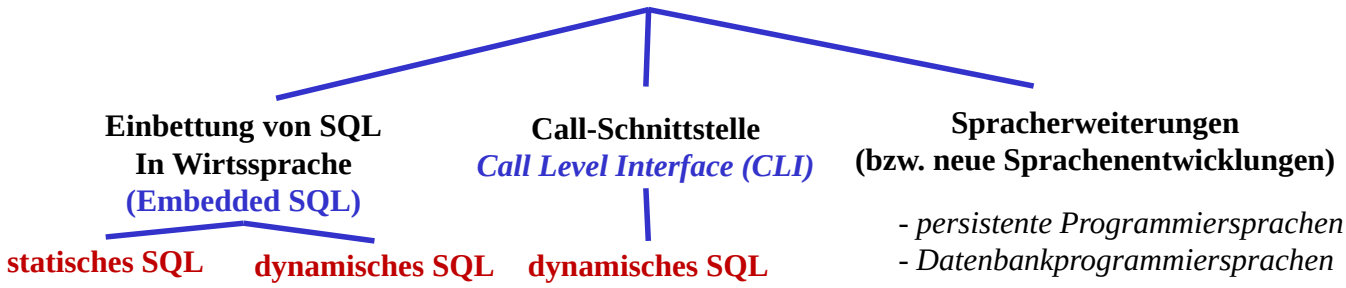
8. Kopplung Programmiersprache - SQL

- Einleitung:
- Eingebettetes SQL
  - Cursor-Konzept
  - positionierte Änderungsoperationen (UPDATE, DELETE)
  - Dynamisches SQL
- Call-Level-Interface

Vertiefung in DBS2:

- Java-Unterstützung: JDBC, SQLJ
- Stored Procedures
- Web-Einbindung von Datenbanken (Servlets, JSP, Skriptsprachen, …)

Kopplung mit einer Wirtssprache



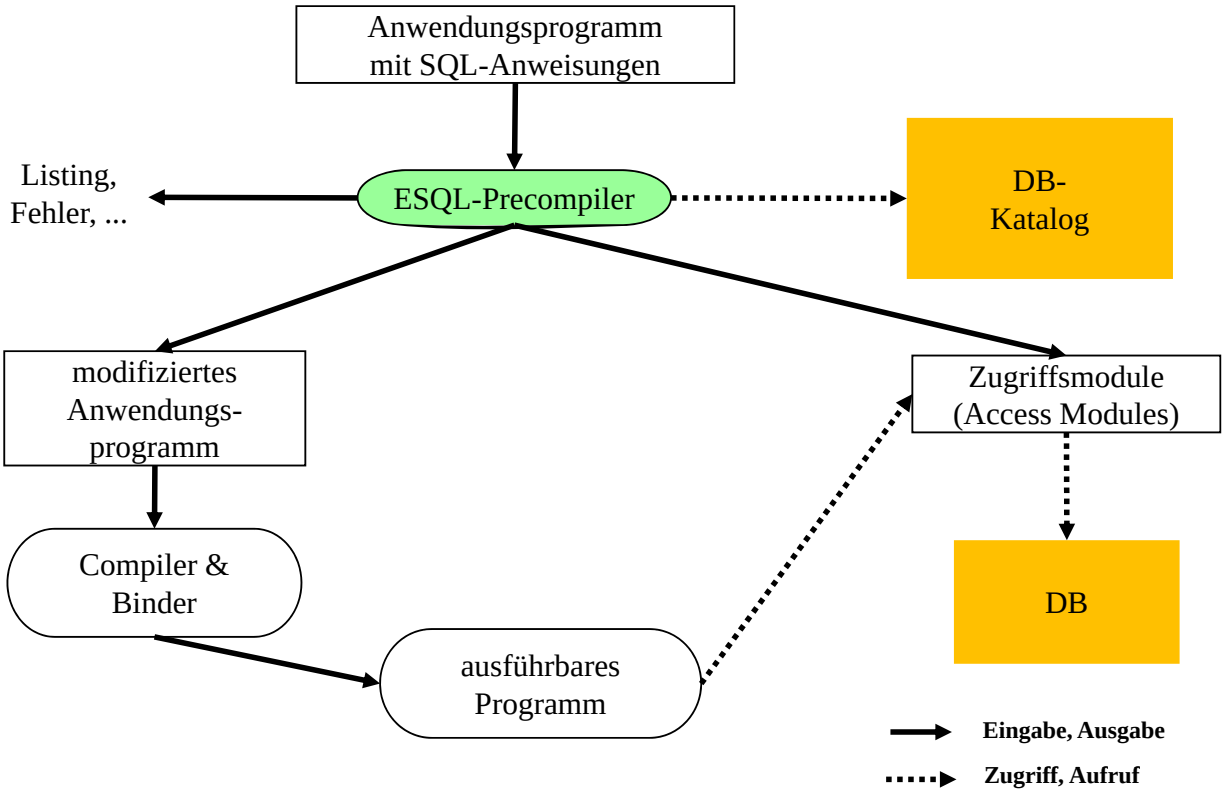
- Einbettung von SQL (Embedded SQL)
  - Spracherweiterung um spezielle DB-Befehle (EXEC SQL ...)
  - Vorübersetzer (Prä-Compiler) wandelt DB-Aufrufe in Prozeduraufrufe um
- Call-Schnittstelle (CLI)
  - DB-Funktionen werden durch Bibliothek von Prozeduren realisiert
  - Anwendung enthält lediglich Prozeduraufrufe
  - weniger komfortable Programmierung als mit Embedded SQL
- Statisches SQL: Anweisungen müssen zur Übersetzungszeit feststehen
  - Optimierung zur Übersetzungszeit ermöglicht hohe Effizienz (Performance)
- Dynamisches SQL: Konstruktion von SQL-Anweisungen zur Laufzeit

