Homework 3

By Sara Rydell and Filip Northman

Task (P)

Task 1

As we mentioned in the comments of the last homework, we solved some minor errors by adding the attribute ISBN to the physical_book schema and we added the schema book_author with attributes ISBN and Author_ID for the junction table connection between books and authors since we had forgotten those aspects. Other than that we didn't receive feedback that we had to change anything from our previous homework.
We also don't want to make any additional changes.

Task 2

Our normalized schemas:

- Book(ISBN:integer, Title:string, Edition:string, Language:string, Publisher:string, Publication_Date:date, Prequel_ISBN:integer)
- Book_Genre(ISBN:integer, Genre:string)
- Author(Author_ID:integer, Author_Name:string)
- Book_Writer(ISBN:integer, Author_ID:integer)
- Physical_Book(Physical ID:integer, ISBN:integer, Damage:integer)
- User(User ID:integer, Full_Name:string, Email:string, Address:string)
- Student(Student_ID:integer, Programme:string)
- Admin(Admin_ID:integer, Department:string, Phone_Number:integer)
- Loan(Borrowing_ID:integer, Physical_ID:integer, User_ID:integer, Borrowing_Date:date, Max_Time:integer, Return_Date:date)
- Fine(Borrowing_ID:integer, Amount:integer)
- Transaction(Transaction_ID:integer, Borrowing_ID:integer, Amount:integer, Payment_Date:date, Payment_Method:string)

Task 3

a) A list of Physical IDs of all the books currently being borrowed.

SQL code:
SELECT physical_id FROM loan WHERE return_date IS NULL;
Algebra expression:
$\boldsymbol{\pi}_{Physical\ ID}(\sigma_{Return\ date = NULL}\ Loan)$
Motivation:
From the table Loan we select all tuples where the return date is null. This is because the books that are currently being borrowed will not have a return date. We then project the Physical IDs of these tuples to get just the Physical IDs for our wanted list, giving us our solution.

b) A list of User IDs of all the users that have not borrowed a single book yet.

SQL code:
SELECT User_ID FROM users EXCEPT SELECT user_ID FROM loan;
Algebra expression:
$\pi_{User\ ID} Users\ -\ \pi_{User\ ID} Loan$
Motivation:
We get the projection of all User IDs in the table Users as well as the projection of the User IDs present in the Loan table since those users have previously borrowed books. The difference between all User IDs and the User IDs in the Loan table will give all users that haven't yet borrowed a single book, which gives us our solution.

c) A list of Physical IDs of all the books that have not been borrowed yet.

SQL code:
SELECT Physical_ID FROM Physical_book EXCEPT SELECT Physical_ID FROM loan;
Algebra expression:
$\pi_{Physical\ ID} Physical\ book\ -\ \pi_{Physical\ ID} Loan$
Motivation:
We get the projection of all Physical IDs of all books in the library as well as the projection of all Physical IDs from the Loan table which gives us the book copies that previously have been borrowed. By taking the difference between all Physical IDs and the Physical IDs of books previously borrowed, we get the list of the Physical IDs of books that haven't been borrowed yet.

d) A list of User IDs of all users with 4 or more fines.

Algebra expression:
$\pi_{User\ ID}(\sigma_{(COUNT(User\ ID)=>4}\ (Fine \bowtie Loan))$
Motivation:

We do a natural join on the tables Fine and Loan and since they both have the mutual attribute Borrowing ID we get all the tuples in Loan that resulted in a fine. We then check if any of the User IDs from this table appears 4 or more times with the count statement and then select those tuples among all since the users with these User IDs will have 4 or more fines. After that we project the User IDs that meet this requirement which in the end gives us the wanted list.

e) A list of all user names that returned the third Harry Potter book between the year 2015 and 2020 (2015 and 2020 included).

Algebra expression:

$\pi_{Full\ name}([User \bowtie_{2015-01-01 < return\ date < 2020-12-30} Loan]$

$\bowtie [\sigma_{Physical\ ID}(Physical\ Book \bowtie_{Title = "Harry\ Potter\ and\ the\ Prisoner\ of\ Azkaban"} Book)])$

Motivation:

We use a natural join between the Physical books and the Books with the wanted title. Now since they both have the mutual attribute ISBN we get the joined tuples of Physical books that have the ISBN if our wanted book. We then select the Physical IDs from this table to use in the other part of the relation.

