Skript zur Vorlesung

Datenbanken I

(Einführung Datenbanken)

Fachhochschule Stralsund
Fachbereich Elektrotechnik & Informatik

Prof. Dr. Uwe Hartmann

1. Datenbanksysteme

Literaturhinweise

Schwinn
'Relationale Datenbanksysteme', Carl Hanser

Date,C.J.
'An Introduction to Database Systems I' und
'An Introduction to Database Systems II', Addison-Wesley

Vossen
'Datenmodelle, Datenbanksprachen und Datenbank-Management-Systeme', Addison-Wesley

Wedekind
'Datenbanksysteme I' und 'Datenbanksysteme II', BI Wissenschaftsverlag

van der Lans
'Das SQL-Lehrbuch' , Addison-Wesley

Misgeld
'SQL', Hanser

Chen
'Der Entity-Relationship-Ansatz zum log. Entwurf', BI Wissenschaftsverlag

Heuer
'Objektorientierte Datenbanksysteme', Addison-Wesley

Wedekind
'Objektorientierte Schemaentwicklung', BI Wissenschaftsverlag

Vossen
'Das DB2-Handbuch', Addison-Wesley

Chamberlin
'DB2 Universal Database', Addison-Wesley

Hein
'Das ORACLE Handbuch', Addison-Wesley

Rautenstrauch
'Effiziente Systementwicklung mit ORACLE', Addison-Wesley

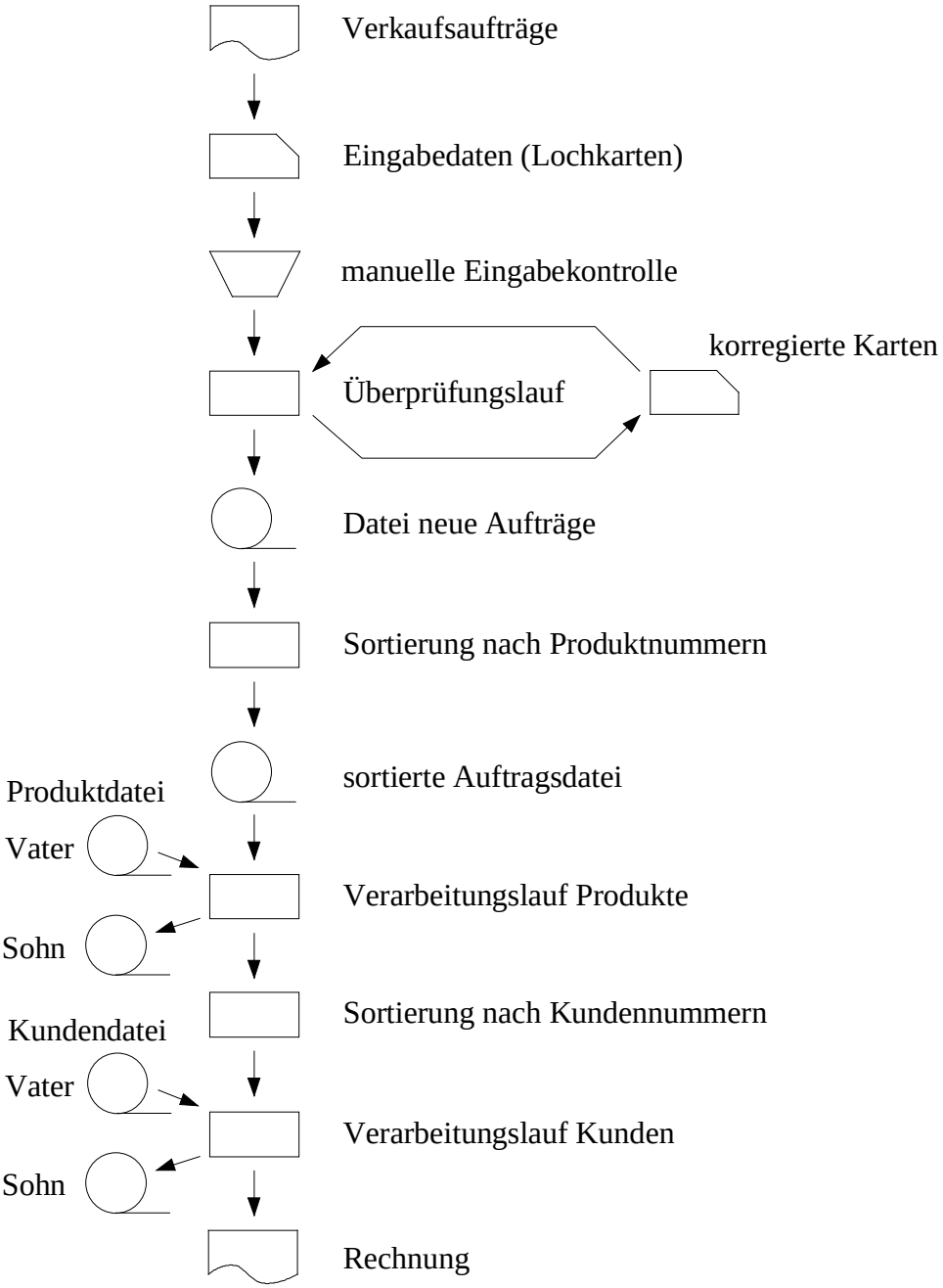
Petkovic
'INGRES-Das relationale Datenbanksystem', Addison-Wesley

Petkovic
'Informix-Das relationale Datenbanksystem', Addison-Wesley

1.1 Generationen der Entwicklung von Datenbanken

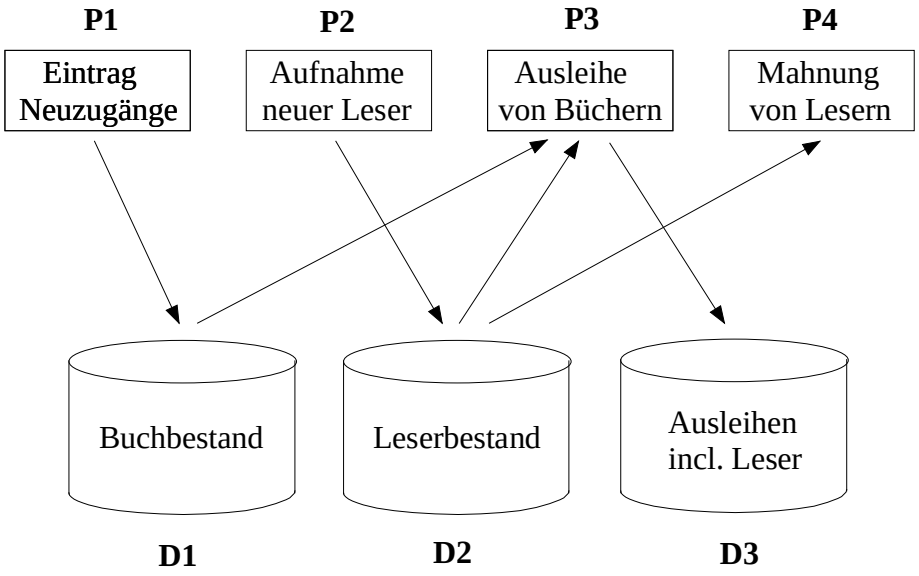
- **1. Generation**
  Zeitraum der fünfziger Jahre
  Filesysteme auf Band
  Stapelverarbeitung

Beispiel



- **2. Generation**
  Zeitraum der sechziger Jahre
  Filesysteme auf Platte
  Direktzugriff

Beispiel
Bibliothekssystem

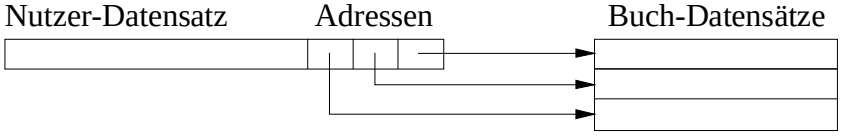


- Aus der mit der Dateiverarbeitung resultierende Probleme:
  1. Hohe Redundanz
  2. Gefahr von Inkonsistenzen
  3. Geringe Produktivität der Programmierer

- **3. Generation**
  Zeitraum der siebziger Jahre
  Einheitliche Strukturierungsprinzipien der Daten: sog. Datenmodelle
  – Hierarchisches Modell
  – Netzwerkmodell

- Objekte und Beziehungen zwischen Objekten eines Anwendungsbereichs werden in Datensätze und physische Links zwischen den Datensätzen in den Rechner (Speicher) abgebildet

Beispiel
ein Nutzer leiht mehrere Bücher aus



- **4. Generation**
  Zeitraum der achtziger Jahre
  Relationales Datenmodell
  strikte Trennung von log. Modell und phys. Speicherung

1.2 Datenbankarchitektur

- Ziel einer DB-Anwendung: Abbildung eines für eine Anwendung relevanten Realitätsausschnittes (**UoD** = Universe of Discourse, Diskursbereich) in einen Rechner und Recherche/Manipulation in diesem Modell

- Datenbanksystem (DBS) = Datenbank (DB) + Datenbankbetriebssystem (DBMS)

- DBMS
  Menge von Programmen (Programmsystem) zur Verwaltung, Strukturierung, Manipulation und zum Auffinden von Daten

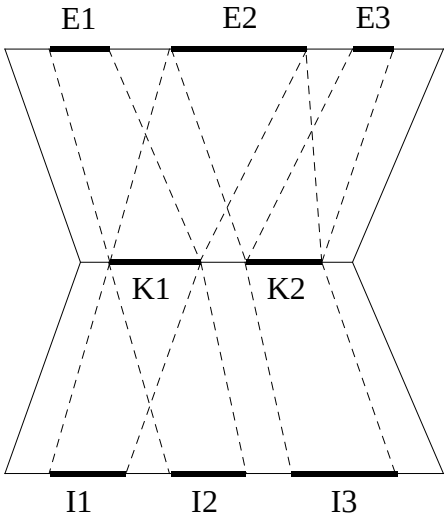
- DB
  Menge von Daten, die Objekte eines UoD abbilden, die logisch untereinander in Beziehungen stehen. Die Daten werden nach einem einheitlichen Verfahren (Datenmodell) strukturiert.

- Anforderungen an ein Datenbanksystem
  - (weitestgehende) Redundanzfreiheit
  - Integritätserhaltung durch das System
  - Vermeidung von Inkonsistenzen
  - Zugriffsschutz
  - anwendungsbezogene Sichen

- Datenunabhängigkeit
  - physische Datenunabhängigkeit
  - logische Datenunabhängigkeit

3-Ebenen-Modell

- Standardisierung von DBMS 1972 durch Standard Planning and Requirements Committee des American National Standard Institute (ANSI/X3/SPARC)



externe Ebene

konzeptuelle Ebene

interne Ebene

- **Konzeptuelle Ebene**: Modelle des gesamten Anwendungsbereichs (UoD)
- **externe Ebene**: Sichten der einzelnen Anwendungen auf die Datenbank
- **interne Ebene**: interne physische Speicherung der Daten

Beispiel

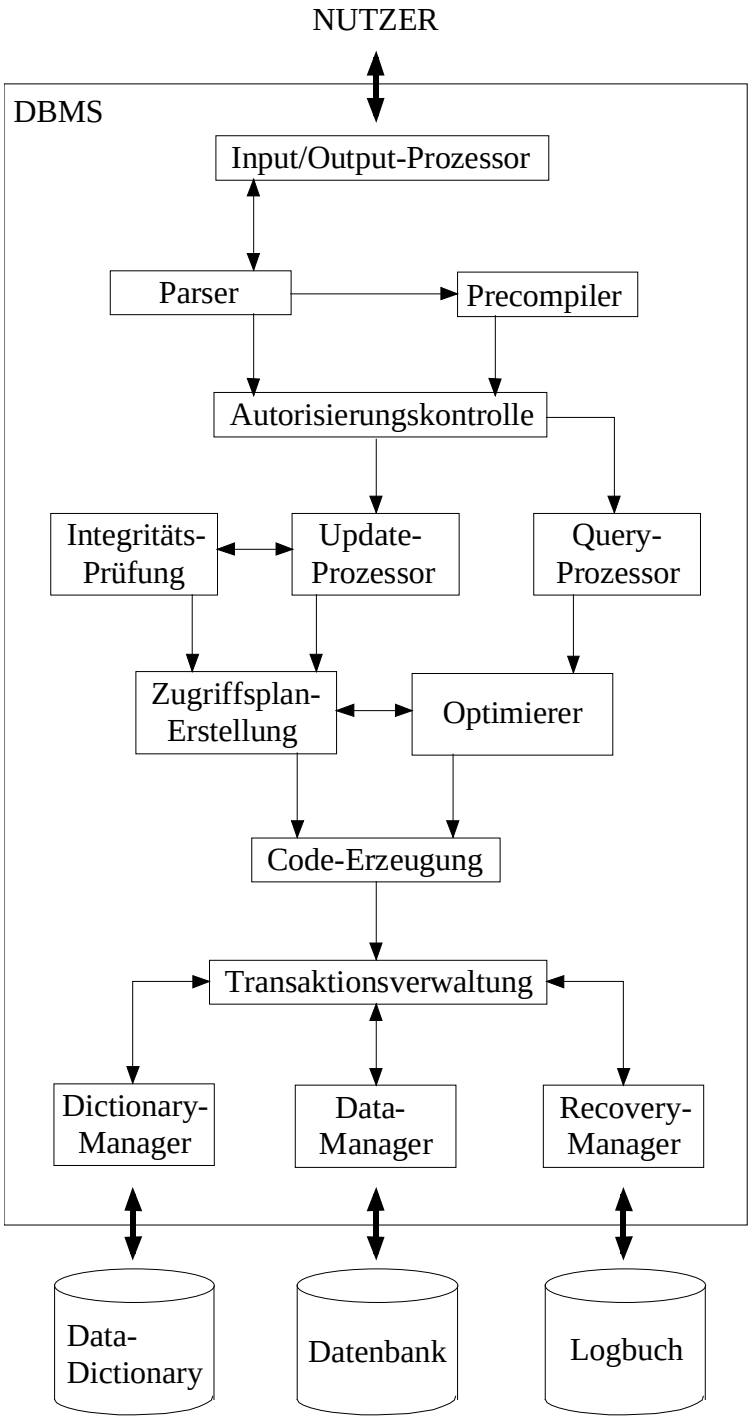
Bibliothek
konzeptuelle Ebene
K1: Benutzerdaten (Name, Anschrift ...)
K2: Bücherdaten (Autor, Titel ...)
externe Ebene (Anwender/Bibliotheksbereiche)
E1: Sicht auf Benutzer
E2: Sicht auf Buch-Ausleihen (Beziehung zwischen Benutzern und Büchern)
E3: Sicht auf Buchbestand
interne Ebene (Speicherstrukturen)
I1: (Name, Straße) der Benutzerdaten als sequentielle Datei
I2: (PLZ, Ort) der Benutzerdaten als Indexsequentielle Datei, mit I1 verkettet
I3: Buchdaten als B*-Baum

- **Datenmodell**
  Menge von log. Konstrukten und Regeln zur Strukturierung von Objekttypen/Beziehungstypen eines UoD auf der konzeptuellen Ebene einer Datenbank

- **Datenbanksprache**
  Menge sprachlicher Konstrukte zur
  a) Strukturierung von Datentypen (DDL: Data Description Language)
  b) Manipulation von Daten (DML: Data Manipulation Language, QL: Query Language)
  c) Definition von
     - Integritätsbedingungen (Trigger)
     - Zugriffsrechten von DB-Nutzern
     - Sichten (Views)
     - internen Speicherstrukturen/-verfahren ...

