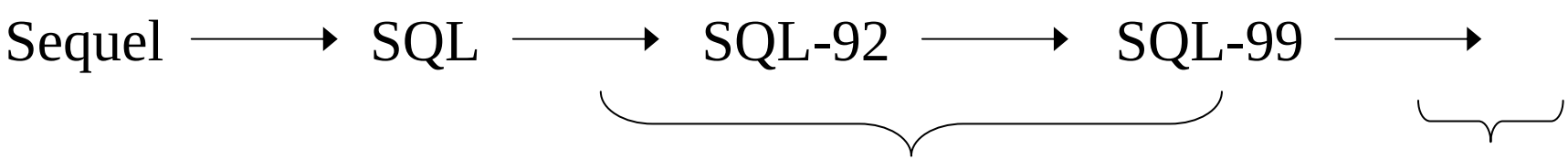
Kapitel 7: Die Datenbanksprache SQL

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

- Σ: 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 $L(G) \subseteq \Sigma^* =$
$\{w \in \Sigma^* \mid$ es gibt endliche Ableitung
$S = x0 \rightarrow x1 \rightarrow ... \rightarrow xn=w$ mit $xi \in (V \cup \Sigma)^*$
und $x(i-1) \rightarrow xi$ als Anwendung einer Regel aus P

Exkurs: Syntaxbeschreibung – einfaches Beispiel

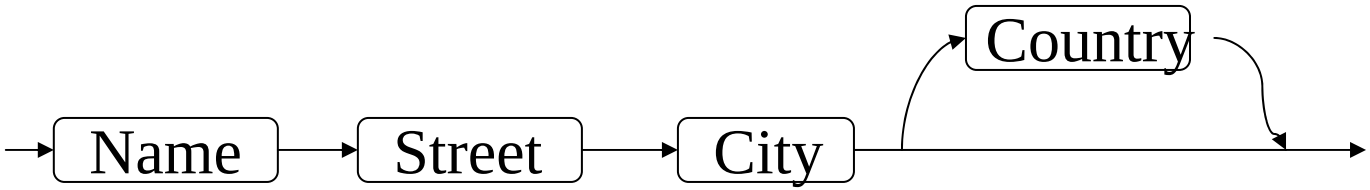
V={Address, Name, Firstname, Lastname, Street, City, Country,
String, Letter, Number, Digit},
Σ={a, b, c, ..., 0, 1, …},
S=Address,
P={Address → Name Street City,
Address → Name Street City Country,
Name → Firstname Lastname,
Name → Letter Lastname,
Firstname → String,
Lastname → String,
Street → String Number,
Country → String,
String → Letter,
String → Letter String,
Number → Digit Number,
Number → ε,
Letter → a, Letter → b, …,
Digit → 0, Digit → 1, …}

Exkurs: Syntaxbeschreibung – EBNF

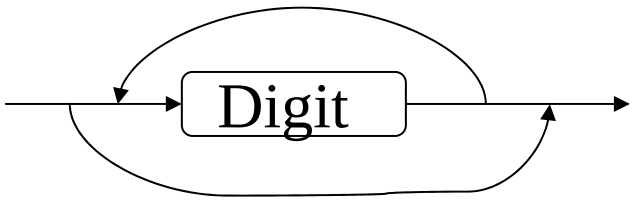
Address ::= Name Street City [Country],
String ::= Letter {Letter …},
Number ::= {Digit …},
Digit ::= 0|1|2|3|4|5|6|7|8|9,
usw.

Exkurs: Syntaxbeschreibung – Syntaxdiagramme

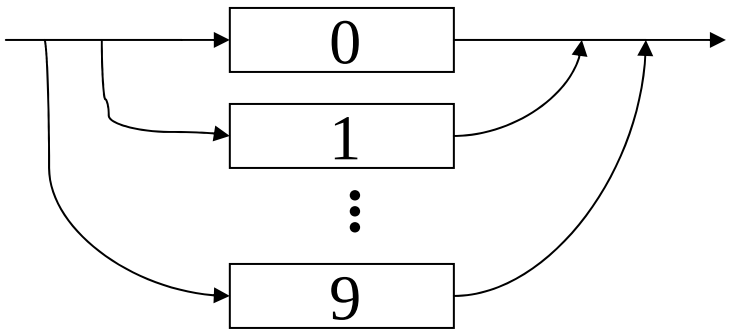
Address ::=



Number ::=



Digit ::=



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)

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

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

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 Rechnungssume 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* θ*ANY Menge* mit $\theta \in \{\{=, \neq, <, >, \leq, \geq\}$ ist erfüllt, wenn es in der Menge ein Element gibt, für das *Wert* θ *Element* gilt. (=ANY ist äquivalent zu IN.)
- Die Bedingung *Wert* θ*ALL Menge* mit $\theta \in \{=, \neq, <, >, \leq, \geq\}$ ist erfüllt, wenn für alle Elemente der Menge gilt: *Wert* θ *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 < 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 → trc query

von select_block-Konstrukten auf TRK-Anfragen
unter Verwendung der Funktion

sql2trc': sql where clause → trc formula

von search_condition-Konstrukten auf TRK-Formeln.

