# Advanced Database
# Basket-ball Management System
# Project Report

Nguyen Huu Phat, Le Xuan Hoan
Supervisor: Prof. Jérôme Charton

**Abstract**

One of the most important advantages of Object-Oriented method (OOM) is its ability to model very naturally many real world problems. Static aspects (such as class, objects, relationships ... ) as well as dynamic aspects (also known as behavioral aspects) of OOM helps to perceive complex system effectively. In the filed of DBMS, many Object-Oriented DBMS (OODBMS) has been developed within the past decade. However despite of the superiorities and benefits of OODBMS over Relational DBMS (RDBMS), OODBMS is still not widely used because RDBMS has its own strong points, it's simple and considered very mature. These above observation has motivated for a new approach appears, and promised as a ideal solution, Object-Relational method. In general and for simplicity, this method combines the strengths of each as well as eliminate their shortcomings. In this report, we will present about applying Object-Relation feature of Oracle DBMS to build a Basket-Ball Meeting Management System as well as Java techniques to interact with an Oracle Object-Relational database. Furthermore we also mention another interesting feature of Oracle DBMS, support for management of spatial data

## 1 Introduction

This is the final report for a project in Advanced Database (AD) course at PUF. The requirements of this project is to build a Basket-Ball Meeting Managemet System (BMS) with Oracle Object-Relational database; apply knowledge which we learnt from AD course into a real-world problem. The project designs a object oriented database within Oracle 10g, using Java as programming language, and also apply Oracle spatial feature to manage spatial data

In the next sections, we will give a brief overview about modeling our system, including modelization of the data, create an Object-Relational database and implement an application in JAVA.

## 2 Conceptual model

Based on customer requirements, we design a conceptual model for BMS. Note that as project's requirements, BMS is implemented on Oracle Object Oriented database, to each entity in conceptual database will be implement as Abtract Data Type in Oracle.

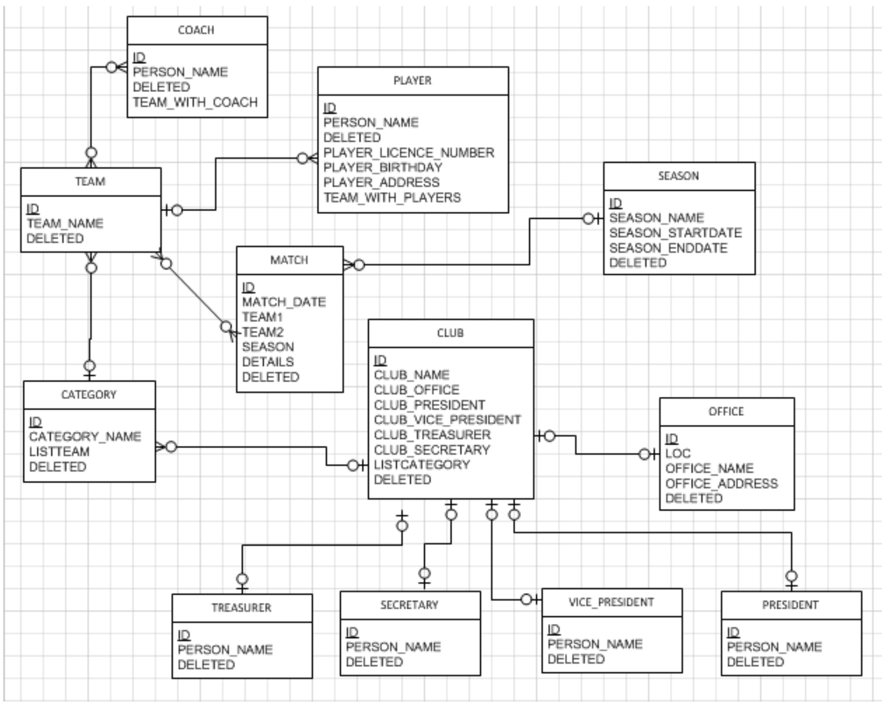

Figure 1: Conceptual model of BMS

## 3 Data modeler

Please see file DataModeler.pdf for details. In that file, we will present clearly about structure of our database based on Object Oriented concepts

## 4 Oracle Spatial

Spatial data support is an advantage of Oracle DB. Of course we do not plan to make a GIS like Google Maps, however we decided to apply somewhat this interesting feature in our project. Specifically our Office object holds a SDO_GEOMETRY attribute, and a function distance to calculate length (in km) between two offices. When create demo data for the program, we also insert to Office objects coordinate of some cities in Viet Nam (these coordinates (longitude and latitude) we extract from Google Maps). Please try our program to see the result of distance calculation

