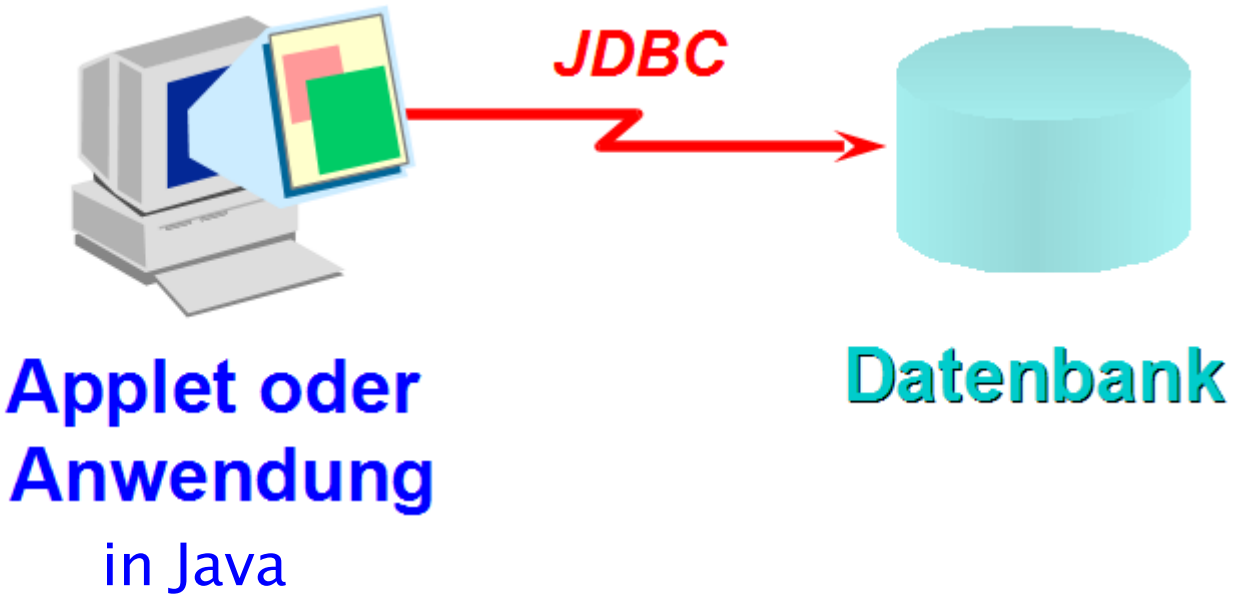
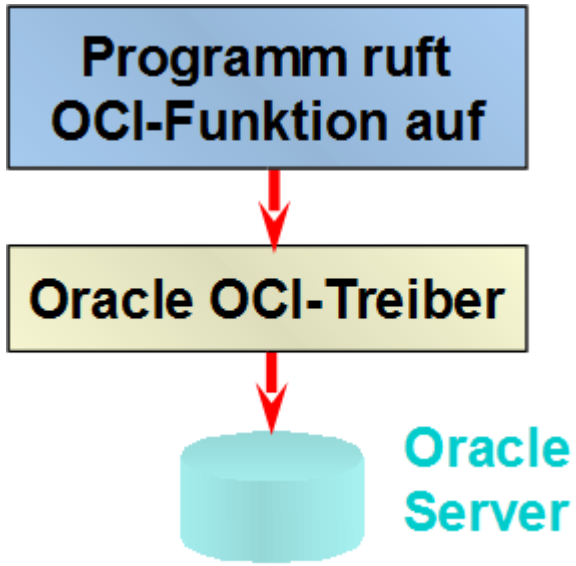
5.5 Datenbankprogrammierung mit Java/JDBC



