Project Specifications

For our database, we chose to represent it with the following 9 entities.

Item {item_id, title, author, type, branch_id, status, dewey_decimal, borrow_date, due_date}
Customer {cust_id, first_name, last_name, email, phone, membership_status}
Borrowed {item_id, cust_id }
Fine {fine_id, cust_id, item_id, amount, paid_status}
Event {event_id, room_id, branch_id, event_name, event_type, date, time, recommended_audience}
Registered {cust_id, event_id}
Room {room_id, branch_id, location, capacity}
Employee {employee_id, branch_id, first_name, last_name, role, email, phone }
Branch {branch_id, phone}

The 'Item' entity allows for lots of flexibility when handling different types of items a library can hold. The 'type' attribute is used to represent the type of the item: [Print, Online, Magazine, Journal, CD, Record]. We'll use a SQL Check Constraint to enforce this. The specific types are subject to what the library offers, but we've limited it to the list above. Since different types of items can have different specific attributes, we considered using an 'is-a' relationship with inheritance to separate the different types, but we realized that this would lead to too many extra tables with overlapping attributes, and over complicate our schema unnecessarily. An item also has a status, limited to [Available, Borrowed, Reserved, Pending]. The first three are self explanatory, but the 'pending' status refers to items that are planned to arrive in the future, but have not arrived yet.

The 'Customer' entity represents all of the library patrons. The membership status can be in one of two states: [Active, Suspended]. This denotes whether or not a customer is allowed to borrow items from the library. Each new customer starts with a status of 'Active', but this is automatically changed to 'Suspended' upon receiving 3 or more fines. We'll use a Trigger to automatically update the Customer's status upon receiving too many fines, and another to restore a customer's 'Active' status upon paying for the fines.

'Borrowed' is a relation used to denote customer item pairs representing which customer is borrowing which item. A customer must have the 'Active' status in order to borrow items, and so we'll use another trigger to check if a customer has the 'Active' status, and either allow the transaction to proceed, or abort it. We will also use a trigger to update the status of an 'Item' if it has been borrowed. We should also include a trigger to update the status of an 'Item' if the borrowed relation is deleted, making the item available again. Lastly, we'll need a trigger to set the borrow date to the current date when an item is borrowed, as well as set its corresponding due dates. We'll set these back to 'NULL' once an item is returned.

A 'Fine' denotes when a Customer is still borrowing an item after the item's due date. We could use a Trigger to check the current date and the item's date every day, but for the purposes of this project, our trigger will update when a customer eventually does return the item, and the fine will be a constant $10.00.

An ‘Event’ is an event hosted by a specific branch in a specific room. Each room has a capacity which limits the number of attendees, which is discussed more below.

