Aufgabenblatt 4 - Musterlösung

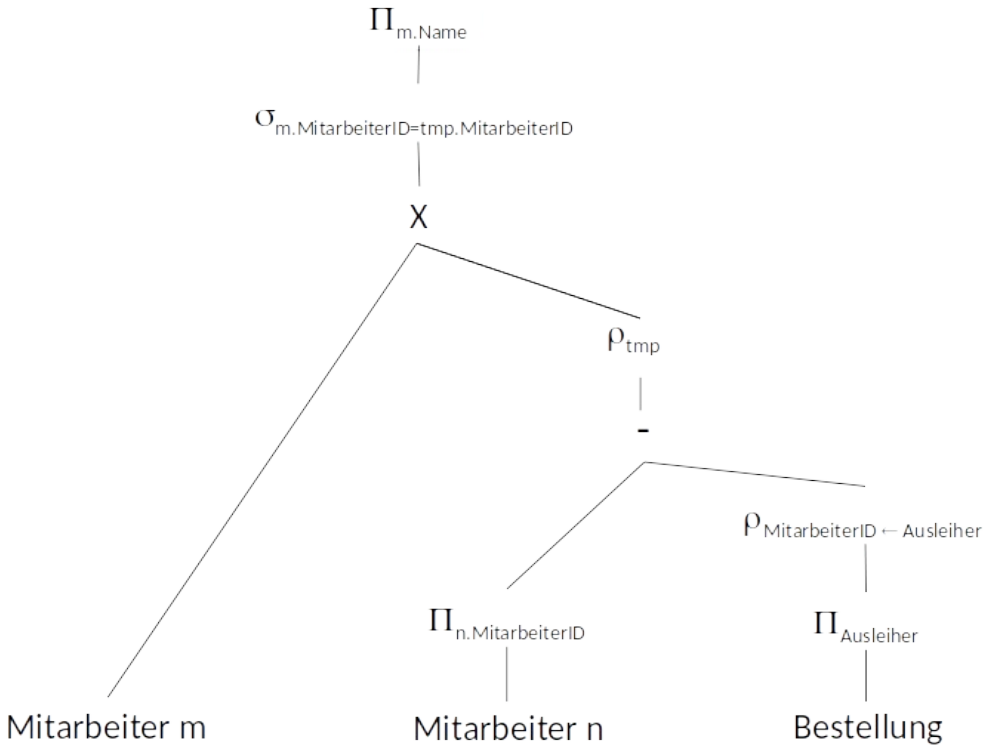
Aufgabe 1: Gegeben sei die folgende Datenbank zur Verwaltung der Bestellungen von Autos innerhalb eines Unternehmensfuhrparks. Sie besteht aus den Tabellen *Mitarbeiter*, *Auto*, *Bestellungen*, und *Rechnungen*. Die Tabellen *Auto* und *Mitarbeiter* speichern grundlegende Informationen zu den Entitäten. Autos und Mitarbeiter werden je nach ihrem Typ (einfacher Angestellter bis Chef und Kleinwagen bis Luxusauto) in verschiedene Stufen eingeteilt. Eine höhere Stufe bedeutet einen höheren Rang im Unternehmen bzw. einen höheren Wert des Autos. Wichtig bei der Einteilung ist, dass ein Mitarbeiter, der eine bestimmte Stufe erreicht hat, kein Auto bestellen darf, das nicht seiner Stufe entspricht (umgekehrt ist unkritisch).

Weiter werden alle anfallenden Rechnungen (z.B. Tanken, Autobahngebühr) gespeichert. Jede Rechnung enthält dabei einen Bezahler (ein Mitarbeiter, der nicht zwangsläufig derjenige sein muss, der das Auto bestellt hat) und wird einem Leihvorgang zugeordnet. Zusätzlich enthält jede Rechnung einen Betrag und eine Beschreibung des Rechnungsgrundes.

Alle Tabellen enthalten einen eindeutigen Identifikator (ID) für die eindeutige Zuordnung eines Datensatzes. Die Datenbank ist mit dem folgenden Script aufgebaut:

