VICTORIA UNIVERSITY OF WELLINGTON
Te Whare Wananga o te Upoko o te Ika a Maui

Department of Computer Science

PO Box 600
Wellington
New Zealand

Tel: +64 4 471 5328
Fax: +64 4 495 5232
Internet: Tech.Reports@comp.vuw.ac.nz

Towards normalization in object-oriented databases

Gillian Dobbie

Technical Report CS-TR-96/16

Abstract

There are two common approaches to designing object-oriented database schemas. The first is to follow an object-oriented analysis and design methodology, and the second is to translate an extended entity relationship diagram to an object-oriented schema. Neither of these approaches provide a way of choosing between or evaluating alternative object-oriented designs.

In this paper, we summarize what normalization provides for relational databases and consider if a concept like normalization could provide a way of evaluating alternative object-oriented designs.

Author Information

Gillian Dobbie
Department of Computer Science
Victoria University of Wellington
PO Box 600
New Zealand
gill@comp.vuw.ac.nz

1 Introduction

There are two common approaches for designing object-oriented database schemas. The first is to follow an object-oriented analysis and design methodology like Booch [1] or Rumbaugh [4]. Typical steps in these methodologies involve:

- finding objects
- classifying objects to classes
- defining the internal structure of classes
- modelling static connections between classes (instance and inheritance connections)
- modelling structural connections (aggregation connections)
- describing message passing between objects
- describing the internal behaviour of classes.

The second is to translate an extended entity relationship (EER) diagram to an object-oriented schema [2].

Neither of these approaches provides a way of choosing between or evaluating alternative object-oriented database (OODB) designs. In the relational database area, there is a well understood method of evaluating alternative relational database designs, namely normalization. In this paper, we use examples to show how some of the ideas of normalization can be applied to object-oriented database design. Currently, there is a dearth of research in this area and this paper is merely a starting point. Papers that relate to design or functional dependencies in object-oriented databases do not give any details of how to evaluate alternative OODB designs.

In [5], the authors compare relational database design with object-oriented database design. They conclude that there are some advantages in using an OODB design over a relational design. The main advantage they give is that using object-oriented concepts in the design phase allows one to capture more application semantics in an early stage of the software life cycle process. They also express the need for a theoretical background for OODB design, a collection of quality metrics measuring relevant properties of an OODB design, and transformation rules which can be applied to object database schemata to reach a higher degree of quality. They do not provide any of these.

In [3], the authors describe a kind of entity relationship (ER) diagram called a normal form object-oriented entity relationship diagram. The work they describe extends previous work they have done in normalizing ER diagrams. Moreover, the authors do not consider what normalization means in the object-oriented setting.

The author in [6] proposes a more general functional dependency constraint for the object-oriented data model, called path functional dependency. However, the author does not consider how these functional dependencies can be used to resolve redundancies in an object-oriented schema.

In the following section, we consider three separate relation schema examples that are not in 1NF (first normal form), 2NF (second normal form) and 3NF (third normal form) respectively. We demonstrate how these examples could be modelled equally poorly in object-oriented databases and then introduce alternative schemas with less redundancy. Features of object-oriented databases, that don't exist in relational databases, and the redundancies that they may introduce are discussed in section 3. Finally, we summarize our findings in section 4.

2 Normalization in object-oriented databases

Normalization can be seen as enforcing certain constraints on a database. The constraints disallow certain kinds of redundancy. Redundancy in a database leaves the database open to update anomalies. In order to maintain consistency in a database, where data is stored in more than one place then and that data is updated, it must be updated in more than one place.

In this section, we give definitions for 1NF, 2NF and 3NF in relational databases, consider examples that are not in these normal forms and decompose the schemas until they conform to the normal forms. We then consider how these examples could be modelled in an object-oriented database and if the same kind of redundancies exist in the object-oriented setting. The definitions and examples are based on those in [2].

2.1 1NF

A relation is in 1NF if the value of any attribute in a tuple is atomic.

Example 2.1 Consider the situation where each employee may work on more than one project, where an employee is represented by their social security number and their name, a project is identified by a project number, and an employee spends a certain number of hours on each project with which they are involved. This situation can be modelled by the schema **EMP_PROJ** in which the attribute **projs** is a multivalued composite attribute. The schema **EMP_PROJ** is not in 1NF because the value of **projs** is not atomic.

EMP_PROJ

ssn	emp_name	projs	
		proj_num	hrs

One way that this schema can be flattened to a relational model is to repeat values for **ssn, emp_name** for every **proj_num, hrs** pair. This flattening obviously introduces redundancy. The redundancy can be removed by decomposing the schema **EMP_PROJ** into the two relations **EMP** and **E_P** shown below. These relation schemas are in 1NF.

