

DATENBANKEN

grundlegende Begriffe

- ◆ *Objekte* bilden die elementare Grundlage unserer Betrachtung (= einzelne Tupel)
Wohlunterscheidbarkeit von der Definition abhängig: 100 Nägel = 1 Objekt mit Eigensch. 'Anzahl'=100
- ◆ *Beziehungen* sind über Objekten oder anderen Beziehungen definiert und entstehen durch In-Bezug-Setzen von Objekten. (z.B. Kursbelegung durch Tabelle Kurs und Student)
- ◆ Menge der Objekte und Beziehungen bilden eine sog. Miniwelt.
- ◆ Fremdschlüssel: Objektreferenzen in Beziehungen
- ◆ Datenbank: strukturierte Menge dauerhafter Daten. Sie geben Informationen über einen Zustand aller Objekte und Beziehungen eines Anwendungsbereiches (Miniwelt)
- ◆ Relationen: Strukturierungskonzept einer (relationalen) Datenbank
- ◆ In relationalen Datenbanken werden Objekte und Beziehungen durch einzelne *Tupel* repräsentiert. Die Eigenschaften der Objekte und Beziehungen ergeben die *Attribute*.
- ◆ Anwendungsprogramme kommunizieren mit einer Datenbank, indem sie Anfragen über gespeicherte Zustände der Miniwelt stellen und diese Zustand durch Ändern, Einfügen oder Löschen von Daten verändern.
- ◆ Anfragen (DB-Anfragesprache) werden durch mengenwertige, deklarative Ausdrücke formuliert: Das Ergebnis einer Anfrage ist eine *Menge* von Tupeln. Die Anfrage definiert, was für Zusammenhänge aus den Daten der Datenbank gebildet werden sollen, ohne dass die algorithmische Vorgehensweise hierzu spezifiziert werden muss.
SQL ist eine Weise diese Ausdrücke komfortabler (nicht nur algebraisch) zu formulieren
- ◆ Die Daten einer Datenbank sind global. Datenbanksysteme besitzen einen Schutzmechanismus / Zugriffsberechtigung.
- ◆ Transaktion: Eine Ausführung (Prozess) eines Anwendungsprogramms über einer Datenbank (=abgeschlossener Verarbeitungsschritt innerhalb der Anwendung).
Sie transformieren einen gegebenen Datenbankzustand in einen neuen.
Problematik: bspw. Überweisung. T1 = Geld abbuchen; Absturz vor T2 = Geldgutschrift
oder 2 Händler verkaufen restliche Lagermenge zwei mal
oder: 2 Anwender erhöhen Ausgangswert '10' um jeweils 1 und laden dazu den scheinbar aktuellen Wert '10' in den jeweils lokalen Arbeitsspeicher, erhöhen und speichern ihn.
Lösung: Sperung (jedoch nur minimal vieler Objekte, damit ander user arbeiten können)
- ◆ Datenbanken enthalten Integritätsbedingungen, getrennt von den einzelnen Programmen und Daten, die die Daten in jedem Zustand der Datenbank erfüllen müssen.
- ◆ ACID-Eigenschaften der Transaktionen
 - Atomicity: Änderungen werden als atomar betrachtet und daher entweder vollständig oder gar nicht vorgenommen (all-or-nothing-Prinzip)
 - Consistency: Eine Transaktion bewirkt immer einen konsistenten Zustandsübergang
Inkonsistente Zwischenzustände sind zulässig.
Bspw. muss jeder Student in genau einer Übungsgruppe eingetr. sein.
Beim Wechsel der Gruppe ist er kurzzeitig in zwei oder keiner Gruppe.
Da die Umbuchung eine atomare Anfrage darstellt, entsteht keine Ink.
Verletzt eine Transaktion die Integrität, so wird sie **zurückgesetzt** (d.h. die durch sie bewirkten Effekte werden wieder entfernt).
 - Isolation: Datenbanksystem simuliert dem Anwender den Einbenutzerbetrieb.
Zeitlich verzahnte Transaktionen anderer Benutzer bleiben verborgen.
 - Durability: Wenn eine Transaktion erfolgreich ihr Ende erreicht hat, dann sind alle von ihr verursachten Änderungen dauerhaft (persistent).

DATENBANKEN

grundlegende Begriffe

- ◆ Ein Datenbankmanagementsystem DBMS ist erforderlich und charakterisiert durch:
 - Programme die alles integriert/automatisch verwalten, sicher, effizient und bequem sind.
 - globaler persistenter Datenbestände
 - gewährleisteter Konsistenz der Daten
 - weitgehende Unabhängigkeit zw. Definition der Daten und ihrer Abspeicherung
 - deklarative, mengenorientierte Anfragespr. ergänzt um einen Anfrageoptimierer
 - kontrollierten Mehrbenutzerbetrieb, Schnittstellen liefern
 - Datensicherheit bei Fehlerfällen
 - Schutz vor Missbrauch der Daten.
- ◆ Datenbanksystem DBS = DB + DBMS
Die Zugriffe der Transaktionen erfolgen nur über das DBMS und nicht direkt über die DB!
- ◆ Warum sollten Integritätsbedingungen nicht in Anwendungs- bzw. Transaktionsprogrammen selbst realisiert werden, sondern zentral durch die Datenbankverwaltungssoftware (DBVS) verwaltet und geprüft werden ?
 - Neue Integritätsbedingungen gelten für alle Daten und müssen von allen Anwendungsprogrammen berücksichtigt werden.
 - Die Einführung neuer Integritätsbedingungen führt zu Änderungen in allen betroffenen Anwendungsprogrammen.
 - Es muss garantiert/überprüft werden, dass sie überall richtig implementiert sind.
- ◆ Relationen/Datentypen werden mittels *Datendefinitionssprache* DDL definiert
--> Definition/Schema einer Relation (im Gegensatz zum akt. Inhalt/Zustand/Instanz)
- ◆ Relationsschemata (zeitunabhängig): **Student**(MatrNr:INTEGER,Name:STRING)
- ◆ Zustandsübergänge/Operationen auf Datentypen werden mittels Anfragesprachen bzw. *Datenmanipulationspr.* DML programmiert --> Instanzen (Menge von Tupeln)
- ◆ Werte der Instanzen (ungenau gesagt: Relationen) sind zeitabhängig

DATENBANKEN

Constraints

- ◆ Integritätsbed. (konsistente Zustände) werden mittels Anfrageausdrücken formuliert. Bedingungen (Constraints) werden als Teil der DDL unter mit Hilfe der DML erstellt. Es existieren sowohl **statische**, (mit einem festen Zustand der DB) als auch **dynamisch (temporale) Bedingungen** (einen Zustandsübergang betreffend). Man bezeichnet sie auch als Integritätsbed. (integrity constraints), z.B. Studentenzahl > Prof.zahl, Semesterzahl steigend (Zustands-Übergang)
Constraints können in **strukturelle** (Typ, Wertebereichseinschränkungen, Bedingung NOT NULL, Schlüsselbedingungen, ref. Integrität) und **semantische** (anwendungsbezogen, z.B. Semesterzahl muss monoton zunehmen) Bedingungen eingeteilt werden.
Integritätsbedingungen werden als Teil des Schemas abgelegt und können mittels ADD und DROP einem Schema hinzugefügt, oder auch aus ihm entfernt werden.
- ◆ **Semantische** Bedingungen definieren mittels logischer Ausdrücke die zulässigen Instanzen einer Relation. Eine Instanz erfüllt eine semantische Bedingung, wenn dieser 'true' ergibt.
Belegung (MatrNr: INTEGER, Note: STRING) **CHECK** (Note IN ('1', '2', '3', '4', '5'))
CHECK (MatrNr > 1000000 AND MatrNr < 10000000)
- ◆ Prof (PersNr: int, Name: **string**) **CHECK** ((select count(*) from Prof) < (select count(*) from Student))
Die Bedingung Professoren < Studenten muss jedoch nicht an die Relation Prof gebunden werden, sondern kann mittels **ASSERTION** eigenständig für die gesamte Datenbankrelation formuliert werden. Andernfalls wird sie nur bei Änderungen in 'prof' geprüft. Folge von 'delete from student' ?
ASSERTION CHECK ((select count(*) from Prof) < (select count(*) from Student))

