Solution to Exercise 3b (Schema).

There are two main possibilities:

a) Use a relation \textit{Loan} which only stores the historical loans.

b) Use a relation \textit{Loan} which stores current and historical loans.

Both of them are OK
Solution to Exercise 3b (Schema).

USER(ucode, name, address, tel, total_loans)
  PK= {ucode}

BOOK(bcode, title, topic, ucode, loan_date)
  PK= {bcode}
  NNV= {topic}
  FK= {ucode} → USER

AUTHOR(bcode, author)
  PK= {bcode, author}
  FK= {bcode} → BOOK

LOAN(ucode, bcode, loan_date, return_date)
  PK= {bcode, loan_date}
  NNV= {ucode, loan_date, return_date}
  FK= {bcode} → BOOK
  FK= {ucode} → USER
Solution to Exercise 3b (Schema).

USER(ucode, name, address, tel, total_loans)
   PK= {ucode}

BOOK(bcode, title, topic)
   PK= {bcode}
   NNV= {topic}

AUTHOR(bcode, author)
   PK= {bcode, author}
   FK= {bcode} → BOOK

LOAN(ucode, bcode, loan_date, return_date)
   PK= {bcode, loan_date}
   NNV= {ucode, loan_date}
   FK= {bcode} → BOOK
   FK= {ucode} → USER
Solution to Exercise 3b (Schema).

R1:
LX:BOOK
∀LX ((BOOK(LX) ∧ (¬null(LX.topic))
    → ( (LX.topic = ‘Physics’) ∨
        (LX.topic = ‘Optics’) ∨
        (LX.topic = ‘Mechanics’) ∨
        (LX.topic = ‘Electricity’)))

R2:
PX:LOAN
∀PX ((LOAN(PX) ∧ (¬null(PX.return_date))
    → (PX.loan_date ≤ PX.return_date))

There are other constraints that should be met, e.g. there cannot be two loans of the same book with overlapped dates.
Solution to Exercise 3b (Schema).

create table User2(
    ucode char(5) constraint cp_User primary key deferrable,
    name varchar2(60),
    address varchar2(50),
    tel varchar2(20),
    total_loans number(3) default 0 not null deferrable);

create table Book(
    bcode char(5) constraint cp_Book primary key deferrable,
    title varchar(100),
    topic varchar(15) constraint null_topic not null deferrable,
    constraint c_topic
        check (topic in ('physics','electricity','mechanics','optics'))
        deferrable);
create table Author(  
    bcode char(5) constraint ca_author_Book references Book(bcode) 
               deferrable,  
    author varchar2(40),  
    constraint cp_Author primary key(bcode, author) deferrable);  
create table Loan(  
    ucode char(5) constraint ca_pre_User2 references User2(ucode) deferrable 
               constraint null_ucode not null deferrable,  
    bcode char(5) constraint ca_pre_Book references Book(bcode) deferrable,  
    loan_date date constraint null_loan_date not null deferrable,  
    return_date date,  
    constraint cp_loan primary key(bcode,loan_date) deferrable,  
    constraint dates check ((return_date is null) or (return_date>loan_date)) 
               deferrable);
Solution to Exercise 3b (Derived attribute).

The attribute “total_loans” must be maintained...

Solution: TRIGGERS
**Solution to Exercise 3b (Derived attribute).**

EVENTS which affect “total_loans”:

<table>
<thead>
<tr>
<th>EVENT</th>
<th>Table</th>
<th>Attribute</th>
<th>CONDITION</th>
<th>ACTION</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Instruction</strong></td>
<td><strong>Table</strong></td>
<td><strong>Attribute</strong></td>
<td><strong>CONDITION</strong></td>
<td><strong>ACTION</strong></td>
</tr>
<tr>
<td>INSERT</td>
<td>LOAN</td>
<td>-</td>
<td>null(return_date)</td>
<td>+1</td>
</tr>
<tr>
<td>UPDATE</td>
<td>LOAN</td>
<td>return_date</td>
<td>from null to not null</td>
<td>-1</td>
</tr>
<tr>
<td>UPDATE</td>
<td>LOAN</td>
<td>ucode</td>
<td>from not null to null</td>
<td>+1 (or forbid)</td>
</tr>
<tr>
<td>DELETE</td>
<td>LOAN</td>
<td>-</td>
<td>null(return_date)</td>
<td>-1 (or forbid)</td>
</tr>
<tr>
<td>INSERT</td>
<td>USER2</td>
<td>-</td>
<td>if &lt;&gt; 0</td>
<td>forbid</td>
</tr>
<tr>
<td>UPDATE</td>
<td>USER2</td>
<td>total_loans</td>
<td>if wrong</td>
<td>forbid</td>
</tr>
</tbody>
</table>
Solution to Exercise 3b (Derived attribute).

