Lecture 14 - Designing a Simple Database

The following notes will walk you through how to design a simple database solution for a Library. For simplicity, the data model we use will not be completely representative of what you would use in reality as this would require more complexity.

We will build this using our intuition and common sense, but in the next lecture we will employ normalisation as a more systematic approach to building a database.

1. Scenario

A small local library currently uses a paper-based system. Each book (on the shelf) in the library has a unique ID and there can be multiple copies of the same book. Each member of the library has a member card with an ID. When a member wants to loan out a copy of a book the librarian will stamp the book with the current date and also fill out an entry in the ledger as follows:

```
Member ID, Book ID, Date of Loan, Expected Date of Return
```

When the member returns the book, the librarian will stamp the book with the return date. The librarian will also look at the date of the loan (previous stamp) and then look up the date and entry in the ledger and amend as:

```
Member ID, Book ID, Date of Loan, Expected Date of Return, Date of Return
```

If the book is late then a fine will be issued:

```
Member ID, Book ID, Date of Loan, Expected Date of Return, Date of Return, Fine Issued
```

Renewal of a book is just treated as a new loan. However, you should consider the limitations of this. It means that it is hard to track how many times. Ideally, the library should have a way of amending the `Expected Date of Return` and tracking the number of renewals. This would allow them to check that someone wasn't unfairly keeping a copy of a book.

- We treat renewals as a new loan entry of the same book for the same member.
- We treat each book loaned out as a single entry. e.g. if a member loans out 3 books in a visit that is 3 entries.

2. A First Attempt

We could just digitise the ledger used by the library. This is just a table in Excel, Access or anything that deals with tables and data entry.

`Loans` Table

MemberID	BookID	DateOfLoan	ExpectedDateOfReturn	DateOfReturn	FineIssued
12345	ABC123	17/02/24	02/03/24	02/03/24	0.00
54321	XYZ321	17/02/24	02/03/24	03/03/24	0.50
...					
34512	ABC451	22/02/24	07/03/24	04/03/24	0.00
12345	BFG135	23/02/24	08/03/24	08/03/24	0.00

This does indeed capture the data required but this solution on it's own has considerable limitations:

1. Each member has an ID, but how is it computed?
2. The ledger captures the member ID, but do we know any more information about the member? Would this be useful?
3. Each book has an ID, but how is that computed?
4. The ledger captures the book ID, but what book is it?
5. Which books are currently out on loan?
6. How many books does a member have out? Maybe there is a limit?
7. Which books are available in the library?
8. How many copies are there of a given book?

1 and 2 should be screaming at you, as should 3 and 4. We could put this information in the table, but that would mean entering that information again and again. No thanks!

We could provide answers to 5 and 6 using our system, but it requires a lot of work. 7 and 8 we cannot as we are only recording loans, not the book inventory.

We will see by extending for points 1,2,3 and 4 that answers to 5,6,7 and 8 will come more readily.

3. A Second Attempt - Additional Tables

The above points demand two other tables, one for members and another for books (inventory). i.e.

`Members` Table

MemberID	Name	DateOfBirth	Email
12345	Joe Bloggs	TODO	TODO
54321	TODO	TODO	TODO
34512	TODO	TODO	TODO

`Books` Table

BookID	Title	Author	PublicationYear
TODO	TODO	TODO	TODO
TODO	TODO	TODO	TODO
TODO	TODO	TODO	TODO

This is now a single place for this information. You can now look up the information for a given member or book from the corresponding tables `Members` and `Books`. You can also cross-reference information, i.e. if you look at an entry in the `Loans` table you can now look up both the member and the book information.

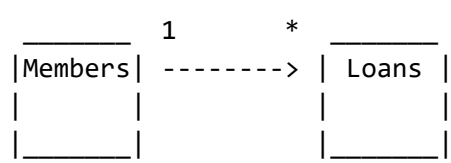
3.1 Linking Tables (Relationships)

The above illustrates that these are linked. How are they linked?

Consider the `Members` and `Loans` tables.

- A member can have many loans
  - This makes sense as a member can make many loans.
- A loan can only have one member.
  - This makes sense as a single loan can't be made by many members.

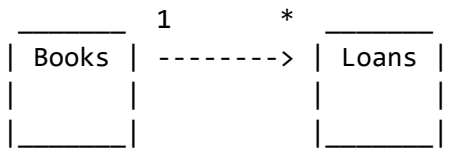
This means there is a one-to-many relationship between `Members` and `Loans`.