- Ein **DBMS** ist ein Softwaresystem, das Anweisungen in einer Datenbanksprache in eine nach einem Datenmodell strukturierte DB abbildet

Komponenten eines DBMS



2. Das Relationenmodell

- E. F. Codd:
  "A relational model of data for large shared data banks"
  Comm. ACM, 1970

2.1 Relationen

♦ **Relation (als Wertemenge)**
Eine Relation $\overline{R}$ ist eine Menge

$$\overline{R} \subseteq M_1 \times M_2 \times ... \times M_n = \bigtimes_{i=1}^{n} M_i$$

M_i – Wertemengen

Beispiel
M1 = (X,Y,Z)
M2 = (U,W)
$\overline{R} \subseteq M1 \times M2$
$\subseteq \{(X,U),(X,W),(Y,U),(Y,W),(Z,U),(Z,W)\}$

$\overline{R}1 = \{(X,U),(Y,W),(Z,U)\}$
$\overline{R}2 = \{(X,W),(Y,U)\}$
$\overline{R}2 = \{\}$

♦ **Tupel**
Ein Element r einer Relation $\overline{R}$ heißt Tupel

$r \in \overline{R}$
$r = (m_1, m_2, ..., m_n)$ mit $m_i \in M_i$

♦ **Wertebereich**
Eine Menge M_i ist Wertebereich des Attributs A_i

$M_i = \text{dom}(A_i)$

♦ **Attributkombination**
Eine Attributkombination ist eine geordnete Menge von Attributen

$\overline{A} = (A_1, A_2, ..., A_n)$

♦ **Relationenstruktur**
Eine Relationenstruktur wird dargestellt als

$R(A_1, A_2, ..., A_n)$

mit R - Name der Relation

A_i - Attribut(namen)

♦ **Grad einer Relation**
Die Anzahl der Attribute einer Relation

$R(A_1, ..., A_n)$

wird als Grad (degree, Stelligkeit) der Relation

$deg(R) = n$

bezeichnet. (n-stellige Relation)

♦ **Dimension/Kardinalität**
Die Anzahl der Tupel einer Relation R heißt Dimension oder Kardinalität von R:

dim(R) oder card(R)

♦ **Datenbankrelation**
Eine Datenbankrelation (wir verwenden künftig nur den Begriff Relation) wird dargestellt als

$R(\overline{A}, \overline{R}) = R((A_1, ..., A_n), \{r | r \in \overline{R}\})$

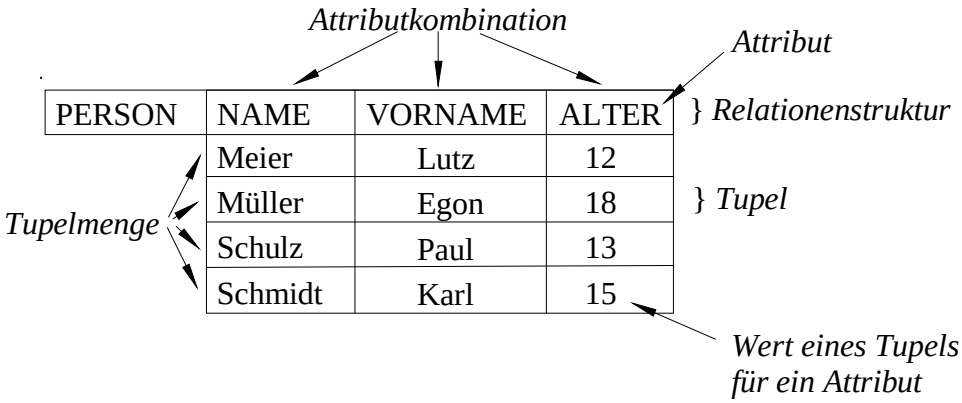
mit R - Relationenname

$\overline{A}$ - Attributkombination

$\overline{R}$ - Tupelmenge

♦ **Wert eines Tupels**
Der Wert eines Tupels r des Attributs A_i wird dargestellt als $r[A_i]$

• **Tabellendarstellung einer Datenbankrelation**



- **Eigenschaften von Relationen**
  - Die Reihenfolge der Tupel einer Relation ist für deren Informationsgehalt ohne Bedeutung
  - Die Reihenfolge der Attribute (Attributkombination) ist für den Informationsgehalt der Relation ohne Bedeutung

♦ **Äquivalente Relationen**
Zwei Relationen

$$R(\overline{A}, \overline{R}), \quad \overline{A} = (A_1, \ldots, A_n) \text{ und}$$
$$S(\overline{B}, \overline{S}), \quad \overline{B} = (B_1, \ldots, B_m)$$

heißen äquivalent strukturiert, wenn gilt

$$\{A_1, \ldots, A_n\} = \{B_1, \ldots, B_m\}$$

R und S heißen äquivalent, wenn sie äquivalent strukturiert sind und $\overline{R} = \overline{S}'$ ist, wobei $\overline{S}'$ aus $\overline{S}$ durch Umstellung der Tupel-Elemente entsprechend der Attributkombination von R gewonnen wird.

♦ **Relationale Datenbank**
Eine relationale Datenbank ist eine Menge von Relationen

$$DB = \{R_1, \ldots, R_k\} \text{ mit } R_i(\overline{A}_i, \overline{R}_i)$$

2.2 Relationenalgebra

- DB-Anweisungen relationaler Systeme basieren auf zwei Typen von Anfragesprachen
  - Relationenalgebra (algebraorientierte Sprachen)
  - Relationenkalkül (kalkülorientierte Sprachen)

2.2.1 Algabraoperationen

- Relationenalgebra: Satz von Algebraoperationen

zwei Typen von **Algebraoperationen**:

1) klassische Mengenoperationen
   - Vereinigung
   - Durchschnitt
   - Differenz
   - (Kreuzprodukt)

2) spezielle relationale Operationen
   - Restriktion (Selektion)
   - Projektion

- Join (Verbund)
- Division

- Die Anwendung einer Algebraoperation auf eine oder zwei Relationen erzeugt im Ergebnis eine Relation

für die folgenden Definitionen der Algebraoperationen seien 2 Relationen gegeben:

$$R((A_1,...,A_n),\overline{R})$$
$$S((B_1,...,B_m),\overline{S})$$

- **Vereinigungsverträglichkeit**
  Zwei Relationen R und S heißen vereinigungsverträglich, wenn sie äquivalent strukturiert sind.

- Es gilt: Die Mengenoperationen Vereinigung, Durchschnitt und Differenz sind nur auf vereinigungsverträgliche Relationen anwendbar

Vereinigung

Die Vereinigung der Relationen R und S ist eine Relation

$$R \cup S = \{ r \mid r \in R \vee r \in S \}$$

Beispiel:

Vereinigung zweier Relationen $T = R \cup S$

R	A1	A2
	a	1
	b	2

S	A1	A2
	a	1
	a	2
	b	4

T	A1	A2
	a	1
	b	2
	a	2
	b	4

Durchschnitt

Der Durchschnitt der Relationen R und S ist eine Relation

$$R \cap S = \{ r \mid r \in R \wedge r \in S \}$$

Beispiel

Durchschnitt zweier Relationen $T = R \cap S$

R	A1	A2
	a	1
	b	2

S	A1	A2
	a	1
	a	2
	b	4

T	A1	A2
	a	1

Differenz

Die Differenz der Relationen R und S ist eine Relation

$$R - S = \{ r \mid r \in R \wedge r \notin S \}$$

Beispiel

Differenz zweier Relationen T1 = R – S bzw. T2 = S – R

R	A1	A2
	a	1
	b	2

S	A1	A2
	a	1
	a	2
	b	4

T1	A1	A2
	b	2

T2	A1	A2
	a	2
	b	4

Projektion

Die Projektion einer Relation R auf eine Attributkombination $C_1,\ldots,C_k$ von R ist eine Relation

$$\prod_{C_1,\ldots,C_k}(R) = \{ (m_1,\ldots,m_k) \mid r \in R \wedge m_1 = r[C_1] \wedge \ldots \wedge m_k = r[C_k] \wedge \{ C_1,\ldots,C_k \} \subset \{ A_1,\ldots,A_n \} \}$$

Beispiel

Projektion einer Relation R auf die Attribute A2 und A3: $T = \prod_{A2,A3}(R)$

R	A1	A2	A3
	a	1	x
	a	1	y
	b	1	x
	c	2	z

T	A2	A3

1	x
1	y
2	z

Restriktion

Die Restriktion (Selektion) einer Relation R mit einer Bedingung P ist eine Relation

$$\sigma_{(P)}(R) = \{\, r \mid r \in R \wedge P(r) = \text{true} \,\}$$

mit P(r) ist:

a) ein Vergleich eines Attributs mit einer Konstanten
$A_i \; \Theta \; m$: $P(r) = \text{true}$ g.d.w. $r[A_i] \; \Theta \; m$ wahr ist

b) ein Vergleich zweier Attribute
$A_i \; \Theta \; A_j$: $P(r) = \text{true}$ g.d.w. $r[A_i] \; \Theta \; r[A_j]$ wahr ist
wobei Θ : $=, \neq, <, >, \leq, \geq$

c) logische Verknüpfung aus Prädikaten
$P(r) = P_1(r) \Delta P_2(r) \Delta P_3(r) \ldots$ mit Δ ist $\wedge, \vee$

Beispiel

Restriktion einer Relation R mit dem Prädikat A1='a': $T = \sigma_{A1='a'}(R)$

R	**A1**	**A2**	**A3**
	a	1	x
	a	1	y
	b	1	x
	c	2	z

T	**A1**	**A2**	**A3**
	a	1	x
	a	1	y

Produkt

Das Produkt (Kreuzprodukt) zweier Relationen R und S ist eine Relation

$$R \times S = \{\, r \mid r \in T((\overline{A}, \overline{B}), \overline{T}) \;\wedge\; \prod_{\overline{A}}(T) = R \;\wedge\; \prod_{\overline{B}}(T) = S \;\wedge\; \dim(T) = \dim(R) * \dim(S) \,\}$$

Beispiel

Produkt einer Relation R mit einer Realtion S: $T = R \times S$

R	A1	A2
	a	1
	b	2

S	B1
	x
	y
	z

T	A1	A2	B1
	a	1	x
	a	1	y
	a	1	z
	b	2	x
	b	2	y
	b	2	z

Join

Der Join (Verbund) zweier Relationen R und S ist eine Relation

$$R \underset{A_i \Theta B_j}{\otimes} S = \left\{ r \mid r \in R \times S \ \wedge \ r[A_i] \, \Theta \, r[B_j] \right\}$$

Beispiel:

Join zweier Relationen R und S mit der Bedingung A2>B2: $T = R \underset{A2>B2}{\otimes} S$

R	A1	A2	A3
	a	1	x
	b	2	y
	b	4	x

S	B1	B2
	x	1
	x	4
	y	3
	z	8

T	A1	A2	A3	B1	B2
	b	2	y	x	1
	b	4	x	x	1
	b	4	x	y	3

- **Equi-Join**
  Ein Equi-Join ist ein (Theta-)Join, bei dem Θ das '=' ist.

Beispiel:

Equi-Join zweier Relationen R und S mit der Bedingung A3=B1: $T = R \underset{A3=B1}{\otimes} S$

R	A1	A2	A3
	a	1	x
	b	2	y
	b	4	x

S	B1	B2
	x	1
	x	4
	y	3
	z	8

T	A1	A2	A3	B1	B2
	a	1	x	x	1
	a	1	x	x	4
	b	2	y	y	3
	b	4	x	x	1
	b	4	x	x	4

- **Natural-Join**

  Seien R und S zwei Relationen mit genau zwei gleichen Attributen $A_i = B_j$, dann ist der Natural-Join (natürlicher Verbund) von R und S eine Relation

$$R \otimes S = \left\{ r \mid r \in \prod_{\overline{A},\overline{B}'} (R \underset{A_i=B_j}{\otimes} S) \wedge \overline{B}' = \overline{B} - B_j \right\}$$

Beispiel:

Natural-Join zweier Relationen R und S: $T = R \otimes S$

R	A1	A2	A3
	a	1	x
	b	2	y
	b	4	x

S	B1	A2
	x	1
	x	4
	y	3
	z	4

T	A1	A2	A3	B1
	a	1	x	x
	b	4	x	x
	b	4	x	z

- **Sonderformen des Join**

  Semi-Join

$$R \overset{Semi}{\otimes} S = \prod_{\overline{A}} (R \otimes S)$$

Auto-Join

$$R \underset{A_i \Theta A_j}{\otimes} R$$

- **Outer-Join**
  Der Left-/Right-/Full-Outer-Join ist ein Join, bei dem alle Tupel der linken/rechten/beider Relationen in der Ergebnisrelation enthalten sind. Soweit die Tupel der beteiligten Relationen die Join-Bedingung nicht erfüllen, werden ihre Werte auf NULL gesetzt.

Beispiel:

Left-Outer-Join zweier Relationen R und S: $T = R \overset{Left-Outer}{\otimes} S$

R	A1	A2	A3
	a	1	x
	b	2	y
	b	4	x

S	B1	A2
	x	1
	x	4
	y	3
	z	4

T	A1	A2	A3	B1
	a	1	x	x
	b	2	y	?
	b	4	x	x
	b	4	x	z

- Grad und der Dimension der Ergebnisrelation T bei Anwendung relationaler Operationen auf eine Relation R bzw. auf R und S:

Operation	deg(T)	dim(T)
Restriktion	=deg(R)	<=dim(R)
Projektion	<=deg(R)	<=dim(R)[1]
Produkt	=deg(R)+deg(S)	=dim(R)*dim(S)
Vereinigung	=deg(R)=deg(S)	<=dim(R)+dim(S)[2]
Durchschitt	=deg(R)=deg(S)	<=min(dim(R),dim(S))[3]
Differenz	=deg(R)=deg(S)	>=max(dim(R)-dim(S),0)[4]
Theta-Join	=deg(R)+deg(S)	<=dim(R)*dim(S)[5]

2.2.2 Rechenregeln

Seien $R(A_1,...,A_n)$ und $S(B_1,...,B_m)$ zwei Relationen mit den Attributkombinationen (Attributmengen) $\overline{A} = (A_1,...,A_n)$ und $\overline{B} = (B_1,...,B_m)$

Vertauschbarkeit von Restriktionen

$$\begin{aligned}\sigma_{P_1}(\sigma_{P_2}(R)) &= \sigma_{P_2}(\sigma_{P_1}(R)) \\ &= \sigma_{P_1}(R) \cap \sigma_{P_2}(R) \\ &= \sigma_{P_1 \wedge P_2}(R)\end{aligned}$$

mit P_1 und P_2 Prädikate

Vertauschbarkeit von Projektionen

Seien $L_1, L_2 \subseteq \overline{A}$, dann gilt

$$\begin{aligned}\prod_{L_1}(\prod_{L_2}(R)) &= \prod_{L_2}(\prod_{L_1}(R)) \\ &= \prod_{L_1 \cap L_2}(R)\end{aligned}$$

mit $L_1 \cap L_2$: Durchschnitt der Attributmengen

Sonderfall:

Seien $L_1 \subseteq L_2 \subseteq \overline{A}$, dann gilt

$$\prod_{L_2}(\prod_{L_1}(R)) = \prod_{L_1}(R)$$

Vertauschbarkeit von Projektion und Restriktion

Sei $L \subseteq \overline{A}$, dann gilt

$$\prod_{L}(\sigma_{P}(R)) = \sigma_{P}(\prod_{L}(R))$$

Kommutativität des Join

$$R \otimes S = S \otimes R$$

Assoziativität des Join

$$R_1 \otimes (R_2 \otimes R_3) = (R_1 \otimes R_2) \otimes R_3$$

Joinverfälschung (eine Relation)

Sei $L_1 \cup L_2 = \overline{A}$ und $L_1 \cap L_2 = A_j$ (A_j ist das Join - Attribut), dann gilt

a) $R \subseteq \prod_{L_1}(R) \otimes \prod_{L_2}(R)$

b) $\prod_{L_1}(R) = \prod_{L_1}(\prod_{L_1}(R) \otimes \prod_{L_2}(R))$

Joinverlust

$$\prod_{\overline{A}}(R \otimes S) \subseteq R$$

2.3 Schlüssel

Schlüssel

Eine Attributmenge K einer Relation $R(A_1,...,A_n)$ mit $K \subseteq (A_1,...,A_n)$ heißt Schlüssel von R, wenn gilt:

a) für beliebige Tupel $r,s \in R$ gilt,
wenn $\prod_K(r) = \prod_K(s)$ dann r = s und

b) es existiert kein $K' \subset K$ mit der Eigenschaft a)

Primärschlüssel

Seine K und L, $K \neq L$, Schlüssel von R, so heißt derjenige Schlüssel, der in einer DB-Anwendung zur Identifikation der Tupel ausgewählt wird, Primärschlüssel.

Beispiel:
STUDENT(Matr-Nr, Name, Vorname, Wohnsitz, Geburt, Geschlecht, Beruf)
zwei mögliche Schlüssel:
STUDENT(Matr-Nr, Name, Vorname, Geburt, Wohnsitz, Geschlecht, Beruf)
oder
STUDENT(Name, Vorname, Geburt, Matr-Nr, Wohnsitz, Geschlecht, Beruf)

2.4 Semantik relationaler Operationen: Datenbankanfragen

- Gegeben ist eine Kunst-Datenbank mit den Relationen

  Maler: K(KNR, Name, Geburt, Geb_Ort, Tod)
  Museum: M(ENR, Bezeichnung, Sitz)
  Galerie: G(ENR, Bezeichnung, Sitz)
  Bild: B(BNR, Titel, Wert, KNR, ENR)

  - Die Relation Maler enthält den Namen sowie Geburts- und Sterbejahren und den Geburtsort der Künstler. Ein NULL-Wert (-) im Sterbejahr besagt, daß der Maler noch lebt. Eine Schlüsselnummer (KNR) identifiziert einen Künstler eindeutig.
  - Museen und Galerien sind Einrichtungen, die Bilder besitzen. Sie werden durch einen Namen (Bezeichnung) und durch ihren Sitz beschrieben. Jede Einrichtung besitzt eine identifizierende Nummer (ENR).
  - Die Relation Bild enthält Angaben zu Titel, dem Entstehungsjahr und zum Wert (in TDM) von Bildern. Jedes Bild wird durch eine Bildnummer (BNR) identifiziert. KNR ist die Nummer des Malers eines Bildes. Ein NULL-Wert bedeutet, daß der Maler unbekannt ist. ENR ist die Nummer der Einrichtung (Galerie oder Museum), in deren Besitz sich das Bild befindet.

