AC 2008-545: A COMPARISON OF IMPLEMENTING A SINGLE PROBLEM IN RELATIONAL, OBJECT-RELATIONAL AND OBJECT-ORIENTED DATABASE SYSTEMS

Floyd Wilkes, Utah Valley State College

Reza Sanati-Mehrizy, Utah Valley State College

A Comparison of Implementing a Single Problem in Relational, Object-Relational and Object-Oriented Database Systems

ABSTRACT

Several database textbooks were studied to determine how extensively the three database models Relational, Object-Relational and Object-Oriented were been covered [1]. From this study it was determined that some database textbooks either do not cover all three models or they cover them very briefly. None of the textbooks in this study provide a complete database that is represented in all these three models. The authors assert that students should see the benefits and difficulties of using each of the models since problems can be solved more easily in one than in another.

Keywords

Database Design, Relational databases, Object-Relational databases, Object-Oriented databases.

1. INTRODUCTION

The database world is in a state of change. After E. F. Codd published his seminal paper on the relational model [2], this model dominated the database world for the next 20 years. Since the 1990s however, other forces have been emerging which require recognition. First, the programming world shifted from a structured-procedural to an object-oriented paradigm with its attendant shift from storing data outside the program to storing data inside program objects. Using a relational database with an Object Oriented program requires moving data from inside objects to the relational structure which requires additional processing and adds complexity to a program. A second change concerns the nature of the data being stored. The problems being solved with computers have become more complex as has the data associated with these problems. Mapping complex data into the atomic data types of the relational model is not an adequate solution for these problems.

Two solutions for these changing conditions have emerged. One has been the development of Object-Oriented databases (OODB). Using an OODB allows a developer to store objects in the database as objects rather than mapping their data into a relational structure [2]. A second approach, usually referred to as the Object-Relational (ORDB) approach, has required modifying or extending the relational model to allow complex data to be stored in the database. This is accomplished through a data structure called a user defined data types (UDT) [3]. Many of the ORDB concepts have been incorporated into the SQL 1999 standard [4].

2. BACKGROUND

Our teaching experience indicates to us that most students understand relational database concepts quite quickly. But, understanding Object Relational and Object-Oriented database concepts is more challenging for many of them. It is our contention that understanding these latter models would be easier for students if they saw the same database example presented in all these three models. Thus, compare and visualize how the implementation changes from a model to another.

This paper discusses a patient database that we have implemented in all three database models to ease the understanding of Object Relational and Object Oriented database concepts. Also, a set of queries will be executed using these three implementations and the results will be shown for comparison.

3. THE PROBLEM DOMAIN

The Database Management Systems used when developing material for this paper were: Relational, Oracle 10g (only features available prior to SQL 99[4]); Object-Relational, Oracle10g (full features of the system); Object-Oriented, db4o, an open-source Object-Oriented database available at www.db4o.com[7] that supports the Object Oriented features of Java and C#.

The problem we have used for our illustration is a medical database that might used to support patient care. Part of the basis for this problem was borrowed from Hoffer [3]. The database is not intended to be a "real world" solution to the problem, rather to illustrate differences of implementation in the three models. The tables or classes defined for the problem domain are shown in Figure 1.

The business process supported by this database assumes that patients present themselves for treatment. Before treatment can begin, the patient must receive a diagnosis from a physician. Physicians may or may not be involved in diagnosing patients. From the diagnosis and the severity of the patient's condition, it would be determined whether the patient is to become an In-patient or an Out-patient. In-patients are assigned to a bed and Out-patients are simply scheduled for a follow-up visit, if needed.

