# DB-Programmierung





## Ziele

- static SQL verstehen
  - Build-Prozess / Art des Datenzugriffs
  - Host-Variablen
  - Vor- / Nachteile
- dynamic SQL verstehen
  - Build-Prozess / Art des Datenzugriffs
  - Parameter Marker
  - Vor- / Nachteile
- DB2 – Zugriff mit Java selbst programmieren

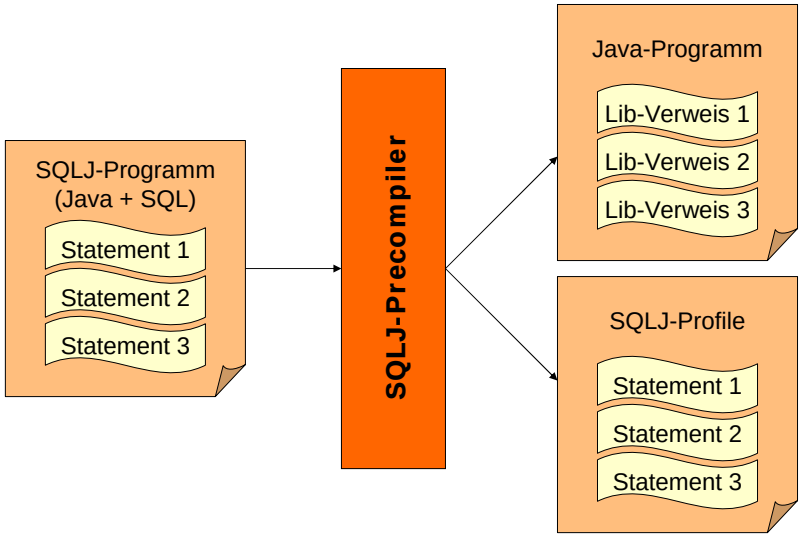
# Dynamisch vs. Statisch

- Dynamisch
  - Statement wird ausgeführt wie im Programm stehend, mit den Rechten des ausführenden Nutzers
  - Java-Speak: JDBC
- Statisch
  - SQL direkt im Code, durch Präprozessor zu parsen
  - 2-pass-Compilierung nötig
    - „Vorkompilierung“: Alle Statements werden erkannt, in ein DB-internes Format umgewandelt und durch Referenzen auf dieses Format ersetzt. Ergebnis: nativer Code (z.B. Java oder C)
    - Eigentliche Kompilierung durch den Compiler der Sprache
  - Java-Speak: SQLJ

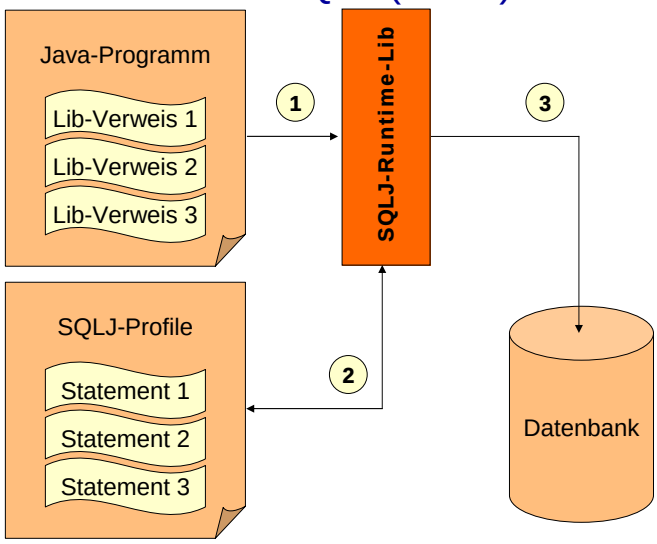




# SQLJ (cont.)



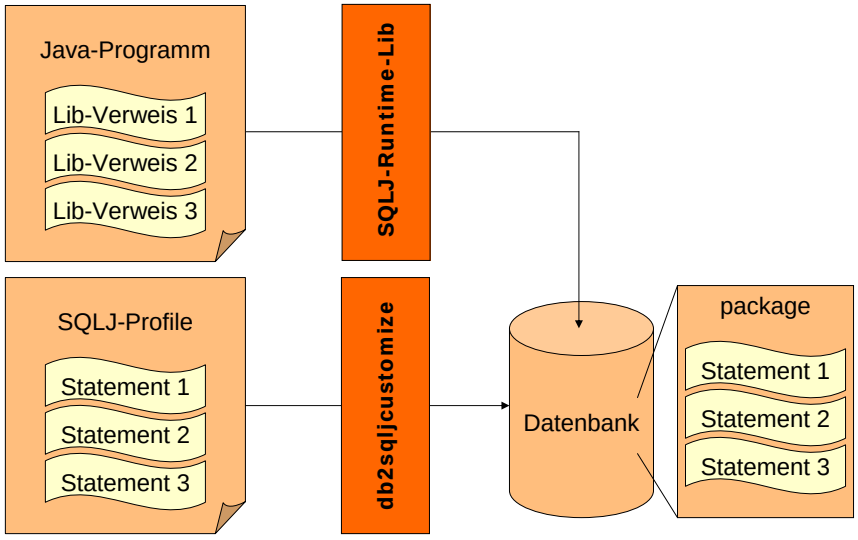
## SQLJ (cont.)