DATENBANKEN

Datenmodell

- ◆ DDL und DML zusammen bezeichnet man als ein *Datenmodell*. Dieses ermöglicht Zugang zu einer Datenbank auf abstrakten Ebene, da man nicht mit 0en und 1en arbeiten möchte. Dazu abstrahiert man eine Menge gleichartiger, zusammengehörender Daten zu einer *Datei*. Dies ist unzureichend, da eine Programmierung auf Dateien Abhängigkeiten von Datenformaten, oder auch speziellen effizienten Repräsentationstechniken mit sich bringt.
- ◆ 3 *Abstraktionsebenen*: physikalische, konzeptuelle und externe Ebene.
- ◆ Auf der *physikalischen Ebene* wird festgelegt, wo und wie die Daten auf den Speichermedien physisch abgelegt werden und welche Hilfsstrukturen (Indexdateien) zur Effizienzsteigerung zur Verfügung gestellt werden sollen. Grundlage der konzept. Ebene.
 - Das Schema beschreibt die vom Administrator festgelegte physische Datenorganisation.
 - Das Schema beinhaltet alle Informationen über den Aufbau der abgespeicherten Daten, über die Speicherung der Daten und über die Zugriffspfade.
 - Die Erstellung des zugehörigen Schemas hängt von den statistischen Informationen über die Häufigkeit von Anwendungen, von Zugriffen auf Objekten, von Zeitbeschränkungen für Anwendungen usw. ab.
- ◆ Die Datentypen der *konzeptuellen Ebene* werden unabhängig von den Details einer physischen Implementierung definiert. Sie definieren das zugrunde liegende globale Gesamtmodell der Miniwelt als Grundlage für die Anwendungsprogrammierung.
 - Das Schema stellt einen stabilen Bezugspunkt für alle Anwendungen dar.
 - Es ändert sich nur, wenn sich die Vorstellungen über das Unternehmen ändern.
 - Das Schema schafft die Voraussetzung für die Datenunabhängigkeit der Anwendungsprogramme
 - Das Schema beschreibt die Gesamtheit der Daten, die innerhalb der Datenbank verwaltet werden.
 - Dazu gehören die Festlegung von Beziehungen, Integritätsbedingungen und Operationen.
 - Das zugehörige Schema hängt von den in der Realität existierenden Strukturen ab, bspw. den wesentlichen Daten eines Unternehmens und den wesentlichen Beziehungen zwischen ihnen.
- ◆ Die Datentypen der *externen Ebene* werden speziell für einzelne Benutzergruppen abgefasst und erlauben Sichten auf die konzeptuelle Ebene als Basis für die Anwendungsprogrammierung.
 - Das Schema beschreibt die einzelnen Sichten der Benutzer, so dass jeder Benutzer nur die für ihn relevanten Daten sieht.
 - Die Erstellung des entsprechenden Schemas hängt von den Anforderungen und auch den Befugnissen (Zugriffsrechte etc.) des jeweiligen Benutzers ab.
- ◆ Die Menge der Datentypen einer Abstraktionsebene bezeichnet man als Schema: physikalische Schema, konzeptuelle Schema, externe Schemata (Benutzersichten).
- ◆ *Physische Datenunabhängigkeit*: z.B. Sind Kundendaten in 20 Dateien verteilt. Das DBMS bildet Operationen der konzeptuellen Ebene automatisch in Operationen des Dateisystems ab. Änderungen der physikalischen Repräsentation bewirken keine Änderungen in den Anwendungsprogrammen.
- ◆ *Logische Datenunabhängigkeit*: Das DBMS bildet Operationen der externen Ebene automatisch in Operationen der konzeptuellen Ebene ab. Änderungen der konzeptuellen Repräsentation (z.B. Aufspaltung der Kundendaten in 2 Tabellen) bewirken keine Änderungen in den Anwendungsprogrammen und umgekehrt.

DATENBANKEN

Relationenalgebra

- ◆ $X = \{A_1, \dots, A_k\}$ ist endliche *Attributmenge*
- ◆ Jedes Attribut $A \in X$ besitzt einen nicht-leeren *Wertebereich* $\text{dom}(A)$
- ◆ Vereinigung aller Wertebereiche ergibt $\text{dom}(X) = \cup_{A \in X} \text{dom}(A)$
- ◆ Ein *Tupel* μ über X ist eine Abb. $\mu : X \rightarrow \text{dom}(X)$, wobei $\forall A \in X: \mu(A) \in \text{dom}(A)$.
 $\mu_1 = \{\text{id} \rightarrow 1, \text{Name} \rightarrow \text{Bernauer}\}$
- ◆ $\text{Tup}(X)$ ist die Menge *aller* Tupel über X
- ◆ Eine *Relation* r über X ist eine *endliche* Menge von Tupeln: $r \subseteq \text{Tup}(X)$.
- ◆ $\text{Rel}(X)$ ist die unendliche Menge *aller* Relationen über X
- ◆ $r \in \text{Rel}(X)$ ist eine *Instanz* zu X
- ◆ Sei R ein Relationsbezeichner. Ein (Relations)-*Schema* zu R hat die Form $R(X)$
 $= R(\{A_1, \dots, A_k\}) = R(A_1, \dots, A_k) = R(A_1:\text{dom}(A_1), \dots, A_k:\text{dom}(A_k))$ mit X als endliche Attributmenge bzw. sog. *Format* des Schemas und Stelligkeit k
- ◆ Sei $R(X)$ Rel.-Schema. Ein *Schlüssel* K zu R ist eine Teilmenge des Formates, $K \subseteq X$
Ein Schlüssel ist eine *minimale* Teilmenge K der Attribute eines Rel.Schemas, mit der die Tupel *aller* zulässigen Instanzen eindeutig identifiziert werden können. (ist bspw. nur eine sander in der *aktuellen Instanz* vorhanden, reicht name dennoch nicht als key, da weitere sander hinzukommen könnten) Wird ein Attribut der Schlüsselmenge gestrichen, kann es daher kein Schlüssel mehr sein. Eine Instanz erfüllt die Schlüsselbed., wenn es zu jedem Schlüssel nur einen jeweiligen Tupel aus der Instanz gibt. Gibt es mehrere Möglichkeiten, den Schlüssel zu wählen, spricht man von sog. Schlüsselkandidaten, beim Ausgewählten von Primärkey.
- ◆ Ist ein Schlüssel (oder auch mehrere) für ein Schema festgelegt, dann sind nur noch solche Instanzen zu diesem Schema zulässig, in denen keine zwei unterschiedlichen Tupel in sämtlichen Attributen eines Schlüssels gleich sind \rightarrow Einschränkung zulässiger Instanzen / spezielle Integritätsbedingung
- ◆ Fremdschlüssel: 2 Relationsschemata R, S . K Schlüssel von R und L Teilmenge der Attribute von S mit einer Bijektion zwischen K und L
Instanzen r und s erfüllen die Fremdschlüsselbedingung "L references K" wenn zu jedem Tupel $\mu \in s$ ein Tupel $\mu' \in r$ ex., so dass μ und μ' dieselben Werte bzgl. K und L haben.
- ◆ Ein (relationales) Datenbank-Schema R ist gegeben durch eine endl. Menge von RelationsSchemata,
 $R = \{R_1(X_1), \dots, R_m(X_m)\} = \{R_1, \dots, R_m\}$
- ◆ Eine Instanz I zu einem relationalen Datenbankschema $R = \{R_1, \dots, R_m\}$ ist eine Menge von endlichen Relationen $I = \{r_1, \dots, r_m\}$, wobei r_i Instanz zu R_i ... oder als Abbildung definiert: $I(R_i) = r_i$
- ◆ Eine Anfrage definiert zu einer gegebenen Instanz eine neue Relation als Antwort
Eine Anfrage ist eine Transformation, die angewendet auf eine Datenbank-Instanz I eine aus den Tupeln in I gebildete Relation definiert.
- ◆ Sei Q eine Anfrage ist $I(Q)$ die Antwort auf Q bezgl. der Instanz I
- ◆ Interpretation des *Nullwertes*: Wert existiert, jedoch noch nicht bekannt, Wert existiert erst in der Zukunft, Wert prinzipiell unbekannt, oder Attribut nicht anwendbar.
- ◆ Ein Relationsschema besteht aus einem Relationsbezeichner, einer Menge von Attributen, dem Format, und einem Schlüssel. Jedem Attribut ist ein Wertebereich zugeordnet.
- ◆ Eine endliche Menge von Relationsschemata ergibt ein Datenbankschema.
- ◆ Eine Instanz einer Datenbank enthält zu jedem Relationsschema eine endliche Relation, die den Bedingungen des Schemas genügt.

- ◆ Sei $R(X)$ Rel.Schema, mit $X = R\{A_1, \dots, A_k\}$, $\emptyset \subset Y$ sei Attribut(teil)menge $\subseteq X$, $\mu \in \text{Tup}(X)$ Tupel über X . Der Ausdruck $\mu[Y]$ heißt **Projektion des Tupels** μ auf Y :
 $\mu[Y] \in \text{Tup}(Y)$, wobei $\mu[Y](A) = \mu(A)$, $A \in Y$

Sei $r \subseteq \text{Tup}(X)$ eine Relation und $Y \subseteq X$.

