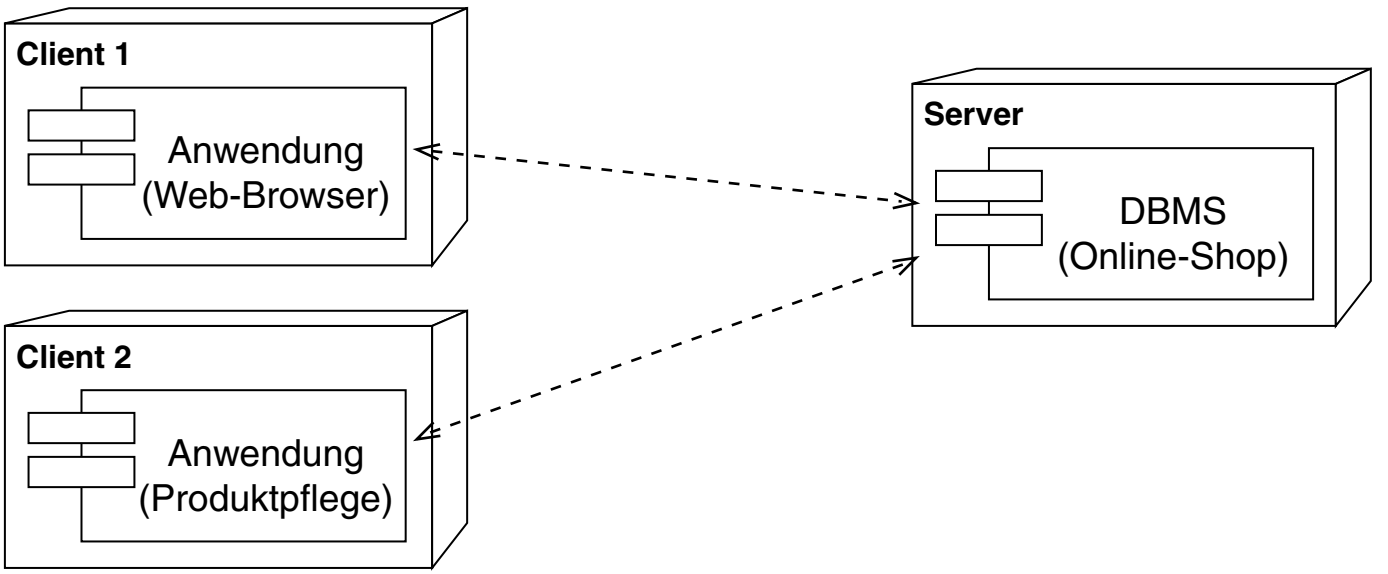
VI. Die relationale Datenbanksprache SQL

