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 may serve 1.-2.

Relational databases serve 1.-4.

Exercise: what's XML specifically good for?

Also: how to remain an OO programmer?

(C) 2010-2013 Prof. Dr. Ralf Lämmel, Universität Koblenz-Landau (where applicable)

Database programming (ignoring OO specifics)

akin to classes + associations

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

Tables = rows / columns of cells

Create, Read, Update, Delete

(C) 2010-2013 Prof. Dr. Ralf Lämmel, Universität Koblenz-Landau (where applicable)

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

The Entity/ Relationship model

Entities

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

There are only attributes of "simple" types.

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

(C) 2010-2013 Prof. Dr. Ralf Lämmel, Universität Koblenz-Landau (where applicable)

Relationships

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

- 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.

(C) 2010-2013 Prof. Dr. Ralf Lämmel, Universität Koblenz-Landau (where applicable)

The relational model

Relations (tables)

	id	name	address	salary	manager	cid	did
▶	1	Craig	Redmond	30864	1	1	1
	2	Ray	Redmond	58641.75	1	1	2
	3	Klaus	Boston	5864	1	1	3
	4	Karl	Riga	586.25	1	1	4
	5	Erik	Utrecht	3086.25	0	1	1
	6	Ralf	Koblenz	308.5	0	1	1
	7	Joe	Wifi City	586	0	1	4

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

(C) 2010-2013 Prof. Dr. Ralf Lämmel, Universität Koblenz-Landau (where applicable)

Relational schemas

Key terms

- Attributes (names)
- Attribute domains (types)
- **Relational schema (attribute-domain pairs)**
- Instance of relational schema (sets of tuples)

(C) 2010-2013 Prof. Dr. Ralf Lämmel, Universität Koblenz-Landau (where applicable)

Relational algebra: compute relations

	id	name	address	salary	manager	cid	did
▶	1	Craig	Redmond	30864	1	1	1
	2	Ray	Redmond	58641.75	1	1	2
	3	Klaus	Boston	5864	1	1	3
	4	Karl	Riga	586.25	1	1	4
	5	Erik	Utrecht	3086.25	0	1	1
	6	Ralf	Koblenz	308.5	0	1	1
	7	Joe	Wifi City	586	0	1	4

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

(C) 2010-2013 Prof. Dr. Ralf Lämmel, Universität Koblenz-Landau (where applicable)

The relational schema for 101companies

- Relational schemas (names only)
  - company (<u>id</u>, name)
  - department (<u>id</u>, name, *cid*, *did*)
  - employee (<u>id</u>, name, address, salary, manager, *cid*, *did*)

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

(C) 2010-2013 Prof. Dr. Ralf Lämmel, Universität Koblenz-Landau (where applicable)

Map ER to relations

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

- 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.

(C) 2010-2013 Prof. Dr. Ralf Lämmel, Universität Koblenz-Landau (where applicable)

SQL DDL

DDL statement for company

```
CREATE TABLE company (
    id INTEGER,
    name VARCHAR(100)
)
```

(C) 2010-2013 Prof. Dr. Ralf Lämmel, Universität Koblenz-Landau (where applicable)

More details

```
CREATE TABLE company (
    id INTEGER PRIMARY KEY,
    name VARCHAR(100) UNIQUE NOT NULL
)
```

(C) 2010-2013 Prof. Dr. Ralf Lämmel, Universität Koblenz-Landau (where applicable)

More details

```
CREATE TABLE company (
    id INTEGER AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100) UNIQUE NOT NULL
)
```

(C) 2010-2013 Prof. Dr. Ralf Lämmel, Universität Koblenz-Landau (where applicable)

```
CREATE TABLE department (
    id INTEGER,
    name VARCHAR(100),
    cid INTEGER,
    did INTEGER,
)
```

(C) 2010-2013 Prof. Dr. Ralf Lämmel, Universität Koblenz-Landau (where applicable)

More details

```
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)
)
```

(C) 2010-2013 Prof. Dr. Ralf Lämmel, Universität Koblenz-Landau (where applicable)

More details

```
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) 2010-2013 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)
)
```

(C) 2010-2013 Prof. Dr. Ralf Lämmel, Universität Koblenz-Landau (where applicable)

DDL language summary

- CREATE TABLE
  - NOT NULL
  - PRIMARY / FOREIGN KEY
  - ON DELETE / UPDATE CASCADE

(C) 2010-2013 Prof. Dr. Ralf Lämmel, Universität Koblenz-Landau (where applicable)

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)

(C) 2010-2013 Prof. Dr. Ralf Lämmel, Universität Koblenz-Landau (where applicable)

http://101companies.org/wiki/

Contribution:mySqlMany

DEMO

We use a local database server and SQL monitor; see the online documentation for the contribution.

