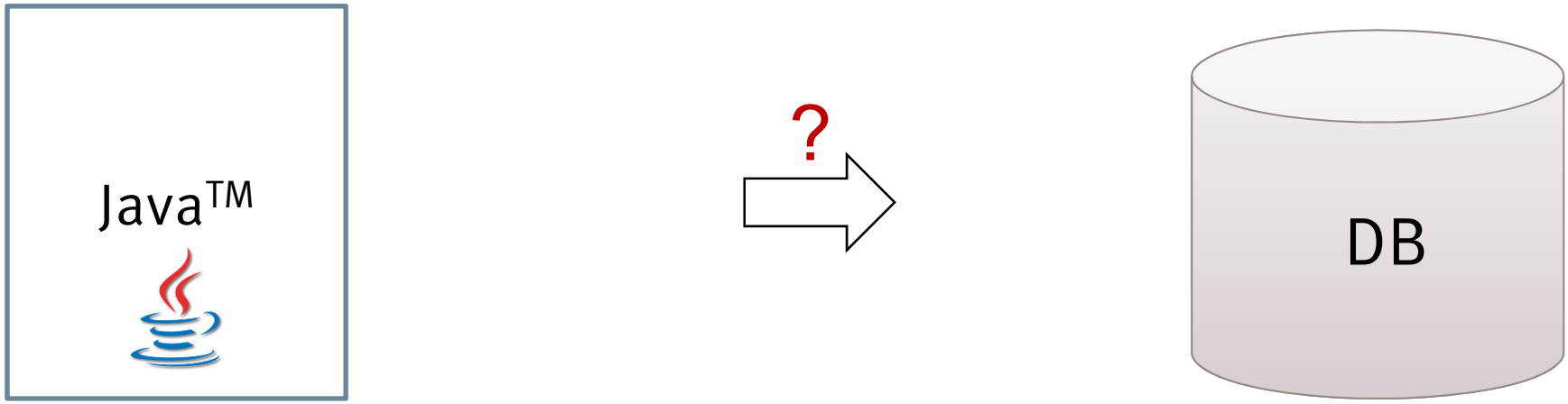
JDBC

Die JDBC-Verbindung

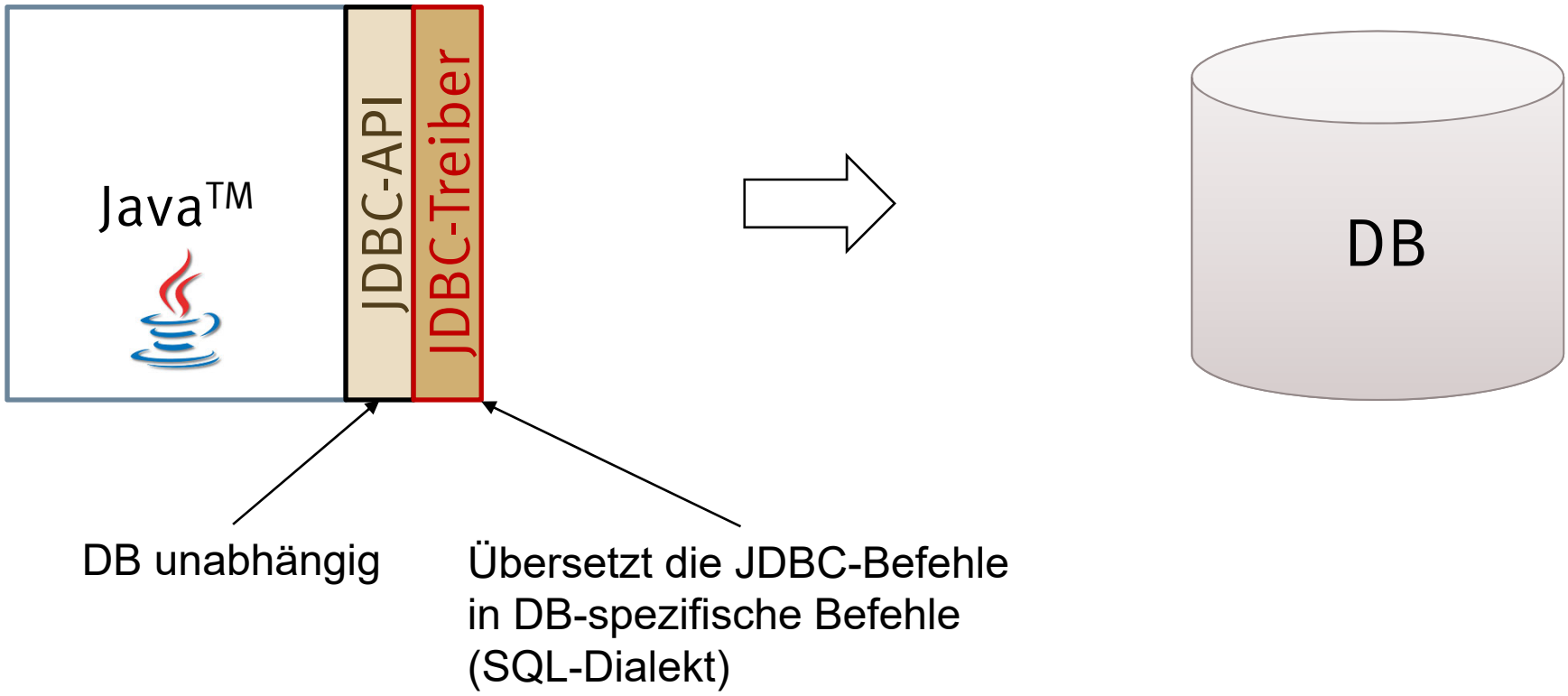
Fachhochschule
Dortmund
University of Applied Sciences

