2002

Migration from relational DBMS to object-relational DBMS

Irina Ispravnikova

University of Wollongong

UNIVERSITY OF WOLLONGONG

COPYRIGHT WARNING

You may print or download ONE copy of this document for the purpose of your own research or study. The University does not authorise you to copy, communicate or otherwise make available electronically to any other person any copyright material contained on this site. You are reminded of the following:

This work is copyright. Apart from any use permitted under the Copyright Act 1968, no part of this work may be reproduced by any process, nor may any other exclusive right be exercised, without the permission of the author.

Copyright owners are entitled to take legal action against persons who infringe their copyright. A reproduction of material that is protected by copyright may be a copyright infringement. A court may impose penalties and award damages in relation to offences and infringements relating to copyright material. Higher penalties may apply, and higher damages may be awarded, for offences and infringements involving the conversion of material into digital or electronic form.

Unless otherwise indicated, the views expressed in this thesis are those of the author and do not necessarily represent the views of the University of Wollongong.

Recommended Citation


Research Online is the open access institutional repository for the University of Wollongong. For further information contact the UOW Library: research-pubs@uow.edu.au
MIGRATION FROM RELATIONAL DBMS TO OBJECT-RELATIONAL DBMS

A thesis submitted in fulfillment of the requirements for award of the degree

HONOURS MASTERS of SCIENCE (COMPUTER SCIENCE)

from

UNIVERSITY OF WOLLONGONG

by

Irina Ispravnikova

MSc, University of Wollongong
BEng (Mch), TUOG (Tuymen, Russia)

Department of Information Technology and Computer Science

2002
Abstract

In the last few years, many companies have begun using their database systems for non-traditional applications because of demands such as storing images and multimedia objects in the database. Consequently, the objects and related operations are becoming more complex. A new technology has evolved in which relational and object-oriented concepts have been combined or merged - Object-Relational database systems.

With the increasing popularity of Object-Relational technology, it becomes necessary to have a methodology that allows database designers to migrate from existing Relational database technology to Object-Relational database technology. The objective of this thesis is to develop such transformation methodology. There are a few tasks in the methodology, which include transformation of database structures, from relational database into a conceptual (UML) model and then to an object-relational database; transformation of database applications, relational SQL statements into O-R SQL statements; and data migration, from relational tables into object-relational tables with objects. Transformation rules are developed for each task.
Performance comparisons between queries run on relational databases and queries run on object-relational databases are also carried out. Results of the comparisons establish the base for proceeding with the migration methodology.
ACKNOWLEDGMENT

I would like to extend my sincere thanks to my supervisor Dr Janusz Getta without whose invaluable assistance this thesis would not have been possible.

Sincere thanks are extended to my parents for providing the perfect environment in which to grow and learn; for providing and nurturing me with love for education.

Last and not list I am grateful to my husband and son for their patience and support throughout this work.
Table of Contents

Abstract ............................................................................................................................ iii
List of Figures ................................................................................................................. ix
List of Tables ................................................................................................................... xi
1. Introduction .................................................................................................................. 1
   1.1. Historical review ................................................................................................. 1
   1.2. The problem ......................................................................................................... 4
   1.3. Strategy of solution .............................................................................................. 5
   1.4. Outline of the Thesis ........................................................................................... 6
   1.5. Contribution ......................................................................................................... 9
2. Object-Relational Databases .................................................................................... 10
   2.1. Object Data Model ........................................................................................... 13
   2.2. Advantages of Object-Relational DBMS ......................................................... 16
   2.3. Disadvantages of Object-Relational DBMS ..................................................... 17
   2.4. Query language for Object-Relational Databases ............................................. 17
       2.4.1. Data Management Language (DML) .................................................... 19
   2.5. Existing Object-Relational Database Management Systems ......................... 20
       2.5.1. Oracle9 .................................................................................................... 20
       2.5.2. PostgreSQL Object-Relational Database ............................................. 23
       2.5.3. Informix’ Dynamic Server .................................................................... 25
       2.5.4. Predator ................................................................................................... 26
   2.6. Existing research work in Object-Relational Databases .................................... 27
3. Transformation of Database Structure ................................................................. 30
   3.1. Transformation of Database structures ......................................................... 30
   3.2. Transformation of Relational Schema into Conceptual schema .................... 30
       3.2.1. Transformation rules ........................................................................ 31
       3.2.2. Applying transformation rules to transform a relational database into a conceptual (UML) models .......................................................... 43
   3.3. Transformation of Conceptual schema into an Object-Relational schema ....... 46
       3.3.1. Application access .............................................................................. 47
       3.3.2. Transformation rules ........................................................................ 52
       3.3.3. Applying transformation rules to transform the conceptual (UML) model into an object-relational database ......................................................... 60
   3.4. Transformation of Relational schema into Object-Relational schema ......... 63
4. Correctness of transformation rules .................................................................... 70
   4.1. Statements that change database state .......................................................... 70
   4.2. Statements that change state of variables .................................................... 71
   4.3. Correctness of transformation rules ............................................................. 71
   4.4. Correctness of transformation rules for data migration ............................... 73
5. Translation of relational SQL statements into object-relational SQL statements 75
   5.1. Data Definition Statements ........................................................................ 75
       5.1.1. CREATE INDEX statements .............................................................. 75
       5.1.2. CREATE VIEW .............................................................................. 76
       5.1.3. CREATE TABLE statement ............................................................. 78
   5.2. Data Manipulation Statements ..................................................................... 85
       5.2.1. Transformation rules ........................................................................ 85
   5.3. Data Retrieval Statements .......................................................................... 101
       5.3.1. Transformation rules ........................................................................ 101
List of Figures

Chapter 2 Object-Relational Database

Figure 2.1 A matrix for classifying DBMS applications ......................... 12
Figure 2.2 Object-relational tables ......................................................... 14
Figure 2.3 Nested tables ....................................................................... 14
Figure 2.4 Single reference and double reference associations ............... 14
Figure 2.5 Pointer reference ................................................................. 15

Chapter 3 Transformation of Database Structures

Figure 3.1 Conceptual (UML) model.................................................... 46
Figure 3.2 References .......................................................................... 48
Figure 3.3 Object-relational tables ....................................................... 49
Figure 3.4 Frequency of access ............................................................. 51
Figure 3.5 Conceptual (UML) model, example .................................... 61
Figure 3.6 Object-relational database schema ..................................... 63

Chapter 4 Correctness of transformation rules

Figure 4.1 Translation of statements .................................................... 72
Figure 4.2 Data migration .................................................................... 74

Chapter 5 Transformation of relational SQL statements to object-relational SQL statements

Figure 5.1 Relational tables Applicant and Skills .................................. 107
Figure 5.2 Object-relational nested table Applicant ............................... 107
Figure 5.3 Relational table Applicant and Position ............................... 113
Figure 5.4 Object-relational tables Applicant and Position ................... 114
Figure 5.5  Relational table Applicant and Position .................................. 118

Figure 5.6  Object-relational tables Applicant and Position featuring
double reference association ...................................................... 118

Figure 5.7  Replacement of SQL statements with O-R SQL statements .... 123

Chapter 6  Data migration

Figure 6.1  Data migration .......................................................................... 124

