Seminar: DB-Anwendungsentwicklung und objekt-relationale DBVS

OR-Anwendungsentwicklung

Robert Klensch

08.02.2001

UNIVERSITÄT KAISERSLAUTERN

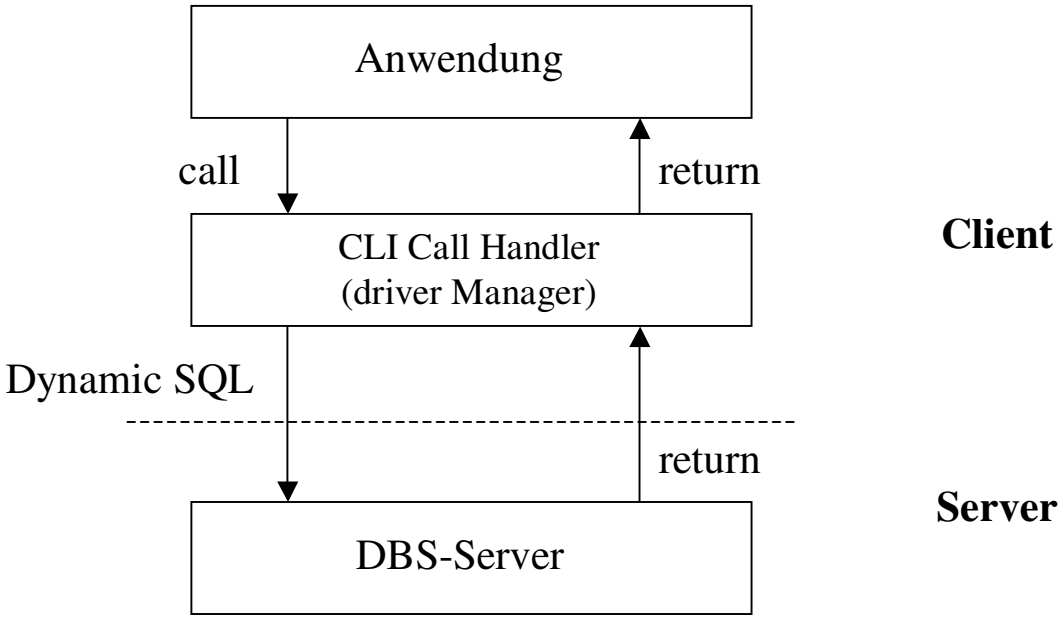
Fachbereich Informatik
AG Datenbanken & Informationssysteme
http://wwwdbis.informatik.uni-kl.de/

DBIS Datenbanken und Informationssysteme

Gliederung

1. Call Level Interfaces
   - ODBC
   - JDBC
2. Beispieldatenbank
3. Anwendungsbeispiel
   - JDBC
4. SQLJ
   - Part 0: Embedded SQL in Java
   - Part 1: SQL Routines using the Java Programming Language
   - Part 2: SQL Types Using the Java Programming Language
5. Zusammenfassung

Call Level Interfaces



Call Level Interfaces

Vorteile (gegenüber embedded SQL)

