Normalized Relational Schema (BCNF)

All the tables are in BCNF form. When we converted our ER model to schema we made them in such a way that they satisfy BCNF properties i.e every attribute is not dependent on any proper subset of the candidates key. Each table contains non-trivial FDs are of the form a->b where a is super-key so we choose BCNF. Since we don't have any multivariate attribute so we don't need 4NF any table.

```
create table book_shelf (
        shelf_id varchar(20),
        capacity int not null default 0,
        rem_capacity int,
        primary key (shelf_id)
);

create table authors (
        author_id varchar(20),
        name varchar(30) not null default '',
        primary key (author_id)
);

create table publisher (
        publisher_id varchar(20),
        name varchar(20) not null default '',
        street_number varchar(20),
        building_number varchar(20),
        city varchar(20) not null default '',
        state varchar(20) not null default '',
        zip_code int not null default 0,
        primary key (publisher_id)
);

create table books (
        isbn varchar(20),
        copy_number int default 1,
        shelf_id varchar(20),
        status varchar(20),
        primary key(isbn, copy_number),
        foreign key (isbn) references books_info on delete cascade,
        foreign key (shelf_id) references book_shelf on delete cascade
);

create table books_info (
        isbn varchar(20),
        title varchar(50) not null default '',
        year_of_publication date not null default '00-00-0000',
        publisher_id varchar(10),
        primary key(isbn),
        foreign key (publisher_id) references publisher on delete cas
        cade
);

create table books_authors (
        author_id varchar(20),
        isbn varchar(20),
        primary key (author_id, isbn),
        foreign key (author_id) references authors on delete cascade,
        foreign key (isbn) references books_info on delete cascade
);

create table users (
        user_id varchar(20),
        user_name varchar(20) not null,
        password varchar(20) not null,
        email_id varchar(40) not null,
        house_number varchar(20),
        street_number varchar(20),
```

```sql
        city varchar(20) not null default '',
        state varchar(20) not null default '',
        zip_code int not null default 0,
        total_fine_pending float default 0,
        primary key (user_id)
);

create table librarian (
        user_id varchar(20),
        librarian_id varchar(20),
        working_hours date not null,
        primary key (user_id, librarian_id),
        foreign key (user_id) references users on delete cascade
);

create table student (
        user_id varchar(20),
        student_id varchar(20),
        primary key (user_id, student_id),
        foreign key (user_id) references users on delete cascade
);

create table faculty (
        user_id varchar(20),
        faculty_id varchar(20),
        primary key (user_id, faculty_id),
        foreign key (user_id) references users on delete cascade
);

create table review (
        user_id  varchar(20),
        isbn varchar(20),
        rating float default 0,
        review varchar(1000) default '',
        genre varchar(10) not null,
        primary key (user_id, isbn),
        foreign key (user_id) references users on delete cascade,
        foreign key (isbn) references books_info on delete cascade
);

create table books_on_hold (
        user_id varchar(20),
        isbn varchar(20),
        hold_date date,
        primary key (user_id, isbn, hold_date),
        foreign key (user_id) references users on delete cascade,
        foreign key (isbn) references books_info on delete cascade
);

create table personal_shelf (
        user_id varchar(20),
        isbn varchar(20),
        reading_status varchar(20) not null,
        primary key (user_id, isbn),
        foreign key (user_id) references users on delete cascade,
        foreign key (isbn) references books_info on delete cascade
);

create table books_on_loan (
        user_id varchar(20),
        copy_number int,
        isbn varchar(20),
        due_date date,
        issue_date date,
        returned_date date,
        is_lost int not null default 0,
        fine_paid int not null default 0,
        last_reminder_date date,
        primary key (user_id, isbn, copy_number due_date, issue_date),
        foreign key (user_id) references users on delete cascade,
        foreign key (isbn, copy_number) references books on delete cascade,
        foreign key (isbn) references books_info on delete cascade
);

create table friends (
        user_id varchar(20),
        friend_id varchar(20),
        status varchar(10) not null default 'pending',
        primary key (user_id, friend_name),