Programm-Zugriffe auf Datenbanken in der Nicht-Java-Welt

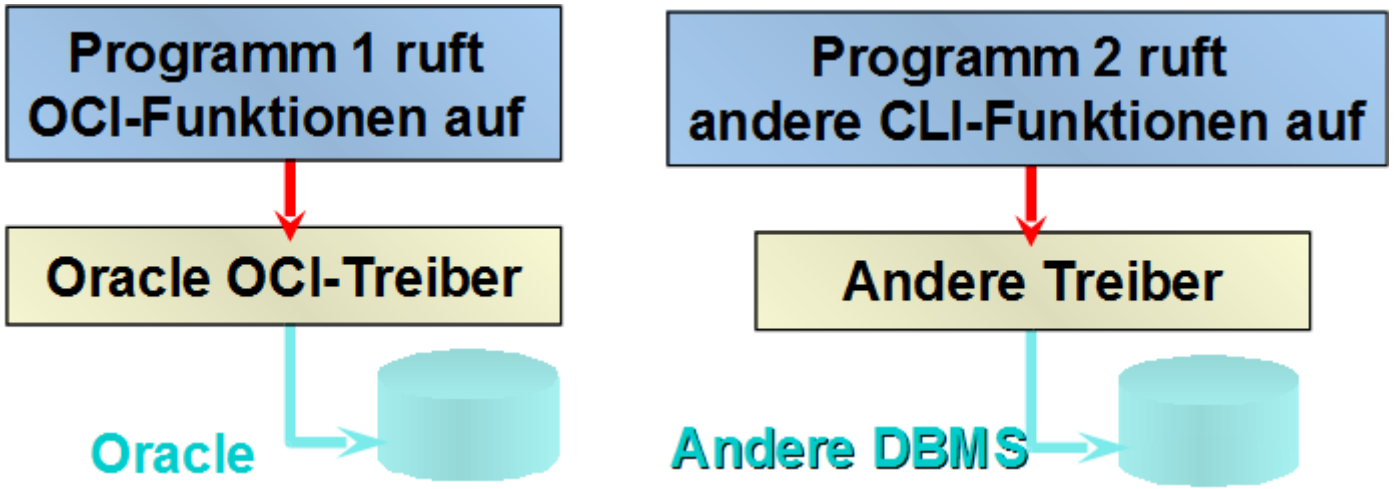
- Anbieter von Datenbanken stellen APIs zur Verfügung, um aus Programmen auf eine Datenbank zuzugreifen:
  - bekannt als Call Level Interface (CLI)
  - zum Beispiel: Oracle Call Interface (OCI)

- Anbieter bieten Treiber an, die CLI-Aufrufe erhalten und an die Datenbank weiterleiten können



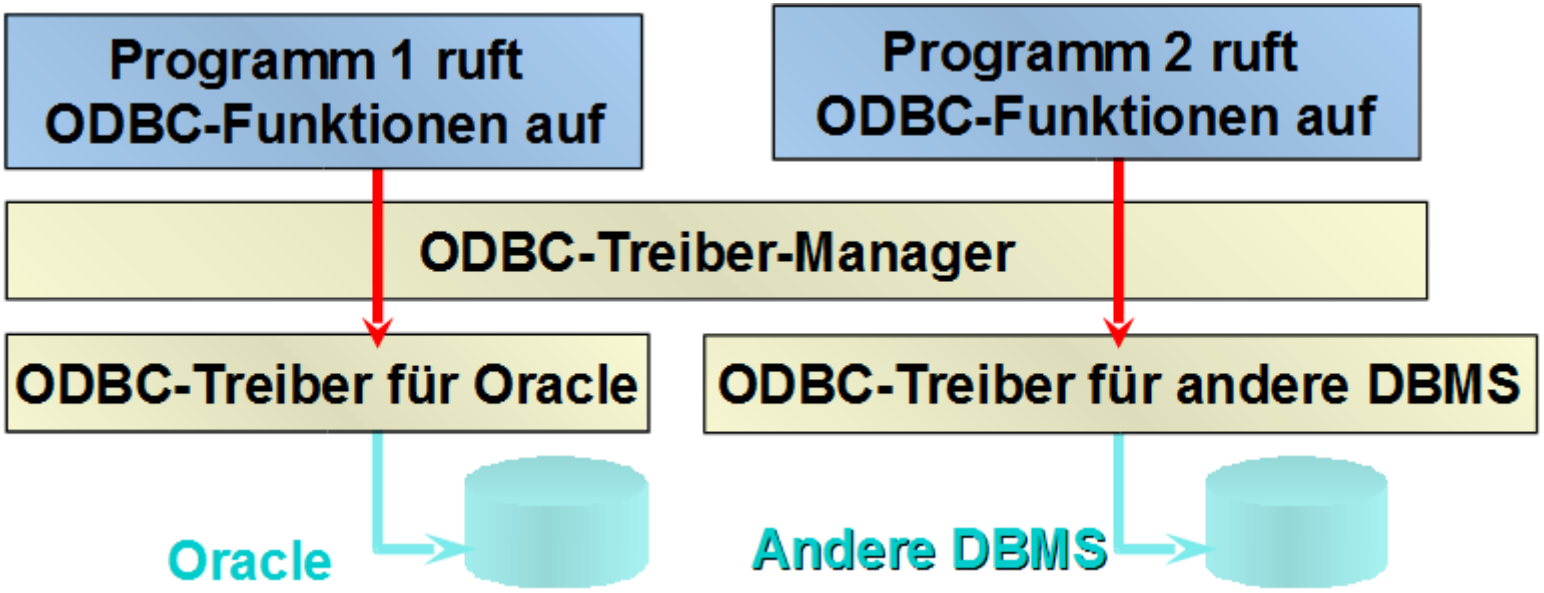
Programm-Zugriffe auf Datenbanken in der Nicht-Java-Welt (Forts.)

