Aufgabe 1:
Wieso verwendet man SQL…?

- SQL oder Structured Query Language ist eine Programmiersprache, die benutzt wird, relationale Datenbanken zu verwalten, während Relational Algebra ist eine Query Sprache und kein Datenbanken Design Tool! Obwohl gute RA Kenntnisse vor allem wie die RA Operationen funktionieren, helfen sehr gute Analysis und Design Entscheidungen zu treffen! Deshalb benutzt man SQL, da RA allein nicht reicht, um Datenbanken in der Realität zu verwalten!

Vorteile:

- SQL ist praktisch und erlaubt High Level Query Ausdrücke.
- Mit SQL-Integritätsbedingung (Bsp. Primärschlüssel, Fremd Schlüssel, not null, check) kann man das Auftreten von Redundanzen verhindern.
- Damit werden auch Anomalien verhindert
- Aggregationen wie (Sum, Count, Average, max, min…) sind in SQL erlaubt.
- In SQL kann man arithmetische Operationen anwenden, Bsp.: um neue Spalten als Resultat zu definieren
  (`SELECT unit_mass * quantity AS total_mass FROM m`)

- Relational Algebra ist eine Theoretische Grundlage bzw. Fundament für die relationalen Datenbanken
- RA hat einen starken mathematischen Hintergrund und kann Mehrdeutigkeiten bzw. Ambiguitäten beseitigen.
- Mit RA kann man komplexere mathematische Operationen (Bsp. Quotient ÷) direkt anwenden.

Nachteile:

- Operationen wie Quotient ÷ kann man in SQL direkt nicht anwenden.
- Obwohl SQL manche Prozedurale Elemente hat, ist es eine Deklarative Sprache, was den Umgang (Denkweise) manchmal schwierig macht.

- Je größer die Anzahl der Tabellen, zwischen den eine Relation gibt, desto aufwändiger wird RA zu nutzen.
- Das Anwenden von Operationen auf eine Spalte wie das Summieren ihrer Elemente ist nicht möglich mit RA
- Bisher gibt es keine Möglichkeit Berechnungen in RA zu führen (bsp. Arthmetische Operationen „+,-,*,/“)

Aufgabe 2

```sql
CREATE TABLE PERSON (
    Personen-ID INTEGER NOT NULL PRIMARY KEY,
    Vorname VARCHAR(50),
    Nachname VARCHAR(50),
);

CREATE TABLE Band (
    Bandname VARCHAR(50) NOT NULL PRIMARY KEY,
    Genre VARCHAR(50)
);


CREATE TABLE Musiker (
    Steuer-ID INTEGER NOT NULL PRIMARY KEY CHECK(Steuer-ID > 1407),
    Stimmlage VARCHAR(50),
    Instrument-ID INTEGER,
    Bezeichnung VARCHAR(50),
    Bandname VARCHAR(50) REFERENCES Band(Bandname) ON DELETE SET NULL,
    SeitJahr INTEGER CHECK(SeitJahr>=1900 AND SeitJahr<=2022)
);

CREATE TABLE Ticketkauf(
    Personen-ID INTEGER REFERENCES PERSON (Personen-ID) ON DELETE CASCADE,
    Bandname VARCHAR(50) REFERENCES Band (Bandname) ON DELETE CASCADE,
    Hallenname VARCHAR(50) NOT NULL,
    Stadt VARCHAR(50) NOT NULL,
    Preis DECIMAL(6,2) CHECK(Preis > Rabatt),
    Rabatt DECIMAL(6,2),
    FOREIGN KEY (Hallenname, Stadt) REFERENCES Konzerthalle ON DELETE CASCADE,
    PRIMARY KEY (Personen-ID, Bandname, Konzerthalle)
);


CREATE TABLE Konzerthalle (
    Hallenname VARCHAR(50) NOT NULL,
    Stadt VARCHAR(50) NOT NULL,
    PLZ INT CHECK(PLZ>0),
    PRIMARY KEY (Hallenname, Stadt)
);

CREATE TABLE PersonKennt(
    Person_1 REFERENCES PERSON (Personen-ID) ON DELETE CASCADE,
    Person_2 REFERENCES PERSON (Personen-ID) ON DELETE CASCADE,
    PRIMARY KEY (Person_1, Person_2)
);


CREATE TABLE PersonIstEinMusiker(
    Person-ID REFERENCES PERSON (Personen-ID) ON DELETE CASCADE,
    Steuer-ID REFERENCES Musiker (Steuer-ID) ON DELETE CASCADE,
    PRIMARY KEY (Person-ID, Steuer-ID)
);
```

