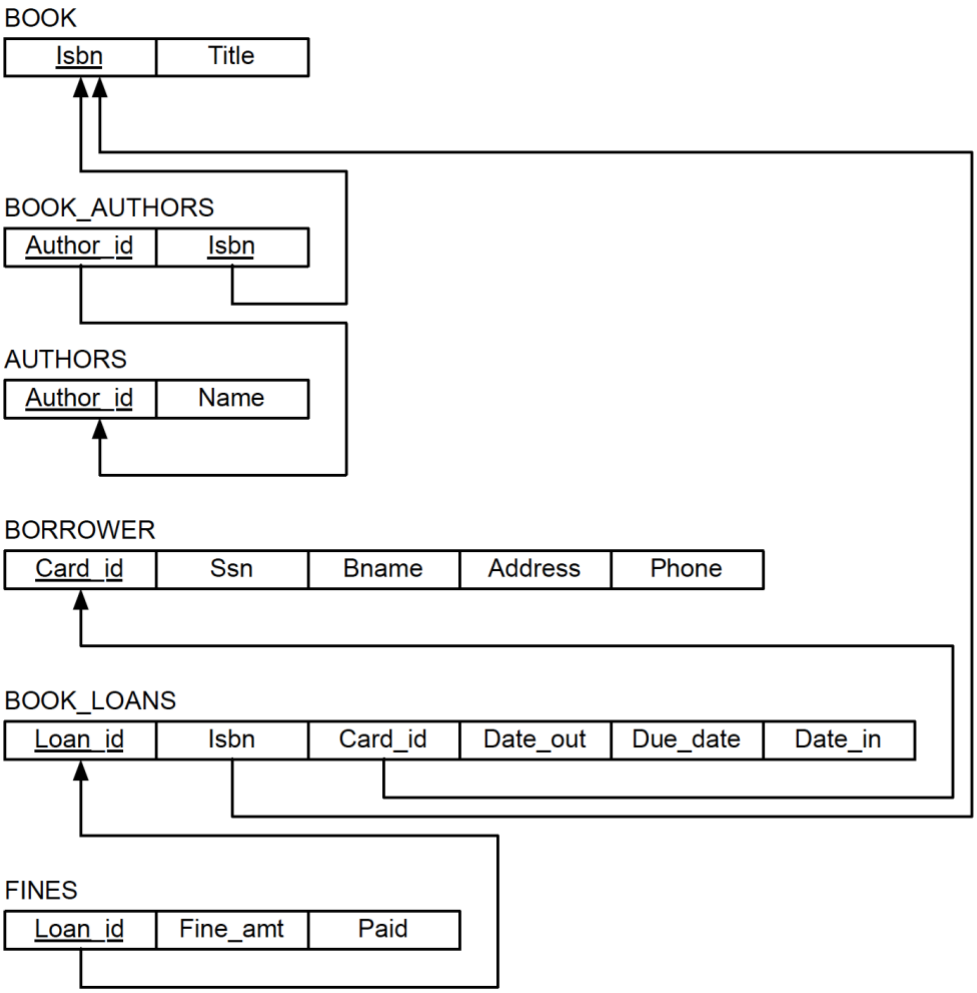
Database Design Document：

1、I simply design the database by using the original library database scheme and that is compatible. We have the scheme below.



2、create the database models according to the scheme above. Here I use django's models, makes us easier to create a database scheme in our database.

```python
from django.db import models

# Create your models here.


class Book(models.Model):
    title = models.CharField(max_length=200)
    ISBN=models.CharField(max_length=10)

class AUTHORS(models.Model):
    Name=models.CharField(max_length=100)


class Book_AUTHORS(models.Model):
    Authors_id=models.ForeignKey(AUTHORS,on_delete=models.CASCADE)
    ISBN = models.ForeignKey(Book, on_delete=models.CASCADE)

class BORROWER(models.Model):
    ssn=models.CharField(max_length=15)
    Bname=models.CharField(max_length=100)
    Address=models.CharField(max_length=100)
    Phone=models.CharField(max_length=100)

class BOOK_LOANS(models.Model):
    ISBN=models.ForeignKey(Book,on_delete=models.CASCADE)
    Card_id=models.ForeignKey(BORROWER,on_delete=models.CASCADE)
    Date_out=models.DateTimeField(max_length=20)
    Due_date=models.DateTimeField(max_length=20)
    Date_in=models.DateTimeField(max_length=20)

class FINES(models.Model):
    Loan_id=models.ForeignKey(BOOK_LOANS,on_delete=models.CASCADE)
    Fine_amt=models.DecimalField(max_digits=5,decimal_places=2)
    Paid=models.BooleanField(max_length=10)
```