We can add a by-default value for total_loans in the definition of the table “USER2” as follows:

```sql
    total_loans DEFAULT 0
```

And, more importantly, we must create the following triggers to maintain the derived attribute “total_loans”.

The following trigger controls the insertion of new LOANs:

CREATE TRIGGER T1
AFTER INSERT ON LOAN
FOR EACH ROW
BEGIN
    IF :new.return_date IS NULL THEN
        UPDATE USER2 SET total_loans = total_loans + 1
        WHERE ucode = :new.ucode;
    END IF;
END;
Solution to Exercise 3b (Derived attribute).

The following trigger controls the modification of the attribute “return_date” in LOAN (a book is borrowed or returned):

CREATE TRIGGER T2
AFTER UPDATE OF return_date ON LOAN
FOR EACH ROW
BEGIN
    IF :new.return_date IS NULL AND :old.return_date IS NOT NULL THEN
        UPDATE USER2 SET total_loans = total_loans +1 WHERE ucode = :new.ucode;
    ELSE IF :new.return_date IS NOT NULL AND :old.return_date IS NULL THEN
        UPDATE USER2 SET total_loans = total_loans -1  WHERE ucode = :old.ucode;
    END IF;
END IF;
END;
Solution to Exercise 3b (Derived attribute).

The following trigger controls the modification of the attribute “ucode” in LOAN (a LOAN is changed to a different USER):

CREATE TRIGGER T3
AFTER UPDATE OF ucode ON LOAN
FOR EACH ROW
BEGIN
    IF :new.return_date IS NULL AND :old.return_date IS NULL THEN
        UPDATE USER2 SET total_loans = total_loans +1 WHERE ucode = :new.ucode;
        UPDATE USER2 SET total_loans = total_loans -1 WHERE ucode = :old.ucode;
    END IF;
END;
Solution to Exercise 3b (Derived attribute).

The following trigger controls the deletion of a LOAN (although another option is to forbid this: this can only be done if the book has been returned):

CREATE TRIGGER T4
AFTER DELETE ON LOAN
FOR EACH ROW
BEGIN
    IF :old.return_date IS NULL THEN
        UPDATE USER2 SET total_loans = total_loans -1
        WHERE ucode = :old.ucode;
    END IF;
END;
Solution to Exercise 3b (Derived attribute).

The following trigger controls that the number of loans must be 0 when inserting a new USER:

CREATE TRIGGER T5
AFTER INSERT ON USER2
FOR EACH ROW
BEGIN
  IF :new.total_loans <> 0 then
    RAISE_APPLICATION_ERROR(-20000, 'When a USER is inserted, the number of loans s/he has must be 0.');
  END IF;
END;
The following trigger controls that the number of loans cannot be modified:

```sql
CREATE TRIGGER T6
AFTER UPDATE OF total_loans ON USER2
FOR EACH ROW
DECLARE
N NUMBER;
BEGIN
SELECT COUNT(*) INTO N
FROM LOAN P
WHERE P.return_date IS NULL AND P.ucode = :new.ucode;
IF N <> :new.total_loans THEN
   RAISE_APPLICATION_ERROR(-20000, 'Inconsistent modification. ');
END IF;
END;
```

This trigger is problematic because it can be indirectly invoked by other triggers.
Additionally it has “mutant tables”.

Another option would be to forbid the modification of total_loans to every user:

```sql
REVOKE UPDATE(total_loans) TO ALL;
```

Solution to Exercise 3b (Derived attribute).