



Kommunikation und Datenhaltung

Datenbankanwendungsentwicklung



Überblick über den Datenhaltungsteil

- Einleitung
 - Motivation und Grundlagen
 - Architektur von Datenbanksystemen
- Datenbankabfragen
 - Relationenmodell und Relationenalgebra
 - Relationale Datenbanksprachen (SQL)
- Datenbankentwurf
 - ER- und EER-Modell
 - Abbildung von ER-Modellen auf das Relationenmodell
 - Relationaler Entwurf
 - Sprachen zur Datenbankdefinition
- Anfrageoptimierung
- Transaktionsverwaltung
- **Datenbankanwendungsentwicklung**

Datenhaltung vs. Anwendungssysteme

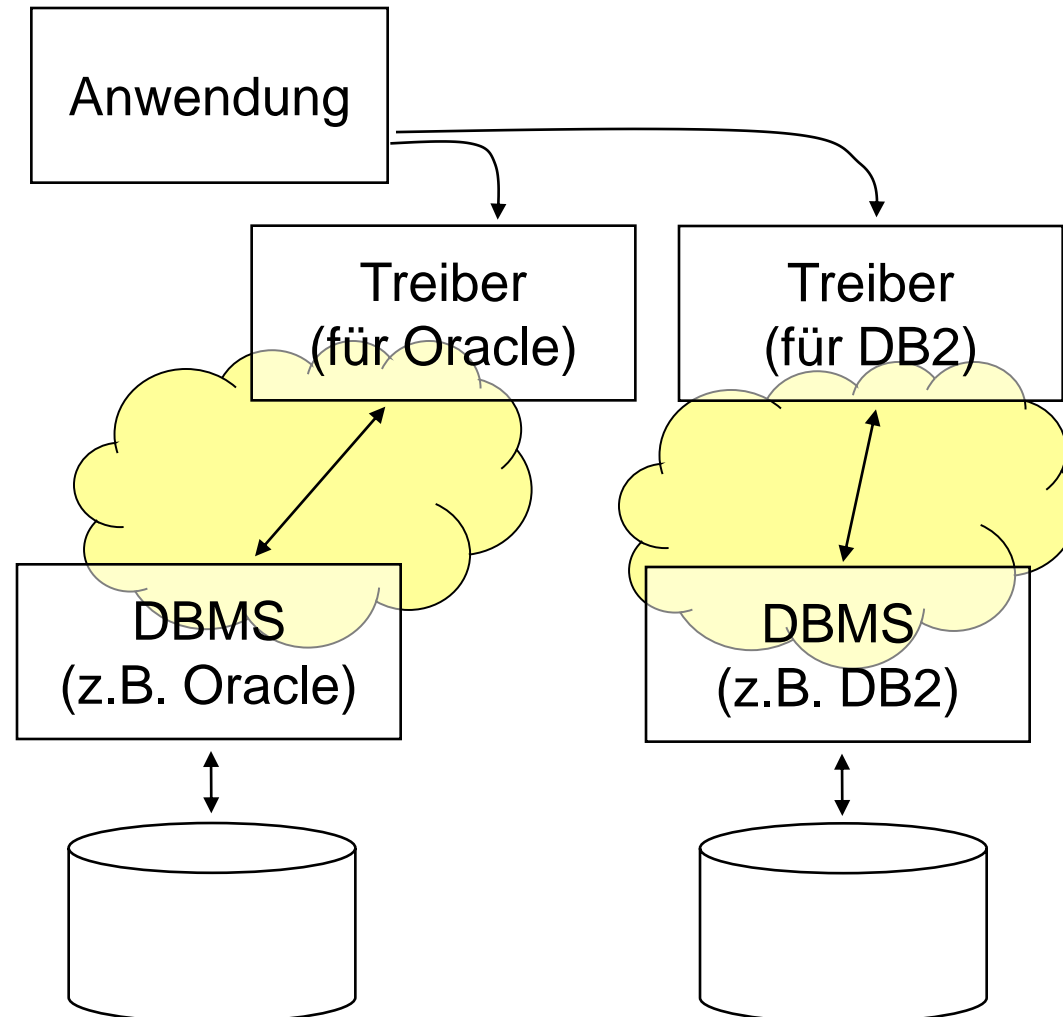
- Bisher:
Datenhaltung in
Datenbanken
 - Unterstützung
direkter Anfragen
mittels SQL
 - Ausschließlich
deklarative Anfragen
 - Ausdrucksmäch-
tigkeit ist eingeschränkt
(umständlich)
- Jetzt: Anbindung
von Programmier-
sprachen an DB
 - Benutzer verlangen
GUI, z.B. Web-
Frontend
 - Anwendungslogik
arbeitet mit Daten
 - Programmier-
sprachen sind
berechnungs-
universell