Chapter 7  Conclusion

Figure 7.1  Performance comparison, relational v object-relational......... 129

Appendix A

Figure A.1  Performance comparison in object-relational tables .......... 140

Figure A.2  Performance comparison in object-relational v
relational tables ............................................................................ 142

Figure A.3  Performance comparison of nested tables v relational tables ... 146

Appendix C

Figure C.1  Classes ......................................................................................... 162

Figure C.2  An association ........................................................................... 163

Figure C.3  Notation for multiplicity ............................................................. 163

Figure C.4  A link attribute is a property of the association ................. 164

Figure C.5  Generalization ........................................................................ 164

Figure C.6  Attribute multiplicity ................................................................. 165

Figure C.7  Candidate key for a class .......................................................... 165

Figure C.8  A ternary association ................................................................. 166
List of Tables

Chapter 3 Transformation of Database Structures

Table 3.1 Worthiness of single reference and double reference associations ............................................................................................................................ 52

Chapter 5 Translation of relational SQL statements into object-relational SQL statements

Table 5.1 CREATE TABLE statements for relational and object-relational (nested) tables........................................................................................................................................ 80

Table 5.2 CREATE TABLE statements for relational and object-relational with single reference association tables .................................................................................. 82

Table 5.3 CREATE TABLE statements for relational and object-relational with double reference association tables.......................................................................................... 85

Table 5.4 INSERT statements for relational and object-relational (nested) tables ............................................................................................................................................. 87

Table 5.5 INSERT statements for relational and object-relational with single reference association tables ........................................................................................................ 88

Table 5.6 INSERT statements for relational and object-relational with double reference association tables ........................................................................................................ 90

Table 5.7 UPDATE statements for relational and object-relational (nested) tables ............................................................................................................................................. 92

Table 5.8 UPDATE statements for relational and object-relational with single reference association tables ........................................................................................................ 94
Table 5.9 UPDATE statements for relational and object-relational with
double reference association tables ............................................................ 96
Table 5.10 DELETE statements for relational and object-relational (nested)
tables .................................................................................................................... 98
Table 5.11 DELETE statements for relational and object-relational with
single reference association tables ................................................................. 99
Table 5.12 DELETE statements for relational and object-relational with
double reference association tables ................................................................. 101
Table 5.13 SELECT statements for relational and object-relational (nested)
tables .................................................................................................................. 103
Table 5.14 SELECT statements for relational and object-relational with
single reference association tables ................................................................. 104
Table 5.15 SELECT statements for relational and object-relational with
double reference association tables ................................................................. 106
Table 5.16 Example of Insert statements in nested tables ....................... 109
Table 5.17 Example of Update statement in nested tables ..................... 110
Table 5.18 Example of Delete statement in nested tables ..................... 111
Table 5.19 Example of Select statements in nested tables ..................... 112
Table 5.20 Example of Insert statements in unidirectional associations .... 114
Table 5.21 Example of Update statements in unidirectional associations .. 115
Table 5.22 Example of Delete statements in unidirectional associations ... 116
Table 5.23 Example of Select statements in unidirectional associations.... 117
Table 5.24 Example of Insert statements in bi-directional associations ..... 119
Table 5.25 Example of Update statements in bi-directional associations... 120
Table 5.26  Example of Delete statements in bi-directional associations.... 121

Table 5.27  Example of Select statements in bi-directional associations .... 122

Appendix A

Table A.1  Queries results...............................................................................136
Table A.2  Queries results...............................................................................138
Table A.3  Queries results...............................................................................142
Chapter 1
Introduction

1.1 HISTORICAL REVIEW

The relational data model was first introduced by Codd in 1970[7][14]. During the mid 1970’s and until the mid 1980’s relational database theory dominated the output of the database research resulting in a strong mathematical foundation for the relational data model.

Relational Database represents data as a collection of rows and columns in a two-dimensional table and can only represent data in binary form. Relational Systems have supported only certain simple, predefined datatypes and operators. Values in a relational table’s foreign key column can refer only to rows in one other table due to cardinality constraints. Relational Database only caters for one kind of relationship: an association between two tables using primary key/foreign key (PK/FK). These restrictions have limited the usefulness of Relational Systems for complex applications.

During the late 1980’s and 1990’s shortcomings of the relational data model in areas such as scientific and statistical applications, expert systems, text handling, multimedia, office automation, manipulating temporal and spiral data, and
computer-aided design, gave rise to several new proposals and extensions to the relational data model. The main extensions to the relational data model include complex object data models such as the nested relational model [14].

Nested relational data models based on 'ignoring' the First Normal Form (1NF) – the assumption that keeps the tabular structure of the relations simple and allows relational query language to be able to refer to attribute values in tuples of relations in a straightforward manner [14]. The nested relational model did not use record identity and links; instead it used a nested structure of a hierarchical model in conjunction with a concept of primary-foreign key.

Another step in database development has been a functional data model that uses a data-access language based on mathematical function notation, with declarative functional query language, analogous to relational query language, to define the function values [4]. The basis being objects and functions. Functions map objects onto other objects and data values. Relationships between objects, attributes of objects, and procedures associated with objects are all represented by functions.

Further work in database development was concentrated in Object-Oriented Databases. The appearance of Object-Oriented Databases was as significant as an appearance of relational databases. It is viewed as an alternative to the Relational Databases. Standardization work for object databases began in 1991 [5]. Object-Oriented Databases supported by object-oriented languages, including
encapsulation, inheritance, and polymorphism. Object-Oriented Systems can store complex user-defined objects, and can capture the behavior as well as the state of these objects. Object ID (OID) used for accessing data in objects that is faster and more efficient.

In the last few years, many companies have begun using their database systems for non-traditional applications because of demands such as storing images and multimedia objects in the database. Consequently, the objects and related operations are becoming more complex. Some examples of complex data are images, geographical information systems, multimedia objects, and spatial, 3-D, and temporal data. A new technology has evolved in which relational and object-oriented concepts have been combined or merged. These systems are called Object-Relational database systems.

Object-Relational Database (ORDB) technology had been first commercially implemented in 1997[16]. It inherited the robust transaction- and performance-management features of its Relational Database model and the flexibility of the Object-Oriented model. The term object-relational database is used to describe a database that has evolved from the relational model into a hybrid database that contains both relational technology and object technology.

These are just a few of the advantages that the Object-Relational model can be associated with: (1) Enhance a system's overall performance. (2) Improve the
flexibility of the overall system. (3) Provide maintainability. (4) Increase functionality, allow users to define data types, functions and operators. (5) Support extensible data types. (6) Adopt query-centric approach to data management.

1.2 THE PROBLEM

The simplicity of the Relational Database model is a benefit in ease of use and mathematical tractability, but it is also a limitation. The application's data are too complex to represent using relational tables and queries.

On the other hand Object-Oriented Database model structure is complex and there is no method existing to migrate Relational Databases into Object-Oriented Databases.

The need for Object-Relational Databases are:

- Modeling issues - most of the applications are written in Object-Oriented languages and most of databases that are in use are Relational Databases.

