Relationales Datenbankpraktikum

Dr. M. Hartung, L. Kolb

Aufgaben 2 und 3 – Agenda

- Überblick zur Aufgabenstellung
- Materialien

- XML-Verarbeitung
- Datenbanksystem DB2
- Datenbankzugriff mittels Java-Programm
- DB2-XML-Funktionen
- Hibernate

Organisation

Jede Gruppe erhält einen Account (dbprak01, …)

- Für Informatik-Domäne (ssh auf userv1-5, Pools 4. Etage Augusteum)
- Für Datenbankverbindung

Zu importierende Daten (zwei Verfahrensvarianten)

- Download von Praktikums-Webseite „Teil 2“
- Alternative für WFB: XML Import von dbprak21.XMLDATEN

Dokumentation zu den Themen

- Literaturverzeichnis auf Praktikums-Website

Testate

- Lauffähige Programme auf einem Rechner Ihrer Wahl
- “Unsere” DB2-Installation
- Termine siehe Praktikumswebsite und Absprache Betreuer

Rückblick Aufgabe 1: Modellierung

Miniweltbeschreibung und *mögliche* Fragen beachten!

- Entity Relationship Diagramm erstellen
  - Entity-Mengen
  - Attribute
  - Beziehungen
  - Primärschlüssel (auch über mehrere Attribute)
  - evlt. Datentypen
  - künstliche ID-Attribute nur in Ausnahmefällen
  - ER Diagramm kennt keine Fremdschlüssel, nur Beziehungen!

- Transformation in Relationenmodell
  - Entitity-Mengen zu Relationen
  - N:M-Beziehungen zu eigene Relationen
  - ER-Beziehungen zu Fremdschlüssel

Aufgaben 2 und 3

Aufgabe 2: Datenimport und -aufbereitung

- Schema laden
  - Datei „schema.sql“ anlegen
- Daten laden
  - Entweder aus Datenbank dbprak21 oder aus XML- und CSV-Dateien
  - Korrektheits-/Konsistenzprüfung, Abhängigkeiten/Ladereihenfolge
- SQL Anfragen
- Logging-Mechanismus für Änderungen entwerfen/implementieren

Aufgabe 3: Anwendung

- Anbinden der Datenbank an gegebene Java-Applikation (GUI)
- Realisierung einer Middleware unter Verwendung von Hibernate

Werkzeuge für SQL und Java

- DB-Schema-Viewer, interaktive SQL-Tools
  - IBM Data Developer Workbench (Eclipse-basiert, 500MB)
  - Execute Query (Download via executequery.org)
  - SquirreL SQL Client (Download via squirrel.org)
- IDEs zur Java-Entwicklung
  - Eclipse, Netbeans, IntelliJ, VIM, Emacs, ...
- Bei Bedarf: eigene DB2-Installation DB2-Express