Then by two instruction “python manage.py makemigrations” and “python manage.py migrate”, we can easily connect to the database and help us to create those tables based on the fields and variable we define above.

```python
import ...


class Migration(migrations.Migration):

    initial = True

    dependencies = [
    ]

    operations = [
        migrations.CreateModel(
            name='AUTHORS',
            fields=[
                ('id', models.AutoField(auto_created=True, primary_key=True, serialize=False, verbose_name='ID')),
                ('Name', models.CharField(max_length=100)),
            ],
        ),
        migrations.CreateModel(
            name='Book',
            fields=[
                ('id', models.AutoField(auto_created=True, primary_key=True, serialize=False, verbose_name='ID')),
                ('title', models.CharField(max_length=200)),
                ('ISBN', models.CharField(max_length=10)),
            ],
        ),
        migrations.CreateModel(
            name='Book_AUTHORS',
            fields=[
                ('id', models.AutoField(auto_created=True, primary_key=True, serialize=False, verbose_name='ID')),
                ('Authors_id', models.ForeignKey(on_delete=django.db.models.deletion.CASCADE, to='lib.AUTHORS')),
                ('ISBN', models.ForeignKey(on_delete=django.db.models.deletion.CASCADE, to='lib.Book')),
            ],
        ),
        migrations.CreateModel(
            name='BOOK_LOANS',
            fields=[
                ('id', models.AutoField(auto_created=True, primary_key=True, serialize=False, verbose_name='ID')),
                ('Date_out', models.CharField(max_length=10)),
                ('Due_date', models.CharField(max_length=10)),
                ('Date_in', models.CharField(max_length=10)),
            ],
        ),
```

Note: In same models (corresponding to the table one by one) the Django may help us to create the unique 'id', and use as a primary key. For example, in the book model, except for the 'title' and 'ISBN', the system creates a field 'id' as primary key. Nonetheless, the ISBN and the id is a one to one relationship. So actually, the ISBN can still be the primary key, which will not change the scheme we have mentioned above.

3、Load all the data we need from the two csv file "books.csv borrowers.csv".Here is how I did.

There are lib_book, lib_borrower, *lib_authors,* lib_book_authors 4 tables we should enter data into.

a) LOAD DATA From 'books.csv' to lib_book:
lib_book have 3 fields (id(Autoincrement),ISBN and title), id will increase as we LOAD the DATA

LOAD DATA INFILE 'books.csv'
INTO TABLE lib_book IGNORE 1 LINES (ISBN, @dummy,title,@dummy,@dummy,@dummy,@dummy)

b) Load data from 'borrowers.csv' to lib_borrower

LOAD DATA INFILE 'borrowers.csv' INTO TABLE lib_borrower FIELDS TERMINATED BY ',' IGNORE 1 LINES (id,ssn,@var1,@var2,@dummy,@Address,@dummy1,@dummy2,phone) SET Bname=CONCAT_WS(",",@var1,@var2), Address=CONCAT_WS(",",@Address,@dummy1,@dummy2);

c) We first use a **new_table (id,Aname,ISBN)** and load the corresponding Author name and ISBN in this table. However, the column "Aname"(Author name) can have several authors which is separated by the ',' then I have to split them and generate a new table named **ISBN_AUTHOR(id,ISBN,name)** while we one ISBN can have multiple name and one name can have multiple ISBN.

Insert into ISBN_AUTHOR (select a.id,a.ISBN,substring_index(substring_index(a.Aname,',',b.help_topic_id+1),',',-1) name from new_table a join mysql.help_topic b on b.help_topic_id < (length(a.Aname) - length(replace(a.Aname,',',''))+1) order by a.id);

d) Now we need to Load in the table Author, insert them into table lib_authors.