- Relational Database model does not provide satisfactory performance. (Performance comparison presented in details in Appendix A)

- Object-Oriented model is not the target, because there is a big conceptual gap existing between Relational and Object-Oriented Database models, and most of the data is in Relational Databases.
• No methodology exists to migrate data from Relational database into an Object-Oriented Database.

So far application developers have tended to choose a Relational or Object-Oriented Database system depending on which set of advantages is more important for a specific application. In the last few years, however, applications have begun to appear which have strong requirements for both data independence and high-level query language of a relational system and the complex user defined semantics of an object-oriented system.

When an existing Relational Database needs to be updated to accommodate object-oriented features, such as extended data types, there is no methodology in existence for migrating from Relational Database application to Object-Relational Database application.

1.3 STRATEGY OF SOLUTION

The main question that needs to be answered is how to transform Relational Database technology into Object-Relational Database technology. Development of a method for the migration is the aim of this research.

There are three steps required to achieve the goal:
1. Translate Relational Database structure into Object-Relational Database structure. This step covers the transformation from an existing relational database to a conceptual or UML model. Then from a conceptual or UML model into a Object-Relational Database structure.

2. Translate relational SQL statements into object-relational SQL statements. This step covers the translation of Data Manipulation, Data Definition and Data Retrieval statements.

3. Migration of the Database content. This step covers the migration of data from Relational database into Object-Relational database.

1.4 OUTLINE OF THE THESIS

This thesis is divided into three parts.

- Chapter 1 Introduction

This chapter provides a brief description of the Relational DBMS and Object-oriented DBMS, and states advantages and disadvantages of both. It also states the main problems and the strategy of the solution.

- Chapter 2 Overview of previous work
This chapter describes Object-Relational DBMS and provides a brief description of existing Object-Relational DBMS.

- **Chapter 3 Transformation of Database structures**

  This chapter presents a thorough discussion of how to transform Relational Database structure into Object-Relational Database structure. Each step towards the solution takes a separate sub-chapter. As well as theoretical solutions it shows an example for each step.

- **Chapter 4 Translation of relational SQL statements into object-relational SQL statements**

  This chapter presents a thorough discussion of how to translate relational SQL statements into Object-Relational SQL statements, including Data Manipulation statements, Data Definition statements and Data Retrieval statements (SELECT statement). Each task towards the solution is presented with examples and takes a separate sub-chapter.

- **Chapter 5 Data migration**

  This chapter describes steps required to perform to migrate data from Relational Database to Object-Relational Database. Each step is supported by example with thorough explanations.
Appendices

The Thesis contains three appendices, which show examples

Appendix A: Performance comparison

This appendix presents performance comparison from queries run on relational tables, object-relational tables and nested tables. The results serve as a base for the research.

Appendix B: Transformation of Relational Database into Object-Relational Database, an example

This appendix presents a full example of how to migrate from an existing Relational database into an Object-Relational database.

Appendix C: Modeling notations

This appendix presents notations used by UML models.

1.5 CONTRIBUTION

The thesis contains my work in developing methods and transformation rules for translating relational database structure to object-relational database structure, also
translation of relational SQL statements to O-R SQL statements and transformation of relational database data content to object-relational database.
Chapter 2
Object-Relational Databases

An Object-Relational Database Management System (ORDBMS) is a database that supports both relational and object-relational models in an integrated fashion. The user can define, manipulate, and query both relational data and objects while using common interfaces (such as SQL).

The main objective of Object-Relational DBMS is to achieve the benefits of both the relational and the object models such as scalability and support for extensible data types. ORDBMS employs a data model that incorporates Object-Oriented features into RDBMS. A database is still a collection of relational tables, but some of the tabular entries may have richer data structure, called abstract data types (ADTs) also known as user defined types (UDT) [8]. The ORDBMS has the relational model in it because the data is stored in the form of tables having rows and columns and SQL is used as the query language and the result of a query is also table or tuples (rows).

Hence the characteristics of ORDBMSs are:

- Base datatype extension,
- Support complex objects,
- Inheritance,
Object-Relational DBMS combines the features of RDBMS with the best characteristics of Object-Oriented Database Management.

ORDBMS allows users to define datatypes, functions and operators. As a result, the functionality of the ORDBMSs increases along with its performance.

- ORDBMS allow developers to embed new classes of data objects into the relational data model.

- ORDBMS schema has additional features not presented in RDBMS schema. Several object-oriented structural features such as inheritance and polymorphism are part of the OR data model.

- ORDBMS adopt the RDBMS query-centric approach to data management. All data access in an ORDBMS is handled with declarative SQL statements.

Dr. Michael Stonebraker [23] has classified the DBMS applications into four types: simple data without query, simple data with query, complex data without query, and complex data with query. These four types describe file systems, Relational DBMSs, Object-Oriented DBMSs, and object-Relational DBMS, respectively.
Also, Stonebraker [23] predicts that applications from Relational DBMS (simple data with query) will slowly move towards the Object-Relational DBMS (complex data with query).

The five architectural options given by Dr. Stonebraker [23], listed in ascending order of practicality, performance, and general desirability are:

- Supply plug-in code to make function calls to other applications.
- Add separate API's and server subsystems to support object functionality.
- Simulate specialized object-relational functionality in a middleware layer.
• Completely redesign the database engine.

• Add a new object-oriented layer to support rich datatypes atop a proven relational database engine.

With advantages like large storage capacity, access speed, and the manipulation power of object databases, ORDBMS are set to conquer the database market.

INFORMIX Dynamic Server, developed by Informix Corporation, belongs to the fourth category. The other current ORDBMSs include ORACLE9, from Oracle Corporation, and POSTGRESQL Object-Relational Database by The PostgreSQL Global Development Group.

2.1 OBJECT-RELATIONAL DATA MODEL

Object-relational schema is a binary graph \((N, E)\) where \(N\) is a set of nodes and \(E\) is a set of edges. A set of edges \(E\) consists of single-arrow edges and double arrow edges. Single-arrow edges represent single reference association and double-arrow edges represent double reference association.

A schema of object-relational table in graphical representation will be denoted by rounded rectangle with the table name inside the rectangle. Optional, attributes follow the name of object-relational schema after a semicolon, Figure 2.2. The table name is underlined by a single line.
Schema of nested table in graphical representations will be denoted as an attribute to the object-relational schema. The name of the nested table is underlined with a double line and the attributes of the nested table follow the name after semicolon, Figure 2.3.

Links implement associations or relationships from a conceptual schema. An association connects related object-relational tables and is denoted by single or double arrows, Figure 2.4. Single arrows represent single reference association and double arrows represent double reference association.

Direction of references, represented by arrows, shows the way object-relational tables are connected.

Figure 2.4 Single reference and double reference associations
In Oracle9 REF is a logical ‘pointer’ to a row object REFs and a collection of REFs provides associations among objects illuminating the need for foreign keys [1]. REFs allow easy navigation between objects. Pointer references are denoted by @.

Figure 2.5

Applicant: A#, Name, @Position

Figure 2.5 Pointer references

In ORDBMS, objects are represented using the user-defined type (UDT) and user-defined function (UDF) features. The only way to address or manipulate an object is through the set of methods making up the object’s interface. The only way to work with ORDBMS data is by applying UDT to type instances in SQL queries.

