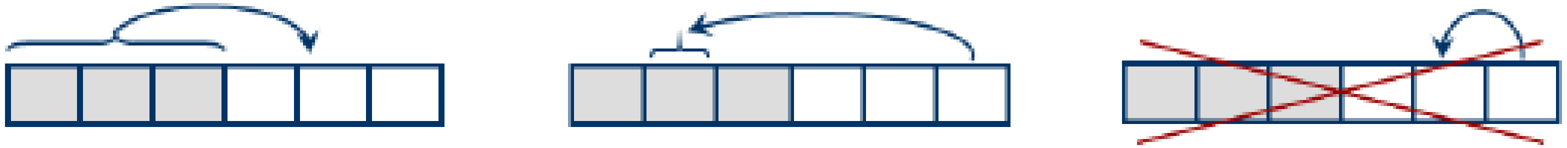
Normalisierung

Normalformen - Zusammenfassung

1NF – alle Attribute sind atomar

2NF – alle Nichtschlüsselattribute sind voll funktional abhängig von jedem Kandidatenschlüssel (keine partiellen Abhängigkeiten)

3NF – in 2NF und alle Nichtschlüsselattribute sind nur von Kandidatenschlüssel abhängig (keine transitiven Abhängigkeiten)



BCNF – jede Determinante ist ein Superschlüssel (alle FDs werden von Kandidatenschlüsseln bestimmt)

3NF-Zerlegung

- Normalerweise: per Hand gemacht und überprüft

- Eine 3NF Zerlegung, die verlustlos und abhängigkeitsbewahrend ist, ist möglich, aber wir müssen dafür nicht die ganze Menge der FDs benutzen, sondern eine „minimale“ Überdeckung von F
- Wichtigster Algorithmus: **3NF-Synthesealgorithmus**
- Braucht als Eingabe eine redundanzfreie Menge von FDs (**kanonische Überdeckung**)

Berechnung der kanonischen Überdeckung

- Schritt 1 : Linksreduktion
- Schritt 2 : Rechtsreduktion
- Schritt 3 : Entferne die FDs der Form $A \to \emptyset$
- Schritt 4 : Ersetze alle FDs der Form

  $A \to B_1, \ldots, A \to B_k$ durch $A \to B_1 \cup \cdots \cup B_k$

Kanonische Überdeckung - Beispiel

- F = {ABCD → E, E → D, A → B, AC → D}
- Schritt 1. Linksreduktion:
  - B und D sind überflüssig in ABCD → E ⇒ {AC → E, E → D, A → B, AC → D}
- Schritt 2. Rechtsreduktion:
  - D ist überflüssig in AC → D ⇒ {AC → E, E → D, A → B, AC → ∅}
- Schritt 3. ⇒ {AC → E, E → D, A → B}
- Schritt 4. –

$$\Rightarrow \quad F_c = \{\text{AC} \to \text{E}, \text{E} \to \text{D}, \text{A} \to \text{B}\}$$

Kanonische Überdeckung

- **Hausaufgabe:**
  - Berechne die kanonische Überdeckung von

    $F = \{A \rightarrow B, B \rightarrow C, AB \rightarrow C\}$
- **Bemerkung.** Die kanonische Überdeckung ist nicht eindeutig (es hängt von dem Auswahl der überflüssigen Attributen ab)

3NF - Synthesealgorithmus

- Ziel des Synthesealgorithmus:
  - Zerlegung einer Relation R mit funktionalen Abhängigkeiten F in Relationen $R_1,...,R_n$, so dass folgende Bedingungen erfüllt sind:
    - Kein Informationsverlust
    - Bewahrung der funktionalen Abhägigkeiten
    - $R_1,...,R_n$ in 3NF

