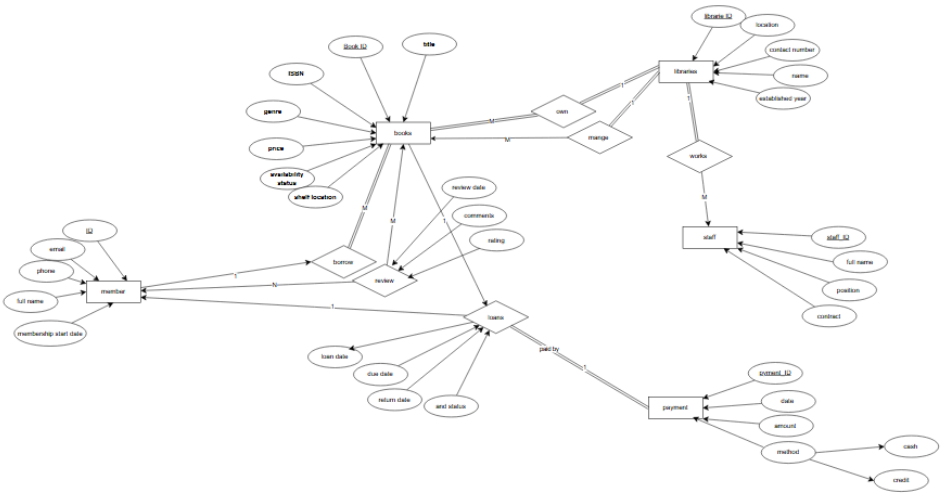
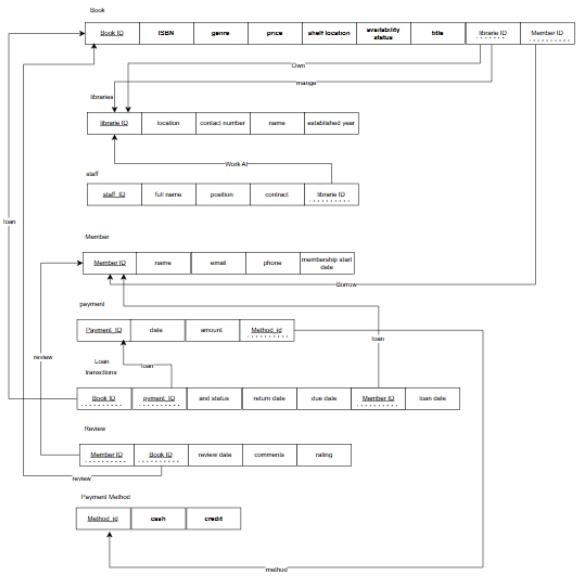
Library Database Design Comparison



normalized



Option 1 – Basic ERD with Normalized Schema

Normalization

- 1NF: Met – each field contains atomic values.
- 2NF: Met – no partial dependencies since most tables have simple primary keys.
- 3NF: Partially met – Loan as a relationship limits clean separation of attributes.

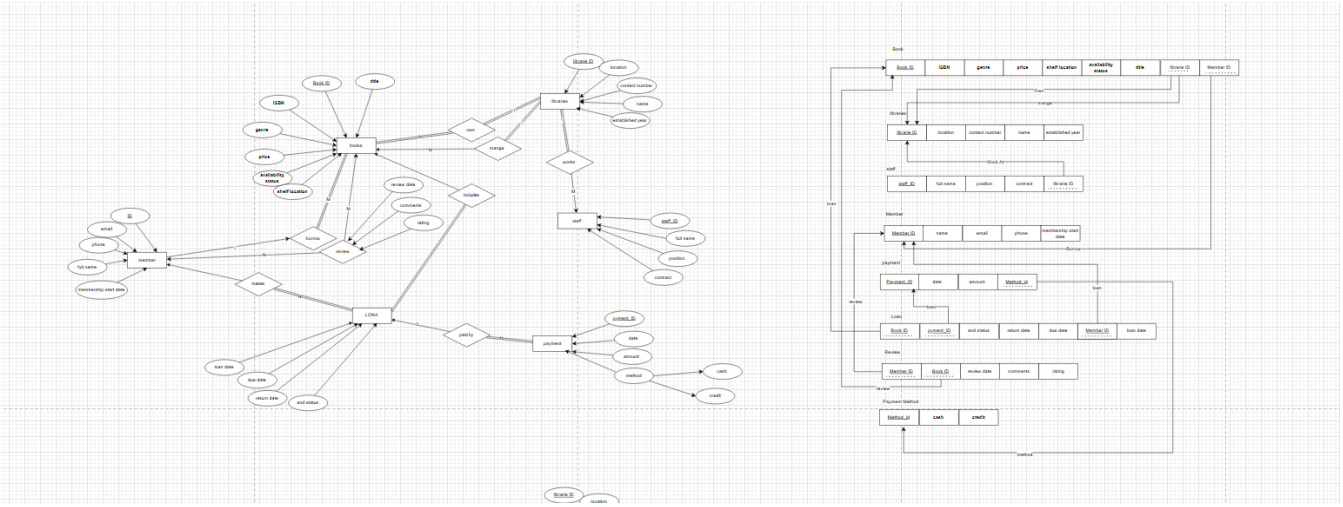
Pros:

- Clear ERD structure.
- Normalized relational schema.
- Core relationships (`borrow`, `review`, `payment`) are well mapped.

