

Library Management System

Academic project file | Proposed desktop prototype

ASSIGNEASE STUDIO

Typed. Colorful. Clearly organised. A practice document for demonstrating assignment layout and writing.

Submission details	To be completed
Student name	
Course / semester	
Roll number	
College / department	
Submitted to	
Academic session	

Practice sample - not an assessed submission. Replace blank details and follow the college's own requirements before use.

Certificate

Template for college completion

This page is a certificate template for the proposed project titled "Library Management System". Student, guide and institution details must be filled only after the project has been reviewed under the college's requirements.

Field	Details
Student name	_____
Roll number	_____
Course / semester	_____
Department / institution	_____
Project guide	_____
Academic session	_____

CERTIFICATION WORDING FOR AUTHORISED USE

This is to certify that the above-named student has submitted the project titled "Library Management System" under the supervision of the undersigned, subject to the institution's review and assessment requirements.

Guide signature: _____

Head of department: _____

Date: _____

Institution seal: _____

No signature, approval or completed implementation is claimed by this practice sample.

Acknowledgement

Preparing this project design helped me look at an ordinary library counter as a set of connected decisions. A book is not simply "available" or "unavailable"; each physical copy has its own history, and each issue or return changes what the next reader can borrow.

I am thankful for the learning resources that explain database design, interface development and testing in a clear way. The Python and SQLite documentation was useful for understanding the proposed technical approach. Their links are included in the bibliography.

The names of the project guide, department and institution have been left blank because this is a practice file. In a real submission, this acknowledgement should be edited to recognise the people who actually supported the work, rather than adding names or help that did not exist.

PROJECT STATUS

Design and documentation sample. The forms are illustrative mockups, and the test cases are planned tests. The file does not claim a completed application or measured performance.

Index

Section	Page
1. Introduction	4
2. Objectives and scope	4
3. Existing and proposed system	5
4. Requirements	5
5. Feasibility and risks	6
6. SDLC methodology	7
7. Architecture and modules	8
8. Use cases and workflows	9
9. Entity relationship diagram	10
10. Data flow diagrams	11
11. Database tables and constraints	12
12. Issue and return logic	13
13. Interface mockups	15
14. Validation and error handling	16
15. Testing plan	17
16. Deployment and maintenance	18
17. Limitations and future scope	19
18. Conclusion and bibliography	20

1. Introduction

A library management system keeps track of books, members and borrowing transactions in one organised record. The proposed system in this project is meant for a small college library. It would support the librarian's daily work rather than replace the librarian's judgment about damaged books, exceptional borrowing requests or disputes.

The project starts with a familiar situation: a student asks for a book, the librarian checks the shelf or register, and an issue entry is written by hand. This can work in a small collection, but it becomes harder to answer simple questions quickly. Which copy is still with a student? Is a requested title available? Which loans are overdue? Are two records describing the same book?

The design separates a book title from a physical copy. A title such as a database textbook may have several copies. Each copy receives an accession number and can be borrowed independently. This distinction is the basis of a dependable circulation record. Counting titles alone would give the wrong answer when one copy is on the shelf and another is on loan.

The proposed prototype uses a desktop interface built with Python and Tkinter and a local SQLite database. This is a design choice for the assignment, not a statement that every library should use this stack. It keeps the project small enough to understand while still allowing forms, validation, relational tables and transaction handling to be explained.

This file covers the problem, requirements, development method, data model, diagrams, screen designs, rules and planned tests. It deliberately avoids claiming that a working system has already been installed. That distinction matters: a clear design is valuable, but it is not evidence that the design has passed real testing.

WORKING ASSUMPTION

One small college library, one circulation desk and a local database. All member IDs, copy numbers and examples in this file are fictional demonstration data.

2. Objectives and scope

The main objective is to keep a reliable record of who has borrowed each physical copy and when it should be returned. A second objective is to make routine enquiries easier: searching the catalogue, checking availability and listing overdue loans should not require reading several handwritten pages.

The system should also reduce avoidable mistakes. It should reject an issue request for an inactive member, stop the same copy from being issued twice and prevent a return from being recorded against the wrong loan. These checks are more important than adding many decorative buttons to the interface.

The first version is intentionally limited. It includes catalogue maintenance, member registration, issue and return, reservation recording and basic reports. It does not include online payments, inter-library borrowing, RFID hardware or a public website. Those features would introduce new permissions, security concerns and testing requirements.

A good project objective must be testable. "Make the library modern" is too broad. "A copy with an open loan cannot receive another open loan" is specific and can be checked. The objectives below use that approach. Performance targets must be measured during implementation rather than presented as achievements in a design file.

Objective	Acceptance criterion
Track physical copies	Every copy has one unique accession number

Protect circulation records	No copy has two open loans
Search the catalogue	Search by title, author or accession number
Manage members	Inactive members cannot borrow
Report overdue items	Open loans past due date appear correctly
Preserve records	Historical loans remain after a return
Recover from failure	A backup can be restored in a planned test

3. Existing and proposed system

For this project, the assumed existing system is a manual register with separate notes for members and stock. This is a model of a common small-library workflow, not an observation of a named institution. Manual records are understandable and require little equipment, but consistency depends on the person updating them each time.

