Database Programming

Prof. Dr. Ralf Lämmel
Universität Koblenz-Landau
Software Languages Team
Elevator speech

Think of information systems and data processing!

1. How to persist data?
2. How to separate data and functionality?
3. How to deal with a lot of data efficiently?
4. How to implement entity relationships?

XML and JSON may serve 1.-2.
Relational databases serve 1.-4.

Exercise: what’s XML specifically good for?

Also: how to remain an OO programmer?
Database programming (ignoring OO specifics)

1. Model data via **entity-relationship** (ER) model
2. Map ER model to relational model (**tables**)
3. Implement relational model via **SQL**
4. Implement **CRUD** functionality

(C) 2010-2014 Prof. Dr. Ralf Lämmel, Universität Koblenz-Landau (where applicable)
The Entity/Relationship model

Just a quick ride; see your DB course for details.
Entities

- company: name
- department: name
- employee: name, address, salary, manager

There are only attributes of "simple" types.

Just a quick ride; see your DB course for details.
Relationships

- The company of a department.
- The super-department of a sub-department.
- The company of an employee.
- The department of an employee.

Exercise: figure out cardinalities for the listed relationships.

Just a quick ride; see your DB course for details.
The relational model
Relations (tables)

- Relation
  - Vertically: set of tuples ("rows")
  - Horizontally: set of columns
- Each cell is of some type
  - Strings
  - Numbers
  - Row IDs (numbers again)
Relational schemas

Key terms

- Attributes (names)
- Attribute domains (types)
- Relational schema (attribute-domain pairs)
- Instance of relational schema (sets of tuples)
The relational schema for 101 companies

- Relational schemas (names only)
  - company (id, name)
  - department (id, name, cid, did)
  - employee (id, name, address, salary, manager, cid, did)

**Key constraints:**
Primary key (underlined) for identification of tuple
Foreign key (italics) for reference to another table
Variation

- Relational schemas (names only)
  - company (id, name)
  - department (id, name, cid, did)
  - employee (id, name, address, salary, manager, cid, did)

Key constraints:
Primary key (underlined) for identification of tuple
Foreign key (italics) for reference to another table
Variation

- Relational schemas (names only)
  - company (id, name)
  - department (id, name, cid, did, mid)
  - employee (id, name, address, salary, manager, cid, did)

Key constraints:
- Primary key (underlined) for identification of tuple
- Foreign key (italics) for reference to another table
Relational algebra: compute relations

- **Projection** (narrow down on certain columns)
- **Selection** (narrow down on certain rows)
- **Join** (compose two tables by condition)
Map ER to relations

- Every entity becomes a table.
- Relationships
  - 1:1 use foreign key
  - otherwise (mostly) use extra table.
- Compare with implementation of UML class diagrams.

We also speak of tables instead of relations.

Just a quick ride; see your DB course for details.
SQL DDL
CREATE TABLE company (  
id INTEGER,  
name VARCHAR(100)  
)
CREATE TABLE company (  
id INTEGER PRIMARY KEY,  
name VARCHAR(100) UNIQUE NOT NULL  
)
CREATE TABLE company ( 
  id INTEGER AUTO_INCREMENT PRIMARY KEY, 
  name VARCHAR(100) UNIQUE NOT NULL 
)
CREATE TABLE department (  
id INTEGER,  
name VARCHAR(100),  
cid INTEGER,  
did INTEGER,  
)
CREATE TABLE department (  
id INTEGER PRIMARY KEY,  
name VARCHAR(100) NOT NULL,  
cid INTEGER NOT NULL,  
did INTEGER,  
FOREIGN KEY (cid) REFERENCES company(id),  
FOREIGN KEY (did) REFERENCES department(id)  )
CREATE TABLE department (  
id INTEGER PRIMARY KEY,  
name VARCHAR(100) UNIQUE NOT NULL,  
cid INTEGER NOT NULL,  
did INTEGER,  
FOREIGN KEY (cid) REFERENCES company(id)  
    ON DELETE CASCADE ON UPDATE CASCADE,  
FOREIGN KEY (did) REFERENCES department(id)  
    ON DELETE CASCADE ON UPDATE CASCADE)