- Unterschiedliche Datenbanken haben ihre eigenen CLIs.
  - ⇒ Notwendigkeit, verschiedene CLIs zu nutzen
  - ⇒ Notwendigkeit eines Treibers für jedes CLI



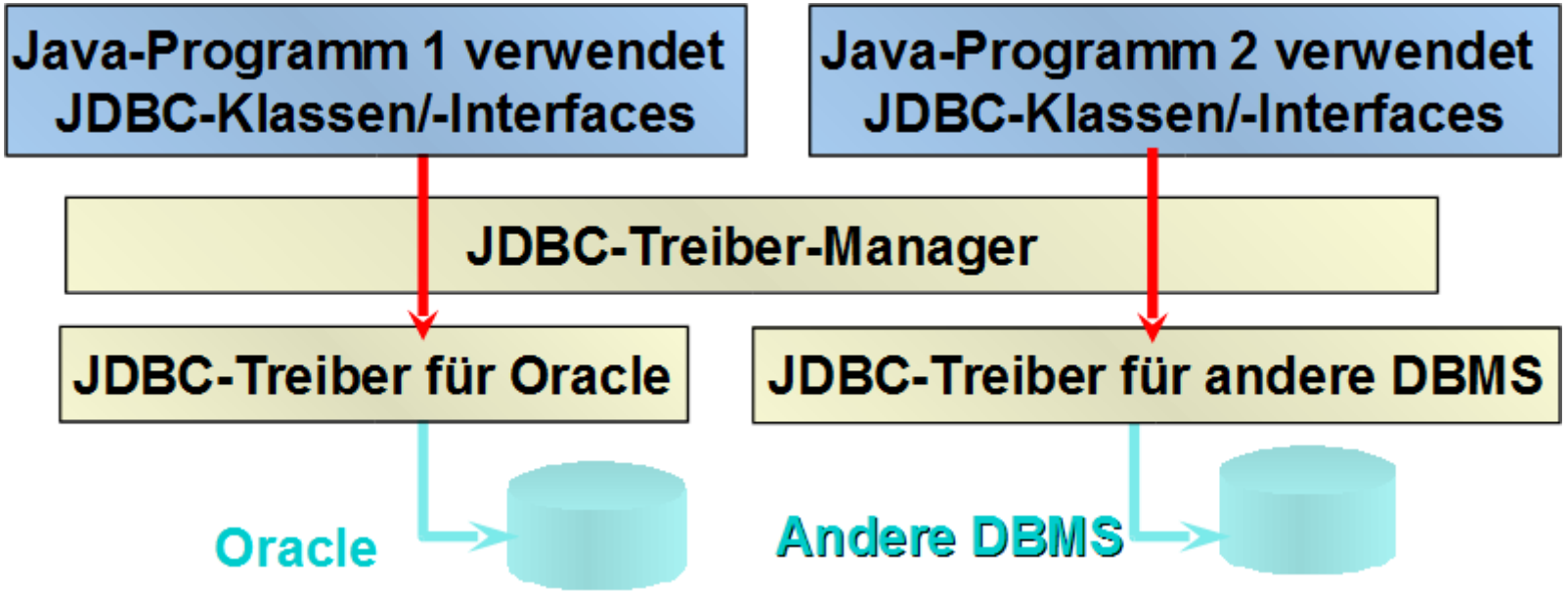
Programm-Zugriffe auf Datenbanken in der Nicht-Java-Welt (Forts.)

