

# **Kapitel 7: Die Datenbanksprache SQL**

## **7.1 Datendefinition**

## **7.2 Einfache Anfragen**

## **7.3 Semantik einfacher SQL-Anfragen**

## **7.4 Erweiterte Anfragen**

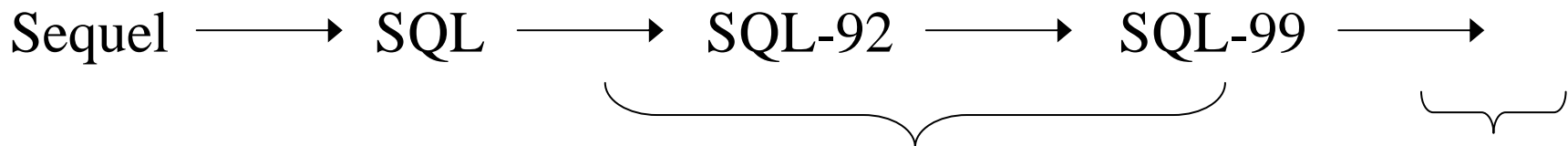
## **7.5 Datenmodifikation**

# SQL-Sprachumfang

SQL (Structured Query Language) umfasst:

- Interaktives („stand-alone“) SQL
- Eingebettetes („embedded“) SQL
- Anweisungen zur Integritätssicherung, Zugriffskontrolle
- Anweisung zum Tuning

Historie:



Produkte:

Oracle, DB2, Informix,  
Sybase, SQL Server,  
(MySQL), ...

# 5.1 Datendefinition

## “Grobsyntax”:

```
CREATE TABLE [user .] table ( column_element {, column_element ...}  
                                {, table_constraint ...} )
```

mit:

```
column_element ::=  
    column data_type [DEFAULT expr] [column_constraint]
```

```
column_constraint ::=  
    [NOT NULL] [PRIMARY KEY | UNIQUE]  
    [REFERENCES [user .] table [ ( column ) ] ]  
    [CHECK ( condition ) ]
```

```
table_constraint ::=  
    [ {PRIMARY KEY | UNIQUE} ( column {, column ...} ) ]  
    [ FOREIGN KEY ( column {, column ...} )  
      REFERENCES [user .] table [ ( column {, column ...} ) ]  
    [CHECK ( condition ) ]
```

**Semantik:** Anlegen von (leeren) Relationen und  
Festlegen von Integritätsbedingungen

# Exkurs: Syntaxbeschreibung – kontextfreie Grammatiken

**Kontextfreie Grammatik**  $G = (\Sigma, V, P, S)$  mit

- $\Sigma$ : Terminalsymbole (Literale)
- $V$ : Nonterminalsymbole (Variablen) (mit  $V \cap \Sigma = \emptyset$ )
- $P \subseteq V \times (V \cup \Sigma)^*$  : Produktionsregeln
- $S \in V$ : Startsymbol.

Die von  $G$  erzeugte Sprache  $L(G) \subseteq \Sigma^* =$

$\{w \in \Sigma^* \mid \text{es gibt endliche Ableitung}$

$S = x_0 \rightarrow x_1 \rightarrow \dots \rightarrow x_n = w \text{ mit } x_i \in (V \cup \Sigma)^*$

und  $x_{(i-1)} \rightarrow x_i$  als Anwendung einer Regel aus  $P$

# Exkurs: Syntaxbeschreibung – einfaches Beispiel

$V = \{\text{Address, Name, Firstname, Lastname, Street, City, Country, String, Letter, Number, Digit}\},$

$\Sigma = \{a, b, c, \dots, 0, 1, \dots\},$

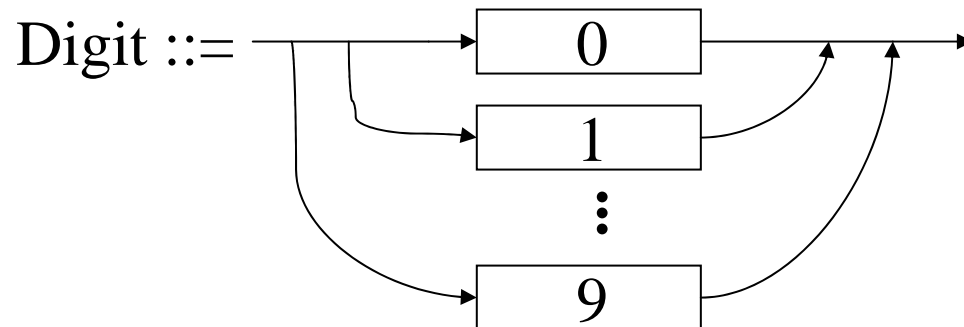
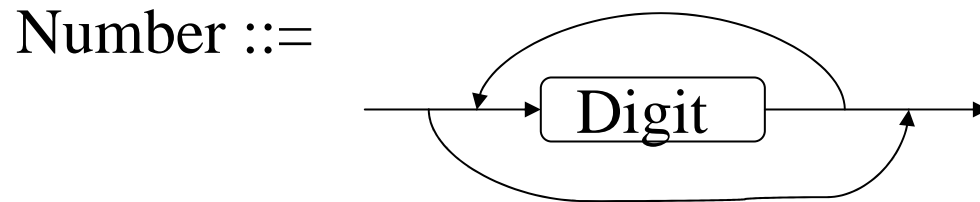
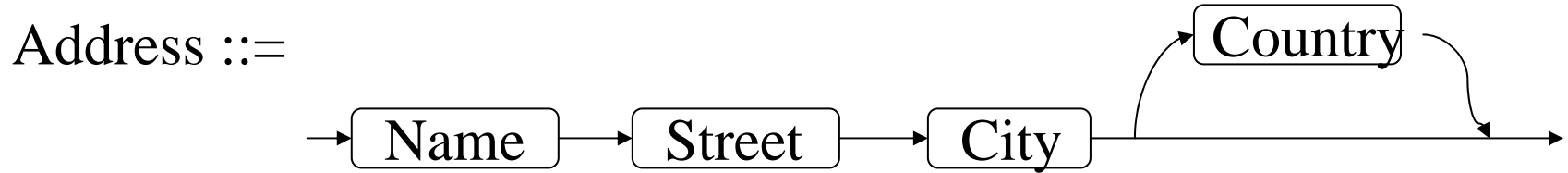
$S = \text{Address},$