A register can record a transaction correctly and still be difficult to search. If the student's name is written differently on two occasions, borrowing history can become fragmented. If a book is described only by its title, the librarian may not know which physical copy was issued. Correcting an old entry may also obscure what changed and why.

The proposed system replaces these scattered entries with linked records. Member IDs identify borrowers. Book IDs identify bibliographic records. Copy IDs identify physical items. Loan IDs identify issue events. The librarian works through forms, while the database enforces relationships behind them.

The proposed approach has costs too. It depends on electricity, a maintained device and a backup process. A badly designed application could lose more data at once than a damaged register. The recommendation is therefore to build carefully, test recovery and keep a short transition plan rather than assume digital records are automatically safer.

Area	Assumed manual process	Proposed process
Availability	Check shelf and register	Search copies and open loans
Member history	Find entries by name	Read transactions by member ID
Duplicate copies	Repeated title entries	One title with multiple copy records
Overdue report	Compare dates manually	Filter open loans by due date
Correction	Overwrite or annotate register	Restricted edit with reason
Recovery	Protect physical registers	Verified backup and restore

4. Requirements

Requirements describe what the system should do and the conditions under which it should work. They are separated into functional requirements, which concern actions, and non-functional requirements, which concern quality. This keeps a working button from being mistaken for a complete feature.

The librarian is the main user of the prototype. A member can make a borrowing request at the desk, but does not directly edit records in this version. A read-only search interface could be added later.

Administrative actions such as changing policies or restoring backups need a narrower role than ordinary searching.

Each requirement should have a clear failure path. If a copy number is not found, the interface should show a useful message and leave the database unchanged. If an operation fails halfway through, the system should roll it back rather than leave a loan without the matching copy state.

ID	Functional requirement	Priority
FR01	Create and edit a book record	Must
FR02	Register a physical copy with unique accession number	Must
FR03	Create a member and set active/inactive status	Must
FR04	Search titles, authors and copies	Must
FR05	Validate eligibility before issue	Must
FR06	Record return without deleting loan history	Must
FR07	List open and overdue loans	Must
FR08	Record and cancel a title reservation	Should
FR09	Export a librarian-selected report	Should
FR10	Record administrative changes with a reason	Should

Quality requirement	Design response	How to verify
Usability	Clear labels, keyboard navigation	Observe a planned desk-workflow trial
Integrity	Foreign keys and unique checks	Try invalid and duplicate records
Reliability	Atomic circulation transactions	Simulate an interrupted update
Privacy	Minimum member details and restricted reports	Review fields and role permissions
Recovery	Local backup with restore instructions	Restore into a test database
Maintainability	Separate UI, rules and data access	Review module dependencies

5. Feasibility and risks

The technical approach is feasible for a learning prototype because the chosen tools support a desktop interface and a local relational database. Python's documentation describes Tkinter as the standard Python interface to the Tcl/Tk GUI toolkit. SQLite's documentation describes how foreign keys can enforce relationships between records. The project still needs actual implementation to establish compatibility on the target computer.

Operational feasibility depends on the librarian's workflow. A technically correct form can fail in practice if issuing a book requires too many steps or error messages hide the next action. A short trial with realistic requests should be part of development. Staff should also know what to do when the computer is unavailable.

This sample makes no monetary estimate because equipment, training and support costs are not known. A real proposal should separate one-time setup from recurring maintenance and should count staff time. Saying that open-source tools make the project "free" would ignore those costs.

The main risks are duplicate migration records, accidental deletion, inadequate backup and incorrect borrowing rules. Each risk needs an owner and a control. The software cannot settle an institutional policy that has not been agreed, such as whether overdue members may renew or how damaged books are handled.

Risk	Control	Remaining concern
Duplicate accession numbers	Unique database constraint	Old register needs review
Loss of local computer	Scheduled backup and tested restore	Backup can also be damaged
Wrong issue policy	Configurable agreed rules	Policy approval still needed
Unauthorised member-data export	Restricted reports and access	Copies of exports need protection
Mid-operation failure	Transaction rollback	Recovery must be tested
Damaged or missing book dispute	Librarian review and notes	Cannot be decided by status alone

6. SDLC methodology

The project follows an iterative software development life cycle. Requirements are written first, then a small working slice is designed, implemented and tested before the next feature is added. This is preferable to building every screen first and discovering at the end that the database cannot represent the library's actual rules.

During requirements analysis, the developer would confirm borrowing periods, maximum open loans, membership status rules and the treatment of damaged copies with the library. This sample uses a 14-day loan period and a limit of three open loans only as demonstration policies. They must not be treated as the rules of a real college.

During design, entities and relationships are settled before form layouts. A circulation transaction should refer to a member and a physical copy, not to free-text names. The ER diagram checks those relationships. The DFD then checks where information enters, is processed, is stored and is returned to the user.

Implementation would start with catalogue and member records, then add issue and return. Reports and reservations come after the core circulation rules are stable. This order gives early tests something meaningful to check. A visually complete dashboard with unreliable loans would not be an acceptable first milestone.

Testing happens at each stage. Small rule tests check dates and eligibility. Integration tests check whether the database and business rules work together. User acceptance tests check whether a librarian can complete a realistic task and understand a failure message. A feature is finished only when both normal and failure paths have been checked.