3NF - Synthesealgorithmus

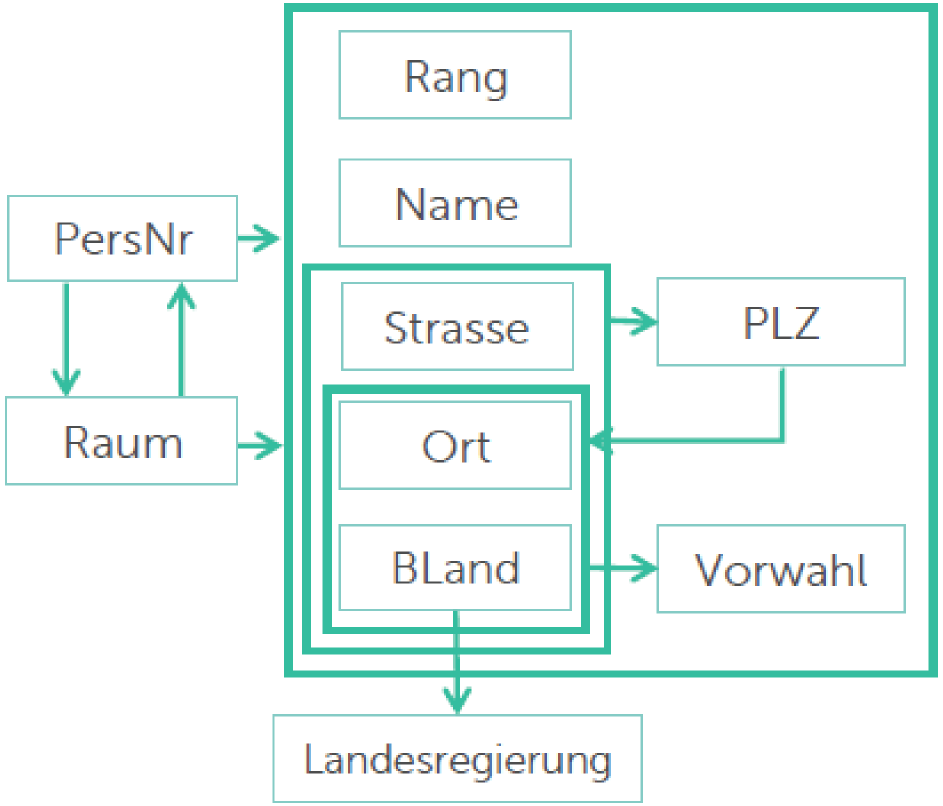
- Bestimme die kanonische Überdeckung F_C der Menge F
- Für jede FD $A \rightarrow B$ in F_c:
  - Erzeuge eine Relation $R_A = A \cup B$ und ordne R_A die FDs
    $F_A=\{C \rightarrow D \in F_C | C \rightarrow D \subseteq RA\}$ zu
  - Falls keine der erzeugten Relationen einen Schlüsselkandidaten des ursprunglichen Relation R enthalten:
    erzeuge zusätzlich eine neue Relation $R_K = K$ und $FK = \emptyset$, wobei K ein Schlüsselkandidat von R ist
- Entferne die Relationen R_A, die in einem anderen Schema enthalten sind, d.h. $R_i \subseteq R_j$

Synthesealgorithmus - Beispiel

- R(A,B,C,D,E)
- F = {ABCD → E, E → D, A → B, AC → D}
- Kanonische Überdeckung $F_c = \{AC \rightarrow E, E \rightarrow D, A \rightarrow B\}$
- Einzige Kandidatenschlüssel : AC ($AC^+ = R$)
- R ist nicht 3NF wegen A → B
- 3NF Zerlegung von R:
  - $R_1(\underline{A,C},E)$ mit $F_1=\{AC \rightarrow E\}$, $R_2(\underline{E},D)$ mit $F_2=\{E \rightarrow D\}$, $R_3(\underline{A},B)$ mit $F_3=\{A \rightarrow B\}$
  - R_1 enthält den Schlüssel AC

Synthesealgorithmus

- **Hausaufgabe**
  - Professoren(PersNr, Raum, Rang, Name, Strasse, Ort, BLand, Landesregierung, PLZ, Vorwahl)
  - FDs – in der Figur
  - Ist die Relation in 3NF? Warum ja/nicht?
  - Falls nicht, finde eine 3NF Zerlegung.



3NF Zerlegung

- **Bemerkung**. Eine 3NF Zerlegung ist nicht eindeutig. Es hängt vom Folgenden ab:
  - Auswahl der kanonischen Überdeckung
  - Auswahl der Relation, die in einer anderen enthalten ist, zum Eliminieren
- **Bemerkung.** Zerlegung ist eine Lösung für Redundanzen und Anomalien, **aber** zu viel Zerlegung kann schlecht sein aus Leistungsgründe.
- Beispiel.
  - R(Prof, Institut, Tel, Raum) mit F ={Prof $\rightarrow$ (Institut, Tel, Raum)}
  - Die Zerlegung R_1 = {Prof, Institut}, R_2= {Prof, Tel}, R_3= {Prof, Raum} ist korrekt, aber nicht notwendig

Mehrwertige Abhängigkeiten / multivalued dependencies (MVD)

- Verallgemeinerung funktionaler Anhängigkeiten
- Beispiel:
  - Fähigkeiten(PersNr, Sprache, Programmiersprache)
  - In diesem Schema gibt es keine FDs → BCNF, trotzdem haben wir Redundanz
  - Eine Person kann mehrere Sprachen sprechen
  - Eine Person beherrscht mehrere Programmiersprachen
  - Sprache und Programmiersprache sind mehrwertig abhängig von PersNr

