

Teil X

Anwendungsprogrammierung

Anwendungsprogrammierung

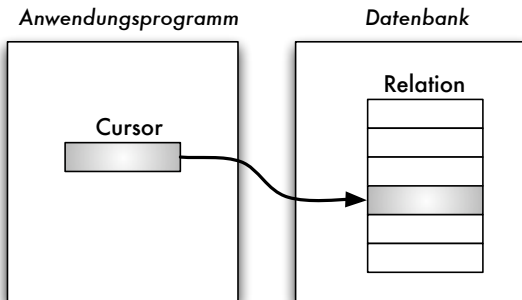
- 0 Programmiersprachenanbindung
- 1 JDBC
- 2 SQLJ
- 3 LINQ

Programmiersprachenanbindung

- Kopplungsarten:
 - ▶ prozedurale oder CALL-Schnittstellen (**call level interface**)
 - ★ Beispiele: SQL/CLI, ODBC, JDBC, ...
 - ▶ Einbettung einer DB-Sprache in Programmiersprachen
 - ★ statische Einbettung: **Vorübersetzer-Prinzip**
 - ↪ SQL-Anweisungen *zur Übersetzungszeit* festgelegt
 - ★ Beispiele: Embedded SQL, SQLJ
 - ★ dynamische Einbettung:
 - ↪ Konstruktion von SQL-Anweisungen zur Laufzeit
 - ▶ **Spracherweiterungen** und neue *Sprachentwicklungen*
 - ★ Beispiele: SQL/PSM, PL/SQL, Transact-SQL, PL/pgSQL

Cursor-Konzept

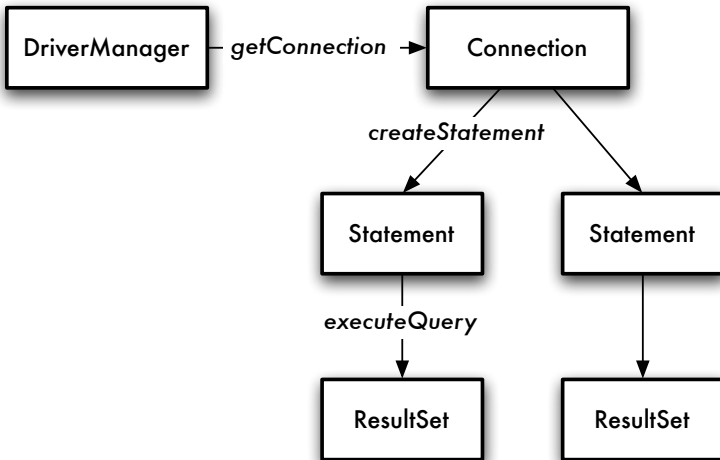
- **Cursor**: Iterator über Liste von Tupeln (Anfrageergebnis)



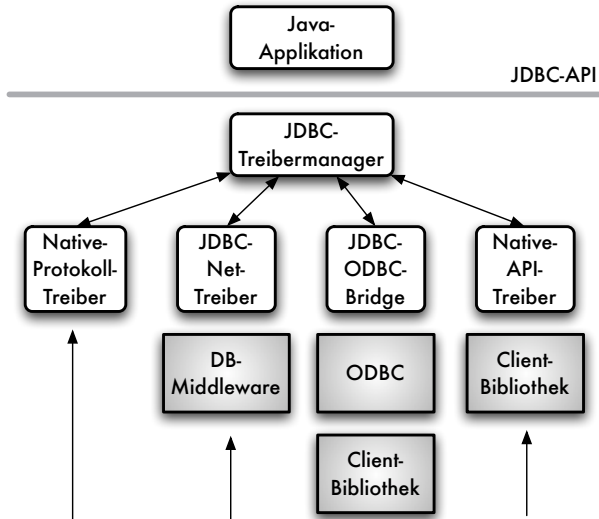
JDBC: Überblick

- Datenbankzugriffsschnittstelle für Java
- abstrakte, datenbankneutrale Schnittstelle
- vergleichbar mit ODBC
- Low-Level-API: direkte Nutzung von SQL
- Java-Package `java.sql`
 - ▶ `DriverManager`: Einstiegspunkt, Laden von Treibern
 - ▶ `Connection`: Datenbankverbindung
 - ▶ `Statement`: Ausführung von Anweisungen über eine Verbindung
 - ▶ `ResultSet`: verwaltet Ergebnisse einer Anfrage, Zugriff auf einzelne Spalten

JDBC: Struktur



JDBC: Treiberkonzept



JDBC: Ablauf

- 1 Aufbau einer Verbindung zur Datenbank
 - ▶ Angabe der Verbindungsinformationen
 - ▶ Auswahl und Laden des Treibers
- 2 Senden einer SQL-Anweisung
 - ▶ Definition der Anweisung
 - ▶ Belegung von Parametern
- 3 Verarbeiten der Anfrageergebnisse
 - ▶ Navigation über Ergebnisrelation
 - ▶ Zugriff auf Spalten

JDBC: Verbindungsaufbau

1 Treiber laden

```
Class.forName ("com.company.DBDriver");
```

2 Verbindung herstellen

```
Connection con;  
String url = "jdbc:subprotocol:datasource";  
con = DriverManager.getConnection  
    (url, "scott", "tiger");
```

JDBC-URL spezifiziert

- Datenquelle/Datenbank
- Verbindungsmechanismus (Protokoll, Server und Port)

JDBC: Anfrageausführung

1 Anweisungsobjekt (Statement) erzeugen

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

2 Anweisung ausführen

```
String query =  
    "select Name, Jahrgang from WEINE";  
ResultSet rset = stmt.executeQuery (query);
```