(C) 2010-2013 Prof. Dr. Ralf Lämmel, Universität Koblenz-Landau (where applicable)

SQL DML

CRUD

- C: Create (SQL: Insert)
- R: Read (SQL: Select)
- U: Update
- D: Delete

(C) 2010-2013 Prof. Dr. Ralf Lämmel, Universität Koblenz-Landau (where applicable)

```
INSERT INTO company (name)
 VALUES ("Acme Corporation")
```

Insert a new company into the corresponding table.

(C) 2010-2013 Prof. Dr. Ralf Lämmel, Universität Koblenz-Landau (where applicable)

```
INSERT INTO department (name,cid)
 VALUES ("Research",1)
INSERT INTO department (name,cid)
 VALUES ("Development",1)
...
```

Insert several departments into the corresponding table.

(C) 2010-2013 Prof. Dr. Ralf Lämmel, Universität Koblenz-Landau (where applicable)

```
SELECT * FROM DEPARTMENT
```

id,name,cid,did
1,Research,1,NULL
2,Development,1,NULL
3,Dev1,1,2
4,Dev1.1,1,3

List of tuples of the department table.

(C) 2010-2013 Prof. Dr. Ralf Lämmel, Universität Koblenz-Landau (where applicable)

SELECT SUM(salary) FROM employee

Select all employees, project to their salaries, and sum them up.

(C) 2010-2013 Prof. Dr. Ralf Lämmel, Universität Koblenz-Landau (where applicable)

```
SELECT SUM(salary) FROM employee
 WHERE cid = 1
```

Retrieve only salaries of a specific company.

(C) 2010-2013 Prof. Dr. Ralf Lämmel, Universität Koblenz-Landau (where applicable)

```
SELECT SUM(salary) FROM employee
 WHERE cid =
  (SELECT id FROM company
   WHERE name = "Acme Corporation")
```

Use a nested query to determine the company id.

(C) 2010-2013 Prof. Dr. Ralf Lämmel, Universität Koblenz-Landau (where applicable)

```
UPDATE employee
 SET salary = salary / 2
```

Cut all salaries in half.

(C) 2010-2013 Prof. Dr. Ralf Lämmel, Universität Koblenz-Landau (where applicable)

```
UPDATE employee
 SET salary = salary / 2
 WHERE cid = 1
```

Limit update to employees with company id = 1.

(C) 2010-2013 Prof. Dr. Ralf Lämmel, Universität Koblenz-Landau (where applicable)

```
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.

(C) 2010-2013 Prof. Dr. Ralf Lämmel, Universität Koblenz-Landau (where applicable)

Embedding SQL with JDBC

Total all salaries

```
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) 2010-2013 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

(C) 2010-2013 Prof. Dr. Ralf Lämmel, Universität Koblenz-Landau (where applicable)

Embedded SQL

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

(C) 2010-2013 Prof. Dr. Ralf Lämmel, Universität Koblenz-Landau (where applicable)

Cut all salaries in half

```
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.

(C) 2010-2013 Prof. Dr. Ralf Lämmel, Universität Koblenz-Landau (where applicable)

http://101companies.org/wiki/**Contribution:jdbc**

DEMO

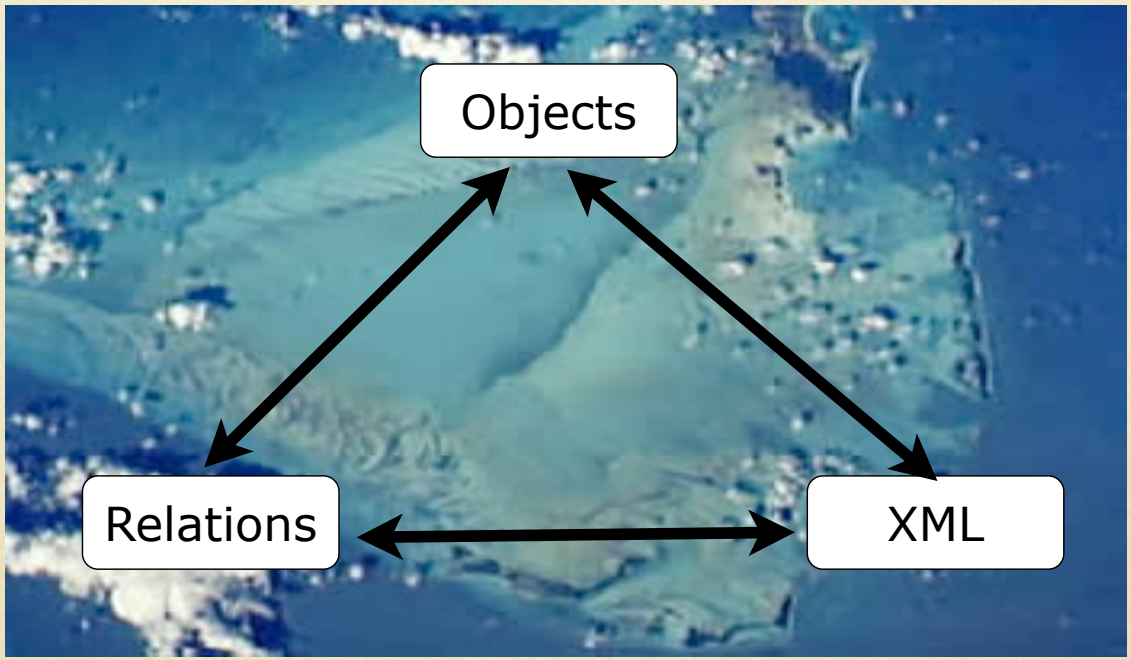
(C) 2010-2013 Prof. Dr. Ralf Lämmel, Universität Koblenz-Landau (where applicable)

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.

(C) 2010-2013 Prof. Dr. Ralf Lämmel, Universität Koblenz-Landau (where applicable)
Thanks to Erik Meijer for contributing this slide or part thereof.

Object model for 101companies System

