAO catches it and throws our unchecked `DataAccessException`, so the controller and GUI never import `java.sql`.

5. Plugging into MVC

The controller keeps the model **and** the database in step, in a careful order:

```

public void addItem(String type, String id, String title, String detail) {
    LibraryItem item = build(type, id, title, detail); // 1. validate by constructing
    itemDAO.insert(item);                             // 2. save
    library.addItem(item);                             // 3. only then change memory
}

public void borrow(String itemId, String memberId) throws ItemNotAvailableException {
    library.borrowItem(itemId, memberId);              // rules first
    try { loanDAO.recordLoan(itemId, memberId); }      // then persist
    catch (RuntimeException e) { library.returnItem(itemId); throw e; } // undo on failure
}

```

At start-up, `loadFromDatabase()` rebuilds the model, **replaying open loans** through the model's own rules.

6. Proof it persists

Borrow a few items, press **Loan History** (the JOIN), close the application, start it again:

Library Management System

Search:

New item

Type: Book

ID:

Title:

Detail:

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

G001 borrowed by M002

Member: M002 - Mr. Kasun Silva

Library Management System

Search:

New item

Type: Book

ID:

Title:

Detail:

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

Ready

Member: M001 - Nadeesha Perera

7. Debugging JDBC

Error	Usual cause
No suitable driver	The driver jar is not on the classpath
no such table: items	<code>Database.init()</code> was never called
FOREIGN KEY constraint failed	Inserting a loan for an id that is not in <code>items</code> / <code>members</code>
UNIQUE constraint failed	Inserting the same primary key twice
database is locked	A second program still has <code>lms.db</code> open

Test DAOs with a small console program (DaoDemo) **before** touching the GUI.

8. Lab 8 preview (Week 11)

Create `lms.dao` with `Database`, `ItemDAO`, `MemberDAO`, `LoanDAO`; wire them into the controller; seed the database on first run; add **New Member** and **Loan History**. Commit the ER diagram. **Do not commit** `lms.db`. Branch `week11-jdbc`.

9. Self-check

1. What stops a loan from referring to a member who does not exist?
2. Rewrite `"... WHERE name = '" + name + "'"` safely.
3. Why does the DAO throw `DataAccessException` instead of letting `SQLException` escape?
4. Why does `borrow()` undo the in-memory loan if the database call fails?