- Maler (K)

KNR	Name	Geburt	Geb_Ort	Tod
K1	Lorrain	1734	Paris	1812
K2	Endler	1931	München	-
K3	Veccio	1480	Paris	1528
K4	Runge	1777	Berlin	1810
K5	van Goyen	1922	Delft	-
K6	Davis	1940	London	-
K7	Huysman	1822	Delft	1925
K8	Tissot	1722	Paris	1785

- Museum (M)

ENR	Bezeichnung	Sitz
E1	Louvre	Paris
E3	Nationalmuseum	Kopenhagen
E4	Deutsches Museum	Berlin

- Galerie (G)

ENR	Bezeichnung	Sitz
E2	National Gallery	London
E5	Le Corbet	Paris
E6	Galerie Villon	Paris
E7	Nationalgalerie	Berlin

- Bild (B)

BNR	Titel	Jahr	Wert	KNR	ENR
B1	Landschaft mit Pfeilen	1965	150	K2	E2
B2	Wasserlandschaft	1946	200	K5	E6
B3	Maria am Berge	1803	550	K4	E1
B4	Dame mit Hündchen	1784	1200	K1	E5
B5	Bildnis Dirk van Bongen	1877	800	K7	E1
B6	Verkündigung der Maria	1788	1400	K1	E1
B7	Peng!	1996	120	K2	E6
B8	Maria Magdalena	1422	1000	-	E5
B9	Landschaft ohne Pfeile	1965	100	K2	E2
B10	Bildnis des Dr. Franke	1977	250	K5	E6
B11	Hinter den Bergen	1792	750	K4	E1
B12	Auferstehung	1572	800	-	E4
B13	Dorfszene bei Gewitter	1888	450	K7	E1
B14	Das Frühstück	1756	800	K1	E5
B15	Ansicht vom Gendarmenmarkt	1805	450	K4	E7
B16	Portrait Marte F.	1910	600	K7	E7

Anfragen an eine Relation

Anfrage 1

"Gesucht sind Titel und Wert aller Bilder."

$$R = \prod_{Titel,Wert}(B)$$

Lösung R

Titel	Wert
Landschaft mit Pfeilen	150
Wasserlandschaft	200
. . .	. . .
Portrait Marte F.	600

Anfrage 2

"Gesucht sind die Namen aller Pariser Galerien."

$$R = \prod_{Bezeichnung}(\sigma_{Sitz='Paris'}(G))$$

Lösung R

Bezeichnung
Le Corbet
Galerie Villon

Anfragen über mehrere Relationen

Anfrage 3

"Gesucht sind die Orte, an denen sich Museen oder Galerien befinden."

$$R = \prod_{Sitz}(M \cup G)$$

Lösung R

Sitz
Paris
Kopenhagen
Berlin
London

Anfrage 4

"Gesucht sind die Namen von Museen, die Bilder besitzen, sowie deren Titel und Wert."

$$R = \prod_{Bezeichnung,Titel,Wert} (M \otimes B)$$

Lösung R

Bezeichnung	Titel	Wert
Louvre	Maria am Berge	550
Louvre	Dorfszene bei Gewitter	450
Louvre	Hinter den Bergen	750
Louvre	Verkündigung der Maria	1400
Louvre	Bildnis Dirk van Bongen	800
Deutsches Museum	Auferstehung	800

Anfrage 5

"Gesucht sind alle Angaben zu Malern, deren Bilder sich in Museumsbesitz befinden."

$$R = \prod_{KNR,Name,Geburt,Geb\_Ort,Tod} (K \otimes B \otimes M)$$

Lösung R

KNR	Name	Geburt	Geb_Ort	Tod
K1	Lorrain	1734	Paris	1812
K4	Runge	1777	Berlin	1810
K7	Huysman	1822	Delft	1925

Anfrage 6

"Gesucht sind die Namen der Museen, an deren Sitz sich auch Galerien befinden sowie die Namen dieser Galerien"

$$R = \prod_{M.Bezeichnung,G.Bezeichnung} (M \underset{M.Sitz=G.Sitz}{\otimes} G)$$

Anfrage 7

"Gesucht sind die BNR und Titel der Bilder, deren Wert höher ist als der Wert des Bildes 'Das Frühstück'."

$$R = \prod_{B_1.Titel} (B_1 \underset{B_1.Wert>B_2.Wert}{\otimes} \sigma_{B_2.Titel='Das\ Frühstück'}(B_2))$$

Anfrage 8

"Gesucht sind ENR und Bezeichnung <u>aller</u> Museen und die Titel der Bilder in ihrem Besitz."

$$R = \prod_{ENR,Bezeichnung,Titel} (M \overset{Left-Outer}{\otimes} B)$$

Lösung R

ENR	Bezeichnung	Titel
E1	Louvre	Maria am Berge
E1	Louvre	Dorfszene bei Gewitter
E1	Louvre	Hinter den Bergen
E1	Louvre	Verkündigung der Maria
E1	Louvre	Bildnis Dirk van Bongen
E3	Nationalmuseum	-
E4	Deutsches Museum	Auferstehung

Anfrage 9

"Gesucht sind die KNR der Maler, von denen sich kein Bild im Besitz einer Berliner Einrichtung befindet."

$$R = \prod_{KNR}(K) - \prod_{KNR}(B \otimes \sigma_{Sitz='Berlin'}(M \cup G))$$

Anfrage 10

"Gesucht sind die ENR der Einrichtungen, in deren Besitz sich nur Bilder lebender Maler befinden."

$$R = \prod_{ENR}(B) - \prod_{ENR}(B \otimes (\sigma_{Tod \neq NULL}(K)))$$

3. SQL: Die Anfragesprache

- IBM: System/R-Projekt mit der Sprache SEQUEL (*S*tructured *E*nglish *Q*uery *L*anguage)
- 1981 gelangte die Sprache mit dem SQL/Data System von IBM zum ersten mal auf den Markt.
- 1986 erster ISO-Standard der Sprache SQL (SQL1 oder SQL86)
- 1992 aktueller ISO-Standard von SQL (SQL2 oder SQL92): Norm ISO/IEC 9075:1992

1993 Deutsches Institut für Normung: DIN 66 315.

- **Sprachanteile von SQL**
  - der Anfragesprache (Query Language, QL): Formulierung von Informationsanforderungen
  - der Manipulationssprache (Data Manipulation Language, DML): Speicherung und Veränderung von Informationen
  - der Beschreibungssprache (Data Description Language, DDL): Definition von Informationstrukturen

- **Syntaxnotation für die Definition von SQL-Statements**

Symbol	Bedeutung
GROßBUCHSTABEN	Schlüsselworde der Sprache, Terminale, z.B. SELECT
kleinbuchstaben	Syntaktische Elemente, Nicht-Terminale, z.B query-expression
::=	Definitionsoperator, z.B. bei *x* ::= *y* wird *x* durch *y* definiert
|	Alternative, z.B. bei *x* | *y* | *z* muß *x* oder *y* oder *z* angegeben werden.
[ ]	Option, z.B. bei [*x*] muß *x* nicht angegeben werden
{ }	Gruppierung, z.B. bei { *x* | *y* } muß *x* oder *y* angegeben werden
{ }...	Wiederholung, z.B. bei { *x* }... wird *x* ein oder mehrmals wiederholt

3.1 Das SELECT-Grundformat

♦ **Grundstruktur der SELECT-Anweisung**

SELECT *select-list*
FROM *table-reference*
WHERE *search-condition*

- Ergebnis einer SELECT-Anweisung (Anfrage) ist eine Relation:

select-list:	Attibutliste der Ergebnisrelation
table-reference:	Liste von Relationen, die zur Erzeugung der Ergebnisrelation verwendet werden
search-condition:	Bedingung (Eigenschaften), die alle Tupel der Ergebnisrelation erfüllen müssen

Beispiel

Sei R eine Relation mit den Attributen A, B und C: R(A, B, C)

„Gesucht sind die A- und B-Werte der Tupel der Relation R, bei denen die C-Werte größer als ihre zugehörigen B-Werte sind.“

Algebraausdruck:

$$\prod_{A,\,B}(\sigma_{C>B}(R))$$

SELECT-Anweisung:

```
SELECT  A, B
FROM  R
WHERE  C > B
```

Syntax

query-specification ::=

SELECT [ALL | DISTINCT] select-list
FROM table-reference [{ , table-reference }...]
[WHERE search-condition]
[group-by-clause]
[having-clause]
[order-by-clause]

select-list ::=

*
| select-sublist [{ , select-sublist }...]

select-sublist ::=

value-expression [[AS] column-name]
| table-name.*
| range-variable.*

- DISTINCT: unterdrückt die mehrfache Ausgabe gleicher Tupel
- * in der select-list: alle Attribute aller Ausgangsrelationen in der Ergebnisrelation
- AS column-name: selbstdefinierter Name eines Attributs der Ergebnisrelation
- * nach Namen: aller Attribute einer Ausgangsrelation in die Ergebnisrelation

Syntax

table-reference ::=

table-name [range-variable]
| (query-expression) range-variable
| joined-table

- range-variable: Tupelvariable als Aliasname zur Bezeichnung einer Ausgangsrelation zur Gewährleistung der Eindeutigkeit von Namen

Beispiel: (obere Anfrage)

```
SELECT  x.A, x.B
FROM  R x
WHERE  x.C > x.B
```

- query-expression: SELECT-Anweisung, die eine temporäre Relation erzeugt. Wird über eine Tupelvariable referenziert

- joined-table: Spezialform einer Join-Anfrage (siehe unten)

Beispiel 3.1a

„Gesucht sind alle Angaben über die Maler“

```
SELECT *
FROM  Maler
```

Ergebnis

KNR	Name	Geburt	Geb_Ort	Tod
K1	Lorrain	1734	Paris	1812
K2	Endler	1931	München	-
K3	Veccio	1480	Paris	1528
K4	Runge	1777	Berlin	1810
K5	van Goyen	1922	Delft	-
K6	Davis	1940	London	-
K7	Huysman	1822	Delft	1925
K8	Tissot	1722	Paris	1785

oder

```
SELECT  x.*
FROM  Maler  x
```

oder

```
SELECT  Maler.*
FROM  Maler
```

Beispiel 3.1b

„Gesucht sind Bezeichnung und Ort der Museen“

```
SELECT Bezeichnung, Sitz
FROM  Museum
```

Ergebnis

Bezeichnung	Sitz
Louvre	Paris
Nationalmuseum	Kopenhagen
Deutsches Museum	Berlin

Beispiel 3.1c

„Gesucht sind die Sitze (umbenannt zu Orte), an denen sich Galerien befinden“

SELECT DISTINCT Sitz AS Orte
FROM Galerie

Ergebnis

Orte
London
Paris
Berlin

3.2 Werte-Ausdrücke

Syntax

value-expression ::=

numeric-value-expression

| string-value-expression

| datetime-value-expression

| interval-value-expression

- numeric-value-expression: berechnet einen numerischen Wert.
- string-value-expression: erzeugt einen Character-String oder einen Bit-String durch Verkettungsoperationen.
- datetime-value-expression: erzeugt einen Datum-/Zeit-Wert.
- interval-value-expression: erzeugt ein Zeitintervall-Wert.

Syntax

numeric-value-expression ::=

term

| numeric-value-expression { + | - } term

term ::=

factor

| term { * | / } factor

factor ::=

[+ | -] numeric-primary

numeric-primary ::=

unsigned-numeric-value

| column-name

| set-function-specification

| numeric-value-function

| (numeric-value-expression)

- Numerische Ausdrücke sind einfache algebraische Ausdrücke, die den bekannten Rechenregeln genügen.
- Set-Funktionen dienen der Aggregierung von Attributwerten und werden in Datenbanksystemen deshalb auch als Aggregatfunktionen bezeichnet.

Syntax

set-function-specification ::=

COUNT (*)

| set-function-type ([ALL | DISTINCT] numeric-value-expression)

set-function-type ::=

AVG | MAX | MIN | SUM | COUNT

- COUNT(*): Anzahl der signifikanten Tupel einer Relation.
- AVG: Durchschnitt der Werte eines Attributs der signifikanten Tupel einer Relation.

- MAX und MIN: Maximum- bzw. Minimum der Werte eines Attributs der signifikanten Tupel einer Relation.
- SUM: Summe der Werte eines Attributs der signifikanten Tupel einer Relation.
- COUNT: Anzahl der Werte eines Attributs der signifikanten Tupel einer Relation.
- NULL-Werte bleiben bei der Aggregierung unberücksichtigt.

Beispiel 3.2a

„Gesucht ist der Durchschnittswert aller Bilder“

SELECT AVG(Wert)
FROM Bild

Ergebnis

1
601

Beispiel 3.2b

„Gesucht ist die Anzahl der (unterschiedlichen) Orte, an denen sich Galerien befinden“

SELECT COUNT(DISTINCT Sitz) AS Anzahl
FROM Galerie

Ergebnis

Anzahl
3

3.3 WHERE-Bedingungen

- Ein Tupel ist dann und nur dann Bestandteil einer Ergebnisrelation als Antwort auf eine SELECT-Anfrage mit WHERE-Bedingung, wenn diese Bedingung für dieses Tupel wahr ist.

Syntax

search-condition ::=

boolean-term

| search-condition OR boolean-term

boolean-term ::=

[NOT] boolean-primary

| boolean-term AND [NOT] boolean-primary

boolean-primary ::=

predicate

| (search-condition)

- WHERE-Bedingung: Folge logisch durch *OR* und *AND* verknüpfter Einzelbedingungen (predicate)

Syntax

predicate ::=

comparsion-predicate

| between-predicate

| in-predicate

| like-predicate

| null-predicate

| quantified-comparsion-predicate

| exists-predicate

- Die in der Syntax dargestellten Prädikate lassen zum Teil die Bildung von Unteranfragen (*subqueries*) zu.

3.4 Anfragen an eine Relation

3.4.1 Vergleichsoperatoren

Syntax

comparsion-predicate ::=

row-value-constructor comp-op row-value-constructor

comp-op ::=

= | <> | < | > | <= | >=

row-value-constructor ::=

value-expression

| row-subquery

- im einfachsten Fall: Vergleiche zwischen (Ausdrücken aus):
  1. Konstanten und Konstanten
  2. Variablen und Konstanten
  3. Variablen und Variablen

Beipiel 3.4a

„Gesucht sind die BNR und Titel der Bilder, deren Wert >1000 TDM ist."

SELECT BNR, Titel
FROM Bild
WHERE Wert>1000

Ergebnis

BNR	Titel
B4	Dame mit Hündchen
B6	Verkündigung der Maria

Beispiel 3.4b

„Gesucht sind die Namen der Maler, die über hundert Jahre alt geworden sind."

SELECT Name
FROM Maler
WHERE Tod>(Geburt+100)

Ergebnis

Name
Huysman

Beispiel 3.4c

„Gesucht ist die Anzahl der Bilder, die einen Wert von 700 bis 1000 € haben“

SELECT COUNT(*) AS Anzahl
FROM Bild
WHERE Wert>=700
AND Wert<=1000

Ergebnis

Anzahl
5

3.4.2 Das NULL-Prädikat

Syntax

null-predicate ::=

row-value-constructor IS [NOT] NULL

- Der IS-NULL-Ausdruck ist immer dann wahr, wenn *row-value-constructor* den Wert NULL (nicht vorhanden) besitzt.

Beispiel 3.4d

„Gesucht sind Name und Geburtsjahr der derzeit lebenden Maler“

SELECT Name, Geburt
FROM Maler
WHERE Tod IS NULL

Ergebnis

Name	Geburt
Endler	1931
van Goyen	1922
Davis	1940

3.4.3 Der BETWEEN-Operator

Syntax

between-predicate ::=

row-value-constructor [NOT] BETWEEN

row-value-constructor AND row-value-constructor

- Ein *between-predicate* ist wahr, wenn der Wert des ersten *row-value-constructor*s im Bereich zwischen den Werten der beiden anderen liegt.