```
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) 2010-2013 Prof. Dr. Ralf Lämmel, Universität Koblenz-Landau (where applicable)

XSD for l0l companies 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>
```

(C) 2010-2013 Prof. Dr. Ralf Lämmel, Universität Koblenz-Landau (where applicable)

Relational schema (SQL DDL)

```
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).

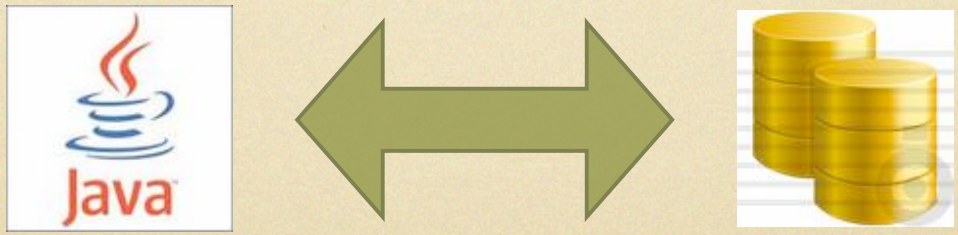
(C) 2010-2013 Prof. Dr. Ralf Lämmel, Universität Koblenz-Landau (where applicable)

Elevator pitch

How to persist objects in database tables?

How to map object models to relational schemas?

... or the other way around?



(C) 2010-2013 Prof. Dr. Ralf Lämmel, Universität Koblenz-Landau (where applicable)

Exercises

Bing for this statement and read about it:

"**O/R Mapping … is the Vietnam of Computer Science**"!

Read about the "O/R impedance mismatch"!

(C) 2010-2013 Prof. Dr. Ralf Lämmel, Universität Koblenz-Landau (where applicable)

Hibernate

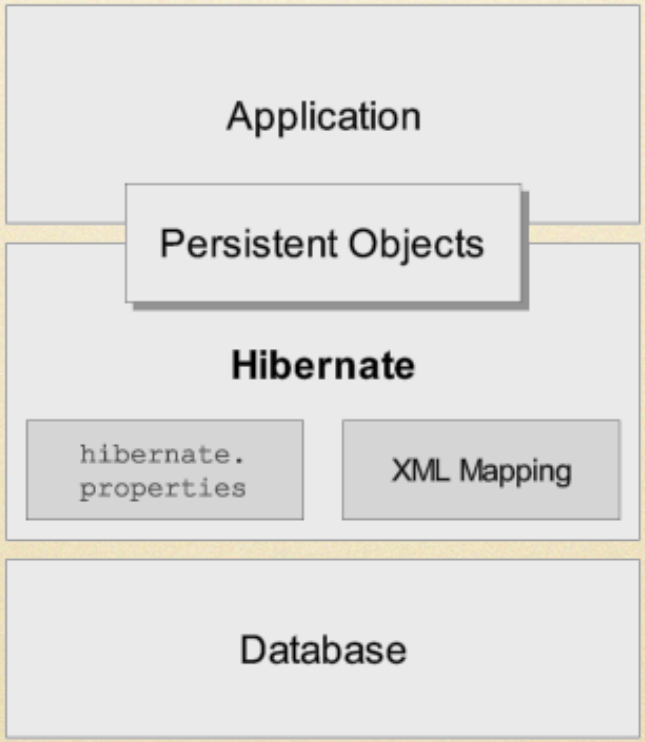
Hibernate: Simplifying persistence in Java