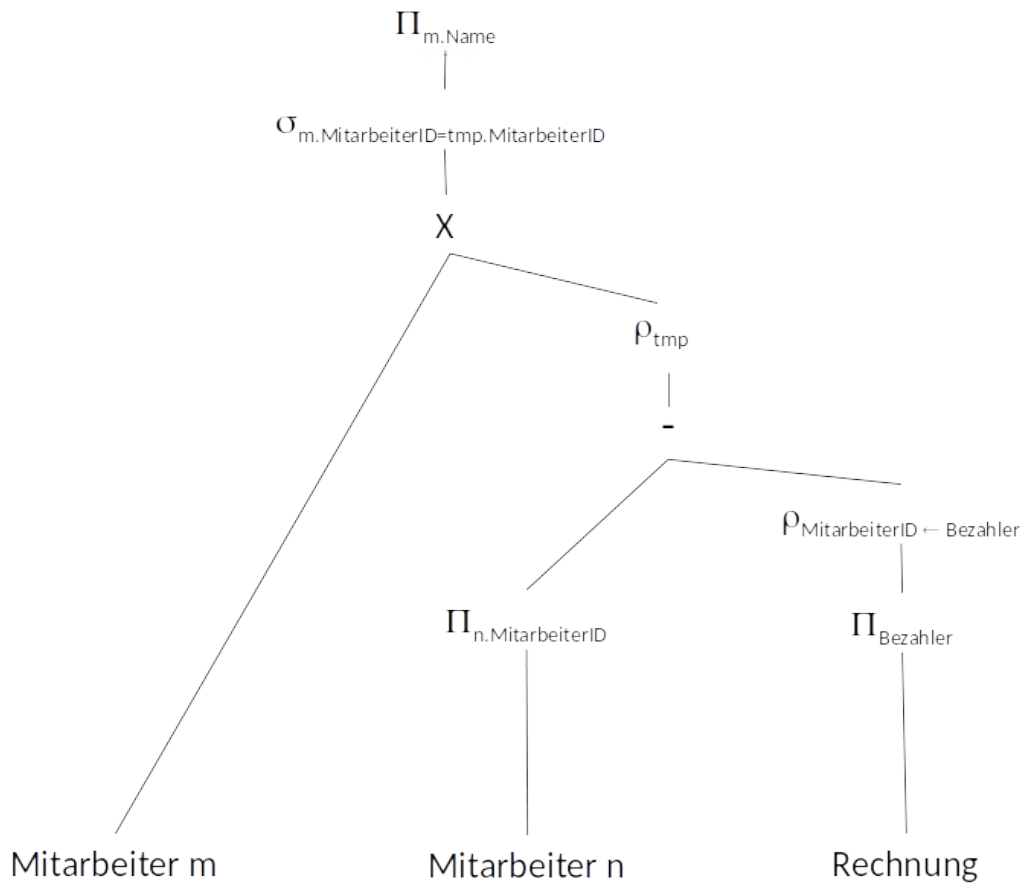
```
create table Auto(
AutoID  integer primary key,
Hersteller  VARCHAR(10) check (Hersteller in (‘Mercedes’, ‘Opel’, ‘VW’ ,’Ford’, ’Mazda’)),
Modell  VARCHAR(20),
Stufe integer check(Stufe between 1 and 6));

create table Mitarbeiter(
MitarbeiterID integer primary key,
Name VARCHAR(20),
Vorname VARCHAR(20),
Stufe integer check(Stufe between 1 and 6));

create table Bestellungen(
BestellID integer primary key,
Auto integer references Auto(AutoID) on delete cascade,
Ausleiher integer references Mitarbeiter(MitarbeiterID) on delete cascade,
von DATE,
bis DATE);

create Table Rechnungen (
RechnungsID integer primary key,
Bezahler integer references Mitarbeiter(MitarbeiterID) on update cascade on delete set null,
Bestellung integer references Bestellungen(BestellID) on update cascade on delete set null,
Betrag numeric (9,2),
Begruendung VARCHAR (100));
```

a) Erzeugen Sie den Operatorbaum sowie die korrespondierende SQL-Abfrage für die folgende Anfrage: Ermitteln Sie alle Mitarbeiter (Name), die noch kein Auto ausgeliehen haben. Wandeln Sie den Baum in eine SQL-Anfrage um.



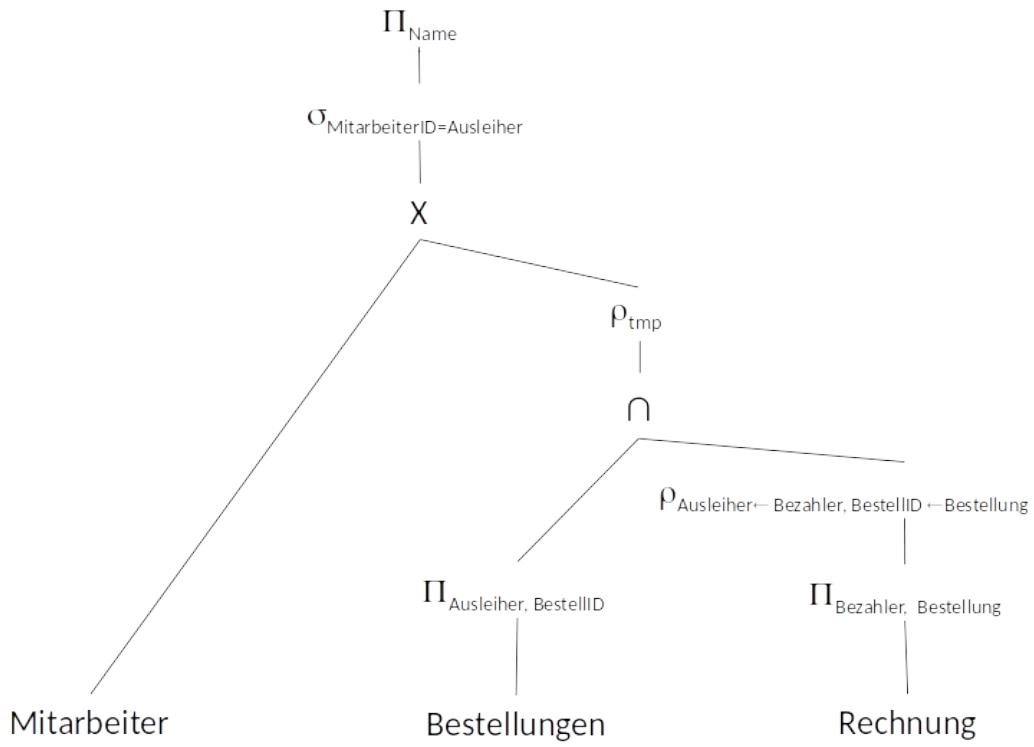
SELECT name FROM Mitarbeiter m, ((SELECT mitarbeiterid FROM mitarbeiter) except (SELECT ausleiher as mitarbeiterid FROM bestellungen)) as tmp where m.mitarbeiterid=tmp.mitarbeiterid

