BEUTH HOCHSCHULE FÜR TECHNIK BERLIN
University of Applied Sciences

Datenbank-programmierung und Datenintegration

Medieninformatik Bachelor
Modul 9:
Datenbanksysteme

Prof. Dr. Petra Sauer, Prof. Dr. habil. Alexander Löser

Sie erinnern sich: Ihre Aufgaben

- **Aufgabe 1 Anfragen & Modellierung“**
  Denken Sie mal darüber nach, welche Anfragen Sie an die AOL Daten stellen möchten. Bitte Sie bitte ein logisches und physisches Schema zur Beantwortung dieser Anfragen.
- **Aufgabe 2 „SQL und Anfrageausführung“**
  Bitte formulieren Sie für Ihre Analyseideen aus 1.) die SQL Anfragen. Sie verstehen auch Möglichkeiten der Anfrageausführung bzw. Optimierung.
- **Aufgabe 3 „Datenintegration“**
  Zur Ausführung der Ausführung fehlen Ihnen noch externe Daten, z.B. aus dem Internet Archive, DMOZ oder Freebase.org. Bitte ergänzen Sie Ihr Schema und die Datenbasis.
- **Aufgabe 4 „Analyse, Erkenntnisgewinn und Wert“**
  Stellen Sie in 5 Minuten die wichtigsten Erkenntnisse aus den Daten vor. Bewerten Sie den Erkenntnisgewinn, z.B. gegenüber Ihren Kommilitonen oder der Literatur! Welche Erkenntnisse hätten einen kommerziellen Wert?

Die Themen

- Was sind Datenbanken?
  - Motivation, Historie, Datenunabhängigkeit, Einsatzgebiete
- Datenbankentwurf im ER-Modell & Relationaler Datenbankentwurf
  - Entities, Relationships, Kardinalitäten, Diagramme
  - Relationales Modell, ER -> Relational, Normalformen, Transformationseigenschaften
- Relationale Algebra & SQL
  - Kriterien für Anfragesprachen, Operatoren, Transformationen
  - SQL DDL, SQL DML, SELECT … FROM … WHERE …
- Datenintegration & Transaktionsverwaltung
  - JDBC, Cursor, ETL, ORM
  - Mehrbenutzerbetrieb, Serialisierbarkeit, Sperrprotokolle, Fehlerbehandlung, Isolationsebenen in SQL
- Ausblick
  - Map/Reduce, HDFS, Hive …
  - Wert von Daten

IHR ERSTER TAG IN DER NEUEN FIRMA ALS DATENBANK- BZW. SOFTWAREENTWICKLER …

Persistenz mit Java und SQL: Ihre Aufgabe

Unter Persistenz verstehen wir die dauerhafte Speicherung von flüchtigen Daten, wie zum Beispiel den Objekten in einer Java-Anwendung. Als Speichermedium kommen das Dateisystem oder Datenbanken in Frage.

- Sie sind DB-Entwickler. Ihr Kunde verfügt über ein RDBMS mit der Tabelle
  - PERSON(person_id, first_name, last_name, born_date)
- Der Kunde möchte von einer Java Umgebung die Tabelle PERSON lesen …
- … *regelmäßig* nach Personen mit einem bestimmten Geburtsjahr suchen …
- … und *möglichst zeitnah* viele neue Datensätze einfügen.
- Das Schlüsselattribut *person.person_id* soll in der Klasse *Person_Logic.class* über die folgende Methode erzeugt werden.
  - *long generatePerson_ID(String first_name, String last_name, Date born_date)*

Gliederung

Einführung in die Datenbankanwendungsprogrammierung

- **Möglichkeiten zur Datenbankanwendungsprogrammierung**
- Cursor-Konzept
- Standard Java-SQL-Schnittstellen: JDBC

Beschreibung der Problematik:

- Bisher SQL kennen gelernt, dennoch **kein einziges Programm entwickelt**
- Was **können** DB-Sprachen?
  - DB-Strukturen, -inhalte, -nutzer, -rechte etc. managen (mit allen möglichen DB-Objekten umgehen)
  - Garantie:
    - Terminierung der Operationen
    - Endlichkeit der Ergebnisse von Operationen
    - Optimierbarkeit der Operationen
- Was **fehlt** den DB-Sprachen?
  - Kontrollstrukturen
  - Ein- und Ausgabefunktionalität (Bibliotheken evtl.?)
  - Möglichkeiten der Gestaltung der GUI?

Problem: um DB-Anwendungen zu entwickeln, sind reine **DB-Sprachen nicht ausreichend**

Lösungsansätze für das Problem (1):

- **Anbindung der DB-Sprache an konventionelle Programmiersprache**
  - **Einbettung** von Operationen einer DB-Sprache in eine Programmiersprache, Vorübersetzer

    Beispiel: Embedded SQL (Java-Einbettung: SQLJ)
  - Bereitstellung von **Funktionen zum Aufruf von DB-Operationen** (API / Treiberkonzept) um Daten aus einer relationalen Datenbank abzufragen und CRUD-Operationen (Create / Update / Delete) durchzuführen.

    Zum Beispiel:
  - C++ → ODBC-API (Open Database Connectivity) *Microsoft*
  - Java → JDBC-API (Java Database Connectivity) *Open Source*
  - C# → ADO.NET (ActiveX Data Objects) *Microsoft*
  - VB.NET → ADO.NET (ActiveX Data Objects) *Microsoft*
- **Charakteristik**: **DB-Sprache für Anwendungsentwickler sichtbar**

Lösungsansätze für das Problem (2):

- **Erweiterung der DB-Sprache** um Kontrollstrukturen bzw. Kopplung mit einer Makrosprache
  - Beispiel: 4GL-Werkzeuge zur Entwicklung von maskenbasierten Anwendungen mit hohem Datenaufkommen
  - Beispiel: SQL/PSM (SQL-Standard) und PL/SQL (Oracle)
- **Erweiterung** einer existierenden **Programmiersprache** zu einer persistenten Programmiersprache, Unterscheidung zwischen transienten und persistenten Variablen und Objekten

  Beispiel: Erweiterungen von C++, Java etc. um ein Persistenzkonzept (Weg zu OODB → db4o, ObjectStore, Versant etc.)
- **Charakteristik**:
  - **Sprache für Persistenzmechanismus und Programmiersprache vereinheitlicht**
  - **Keine Aufgabenteilung**
  - **Es gab keinen Standard für OODB → OODB „ausgestorben“**

Lösungsansätze für das Problem (3):

- Entwicklung neuer Programmiermodelle für den Zugriff auf Datenbanken über „**transparente Objektpersistenz**“ – mit einer Middleware-Technologie, die als **Object-Relational-Mapper (ORM)** bezeichnet wird
  - Beispiel: JDO, JPA, Hibernate, EclipseLink, TopLink

- **Charakteristik**:
  - **DB-Sprache für Anwendungsentwickler nur bedingt sichtbar**
  - **Lösung für „Object-relational impedance mismatch“**

PARADIGMENBRUCH: OO/JAVA UND SQL WELT

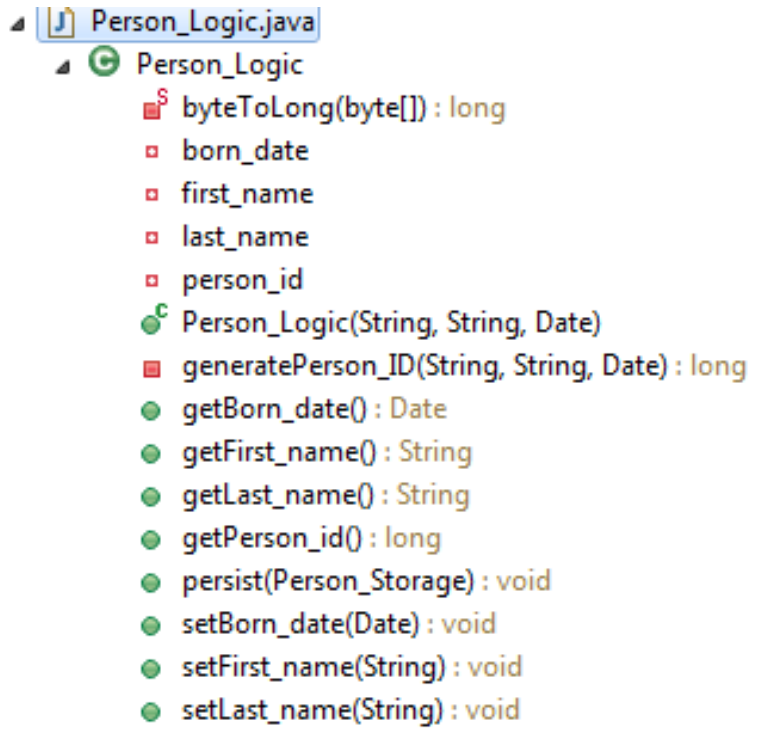
Gliederung

Einführung in die Datenbankanwendungsprogrammierung

- Möglichkeiten zur Datenbankanwendungsprogrammierung
- **Cursor-Konzept**
- Standard Java-SQL-Schnittstellen: JDBC

Unser Beispiel: Welche Probleme sehen Sie?

```
create table person (
person_id BIGINT,
first_name VARCHAR(100),
last_name VARCHAR(100),
born_date DATE
);
```




```
============================================================
select * from PERSON
Execution Time: 0ms
============================================================
PERSON_ID           FIRST_NANE      LAST_NAME         BORN_DATE
------------------  --------------- ------------    ----------
6144000434943451206  Bill           Smith           1969-01-01
297806464197500561   Inga           Jones           1970-09-24
2208189410429574150  Moritz         Youngster       1997-02-24
7276586290201554280  Bob            Builder         1976-04-20
1658111608567355835  Homer          Simpson         1995-01-01
2958682298118957594  Jesus          Christus        0001-12-24
4595116680171429517  Mathilde       Baby            2011-01-19
============================================================
```

Paradigmenbruch

Wir bezeichnen die Unterschiedlichkeit der Konzeptes ‚objektorientierte Softwareentwicklung' und ‚relationale Datenbank' als Paradigmenbruch.

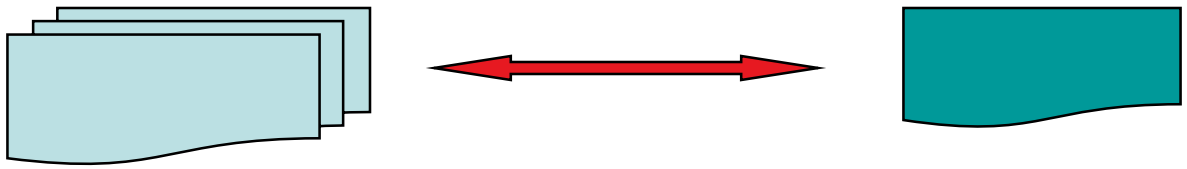
- Objektorientierung: Über (gerichtete) Referenzen miteinander verknüpfte Objekte
- Datenbanken: tabellarische Darstellung, relationale Algebra

- Mapping
  - Jede Klasse eine eigene Tabelle
  - Attribute der Klasse = Attribute/Spalten der Tabelle
  - Abbildung der Referenzen auf andere Objekte über Fremdschlüssel

- Typische Probleme
  - Missmatch ‚Basisdatentypen' Java und SQL
  - Objekt-Identität vs. Primary Key
  - Mengenwertige Attribute in Java vs. 1. Normalform
  - Kardinalitäten
  - Vererbung
  - Performance, sehr kleine Tabellen, Datennachladen …

Generelles Problem der Kopplung von DB-Sprache und (imperativer) Programmiersprache: Unverträglichkeit der Typsysteme

DB-Sprache (SQL)	**Programmiersprache x**
Datenstruktur Relation	Datenstruktur Tupel
(= Menge von Tupeln)	



„Impedance Mismatch“

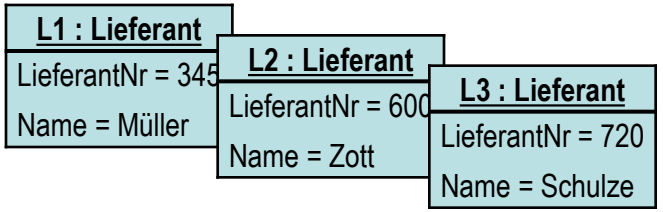
Lösung: **Cursor-Konzept**

Das Cursor-Konzept

Illustration Impedance Mismatch am Beispiel von SQL:

- Die Relationale Algebra bzw. die Sprache SQL ist mengenorientiert. Wird beispielsweise eine Anfrage wie „*Selektiere mir alle Lieferanten, die Milch liefern*“ abgesetzt, so wird eine Menge von „flachen“ Tupeln als Ergebnis geliefert.
- Objektorientierte Programmiersprachen sind jedoch satzorientiert. Das heißt Objekte können nur einzeln bearbeitet werden, selbst wenn sie in Kollektionen wie etwa Sets oder Listen zusammengefasst sind.