Projektion der Relation r auf $Y = \pi[Y]r = \{\mu \in \text{Tup}(Y) \mid \exists \mu' \in r, \text{ so dass } \mu = \mu'[Y]\}$

$$\text{z.B. } r = \begin{array}{|c|c|c|} \hline A & B & C \\ \hline a & b & c \\ a & a & c \\ c & b & d \\ \hline \end{array} \quad \pi[A, C](r) = \begin{array}{|c|c|} \hline A & C \\ \hline a & c \\ c & d \\ \hline \end{array}$$

Die Anzahl der Tupel kann abnehmen, wenn das unterscheidende Attribut herausprojiziert wurde

- ◆ (atomare) **Selektionsbedingung** α bezgl. X : $A \theta B$, $A \theta a$, $a \theta A$.
 Ein Tupel $\mu \in \text{Tup}(X)$ erfüllt eine Selektionsbedingung α , wenn $\mu(A) \theta \mu(B)$, $\mu(A) \theta a$ oder $a \theta \mu(A)$ mit a Konstante aus dem Wertebereich. Atomare Selektionsbed. können mittels $\wedge, \vee, \neg, (,)$ zu Formeln verallgemeinert werden.

$$\text{z.B. } X = \{A, B, C\}$$

$$\mu_1 = (A \rightarrow 2, B \rightarrow 2, C \rightarrow 3)$$

$$\mu_2 = (A \rightarrow 2, B \rightarrow 3, C \rightarrow 2)$$

$$\alpha_1 = (A=B) \quad \alpha_2 = (C > 1)$$

$$\mu_1, \mu_2 \text{ erfüllen } \alpha_2, \mu_1 \text{ erfüllt } \alpha_1$$

$r \subseteq \text{Tup}(X)$ Relation, α Selektionsbed. zu X . $\sigma[\alpha]r$ heißt **Selektion** der Relation r bzgl. α . $\sigma[\alpha]r = \{\mu \in \text{Tup}(X) \mid \mu \in r \wedge \mu \text{ erfüllt } \alpha\}$.

$$\text{z.B. } r = \begin{array}{|c|c|c|} \hline A & B & C \\ \hline a & b & c \\ d & a & f \\ c & b & d \\ \hline \end{array} \quad \sigma[B=b](r) = \begin{array}{|c|c|c|} \hline A & B & C \\ \hline a & b & c \\ c & b & d \\ \hline \end{array}$$

- ◆ $X=Y$ Attributmengen, $r \subseteq \text{Tup}(X)$, $s \subseteq \text{Tup}(Y)$ Relationen.

Vereinigung: $r \cup s = \{\mu \in \text{Tup}(X) \mid \mu \in r \vee \mu \in s\}$.

Fügt die Zeilen zweier Tabellen mit gleicher Spaltenzahl (Tupel gleicher Struktur, d.h. Stelligkeit UND Typ) zusammen. Duplikate werden automatisch gestrichen.

SELECT * FROM r UNION SELECT * FROM s

Mit **UNION ALL** werden Duplikate zugelassen.

- ◆ **Differenz:** $r - s = \{\mu \in \text{Tup}(X) \mid \mu \in r, \text{ wobei } \mu \notin s\}$.
 $= \text{SELECT * FROM } r \text{ MINUS SELECT * FROM } s$
 $= \text{SELECT * FROM } r \text{ WHERE A NOT IN(SELECT A FROM } s)$

$$r = \begin{array}{|c|c|c|} \hline A & B & C \\ \hline a & b & c \\ d & a & f \\ c & b & d \\ \hline \end{array} \quad s = \begin{array}{|c|c|c|} \hline A & B & C \\ \hline b & g & a \\ d & a & f \\ \hline \end{array} \quad r \cup s = \begin{array}{|c|c|c|} \hline A & B & C \\ \hline a & b & c \\ d & a & f \\ c & b & d \\ b & g & a \\ \hline \end{array} \quad r - s = \begin{array}{|c|c|c|} \hline A & B & C \\ \hline a & b & c \\ c & b & d \\ \hline \end{array}$$

- ◆ **Join:** X,Y Attributmengen, XY Kurzschreibweise für $X \cup Y$, $r \subseteq \text{Tup}(X)$, $s \subseteq \text{Tup}(Y)$
Relationen **natürlicher Verbund** \diamond von r und s: $r \diamond s = \{\mu \in \text{Tup}(XY) \mid \mu[X] \in r \wedge \mu[Y] \in s\}$.
d.h. die Projektion auf den X-Anteil eines beliebigen Tupel aus der neuen Verbundtabelle ergibt einen Tupel aus der ursprünglichen Relation r, die Y-Projektion einen Tupel aus s.

$r = \begin{array}{ c c c } \hline \underline{A} & \underline{B} & \underline{C} \\ \hline 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 6 \\ \hline \end{array}$	$s = \begin{array}{ c c } \hline \underline{C} & \underline{D} \\ \hline 3 & 1 \\ 6 & 2 \\ 4 & 5 \\ \hline \end{array}$	$r \diamond s = \begin{array}{ c c c c c } \hline \underline{A} & \underline{B} & \underline{C} & \underline{(C)} & \underline{D} \\ \hline 1 & 2 & 3 & (3) & 1 \\ 4 & 5 & 6 & (6) & 2 \\ 7 & 8 & 6 & (6) & 2 \\ \hline \end{array}$
---	---	--

Seien X_i Formate. $X_1 \cap X_2 = \emptyset \Rightarrow r \diamond s = r \times s$ (Schnitt der *Formate* disjunkt, nicht der Inhalte)

d.h. das **kartesische Kreuzprodukt** erhält man durch den Verbund, wenn die Attributmengen $X = \{A, B, C\}$, $Y = \{D, E\}$ disjunkt sind. Ansonsten Umwandlung von $Y = \{A, B\}$ in $Y = \{A \rightarrow D, B \rightarrow E\}$

- ◆ **Allg. natürlicher Verbund** $\diamond_{i=1}^n r_i = \{\mu \in \text{Tup}(\cup_{i=1}^n X_i) \mid \mu[X_i] \in r_i\}$ (für n Relationen)
Der natürl. Verbund basiert auf der Gleichheit der Attribute (join where r.C = s.A)
Möchte man andere Operatoren verwenden (<, >, ...), benötigt man den θ -Verbund

- ◆ $X = \{A_1, \dots, A_k\}$, $Y = \{B_1, \dots, B_k\}$ Formate. δ ist eine umkehrbar eindeutige Abbildung von X nach Y, wobei $\text{dom}(A) = \text{dom}(\delta(A))$. Gilt $\delta(A) = B$, so schreiben wir $A \rightarrow B$.

Sei $r \subseteq \text{Tup}(X)$ Relation zu X.

- ◆ **Umbenennung** $\delta[X, Y]$ bezgl. r: $\delta[X, Y]r = \{\mu \in \text{Tup}(Y) \mid \exists \mu' \in r, \text{ so dass } \mu'(A_i) = \mu(\delta(A_i))\}$

z.B. $X = \{A, B, C\}$, $Y = \{D, E, C\}$, $\delta = \{A \rightarrow D, B \rightarrow E \text{ und } C \rightarrow C\}$

$$r = \begin{array}{|c|c|c|} \hline \underline{A} & \underline{B} & \underline{C} \\ \hline a & b & c \\ \hline \end{array} \quad \delta[X, Y]r = \begin{array}{|c|c|c|} \hline \underline{D} & \underline{E} & \underline{C} \\ \hline a & b & c \\ \hline \end{array} = [A \rightarrow D, B \rightarrow E]r$$

- ◆ **Basisoperatoren der Relationenalgebra:** Selektion, Projektion, Vereinigung, Differenz, Verbund und Umbenennung. Die Anwendung dieser Operatoren auf Relationen liefert als Ergebnis wiederum eine Relation. Die zulässigen Ausdrücke der Relationenalgebra können ausgehend von den Basisoperatoren induktiv definiert werden.

- ◆ Seien X_i Attributmengen und $r_i \subseteq \text{Tup}(X_i)$ Relationen.

Durchschnitt: Sei $X_1 = X_2$. $r_1 \cap r_2 = r_1 - (r_1 - r_2)$.

Oder durch den Verbund, wenn die Formate X_1, X_2 gleich sind: $r_1 \cap r_2 = r_1 \diamond r_2$

- ◆ **θ -Verbund.** Sei $X_1 \cap X_2 = \emptyset$ und sei α eine beliebige Selektionsbedingung über $X_1 \cup X_2$.
 $r \diamond_{\alpha} s = \sigma[\alpha](r \times s)$. Enthält α aussch. Gleichheitsvergl., redet man von einem *Equi-Verbund*.