In RDBMS, to group data into sets and arrays is forbidden by the First Normal Form rule (1NF). This forced developers into awkward data models and complex queries. There is no such limitation in existence in ORDBMS, where elements or attributes can be sets or arrays of other object instances. Because the ORDBMS provides support for extensible data types, there is the possibility to manage other kinds of collections, such as stacks, frames, queues, timeseries data, etc.

Relational primary and foreign key constraints are limited to table relationships, rather than relationship between objects. In addition, values in a relational table’s
foreign key column can refer only to rows in one other table. Object identity permits references between multiple tables. It is more flexible than foreign key constraints, and a small number of straightforward extensions to SQL allows developers to 'chase the pointers'.

Inheritance is used for representing situations in which one kind of data is related to another kind of data in a particular way. Traditionally, the relational model only catered to one kind of relationship: an association between two tables that is defined using a primary key / foreign key reference. Not all relationships can be represented this way. Also, the extensible type system can permit the introduction of a huge number of new objects into the database. To make data more manageable is to reuse the implementation of one data type in the implementation of another, which is called inheritance.

2.2 ADVANTAGES OF OBJECT-RELATIONAL DATABASES

The following advantages are associated with Object-Relational DBMS.

- To enhance a system's overall performance. ORDBMS can achieve a higher level of data throughput or better response time than is possible using RDBMS technology. This effect is particularly pronounced for data-intensive applications such as decision support systems and in situations in which the objects in the application are large, such as digital signal or time series data.
• Improve the flexibility of the overall system. Multiple unrelated object
definitions can be combined within a single ORDBMS database. At runtime,
they can be put together within a query expression created to answer some high-
level question. Such flexibility is very important because it reduces the costs
associated with information system development and ongoing maintenance.

• Provide maintainability. Over time, as new functionality is added to the
application and as the programming staff changes, the system’s catalogue can be
used to determine the extent of the current system’s functionality and how it all
fits together.

2.3 DISADVANTAGES OF OBJECT-RELATIONAL DATABASES

In spite of many advantages, ORDBMS does also have a drawback.

• Complexity

• ‘Unsafe deletion’. The Object-Relational Database model does not have any
cardinality constraints that will prevent it from deleting a tuple (row) from the
object.

2.4 QUERY LANGUAGE FOR OBJECT-RELATIONAL DATABASES
To manipulate data stored in an Object-Relational DBMS an extended version of the SQL language, OR-SQL, is used. OR-SQL expressions describe what it is that is wanted (nonprocedural), rather than a procedural, step-by-step algorithm defining how the task is to be accomplished, so it is still the same way of operating on a database as in traditional SQL.

OR-SQL expresses operations over relational schema objects such as tables and views. Queries make no reference to physical data management details, such as pointers, or to memory management. The actual operational schedule depends on factors such as the physical configuration of the database, the presence of indices on certain columns, the number of rows the ORDBMS needs to manage at various stages of the query processing, and so on.

OR-SQL used both to create the database structure, and to write queries, which are statement that retrieve or update the data stored in the database. SQL has its roots in SEQUEL (Structured English QUEry Language), a language created as part of the IBM System Relational DBMS prototype during the late 1970. The SQL language has undergone several revisions, with each iteration adding new functionality. The latest revision, called SQL-1999 or SQL3 is the foundation for Object-Relational database management systems [10].

OR-SQL is divided into two sub-parts: Data Definition Language (DDL) and Data Manipulation Language (DML).
• DDL is used to define or modify the structure of schema features such as tables, views, indices, functions, procedures, data types, and so on. The principle DDL language statements are CREATE, ALTER and DROP.

• DML is used to read and modify data values accessed through the database. The four main DML operations are SELECT, INSERT, UPDATE, and DELETE.

2.4.1 DATA MANAGEMENT LANGUAGE (DML)

Below are Data Management Language operations in details.

• An INSERT statement can add a row by addressing the table directly. In its simplest form the INSERT statement must provide a target table name and a list of constant data values of the appropriate type or values computed by user-defined functions for each of its columns. More sophisticated INSERT statements can append the row values from a SELECT query to the target table.

• An UPDATE statement modifies data values in a table. Each UPDATE needs to identify the table whose rows are being changed, how the affected columns are to be changed, and it uses a WHERE clause to identify the row(s) to be modified. The change the UPDATE statement makes, which columns are affected and the new values they are to have, is specified in the SET clause. Each UPDATE can change a single row, a set of rows, a single column, a set of columns, or a set of columns in a set of rows.
• The DELETE statement removes records from a database table and reclaims the row's storage space for new data.

• A SELECT query's result can be thought of as a transient table, which is a set of identically structured rows. Result rows only exist for as long as the query is active.

2.5 EXISTING OBJECT-RELATIONAL DATABASE MANAGEMENT SYSTEMS

2.5.1 ORACLE9

The ORACLE9 universal data server natively supports an object-relational data model and lays the foundation for an extensible framework for supporting domain-specific data and operations within the database. ORACLE9 is the first industrial-strength object-relational server in the database marketplace.

Design goals of the ORACLE9 universal data server are [16] to:

• Provide users with the ability to model their business objects in the database by enhancing the type system to provide support for user-defined types. These types are meant to closely model application objects and are treated as built-in types, such as number and character, by the database server.
• Provide an infrastructure to facilitate object-based access to object data stored in an Oracle database and minimize the potential mismatch between the data model used in an application and the data model supported by a database.

• Provide built-in support for new data types needed in multi-media, financial and spatial applications.

• Provide a framework for database extensibility so that new multimedia data types can be supported and managed natively in the database. This framework provides the infrastructure needed to allow extensions of the data server by third parties via data

A major benefit of object orientation is inheritance, i.e., the ability to extend the specification and behavior of an object. ORACLE9’s type system supports inheritance on types. A type can be extended by creating subtypes which would reuse the specification and implementation (attributes and methods) of the type it derives from (parent). In a subtype, the ability to enhance the structure of the data as inherited from its supertype by defining other attributes, is referred to as data inheritance. The ability to add other methods in addition to ones inherited from its supertype is referred to as method inheritance.

ORACLE9’s type system allows relationships between two entities to be modeled natively in the database via the concept of a reference. A reference is specified in
the source object type with the keyword REF and the target object type name. A REF is a generalization of the relational foreign key concept.

ORACLE9's type system supports varying length arrays and nested tables (multisets of rows) as collection types. Attributes of types and columns of tables can be of a collection type. By using varying arrays and nested tables, application developers can model one-to-many and many-to-many relationships natively in their database schema.

ORACLE9 provides full integration of relational and object-based applications and schemas. Any kind of object columns may be added to existing relational tables. Also, the view mechanism is extended to support new object-oriented applications and tools to provide access to data stored in relational and object tables in a uniform manner. Conversely, new and existing relational applications and tools can access data in object and relational tables uniformly.

ORACLE9 allows methods to be implemented in PL/SQL, C/C++ or Java. Decoupling of the specification of a method in SQL from its implementation provides a uniform way to invoke methods on object types, even though these object types can be implemented in various programming languages.