- keine Precompilierung

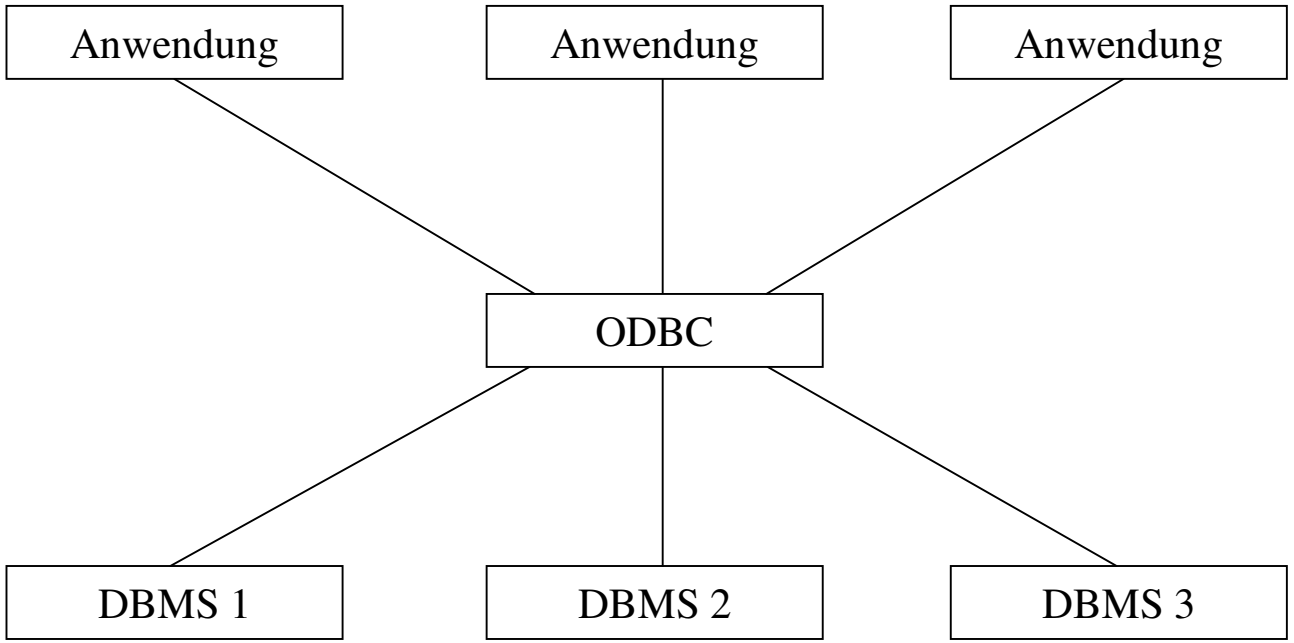
Nachteile (gegenüber embedded SQL)

- komplexere Programmierung
- kein "Precompilierungsvorteil"

CLI-Standard umfasst über 40 Routinen

- Ressourcen-Allokation und -Deallokation
- Ausführen von SQL-Befehlen
- Zugriff auf Diagnoseinformationen
- Transaktionsklammerung
- Informationsanforderung zur Implementierung