Beispiel 3.4e

„Gesucht sind die Namen, Geburts- und Todesjahre der Maler, die im 18. Jh. in Paris geboren wurden“

SELECT Name, Geburt, Tod
FROM Maler
WHERE Geb_Ort='Paris'
AND Geburt BETWEEN 1700 AND 1799

Ergebnis

Name	Geburt	Tod
Lorrain	1734	1812
Tissot	1722	1785

3.4.4 Der LIKE-Operator

Syntax

like-predicate ::=

character-value-expression [NOT] LIKE character-value-expression

- In der Regel ist der linke *character-string-expression* ein Attribut (*column-name*), der rechte ein Muster.

- Globalsysmbole in Zeichenketten:
  - % (Prozent), steht für eine beliebige Anzahl beliebiger Zeichen im String
  - _ (Unterstrich), steht für ein einzelnes beliebiges Zeichen im String

Beispiel 3.4f

„Gesucht sind alle Angaben über Bilder, deren Titel etwas über 'Landschaft' aussagt.“

```
SELECT *
FROM Bild
WHERE Titel LIKE '%landschaft%'
OR Titel LIKE '%Landschft%'
```

Ergebnis

BNR	Titel	Jahr	Wert	KNR	ENR
B1	Landschaft mit Pfeilen	1965	150	K2	E2
B2	Wasserlandschaft	1946	200	K5	E6
B9	Landschaft ohne Pfeile	1965	100	K2	E2

3.4.5 Der IN-Operator

Syntax

in-predicate ::=

row-value-constructor [NOT] IN in-predicate-value

in-predicate-value ::=

(value-expression [{ , value-expression }...])

| table-subquery

- Ein in-predicate-Ausdruck ist wahr, wenn der aktulle Wert des row-value-constructor in der Menge der in-predicate-values enthalten ist.

Beispiel 3.4g

„Gesucht sind die Bezeichnungen der Pariser und Berliner Galerien“

SELECT Bezeichnung
FROM Galerie
WHERE Sitz IN (‘Paris’, ‘Berlin’)

Ergebnis

Bezeichnung
Le Corbet
Galerie Villon
Nationalgalerie

3.5 Ordnen von Daten

3.5.1 Die ORDER-BY-Klausel

Syntax

order-by-clause ::=

ORDER BY column-name [ASC | DSC] [{ , column-name [ASC | DSC]}...]

- *ORDER BY* ordnet die Tupel der Ergebnisrelation der SELECT-Anweisung aufsteigend (ASC) oder absteigend (DSC) nach den Werten des ORDER-BY-Attributs (*column-name*). Enthält die order-*by-clause* eine Liste von Attributnamen so wird zunächst nach gleichen Werten des ersten Attributs geordnet, innerhalb einer solchen Gruppe dann nach den Werten des zweiten Attributs u.s.w.

Beispiel 3.5a

„Gesucht sind die Namen und Geburtsjahre aller Maler in alphabetischer Reihenfolge“

SELECT Name, Geburt
FROM Maler
ORDER BY Name

Ergebnis

Name	Geburt
Davis	1940
Endler	1931

Huysman	1822
Lorrain	1734
Runge	1777
Tissot	1722
van Goyen	1922
Veccio	1480

3.5.2 Die GROUP-BY-Klausel

Syntax

group-by-clause ::=

GROUP BY column-name [{ , column-name }...]

- *GROUP BY* gruppiert (ordnet) die Tupel der Ergebnisrelation der SELECT-Anweisung aufsteigend nach den Werten des GROUP-BY-Attributs (*column-name*). Enthält die *group-by-clause* eine Liste von Attributnamen so wird zunächst nach den Werten des ersten Attributs gruppiert, innerhalb einer solchen Gruppe dann nach den Werten des zweiten Attributs u.s.w.

- Die *select-list* der SELECT-Anweisung darf keinen anderen *column-name* enthalten als die zugehörige GROUP-BY-Klausel.

Beispiel 3.5b

„Gesucht sind die Namen aller Galerien nach ihren Sitzen gruppiert“

SELECT Sitz, Bezeichnung
FROM Galerie
GROUP BY Sitz, Bezeichnung

Ergebnis

Sitz	Bezeichnung
Berlin	Nationalgalerie
London	National Gallery
Paris	Galerie Villon
Paris	Le Corbet

- Enthält die *select-list* der SELECT-Anweisung Aggregatfunktionen, dann erfolgt die Aggregierung innerhalb einer jeden einzelnen Gruppe.

Beispiel 3.5c

„Gesucht ist die Anzahl der Bilder, die sich im Besitz einer jeden Einrichtung befinden sowie deren Gesamtwert "

```
SELECT ENR, COUNT(*) AS Anzahl, SUM(Wert) AS Gesamtwert
FROM Bild
GROUP BY ENR
```

Ergebnis

ENR	Anzahl	Gesamtwert
E1	5	3950
E2	2	250
E4	1	800
E5	3	3000
E6	3	570
E7	2	1050

3.5.3 Die HAVING-Klausel

Syntax

having-clause ::=

HAVING search-condition

- search-condition ist syntaktisch äquivalent zur Suchbedingung im WHERE einer SELECT-Anweisung. HAVING wirkt auf Gruppen von Tupeln einer Relation wie WHERE auf die Relation als Ganzes.

Beispiel 3.5e

„Gesucht sind die Orte, in denen mehrere Galerien ihren Sitz haben"

```
SELECT Sitz AS Ort
FROM Galerie
GROUP BY Sitz
HAVING COUNT(*)>1
```

Ergebnis

Ort
Paris

3.6 Ungeschachtelte Anfragen über mehrere Relationen

3.6.1 Mengenoperationen auf Relationen

Syntax

query-expression ::=

query-term

| query-expression { UNION | EXCEPT } query-term

query-term ::=

query-primary

| query-term INTERSECT query-primary

query-primary ::=

query-specification

| (query-expression)

- *UNION* bildet die Vereinigungs-, *EXCEPT* die Differenz- und *INTERSECT* die Schnittmenge der beteiligten Teilergebnisrelelationen.

Beispiel 3.6a

„Gesucht sind Bezeichnungen und Sitze aller Einrichtungen“

SELECT Bezeichnung, Sitz
FROM Museum
UNION
SELECT Bezeichnung, Sitz
FROM Galerie

Ergebnis

Bezeichnung	Sitz
Louvre	Paris
Nationalmuseum	Kopenhagen
Deutsches Museum	Berlin
National Gallery	London
Le Corbet	Paris
Galerie Villon	Paris
Nationalgalerie	Berlin

Beispiel 3.6c

"Gesucht sind die ENR der Museen, in deren Besitz sich keine Bilder befinden“

```
SELECT ENR
FROM Museum
EXCEPT
SELECT ENR
FROM Bild
```

Ergebnis

ENR
E3

3.6.2 Join-Anfragen

Der implizite Join

Beispiel 3.6e

„Gesucht sind die Titel der Bilder und die Namen Ihrer Maler“

```
SELECT Titel, Name
FROM Bild, Maler
WHERE Bild.KNR=Maler.KNR
```

Ergebnis

Titel	Name
Landschaft mit Pfeilen	Endler
Wasserlandschaft	van Goyen
Maria am Berge	Runge
Dame mit Hündchen	Lorrain
Bildnis Dirk van Bongen	Huysman
Verkündigung der Maria	Lorrain
Peng!	Endler
Landschaft ohne Pfeile	Endler
Bildnis des Dr. Franke	van Goyen
Hinter den Bergen	Runge
Dorfszene bei Gewitter	Huysman
Das Frühstück	Lorrain
Ansicht vom Gendarmenmarkt	Runge
Portrait Marte F.	Huysman

Beispiel 3.6g

„Gesucht sind Namen von lebenden Malern, von denen sich Bilder mit einem Wert größer 200 in Galeriebesitz befinden sowie die Bezeichnungen und Sitze dieser Galerien“

SELECT DISTINT x.Name, z.Bezeichnung AS Galerie, z.Sitz
FROM Maler x, Bild y, Galerie z
WHERE x.KNR=y.KNR
AND y.ENR=z.ENR
AND y.Wert>200
AND x.Tod IS NULL

Ergebnis

Name	Galerie	Sitz
van Goyen	Galerie Villon	Paris

Beispiel 3.6h

Gesucht sind die BNR und Titel aller Bilder, die in dem gleichen Jahr entstanden wie das Bild mit der Nummer 'B1'

SELECT x.BNR, x.Titel
FROM Bild x, Bild y
WHERE x.Jahr=y.Jahr
AND y.BNR='B1'

Ergebnis

BNR	Titel
B1	Landschaft mit Pfeilen
B9	Landschaft ohne Pfeile

Beispiel 3.6j

„Gesucht sind alle Angaben über Galerien, die Bilder derselben Maler besitzen, die auch die Galerie Villon besitzt sowie die Namen der Maler dieser Bilder“

SELECT DISTINCT u., v.Name AS Maler*
FROM Galerie x, Bild y, Bild z, Galerie u, Maler v
WHERE x.Bezeichnung='Galerie Villon'
AND x.ENR=y.ENR
AND y.KNR=z.KNR
AND z.ENR=u.ENR
AND z.KNR=v.KNR

Ergebnis

ENR	Bezeichnung	Sitz	Maler
E2	National Gallery	London	Endler
E6	Galerie Villon	Paris	Endler
E6	Galerie Villon	Paris	van Goyen

Beispiel 3.6k

„Gesucht sind die Galerien, die keine Bilder von Malern aus dem 18. Jh. besitzen“

```
SELECT *
FROM  Galerie
EXCEPT
SELECT x.*
FROM Galerie x, Bild y, Maler z
WHERE x.ENR=y.ENR
AND y.KNR=z.KNR
AND z.Geburt<1800
AND z.Tod>1699
```

Ergebnis

ENR	Bezeichnung	Sitz
E2	National Gallery	London
E6	Galerie Villon	Paris

Der explizite Join

Syntax (SQL2)

joined-table ::=

table-reference CROSS JOIN table-reference

| table-reference [NATURAL] [join-type] JOIN table-reference [join-spec]

| (joined-table)

join-spec ::=

ON search-condition

| USING (column-name [{ , column-name }...]

join-type ::=

INNER

| { LEFT | FULL | RIGHT } [OUTER]

- table-reference ist im einfachsten Fall ein Relationenname, kann aber auch eine SELECT-Anweisung oder eine joined-table selbst sein. Es ist damit insbesondere möglich, mehrere explizite Joins in einem Ausdruck miteinander zu verknüpfen.
- CROSS JOIN: kartesische Produkt (Kreuzprodukt) zweier Relationen.
- NATURAL natürlicher Verbund.

- ON explizite Angabe der Join-Bedingung.
- USING: Auswahl von Join-Attributen
- INNER: Inner-Join
- OUTER LEFT/RIGHT/FULL: linker/rechter/voller Outer-Join

Beispiel (siehe oben)

„Gesucht sind die Titel der Bilder und die Namen ihrer Maler“

```
SELECT Titel, Name
FROM Bild NATURAL JOIN Maler
```

oder

```
SELECT Titel, Name
FROM Bild JOIN Maler USING (KNR)
```

oder

```
SELECT Titel, Name
FROM Bild JOIN Maler ON Bild.KNR=Maler.KNR
```

Syntax (DB2)

```
joined-table ::=
    table-reference [ join-type ] JOIN table-reference ON search-condition
  | ( joined-table )

join-type ::=
    INNER
  | { LEFT | FULL | RIGHT } [ OUTER]
```

Beispiel 3.6l

„Gesucht sind Name und Sitz der Galerien und die Namen der Maler, deren Bilder sie besitzen“

```
SELECT x.Bezeichnung, x.Sitz, z.Name
FROM Galerie x JOIN Bild y ON x.ENR=y.ENR JOIN Maler z ON y.KNR=z.KNR
```

Beispiel 3.6o

„Gesucht sind ENR und Namen aller Museen und die Titel der Bilder, die sie besitzen“

```
SELECT x.ENR, x.Bezeichnung, y.Titel
FROM Museum x LEFT JOIN  Bild y ON x.ENR=y.ENR
```

Ergebnis

ENR	Bezeichnung	Titel
E1	Louvre	Maria am Berge
E1	Louvre	Dorfszene bei Gewitter
E1	Louvre	Hinter den Bergen
E1	Louvre	Verkündigung der Maria
E1	Louvre	Bildnis Dirk van Bongen
E3	Nationalmuseum	-
E4	Deutsches Museum	Auferstehung

3.7 Geschachtelte Anfragen

- Geschachtelte Anfragen sind SELECT-Anweisungen, die wiederum SELECT-Anweisungen als Unteranfragen (subqueries) enthalten.
- 2 Formen von Unteranfragen:

Syntax

row-subquery ::=

(query-expression)

Syntax

table-subquery ::=

(query-expression)

- row-subquery: erzeugt einen einzelnen Attributwert oder ein einzelnes Tupel
- table-subquery: erzeugt eine Wertemenge oder eine Relation.

3.7.1 Vergleichsanfragen

Syntax

comparsion-predicate ::=
value-expression comp-op row-subquery

- die row-subquery ist hier eine SELECT-Anweisung, die einen einfachen Wert erzeugt.

Beispiel 3.7a

„Gesucht sind die Titel der Bilder mit einem überdurchschnittlichem Wert“

```
SELECT Titel
FROM Bild
WHERE Wert >
      (SELECT AVG(Wert)
       FROM Bild)
```

Ergebnis

Titel
Dame mit Hündchen
Bildnis Dirk van Bongen
Verkündigung der Maria
Maria Magdalena
Hinter den Bergen
Auferstehung
Das Frühstück

Beispiel 3.7b

„Gesucht sind die Namen der Maler, die mindestens so0 viele Bilder gemalt haben wie Huysman“

```
SELECT x.Name
FROM Maler x, Bild y
WHERE x.KNR=y.KNR
GROUP BY x.KNR, x.Name
HAVING COUNT(y.BNR) >=
      (SELECT COUNT(z.BNR)
       FROM Bild z, Maler u
       WHERE z.KNR=u.KNR
       AND u.Name = ‘Huysman')
```

Ergebnis

Name
Endler
Huysman
Lorrain
Runge

3.7.2 Quantifizierte Anfragen

Syntax

quantified-comparsion-predicate ::=

row-value-constructor comp-op { ALL | { SOME | ANY } } table-subquery

- Ein Vergleich *comp-op ALL* ist wahr, wenn der aktuelle Wert des *row-value-constructor* mit allen der durch die Unteranfrage erzeugten Werte die Vergleichsbedingung erfüllt oder wenn die Unteranfrage keinen Wert erzeugt.

- *SOME* und *ANY* sind synonym. Ein Vergleich *comp-op SOME* oder *comp-op ANY* ist dann wahr, wenn der aktuelle Wert des *row-value-constructor* mit mindestens einem der durch die Unteranfrage erzeugten Werten die Vergleichsbedingung erfüllt. Der Vergleich ist falsch, wenn die Unteranfrage keinen Wert erzeugt.

Beispiel 3.7c

„Gesucht sind die Namen der Museen, in deren Besitz sich keine Bilder befinden."

```
SELECT Bezeichnung
FROM Museum
WHERE ENR <> ALL
      (SELECT ENR
      FROM Bild)
```

Ergebnis

Bezeichnung	Sitz
Nationalmuseum	Kopenhagen

Beispiel 3.7e

„Gesucht sind die Namen der Museen, in deren Besitz sich Bilder befinden."

```
SELECT Bezeichnung
FROM Museum
WHERE ENR = ANY
      (SELECT ENR
       FROM Bild)
```

Ergebnis

Bezeichnung
Louvre
Deutsches Museum

3.7.3 Die IN-Bedingung

Beispiel 3.7f (entspricht Beispiel 3.7e)

„Gesucht sind die Namen der Museen, in deren Besitz sich Bilder befinden."

```
SELECT Bezeichnung
FROM Museum
WHERE ENR IN
      (SELECT ENR
       FROM Bild)
```

Beispiel 3.7g

„Gesucht sind die Namen der Museen, in deren Besitz sich keine Bilder befinden."

```
SELECT Bezeichnung
FROM Museum
WHERE ENR NOT IN
      (SELECT ENR
       FROM Bild)
```

Beispiel 3.7h

„Gesucht sind Namen und Sitz von Galerien, die Bilder von Malern des 18. Jh. besitzen."

```
SELECT Bezeichnung, Sitz
FROM Galerie
WHERE ENR IN
      (SELECT ENR
       FROM Bild
       WHERE KNR IN
            (SELECT KNR
             FROM Maler
             WHERE Geburt<1800
             AND Tod>1699))
```

3.7.4 Die EXISTS-Bedingung

Syntax

exists-predicate ::=

EXISTS table-subquery

- Das *exist-predicate* ist wahr, wenn die *table-subquery* ein Ergebnis, d.h. mindestens ein Tupel liefert.