PersNr	**Sprache**	**ProgSprache**
3002	Deutsch	C
3002	Englisch	C
3002	Englisch	Java
3002	Deutsch	Java
3005	Englisch	C
3005	Deutsch	C

Mehrwertige Abhängigkeiten

- In dem vorigen Schema werden **voneinenader unabhängige Konzepte** vermischt
- Diese Konzepte haben nur den Schlüsselattribut gemeinsam
- Informelle Definition: ein Attribut (hier, PersID) bestimmt mehrere voneinander unabhängige Attribute

- Kompaktere Speicherung:

PersNr	Sprache
3002	Deutsch
3002	Englisch
3005	Englisch
3005	Deutsch

PersNr	ProgSprache
3002	C
3002	Java
3005	C

Mehrwertige Abhängigkeiten

- **Definition.** Sei R eine Relation und $\alpha, \beta \subseteq$ R, γ = R - $\alpha\beta$ Dann ist β **mehrwertig abhängig** von α, $\boldsymbol{\alpha} \rightarrow\rightarrow \boldsymbol{\beta}$, wenn in jeder Ausprägung r der Relation R gilt:

$$t_1, t_2 \in r \text{ und } \pi_\alpha(t_1) = \pi_\alpha(t_2) \Rightarrow$$

$$\exists\, t_3 \in r \text{ so dass } \pi_{\alpha\beta}(t_1) = \pi_{\alpha\beta}(t_3) \text{ und } \pi_\gamma(t_2) = \pi_\gamma(t_3)$$

Als Konsequenz daraus, wenn wir t_2 und t_1 betrachten, folgt dass

$$\exists\, t_4 \in r \text{ so dass } \pi_{\alpha\beta}(t_2) = \pi_{\alpha\beta}(t_4) \text{ und } \pi_\gamma(t_1) = \pi_\gamma(t_4)$$

- Intuition: Für alle Tupel mit gleichem Wert für A kommen alle B-C Kombinationen vor

Inferenzregeln

- Komplement: $X \rightarrow\rightarrow Y \Rightarrow X \rightarrow\rightarrow R\text{-}XY$
- Mehrwertige Verstärkung: $X \rightarrow\rightarrow Y$, $Z \subseteq W \Rightarrow WX \rightarrow\rightarrow YZ$
- Mehrwertige Transitivität: $X \rightarrow\rightarrow Y$, $Y \rightarrow\rightarrow Z \Rightarrow X \rightarrow\rightarrow Z\text{-}Y$
- Verallgemeinerung: $X \rightarrow Y \Rightarrow X \rightarrow\rightarrow Y$
- Koaleszenz: $X \rightarrow\rightarrow Y$, $W \cap Y = \emptyset$, $W \rightarrow Z$, $Z \subseteq Y \Rightarrow X \rightarrow Y$ (Coalescence)

4. Normalform

- **Definition.** Sei R eine Relation, $X, Y \subseteq R$. R ist in $\mathbf{4NF}$ wenn für jede MVD $X \to \to Y$ eine der folgenden Bedingungen gilt:
  - $Y \subseteq X$ (1) oder
  - $XY = R$ (2) oder
  - X ist ein Superschlüssel (X enthält einen Schlüssel von R) (3)
- Bem. Wenn (1) oder (2) gilt, dann heißt die Abhängigkeit $X \to \to Y$ **trivial.**
- **Äquivalente Def.** Eine Relation R ist in 4NF, wenn sie in der BCNF ist und nur noch triviale MVDs hat.
- 4NF ist eine Verstärkung der BCNF

4NF - Beispiel

- Fähigkeiten(PersNr, Sprache, Programmiersprache)
  - ist in der BCNF
  - ist nicht in der 4NF
    - $\text{PersNr} \rightarrow \rightarrow \text{Sprache}$ verletzt 4NF
- Zerlegung:
  - Sprachen(PersNr, Sprache) – 4NF

    MVD: PersNr $\rightarrow \rightarrow$ Sprache
  - ProgrSprachen(PersNr, Programmiersprache) – 4NF

    MVD: PersNr $\rightarrow \rightarrow$ Programmiersprache

5. Normalform

- **Definition(Verbund-Abhängigkeit/ Join Dependency).** Eine Relation R genügt der Verbund-Abhängigkeit(JD) $\otimes$ $\{R_1, ..., R_n\}$ genau dann, wenn $\{R_1, ..., R_n\}$ eine verlustlose Zerlegung von R ist.
- Eine MVD $X \rightarrow\rightarrow Y$ der Relation R kann als Join-Abhängigkeit representiert werden wie folgt: $\otimes \{XY, X(R\text{-}Y)\}$