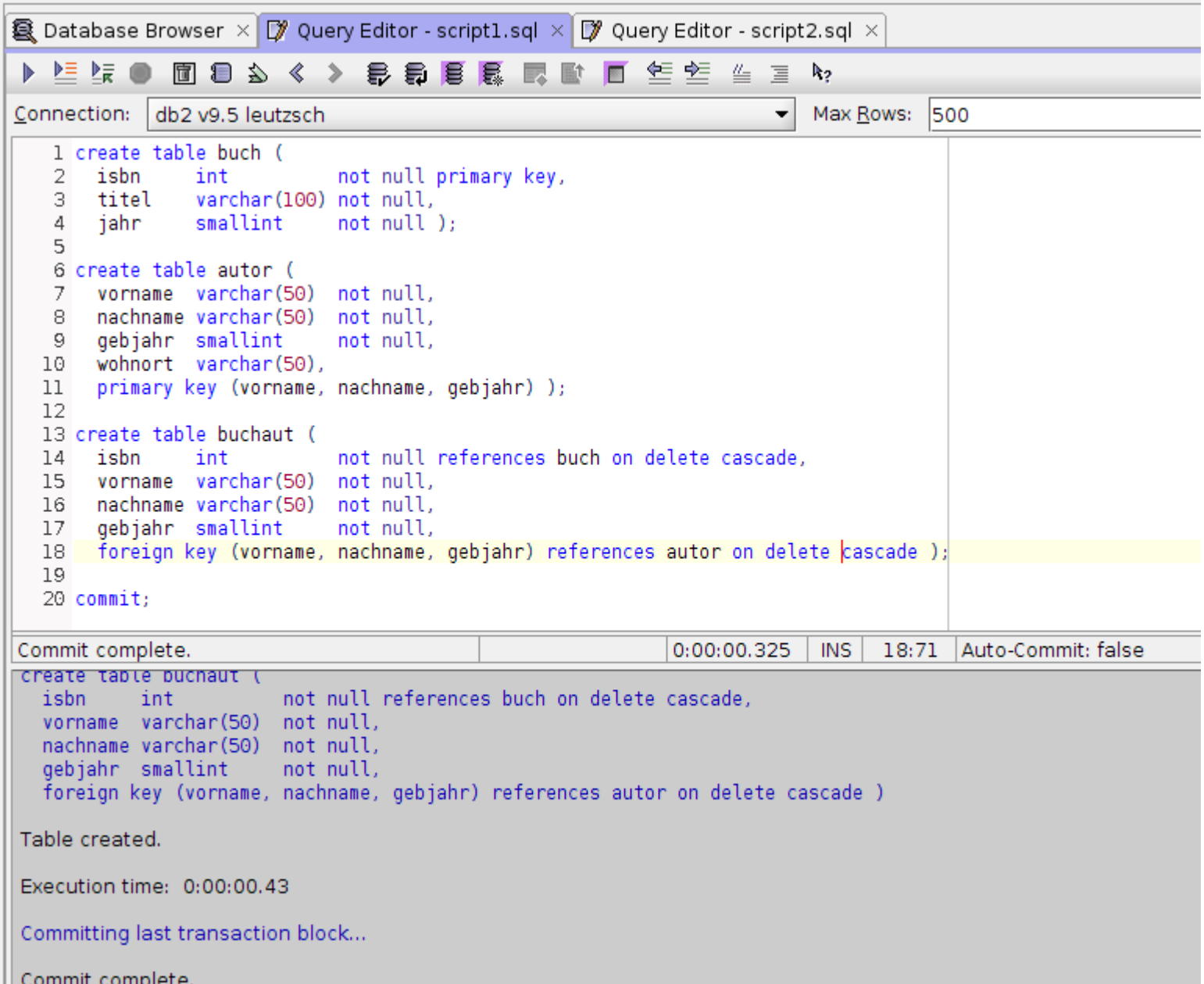
Datenbanksystem: DB2

- Technische Daten

	WFB	MS
DBS	DB2 V9.5 for UNIX	DB2 V10 for UNIX
Server	leutzsch.informatik.uni-leipzig.de	anger.informatik.uni-leipzig.de
DB	PRAK13A	PRAK13B
Port	50001	50001
Nutzer	dbprak01 - dprak10	dbprak11 - dbprak21

- Zugriff auf Datenbank via JDBC
  - Treiberdateien (JAR-Files) Download von Praktikums-Website
  - Treiberdateien in ExecuteQuery/SQirreL/IBM Bench einbinden
  - **Beispielprogramm DB2Query.java**