b) Erzeugen Sie den Operatorbaum sowie die korrespondierende SQL-Abfrage für die folgende Anfrage: Ermitteln Sie den Namen und Vornamen aller Mitarbeiter, die noch keine Rechnung bezahlt haben. Wandeln Sie den Baum in eine SQL-Anfrage um.



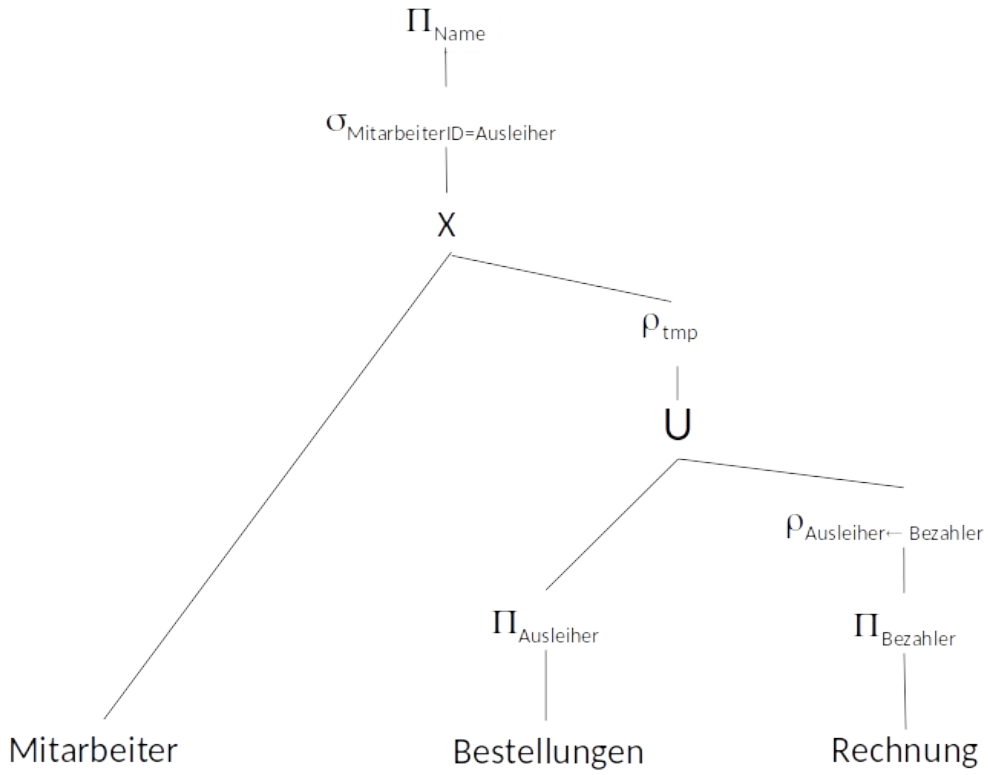
SELECT name FROM mitarbeiter natural join ((SELECT mitarbeiterid FROM mitarbeiter) except (SELECT bezahler AS mitarbeiterid from Rechnungen)) AS tmp

c) Erzeugen Sie den Operatorbaum sowie die korrespondierende SQL-Abfrage für die folgende Anfrage: Ermitteln Sie alle Mitarbeiter, die Rechnungen bezahlt haben für Autos, die sie auch ausgeliehen haben. Wandeln Sie den Baum in eine SQL-Anfrage um.



SELECT name from Mitarbeiter AS m, ((SELECT ausleiher, bestellid from bestellungen) intersect (SELECT bezahler AS ausleiher, bestellung AS bestellid FROM rechnungen)) AS tmp WHERE m.mitarbeiterid=tmp.ausleiher

d) Erzeugen Sie den Operatorbaum sowie die korrespondierende SQL-Abfrage für die folgende Anfrage: Ermitteln Sie alle Mitarbeiter, die schon mal ein Auto ausgeliehen haben oder eine Rechnung bezahlt haben. Verwenden Sie dabei den Union Operator. Wandeln Sie den Baum in eine SQL-Anfrage um.



SELECT name FROM mitarbeiter, ((SELECT ausleiher FROM bestellungen) UNION (SELECT bezahler AS ausleiher FROM rechnungen)) as tmp WHERE mitarbeiterid=ausleiher