## SQLJ (cont.)



# SQLJ (cont.)

```
public class Select {

        public void employee(String empNo) {

                String firstName = null;
                String lastName = null;

                // use the DB2 SAMPLE database
                String url = "jdbc:db2:SAMPLE";

                // Get the connection
                con = DriverManager.getConnection(url);

                //Lookup the employee given the employee number
                #sql { SELECT FIRSTNME, LASTNAME INTO :firstName, :lastName
                       FROM EMPLOYEE WHERE EMPNO = :empNo } ;

                System.out.println ("Employee " + firstName + "  " + lastName);

                con.close();
        }
```





# SQLJ (cont.)

```
public void listStudents( ... )
{
  //Iterator definieren
  #sql iterator StudentenItr ( String Name, int Semester );
  ...

  StudentenItr Stud;

  //SQL-Anfrage deklarieren; wird in Datenbank übernommen;
  //Referenz im Code
  #sql Stud = { SELECT Name, Semester FROM Studenten
                where Semester >13 };

  //Ausgabe der Ergebnisse
  while ( Stud.next() ) {
        System.out.pintln( Stud.Name() + ": " + Stud.Semester() )
  ...
}
```

## SQLJ (cont.)

- SQLJ programs require fewer lines of code (shorter, and hence easier to debug).
- SQLJ can perform syntactic and semantic checking on the code, using database connections at compile time.
- SQLJ provides strong type-checking of query results and other return parameters.
- SQLJ provides a simplified way of processing SQL statements.

- JDBC provides finer-grained control over the execution of SQL statements and offers true dynamic SQL capability

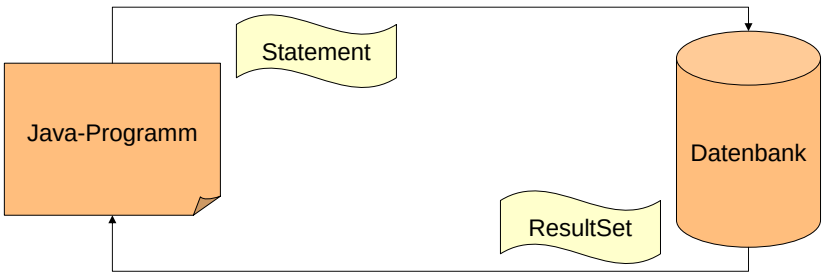




## SQLJ (cont.)

- Vorteile:
  - (Vor-)Compilierung / Optimierung der Statements
  - Statements werden mit den Rechten des „Binders“ ausgeführt
- Nachteile:
  - Ungleich aufwändiger in der Entwicklung durch zusätzliche Build-Schritte
  - Keine aktuellen Statistiken für Optimierung

# JDBC








# JDBC

- Dynamischer Zugriff
  - Anfragen werden mittels DB-Methoden direkt ausgeführt
  - Vorteil: Immer aktuelle Statistiken für Optimizer
  - Nachteil: Statement muss noch kompiliert werden. (Jedoch Kompensation: Statement – Pool)
  - Beispiel:

```
Public static void main() {
  Connection con = DriverManager.getConnection()
  PreparedStatement stmt = null;
  stmt = con.prepareStatement („SELECT name FROM person“);
  ResultSet rs = stmt.executeQuery();
  rs.print();
}
```

## JDBC – Objekte

java.sql.DriverManager

```
Class.forName("COM.ibm.db2.jdbc.app.DB2Driver").newInstance();
```

java.sql.Connection

```
Connection con = DriverManager.getConnection("jdbc:db2:dbprak",username,password);
```

java.sql.PreparedStatement

```
PreparedStatement stmt = con.prepareStatement („SELECT name FROM person“);
```

java.sql.ResultSet

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





## DriverManager

- Singleton-Klasse, nur static Methoden, kümmert sich um die Treiberverwaltung
- Wird eine Treiberklasse geladen, muss sich diese selbst beim DriverManager anmelden (Konvention)
- Ein getConnection() auf dem DriverManager kümmert sich dann anhand der URL um den richtigen Treiber.