LieferantNr	Name
345	Müller
720	Schulze
600	Zott



- Liefert also die SQL-Abfrage mehr als einen Datensatz als Ergebnismenge so muss beim Wechsel der Objekte von der Datenbankdarstellung zurück in die Hauptspeicherdarstellung der Anwendung eine Transformation in eine zum Objektmodell passende Mengendarstellung erfolgen (zum Beispiel collections, sets, arrays).

Das Cursor-Konzept

Cursor

- „Current Set of Records“
- Iterator über eine Liste von Tupeln
- fest an eine Anfrage gebunden

Cursoroperationen im Überblick:

- Cursor **deklarieren** (Zuordnung des select-Statements)
- Cursor **öffnen** (Tupel der Anfrage werden zur Verfügung gestellt)
- Operation **fetch** auf dem geöffneten Cursor verwenden (Zugriff auf ein Tupel des Cursors und Datentransfer in das Anwendungsprogramm sowie Weitersetzen des Cursor-Zeigers)
- Cursor **schließen** (Speicherplatzfreigabe)
- (positioniertes **update** und **delete** - in Bezug auf das gerade im Zugriff befindliche Tupel mit Selektionsprädikat …**current of** …)

Das Cursor-Konzept

Cursordeklaration (SQL-92):

declare <*cursorname*> *[**insensitive**] [**scroll**] **cursor for*** <*select-Statement*>*;*

Beispiel:

declare *kunde* ***cursor for select*** * ***from*** *kdst;*

Für Reservierungsabfragen müssen Sperren erworben werden -> Erweiterung des select-Statements um ***for update of*** *{*<*relation*> | <*spaltenliste*>*}*

Beispiel:

declare *kunde* ***cursor for***

select * ***from*** *kdst*

for update of *umshaben;*

Das Cursor-Konzept

Cursorverwendung:

- Öffnen des Cursors

  open *kunde;*

- Transfer Tupel in Variablen des Anwendungsprogramms über fetch (Variablendeklaration erforderlich!)

  fetch *kunde* ***into*** *:kdnr, :firmenname, :ort, …;*

- Änderungen von Tupeln durchführen über update bzw. delete

  update *kdst*
  set *umshaben = 0*
  where current of *kunde;*

  delete from *kdst*
  where current of *kunde;*

- Schlíeßen des Cursors

  close *kunde;*

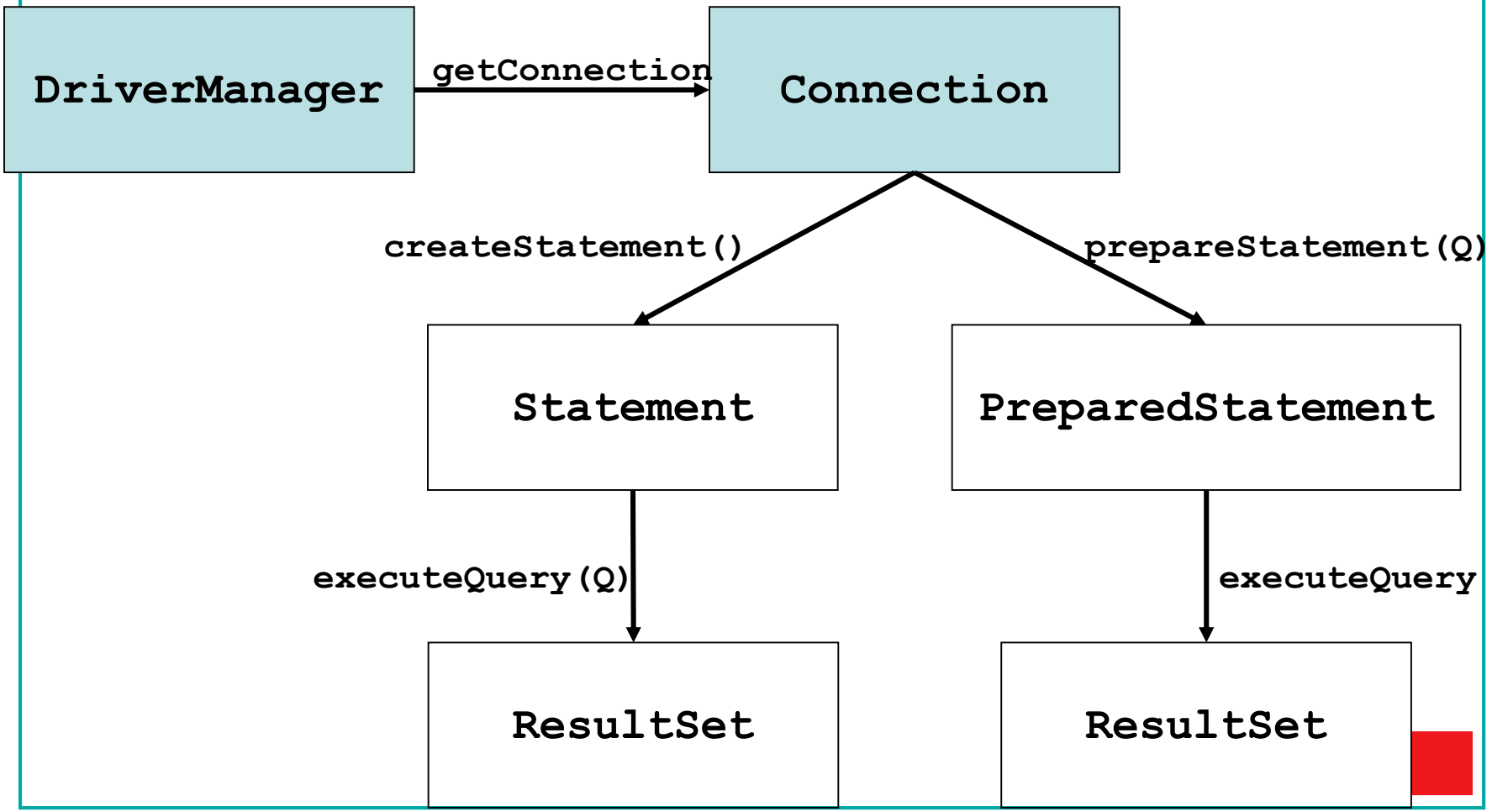
Einführung in die Datenbankanwendungsprogrammierung

- Möglichkeiten zur Datenbankanwendungsprogrammierung
- Cursor-Konzept
- **Standard Java-SQL-Schnittstellen: JDBC**

DER ZUGRIFF AUF DAS RDBMS MIT JAVA/JDBC

JDBC Überblick

JDBC stellt Basisoperationen für die DB-Verbindung, zum Typ-Mapping, für statische und dynamische Anfragen und zur Navigation in Resultaten bereit.



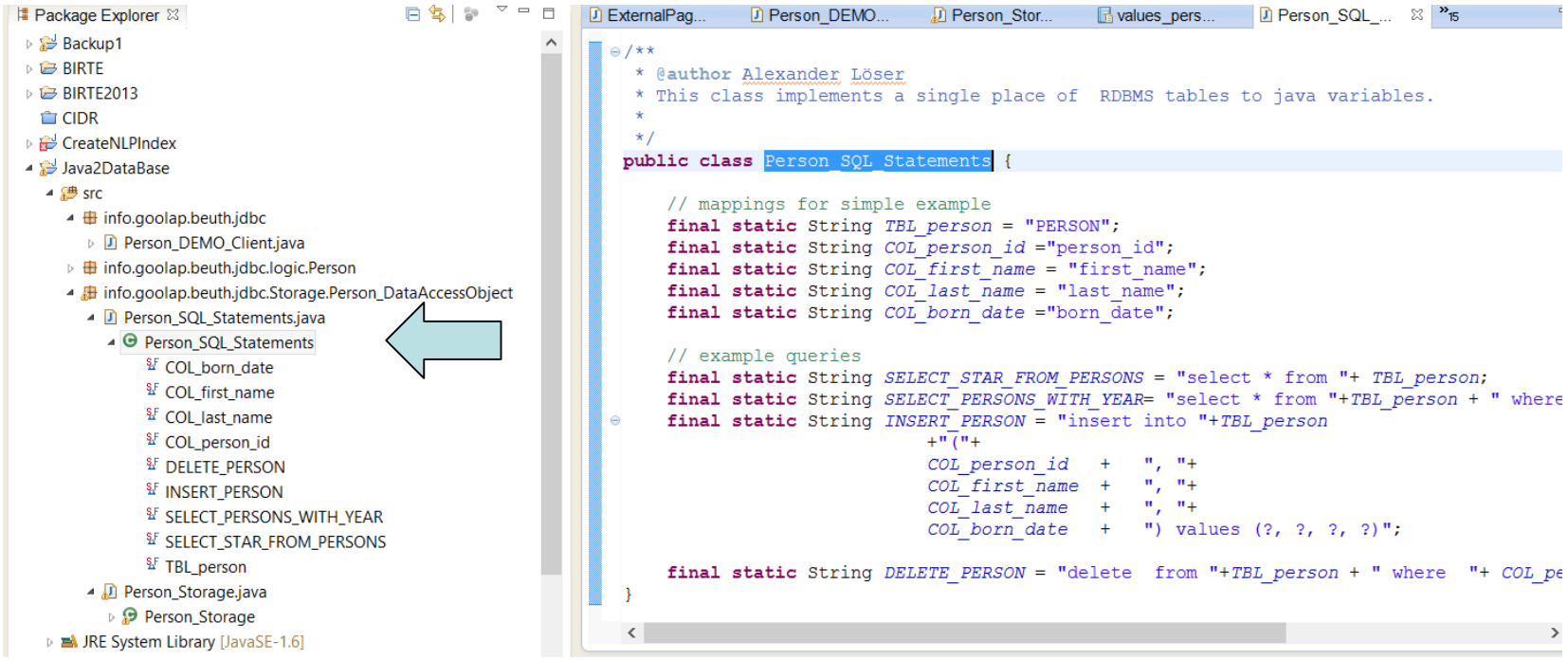
Erste Schritte: Verbindung zum RDBMS

Wir verwenden den von MySQL (oder Oracle) bereitgestellten JDBC Treiber.

- Treiber für das DBMS einbinden
  - In BUILD_PATH: **`mysql-connector-java-5.1.22-bin.jar`**
- Treiber instanziieren
  - **`Class.forName("com.mysql.jdbc.Driver");`**
- Verbindung zur Datenbank aufbauen
  - **`Connection meineConnection = DriverManager.getConnection(URL, name, password);`**
  - URL ist DBMS und Datenbank-spezifisch
    - "jdbc:subprotocol:datasource";
  - **`Connection meineConnection = DriverManager.getConnection ("jdbc:ids://<host>/<database>", "<user>","<passwd>");`**