EMP

ssn	emp_name

E_P

ssn	proj_num	hrs

One of the features of object-oriented databases is that attributes can be multi-valued and composite. For this reason, the constraint defined in 1NF need not be enforced in the design of object-oriented databases. The relation schema defined above can be modelled as follows in an OODB.

```
EMP_PROJ
  ssn : integer;
  emp_name : string;
  projs : {(proj_num:integer, hrs:integer)};
```

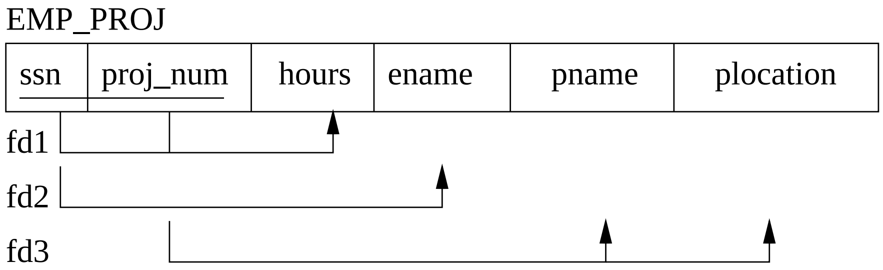
In the above object-oriented schema, **EMP_PROJ** is a class with attributes **ssn, emp_name** and **projs**. Attribute **projs** is a multi-valued attribute where each value of the attribute is a tuple. This tuple has attributes **proj_num** and **hrs**.

2.2 2NF

A relation schema R is in 2NF if it is in 1NF and every non-prime attribute in R is fully functionally dependent on the primary key of R.

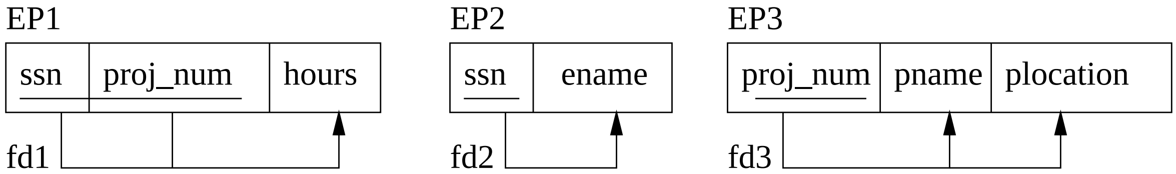
A **non-prime attribute** is an attribute that is not part of the primary key. A **functional dependency**, denoted by $X \rightarrow Y$, between two sets of attributes X and Y means that the values of the Y component of a tuple in a relation depend on, or are determined by, the values of the X component. A functional dependency $X \rightarrow Y$ is a **full functional dependency** if removal of any attribute A from X means that the dependency does not hold anymore.

Example 2.2 Consider the relation schema **EMP_PROJ** below. The primary key is **ssn, proj_num**, where **ssn** is the employee's social security number and **proj_num** is the project number. An employee may work on more than one project. The attribute **hours** is the time that an employee spends on a project and is functionally dependent on **ssn, proj_num**. The attribute **ename** is the employee's name and is functionally dependent on **ssn**. The attributes **pname** (project name) and **plocation** (project location) are functionally dependent on **proj_num**.



If more than one employee works on a project, then the same **pname** and **plocation** must be repeated for that project. There is obviously redundancy in this design, and this redundancy can lead to update anomalies.

The relation schema **EMP_PROJ**, which is not in 2NF, can be decomposed into three relation schemas **EP1, EP2** and **EP3** such that the resulting relation schemas are in 2NF.



The relation schema **EMP_PROJ** could be modelled as one class in an OODB as shown below.

EMP_PROJ
  ssn : integer;
  ename : string;
  proj_num : integer;
  hours : integer;
  pname : string;
  plocation : string;

Obviously the same redundancies exist that existed with the relational schema **EMP_PROJ** and therefore the same update anomalies are possible. Following a naive design methodology, the object-oriented database designer would most likely come up with two classes, **EMPLOYEE** and **PROJECT** as shown below. The value of the multi-valued attribute

project in class **EMPLOYEE** is a set of tuples. The first attribute in this tuple is a pointer to an object of class **PROJECT**.