On the other part of the expression we do a natural join on the User table and the tuples from the Loan table where the return date is between 2015 and 2020. Here User ID is a mutual attribute and thus we will get the joined tuples for all loans made between the years 2015 and 2020.

Now when we naturally join these two expressions, the mutual attribute will be Physical ID and as a result we will get all tuples that meet all our criterias and from these tuples we project Full name to get our wanted list.

f) A list of Physical_IDs of all the books that have both Horror and Fantasy as genres published before the year 2010.

Algebra expression:

$\pi_{Physical\ ID}([Physical\ Book \bowtie_{Publication\ date < 2010-01-01} Book]$

$\bowtie [\pi_{ISBN}(\sigma_{Genre='Fantasy'\ AND\ Genre = "Horror"} Book\ genre)])$

Motivation:

We select all books that have both Fantasy and Horror as genres from the table Books. Afterwards we remove everything but the ISBN by projecting the ISBN from these books so that we don't get any duplicates from several genres.

In the other part we combined Physical Book through a natural join with tuples of the table Book where the publication dates are before 2010. Then we get all Physical books that were published before 2010.

Then we use a natural join between these two expressions, the remaining ISBNs of books having the two genres and the Physical books that were published in the correct timespan.

In this way we get all books that meet our requirements and lastly we project the Physical IDs to get the wanted list.

Task (P+)

a) **Referential integrity constraint**
Referential integrity is a constraint in the database that enforces a relationship between two tables. For instance, foreign key constraints specify that the key only can contain values that are present in the referenced primary key, and thus ensures the referential integrity of data that is joined on the two keys.

Example:
User ID being referenced in the table Student as Student ID

Relational algebra expression:
$\pi_{Student\ ID}(Student) \subseteq \pi_{User\ ID}(Users)$

Motivation:
We use this expression since for each Student ID in the table Students there must be a corresponding User ID in the table Users. This is because Student and User have a “isa” relation between each other where every unique Student ID is referencing a unique User ID.

We are using the same format as in the course book at page 60.

b) **Domain constraint**
Domain constraints are constraints that determine which data type can be added by the user to specific columns. If the input is of the incorrect value the user will be given a message that the column is not fulfilled properly.

Example:
We have the attribute ISBN in Book which we defined as domain type BIGINT.
We also have an attribute, User_ID in Users for instance, that has the domain type INT.

Relational algebra expression:
BIGINT: $\pi_{ISBN > 2^{63}-1\ AND\ ISBN < -2^{63}}(Book) = \emptyset$
INT: $\pi_{User\ ID < -2^{31} AND\ User\ ID > 2^{31}-1}(Users) = \emptyset$

Motivation:
We define the BIGINT and INT constraint in the example above. We do this by choosing ISBN and defining that if the Value of ISBN is above $2^{63} - 1$ or below -2^{63} the result will be the empty set Ø since it can’t be in this domain.
The same can be said for User ID but instead with INT and therefore no value above $2^{31} - 1$ and no value below -2^{31} are part of this domain, which will then give us the empty set Ø.

c) **Key constraint**
Key constraints specify that in all relations all primary keys must be unique and that the primary key can't be null.

Exemple:
ISBN is the primary key for the table Book.

Relational algebra expression:
$\sigma_{Book1.ISBN} = \sigma_{Book2.ISBN} \; AND \; Book1.Title \neq Book2.Title \; (Book1 \times Book2) = \emptyset$

Motivation:
We begin by constructing all pairs of Book tuples (t1, t2) we should not find a pair that have all identical attributes but a different ISBN. To construct these pairs we use the cartesian product and to search for pairs that violate this we use selection. Lastly we declare the constraint by setting the result equal to Ø. Since we are taking the product of the Book relation itself we rename at least one copy so that we get names for the attributes of our product. We name them Book1 and Book2 respectively.

We are using the same format as in the course book at page 61.