- **OBDC ("Open Database Connectivity")** stellt ein Standard-Interface zu beliebigen Datenbanken zur Verfügung.



Von ODBC zu JDBC

- **JDBC ("Java Database Connectivity")** wurde nach dem Vorbild ODBC für Java-Programme entwickelt.
- JDBC definiert Standard-Datenbank-Interfaces und -Klassen, die man aus Java aufrufen kann, um mit SQL-Datenbanken zu arbeiten[5].



[5] JDBC unterstützt mindestens SQL92-Syntax und -Typen. Es ermöglicht Anbieter-spezifische Erweiterungen; so gibt es viele Oracle-Erweiterungen, um Performance und Flexibilität zu verbessern.

Was bietet JDBC?

- JDBC definiert Standard-Datenbank-Interfaces. Um JDBC in einer Java-Anwendung zu nutzen, muss das java.sql-Paket importiert werden.
- Diese Interfaces sind durch JDBC-Treiber implementiert:

JDBC-Interfaces

```
interface Driver ...
interface Connection ...
interface Statement ...
interface ResultSet ...
```

JDBC-Treiber, z.B. für Oracle

```
class AAA
  implements Driver ...
class BBB
  implements Connection ...
etc.
```

- Aufgaben:
  - Verbinden mit einer Datenbank
  - Ausführen von DML-Operationen inklusive Anfragen und von DDL-Operationen, ähnlich Dynamic SQL
  - “Prepared Statements” und “Stored Procedures”

Verwendung von JDBC

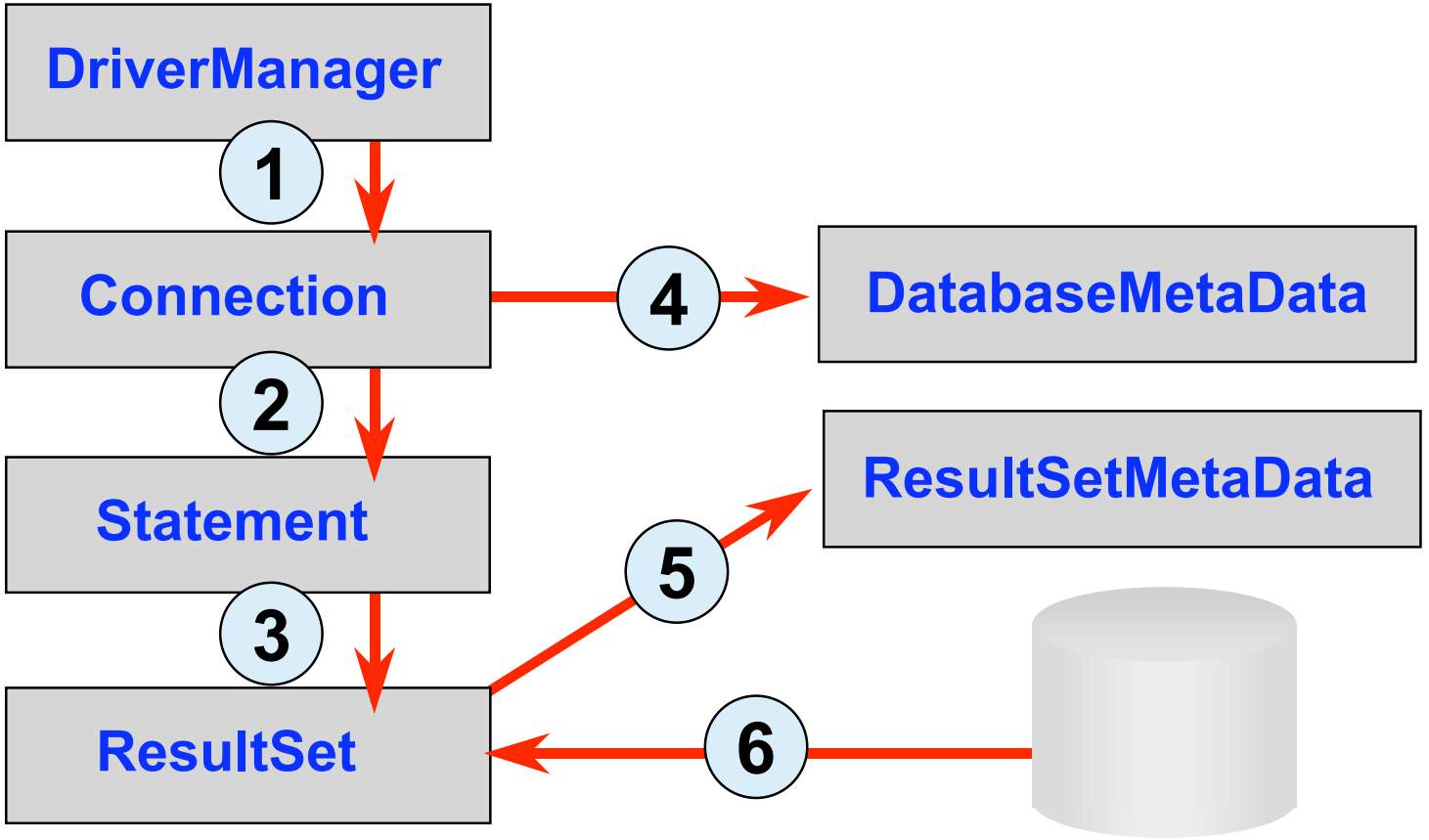
Ein vereinfachtes Beispiel:

```
import java.sql.*;
public class MyClass {
  public void MyMethod() {
     Connection conn =  DriverManager.getConnection(...);
     Statement   st  = conn.createStatement();
     ResultSet    res;
     res = st.executeQuery("select * from EMPLOYEES " +
                           "where salary > 10000 " +
                           "order by hire_date");
     //  JDBC nutzt Dynamisches SQL !!!

     ...  //  folgt: zeilenweises Durchlaufen des ResultSets

     res.close(); st.close(); conn.close();
  }
}
```