EMPLOYEE
  ssn : integer;
  ename : string;
  project : {(p:PROJECT, hours:integer)};

PROJECT
  proj_num : integer;
  pname : string;
  plocation : string;
  employee : {(e:EMPLOYEE, hours:integer)};

This schema contains redundancy, since **hours** is stored twice. We suggest that the following is a better design, in which there are three classes, **EMPLOYEE, PROJECT** and **EMP_PROJ** because there is no redundancy. Notice the similarity between this schema and that with the three relations **EP1, EP2** and **EP3** above. This schema illustrates how many-to-many relationships can be modelled in object-oriented database design. There is a many-to-many relationship between **EMPLOYEE** and **PROJECT**. As with other many-to-many relationships in object-oriented databases, it is useful to introduce a class that represents the relationship between the classes that model the two real world entities.

EMPLOYEE
  ssn : integer;
  ename : string;

PROJECT
  proj_num : integer;
  pname : string;
  plocation : string;

EMP_PROJ
  employee : EMPLOYEE;
  project : PROJECT;
  hours : integer;

This design may lead to slower performance than a design in which there is redundancy. We would argue that redundancy can then be included in the schema again but in a controlled way. Controlled redundancy is also introduced in relational database designs for performance reasons. There are two mechanisms provided in object-oriented database management systems (OODBMSs) that allow this redundancy to be handled in a controlled way. Often object-oriented database management systems (OODBMSs) provide an inverse relationship which automatically updates an attribute if the other side of the attribute is updated. The following schema illustrates controlled redundancy if the inverse relationship is maintained by the OODBMS.

EMPLOYEE
  ssn : integer;
  ename : string;
  projects : {ep : EMP_PROJ};

```
PROJECT
  proj_num : integer;
  pname : string;
  plocation : string;
  employees : {ep : EMP_PROJ};


EMP_PROJ
  employee : EMPLOYEE;
  project : PROJECT;
  hours : integer;
```

Another feature that OODBMSs have that allow you to control redundancy is methods. Consider the example schema above in which **hours** is stored twice. This redundancy could be avoided if one of the attributes is a derived attribute. Consider storing **hours** in one class, say **EMPLOYEE** and having a method in the other class **PROJECT** that derives the value for **hours** from the value stored in class **EMPLOYEE**. So, when the value for **hours** is updated in **EMPLOYEE** then the view of it from **PROJECT** is automatically updated.

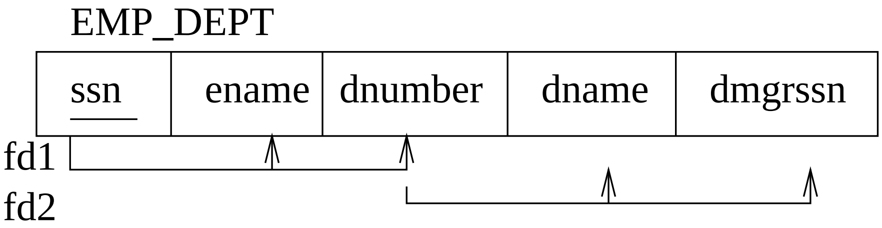
```
EMPLOYEE
  ssn : integer;
  ename : string;
  projects : {(p:PROJECT, hours:integer)};


PROJECT
  proj_num : integer;
  pname : string;
  plocation : string;
  employees : {e:EMPLOYEE},
  int method hours(e:EMPLOYEE)
  = select pair.hours
    from e.projects as pair
    where pairs.p = this
```

2.3 3NF

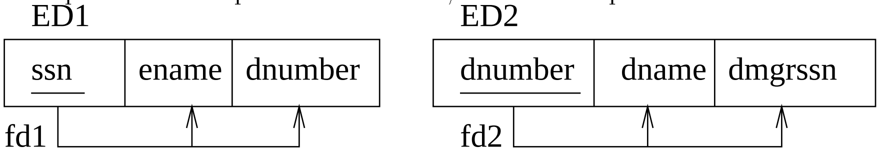
A relational schema R is in 3NF, if it is in 2NF and no non-prime attribute of R is transitively dependent on the primary key.

A functional dependency $X \rightarrow Y$ is a **transitive dependency** if there is a set of attributes Z and both $X \rightarrow Z$ and $Z \rightarrow Y$ hold.

Example 2.3 Consider the relation schema **EMP_DEPT**. The attribute **ssn** is the social security number of an employee, *ename* is the employee's name, **dnumber** is the identifier of the department in which they work, **dname** is the name of the department and **dmgrssn** is the social security number of the manager of that department. It is not in third normal form because **dname** and **dmgrssn** are transitively dependent on **ssn**. Notice that if many employees work in the same department, and we wish to update that department's name, then we'll have to update it in more than one place.