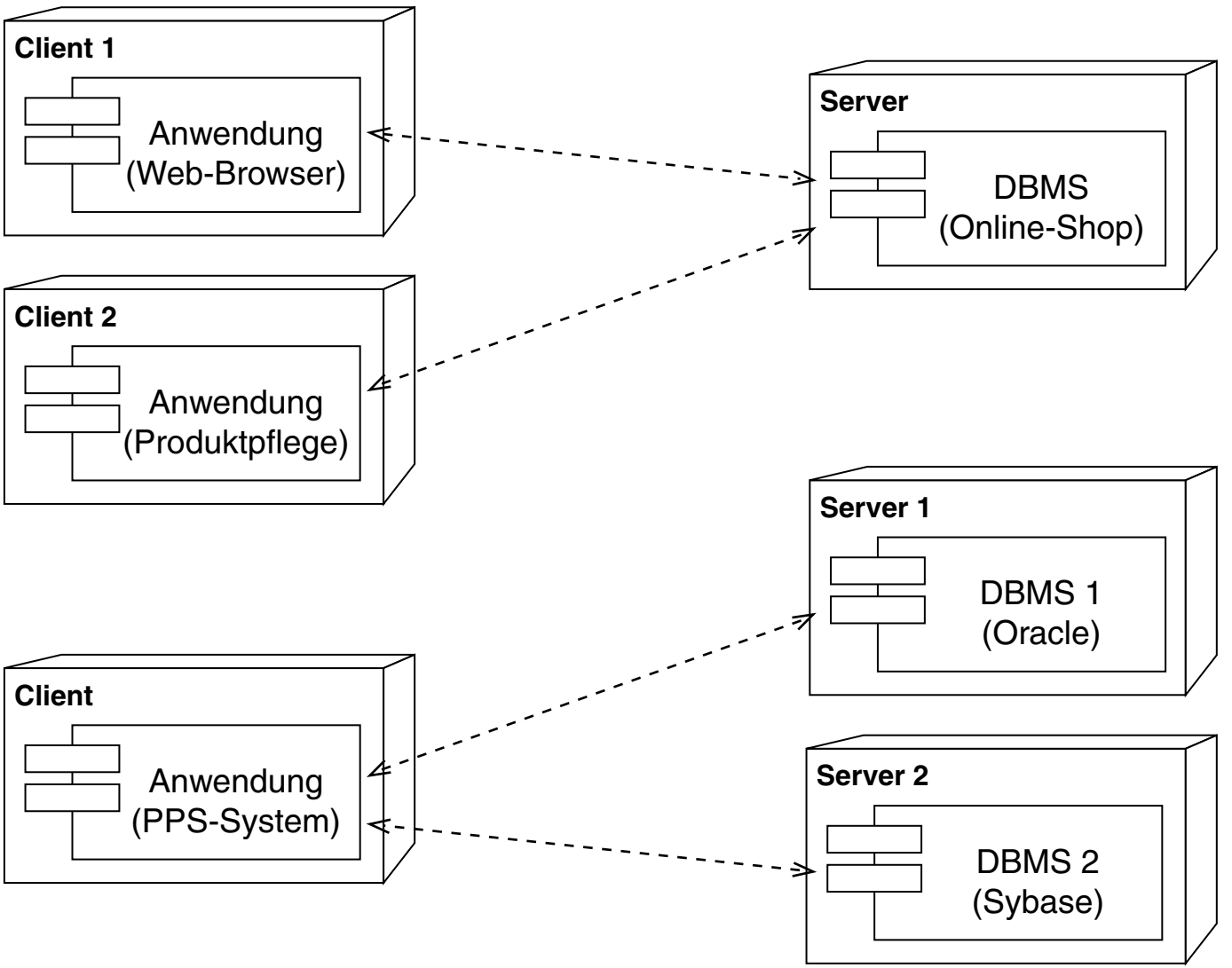
SQL vom Programm aus

Anwendungsszenarien



SQL vom Programm aus

Anwendungsszenarien



SQL vom Programm aus

Prinzipien zur DBMS-Anbindung

1. Anreicherung von Programmiersprachen durch Datenbankoperationen

2. Einbettung von Datenbanksprachen in Programmiersprachen
   - Prinzip: SQL-Statements werden im Programmquelltext ausgezeichnet
   - Vorteil: SQL-Statements lassen sich zur Übersetzungszeit prüfen und optimieren
   - Beispiel: Embedded SQL; die Realisierung für Java heißt SQLJ
   - Erweiterung: Dynamic SQL

SQL vom Programm aus

Prinzipien zur DBMS-Anbindung

1. Anreicherung von Programmiersprachen durch Datenbankoperationen

2. Einbettung von Datenbanksprachen in Programmiersprachen
   - Prinzip: SQL-Statements werden im Programmquelltext ausgezeichnet
   - Vorteil: SQL-Statements lassen sich zur Übersetzungszeit prüfen und optimieren
   - Beispiel: Embedded SQL; die Realisierung für Java heißt SQLJ
   - Erweiterung: Dynamic SQL

3. Programmierschnittstelle (Application Programming Interface, API)
   - Prinzip: SQL-Anweisungen werden als zur Programmausführungszeit generierbarer Text an das Datenbanksystem übergeben
   - Vorteil: hohe Flexibilität

Bemerkungen:

- Eine generelle Problematik bei Programmierschnittstellen ist die Verarbeitung von Tupelmengen, die als Ergebnis einer Anfrage geliefert werden. Eine Lösung hierzu bietet das *Cursor-Prinzip*, das in Java (SQLJ, JDBC) als Iterator-Objekt realisiert ist.
- Java Database Connectivity, JDBC, ist eine Programmierschnittstelle (API) der Java-Plattform, die einen einheitlichen Zugriff auf (relationale) Datenbanken verschiedener Hersteller bietet. [Wikipedia]
- Beispielanfrage in SQLJ (2. Prinzip):

