CSE-A1200 Databases

Project Report

Database for Helsinki area public library system

Kamyar Hasanzadeh

kamyar.hasanzadeh@aalto.fi **(336295)**

Anna Gorodetskaya

anna.gorodetskaya@aalto.fi **(336172)**

Spring 2014

Part 1. E/R Data Model and Relational Data Model

Tasks

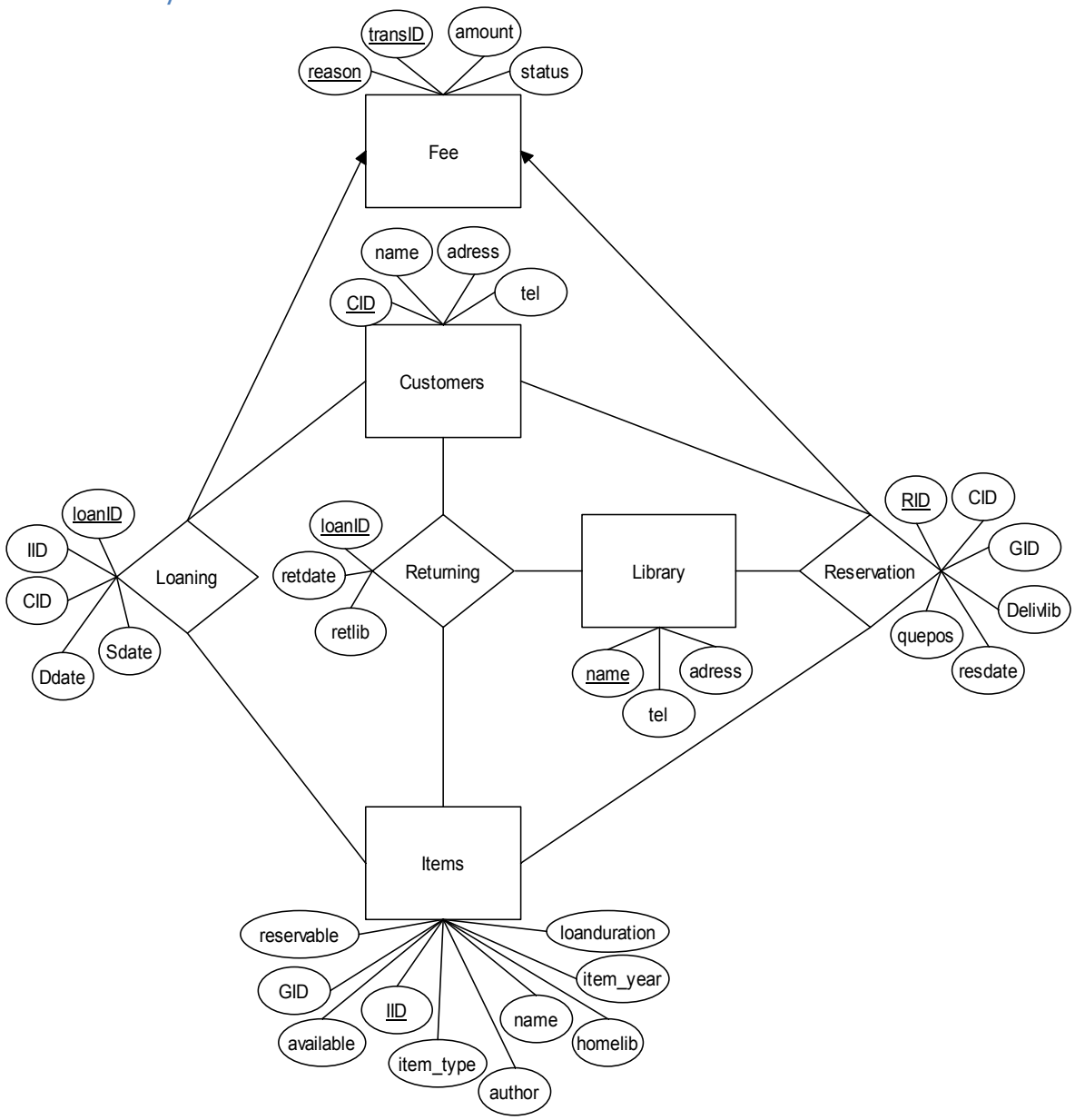
• Draw an E/R-diagram based on the information in the following sections.

• Convert the E/R-diagram to the relational data model. Present the schemas of the relations and underline the attributes which form the key for each relation.