Agenda

- **Call Level Interface (CLI):
Java und JDBC**
- **Objekt-Relationales Mapping (ORM):
Java und Hibernate**
- **Prozedurale Datenbanksprachen:
Oracle PL/SQL**

Call Level Interface (CLI): Architektur



JDBC
Hibernate
PL/SQL



JDBC: Einleitung

- Datenbankzugriffsschnittstelle für Java,
- unabhängig vom DBMS,
- vergleichbar mit ODBC (Microsoft's CLI),
- Low-Level-API:
Direkte Nutzung von SQL
- Unterstützung von Transaktionen

JDBC
Hibernate
PL/SQL

JDBC: Überblick

- Java-Package `java.sql` – wichtige Klassen dieses Packages:
 - **DriverManager**: Einstiegspunkt, Laden von Treibern und Grundlage für Aufbau der Datenbankverbindung,
 - **Connection**: Datenbankverbindung,
 - **Statement**: Ausführung von Anweisungen über eine Verbindung,
 - **ResultSet**: verwaltet Ergebnisse einer Anfrage in Form einer Relation, Zugriff auf einzelne Spalten.
- Außerdem: **SQLException**

JDBC
Hibernate
PL/SQL

JDBC: Verbindungsaufbau I

- Treiber laden, z.B.:
 - `Class.forName ("oracle.jdbc.driver.OracleDriver") ;`
 - Initialisiert den Treiber zum Zugriff auf die Datenbank.
 - Treiber werden in Form von Java-Klassen von Datenbank-Herstellern zur Verfügung gestellt.

JDBC
Hibernate
PL/SQL

JDBC: Verbindungsaufbau II

- Verbindung herstellen:

```
Connection con;  
con = DriverManager.getConnection  
("jdbc:oracle:thin:@benriach:1521:tox",  
 "user", "passwd");
```

Name
Rechner

Name der
Datenbank

- Erster Parameter spezifiziert
 - zu verwendender Treiber
 - Server-Host und Port
 - Datenbank
- Weitere Parameter
 - Benutzer
 - Passwort

JDBC
Hibernate
PL/SQL

JDBC: Anfrageausführung

- Anweisungsobjekt (Statement) erzeugen:

```
Statement stmt = con.createStatement();
```

- Anweisung ausführen:

```
String table = "buecher";  
ResultSet rs = stmt.executeQuery(  
    "SELECT titel, preis FROM " + table  
);
```

- Änderungsanweisung ausführen:

```
executeUpdate(string)
```

JDBC
Hibernate
PL/SQL

JDBC: Ergebnisverarbeitung

- Navigation über Ergebnismenge (Cursor-Prinzip):

```
while (rs.next ()) {  
    // Verarbeitung der einzelnen Tupel  
    ...  
}
```