- ◆ Seien X_1, X_2 Formate. $X_1 \subset X_2$, $Z = X_2 - X_1$, $r \neq \emptyset$

Division $r_1 \div r_2 = \{\mu \in \text{Tup}(Z) \mid \{\mu\} \times r_2 \subseteq r_1\} = \pi[Z]r_1 - \pi[Z](\pi[Z]r_1 \times r_2)$

$r = \begin{array}{ c c c c } \hline \underline{A} & \underline{B} & \underline{C} & \underline{D} \\ \hline a & b & c & d \\ a & b & e & f \\ b & c & e & f \\ e & d & c & d \\ e & d & e & f \\ a & b & d & d \\ \hline \end{array}$	$s = \begin{array}{ c c } \hline \underline{C} & \underline{D} \\ \hline c & d \\ e & f \\ \hline \end{array}$	$r \div s = \begin{array}{ c c } \hline \underline{A} & \underline{B} \\ \hline a & b \\ e & d \\ \hline \end{array}$
--	--	---

(b,c) kommt nicht in $(r \div s)$ vor, da $(r \div s) \times s$ sonst auch das Tupel (b,c,c,d) ergäbe (not in r!!!)

z.B. $\pi[\text{MatrNr}](\text{Belegung} \div \pi[\text{KursNr}]\text{Kurs})$

mit $X_1 = \{A,B,C,D\}$, $X_2 = \{C,D\}$, $Z = \{A,B\}$ folgt:

$$\begin{aligned}
r_1 \div r_2 &= r \div s = \pi[A,B]r_1 - \pi[A,B]((\pi[A,B]r_1) \times r_2) - r_1 \\
&= \{(a,b), (b,c), (e,d)\} - \pi[A,B](\{(a,b), (b,c), (e,d)\} \times \{(c,d), (e,f)\} - r_1) \\
&= \{(a,b), (b,c), (e,d)\} - \pi \\
[A,B](\{(a,b,c,d), (b,c,c,d), (e,d,c,d), (a,b,e,f), (b,c,e,f), (e,d,e,f)\} - r_1) \\
&= \{(a,b), (b,c), (e,d)\} - \pi[A,B](\{(b,c,c,d)\}) = \{(a,b), (b,c), (e,d)\} - \{(b,c)\} = \\
&= \{(a,b), (e,d)\}
\end{aligned}$$

Algebra als Anfragesprache (nicht wie bisher Ausdrücke über Instanzen/Relationen)

- ◆ Dazu ersetzen wir *in* den Ausdrücken die konkreten Relationen (z.B. $\{(a,b), \dots\}$) durch Relationsbezeichner (z.B. r). Der bisherige Ausdruck wird somit zu *Anfrage*
- ◆ Jede Anfrage Q definiert zu einer konkreten Instanz I eine Relation $I(Q)$ als *Antwortmenge* $Nicht\ Q(I)$, da eine Instanz eine Abbildung ist, die einem Rel.bezeichner eine Relation zuweist. Daher wird mit $I(Q)$ die Zuweisung auf Anfrageausdrücken erweitert.
- ◆ Es können nicht alle berechenbaren Transformationen über den Instanzen zweier Datenbankschemata mittels der Algebra ausgedrückt werden. Das bekannteste Beispiel für diese eingeschränkte Mächtigkeit ist das Problem, zu einer beliebigen binären Relation die transitive Hülle zu berechnen, also aus den Tupeln (a,b) , (b,c) den Wert (a,c) zu ermitteln (wenn die Größe der Relation unbekannt)
- ◆ Gegeben sei die Relation $R(\{A,B\})$ mit der folgenden Instanz $r = \{(1,2), (2,3)\}$
 - a) Geben Sie einen Algebra-Ausdruck an, der die transitive Hülle von r berechnet.
 $r \cup \delta[C \rightarrow B](\pi[A,C](r \hat{\cup} (\delta[A \rightarrow B, B \rightarrow C]r))) = \{(1,2), (2,3), (1,3)\}$
 - b) es existiert kein Ausdruck der relationalen Algebra, der für beliebige Instanzen einer solchen binären Relation R die transitive Hülle berechnet. Ist das obige Beispiel ein Widerspruch zu dieser Aussage?
Nein. Obiger Ausdruck berechnet zur Instanz $r = \{(1,2), (2,3), (3,4)\}$ die Relation $\{(1,2), (2,3), (3,4)\} \cup \delta[C \rightarrow B](\pi[A,C](\{(1,2,3), (2,3,4)\})) = \{(1,2), (2,3), (3,4), (1,3), (2,4)\}$
Der Ausdruck $(1,4)$ fehlt, um die transitive Hülle zu berechnen.

Zulässige Ausdrücke der Relationsalgebra

- ◆ gegeben: Relationsschema $\mathbf{R} = \{R_1(X_1), \dots, R_k(X_k)\} = \{R_1, \dots, R_k\}$ mit Instanzen $I(R_i) = r_i \subseteq \text{Typ}(X_i)$
- ◆ Jedem zulässigem Ausdruck Q wird ein Format X_Q zugeordnet
- ◆ I weist Q die Relation $I(Q) = \text{Typ}(X_Q)$ zu
- ◆ **induktive Definition der zulässigen Ausdrücke und Relationen:**
 - I.A.:** Jeder **Relationsbezeichner** \mathbf{R} ist ein zulässiger Ausdruck der Relationenalgebra.
Das Format des Ausdrucks \mathbf{R} und $I(\mathbf{R})$ sind bereits definiert.
Sei A ein Attribut und $a \in \text{dom}(A)$ eine Konstante.
Die **Relationskonstante** $\{a\}$ ist ein Ausdruck der Relationenalgebra.
Das Format von $\{a\}$ ist $\{A\}$ und es gilt $I(\{a\}) = \{a\}$
Ausdrücke der Form \mathbf{R} und $\{a\}$ sind die **atomaren** Ausdrücke der Relationenalgebra.
 - I.S.:** Die Menge der zulässigen Ausdrücke der Relationenalgebra ist die Menge aller atomarer Ausdrücke oder die durch Anwendung folgenden Regeln entstanden:
- ◆ **Vereinigung**
Die Formate zweier Anfragen Q_1, Q_2 seien gleich: $X_{Q_1} = X_{Q_2}$
 $Q = (Q_1 \cup Q_2)$ ist ein zulässiger Ausdruck mit dem Format $X_Q = X_{Q_1}$ und
Ergebnisrelation $I(Q) = I(Q_1) \cup I(Q_2)$
- ◆ Differenz, Projektion, Verbund und Umbenennung
- ◆ Eine Anfragesprache, die mindestens die Mächtigkeit der Algebra besitzt, heißt **relational vollständig**. Damals gab es Anfragesprachen die den Verbundoperator nicht beherrschten und daher nicht wie das heute SQL rel. Vollständig waren.
Die Algebra ist nicht Turing-vollständig (transitive Hülle nicht berechenbar, Anzahl der Tupel einer Relation lässt sich nicht mit diesen Basisoperatoren ermitteln, etc.)

Zwei Ausdrücke der Algebra Q, Q' heißen **äquivalent**, gdw. für jede Instanz I gilt: $I(Q) \equiv I(Q')$

Wozu wichtig? Finden von Anfragen Q' , die effizienter und schneller ausgewertet werden können
 Sei $\text{attr}(\alpha)$ die in einer Selektionsbedingung α verwendete Attributmenge und seien R, S, T Rel.bezeichner mit Formaten X, Y, Z .

- ◆ $R \diamond R = R$
- ◆ $R \diamond S = S \diamond R$
- ◆ $(R \diamond S) \diamond T = R \diamond (S \diamond T)$
- ◆ $X=Y \Rightarrow R \cap R = R \diamond R$
- ◆ $\pi[Z](\pi[Z_2]R) = \pi[Z_1 \cap Z_2]R$ mit $Z_1 \subseteq Z_2 \subseteq X$
 zweite Ausdruck effizienter, da die Projektion auf die kleinere Attributmenge sofort stattfindet
- ◆ $\sigma[\alpha](R \diamond S) = (\sigma[\alpha]R) \diamond S$ mit $\text{attr}(\alpha) \subseteq X, \text{attr}(\alpha) \cap Y = \emptyset$
 zweite Ausdruck effizienter, da große Zwischenergebnisse vorherige Selection verhindert werden
- ◆ $\pi[Y](\sigma[\alpha]R) = \sigma[\alpha](\pi[Y]R)$ nur wenn $\text{attr}(\alpha) \subseteq Y \subseteq X$, da sonst die Attribute, auf die sich die Selection bezieht, zuvor herausprojiziert würden

- ◆ Seien $R(A,B,C), S(A,E,F)$ und $T(A,H)$ Relationsschemata. Zeigen Sie die Äquivalenz der Ausdrücke $\pi[E,H](\sigma[B=10]((R \diamond T) \diamond S))$ und $\pi[E,H](\sigma[B=10]R \diamond ((\pi[S,E]S) \diamond T))$
 - Join \diamond ist assoziativ: $\pi[E,H](\sigma[B=10](R \diamond (T \diamond S)))$
 - B ex. nur in R: $\pi[E,H](\sigma[B=10]R) \diamond (T \diamond S)$
 - F wird nicht benötigt: $\pi[E,H](\sigma[B=10]R \diamond (T \diamond (\pi[A,E]S)))$
 - Join \diamond ist kommutativ: $\pi[E,H](\sigma[B=10]R \diamond ((\pi[A,E]S) \diamond T))$

