CSC 370

Activity Worksheet: Expressing Constraints in SQL

Mr. Yichun Zhao

Fall 2022

You have already completed this worksheet to convert constraints from plain English into relational algebra. Now you should convert it into a third format by writing these relational algebra constraints in SQL. As before, all seven constraints refer to a relational schema with the following structure:

BankSystem Schema

Member(member_id, social_insurance_number, date_joined)
Account(account_id, member_id, balance, credit_limit)
Transactions(transaction_id, send_account_id, recipient_account_id, data, amount)

Questions

1. Non-Negative Balances

   Constraint: No account can have a balance below zero.

   $$\sigma_{\text{balance}<0}(\text{Account}) = \emptyset$$

   Solution:

2. Within Credit Limits

   Constraint: No account can have negative balance that exceeds the credit limit.

   $$\sigma_{\text{balance}>\text{credit\_limit}*-1}(\text{Account}) = \emptyset$$

   Solution:

3. CRA Agrees With Your Member Count

   Constraint: No two members can have the same social insurance number.

$$\rho_A(\text{Member}) \bowtie_{\text{A.member\_id} \neq \text{B.member\_id AND A.sin=B.sin}} \rho_B(\text{Member}) = \emptyset$$

   Solution:

4. No Dangling Members

   Constraint: Every member must have at least one account.

$$\pi_{\text{member\_id}}(\text{Member}) \setminus \pi_{\text{member\_id}}(\text{Account}) = \emptyset$$

   Solution:

5. No Transfers Within An Account

   Constraint: Every transaction has a unique sender and recipient account.

$$\sigma_{\text{sender\_account\_id=recipient\_account\_id}}(\text{Transactions}) = \emptyset$$

   Solution:

6. No Self-Transfers

   Constraint: Every transaction is between a unique sending member_id and recipient member_id.

$$\rho_{A1}(\text{Account}) \bowtie_{\text{A1.mid=A2.mid}} \rho_{A2}(\text{Account}) \bowtie_{\text{A1.aid=said AND A2.aid=raid}} \text{Transactions} = \emptyset$$

   Solution:

7. Junior Account Limits

   Constraint: No member who joined within the past two years can have an account with a credit limit over $5000.

$$\sigma_{\text{date\_joined}>'2019-09-28'}(\text{Member}) \bowtie \sigma_{\text{credit\_limit}>5000}(\text{Account}) = \emptyset$$

   Solution:

8. Daily Limit

   Constraint: Each account is limited to three outgoing transactions per day.

   Note: This might sound very artificial, but that is because we have not yet learned about aggregation functions so we cannot create constraints on the sum of the transaction in a day.

$$\begin{array}{r}
\sigma_{\text{T1.tid}\neq\text{T2.tid}\neq\text{T3.tid}\neq\text{T4.tid}} \\
(\rho_{\text{T1}}(\text{Transactions}) \bowtie_{\text{T1.date=T2.date AND T1.said=T2.said}} \\
\rho_{\text{T2}}(\text{Transactions}) \bowtie_{\text{T1.date=T3.date AND T1.said=T3.said}} \\
\rho_{\text{T3}}(\text{Transactions}) \bowtie_{\text{T1.date=T4.date AND T1.said=T4.said}} \\
\rho_{\text{T4}}(\text{Transactions})) = \emptyset
\end{array}$$

   Solution:

Solutions

Question 1

This constraint applies only to one attribute of a table; so, we can use an attribute check constraint:

```
ALTHER TABLE Account
    MODIFY balance INT CHECK(balance >= 0);
```

Question 2

This constraint involves multiple attributes from the same table, so we need a tuple check constraint:

```
ALTHER TABLE Account
    ADD CONSTRAINT credit_limit
    CHECK(balance >= -1 * credit_limit);
```

Question 3

This is a key constraint. We have already defined a primary key for this relation; so, we wish to define an auxiliary key. Since it is only one attribute, we can simply modify that attribute to be unique:

```
ALTHER TABLE Member
    MODIFY social_insurance_number CHAR(9) UNIQUE;
```

Question 4

This is a referential integrity constraint. We can enforce it with a foreign key (with default reject or with cascade policy).

```
ALTHER TABLE Member
    ADD CONSTRAINT fk_member_account (member_id)
    REFERENCES Account(member_id);
```

Question 5

This constraint references two attributes from the same table. Thus, we can represent it with a tuple check constraint.

```
ALTHER TABLE Transactions
    ADD CONSTRAINT unique_send_recipient
    CHECK(sender_account_id <> recipient_account_id;
```

Question 6

This is a complicated constraint that involves multiple tables. We cannot represent this any better than with a whole-database assertion.

```
CREATE ASSERTION no_self_transfers
CHECK(NOT EXISTS(
    SELECT *
    FROM Transactions
        JOIN Account AS A1
            ON (A1.account_id = send_account_id)
        JOIN Account AS A2
            ON (A2.account_id = recipient_account_id
              AND A1.member_id = A2.member_id)));
```

Question 7

Again, we need multiple tables to describe this constraint: we will need to use an assertion again.

```
CREATE ASSERTION junior_credit_limit
CHECK( NOT EXISTS(
    SELECT *
    FROM Account
        NATURAL JOIN Member
    WHERE credit_limit > 5000
        AND date_joined > DATE_SUB(CURDATE(), INTERVAL 2 YEAR)));
```

Question 8

This constraint only applies to one table, but it cannot be applied to a single tuple. Thus, we still need an assertion rather than a tuple constraint.

```
CREATE ASSERTION transaction_limit_of_3
CHECK(NOT EXISTS(
    SELECT NULL
    FROM Transaction
    GROUP BY send_account_id
    HAVING COUNT(*) > 3));
```