- Zugriff auf Spaltenwerte des aktuellen Tupels über **getXYZ**-Methoden

- über Spaltenindex:

```
String titel = rs.getString(1);
```

- über Spaltenname:

```
int anzahl = rs.getInt(„anzahl“);
```

JDBC
Hibernate
PL/SQL



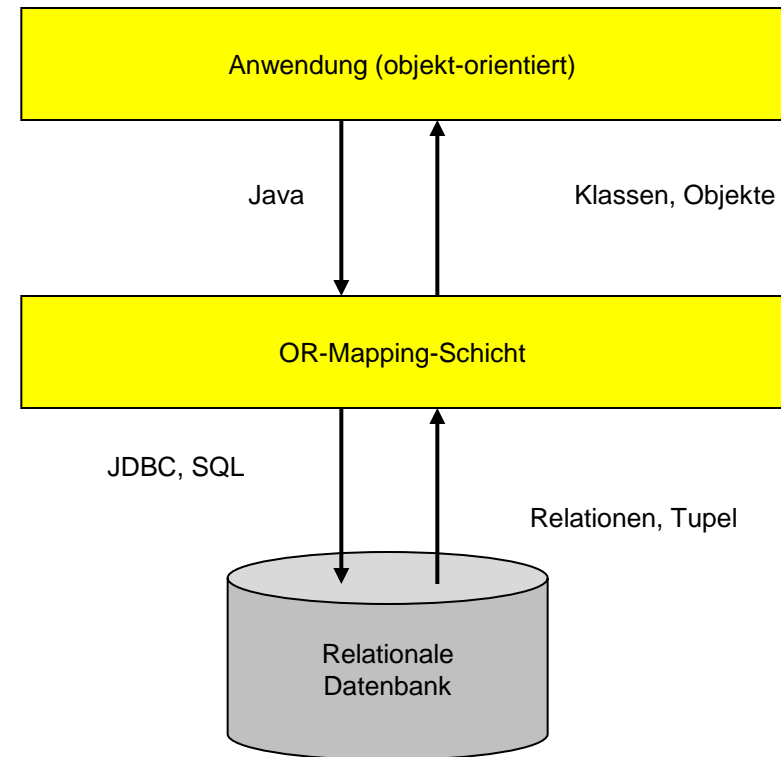
Agenda

- Call Level Interface (CLI):
Java und JDBC
- **Objekt-Relationales Mapping (ORM):
Java und Hibernate**
- Prozedurale Datenbanksprachen:
Oracle PL/SQL

Objekt-Relationales-Mapping (ORM)

- Speichern von Objekten
- Keine direkte Verwendung von SQL nötig (in aller Regel aber möglich)
- Vermittlung zwischen objektorientierter Anwendung und relationalem DBMS
- Abbildung objektorientierte auf ‚flache‘ relationale Strukturen (und umgekehrt)

JDBC
Hibernate
PL/SQL



Warum ORM?

- ORM: Persistenzverwaltung für (Daten-)Objekte
 - Weniger Anwendungs-Code
 - Bessere Wartbarkeit, einfaches Testen
- Hohe Performanz: Caching, Lazy Loading etc.
- Nutzung vorhandener relationaler DBMS
- Objekte profitieren von relationalem DBMS:
 - Indizierung
 - Konsistenzsicherung
 - hohe Verfügbarkeit
 - Mehrbenutzerbetrieb
- Erleichtert Integration vorhandener Datenbestände

JDBC
Hibernate
PL/SQL

Mapping: Idee

- Abbildung von ... auf ...
 - Klassen
 - Eine oder mehrere Tabellen
 - Mehrere Klassen auf eine Tabelle
 - Primitiven Attributen (Members)
 - Spalten von Tabellen mit Datentypen des DBMS
 - Mengenwertigen Attributen
 - Neue Tabelle (1:n-Beziehung)
 - Attribute anderen Typs
 - Als Fremdschlüssel auf andere Tabelle
 - Beziehungen
 - Fremdschlüsselbeziehungen

JDBC
Hibernate
PL/SQL

Beispiel ORM: Hibernate

- ORM für Java
- De facto Standard
- Open Source
- Unterstützt alle gängigen Datenbanksysteme
- Unterstützte Vorgehensweisen
 - Top-Down:
Generierung von DB-Schema aus Klassenmodell
 - Bottom-Up:
Generierung von Klassen aus DB-Schema
 - Meet-In-The-Middle:
Mapping zwischen vorhandenem Klassenmodell und vorhandenem DB-Schema

JDBC
Hibernate
PL/SQL

Funktionsweise von Hibernate

- Mapping wird (automatisch) spezifiziert
 - Mapping wird in XML-Datei hinterlegt
 - Alternativ: Annotation im Java-Code
- (Komplexe) Java-Objekte können persistent gemacht werden
 - Hibernate bildet Objekte gemäß Mapping-Regeln auf eine oder viele Relationen ab
- Objekte können wieder aus Datenbank geladen werden