Statisches SQL: Beispiel für C

```
exec sql include sqlca; /* SQL Communication Area */
main ()
{
exec sql begin declare section;
   char  X[8];
   int   GSum;
exec sql end declare section;
exec sql connect to dbname;
exec sql insert into PERS(PNR,PNAME,GEHALT) values (4711,'Ernie', 32000);
exec sql insert into PERS(PNR,PNAME,GEHALT) values (4712,'Bert',  38000);
printf("ANR ? "); scanf(" %s", X);
exec sql select sum (GEHALT) into :GSum from PERS where ANR = :X;
printf("Gehaltssumme: %d\n", GSum)
exec sql commit work;
exec sql disconnect;
}
```

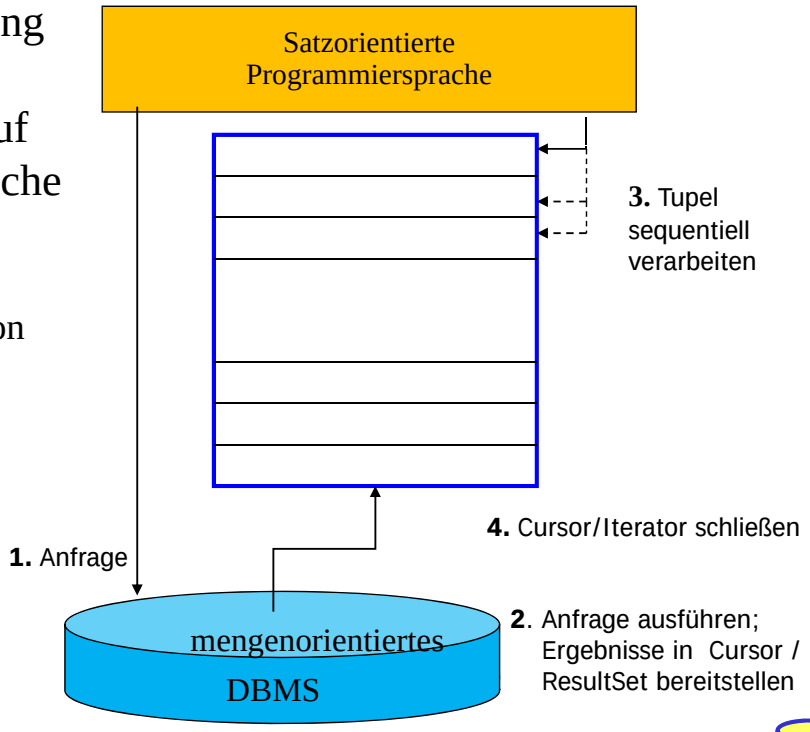
- eingebettete SQL-Anweisungen werden durch "EXEC SQL" eingeleitet und spezielles Symbol (hier ";") beendet, um Compiler Unterscheidung von anderen Anweisungen zu ermöglichen
- Verwendung von AP-Variablen in SQL-Anweisungen verlangt Deklaration innerhalb eines "declare section"-Blocks sowie Angabe des Präfix ":" innerhalb von SQL-Anweisungen
- Werteabbildung mit Typanpassung durch INTO-Klausel bei SELECT
- Kommunikationsbereich SQLCA (Rückgabe von Statusanzeigern u. ä.)