Fragestellung

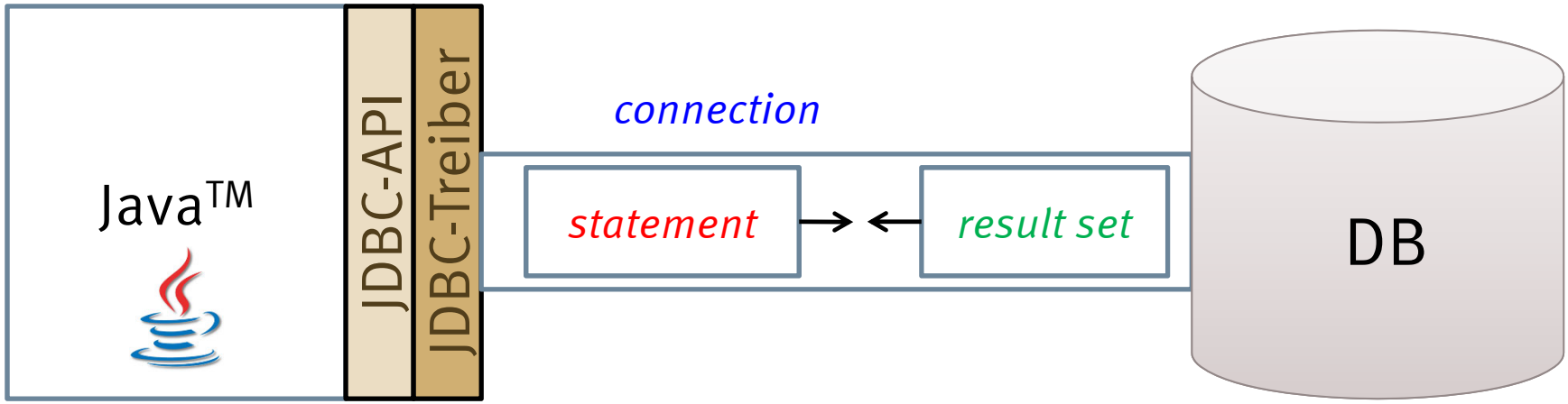


Wie kann aus einem Javaprogramm auf die Datenbank zugegriffen werden?

Lösungsidee: **J**ava **D**ata**b**ase **C**onnectivity



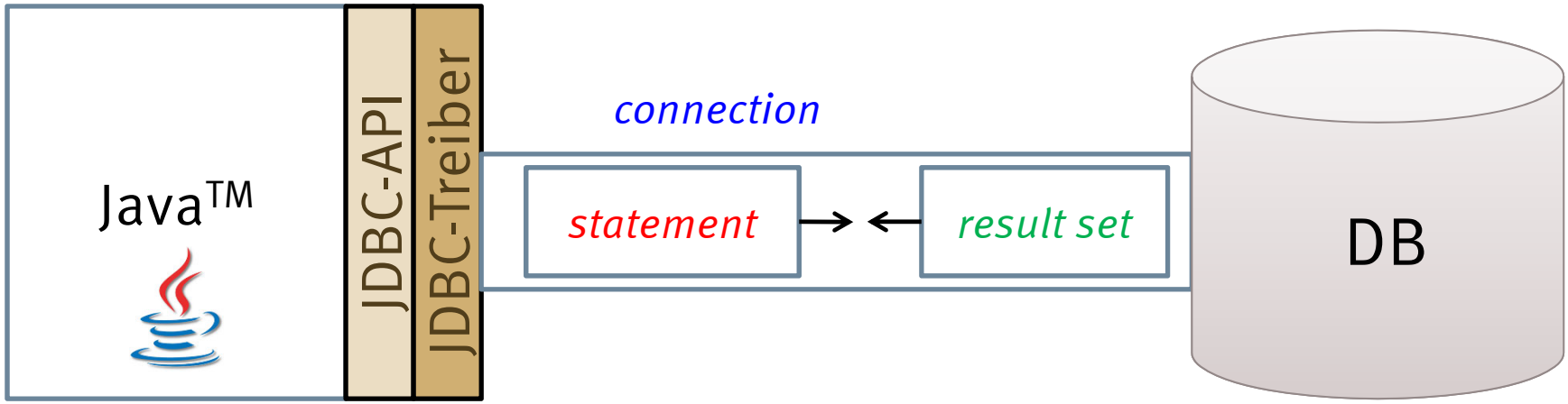
Prinzip JDBC - Connection



Schritte

- Aufbau der Verbindung (*connection*) zur DB
- Senden der SQL-Anweisung (*statement*)
- Verarbeiten der Anfrageergebnisse (result set)
- Schließen der Verbindung (*connection*) zur DB