Abbildung auf TRK (1)

sql2trc [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)

$= \{ri_1.A1, ri_2.A2, ... \mid r1 \in REL1 \wedge r2 \in REL2 \wedge \ ... \wedge rm \in RELm$
$\wedge \exists t1 ... \exists tk (t1 \in TAB1 \wedge ... \wedge tk \in TABk \wedge sql2trc'[F])\}$

sql2trc [select-block1 UNION select-block2]

(mit select-block1: SELECT A1, A2, … FROM REL1 R1, …, RELm Rm,
TAB1 T1, …, TABk Tk WHERE F
und select-block2: SELECT B1, B2, ... FROM SET1 S1, ..., SETm' Sm',
PAR1 P1, ..., PARk' Pk' WHERE G)

$= \{u1, u2, ... \mid$
$(\exists r1 ... \exists rm (u1 = r1.A1 \wedge \ u2 = r2.A2 \wedge \ldots \wedge$
$r1 \in REL1 \wedge r2 \in REL2 \wedge \ ... \wedge rm \in RELm \wedge$
$\exists t1 ... \exists tk \ (t1 \in TAB1 \wedge ... \wedge tk \in TABk \wedge sql2trc'[F]))$
$\vee \ (\exists s1 ... \exists sm' (u1 = s1.B1 \wedge \ u2 = s2.B2 \wedge \ldots \wedge$
$s1 \in SET1 \wedge s2 \in SET2 \wedge \ ... \wedge sm' \in SETm' \wedge$
$\exists p1 ... \exists pk' (p1 \in PAR1 \wedge ... \wedge pk' \in PARk' \wedge sql2trc'[G]))\}$

Abbildung auf TRK (2)

sql2trc' [Ri.Aj θ Tk.Bl] = ri.Aj θ tk.Bl
sql2trc' [Ri.Aj θ c] (mit einer Konstanten c) = ri.Aj θ c
sql2trc' [F AND G] = sql2trc'[F] $\wedge$ sql2trc'[G]
sql2trc' [F OR G] = sql2trc'[F] $\vee$ sql2trc'[G]
sql2trc' [NOT F] = $\neg$ sql2trc'[F]

sql2trc' [Ri.Aj IN subquery]
(so daß subquery die Form SELECT Qk.C
FROM QUELL1 Q1, ..., QUELLm' Qm' WHERE H hat)
= $\exists$ q1 … $\exists$ qm' (qk.C = ri.Aj $\wedge$ q1$\in$QUELL1 $\wedge$ … qm'$\in$QUELLm'
$\wedge$ sql2trc'[H])

Abbildung auf TRK (3)

sql2trc' [Ri.Aj θANY subquery]

$= \exists qk\,(qk \in QUELLk \wedge (\exists q1 \ldots \exists q(k-1)\ \exists q(k+1) \ldots \exists qm'$
$(ri.Aj\ \theta\ qk.C\ \wedge q1 \in QUELL1 \wedge \ldots \wedge q(k-1) \in QUELL(k-1) \wedge$
$q(k+1) \in QUELL(k+1) \wedge \ldots \wedge qm' \in QUELLm' \wedge$ sql2trc'[H])))

sql2trc' [Ri.Aj θALL subquery]

$= \forall qk\,(\,(\,qk \in QUELLk \wedge\ (\exists q1 \ldots \exists q(k-1)\ \exists q(k+1) \ldots \exists qm'$
$(q1 \in QUELL1 \wedge \ldots \wedge q(k-1) \in QUELL(k-1) \wedge$
$q(k+1) \in QUELL(k+1) \wedge \ldots \wedge qm' \in QUELLm' \wedge$ sql2trc'[H])))
$\Rightarrow (\ ri.Aj\ \theta\ qk.C\ )\ )$

sql2trc' [EXISTS subquery]

(so daß subquery die Form SELECT C1, C2, ...
FROM QUELL1 Q1, ..., QUELLm' Qm' WHERE H hat)

$= \exists q1 \ldots \exists qm'\ (q1 \in QUELL1 \wedge \ldots \wedge qm' \in QUELLm'$
$\wedge$ 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]

$= \{k.KNr, k.Name \mid k \in Kunden \wedge sql2trc'[K.Ort=... AND NOT EXISTS \ldots]\}$

$= \{k.KNr, k.Name \mid k \in Kunden \wedge k.Ort=... \wedge \neg\ sql2trc'[EXISTS ...]\}$

$= \{k.KNr, k.Name \mid k \in Kunden \wedge k.Ort=... \wedge \neg\ (\exists\ p\ \exists\ b\ (\ p \in Produkte \wedge b \in Bestellungen \wedge sql2trc'[P.Preis>... AND ... AND ...]\ ))\ \}$

$= \{k.KNr, k.Name \mid k \in Kunden \wedge k.Ort=... \wedge \neg\ (\exists\ p\ \exists\ b\ (\ p \in Produkte \wedge b \in Bestellungen \wedge p.Preis>... \wedge p.PNr=b.PNr \wedge b.KNr=k.KNr\ ))\ \}$