Verarbeitung von ESQL-Programmen



Mengenorientierte Anfragen

- Queries mit nur einem Ergebnissatz
  - einfache Übernahme der Ergebnise in Programmvariable (SELECT attr INTO :var)
- Kernproblem bei SQL-Einbettung in Programmiersprachen: Abbildung von Tupelmengen auf Variablen der Programmiersprache
  - Nutzung von Cursor/Iteratoren bzw. Result-Sets zur satzweisen Bereitstellung und Abarbeitung von DBS-Ergebnismengen



Cursor-Konzept in Embedded SQL

- Cursor ist ein Iterator, der einer Anfrage (Relation) zugeordnet wird und mit dessen Hilfe die Tupeln des Ergebnismenge einzeln (one tuple at a time) im Programm bereitgestellt werden
  - Trennung von Query-Spezifikation (Cursor-Deklaration) und Bereitstellung/Verarbeitung von Tupeln im Query-Ergebnis
- Operationen auf einen Cursor C1
  - DECLARE C1 CURSOR FOR table-exp
  - OPEN C1
  - FETCH C1 INTO VAR1, VAR2, . . ., VARn
  - CLOSE C1
- Anbindung einer SQL-Anweisung an die Wirtssprachen-Umgebung
  - Übergabe der Werte eines Tupels mit Hilfe der INTO-Klausel bei FETCH
    => INTO target-commalist (Variablenliste d. Wirtsprogramms)
  - Anpassung der Datentypen (Konversion)
- kein Cursor erforderlich für Select-Anweisungen, die nur einen Ergebnissatz liefern (SELECT INTO)

Cursor-Konzept (3)

- Beispielprogramm in C (vereinfacht)

```
...
   exec sql begin declare section;
        char  X[50];
        char  Y[8];
        double G;
   exec sql end declare section;
   exec sql declare c1 cursor  for
            select NAME, GEHALT from PERS where ANR = :Y;
   printf("ANR ? "); scanf(" %s", Y);
   exec  sql  open  C1;
   while  (sqlcode == ok) {
            exec  sql  fetch  C1  into  :X, :G;
            printf("%s\n", X)}
   exec  sql  close  C1;
...
```

- DECLARE C1 . . . ordnet der Anfrage einen Cursor C1 zu
- OPEN C1 bindet die Werte der Eingabevariablen
- Systemvariable SQLCODE zur Übergabe von Fehlermeldungen (Teil von SQLCA)

Scroll-Cursor

```
DECLARE cursor [SCROLL] CURSOR FOR table-exp
          [ORDER BY order-item-commalist]
          [FOR {READ ONLY | UPDATE [OF column-commalist]}]
```

- Erweiterte Positionierungsmöglichkeiten durch SCROLL
  - Cursor-Definition (Bsp.):

```
EXEC SQL DECLARE  C2  SCROLL  CURSOR FOR
SELECT NAME, GEHALT FROM PERS
ORDER BY GEHALT ASCENDING
```

- Erweitertes FETCH-Statement:

```
EXEC SQL FETCH [[<fetch orientation>] FROM <cursor>  INTO  <target list>

Fetch orientation:  NEXT, PRIOR, FIRST, LAST,
                    ABSOLUTE  <expression>, RELATIVE <expression>
```

- Beispiele:

DB-Aktualisierung über Cursor

- Wenn die Tupeln, die ein Cursor verwaltet (active set), eindeutig Tupeln einer Relation entsprechen, können sie über Bezugnahme durch den Cursor geändert werden

```
positioned-update ::= UPDATE  table  SET  update-assignment-commalist
                      WHERE CURRENT OF  cursor

positioned-delete ::= DELETE  FROM  table WHERE CURRENT OF cursor
```

- Beispiel:

```
while  (sqlcode == ok) {
   exec sql fetch  C1  into  :X, :G;
         /* Berechne das neue Gehalt in Z */
   exec sql update    PERS
               set  GEHALT  =  :Z
               where  current  of  C1;
  }
```

- keine Bezugnahme bei INSERT möglich!