ORACLE9 object-relational database introduces innovative technology for supporting new data types and a new object-oriented data model. It provides an integrated and scalable solution for storage, manipulation, and management of text,
spatial, image, and time series data. The support provided for Java in the server and in the client interfaces, for management of large binary and character data and the integration of object and relational data, are truly unique in the database industry. By providing comprehensive support for storing, accessing, and managing objects natively in both the client and server tiers, ORACLE8 truly brings the object paradigm to the mainstream of corporate computing environments.

2.5.2 POSTGRESQL OBJECT-RELATIONAL DATABASE

PostgreSQL is an object-relational database management system (ORDBMS) based on POSTGRES, Version 4.2, developed at the University of California at Berkeley Computer Science Department [19]. The POSTGRES project, led by Professor Michael Stonebraker.

PostgreSQL is an open-source descendant of this original Berkeley code. It provides SQL92/SQL99 language support and other modern features.

PostgreSQL offers substantial additional power by incorporating the following additional concepts in such a way that users can easily extend the system:

• inheritance
• data types
• functions
Other features provide additional power and flexibility:

- constraints
- triggers
- rules
- transactional integrity

These features put PostgreSQL into the category of databases referred to as object-relational.

Major enhancements in PostgreSQL include:

- Table-level locking has been replaced by multiversion concurrency control, which allows readers to continue reading consistent data during writer activity and enables hot backups from pg_dump while the database stays available for queries.

- Important backend features, including subselects, defaults, constraints, and triggers, have been implemented.

- Additional SQL92-compliant language features have been added, including primary keys, quoted identifiers, literal string type coercion, type casting, and binary and hexadecimal integer input.
• Built-in types have been improved, including new wide-range date/time types and additional geo-metric type support.

• Overall backend code speed has been increased by approximately 20-40%, and backend start-up time has decreased by 80%.

2.5.3 INFORMIX DYNAMIC SERVER

Informix Dynamic Server (IDS) is an Object-Relational Database Management System implemented by the INFORMIX Corporation [3].

In addition to core functionality, which is similar for almost all Object-Relational Databases, users can have at their disposal a number of precoooked bundles of user-defined extensions, called DataBlade products. The list of DataBlades changes constantly. INFORMIX partners bring out new DataBlade products regularly.

DataBlades products are classified based on the kind of problem they are design to solve. In all cases, a DataBlade extends the set of data types that can be used in database tables and the set of expressions that can be used in SQL queries. Some DataBlade modules use the indexing facilities provided by the ORDBMS; others require specialist access methods. The text management DataBlade products all include an indexing access method of some kind, and the Timeseries datablade uses its own physical storage algorithms instead of the ones used for table data.
In addition to C, developers can use Java to implement user-defined extensions. Certain features of Java – its performance, safety, portability, and the fact it is an open language standard – make it an excellent choice for this kind of work. Once embedded within the IDS product, Java methods can implement user-defined functions used in SQL query statements, or the engine can invoke a self-contained Java class. Surfacing its methods as SQL expressions, it is even possible to create an ORDBMS data types that corresponds to a Java class.

2.5.4 PREDATOR

PREDATOR is an Object-Relational Database created by the Computer Science Department of Cornell University [20]. Version 2.0 of its database code was released in March 2000. The goal has been to build a research and educational vehicle that can handle real-life database problems.

The system is built using C++, and makes use of inheritance and encapsulation. It uses the Shore storage manager (a database management library developed at Wisconsin University) as the underlying data repository.

PREDATOR is a multi-user client-server database system. PREDATOR implements many features found in commercial relational and object-relational database systems. The emphasis has been primarily on query processing, although transaction processing is also supported. The beta code release includes the ability to execute a large subset of SQL, multiple join algorithms, storage management over
large data volumes, indexing, a cost-based query optimizer, and a variety of object-relational features. Basic concurrency control and low-level storage management is handled using the Shore storage manager. A WWW-based graphical user interface allows any Java-enabled Web browser to act as a database client.

A major theme in PREDATOR is extensibility of the system -- adding the ability to process new kinds of data. The primary technical contribution in PREDATOR is the notion of Enhanced Abstract Data Types (E-ADTs). An E-ADT ‘enhances’ the notion of a database ADT by exposing the semantics of the data type to the database system. These semantics are primarily used for efficient query processing, although they serve other purposes too.

The goal of extensibility is pursued at many levels of the code, but mainly in the type, language and query processing systems. There are two basic components in the system: one is an extensible table of Enhanced Abstract Data Types (E-ADTs), which defines the kinds of data that the system can manipulate. The other component is an extensible table of query processing engines (QPEs); currently SQL is the only supported language.

2.6 EXISTING RESEARCH WORK IN OBJECT-RELATIONAL DATABASES

Research has been conducted not only in storage and data retrieval in object-relational databases suitable for financial and other types of applications, but in areas
such as medical and pharmaceutical application. Examples of such work can be found at BioImage database [15], B-SPID project [9] and integration of molecule data type [6].

The BioImage database is a new scientific database for multidimensional microscopic images of biological specimens, which is available through the World Wide Web (WWW) [15]. Pointers are used to reference information stored in other databases. The database management system is the Informix Dynamic Server with Universal Data Option. This object-relational system allows the handling of complex data using features such as collection types, inheritance, and user-defined data types. Informix datablades are used to provide additional functionality: the Web Integration Option enables WWW access to the database; the Video Foundation Blade provides functionality for video handling.

The B-SPID project concerns the processing of neuroimages and attached components stored in an object-relational multimedia database management system (DBMS) [9]. Advanced bioinformation concepts are exploited in this project such as large scale data storage, high level graphical user interfaces and 3D graphical processing and display of data. Queries on this database are designed to obtain and display from neuro-imaging data, several types of results (pictures, text, or 3D graphical shapes) on heterogeneous systems.
Ontogen Corporation and Daylight Chemical Information Systems have recently integrated a molecule data type in an object-relational database management system [6]. This innovation unites chemical and biological data management into a single universal database. All chemical information storage, searching and retrieval are performed using a standard structured query language. These can simplify and standardize the management of pharmaceutical data.

Existing research into object-relational databases has included establishing a method for designing a database in particular modeling relationships [22]; and designing a methodology for transforming inheritance relationships to relational tables [21].

However, there is no known work in the area related to transformation of Relational databases into Object-Relational databases. It makes this thesis very important in the subject of object-relational databases and laid groundwork for further research.
3.1 TRANSFORMATION OF DATABASE STRUCTURES

Transformation of database structures is performed in two stages.

- Stage one: transformation of relational schemas into conceptual schemas
- Stage two: transformation of conceptual schemas into object-relational schemas

It is necessary to perform the transformation in two stages because a conceptual model from an abstraction point of view is closer to an object-relational model than a relational model in the depth of application understanding.

3.2 TRANSFORMATION OF RELATIONAL SCHEMA INTO CONCEPTUAL SCHEMA

The process of transformation of Relational Database schema into conceptual model generally known as Database Reverse Engineering.
During database reverse engineering it is required to take an existing design or an implementation that embodies a design and extracts the essential application problem content. The database reverse engineering omits design optimizations and implementation decisions.