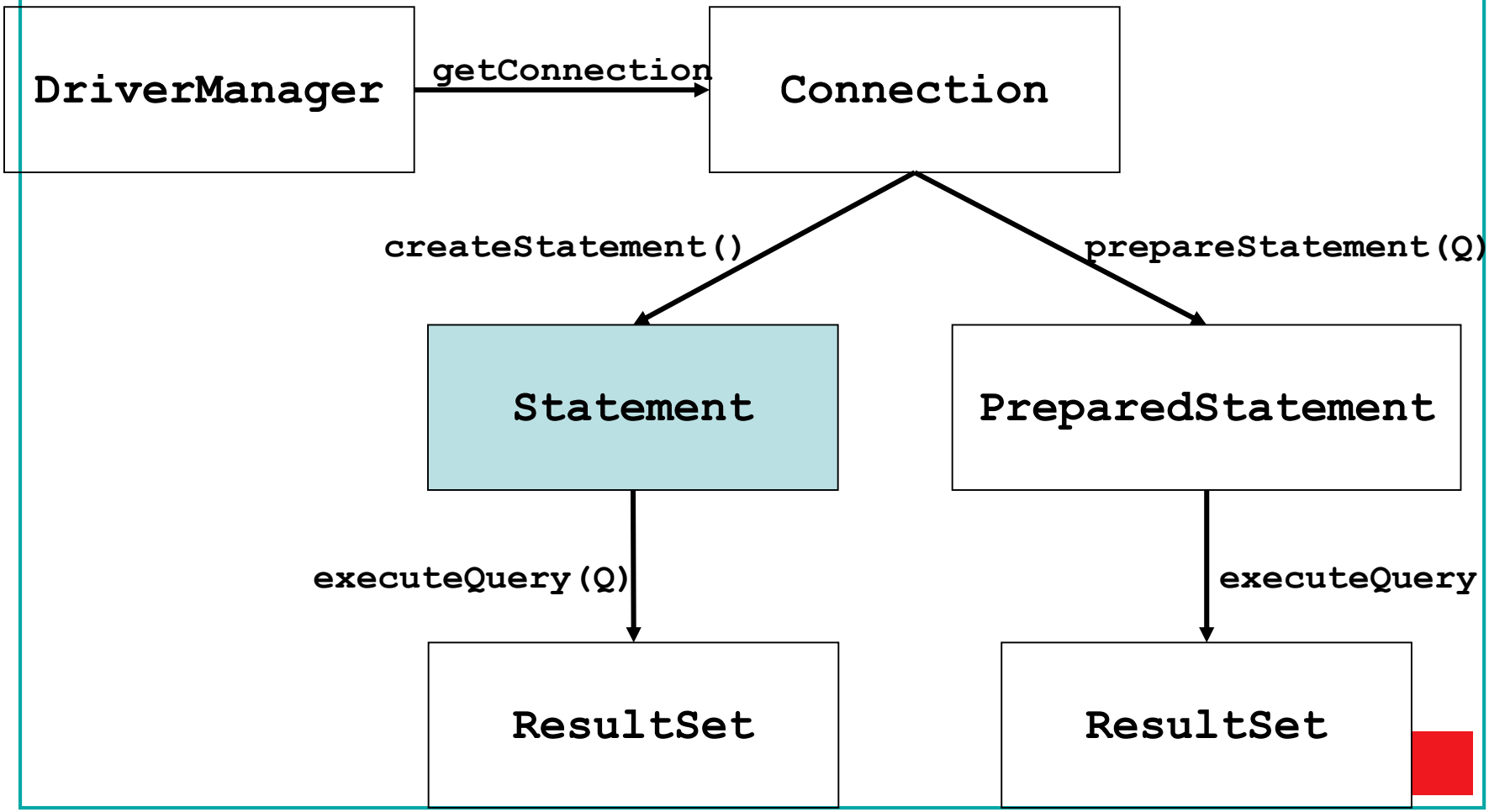
Wo speichern wir die SQL Anfragen

Möglichkeiten: A) separate Klasse, B) Datei, C) Datenbank. Wir vermeiden SQL Anfragen über den Java Quellcode zu „verstreuen“.



JDBC Überblick

JDBC stellt Basisoperationen für die DB-Verbindung, zum Typ-Mapping, für statische und dynamische Anfragen und zur Navigation in Resultaten bereit.



Dynamisches SQL

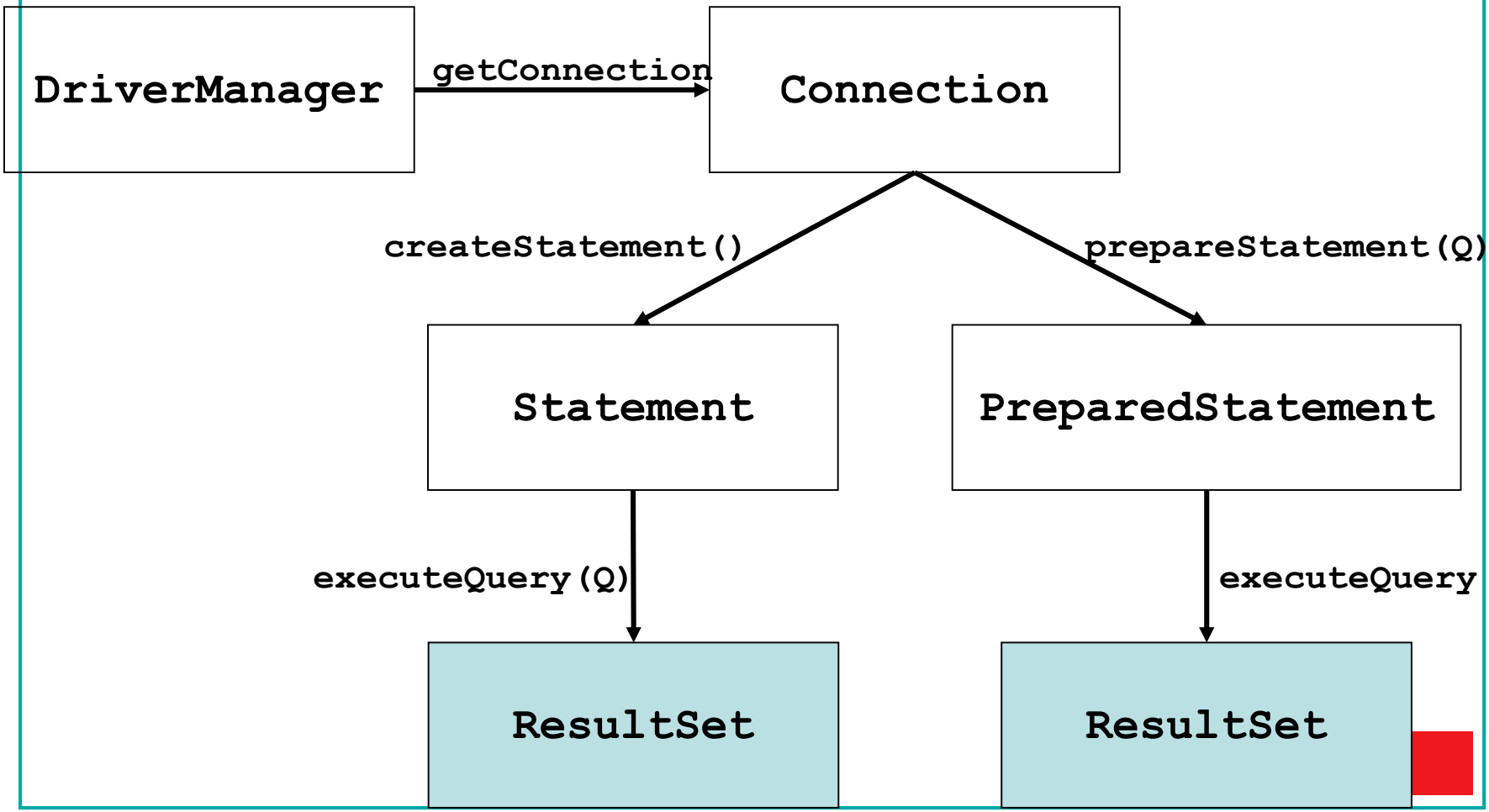
Für Ad-hoc Anfragen stellt JDBC das dynamische SQL zur Verfügung. Einmal kompilierte Anfragepläne werden zeitlich begrenzt im Cache abgespeichert.

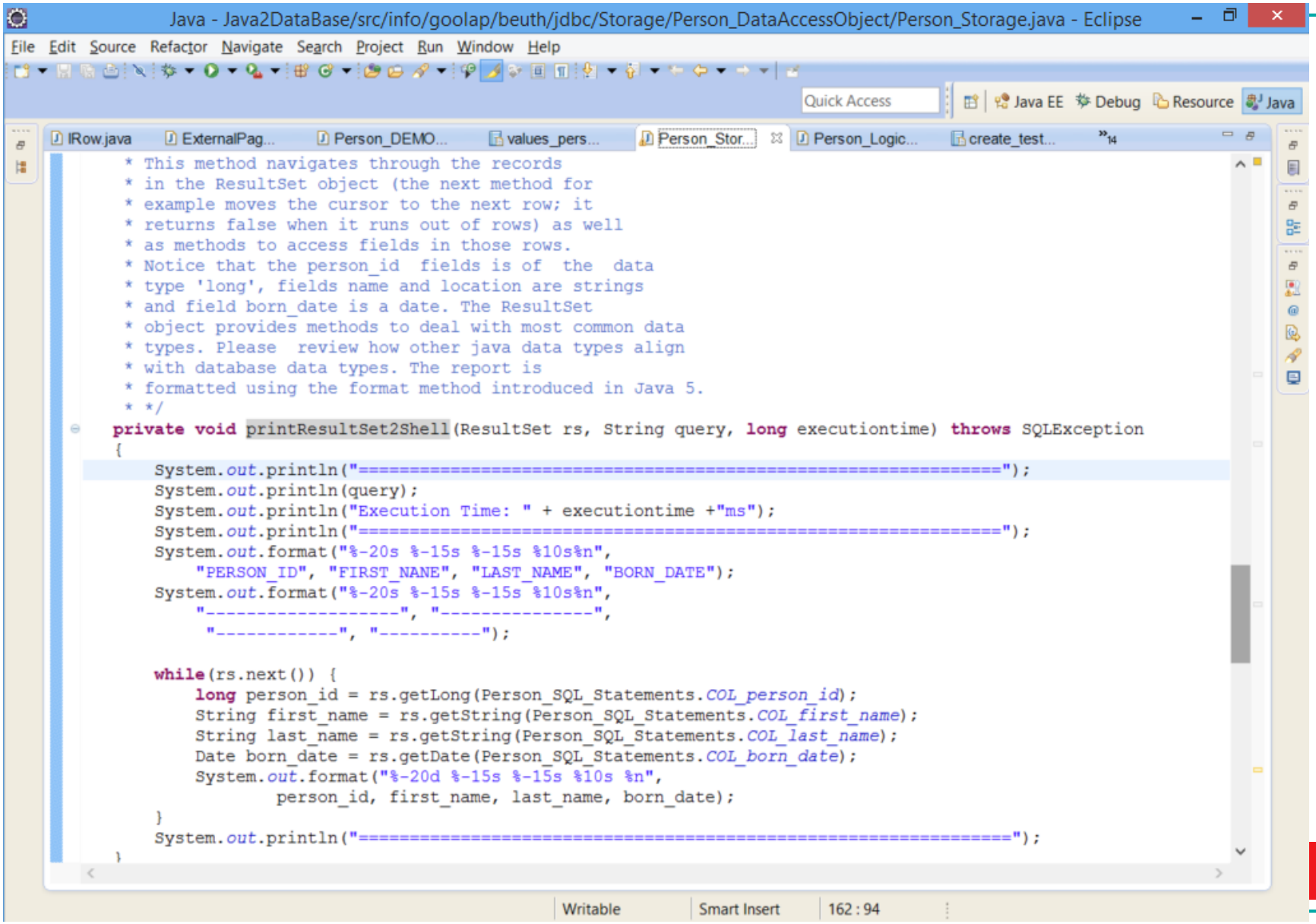
- SQL Statements werden
  - als Strings an die Datenbank geschickt, …
  - dann kompiliert …
  - … und danach ausgeführt
- Physikalischer Plan des Statement liegt für eine Weile im Cache
- Wiederholte Ausführung desselben Statements erspart die Optimierungsphase