Prinzip JDBC - Connection



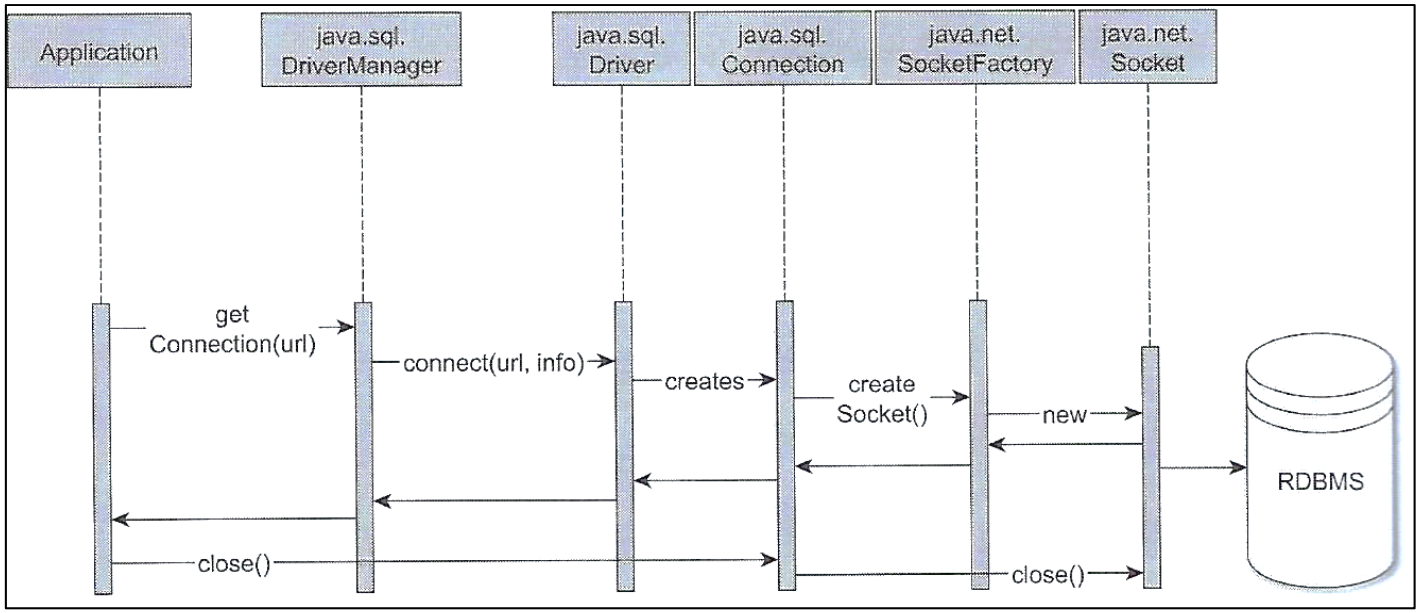


Abb. aus V. Mihalcea, High-Performance Java Persistence, S.14

Connection Pool

Eine Menge von Pooled Connections werden durch den Connection Pool verfügbar gemacht, die bei Bedarf aus dem Pool angefordert und nach der Verwendung wieder in den Pool zurückgestellt werden. Alle Verbindungen im Connection Pool sind dabei mit derselben Datenbank verbunden.

Beispiel:
Änderung der maximalen Anzahl von Connections auf 200
(Oracle Express Ed.):

```
ALTER SYSTEM
SET processes=200
scope=spfile;
```

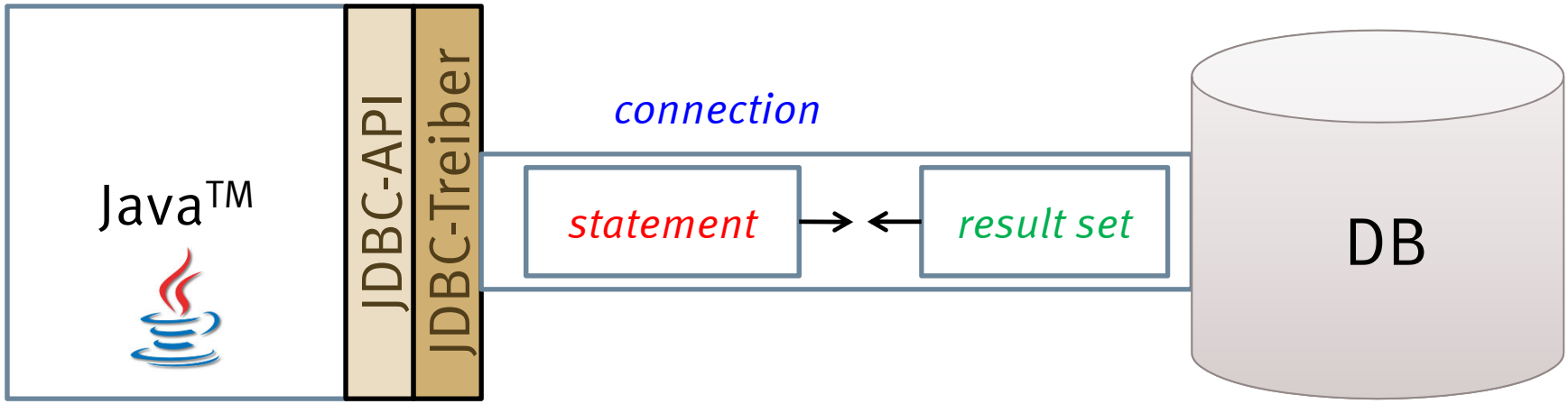
Connection-Pooling

Eine Pooled Connection ist eine wiederverwendbare Datenbankverbindung. Sie wird beim Aufruf von close() nicht physisch geschlossen, sondern lediglich freigegeben und bleibt für nachfolgende Client-Zugriffe erhalten. In einem Connection Pool werden diese gespeichert.