The process of a database reverse engineering a relational database depends on primary and foreign keys. The transformation rules for reverse engineering of relational schemas into a conceptual schema are described in the next section.

3.2.1 TRANSFORMATION RULES

A conceptual model adopts a unified modeling language (UML) [2,13]. Because the UML is so semantically deep, it’s possible to use it to visualize and specify the configuration of database system, such as found at the boundary of object model and relational model. This makes it possible to track the migration of a relational database model to an object model. In the presence of tools that support database round trip engineering, it is then possible to create a data model based on the database structures through forward engineering or to create a database based on the data model through reverse engineering. All the semantics relevant to the data tables, columns, constraints, indices and more – can be preserve through such transformations. The following rules apply when transforming relational database schema into object class in conceptual (UML) model.

Rule R.1
If there exists a relational schema then it can be transformed into a class.

**Input:** Relational table $R(P_1, F_1, a_1, \ldots a_n)$, where $P_1$ is a primary key and $F_1$ is a foreign key.

**Output:** UML interpretation of the class

$$
\begin{array}{|c|}
\hline
C_R \\
\hline
a_1 \\
\vdots \\
a_j \\
a_n \\
\hline
\end{array}
$$

Note: Foreign key $F_1$ is dropped from the class, because foreign keys serve the purpose of creating an association between classes. Primary key $P_1 = (a_i, \ldots, a_j)$ is a composite attribute.

**Rule R.2**

Add one-to-many or one-to-one association to the pair of classes.

**Input:** Schemas of relational tables $R(P_1, a_1, \ldots a_n)$ and $S(P_2, F, b_1, \ldots b_n)$, where $P_1$ and $P_2$ are primary keys of schemas R and S respectively, F is a foreign key of schema S that references the primary key $P_1$ of schema R.
Output: Add one-to-many association (when an attribute in a class that is on 'one' side of an association has a connection with one or many attributes in a class that is on 'many' side and each attribute in a class that is on 'many' side has an association with only one attribute in a class that is on 'one' side) or one-to-one (when one attribute in one class has an association with one and only one attribute in another class) association to the \( C_R \) and \( C_S \) classes in conceptual (UML) model.

One-to-many association

One-to-one association

EXAMPLE 3.1

The primary key in relational table is underlined with a solid line and the foreign key is marked by dashed underlining.

CUSTOMER (CUSTOMER_ID, NAME)
ORDER (ORDER_ID, ORDER_DATE, CUSTOMER_ID)
Rule R.3

Add many-to-many associations with link attributes to classes.

**Input:** Schemas of relational tables $R(P_1, a_1, \ldots a_n)$, $S(P_2, b_1, \ldots b_n)$ and $T(F_1, F_2, c_1, \ldots c_n)$ where $P_1$ and $P_2$ are primary keys of schemas $R$ and $S$ respectively, $F_1$ and $F_2$ are foreign keys of schema $T$ that references the primary keys $P_1$ of schema $R$ and $P_2$ of schema $S$.

**Output:** Add many-to-many association (when many attributes in one class have an association with many attributes in another class) to $C_R$, $C_S$ and $C_T$ classes in conceptual (UML) model. Relational tables that contains two foreign keys is transformed into a link attribute (which is a property of an association).

\[ \text{EXAMPLE 3.2} \]
Rule R.4

Add n-ary association to classes.

Input: Schemas of relational tables $R(P_1, a_1, \ldots a_n)$, $S(P_2, b_1, \ldots b_n)$, $V(P_3, c_1, \ldots c_n)$ and $T(F_1, F_2, F_3, d_1, \ldots d_n)$ where $P_1, P_2$ and $P_3$ are primary keys of schemas $R$, $S$ and $V$ respectively, $F_1, F_2$ and $F_3$ are foreign keys of schema $T$ that referencing primary keys $P_1$ of schema $R$, $P_2$ of schema $S$ and $P_3$ of schema $V$.

Output: Add n-ary association (when an attribute in one class that is on 'one' side of an association has a connection with multiple attributes in multiple classes, 'many' side) to $C_R$, $C_S$, $C_V$ and $C_T$ classes in a conceptual (UML) model. Relational table
that contains multiple number of foreign keys is transformed into a link attribute, which is a property of an association.

EXAMPLE 3.3

PATIENT (PATIENT_ID, PATIENT_NAME)
PHYSICIAN (PHYSICIAN_ID, PHYSICIAN_NAME)
TREATMENT (TREATMENT_CODE, DESCRIPTION)
PATIENT_TREATMENT (PATIENT_ID, PHYSICIAN_ID, TREATMENT_CODE, DATE, RESULT)
Rule R.5

Add one-to-many association to classes when relational table has foreign key (FK) but without it table cannot exist independently.

Input: Schemas of relational tables R(P₁, a₁, ...aₙ), S(P₂, F₁, b₁, ...bₙ) where P₁ and P₂ are primary keys of schemas R and S respectively, F₁ is foreign key of schema S that referencing primary key P₁ of schema R.

Output: Add one-to-many association to C_R and C_S classes in conceptual (UML) model.
EXAMPLE 3.4

**EMPLOYEE** (EMPLOYEE_ID, NAME, EMPLOYEE_TYPE, DATE_HIRED)
**DEPENDENT** (FIRST_NAME, LAST_NAME, EMPLOYEE_ID, DOB)

**Rule R.6**

*Add multivalued attribute to class.*

**Input:** Schemas of relational tables \( R(P_1, a_1, \ldots, a_n) \) and \( S(K_1, b_1, \ldots, b_n) \) where \( P_1 \) and \( K_1 \) are primary keys of schemas \( R \) and \( S \) respectively and \( K_1 \) is the foreign key of schema \( S \) that references the primary key \( P_1 \) of schema \( R \). Attributes \( (b_1, \ldots, b_n) \) can be viewed as attributes to relational table schema \( R \).

**Output:** Add multivalued attributes to class \( C_R \) in conceptual (UML) model.
EXAMPLE 3.5

\[
\begin{array}{|c|}
\hline
C_R \\
\hline
a_1 \\
\hline
\ldots \\
\hline
a_m \\
\hline
a_n \\
\hline
b_{[1\ldots n]} \\
\hline
\end{array}
\]

**Rule R.7**

Add unary one-to-many association to class.

**Input:** Schemas of relational tables \(R(P_1, F_1, a_1, \ldots a_n)\) where \(P_1\) is a primary key of schema \(R\) and \(F_1\) is foreign key of schema \(R\) that references the primary key \(P_1\) of schema \(R\).
**Output:** Add unary one-to-many association (it is a one-to-many association between the instances of the same relational table) to $C_R$ class in conceptual (UML) model.

![Diagram of the CR class with attributes a_1, ..., a_n]

**EXAMPLE 3.6**

EMPLOYEE (EMPLOYEE_ID, NAME, DOB, MANAGER_ID)

![Diagram of Employee and Managers with attributes Employee_ID (CK1), Name, DOB and Managers]

**Rule R.8**

Add unary many-to-many association to class.
**Input**: Schemas of relational tables $R(P_1, a_1, \ldots, a_n)$ and $S(P_2, F_1, b_1, \ldots, b_n)$ where $P_1$ and $P_2$ are primary keys of schemas $R$ and $S$ respectively, $P_2$ takes its value from $P_1$ of schema $R$ and $F_1$ is foreign key of schema $S$ that references the primary key $P_1$ of schema $R$.

