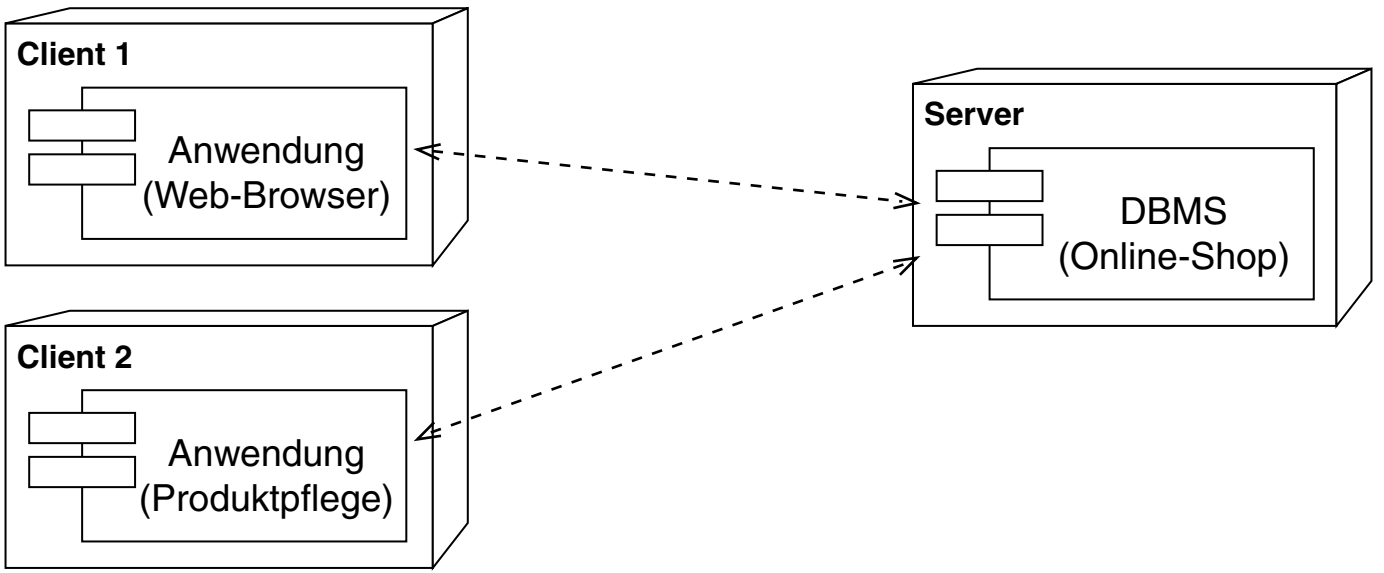
Kapitel DB:V

V. Die relationale Datenbanksprache SQL

- Einführung
- SQL als Datenanfragesprache
- SQL als Datendefinitionssprache
- SQL als Datenmanipulationssprache
- Sichten
- SQL vom Programm aus