Insert into lib_authors(Name) select distinct name from ISBN_AUTHOR;

e) Then we need to insert the lib_book_authors(ISBN_id,Authors_id_id). We pair the ISBN_id with Authors_id_id where ISBN_AUTHOR.name=lib_authors.Name

Insert into lib_book_authors (ISBN_id,Authors_id_id) select ISBN_AUTHOR.id,lib_authors.id from ISBN_AUTHOR,lib_authors where ISBN_AUTHOR.name=lib_authors.Name;

4、Design the GUI

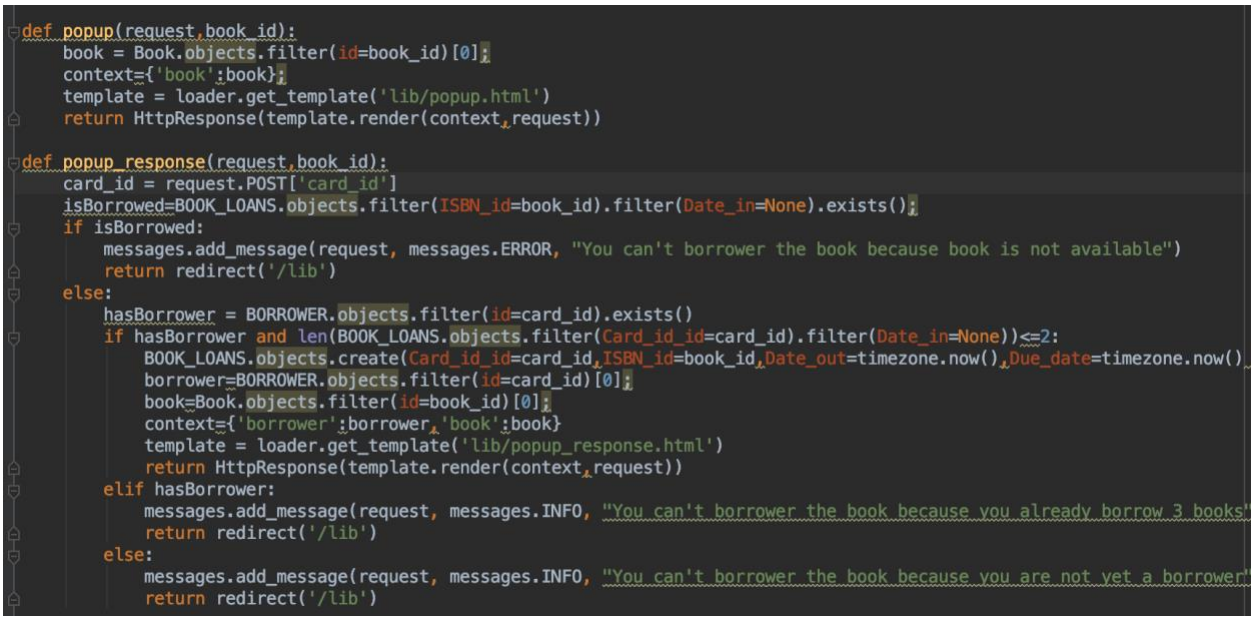
Using template, which is widely used in Django. Since this is a database course, we talk less about the GUI. All the GUI are shown in the Quick Start.

5、backend Logic design

In Django, all the logic are shown in the view.py. I will show it below.

```
def result(request):
    template=loader.get_template('lib/result.html')
    mytext=request.POST['text']

    content = {}

    Booklist1 = Book.objects.filter(title__icontains=mytext);
    Booklist2=Book.objects.filter(ISBN__icontains=mytext);
    Authorlist=AUTHORS.objects.filter(Name__icontains=mytext);
    #
    Book_AUTHORSList=Book_AUTHORS.objects.filter(Authors_id_id__in=[q.id for q in Authorlist]);
    Booklist3=Book.objects.filter(id__in=[r.ISBN_id for r in Book_AUTHORSList]);
    #

    Booklist=Booklist1|Booklist2|Booklist3;
    content['books']=Booklist;
    content['authors'] = getAuthors(content['books']);
    content['isAvail']=isAvail(content['books'])
    content['total']=zip(content['books'],content['authors'],content['isAvail']);
    return HttpResponse(template.render(content,request))


def getAuthors(book):
    authors = [];
    for b in book:
        temp1 = Book_AUTHORS.objects.filter(ISBN_id=b.id);
        temp2 = ''
        for t in range(len(temp1)):
            temp3 = AUTHORS.objects.get(id=temp1[t].Authors_id_id)
            if t != 0:
                temp2+=' , '
            temp2 = temp2+temp3.Name;

        authors.append(temp2)

    return authors;


def isAvail(book):
    isAvail=[];
    for b in book:
        if BOOK_LOANS.objects.filter(ISBN_id=b.id).filter(Date_in=None).exists():
            isAvail.append(False);
        else:
            isAvail.append(True)
    return isAvail;
```