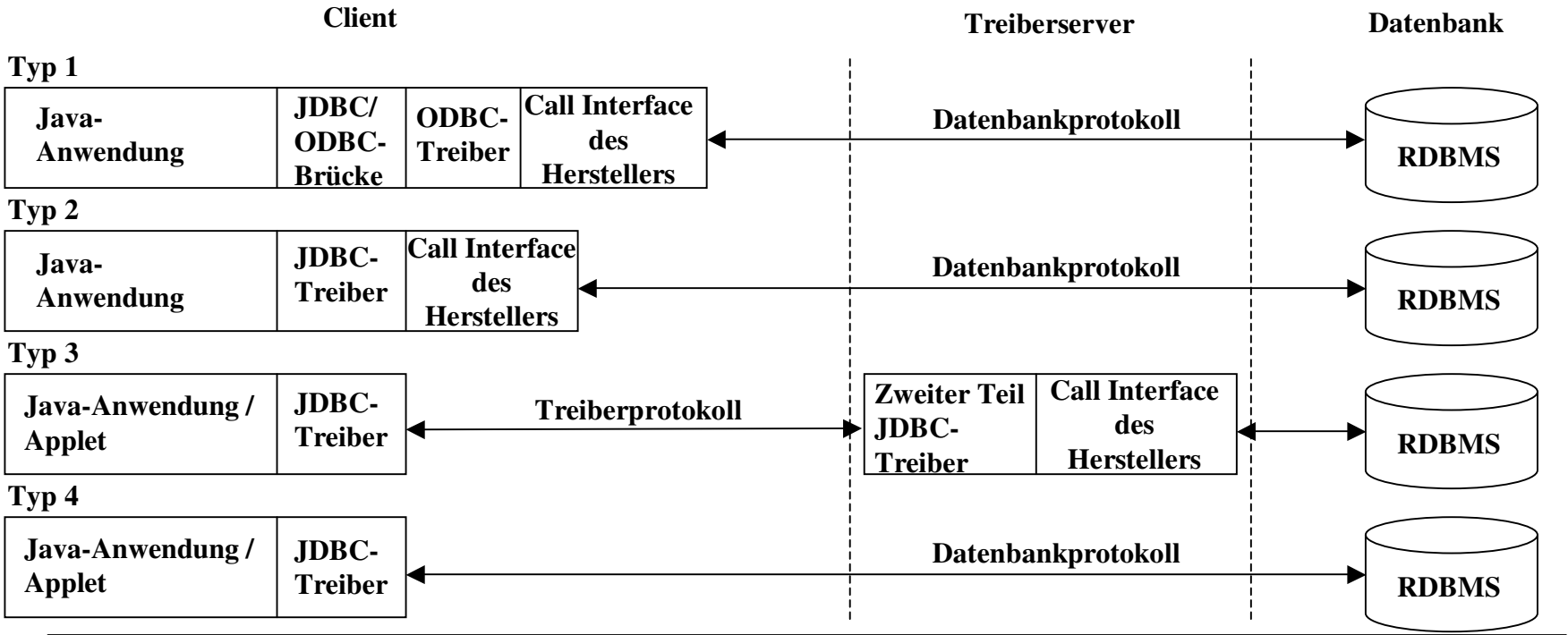
ODBC



JDBC

4 Architekturen

- Typ 1 - JDBC-ODBC Bridge
- Typ 2 - Native-API Partly Java Driver
- Typ 3 - Net-Protocol All-Java Driver
- Typ 4 - Native-Protocol All-Java Driver



Beispieldatenbank

Ziel: Abspeichern von CD-Informationen

Verwendete Attribute:

- Albumname Varchar(50)
- Titel ARRAY OF TitelStruktur
- Preis DM
- Coverbild BLOB
- Musiker REF(KuenstlerTabelle)

Datentyp Titelstruktur:

```
CREATE TYPE TitelStruktur AS (
    Titelname   VARCHAR(100),
    Spielzeit   TIME
)
NOT FINAL;
```

Beispieldatenbank

Datentyp DM:

```
CREATE TYPE DM AS DOUBLE FINAL;
```

Datentyp Kuenstler:

```
CREATE TYPE KuenstlerType AS (
   Name            VARCHAR(50),
   Geburtsdatum    DATE,
   Nationalitaet   VARCHAR(50)
)
NOT FINAL
REF IS SYSTEM GENERATED;
```

Tabelle KuenstlerTabelle:

```
CREATE TABLE KuenstlerTabelle OF KuenstlerType (
   REF IS aref SYSTEM GENERATED );
```

Beispieldatenbank

CDTabelle:

```
CREATE TABLE CDTabelle (
        Albumname       VARCHAR(50) PRIMARY KEY,
        Titel           TitelStruktur ARRAY[20],
        Preis           DM,
        Coverbild       BLOB(1M),
        Musiker         REF(KuenstlerTabelle)
);
```

Anwendungsbeispiel:JDBC

```
Connection con = null;

String URL = "jdbc:sybase:TDS:localhost:2244";

try {
        Class.forName("com.sybase.jdbc.SybDriver");
}
catch (ClassNotFoundException cnf) {
        System.out.println(cnf.getMessage());
}

try {
        con = DriverManager.getConnection (URL,"user","password");
}
catch (SQLException sql) {
        System.out.println(sql.getMessage());
}
```

Anwendungsbeispiel:JDBC

```
Statement stmt = con.createStatement();

String stmtSource = "SELECT Albumname,Titel,Preis,Coverbild,Musiker
                     FROM CDTabelle
                     WHERE Albumname = 'Fields of Gold' ";

ResultSet rs = stmt.executeQuery(stmtSource);
```

Anwendungsbeispiel:JDBC

```
while (rs.next()) {
        String JAlbumname = rs.getString("Albumname");
        Array JTitel = rs.getArray(2);
        Struct[] structArray = (Struct[]) JTitel.getArray();
        for (int i=0;i < structArray.length;i++) {
                Object[] Attribute = structArray[i].getAttributes();
                for (int j=0; j<Attribute.length;j++) {
                        System.out.println(Attribute[j]);
                }
        }
        Blob Coverbild = rs.getBlob(4);
        long CoverbildLaenge = Coverbild.length();
        double JPreis = rs.getDouble("Preis");
        Ref JRef = rs.getRef("MusikerTabelle");
}
```