Beziehungen zwischen JDBC-Klassen



Beziehungen zwischen JDBC-Klassen (Forts.)

- DriverManager (*dt. Treiber-Verwalter,* Singleton-Klasse bzw. Objekt) ermöglicht den Zugriff auf registrierte JDBC-Treiber.

  (1): DriverManager handelt die Verbindung zu einer spezifischen Datenquelle über seine getConnection()-Methode aus.

- Die Connection-Klasse (*dt. Verbindung*) wird vom JDBC-Treiber angeboten, genau wie alle im folgenden erwähnten Klassen. Ein Connection-Objekt stellt eine Sitzung mit einer Datenbank dar.

  (2): Ein Connection-Objekt kann Statement-Objekte mittels createStatement() erzeugen.

- Ein Statement-Objekt (*dt. Anweisung*) behandelt eine SQL-Anweisung.

  (3): Ein Statement-Objekt kann z. B. eine Anfrage mittels executeQuery()-Methode ausführen und das Ergebnis in einem ResultSet-Objekt bereitstellen. Ein DML-Anweisung (z.B. **update**) würde mittels executeUpdate() ausgeführt.

Beziehungen zwischen JDBC-Klassen (Forts.)

- Ein ResultSet-Objekt (*dt. Ergebnismenge*) ermöglicht den Zugriff auf eine Tabelle mit Ergebnisdaten, die durch Ausführung einer Anfrage erstellt wurde.

  (6): Die Zeilen werden mit der next()-Methode der Reihe nach durchlaufen. In jeder Zeile kann mit get...(...)-Methoden auf die Spaltenwerte in beliebiger Reihenfolge zugegriffen werden.

- DatabaseMetaData und ResultSetMetaData liefern Metadaten über die Datenbank bzw. ein ResultSet zurück.

  (4,5): Dazu erzeugt getMetaData() von Connection bzw. ResultSet ein entsprechendes Metadaten-Objekt.

Registrierung eines JDBC-Treibers

- JDBC-Treiber müssen sich selbst beim DriverManager registrieren.
- Treiber registrieren sich automatisch, wenn sie geladen werden.

```
try {
   Class c = Class.forName("oracle.jdbc.driver.OracleDriver");
}
catch (ClassNotFoundException e) {
   e.printStackTrace();
}
```

Datenbank-Verbindung (Connection)

- DriverManager wird benutzt, um eine Verbindung zu einer Datenbank zu öffnen.
- Die Verbindung wird spezifiziert durch eine URL, die den JDBC-Treiber (protocol) und die jeweilige Datenbank-Server-Instanz (connectString) identifiziert.

  jdbc:<protocol>:<connectString>
  z.B. jdbc:oracle:thin:@<host>:<port>:<systemid>