Deployment includes installing dependencies, creating the database, checking roles and restoring a sample backup. Maintenance then covers fixing defects, updating policy settings and reviewing backups. A project does not stop needing care once the first issue receipt is printed.

Phase	Output	Review question
Analysis	Requirements and agreed rules	Are policy assumptions confirmed?
Design	Schema, diagrams and forms	Do all records and flows agree?

Implementation	Small working modules	Does each module enforce its rules?
Testing	Executed tests and defect log	Do failed actions leave data unchanged?
Deployment	Installation and recovery guide	Can the desk operate safely?
Maintenance	Change and backup records	Can updates be traced and reversed?

SUGGESTED DEVELOPMENT SEQUENCE

Iteration 1: catalogue + members. Iteration 2: issue + return. Iteration 3: reservations + reports. Iteration 4: role checks + backup trial + usability review. This is a plan, not a record of completed work.

7. Architecture and modules

The proposed architecture has three layers. The presentation layer contains Tkinter forms. The service layer contains borrowing rules and validation. The data layer performs database operations. Keeping these layers separate prevents a form button from becoming the only place where an important rule is enforced.

For example, the issue form collects a member ID and copy accession number. It should not directly insert a loan merely because both boxes contain text. The service layer checks membership, open-loan count, copy status and any reservation restriction. Only then does the data layer commit the new loan and related state changes together.

A catalogue module manages book titles and physical copies. A member module manages active borrowers. A circulation module handles issue, return and renewal rules. A reservation module keeps a queue at title level. A reporting module produces read-only summaries. An administration module controls settings and backup operations.

The prototype assumes one desk computer. If multiple circulation desks are introduced, concurrency and shared access must be reconsidered. A local database file copied casually between computers would create conflicting records. A networked version needs an intentional server, permissions and transaction strategy, not just a shared folder.

Module	Inputs	Outputs
Catalogue	Title details, accession number	Search results, copy record
Members	Member details and status	Member record, eligibility status
Circulation	Member ID, copy ID, action	Loan record, return confirmation
Reservations	Member ID, title ID	Queue entry, cancellation status
Reports	Date range and report type	Open-loan or overdue list
Administration	Authorised setting or backup request	Change record or backup result

The database should be opened through one small access layer rather than separately in every form. That layer enables foreign-key checks for each connection and returns clear results or errors. The service layer should use parameterised queries so that user-entered text is treated as data. Validation in the interface is helpful, but it does not replace constraints and checks in the lower layers.

The system can present an action result as a simple structure: success flag, message and optional record ID. A form uses this result to update the screen. This keeps internal database error details away from ordinary users while preserving enough information in an appropriate diagnostic log for maintenance.

8. Use cases and workflows

A use case describes a goal from the user's point of view. The main actor is the librarian, who acts on a member's request. The design should keep the member informed of the result without allowing a borrowing request to bypass checks.

For searching, the librarian enters a title, author or accession number. Results show bibliographic details and copy availability. A search result should distinguish "no matching title" from "matching title with no available copy". These require different next steps: checking spelling in one case and considering a reservation in the other.

For issuing, the librarian identifies the member and copy, checks the displayed information and confirms the request. The service layer validates eligibility. On success, the system records dates and displays the loan ID. On failure, it explains the relevant rule and changes nothing. The form must not show a success message before the transaction has committed.

For returning, the librarian identifies the open loan and checks the returned physical copy. The system records the return date and closes the loan. A damaged copy is not automatically returned to the available pool; it can be marked for review according to the agreed policy. The software records the decision, but the librarian makes the physical assessment.

Use case	Precondition	Success result	Failure example
Search catalogue	Catalogue loaded	Matching titles and copy status	No result for query
Issue copy	Active member and eligible copy	One open loan created	Borrowing limit reached
Return copy	Open loan exists	Loan closed; copy status reviewed	No open loan for copy
Reserve title	Active member, no duplicate active request	Queue entry recorded	Reservation already active
Overdue report	Authorised role	Read-only open overdue list	Invalid date filter
Restore backup	Authorised administrator; desk paused	Verified database restored	Backup validation fails

The reservation workflow needs particular care. Reservations are attached to a title because a reader normally wants any suitable copy of that edition, not one arbitrary accession number. When a copy returns, the earliest eligible reservation can be offered the copy. A future version may add a hold deadline; the present prototype records the queue and requires librarian review before issue.

Renewal is not included as an automatic action in the first version. If later added, it should check for active reservations and apply an agreed maximum renewal count. Changing the due date should leave a trace so that the original agreement and the extension can both be understood.