$P = \{\text{Address} \rightarrow \text{Name Street City},$   
 $\text{Address} \rightarrow \text{Name Street City Country},$   
 $\text{Name} \rightarrow \text{Firstname Lastname},$   
 $\text{Name} \rightarrow \text{Letter Lastname},$   
 $\text{Firstname} \rightarrow \text{String},$   
 $\text{Lastname} \rightarrow \text{String},$   
 $\text{Street} \rightarrow \text{String Number},$   
 $\text{Country} \rightarrow \text{String},$   
 $\text{String} \rightarrow \text{Letter},$   
 $\text{String} \rightarrow \text{Letter String},$   
 $\text{Number} \rightarrow \text{Digit Number},$   
 $\text{Number} \rightarrow \varepsilon,$   
 $\text{Letter} \rightarrow a, \quad \text{Letter} \rightarrow b, \quad \dots,$   
 $\text{Digit} \rightarrow 0, \quad \text{Digit} \rightarrow 1, \quad \dots\}$

# Exkurs: Syntaxbeschreibung – EBNF

Address ::= Name Street City [Country],  
String ::= Letter {Letter ...},  
Number ::= {Digit ...},  
Digit ::= 0|1|2|3|4|5|6|7|8|9,  
usw.

# Exkurs: Syntaxbeschreibung – Syntaxdiagramme



# CREATE TABLE: Beispiele (1)

```
CREATE TABLE Kunden
```

```
(KNr INTEGER CHECK (KNr>0) PRIMARY KEY,  
Name VARCHAR(30),  
Stadt VARCHAR(30),  
Saldo FLOAT, /* oder besser: Saldo MONEY, */  
Rabatt FLOAT CHECK (Rabatt >= 0.0) )
```

```
CREATE TABLE Produkte
```

```
(PNr INTEGER CHECK (PNr>0) PRIMARY KEY,  
Bez VARCHAR(30) NOT NULL UNIQUE,  
Gewicht FLOAT CHECK (Gewicht > 0.0),  
Preis FLOAT CHECK (Preis > 0.0),  
Lagerort VARCHAR(30),  
Vorrat INTEGER CHECK (Vorrat >= 0) )
```



# CREATE TABLE: Beispiele (2)

```
CREATE TABLE Bestellungen (  
    BestNr INTEGER CHECK (BestNr > 0) PRIMARY KEY,  
    Monat INTEGER NOT NULL  
        CHECK (Monat BETWEEN 1 AND 12),  
    Tag INTEGER NOT NULL CHECK (Tag BETWEEN 1 AND 31),  
        /* oder besser: Datum DATE, */  
    KNr INTEGER CHECK (KNr > 0),  
    PNr INTEGER CHECK (PNr > 0),  
    Menge INTEGER CHECK (Menge > 0),  
    Summe FLOAT,  
    Status VARCHAR(9) CHECK (Status IN ('neu', 'geliefert', 'bezahlt')),  
    FOREIGN KEY (PNr) REFERENCES Produkte (PNr),  
    FOREIGN KEY (KNr) REFERENCES Kunden (KNr),  
    UNIQUE (Monat, Tag, PNr, KNr) )
```

## 5.2 Einfache Anfragen

### “Grobsyntax”:

```
select_block { { UNION | INTERSECT | EXCEPT } [ALL]  
              select_block ...}  
[ORDER BY result_column [ASC | DESC]  
          {, result_column [ASC | DESC] ...}]
```

```
mit select_block ::=  
SELECT [ALL | DISTINCT]  
      {column | {expression [AS result_column]}}  
      {, {column | {expression [AS result_column]}}} ...}  
FROM table [correlation_var] {, table [correlation_var] ...}  
[WHERE search_condition]  
[GROUP BY column {, column ...} [HAVING search_condition] ]
```

# Abbildung auf TRK und RA

## “Grobsemantik”:

SELECT A, B, ... FROM R, S, ..., T, ... WHERE F

(so daß A, B, ... zu R, S, ... gehören, nicht aber zu T, ...,  
und F über R, S, ..., T, ... definiert ist)

→ RA:  $\pi[A, B, \dots] ( \sigma [F] ( R \times S \times \dots \times T \times \dots ) )$

→ TRK:  $\{x.A, y.B, \dots \mid x \in R \wedge y \in S \wedge \dots \wedge \exists z: z \in T \dots$   
 $\wedge F(x, y, \dots, z, \dots)\}$

# Einfache SQL-Anfragen: Beispiele (1)

1) Finden Sie (die Namen) alle(r) Kunden mit negativem Saldo.