• Provide answers to the following questions: What are the functional dependencies of the database? Are there any form of redundancy or other anomalies in the database structure? Is your database in Boyce-Codd Normal Form? If it is not, please use the decomposition algorithm.

Solution

1. E/R Data model



2. Relational model

Our database consists of several entities: libraries, customers, items and fees.

Each library has a unique name with address and phone number that is recorded in library relational table.

Information about all the items from all the libraries is collected in one relational table. Each item has its unique ID and information about its name, type, author, year, and home library. The items with the same name and type have the same group ID (GID). Additionally, it is possible to know if the item is available in exact moment and if it is reservable or not (consequently available and reservable attributes). Some items can be loaned but not reserved and in opposite. For each item also the duration of loan is recorded.

All customers have their unique customer ID, information about their names, home address and phone number.

Our relational model saves the information about following operations in library system: loaning, reserving, collecting fees and returning.

For each loan with unique ID the information about customer and borrowed item is recorded together with the start and the due date of the loan and information about returning status of the item (returned or not). In case if the due date is over than today`s date and the returned status is 'not returned', the customer is started to be charged for overdue loan and information about fee is recorded into Fee relational table.

Each reservation is also recorded with unique reservation ID, customer ID and group ID: because for customer the title of the item is more important rather than the exact item itself. So the customer can reserve any available in exact moment item from any library and point out the most convenient library for picking out the item (where the item will be delivered from home library). Additionally customer gets the position in the queue of people who wants to reserve the item with the same name. The date of reservation is also fixed and the fee for reservation service is recorded for reserving customer in Fee relational table.

In case of returning, the information about loan ID, returning date and returning library is recorded.

The relational table Fee contains information about the transaction ID, reason of fee (whether it is fee for loan overdue or for reservation), amount of money and status of fee (was it paid or not). Until the customer hasn`t paid for overdue loan and hasn`t returned the book to the library, the sum of fee is growing every day by a fixed amount.

Our relational model consists of the following relational tables (the keys for each table are underlined):

Library (name, address, tel)

Customers (CID, name, address, tel)

Items (IID, GID, available, item_type, name, author, item_year, loanduration, reservable, homelib)

Loans (loanID, IID, CID, Sdate, Ddate, returned)

Reserve (RID, CID, GID, DelivLib, resdate, quepos)

Fee (reason, transID, amount, status)

Returning (loanID, retdate, retlib)

3. Functional dependencies

The existing functional dependences for our database are following:

Library:
name → address, tel

Customers:
CID → name, address, tel

Items:
IID → GID, available, item_type, name, author, item_year, loanduration, reservable, homelib

Loans:
loanID → IID, CID, Sdate, Ddate, returned

Reserve:
RID → CID, GID, DelivLib, resdate, quepos

Fee:
reason, transID → amount, status

Returning:
loanID → retdate, retlib

4. Redundancy and other anomalies, following BCNF.

According to (Garcia-Molina et al., 2008)[1]: "A Relation R is in BCNF if and only if: whenever there is a nontrivial FD $A_1\ A_2\ ...\ A_n \rightarrow B_1\ B_2\ ...\ B_m$ for R, it is the case that $\{A_1\ A_2\ ...\ A_n\}$ is a superkey for R".

Our relational model with its functional dependences is following the above mentioned condition that means that our relation is already in Boyce-Codd Normal Form (BCNF). This fact is eliminating the presence of redundancy or other anomalies in our case.

[1] Garcia-Molina, H.; Ullman, J. D.; Widom, J. 2008. Database Systems – The Complete Book. Second Edition, New Jersey, NY: Pearson-Prentice Hall.

Part 2. SQL

Tasks

• Define the relation schema in SQL. The schema must contain same information as the project part 1 (library database). Write the CREATE statements to the document. Use data types that are reasonable. Justify your solutions.
• Check the keys and other constraints in your database. Verify that the primary keys and foreign keys are well defined.
• Which kind of SQL queries might be typical for the library information system? Create reasonable indexing which supports the purposeful use of the database. In addition, add at least one useful view definition to your database.
• Create some use cases for the library information system. Write a description of the use case, explain which information needs to be selected/updated/inserted and write the SQL queries that are needed to fulfill the use case. The number of SQL queries written in the document must be at least 15. Try to find at least one use case where you need to use a little bit more complicated SQL query.
• Run the SQL statements (creating the tables, inserting example data to them and the SQL queries) in SQLite environment. Sqlite3 is available in Aalto IT Linux computers (at least in most of them) and you can also download it to your own computer, see www.sqlite.com. Note that there are some limitations in SQL statements implemented by SQLite.
Justify all your solutions carefully!
Note that you are not asked to write a user interface for the database. You can create the database and perform all your queries by just writing the SQL statements.

