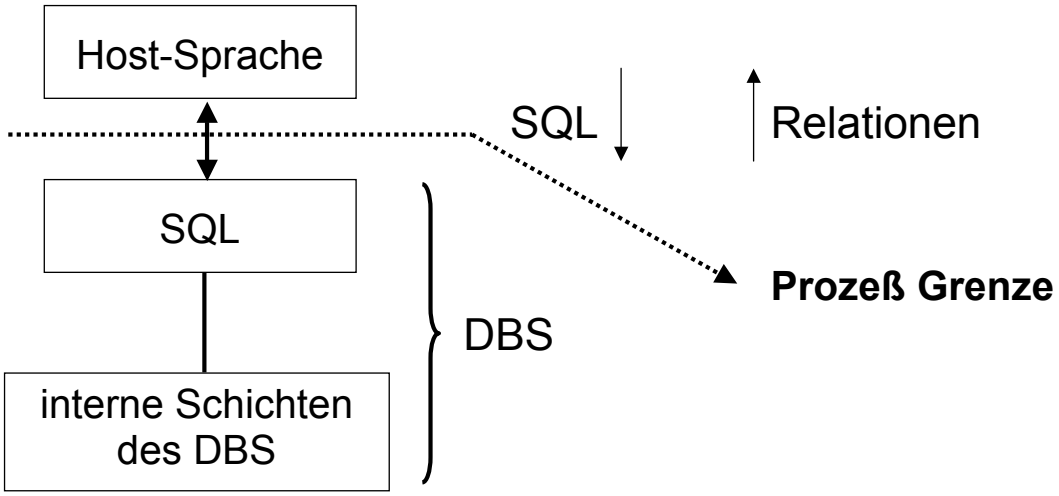
Kapitel 8 Host Sprachen mit Embedded und Dynamic SQL

Kap. 8.1 Statisches SQL

Problem: SQL ist nicht Turing-vollständig, z. B.

- keine Rekursion,
- keine Iteration,
- keine bedingten Anweisungen,
- keine bedingten Sprünge,
- analog zu Sprache der arithmetischen Ausdrücke.

$\Rightarrow$ z. B. transitive Hülle oder Stücklisten-Problem nicht formulierbar

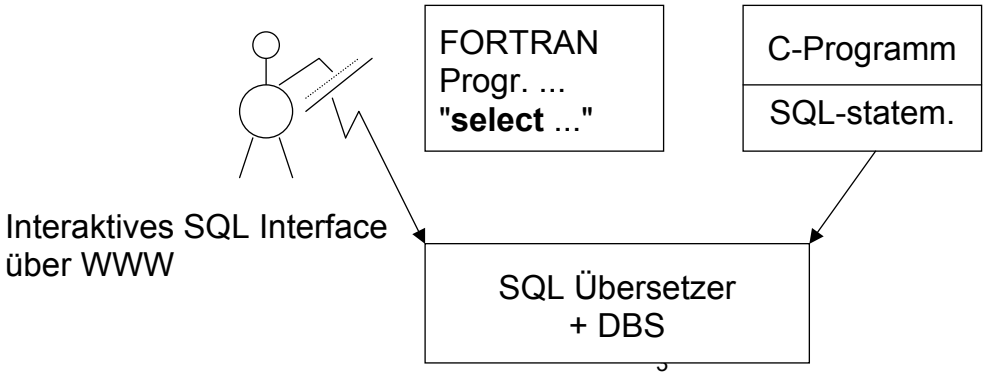


↓ call SS mit Übergabe vollständiger SQL-statements an DBS

↑ tupelweise Rückgabe von Ergebnissen

Embedded SQL: statisches SQL, statements fest im Programmtext eingebettet ⇒ Vorübersetzung und Optimierung durch Präprozessor möglich! (plus Parametrisierung)

Dynamic SQL: SQL statements durch laufendes Programm erst erzeugt ⇒ dynamische Übersetzung und Optimierung



Probleme:
- Sprachkopplung (Erweiterung)
- Compiler - Kopplung
- Übergabe von Ergebnissen
- Prametrisierung von SQL

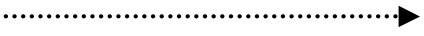
Embedded SQL:

- SQL statements explizit im Programm-Text
- Kopplung über Host-Variablen für Parameter und Ergebnisse

Kopplungsprobl. und Bedingungen für statische Übersetzbarkeit

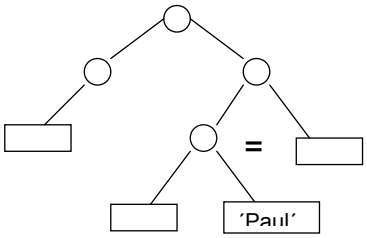
Appl. Progr

select ... **where** Name = 'Paul'



Prozeß-Grenze

DBS mit SQL



Ersetzung von 'Paul' durch 'Bayer' ändert Operatorbaum und Optimierung nicht

⇒ Konstante in SQL durch Host-Variablen ersetzbar!

Hinweis: AP und DBS verschiedene Prozesse, Host-Variablen für DBS nicht zugreifbar.