→ `SELECT KNr, Name, Stadt, Saldo, Rabatt FROM Kunden  
WHERE Saldo < 0.0`

oder: `SELECT * FROM Kunden WHERE Saldo < 0.0`

bzw.: `SELECT Name FROM Kunden WHERE Saldo < 0.0`

2) Namen von Kunden mit unbezahlter Bestellung vor Anfang Oktober

→ `SELECT Name FROM Kunden, Bestellungen  
WHERE Monat < 10 AND Status <> 'bezahlt'  
AND Kunden.KNr = Bestellung.KNr`

3) Namen der Homburger Kunden, die seit Anfang Sept. ein Produkt aus Homburg bekommen haben, jeweils mit der Bezeichnung des Produkts

→ `SELECT Name, Bez FROM Kunden, Bestellungen, Produkte  
WHERE Stadt='Homburg'  
AND Monat>=9 AND Status<>'neu'  
AND Lagerort='Homburg'  
AND Kunden.KNr=Bestellungen.KNr  
AND Bestellungen.PNr=Produkt.PNr`

# Einfache SQL-Anfragen: Beispiele (2)

- 4) Finden Sie die Rechnungssumme der Bestellung mit BestNr 111  
(ohne auf das Attribut Summe der Relation Bestellungen zuzugreifen).  
→ `SELECT Menge*Preis*(1.0-Rabatt) AS Rechnungssumme  
FROM Bestellungen, Produkte, Kunden  
WHERE BestNr=111  
AND Bestellungen.PNr=Produkte.PNr  
AND Bestellungen.KNr=Kunden.KNr`

# Allgemeinere Form der FROM-Klausel

## Korrelationsvariablen (Tupelvariablen):

Finde alle Paare von Kunden, die in derselben Stadt wohnen.

→ SELECT K1.Name, K2.Name FROM Kunden K1, Kunden K2  
WHERE K1.Stadt=K2.Stadt AND K1.KNr < K2.KNr

## Outer Joins:

Kunden mit 5 % Rabatt zusammen mit ihren Bestellungen und den entsprechenden Produkten, inkl. Kunden ohne Bestellungen

→ SELECT \*  
FROM Kunden FULL OUTER JOIN Bestellungen  
ON (Kunden.KNr=Bestellungen.KNr),  
Produkte  
WHERE Rabatt = 0.05  
AND Produkte.PNr=Bestellungen.PNr

# Allgemeinere Form der WHERE-Klausel (1)

## **Bereichsanfragen:**

Kunden, deren Rabatt zwischen 10 und 20 Prozent liegt.

→ SELECT \* FROM Kunden  
WHERE Rabatt BETWEEN 0.10 AND 0.20

## **String-Pattern-Matching:**

Kunden, deren Name mit A beginnt.

→ SELECT \* FROM Kunden WHERE Name LIKE 'A%'

Kunden mit Namen Meier, Maier, Meyer, ...

→ SELECT \* FROM Kunden WHERE Name LIKE 'M\_\_er'

## **Test auf Nullwert:**

Produkte, die grundsätzlich in keinem Lager geführt werden.

→ SELECT \* FROM Produkte WHERE Lagerort IS NULL

# Allgemeinere Form der WHERE-Klausel (2): Subqueries

## Test auf Mitgliedschaft in Menge:

Finden Sie alle Kunden aus Homburg, Merzig und Saarlouis.

```
→ SELECT * FROM Kunden  
   WHERE Stadt IN ('Homburg', 'Merzig', 'Saarlouis')
```

## mit Subqueries:

Namen von Kunden mit unbezahlter Bestellung vor Anfang Oktober

```
→ SELECT Name FROM Kunden  
   WHERE KNr IN ( SELECT KNr FROM Bestellungen  
                  WHERE Status <> 'bezahlt' AND Monat < 10 )
```

## mit korrelierten Subqueries:

Kunden aus Städten, in denen es mindestens zwei Kunden gibt.

```
→ SELECT * FROM Kunden K1  
   WHERE Stadt IN (SELECT Stadt FROM Kunden K2  
                  WHERE K2.KNr <> K1.KNr )
```



# Allgemeinere Form der WHERE-Klausel (3): “quantifizierte” Subqueries

- Die Bedingung *Wert*  $\theta$  *ANY Menge* mit  $\theta \in \{=, \neq, <, >, \leq, \geq\}$  ist erfüllt, wenn es in der Menge ein Element gibt, für das *Wert*  $\theta$  *Element* gilt.  
(=ANY ist äquivalent zu IN.)
- Die Bedingung *Wert*  $\theta$  *ALL Menge* mit  $\theta \in \{=, \neq, <, >, \leq, \geq\}$  ist erfüllt, wenn für alle Elemente der Menge gilt: *Wert*  $\theta$  *Element*.  
( $<>$ ALL ist äquivalent zu NOT IN.)
- Die Bedingung *EXISTS Menge* ist erfüllt, wenn die Menge nicht leer ist  
(dies ist äquivalent zur Bedingung  $0 < \text{SELECT COUNT(*) FROM ...}$ )

Beispiele:

Finden Sie die Kunden mit dem geringsten Rabatt.