The relation schema **EMP_DEPT** can be decomposed to **ED1** and **ED2**. The resulting relation schemas are in 3NF. Notice now that if many employees work in the same department, and we wish to update that department's name, then we'll update the name in one place only.



One possible object-oriented design for this is one class called **EMP_DEPT** as shown below.

EMP_DEPT
  ssn : integer;
  ename : string;
  dnumber : integer;
  dname : string;
  dmgrssn : integer;

Such a design allows update anomalies to occur, similar to those that could occur in the above relation schema that is not in 3NF. A better design, in which there is no redundancy, is to decompose the class **EMP_DEPT** to two classes **EMPLOYEE** and **DEPARTMENT**.

EMPLOYEE
  ssn : integer;
  ename : string;
  dept : DEPARTMENT;

DEPARTMENT
  dnumber : integer;
  dname : string;
  dmgrssn : integer;

3 Other object-oriented features

In this section we look at an object-oriented feature that cannot be directly modelled in the relational model, namely aggregation.

The aggregate relationship is sometimes called the **has_a** relationship. For example, a shop **has** many products, like milk, bread and butter. The milk, bread and butter in turn have their own attributes. It would be recognised that the attributes of milk would be transitively dependent on the shop key. Because of this transitive dependency, we believe there is a link between aggregation and 3NF.

The aggregation relationship describes an aggregation hierarchy. Often, it is not obvious which way the aggregation relationship should be. Consider the EMPLOYEE and DEPARTMENT classes in section 2. Another design is shown below.

EMPLOYEE
  ssn : integer;
  ename : string;

DEPARTMENT
  dnumber : integer;
  dname : string;
  dmgrssn : integer;
  employee: {EMPLOYEE};

We believe that this design is better because there is no redundancy in this schema and it is more representative of what actually happens in the real world. Unfortunately, functional dependencies only guide us to possible redundancies and can provide no information of which way around such aggregate relationships should be modelled.

4 Conclusions and future work

This paper has shown that redundancies can arise in object-oriented database schemas and that the heuristics provided by object-oriented design and analysis methodologies do not provide sufficient guidance to measure what redundancies do exist in a particular object-oriented database design. We have also shown that normalization is a possible starting point for such a measure. In section 2, we showed that the constraint enforced by 1NF is not applicable in object-oriented database design. However, the constraints enforced in 2NF and 3NF (without the 1NF constraint) are useful in the design of object-oriented database systems, and further that the normal forms could be used as a starting point as a means of measuring the appropriateness of a design. We acknowledge that normalization alone does not provide the answer. Furthermore, we found there is no natural way to handle many-to-many relationships in object-oriented database design. There are many features in object-oriented databases that are not present in relational databases that are not covered by normalization. For example, it would be interesting to investigate if the concepts of functional dependencies and normalization could be extended to evaluate different object-oriented database designs that involved inheritance.

We now raise two other issues that require close consideration. The first issue is that objects do not necessarily have primary keys. We do not believe that this is an insurmountable problem but it must be taken into consideration when applying normalization to object-oriented schemas.

The other issue is methods. We have already shown how methods can provide controlled redundancy. Do they need to be treated differently in a normalization-like process?

So far, we have considered only 1NF, 2NF and 3NF. We have shown the constraints of 2NF and 3NF can be used to evaluate alternative object-oriented designs. It now opens the question of how useful other normal forms are in this setting.

Acknowledgements

I would like to thank the SERG group for their input, especially Bryan Genet, Peter Andreae and Lindsay Groves. Thanks also to Rodney Topor for his helpful comments from afar.

References

[1] G. Booch. *Object-oriented design with applications.* The Benjamin/Cummings Publishing Company, Inc., 1992.

[2] R. Elmasri and S. Navathe. *Fundamentals of Database Systems.* The Benjamin/Cummings Publishing Company, Inc., 1994.

[3] T. W. Ling and P. K. Teo. A normal form object-oriented entity relationship diagram. In *Proceedings of the 13th International Conference on the Entity-Relationship Approach*, pages 241–258, Manchester, UK, 1994.

[4] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen. *Object-oriented modeling and design.* Prentice Hall, Englewood Cliffs, NJ, 1992.

[5] G. Saake, S. Conrad, I. Schmitt, and C. Türker. Object-oriented database design: What is the difference with relational database design? In *ObjectWorld Frankfurt'95 - Conference Notes, Software Development Track*, page Software Development Track SD.6, Frankfurt, Germany, 1995.

[6] G. Weddell. A theory of functional dependencies for object-oriented data models. In *Proceedings of the First International Conference on Deductive and Object-Oriented Databases*, Kyoto, Japan, 1989.