⇒ parametrisierter Op.-Baum; übersetzbar und optimierbar zur Übersetzungszeit des Host-Programms

⇒ Übergabe der Werte der Host-Varibalen als Parameter zur Ausführungszeit durch Zusatz-SW (libraries) im AP, erfordert Prozesskommunikation, Einbau in vorübersetzte und optimierte Operatorbäume.

Sehr einfache Kopplung, nur Ersetzung von SQL-Konstanten durch Host-Var., aber z. B. keine frei formulierten SQL-Ausweisungen vom Benutzer.

1. Vereinbarung von Host Variablen:

```
EXEC SQL BEGIN DECLARE SECTION
    V1          CHAR
    V2          SMALLINT
    V3          FLOAT
    :
    :
EXEC SQL END DECLARE SECTION
```

Host Variablen sind frei vereinbar.

2.Kopplungsproblem

Austausch von Status-Info über Erfolg von SQL-Anweisungen.
Für Applikation und für bestimmtes DBMS gemeinsame
feste Struktur (Standardisierung schwach, ähnlich wie System-
tabellen!).

```
typedef struct  sqlca  "sql communication area"
{   long  sqlcode  ;
    long  sqlerrd  ;
    char  sqlwarn 0 ;
    char  sqlwarn 1 ;
    :
    long  errline  ;
    ld    taid     ;
    state tastate  ;
    :
    :
}   sqlca
EXEC SQL INCLUDE SQLCA
```


Bedeutungen:

- Erfolg einer Transaktion: sqlcode = 0
  - < 0 : Fehler
  - > 0 : erfolgreich mit condition,
    z. B. "end of table"
- div. Warnungen: z. B. **null** values gelesen, erzeugt
- NOT FOUND : ≈ sqlcode = 100
  - z. B. bei FETCH am Ende eines Cursors
    (Bedingung für Schleifen-Abbr.)
- INSERT, UPDATE, DELETE
  - ohne betroffene Tupel

- **select** ... **into** mit leerem Ergebnis

- Transaktions - Id
- DB - Id

} bei parallelen und verteilten Anwendungen!

Für Details siehe Manuale!

Beispiel: *gemeinsame Variablen zwischen Appl. Und DBMS*

```
EXEC SQL BEGIN DECLARE SECTION
    string    lief# ;
    string    teil# ;
    integer   anzahl
EXEC SQL END DECLARE SECTION;
```



```
begin
    print ('teil # eingeben: ') ;
    read (teil#) ;
EXEC SQL
    select count (*) into   :anzahl
      from LT
      where T# =   :teil#
…   if sqlcode ≠ 0 then Fehlerbehandlung
```

Variablen der Host-Sprache inerhalb von SQL mit Doppelpunkt

Einzelergebnisse: werden in Host-Variablen an Appl. übergeben
Mengenergebnisse: siehe "Cursor"

Update Beispiel: *Status einer Stadt ändern*

```
... BEGIN DECLARE  ...
        integer upgrade;
        string city
... END DECLARE  ...
...   read (upgrade); read (city)     ...

EXEC SQL
    update    LIEF
      set STATUS = STATUS +  :upgrade
      where STADT =  :city  ;
```

Anzahl betroffener Tupel aus SQLCA Komponente ersichtlich.

Lesen eines einzelnen Tupels:

```
EXEC SQL   select A, B, C
                    into :a, :b, :c
                    from ... where ...
```

3.Kopplungsproblem:

Cursor Konzept zum Lesen mehrerer Tupel:

Cursor ~ Ergebnis – Folge von Tupeln in sequentieller Datei

EXEC SQL DECLARE ⟨name⟩ CURSOR
FOR ⟨query⟩

z. B.

```
EXEC SQL  DECLARE liefert-rote-Teile CURSOR
          FOR   select LNAME, TNAME
                from LIEF, LT, TEILE
                where ... and Farbe = 'rot'
```

⇒ veranlaßt Übersetzung, Optimierung zum ***Übersetzungszeitpunkt*** des Host-Programms

EXEC SQL OPEN liefert-rote-Teile

⇒ veranlaßt Ausführung und Bereitsstellung des Ergebnisses über Op.-Baum, Datei.

liefert-rote-Teile ist wie File-Pointer auf Query-Ergebnis!

Hinweis: Zielstruktur von CURSOR wird durch Projektionsliste von ⟨query⟩ festgelegt, z.B. LNAME, TNAME

Holen des nächsten Ergebnis-Tupels:

```
EXEC SQL FETCH liefert-rote-Teile
  INTO  :lief-name,  :teil-name
```

Schleife, um alle Ergebnis-Tupel zu holen:

```
  while sqlca.sqlcode = 0 do
     EXEC SQL FETCH liefert-rote-Teile
     INTO  :lief-name,  :teil-name;
     print (lief-name, teil-name) od
```

Beendigung: um CURSOR zu schließen und SQLCA zu resetten:

```
EXEC SQL CLOSE liefert-rote-Teile
```

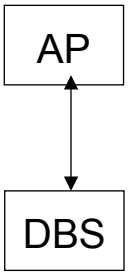
allgemeiner Fall: parametrisierter CURSOR mit Host-Variable "colour"

```
EXEC ...    liefert-farbe CURSOR FOR
       ...  where TEILE.FARBE = :colour
                                 ↑
                            Host-Variable
```

⇒ Bindung der Host-Variablen erst bei
EXEC SQL OPEN liefert-farbe
über parametrisierten Operator-Baum.