→ `SELECT * FROM Kunden`  
`WHERE Rabatt <= ALL ( SELECT Rabatt FROM Kunden )`

Finden Sie die Kunden, für die keine Bestellung registriert ist.

→ `SELECT * FROM Kunden`  
`WHERE NOT EXISTS (SELECT * FROM Bestellungen`  
`WHERE Bestellungen.KNr = Kunden.KNr )`

# Allgemeinere Form der WHERE-Klausel (4): Simulation allquantifizierter Suchprädikate

Finden Sie die Kunden, die alle überhaupt lieferbaren Produkte irgendwann bestellt haben:

```
→ SELECT * FROM Kunden
   WHERE NOT EXISTS
       ( SELECT * FROM Produkte
         WHERE NOT EXISTS
             ( SELECT * FROM Bestellungen
               WHERE Bestellungen.PNr = Produkte.PNr
                 AND Bestellungen.KNr = Kunden.KNr ) )
```

## 5.3 Präzise Semantik einfacher SQL-Anfragen: Abbildung auf TRK

Voraussetzungen:

- Vernachlässigung von Multimengen, Nullwerten u.ä.
- Eindeutige Benennung von Tupelvariablen und Zuordnung von Attr.

Beispiel: SELECT A FROM REL

WHERE EXISTS (SELECT B FROM REL WHERE B>0)

umbenennen in

SELECT R1.A FROM REL R1 WHERE EXISTS

(SELECT R2.B FROM REL R2 WHERE R2.B>0)

Definition einer Abbildungsfunktion

*sql2trc: sql query  $\rightarrow$  trc query*

von select\_block-Konstrukten auf TRK-Anfragen

unter Verwendung der Funktion

*sql2trc': sql where clause  $\rightarrow$  trc formula*

von search\_condition-Konstrukten auf TRK-Formeln.

# Abbildung auf TRK (1)

$\text{sql2trc} [\text{SELECT } A_1, A_2, \dots \text{ FROM REL1 } R_1, \text{REL2 } R_2, \dots, \text{RELm } R_m,$   
 $\text{TAB1 } T_1, \dots, \text{TABk } T_k \text{ WHERE } F]$   
 (so daß  $A_1, A_2, \dots, A_n$  zu  $\text{REL1}, \text{REL2}, \dots, \text{RELm}$  gehören,  
 nicht aber zu  $\text{TAB1}, \dots, \text{TABk}$   
 und  $F$  über  $\text{REL1}, \dots, \text{RELm}, \text{TAB1}, \dots, \text{TABk}$  definiert ist)  
 $= \{ r_{i_1}.A_1, r_{i_2}.A_2, \dots \mid r_1 \in \text{REL1} \wedge r_2 \in \text{REL2} \wedge \dots \wedge r_m \in \text{RELm}$   
 $\wedge \exists t_1 \dots \exists t_k (t_1 \in \text{TAB1} \wedge \dots \wedge t_k \in \text{TABk} \wedge \text{sql2trc}'[F]) \}$

$\text{sql2trc} [\text{select-block1 UNION select-block2}]$   
 (mit select-block1:  $\text{SELECT } A_1, A_2, \dots \text{ FROM REL1 } R_1, \dots, \text{RELm } R_m,$   
 $\text{TAB1 } T_1, \dots, \text{TABk } T_k \text{ WHERE } F$   
 und select-block2:  $\text{SELECT } B_1, B_2, \dots \text{ FROM SET1 } S_1, \dots, \text{SETm}' S_m',$   
 $\text{PAR1 } P_1, \dots, \text{PARk}' P_k' \text{ WHERE } G$ )  
 $= \{ u_1, u_2, \dots \mid$   
 $(\exists r_1 \dots \exists r_m (u_1 = r_1.A_1 \wedge u_2 = r_2.A_2 \wedge \dots \wedge$   
 $r_1 \in \text{REL1} \wedge r_2 \in \text{REL2} \wedge \dots \wedge r_m \in \text{RELm} \wedge$   
 $\exists t_1 \dots \exists t_k (t_1 \in \text{TAB1} \wedge \dots \wedge t_k \in \text{TABk} \wedge \text{sql2trc}'[F]))$   
 $\vee (\exists s_1 \dots \exists s_m' (u_1 = s_1.B_1 \wedge u_2 = s_2.B_2 \wedge \dots \wedge$   
 $s_1 \in \text{SET1} \wedge s_2 \in \text{SET2} \wedge \dots \wedge s_m' \in \text{SETm}' \wedge$   
 $\exists p_1 \dots \exists p_k' (p_1 \in \text{PAR1} \wedge \dots \wedge p_k' \in \text{PARk}' \wedge \text{sql2trc}'[G])) \}$

# Abbildung auf TRK (2)

$\text{sql2trc}' [ Ri.Aj \theta Tk.B1 ] = ri.Aj \theta tk.B1$

$\text{sql2trc}' [ Ri.Aj \theta c ]$  (mit einer Konstanten  $c$ )  $= ri.Aj \theta c$

$\text{sql2trc}' [ F \text{ AND } G ] = \text{sql2trc}'[F] \wedge \text{sql2trc}'[G]$