JDBC
Hibernate
PL/SQL

Beispiel: Objekt-Persistenz

- Java Daten-Klassen können ganz normal verwendet werden
 - Beispiel: Klasse `Message`
 - Keine Hibernate-spezifischen Erweiterungen in `Message` (im Gegensatz zu anderen ORMs)
- **Instanzen der Klasse können als normale Java Objekte behandelt werden**
- Für Persistenzverwaltung zusätzliche Funktionen benötigt

```
public class Message {  
    private String text;  
    public Message(String text) {  
        this.text = text;  
    }  
    public String getText() {  
        return text;  
    }  
}
```

```
Message message =  
    new Message("Hello World");  
System.out.println(  
    message.getText());
```

JDBC
Hibernate
PL/SQL

Speichern eines Objektes

- Java-Ebene
 - Einfaches Speichern
 - Keinerlei SQL
 - Verwendung von Transaktionen möglich

```
Session session =  
    getCurrentSession();  
  
Message message =  
    new Message("Hello World!");  
  
session.save(message);
```

- SQL-Ebene
 - SQL-Statement automatisch erzeugt
 - Primärschlüssel MESSAGE_ID automatisch generiert

```
INSERT INTO MESSAGES  
    (MESSAGE_ID, MESSAGE_TEXT)  
VALUES (1, 'Hello World')
```

JDBC
Hibernate
PL/SQL

Laden eines Objektes

- Java-Ebene
 - Anfrage ist komplett objektorientiert.
 - Auch andere Sprachen:
 - SQL
 - HQL (Verwendung von Klassennamen anstatt relationaler Ausdrücke)
 - Ergebnis ist Java-Liste
- Innerhalb Hibernate
 - Automatische Abbildung der Ergebnismenge auf Objektliste
- SQL-Ebene
 - Automatische Erzeugung von SQL-Statements

```
boolean german = false;
Session s = getCurrentSession();
Criteria c =
    s.createCriteria(Message.class);

if (german)
c.add(Restrictions.like(
    "text", "%Welt%"));
else c.add(Restrictions.like(
    "text", "%World%"));
// "text" ist Member der Klasse

List<Message> messages = c.list();
for (Message message:messages)
    System.out.println(
        message.getText());
```

```
SELECT * FROM MESSAGES
```

JDBC
Hibernate
PL/SQL



Agenda

- Call Level Interface (CLI):
Java und JDBC
- Objekt-Relationales Mapping (ORM):
Java und Hibernate
- **Prozedurale Datenbanksprachen:**
Oracle PL/SQL

Prozedurale Datenbanksprachen

- Manchmal ist es sinnvoll, ausgewählte Teile der Anwendungslogik im DBMS auszuführen:
 - Transfers von Daten zwischen DBMS und Anwendungsserver entfallen
 - Verwendung nativer Datentypen des DBMS ist effizient, außerdem ggf. Optimierung möglich
 - Anwendungslogik kann zentral vorgehalten und gewartet werden -> Konsistenzsicherung
 - Ausführung gesteuert durch Trigger
- SQL allein reicht aber nicht:
 - Ist als deklarative Anfragesprache bewusst einfach und i.A. nicht berechnungsuniversell
- Erweiterung um prozedurale Konstrukte

JDBC
Hibernate
PL/SQL



PL/SQL – Struktur

- Mechanismus für gespeicherte Prozeduren in Oracle.
- Block – Strukturierungseinheit in PL/SQL; können hintereinander geschachtelt sein.
- Aufbau eines Blocks:

```
DECLARE
```

```
...
```

```
BEGIN
```

```
...
```

```
EXCEPTION
```

```
...
```

```
END;
```

```
.
```

```
run
```

JDBC
Hibernate
PL/SQL

Variablen und Typen I

JDBC
Hibernate
PL/SQL

- Informationsaustausch zwischen PL/SQL-Programm und Datenbank durch Variablen, z.B.

```
DECLARE
    preis NUMBER;
    meinBier VARCHAR(20);
```