## 5 Access Object-Relational schema through JDBC

### 5.1 Weakly Typed Objects

This approach will materialize DB objects to Java objects that implement interface java.sql.Struct. These Struct instances contain an array of attributes, which are corresponding to attribute of DB objects, in the same order as we defined DB objects, following figure will help a visual explanation

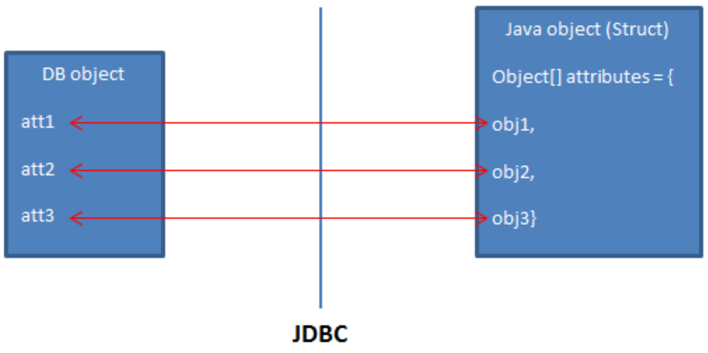

Figure 2: Weakly Typed mapping

### 5.2 Strongly Typed Objects

This approach provides a stronger mapping between DB object model and Java object model. Unlike in weekly typed mapping, every DB object is uniformed to Struct-based objects, with strongly typed mapping, every DB object is corresponding to a Java class. Actually, for each DB object, we have a pair of Java classes to represent the DB object itself and its reference, following give a graphical illustration (for Player)

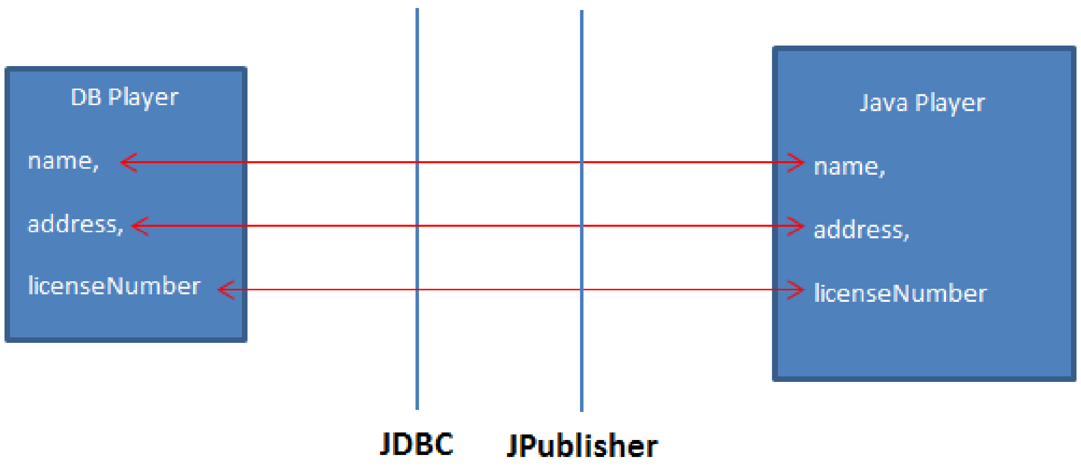

Figure 3: Strongly Typed mapping

We chose the second way, the reason is it's more convenient and less error-prone than using weekly typed objects (obviously processing of reference in weekly typed mapping could be a complicated problem). Additionally we don't need to create these Java classes manually, with the help of an utility from Oracle, JPublisher, these classes are generated automatically

### 5.3 JPublisher

JPublisher is a tool that can generate representation in Java (as Java classes) of DB entities such as objects, PL/SQL packages. In general, JPublisher can help to map following DB entities:

- DB objects (built-in or user-defined)
- Object references
- Collection types
- PL/SQL packages
- Server-side Java Classes
- SQL queries and DML statements

For this project, we will only focus on three first entities. For more information about JPublisher, please refer to Oracle Database JPublisher User's Guide (10g Release 1), supplied by Oracle

JPublisher takes an inout file consisting of one or many translation statements. A translation statement instructs JPublisher to map between a DB entity and Java classes. Following is syntax of translation statement

You can find our translation statements for the DB model in the appendix

## 6 Java object model

After using JPublisher to generate corresponding Java object model, we modify this model a little to satisfy better our requirements. We explain it in detail right below this class diagram