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

(C) 2010-2013 Prof. Dr. Ralf Lämmel, Universität Koblenz-Landau (where applicable)

The Hibernate architecture

Source: Hibernate Reference Documentation



(C) 2010-2013 Prof. Dr. Ralf Lämmel, Universität Koblenz-Landau (where applicable)

A persistent class

```
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

(C) 2010-2013 Prof. Dr. Ralf Lämmel, Universität Koblenz-Landau (where applicable)

Metadata for O/R mapping

```
<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>
```



(C) 2010-2013 Prof. Dr. Ralf Lämmel, Universität Koblenz-Landau (where applicable)

A database table

The CAT table in the database

```
 Column | Type                        | Modifiers
--------+-----------------------------+-----------
 cat_id | character(32)               | not null
 name   | character varying(16)       | not null
 sex    | character(1)                |
 weight | real                        |
```

Indexes: cat_pkey primary key btree (cat_id)

(C) 2010-2013 Prof. Dr. Ralf Lämmel, Universität Koblenz-Landau (where applicable)

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();
```

Set up session and begin transaction

Regular OO code

Make a n object persistent and commit changes.

(C) 2010-2013 Prof. Dr. Ralf Lämmel, Universität Koblenz-Landau (where applicable)

How to retrieve persistent objects?
Use HQL (Hibernate query language).

```
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() );
}
```

(C) 2010-2013 Prof. Dr. Ralf Lämmel, Universität Koblenz-Landau (where applicable)

So what's O/R mapping?

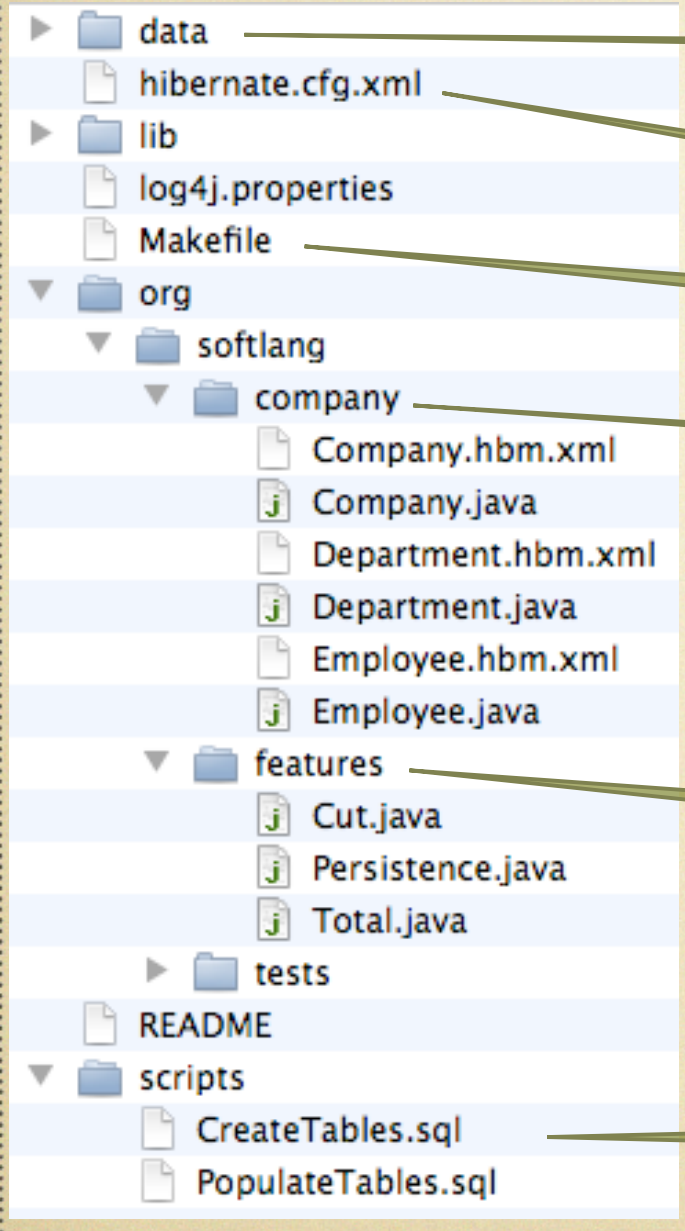
Wikipedia's definition

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.