Here is the logic to return all the books we search to the templates. We receive the word that the user type by using request.POST method, and cause the content that the user type can be anthor_name, title, or ISBN, we should search it in the database and union together. Then we should get the corresponding Authors by using getAuthors, and corresponding isAvail to decide if the book is still available for borrow.



```
def popup(request,book_id):
    book = Book.objects.filter(id=book_id)[0];
    context={'book':book};
    template = loader.get_template('lib/popup.html')
    return HttpResponse(template.render(context,request))

def popup_response(request,book_id):
    card_id = request.POST['card_id']
    isBorrowed=BOOK_LOANS.objects.filter(ISBN_id=book_id).filter(Date_in=None).exists();
    if isBorrowed:
        messages.add_message(request, messages.ERROR, "You can't borrower the book because book is not available")
        return redirect('/lib')
    else:
        hasBorrower = BORROWER.objects.filter(id=card_id).exists()
        if hasBorrower and len(BOOK_LOANS.objects.filter(Card_id_id=card_id).filter(Date_in=None))<=2:
            BOOK_LOANS.objects.create(Card_id_id=card_id,ISBN_id=book_id,Date_out=timezone.now(),Due_date=timezone.now()
            borrower=BORROWER.objects.filter(id=card_id)[0];
            book=Book.objects.filter(id=book_id)[0];
            context={'borrower':borrower,'book':book}
            template = loader.get_template('lib/popup_response.html')
            return HttpResponse(template.render(context,request))
        elif hasBorrower:
            messages.add_message(request, messages.INFO, "You can't borrower the book because you already borrow 3 books"
            return redirect('/lib')
        else:
            messages.add_message(request, messages.INFO, "You can't borrower the book because you are not yet a borrower"
            return redirect('/lib')
```

Check out:

Popup and popup_response() can be used to check out. Here I will explain the popup_response(). First, we should check that if the book has been borrowed by using the book_id, which is passed by click the corresponding button for that book. And if the book is not there, we add message saying that the book is not available and we can't borrow it.

Second, if the book is available ,we should check if the card_id is in our database (the Boolean hasBorrower).If we have borrower and he/she have borrowed less than 2 book(which didn't return yet) and then he/she can borrow, else if he have been borrowed 3 books and can't borrowed one more.

```
def checkin(request):
    template = loader.get_template('lib/checkin.html')
    return HttpResponse(template.render(None,request))
def checkinresult(request):
    text = request.POST['text']
    option=request.POST['inlineRadioOptions']

    content = {};
    if(option=="BOOKS.book_id"):
        book_loan=BOOK_LOANS.objects.filter(ISBN_id=text,Date_in=None)
    elif(option == "BORROWER.card_no"):
            book_loan = BOOK_LOANS.objects.filter(Card_id_id=text,Date_in=None)
    elif (option == "BORROWER name"):
        borrowers=BORROWER.objects.filter(Bname__icontains=text);
        book_loan=[];
        for b in borrowers:
            book_loansub=BOOK_LOANS.objects.filter(Card_id_id=b.id,Date_in=None)
            for q in book_loansub:
                book_loan.append(q);
    template = loader.get_template('lib/checkinresult.html')
    content['book_loan'] = book_loan;
    return HttpResponse(template.render(content, request))

def checkinfinish(request):
    book_idCheck = request.POST.getlist('book_id')
    if book_idCheck:
        for b in book_idCheck:
            BOOK_LOANS.objects.filter(ISBN=b).update(Date_in=timezone.now())
        messages.add_message(request, messages.INFO, "Check in succeed!");
        return redirect('/lib')
    else:
        return HttpResponse("You didn't select at all");
```

Check in:

The Check in logic is quite straightforward. First, we search for the BOOK_LOANS Record according to the book_id, card_id or borrower name. Then, once we select the corresponding book, the book_id is passed and using the book_id we can find the corresponding record in BOOK_LOANS because BOOK_LOANS has a foreign key ISBN which is the primary key in BOOK.

```
def createborrower(request):
    template = loader.get_template('lib/CreateBorrower.html')
    return HttpResponse(template.render(None, request));

def newborrower(request):
    ssn = request.POST['ssn']
    name=request.POST['name']
    address=request.POST['address']
    phone=request.POST['phone']
    if ssn and name and address:
        if BORROWER.objects.filter(ssn=ssn).exists():
            return HttpResponse("Your SSN has been used, try another one ");
        else:
            if BORROWER.objects.filter(ssn=ssn).exists():
                return HttpResponse("SSN has already taken");
            else:
                BORROWER.objects.create(ssn=ssn, Bname=name, Address=address,Phone=phone)
                return HttpResponse("Create new Borrower");
    else:
        return HttpResponse("You haven't enter all fields");
```

The logic to create new borrower is straightforward too. We need to judge if ssn, name, address is empty or not. If it is empty, we return some message saying we didn't enter all fields.