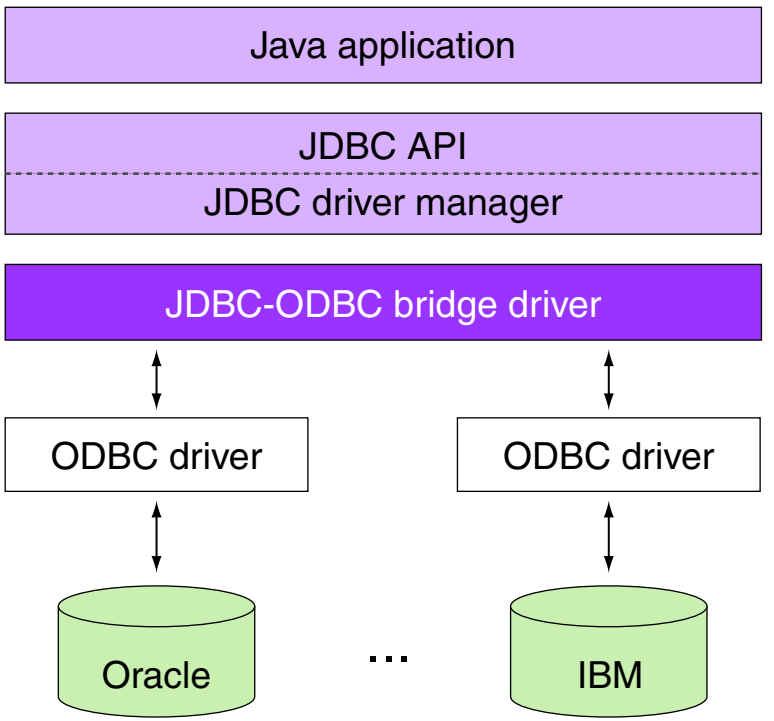
```
#sql [ctx] {
    SELECT max(Gebuehr)
    INTO :maxGebuehr
    FROM Kursgebuehr
};
```

- Entsprechende Anfrage in JDBC (3. Prinzip):

```
PreparedStatement stmt = myConnection.prepareStatement(
    "SELECT max(Gebuehr) FROM Kursgebuehr"
  );
ResultSet rs = statement.executeQuery();
int maxGebuehr = rs.getInt(1);
rs.close();
stmt.close();
```

SQL vom Programm aus

JDBC-API: Treibertypen



JDBC-Typ-1-Treiber. Übersetzung von JDBS-Aufrufen in ODBC-Aufrufe mittels eines sogenannten JDBC-ODBC-Bridge-Treibers. [Typ 2]

JDBC = Java Database Connectivity

ODBC = Open Database Connectivity

SQL vom Programm aus

JDBC-API: Vergleich mit ODBC

Gegenüberstellung wichtiger Anwendungsoperationen, ODBC-Funktionsnamen und der JDBC-Implementierung:

Operation (Anwendungssicht)	ODBC-Funktionsname	Implementierung in JDBC <class>:<method>
Verbindung zu DBMS aufbauen	SQLConnect	DriverManager: getConnection()
SQL-Anfrage ausführen	SQLExecute	Statement: executeQuery()
Ergebnisse abholen	SQLFetch	ResultSet: next()
Fehlermeldung abfragen	SQLError	SQLException
Transaktion deklarieren	SQLTransact	Connection: setAutoCommit()
Transaktion ausführen	SQLTransact	Connection: commit()
Transaktion zurücknehmen	SQLTransact	Connection: rollback()
Verbindung zu DBMS trennen	SQLDisconnect	Connection: close()

Bemerkungen:

- ODBC is a standard programming language middleware API for DBMS. ODBC accomplishes DBMS independence by using an ODBC driver as a translation layer between the application and the DBMS. The application uses ODBC functions through an ODBC driver manager with which it is linked, and the driver passes the query to the DBMS. An application that can use ODBC is referred to as “ODBC-compliant”.