(C) 2010-2014 Prof. Dr. Ralf Lämmel, Universität Koblenz-Landau (where applicable)
CREATE TABLE employee (  
id INTEGER PRIMARY KEY,
name VARCHAR(50) NOT NULL,
address VARCHAR(50) NOT NULL,
salary DOUBLE NOT NULL,
manager BOOL NOT NULL,
cid INTEGER NOT NULL,
did INTEGER NOT NULL,
FOREIGN KEY (cid) REFERENCES company(id),
FOREIGN KEY (did) REFERENCES department(id)
)
DDL language summary

- CREATE TABLE
  - INTEGER, VARCHAR(...), DOUBLE
  - NOT NULL
  - UNIQUE
  - PRIMARY / FOREIGN KEY
  - ON DELETE / UPDATE CASCADE
Database programming with SQL
(Structured Query Language)

- Represent schema in DDL subset of SQL
  - DDL - Data Definition Language
    - Part of SQL for data definition
- Represent population in DML subset of SQL
  - DML - Data Manipulation Language
    - Part of SQL for CRUD (Create, Read, Update, Delete)
We use a local database server and SQL monitor; see the online documentation for the contribution.
SQL DML
INSERT INTO company (name) VALUES ("Acme Corporation")

Insert a new company into the corresponding table.
INSERT INTO department (name,cid) VALUES ("Research",1)
INSERT INTO department (name,cid) VALUES ("Development",1)
...

Insert several departments into the corresponding table.
SELECT * FROM DEPARTMENT

<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
<th>cid</th>
<th>did</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Research</td>
<td>1</td>
<td>NULL</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Development</td>
<td>1</td>
<td>NULL</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Dev1</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Dev1.1</td>
<td>1</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>

List of tuples of the department table.
SELECT SUM(salary) FROM employee

Select all employees, project to their salaries, and sum them up.
SELECT SUM(salary) FROM employee
WHERE cid = 1

Retrieve only salaries of a specific company.
SELECT SUM(salary) FROM employee
WHERE cid =
(SELECT id FROM company
WHERE name = "Acme Corporation")

Use a nested query to determine the company id.
UPDATE employee
SET salary = salary / 2

Cut all salaries in half.
UPDATE employee
SET salary = salary / 2
WHERE cid = 1

Limit update to employees with company id = 1.
UPDATE employee
SET salary = salary / 2
WHERE cid =
(SELECT id FROM company
WHERE name = "Acme Corporation")

Use a nested query to determine the company id.
Embedding SQL with JDBC
Total all salaries

```java
public static double total(MyConnection myConnection, String companyName){
    double total = 0;
    try {
        String query = "SELECT salary FROM employee "
                        + "WHERE cid = (SELECT id FROM company WHERE name = ?);";
        PreparedStatement pstmtEmployees = myConnection.getConn()
                                        .prepareStatement(query);
        pstmtEmployees.setString(1, companyName);
        ResultSet salaries = pstmtEmployees.executeQuery();
        while (salaries.next())
            total += salaries.getDouble("salary");
    } catch (SQLException e){
        e.printStackTrace();
    }
    return total;
}
```

We do not use SQL’s SUM here so that we can demonstrate JDBC’s ResultSets with the example.

(C) 2010-2014 Prof. Dr. Ralf Lämmel, Universität Koblenz-Landau (where applicable)
Library support for database programming

- JDBC (part of Java Core API)
  - Connections to databases
  - Submit SQL statements
  - Retrieve results

- MySQL connector (part of demo project)
  - JDBC-based driver for MySQL
Embedded SQL

- Important JDBC types
  - Connection
  - Statement (different forms thereof)
  - ResultSet (for queries in particular)
  - SQL Exception
Cut all salaries in half

```java
public static void cut(MyConnection myConnection, String companyName) {
    try {
        // cut salaries in all employee columns
        String sqlCut = "UPDATE employee SET salary = salary / 2 "
                        + "WHERE cid = (SELECT id FROM company WHERE name = ?)";
        PreparedStatement pstmtEmployees = myConnection.getConn()
                                    .prepareStatement(sqlCut);
        pstmtEmployees.setString(1, companyName);
        pstmtEmployees.executeUpdate();
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
```

**Exercise**: understand the notion of injection attacks and argue how “prepared statements” help avoiding the problem.
http://101companies.org/wiki/Contribution:simplejdbc

DEMO
Object/Relational Mapping

We travel between the O and R spaces here. We traveled between O and X spaces before.
Context: X/O/R mapping
The Bermuda Triangle of data processing