## DriverManager (cont.)

```
// Laden des Datenbanktreibers

static {
        try {
                Class.forName ("COM.ibm.db2.jdbc.app.DB2Driver").newInstance ();
        } catch (Exception e) {
                System.out.println ("\n  Error loading DB2 Driver...\n");
                System.out.println (e);
                System.exit(1);
        }
}
```





## Connection

- Repräsentiert eine Verbindung in die Datenbank
- „erzeugt“ Statements etc.
- Stellt Meta-Daten über die Datenbank bereit (resultset..)
- Setzt Transaktionseigenschaften

## Connection URLs

- Application Driver (Type2) (db2java.zip):
  - Class: COM.ibm.db2.jdbc.app.DB2Driver
  - URL: „jdbc:db2:<dbname>“
- Applet Driver (Type3) (*obsolet*):
  - Class: COM.ibm.db2.jdbc.net.DB2Driver
  - URL: "jdbc:db2://<server>:<port>/<dbname>„
  - Port-Default: 6789
- Native Driver (Universal Driver) (db2jcc.jar):
  - Class: com.ibm.db2.jcc.DB2Driver
  - URL: "jdbc:db2://<server>:<port>/<dbname>„
  - Port-Default: 50004





## Statement-Interfaces 1

- java.sql.Statement
  - SQL-Statement, welches ausgeführt werden soll
- Aufrufmethoden:
  - execute (String sql, ...): Führt beliebiges Statement aus, kann mehrere ResultSets zurückliefern
  - executeQuery (String sql, ...): Führt das Statement aus, liefert ein ResultSet zurück
  - executeUpdate (String sql, ...): Führt ein INSERT / UPDATE / DELETE oder DDL statement aus. Kein ResultSet.

## Statement-Interfaces 2

- PreparedStatement extends Statement
  - Statement wird von der Datenbank vorbereitet
  - Statement darf Parameter-Marker enthalten
  - Vorbereitetes Statement kann mehrfach (mit unterschiedlichen Bindings) ausgeführt werden
- Beispiel

```
stmt = con.prepareStatement(„select * from person where
  username=?“);
stmt.setString(1, ´Mueller´);
rs = stmt.executeQuery();
```





## Parameter-Marker

- Parameter für eine Query werden als Platzhalter „?“ in die Query eingefügt
- Werte werden an diese Platzhalter gebunden
- Vorteile:
  - Vermeidung von SQL Injection
  - Wiederverwendbarkeit des Statements
  - Statement-Caching in der Datenbank

## Host-Variablen

- SQLJ-Äquivalent zu Parameter-Markern
- Beispiel:

```
#sql Stud = {SELECT Name, Semester FROM Studenten where Semester > :sem };
```

- Achtung bei Host-Variablen in C!





## ResultSet

- Repräsentiert eine Antwort-Tabelle auf eine Query
- I.d.R. forward-only, not updatable

```
Statement stmt = con.createStatement();
ResultSet rs = stmt.executeQuery("SELECT a, b FROM daten");
while (rs.next()) {
        System.out.println (rs.getInt ("b"));
        System.out.println (rs.getString (1));
}
```

## Exception Handling

- Fast alle SQL – Rufe *werfen* SQLException
- Gutes Error-Handling also wichtig
- Siehe getErrorCode(), getSQLState()





## JDBC revisited

```
try{
  Class.forName("COM.ibm.db2.jdbc.app.DB2Driver").newInstance();

  Connection con = DriverManager.getConnection(“jdbc:db2:dbprak“);

  PreparedStatement stmt = null;

  stmt = con.prepareStatement (“SELECT name FROM person WHERE id = ?“);

  stmt.setInt (1, 25);

  ResultSet rs = stmt.executeQuery();

  while (rs.next()) {

    System.out.println (rs.getString(1));

  }
catch (SQLExeption se){ System.out.println(“Error:“ + se)}
```

## Fallstricke

- db2profile / db2cshrc *sourcen*!!
- Auf korrekten Classpath achten
  - Sind db2java.zip, die eigenen Klassen und TextIO erreichbar?
- Ist der Java-Compiler im Pfad erreichbar?





## Ziele

- static SQL verstehen
  - Build-Prozess / Art des Datenzugriffs
  - Host-Variablen
  - Vor- / Nachteile
- dynamic SQL verstehen
  - Build-Prozess / Art des Datenzugriffs
  - Parameter Marker
  - Vor- / Nachteile
- DB2 – Zugriff mit Java selbst programmieren

## Nächstes Praktikum

- Nächste Woche: Fällt aus
- Übernächste Woche: Fällt aus

- 14./15. Februar: Wiederholung ($\rightarrow$ Klausur)