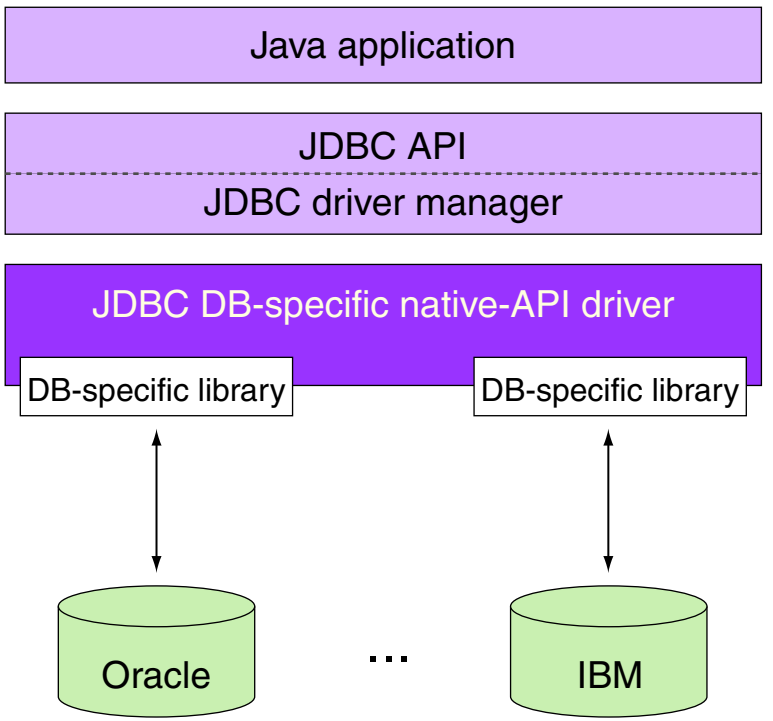
  Any ODBC-compliant application can access any DBMS for which a driver is installed. Drivers exist for all major DBMSs and even for text or CSV files. [Wikipedia]

- Ein ODBC-Treiber macht eine Datenquelle (z.B. eine MySQL-Datenbank oder eine Textdatei) zu einer ODBC-Datenquelle, die ODBC-Funktionsaufrufe versteht. ODBC ist von zwei Standpunkten aus zu betrachten:

  1. Aus Sicht der Anwendung, die in der Lage ist, mit einer ODBC-Datenquelle zu kommunizieren.
  2. Aus Sicht der Datenquelle, die ODBC-Anfragen verstehen und bedienen kann.

SQL vom Programm aus

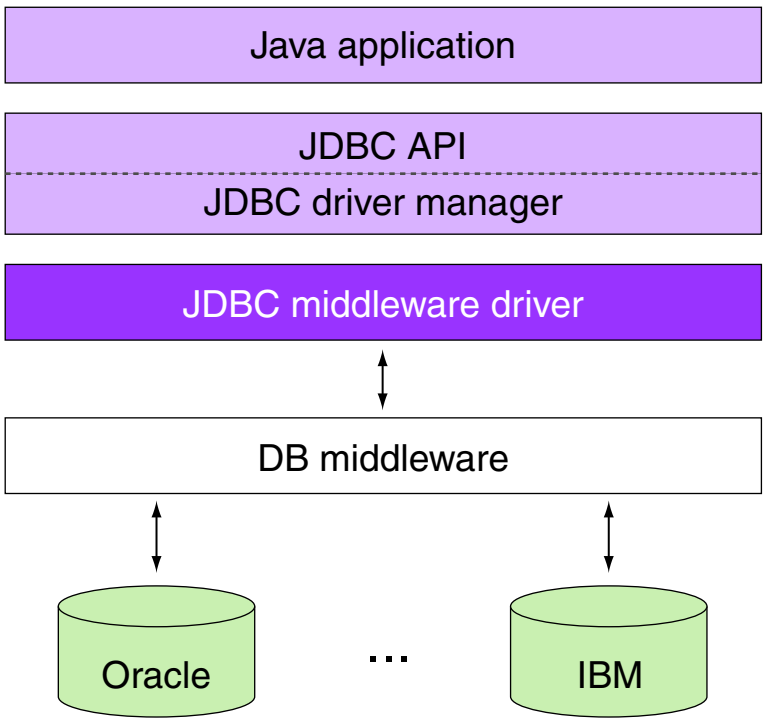
JDBC-API: Treibertypen (Fortsetzung)



JDBC-Typ-2-Treiber. Übersetzung von JDBS-Aufrufen in Aufrufe für einen Datenbankserver mittels einer plattform- und datenbankspezifischen Programmbibliothek. [Typ 1]