In fact, there are further technical spaces.
public class Company {
    private String name;
    private List<Department> depts = new LinkedList<Department>();
    public String getName() { return name; }
    public void setName(String name) { this.name = name; }
    public List<Department> getDepts() { return depts; }
}

public class Department { ... }

public class Employee { ... }

(C) 2010-2014 Prof. Dr. Ralf Lämmel, Universität Koblenz-Landau (where applicable)
XSD for 101companies System

<xs:element name="company">
   <xs:complexType>
      <xs:sequence>
         <xs:element ref="name"/>
         <xs:element maxOccurs="unbounded"
                      minOccurs="0"
                      ref="department"/>
      </xs:sequence>
   </xs:complexType>
</xs:element>

<xs:element name="department"> ... </xs:element>

<xs:complexType name="employee"> ... </xs:complexType>
CREATE TABLE company ( 
    id INTEGER PRIMARY KEY,
    name VARCHAR(100) UNIQUE NOT NULL
)

CREATE TABLE department ( ... )
CREATE TABLE employee ( ... )

Observe one detail: companies do not refer to departments (but vice versa).
Elevator pitch

How to persist objects in database tables?

How to map object models to relational schemas?

... or the other way around?
Bing for this statement and read about it:

“O/R Mapping ... is the Vietnam of Computer Science”!

Read about the “O/R impedance mismatch”!
Hibernate
Hibernate:
Simplifying persistence in Java

Capabilities

- Store objects in database tables, one object per row.
- Restore objects in different programs / runs.
The Hibernate architecture

Source: Hibernate Reference Documentation
A persistent class

```java
public class Cat {
    private String id;
    private String name;
    private char sex;
    private float weight;
    public String getId() { return id; }
    private void setId(String id) { this.id = id; }
    // ... other getters and setters ...
}
```

Used for the primary key
Metadata for O/R mapping

```xml
<hibernate-mapping>
  <class name="Cat" table="CAT">  
    <id name="id" type="string" unsaved-value="null" >
      <column name="CAT_ID" sql-type="char(32)" not-null="true"/>
      <generator class="uuid.hex"/>
    </id>
    <property name="name">  
      <column name="NAME" length="16" not-null="true"/>
    </property>
    <property name="sex"/>
    <property name="weight"/>
  </class>
</hibernate-mapping>
```

Map Java String type to SQL type with length
A database table

The CAT table in the database

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Modifiers</th>
</tr>
</thead>
<tbody>
<tr>
<td>cat_id</td>
<td>character(32)</td>
<td>not null</td>
</tr>
<tr>
<td>name</td>
<td>character varying(16)</td>
<td>not null</td>
</tr>
<tr>
<td>sex</td>
<td>character(1)</td>
<td></td>
</tr>
<tr>
<td>weight</td>
<td>real</td>
<td></td>
</tr>
</tbody>
</table>

Indexes: cat_pkey primary key btree (cat_id)
A Hibernate session
(in Java code)

Session session = HibernateUtil.currentSession();
Transaction tx = session.beginTransaction();

Cat princess = new Cat();
princess.setName("Princess");
princess.setSex('F');
princess.setWeight(7.4f);

session.save(princess);
tx.commit();
HibernateUtil.closeSession();
How to retrieve persistent objects?
Use HQL (Hibernate query language).

```java
Query query = session.createQuery("select c from Cat as c where c.sex = :sex");
query.setCharacter("sex", 'F');
for (Iterator it = query.iterate(); it.hasNext();) {
    Cat cat = (Cat) it.next();
    out.println("Female Cat: " + cat.getName() );
}
```
So what’s O/R mapping?

Wikipedia’s definition

Ralf’s definition (relatively naïve version):
• Category 1:
  – Start from (idiomatic) classes.
  – Map object model to relational schema.
  – Deploy relational schema in database.
  – Encode CRUD in OO code.
  – Add transactions to OO code.
• Category 2:
  – Start from database (schema, instance, SPROC).
  – Derive object model to encapsulate data access.
  – Continue as above ...
• Category 1 + 2: classes and tables given, mapping wanted.
• Category 2’:
  – Like Category 2 but ...
  – ER/relational model-level mapping.
  – Coverage of distributed database and data integration.
http://10lcompanies.org/wiki/Contribution:hibernate
Data dir to be used by HSQLDB

Hibernate configuration

See Makefile for usage scenario

Hibernate-enabled object model with mapping files

The usual features.

SQL scripts with relational schema and instance data.