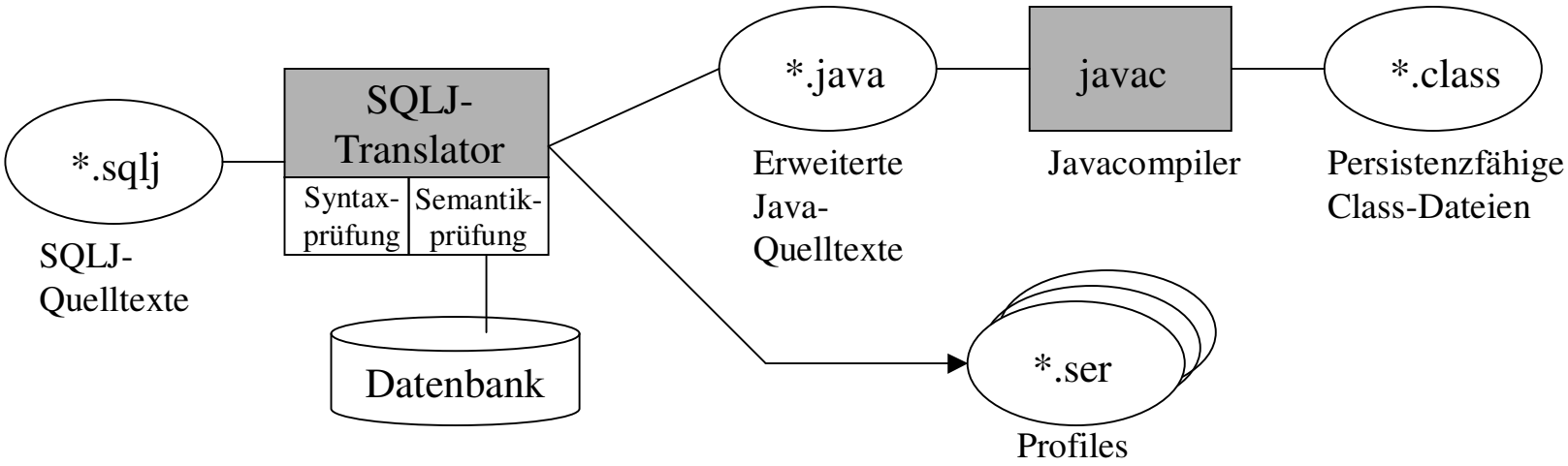
SQLJ - Part 0: Embedded SQL in Java

Vorteile (gegenüber JDBC)

- kurze und prägnante Schreibweise
- komfortables Programmieren von Datenbankanfragen
- Syntaxprüfung zur Übersetzungszeit
- Semantikprüfung (Vergleiche mit bestehenden Datenbankschemata) zur Übersetzungszeit
- Anpassbarkeit an ein Zieldatenbanksystem, ohne den Quellcode zu ändern

Nachteil (gegenüber JDBC)

- statisch



SQLJ - Part 0: Embedded SQL in Java

Verbindungskontext

```
try {
        Class.forName("com.sybase.jdbc.SybDriver");
        String URL = "jdbc:sybase:TDS:localhost:2244";
        DefaultContext defCtxt = new DefaultContext(URL,"user",
                                                    "password",false);
        DefaultContext.setDefaultContext(defCtxt);

        #sql { DELETE
               FROM CDTabelle
               WHERE Albumname='Fields of Gold'
             };
        }
catch(SQLException sqlex) {
        System.out.println(sqlex.getMessage());
}
catch (Exception ex) {
        System.out.println(ex.getMessage());
}
```

SQLJ - Part 0: Embedded SQL in Java

Explizite Definition von Verbindungskontexten:

```
#sql context ContextName;
...
ContextName ContextNameInstanz =
    new ContextName(URL,"user","password",false);
...
#sql [ContextNameInstanz] {
    /* SQL-Anweisung */
 }
```

SQLJ - Part 0: Embedded SQL in Java

Java Variablen in embedded SQL-Anweisungen

```
String JAlbumname;
int i=35;
#sql { SELECT   Albumname
       INTO     :JAlbumname
       FROM   CDTabelle
       WHERE   Preis = :i
};
```