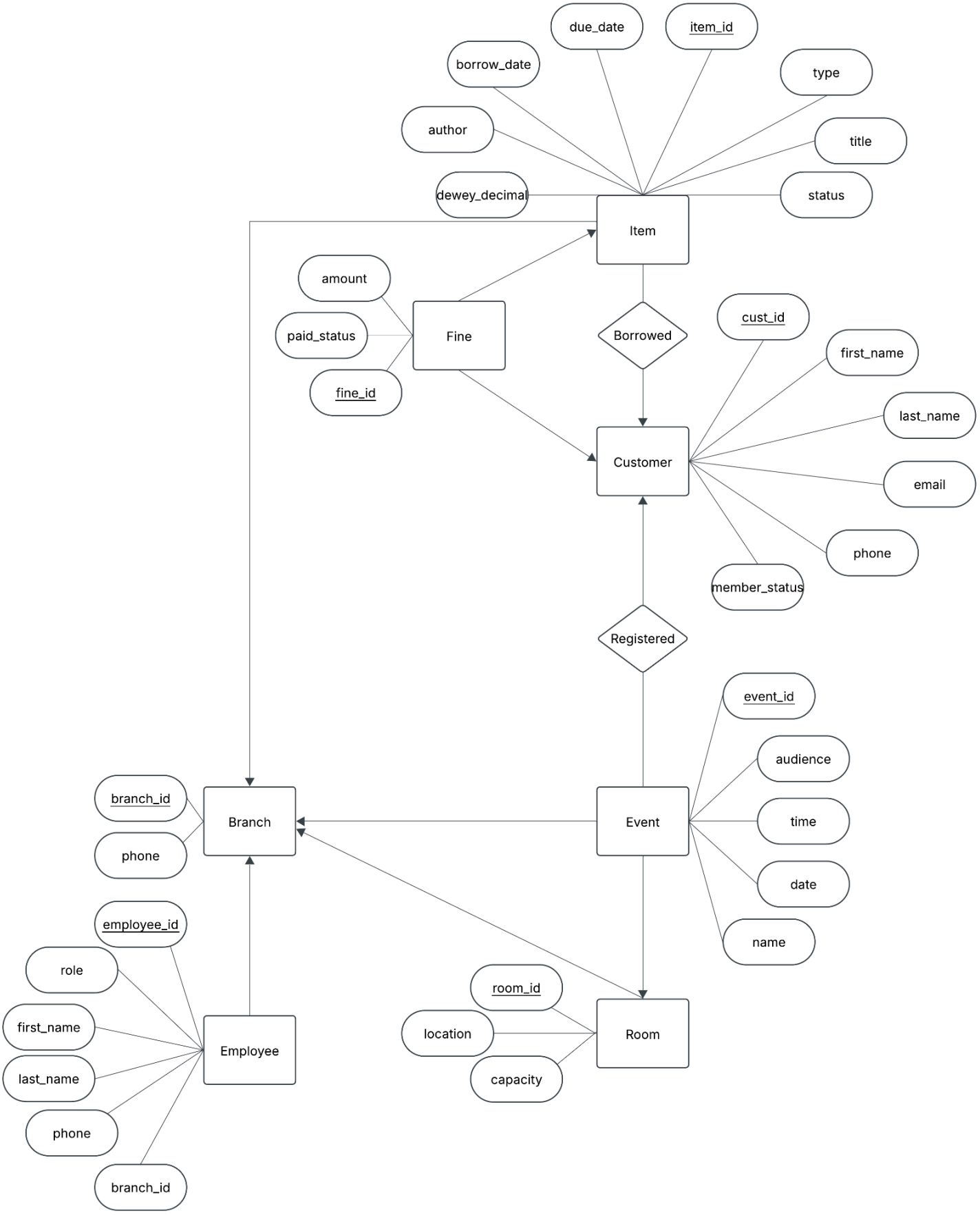
‘Registered’ represents the relation between customers, and the events they’re registered for. Since each event has a specific room, and each room has a maximum capacity, we’ll use a trigger to enforce the constraint that when registering for a new event, the total number of customers registered for that specific event must be less than the event room’s capacity.

A ‘Room’ is a social room in a specific library branch used to host events. It has a location within the branch, and a capacity, which is used to constrain the maximum number of attendees to an event.

An ‘Employee’ has a specific role at a specific branch. For the purpose of this project, we’ve limited the role of employees to [Librarian, Assistant, Event Coordinator, Volunteer]. Volunteers are automatically added upon application, and can be filtered out easily.

Lastly, a ‘Branch’ denotes one of many physical locations of a library.

E/R Diagram


due_date
item_id
borrow_date
type
author
title
dewey_decimal
status
Item
amount
paid_status
fine_id
Fine
Borrowed
cust_id
first_name
last_name
email
phone
member_status
Customer
Registered
event_id
audience
time
date
name
Event
branch_id
phone
Branch
employee_id
role
first_name
last_name
phone
branch_id
Employee
room_id
location
capacity
Room

Anomalies

Our design does not allow anomalies. The following is a list of functional dependencies from our database schema.

item_id → title, author, type, branch_id, status, dewey_decimal, borrow_date, due_date
cust_id → first_name, last_name, email, phone, membership_status
(cust_id, item_id) → cust_id, item_id *[Trivial]*
fine_id → cust_id, item_id, amount, paid_status
event_id → event_name, event_type, date, time, recommended_audience, room_id, branch_id
(cust_id, event_id) → cust_id, event_id *[Trivial]*
room_id → branch_id, location, capacity
employee_id → first_name, last_name, role, email, phone, branch_id
branch_id → phone

The underlined terms represent the keys in our database. For each of these functional dependencies, we notice that the left hand side always consists of a key (or superkey), or the relation is trivial. This means that our schema is in BCNF, as long as there are no more hidden functional dependencies, not explicitly stated by our design. We argue that there aren't any hidden functional dependencies, since for each table schema, all of its attributes depend solely on the key. As an example, the author of a book does not solely depend on its title (books may have the same or similar names, but different authors and the same author can write multiple different books). We note that each of our tables has a specific id unique to that table, with the exception of the relations (which are by design, trivial).