(C) 2010-2014 Prof. Dr. Ralf Lämmel, Universität Koblenz-Landau (where applicable)
Employee POJOs

```java
public class Employee {

    private long id;
    private String name;
    private String address;
    private double salary;
    private boolean manager;

    public long getId() {
        return id;
    }

    @SuppressWarnings("unused")
    private void setId(long id) {
        this.id = id;
    }

    ...

}
```

(C) 2010-2014 Prof. Dr. Ralf Lämmel, Universität Koblenz-Landau (where applicable)
Mapping for employees

<hibernate-mapping>

<class name="org.softlang.company.Employee" table="EMPLOYEE">

  <id name="id" column="ID">
    <generator class="native" />
  </id>

  <property name="name" />
  <property name="address" />
  <property name="salary" />
  <property name="manager" />

</class>

</hibernate-mapping>
Mapping for departments

<hibernate-mapping>

<class name="org.softlang.company.Department" table="DEPARTMENT">

  <id name="id" column="ID">
    <generator class="native" />
  </id>

  <property name="name" />

  <set name="employees" cascade="all">
    <key column="DEPT_ID" />
    <one-to-many class="org.softlang.company.Employee" />
  </set>

  <set name="subdepts" cascade="all">
    <key column="DEPT_ID" />
    <one-to-many class="org.softlang.company.Department" />
  </set>

</class>

</hibernate-mapping>
Hibernate configuration

<hibernate-configuration>
  <session-factory>
    <!-- Database connection settings. -->
    <property name="connection.driver_class">org.hsqldb.jdbcDriver</property>
    <property name="connection.url">jdbc:hsqldb:hsql://localhost</property>
    <property name="connection.username">sa</property>
    <property name="connection.password"></property>

    <!-- SQL dialect -->
    <property name="dialect">org.hibernate.dialect.HSQLDialect</property>

    <!-- Create the database schema, if needed; update otherwise -->
    <property name="hbm2ddl.auto">update</property>

    <!-- Mapping files in the project -->
    <mapping resource="org/softlang/company/Company.hbm.xml" />
    <mapping resource="org/softlang/company/Department.hbm.xml" />
    <mapping resource="org/softlang/company/Employee.hbm.xml" />

    ...
  </session-factory>
</hibernate-configuration>
http://101companies.org/wiki/Contribution:hibernate

DEMO
After starting DB and GUI

Non-default selection
After JUnit test Load and Refresh
After executing PopulateTables.sql

insert into COMPANY (NAME) values ('meganalysis'); -- ID = 1

insert into DEPARTMENT (NAME, COMP_ID) values ('Research',1); -- ID = 1
insert into DEPARTMENT (NAME, COMP_ID) values ('Development',1); -- ID = 2

insert into DEPARTMENT (NAME, DEPT_ID) values ('Dev1',2); -- ID = 3
insert into DEPARTMENT (NAME, DEPT_ID) values ('Dev1.1',3); -- ID = 4

insert into EMPLOYEE (NAME, DEPT_ID, ADDRESS, SALARY, MANAGER) values ('Craig', 1, 'Redmond', 123456, true);
insert into EMPLOYEE (NAME, DEPT_ID, ADDRESS, SALARY, MANAGER) values ('Erik', 1, 'Utrecht', 12345, false);
insert into EMPLOYEE (NAME, DEPT_ID, ADDRESS, SALARY, MANAGER) values ('Ralf', 1, 'Koblenz', 1234, false);
insert into EMPLOYEE (NAME, DEPT_ID, ADDRESS, SALARY, MANAGER) values ('Ray', 2, 'Redmond', 234567, true);
insert into EMPLOYEE (NAME, DEPT_ID, ADDRESS, SALARY, MANAGER) values ('Klaus', 3, 'Boston', 23456, true);
insert into EMPLOYEE (NAME, DEPT_ID, ADDRESS, SALARY, MANAGER) values ('Karl', 4, 'Riga', 2345, true);
insert into EMPLOYEE (NAME, DEPT_ID, ADDRESS, SALARY, MANAGER) values ('Joe', 4, 'Wifi City', 2344, false);

update count
1

(C) 2010-2014 Prof. Dr. Ralf Lämmel, Universität Koblenz-Landau (where applicable)
Running a simple SQL query

```
select * from employee;
```