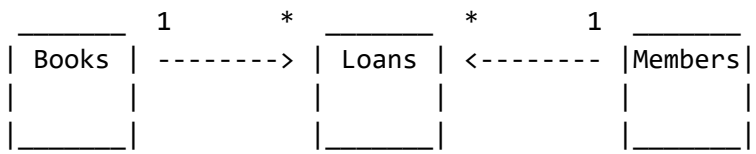
How about the `Books` and the `Loans` tables?

- A book can have many loans
  - This makes sense as a book can be loaned out many times.
- A loan can only have one book.
  - This makes sense as a loan entry as defined above only records a single book.

This means there is a one-to-many relationship between `Books` and `Loans`.



Putting these together we get the following Entity Relationship Diagram:



3.2 Duplicate Data

We could now look to build this, however, if we look at the `Books` Table again:

BookID	Title	Author	PublicationYear
TODO	TODO	TODO	TODO
TODO	TODO	TODO	TODO
TODO	TODO	TODO	TODO

We notice that we could have duplicate information for the multiple copies of a book that the library has.

There are two main reasons to avoid this:

1. Data duplication (we are unnecessarily storing the same data multiple times). The only difference is the BookID.
2. Updating book information (e.g. adding info or incorrect information) requires doing this for all entries in the `Books` table.

If we think about the entities involved there are two. The **book** (it's contents and information) and the physical **copies**.

We can therefore add another table `Copies` that contains details information about the copy (e.g. condition, is it borrowed?) and link that to the `Book` information.

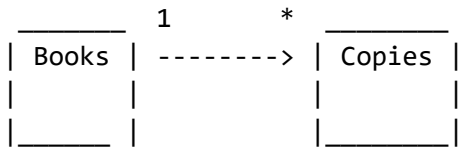
CopyID	BookID	Condition	Borrowed
TODO	TODO	TODO	TODO
TODO	TODO	TODO	TODO
TODO	TODO	TODO	TODO

Now as long as we don't duplicate information in the `Books` table, a book will have a unique `Book ID` which we can then store in the `Copies` table multiple times.

If the relationship isn't obvious, let's examine it.

- A book can have many copies
  - This makes sense as there can by multiple copies of the same book on a shelf.
- A copy can only have one book.
  - This makes sense as a copy is a single book!

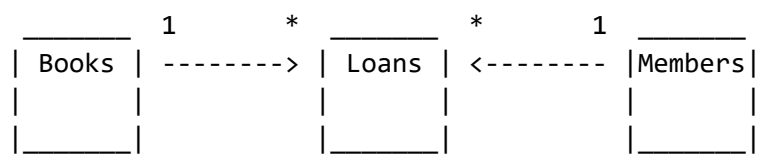
This is a one-to-many relationship between `Books` and `Copies`.



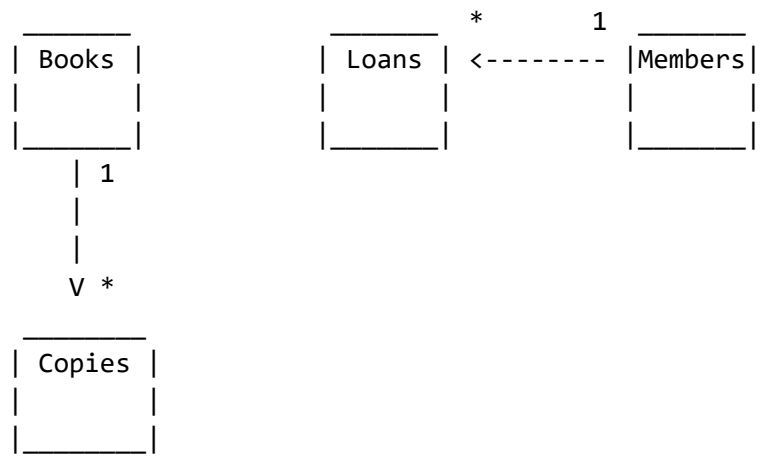
- The `Books` table no longer represents copies of a book, thus it isn't linked to `Loans`.

3.3 A Final Entity Relationship Diagram (ERD)

Let's update our existing ERD (given below).



- The `Books` table is linked to `Copies` (one-to-many).
- The `Books` table no longer represents copies, thus it isn't linked to `Loans`.