Beispiel 3.7j (entspricht Beispiel 3.7f)

„Gesuch sind Namen der Museen, in deren Besitz sich Bilder befinden.“

```
SELECT x.Bezeichnung
FROM Museum x
WHERE EXISTS
      (SELECT *
       FROM Bild y
      WHERE x.ENR = y. ENR)
```

Beispiel 3.7.k (entspricht Beispiel 3.7h)

„Gesucht sind Namen und Sitz von Galerien, die Bilder von Malern des 18. Jh. besitzen.“

```
SELECT x.Bezeichnung, x.Sitz
FROM Galerie x
WHERE EXISTS
      (SELECT *
       FROM Bild y
       WHERE x.ENR=y.ENR
       AND EXISTS
            (SELECT *
             FROM Maler z
             WHERE y.KNR=z.KNR
             AND Geburt<1800
             AND Tod>1699))
```

Beispiel 3.7l (entspricht Beispiel 3.7g)

„Gesucht sind Namen der Museen, in deren Besitz sich keine Bilder befinden.“

```
SELECT x.Bezeichnung
FROM Museum x
WHERE NOT EXISTS
      (SELECT *
       FROM Bild y
      WHERE x.ENR = y. ENR)
```

Beispiel 3.7m

„Gesucht sind die Namen der Galerien, die ausschießlich Bilder von Malern des 18.Jh. besitzen."

```
SELECT x.Bezeichnung
FROM Galerie x
WHERE NOT EXISTS
      (SELECT *
       FROM Bild y
       WHERE x.ENR=y.ENR
       AND NOT EXISTS
```

```
(SELECT *
 FROM Maler z
 WHERE y.KNR = z.KNR
 AND z.Geburt<1800
 AND z.Tod>1699))
```

4. SQL: Die Manipulationssprache

4.1 Die INSERT-Anweisung

Syntax

insert-statement ::=

INSERT INTO table-name [(column-name [{ , column-name }...])] insert-source

insert-source ::=

VALUES (value-expression [{ , value-expression }...])

| query-expression

- Die *INSERT*-Anweisung fügt Attributwerte aus einer *insert-source* in eine Relation *table-name* (Target-Relation) ein. Diese Relation muß bereits definiert sein

- Dem Namen der Target-Relation kann optional eine Liste von Attributnamen dieser Relation (*column-name*) folgen, die jedoch nicht alle Attributnamen der Target-Relation enthalten muß. Wird eine solche Liste angegeben, so muß die Anzahl der durch die *insert-source* erzeugten Werte (eines Tupels) mit der Anzahl der in der Liste angegebenen Attributnamen übereinstimmen.

- *VALUES*: explizite Angabe von einzufügenden Attributwerten durch eine in Klammern gesetzte Liste

- *query-expression*: erzeugen einer Menge von Wertetupeln durch eine SELECT-Anweisung (*query-specification* oder *join-table*).

Beispiel 4.1a

Der Relation Museum ist ein Tupel mit der ENR 'E8', der Bezeichnung 'Prado' und dem Sitz 'Madrid' hinzuzufügen

```
INSERT INTO Museum
VALUES ('E8', 'Prado', 'Madrid')
```

oder auch

INSERT INTO Museum(ENR, Bezeichnungng, Sitz)
VALUES ('E8', 'Prado', 'Madrid')

Ergebnis (Relation Museum)

ENR	Bezeichnung	Sitz
	. . .	
E4	Deutsches Museum	Berlin
E8	Prado	Madrid

Beispiel 4.1b

Der Bild-Relation ist ein Tupel mit der Bildnummer 'B17', dem Titel 'Pfeile ohne Landschaft' des Malers mit der Nummer 'K2' hinzuzufügen

INSERT INTO Bild(BNR, Titel, KNR)
VALUES ('B17', 'Pfeile ohne Landschaft', 'K2')

Ergebnis (Relation Bild)

BNR	Titel	Jahr	Wert	KNR	ENR
	. . .				
B16	Portrait Marte F.	1910	600	K7	E7
B17	Pfeile ohne Landschaft	-	-	K2	-

Beispiel 4.1c

Gegeben sei eine (zunächst leere) Relation Werke(KNR, Maler, Anzahl), in die für jeden Maler seine ENR, der Name und die Anzahl der von ihm gemalten Bilder durch Übernahme der Daten aus der Kunst-Datenbank eingetragen werden soll.

INSERT INTO Werke(KNR, Maler, Anzahl)
SELECT x.KNR, x.Name, COUNT(y.bnr)
FROM Maler x, Bild y
WHERE x.KNR=y.KNR
GROUP BY x.KNR, x.Name

Ergebnis (Relation Werke)

KNR	Maler	Anzahl
K1	Lorrain	3
K2	Endler	3
K4	Runge	3
K5	van Goyen	2
K7	Huysman	3

4.2 Die DELETE-Anweisung

Syntax

delete-statement ::=

DELETE FROM table-name [range-variable] [WHERE search-condition]

- Die *DELETE*-Anweisung löscht in einer Relation *table-name* alle Tupel, die die *search-condition* erfüllen.
- Ist keine WHERE-Bedingung angegeben, so werden alle Tupel der Relation gelöscht. Die Relationenstruktur bleibt jedoch erhalten.

Beispiel 4.2a

„Aus der Relation Bild sind alle Bilder mit unbekanntem Maler zu entfernen“

```
DELETE FROM Bild
WHERE KNR IS NULL
```

Beispiel 4.2b

„Aus der Maler-Relation sind alle Maler zu entfernen, die keine Bilder gemalt haben (nicht in der Bild-Relation vorkommen)“

```
DELETE FROM Maler
WHERE KNR NOT IN
      (SELECT KNR
       FROM Bild
       WHERE KNR IS NOT NULL)
```

Beispiel 4.2c

„Aus der Bild-Relation sind die Bilder zu löschen, deren Werte unter dem Durchschnittswert liegen“

```
DELETE FROM Bild
WHERE Wert <
      SELECT AVG(Wert)
      FROM Bild
```

4.3 Die UPDATE-Anweisung

Syntax

update-statement ::=

UPDATE table-name [range-variable]

SET coulmn-name = row-value-constructor [{ , column-name = row-value-constructor}...]

[WHERE search-condition]

- Es werden die in einer *UPDATE*-Anweisung angegebenen Werte der Attribute *column-name* durch die Werte des zugewiesenen row-value-constructor ersetzt, wenn die angegeben WHERE-Bedingung erfüllt ist.

- Ist keine WHERE-Bedigung angegeben, werden in allen Tupeln der Relation die Attributwerte ersetzt.

Beispiel 4.3a

Der Maler Endler ist 1999 verstorben, die Daten sind entsprechend zu ändern.

```
UPDATE Maler
SET Tod = 1999
WHERE Name = ‘Endler’
```

Beispiel 4.3b

Alle Bilder des Malers Endler sind in ihrem Wert auf das dreifache gestiegen, die Relation Bild soll entsprechend verändert werden“

```
UPDATE Bild
SET Wert=3*Wert
WHERE KNR =
      (SELECT KNR
       FROM Maler
       WHERE Name='Endler')
```

Beispiel 4.3d

Der Wert der Bilder von Huysman wir neu festgelegt. Jedes seiner Bilder erhält als Wert den bisherigen Durchschnittswert aller seiner Bilder.

```
UPDATE Bild x
SET x.Wert =
      (SELECT AVG(Wert)
       FROM Bild y
       WHERE y.KNR=x.KNR)
WHERE x.KNR =
      (SELECT KNR
       FROM Maler
       WHERE Name='Huysman')
```

5. SQL: Die Definitionssprache

Syntax

table-definition ::=

CREATE TABLE table-name

(column-definition [{ , column-definition }...]

[table-constraint [{ , table-constraint }...]])

column-definition ::=

column-name data-type [column-constraint [{ , column-constraint }...]]

- Definition einer Relationenstruktur (Relationenname, Attribute und Datentypen)
- Datentypen (Auswahl DB2)

Datentyp	Beschreibung
SMALLINT	16-Bit-Integer
INTEGER	32-Bit-Integer
DECIMAL(p,s)	Dezimalzahl, p Ziffern, s Ziffern hinter Dezimalpunkt
CHAR(n)	String fester Länge mit n Zeichen, n<=254
VARCHAR(n)	String variabler Länge mit maximal n Zeichen, n<=4000
DATE	Datum 'dd.mm.yyyy'
TIME	Zeit 'hh:mm:ss'

Beispiel

Relationenstruktur Firma (FNR, Name, Sitz, Umsatz, Gewinn)

```
CREATE TABLE Firma
   (FNR      SMALLINT,
    Name     VARCHAR(20),
    Sitz     CHAR(3),
    Umsatz   DECIMAL(7),
    Gewinn   DECIMAL(7))

INSERT INTO Firma (FNR,Name,Sitz,Umsatz,Gewinn)
   VALUES(1,'TechnoCom','HH',1000000,-500000);

INSERT INTO Firma (FNR,Name,Sitz,Umsatz,Gewinn)
   VALUES(2,'ProCom','Rostock',1000000,250000);
```

- Eine INSERT/UPDATE/DELETE-Anweisung wird nur ausgeführt, wenn keine der für eine Relation definierten Integritätsbedingungen verletzt ist

Attributbedingungen

Syntax

column-constraint ::=

| NOT NULL

| UNIQUE

| CHECK (check-condition-1)

| DEFAULT { const | NULL }

- NOT NULL: Attributwert muß vorhanden sein
- UNIQUE: Attributwert muß eindeutig sein
- CHECK (check-condition-1): Attributwert muß die angegebene Bedingung erfüllen
- DEFAULT { const | NULL }: wird bei einem INSERT für dieses Attribut kein Wert angegeben, so erhält es vom System den Wert NULL oder const.

Beispiel

Relationenstruktur Firma (FNR, Name, Sitz, Umsatz, Gewinn) mit Bedingungen

```
CREATE TABLE Firma
   (FNR      SMALLINT NOT NULL UNIQUE,
    Name     VARCHAR(20) NOT NULL,
    Sitz     CHAR(3) CHECK(Sitz IN ('HRO','HH','B','M')) DEFAULT NULL,
    Umsatz   DECIMAL(7) CHECK(Umsatz<10000000 AND Umsatz>=0),
    Gewinn   DECIMAL(7) DEFAULT 0);

INSERT INTO Firma (FNR,Name,Sitz,Umsatz,Gewinn)
   VALUES(1,'TechnoCom','HH',1000000,-500000);

INSERT INTO Firma (FNR,Name,Sitz,Umsatz,Gewinn)
   VALUES(1,'ProTech','HR0',1000000,15000);

INSERT INTO Firma (FNR,Sitz,Umsatz,Gewinn)
   VALUES(2,'HR0',1000000,15000);

INSERT INTO Firma (FNR,Name,Sitz,Umsatz,Gewinn)
   VALUES(2,'ProTech','HST',1000000,15000);

INSERT INTO Firma (FNR,Name,Sitz,Umsatz,Gewinn)
   VALUES(2,'TechnoCom','HH',-1000000,-500000);

INSERT INTO Firma (FNR,Name,Umsatz)
   VALUES(2,'ProTech',1000000);
```

Syntax

table-constraint ::=

| CHECK (check-condition-2)

| UNIQUE (column-name [{ , column-name }...]

- CHECK (check-condition-2): wie oben, kann sich jedoch auf mehrere Attributwerte eines jeden Tupels beziehen

- UNIQUE (column-name [{ , column-name }...]: wie oben, kann sich jedoch auf mehrere Attributwerte eines jeden Tupels beziehen

Beispiel

Relationenstruktur Firma (FNR, Name, Sitz, Umsatz, Gewinn) mit Bedingungen

```
CREATE TABLE Firma
   (FNR       SMALLINT NOT NULL UNIQUE,
    Name      VARCHAR(20) NOT NULL,
    Sitz      CHAR(3) CHECK(Sitz IN ('HRO','HH','B','M')) DEFAULT NULL,
    Umsatz    DECIMAL(7) CHECK(Umsatz<10000000 AND Umsatz>=0),
    Gewinn    DECIMAL(7) DEFAULT 0,
    CHECK(Umsatz>Gewinn));

INSERT INTO Firma (FNR,Name,Sitz,Umsatz,Gewinn)
   VALUES(1,'TechnoCom','HH',1000000,2000000);
```

Primärschlüsselbedingungen (intrarelationale IB)

Syntax (Fortführung)

table-constraint ::=

. . .

| PRIMARY KEY (column-name [{ , column-name }...])

- Definition eines Attributs oder einer Attributkombination als Primärschlüssel

Beispiel

Relationenstruktur Firma (FNR, Name, Sitz, Umsatz, Gewinn) mit Primärschlüssel

```
CREATE TABLE Firma
   (FNR       SMALLINT NOT NULL,
    Name      VARCHAR(20) NOT NULL,
    Sitz      CHAR(3) CHECK(Sitz IN ('HRO','HH','B','M')) DEFAULT NULL,
    Umsatz    DECIMAL(7) CHECK(Umsatz<10000000 AND Umsatz>=0),
    Gewinn    DECIMAL(7) DEFAULT 0,
    CHECK(Umsatz>Gewinn),
    PRIMARY KEY(FNR));
```

Fremdschlüsselbedingungen (interrelationale IB)

Fremdschlüssel

Ein Fremdschlüssel $F \subseteq (A_1,...,A_n)$ einer Relation $R_1(A_1,...,A_n)$ ist eine Attributkombination, die

a) kein Schlüssel von R_1 ist und

b) Schlüssel einer weiteren Relation R_2 ist.

Beispiel

Firma (FNR, Name, Sitz, Umsatz, Gewinn)
Artikel (ANR, Bezeichnung, *Hersteller*, Herst_Ort)
Verkauf (*FNR*, *ANR*, Menge)
Abteilung (*FNR*, Name, Leiter)

Syntax (Fortführung)

table-constraint ::=

. . .

| FOREIGN KEY (column-name [{ , column-name }...]) reference-specification

reference-specification ::=

REFERENCES table-name [(column-name [{ , column-name }...])]

[ON DELETE { CASCADE | SET NULL | RESTRICT | NO ACTION }]

[ON UPDATE { RESTRICT | NO ACTION }]

- Definition eines Attributs oder einer Attributkombination als Fremdschlüssel
- REFERENCES: Relation (Vaterrelation), in der der Fremdschlüssel der in der CREATE-Anweisung definierten Relation (Kindrelation) als Primärschlüssel auftritt, ggf. explizite Angabe der Schlüsselattribute
- ON DELETE/UPDATE: legt die Maßnahmen des Systems beim Löschen/Ändern eines Tupels der Vaterrelation fest

Integritätserhaltende Maßnahmen des Systems

1. Das Hinzufügen (INSERT) eines Kind-Tuples, dessen Fremdschlüsselwert mit keinem Primärschlüsselwert der Vaterrelation übereinstimmt, wird abgewiesen.
2. Das Ändern (UPDATE) eines Fremschlüsselwertes eines Kind-Tupels auf einen Wert (ungleich NULL), der mit keinem Primärschlüsselwert der Vaterrelation übereinstimmt, wird abgewiesen.

3. Das Löschen eines Vater-Tupels, dessen Primärschlüsselwert mit Fremschlüsselwerten der Kind-Relation übereinstimmt, wird bei:
   - ON DELETE NO ACTION: abgewiesen
   - ON DELETE SET NULL: ausgeführt und die betroffenen Fremdschlüsselwerte der Kind-Tupel (wenn möglich) auf NULL gesetzt.
   - ON DELETE CASCADE: ausgeführt und die betroffenen Kind-Tupel ebenfalls gelöscht.

4. Das Ändern des Primärschlüsselwertes eines Vater-Tupels, wird bei ON UPDATE NO ACTION nur abgewiesen, wenn kein Kind-Tupel mit diesem neuen Wert als Fremdschlüsselwert existiert.