$\text{sql2trc}' [ F \text{ OR } G ] = \text{sql2trc}'[F] \vee \text{sql2trc}'[G]$

$\text{sql2trc}' [ \text{NOT } F ] = \neg \text{sql2trc}'[F]$

$\text{sql2trc}' [ Ri.Aj \text{ IN subquery} ]$

(so daß subquery die Form  $\text{SELECT } Qk.C$

$\text{FROM QUELL1 } Q1, \dots, \text{QUELLm'} Qm' \text{ WHERE } H \text{ hat})$

$= \exists q1 \dots \exists qm' (qk.C = ri.Aj \wedge q1 \in \text{QUELL1} \wedge \dots qm' \in \text{QUELLm'} \\ \wedge \text{sql2trc}'[H] )$

# Abbildung auf TRK (3)

sql2trc' [Ri.Aj  $\theta$ ANY subquery]

$$= \exists q_k (q_k \in \text{QUELL}_k \wedge (\exists q_1 \dots \exists q_{(k-1)} \exists q_{(k+1)} \dots \exists q_{m'} \\ (r_i.A_j \theta q_k.C \wedge q_1 \in \text{QUELL}_1 \wedge \dots \wedge q_{(k-1)} \in \text{QUELL}_{(k-1)} \wedge \\ q_{(k+1)} \in \text{QUELL}_{(k+1)} \wedge \dots \wedge q_{m'} \in \text{QUELL}_{m'} \wedge \text{sql2trc}'[H])))$$

sql2trc' [Ri.Aj  $\theta$ ALL subquery]

$$= \forall q_k ( ( q_k \in \text{QUELL}_k \wedge (\exists q_1 \dots \exists q_{(k-1)} \exists q_{(k+1)} \dots \exists q_{m'} \\ (q_1 \in \text{QUELL}_1 \wedge \dots \wedge q_{(k-1)} \in \text{QUELL}_{(k-1)} \wedge \\ q_{(k+1)} \in \text{QUELL}_{(k+1)} \wedge \dots \wedge q_{m'} \in \text{QUELL}_{m'} \wedge \text{sql2trc}'[H] ) ) ) \\ \Rightarrow ( r_i.A_j \theta q_k.C ) )$$

sql2trc' [EXISTS subquery]

(so daß subquery die Form SELECT C1, C2, ...

FROM QUELL1 Q1, ..., QUELL<sub>m'</sub> Q<sub>m'</sub> WHERE H hat)

$$= \exists q_1 \dots \exists q_{m'} (q_1 \in \text{QUELL}_1 \wedge \dots \wedge q_{m'} \in \text{QUELL}_{m'} \\ \wedge \text{sql2trc}'[H] )$$

# Abbildung auf TRK: Beispiel

query = SELECT K.KNr, K.Name FROM Kunden K  
WHERE K.Ort='Saarbrücken'  
AND NOT EXISTS  
(SELECT P.PNr FROM Produkte P, Bestellungen B  
WHERE P.Preis > 100 AND P.PNr=B.PNr AND B.KNr=K.KNr)

sql2trc [query]  
= {k.KNr, k.Name | k ∈ Kunden ∧ sql2trc'[K.Ort=... AND NOT EXISTS ...]}  
= {k.KNr, k.Name | k ∈ Kunden ∧ k.Ort=... ∧ ¬ sql2trc'[EXISTS ...]}  
= {k.KNr, k.Name | k ∈ Kunden ∧ k.Ort=... ∧  
¬ (∃ p ∃ b ( p ∈ Produkte ∧ b ∈ Bestellungen  
∧ sql2trc'[P.Preis>... AND ... AND ...] )) }  
= {k.KNr, k.Name | k ∈ Kunden ∧ k.Ort=... ∧  
¬ (∃ p ∃ b ( p ∈ Produkte ∧ b ∈ Bestellungen  
∧ p.Preis>... ∧ p.PNr=b.PNr ∧ b.KNr=k.KNr )) }

# Präzise Semantik einfacher SQL-Anfragen: Abbildung auf RA

Voraussetzungen:

- Vernachlässigung von Multimengen, Nullwerten u.ä.
- Eindeutige Benennung von Tupelvariablen und Zuordnung von Attr.

Definition einer Abbildungsfunktion

***sql2ra: sql query  $\rightarrow$  ra query***

von select\_block-Konstrukten auf RA-Anfragen  
unter Verwendung der Funktion

***sql2ra': sql where clause  $\times$  ra query  $\rightarrow$  ra query***

von search\_condition-Konstrukten auf RA-Ausdrücke  
sowie der Hilfsfunktion

***sql2ra-: sql where clause  $\times$  ra query  $\rightarrow$  ra query***

mit sql2ra-  $[F](E) = E - \pi[\text{sch}(E)] (\text{sql2ra}'[F](E))$

Erweiterung auf Multirelationen relativ leicht möglich.