9. Entity relationship diagram

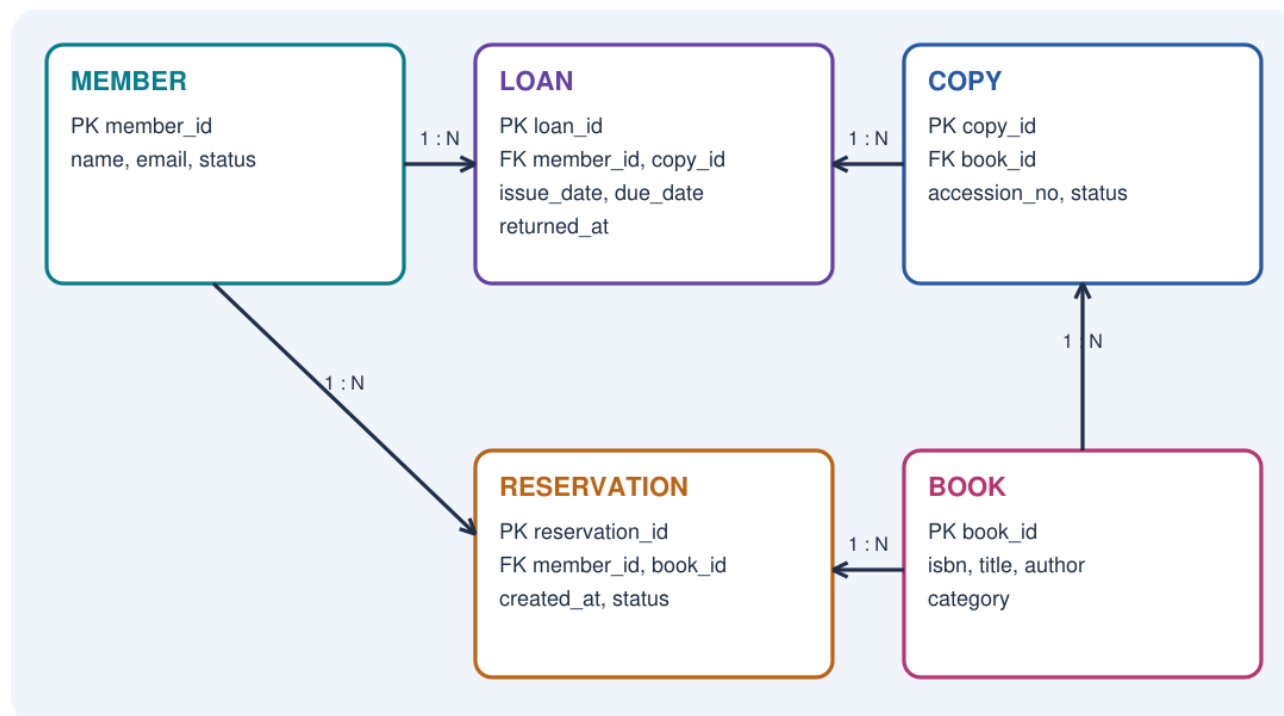


Figure 3. Proposed logical ER diagram. PK = primary key; FK = foreign key. 1:N labels refer to historical records, not simultaneous open loans.

The BOOK entity stores a bibliographic record. COPY stores each physical item and refers to BOOK. MEMBER identifies a borrower. LOAN records a circulation event and refers to a MEMBER and a COPY. RESERVATION records a title-level request and refers to a MEMBER and a BOOK.

One book can have many copies, but each copy belongs to one book record. One member can have many historical loans, and one copy can appear in many historical loans. The additional business rule is that a copy may have only one open loan at a time. That restriction is not expressed by the historical 1:N relationship alone and must be enforced separately.

A member can have many reservations over time. A book can also have many reservations. RESERVATION resolves this many-to-many relationship by storing each request as its own row, with a creation time and status. Queue order is based on creation time and reservation ID as a tie-breaker.

The logical diagram is deliberately small. Staff credentials and administrative logs are supporting records rather than part of the main borrowing relationships shown here. They would be added to a full implementation schema. Removing a member with historical loans should not be allowed casually; making the member inactive preserves the history without permitting new borrowing.

MODEL CHECK

A BOOK is a title or edition record. A COPY is a physical item. A LOAN is one issue event. Do not merge these three ideas into one table simply because a form displays them together.

10. Data flow diagrams

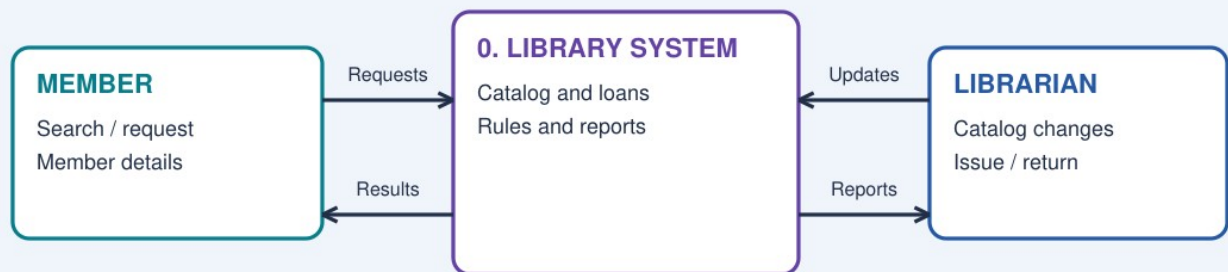


Figure 4. Context DFD (level 0): member and librarian interact with the library system. Arrows show data, not physical movement of books.

The context diagram treats the application as one process. A member provides search or borrowing requests and member details. The application returns results and transaction information. The librarian provides catalogue changes and circulation actions and receives confirmations and reports. Internal tables are not shown at this level.

The level 1 diagram expands the internal processing into catalogue, circulation and reports. Catalogue processing reads and updates **BOOK** and **COPY** records. Circulation reads members, validates a selected copy and writes transaction records. Reporting reads stored records without changing the underlying borrowing state.