**Output**: Add unary many-to-many association (it is a many-to-many association between instances of the same relational table) to $C_R$ class in conceptual (UML) model.

**EXAMPLE 3.7**

ITEM (ITEM_ID, NAME, UNIT_COST)  
COMPONENT (ITEM_ID, COMPONENT_ID, QUANTITY)
Rule R.9

Add one-to-one association (supertype/subtype) to class.

Input: Schemas of relational tables \( R(P_1, a_1, ...a_n) \) and \( S(K_1, b_1, ...b_n) \) where \( P_1 \) and \( K_1 \) are primary keys of schemas \( R \) and \( S \) respectively and \( K_1 \) is a foreign key of schema \( S \) that references the primary key \( P_1 \) of schema \( R \).

Output: Add one-to-one association to \( C_R \) and \( C_S \) classes in conceptual (UML) model.

EXAMPLE 3.8

\begin{verbatim}
EMPLOYEE (EMPLOYEE_ID, NAME, EMPLOYEE_TYPE, DATE_HIRED)
HOURLY_EMPLOYEE (H_EMPLOYEE_ID, HOURLY_RATE)
SALARIED_EMPLOYEE (S_EMPLOYEE_ID, ANNUAL_SALARY)
\end{verbatim}
AUTOMATED TRANSFORMATION OF DATABASE STRUCTURES

The process of reverse engineering of a relational database can be automated. Some commercial products exist (Rational Rose, OR-Compass, Oracle Database Designer, InfoModeler), which provide tools for conceptual modeling and also generate physical schemas either directly towards the database or inside a data definition language file [22]. According to Grimes [11,12] these tools are still immature and functionally incomplete. In particular they do not provide enough methodological modeling assistance in deciding how to implement semantic relationships.

3.2.2 APPLYING TRANSFORMATION RULES TO TRANSFORM A RELATIONAL DATABASE INTO A CONCEPTUAL (UML) MODEL
Below is a relational database

```sql
EMPLOYEE (EMPLOYEE_ID, NAME, ADDRESS, CITY, STATE, CODE,
EMPLOYEE_TYPE, DATE_HIRED, MANAGER_ID);

DEPENDENT (FIRST_NAME, MIDDLE_NAME, LAST_NAME, EMPLOYEE_ID,
GENDER, DOB);

HOURLY_EMPLOYEE (H_EMPLOYEE_ID, HOURLY_RATE);

SALARIED_EMPLOYEE (S_EMPLOYEE_ID, ANNUAL_SALARY,
STOCK_OPTIONS);

CONSULTANT (C_EMPLOYEE_ID, BILLING_RATE, CONTACT_NUMBER);

DEPARTMENT_CENTRE (DEP_CENT_ID, LOCATION, DATE_ASSIGNED,
EMPLOYEE_IN_CHARGE);

EMPLOYEE_SKILL (EMPLOYEE_ID, SKILL);
```

Relational table EMPLOYEE_SKILL has primary key EMPLOYEE_ID that plays role of foreign key as well and it reference primary key EMPLOYEE_ID of EMPLOYEE relational table. At the same time attribute SKILL in EMPLOYEE_SKILL relational table can be viewed as an attribute of EMPLOYEE, so according to rule (R.6) relational table EMPLOYEE_SKILL is a multivalued attribute of class EMPLOYEE.
In relational table EMPLOYEE, foreign key MANAGER_ID references primary key EMPLOYEE_ID in the same relational table, then according to rule (R.7) construct unary one-to-many association.

Relational table DEPENDENT has foreign key EMPLOYEE_ID that reference EMPLOYEE relational table but without it can not exist independently, because primary key DEPENDENT_NAME does not uniquely identify each dependent. So according to rule (R.5) construct one-to-many association from relational tables DEPENDENT and EMPLOYEE.

Relational table DEPARTMENT_CENTRE has one primary key DEP_CENT_ID and one foreign key EMPLOYEE_ID which references the primary key of EMPLOYEE relational table; each employee assigned to one department centre so according to rule (R.2) construct one-to-one association between these two relational tables.

Relational tables HOURLY_EMPLOYEE, SALARIED_EMPLOYEE and CONSULTANT each have primary keys that are playing the role of foreign keys as well and reference primary key of EMPLOYEE relational table. According to rule (R.9) a subtype/supertype relationship between these four relational tables is constructed.

Figure 3.1 presents conceptual (UML) model of the relational database.
3.3 TRANSFORMATION OF CONCEPTUAL SCHEMA INTO OBJECT-RELATIONAL SCHEMA

The process of transformation from conceptual (UML) model to Object-Relational Database schema best known as *Database Forward Engineering*.

During database forward engineering it is required to take a conceptual model or model obtained from a database reverse engineering process and follow certain steps to perform the creation of an object-relational model.
The process of database forward engineering to an object-relational database schema depends on the understanding of the conceptual model, frequency of traversal access between two object-relational tables in database applications and personal preferences in the object-relational design of the database.

### 3.3.1 APPLICATION ACCESS

The direction of traverse from one object-relational table to another and frequency of query access is very important when constructing an object-relational database, because it influences the type of association, either single reference or double reference association, and the direction of pointer references when navigating from one object-relational table with objects to another.

Consider the following example of two relational tables \( R (P_1, a_1, ..., a_n) \) and \( S (P_2, F_1, b_1, ...b_n) \) with one-to-one association. After transforming relational tables into object-relational tables \( T (a_1, ..., a_n) \) and \( V (b_1, ..., b_n) \), references between tables can be created in three ways: single reference from \( T \) to \( V \), single reference from \( V \) to \( T \) or double reference from \( T \) to \( V \) and from \( V \) to \( T \), Figure 3.2.

The direction that the heads of the arrows points determined by how 'important' (how frequent accessed) the object-relational table is; the base of the arrow is attached to the more 'important' table. For example in the pair of tables Employee and Dependent, table Employee accessed 10 times a day and table Dependent accessed 3 times a year, so table Employee is more 'important'.
Decision on the single or double reference must be as simple as possible. So, it can be either (A) or (B), see Figure 3.2.

In the following example there are two object-relational tables Employee and Dependent, Figure 3.3. Original relational tables Employee and Dependent have one-to-many associations. Object-relational tables have single reference association from Dependent to Employee (assuming that object-relational table Dependent accessed more frequently than the table Employee).
If the access path runs in the same way as the direction of reference then the query is very simple.

Find employee details whose dependent date of birth is 2-Dec-84.

```
SELECT DEREF(Q)
FROM DEPENDENT D
WHERE D.DOB = 2-DEC-84;
```

Explain plan for the above query (query A) shows that both object-relational tables 'work' like one table, due to reference pointers. The first table Dependent is accessed and the row with dependent details whose date of birth is 2-Dec-84 is selected; then SELECT statement is executed that finds the employees details for the relevant dependent.

```
SELECT STATEMENT

TABLE ACCESS FULL DEPENDENT
```
If the access path is in the opposite direction to the reference association then the query is becoming more complex.