Solution

1. Creating tables

In this section we defined the schemas and created the tables. Moreover, the necessary constraints and references were included in order to assure the redundancy of the database. The following code was used for these purposes.

```
CREATE TABLE library (
name CHAR(50),
address CHAR(100),
tel char(20),
PRIMARY KEY (name)
);
CREATE TABLE customers (
CID INT,
name CHAR (50),
address CHAR (100),
tel char(20),
PRIMARY KEY (CID)
);
CREATE TABLE items(
IID INT,
GID INT,
```

```
available BOOLEAN,
item_type CHAR (20),
name CHAR(70),
author CHAR (50),
item_year INT,
loanduration INT,
reservable BOOLEAN,
homelib CHAR(50) REFERENCES library(homelib),
PRIMARY KEY (IID)
);
CREATE TABLE loans (
loanID INT,
IID INT REFERENCES items(IID),
CID INT REFERENCES customers(CID),
Sdate DATE,
Ddate DATE,
returned BOOLEAN;
PRIMARY KEY (loanID)
);
CREATE TABLE reserve (
RID INT,
CID INT REFERENCES customers(CID),
GID INT,
DelivLib CHAR (50),
resdate DATE,
quepos INT,
PRIMARY KEY (RID)
);
CREATE TABLE fee (
reason CHAR (20) CHECK(reason IN (‘reservation’,’delayed loan’),
transID INT,
amount FLOAT,
status CHAR(40) CHECK(status IN ('paid','pending')),
PRIMARY KEY (reason, transID)
);
CREATE TABLE returning (
loanID INT REFERENCES loans(loanID),
retdate DATE,
retlib CHAR (50) REFERENCES library(retlib),
PRIMARY KEY (loanID)
);
```

As it can be speculated in the SQL code above, the tables are created using the CREATE TABLE statement and the keys are defined by PRIMARY KEY. Moreover, for each attribute the desirable data type has been specified. The attributes containing strings are considered as CHAR and an appropriate length has been assigned. In cases where the string may be long (such as addresses or names of the items) a longer sized has been specified. In addition, some attributes are defined as BOOLEAN, meaning that they can only be either 0 or 1 indicating TRUE or FALSE. For example ‘reservable’ is a BOOLEAN value where 1

indicates that the item is reservable and 0 indicates that it is not reservable. Furthermore, DATE type has been assigned to the attributes which are intended to contain dates. For the attributes containing numbers an INT type has been specified in most cases. The only exception is the 'amount' which has been defined as FLOATE in order to be able to handle the case where the prices contain decimals.

In cases where a table contains the same attribute as another table, in order to assure the consistency of the values, the REFERENCES have been added. In addition, some attributes can only contain certain values. For instance the 'status' attribute in table 'fee' can only be either *paid* or *pending.* Similarly, the 'reason' attribute can only have two values as specified in the code above. In order to assure the assignment of the right values to these attributes, a constraint has been applied using the CHECK statement.

2. Automatic maintenance of the tables

In order to make the database more feasible to use, we decided to automate certain functionalities of the tables. This can potentially contribute to the consistency of the tables and its ease of use. This automation is done by using triggers. Here we will discuss each trigger's functionalities and purposes.

```
CREATE TRIGGER loansetting2 AFTER
INSERT ON loans
FOR EACH ROW BEGIN

DELETE FROM loans WHERE IID= NEW.IID  AND CID=NEW.CID
 AND ((SELECT available FROM items WHERE IID=NEW.IID)=0);


UPDATE loans SET Sdate= (SELECT date('now')) WHERE CID=NEW.CID AND IID=NEW.IID
 AND ((SELECT available FROM items WHERE IID=NEW.IID)=1);

UPDATE loans SET Ddate=  (SELECT date(julianday('now')+  (SELECT loanduration
 FROM items WHERE items.IID=NEW.IID))) WHERE CID=NEW.CID  AND IID=NEW.IID
 AND ((SELECT available FROM items WHERE IID=NEW.IID)=1);

UPDATE loans SET loanID= (SELECT COUNT (*) FROM loans)) WHERE CID=NEW.CID
 AND IID=NEW.IID  AND ((SELECT available FROM items WHERE IID=NEW.IID)=1);

UPDATE items SET available= 0 WHERE items.IID=NEW.IID  AND  ((SELECT returned FROM
loans WHERE IID=new.IID)=0);
 END;
```

This trigger handles the following functionalities:

- Prohibits loaning an item which is not available (deletes insertion).
- Updates the Starting date of loan to today's date so the user should not worry about the date.
- Calculates the due date of a loan automatically
- Assigns a unique loaned to each loan. This is important as the loaned is the key of loans table.
- Alters the availability of the item to 0 after a successful loan.