<table>
<thead>
<tr>
<th>ID</th>
<th>NAME</th>
<th>ADDRESS</th>
<th>SALARY</th>
<th>MANAGER</th>
<th>DEPT_ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Craig</td>
<td>Redmond</td>
<td>123,456</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>Erik</td>
<td>Utrecht</td>
<td>12,345</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>Ralf</td>
<td>Koblenz</td>
<td>1,234</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>Ray</td>
<td>Redmond</td>
<td>234,567</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>Klaus</td>
<td>Boston</td>
<td>23,456</td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>Karl</td>
<td>Riga</td>
<td>2,345</td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>7</td>
<td>Joe</td>
<td>Wifi City</td>
<td>2,344</td>
<td></td>
<td>4</td>
</tr>
</tbody>
</table>

(C) 2010-2014 Prof. Dr. Ralf Lämmel, Universität Koblenz-Landau (where applicable)
Summary

- Use OOP as the programming paradigm.
- Use relational DBs as the data management paradigm.
- OO programs can use embedded SQL for database access.
  - SQL may be represented as strings.
  - Query results may be as weakly typed collections.
- Object models may be mapped to relational schemas.
  - Relational data may be loaded into objects.
  - Object changes may be saved back to database.
  - Beware of the O/R impedance mismatch.
Left-over material
Hibernate topics not covered

- Use annotations instead of mapping files
- R-to-O
  - Database schema already present
  - No need to manually write mapping
  - No need to manage constraints
  - Use Hibernate’s generator tools
- Object queries
  - Use Hibernate’s language for object queries
    - EJB-QL
Developer’s view on using Hibernate

The content on this slide is covered “in passing” in the lecture.
Developer’s view on using Hibernate

1. Link Hibernate libraries
2. Configure database
3. Hibernate-enable classes
4. Define a mapping
5. Write CRUD code

The content on this slide is covered “in passing” in the lecture.
Developer’s view on using Hibernate

1. Link Hibernate libraries
2. Configure database
3. Hibernate-enable classes
4. Define a mapping
5. Write CRUD code

The content on this slide is covered “in passing” in the lecture.
Download Hibernate

http://www.hibernate.org/

The content on this slide is covered “in passing” in the lecture.
Required for Hibernate 3.2 (+ HSQLDB support)

- antlr-2.7.6.jar
- asm.jar
- asm-attrs.jar
- cglib-2.1.3.jar
- commons-collections-2.1.1.jar
- commons-logging-1.0.4.jar
- dom4j-1.6.1.jar
- ehcache-1.2.3.jar
- jta.jar
- log4j-1.2.11.jar

- hibernate3.jar
- hsqldb.jar

Tip: create a lib dir within your project and place all those jars over there. Then, make sure your project’s build path references the jars.
Developer’s view on using Hibernate

1. Link Hibernate libraries
2. Configure database
3. Hibernate-enable classes
4. Define a mapping
5. Write CRUD code

The content on this slide is covered “in passing” in the lecture.
Hibernate relies on a RDBMS

- Simple option
  - Use hsqldb (HyperSQL DB engine)
    - hsqldb.jar
- More flexible option
  - Use any SQL database via JDBC
Using hsqldb

- Create a “data” subdirectory in your project directory.
- Start the DB engine from within the “data” directory:
  
  `java -cp ../lib/hsqldb.jar org.hsqldb.Server`

- Keep it running in the background.
- Use database manager to monitor database.

  `java -cp lib/hsqldb.jar org.hsqldb.util.DatabaseManagerSwing`
Developer’s view on using Hibernate

1. Link Hibernate libraries
2. Configure database
3. Hibernate-enable classes
4. Define a mapping
5. Write CRUD code

The content on this slide is covered “in passing” in the lecture.
Hibernate-enable classes

- Use POJOs
  - Provide a default constructor
  - Model the following field
    private Long id;
    + public getter
    + private setter

The content on this slide is covered “in passing” in the lecture.
Developer's view on using Hibernate

1. Link Hibernate libraries
2. Configure database
3. Hibernate-enable classes
4. Define a mapping
5. Write CRUD code

The content on this slide is covered "in passing" in the lecture.
O/R mapping with Hibernate

- Defined in an XML file
  - MyClass.java
  - MyClass.hbm.xml

The content on this slide is covered “in passing” in the lecture.
Developer’s view on using Hibernate

1. Link Hibernate libraries
2. Configure database
3. Hibernate-enable classes
4. Define a mapping
5. Write CRUD code

The content on this slide is covered “in passing” in the lecture.