Beispiel

```
CREATE TABLE Artikel
   (ANR          SMALLINT NOT NULL,
    Bezeichnung  VARCHAR(20) NOT NULL,
    Herst_Firma  SMALLINT,
    Herst_Datum DATE,
    Herst_Ort    VARCHAR(15),
    PRIMARY KEY(ANR),
    FOREIGN KEY(Herst_Firma) REFERENCES Firma(FNR) ON DELETE SET NULL);

CREATE TABLE Verkauf
   (FNR       SMALLINT NOT NULL,
    ANR       SMALLINT NOT NULL,
    MENGE     SMALLINT CHECK(Menge < 10000),
    PRIMARY KEY(FNR,ANR),
    FOREIGN KEY(FNR) REFERENCES Firma ON DELETE CASCADE,
    FOREIGN KEY(ANR) REFERENCES Artikel ON DELETE CASCADE );
```

Ändern und Löschen von Relationenstrukturen

Syntax

alter-table-statement ::=

ALTER TABLE table-name action

action ::=

ADD column-definition

| ADD table-constraint

- Hinzufügen neuer Attribute und Bedingungen

Syntax

rename-table-statement ::=

RENAME TABLE table-name TO new-table-name

- Umbenennen eines Relationennamens

Syntax

drop-table-statement ::=

DROP TABLE table-name

- Löschen einer Relation

6. Datenbank-Entwurf

Datenbankentwurf

Abbildung von Objekttypen und Beziehungstypen eines UoD in Relationenstrukturen einer Datenbank

Problemanalyse

- Informationsbedarf
- Umfang der UoD
- Struktur des UoD

6.1 Das Entity-Relationship-Modell

- P. Chen
  "The Entity-Relationship-Model - Toward a Unified View of Data"
  ACM TODS 1976

- Ein **Entity** (Entität, Objekt) ist ein abgrenzbarer realer oder mentaler Gegenstand des UoD, der identifiziert werden kann und über den Informationen in der Datenbank gespeichert werden sollen. Es wird durch seine Eigenschaften (Attributwerte) beschrieben.

- Ein **Entity-Typ** wird durch einen Namen bezeichnet, durch eine Menge von Attributen beschrieben und durch eine Teilmenge dieser Attribute (Schlüsselattribute) identifiziert.

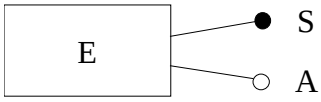
- Ein **Relationship** (Beziehung) ist ein Zusammenhang zwischen zwei oder mehreren Entities, über den Informationen in der Datenbank gespeichert werden sollen. Er kann durch seine Eigenschaften (Attributwerte) beschrieben werden.

- Ein R**elationship-Typ** wird durch einen Namen bezeichnet und kann durch eine Menge von Attributen beschrieben werden. Er wird identifiziert durch die Schlüsselattribute der bezogenen Entity-Typen.

- Entities existieren unabhängig voneinander, Relationships nur dann, wenn die bezogenen Entities existieren (Integritätsbedingung)

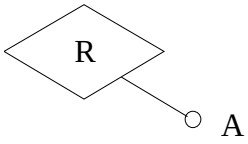
Entity-Relationship-Diagramm

- **Entity-Typ**



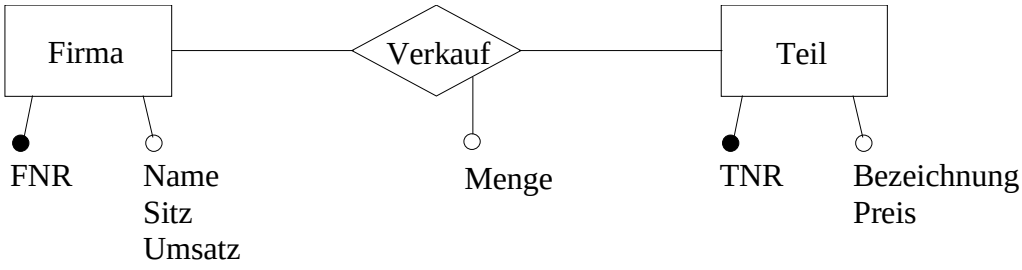
E: Entity-Typ-Name
S: Schlüsselattribut(e)
A: weitere optionale Attribute

- **Relationship-Typ**



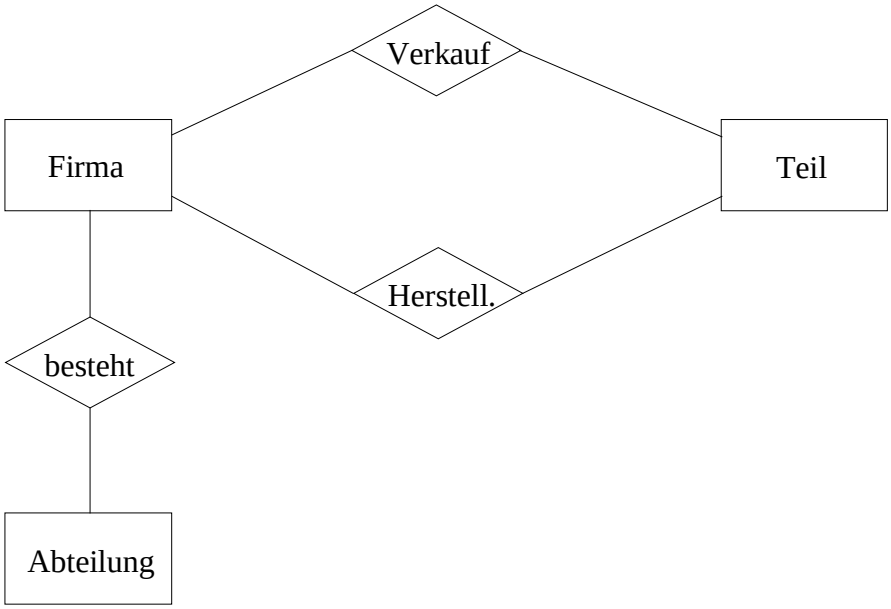
R: Relationship-Typ-Name
A: optionale Attribute

Beispiel
Firmen verkaufen Teile



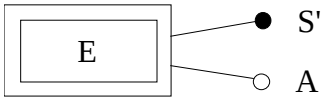
Beispiel

Firmen verkaufen Teile
Firmen bestehen aus Abteilungen
Teile werden durch Firmen hergestellt



- Ein **schwacher Entity-Typ** ist ein Entity-Typ, der sich auf einen anderen Entity-Typ bezieht. Er wird durch einen Namen bezeichnet und durch eine Menge von Attributen beschrieben. Sei Schlüssel setzt sich zusammen aus einer Teilmenge dieser Attribute den Schlüsselattributen des Entity-Typs, auf den er sich bezieht.

- **schwacher Entity-Typ**



E: Entity-Typ-Name
S': eigene Schlüsselattribut(e)
A: weitere optionale Attribute

Beispiel

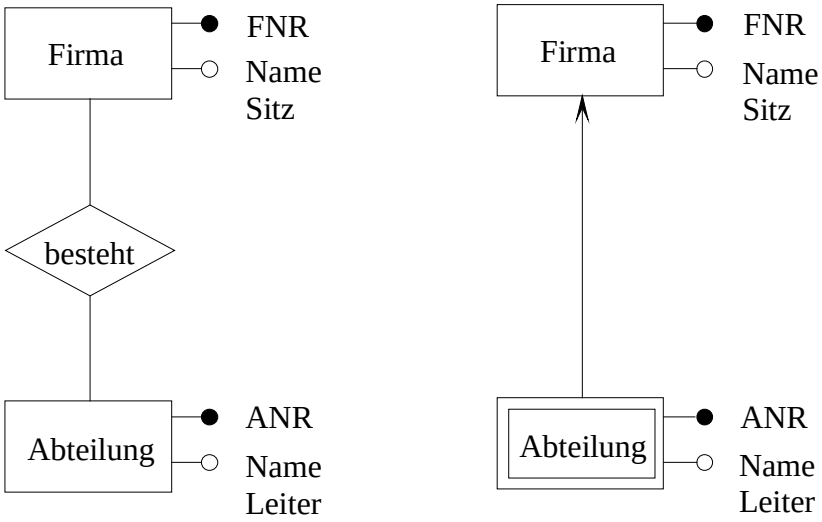
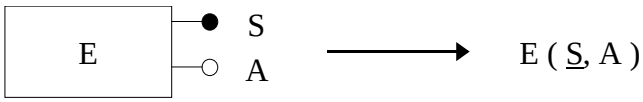
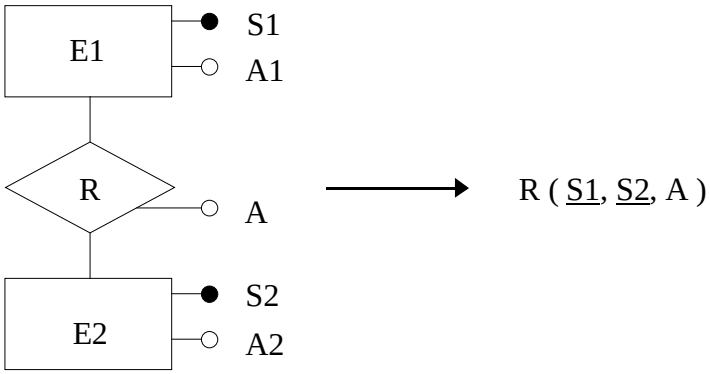


Abbildung von E-R-Diagrammen in Relationenstrukturen

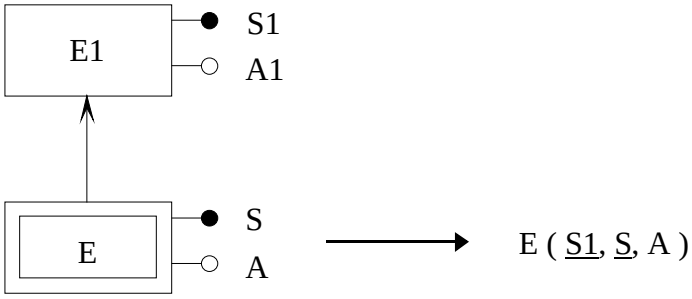
- jeder Entity-Typ wird in genau eine Relation abgebildet



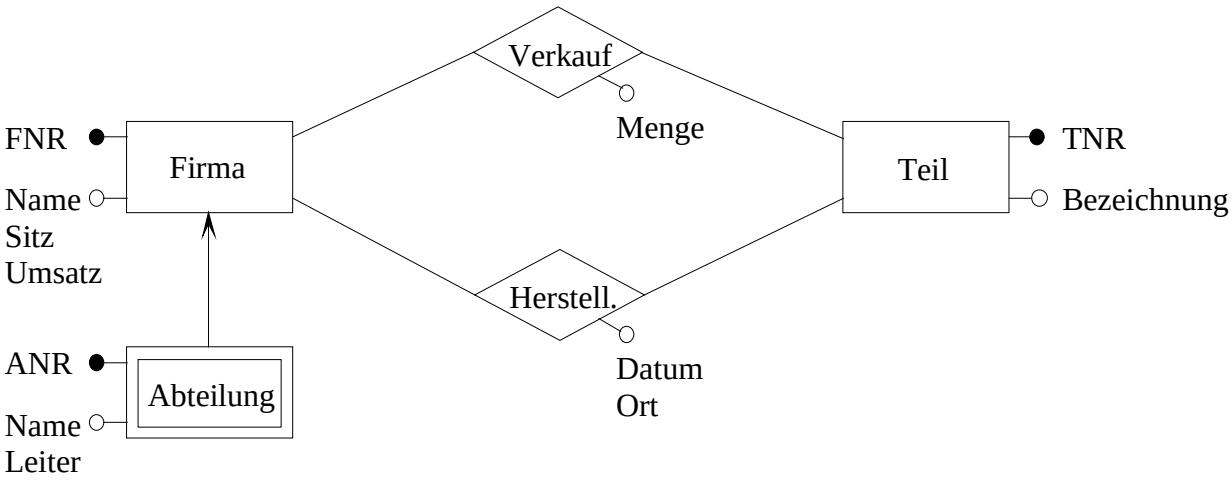
- jeder Relationship-Typ wird in genau eine Relation abgebildet. Der Schlüssel der Relation setzt sich aus den Schlüsseln der bezogenen Entity-Typen zusammen



- jeder schwache Entity-Typ wird in genau eine Relation abgebildet. Der Schlüssel der Relation setzt sich aus seinen eigenen Schlüsselattributen und dem Schlüssel des Entity-Typs, auf den er sich bezieht, zusammen.



Beispiel

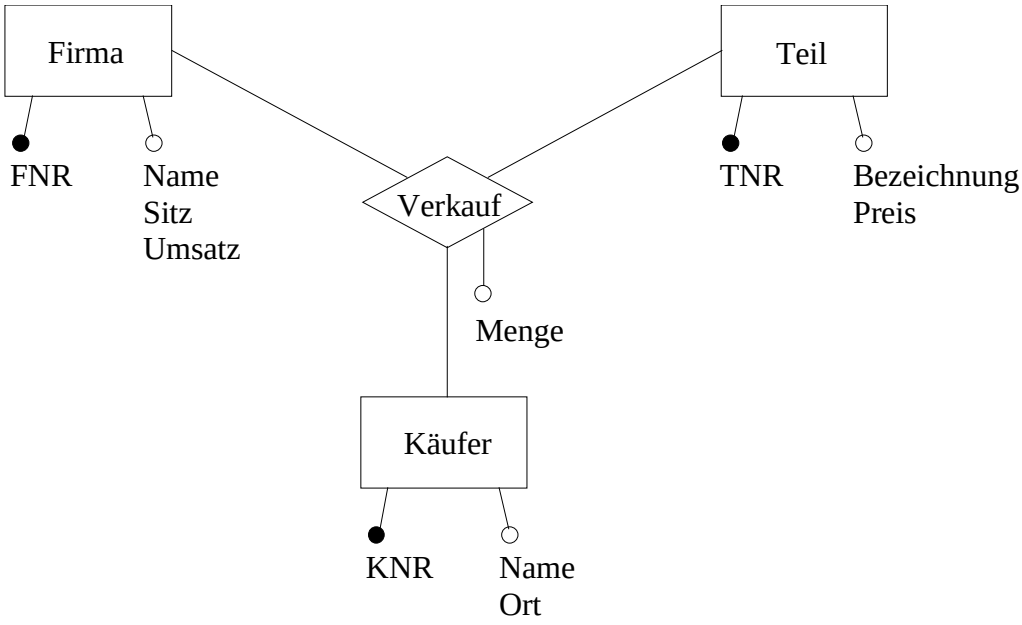


Relationenschema:

Firma (FNR, Name, Sitz, Umsatz)
Teil (TNR, Bezeichnung)
Verkauf (FNR, TNR, Menge)
Herstellung (FNR, TNR, Datum, Ort)
Abteilung (FNR, ANR, Name, Leiter)

Beispiel

Teile werden von Firmen an Käufer verkauft

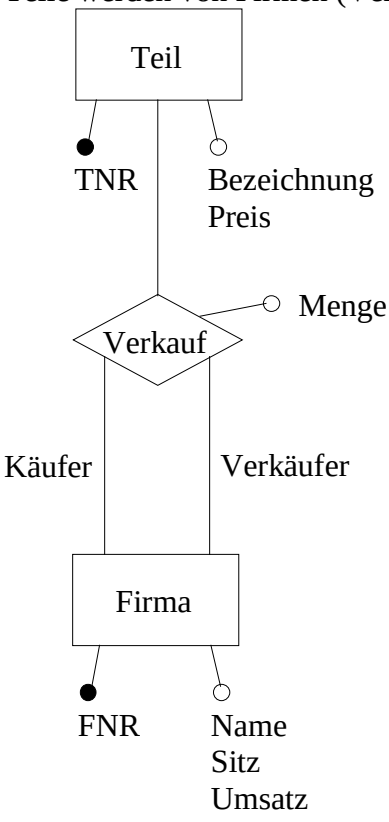


Relationenschema:

Firma (FNR, Name, Sitz, Umsatz)
Teil (TNR, Bezeichnung)
Käufer (KNR, Name, Ort)
Verkauf (FNR, TNR, KNR, Menge)

Beispiel

Teile werden von Firmen (Verkäufer) an Firmen (Käufer) verkauft



Relationenschema:

Teil (TNR, Bezeichnung, Preis)
Firma (FNR, Name, Sitz, Umsatz)
Verkauf (V-FNR, K-FNR, TNR, Menge)

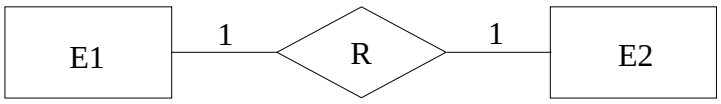
Arten von Relationships

- Sei R ein Relationship zwischen den Entity-Typen E1 und E2

  Ein Relationship heißt **einfach** (zu-eins-Relationship), wenn zu einem e1∈ E1 höchsten ein e2 ∈ E2 in Beziehung stehen kann.

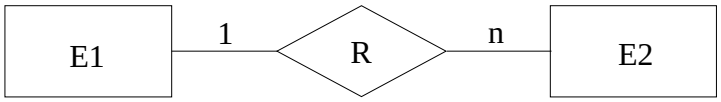
  Ein Relationship heißt **vielfach** (zu-viele-Relationship), wenn zu einem e1∈ E1 mehrere e2 ∈ E2 in Beziehung stehen können.