# Abbildung auf RA (1)

sql2ra [SELECT A1, A2, ... FROM REL1 R1, REL2 R2, ..., RELm Rm,  
TAB1 T1, ..., TABk Tk WHERE F ]

(so daß A1, A2, ..., An zu REL1, REL2, ..., RELm gehören,  
nicht aber zu TAB1, ..., TABk  
und F über REL1, ..., RELm, TAB1, ..., TABk definiert ist)

= R1 := REL1; ...; Rm := RELm; T1 := TAB1; ...; Tk := TABk;  
 $\pi[R_{i_1}.A1, R_{i_2}.A2, \dots] ( \text{sql2ra}'[F](R1 \times \dots \times Rm \times T1 \times \dots \times Tk) )$

sql2ra [ select-block1 UNION select-block2 ]

(mit select-block1: SELECT A1, ... FROM REL1 R1, ..., RELm Rm,  
TAB1 T1, ..., TABk Tk WHERE F  
und select-block2: SELECT B1, ... FROM SET1 S1, ..., SETm' Sm',  
PAR1 P1, ..., PARK' Pk' WHERE G)

= sql2ra[select-block1]  $\cup$  sql2ra[select-block2]  
(mit ggf. notwendigen Umbenennungen von Attributen)

# Abbildung auf RA (2)

$\text{sql2ra}' [ Ri.Aj \theta Tk.B1 ] (E) = \sigma[Ri.Aj \theta Tk.B1](E)$   
 $\text{sql2ra}' [ Ri.Aj \theta c ] (E) \text{ (mit einer Konstanten } c) = \sigma[Ri.Aj \theta c](E)$   
 $\text{sql2ra}' [ F \text{ AND } G ] (E) = \text{sql2ra}'[F](E) \cap \text{sql2ra}'[G](E)$   
 $\text{sql2ra}' [ F \text{ OR } G ] (E) = \text{sql2ra}'[F](E) \cup \text{sql2ra}'[G](E)$   
 $\text{sql2ra}' [ \text{NOT } F ] (E) = \text{sql2ra}'[F](E) = E - \pi[\text{sch}(E)] (\text{sql2ra}'[F](E) )$

$\text{sql2ra}' [ Ri.Aj \text{ IN subquery} ] (E)$   
(so daß subquery die Form  $\text{SELECT } Qk.C$   
 $\text{FROM QUELL1 } Q1, \dots, \text{QUELLm' } Qm' \text{ WHERE } H \text{ hat}$ )  
 $= Q1 := \text{QUELL1}; \dots Qm' := \text{QUELLm'};$   
 $\pi[\text{sch}(E)] ( \text{sql2ra}'[H] ( \sigma[Ri.Aj = Qk.C] ( E \times Q1 \times \dots \times Qm' ) ) )$

# Abbildung auf RA (3)

sql2ra' [Ri.Aj  $\theta$  ANY subquery]  
= Q1 := QUELL1; ... Qm' := QUELLm';  
 $\pi[\text{sch}(E)] ( \text{sql2ra}'[H] ( \sigma[Ri.Aj \theta Qk.C] ( E \times Q1 \times \dots \times Qm' ) ) )$

sql2ra' [Ri.Aj  $\theta$  ALL subquery] (E)  
= sql2ra- [Ri.Aj  $\theta'$  ANY subquery](E)  
mit  $\theta'$  gleich  $\neq$  für  $\theta$  gleich  $=$ ,  $=$  für  $\neq$ ,  $<$  für  $\geq$ ,  $>$  für  $\leq$ , usw.  
=  $E - \pi[\text{sch}(E)] ( \text{sql2ra}'[H] ( \sigma[Ri.Aj \theta' Qk.C] ( E \times Q1 \times \dots \times Qm' ) ) )$

sql2ra' [EXISTS subquery]  
(so daß subquery die Form SELECT C1, C2, ...  
FROM QUELL1 Q1, ..., QUELLm' Qm' WHERE H hat)  
= Q1 := QUELL1; ... Qm' := QUELLm';  
 $\pi[\text{sch}(E)] ( \text{sql2ra}'[H] ( E \times Q1 \times \dots \times Qm' ) )$

# Abbildung auf RA: Beispiel

query = SELECT K.KNr, K.Name FROM Kunden K  
WHERE K.Ort='Saarbrücken'  
AND NOT EXISTS  
(SELECT P.PNr FROM Produkte P, Bestellungen B  
WHERE P.Preis > 100 AND P.PNr=B.PNr AND B.KNr=K.KNr)

sql2ra [query]

= K := Kunden; B := Bestellungen; P := Produkte;

$\pi[K.KNr, K.Name] (\text{sql2ra}'[K.Ort=\dots \text{ AND NOT EXISTS } \dots] (K))$

= ...  $\pi[K.KNr, K.Name] (\sigma[Ort=\dots](K) \cap \text{sql2ra}'[NOT EXISTS \dots](K))$

= ...  $\pi[K.KNr, K.Name] (\sigma[Ort=\dots](K) \cap (K - \pi[\text{sch}(E)](\text{sql2ra}'[EXISTS \dots](K))))$

= ...  $\pi[K.KNr, K.Name] (\sigma[Ort=\dots](K) \cap (K - \pi[\text{sch}(K)](\text{sql2ra}'[P.Preis\dots\text{AND}\dots\text{AND}\dots](K \times P \times B))))$

= ...  $\pi[K.KNr, K.Name] (\sigma[Ort=\dots](K) \cap (K - \pi[\text{sch}(K)](\sigma[P.Preis>100](K \times P \times B) \cap \sigma[P.PNr=B.PNr](K \times P \times B) \cap \sigma[P.PNr=B.PNr](K \times P \times B))))$

## 5.4 Erweiterte SQL-Anfragen: Aggregationsfunktionen

"Grobsyntax":