- Der folgende Beispielcode erzeugt dafür ein Connection-Objekt:

```
Connection conn;
try {
  conn =  DriverManager.getConnection(
      "jdbc:oracle:thin:@myhost:1521:prod1",
      "<user>", "<password>");
}
catch (SQLException e) {...}
```

Datenbank-Metadaten (DatabaseMetaData)

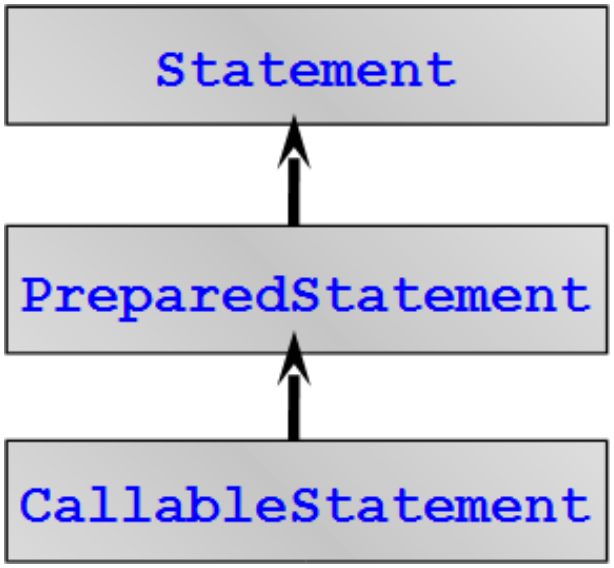
- Ein Connection-Objekt kann ein DatabaseMetaData-Objekt erzeugen.
- Dieses bietet eine Reihe von Methoden an, um Metadaten über eine Datenbank zu bekommen, z.B.:

```
Connection conn;
. . .
try {
  DatabaseMetaData dm = conn.getMetaData();
  String s1 = dm.getURL();
  String s2 = dm.getSQLKeywords();
  boolean b1 = dm.supportsTransactions();
  boolean b2 = dm.supportsSelectForUpdate();
}
catch (SQLException e) {...}
```

SQL-Anweisungen (Statement)

- Für die Übergabe, Kompilierung und Ausführung von SQL-Anweisungen sind drei Interfaces definiert, die folgende Fähigkeiten anbieten:

- Anfragen und andere DML-/ DDL-Operationen ausführen
- Vorkompilierte Anweisungen ausführen
- Gespeicherte Prozeduren aufrufen



SQL-Anweisungen (Statement) (Forts.)

- Die executeQuery()-Methode von Statement führt eine SQL-Anfrage aus und gibt ein ResultSet zurück.

```
try {
    Statement stmt = conn.createStatement();
    ResultSet rset = stmt.executeQuery
        ("select first_name||' '||last_name name, salary from EMPLOYEES");
}
catch (SQLException e) {...}
```

Verarbeitung von Anfrageergebnissen (ResultSet)

- Zum Durchlaufen eines ResultSet dient insbes. die next()-Methode, die einen Zeiger auf die erste/nächste Ergebniszeile setzt und gleichzeitig meldet, ob das erfolgreich war.
- JDBC bietet auch noch andere Bewegungen im ResultSet an.
- Zum Holen der Ergebnisfelder sind get...(...)-Methoden zu verwenden, die die Daten auch in äquivalente Java-Typen wandeln.

```
try {
    ResultSet rset = stmt.executeQuery(" ...  wie oben ... ");
    while (rset.next()) {
        String empname = rset.getString(1);
        BigDecimal salary = rset.getBigDecimal(2);
        //  oder Spaltenzugriffe über Namen:
        String empname = rset.getString("name");
        BigDecimal salary = rset.getBigDecimal("salary");
    }
} catch (SQLException e) {...}
```

Ergebnis-Metadaten (ResultSetMetaData)

- Ein ResultSet-Objekt kann ein ResultSetMetaData-Objekt erzeugen.
- Dieses bietet Methoden an, um Metadaten über das Ergebnis, insbes. über dessen Spaltenanzahl/-namen/-typen zu bekommen, z.B.:

```
try {
    ResultSet rset = ...;
    ResultSetMetaData md = rset.getMetaData();
    while (rset.next()) {
        for (int c = 0; c < md.getColumnCount(); c++) {
            String lbl = md.getColumnLabel();
            String typ = md.getColumnTypeName(); ...
            if (typ equals "String")) {
              System.out.println("Column " + lbl + " has value " + rset.getString(lbl));
            } ...
        }
    }
} catch (SQLException e) {...}
```