        foreign key (user_id) references users on delete cascade,
        foreign key (friend_id) references users(user_id) on delete cascade
);

create table rules (
        rule_id varchar(10),
        min_days int not null default 0,
        max_days int,
        fine float not null default 0,
        primary key (rule_id)
);
```

```
create function update_fine() returns trigger as $update_fine$
          begin
                        update users set total_fine_pending =
select(sum(fine_paid) from books_on_loan where user_id = current.user_id and
returned_date is null and current_date() > due_date);
                        return null;
          end;
        $update_fine$ language plpgsql;

create trigger update_fine every day on users
        for each row execute procedure update_fine();
```

```
Assumptions here :
        1. "current" refers to current row of the table
        2. Below trigger executes for each row of books_on_hold table
        3. Below trigger executes everyday
```

```
create function book_unhold() returns trigger as $book_unhold$
          begin
                if (select (to_days(current_date()))-to_days(returned_date) from
books_on_loan where current.isbn=isbn and current.copy_number=copy_number order
by returned_date limit 1) >10 then
                        if(select count(user_id) from books_on_hold where
current.isbn=isbn and current.copy_number=copy_number and user_id in (select
user_id from student)) <2 then
                                update books set status="on_shelf" where
current.isbn=isbn and current.copy_number=copy_number
                        end if;
                        delete from books_on_hold where current.user_id =
user_id and current.isbn=isbn and current.copy_number=copy_number;
                        return null;
                end if;
          end;
        $check_event$ language plpgsql;

create trigger check_event every day on books_on_hold
          for each row execute procedure book_unhold();
```

```
create function allow_loan() returns trigger as $allow_loan$
          begin
                if (new.user_id in (select user_id from users where
total_fine_pending > 1000)) then
                        RAISE EXCEPTION 'Your total_fine is more than 1000,
please pay before any other loan';
                        return null;
                end if;
                return new;
          end;
        $allow_loan$ language plpgsql;

create trigger allow_loan before insert on books_on_loan
        for each row execute procedure allow_loan();
```

```
create function withdraw_lmt() returns trigger as $withdraw_lmt$
          begin
                if (new.user_id in (select user_id from students where (select
count(user_id) from books_on_loan where user_id=new.user_id and returned_date is
not null) =3) then
                        RAISE EXCEPTION '3 books limit reached';
                        return null;
                end if;
                return new;
          end;
        $allow_loan$ language plpgsql;

create trigger withdraw_lmt before insert on books_on_loan
        for each row execute procedure withdraw_lmt();
```

```
create function update_email_date() returns trigger as $update_email_date$
          begin
                if ((to_days(current_date())-to_days(current.due_date))>31 &&
(to_days(current_date())-to_days(current.last_reminder_date)) > 15 ||
current.last_reminder_date is null && returned_date is null)then
                                update books_on_loan set
last_reminder_date=current_date();
                                send_mail(user_id of email);
                        return null
                end if;
          end;
        $update_email_date$ language plpgsql;

create trigger update_email_date every day on books_on_loan
        for each row execute procedure update_email_date();
```

Assumptions

1. A Librarian is a faculty, i.e., librarian can also take books from library.
2. There is a mechanism by which triggers are executed regularly.
3. Function current_date() gives the current date.
4. User should enter the genre of the book, whenever he/she reviews the book. Genre uniquely determines the type of book and will be used for suggesting the books to the user (based on his/her highest rated genre).
5. There exists a method which sends the email in and then triggers the `update_email_date()` every day.
6. There exists a mechanism that daily updates the fine_paid on books_on_loan for the issued books, by the amount as per the library_rules.
7. Trigges with every day as the execution constraint, somehow executes at the starting of each day.
8. Books with same ISBN number are given unique copy_id to distinguish them. If library have single copy of some book then default copy_number 1 is allocated to that book.

Thank You !!!