Connection Pooling durch den Oracle-JDBC-Treiber

Programmlauf	1. Verbindung	bei Wiederverwendung (Mittelwert aus N=10)
1.	404 ms	16 ms
2.	431 ms	16 ms
3.	488 ms	20 ms
4.	403 ms	16 ms
5.	536 ms	15 ms

Zusammenfassung



Schritte

- Aufbau der Verbindung (*connection*) zur DB
- Senden der SQL-Anweisung (*statement*)
- Verarbeiten der Anfrageergebnisse (result set)
- Schließen der Verbindung (*connection*) zur DB

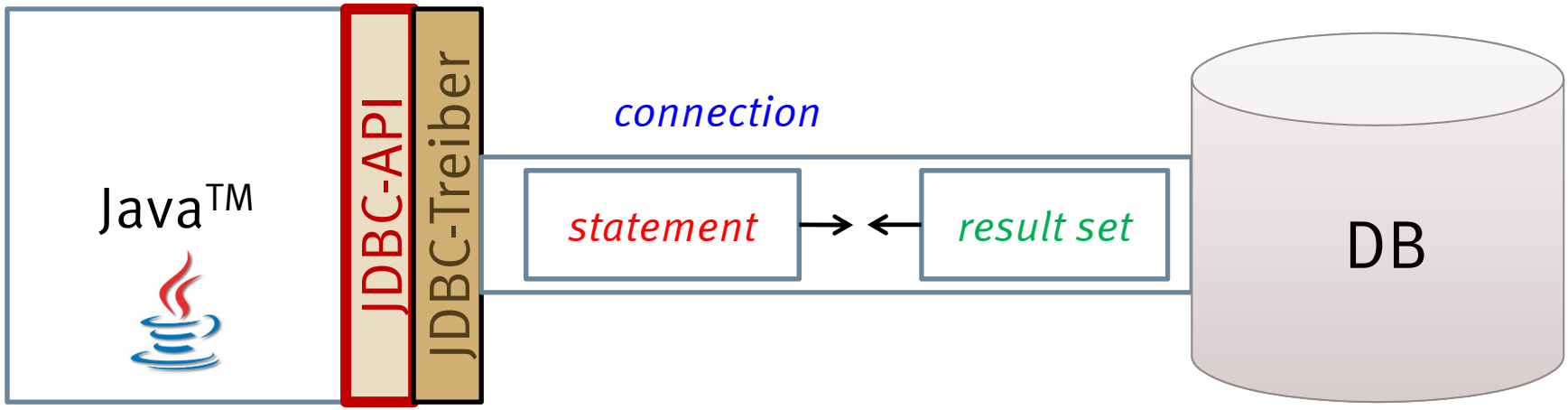
we focus on students

JDBC

Die API

Fachhochschule Dortmund
University of Applied Sciences

Lösungsansatz

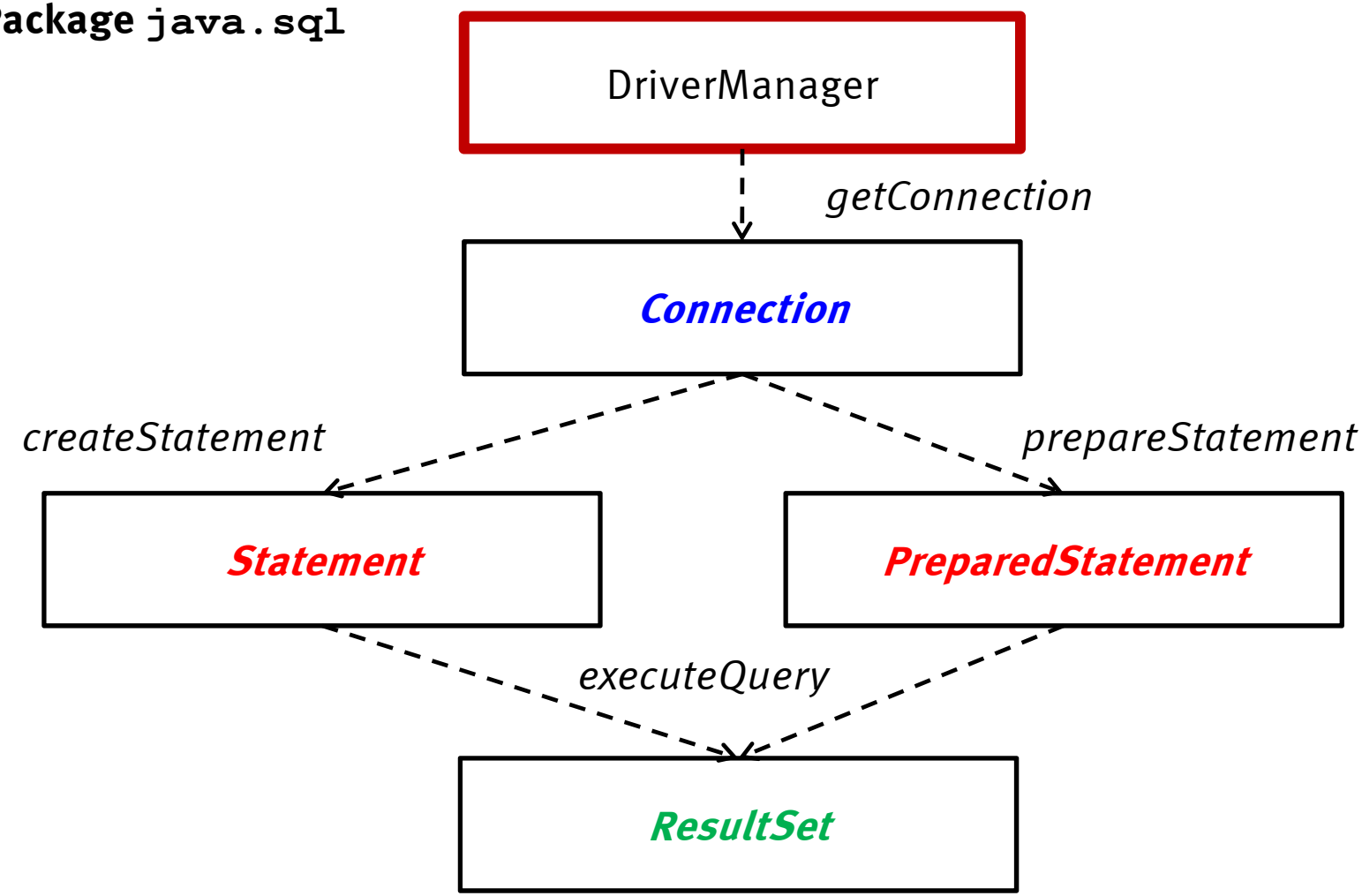


Schritte

- Aufbau der Verbindung (*connection*) zur DB
- Senden der SQL-Anweisung (*statement*)
- Verarbeiten der Anfrageergebnisse (result set)
- Schließen der Verbindung (*connection*) zur DB

Wie wird die JDBC-API verwendet?

Package `java.sql`



DriverManager

DriverManager

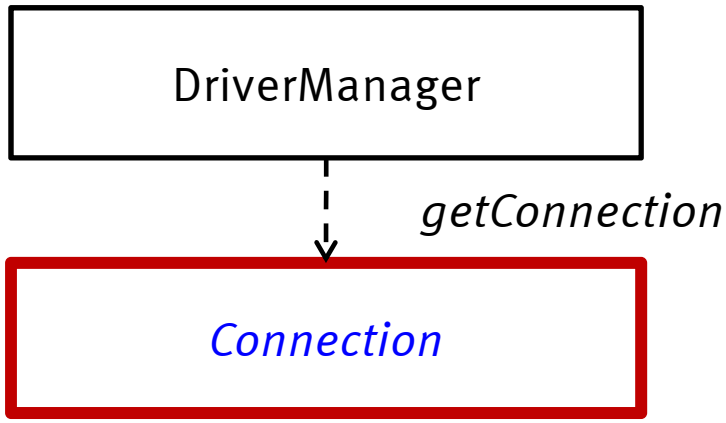