- ◆ Seien $R(X), S(Y)$ Relationsschemata. Zeigen oder widerlegen Sie die Äquivalenz
 - Sei $X \cap Y = \emptyset$. Dann gilt $(R \diamond S) \div S = R$
 $(R \diamond S) \div S = (R \times S) \div S$ Rel. $R(X), S(Y)$ haben keine gem. Attribute
 $= \pi[R](R \times S) \setminus \pi[R](\pi[R](R \times S) \times S) \setminus R \times S$ nach Definition der Division
 $= R \setminus \pi[R](R \times S) \setminus (R \times S) = R \setminus \pi[R](\emptyset) = R$
 - Sei $X=Y$ und $Z \subseteq X$. Dann gilt: $\pi[Z](R-S) = \pi[Z]R - \pi[Z]S$
 z.B. $X=Y=\{A, B\}, Z \subseteq X = \{A\}, R = \{(1,2)\}$ und $S = \{(1,3)\}$
 $\pi[A](R-S) = \{<1>\}$ und $\pi[A](R) - \pi[A](S) = \emptyset$
 - Sei $Y \subseteq X$ und $Z = X \setminus Y$. Dann gilt $(R \div S) \times S \subseteq R$ und $R \div S$ ist maximal im Sinn, dass für ein beliebiges Schema $T(Z)$ gilt: $T \times S \subseteq R \Rightarrow T \subseteq R \div S$

DATENBANKEN

Relationenkalkül

Die (R-)Formeln des **Relationenkalküls** werden aus Konstanten, Variablen, Relationsbezeichnern, Junktoren (Negation, Konjunktion, Disjunktion) \neg, \wedge, \vee , Quantoren \forall, \exists und Hilfszeichen (,), gebildet.

- ◆ Formate im bzgl. dem Kalkül sind immer geordnet (Reihenfolge der Attribute)
- ◆ $R(X,Y) \wedge X=Y$ alle Tupel aus R wo $X=Y$ gilt
- $R(X,Y) \wedge S(Y)$ alle XY-Paare aus R, deren Y in S existiert
- $R(X,Y) \vee (X=b \wedge S(Y))$ alle Tupel aus R oder alle ('b',Y), deren Y in S existieren
- $R(X,Y) \wedge \neg(X=b \wedge S(Y))$ alle Tupel aus R deren X nicht b ist oder in S liegt
- $\forall Y(S(Y) \Rightarrow R(X,Y))$
- ◆ Sei R ein Relationsbezeichner der Stelligkeit k und a_1, \dots, a_k Konstanten oder Variablen.
 $R(a_1, \dots, a_k)$ ist eine (**atomare**) R-Formel, z.B. $R(X,Y, '1981')$, $R(X,X,Y)$, ...
- ◆ Eine Selektionsbedingung ist der Form $X\theta Y$, $X\theta a$ oder $a\theta X$ mit X,Y als Variablen,
a als Konstante und $\theta \in \{=, \neq, <, \leq, \geq, >\}$ als Vergleichsoperator und ist eine **atomare** R-Formel.
- ◆ F sei R-Formel $\Rightarrow \neg F$ ist R-Formel (nicht atomar).
- ◆ X sei Variable, F sei R-Formel, die X enthält, jedoch keinen Ausdruck der Form $\exists X$, bzw. $\forall X$.
X heißt in diesem Fall **frei** (keine Quantoren) in der R-Formel F. Anderenfalls **gebunden** in F
- $\exists X$ F ist eine (\exists -quantifizierte) R-Formel.
- $\forall X$ F ist eine (\forall -quantifizierte) R-Formel.
- F ist der Wirkungsbereich des \forall -, bzw. \exists -Quantors.
- ◆ Seien F und G R-Formeln und sei v_F , bzw. v_G die Menge der in F, bzw. G vorhandenen Variablen, wobei die Variablen in $v_F \cap v_G$ sowohl in F als auch in G frei sind.
Die Konjunktion $(F \wedge G)$ ist eine R-Formel.
Die Disjunktion $(F \vee G)$ ist eine R-Formel
Für Konjunktion und Disjunktion gilt das Assoziativgesetz
- ◆ >>> Im Ergebnis kommen nur freie Variablen vor.
Der \exists -Quantor erfüllt nur den Zweck, Variablen aus dem Ergebnis herauszuprojeziert
- ◆ Beispiel: Relationsschemata $R(A,B)$, $S(B)$ mit Instanzen $r = \{(a,a), (d,a), (c,b)\}$, $s = \{b,c\}$
Anfrage $\exists Y R(X,Y)$ ergibt Menge aller X, zu denen ein Y existiert: $\{a,d,c\}$.
 $R(X,Y) \wedge S(Y)$ ergibt $\{(c,b)\}$
- ◆ Anfrage $R(X,Y)$ zu einem Schema $R(X,Y,Z)$ wäre syntaktisch nicht korrekt.
- ◆ Eine Kalkül-Anfrage Q über einem Datenbank-Schema R im Relationenkalkül hat die Form: $Q = \{(a_1, \dots, a_n) | F\}$, F ist R-Formel zu R und a_1, \dots, a_n Variablen und Konstanten.
Menge der Variablen unter den a_i muss der Menge der freien Variablen in F entsprechen.
- ◆ Format $\{A_1, \dots, A_n\}$ für die Menge der Antworten: $\{(A_1:a_1, \dots, A_n:a_n) | F\}$.
- ◆ Beispiele: $R(A,B)$ und $S(B, C)$ gegebene Relationsschemata. r, s seien hierzu betrachtete Instanzen.
 - $\pi[A]R$ = $\{(A:X) | \exists Y R(X,Y)\} = \{X | \exists Y R(X,Y)\}$
 - $\pi[A](\sigma[B = 5]R)$ = $\{(A:X) | \exists Y R(X,5)\}$
 - $\sigma[A=B]R$ = $\{(A:X, B:Y) | R(X,Y) \wedge X=Y\}$
 - $R \oslash S$ = $\{(A:X, B:Y, C:Z) | R(X,Y) \wedge S(Y,Z)\}$
 - $R \cup \delta[C \rightarrow A]S$ = $\{(A:X, B:Y) | R(X,Y) \vee S(Y,X)\}$
 - $R - \delta[C \rightarrow A]S$ = $\{(A:X, B:Y) | R(X,Y) \wedge \neg S(Y,X)\}$
 - $R \div \pi[B]S$ = $\{(A:X) | \forall Y \exists Z (S(Y, Z) \Rightarrow R(X,Y))\}$
 - $R \cup (\{1\} \times \pi[B]S)$ = $\{(X,Y) | R(X,Y) \vee (X=1 \wedge S(Y))\}$
- ◆ F ist R-Formel mit Variablenmenge V_F . Eine Variablenbelegung v zu F ist eine Funktion

über V_F : $v: V_F \rightarrow \text{dom}$.

Erweiterung von v um die Identität für Konstanten, d.h. für Konstanten a gilt $v(a) = a$
 v angewendet auf eine [a) Variable | b) Konstante] ergibt [a) eine | b) die selbe] Konstante

- ◆ **Atomare Kalkülanfragen:** $Q = \{(a_1, \dots, a_n) \mid R(a_1, \dots, a_n)\}$ über Rel.Schema $R(A_1, \dots, A_n)$
- ◆ Sei i Instanz zu R und $F = R(A_1, \dots, A_n)$. Die **Antwortmenge** zu Q bezgl. $(i, Q(i))$, ist definiert zu:
 $Q(i) = \{(v(a_1), \dots, v(a_n)) \mid v \text{ Variablenbelegung zu } V_F, \text{ so dass } (v(a_1), \dots, v(a_n)) \in i\}$
- ◆ Sei $Q = \{(a_1, \dots, a_n) \mid F\}$ eine beliebige Kalkülanfrage, wobei a_i Variablen und Konstanten.
 Die Antwort zu Q bzgl. einer gegebenen Instanz I ist $Q(I)$ wie folgt:
 $Q(I) = \{(v(a_1), \dots, v(a_n)) \mid v \text{ Variablenbelegung zu } V_F \text{ so, dass } F \text{ unter } v \text{ wahr bzgl. } I\}$

Beispiel	$r = \begin{array}{cc} \underline{A} & \underline{B} \\ 1 & 2 \\ 2 & 2 \\ 2 & 1 \end{array}$	$s = \begin{array}{cc} \underline{C} & \underline{D} \\ 1 & 1 \\ 1 & 2 \\ 3 & 1 \end{array}$	$Q \Rightarrow \begin{array}{cccc} \underline{A} & \underline{B} & \underline{C} & \underline{D} \\ 1 & 2 & 1 & 1 \\ 1 & 2 & 1 & 2 \\ 2 & 2 & 1 & 1 \\ 2 & 2 & 1 & 2 \end{array}$
----------	--	--	---

$R(A,B), S(C,D)$ Relationsschemata mit Instanzen r, s .