The following code is intended for the reservation table:

```
CREATE VIEW itemstatus AS
SELECT GID, reservable FROM items;

CREATE TRIGGER rs1
AFTER INSERT ON reserve
FOR EACH ROW
BEGIN
UPDATE reserve SET RID=  (SELECT COUNT (*)  FROM reserve)WHERE CID=NEW.CID  AND
GID=NEW.GID;

 UPDATE reserve  SET quepos=((SELECT COUNT (*)FROM reserve WHERE GID=NEW.GID))
WHERE CID=NEW.CID  AND GID=NEW.GID  AND ((SELECT reservable FROM itemstatus
 WHERE GID=NEW.GID)=1);

 UPDATE reserve  SET resdate = (SELECT date('now')) WHERE CID=NEW.CID
 AND GID=NEW.GID;

 DELETE  FROM reserve WHERE GID= NEW.GID  AND CID=NEW.CID  AND (
      (SELECT reservable FROM itemstatus WHERE GID=NEW.GID)=0); END;


CREATE TRIGGER rs2
AFTER
INSERT ON reserve
FOR EACH ROW
WHEN ((SELECT reservable FROM itemstatus WHERE GID=NEW.GID)=1) BEGIN
INSERT INTO fee VALUES('Reservation', (SELECT COUNT (*) FROM reserve),1,'pending');
END;
```

In the items table each copy of an item has a value as reservable or not. The value is the same for all the copies of an item and since we are doing the reservations based on the GID (items) and not copies, we needed the reservability value for each GID rather than each item.

Therefore a VIEW was used to generalize this information. Moreover, the two triggers used on this table are intended to handle the following functionalities:

- Assign an automatic and unique RID (key) to each reservation.
- Calculate the position of each a customer in the queue after reservation.
- Use today's date as the reservation date.
- Prohibit reservation if the item is not reservable.
- Charge the customer (insert into fee table) for each reservation. Reservation is a paid service.

The following triggers are applied to the returning table:

```
CREATE TRIGGER ret1
AFTER INSERT ON returning
FOR EACH ROW
BEGIN

UPDATE returning SET retdate=  (SELECT date('now'))WHERE loanID=NEW.loanID;

 UPDATE items  SET available=1 WHERE IID=  (SELECT IID  FROM loans
 WHERE loans.loanID=NEW.loanID);

 UPDATE loans  SET returned=1 WHERE loans.loanID=NEW.loanID; END;


CREATE TRIGGER ret2
AFTER
INSERT ON returning
FOR EACH ROW
when(abs((SELECT julianday('now')) - (SELECT julianday(Sdate)FROM loans
 WHERE loanID=NEW.loanID))> (SELECT loanduration FROM items WHERE IID=
(SELECT IID FROM loans WHERE loanID=NEW.loanID)))
BEGIN

INSERT INTO fee VALUES('Delayed loan', NEW.loanID, 1*(abs(
 (SELECT julianday('now')) -(SELECT julianday(Sdate) FROM loans
WHERE loans.loanID=NEW.loanID)) - (SELECT loanduration
 FROM items  WHERE IID= (SELECT IID FROM loans  WHERE loans.loanID=NEW.loanID))),
      'pending');

UPDATE loans SET returned=1 WHERE loans.loanID=NEW.loanID;

END;
```

The two triggers used on this table are intended to handle the following functionalities:

- Assign the return date to today's date automatically.
- Alter the availability of the item to 1.
- Set loan as returned.
- Check if the loan has been overdue and if so, the customer will be charged as a (number of delayed days * base fee)

Ultimately, it should also be noted that for testing, tables were imported into the database from CSV files by running the following commands in sqlite3:

```
.mode csv
.import items.csv items
.import library.csv library
.import customers.csv customers
```

Moreover, in order to add to the efficiency of searches done by users through the database, we planned to add an index. In the case of library the most common search is the name of the items therefore it will be most efficient if we index the 'name' attribute in the 'items' table. For this purpose we ran the following code:

```
CREATE INDEX itemsearch ON items (name);
```

3. Using the database

In order to insert a new loan from customer, we need the information about customer ID, item ID and condition of the book (weather it is returned and available for loan or not). The information about loan ID, starting date and due date will be inserted for a loan automatically.