- JDBC-Treiber initialisieren

  ```
  Class.forName("oracle.jdbc.driver.OracleDriver");
  ```

- Registrierung des Treiber (erfolgt ab Java 6 automatisch)

  ```
  DriverManager.registerDriver (new oracle.jdbc.driver.OracleDriver());
  ```

Verbindungsaufbau



Die Klasse Connection bietet Methoden um:

- Verbindung zur Datenbank herzustellen und zu schließen und die Verbindungseigenschaften abzufragen
- SQL-Anweisungen zu kapseln und an das DBMS zu senden
- auf den Katalog (Data Dictionary) zuzugreifen
- Transaktionen zu steuern (commit, rollback)

Verbindungsaufbau

- Aufbau einer URL (Unified Ressource String) (Oracle)

```
url="jdbc:oracle:thin:@172.22.112.100:1521:fbpool";
```

Treiber | IP-Adresse | Port | Datenbankschema (oracle: SID)

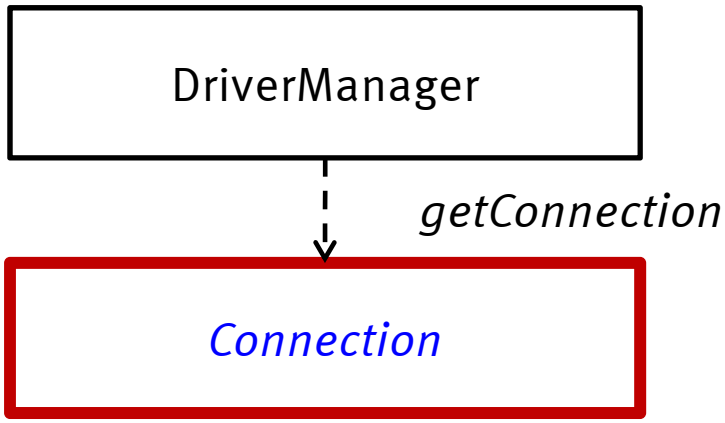
- Verbindung öffnen

```
Connection con=
        DriverManager.getConnection(url,“userid“,“password“);
```