PersID	**Laboratory Class**	**Room**
7223	Electrotechnics	344
7223	Electrotechnics	347
7223	Web Development	655
3111	Electrotechnics	344
3111	Web Development	601

5. Normalform

- **Definition.** Eine Relation R ist in 5NF falls für jede Verbund-Abhängigkeit JD in R eine der folgenden Bedingungen gilt:
  - R_i = R für eine i (JD ist trivial)
  - JD wird durch Schlüssel von R verursacht
- Eine JD $\otimes$ {R_1, ..., R_n} wird durch Schlüssel von R verursacht, wenn jeder R_i ein Superschlüssel ist (ein Kandidatschlüssel enthält)

5. Normalform

- Relation Angestellte *(<u>PersNr</u>, Abteilung, Name, Vorname, Alter)*
- Die Relation könnte durch folgende Verbunde entstanden sein:
  - ((*PersNr, Abteilung*), (*PersNr, Name, Vorname, Alter*))
  - ((*PersNr, Abteilung*), (*PersNr, Name*), (*PersNr, Vorname*), (*PersNr, Alter*))

- Ein Verbund mit Hilfe von Schlüsselattributen (*PersNr*) ist problemlos, da jedem Schlüsselwert der einen, ein Schlüsselwert der anderen entspricht. In der Ergebnisrelation kommen auf diese Weise auch nur die Tupel der verschiedenen Relationen, die einen gemeinsamen Schlüssel haben.

Normalisierung – wie weit?

- In der Praxis spielt 5NF nur in seltenen Sonderfällen eine Rolle.
- Es ist nicht immer sinnvoll alle Normalisierungsschritte durchzuführen.
- Die Realisierung von BCNF ist zu empfehlen und die Verhinderung einer mehrwertigen Abhängigkeit auch, so dass die 4NF immer sichergestellt ist.

In welcher Normalform ist folgende Relation? Ist es sinnvoll weiter zu normalisieren?

- R1 (patientID, diagnosis, treatment)
- R2 (CNP, patientID, patientName, patientAddress)
- R3 (patientID, prescriptionID, patientName, treatment, cost)
- R4 (hospitalID, patientID, hospitalName, patientName)

Schreibe ein Query, welches die Diagnosen mit den, im Durchschnitt, teuersten Behandlungen, haben.

In welcher Normalform ist folgende Relation? Ist es sinnvoll weiter zu normalisieren?

Bemerkung: Kunden und Lieferanten sind Firmen

- R1 (<u>clientID</u>, <u>productID</u>, orderQty, productName)
- R2 (<u>clientID</u>, clientName, clientContactPerson)
- R3 (<u>productID</u>, productName, supplierID, supplierContactPerson)
- R4 (<u>clientID</u>, clientName, contactPersonID, clientContactPerson)
- R5 (<u>productID</u>, <u>supplierID</u>, price, supplierName)

Relationale Algebra

Relationale Abfragesprachen / Relational Query Languages (QL)

- Abfragesprachen: Daten aus einer Datenbank zu manipulieren und abzufragen (retrieve information)
- Das relationalle Modell hat einfache und leistungsfähige Abfragesprachen (man kann viel optimieren)
- Abfragesprache ≠ Programmiersprache
- Abfragesprachen:
  - Nicht für komplexe Operationen
  - Erlaubt einfacher und effizienter Zugriff zu großen Datensätze

Formale Relationale Abfragesprachen / Query Languages

- Zwei mathematische Abfragesprachen stellen die theoretische Grundlage der „reelen" Abfragesprachen (wie z.B. SQL) in relationalen Datenbanken:
    - Relationale Algebra:
        - kann Ausführungspläne beschreiben (operational)
    - Relationale Kalküle:
        - Der Benutzer kann beschreiben, was er haben will und nicht wie es berechnet werden soll (non-operational, deklarativ)
        - Domänenkalkül, Tupelkalkül

Relationale Algebra

- Theoretische Grundlage für Datenbanken
- Basiert auf Mengenlehre und algebraische Strukturen
  - Relation – Tabelle
  - Tupel – Zeile
  - Attribut – Spalte

Codd, E. F. (1970). A relational model of data for large shared data banks. Communications of the ACM, 13(6), 377–387.