- **1:1-Relationship**



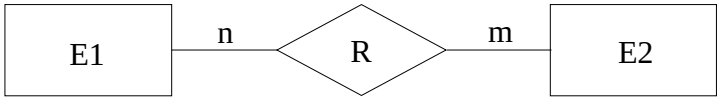
ein e1 kann zu höchstens einem e2 und ein e2 zu höchstens einem e1 in Beziehung stehen.

- **1:n-Relationship**



ein e1 kann zu vielen e2 und ein e2 zu höchstens einem e1 in Beziehung stehen.

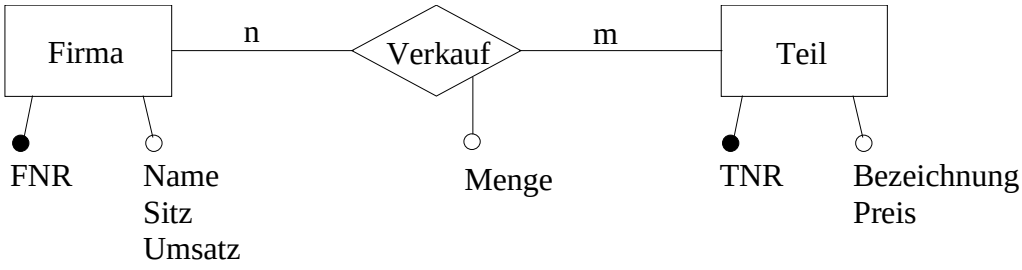
- **n:m-Relationship**



ein e1 kann zu vielen e2 und ein e2 zu vielen e1 in Beziehung stehen.

Beispiel

Eine Firma verkauft mehrere Teile(typen) und ein Teil(typ) wird von mehreren Firmen verkauft



Integritätsbedingung: ein Teil(typ) wird immer nur von einer Firma verkauft

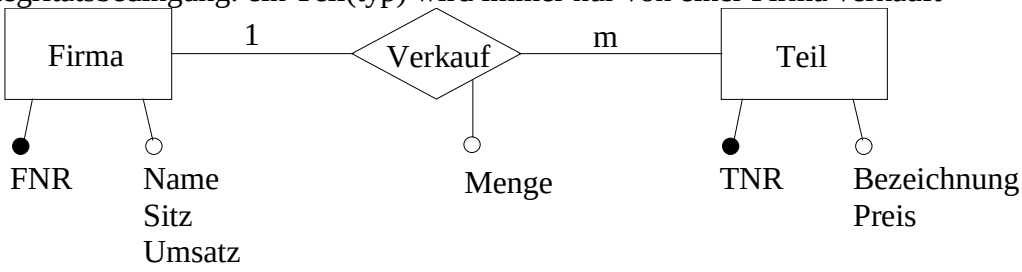
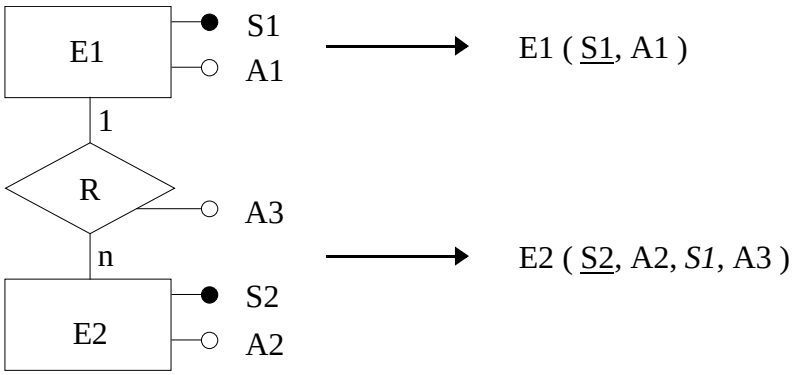


Abbildung von 1:n-Relationships in Relationenstrukturen

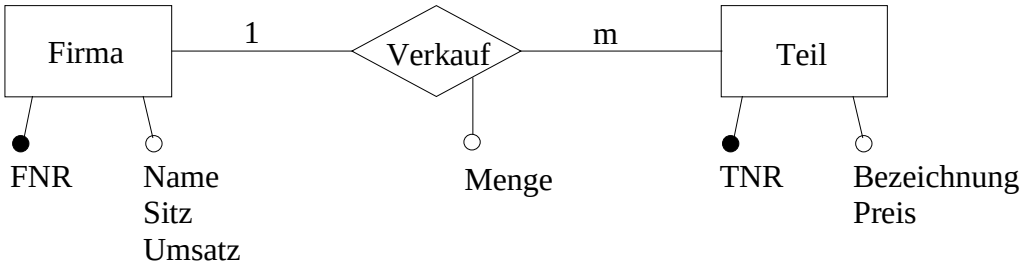
- **Fremdschlüssel**
  Ein Fremdschlüssel $F \subseteq (A_1,...,A_n)$ einer Relation $R_1(A_1,...,A_n)$ ist eine Attributkombination, die
  a) kein Schlüssel von R_1 ist und
  b) Schlüssel einer weiteren Relation R_2 ist.

- jede 1:n-Relationship als Fremdschlüsselbeziehung in die in vielfacher Beziehung stehende Entity-Relation abgebildet:



(Fremdschlüssel kursiv)

Beispiel



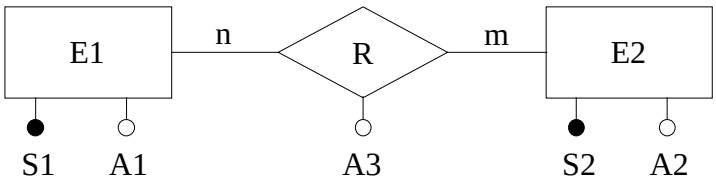
Relationenschema:

Firma (FNR, Name, Sitz, Umsatz)
Teil (TNR, Bezeichnung, Preis, *FNR*, Menge)

(Fremdschlüssel kursiv)

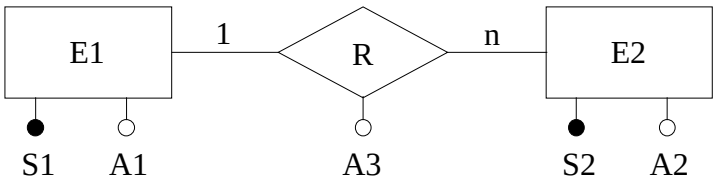
Abbildungen der Konzepte des E-R-Modells in Relationen

- **n:m-Beziehungen**



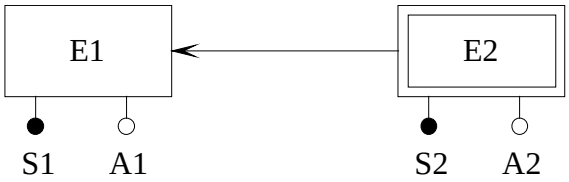

```
CREATE TABLE  E1
   ( S1   data-type    NOT NULL,
     A1   data-type,
     PRIMARY KEY(S1) );
CREATE TABLE  E2
   ( S2   data-type    NOT NULL,
     A2   data-type,
     PRIMARY KEY(S2) );
CREATE TABLE  R
   ( S1   data-type    NOT NULL,
     S2   data-type    NOT NULL,
     A3   data-type,
     PRIMARY KEY(S1,S2),
     FOREIGN KEY(S1) REFERENCES E1(S1) ON DELETE CASCADE,
     FOREIGN KEY(S2) REFERENCES E2(S2) ON DELETE CASCADE );
```

- **1:n-Beziehungen**



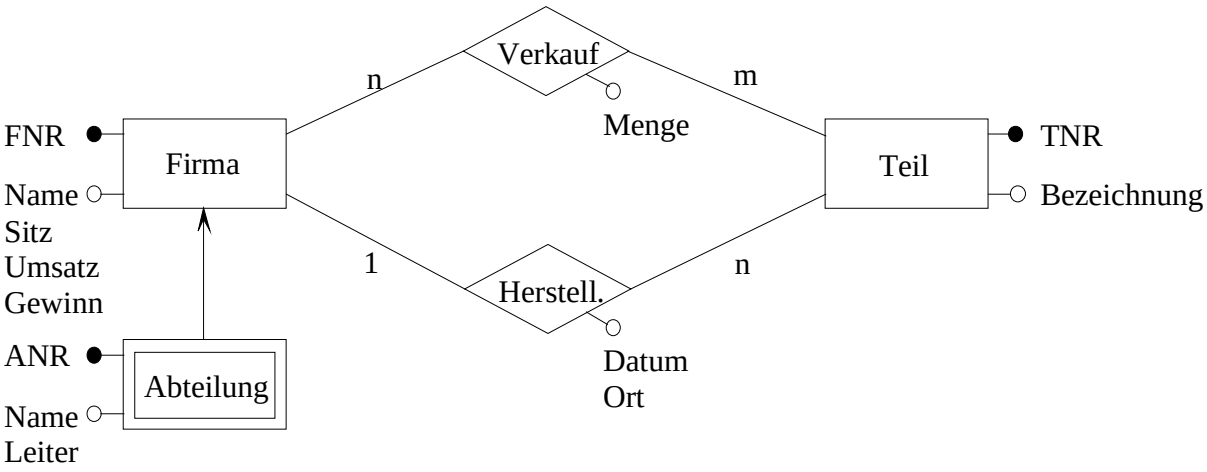

```
CREATE TABLE  E1
   ( S1   data-type    NOT NULL,
     A1   data-type,
     PRIMARY KEY(S1) );
CREATE TABLE  E2
   ( S2   data-type    NOT NULL,
     A2   data-type,
     S1   data-type,
     A3   data-type,
     PRIMARY KEY(S2)
     FOREIGN KEY(S1) REFERENCES E1(S1) ON DELETE SET NULL );
```

- **schwache Entity-Typen**




```
CREATE TABLE  E1
   ( S1   data-type    NOT NULL,
     A1   data-type,
     PRIMARY KEY(S1) );
CREATE TABLE  E2
   ( S1   data-type    NOT NULL,
     S2   data-type    NOT NULL,
     A2   data-type,
     PRIMARY KEY(S1,S2)
     FOREIGN KEY(S1) REFERENCES E1(S1) ON DELETE CASCADE );
```

Beispiel




```
CREATE TABLE Firma
   (FNR          SMALLINT NOT NULL,
    Name         VARCHAR(20) NOT NULL,
    Sitz         CHAR(3) CHECK(Sitz IN ('HRO','HH','B','M')) DEFAULT NULL,
    Umsatz       DECIMAL(7) CHECK(Umsatz<10000000 AND Umsatz>=0),
    Gewinn       DECIMAL(7) DEFAULT 0,
    CHECK(Umsatz>Gewinn),
    PRIMARY KEY(FNR));

CREATE TABLE Teil
   (TNR          SMALLINT NOT NULL,
    Bezeichnung  VARCHAR(20) NOT NULL,
    Herst_Firma  SMALLINT,
    Herst_Datum DATE,
    Herst_Ort    VARCHAR(15),
    PRIMARY KEY(TNR),
    FOREIGN KEY(Herst_Firma) REFERENCES Firma(FNR) ON DELETE SET NULL);

CREATE TABLE Verkauf
   (FNR       SMALLINT NOT NULL,
    TNR       SMALLINT NOT NULL,
    MENGE     SMALLINT CHECK(Menge < 10000),
    PRIMARY KEY(FNR,TNR),
    FOREIGN KEY(FNR) REFERENCES Firma ON DELETE CASCADE,
    FOREIGN KEY(TNR) REFERENCES Teil ON DELETE CASCADE );

CREATE TABLE Abteilung
   (FNR       SMALLINT NOT NULL,
    ANR       SMALLINT NOT NULL,
    Name      VARCHAR(20) NOT NULL,
    Leiter    VARCHAR(15) DEFAULT 'n.n.',
    PRIMARY KEY(FNR,ANR),
    FOREIGN KEY(FNR) REFERENCES Firma ON DELETE CASCADE );
```

6.2 Normalisierung

- **Normalisierung**

  Analytisches Verfahren zur Reduktion von (Informations-) Redundanzen in relationalen Strukturen durch Aufspaltung von Relationen

Beispiel

Person	Name	Vorname	Wohnort	Kind	K_Alter
	...	...	...	...	...
	Meier	Max	HST	Paul	4
	Meier	Max	HST	Willi	12
	Meier	Max	HST	Anna	17
	Müller	Liese	HRO	Hannes	15
	...	...	...	...	...

- unnormalisierte Relationen (redundante Informationen) führen in der Regel zu Anomalien (Mehraufwand) bei der Datenmanipulation in einer Datenbank.

6.2.1 Grundlagen

- Die **Attributkombination A einer Relation R** ist die Menge der Attribute von R

  $A = (A_1, \ldots, A_n)$ von $R(A_1, \ldots, A_n)$

- Eine **Attributkombination X bezüglich einer Relation R** ist eine Teilmenge der Attributkombination A von R

  $X \subseteq A$

Funktionale Abhängigkeit (FD)

Seien X und Y zwei Attributkombinationen bezüglich einer Relation R. Y heißt funktional abhängig von X:

$X \rightarrow Y$

wenn für beliebige Tupel $t_1, t_2 \in R$ gilt:

wenn $\prod_X(t_1) = \prod_X(t_2)$ dann $\prod_Y(t_1) = \prod_Y(t_2)$

Anmerkung
1. Die Implikation gilt nicht umgekehrt.
2. Die Disjunktheit von X und Y ist nicht verlangt.

Beispiel

Firma(FNR, Name, Sitz, Umsatz, Gewinn)
(FNR) $\rightarrow$ (Name, Sitz, Umsatz, Gewinn)
(FNR, Name) $\rightarrow$ (Umsatz, Gewinn)

Schlüssel

Eine Attributkombination S bezüglich einer Relation R mit der Attributkombination A ist Schlüssel von R, wenn gilt:

a) $S \rightarrow A$ und

b) Es existiert kein $S' \subset S$ mit $S' \rightarrow A$

Volle funktionale Abhängigkeit (FFD)

Seien X und Y zwei Attributkombinationen bezüglich einer Relation R. Y heißt voll funktional abhängig von X, wenn Y von X und von keiner echten Teilmenge von X funktional abhängig ist.

Beispiel

Firma(FNR, Name, Sitz, Umsatz, Gewinn)
(FNR, Name) $\overset{fd}{\rightarrow}$ (Umsatz, Gewinn)
(FNR) $\overset{ffd}{\rightarrow}$ (Name, Sitz, Umsatz, Gewinn)

Funktionale Abhängigkeit der Komponenten

Seien X und Y zwei Attributkombinationen bezüglich einer Relation R mit der Attributkombination A und $Y = (B_1, \ldots, B_m)$ mit $B_i \in A$, dann gilt

$$X \rightarrow Y \quad \text{g.d.w.} \quad X \rightarrow B_1 \wedge X \rightarrow B_2 \wedge \ldots \wedge X \rightarrow B_m$$

Folgerung

Eine Attributkombination S bezüglich einer Relation $R(A_1, \ldots, A_n)$ ist Schlüssel von R, wenn gilt:

$$S \rightarrow A_1 \wedge S \rightarrow A_2 \wedge \ldots \wedge S \rightarrow A_n$$

- **Verkettung** zweier Attributkombinationen $X = (B_1, \ldots, B_m)$ und $Y = (C_1, \ldots, C_n)$

$$XY = (B_1, \ldots, B_m) \cup (C_1, \ldots, C_n)$$

Projektivität

Wenn $Y \subseteq X$ dann $X \rightarrow Y$

Beispiel

Firma-Relation: wenn X = (Name, Sitz) dann (Name, Sitz) → Name und (Name, Sitz) → Sitz

Distributivität

Wenn $X \rightarrow YZ$ dann $X \rightarrow Y \wedge X \rightarrow Z$

Additivität

Wenn $X \rightarrow Y \wedge X \rightarrow Z$ dann $X \rightarrow YZ$

Transitivität

Wenn $X \rightarrow Y \wedge Y \rightarrow Z$ dann $X \rightarrow Z$

Beispiel

Mitarbeiter(MNR, Name, Abteilung, Chef)
MNR → (Name, Abteilung)
Abteilung → Chef

Transitive Abhängigkeit (TD)

Seien X, Y und Z paarweise disjunkte Attributkombinationen bezüglich einer Relation R. Z heißt transitiv abhängig von X (also X → Z), wenn

$X \rightarrow Y$ und $Y \rightarrow Z$ aber nicht $Y \rightarrow X$

Verlustfreier Join

Seien X, Y und Z paarweise disjunkte Attributkombinationen bezüglich einer Relation $R(A_1, \ldots, A_n)$ mit $XYZ = (A_1, \ldots, A_n)$. Dann gilt:

Wenn $X \rightarrow Y$ oder $X \rightarrow Z$ dann $R = \prod_{XY}(R) \otimes \prod_{XZ}(R)$

Beispiel

Firma(FNR, Name, Sitz, Umsatz, Gewinn)
FNR → (Umsatz, Gewinn)

$$\text{Firmendaten} = \prod_{FNR,Name,Sitz}(\text{Firma})$$

$$\text{Firmenbilanzen} = \prod_{FNR,Umsatz,Gewinn}(\text{Firma})$$

$$\text{Firma} = \text{Firmendaten} \otimes \text{Firmenbilanzen}$$

6.2.2 Normalformen

Erste Normalform

Eine Relation R befindet sich in der 1.NF, wenn jedes Attribut von R nur über elementare Ausprägungen (Werte) verfügt.