- Elegante Möglichkeit, um sicherzustellen, daß Attributtyp in Datenbank und Variablen-Typ in PL/SQL-Programm identisch – Illustration:

```
DECLARE
    meinBier Biere.name%Type;
```

- Variable mit Record-Typ, bestehend aus mehreren getypten Feldern – Beispiel:

```
DECLARE
    bierTupel Biere%ROWTYPE;
```



```
Oracle SQL*Plus
Datei Bearbeiten Suchen Optionen Hilfe
SQL> create table t1(e integer, f integer);
Tabelle wurde angelegt.
SQL> delete from t1;
0 Zeilen wurden gelöscht.
SQL> insert into t1 values(1,3);
1 Zeile wurde erstellt.
SQL> insert into t1 values(2,4);
1 Zeile wurde erstellt.
SQL> declare
2 a NUMBER;
3 b NUMBER;
4 begin
5 select e,f into a,b from t1 where e>1;
6 insert into t1 values(b,a);
7 end;
8 .
SQL> run;
1 declare
2 a NUMBER;
3 b NUMBER;
4 begin
5 select e,f into a,b from t1 where e>1;
6 insert into t1 values(b,a);
7* end;
PL/SQL-Prozedur wurde erfolgreich abgeschlossen.
SQL> select * from t1;
-----
          E          F
-----
          1          3
          2          4
          4          2
SQL>
```

JDBC
Hibernate
PL/SQL

Kontrollfluß I

- Beispiele:

- DECLARE

BEGIN

```
a NUMBER;  
b NUMBER;
```

```
SELECT e, f INTO a, b  
FROM T1 WHERE e>1;
```

```
IF b=1 THEN
```

```
    INSERT INTO T1  
        VALUES (b, a);
```

```
ELSE
```

```
    INSERT INTO T1  
        VALUES (b+10, a+10);
```

```
END IF;
```

END;

•
run;

JDBC
Hibernate
PL/SQL



Kontrollfluß II

- Beispiele (Fortsetzung):

- **DECLARE**

```
    i NUMBER := 1;
```

```
BEGIN
```

```
    LOOP
```

```
        INSERT INTO T1
```

```
            VALUES (i, i);
```

```
        i := i+1;
```

```
        EXIT WHEN i>100;
```

```
    END LOOP;
```

```
END;
```

```
•
```

```
run;
```

JDBC
Hibernate
PL/SQL

Cursors

- Tupel an der aktuellen Cursor-Position kann i. a. auch modifiziert und gelöscht werden.

```
DECLARE
  a T1.e%TYPE;  b T1.f%TYPE;
  CURSOR T1Cursor IS
  SELECT e, f FROM T1 WHERE e < f FOR UPDATE;
BEGIN
  OPEN T1Cursor;
  LOOP
    FETCH T1Cursor INTO a, b;
    EXIT WHEN T1Cursor%NOTFOUND;
    DELETE FROM T1 WHERE CURRENT OF
      T1Cursor;
    /* Insert the reverse tuple: */
    INSERT INTO T1 VALUES(b, a);
  END LOOP; /* Free cursor used by the
    query. */
  CLOSE T1Cursor;
END;
```

•
run;

JDBC
Hibernate
PL/SQL



Weitere prozedurale SQL-Erweiterungen

- Standardisiert: SQL/PSM
- IBM DB2: SQL/PSM, PL/SQL (seit 2009)
- Microsoft SQL Server: Transact-SQL
- Externe Routinen:
Implementiert z. B. in C, Java.
- Java als Sprache für Datenbank-Prozeduren

JDBC
Hibernate
PL/SQL



Veranstaltungen nächstes Semester

Praktika WS/SS	Datenbank- praktikum	Praktikum Verteilte Datenhaltung	Praktikum Data Warehousing und Mining
Vertiefungs- vorl. SS	Datenschutz und Privatheit in vernetzten Informationssystemen	Datenbankimplementierung und -Tuning	Die digitale Bibliothek
Vertiefungs- vorl. WS	Moving Objects Databases	Workflow-Management Systeme	Informationsintegration und Web-Portale
	Datenbankeinsatz	Verteilte Datenhaltung	Data Warehousing und Mining
Kernvorl. SS	Kommunikation und Datenhaltung		