- Verbindung schließen

```
con.close();
```

Verbindungsaufbau



Deklarierung von Ressourcen (z.B. Connection)

```
70          try (Connection con = DriverManager.getConnection(url,"saatzbh", "oracle")) {
71
72
73
74
75          } catch (SQLException ex1) {
76              printSQLException(ex1);
77          }
```

ab Java 7

Ressourcen werden automatisch freigegeben

Statement

- Statement

```
String      sqlString=  "UPDATE Kunde "
                        + "SET Nachname='Curie', Geburtsdatum=NULL"
                        + "WHERE Kundennummer=8365";
Statement stmt = con.createStatement();

int anzahl = stmt.executeUpdate(sqlString);
```

- PreparedStatement

Hier wird das SQL-Statement zunächst vorbereitet (compiliert). Dies beschleunigt die Ausführung mehrerer gleichartiger Statements hintereinander.

```
String      sqlString=  "UPDATE Kunde "
                        + "SET Nachname=?, Geburtsdatum = ? "
                        + "WHERE Kundennummer=?";
PreparedStatement stmt = con.prepareStatement(sqlString);
stmt.setString(1,'Curie');
stmt.setNull(2,"java.sql.Date");
stmt.setInt(3,8365);

int anzahl = stmt.executeUpdate();
```

```
------------- ERGEBNIS ----------------
JDBC              1. Ausführng: 612 ms
JDBC PS           1. Ausführng: 123 ms
```

ResultSet

- SQL

```
SELECT Kundennummer, Nachname, Anrede
FROM Kunde
```

Kundennummer 1	Nachname 2	Anrede 3
2310	Meitner	Frau
7562	Einstein	Herr
8365	Curie	Frau
8523	Dekanat Informatik	NULL

- JDBC

```
ResultSet rs = stmt.executeQuery(sqlString);
while (rs.next()){
        String knr = rs.getString(1);
        // interne Datenkonvertierung: Integer.toString(rs.getInt(1));
        String name   = rs.getString(2);
        String anrede = rs.getString(3);
        System.out.println(knr + ":"+anrede + " "+ name);
}
```

Nr. der Spalte

Zeilenweise Verarbeitung

ResultSet - Metadaten

- Beschreibung des ResultSets – ResultSetMetaData
- getColumnCount()
- getColumnName(nr)
- getColumnType(nr)
- getPrecision(nr)
- isNullable(nr)
- isAutoIncrement(nr)
- isWriteable(nr)
- isCaseSensitive(nr)
- ...

Spalte 1 Kundennr.	Spalte 2 Nachname	Spalte 3 Anrede
2310	Meitner	Frau
8523	Dekanat	NULL