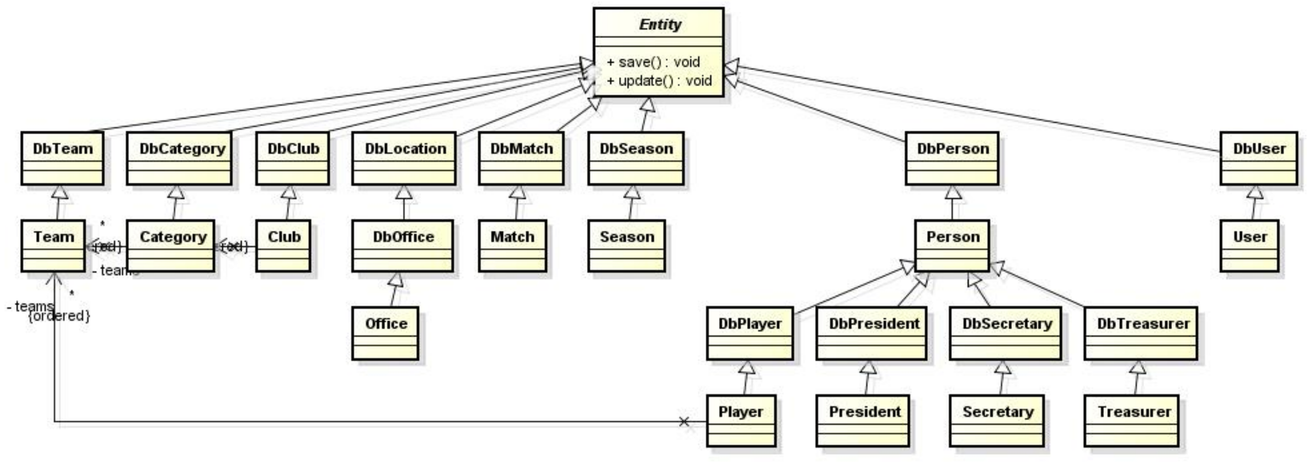

Figure 4: Class diagram of most important part of JAVA model

In this diagram, the classes prefixed by DB (DBPlayer, DBMatch ...) are corresponding classes of DB objects such as Player, Match, we denote this kind of classes as DB-mapped class. For each of these classes, we extend a sub-class (see in the diagram). These sub-classes will contain our custom source code (such as save(), update() ...). That means we can separate our custom source code and JPublisher generated source code. By this way, when our DB model changes, we only need to run JPublisher to regenerate DB-mapped classes, our custom code (e.g. save(), update() method) is preserved.

Moreover, we also create a super class for all of DB-mapped classes, Entity. This super class is abstract and contains generic methods such as save(), update(). For most of the case, these methods are enough to save, update DB-mapped classes to DB.

**EntityUtils**
This is a utility class that has an important generic method, loadByCondition

This method receives a Condition c, a Class T and returns a list of objects corresponding to Class T and satisfy Condition c.

Condition is our abstraction for where condition in SQL statement, we can use it to denote conditions without grouping (without parentheses), for example: att1 = val1 and att2 = val2 or att3 = val3

## 7 Used tools

- JDK 7
- Eclipse Indigo 3.7.2
- Oracle 10g Express Edition
- Oracle SQL Developer 3.1.07

- Oracle SQL Developer Data Modeler 3.1.1.703
- JPublisher 10g Release 10.2

## 8 Conclusion

Basically, the program meets our advisor's requirements, it is able to work as a simple basket-ball management system. However, as mentioned above, there are still some points waiting to improve, especially about application advanced subjects in Object-Relational and spatial features of Oracle DBMS.

The usability should be improved too.The program should implement more real-world feature for a Basket Management System

## 9 Acknowledgements

We would like to express a tremendous thank to Prof. Jérôme Charton, who has raised a very interesting project for us. Object-Relational database and spatial data were not familiar for us before and really not easy for us during the project. But with a continuous support from the advisor, we have gained much knowledge about it and made our own interest in this subject

We also thank our classmates for their encouragements during our project.

## References

[1] R. M. Menon, "Expert Oracle JDBA Programming", Apress (June 2, 2005), ISBN: 978-1590594070

[2] Ravi Kothuri, Albert Godfrind and Euro Beinat, "Pro Oracle Spatial for Oracle Database 11g", Apress (October 29, 2007), ISBN: 978-1590598993

[3] "Working with Oracle Object References", Oracle9i JDBC Developer's Guide and Reference; Release 2 (9.2); Part Number A96654-01, http://docs.oracle.com/cd/B10500_01/java.920/a96654/oraoor.htm

[4] Wenny Rahayu, David Taniar, Eric Pardede, "Object- Oriented Oracle", IRM Press (August 16, 2005), ISBN: 978-1591408109