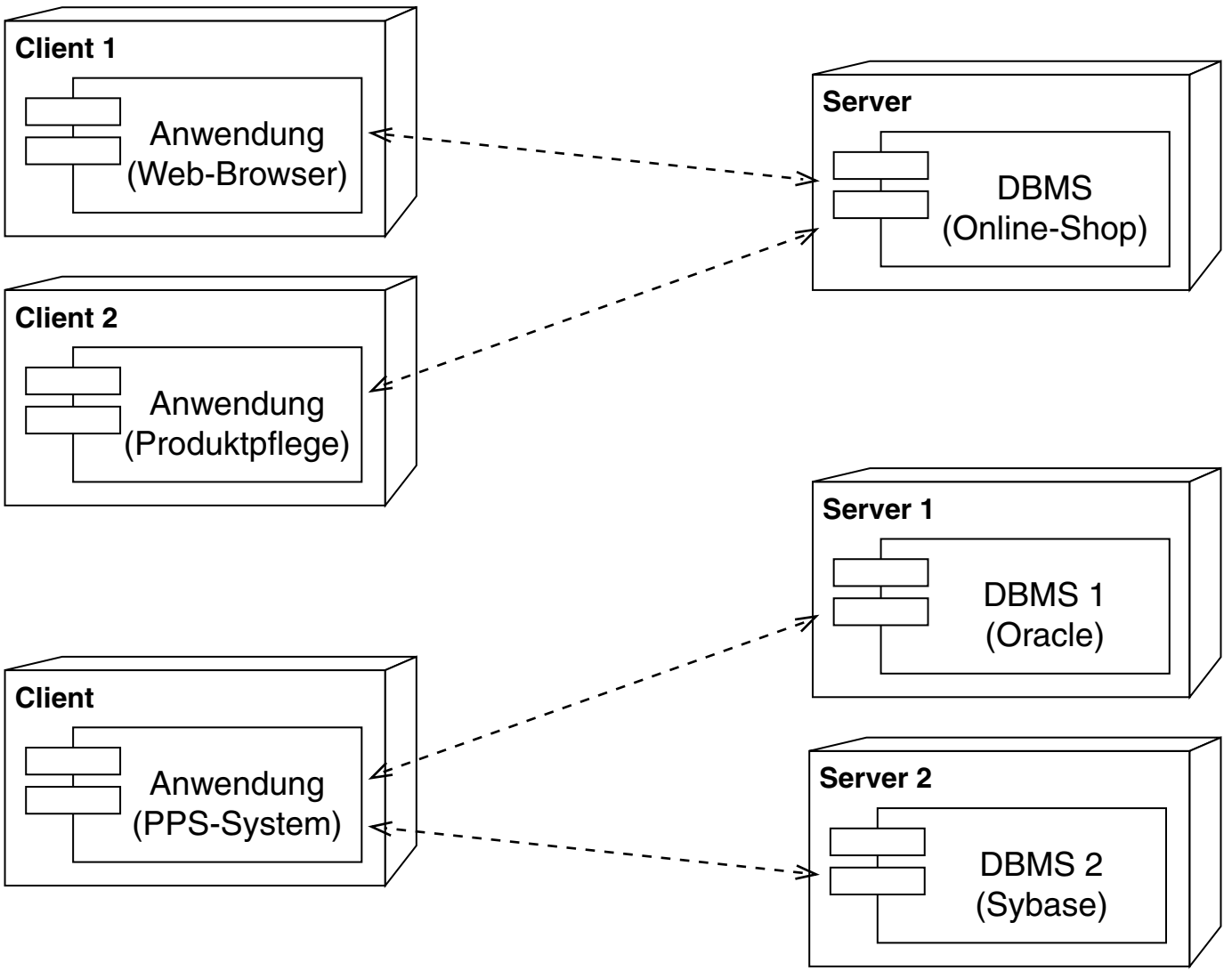
SQL vom Programm aus

Anwendungsszenarien



SQL vom Programm aus

Anwendungsszenarien



SQL vom Programm aus

Prinzipien zur DBMS-Anbindung

1. Anreicherung von Programmiersprachen durch Datenbankoperationen

2. Einbettung von SQL in Programmiersprachen: Embedded SQL
   - Prinzip: SQL-Statements werden im Programmquelltext ausgezeichnet
   - Vorteil: SQL-Statements lassen sich zur Programmübersetzungszeit prüfen und optimieren
   - Realisierungen: SQLJ (Java), Pro*C (C, C++)

SQL vom Programm aus

Prinzipien zur DBMS-Anbindung

1. Anreicherung von Programmiersprachen durch Datenbankoperationen

2. Einbettung von SQL in Programmiersprachen: Embedded SQL
   - Prinzip: SQL-Statements werden im Programmquelltext ausgezeichnet
   - Vorteil: SQL-Statements lassen sich zur Programmübersetzungszeit prüfen und optimieren
   - Realisierungen: SQLJ (Java), Pro*C (C, C++)

3. Programmierschnittstelle (Application Programming Interface, API)
   - Prinzip: SQL-Anweisungen werden als zur Programmausführungszeit generierbarer Text an das Datenbanksystem übergeben [Beispiel: Java, Python]
   - Vorteil: hohe Flexibilität
   - Realisierungen: ODBC (plattformunabhängig), JDBC (Java)

Bemerkungen:

- Beispielanfrage in SQLJ (Prinzip 2):

```
int maxGebuehr = 0;
#sql [ctx] {
    SELECT max(Gebuehr)
    INTO :maxGebuehr
    FROM Kursgebuehr
};
```

- Obige Beispielanfrage in JDBC (Prinzip 3):

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

Bemerkungen: (Fortsetzung)

- Open Database Connectivity, 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 und ermöglicht einer Anwendung den Zugriff hierauf:

  1. Client-Sicht. Eine Anwendung kann Anfragen an ODBC-Datenquellen senden.
  2. Server-Sicht. Eine Datenquelle kann ODBC-Anfragen beantworten.