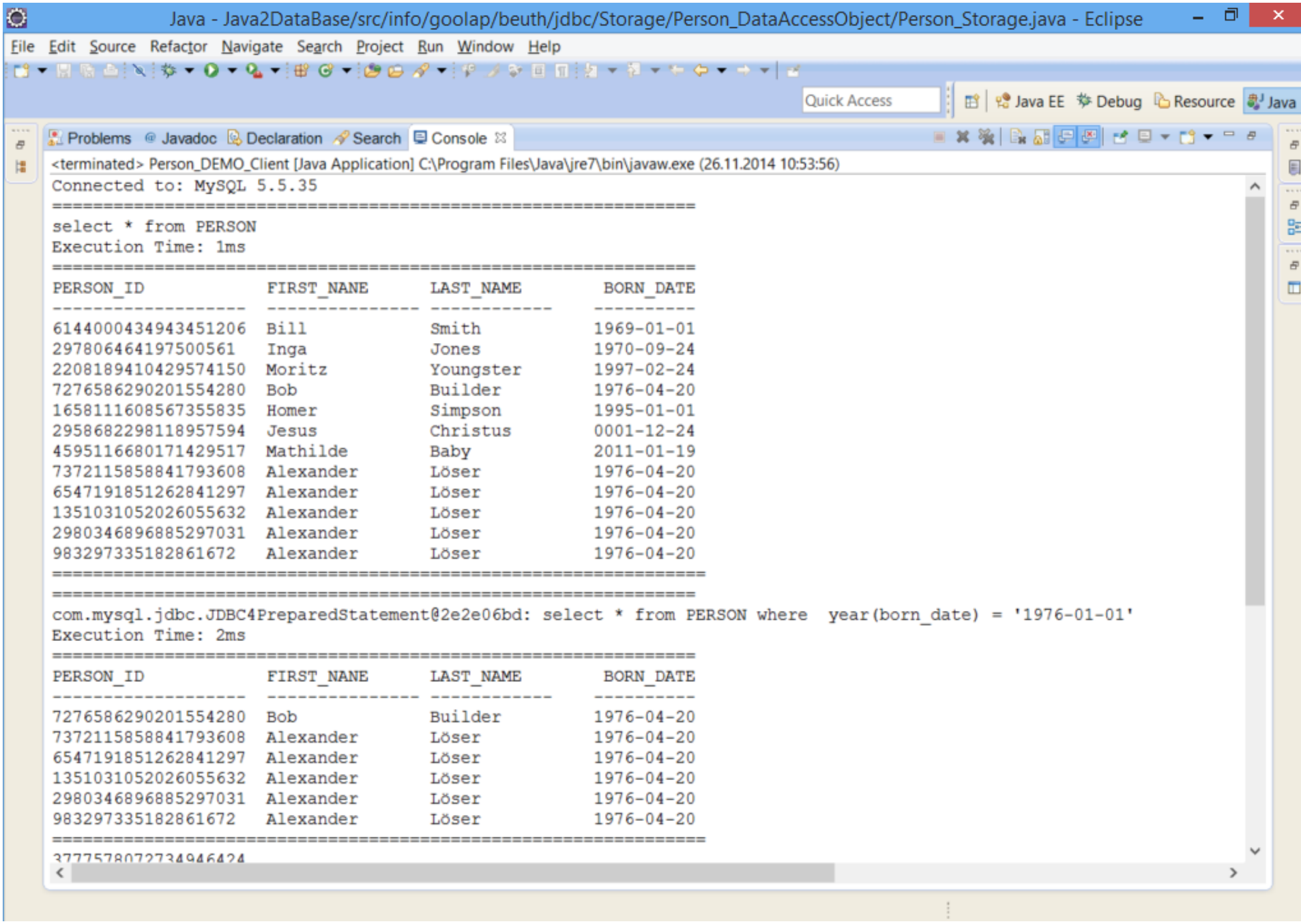
```
/** This method prints out the records of table person. Note the try
 *  and catch. Virtually all JDBC methods throw a
 *  SQLException that must be tended to. The connection
 *  object is used to create a statement object.
 *  The executeQuery method is used to submit a
 *  SELECT SQL query. The executeQuery method returns a ResultSet object.
 */
public void printAllPersons() {
   String query = Person_SQL.SELECT_STAR_FROM_PERSONS;
   ResultSet rs = null;
   try {
      Statement s = con.createStatement();
      long begin = System.currentTimeMillis();
      rs = s.executeQuery(query);
      long executiontime = System.currentTimeMillis()-begin;
      this.printResultSet(rs,query,executiontime);
      s.close();
   }
   catch(SQLException ex) {
         // handle any errors
         System.out.println("SQLException: " + ex.getMessage());
         System.out.println("SQLState: " + ex.getSQLState());
         System.out.println("VendorError: " + ex.getErrorCode());
         System.exit(-1);
   }
}
```

JDBC Überblick

JDBC stellt Basisoperationen für die DB-Verbindung, zum Typ-Mapping, für statische und dynamische Anfragen und zur Navigation in Resultaten bereit.



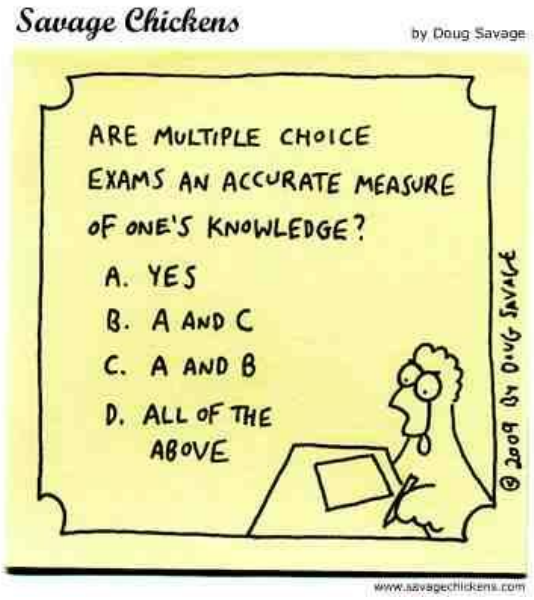

Java - Java2DataBase/src/info/goolap/beuth/jdbc/Storage/Person_DataAccessObject/Person_Storage.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
Java EE Debug Resource Java
IRow.java ExternalPag... Person_DEMO... values_pers... Person_Stor... Person_Logic... create_test...
* This method navigates through the records
* in the ResultSet object (the next method for
* example moves the cursor to the next row; it
* returns false when it runs out of rows) as well
* as methods to access fields in those rows.
* Notice that the person_id fields is of the data
* type 'long', fields name and location are strings
* and field born_date is a date. The ResultSet
* object provides methods to deal with most common data
* types. Please review how other java data types align
* with database data types. The report is
* formatted using the format method introduced in Java 5.
* */
private void printResultSet2Shell(ResultSet rs, String query, long executiontime) throws SQLException
{
System.out.println("==");
System.out.println(query);
System.out.println("Execution Time: " + executiontime +"ms");
System.out.println("==");
System.out.format("%-20s %-15s %-15s %10s%n",
"PERSON_ID", "FIRST_NANE", "LAST_NAME", "BORN_DATE");
System.out.format("%-20s %-15s %-15s %10s%n",
"--------------------", "---------------",
"------------", "----------");
while(rs.next()) {
long person_id = rs.getLong(Person_SQL_Statements.COL_person_id);
String first_name = rs.getString(Person_SQL_Statements.COL_first_name);
String last_name = rs.getString(Person_SQL_Statements.COL_last_name);
Date born_date = rs.getDate(Person_SQL_Statements.COL_born_date);
System.out.format("%-20d %-15s %-15s %10s %n",
person_id, first_name, last_name, born_date);
}
System.out.println("==");
}
Writable Smart Insert 162 : 94


Java - Java2DataBase/src/info/goolap/beuth/jdbc/Storage/Person_DataAccessObject/Person_Storage.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
Java EE Debug Resource Java
Problems Javadoc Declaration Search Console
<terminated> Person_DEMO_Client [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (26.11.2014 10:53:56)
Connected to: MySQL 5.5.35
==========
select * from PERSON
Execution Time: 1ms
==========
PERSON_ID FIRST_NANE LAST_NAME BORN_DATE
6144000434943451206 Bill Smith 1969-01-01
297806464197500561 Inga Jones 1970-09-24
2208189410429574150 Moritz Youngster 1997-02-24
7276586290201554280 Bob Builder 1976-04-20
1658111608567355835 Homer Simpson 1995-01-01
2958682298118957594 Jesus Christus 0001-12-24
4595116680171429517 Mathilde Baby 2011-01-19
7372115858841793608 Alexander Löser 1976-04-20
6547191851262841297 Alexander Löser 1976-04-20
1351031052026055632 Alexander Löser 1976-04-20
2980346896885297031 Alexander Löser 1976-04-20
983297335182861672 Alexander Löser 1976-04-20
==========
==========
com.mysql.jdbc.JDBC4PreparedStatement@2e2e06bd: select * from PERSON where year(born_date) = '1976-01-01'
Execution Time: 2ms
==========
PERSON_ID FIRST_NANE LAST_NAME BORN_DATE
7276586290201554280 Bob Builder 1976-04-20
7372115858841793608 Alexander Löser 1976-04-20
6547191851262841297 Alexander Löser 1976-04-20
1351031052026055632 Alexander Löser 1976-04-20
2980346896885297031 Alexander Löser 1976-04-20
983297335182861672 Alexander Löser 1976-04-20
==========
3777578072734946424

Create your Own Exam: Anfrageoptimierung

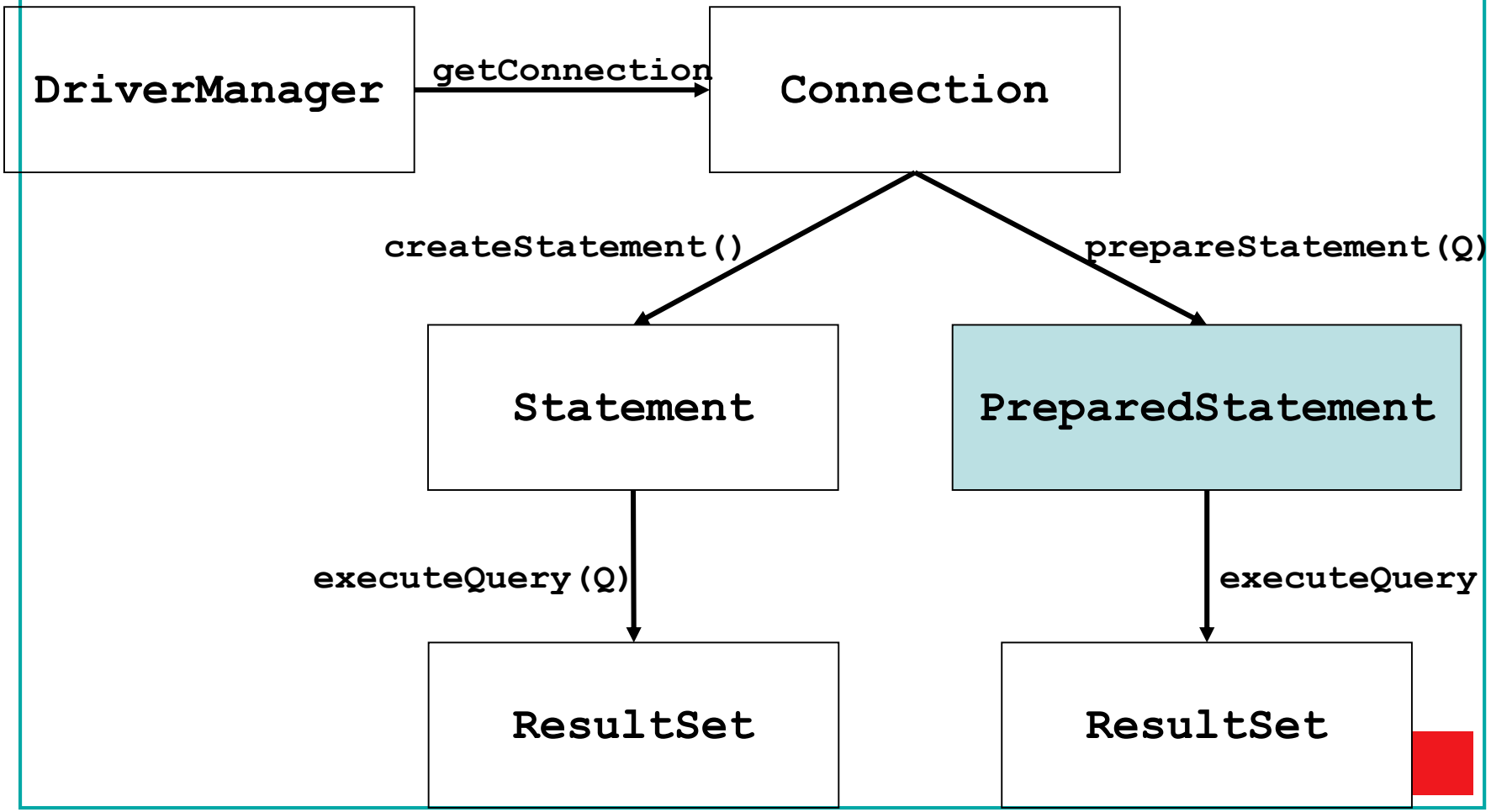
- Bitte erstellen Sie eine Multiple Choice Aufgabe zum Thema Cursor/ JDBC
  - Formulieren Sie eine Frage und 3 Antworten (A, B, C)
  - Davon sollte mindestens eine Antwort richtig und mindestens eine Antwort falsch sein
- Geben Sie die Aufgabe an Ihren rechten Nachbarn. Diskutieren Sie gemeinsam und markieren Sie die richtigen Lösungen
- Geben Sie am Ende der Vorlesung Ihre Aufgabe bei mir ab

5 min



JDBC Überblick

JDBC stellt Basisoperationen für die DB-Verbindung, zum Typ-Mapping, für statische und dynamische Anfragen und zur Navigation in Resultaten bereit.