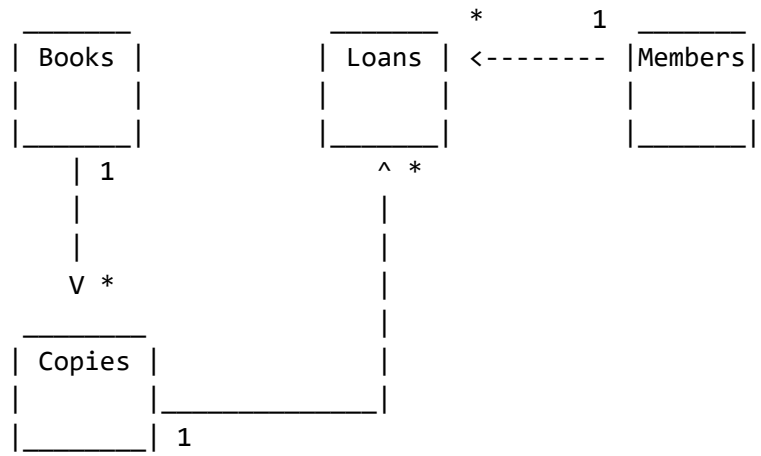


Now we need to reconnect the `Loans` table. If we think about want a loan represents, it represents taking out a physical copy of the book. That is the `Copies` table.

- A copy can have many loans
  - This makes sense as a copy of a book can be loaned out many times.
- A loan can only have one copy of a book.
  - This makes sense as a loan entry as defined above only records a single copy of a book.

This is a one-to-many relationship between `Copies` and `Loans`.

Thus our final ERD is:



4. Implementing the Tables

Now that we have our ERD we can define our primary and foreign keys.

4.1 Primary Keys

- A primary key is the unique field (attribute) that uniquely identifies an entry in a table.

For `Members`, `Copies` and `Books` this is simply their IDs.

- `MemberID`
- `CopyID`
- `BookID`

`Loans` is more difficult as it is perfectly reasonable for the same member to take out the same book on different occasions. We could use the composite key (`MemberID`, `CopyID`, `DateOfLoan`) as this is unique.

However, as a rule, we won't use composite keys. For the sake of an occasional additional field, it will save complexity. We will instead introduce a `LoanID`.

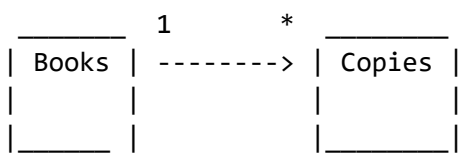
4.2 Foreign Keys

- A foreign key is a field (attribute) that appears in another table as the primary key.

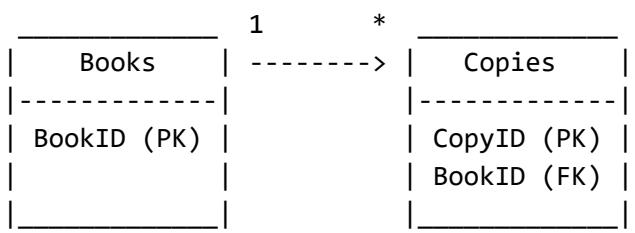
You can think of this another way. If two tables A and B have a one-to-many relationship then the following must be true.

- The primary key of A must appear as a foreign key in B.

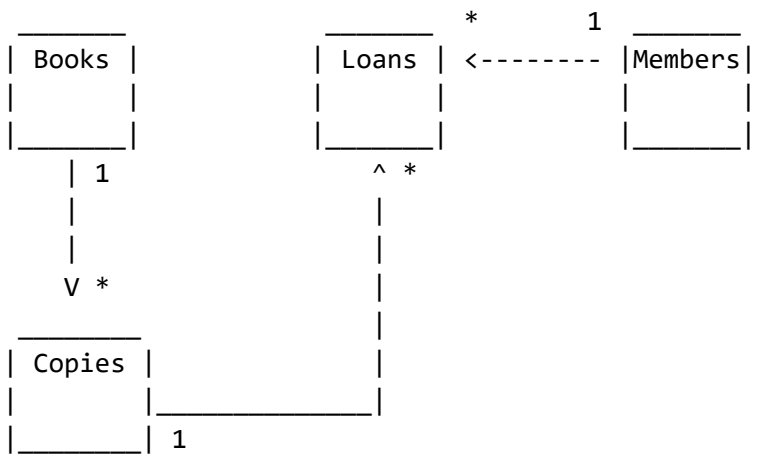
For example, `Books` and `Copies` have a one-to-many relationship.



Thus the primary key of `Books` must appear as the foreign key of `Copies`.



We can do this for our ERD:



Which gives us the following, this now makes implementing this very easy.