- Java Database Connectivity, JDBC, ist eine Programmierschnittstelle (API) der Java-Plattform, die einen einheitlichen Zugriff auf relationale Datenbanken verschiedener Hersteller bietet. [Wikipedia: JDBC, JDBC-Treiber]

- Programmierschnittstellen verwenden sogenannte “Cursor”, um über Tupelmengen zu iterieren, die als Ergebnis einer Anfrage geliefert werden. Die entsprechende Umsetzung in JDBC geschieht durch ein Iterator-Objekt.

SQL vom Programm aus

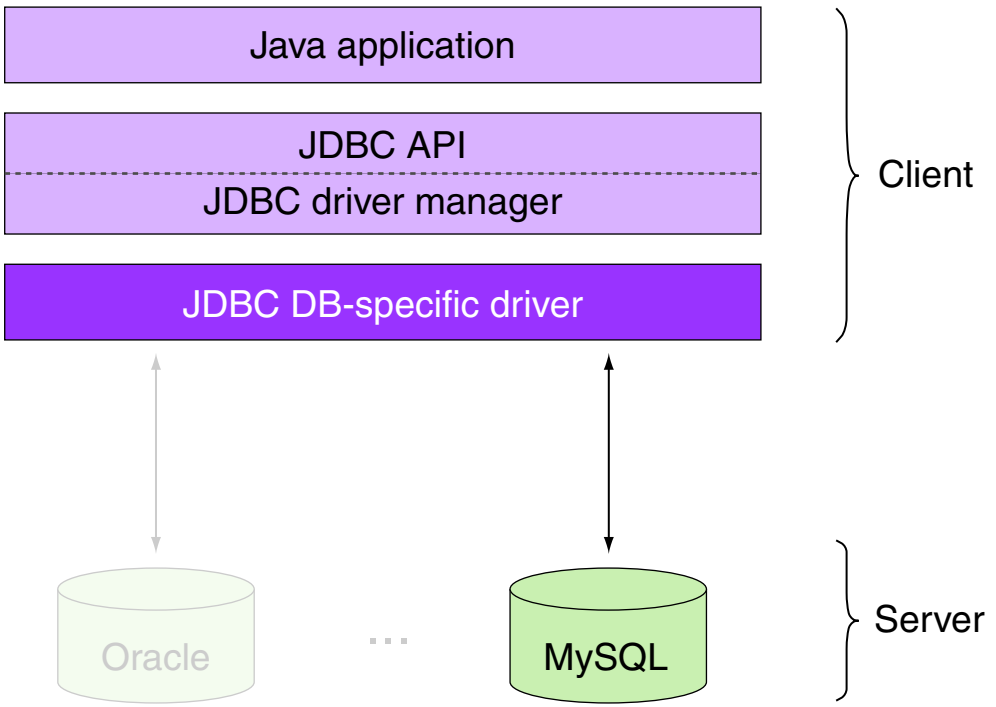
ODBC versus JDBC

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()`

SQL vom Programm aus

JDBC Native Protocol Driver [Python + SQLite]



Übersetzung von JDBC-Aufrufen für einen MySQL-Datenbankserver ohne Verwendung einer datenbankspezifischen Programmbibliothek.

[Wikipedia: Vorteile / Nachteile]

SQL vom Programm aus

JDBC-API: Beispiel

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

public class JdbcDemo {
  public static void main(String[] args) throws Exception {
   JdbcDemo demo = new JdbcDemo();
   String db     = "mitarbeiterdb";
   String url    = "jdbc:mysql://localhost/" + db;  // "mysql" or "sqlite"
   String user   = "db-user";
   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);
   }
 }

 public ResultSet submitQuery ...
}
```

SQL vom Programm aus

JDBC-API: Beispiel (Fortsetzung)

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

public class JdbcDemo {
  public static void main ...

  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;
  }
}
```

SQL vom Programm aus

JDBC-API: Beispiel (Fortsetzung)