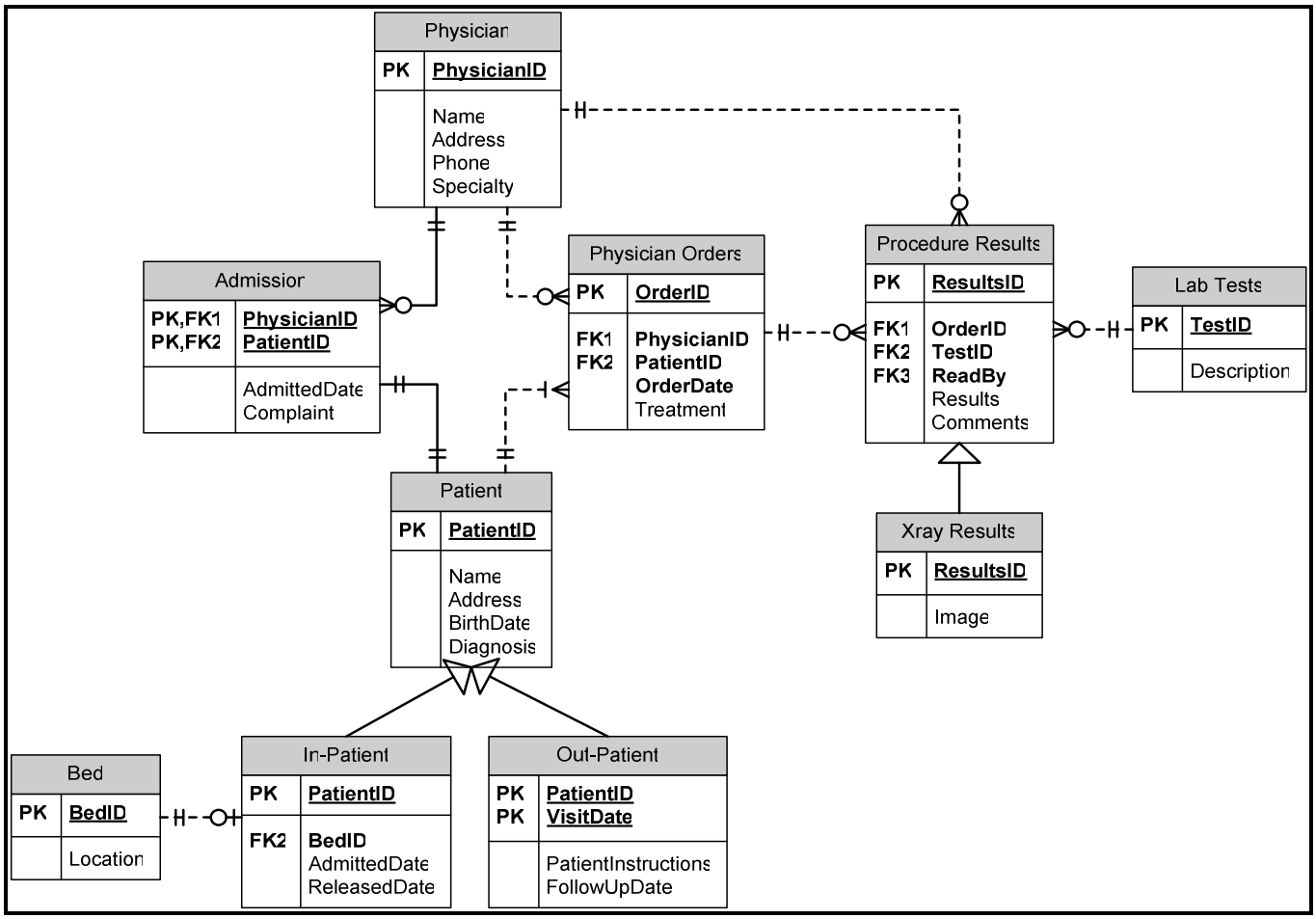

Figure 1. Medical Database Schema

After being admitted, a treating physician orders various evaluations and treatments for a patient which may include laboratory tests and X-ray procedures. Available procedures are listed in the Lab Tests and X-ray Procedures tables. Treatment orders are recorded in the Physicians Orders table and may include multiple procedures on a single day or recurring procedures on successive days. Each treatment procedure is provided by a department in the hospital. In the real world treatments could include surgical, pharmacological and other types of treatment; however, for purposes of this example, we have excluded other treatment modalities. The results from treatment procedures are recorded in the database for reference by the physician and treatment staff.

As part of our illustration, we have also defined these four questions that to be answered by queries:

1. List patients by patient type (In-Patient, Out-Patient), location, and disorder.
2. What are the patient's name and the treating physician's name of the patient in bed number 16?
3. What are the lab test results for patient X?
4. What are the names of patients that have diagnosis Y?

In the sections that follow, we show coding examples of using the three different models. Due to space limitations in this paper, we will only discuss creation and data manipulation operations concerned with the Patient, In-Patient and out-Patient portions of the database. For those who may be interested, an extended version of the database and related operations is available from the authors.

3.1 A Relational Example

For the relational example, the code shown in this section is used to create, populate and query the Patient, In-Patient and Out-Patient tables. All code in this section conforms to pre-SQL99 standards. All data types are atomic. All code in this section and the Object-Relational section conforms to code specifications found in the Oracle Database 10g: Complete Reference[5].