We created the trigger which checks every time when the loan is inserted if this item is returned before. If it is found that this item is already loaned, database stops the following actions and loan is not possible. If the item is returned and customer can borrow it, database updates the starting date where the today`s day is inserted and due date where it inserts the date taken as a sum of today`s date and duration of a loan. Duration of a loan is taken from the items relational table and is different for each item. At the same moment database

changes the status of an item in table items by setting returned equals zero. It means - the book is on loan. The loan ID in relational table loans is updated automatically by adding 1 to counted already existed loans. However, since it is key, you can`t just leave for this field just 'null' but should write in insertion query any number that will be changed automatically later.

Every loaned item has its own loan ID. So in case if the customer takes from the library more than one item at once, (s)he gets as many loan IDs as the number of borrowed items.

For example, we want to insert a new loan for customer with ID=1029 a new item with ID=2188:

```
INSERT into loans values (1, 2188, 1029, null, null, 0)
```

After the insertion this loan in the database will look like:

```
4|2188|1029|2014-03-28|2014-04-22|0
```

In order to insert a new reservation we need to know customer ID, item group ID (it shows the ID for a group of the same items) and library where this item will be delivered. The reservation ID, date of the reservation, position of the customer in the queue will be inserted automatically. For this case we created a trigger that first counts the number of previous reservations and adds one to this number assigning the ID to a new reservation. Then it is counted the number of people in reservation list who reserved the same item and adds one to it - this is the queue position of the customer. When the position is 1 it means that the customer can borrow the book and after that customer will be deleted from the reservation list. The reservation date is set as a today`s date automatically. The trigger also checks if the item that the customer wants to borrow is reservable. If it is not reservable the reservation will be unsuccessful and there will be no insertion.

Additionally we created a trigger that adds a new insertion into a fee relational table in case of a new reservation. Since the reservation is a paid service, for each reservation case the fee for the customer who wants to reserve an item will be charged.

For example the customer with ID=1097 wants to borrow the item with group ID=38 and after that the customer with ID=1088 wants to borrow the same item these insertions will be look like following:

```
insert into reserve values (1, 1097,38, 'Library Apple',null, null)
insert into reserve values (2, 1088,38, 'Library Apple',null, null)
```

After these insertions the database will look like:

1|1097|38|Library Apple|2014-03-28|1
2|1088|38|Library Apple|2014-03-28|2

In order to return a book we need to know the information about loan ID and return library. The date of return will be inserted automatically as today`s date in one of our triggers. In this trigger when the new return is inserted, the loans table is updated - the value of 'returned' attribute gets 1- and the items table is updated - the value of 'available' attribute gets 1. It means that the item now is available and can be loaned.

The second trigger checks the difference between the due date and returning date. If it is bigger than the loan duration that means 'Delayed loan', the fee is charged from the customer (a new insertion happens in fee relational table).

For example the customer with loan ID=2 wants to return an item in Malmi Library

```
insert into returning values (2, null, 'Malmi Library')
```

After an insertion of this returning, the database will looks like this:

2|2014-03-28|Malmi Library

4. Use cases

1. Updating tables

We want to update the address information of a particular customer. The person's name is James Butt and his customer ID (CID) is 1001. The following statement does this:

```
UPDATE customers set Address='Jamerantaival 11 A' where CID=1001;
```

2. Transportation of items

We are interested to find out about the transportations that need to be done in the cases that the customers returns the item to a library other than its home library. The following query can provide this information:

```
SELECT returning.loanID,homelib,retlib
FROM returning, loans,items
WHERE returning.loanID=loans.loanID AND loans.IID=items.IID AND homelib<>retlib;
```

The output of this query is:

2|Lauttasaari Library |Malmi Library

3. In order to see a list of customers who have previously borrowed an item we can use the following query. The results are ordered by the due date of each loan.

```
SELECT CID
from loans
where IID = 1023 AND returned = 1
order by Ddate;
```

The output of this query is:

```
1001|James Butt|Jamerantaival 11 A|504-621-8927
1002|Josephine Darakjy|4 B Blue Ridge Blvd|810-292-9388
1003|Art Venere|8 W Cerritos Ave #54|856-636-8749
1004|Lenna Paprocki|639 Main St|907-385-4412
```