$Q = \{(A:X, B:Y, C:V, D:W) \mid R(X,Y) \wedge S(V,W) \wedge Y > V\} = \text{Anfrage}$

- ◆ Anfrage $Q = \{X \mid \neg R(X)\} = \{\text{Menge aller } X, \text{ die nicht in } R \text{ enthalten sind}\}$, wobei $I(R) = \{1\}$.
 Für $\text{dom}(A) = \text{INTEGER}$ ist die Antwort $Q(I)$ im uneingeschr. Relationenkalkül *unendlich*, da Integer ein unendlicher Wertebereich aller natürlicher Zahlen ist (String unendliche Menge aller Strings).
 Dies ist unerwünscht, da unendliche Relationen in der Realität unbrauchbar.
 Triviale Vermeidung durch $Q = \{X \mid \neg R(X) \wedge R(X)\}$
- ◆ Anfrage $Q = \{(X, Z) \mid \exists Y (R(X,Y) \vee S(Y, Z))\}$, wobei $I(R) = \{(1,1)\}$ und $I(S) = \emptyset$.
 $Q(I)$ unendlich: $\{(1, \text{irgendwas aus dom}(Z))\}$, z.B. $\{(1,1), (1,2), (1,3), \dots\}$ da die AnfrageFormel bereits durch die erste Komponente $(1, ..)$ wahr gemacht wird
- ◆ Sei $Q := \{(a_1, \dots, a_n) \mid F\}$, I eine Instanz zu R und adom diejenige Menge, die gerade alle Konstanten in Q und alle Konstanten aus I enthält. adom ist der aktive *Wertebereich* von Q . Da die Konstanten in der Anfrage und die der (endlichen) Relation endlich sind, ist somit auch adom und damit $Q(I)$ endlich
- ◆ Q heißt *wertebereichsunabhängig*, wenn für jede beliebige Menge $D \supset \text{adom}$ gilt: $Q(I, \text{adom}) = Q(I, D)$
 d.h. wenn bei jeder beliebigen Obermenge D zu adom , die selben Ergebnisse herauskommen, wie bei adom selbst. Wertebereichsunabhängige Anfragen erzeugen nur endlich große ErgebnisRelationen.
 Wie kann man herausbekommen ob Q wertebereichsunabhängig ist? Nicht-entscheidbares Problem!
 Daher werden sog. sichere Formeln als hinreichende Bedingung eingeführt, die die Menge der Kalkülanfragen zwar (sicher) einschränken, diese jedoch äquivalent zur uneingeschränkten Algebra bleibt.
- ◆ R-Formel F sicher
 $\Rightarrow F$ auch wertebereichsunabhängig
 $\Rightarrow F$ enthält **keine** \forall -Quantoren. (jedoch äquivalent: ' $\forall X$ gilt' \equiv 'es gibt kein X für das *nicht* gilt')
 $\Rightarrow (F_1 \vee F_2 \text{ Teilformel von } F \Rightarrow F_1 \text{ und } F_2 \text{ müssen dieselben freien Variablen besitzen})$
 - Eine Teilformel G einer Formel F heißt **maximal konjunktiv**, wenn F keine konjunktive Teilformel der Form $H \wedge G$ oder $G \wedge H$ enthält. Sei $F_1 \wedge \dots \wedge F_m$ eine maximal konjunktive Teilformel von F (größte ver-undete Teilstücke, die sich nicht mehr verlängern lassen).
 Alle freien Variablen X müssen im folgenden Sinn begrenzt (Ersetzungen endl. begrenzt) sein:
 - Falls X in einer Formel F_j frei ist, wobei F_j weder ein Vergleichsausdruck noch negiert ist, dann ist X begrenzt.
 - Falls F_j die Form $X=a$ oder $a=X$ hat und a ist eine Konstante, dann ist X begrenzt.
 - Falls F_j die Form $X=Y$ oder $Y=X$ hat und Y ist begrenzt, dann ist auch X begrenzt.
- ◆ $\{(X,Y) \mid X=Y \vee R(X,Y)\}$ ist nicht sicher, da die max.konj. Teilformel $X=Y$ nicht begrenzt ist
 $\{(X,Y) \mid X=Y \wedge R(X,Y)\}$ ist sicher, da es eine einzige max.konj. Teilformel gibt, die begrenzt ist.
 $\{(X,Y,Z) \mid R(X,Y,Z) \wedge \neg(S(X,Y) \vee T(Y,Z))\}$ ist nicht sicher, da die freien Variablen X,Y,Z disjunktiv verknüpft sind aber nicht in beiden Teilformeln auftauchen
 äquivalenten und sicher: $\{R(X,Y,Z) \wedge \neg S(X,Y) \wedge \neg T(Y,Z)\}$
 Division = $\{X \mid \forall Y (S(Y) \Rightarrow R(X,Y))\} = \{X \mid \neg \exists Y (S(Y) \wedge \neg R(X,Y))\} = \text{nicht sicher, wegen dem}$

all-Quantor und der unbegrenzten, freien Variable X, jedoch ex. die äquivalente und sichere Formulierung $\{X \mid \neg \exists Y (S(Y) \wedge \neg R(X, Y)) \wedge \exists Z R(X, Z)\} \wedge \exists U \exists V R(U, V) \wedge X=U$
 Durch Hinzunahme von $\exists Z R(X, Z)$ wird X beschränkt ohne das Ergebnis zu verändern
 $R(X, Z)$, $R(U, V)$, $\exists V R(U, V)$, $[S(Y) \wedge \neg R(X, Y) \wedge \exists Z R(X, Z)]$ sind max.konj. Teilformeln, da sie nicht mehr konj. verlängert werden können, ohne als nächstes das stärker bindende \exists zu verwenden

- ◆ Sicherheit ist rein syntaktisch definiert!
- ◆ Welche folgender Formeln im Relationenkalkül ist wertebereichsunabhängig oder sicher?
 Sei DOM_A die Menge der natürlichen Zahlen und $DOM_B = \{1, \dots, 5\}$
 - a) $F(X, Z) = (\exists Y)(p(X, Y) \vee q(Y, Z))$
 nicht sicher, da die Teilformeln p und q nicht dieselben freien Variablen besitzen.
 nicht wbua, da die Antwortmenge sich unterscheidet, je nachdem ob bzgl. X,Z DOM_A oder DOM_B zugrundeliegt
 - b) $F(X, Z) = (\exists Y)(p(X, Y) \wedge q(Y, Z))$
 sicher, da keine Negation oder Disjunktion auftritt und alle freien Variablen begrenzt sind
 aus der Sicherheit folgt die Wertebereichsunabhängigkeit
 - c) $F(X, Z) = (\exists Y)(p(X, Y) \wedge \neg q(Y, Z))$
 nicht sicher, da die freie Variable Z unbegrenzt ist.
 Nicht wbua, da die Antwortmenge abhängig, ob bzgl. Z DOM_A oder DOM_B zugrundeliegt
 - d) $F(X, Y) = (\exists Z)(r(X, Z) \wedge q(Z, Y) \wedge \neg p(X, Z))$
 sicher, da alle freien Variablen begrenzt sind. Aus der Sicherheit folgt die Wertebereichsunabh.
 - e) $F(X, Y) = (\exists Z)(r(X, Z) \vee p(X, Y)) \wedge (\exists Z)(\neg s(X, Y, Z))$
 nicht sicher: $(\exists Z)(r(X, Z) \vee p(X, Y))$ ist unsicher, da die Teilformeln r und p nicht dieselben freien Variablen besitzen.
 - f) $F(X, Y) = (\forall Z)(p(X, Y, Z))$
 nicht sicher, da die Formel einen all-Quantor enthält. Nicht wertebereichsunabhängig.
 Umformung: $F(X, Y) = \neg(\exists Z)\neg(p(X, Y, Z))$
- ◆ Relationenalgebra und -kalkül sind beide grundlegende Sprachkonzepte relationaler Datenbanken.
- ◆ Die Basisoperatoren der Relationenalgebra sind Vereinigung \cup , Differenz -, Projektion π , Selektion σ , Verbund \Join und Umbenennung δ . (grundlegend für Datenbankanfragesprachen) Eine Anfragesprache, die mindestens diese Operatoren erlaubt, heißt relational vollständig.
 Die Relationenalgebra ist nicht turing-vollständig. Es können somit für gegebene Datenbankschemata R_1, R_2 nicht alle berechenbaren Transformationen von der Menge der Instanzen zu R_1 in die Menge der Instanzen zu R_2 ausgedrückt werden.
- ◆ Eine Anfrage im Relationenkalkül wird als logische Formel über Relationsbezeichnern einer Menge von Relationsschemata geschrieben. Die Antwort zu einer Anfrage ergibt sich aus denjenigen Belegungen der freien Variablen der Formel, die die Formel bezgl. der betrachteten Instanzen der Relationsschemata wahr machen.
- ◆ Ist die Anfrageformel Q sicher, so ist die Antwort dieser Kalkülanfrage immer endlich!
- ◆ Der sichere Relationenkalkül und die Relationenalgebra haben die gleiche Mächtigkeit, d.h. zu jedem Ausdruck Q der Relationenalgebra existiert eine äquivalente sichere Anfrage Q' des Relationenkalküls und umgekehrt. Q und Q' definieren für jede beliebige Instanz I dieselbe Relation.
 Der Relationenkalkül ist somit relational vollständig.