Relationale Algebra

- Fünf Basisoperationen:
  - **Projektion** (π) : wählt bestimmte Spalten aus der Relation und gibt diese als neue Relation aus („löscht“ die anderen Spalten)
  - **Selektion** (σ) : wählt bestimme Zeilen aus der Relation und gibt diese als neue Relation aus („löscht“ die anderen Zeilen)
  - **Kartesisches Produkt** ($\times$) : erlaubt die Verknüpfung zweier Relationen
  - **Differenz** (-) : gibt die Tupeln aus der ersten Relation, die sich nicht in der zweiten Reltion befinden, aus
  - **Vereinigung** (∪) : gibt die Tupeln aus der ersten und zweiten Relation aus
- Zusätzliche Operatoren: Umbenennen, Durchschnitt, Division, Verbund
- Die Operationen können zusammengesetzt sein (jede Operation hat eine Relation als Ergebnis)

Projektion

- **Definition.** Sei $L = (A_1, \ldots, A_n)$ eine Teilmenge von Attributen (Spalten) aus der Relation R. Die Projektion der Attribute L einer Relation R ist definiert als die Relation $R'(A_1, \ldots, A_n)$ mit:

$$R' = \pi_L(R) = \{\ t' \mid t \in R \wedge t'.A_1 = t.A_1 \wedge \ldots \wedge t'.A_n = t.A_n\}$$

- Oder, anders gesagt:
  - Die Projektion aus einem Tupel $t \in R$ ist definiert als das Tupel

$$\pi_L(t) = (t(A_1), \ldots, t(A_n))$$

  - Die Projektion der Relation R ist definiert als die Relation

$$\pi_L(R) = \{\pi_L(t) \mid t \in R\ \}$$

Projektion - Beispiel

Studenten

MatrikelNr	Name	Vorname	Vorname2	Geburt	Ort	SgNr	Bafoeg
1001	Schmidt	Hans	Peter	24.2.1990	Würzburg	2	200
1002	Meisel	Dirk	Helmut	17.8.1989	Schweinfurt	3	500
1003	Schmidt	Amelie		19.9.1992	Rimpar	1	0
1004	Krause	Christian	Johannes	3.5.1990	Würzburg	1	100
1005	Schäfer	Julia		30.3.1993	Kitzingen	5	0
1006	Rasch	Lara		30.3.1992	Würzburg	3	0
1007	Bakowski	Juri		15.7.1988	Schweinfurt	4	400

$$\pi_{Name,Ort}(Studenten)$$

Name	Ort
Schmidt	Würzburg
Meisel	Schweinfurt
Schmidt	Rimpar
Krause	Würzburg
Schäfer	Kitzingen
Rasch	Würzburg
Bakowski	Schweinfurt

Projektion in SQL

- Ist $\pi_{Name,Ort}$(Studenten) äquivalent mit

  `SELECT Name, Ort FROM Studenten` ?

- NEIN!
  - Relationale Algebra funktioniert mit Mengen $\Rightarrow$ keine Duplikate (keine identischen Tupel)
  - Das ist in SQL nicht standardmäßig so!
  - Äquivalent:

```
SELECT DISTINCT Name, Ort FROM Studenten
```

Selektion / Restriktion

- **Definition.** Die Selektion einer Relation R ist definiert als die Menge aller Tupel aus R, die der Selektionsbedingung P genügen:

$$\sigma_P(R) = \{ t \mid t \in R \land P(t) \}$$

- Die Bedingung P setzt sich zusammen aus:
  - Operanden: Konstanten oder Name eines Attributs
  - Vergleichsoperatoren: $=, \neq, <, \leq, >, \geq$
  - Boolsche Operatoren: $\lor, \land, \neg$

Selektion - Beispiel

Studenten

MatrikelNr	Name	Vorname	Vorname2	Geburt	Ort	SgNr	Bafoeg
1001	Schmidt	Hans	Peter	24.2.1990	Würzburg	2	200
1002	Meisel	Dirk	Helmut	17.8.1989	Schweinfurt	3	500
1003	Schmidt	Amelie		19.9.1992	Rimpar	1	0
1004	Krause	Christian	Johannes	3.5.1990	Würzburg	1	100
1005	Schäfer	Julia		30.3.1993	Kitzingen	5	0
1006	Rasch	Lara		30.3.1992	Würzburg	3	0
1007	Bakowski	Juri		15.7.1988	Schweinfurt	4	400

$$\Downarrow \quad \sigma_{Name='Schmidt'}(Studenten)$$

MatrikelNr	Name	Vorname	Vorname2	Geburt	Ort	SgNr	Bafoeg
1001	Schmidt	Hans	Peter	24.2.1990	Würzburg	2	200
1003	Schmidt	Amelie		19.9.1992	Rimpar	1	0

Selektion in SQL

$$\sigma_{\texttt{Name = 'Schmidt'}}(\texttt{Studenten})$$

$$\downarrow$$

```
SELECT DISTINCT * FROM Studenten
WHERE Name = 'Schmidt'
```

Aufpassen

- Nicht verwechseln:

Projektion - $\pi_{\text{Attr1, Attr2}}(\text{Relation})$

```
SELECT DISTINCT Attr1, Attr2
FROM Relation
WHERE c
```

Selektion - $\sigma_c(\text{Relation})$

Zusammensetzung von Projektion und Selektion

$$\pi_{\text{Name, Vorname, Ort}}(\sigma_{\text{Name = 'Schmidt'}}(\text{Studenten}))$$

```
SELECT DISTINCT Name, Vorname, Ort
FROM Studenten
WHERE Name = 'Schmidt'
```

$$\sigma_{\text{Name = 'Schmidt'}}(\pi_{\text{Name, Vorname, Ort}}(\text{Studenten}))$$

- Welches ist das äquivalente SQL Query?
- Kann man immer die Reihenfolge der Projektion und Selektion wechseln?
- Nein → die Selektion kann nach der Projektion ausgeführt werden, nur dann wenn die Selektionsbedingung nur Attribute aus der Projektion enthält

Vereinigung, Durchschnitt, Differenz

- Vereinigung: $R_1 \cup R_2 = \{ t \mid t \in R_1 \vee t \in R_2 \}$
- Durchschnitt: $R_1 \cap R_2 = \{ t \mid t \in R_1 \wedge t \in R_2 \}$
- Differenz: $R_1 - R_2 = \{ t \mid t \in R_1 \wedge t \notin R_2 \}$

- R_1 und R_2 müssen für alle diese Operationen gleiches Relationenschema besitzen
- Wertebereiche müssen kompatibel oder vereinigungsverträglich sein
- Bem. Es gilt $R_1 \cap R_2 = R_1 - (R_1 - R_2)$

Vereinigung, Durchschnitt, Differenz in SQL

$R_1 \cup R_2$

```
SELECT DISTINCT *
FROM R1
UNION
SELECT DISTINCT *
FROM R2
```

$R_1 \cap R_2$

```
SELECT DISTINCT *
FROM R1
INTERSECT
SELECT DISTINCT *
FROM R2
```

$R_1 - R_2$

```
SELECT DISTINCT *
FROM R1
EXCEPT
SELECT DISTINCT *
FROM R2
```

Kartesisches Produkt

- Das kartesische Produkt zweier Relationen $R_1(A_1, \ldots, A_n)$ und $R_2(B_1, \ldots, B_m)$ ist definiert als Relation:

$$R_1 \times R_2 = \{\, t \mid t_1 \in R_1 \wedge t_2 \in R_2$$
$$\wedge\ t.A_1 = t_1.A_1 \wedge \ldots \wedge t.A_n = t_1.A_n$$
$$\wedge\ t.B_1 = t_2.B_1 \wedge \ldots \wedge t.B_m = t_2.B_m \}$$

SQL:

```
SELECT DISTINCT *
FROM R1, R2
```

Kartesisches Produkt - Beispiel

A_1	A_2
1	A
2	B
3	C

X

B_1	B_2
1	X
2	Y
4	Z

=

A_1	A_2	B_1	B_2
1	A	1	X
1	A	2	Y
1	A	4	Z
2	B	1	X
2	B	2	Y
2	B	4	Z
3	C	1	X
3	C	2	Y
3	C	4	Z

θ-Join (Theta-Verbund)

- Auswahl bestimmter Tupel aus dem kartesischen Produkt $R_1 \times R_2$
- Basis der Verknüpfung der Relationen: eine Bedingung c

$$R_1 \bowtie_c R_2 = \sigma_c (R_1 \times R_2)$$

Bsp.

$$\text{Studenten} \bowtie_{\text{Studenten.MatrikelNr = Enrolled.MatrikelNr}} \text{Enrolled}$$

SQL:

```
SELECT DISTINCT *
FROM Studenten, Enrolled
WHERE Studenten.MatrikelNr
= Enrolled.MatrikelNr
```

oder

```
SELECT DISTINCT *
FROM Studenten
INNER JOIN Enrolled ON
Studenten.MatrikelNr =
Enrolled.MatrikelNr
```

Equi-Join

- Einen θ-Join der Form $R_1 \bowtie_{R_1.A_i = R_2.B_j} R_2$ nennt man Equi-Join
- Notation für Equi-Join um zu unterscheiden: $R_1 \bowtie_{E(R_1.A_i = R_2.B_j)} R_2$
- Die Bedingung muss der Form einer Gleichwertigkeit zwischen Attribute der ersten und der zweiten Relation sein
- Das Ergebnis enthält nur einen der Attribute, da es redundant ist beide zu behalten (die Attribute sind gleich)