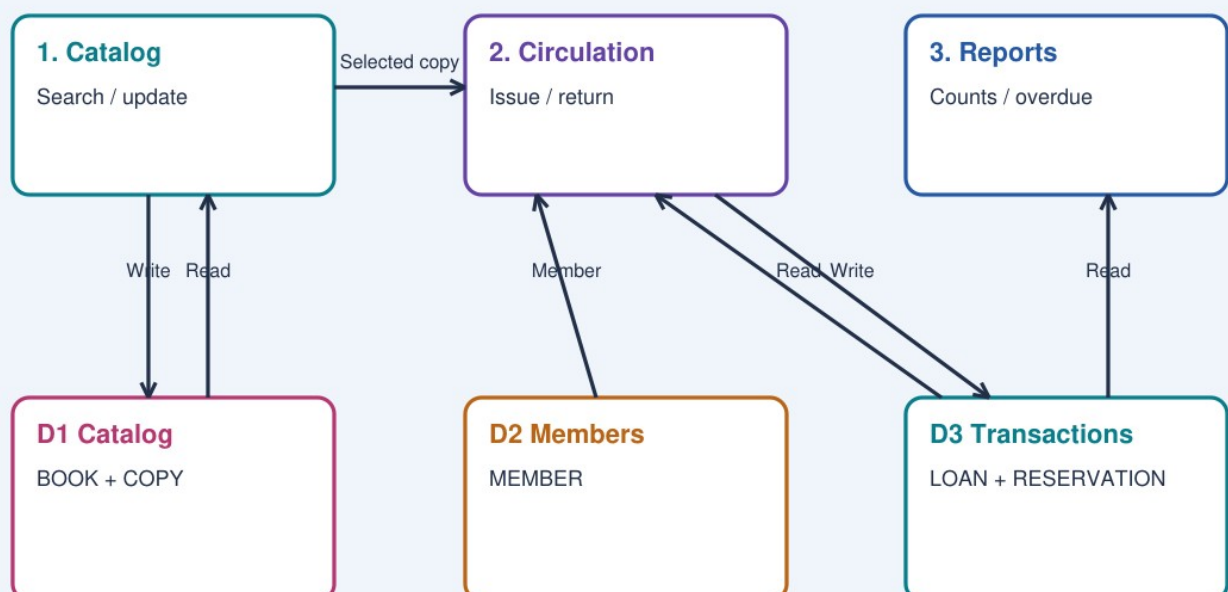


Figure 5. Simplified level 1 internal DFD. External member/librarian inputs follow the context diagram; stores are grouped for readability.

A DFD is not a program flowchart. It explains the movement and storage of information, not the order of every conditional statement. The "selected copy" flow means catalogue information is passed to circulation. The reports process reads transaction records to produce a summary; it does not issue a book.

The simplified diagram groups related records into three stores. D1 contains titles and copies, D2 contains members, and D3 contains loans and reservations. An implementation can use separate tables inside these groups. Diagrams and schema should use the same names so that a reader can follow one example from the form to the stored record.

During implementation review, the developer should check balancing: the externally visible requests and results in the context diagram must still be explained at the lower level. No process should create a meaningful output without an input or stored information that explains it.

11. Database tables and constraints

The data dictionary below defines the core fields. Identifiers are integer keys inside the database; readable codes such as M001 and CP010 are presentation examples. Dates should be stored in a consistent ISO-style representation so that comparison does not depend on how a date is written on the screen.

The schema is designed to avoid repeated data. A book's title is stored in BOOK, not copied into every loan. A loan stores the copy ID, and the copy leads to the relevant book record. If a spelling mistake in a title is corrected, the bibliographic record changes without rewriting the circulation history.

A physical copy can be withdrawn, damaged or available for circulation. Availability is best checked against both its condition/status and the absence of an open loan. A single "available" label maintained independently of loans can become inconsistent. The service layer must treat the two checks as one rule.

BOOK field	Type	Purpose / rule
book_id	INTEGER PK	Internal title/edition identifier
isbn	TEXT, optional	Use when known; not every item has one
title	TEXT NOT NULL	Trim; reject empty title
author	TEXT	Display author or editor
category	TEXT	Local classification label

COPY field	Type	Purpose / rule
copy_id	INTEGER PK	Physical copy identifier
book_id	INTEGER FK	References BOOK
accession_no	TEXT UNIQUE NOT NULL	Library stock number
status	TEXT NOT NULL	circulating, damaged or withdrawn

MEMBER field	Type	Purpose / rule
member_id	INTEGER PK	Borrower identifier
name	TEXT NOT NULL	Display name; not a unique identifier
email	TEXT, optional	Use only when needed

status	TEXT NOT NULL	active or inactive
--------	---------------	--------------------

LOAN field	Type	Purpose / rule
loan_id	INTEGER PK	One issue event
member_id	INTEGER FK	References MEMBER
copy_id	INTEGER FK	References COPY
issue_date	TEXT NOT NULL	Issue date in YYYY-MM-DD form
due_date	TEXT NOT NULL	Must not precede issue date
returned_at	TEXT, nullable	NULL means loan is still open

RESERVATION field	Type	Purpose / rule
reservation_id	INTEGER PK	Queue request identifier
member_id	INTEGER FK	References MEMBER
book_id	INTEGER FK	References BOOK
created_at	TEXT NOT NULL	Timestamp used for queue order
status	TEXT NOT NULL	active, fulfilled or cancelled

Primary keys identify rows; foreign keys enforce valid references. SQLite's documentation states that foreign-key enforcement must be enabled on each database connection. The proposed access layer therefore runs `PRAGMA foreign_keys = ON` before performing application operations. A successful table definition alone is not enough evidence that relationships are being enforced.