SQLJ - Part 0: Embedded SQL in Java

Iteratoren - Named iterator

```
#sql iterator CdIterator (String Albumname, double Preis);
...
CdIterator cditerator1;
#sql cditerator1 = { SELECT Albumname,Preis
                     FROM CDTabelle
                     WHERE Musiker->name ='Sting' };
...
while (cditerator1.next()) {
    System.out.println(cditerator1.Albumname()+"kostet :"
    +cditerator1.Preis()+"DM");
}
cditerator1.close();
```

SQLJ - Part 0: Embedded SQL in Java

Iteratoren - Positioned iterator

```
#sql iterator CdIterator2(String,double);
...
CdIterator cditerator2;
String JAlbumname;
Double JPreis;
#sql cditerator2 = { SELECT Albumname,Preis
                     FROM CDTabelle
                     WHERE Musiker->name ='Sting' };
while(true) {
    #sql {FETCH :cditerator2 INTO :JAlbumname,JPreis};
    if (cditerator2.endFetch()) break;
    System.out.println(Albumname+"kostet :"+JPreis+"DM");
}
cditerator2.close();
```

SQLJ - Part 0: Embedded SQL in Java

Mehrere ResultSets

```
ExecutionContext execCont = new ExecutionContext();
#sql [execCont] {CALL Prozedur() };
ResultSet rs;
while ((rs = execCont.getNextResultSet()) = null) {
    while(rs.next()) {
            System.out.println(rs.getString(1));
    }
}
```

SQLJ - Part 1: SQL Routines using the Java Programming Interface

- 1999 verabschiedet
- Spezifikation von "Stored Procedures" und "Stored Functions" in Java
- keine semantischen Unterschiede zwischen herkömmlichen und durch Java Methoden gebildete Prozeduren und Funktionen
- Java Virtual Machine

SQLJ - Part 1: SQL Routines using the Java Programming Interface

Funktionen

```
public static double DMinDollar(double DM) {
                return(DM/2,17);
}
```

In SQL:

```
CREATE FUNCTION SQLDMinDollar (DOUBLE) RETURN DOUBLE
EXTERNAL NAME 'CDTabelle_jar:CDTabelle.DMinDollar(double)'
LANGUAGE JAVA
PARAMETER STYLE JAVA
```

Funktionsaufruf :

```
SELECT   Albumname,SQLDMinDollar(Preis)
FROM    CDTabelle
WHERE Musiker->Name='Sting'
```

SQLJ - Part 1: SQL Routines using the Java Programming Interface

Funktionen - Java Methoden, die SQL-Anweisungen ausführen

```
public static int ZaehleAlben (String Kuenstlername)
                throws SQLException {
    int Anzahl =0;
        String stmt = "SELECT COUNT(*) "+" FROM CDTabelle "
                      +"WHERE Musiker->Name = '"
                      +Kuenstlername+"'";
    Connection con =
        DriverManager.getConnection("jdbc:default:connection");
    ResultSet rs = con.createStatement().executeQuery(stmt);
    rs.next();
    Anzahl = rs.getInt(1);
    return Anzahl;
}
```

In SQL:

```
CREATE FUNCTION SQL_ZAEHLEALBEN(IN Kuenstlername VARCHAR(50))
RETURNS INTEGER
EXTERNAL NAME 'CDTabelle_jar:CDTabelle.ZaehleAlben'
LANGUAGE JAVA
PARAMETER STYLE JAVA
```

SQLJ - Part 1: SQL Routines using the Java Programming Interface

Prozeduren

```
public static void DMinDollar (double DM,double[] Dollar) {
        Dollar[0] = DM / 2.17;
}
```

In SQL:

```
CREATE PROCEDURE DMinDollar (IN DOUBLE, OUT DOUBLE)
EXTERNAL NAME 'CDTabelle_jar:CDTabelle.DMinDollar '
LANGUAGE JAVA
PARAMETER STYLE JAVA
```

SQLJ - Part 1: SQL Routines using the Java Programming Interface

Prozeduren - ResultSets in Prozeduren

```
public static void Alben (String Name,ResultSet[] rs) throws
    SQLException {
    String stmt = "SELECT Albumname"
                  +"FROM CDTabelle"
                  +"WHERE Musiker->Name = "+"'"+Name+"'";
    Connection con = DriverManager.getConnection
        ("jdbc:default:connection");
    rs[0] = con.createStatement().executeQuery(stmt);
}
```