Bedeutung von OPEN und FETCH:

OPEN Ergebnismenge wird fixiert, z. B. in Hilfsdatei in DBS



i.e. 2 Prozesse, jedes FETCH erfordert Prozeß-Kommunikation
⇒ Optimierung durch Pufferung!

Verallgemeinerungen: CURSOR vorwärts, rückwärts, scrollable; mehrere CURSOR-Zeiger auf 1 CURSOR.

Kap. 8.2 Dynamisches SQL

Hauptparadigma: Erzeugung des SQL-strings per Programm in einer Host-Variablen, z. B. bei UFI

Beispiel:

1) embedded SQL:

```
print ('Enter employee number:');
read (emplno);

EXEC SQL UPDATE EMPLOYEES
    set base-pay = base-pay + 100
    where employee-no = :emplno;
```


2) mit dynamischem SQL mit beliebiger Bedingung:
über **string** Variablen *part1, part2*, …

```
part 1:  = 'update EMPLOYEES'
part 2:  = 'set base-pay = base-pay + 100'
print ('Enter condition for raise');
read (part 3);
strcat (part 1, part 2); strcat (part 1, part 3);
{ part1 = 'update EMPLOYEES
            set base-pay = base-pay + 100
            where …'  }

EXEC SQL EXECUTE IMMEDIATE :part 1
```

Executable-Statement ::=

Connect-Statement
| Disconnect-Statement
| Login-Statement
| Begin-Work-Statement
| Commit-Work-Statement
| Rollback-Work-Statement
| Use-Transaction-Statement
| Use-Database-Statement
| Set-Sortorder-Statement
| Get-Sortorder-Statement
| Set-Data-Dir-Statement
| Set-Timeout-Statement
| Set-Consistency-Statement
| Send-Interrupt-Statement
| Open-Cursor-Statement

| Fetch-Statement
| Delete-Positioned-Statemnet
| Update-Positioned-Statement
| Close-Cursor-Statement
| Select-Into-Statement
| Embedded-TBSQL-Statement
| Dynamic-ESQL-Statement

Ein Declarative-Statement darf überall dort stehen, wo die Vereinbarung einer C Variablen legal ist.

Kap. 8.3 SQL Erweiterung: Gruppierung

Expr ::= Konstante | Field-Spec |
⟨Aufschichtung mit Operatoren⟩

Beispiel für Field-Spec: L.Straße mit **from** LIEF L
für Expr: 1.16*L.Preis oder sum(Anzahl) avg(Preis)

Allg. Form eines Query-Blocks:

select [distinct] Expr. {, Expr} *Projektionsliste*
from Rel-Spec {, Rel-Spec}
where Expr
group by Field-Spec {, Field-Spec}
having Expr
order by {Attribute, die in Projektionsliste vorkommen}

group by: fasse Tupel mit identischen Werten in Field-Spec zu Gruppen (groups) zusammen ("group attributes")

having: Gruppen-Prädikat: bezieht sich nur auf
- Gruppen-Attribute und
- Aggregatfunktionen über Nicht-Gruppen-Attribute.
  Führt zu Untermenge von vollständigen Gruppen.

Projektionsliste: darf nur noch Gruppen-Attribute oder Aggregatfunktionen über Nicht-Gruppen-Attr. enthalten.
⇒ im Ergebnis nur 1 Tupel pro Gruppe, die das having Prädikat erfüllt.

order by: sortiere nach Attributen, die im Ergebnis vorkommen müssen.

Beispiel: GRUPPEN

A	B	C
1	2	2
1	2	4
1	2	6
1	3	8
1	3	10
1	4	12
1	1	1

```
select A, avg (C), B
    from GRUPPEN
        where C > 2
        group by A, B
            having B < 4 and
                    min (C) < 11
        order by B
```

" *Mittleres Gewicht der roten Teile nach Tname?* "

```
select Tname, avg (Gewicht)
    from TEILE
            where Farbe = 'rot'
    group by Tname
```


" Durchschnittsnote der Informatiker mit mehr als 2 Prüfungen? "

PRÜFUNGEN (M#, Name, StudFach, PrüfFach, Note)

```
select Name, avg (Note), count (*)
    from PRÜFUNGEN
    where Studienfach = 'Informatik'
    group by M#, Name
        having count (*) > 2
    order by Name
```

Hinweis: Ergebnis kann Ausdrücke enthalten, i. e. ohne Bezeichnung

$\Rightarrow$ Referenzierung in **order by** durch Position.

z. B. ... **order by** 2, Name
um nach **avg (Note)** zu sortieren