The open-loan rule can be supported with a partial unique index on `copy_id` where `returned_at` IS NULL. This allows many historical loans for one copy while rejecting two current ones. A similar partial unique index on `member_id` and `book_id` for active reservations prevents duplicate queue entries by the same member.

Deletion rules should preserve history. A book with copies or a member with loans should not be deleted as a routine clean-up action. Status changes are safer for records already referenced by transactions. If an incorrect record must be removed, the administrator should review its dependencies and retain the reason in an audit record.

ILLUSTRATIVE SCHEMA FRAGMENT

```
PRAGMA foreign_keys = ON;
CREATE UNIQUE INDEX one_open_loan_per_copy
ON loan(copy_id) WHERE returned_at IS NULL;
```

This is a design fragment, not a complete executable application.

12. Issue and return logic

The issue operation is the most important transaction in the project. It must establish that the member is active, the copy is in circulating condition, the copy has no open loan and the borrowing limit has not been reached. In this sample, the demonstration limit is three open loans and the demonstration period is 14 days.

Eligibility is checked inside the transaction, not only before the confirmation button is clicked. Otherwise the result can become stale between checking and inserting. Even in a single-desk prototype, centralising the checks makes the rule easier to test and protects future changes.

The application begins a transaction, reads the member and copy, counts open loans and checks reservations. If any rule fails, it rolls back and returns a specific message. If all rules pass, it inserts the loan with issue and due dates and commits. The screen displays success only after the commit succeeds.

A return identifies the open loan for the copy. It records returned_at and, if appropriate, updates the copy's condition status. It does not delete the loan. Closing a loan twice should be rejected or reported as already returned rather than creating another event that confuses the history.

Step	Issue action	Why it matters
1	Begin transaction	Group the operation into one unit
2	Find active member	Avoid borrowing against a wrong ID
3	Check copy and open loan	Prevent duplicate issue
4	Check open-loan limit and reservation	Apply agreed eligibility rules
5	Insert loan and calculate due date	Create the borrowing agreement
6	Commit, then show confirmation	Avoid false success messages
Failure	Rollback and explain the failed rule	Leave records unchanged

The due date calculation should use date arithmetic rather than adding to the day-of-month field. A loan near the end of a month must still receive a valid date. The screen can show a friendly date format, but the database representation remains consistent.

This prototype does not collect fines or payments. It can show days overdue using $\max(0, \text{return_or_today_date} - \text{due_date})$. If a college wants fines, the amount, holidays, waivers and payment handling need their own agreed requirements. A sample system should not invent a fee policy and present it as an institutional rule.

Reservation handling also needs an explicit policy. The sample gives the earliest active request priority, subject to eligibility, and asks the librarian to review the request before issue. When that reservation is fulfilled, the status changes in the same transaction as the loan. If the operation fails, both changes must roll back.

An example makes the relationship clearer. Member M001 borrows copy CP010. The loan points to M001's internal member ID and CP010's copy ID. Another copy of the same title may still be available. Returning CP010 closes that loan, but the original issue and due dates remain in the record for later reporting.

ILLUSTRATIVE PSEUDOCODE

```
BEGIN TRANSACTION
Validate member, copy, limit and reservation.
If invalid: ROLLBACK; return a clear message.
Create the loan; update any fulfilled reservation.
COMMIT; show the loan ID and due date.
On unexpected error: ROLLBACK; preserve a diagnostic record.
```

13. Interface mockups

The interface should resemble a small working desk tool, not a dashboard full of numbers that the librarian rarely needs. The main navigation can contain Catalogue, Members, Circulation and Reports. The current task should be easy to identify, and each screen should give one clear primary action.

The mockups are drawn for this project file. They are not screenshots of a functioning application. Labels and demonstration values illustrate the proposed information flow. A completed prototype would need actual screenshots after implementation and testing.

LIBRARY DESK / Issue a book

INTERFACE MOCKUP - not an application screenshot

Member ID

M001

Accession number

CP010

Due date

14 days after issue

Validate and issue

Figure 6. Issue-book form mockup with fictional values. Due date is calculated by the service layer after validation.

Search catalogue - form specification	Example display
Query	Database systems
Filter	Title / author / accession
Result	Database systems 3 copies 1 available
Selected title	Book ID 12
Action	View copy details or record reservation
Empty state	No matching title. Check spelling or change filter.

Member registration - form specification	Input / behaviour
Name	Required; reject blank after trimming
Member code	Generated or institution-assigned unique code
Email	Optional in this prototype
Status	Active by default only after librarian review
Save	Validate, commit, then show member ID
Cancel	Close without saving

The issue form should show the member's name after the ID is entered and the book title after the accession number is found. This gives the librarian a chance to catch a wrong record before confirming. The member's borrowing count and copy condition should also be visible, but the service layer still checks them during the transaction.