Aufgabe 3

a)

```sql
SELECT Vorname, Nachname
FROM Person
WHERE PLZ < 80798 AND Nachname = "Huber";
```

b)

```sql
SELECT Person.ID, Person.Vorname, Person.Nachname
FROM Mitglied JOIN Person
ON Mitglied.Person = Person.ID
GROUP BY Person.ID
Having COUNT(Distinct Mitglied.Verein)>=3
ORDER BY Person.Vorname DESC;
```

c) i)

```sql
 CREATE VIEW GewinnerView AS
SELECT
ID,
if(Tore1 > Tore2, Verein1, Verein2) as Gewinner
FROM MATCH
WHERE Tore1 != Tore2
```

ii)

```sql
ALTER VIEW MatchView AS
SELECT
ID,
if(Tore 1 = Tore 2, NULL, if(Tore1 > Tore2, Verein1, Verein2)) as Gewinner
FROM MATCH
```

iii)

```sql
SELECT Gewinner, Verein.name, COUNT(Gewinner) as Gewonnen
FROM MatchView JOIN ON MatchView.Gewinner = Verein.ID
GROUP BY Gewinner
ORDER BY Gewonnen DESC
```

d) i)

$\{ id, name \mid \exists\ perID:$
$Verein(id, name, \_, \_, 26985) \land Mitglied(id, perID, \_) \land Person(perID, \_, \_, \_, 26985)\}$

ii) Schema(ver)= Schema(Verein)

$\{[ver.id, ver.name] \mid Verein(ver) \land (\exists\ mitg \in Mitglied, \underline{per} \in Person)$
$(ver.plz = 26985 \land ver.id = mitg.verein \land mitg.person = person.id \land person.plz = 26985)\}$

a) $Sel(\sigma_{R.Alter < 30 \vee R.Alter > 60}(R))$

$= Sel(\sigma_{R.Alter<30}(R)) + Sel(\sigma_{R.Alter>60}(R))$

$= 0.1 + 0.2$

$= 0.3$

$Sel(\sigma_{F.Budget > 500.000 \vee S.Vermögen > 15.000.000}(F \times S))$

$= Sel(\sigma_{F.Budget>500.000}(F \times S)) + Sel(\sigma_{S.Vermögen>15.000.000}(F \times S))$