Equi-Join Beispiel

Kurse

KursId	Titel
Alg1	Algorithmen1
DB1	Datenbanken1
DB2	Datenbanken2

Enrolled

MatrNr	KursId	Note
1234	Alg1	7
1235	Alg1	8
1234	DB1	9
1234	DB2	7
1236	DB1	10

Kurse $\bowtie_{E(\text{Kurse.KursId}=\text{Enrolled.KursId})}$ Enrolled

KursId	Titel	MatrNr	Note
Alg1	Algorithmen1	1234	7
Alg1	Algorithmen1	1235	8
DB1	Datenbanken1	1234	9
DB2	Datenbanken2	1234	7
DB1	Datenbanken1	1236	10

Natürlicher Verbund / Natural join

- Verknüpft zwei Relationen indem alle gleichbenannten Attribute der beiden Relationen betrachtet werden und nur eines der gleichen Attribute kommt im Ergebnis vor (ohne Redundanzen)
- Qualifizierende Tupel müssen für diese gleichbenannten Attribute gleiche Werte aufweisen, um in das Ergebnis einzugehen
- Gibt es kein gemeinsames Attribut so ist das Ergebnis das kartesische Produkt

Kurse

KursId	**Titel**
Alg1	Algorithmen1
DB1	Datenbanken1
DB2	Datenbanken2

Enrolled

MatrNr	**KursId**	**Note**
1234	Alg1	7
1235	Alg1	8
1234	DB1	9
1234	DB2	7
1236	DB1	10

Kurse ⋈ Enrolled

KursId	**Titel**	**MatrNr**	**Note**
Alg1	Algorithmen1	1234	7
Alg1	Algorithmen1	1235	8
DB1	Datenbanken1	1234	9
DB2	Datenbanken2	1234	7
DB1	Datenbanken1	1236	10

Division

- Die Relation R_1 enthält Attribute X und Y und R_2 enthält den Attribut Y.

$$R_1 \div R_2 = \{ \langle X \rangle \mid \forall \langle Y \rangle \in R_2 : \exists \langle X,Y \rangle \in R_1 \}$$

- $R_1 \div R_2$ (oder R_1 / R_2) enthält alle X Tupeln so dass für jedes Y Tupel in $R_{2,}$ ein XY Tupel in R_1 existiert
- X und Y können auch Mengen von Attributen sein

Division - Beispiel

R_1

A	B
4	3
4	1
4	7
8	3
8	1
8	7

R_2

A
4
8

$R_1 \div R_2$

B
3
1
7

Division

- Nicht als primitiver Operator, aber nützlich
- Die Division wird dann eingesetzt, wenn die Frage „**für alle**“ enthält
- Beispielfragestellungen für eine Division:
    - Welche Personen haben eine Kundenkarte von **allen** Filialen?
    - Welche Mitarbeiter arbeiten an **allen** Projekten?
    - Welche Studenten hören **alle** Vorlesungen von Prof. X?

Division

- Darstellung des Quotienten durch die Basisoperatoren:
  - Idee: Berechne alle X Werte, die von irgendeinem Y Wert aus R_2 disqualifiziert wird
  - X wird disqualifiziert wenn für einen Y der Tupel XY nicht in R_1 enthalten ist:

$$\pi_X((\pi_X(R_1) \times R_2) - R_1)$$

  - Der Quotient $R_1 \div R_2$ enthält dann alle X Werte aus R_1, die nicht disqualifiziert sind:

$$R_1 \div R_2 = \pi_X(R_1) - \pi_X((\pi_X(R_1) \times R_2) - R_1)$$

Umbenennen von Relationen und Attributen

- Umbenennung unterscheidet sich von den anderen Operatoren dadurch, dass keine Berechnung vorgenommen wird
- Operator ist aber notwendig, wenn eine Relation mehrfach in einer Anfrage vorkommt (z.B. Join)
- $\rho_S(R)$: Relation R wird in Relation S umbenannt
- $\rho_{B \leftarrow A}(R)$: Attribut A der Relation R wird umbenannt in B
- Das Relationenschema wird nicht geändert (nur eventuell Namen von Attributen)

Zuweisungsoperation

- Die Zuweisungsoperation $\leftarrow$ ist eine Methode komplexe Abfragen zu repräsentieren
- Eine Abfrage kann in einer temporären Variable gespeichert werden