(C) 2010-2013 Prof. Dr. Ralf Lämmel, Universität Koblenz-Landau (where applicable)

http://101companies.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) 2010-2013 Prof. Dr. Ralf Lämmel, Universität Koblenz-Landau (where applicable)

Employee POJOs

```
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) 2010-2013 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>
```

(C) 2010-2013 Prof. Dr. Ralf Lämmel, Universität Koblenz-Landau (where applicable)

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>
```

(C) 2010-2013 Prof. Dr. Ralf Lämmel, Universität Koblenz-Landau (where applicable)

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>
```

(C) 2010-2013 Prof. Dr. Ralf Lämmel, Universität Koblenz-Landau (where applicable)

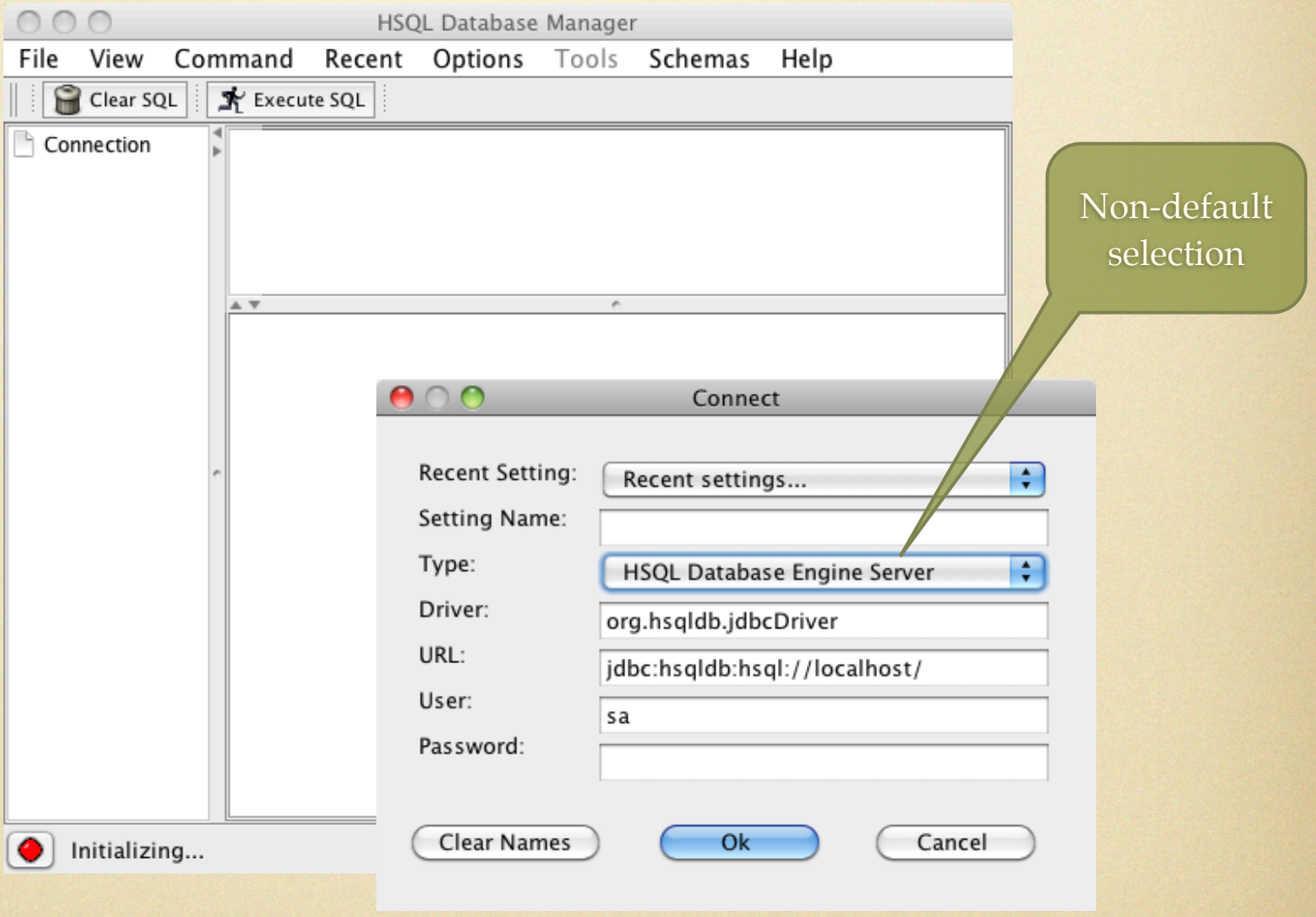
http://101companies.org/wiki/

Contribution:hibernate

DEMO

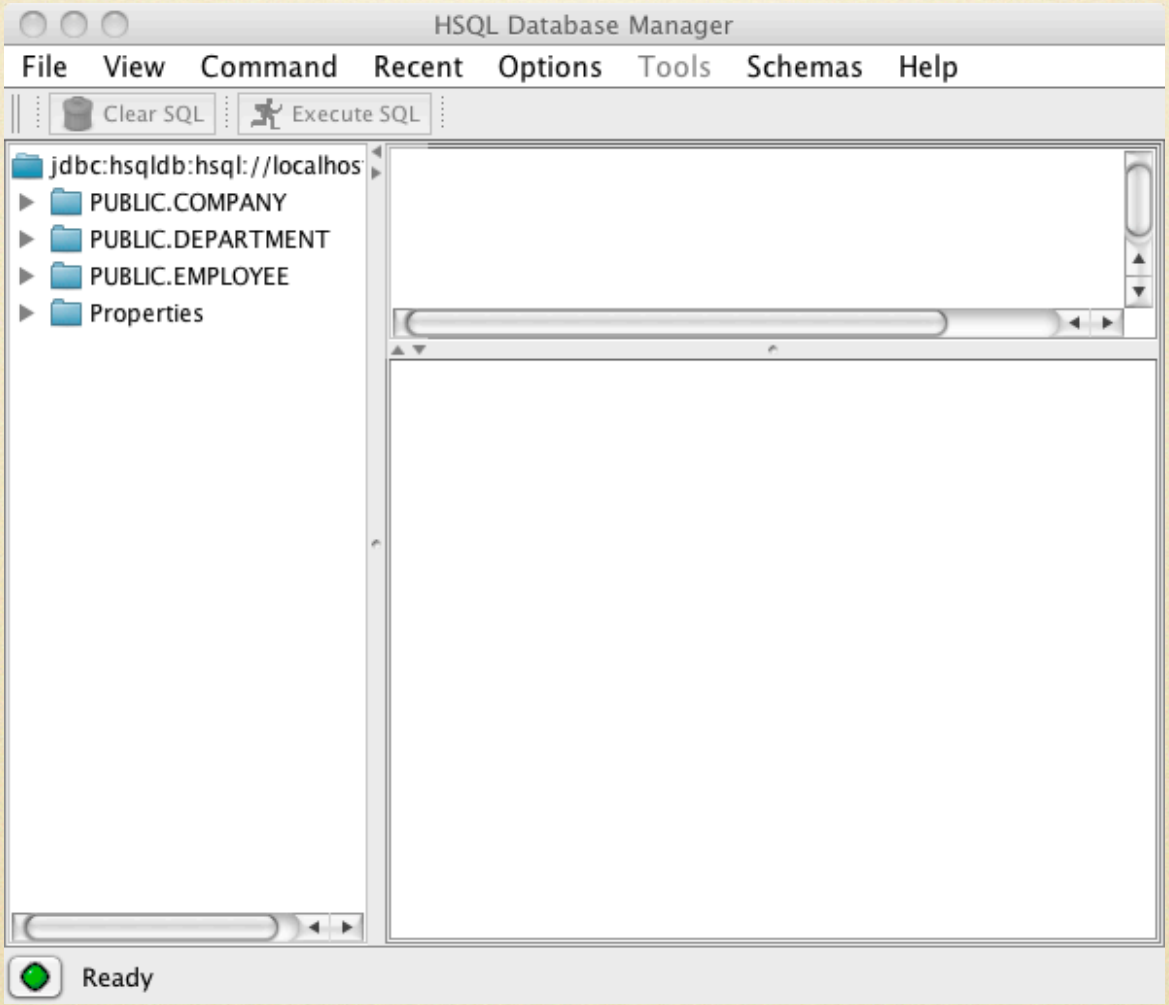
(C) 2010-2013 Prof. Dr. Ralf Lämmel, Universität Koblenz-Landau (where applicable)