{ MAX | MIN | AVG | SUM | COUNT }  
( { ALL | DISTINCT } {column | expression | \*} )

„Grobsemantik“:

Abbildung einer Menge skalarer Werte auf einen skalaren Wert

# Aggregationsfunktionen: Beispiele

- 1) Finden Sie den höchsten (durchschnittlichen) Rabatt aller Kunden.  
→ `SELECT MAX (Rabatt) FROM Kunden`  
→ `SELECT AVG (Rabatt) FROM Kunden`
- 2) An wievielen Lagerorten werden Produkte gelagert?  
→ `SELECT COUNT (DISTINCT Lagerort) FROM Produkte`
- 3) Wieviele Kunden haben einen Rabatt von mehr als 15 Prozent?  
→ `SELECT COUNT (*) FROM Kunden WHERE Rabatt > 0.15`
- 4) Welche Kunden haben einen überdurchschnittlichen Rabatt?  
→ `SELECT * FROM Kunden`  
`WHERE Rabatt > SELECT AVG (Rabatt) FROM Kunden`
- 5) Wie hoch ist der Gesamtumsatz?  
→ `SELECT SUM (Menge*Preis*(1.0-Rabatt))`  
`FROM Bestellungen, Produkte, Kunden`  
`WHERE Bestellungen.PNr=Produkte.PNr`  
`AND Bestellungen.KNr=Kunden.KNr`

# Built-in-Funktionen auf skalaren Werten

Häufig produktspezifisch, z.B.

- Stringmanipulation in Oracle  
SELECT SUBSTR (Name, INSTR(Name, ' ')+1) FROM Kunden
- Umwandlung eines Datums (Datentyp DATE) in einen String  
SELECT TO\_CHAR(SYSDATE, 'DY DD MONTH YYYY, HH24:MI:SS')

usw. usw.

# Oracle-spezifische Funktionen zur Textinhaltssuche

mit Oracle8i interMedia

mit Ranking von Suchresultaten:

```
SELECT PNr, Bez FROM Produkte WHERE  
CONTAINS (Beschreibung, 'hard disk ', 1) > 0 ORDER BY Score(1) DESC  
oder: ... CONTAINS (Beschreibung, 'NEAR(hard, disk, 10) ', 1) > 0 ...
```

mit Reduktion auf Wortstämme:

```
SELECT PNr, Bez FROM Produkte WHERE  
CONTAINS (Beschreibung, '$disk', 1) >= 5 ORDER BY Score(1) DESC
```

inkl. Thesaurus zur Berücksichtigung  
von Synonymen (SYN) und Unterbegriffen (NT):

```
SELECT PNr, Bez FROM Produkte  
WHERE CONTAINS (Beschreibung, 'SYN(disk)', 1) >= 0  
AND CONTAINS (Beschreibung, 'NT(optics)', 2) >= 0  
ORDER BY Score(1)*Score(2) DESC
```



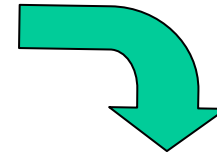
# Aggregation mit Gruppierung

Beispiel: Gesamtverkaufszahl von Produkten (ab Anfang September)

→ SELECT PNr, SUM(Menge) FROM Bestellungen  
WHERE Monat >= 9 GROUP BY PNr

**Zwischenresultat:**

PNr	Gruppe		
	BestNr	...	Menge
1	9		100
2	3		4
3	4		1
4	5		10
5	6		50
	10		50
	11		50
6	7		2
7	8		5
	12		10



**Endresultat:**

PNr	SUM(Menge)
1	100
2	4
3	1
4	10
5	150
6	2
7	15

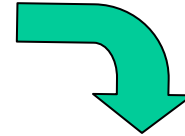
# Aggregation mit Gruppierung und Gruppenauswahl

Kunden mit mindestens zwei Bestellungen seit Anfang Oktober, deren Gesamtwert mindestens 2000 DM betragen hat.

→ `SELECT KNr FROM Bestellungen WHERE Monat >= 10  
GROUP BY KNr HAVING COUNT(*) >= 2 AND SUM(Summe) >= 2000`

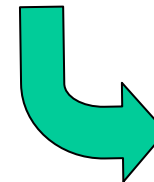
## Zwischenresultat 1:

KNr	Gruppe		
	BestNr	...	Summe
1	6		900.00
	7		180.00
	8		900.00
2	9		1600.00
	10		800.00
3	7		1800.00



## Zwischenresultat 2:

KNr	COUNT(*)	SUM (Summe)
1	3	1980.00
2	2	2400.00
3	1	1800.00



## Endresultat:

KNr
2

# Beispiele für GROUP BY ... HAVING ...

1) Gesamtverkaufszahl von Produkten, die ab Anfang September mehr als 1000-mal verkauft worden sind

→ SELECT PNr, SUM(Menge) FROM Bestellungen  
WHERE Monat >= 9  
GROUP BY PNr HAVING SUM(Menge) > 1000

2) Durchschnittsrabatt für Städte mit mehr als 10 Kunden

→ SELECT Stadt, AVG(Rabatt) AS MittlRabatt FROM Kunden  
GROUP BY Stadt HAVING COUNT(\*) > 10  
ORDER BY 2 DESC bzw. ORDER BY MittlRabatt DESC