4. In case we are interested to know the number of copies existing of an item, we can make the following query.

```
SELECT COUNT (DISTINCT IID)
FROM items
WHERE name= 'The Story of Art';
```

However, in this database a group ID (GID) to the similar items. Therefore another way would be to make the following query:

```
SELECT COUNT (DISTINCT IID) FROM items WHERE GID=159;
```

The result of both queries is:

```
2
```

5. In order to find out the total amount of debt for each customer we can make the following query:

```
CREATE VIEW resdebt AS
SELECT TOTAL(amount) AS fee1,transID
FROM fee
WHERE reason= 'Reservation' AND transID=(SELECT RID FROM reserve WHERE CID
= 1097) AND status='pending';

CREATE VIEW duedebt AS
SELECT TOTAL(amount) AS fee2, transID
FROM fee
WHERE reason= 'Delayed loan' AND transID=(select loanID from loans where CID =
1097) AND status= 'pending';

SELECT fee1+fee2
FROM resdebt,duedebt;
```

The above query calculates the total amount of pending fees of customer whose CID is1097. TOTAL has been used instead of SUM in order to overcome the cases where we have null values (for instance the customer has no reservation debt). The outcome of the query in this case is:

1

6. In case we are interested in having a list of items which have been composed or written by a specific author, we can make the following query:

```
SELECT name
FROM items
WHERE author = 'Tegan Arceo';
```

The result of this query is a list of all items written or composed by 'Tegan Arceo' as below:

A History of Historical Writing

7. In case in case we are interested in having a list of all reservable items we can make the following simple query:

```
SELECT DISTINCT name
FROM items
WHERE reservable = 1;
```

The result is a long list of items from which only a instant is presented here:

An Account of My Hut
Analects
Annals. Histories
Anthology of Chinese Literature
Apology. Crito. Phaedo
Autobiography of Benvenuto Cellini
Autobiography. Poetry and Truth from Mu Own Life
Buddhist Scriptures

8. In order to identify the home library of an item we can simply make the following query:

```
SELECT homelib
FROM items WHERE IID=2001;
```

The result of this query is:

Arabianranta Library

9. A library manager can find out about the overdued loans and the customers by making the following query:

```
SELECT loanID, CID
FROM loans
WHERE DATE('now)> Ddate;
```

The result would be:

```
1|1010
```

10. In some cases the libraries need to know about the loans which are about to be overdue so that they can for example send a reminder to the related customer. The following query can find a list of loans and customers which have less than two days left to their due date:

```
select loanID, CID from loans where (select julianday(Ddate)  from loans where Ddate
not null -julianday('now') )< 2;
```

The result is as following:

```
1|1011
3|1015
```

11. In many services involving payments, customers are interested in viewing a log or record of their previous payments. Thus by making the following the query one can get a list of all paid transactions:

```
create view resdebt as
select total(amount) as fee1,transID
from fee
where reason= 'Reservation' AND transID=(select RID from reserve where CID = 1097)
and status='paid';

create view duedebt as
select total(amount) as fee2, transID
from fee
where reason= 'Delayed loan' AND transID=(select loanID from loans where CID =
1097) and status= 'paid';

select fee1+fee2
from resdebt,duedebt;
```

12. If we want to know which customers are the most active, or another words - make loans most frequently, we can check the loan relational table and group all the loans by customers in descending order:

```
select CID
from loans
group by CID
order by count(loanID) desc;
```

The result is following:

```
1010
1003
```

```
1029
1050
```

13.If some customer or worker of library wants to define the number of people in a queue for specific item group, it is possible to write the following query:

```
select max(quepos)
from reserve
where GID= 38;
```

The result if following:

```
2
```

14. A customer may want to know about his or her location in the reservation queue of an item. For this purpose, we can use the following query:

```
SELECT quepos
FROM reserve
WHERE CID = 1010 AND GID=38;
```

The result is an integer indicating the queue position:

```
1
```

15. If the workers of library want to know about the activity in their library, for example how many loans were done during the last month (approximately 30 days), they can apply the following query:

```
select count (loanID) from loans where (select homelib from loans natural left outer join
items) = 'Malmi Library' AND (select julianday(Sdate) from loans where Sdate not null -
julianday('now') <30);
```

Here we join tables ‘loans’ and ‘items’ on the attribute ‘item ID’ to know the home library and count the number of loans for last 30 days.

The result is following:

```
4
```

Note: All the database and source code files are available and can be provided upon request. The DB is also available on GitHub.