After starting DB and GUI



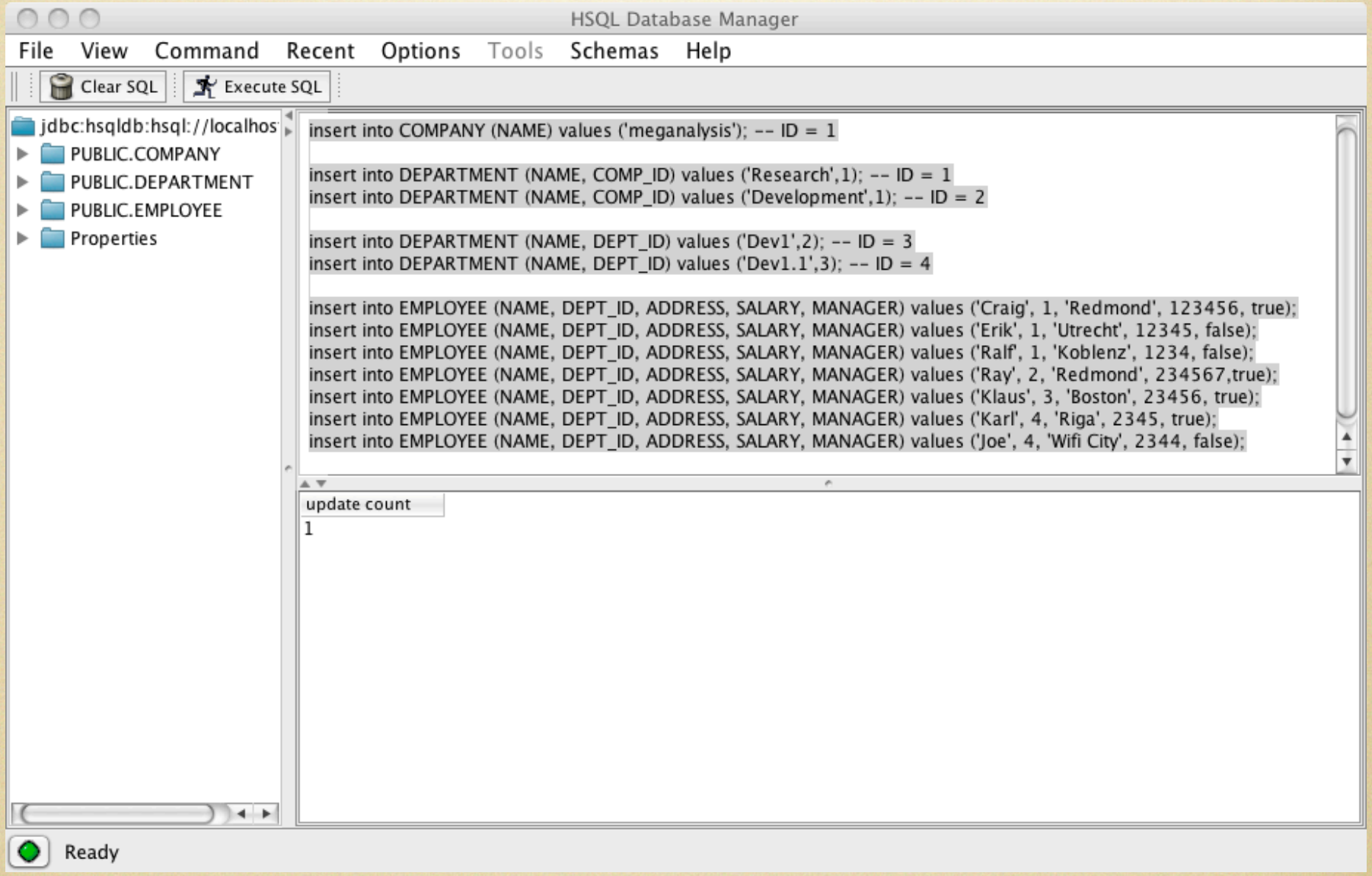
(C) 2010-2013 Prof. Dr. Ralf Lämmel, Universität Koblenz-Landau (where applicable)

