

CS 683 Emerging Technologies
Fall Semester, 2004
Doc 25 O/R Mapping Intro
Contents

References.....	2
Object O/R Mapping.....	3
SQL & Relational Tables.....	4
A GLORP Example.....	8
Person.....	8
Defining the Mapping.....	9
Creating the Table in the Database.....	11
Writing a Person to the Database.....	12
Reading from the Database.....	13
Modifying Objects.....	14

Copyright ©, All rights reserved. 2004 SDSU & Roger Whitney,
5500 Campanile Drive, San Diego, CA 92182-7700 USA.
OpenContent (<http://www.opencontent.org/opl.shtml>) license
defines the copyright on this document.

References

Glorp web site <http://glorp.org/>

Glorp Tutorial, Roger Whitney

CS 580 Lecture notes, Roger Whitney

Object O/R Mapping

Examples

- GLORP (Smalltalk)
Generalized Lightweight Object-Relational Persistence
- Hibernate (Java)

SQL & Relational Tables

students

firstname	lastname	phone	code_id
John	Smith	555-9876	2000
Ben	Oker	555-1212	9500
Mary	Jones	555-3412	9900

codes

code	major
2000	Art
3000	History
9500	Electrical engineering
9900	Computer Science

```
mysql> CREATE TABLE students
(
  firstname CHAR(20) NOT NULL,
  lastname CHAR(20),
  phone CHAR(10),
  code_id INTEGER
);
```

```
mysql> CREATE TABLE codes
(
  code INTEGER,
  major CHAR(20)
);
```

SQL to access table

```
INSERT
  INTO students (firstname, lastname, phone, code_id)
SELECT
  'Roger' AS firstname,
  'Whitney' AS lastname,
  '594-3535' AS phone,
  codes.code AS code_id
FROM
  codes
WHERE
  codes.major = 'Art'
```

```
SELECT
  firstname, lastname, phone, major
FROM
  codes, students
WHERE
  major = 'Art' AND code_id = code
```

JDBC

```
import java.sql.*;

public class SampleConnection
{
    static final String ART_MAJOR_SQL = " SELECT
        firstname, lastname, phone, major
    FROM
        codes, students
    WHERE
        major = 'Art' AND code_id = code";

    public static void main (String args[]) throws Exception
    {
        Connection rugby = getDatabaseConnection();
        Statement getArtMajors = rugby.createStatement();
        ResultSet artMajorsRows =
            getArtMajors.executeQuery(ART_MAJOR_SQL);
        ArrayList artMajors = new ArrayList();
        while (artMajorsRows.next() )
        {
            Student artMajor = new Student();
            artMajors.add( artMajor);
            artMajor.setFirstName(artMajorsRows.getString( "firstname"));
            artMajor.setLastName(artMajorsRows.getString( "lastname"));
            artMajor.setPhone(artMajorsRows.getString( "phone"));
            artMajor.setMajor(artMajorsRows.getString( "major"));
        }
        doSomethingWithArtMajors(artMajors);
        rugby.close();
    }
}
```

JDBC Example Continued

Connection getConnection() throws SQLException

```
{
String dbUrl = "jdbc:mysql://rugby.sdsu.edu:8777/test";
String user = "whitney";
String password = "mylittleSecret";

Class.forName("com.mysql.jdbc.Driver");
return DriverManager.getConnection( dbUrl, user, password);
}
}
```

A GLORP Example

Person

```
Smalltalk defineClass: #Person
  superclass: #{Core.Object}
  indexedType: #none
  private: false
  instanceVariableNames: 'firstName lastName '
  classInstanceVariableNames: "
  imports: "
  category: 'GlorpExperiments'
```

Person class methods

```
first: firstNameString last: lastNameString
  ^self new setFirst: firstNameString last: lastNameString
```

Person instance methods

```
setFirst: firstNameString last: lastNameString
  firstName := firstNameString.
  lastName := lastNameString
```

```
firstName: anObject
  firstName := anObject
```

Defining the Mapping

```
Smalltalk defineClass: # GlorpTutorialDescriptor
  superclass: #{Glorp.DescriptorSystem}
  indexedType: #none
  private: false
  instanceVariableNames: "
  classInstanceVariableNames: "
  imports: 'Glorp.DirectMapping'
  category: 'GlorpExperiments'
```

```
GlorpTutorialDescriptor >>allTableNames
^#( 'PEOPLE' )
```

```
tableForPEOPLE: aTable
  aTable createFieldNamed: 'first_name' type: (platform varChar: 50).
  (aTable createFieldNamed: 'last_name' type: (platform varChar: 50))
  bePrimaryKey.
```

```
descriptorForPerson: aDescriptor
```

```
| table |
table := self tableNamed: 'PEOPLE'.
aDescriptor table: table.
(aDescriptor newMapping: DirectMapping)
  from: #firstName
  to: (table fieldNamed: 'first_name').
(aDescriptor newMapping: DirectMapping)
  from: #lastName
  to: (table fieldNamed: 'last_name')
```

Defining the Mapping Continued

```
classModelForPerson: aClassModel  
  aClassModel newAttributeNamed: #firstName.  
  aClassModel newAttributeNamed: #lastName.
```

```
constructAllClasses  
^(super constructAllClasses)  
  add: Person;  
  yourself
```

Creating the Table in the Database

```
login := Login new database: PostgreSQLPlatform new;
  username: 'whitney';
  password: 'foo';
  connectionString: '127.0.0.1_glorpTests'.
```

```
accessor := DatabaseAccessor forLogin: login.
accessor login.
```

```
session := GlorpSession new.
session system:
  (GlorpTutorialDescriptor forPlatform: login database).
session accessor: accessor.
```

```
session inTransactionDo:
  [session system allTables do:
    [:each |
      accessor
        createTable: each
        ifError: [:error |Transcript show: error messageText]]].
```

One does not have to use GLORP to create tables in the database. One can create them by hand. In many cases that will be the case.

Writing a Person to the Database

```
person := Person first: 'Pete' last: 'Frost'.  
session beginUnitOfWork.  
session register: person.  
session commitUnitOfWork.
```

All objects registered in a unit of work (and with the proper GLORP mapping) will be written to the database in the correct table(s) when the unit of work is completed

Reading from the Database

Reading all Rows

people := session readManyOf: Person .

people will be a collection of Person objects.

Reading one Person

```
foundPerson := session  
    readOneOf: Person  
    where: [:each | each firstName = 'Jose']
```

Modifying Objects

```
session beginUnitOfWork.  
foundPerson := session  
    readOneOf: Person  
    where: [:each | each firstName = 'Jose'].  
foundPerson firstName: 'RamJet'.  
session commitUnitOfWork
```

Any object read from the database in a unit of work is automatically registered with the using of work. Any object registered with a unit of work that is modified during the unit of work will have the modifications saved to the database when the unit of work is committed. The proper rows in all required tables are updated.