Verwaltung von Verbindungen

- Zugriff auf DB erfordert i.a. zunächst, eine Verbindung herzustellen, v.a. in Client/Server-Umgebungen
  - Aufbau der Verbindung mit CONNECT, Abbau mit DISCONNECT
  - jeder Verbindung ist eine Session zugeordnet
  - Anwendung kann Verbindungen (Sessions) zu mehreren Datenbanken offenhalten
  - Umschalten der "aktiven" Verbindung durch SET CONNECTION

```
CONNECT TO   target [AS connect-name] [USER user-name]
```

```
SET  CONNECTION   { connect-name | DEFAULT }
```

```
DISCONNECT   { CURRENT | connect-name | ALL }
```

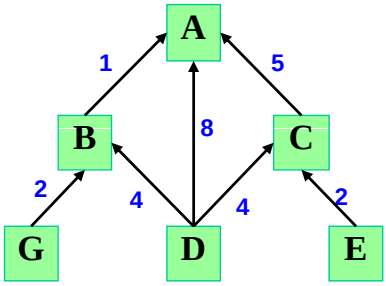

Beispiel: Stücklistenauflösung

- Darstellungsmöglichkeit im RM:

  TEIL (TNR, BEZ, MAT, BESTAND)

  STRUKTUR (OTNR, UTNR, ANZAHL)

Goes Into-Graph



TEIL

TNR	BEZ	MAT	BESTAND
A	Getriebe	-	10
B	Gehäuse	Alu	0
C	Welle	Stahl	100
D	Schraube	Stahl	200
E	Kugellager	Stahl	50
F	Scheibe	Blei	0
G	Schraube	Chrom	100

STRUKTUR

OTNR	UTNR	ANZAHL
A	B	1
A	C	5
A	D	8
B	D	4
B	G	2
C	D	4
C	E	2

- Aufgabe: Ausgabe aller Endprodukte sowie deren Komponenten

Beispiel: Stücklistenauflösung (2)

- max. Schachtelungstiefe sei bekannt (hier: 2)

```
exec sql begin declare section; char T0[10], T1[10], T2[10]; int ANZ;
exec sql end declare section;
exec sql declare C0 cursor for select distinct OTNR from STRUKTUR S1
    where not exists (select * from STRUKTUR S2 where S2.UTNR = S1.OTNR);
exec sql declare C1 cursor for
    select UTNR, ANZAHL from STRUKTUR where OTNR = :T0;
exec sql declare C2 cursor for
    select UTNR, ANZAHL from STRUKTUR where OTNR = :T1;
exec sql open C0;
while (1) {
    exec sql fetch C0 into :T0;
    if (sqlcode == notfound) break;
    printf ("%s\n", T0);
    exec sql open C1;
    while (2) {exec sql fetch C1 into :T1, :ANZ;
               if (sqlcode == notfound) break;
               printf ("  %s: %d\n",T1, ANZ);
               exec sql open C2;
               while (3) {   exec sql fetch C2 INTO :T2, :ANZ;
                             if (sqlcode == notfound) break;
                             printf ("       %s: %d\n",T2, ANZ);  }
               exec sql close C2;  }
    exec sql close C1;  } /* END WHILE */
exec sql close C0;
```

Dynamisches SQL

- dynamisches SQL: Festlegung von SQL-Anweisungen zur Laufzeit
  -> Query-Optimierung i.a. erst zur Laufzeit möglich
- SQL-Anweisungen werden vom Compiler wie Zeichenketten behandelt
  - Dekaration DECLARE STATEMENT
  - Anweisungen enthalten SQL-Parameter (?) statt Programmvariablen
- 2 Varianten: Prepare-and-Execute bzw. Execute Immediate

```
exec sql begin declare section;
   char  Anweisung[256], X[6];
exec sql end declare section;
exec sql declare SQLanw statement;
Anweisung = "DELETE FROM PERS WHERE  ANR = ? AND ORT = ?"; /*bzw. Einlesen
exec sql prepare SQLanw from :Anweisung;
exec sql execute SQLanw using
scanf(" %s", X);
exec sql execute SQLanw using
```