3.1.1 Creating Entities

```
CREATE TABLE Patient(
        patientNo       number(10),
        lastName        VARCHAR2(20),
        firstName       VARCHAR2(20),
        street          VARCHAR2(20),
        city            VARCHAR2(20),
        state           CHAR(2),
        zip             VARCHAR2(10),
        DOB             DATE,
        diagnosis       VARCHAR2(50));
CREATE TABLE InPatient(
        patientNo       number(10),
        dateAdmitted    DATE,
        dateReleased    DATE,
        bedId           number(3));
CREATE TABLE OutPatient(
        patientNo       number(10),
        dateOfVisit     DATE,
        instructions    VARCHAR2(200),
        followUpDate    DATE);
```

3.1.2 Populating Entities

Adding an In Patient to the database requires these two inserts.

```
insert into Patient
values(150, 'Jones', 'Betty', '250 E. Center Street', 'Orem', 'UT', '84097', '27-Jul-50');
insert into InPatient
values (150, '6-Apr-07', '15-May-07', 'E37');
```

3.1.3 Querying Entities

```
SELECT *
FROM    Patient JOIN
          InPatient ON Patient.PatientNo =    InPatient.PatientNo
```

3.2 An Object-Relational Example

The code for the Object-Relational example illustrates the use of UDT's (User Defined Types). In this example an Address Type, a Phone Type and a Name Type are defined first and then used in the definition of the Patient Table. Another difference from the Relational Example is the use of sub-types which allows for inheritance.

3.2.1 Creating Enities

```
CREATE OR REPLACE TYPE AddressType AS OBJECT(
        street      VARCHAR2(50),
        city        VARCHAR2(15),
        state       CHAR(2),
        zip         CHAR(10));
CREATE OR REPLACE TYPE PhoneType AS OBJECT(
        areaCode            CHAR(3),
        telephoneNumber     CHAR(14));
CREATE OR REPLACE TYPE NameType AS OBJECT(
        firstNameVARCHAR2(15),
```

```
    lastName VARCHAR(20));
CREATE OR REPLACE TYPE PatientType AS OBJECT(
    patientNo       number(10),
    name            SYSTEM.NameType,
    address         SYSTEM.AddressType,
    DOB             DATE,
    diagnosis       VARCHAR2(50));
    NOT FINAL;
CREATE OR REPLACE TYPE InPatientType UNDER PatientType(
    dateAdmitted    DATE,
    dateReleased    DATE,
    bedId           REF BedType)
    FINAL;
CREATE OR REPLACE TYPE OutPatientType UNDER PatientType(
    dateOfVisit     DATE,
    followUpDate    DATE,
    instructions    VARCHAR2(200))
    FINAL;
CREATE TABLE Patient of PatientType(
    CONSTRAINT Patient_PatientId_pk PRIMARY KEY (patientNo))
    OBJECT IDENTIFIER PRIMARY KEY;
```

3.2.2 Populating Entities

Here is an example of the code used to insert an Out Patient in the database.

```
insert into Patient values(OutPatientType(100,NameType('Roy','Smith'),AddressType('100 E. Center Street', 'Orem', 'UT','84097'), '27-Jul-50', '21-May-99' ,'23-May-99', 'Do not eat solid food for 24 hours'));
```

3.2.3 Querying Entities

```
select patientno, c.name.firstname, c.name.lastname, c.address.street, c.address.city,c.address.zip from patient c WHERE VALUE(c) IS OF (ONLY OutPatientType);
```

3.3 An Object-Oriented Example

In the Object-Oriented example, we shift from tables to objects. The objects stored in the database are created in the host language which in this case is Java. In our example, a class is defined for Patient, In Patient and Out Patient. After a class has been instantiated as an object, the object is loaded with data before being moved to the db4o database for permanent storage. What is shown in subsubsections 3.3.1 through 3.3.3 are code segments from a Java program that creates, stores and queries the Patient, In Patient and Out Patient objects in the sample database. Code in this section conforms to Java JDK 5[6].