DATENBANKEN

Anfragesprache SQL

- ◆ Ein Anfrageausdruck in **SQL** besteht aus einer SFW-Klausel

```

SELECT  A1, ..., An    (...Attribute der Ergebnisrelation, entspr. π der Algebra)
FROM    R1, ..., Rm    (...benötigte Relationen)
WHERE   F                (...Auswahlbedingung, entspr. σ in der Algebra)
GROUP BY B1, ..., Bk    (...Gruppierungsattribute)
HAVING G                (...Gruppierungsbedingung)
ORDER BY H              (...Sortierordnung)
äquivalenter Algebraausdruck π[A1, ..., An](σ[F](r1 × ... × rm))
äquivalenter Kalkülausdruck {(A1:Xi,1, ..., An:Xi,n) | R1(X1) ∧ ... ∧ Rm(Xm) ∧ F}
    
```

Auswertungsreihenfolge: from vor < where < group < having < order < select-klausel

Im Gegensatz zur Algebra (Mengen) können in SQL (Multimengen) nun Werte mehrmals auftreten (dupl.)
 Bei der Projektion verringerte sich evtl. die Ergebnismenge, wenn das unterscheidende Attribut herausprojiziert wurde. In SQL bleibt die Anzahl der Tupel zunächst gleich, sofern Duplikate nicht explizit verhindert werden

```

[einfache Anfrage]          SELECT SName FROM Stadt  oder  SELECT M.* FROM member M
[Elimination von Duplikaten] SELECT DISTINCT SName FROM Stadt order by area ASC, name DESC
[neue Spalte]              SELECT name, code, capital, area, 'sehr groß' as Bem, code as CD
                           FROM COUNTRY WHERE AREA > 9500000
    
```

Name	Code	Capital	Area	Bem	CD
China	TJ	Beijing	9596960	sehr groß	TJ
Canada	CDN	Ottawa	9976140	sehr groß	CDN
Russia	R	Moscow	17075200	sehr groß	R

```

[Pattern Matching]        SELECT * FROM Land where LName LIKE '%many%'
    
```

```

NAME CODE CAPITAL PROVINCE AREA POPULATION
Germany D Berlin Berlin 356910 83536115
    
```

```

[Pattern Matching]        SELECT name, code from country where name like '__y%'
    
```

```

NAME CODE
Egypt ET
Guyana GUY
Seychel. SY
    
```

[skalare Teilanfr.]

Eine *gekammerte* Teilanfrage wird wie eine *skalare Konstante* verwendet
 Die Teilanfrage darf dann nur genau einen Wert liefert (keine Menge!)
 Welche Länder haben einen geringeren Flächenanteil in Asien als die Türkei?

```

SELECT DISTINCT * FROM county WHERE continent='Asia' AND percentage <
(SELECT percent FROM county WHERE code='TR' AND continent = 'Asia');
    
```

[Verbund, Alias]

```

SELECT DISTINCT city.name AS stadt, country.name AS land FROM city,
country WHERE city.country = country.code order by land
    
```

count.code	STADT	LAND	mit	city.Name	city.country	country.name
	Aachen	Germany	Aachen	GER	Germany	GER
	...	Germany	Budapest	UNG	Hungary	UNG
	Zwickau	Germany	Paris	FR	France	FR
	Athen	Greece	...			

[K.bezeichner optional] SELECT DISTINCT ci.name, co.name FROM city as *ci*, country *co* WHERE ci.country = co.code ORDER BY co.name ASC, ci.name DESC

gleiches Ergebnis wie vorherige Anfrage, nur mit ci als city als verkürzte Schreibweise. Schlüsselwort as ist optional und kann weggelassen werden.

[Korrelat.B zwingend] SELECT DISTINCT L1.LCode AS Land1, L2.LCode AS Land2 FROM **Lage** L1, **Lage** L2 WHERE L1.Kontinent = L2.Kontinent AND L1.LCode < L2.LCode
alle LandPaare (Lage.code × Lage.code) im gleichen Kontinent.
Keine doppelten oder reflexiven Angaben, wie GER:FR & FR:GER oder FR:FR
Korrelationsbezeichner zwingend notwendig, da Kreuzprodukt der Tabelle mit sich selbst erforderlich.

SQL-Ausdruck ... zu $R \cap (S \cup T)$: Select R.x from R, S, T where R.x=S.x or R.x=T.x
liefert nichts wenn T leer ist, da das Kreuzprodukt 'from R,S,T' leer ist.

Algorithmus (nested-loop-Semantik zur Veranschaulichung des SQL-Verhaltens)

```
FOR each Tupel t1 in Relation R1 DO
  FOR each Tupel t2 in Relation R2 DO ...
    FOR each Tupel tm in Relation Rm DO
      IF Die WHERE-Klausel ist erfüllt nach Ersetzen der Attributnamen in F durch
        die entsprechenden Werte der gerade betrachteten Tupel t1, ..., tm
      THEN Bilde ein Antwort-Tupel aus den Werten der in der SELECT-Klausel 1
        angegebenen Attributen A1, ..., An bezgl. der gerade betrachteten Tupel t1, ..., tm
```

SELECT LName, MAX(Einwohner) FROM Land ist syntaktisch nicht zulässig.

Es kann mittels ALL oder ANY getestet werden, wie sich Zeile z zu den Zeilen in T verhält.

Eine Tabelle kann mittels EXISTS getestet werden, ob sie mindestens eine Zeile enthält,
bzw. mittels UNIQUE, ob sie keine Duplikate enthält.

Tests auf Teilmengen, bzw. **Mengengleichheit** können in SQL nicht direkt (per Mengenoperator) ausgedrückt werden. Jedoch gilt: Mengen $A = B$ gdw. $(A \cup B) - (A \cap B) = \emptyset$ oder wenn jedes Element der ersten Menge auch Element der zweiten Menge ist und umgekehrt.

Welche Land-ID's treten in Country oder (und nicht in) der Relation encompasses auf?

(SELECT code FROM Land) EXCEPT (SELECT Land FROM enc);

(SELECT code FROM Land) UNION (SELECT Land FROM enc);

Mittels UNION ALL, INTERSECTION ALL, EXCEPT ALL werden Duplikate der Zeilen berücksichtigt.
Hat der erste Operand n Duplikate eines Tupels und der zweite Operand m, dann hat das Ergebnis n+m, min(n,m), max(n-m,0) Duplikate dieses Tupels.

Bei Verwendung von Korrelationsvariablen (M) von übergeordneten Anfragen wird die Teilanfrage pro Wertekombination der Korrelationsvariablen einmal ausgeführt, d.h.pro Tupel (zeilenweise) aus M (country) wird die darauf folgende Teilanfrage jeweils geprüft und kann auf die Tupelwerte zugreifen.

select lname from land L1 where UNIQUE (select L2.LCode from lage L2 where L1.LCode = L2.LCode)
select lname from land L1 where 1 = (select count(*) from lage L2 where L1.LCode = L2.LCode)

Wie kommt man auf diesen SQL-Ausdruck, wenn man den Hinweis der Division erhält?

- Algebra als Ausgangspunkt mit R(A,B), S(B), T(A) als Schemata:

$$T = R \div S \text{ [}\div \text{ kein SQL-Operator]} = \{x \mid \{x\} \times S \subseteq R\} \text{ [}\subseteq \text{ kein SQL-Operator]} = \{x \mid \{x\} \times S = \emptyset\}$$

Anstatt die Relation S mit der Variablen X durch ein Kreuzprodukt (wie zu realisieren?!) zu erweitern, nur um das gleiche Format wie R(A,B) für die Mengendiff. zu erhalten, kann man R reduzieren.