Dynamisches SQL (2)

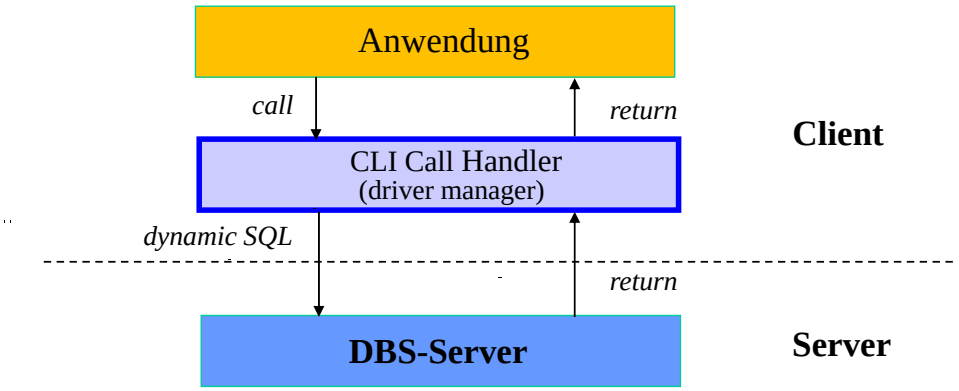
- Variante ohne Vorbereitung: EXECUTE IMMEDIATE
- Beispiel

```
scanf(" %s", Anweisung);
exec sql execute immediate :Anweisung;
```

- Maximale Flexibilität, jedoch potentiell geringe Performance
  - kann für einmalige Query-Ausführung ausreichen

Call-Level-Interface

- alternative Möglichkeit zum Aufruf von SQL-Befehlen innerhalb von Anwendungsprogrammen: direkte Aufrufe von Prozeduren/Funktionen einer standardisierten Bibliothek (API)
- Hauptvorteil: keine Präkompilierung von Anwendungen
  - Anwendungen mit SQL-Aufrufen brauchen nicht im Source-Code bereitgestellt zu werden
  - wichtig zur Realisierung von kommerzieller Anwendungs-Software bzw. Tools
- Einsatz v. a. in Client/Server-Umgebungen



Call-Level-Interface (2)

- Unterschiede in der SQL-Programmierung zu eingebettetem SQL
  - CLI impliziert i.a. dynamisches SQL (Optimierung zur Laufzeit)
  - komplexere Programmierung
  - explizite Anweisungen zur Datenabbildung zwischen DBS und Programmvariablen
  - einheitliche Behandlung von mengenwertigen und einfachen Selects (<-> Cursor-Behandlung bei ESQL)
- SQL-Standardisierung des CLI (Teil von SQL99)
  - starke Anlehnung an ODBC
  - über 40 Routinen:
    - Verbindungskontrolle, Ressourcen-Allokation
    - Ausführung von SQL-Befehlen
    - Zugriff auf Diagnoseinformation, Transaktionsklammerung
- JDBC: neuere Variante

Zusammenfassung

- Cursor-Konzept zur satzweisen Verarbeitung von Datenmengen
  - Operationen: DECLARE CURSOR, OPEN, FETCH, CLOSE
  - Erweiterungen: Scroll-Cursor, Sichtbarkeit von Änderungen
- Statisches (eingebettetes) SQL
  - hohe Effizienz, relativ einfache Programmierung
  - begrenzte Flexibilität (Aufbau aller SQL-Befehle muß zur Übersetzungszeit festliegen, es können nicht zur Laufzeit verschiedene Datenbanken angesprochen werden)
- Call-Level-Interface (z.B. JDBC)
  - keine Nutzung eines Präcompilers
  - Einsatz v.a. in Client-Server-Systemen
  - breite Unterstützung

Vorschau SS2010

- Datenbanksysteme 2
  - Anwendungsprogrammierung (stored procedures, JDBC), Web-Anbindung von DB
  - Objektorientierte DBS, O/R-Mapping (Hibernate)
  - Objektrelationale DBS (SQL99, SQL2003)
  - XML-Datenbanken (XML-Schema, XQuery)
- Relationales Datenbankpraktikum (IBM DB2)
  - Entwurf einer Datenbank für gegebene Aufgabenstellung
  - Einrichtung der DB; initiales Laden mit Testdaten
  - Realisierung von Anwendungsfunktionen (in Java)