The return form needs the copy number, the open loan and a condition choice. "Returned" and "available for circulation" are not identical if the book is damaged. The confirmation should state both the closed loan and the resulting copy status so that the next search result makes sense.

Reports should use a preview before export. An overdue list can contain private borrower information, so exporting it is an administrative decision rather than a background convenience. The selected date range and the report's generated time should appear in the output. There should be no payment button in this version.

Return form element	Purpose
Accession number	Identify the physical item
Open-loan summary	Show member, issue and due dates
Condition selector	Circulating / damaged / withdrawn
Confirm return	Close the correct open loan
Result message	Show return recorded and resulting status

14. Validation and error handling

Validation should happen close to the point where a user makes a mistake, but it must also happen in the service and database layers. A form can prevent a blank title; a database constraint protects the same rule if another code path bypasses the form.

Messages should explain the next step. "Invalid input" is usually not enough. "Member M001 is inactive; review the member record before issuing" tells the librarian what failed without pretending the problem was a technical crash. A missing copy should be distinguished from a copy already on loan.

Unexpected failures need a different response. A database write error should leave the transaction unchanged and show that the operation was not completed. The interface should not expose a long traceback to the librarian. A restricted diagnostic log can retain the error type and operation context without copying unnecessary member details.

Repeated clicks are another common source of trouble. The confirmation button should be disabled while an operation is being processed. The database's open-loan uniqueness rule remains necessary because interface controls alone cannot guarantee consistency.

Situation	User-facing message	Data result
Blank title	Enter a title before saving.	No row inserted
Duplicate accession	That accession number already exists.	No duplicate copy
Inactive member	This member cannot borrow while inactive.	No loan inserted
Copy already borrowed	This copy has an open loan.	No second open loan
Loan limit reached	Return a copy before issuing another.	No loan inserted

No open loan on return	No open loan was found for this copy.	No history changed
Database error	The action was not completed. Try again after checking the system.	Rollback

Input handling should trim surrounding spaces and reject empty values. It should not silently alter meaningful names or titles beyond the agreed rules. A title with an apostrophe is valid text and must be handled through parameterised queries, not string concatenation. Dates should be parsed and checked before storage.

Record edits need similar care. Changing a member's name should not change the member ID. Marking a copy as withdrawn should not erase its historical loans. The form should warn when an edit affects future circulation, and administrative changes should have a recorded reason.

The prototype stores only data it needs. An ordinary borrower record does not require sensitive identity documents or payment-card details. Access to the database file and report exports should be limited. If authentication is implemented, passwords need an appropriate password-hashing method rather than plain text. Security details must be designed and tested before a real deployment.

15. Testing plan

Testing is planned around rules and workflows, not just whether screens open. Each test should specify the starting records, the action, expected result and actual result. Actual results remain "Not executed" in this practice file because no working application is being claimed.

Unit tests would cover date calculations, borrowing eligibility and overdue-day calculations. Integration tests would check that a valid issue creates one loan, an invalid issue creates none and a return preserves history. User acceptance tests would ask a librarian to complete realistic desk tasks without instructions from the developer at each click.

The test database should contain more than ideal records. It needs two copies of one title, an inactive member, a member already at the limit, an overdue loan, a damaged copy and duplicate-input attempts. These cases expose the difference between a form that appears to work and a system that protects its records.

Failure tests are particularly important for circulation. An interrupted operation must not leave a reservation fulfilled without a matching loan. A restore test must demonstrate that the chosen backup can actually be opened and contains the expected records. Merely seeing a backup filename is not proof of recovery.

Test ID	Action / starting condition	Expected result	Status
T01	Save a book with blank title	Reject with clear message	Not executed
T02	Add same accession twice	Reject second copy	Not executed
T03	Issue to active member below limit	Create one open loan	Not executed
T04	Issue same copy again	Reject duplicate open loan	Not executed
T05	Issue to inactive member	Reject; no loan	Not executed
T06	Issue fourth open loan under sample policy	Reject limit breach	Not executed

T07	Issue damaged copy	Reject unavailable condition	Not executed
T08	Return a valid open loan	Set returned_at; keep history	Not executed
T09	Return same copy again	Report no open loan	Not executed
T10	Issue near month end	Calculate valid due date	Not executed
T11	Save apostrophe in title	Store text safely	Not executed
T12	Force failure during circulation	Rollback all changes	Not executed
T13	List overdue loans	Include only open loans past due	Not executed
T14	Restore backup into test location	Expected records available	Not executed
T15	Create duplicate active reservation	Reject duplicate request	Not executed
T16	Fulfil reservation with failed issue	Reservation remains active	Not executed

Defects should be recorded with reproduction steps and their effect on records. A defect that permits duplicate open loans is more serious than a misaligned label. The release decision should reflect that difference. The test report must also record the application version, database version and environment so that a pass is not treated as permanent proof for later changes.

After a fix, the failing case is rerun and related cases are checked for regression. For example, changing return logic can affect availability, overdue reporting and reservation fulfilment. A narrow fix should not be accepted until those linked outcomes have been reviewed.

A useful acceptance demonstration follows one fictional copy through its whole life: add it, search it, issue it, try a second issue, return it, mark it damaged and search again. The database and interface should agree at every stage. This is more persuasive than several isolated screenshots of successful forms.