SQL DDL Schemadefinition auf DB2

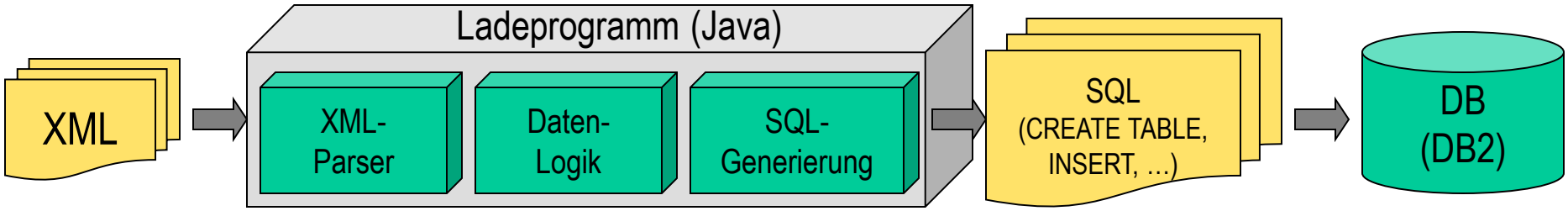


XML: Beispiel Landdaten

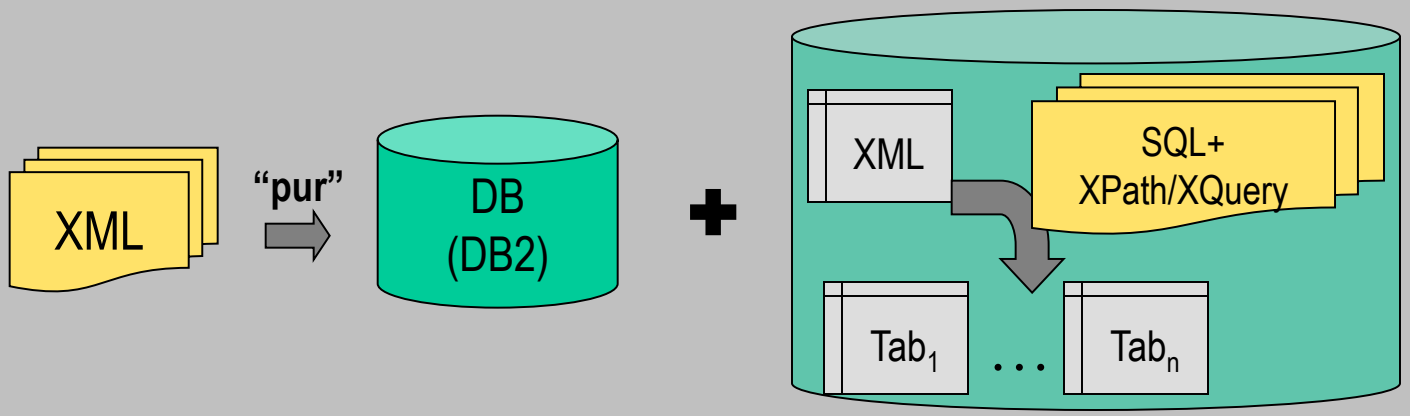
```
<WorldFactbook>
 <country>
  (Beginn der Daten zu einem Land)
  <name>
   (Kurz)-Name des Landes
  </name>
  <Introduction>
   <Background>InfoText</Background>
  </Introduction>
  <Geography>
   <Location>
    Text zur geografischen Lage
   </Location>
  </Geography>
  <Economy>
   <GDP unit="$">
    Bruttoinlandsprodukt in Dollar
   </GDP>
   <Agriculture_products>
    <name>Agrarprodukt 1</name>
    <name>Agrarprodukt 2</name>
     ...
   </Agriculture_products>
  </Economy>
  ...
 </country> (Ende erstes Land)
 <country>   (Beginn naechstes Land)
  ...
 </country>
 ...
</WorldFactbook>
```

Transformation: XML → Relationen

- **Variante 1**: Java-Ladeprogramm



- **Variante 2**: Nutzung der XML-Funktionalität der DB



XML: Verarbeitung mit Java

- Standard-APIs zur Verarbeitung von XML-Dokumenten
  - DOM (Document Object Model)
    - Dokument wird geparsed und als Baumstruktur im Hauptspeicher abgelegt
    - Methoden zur Navigation/Manipulation
  - SAX (Simple API for XML)
    - sequenzielles Parsen und Verarbeiten (auch für größere XML-Daten)
    - Parser ruft call-back-Methoden nach Einlesen eines Dokumentabschnitts auf (Elementstart, Elementende, Textabschnitt, Fehlernachrichten)
- Verarbeitung mit Java mittels Java-XML-Parser
  - hier verwendet: Xerces von Apache
  - Einbinden in CLASSPATH
- Beispiel: SAX
  - Initialisieren des Parsers
  - Festlegen der Klasse, die Call-Back-Methoden implementiert
  - Dokument an Parser übergeben
- **Beispielprogamm XMLParserDemo.java**