Beispiel

Belegung(GastNr, ZimmerNr, Tag, Name, Vorname, Anschrift)
mögliche Schlüssel: (GastNr, Tag) → ZimmerNr
(ZimmerNr, Tag) → GastNr

- INSERT-Anomalie
- DELETE-Anomalie
- UPDATE-Anomalie

- **Primär-Attribute:** alle Attribute einer Relation R, die in irgendeinem Schlüssel von R vorkommen. **Sekundär-Attribute**: alle übrigen Attribute von R.

Zweite Normalform

Eine Relation R befindet sich in 2.NF, wenn sie in 1.NF ist und jedes Sekundär-Attribut von R von jedem beliebigen Schlüssel von R voll funktional abhängig ist.

Folgerung

- besitzt eine Relation nur einen einzigen Schlüssel mit nur einem Attribut, so ist sie in 2.NF
- sind alle Attribute einer Relation Bestandteil der möglichen Schlüssel, so ist sie in 2.NF

Normalisierung 1.NF → 2.NF

Ist eine Attributkombination Y bezüglich einer Relation R mit der Attributkombination A nicht voll funktional abhängig von einem Schlüssel S, $S \cap Y = \varnothing$, so daß ein $S' \subset S$ mit $S' \rightarrow Y$ existiert, dann kann R als verlustfreier Join der Projektionen

$R1 = \prod_{S'Y}(R)$ und

$R2 = \prod_{Z}(R)$ mit $Z = A - Y$

dargestellt werden.

Beispiel

Belegung(GastNr, ZimmerNr, Tag, Name, Vorname, Anschrift)
(GastNr, Tag) → ZimmerNr
GastNr → (Name, Vorname, Anschrift)

$R1 = \prod_{GastNr,Name,Vorname,Anschrift}(Belegung)$

$R2 = \prod_{GastNr,ZimmerNr,Tag}(Belegung)$

Beispiel

Rechnung(<u>RNR</u>, GastNr, Name, Vorname, Anschrift, Betrag)

- INSERT-Anomalie
- DELETE-Anomalie
- UPDATE-Anomalie

Dritte Normalform

Eine Relation R befindet sich in 3.NF, wenn sie in 2.NF ist und kein Sekundär-Attribut von R von einem Schlüssel von R transitiv abhängig ist.

Normalisierung 2.NF → 3.NF

Ist eine Attributkombination Y bezüglich einer Relation R mit der Attributkombination A transitiv von einem Schlüssel S abhängig, so daß eine Attributkombination X mit $S \rightarrow X \ \wedge \ X \rightarrow Y$ und $X \cap Y = \varnothing$ existiert, dann kann R als verlustfreier Join der Projektionen

$R1 = \prod_{XY}(R)$ und

$R2 = \prod_{Z}(R)$ mit $Z = A - Y$

dargestellt werden.

Beispiel

Rechnung(<u>RNR</u>, GastNr, Name, Vorname, Anschrift, Betrag)

RNR → (GastNr, Betrag)

GastNr → (Name, Vorname, Anschrift)

$R1 = \prod_{GastNr,Name,Vorname,Anschrift}(Rechnung)$

$R2 = \prod_{RNR,GastNr,Betrag}(Rechnung)$

7. Datensicherheit

7.1 Sichten

Sicht (View)

Von einer oder mehreren realen oder virtuellen Datenbankrelationen (Basisrelationen/Sichten) abgeleitete virtuelle Relation

- Anwender erhalten ihren spezifischen Ausschnitt einer Datenbank.
- Komplexe Anfragen auf die Basisrelationen der Datenbank können durch Sichten in Teilen vordefiniert werden.
- Änderungen in den Basisrelationen und deren Beziehungen untereinander können dem Anwender verborgen bleiben.

Erzeugen einer Sicht

Syntax

view-definition ::=

CREATE VIEW table-name [(column-name [{ , column-name }...])

AS query-expression

[WITH CHECK OPTION]

- Erzeugt eine Sicht mit dem Namen table-name.
- query-expression: sichterzeugende Anfrage
- Enthält die CREATE-VIEW-Anweisung keine Liste von column-name, so besitzt die Sicht die gleichen Attribute wie die select-list der SELECT-Anweisung. Wird die Liste von column-names angegeben, so erhält die Sicht diese Attribute. select-list und Liste der column-names müssen in der Anzahl der Attribute übereinstimmen.
- Stellt eine Sicht S eine Auswahl von Tupeln einer einzigen Basisrelation R über eine Bedingung search-condition dar

```
CREATE VIEW S AS
    SELECT *
    FROM R
    WHERE search-condition
```

so können Manipulationsoperationen auf diese Sicht uneingeschränkt ausgeführt werden

- WITH CHECK OPTION: bewirkt, daß nur diejenigen Manipulationen auf die Sicht ausgeführt werden, die auch die search-condition der Sichtendefinition erfüllen.

Beispiel

```
CREATE VIEW Alte_Meister AS
    SELECT *
    FROM Bild
    WHERE Jahr < 1800
```

Beispiel

"Gesucht sind die ENR der Museen, die Werke Alter Meister besitzen"

```
SELECT ENR
FROM Museum
WHERE ENR IN
        (SELECT ENR
        FROM Alte_Meister)
```

Ergebnis: (E1, E4)

Beispiel

```
INSERT INTO Alte_Meister
VALUES ('B17', 'Turmbau zu Babel', 1520, 1200, 'K3', 'E4')

INSERT INTO Alte_Meister
VALUES ('B18', 'Neuer Turmbau zu Babel', 1945, 120, 'K5', 'E4')


CREATE VIEW Alte_Meister AS
SELECT *
FROM Bild
WHERE Jahr < 1800
WITH CHECK OPTION
```

Beispiel

```
UPDATE Bilderzahl
SET Anzahl = Anzahl+1
WHERE KNR = 'K1'
```

Beispiel
Bilder(<u>BNR</u>, Titel, Jahr, Wert, *KNR, ENR*)

```
CREATE VIEW Bilder AS
    SELECT *
    FROM Bild
    WHERE ENR IN
        (SELECT ENR
         FROM Museum
         UNION
         SELECT ENR
         FROM Galerie)
WITH CHECK OPTION;

INSERT INTO Bilder VALUES
('B17', 'Bildnis K.', 1911, 500, 'K7', 'E8');
```

Löschen einer Sicht

Syntax

drop-view-statement ::=

DROP VIEW table-name

7.2 Zugriffsschutz

Rechtevergabe

Syntax

grant-statement ::=

GRANT privileg [{ , privileg }...]

ON table-name

TO grantee [{ , grantee }...]

[WITH GRANT OPTION]

privileg ::=

ALL PRIVILEGES

| SELECT

| DELETE

| INSERT [(column-name [{ , column-name }...])]

| UPDATE [(column-name [{ , column-name }...])]

grantee ::=

PUBLIC

| authorization-identifier

- Mit der *GRANT*-Anweisung verleiht ein Rechtegeber (grantor) einem oder mehreren Rechtenehmern (*grantee*) bestimmte Zugriffsrechte (*privileg*) auf eine Relation.

- *PUBLIC*: Rechtegeber kann allen eingetragenen Nutzern der Datenbank diese Rechte verleihen.

- *ALL PRIVILEGES* verleiht er Rechtenehmern alle Rechte, die er selbst besitzt.

- *WITH GRANT OPTION: grantee* zusätzlich das Recht, die ihm in der GRANT-Anweisung vergebenen Zugriffsrechte auf die Relation *table-name* an andere Nutzer weiterzugeben.

- Zur Speicherung der Nutzerrechte 2 Systemrelationen:

TABLE_PRIVILEGES
(GRANTOR, GRANTEE, TABLE, PRIVILEGE, GRANTABLE),

COLUMN_PRIVILEGES
(GRANTOR, GRANTEE, TABLE, COLUMN, PRIVILEGE,GRANTABLE)

mit

GRANTOR:	Rechtegeber,
GRANTEE:	Rechtenehmer
TABLE:	Relation, für die das Recht vergeben wurden
COLUMN:	Attribut der Relation, für das das Rechtvergeben wurden
PRIVILEGE:	Zugriffsrecht
GRANTABLE:	Recht zur Weitergabe

Beispiel

- Anfangszustand von COLUMN_PRIVILEGES

GRANTOR	GRANTEE	TABLE	COLUMN	PRIVILEGE	GR
user_1	user_1	Maler	KNR	s, d, i, u	yes
user_1	user_1	Maler	Name	s, d, i, u	yes

Beispiel

„Nutzer user_1 soll alle seine Rechte an der Relation Maler an Nutzer user_2 weitergeben“

GRANT ALL PRIVILEGES
ON Maler
TO user_2

Ergebnis (Systemrelation COLUMN_PRIVILEGES)

GRANTOR	GRANTEE	TABLE	COLUMN	PRIVILEGE	GR
user_1	user_1	Maler	KNR	s, d, i, u	yes
user_1	user_1	Maler	Name	s, d, i, u	yes
user_1	user_2	Maler	KNR	s, d, i, u	no
user_1	user_2	Maler	Name	s, d, i, u	no

Beispiel

„user_2 und user_3 erhalten von user_1 das SELECT-, INSERT- und DELETE-Recht auf die Relation Maler mit dem Recht zur Weitergabe“

GRANT SELECT, INSERT, DELETE
ON Maler
TO user_2, user_3
WITH GRANT OPTION

Ergebnis (Systemrelation COLUMN_PRIVILEGES)

GRANTOR	GRANTEE	TABLE	COLUMN	PRIVILEGE	GR
user_1	user_1	Maler	KNR	s, d, i, u	yes
user_1	user_1	Maler	Name	s, d, i, u	yes
user_1	user_2	Maler	KNR	s, d, i	yes
user_1	user_2	Maler	KNR	u	no
user_1	user_2	Maler	Name	s, d, i	yes
user_1	user_2	Maler	Name	u	no
user_1	user_3	Maler	KNR	s, d, i	yes
user_1	user_3	Maler	Name	s, d, i	yes

Beispiel

„user_2 gibt das SELECT-Recht an user_4 mit GRANT OPTION weiter“

GRANT SELECT
ON Maler
TO user_4
WITH GRANT OPTION

und

„user_3 gibt das SELECT-Recht und das INSERT-Recht auf die KNR der Relation Maler an User_4 weiter“

GRANT SELECT, INSERT (KNRl)
ON Maler
TO user_4

Ergebnis (Systemrelation COLUMN_PRIVILEGES)

GRANTOR	GRANTEE	TABLE	COLUMN	PRIVILEGE	GR
user_1	user_1	Maler	KNR	s, d, i, u	yes
user_1	user_1	Maler	Name	s, d, i, u	yes
user_1	user_2	Maler	KNR	s, d, i	yes
user_1	user_2	Maler	KNR	u	no
user_1	user_2	Maler	Name	s, d, i	yes
user_1	user_2	Maler	Name	u	no
user_1	user_3	Maler	KNR	s, d, i	yes
user_1	user_3	Maler	Name	s, d, i	yes
user_2	user_4	Maler	KNR	s	yes
user_2	user_4	Maler	Name	s	yes
user_3	user_4	Maler	KNR	s, i	no
user_3	user_4	Maler	Name	s	no

Beispiel
„user_4 gibt das SELECT-Recht an user_5 weiter“

GRANT SELECT
ON Maler
TO user_5

Ergebnis (Systemrelation COLUMN_PRIVILEGES)

GRANTOR	GRANTEE	TABLE	COLUMN	PRIVILEGE	GR
user_1	user_1	Maler	KNR	s, d, i, u	yes
user_1	user_1	Maler	Name	s, d, i, u	yes
user_1	user_2	Maler	KNR	s, d, i	yes
user_1	user_2	Maler	KNR	u	no
user_1	user_2	Maler	Name	s, d, i	yes
user_1	user_2	Maler	Name	u	no
user_1	user_3	Maler	KNR	s, d, i	yes
user_1	user_3	Maler	Name	s, d, i	yes
user_2	user_4	Maler	KNR	s	yes
user_2	user_4	Maler	Name	s	yes
user_3	user_4	Maler	KNR	s, i	no
user_3	user_4	Maler	Name	s	no
user_4	user_5	Maler	KNR	s	no
user_4	user_5	Maler	Name	s	no

Rechteentzug

- indirekt vom grantor erhaltene Rechte:
  Rechte, die ein *grantee* indirekt von einem grantor erhalten hat

Syntax

revoke-statement ::=

REVOKE privileg [{ , privileg }...]

ON table-name

FROM grantee [{ , grantee }...]

- Nutzer (grantor) entzieht anderen Nutzern (*grantee*) bestimmte Rechte oder Teilrechte (*privileg*) auf eine Relation, die der grantor den Nutzern in einer oder in mehreren vorhergehenden GRANT-Anweisungen gegeben hat.

- Als Folge davon entzieht die *REVOKE*-Anweisung auch allen indirekten *grantees* des grantors bestimmte der indirekt von den grantor erhaltenen Rechte.

Beispiel

„user_1 entzieht user_2 das UPDATE-Recht“

REVOKE UPDATE
ON Maler
FROM user_2

Ergebnis (Systemrelation COLUMN_PRIVILEGES)

GRANTOR	GRANTEE	TABLE	COLUMN	PRIVILEGE	GR
user_1	user_1	Maler	KNR	s, d, i, u	yes
user_1	user_1	Maler	Name	s, d, i, u	yes
user_1	user_2	Maler	KNR	s, d, i	yes
user_1	user_2	Maler	Name	s, d, i	yes
user_1	user_3	Maler	KNR	s, d, i	yes
user_1	user_3	Maler	Name	s, d, i	yes
user_2	user_4	Maler	KNR	s	yes
user_2	user_4	Maler	Name	s	yes
user_3	user_4	Maler	KNR	s, i	no
user_3	user_4	Maler	Name	s	no
user_4	user_5	Maler	KNR	s	no
user_4	user_5	Maler	Name	s	no

Beispiel

„user_1 entzieht user_3 das INSERT-Recht“

REVOKE INSERT
ON Maler
FROM user _3

Ergebnis (Systemrelation COLUMN_PRIVILEGES)

GRANTOR	GRANTEE	TABLE	COLUMN	PRIVILEGE	GR
user_1	user_1	Maler	KNR	s, d, i, u	yes
user_1	user_1	Maler	Name	s, d, i, u	yes
user_1	user_2	Maler	KNR	s, d, i	yes
user_1	user_2	Maler	Name	s, d, i	yes
user_1	user_3	Maler	KNR	s, d	yes
user_1	user_3	Maler	Name	s, d	yes
user_2	user_4	Maler	KNR	s	yes
user_2	user_4	Maler	Name	s	yes
user_3	user_4	Maler	KNR	s	no
user_3	user_4	Maler	Name	s	no
user_4	user_5	Maler	KNR	s	no
user_4	user_5	Maler	Name	s	no

Beispiel

„user_3 entzieht user_4 das SELECT-Recht“

REVOKE SELECT
ON Maler
FROM user_4

Ergebnis (Systemrelation COLUMN_PRIVILEGES)

GRANTOR	GRANTEE	TABLE	COLUMN	PRIVILEGE	GR
user_1	user_1	Maler	KNR	s, d, i, u	yes
user_1	user_1	Maler	Name	s, d, i, u	yes
user_1	user_2	Maler	KNR	s, d, i	yes
user_1	user_2	Maler	Name	s, d, i	yes
user_1	user_3	Maler	KNR	s, d	yes
user_1	user_3	Maler	Name	s, d	yes
user_2	user_4	Maler	KNR	s	yes
user_2	user_4	Maler	Name	s	yes
user_4	user_5	Maler	KNR	s	no
user_4	user_5	Maler	Name	s	no

Beispiel

„user_1 entzieht user_2 alle (verbleibenden) Rechte“

```
REVOKE ALL PRIVILEGES
ON Maler
FROM user_2
```

Ergebnis (Systemrelation COLUMN_PRIVILEGES)

GRANTOR	GRANTEE	TABLE	COLUMN	PRIVILEGE	GR
user_1	user_1	Maler	KNR	s, d, i, u	yes
user_1	user_1	Maler	Name	s, d, i, u	yes
user_1	user_3	Maler	KNR	s, d	yes
user_1	user_3	Maler	Name	s, d	yes