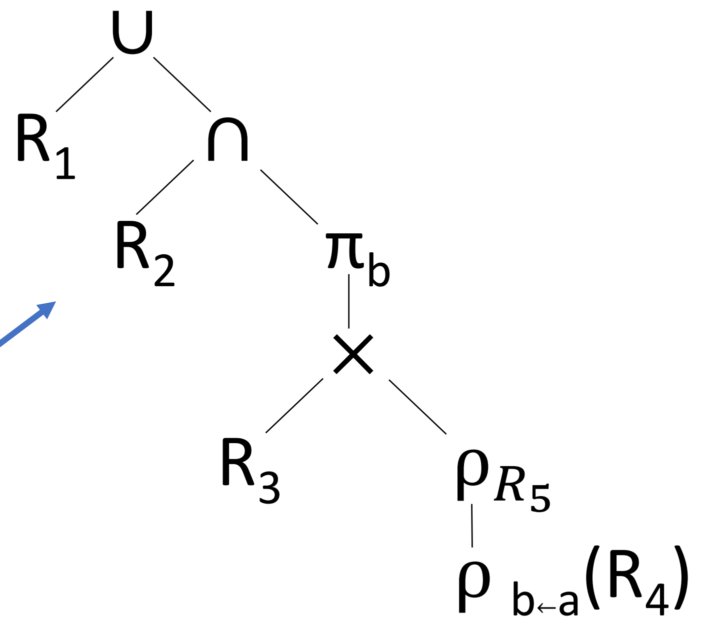
$$\text{Temp} \leftarrow \pi_X(R_1 \times R_2)$$

- Dann kann man diese Variable in weiteren Abfragen benutzen

$$\text{Erg} \leftarrow \text{Temp} - R_3$$

Komplexe Abfragen

$$R_1 \cup (R_2 \cap \pi_b (R_3 \times \rho_{R_5}(\rho_{b \leftarrow a}(R_4))))$$



Ausführungsplan

Erweiterte Relationale Algebra - Operatoren

- Erweiterte Projektion
- Aggregierungsfunktionen
- Outer Join
- Datenbankänderungen

Erweiterte Projektion

- Erweitert die Projektion, indem arithmetische Funktionen als Projektionbedingung benutzt werden können

$$\pi_{F1,\ldots,Fn}(R)$$

- F1, … , Fn sind arithmetische Funktionen, die Konstante oder Attribute der Relation R enthalten

Aggregierungsfunktionen

- Haben mehrere Werte als Input und ein Wert als Output:
  - avg: Mittelwert
  - min: Minimum der Werte
  - max: Maximum der Werte
  - sum: Summe der Werte
  - count: Anzahl der Werte

Aggregierungsfunktionen

$$_{G_{1,}G_2,\ldots,G_n}\vartheta_{F_1(A_1),\, F_2(A_2),\ldots,\, F_n(A_n)}(R)$$

- $G_{1,}G_2,\ldots,G_n$ – eine Liste von Attributen worauf wir gruppieren wollen
- F_i – Aggregatfunktion
- A_i – Name eines Attributes

Aggregierungsfunktionen - Beispiel

Relation R:

A	B	C
a	2	5
b	3	3
a	4	4

$$\vartheta_{sum(C)}(R) \qquad \Rightarrow 12$$

Outer Join

- Erweiterung von Join-Operationen:
  - **Left Outer Join ⟕** - alle Tupel aus der linken Relation, die keinen Join-Partner in der rechten Relation haben, werden trotzdem ausgegeben
  - **Right Outer Join ⟖** - alle Tupel aus der rechten Relation, die keinen Join-Partner in der linken Relation haben, werden trotzdem ausgegeben
  - **Full Outer Join ⟗** - alle Tupel sowohl der linken als auch der rechten Relation, die keinen Join-Partner haben, werden trotzdem ausgegeben
- Null-Werte werden benutzt:
  - Tupeln aus der Relation R, die keinen Join-Partner in der Relation S hatten enthalten Null-Werte für die entsprechenden Spalten der Relation S
  - Ein Null-Wert heißt unbekannt oder inexistent
  - Alle Vergleiche mit einem Null-Wert werden in der Regel als FALSE bewertet

Outer Join - SQL

- **RIGHT JOIN** (alternativ **RIGHT OUTER JOIN**)

```
SELECT *
FROM Studenten RIGHT JOIN Studiengang
  ON Studenten.SgNr = Studiengaenge.SgNr
```

- **LEFT JOIN** (alternativ **LEFT OUTER JOIN**)

```
SELECT *
FROM Studiengaenge LEFT JOIN Studenten
  ON Studenten.SgNr = Studiengaenge.SgNr
```

- **FULL OUTER JOIN**
  - Nicht in allen DB-Systemen verfügbar (z.B. MySQL nicht)

Datenbankänderungen

- Der Inhalt der Datenbank kann durch folgenden Operationen geändert werden:
  - Löschen: $R \leftarrow R - E$
  - Einfügen: $R \leftarrow R \cup E$
  - Aktualisierung/Updating: $R \leftarrow \pi_{F1, \ldots, Fn}(R)$