ResultSet - Methoden

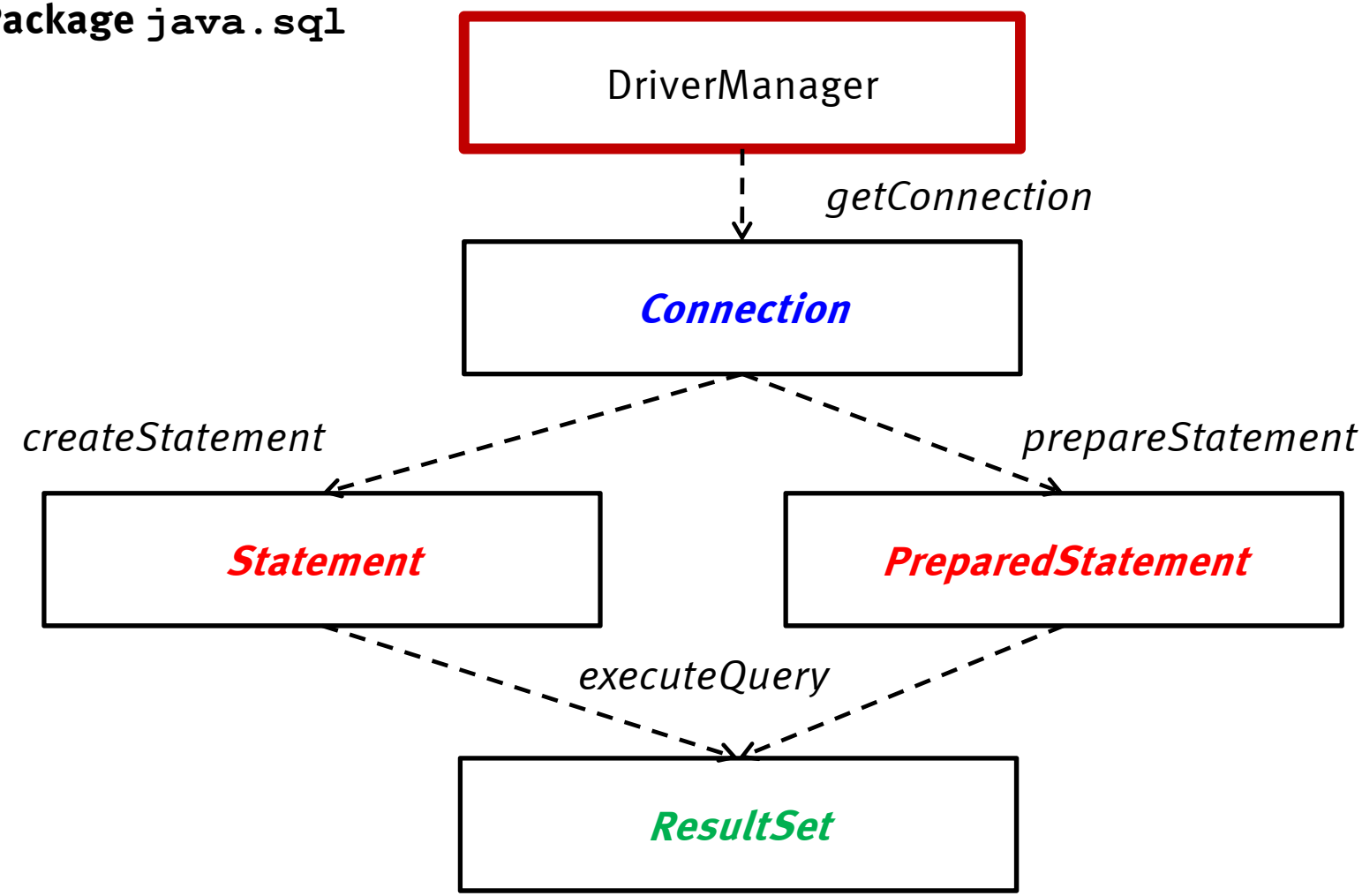
- ResultSet-Methoden
  - next()
  - gettype(nr)
  - gettype(name)
  - wasNull()
  - getMetaData()
- Erweiterungen
  - absolute(nr)
  - afterLast()
  - last()
  - first()
  - beforeFirst()
  - previous()
  - isAfterLast()
  - isLast()
  - ...

beforeFirst() →
next() ↻

Spalte 1	Spalte 2	Spalte 3
2310	Meitner	Frau
7562	Einstein	Herr
8365	Curie	Frau
8523	Dekanat	NULL

Zusammenfassung

Package `java.sql`



JDBC

Datentypen

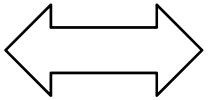
Fachhochschule
Dortmund
University of Applied Sciences

Fragestellung

Java 

SQL

Kunde
int id
String anrede
Date geburtsdatum



JDBC
Schnittstelle

```
CREATE TABLE Kunde(
Kundennr   Integer PRIMARY KEY,
Anrede VARCHAR(4),
Geburstdatum DATE
)
```

Wie werden unterschiedliche Datentypen aufeinander abgebildet?

- NULL-Werte
- Unterschiedliche Längen von Zeichenketten

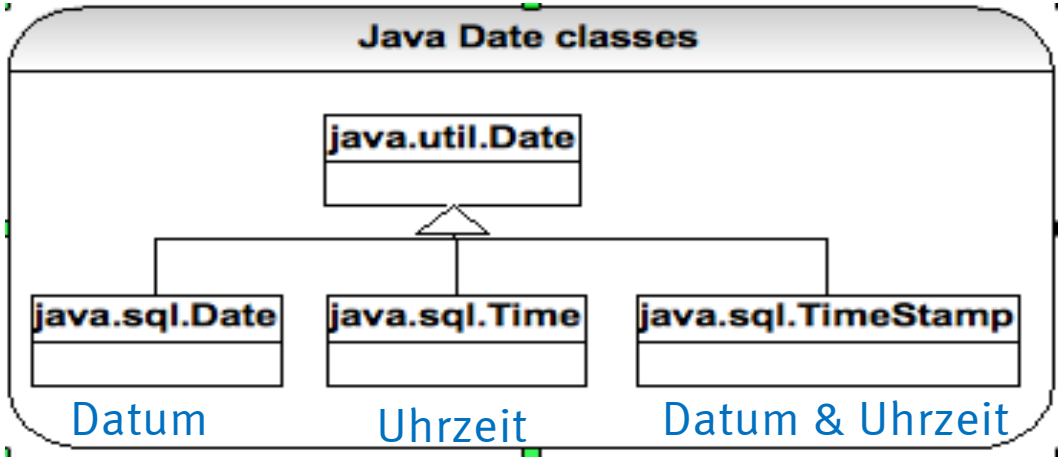
Lösungsidee

Das Package java.sql stellt Klassen bereit, welche die Java-Datentypen auf SQL-Datentypen abbilden.

Java	JDBC (java.sql.Types)
String	CHAR VARCHAR
java.math. BigDecimal	NUMERIC DECIMAL
Boolean	BIT
Int	INTEGER
double	FLOAT DOUBLE
java.sql.Date	DATE
java.sql.Time	TIME
java.sql.Timestamp	TIMESTAMP
java.sql.Clob	CLOB

Vorsicht Falle!