3) Kunden, die alle überhaupt lieferbaren Produkte bestellt haben

→ SELECT KNr FROM Bestellungen  
GROUP BY KNr  
HAVING COUNT (DISTINCT PNr) =  
( SELECT COUNT(\*) FROM Produkte )

# Semantik der Gruppierung

Abbildung auf erweiterte RA (wobei  $A' \subseteq A$  gelten muß):

$$\begin{aligned} & \text{sql2ra} [\text{SELECT } A', f(B) \text{ FROM } \dots \text{ WHERE } \dots \text{ GROUP BY } A] \\ &= R(A, F) := \gamma_+ [A, B, f] (\text{sql2ra} [\text{SELECT } A, B \text{ FROM } \dots \text{ WHERE } \dots]); \\ & \pi_+ [A', F] (R) \end{aligned}$$

Abbildung von

SELECT A', f(B) FROM ... WHERE ...  
GROUP BY A HAVING cond(A, g(C))

auf RA-Programm:

$Q(A, B, C) := \text{sql2ra} [\text{SELECT } \dots \text{ FROM } \dots \text{ WHERE } \dots]; R(A, F) := \emptyset;$

for each  $x \in \pi [A](Q)$  do

$G_x := \pi_+ [A, B, C] (\sigma_+ [A=x] (Q))$

if  $\text{sql2ra}'[\text{cond}](G_x) \neq \emptyset$  then  $y := f(\pi_+ [B](G_x)); R := R \cup_+ \{(x, y)\}$  fi;

od;

$\pi_+ [A', F] (R)$

# Rekursive Anfragen

für transitive Hüllen u.ä.

Beispiel SQL-99-Standard:

```
WITH RECURSIVE Verbindungen (Start, Ziel, Gesamtdistanz) AS
  ( ( SELECT Abflugort, Zielort, Distanz
      FROM Flüge WHERE Abflugort = 'Frankfurt' )
    UNION
    ( SELECT V.Start, F. Ziel, V.Gesamtdistanz + F.Distanz
      FROM Verbindungen V, Flüge F WHERE V.Ziel = F.Abflugort ))
SELECT Ziel, Gesamtdistanz FROM Verbindungen
```

Beispiel Oracle:

```
SELECT F.Zielort FROM Flüge F
START WITH Abflugort = 'Frankfurt'
CONNECT BY Abflugort = PRIOR Zielort
AND PRIOR Ankunftszeit < Abflugzeit - 0.05
```

# 5.5 Datenmodifikation (1)

## Einfügen von Tupeln:

"Grobsyntax":

```
INSERT INTO table [ ( column {, column ...} ) ]  
{ VALUES ( expression {, expression ...} ) | subselect }
```

## Beispiele:

- 1) INSERT INTO Kunden VALUES (7, 'Kunz', 'Neunkirchen', 0.0, 0.0)
- 2) INSERT INTO Kunden (KNr, Name, Stadt)  
VALUES (7, 'Kunz', 'Neunkirchen')
- 3) CREATE TABLE Mahnungen (BestNr ...)  
mit demselben Schema wie Bestellungen  
INSERT INTO Mahnungen  
(SELECT \* FROM Bestellungen  
WHERE Status='geliefert' AND Monat<10)

# Datenmodifikation (2)

## Ändern von Tupeln:

"Grobsyntax":

UPDATE table [correlation\_var]

SET column = expression {, column = expression ...}

WHERE search\_condition

Beispiele:

1) UPDATE Kunden SET Stadt = 'Saarbrücken' WHERE KNr=1

2) UPDATE Kunden SET Rabatt = Rabatt + 0.05

WHERE Saldo > -10000.0

AND KNr IN (SELECT KNr FROM Bestellungen

GROUP BY KNr HAVING SUM(Summe) > 100000.0)

# Datenmodifikation (3)

## Löschen von Tupeln:

"Grobsyntax":

```
DELETE FROM table [correlation_var] [WHERE search_condition]
```

Beispiel: DELETE FROM Bestellungen WHERE Monat < 7

## Schemaänderung:

"Grobsyntax":

```
ALTER TABLE table
```

```
ADD column datatype [column_constraint]  
    {, column data_type [column_constraint] ...}
```

Beispiel:

```
ALTER TABLE Bestellungen ADD Frist INTEGER CHECK (Frist > 0)
```

## Transaktionen:

geklammerte Folgen von SQL-Anweisungen zu einer atomaren Einheit mit Abschluß COMMIT WORK oder ROLLBACK WORK



# Varianten der “Alleskäufer”-Query

- 1) SELECT KNr FROM Kunden  
WHERE ( SELECT PNr FROM Produkte )  
IN ( SELECT PNr FROM Bestellungen  
WHERE KNr = Kunden.KNr )
- 2) SELECT KNr FROM Bestellungen  
WHERE PNr =ALL (SELECT PNr FROM Produkte)
- 3) SELECT \* FROM Kunden K  
WHERE NOT EXISTS ( SELECT \* FROM Produkte P  
WHERE NOT EXISTS  
( SELECT \* FROM Bestellungen B  
WHERE B.PNr = P.PNr  
AND B.KNr = K.KNr ) )
- 4) SELECT KNr FROM Bestellungen GROUP BY KNr  
HAVING COUNT (DISTINCT PNr) =  
(SELECT COUNT(\*) FROM Produkte)