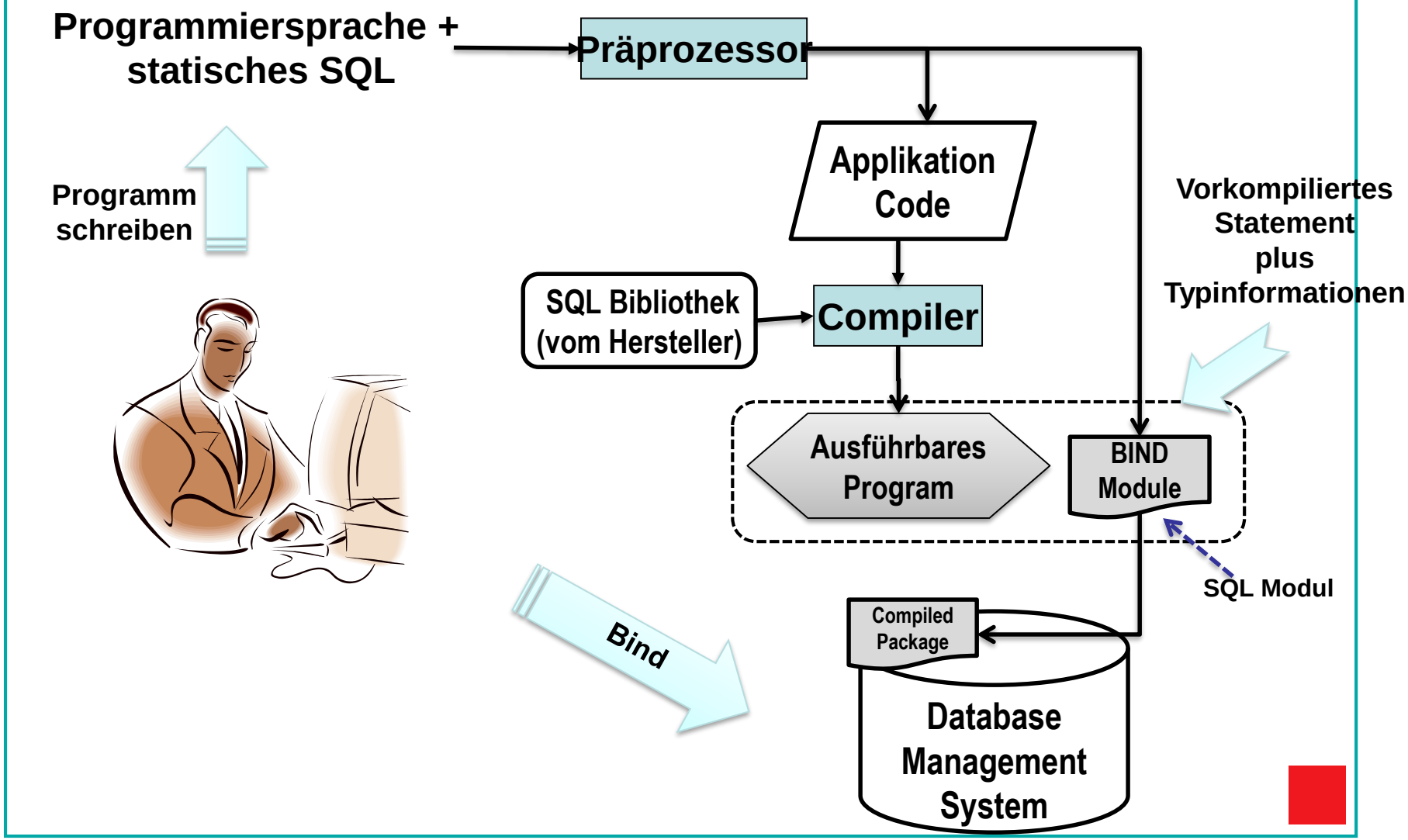


Statisches SQL

Hochtransaktionale Systeme erwarten sehr niedrige Antwortzeiten für einfache Statements in schneller Abfolge. Diese Antwortzeiten erreichen wir durch Umgehung des Overheads für Caches und von Compile Checks.

- Ansatz in JDBC: Prepared Statements
- Statement enthält typischerweise Variablen/Platzhalter für Attributwerte
- Statement wird **einmal kompiliert** wenn die Applikation und die Datenbank verbunden werden (bind-time)
- Applikation hat eine Handle auf das kompilierte Statement und ruft es direkt auf, übergibt Belegung der Variablen
- PREPARED_STATEMENT

Statisches SQL



Statisches SQL: Prepared_Statement

```
  /** This method is used to open the DB and to print out
   * the connection status.
 * @param prepStatement_INSERT_PERSON
   */
  public void openDB() throws SQLException {
      con  = this.connect(DBNAME);
      System.out.println("Connected to: " +
              con.getMetaData().getDatabaseProductName() + " " +
              con.getMetaData().getDatabaseProductVersion()
       );
      prepStatement_INSERT_PERSON = con.prepareStatement(Person_SQL_Statements.INSERT_PERSON);
      prepStatement_SELECT_PERSONS_WITH_YEAR
                  = con.prepareStatement(Person_SQL_Statements.SELECT_PERSONS_WITH_YEAR);
   }
  * @param date
  * This method reads records from the table person for a given date. It executes a prep
  */
public void getPerson_withYear(Date date) {
    ResultSet rs = null;
    try {
        PreparedStatement pst = prepStatement_SELECT_PERSONS_WITH_YEAR;
        pst.setDate(1, date);
        long begin = System.currentTimeMillis();
        rs = pst.executeQuery();
        long executiontime = System.currentTimeMillis()-begin;
        this.printResultSet2Shell(rs, rs.getStatement().toString(), executiontime);
        rs.close();
    }
    catch(SQLException ex) {
            // handle any errors
            System.out.println("SQLException: " + ex.getMessage());
            System.out.println("SQLState: " + ex.getSQLState());
            System.out.println("VendorError: " + ex.getErrorCode());
```

Statisches vs. Dynamisches SQL

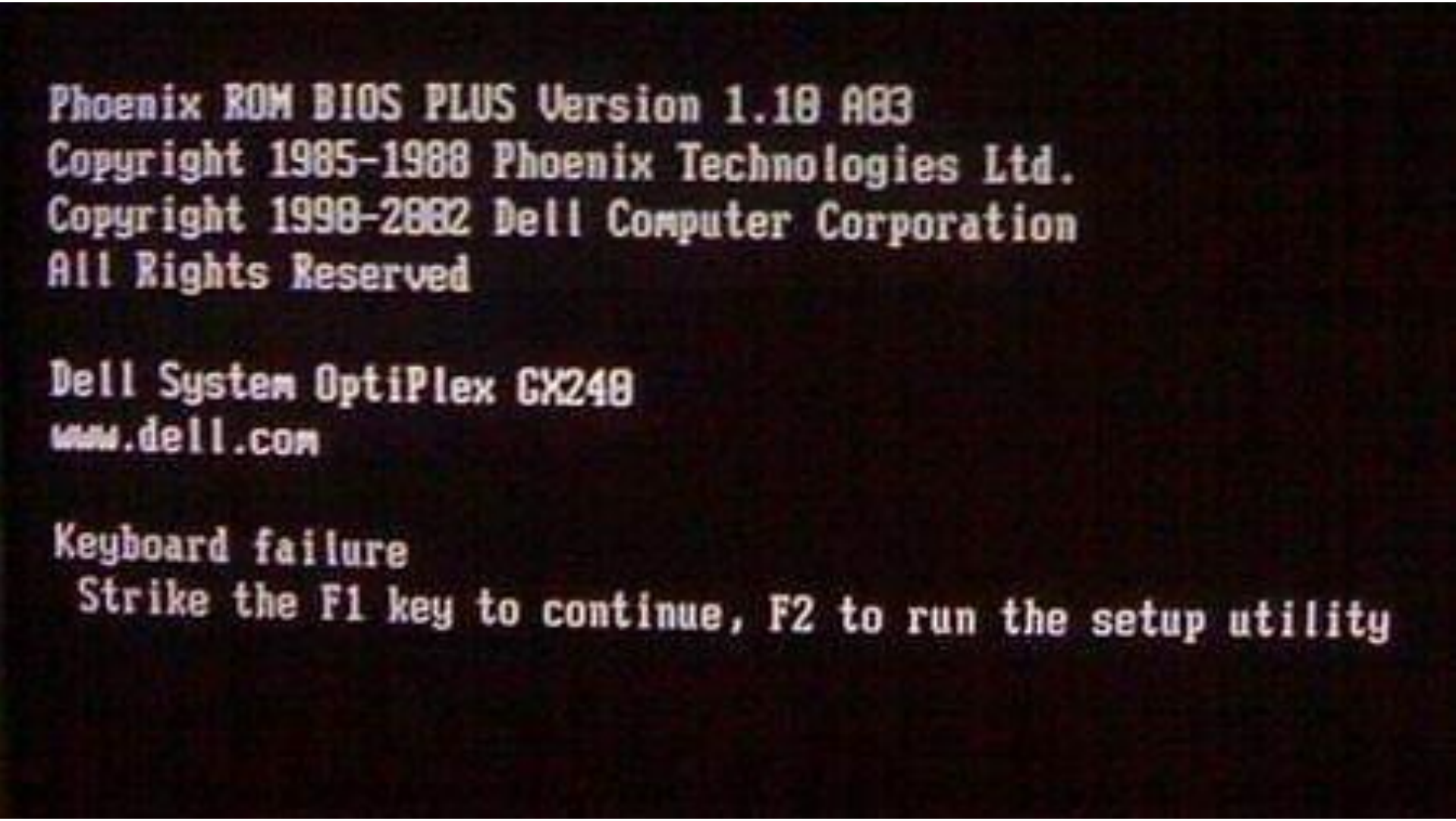
Diese Tabellen geben Orientierungshilfen für die Verwendung statischer oder dynamischer SQL Anfragen mit der JDBC Schnittstelle.

Consideration	Likely Best Choice
Time constraint to run the SQL statement: • Less than 2 seconds • 2 to 10 seconds • More than 10 seconds	 Static Either Dynamic
Data Uniformity • Uniform data distribution • Slight non-uniformity • Highly non-uniform distribution	 Static Either Dynamic
Range (<,>,BETWEEN,LIKE) Predicates • Very Infrequent • Occasional • Frequent	 Static Either Dynamic

Consideration	Likely Best Choice
Repetitious Execution • Runs many times • Runs with highfrequency (transactional) • Ad-Hoc / Runs once	 Either Static Dynamic
Nature of Query • Random • Permanent	 Dynamic Either
Frequency of Statistical Updates • Very infrequently • Regularly • Frequently	 Static Either Dynamic