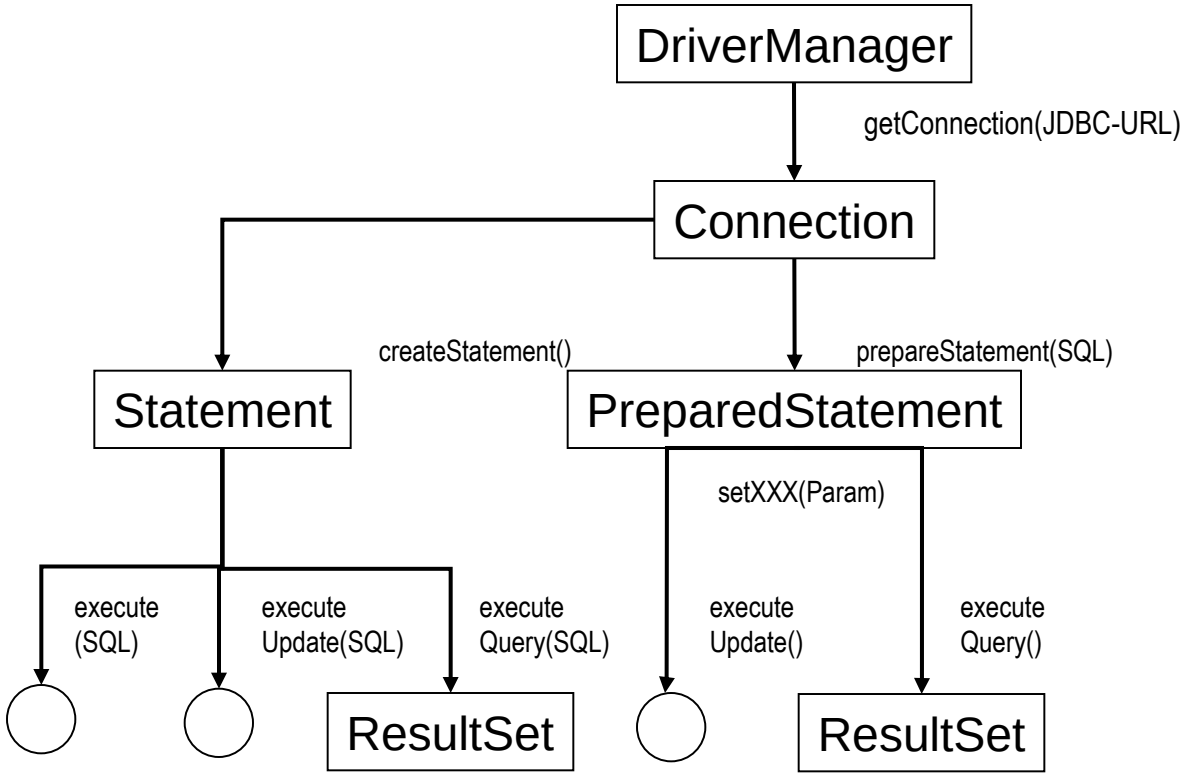
```
public class XMLParserDemo extends DefaultHandler {

  String parserClass = "org.apache.xerces.parsers.SAXParser";

  public void startElement(String namespaceURI, String localName,
                           String rawName, Attributes atts) { ... }

  public void endElement(String namespaceURI, String localName,
                         String rawName) { ... }

  public void characters(char[] ch, int start, int length) { ... }

  public void warning(SAXParseException e) { ... }
  public void error(SAXParseException e) { ... }

  public void doit(String dataFilename) {

    XMLReader parser = null;

    try {
      parser = XMLReaderFactory.createXMLReader(parserClass);
    } catch (SAXException e) {
      System.err.println("Parser Initialisierungsfehler");
      System.exit(1);
    }

    parser.setContentHandler(this);
    parser.setErrorHandler(this);

    try { parser.parse(dataFilename); } catch (SAXException e) {...}

}
```

call-back

init

parse

JDBC Verbindungsaufbau/Anfragen



JDBC: Verbindung zur Datenbank

```
try {
   String jdbcDrv = "com.ibm.db2.jcc.DB2Driver";
   Class.forName(jdbcDrv);
} catch (Exception e) {
   System.err.println("Konnte JDBC-Treiber nicht laden.);
}

Connection con = null; // Verbindungsobjekt zur Datenbank
String url = "jdbc:db2://leutzsch:50001/DBPRAK13A"; // JDBC-URL

try {
   //  Verbindung herstellen
   con = DriverManager.getConnection(url, userid, passwd);
} catch(SQLException e) {
   System.err.println("SQL Fehler aufgetreten: " + e.getMessage());
} finally {
   // Ressourcen freigeben
   try { if (con != null) con.close(); } catch (SQLException e1){};
}
```