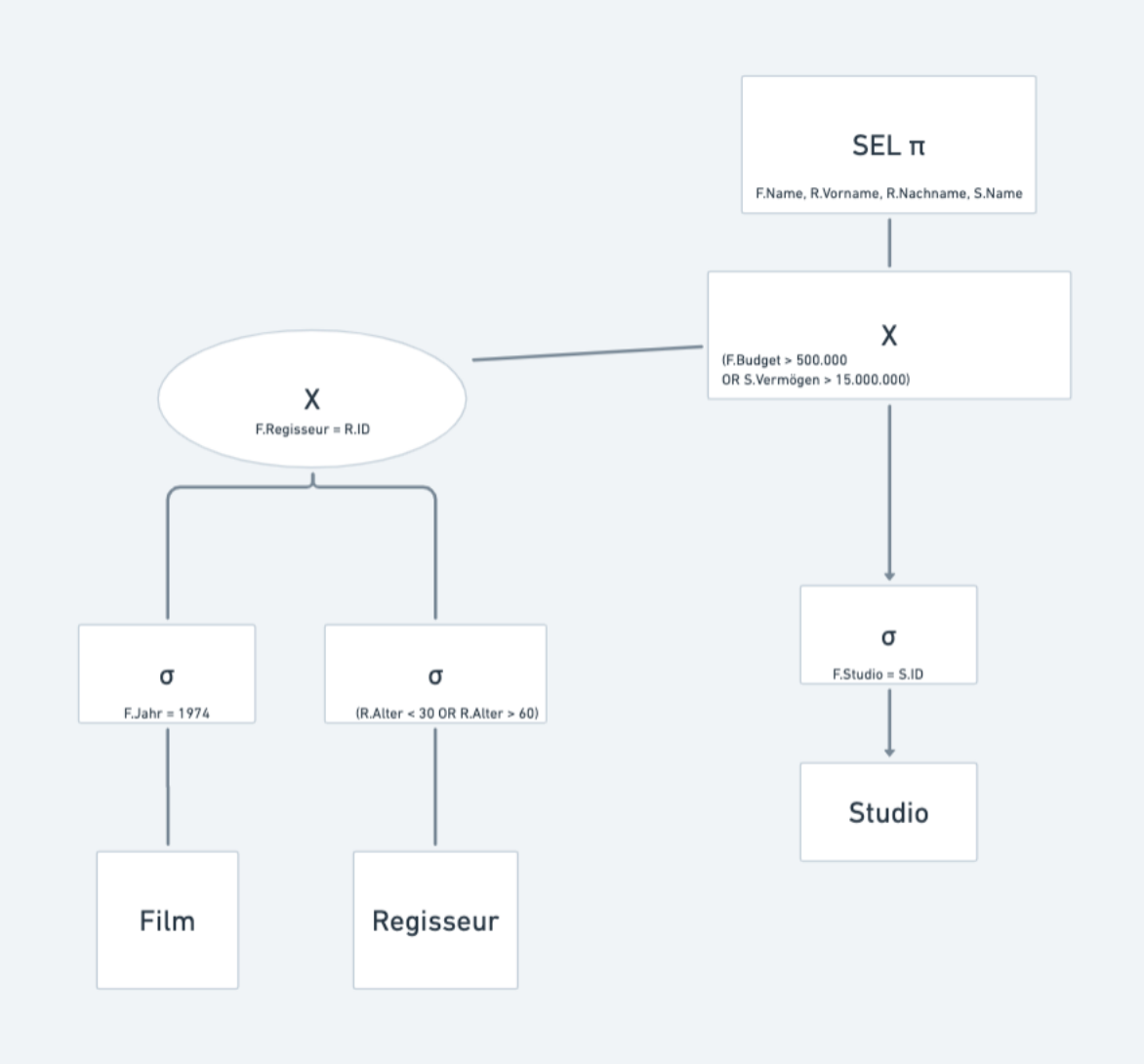
$= Sel(\sigma_{F.Budget>500.000}(F)) + Sel(\sigma_{S.Vermögen>15.000.000}(S))$

$= 1 - Sel(\sigma_{F.Budget \leq 500.000}(F)) + 0.5$

$= 1 - 0.8 + 0.5$

$= 0.2 + 0.5$

$= 0.7$



Aufagabe 5:

1)Kanonische Überdeckung:

	Linksreduktion	Rechtsruduktion
A,C → D	A→ D weil $\{A\}^+ = \{A,C\}^+ = \{A, B, C, D, F\}$	A→ D
A → G	A→G	A→G
A, C → F	A → F weil $\{A\}^+ = \{A,C\}^+ = \{A, B, C, D, F\}$	A→ F
C → F	C → F	C → F
B, A, E → F	A, E → F weil $\{B,A,E\}^+ = \{A,E\}^+ = \{A, B, C, D,E, F\}$	A, E → F
G → B	G → B	G → B
B, E → F, C	B, E → F, C	B, E → C weil F in (F-(B,E→F,C)und(C→F))= {B,E,C,F}))
A → B	A → B	A → weil B in (B-(A→B)und(A→G)und

		(G→B))={ A, B, C, D, F}))

Entfernung rechsleerer Abhängigkeiten: ~~A→~~

Zusammenfassen von Abhängigkeiten mit gleichen linken Seiten:
A→ D und A→G und A→ F wird zu A→D,F,G

2)
Erzeugen eines neuen Relationsschemas:
R1(A, D, F, G) // A→D,F,G
R2(C, F) // C → F
R3(A, E, F) // A, E → F
R4(G, B) // G → B
R5(B, E, C) // B, E → C

3) Rekonstruktion eines Schlüsselkandidaten
Hier passiert wieder nichts da A & E schon zusammen in einer Relation auftreten R3(A, E, F)

4) Elimination überflüssiger Relationen
Hier passiert auch nichts

Ergebnis:
R1(A, D, F, G)
R2(C, F)
R3(A, E, F)
R4(G, B)
R5(B, E, C)