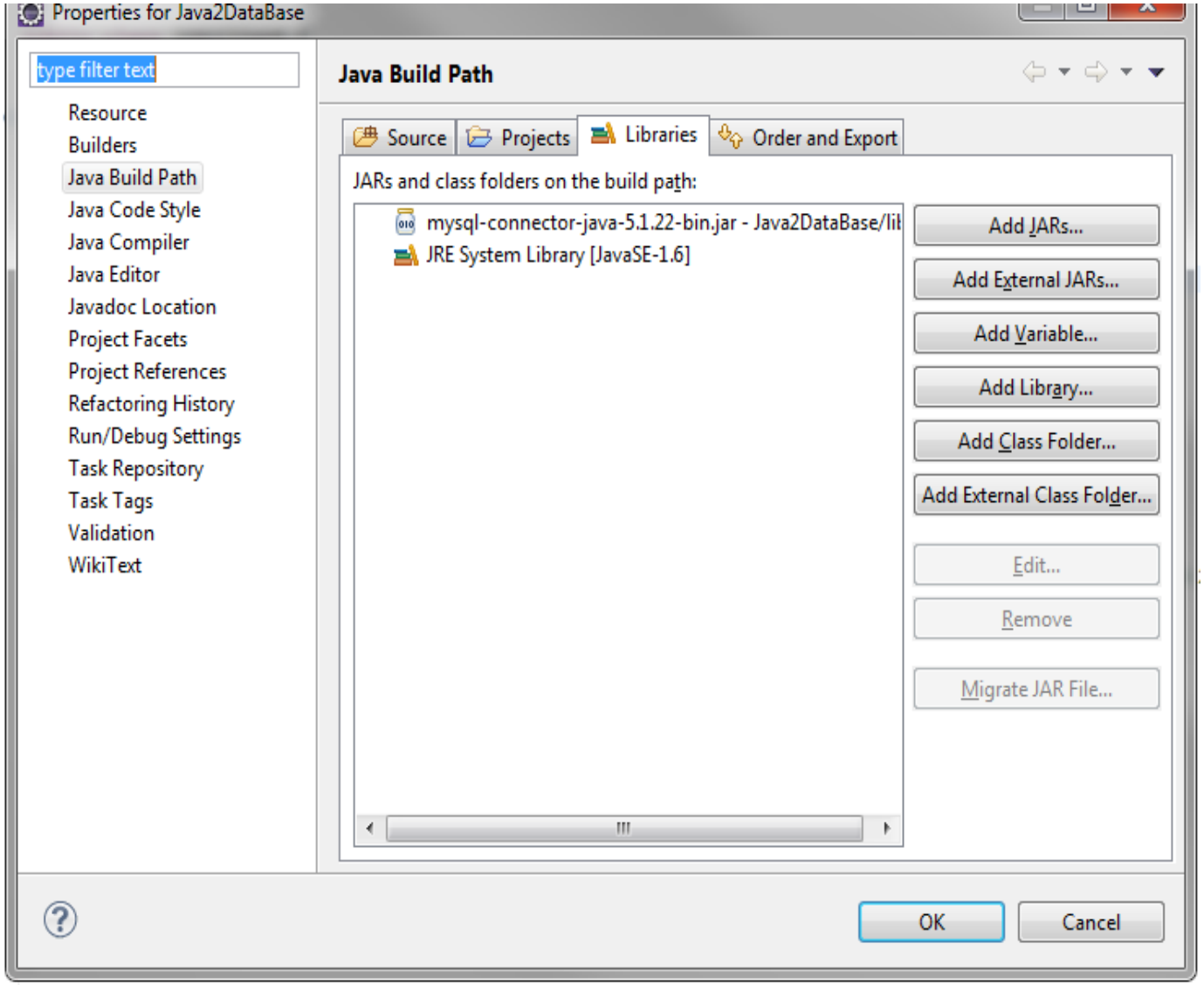
Quelle: Auszug aus der DB2 Dokumentation

IMPLEMENTIERUNG/DEMO



TYPISCHE FEHLERMELDUNGEN

Driver in Java Build_PATH?

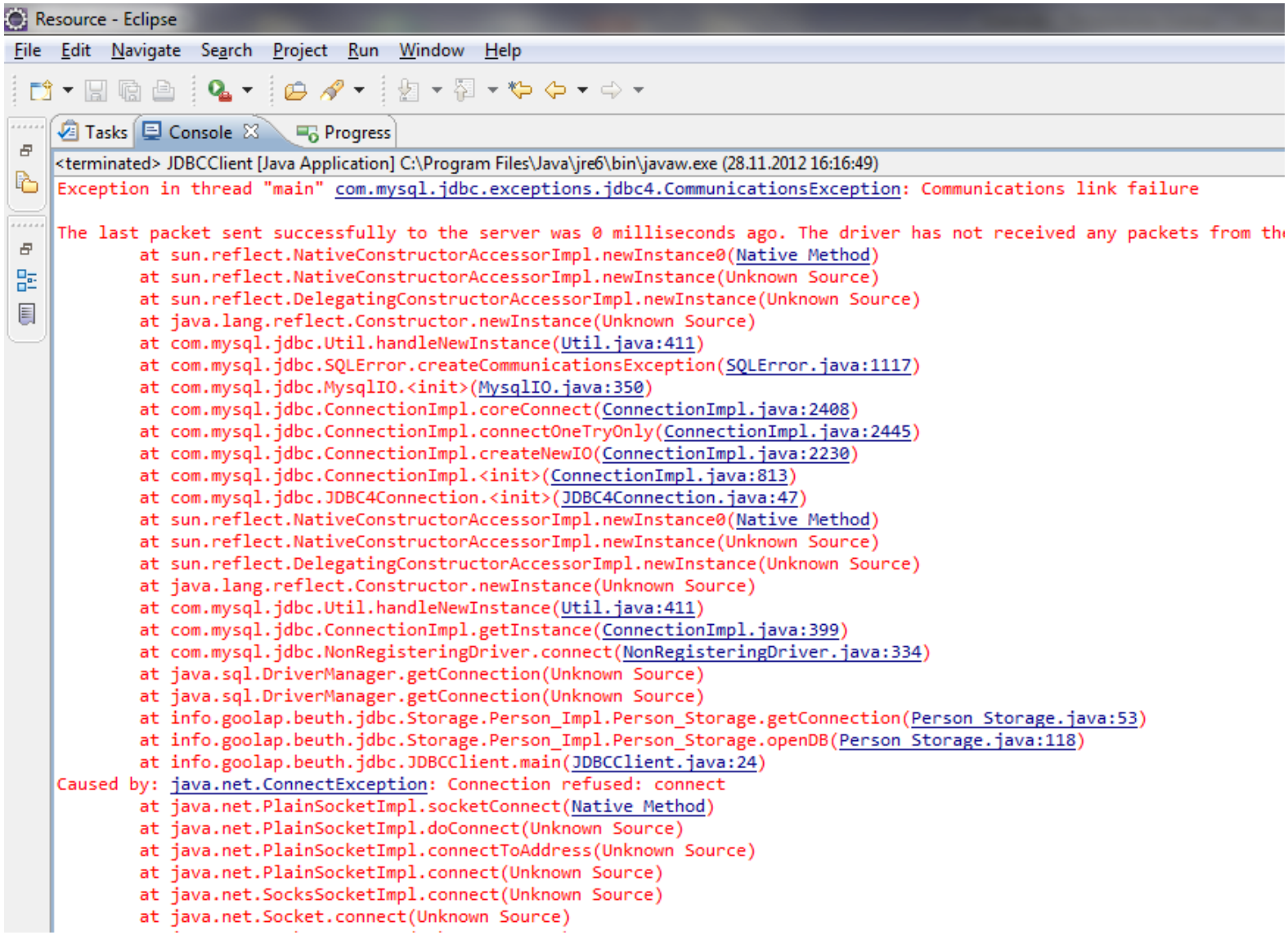


Try … Catch .. Exception

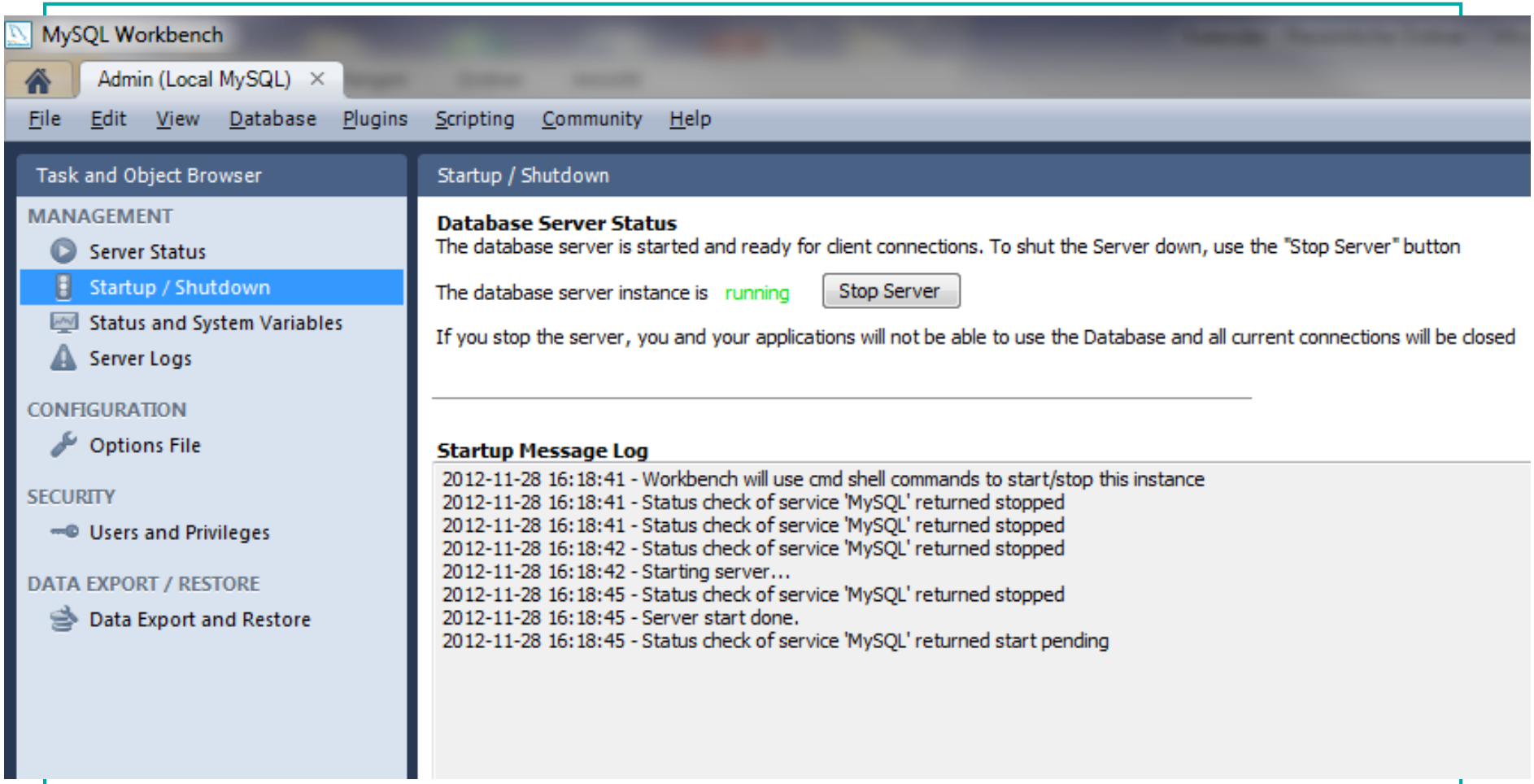
Die Klasse SQLException bietet Getter Methoden für die Nachricht, den SQL STATE und den genauen Error Code.

```
    try {
       Statement s = con.createStatement();
       long begin = System.currentTimeMillis();
       rs = s.executeQuery(query);
       long executiontime = System.currentTimeMillis()-begin;
       this.printResultSet2Shell(rs,query,executiontime);
       rs.close();
       s.close();
    }
    catch(SQLException ex) {
          // handle any errors
          System.out.println("SQLException: " + ex.getMessage());
          System.out.println("SQLState: " + ex.getSQLState());
          System.out.println("VendorError: " + ex.getErrorCode());
          System.exit(-1);
    }
}
```