JDBC: Anfragen, Query-Statement

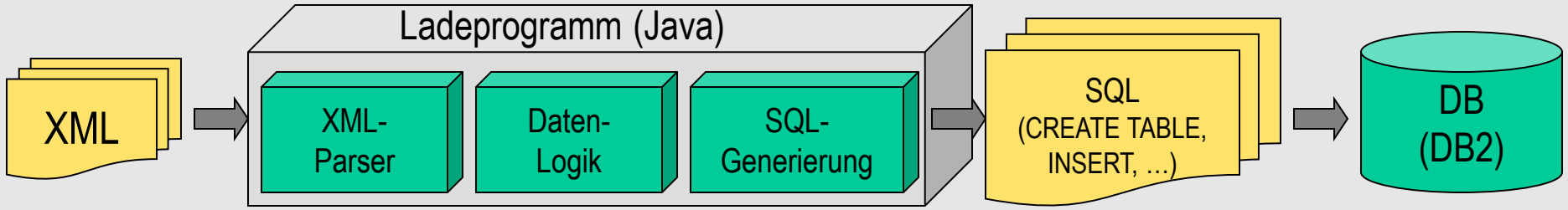
```
Statement stmt = null; // Objekt zum Ausfuehren von Queries
ResultSet rs = null;   // Enthaelt die Query-Ergebnisse
String query = null;

try {
  stmt = con.createStatement();

  query = "SELECT Name FROM dbprak01.Kunde"
  rs = stmt.executeQuery(query);

  while (rs.next()) {  System.out.println(rs.getString(1)); }

} catch(SQLException e) {
  System.err.println("SQL-Fehler: " + e.getMessage());
} finally {
  try { if (stmt != null) stmt.close(); } catch (SQLException e1){};
}
```

JDBC: Prepared-Statement

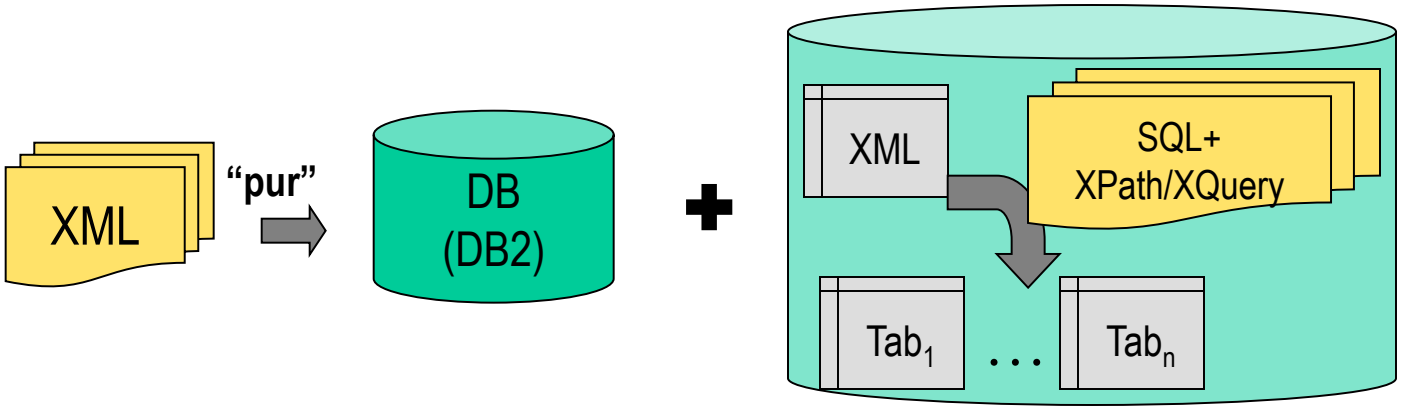
```
PreparedStatement pStmt = null; // Objekt zum Ausfuehren von Queries
ResultSet rs = null;   // Enthaelt die Query-Ergebnisse

try {
  pStmt = con.prepareStatement(
    "SELECT Name FROM dbprak01.Kunde WHERE Wohnort = ?");
  pStmt.setString(1, "Leipzig");
  rs = pStmt.executeQuery();

  while (rs.next()) {  System.out.println(rs.getString(1)); }

} catch(SQLException e) {
  System.err.println("SQL-Fehler: " + e.getMessage());
} finally {
  try { if (pStmt != null) pStmt.close(); }catch(SQLException e1){};
}
```

Transformation: XML → Relationen

- **Variante 1**: Java-Ladeprogramm



- **Variante 2**: Nutzung der XML-Funktionalität der DB



XML in DB: Überblick

Speichern der XML-Dokumente in Datenbank

(für Sie erledigt in Tabelle dbprak21.XMLDATEN)

Anfragen mittels XMLTABLE-Funktion, deren Ergebnis mittels INSERT in (relationalen) Tabellen abgespeichert werden kann