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 → ra query

von select_block-Konstrukten auf RA-Anfragen
unter Verwendung der Funktion

sql2ra': sql where clause × ra query → ra query

von search_condition-Konstrukten auf RA-Ausdrücke
sowie der Hilfsfunktion

sql2ra-: sql where clause × ra query → ra query

mit sql2ra- [F](E) = E - π[sch(E)] (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;
π[Ri_1.A1, Ri_2.A2, ...] (sql2ra'[F](R1 × … × Rm × T1 × … × 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] ∪ sql2ra[select-block2]
(mit ggf. notwendigen Umbenennungen von Attributen)

Abbildung auf RA (2)

sql2ra' [Ri.Aj θ Tk.Bl] (E) = σ[Ri.Aj θ Tk.Bl](E)
sql2ra' [Ri.Aj θ c] (E) (mit einer Konstanten c) = σ[Ri.Aj θ c](E)
sql2ra' [F AND G] (E) = sql2ra'[F](E) ∩ sql2tra'[G](E)
sql2ra' [F OR G] (E) = sql2ra'[F](E) ∪ sql2ra'[G](E)
sql2ra' [NOT F] (E) = sql2ra-[F](E) = E - π[sch(E)] (sql2ra'[F](E))

sql2ra' [Ri.Aj IN subquery](E)
(so daß subquery die Form SELECT Qk.C
FROM QUELL1 Q1, ..., QUELLm' Qm' WHERE H hat)
= Q1 := QUELL1; … Qm' := QUELLm';
π[sch(E)] (sql2ra'[H] (σ[Ri.Aj = Qk.C] (E × Q1 × … × Qm')))

Abbildung auf RA (3)

sql2ra' [Ri.Aj θANY subquery] (E)
= Q1 := QUELL1; … Qm' := QUELLm';
π[sch(E)] (sql2ra'[H] (σ[Ri.Aj θ Qk.C] (E × Q1 × … × Qm')))

sql2ra' [Ri.Aj θALL subquery] (E)
= sql2ra- [Ri.Aj θ'ANY subquery](E)
mit θ' gleich ≠ für θ gleich =, = für ≠, < für ≥, > für ≤, usw.
= E - π[sch(E)] (sql2ra'[H] (σ[Ri.Aj θ' Qk.C] (E × Q1 × … × Qm')))

sql2ra' [EXISTS subquery]
(so daß subquery die Form SELECT C1, C2, ...
FROM QUELL1 Q1, ..., QUELLm' Qm' WHERE H hat)
= Q1 := QUELL1; … Qm' := QUELLm';
π[sch(E)] (sql2ra'[H] (E × Q1 × … × 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;
π[K.KNr, K.Name] (sql2ra’[K.Ort=… AND NOT EXISTS …] (K))
= … π[K.KNr, K.Name] (σ[Ort=…](K) $\cap$ sql2ra’[NOT EXISTS …](K))
= … π[K.KNr, K.Name] (σ[Ort=…](K) $\cap$ sql2ra- [EXISTS ...](K))
= … π[K.KNr, K.Name] (σ[Ort=…](K)
$\cap$ (K - π[sch(E)](sql2ra’ [EXISTS ...](K))))
= ... π[K.KNr, K.Name] (σ[Ort=…](K)
$\cap$ (K - π[sch(K)](sql2ra’[P.Preis...AND…AND…](K $\times$ P $\times$ B)))
= ... π[K.KNr, K.Name] (σ[Ort=…](K)
$\cap$ (K - π[sch(K)](σ[P.Preis>100](K $\times$ P $\times$ B) $\cap$
σ[P.PNr=B.PNr](K $\times$ P $\times$ B) $\cap$
σ[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
```

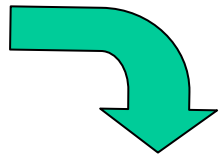
Aggregation mit Gruppierung

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

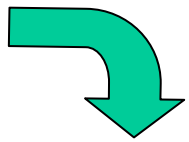
Aggregation mit Gruppierung und Gruppenauswahl

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

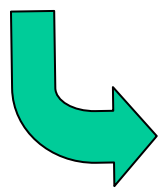
Zwischenresultat 1:

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):

sql2ra [SELECT A', f(B) FROM … WHERE … GROUP BY A]
= R(A, F) := γ_+[A,B,f] (sql2ra[SELECT A,B FROM … WHERE …]);
π_+ [A', F] (R)

Abbildung von

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

auf RA-Programm:

Q(A,B,C) := sql2ra[SELECT … FROM … WHERE …]; R (A,F) := $\emptyset$;
for each x $\in$ π [A](Q) do
Gx := π_+ [A,B,C] (σ_+ [A=x] (Q))
if sql2ra'[cond](Gx) $\neq \emptyset$ then y := f (π_+ [B](Gx)); R := R $\cup_+$ {(x,y)} fi;
od;
π_+[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)
```