After JUnit test Load and Refresh



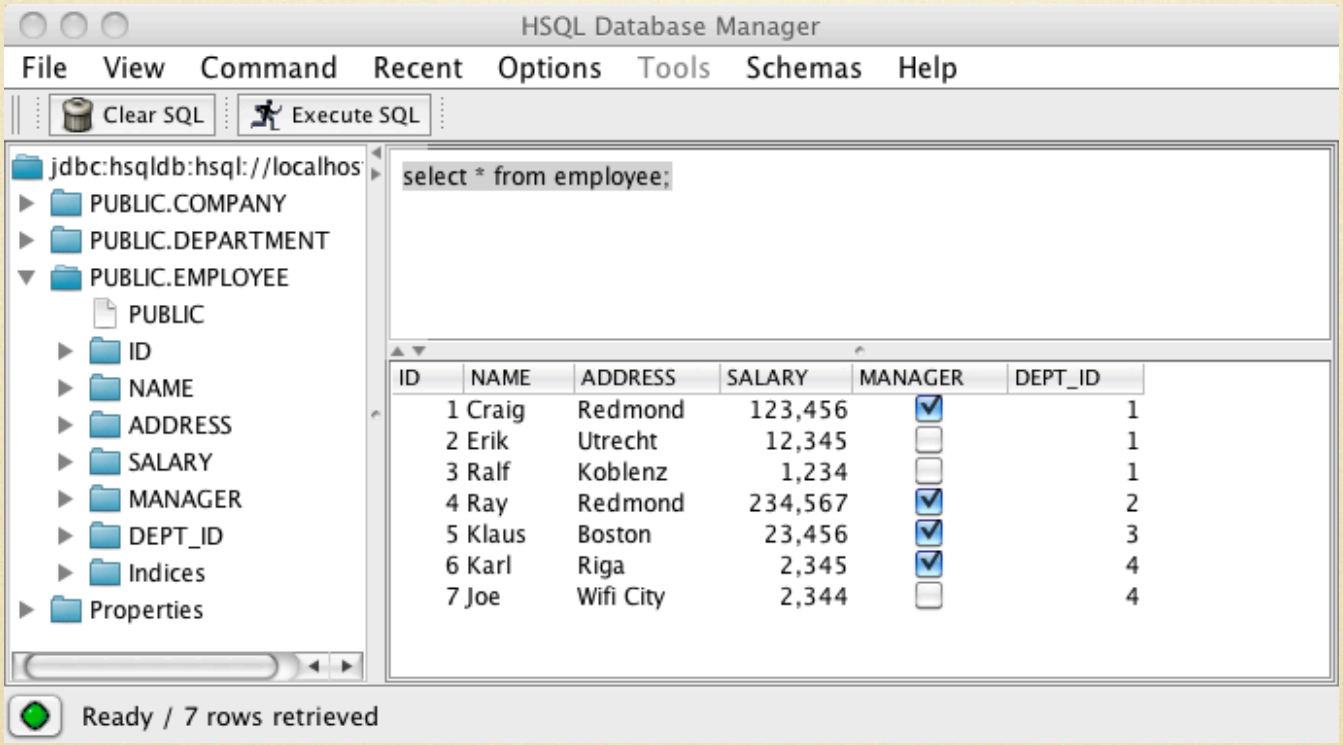
(C) 2010-2013 Prof. Dr. Ralf Lämmel, Universität Koblenz-Landau (where applicable)

After executing PopulateTables.sql



(C) 2010-2013 Prof. Dr. Ralf Lämmel, Universität Koblenz-Landau (where applicable)

Running a simple SQL query



(C) 2010-2013 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.

(C) 2010-2013 Prof. Dr. Ralf Lämmel, Universität Koblenz-Landau (where applicable)

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

(C) 2010-2013 Prof. Dr. Ralf Lämmel, Universität Koblenz-Landau (where applicable)

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

Developer's view on using Hibernate

(C) 2010-2013 Prof. Dr. Ralf Lämmel, Universität Koblenz-Landau (where applicable)

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.

(C) 2010-2013 Prof. Dr. Ralf Lämmel, Universität Koblenz-Landau (where applicable)

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.

(C) 2010-2013 Prof. Dr. Ralf Lämmel, Universität Koblenz-Landau (where applicable)

Download Hibernate

http://www.hibernate.org/

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

(C) 2010-2013 Prof. Dr. Ralf Lämmel, Universität Koblenz-Landau (where applicable)

Required for Hibernate 3.2 (+ HSQLDB support)

- hibernate3.jar
- hsqldb.jar

- 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

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.

(C) 2010-2013 Prof. Dr. Ralf Lämmel, Universität Koblenz-Landau (where applicable)

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.

(C) 2010-2013 Prof. Dr. Ralf Lämmel, Universität Koblenz-Landau (where applicable)

Hibernate relies on a RDBMS

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

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

(C) 2010-2013 Prof. Dr. Ralf Lämmel, Universität Koblenz-Landau (where applicable)

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

(C) 2010-2013 Prof. Dr. Ralf Lämmel, Universität Koblenz-Landau (where applicable)

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.

(C) 2010-2013 Prof. Dr. Ralf Lämmel, Universität Koblenz-Landau (where applicable)

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.

(C) 2010-2013 Prof. Dr. Ralf Lämmel, Universität Koblenz-Landau (where applicable)

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.

(C) 2010-2013 Prof. Dr. Ralf Lämmel, Universität Koblenz-Landau (where applicable)

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.

(C) 2010-2013 Prof. Dr. Ralf Lämmel, Universität Koblenz-Landau (where applicable)

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.

(C) 2010-2013 Prof. Dr. Ralf Lämmel, Universität Koblenz-Landau (where applicable)