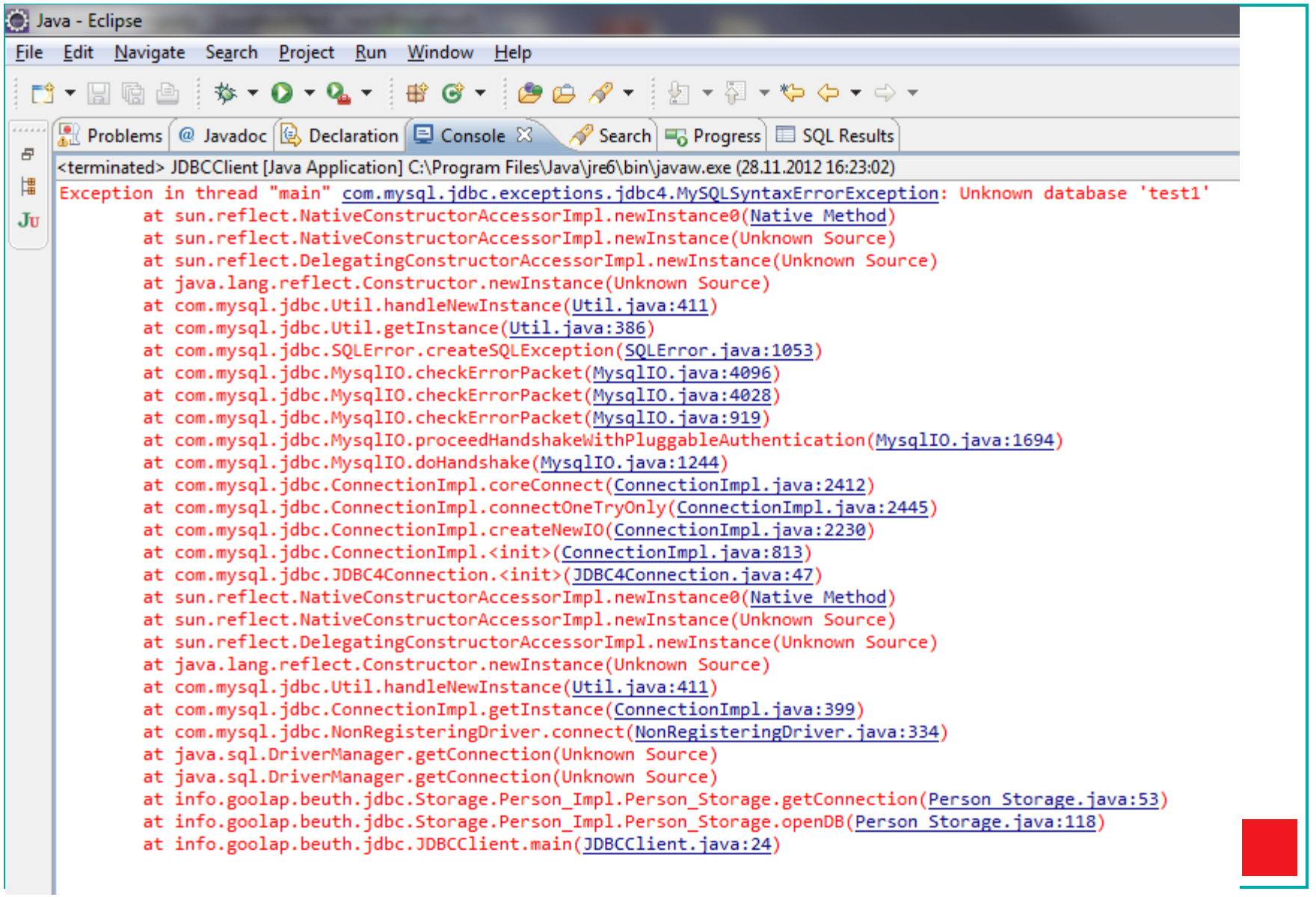
Läuft der Server Prozess?



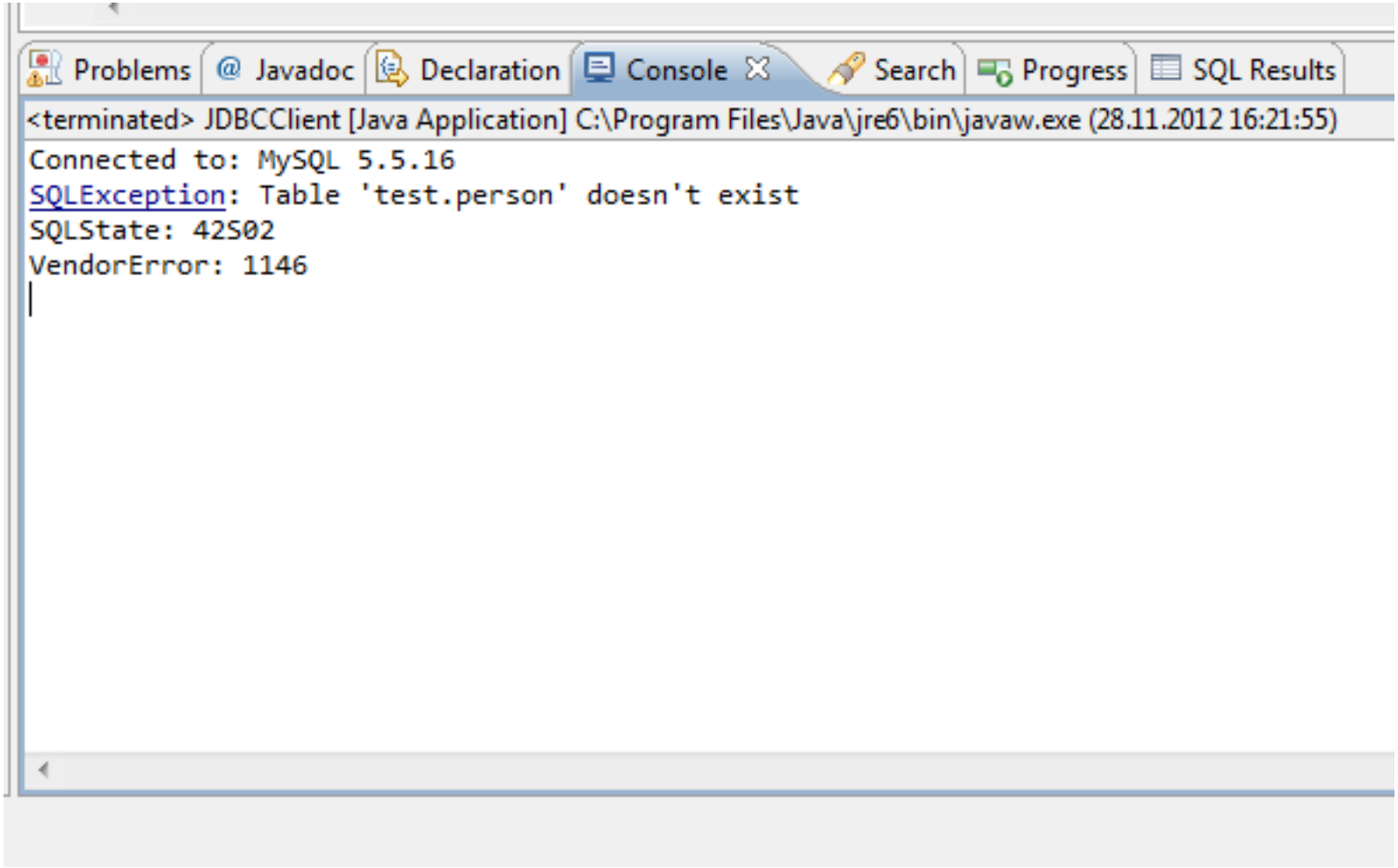
Läuft der Server Prozess?



Sind wir auf der richtigen Datenbank?



Existiert die Tabelle wirklich?



TAKE AWAY MESSAGE

Zusammenfassung

JDBC ermöglicht die Abbildung und die Persistenz von Objekten und deren Referenzen in einem relationalen DBMS.

- Paradigmenbruch: OO/ Java und SQL Welt
- JDBC bietet Persistenz und Zugriff auf das RDBMS mit Java
- Abhängig von Ihren Anforderungen: Statements und Prepared Statements
- SQL Exception bietet Information zu typische Fehlerquellen

- Mögliche Vertiefungen: Persistenz – Frameworks, Transaktionen, Optimierungen, Callable Statements, Verteilte Anfragen …

Sie erinnern sich: Ihre Aufgaben

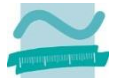
- **Aufgabe 1 Anfragen & Modellierung“**
  Denken Sie mal darüber nach, welche Anfragen Sie an die AOL Daten stellen möchten. Bitte Sie bitte ein logisches und physisches Schema zur Beantwortung dieser Anfragen.
- **Aufgabe 2 „SQL und Anfrageausführung“**
  Bitte formulieren Sie für Ihre Analyseideen aus 1.) die SQL Anfragen. Sie verstehen auch Möglichkeiten der Anfrageausführung bzw. Optimierung.
- **Aufgabe 3 „Datenintegration“**
  Zur Ausführung der Ausführung fehlen Ihnen noch externe Daten, z.B. aus dem Internet Archive, DMOZ oder Freebase.org. Bitte ergänzen Sie Ihr Schema und die Datenbasis.
- **Aufgabe 4 „Analyse, Erkenntnisgewinn und Wert“**
  Stellen Sie in 5 Minuten die wichtigsten Erkenntnisse aus den Daten vor. Bewerten Sie den Erkenntnisgewinn, z.B. gegenüber Ihren Kommilitonen oder der Literatur! Welche Erkenntnisse hätten einen kommerziellen Wert?

Die Themen

- Was sind Datenbanken?
  - Motivation, Historie, Datenunabhängigkeit, Einsatzgebiete
- Datenbankentwurf im ER-Modell & Relationaler Datenbankentwurf
  - Entities, Relationships, Kardinalitäten, Diagramme
  - Relationales Modell, ER -> Relational, Normalformen, Transformationseigenschaften
- Relationale Algebra & SQL
  - Kriterien für Anfragesprachen, Operatoren, Transformationen
  - SQL DDL, SQL DML, SELECT … FROM … WHERE …
- Datenintegration & Transaktionsverwaltung
  - JDBC, Cursor, ETL
  - Mehrbenutzerbetrieb, Serialisierbarkeit, Sperrprotokolle, Fehlerbehandlung, Isolationsebenen in SQL
- Ausblick
  - Map/Reduce, HDFS, Hive …
  - Wert von Daten

Referenzen & HowTos

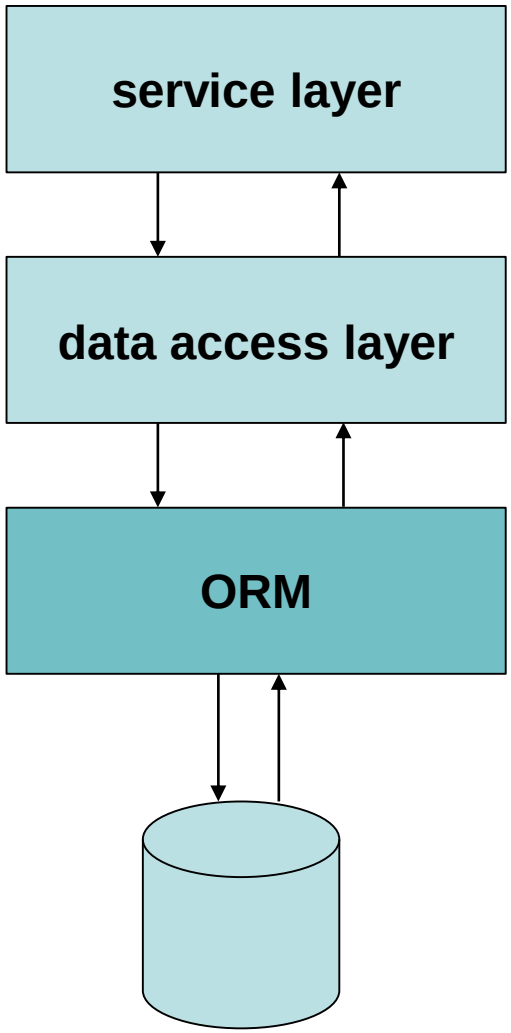
- SUN Java 1.8 API
  - http://docs.oracle.com/javase/8/docs/api/
  - Package java.sql
- Ausführliches Java Buch online - “Java ist auch eine Insel”
  - http://openbook.rheinwerk-verlag.de/javainsel/
  - Auch verfügbar als PDF:
    http://download.galileo-press.de/
    openbook/javainsel8/galileocomputing_javainsel8.zip
- SQLyog: http://code.google.com/p/sqlyog/