- Speicherung wird durch mehrere Anfragen realisiert
- “Nicht alles auf einmal”, evtl. nur Teil des XML-Dokuments relevant

Anfragen realisieren (befüllen) i.A. “typische Teile” des Datenmodells

- Einfache Attribute
  - Beispiel: Produkt hat ASIN und Titel
- Spezialisierungen
  - Beispiel: Musik-CDs und Bücher sind Produkte mit jeweils spezifischen Attributen
- 1:N-Beziehungen = Vater-Kind-Elemente im XML
  - Beispiel: Eine Musik-CD hat mehrere Songs

XML-Dokumente in der Datenbank

- Tabelle “dbprak21.XMLDATEN” enthalten XML-Dokumente
- Verwendung des DB2-Datentyps “XML”
- Beispiel-Datensatz (Dateiname, XML Inhalt)

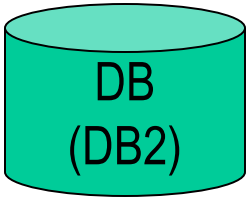
Tabelle dbprak21.XMLDATEN	
datei(VARCHAR)	**dok(XML)**
leipzig	<leipzig><Music asin="B0000668PG“ ean="0721616028027"> <title>In a Pig's Eye: Reflections on the …

Einfache Attribute: Produktdaten

- XMLTABLE-Funktion liefert als Ergebnis relationale Tabelle
  - Kann für INSERT genutzt werden
- Innerhalb von XMLTABLE kann mittels XPath auf den XML-Inhalt zugegriffen werden
- Basiselemente
  - Für jedes Element wird ein Datensatz erzeugt
- Definition des/der XML-Dokumente(s) mittels PASSING

```
SELECT X.ASIN, X.EAN, X.TITLE
FROM dbprak21.XMLDATEN,
XMLTABLE( '$MSXML/leipzig/child::node()'
PASSING dbprak21.XMLDATEN.DOK AS MSXML
   COLUMNS
      "ASIN"    VARCHAR (15)   PATH '@asin',
      "EAN"     VARCHAR (50)   PATH '@ean',
      "TITLE"   VARCHAR (500) PATH 'title'
)AS X
WHERE dbprak21.XMLDATEN.datei='leipzig'
```

Aufgabe 3: Java-Anwendung + DB



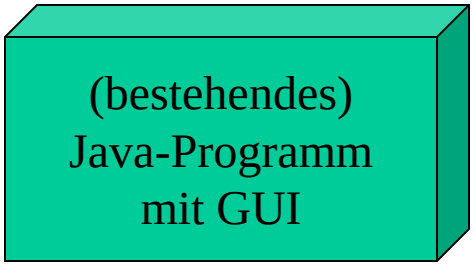
Produkt

Id	Titel	Typ

Angebot

Produkt_Id	Preis	Zustand

```
class Produkt {
  int                 Id;
  string              Titel;
  string              Typ;
  Set<Angebot>        Angebote;
}
```