3.3.1 Creating Entities

```
public class Patient {
    private Integer patientNo;
    private String name;
    private String address;
    private Date dateOfBirth;
    private String diagnosis;
    public Patient(Integer patientNo, String name, String address, Date dateOfBirth, String diagnosis) {
        this.patientNo = patientNo;
        this.name = name;
        this.address = address;
```

```
                this.dateOfBirth = dateOfBirth;
                this.diagnosis = diagnosis;
        }
        public String toString() {
                return patientNo + " " + name + " " + address + " " + dateOfBirth + " " + diagnosis;
        }
        public Integer getPatientNo() {
                return patientNo;
        }
        public void setPatientNo(Integer patientNo) {
                this.patientNo = patientNo;
        }
        public String getName() {
                return name;
        }
        public void setName(String name) {
                this.name = name;
        }
        Note: Additional getters and putters appear here for each attribute in Patient
}
public class Inpatient extends Patient {
        private Date dateAdmitted;
        private Date dateReleased;
        private Bed bed;

        public Inpatient(Integer patientNo, String name, String address, Date dateOfBirth, String diagnosis,  Date
        dateAdmitted, Date dateReleased, Bed bed) {super(patientNo, name, address, dateOfBirth, diagnosis);
                this.bed = bed;
                bed.setPatient(this);
                this.dateAdmitted = dateAdmitted;
                this.dateReleased = dateReleased;
        }

        public Date getDateAdmitted() {
                return dateAdmitted;
        }
        public void setDateAdmitted(Date dateAdmitted) {
                this.dateAdmitted = dateAdmitted;
        }
        Note: Additional getters and putters appear here for each attribute in Inpatient

public class Outpatient extends Patient {
        private Date dateOfVisit;
        private String instructions;
        private Date followUpDate;
```

```
	public Outpatient(Integer patientNo, String name, String address, Date dateOfBirth, String diagnosis, Date
dateOfVisit, String instructions, Date followUpDate) {
		super(patientNo, name, address, dateOfBirth, diagnosis);
		this.dateOfVisit = dateOfVisit;
		this.instructions = instructions;
		this.followUpDate = followUpDate;
	}
	public Date getDateOfVisit() {
		return dateOfVisit;
	}
	public void setDateOfVisit(Date dateOfVisit) {
		this.dateOfVisit = dateOfVisit;
	}
	Note: Additional getters and putters appear here for each attribute in Inpatient
}
```

3.3.2 Populating Entities

The database is populated when objects are built in the host language and then moved to the database. This code accomplishes this task.

```
public static void main(String[] args) {
	ObjectContainer db=Db4o.openFile("temp.db");
	try {
	Inpatient johnAbraham = new Inpatient(16, "John Abraham", "123 Cresent Pl.", new Date(), "Heart Blockage", bed15, new
	Date(), null);
	Outpatient jimWeb = new Outpatient(10, "Jim Web", "8322 N. 220 E.", new Date(), Ulcerated Stomach",
	new Date(), "", null);

	db.set(johnAbraham);
	db.set(jimWeb);
	} finally{
	db.close();
}
```

3.3.3 Querying Entities

```
public class ApplicationLauncher {
	public static void listResult(ObjectSet result) {
		System.out.println(result.size());
		while (result.hasNext()) {
			System.out.println(result.next());
		}
	}
	List <Patient> patients = db.query(Patient.class);
	System.out.println(patients);
	}
```

}

4. COMPARISONS

There are obvious differences not only in the content of the coding for the three models but also in the quantity of implementation code required in each model. This is explainable by the increase in the number entities required by the Object-Relational and the Object-Oriented models. Conversely, the Object-Relational and Object-Oriented models are more compatible with Object-Oriented programming techniques.

5. CONCLUSIONS

This paper presents an abbreviated example of the implementing a common problem in three database models. A more complete example is in preparation for future presentation and for use in the database classroom. The results of this use will be evaluated to determine the effectiveness of presenting a common problem in the three models.

6. ACKNOWLEDGMENTS

This research was partially supported by a Presidential Scholar Award from our institution.

7. REFERENCES

[1] Sanati, R. and Wilkes, F., An International Review of the representation of data in Relational, Object-Relational and Object-Oriented databases, Proceedings of IACIS Conference, October, 2006.

[2] E. F. Codd: "A Relational Model of Data for Large Shared Data Banks," CACM 13, 6 (June 1970).

[3] Hoffer, J., Prescott, M., and McFadden, F., Modern Database Management, 8e. Pearson Prentice Hall, 2007

[4] Eisenberg, A. and Melton, J, SQL:1999, formerly known as SQL3. Retrieved March 3, 2006 from http://dbs.uni-leipzig.de/en/lokal/standards.pdf

[5] http://db4o.com/

[6] Loney, K., Oracle Database 10g: The Complete Reference, McGraw-Hill/Osborne, 2004

[7] http://java.sun.com/javase/downloads/index_jdk5.jsp