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

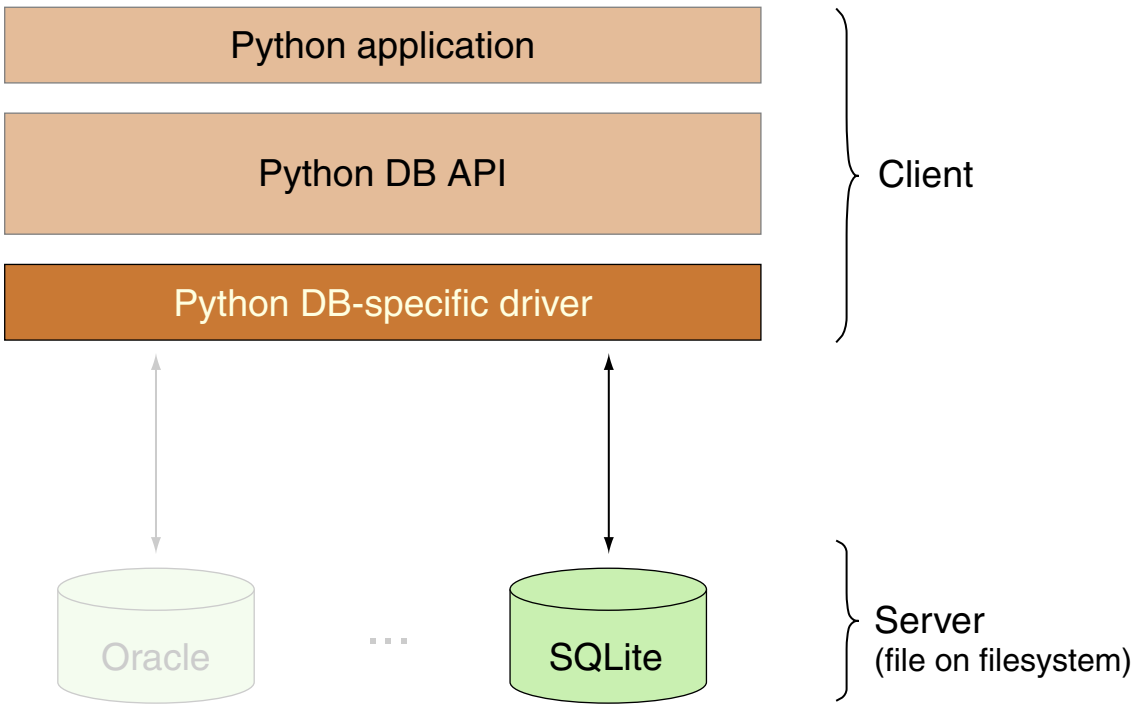
public class JdbcDemo {
  public static void main ...

  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;
  }
}
```

```
user@localhost:~$ javac jdbc/JdbcDemo.java
user@localhost:~$ java -cp .:mysql-connector-java.jar jdbc.JdbcDemo
user@localhost:~$ java -cp .:sqlite-connector-java.jar jdbc.JdbcDemo
Smith 3334
```

SQL vom Programm aus

Python-DB Native Protocol Driver [Java + MySQL]



Übersetzung von Python-DB-Aufrufen für einen SQLite-Datenbankserver (eine Datei auf dem Dateisystem) ohne Verwendung einer datenbankspezifischen Programmbibliothek.

SQL vom Programm aus

Python-DB-API: Beispiel

```
import sqlite3

def submit_query(db: str, query: str):
    connection = sqlite3.connect(db)
    cursor = connection.cursor()
    cursor.execute(query)
    connection.commit()
    result = cursor.fetchall()
    connection.close()
    return result

def main():
    db    = "mitarbeiterdb.db"
    query = "select Name, ChefPersNr " \
            "from mitarbeiter " \
            "where ChefPersNr < 8000"

    result = submit_query(db, query)

    for row in result:
        name = row[0]
        chef_pers_nr = row[1]

        print(str(name) + " " + str(chef_pers_nr))
```

SQL vom Programm aus

MySQL Version 8.x

- Community Downloads [MySQL]
- Storage Engines:
  - ```
    create table ...
    ( ... ) type = InnoDB;
    ```
  - Engines für besondere Tabellentypen [MySQL]
- Einschränkungen (u.a.) :
  - keine Deklaration von Domains
  - Update-Klausel darf keinen SFW-Block enthalten
    ```