Relationale Welt

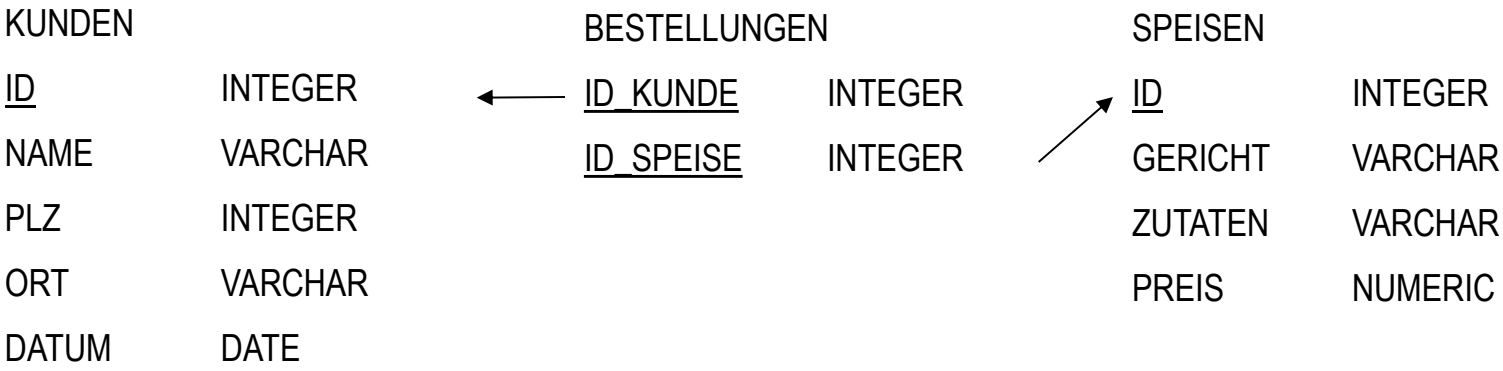
Objektorientierte Welt

Hibernate

- Objekt-Relationales Persistenz-Framework
  - Open-Source-Projekt: www.hibernate.org
  - "Java-Objekt in relationaler Datenbank speichern und laden"

- Objekt-Relationales-Mapping (ORM)
  - Beschreibung durch XML-Konfigurationsdateien
  - Trennung von DB-Anfragen und Java-Code

Hibernate: Beispiel



Datenbank
relational

```
class Kunde {
  int           id;
  string        name;
  int           plz;
  string        ort;
  date          datum;
  Set<Speise>   speisen;

}
```

```
class Speise {
  int           id;
  string        gericht;
  string        zutaten;
  Set<Kunde>    kunden;

}
```

Java
objektorientiert

Hibernate: Beispiel ...

Hibernate.cfg.xml

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
  "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
  "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
  <session-factory>
    <property name="hibernate.connection.driver_class"> ... </property>
    <property name="hibernate.connection.url"> ... </property>
    <property name="hibernate.connection.username"> ... </property>
    <property name="hibernate.connection.password"> ... </property>
    ...
    <mapping resource="Kunde.hbm.xml"/>
    <mapping resource="Speise.hbm.xml"/>
  </session-factory>
</hibernate-configuration>
```

Hibernate: Beispiel ....

Kunde.hbm.xml

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
  "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
  "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
  <class name="mypackage.Kunde" table="KUNDEN">
    <id name="id" column="ID" type="integer">
      <generator class="native"/>
    </id>
    <property name="name"  column="NAME"  type="string" not-null="true"/>
    <property name="plz"   column="PLZ"   type="integer"/>
    <property name="ort"   column="ORT"   type="string"/>
    <property name="datum" column="DATUM" type="date"/>
    <set name="speisen" table="BESTELLUNGEN" lazy="true">
      <key column="ID_KUNDE"/>
      <many-to-many class="mypackage.Speise" column="ID_SPEISE"/>
    </set>
  </class>
</hibernate-mapping>
```

Hibernate: Beispiel .....

Verbindung herstellen

```
import org.hibernate.*;
import org.hibernate.cfg.Configuration;

private SessionFactory sessionFactory;

try {
  System.out.println( "Initializing Hibernate" );
  sessionFactory = new Configuration().configure().buildSessionFactory();
  System.out.println( "Finished Initializing Hibernate" );
} catch( HibernateException ex ) {
    ex.printStackTrace();
    System.exit( 5 );
}
```

Hibernate: Beispiel .....

Speichern

```
try {
  Session sess = sessionFactory.openSession();
  Transaction trx = sess.beginTransaction();

  Kunde kunde = new Kunde();   /* + Werte setzen */
  sess.save( kunde );
  /* + Java-Objekt Speise erzeugen */
  kunde.speisen.add( speise );

  trx.commit();
} catch( HibernateException ex ) {
    if (trx != null)
      try { trx.rollback(); } catch( HibernateException exRb ) {}
    throw new RuntimeException( ex.getMessage() );
} finally {
    try { if( sess != null ) sess.close(); } catch( Exception exCl ) {}
}
```

Hibernate: Beispiel .....

Laden

```
List kunden = sess.createQuery( "from Kunde" ).list();
for( int i=0; i<kunden.size(); i++ ) {
  Kunde kunde = (Kunde)kunden.get( i );
  System.out.println( "Kunde:  " + kunde.name );
  ...
  Iterator itr = kunde.speisen.iterator();
  while( itr.hasNext() ) {
    System.out.print( ((Speise) itr.next()).gericht );
  }
}
```

Zusammenfassung

- Aufgabenstellung & Organisatorisches
  - Accountvergabe

- Hinweise / grober Überblick zu technischen Aspekten
  - XML-Verarbeitung
  - Datenbanksystem DB2
  - Datenbankzugriff mittels Java-Programm
  - DB2-XML-Funktionen
  - Hibernate