Klasse `java.sql.Statement`

- Ausführung von Anfragen (**SELECT**) mit `executeQuery`
- Ausführung von Änderungsanweisungen (**DELETE, INSERT, UPDATE**) mit `executeUpdate`

JDBC: Ergebnisverarbeitung

1 Navigation über Ergebnismenge (Cursor-Prinzip)

```
while (rset.next()) {  
    // Verarbeitung der einzelnen Tupel  
    ...  
}
```

2 Zugriff auf Spaltenwerte über getType-Methoden

- ▶ über Spaltenindex

```
String wname = rset.getString(1);
```

- ▶ über Spaltenname

```
String wname = rset.getString("Name");
```

JDBC: Fehlerbehandlung

- Fehlerbehandlung mittels Exception-Mechanismus
- SQLException für alle SQL- und DBMS-Fehler

```
try {  
    // Aufruf von JDBC-Methoden  
    ...  
} catch (SQLException exc) {  
    System.out.println("SQLException: "+  
        exc.getMessage());  
}
```

JDBC: Änderungsoperationen

- DDL- und DML-Operationen mittels `executeUpdate`
- liefert Anzahl der betroffenen Zeilen (für DML-Operationen)

```
Statement stmt = con.createStatement();  
int rows = stmt.executeUpdate(  
    "update WEINE set Preis = Preis * 1.1 " +  
    "where Jahrgang < 2000");
```

JDBC: Transaktionssteuerung

- Methoden von Connection

- ▶ `commit ()`
- ▶ `rollback ()`

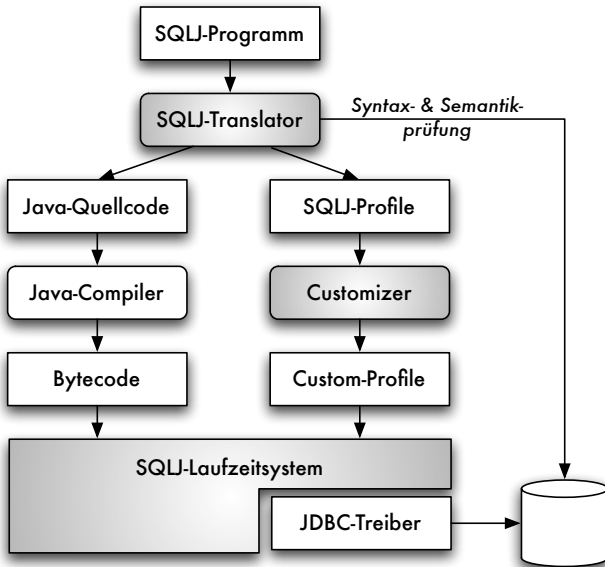
Auto-Commit-Modus

- ▶ implizites Commit nach jeder Anweisung
- ▶ Transaktion besteht nur aus einer Anweisung
- ▶ Umschalten mittels `setAutoCommit (boolean)`

SQLJ: Embedded SQL für Java

- Einbettung von SQL-Anweisungen in Java-Quelltext
- Vorübersetzung des erweiterten Quelltextes in echten Java-Code durch Translator **sqlj**
- Überprüfung der SQL-Anweisungen
 - ▶ korrekte Syntax
 - ▶ Übereinstimmung der Anweisungen mit DB-Schema
 - ▶ Typkompatibilität der für Datenaustausch genutzten Variablen
- Nutzung von JDBC-Treibern

SQLJ: Prinzip



SQLJ-Anweisungen

- Kennzeichnung durch #sql Deklarationen
- Klassendefinitionen für Iteratoren
- SQL-Anweisungen: Anfragen, DML- und DDL-Anweisungen

```
#sql { SQL-Operation };
```

- Beispiel:

```
#sql { insert into ERZEUGER (Weingut, Region) values  
      ( 'Wairau Hills', 'Marlborough' ) };
```

Host-Variablen

- Variablen einer Host-Sprache (hier Java), die in SQL-Anweisungen auftreten können
- Verwendung: Austausch von Daten zwischen Host-Sprache und SQL
- Kennzeichnung durch ":*variable*"
- Beispiel:

```
String name;  
int weinID = 4711;  
#sql { select Name into :name  
      from WEINE where WeinID = :weinID };  
System.out.println("Wein = " + name);
```

Iteratoren

1 Deklaration des Iterators

```
#sql public iterator WeinIter (String Name, String Weingut,  
    int Jahrgang);
```

2 Definition des Iteratorobjektes

```
WeinIter iter;
```

3 Ausführung der Anweisung

```
#sql iter = { select Name, Weingut, Jahrgang from WEINE };
```

4 Navigation

```
while (iter.next()) {  
    System.out.println(iter.Name() + " "+  
        iter.Weingut() + " "+ iter.Jahrgang());  
}
```

Dynamic SQL

- SQL-Statements als zur Laufzeit konstruierte Strings

```
exec sql begin declare section;  
    AnfrageString char(256) varying;  
exec sql end declare section;  
exec sql declare AnfrageObjekt statement;  
AnfrageString =  
    'delete from WEINE where WeinID = 4711';  
...  
exec sql prepare AnfrageObjekt from :AnfrageString;  
exec sql execute AnfrageObjekt;
```

Language Integrated Query (LINQ)

- Einbettung einer DB-Sprache (SQL) in eine Programmiersprache (C#)
- spezielle Klassenmethoden

```
IEnumerable<string> res = weine  
    .Where(w => w.Farbe = "Rot")  
    .Select(w => new { w.Name });
```

- eigene Sprachkonstrukte (ab C# 3.0)

```
IEnumerable<op> res = from w in weine  
    where w.Farbe = "Rot"  
    select new { w.Name };
```