Then we should judge if it is a new SSN, if not , then we will show that "The ssn has been used." If all the fields are ok, we will create a new record for our database.

Actually we set all field to be varchar, so whatever we type, the database will accept no matter if it is the right format. I can improve by using some more logic latter.

```
def updatefine(request):
    book_loan=BOOK_LOANS.objects.all();
    for b in book_loan:
        fine=FINES.objects.filter(Loan_id_id=b.id).exists();
        if not fine:
            if (not b.Date_in and timezone.now() > b.Due_date):
                FINES.objects.create(Fine_amt=0.25 * ((timezone.now() - b.Due_date).days + 1), Paid=False,
                                     Loan_id_id=b.id);
            if ((b.Date_in and b.Date_in > b.Due_date)):
                FINES.objects.create(Fine_amt=0.25 * ((b.Date_in - b.Due_date).days + 1), Paid=False, Loan_id_id=b.id);
        else:
            if not FINES.objects.get(Loan_id_id=b.id).Paid:
                FINES.objects.filter(Loan_id_id=b.id).update(Fine_amt=0.25 * ((timezone.now() - b.Due_date).days + 1))
    return HttpResponse("Update fine finish");
```

We provide a button for update the fine.

First, All the fine should be produced after we make a book loan. So we search every record in the book_loan and their corresponding fine using Loan_id. If we can't fine the corresponding fine, that means before today there are no any fine , we need to check if there are so book that pass the due date today.(if the book not return, compare timezone.now() and Due_date). In case of the library do not update every day , I provide another "if" for those has been returned but still pass the due_date.(If the library update every day, then we actually don't need it.)

```
def showfine(request):
    fine = FINES.objects.filter(Paid=False);
    book_loan=[]
    for f in fine:
        bl=BOOK_LOANS.objects.get(id=f.Loan_id_id);
        book_loan.append(bl)
    total = zip(fine, book_loan);
    template = loader.get_template('lib/showfine.html')
    finebycardid=FINES.objects.values('Loan_id__Card_id').filter(Paid=False).annotate(Sum('Fine_amt'));
    book_loans=[];
    card_id=[];
    amount=[];

    for f2 in finebycardid:
        bl = BOOK_LOANS.objects.filter(Card_id=f2['Loan_id__Card_id']).first();
        book_loans.append(bl);
        card_id.append(f2['Loan_id__Card_id']);
        amount.append(f2['Fine_amt__sum']);

    total2 = zip(book_loans,card_id,amount);
    context={'total':total,'total2':total2}
    return HttpResponse(template.render(context, request));



def makepaymentbyloanid(request,loan_id):
    book_loan=BOOK_LOANS.objects.get(id=loan_id);
    if book_loan.Date_in:
        fine=FINES.objects.get(Loan_id_id=book_loan.id)
        fine.Paid=True;
        fine.save();
        return HttpResponse("Succeed in making payment");
    else:
        return HttpResponse("Can't pay because not yet return the book");
    # return HttpResponse(template.render(None, request));

def makepaymentbycardid(request,card_id):
    book_loan = BOOK_LOANS.objects.filter(Card_id=card_id);
    if canpay(book_loan):
        for bl in book_loan:
            fine = FINES.objects.filter(Loan_id_id=bl.id).filter(Paid=False);
            if fine:
                for f in fine:
                        f.Paid=True;
                        f.save();
        return HttpResponse("We pay it off");
    else:
        return HttpResponse("Can't pay because not yet return all your  fined (all the expire book)books");
```

As for the showfine to show I provide two ways:

1、 show every fine record that haven't been paid yet
2、 show fine record group by card_id

So according to that, we can make payment using two ways:

1、 make payment for one fine record.
2、 Make payment for certain card_id. (Can't pay it if you can have even one book that expire that haven't returned yet.)