Anweisungen II (Prepared Statement)

- Wenn eine Anweisung mehrfach mit verschiedenen Variablen ausgeführt werden soll, wird ein PreparedStatement-Objekt benutzt.
- Dessen SQL-Code wird vorkompiliert.
- Dessen zu bindende Variablen sind mit einem ? zu kennzeichnen.

```
try {
   Connection conn =  DriverManager.getConnection(...);
   PreparedStatement pstmt =
     conn.prepareStatement("update EMP set SAL = ?");
    ...
} catch (SQLException e) {...}
```

Anweisungen II (Prepared Statement) (Forts.)

- Die Variablen sind mit Hilfe von set...(...)-Methoden der Klasse PreparedStatement für die nächste Ausführung zu binden. Ihre Nummerierung ergibt sich durch die Reihenfolge der ?-Platzhalter.
- Die Ausführung erfolgt mit executeQuery() bzw. executeUpdate().

```
try {
    ...
    PreparedStatement pstmt =
      conn.prepareStatement("update EMP set SAL = ?");
    ...
   pstmt.setBigDecimal(1, new BigDecimal(55000));
   pstmt.executeUpdate();
   pstmt.setBigDecimal(1, new BigDecimal(65000));
   pstmt.executeUpdate();
    ...
} catch (SQLException e) {...}
```

Anweisungen III (Callable Statement)

- Um eine gespeicherte Prozedur/Funktion aufzurufen, wird ein CallableStatement-Objekt verwendet.
- in-Parameter werden mit set...(...) wie in einem PreparedStatement gesetzt. — Für Rückgabewerte und out-Parameter wird mit registerOutParameter der Typ spezifiziert.

Gesp. Funktion: `getSal (v_ename in varchar2, v_job out varchar2) return number`

```
...
CallableStatement cs =
    conn.prepareCall( "{? = call getSal(?,?)}");
cs.registerOutParameter(1, Types.NUMERIC);
cs.setString(2, "King");
cs.registerOutParameter(3, Types.VARCHAR);
cs.executeQuery();
System.out.println("King earns " + cs.getFloat(1) +
          " as a " + cd.getString(3));
```

Transaktionen

- Verbindungen werden mit der Eigenschaft AutoCommit=true initialisiert, was zu einer separaten Transaktion pro SQL-Anweisung führt.
- Um die Kontrolle selber zu übernehmen:

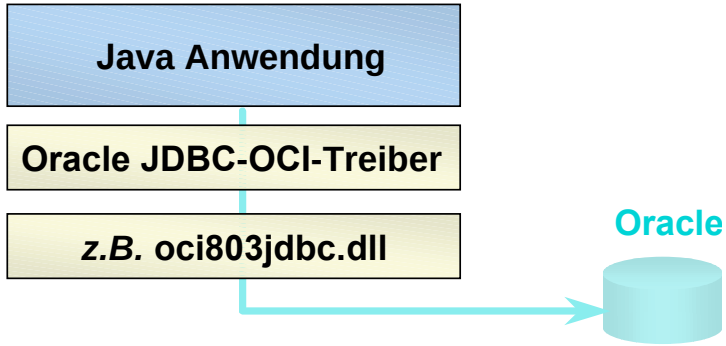
```
 ...
Connection conn =  DriverManager.getConnection(...);
 ...
conn.setAutoCommit(false);  //  ab jetzt keine Autocommits
 ......                     //  Führe SQL-Anweisungen aus
conn.commit();              //  Beende Transaktion.
 oder  conn.rollback();
```

Oracle-JDBC-Treiber

- Oracle bietet zwei Typen von JDBC-Treibern an:
- Der Oracle-JDBC-Thin-Treiber ist in 100% reinem Java geschrieben.
- Er kann über das Netzwerk heruntergeladen werden mit Hilfe eines Java Applets.

- Dieser Treiber ist aus Java-Applets oder Java-Anwendungen nutzbar.

- Der Oracle-JDBC-OCI-Treiber setzt JDBC-Aufrufe in OCI-Aufrufe an einen vorinstallierten "nativen" Treiber auf dem Client um;
  er ist also plattform-abhängig.



- Nur für Java-Anwendungen nutzbar.