RELEASE GATE

No unresolved defect may permit duplicate open loans, missing transaction history or silent partial updates. A verified backup restore and a successful desk-workflow review are required before real use.

16. Deployment and maintenance

Deployment begins with checking the target computer, installed Python/Tk support and the location of the database. The application should create required tables through a controlled setup process and verify its schema version. The librarian should not have to guess which database file the program is using.

Before bringing in real records, use a demonstration database to check search, issue and return. Importing existing registers should be a separate step with validation and review. Duplicate names do not necessarily mean duplicate members, and matching titles do not necessarily mean the same edition. Accession numbers need careful checking because they identify the physical stock.

The first real deployment should have a rollback plan. Preserve the original records, test the imported data and agree how the desk will operate if the application is unavailable. A paper fallback can record temporary

circulation, but those entries must later be reconciled to prevent the same copy from being issued again digitally.

Deployment check	Evidence required
Environment	Application starts on target machine
Database path	Configured location is displayed and documented
Schema	Expected version and required constraints present
Roles	Ordinary user cannot restore or alter settings
Migration	Sample records checked against original source
Circulation	Normal and failure cases reviewed
Backup	Restore into separate test location succeeds
Fallback	Librarian knows temporary-record process

Backups should be made through a supported database backup mechanism rather than by casually copying a live file in the middle of a write. The backup process needs a known destination, date and retention plan. At least one copy should be protected from the same failure that could damage the desk computer. Access should be restricted because backups contain member and borrowing data.

Maintenance includes checking backup results, reviewing defects and documenting changes to borrowing rules. A database migration should be tested on a copy before touching the live records. New code should not simply assume that an old database contains the latest fields.

A restore operation needs administrative authority and a paused circulation desk. Otherwise restoring an older backup could erase transactions made after that backup. The operator should know the recovery point and reconcile any missing temporary records. Recovery is a controlled workflow, not a button to click whenever a report looks odd.

17. Limitations and future scope

The design is limited to a small local prototype. It does not demonstrate network performance, multiple-desk concurrency, RFID integration or payment collection. The interface images are mockups and the test matrix is a plan. A real implementation must provide code, executed tests and actual screenshots before any completion claim is made.

The data model keeps author information simple. A larger catalogue may need separate AUTHOR and BOOK_AUTHOR tables for multiple authors and controlled names. Edition details, publishers and subjects may also deserve richer modelling. That expansion should follow a real catalogue need rather than add complexity only to make the project seem larger.

A future member portal could allow searches and reservation requests. It would require authentication, privacy controls and a clear separation between public catalogue information and private borrowing history. A member should not be able to view another person's loans merely by changing a record ID.

Barcode support is another practical extension. A scanner can reduce typing, but the barcode must still map to the correct accession record. It does not remove the need for membership and availability checks. Similarly, notifications can remind borrowers, but should use confirmed contact details and an agreed communication policy.

A networked version would need a deliberate architecture review. Shared access increases the risk that two desks try to issue the same copy. Server-side transaction rules, role checks and logging would become more important. Such a version should not rely on the single-computer assumptions used in this sample.

Future features should be ranked by the problem they solve. Accurate circulation and recovery come before graphs, mobile apps or fine-payment screens. A modest system that preserves records is more useful than an impressive interface that silently loses them.

Possible extension	Benefit	New work required
Barcode scan	Less manual copy entry	Accession mapping and scan tests
Member portal	Self-service catalogue and requests	Authentication and privacy review
Multiple authors	Better catalogue accuracy	Schema and migration changes
Reminders	Fewer forgotten due dates	Confirmed contact and sending rules
Multi-desk access	Shared circulation workflow	Server and concurrency design
Digital resources	Manage links or licensed access	Rights and access-policy checks

18. Conclusion and bibliography

The project design shows how a simple library workflow can be represented with related records and clear rules. The central idea is to keep titles, physical copies and borrowing events separate. That makes it possible to answer availability questions accurately without losing history.

The proposed system would support catalogue maintenance, member records, circulation, reservations and reports. Its success would depend on more than a friendly interface. Eligibility checks, atomic transactions, privacy controls and verified recovery are part of the core design.

Preparing the diagrams and table definitions also showed where assumptions need to be confirmed. Borrowing periods, limits, reservation priority and damaged-copy handling are institutional decisions. Software can apply them consistently only after the rules are agreed.

This practice file is therefore a starting point for implementation, not a claim that the project has passed testing. The next useful step is to build a small circulation module, run the planned failure cases and then revise the design based on actual results.

Bibliography

Python Software Foundation. (n.d.). tkinter - Python interface to Tcl/Tk. Python documentation.
<https://docs.python.org/3/library/tkinter.html>

SQLite. (n.d.). SQLite foreign key support. <https://www.sqlite.org/foreignkeys.html>

The diagrams, interface mockups, sample policies and test matrix in this file were prepared for this academic practice example. They are not copied from a deployed library system. Documentation links were consulted in October 2026.

BEFORE A REAL SUBMISSION

Fill the student details. Obtain genuine certificate approval if required. Replace mockups with actual screenshots only after implementation. Execute the tests and record results. Check the college's required format and project scope.