Cons:

- `Loan` is modeled as a relationship, not an entity, which limits storing detailed loan data like `due_date`, `status`, etc.
- No `Transaction_Efficiency` entity.
- Less scalable for performance and reporting use cases.
- Cannot support weekly or historical performance tracking.
- If the system later needs `Transaction_Efficiency`, major redesign would be required.
- Difficult to adapt for audit trails or loan-based reporting.

Use Case Fit: Basic systems with simple operations and no performance tracking.



Option 2 – Loan as an Entity

Normalization

- 1NF: Met – atomic values ensured.
- 2NF: Met – full functional dependency exists on primary/composite keys.
- 3NF: Mostly met – data is cleanly separated but lacks performance layers.

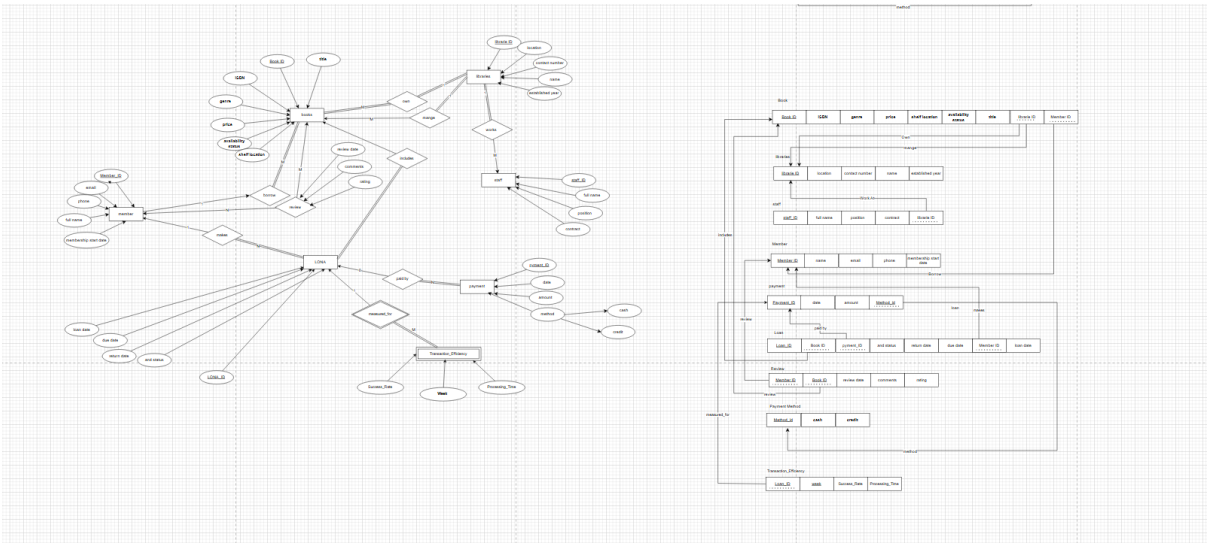
Pros:

- Loan (LONA) is promoted to a full entity, allowing for full attribute tracking.
- Relational schema is normalized and clear.
- Payment, review, and book relations are mapped correctly.

Cons:

- Still missing the Transaction_Efficiency entity.
- Loan_ID or composite key not clearly emphasized.
- No performance tracking or weekly metrics for loans.
- If performance analytics are added in the future, structural changes are needed.
- Less flexible for time-based reporting or efficiency evaluations.

Use Case Fit: Suitable for medium systems that need better loan tracking but no analytics.



Full Design with Performance Tracking (Chosen)

Normalization

- 1NF: Fully atomic fields.
- 2NF: Composite key (Loan_ID, Week) in weak entity ensures full dependency.
- 3NF: No transitive dependencies – best structure.

Pros:

- Loan is a complete entity with all required attributes (loan date, due date, return date, status, payment).
- Transaction_Efficiency is added as a weak entity:
    - Linked to Loan via Loan_ID
    - Composite Primary Key: (Loan_ID, Week)
- Proper use of foreign keys and normalization across all tables.
- All entities are in 3NF.
- Scalable, flexible, and ready for real-world implementation.

Cons:

- Minor layout/labeling improvements could enhance visual clarity (not a structural issue).

Use Case Fit: Full-featured library systems with reporting, analysis, and performance evaluation.

Why Loan is an Entity

- Tracks: `loan_date`, `due_date`, `return_date`, `status`, and `payment_ID`.
- Relationships alone can't hold this level of detail.
- Enables connection to reviews, payments, and efficiency data.

Why `Transaction_Efficiency` is a Weak Entity

- Cannot exist without a corresponding `Loan`.
- Measured per week for each loan.
- Depends on foreign key: `Loan_ID` + `Week`.
- Useful for tracking library performance (e.g., processing time, success rate).

Why `Transaction_Efficiency` is an Entity (Not an Attribute)

- It holds multiple values per loan over time (e.g., Week 1, Week 2, etc.).
- Needs its own structure to allow storing:
  - `Success_Rate`
  - `Processing_Time`
  - `Week`
- Makes performance analysis easier and modular.
- Using it as an entity avoids repeating efficiency data inside the `Loan` table.
- Makes it easier to extend (e.g., future add-ons like `delay_reasons`, `staff_efficiency`, etc.).