EXKURS: OBJECT-RELATIONAL MAPPING

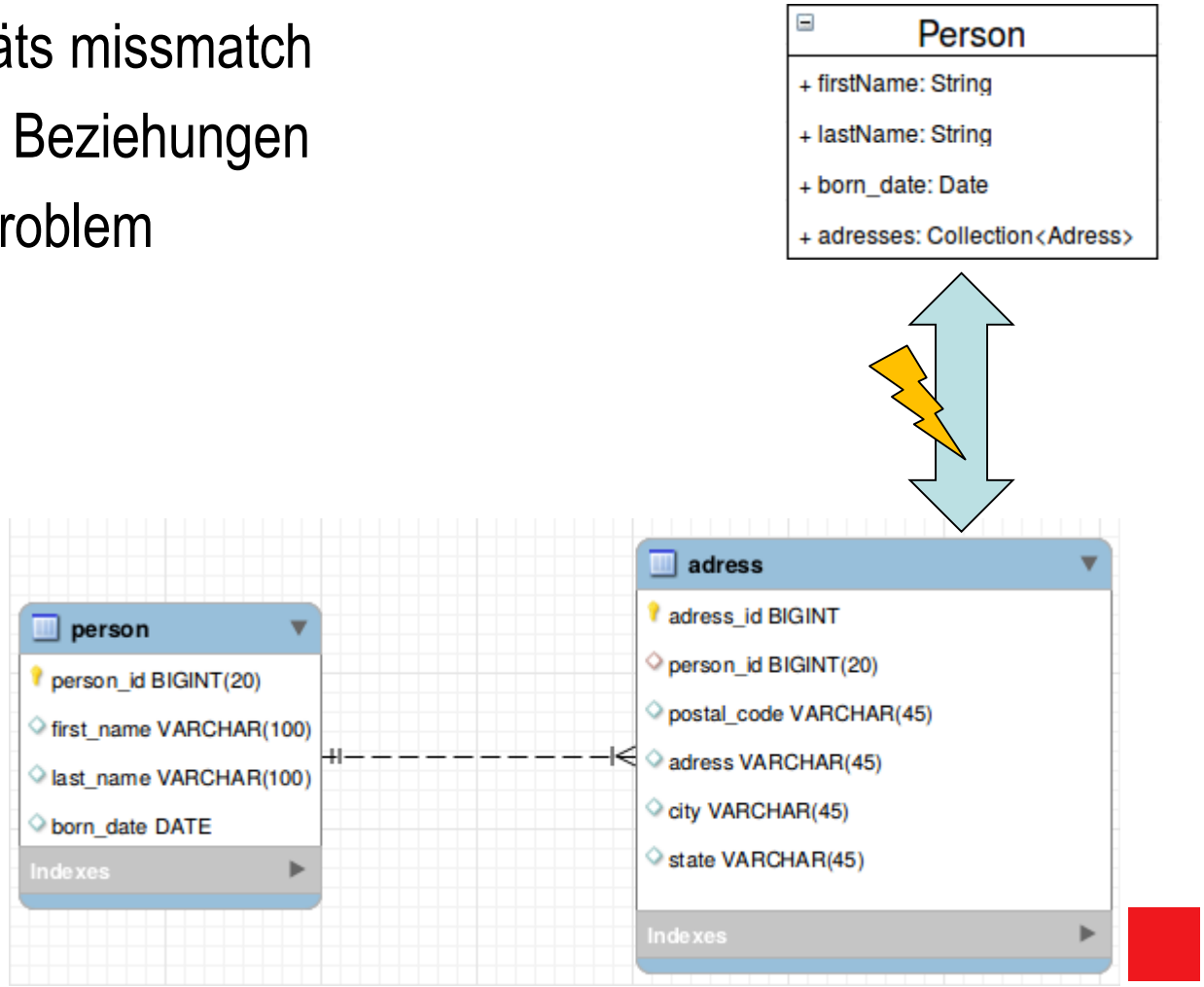
Exkurs: Was ist ORM?

- Lösungsansatz für „Object-relational impedance mismatch“
- Abstraktion der Datenpersistierung
- Fokus des Entwicklers auf Geschäftslogik
- Unabhängigkeit von den Spezifika der Datenbanken



Exkurs: Was ist ORM?

- Durch ORM gelößte Probleme
  - Granularitäts missmatch
  - Gerichtete Beziehungen
  - Identitätsproblem
  - Vererbung



Exkurs: ORM - Eigenschaften

- Was bringt ein ORM Framework mit?
  - API für Metadata definition
  - API für CRUD-Operationen
  - Möglichkeit um Änderungen an Domainobjekten zu verfolgen um “lazy associations“ zu realisieren
  - Query/Criteria API für Suchanfragen und spezielle Datenbankoperationen
- Der Einsatz von ORM kann sehr schwierig und komplex werden.
- **Nicht in allen use-cases sinnvoll!**

Exkurs: ORM – JPA Beispielcode

```java
/**
 * A working shift.
 */
@Entity
public class Shift implements Serializable {

    @Id
    @GeneratedValue
    private Long id;
    /**
     * Employees which are assigned to this shift.
     */
    @ManyToMany(targetEntity = Employee.class, mappedBy = "assignedShifts", fetch = FetchType.EAGER,
            cascade = {CascadeType.MERGE, CascadeType.PERSIST, CascadeType.DETACH})
    private List<Employee> associatedWorker = new ArrayList<>();

    /**
     * Name of the shift.
     */
    private String name;
```


EXKURS: JDBC MIT ORACLE

JDBC – Aufbau einer DB-Verbindung

- **Importieren** der notwendigen **Klassen**: `import java.sql.*;`
- **Laden** des JDBC-**Treibers**

```
Class.forName („oracle.jdbc.driver.OracleDriver“);// Oder:
DriverManager.registerDriver (new
   oracle.jdbc.driver.OracleDriver());
```

- **Verbindungsaufbau** zur DB

```
Connection my_con =
   DriverManager.getConnection(url,user,pwd);
```

- **Erzeugen** eines **Statements**

```
Statement my_stmt = my_con.createStatement();
String query = „select kdnr,firma from kdst“;
```

- **Ausführen** eines Statements

```
ResultSet my_result=my_stmt.executeQuery(query);
```

- **Resultate** verarbeiten

```
while(my_result.next()) {
   int nr = my_result.getInt(1);
   String kd_name = my_result.getString(„firma“); }
```

- **Abmelden** von der DB: `my_con.close();`

JDBC – DML-Statements

```
try { // Verbindungsaufbau
  String url = "jdbc:oracle:thin:@dbl48.beuth-
  hochschule.de:1521:oracle";
// oder "…@localhost:xxxx:oracle" -> bei Nutzung Putty
// oder "…@localhost:1521:xe"-> bei lokaler XE-Inst.

  String user = "matinf";
  String pwd = "xxx";
  con = DriverManager.getConnection (url,user,pwd);

  String ddlop = "insert into artgru (gruppe,grup_txt)
  values (11,'Notebook')";
  stmt = con.createStatement();
  int anzahl = stmt.executeUpdate (ddlop);
  System.out.println(anzahl + "Zeilen hinzugefügt");
}
```

JDBC – DDL-Statements

```
try { // Verbindungsaufbau
…
    con = DriverManager.getConnection (url,user,pwd);

// Tabelle ändern
    String ddlop = "alter table artgru add aufvolum_min number
    check (aufvolum_min > 500)";

    stmt = con.createStatement();
    int anzahl = stmt.executeUpdate (ddlop);
    System.out.println("Alter Table erfolgt");

// Default-Wert hinzufügen
    ddlop = " insert into artgru (gruppe,aufvolum_min) values
    (11,400)";
    stmt = con.createStatement();
    anzahl = stmt.executeUpdate (ddlop);
}
```

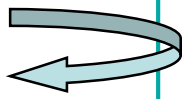
Was passiert hier? 400?

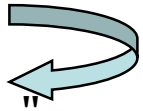
JDBC – Arten von Statements

- **Parameterlose SQL-Operationen**
  - Klasse Statement
    - createStatement();
    - executeQuery(query);
    - executeUpdate(query);
- **Vorübersetzte, parametrisierte SQL-Op.**
  - Klasse PreparedStatement
    - prepareStatement(query);
    - ParameterÜbergabe: setInt();…
    - Parameterlose execute-Methoden
    - NullwertÜbergabe: setNull(1,java.sql.Types.INTEGER); …
- **Gespeicherte Prozeduren / Funktionen**
  - Klasse CallableStatement
    - prepareCall("call aktual_umshaben()");
    - execute();

JDBC – PreparedStatements

```
PreparedStatement stmt = null;
try { // Verbindungsaufbau
  con = DriverManager.getConnection (url,user,pwd);
/* Prepared Statement 1
int aufnummer = 1;
String dmlop = "delete from aufpos where aufnr = ?";
stmt = con.prepareStatement(dmlop);
stmt.setInt(1,aufnummer);*/
// Prepared Statement 2, Arbeit mit Nullwerten
String dmlop = "insert into artgru  values (12,?)";
stmt = con.prepareStatement(dmlop);
// setNull(int index, int jdbcType)
stmt.setNull (1,java.sql.Types.VARCHAR);
int anzahl = stmt.executeUpdate ();
System.out.println("DML-Op. durchgeführt für " + anzahl + "
  Zeilen" );
//+ "oder auch " + stmt.getUpdateCount());
  }
```





JDBC – CallableStatements

```
CREATE OR REPLACE  PROCEDURE
    „MATINF"."AKTUAL_UMSHABEN"   is
   cursor kunden is
     select kdnr,umshaben from kdst
     where not exists (select * from aufkopf
     where kdst.kdnr = aufkopf.kdnr)
   order by kdst.kdnr
 for update of umshaben;
     begin
     for kunden_rec in kunden
     loop
         update kdst set umshaben = 0 where current of kunden;
     end loop;
end;
```

JDBC

```
CallableStatement stmt = null;
try { // Verbindungsaufbau
  con = DriverManager.getConnection (url,user,pwd);
stmt = con.prepareCall("call aktual_umshaben()");
stmt.execute();
int anzahl = stmt.getUpdateCount();
System.out.println("StoredProcedure durchgeführt für " + anzahl + "
  Zeilen" );
  }
```

StoredProcedure mit ResultSet:

```
CallableStatement stmt = null;
try { // Verbindungsaufbau
  con = DriverManager.getConnection (url,user,pwd);

stmt = con.prepareCall("call aktual_umshaben()");
stmt.registerOutParameter (1, OracleTypes.CURSOR);
stmt.execute();
ResultSet rs =((OracleCallableStatement) stmt).getCursor (1);

while (rs.next()) {
  System.out.println("kdnr : " + rs.getInt(1));
  System.out.println("umshaben: " + rs.getInt(2));
// Achtung: keine weiteren Spalten in ResultSet
}
}
```

JDBC – Arten von ResultSets

- **Positionierbare ResultSets**
  - Einstellung über Parameter rsTyp
    - Defaultwert: TYPE_FORWARD_ONLY
    - TYPE_SCROLL_xxxx
- **Dynamische ResultSets** (Sichtbarkeit von Änderungen)
  - Einstellung über Parameter rsTyp
    - TYPE_SCROLL_INSENSITIVE
    - TYPE_SCROLL_SENSITIVE
- **Änderbare ResultSets**
  - Löschen / Ändern des aktuellen Tupels
  - Einfügen eines neuen Tupels
  - Einstellung über Parameter rsConcurrency
    - Defaultwert: CONCUR_READ_ONLY
    - CONCUR_UPDATABLE
- **Methoden**
  - createStatement(int rsTyp, int rsConcurrency)
  - prepareStatement(String sql, int rsTyp,int rsConcurrency)
  - prepareCall(String sql,int rsTyp,int rsConcurrency)

JDBC – Änderbare ResultSets

- **Positionieren** im ResultSet
  - beforeFirst(), afterLast(), previous() etc.
  - Absolute(int i), relative(int i)
- **ArrayFetches**
  - setFetchSize(int i), Default: i = 10
  - (setFetchDirection(FETCH_FORWARD | FETCH_REVERSE | FETCH_UNKNOWN))
- **Änderbare ResultSets**
  - Zweistufiges Update:
    - update*Typ(*int index*, Typ* wert*)* – Vorbereitung Update
    - updateRow() – eigentliches Update
    - deleteRow(), insertRow() – delete / insert

Achtung:

änderbares ResultSet -> änderbare Sicht (keine Join-/Aggregierungs-/Projektionssicht)

Bei Konflikt zwischen Einstellung rsConcurrency und Schema des ResultSet -> automatisches Herabstufen der Änderbarkeit durch Treiber

JDBC – Änderbare ResultSets

```
try { // Verbindungsaufbau …
  String query = "select k.kdnr,firma from kdst k where k.kdnr > ? and ort like
  ?";
  stmt = con.prepareStatement(query,
  ResultSet.TYPE_SCROLL_SENSITIVE,
  ResultSet.CONCUR_UPDATABLE);

// Index des Parameters beginnt mit 1
  stmt.setInt (1, 200);
  stmt.setString (2, "%München%");
  ResultSet rs = stmt.executeQuery ();
  if (rs.isBeforeFirst()) System.out.println("****");
  while (rs.next()) {
    if (rs.isFirst())
          {System.out.println("1. Zeile:" + '\n');
          System.out.println("Firma vor update: " + rs.getString(2) + "( " +
          rs.getRow() + ".    Zeile)");
          rs.updateString(2, "TestName");
          //rs.deleteRow();
          rs.updateRow();
          rs.first();
          System.out.println("Firma nach update: " +        rs.get
          rs.first();          }
    if (rs.isLast()) System.out.println("Letzte Zeile       im ResultSet:" + "( "
  + rs.getRow() + ".Zeile)" + '\n');
    int nr = rs.getInt(1);
    String name = rs.getString(2);
    System.out.println(rs.getRow() + ". Kunde:  " + nr + "   " + name + "   " );
  }
}
```

Setzen der Werte

Update persistent

Positionieren