SQL:

```
CREATE PROCEDURE SQL_Alben (IN NAME VARCHAR(20))
READS SQL DATA
DYNAMIC RESULT SETS 1
EXTERNAL NAME 'CDTabelle_jar:CDTabelle.Alben(String,ResultSet[]) '
LANGUAGE JAVA
PARAMETER STYLE JAVA
```

SQLJ - Part 2: SQL Types Using the Java Programming Language

- Abbilden von Java-Klassen auf SQL-structured types
- Interface Serializable oder java.sql.SQLDATA
- Klassenmember müssen als "public" deklariert werden

SQLJ - Part 2: SQL Types Using the Java Programming Language

```
package CDTabelle;
import java.util.*;
import java.io.*;
import java.sql.*;

public class Kommentar implements java.io.Serializable {
        public String Name;
        public String Kommentar;

        public Kommentar(String Name,String Kommentar) {
                this.Name = Name;
                this.Kommentar = Kommentar;
        }
}
```

SQLJ - Part 2: SQL Types Using the Java Programming Language

```
CREATE TYPE SQL_Kommentar
        EXTERNAL NAME 'CDTabelle_jar:CDTabelle.Kommentar '
        LANGUAGE JAVA
        ( Name          VARCHAR(20) EXTERNAL NAME 'Name'
          Kommentar     VARCHAR(100) EXTERNAL NAME 'Kommentar'

        METHOD SQL_Kommentar(v1 VARCHAR(20), v2 VARCHAR(100))
               RETURNS SQL_Kommentar
               EXTERNAL NAME 'Kommentar'
        )
```

Einsatzbeispiel:

```
INSERT INTO CDTabelle(Kommentar)
VALUES (NEW SQL_Kommentar('MR.X','sehr gutes Album'))
WHERE Albumname = 'Fields of Gold';
```

SQLJ - Part 2: SQL Types Using the Java Programming Language

Verwenden von eigenen Methoden

```
public class Kommentar2 implements java.io.Serializable {
        ...
        public String toString() {
                return( "Der Kommentar von"+Name+"zu diesem Album :"
                        +Kommentar);
        }
}
```

In SQL :

```
CREATE TYPE SQL_Kommentar
        EXTERNAL NAME 'CDTabelle_jar:CDTabelle.Kommentar2'
        LANGUAGE JAVA
        ( ...

        METHOD SQLtoString()
                RETURNS VARCHAR(200)
                EXTERNAL NAME 'toString' )
```

SQLJ - Part 2: SQL Types Using the Java Programming Language

Methodenaufruf:

```
SELECT  Kommentar.SQLtoString
FROM CDTabelle
WHERE Albumname = 'Fields of Gold'

Resultat: "Der Kommentar von MR.X zu diesem Album: sehr gutes
          Album"
```

SQLJ - Part 2: SQL Types Using the Java Programming Language

Sortieren:

Equals only

```
CREATE ORDERING FOR SQL_Kommentar
    EQUALS ONLY BY MAP WITH METHOD SQLtoString
```

Order full

```
CREATE ORDERING FOR SQL_Kommentar
    ORDER FULL BY RELATIVE WITH COMPARABLE INTERFACE
```

Zusammenfassung

- Call Level Interfaces
  - Open Database Connectivity (ODBC) von Microsoft
  - Java Database Connectivity (JDBC) von JavaSoft

- Einführung und Zugriff auf eine Beispieldatenbank
  - Verbindungsaufbau und Anfrageformulierung mittels Beispielanwendung von JDBC
  - Anbinden der Variablen an die Programmiersprache Java
  - schlechte Unterstützung der neuen Datentypen durch ODBC 3.0

- SQLJ
  - SQLJ in drei Teile eingeteilt
  - Teil 0 behandelt embedded SQL in Java
  - Teil 1 erlaubt SQL-Funktionen und –Prozeduren aus Java-Methoden zu verwenden
  - Teil 2 erlaubt es benutzerdefinierte Datentypen aus Java-Klassen als SQL-Datentypen zu verwenden