SQL vom Programm aus

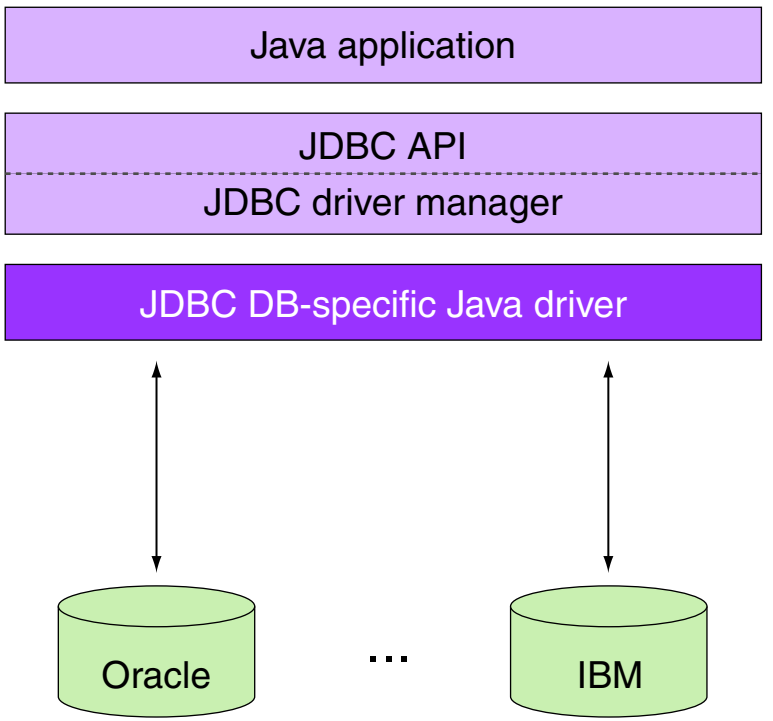
JDBC-API: Treibertypen (Fortsetzung)



JDBC-Typ-3-Treiber. Übersetzung von JDBC-Aufrufen in generische DBMS-Aufrufe sowie Übermittlung an die Middleware eines Anwendungsservers, welche die Aufrufe dann für spezifische Datenbankserver übersetzt.

SQL vom Programm aus

JDBC-API: Treibertypen (Fortsetzung)



JDBC-Typ-4-Treiber. Übersetzung von JDBC-Aufrufen für einen spezifischen Datenbankserver ohne Verwendung einer plattform- und datenbankspezifischen Programmbibliothek.

SQL vom Programm aus

JDBC-API: Beispiel Typ-4-Treiber

```
package jdbc;
import java.sql.*;

public class JdbcDemo {

 public JdbcDemo() throws ClassNotFoundException {
  // Requires MySQL Connector/J.
  Class.forName("com.mysql.jdbc.Driver");
 }

 public ResultSet submitQuery(
         String url, String user, String pass, String query)
         throws SQLException {
  Connection connection = DriverManager.getConnection(url, user, pass);
  Statement statement = connection.createStatement();
  ResultSet result = statement.executeQuery(query);
  return result;
 }
```

Java SE 10 / JDK 10 Dokumentation:

- `java.sql`-Package [Javadoc]
- `DriverManager`-Klasse [Javadoc]

SQL vom Programm aus

JDBC-API: Beispiel Typ-4-Treiber (Fortsetzung)

```
public static void main(String[] args) throws Exception {
   JdbcDemo demo = new JdbcDemo();
   String db     = "mitarbeiterdb";
   String url    = "jdbc:mysql://pcstein.medien.uni-weimar.de/" + db;
   String user   = "stein";
   String pass   = "";
   String query  = "select Name, ChefPersNr "
                 + "from mitarbeiter "
                 + "where ChefPersNr < 8000";

   ResultSet result = demo.submitQuery(url, user, pass, query);

   while (result.next()) {
    String name = result.getString("Name");
    int chefPersNr = result.getInt("ChefPersNr");
    System.out.println(name + ' ' + chefPersNr);
  }
}
```

SQL vom Programm aus

JDBC-API: Beispiel Typ-4-Treiber (Fortsetzung)

```
public static void main(String[] args) throws Exception {
   JdbcDemo demo = new JdbcDemo();
   String db     = "mitarbeiterdb";
   String url    = "jdbc:mysql://pcstein.medien.uni-weimar.de/" + db;
   String user   = "stein";
   String pass   = "";
   String query  = "select Name, ChefPersNr "
                 + "from mitarbeiter "
                 + "where ChefPersNr < 8000";

   ResultSet result = demo.submitQuery(url, user, pass, query);

   while (result.next()) {
    String name = result.getString("Name");
    int chefPersNr = result.getInt("ChefPersNr");
    System.out.println(name + ' ' + chefPersNr);
  }
}
```

```
[stein@pcstein]$ javac jdbc/JdbcDemo.java
[stein@pcstein]$ java -cp .:mysql-connector-java.jar jdbc.JdbcDemo
Smith 3334
```

SQL vom Programm aus

MySQL Version 5.x [download]

Besonderheiten:

- `create table ...`
  `( ... ) type=InnoDB;`

Einschränkungen (u.a.):

- keine Deklaration von Domains
- Update-Klausel darf keinen SFW-Block enthalten