Zunächst interessieren in R alle Tupel die X enthalten (Selektion) mit anschl. Projektion auf B:

$S(X,B) - R(A,B) = S(B) - R(B)$ womit sich folgender Ausdruck (alle Operatoren in SQL verfügbar) ergibt: $T = R \div S = \{x \mid (S - \pi[B])(\sigma[A=X]R) = \emptyset\}$

```
select distinct A from R as R1 as X where not exists
((select distinct B from S) except (select distinct B from R where R.A = R1 as X.A));
```

- Kalkül als Ausgangspunkt: $T = R \div S = \{X \mid \forall Y (S(Y) \Rightarrow R(X,Y))\} = \{X \mid \neg \exists Y (S(Y) \wedge \neg R(X,Y))\}$
 - 1.) $R(X,Y)$. Ist in R ein Tupel (X,Y) enthalten? `select * from R where R.A = X.A and R.B = Y.B`
 - 2.) Ist es nicht enthalten (Ausdruck true)? `Not exists (select * from R where R.A = X.A and R.B = Y.B)`
 - 3.) $(S(Y) \wedge \neg R(X,Y)) =$ `select * from S as Y where not exists (select * from R ... and R.B = Y.B)`
 - 4.) `select distinct A from R as X where not exists (select * from S as Y where not exists (select * from R where R.A = X.A and R.B = Y.B));` **ODER**
`select distinct A from R as X where not exists (select * from S as Y where X.A NOT IN (select * from R where R.B = Y.B));`

Alternative mit nur einem Kreuzprodukt: `select distinct count(*), B.population from Country A, Country B where (A.population >= B.population) group by B.population having count(*) <= 3;`

Die Alternative gibt das gesamte Ranking aus. Variante 1 braucht für x Länder x KreuzProdukte. Count kann andere Aperationen ersetzen, z.B. Testen ob eine Menge leer ist: `count = 0?`

Orthogonalität: Ein Ausdruck, der eine Menge definiert, sollte überall dort zulässig sein, wo ein Relationsbezeichner stehen darf, z.B. *Berechne die Gesamtzahl aller Städte und Länder.*
`Select count(*) from ((select name from city) UNION (select name from country));`

Ein Ausdruck, der einen skalaren Wert definiert, sollte überall dort zulässig sein, wo ein solcher stehen darf: `Select name, pop, (select avg(pop) from city c1 where c1.pop <= c0.pop) as interesting) from city c0;`
[Tabellen erstellen] `CREATE TABLE City (name varchar2(35), latitude NUMBER);`

[Löschen] `optionale Eigenschaften: not null, unique, primary key(attr1, attr2), default 'Fr'`
`delete city; // löscht alle Tupel aus city`
Löschen aller Städte, deren Einwohnerzahl kleiner dem Durchschnitt sind.
`DELETE City WHERE population < (SELECT AVG(population) FROM City);`

Bemerkung: während des Löschens der einzelnen Tupel ändert sich der Durchschnitt AVG(population)! Deshalb werden vor dem Löschen alle zu löschenden Tupel vom DBMS markiert und danach gelöscht.

[Einfügen] `insert into R(A_1, ..., A_n) values (val_1,...,val_n)`
werden nicht alle Werte angegeben, so werden Null- oder die Defaultwerte verw.

Einfügen einer Menge von Tupeln:

Alle Länder die Mitglied in bestimmten Organisationen sind, die aber noch nicht in Country auftreten, werden in diese Relation übernommen.

```
Insert into country select distinct is_member.country from is_member where not exists
(select code from Country L where L.code = is_member.country);
```

```
INSERT INTO Land select DISTINCT M.LCode from Mitglied M
```

Die Angabe der Spaltennamen kann entfallen, wenn die Werte in der Reihenfolge der Spaltennamen in der CREATE-Anweisung definiert werden.

Beim Einfügen von Zeilen muss die Eindeutigkeit des Primärschlüssels gewährleistet sein.

```
CREATE SEQUENCE LandSEQ AS INTEGER START WITH 1 INCREMENT BY 1
MINVALUE 1 MAXVALUE 100000 NO CYCLE
```

```
insert into Land (LandNr, LName, HStadt, Fläche) values (next value for LandSEQ, 'Bayern', 'Muen', 70)
```

```
create table Land LandNr INTEGER GENERATED ALWAYS AS IDENTITY
(START WITH 1 INCREMENT BY 1 MINVALUE 1 MAXVALUE 100000 NO CYCLE),
```

- [Ändern] update R set K where P;
update Mitarbeiter SET Lohn=Lohn/1,95583; WHERE Waehrung = 'DM';
- [Null] - mit IS NULL() oder IS NOT NULL() kann auf null-werte getestet werden
- 1+NULL oder Spalte_A+NULL ergibt NULL
- Aggregierungsfunktionen außer COUNT ignorieren NULL
avg(popul) würde dann nur durch die Anzahl der Attribute exkl. null dividiert
- Count(SpalteA) zählt auch Einträge mit Wert NULL mit.
- Ausdrücke mit Vergleichsoperatoren der Form A=B, A<>B, A<B haben den Wahrheitswert UNKNOWN, wenn einer der Operatoren Null ist
- dreiwertige Logik
- | | | | | | | | | | |
|---|------------|---|---|---|-----------|---|---|---|------------|
| | <u>AND</u> | t | u | f | <u>Or</u> | t | u | f | <u>NOT</u> |
| t | | t | u | f | | t | t | t | f |
| u | | u | u | f | | t | u | u | u |
| f | | f | f | f | | t | u | f | t |
- [expliziter Verbund] SELECT K.Teilnehmer, L.* FROM Kurs K JOIN Kurs L ON M.Code = L.Code WHERE M.Teilnehmer = 'bernauer'
- [natürliche Verbund] SELECT L.Teilnehmer, M.* FROM Kurs L NATURAL JOIN Teilnehmer M WHERE M.Titel = 'AO Chemie'
- [äußerer Verbund] SELECT L.LCode AS Land, M.Organisation AS Org FROM Land L LEFT OUTER JOIN Mitglied M ON L.LCode = M.LCode WHERE L.Fläche > 50

Tarif

Kunden

Typ	Descr.	Teilnehmer		Nachname	Vorname	Geburtsdatum
One	Einsteiger	Bernauer		Kelz	Andreas	21.07.1965
One+	Standard	Huber		Huber	Karl	16.12.1964
Super	Profi	Kelz		Ernsbach	Elli	29.06.1956

- Schnittmenge:** Liefert die Zeilen, die in beiden angegebenen Tabellen enthalten sind.
SELECT * FROM Mitarbeiter **INTERSECT** SELECT * FROM Kunden
äquiv.: Select * from Mitarbeiter where Nachname IN(select Nachname from Kunden)
- Kart. Produkt** Verbindet jede Zeile der ersten Tabelle mit **jeder** Zeile der zweiten Tabelle
SELECT * FROM Kunden **CROSS JOIN** Tarif ⇔ SELECT * FROM Kunden, Tarif
Gleichnamige Spalten werden durch den Tabellennamen referenziert, z.B. Tab1.Name
- Inner Join** Verbindet Tupel zweier Tabellen, sobald ein gemeinsames Feld dieselben Werte enthält.
SELECT * FROM Kunden **INNER JOIN** Tarif ON Kunden.Nachname=Tarif.Teilnehmer
⇔ SELECT * FROM Kunden, Tarif WHERE Kunden.Nachname=Tarif.Teilnehmer
In der Praxis wird man bei einem Inner Join nicht alle (hier liefern Spalte Teilnehmer und Nachname redundante Informationen), sondern nur ausgewählte Spalten anzeigen lassen.
- Natural Join** Verknüpft die beiden Tabellen über die Gleichheit **aller** gleichlautenden Spalten.
Diese werden im Ergebnis nur einmal angezeigt (im Ggs. zu inner Join).
Haben die Tabellen keine gleichlautenden Spalten, wird der Natural Join zum Cross Join.
Gibt es **nur eine** gleichlautende Spalte, so ist der Natural Join ein Inner Join mit anschließender Projektion, bei der gleichnamige Spalten ausgeblendet werden.
- LeftOuter Join** schließt alle Datensätze aus der ersten (linken) Tabelle ein, auch wenn keine entsprechenden Werte für Datensätze in der zweiten Tabelle existiert.
SELECT * FROM Kunden LEFT JOIN Tarif ON (Kunden.NN = Tarif.TN)

Kunden.NN	Kunden.VN	Geburtsdatum	Typ	Descr	Teilnehmer
Kelz	Andreas	21.07.1965	NULL	NULL	NULL
Huber	Karl	16.12.1964	One+	Standard	Huber
Ernsbach	Elli	29.06.1956	NULL	NULL	NULL

LeftOuter Join SELECT * FROM Kunden RIGHT JOIN Tarif ON (Kunden.NN = Tarif.TN)

Kunden.NN	Kunden.VN	Geburtsdatum	Typ	Descr	Teilnehmer
NULL	NULL	NULL	One	Einsteiger	Bernauer
Huber	Karl	16.12.1964	One+	Standard	Huber
NULL	NULL	NULL	Super	Profi	Kelz

Full Join SELECT * FROM Kunden Full JOIN Tarif ON (Kunden.NN = Tarif.TN)

Kunden.NN	Kunden.VN	Geburtsdatum	Typ	Descr	Teilnehmer
NULL	NULL	NULL	One	Einsteiger	Bernauer
NULL	NULL	NULL	Super	Profi	Kelz
Huber	Karl	16.12.1964	One+	Standard	Huber
Kelz	Andreas	21.07.1965	NULL	NULL	NULL
Ernsbach	Elli	29.06.1956	NULL	NULL	NULL

Union Join SELECT * FROM Kunden UNION JOIN Tarif // Keine Bedingung !

Kunden.NN	Kunden.VN	Geburtsdatum	Typ	Descr	Teilnehmer
Kelz	Andreas	21.07.1965	NULL	NULL	NULL
Huber	Karl	16.12.1964	NULL	NULL	NULL
Ernsbach	Elli	29.06.1956	NULL	NULL	NULL
NULL	NULL	NULL	One	Einsteiger	Bernauer
NULL	NULL	NULL	One+	Standard	Huber
NULL	NULL	NULL	Super	Profi	Kelz