JAVA



SQL

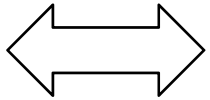
SQL Datentypen	MySQL	Oracle
Datum und Uhrzeit	DATETIME TIMESTAMP	**DATE** TIMESTAMP
Datum MySQL: YYYY-MM-DD Oracle: DD.MM.YYYY	**DATE**	
Uhrzeit	TIME	
Zeitintervalle	INTERVAL	

Beispiel

Java 

SQL

Kunde
int id String anrede Date geburtsdatum



JDBC Schnittstelle

```
CREATE TABLE Kunde(
Id              INTEGER PRIMARY KEY,
Anrede          VARCHAR(4),
Geburstdatum    DATE
)
```

```
int      id= 2310;
String anrede= „Frau“;
```

Id	Anrede	Geburts-datum
2310	Frau	NULL

Beispiel

- Beispiel:

Id	Anrede	Geburts-datum
2310	Frau	NULL

- SQL

```
INSERT INTO  Kunde (Kundennummer, Anrede, Geburtsdatum)
VALUES       (2311, 'Frau', NULL)
```

- JDBC

```
String      sqlString= " INSERT INTO  Kunde "
                       + "(Kundennummer, Anrede, Geburtsdatum)"
                       + " VALUES (?, ?, ?)";
PreparedStatement stmt = con.prepareStatement(sqlString);
stmt.setInt          (1, 2310);
stmt.setString       (2, "Frau");
stmt.setNull         (3,"java.sql.Date");

int anzahl = stmt.executeUpdate();
```

Beispiel

- Beispiel:

Id	Anrede	Geburts-datum
2310	Fräu	NULL

- SQL

```
INSERT INTO  Kunde (Kundennummer, Anrede, Geburtsdatum)
VALUES       (2311, 'Fräu', NULL)
```

- JDBC

```
String      sqlString= " INSERT INTO  Kunde "
                       + "(Kundennummer, Geschlecht, Geburtsdatum)"
                       + " VALUES (?, ?, ?)";
PreparedStatement stmt = con.prepareStatement(sqlString);
stmt.setInt          (1, 2310);
stmt.setString       (2, "Fräulein");
stmt.setNull         (3,"java.sql.Date");

int anzahl = stmt.executeUpdate();
```

Wert wird auf „Fräu“ gekappt
→ SQLWarning

Auswertung von Warnungen

Mit der Methode getWarnings der Klasse ResultSets können Warnungen abgefragt werden.

```
if (e instanceof SQLWarning) {
    System.out.println("\n---SQLWarning---\n");
    SQLException sqlw = (SQLWarning) e;
    System.err.println("Message: " +sqlw.getMessage());
    System.out.println("SQLState: " + sqlw.getSQLState());
    System.out.print("Vendor error code: " + sqlw.getErrorCode());
    System.out.println("");
    //warning = warning.getNextWarning();
}
```

Fehlermeldungen

SQLState beinhaltet den herstellerspezifischen Fehlercode:

```
if (e instanceof SQLException) {
    System.out.println("\n---SQLException---\n");
    SQLException sqlex = (SQLException) e;
    System.err.println("Message: " +sqlex.getMessage());
    System.err.println("SQLState: "+sqlex.getSQLState());
    System.err.println("Error Code: " +sqlex.getErrorCode());
    System.out.println("");
}
```

Beispiel

- Beispiel:

Id	Geschlecht	Geburts-datum
2310	w	NULL

- SQL

```
INSERT INTO  Kunde (Kundennummer, Anrede, Geburtsdatum)
VALUES       (2311, 'w', NULL)
```

- JDBC

```
String      sqlString= " INSERT INTO  Kunde "
                       + "(Kundennummer, Geschlecht, Geburtsdatum)"
                       + " VALUES (?, ?, ?)";
PreparedStatement stmt = con.prepareStatement(sqlString);
stmt.setInt          (1, 2310);
stmt.setString       (2, "w");
stmt.setNull         (3,"java.sql.Date");

int anzahl = stmt.executeUpdate();
```

Behandlung von Null-Werten

- Überprüfen auf NULL-Werte

```
String columnValue = rs.getString(1);
If (rs.wasNull()){
            // Nullwert behandeln
            // (Bezogen auf den letzten ausgelesenen Wert)
}
```

- Überprüfung auf Null-Werte erforderlich bei
  - getXXX-Methoden
    - getString(),
    - getDate(),
    - ...
  - getInt(), getDouble(), ... (NULL-Werte liefern 0)
  - getBoolean() (NULL-Werte liefern false)

Zusammenfassung

Abbildung von Datentypen

Java	JDBC (java.sql.Types)
String	CHAR VARCHAR
java.math. BigDecimal	NUMERIC DECIMAL
Boolean	BIT
Int	INTEGER
double	FLOAT DOUBLE
java.sql.Date	DATE
java.sql.Time	TIME
java.sql.Timestamp	TIMESTAMP
java.sql.Clob	CLOB

- Behandlung von NULL-Werte
- Fehlerbehandlung und SQLWarnings