To find dependent details that is the responsibility of employee with emp# 03.

```
SELECT D.NAME, D.DOB
FROM DEPENDENT D, EMPLOYEE E
WHERE D.EMP# = E.EMP#
AND E.EMP# = 03;
```

Explain plan for the above query (query B) shows that query works like on relational tables. First both tables are sorted over employee number and then joined together; and finally the `SELECT` statement is executed over the selected rows to find the required dependent details.

```
SELECT STATEMENT
  MERGE JOIN
    SORT JOIN
      TABLE ACCESS FULL EMPLOYEE
    SORT JOIN
      TABLE ACCESS FULL DEPENDENT
```

Now, after establishing the importance of direction of reference and the direction of traversal of the application, the real decision on the kind of reference, either single or double, is done considering frequency of application access as well as comparing all costs and benefits for the particular pair of object-relational tables.
Figure 3.4 presents frequency of access of one object-relational table through another.

Figure 3.4 (A) shows the access from object-relational table T to table V — 5 times a day, and from table V to table T — 10 times a day. Access from table V to table T is more important, more frequent, so the direction of pointer references will be from V to T. Figure 3.4 (B) shows that access from object-relational table T to table V is more important, more frequent, so the direction of pointer references will be from T to V.

If the direction of pointer references are difficult to estimate, as in figure 3.4 (C), which shows that access from table T to table V and from table V to table T are equal, so both paths have equal importance. In this case it will be reasonable to create double reference association between tables T and V.

To establish the type of association it is required to estimate the worthiness of each.
<table>
<thead>
<tr>
<th>Single reference association</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Benefits</strong></td>
<td><strong>Costs</strong></td>
</tr>
<tr>
<td>Easier query construction</td>
<td>Information can be more difficult to access in the direction opposite the access path</td>
</tr>
<tr>
<td>Simple association coding</td>
<td>Takes more time in accessing the information than double reference association</td>
</tr>
<tr>
<td></td>
<td>If a row is deleted from one table then it is necessary to delete references to it from another table</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Double reference association</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Benefits</strong></td>
<td><strong>Costs</strong></td>
</tr>
<tr>
<td>Information can be easily accessed from either table to another.</td>
<td>If a row is deleted from one of the tables it is necessary to delete all relevant information and references to it from another table.</td>
</tr>
<tr>
<td>Saves time in accessing the information</td>
<td>Complicated double reference association coding and query construction</td>
</tr>
</tbody>
</table>

*Table 3.1 Worthiness of single reference and double reference associations*

### 3.3.2 TRANSFORMATION RULES

When transforming the conceptual (UML) model into an Object-Relational database the following rules are listed below. There are three situations that are worthy of considering. They are nested tables, single reference associations and double reference associations.

**Rule F.1**
If there is a class $C_R$ then it can be transformed into object-relational schema

**Input:** Conceptual (UML) class $C_R$ (reference rule R.1)

<table>
<thead>
<tr>
<th>$C_R$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_1$</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>$a_n$</td>
</tr>
</tbody>
</table>

**Output:** Object-relational schema $T(a_1, \ldots, a_n)$.

Rule F.2

If there is a multivalued attribute in the class then one can transform it into nested table.

**Input:** Conceptual (UML) model of class $C_R$ containing multivalued attribute (reference rule R.5).

**Output:** Object-relational nested tables $T(a_1, \ldots, a_n, N(a_i, \ldots, a_j))$. 

$T: a_1, \ldots, a_n, N: a_i, \ldots, a_j$
EXAMPLE 3.9

CREATE TYPE SKILL_TYPE AS OBJECT
    (SKILL VARCHAR2(15));
/
CREATE TYPE SKILL_T AS TABLE OF SKILL_TYPE;
/
CREATE TYPE EMPLOYEE_TYPE AS OBJECT
    (EMPLOYEE_ID NUMBER(6),
     NAME VARCHAR2(40),
     ADDRESS VARCHAR2(40),
     SKILLS SKILL_T);

CREATE TABLE EMPLOYEE_TABLE OF EMPLOYEE_TYPE
    NESTED TABLE SKILLS STORE AS NESTED_SKILL;

Rule F.3

If two classes both have one-to-many association and class on the 'many' side
does not have any other association then in object-relational schema it become
a nested table, considering that no application traverses the association in the
direction from 'many' side to 'one' side.

**Input:** Conceptual (UML) model containing two classes $C_R$ and $C_S$ with one-to-
many association (reference rule R.1, rule R.4).

**Output:** Object-relational nested tables $T(a_1, \ldots a_n, N(b_1, \ldots b_n))$.

```
T: a_1, \ldots, a_n
N: a_i, \ldots, a_j
```

**EXAMPLE 3.10**

<table>
<thead>
<tr>
<th>Employee</th>
<th>Dependent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Employee_ID {CK1}</td>
<td>Dependent_Name {CK1}</td>
</tr>
<tr>
<td>Name</td>
<td>DOB</td>
</tr>
<tr>
<td>Address</td>
<td></td>
</tr>
</tbody>
</table>

CREATE TYPE DEPENDENT_TYPE AS OBJECT
    (D_NAME VARCHAR2 (15),
     DOB DATE);
/
CREATE TYPE DEPENDENT_T AS TABLE OF DEPENDENT_TYPE;
/
CREATE TYPE EMPLOYEE_TYPE AS OBJECT
    (EMPLOYEE_ID NUMBER (6),
     NAME VARCHAR2(40),
     ADDRESS VARCHAR2(40),
     DEPENDENTS DEPENDENT_T);
CREATE TABLE EMPLOYEE_TABLE OF EMPLOYEE_TYPE
NESTED TABLE DEPENDENTS STORE AS NESTEDDEPENDENTS;

Employee: Employee_ID, Name, Address, Dependents: D_name, DOB

Rule F.4

If there is one-to-one association between two classes then it depends on the application demand, one may combine two classes into one table in object-relational schema or use single reference or double reference association.

Input: Conceptual (UML) model containing two classes CR and Cs with one-to-one association (reference rule R.1).

Output: Create one object-relational table T(a₁, ...aₙ, b₁, ...bₙ); or two object-relational tables with single reference association T(a₁, ...aₙ @Ref), V(b₁, ...bₙ); or two object-relational tables with double reference association T(a₁, ...aₙ @Ref), V(b₁, ...bₙ, N(@Ref)).

Two object-relational tables with single reference association
EXAMPLE 3.11 of creating two object-relational tables with single reference association.

CREATE TYPE EMPLOYEE_TYPE AS OBJECT
    (EMPLOYEE_ID NUMBER(6),
     NAME VARCHAR2(40),
     ADDRESS VARCHAR2(40));
/
CREATE TYPE DEPARTMENT_CENTRE_TYPE AS OBJECT
    (DEP_ID NUMBER(6),
     LOCATION VARCHAR2(40),
     DATE_ASSIGN DATE,
     D RBF EMPLOYEE_TYPE);
/
CREATE TABLE EMPLOYEE OF EMPLOYEE_TYPE;
CREATE TABLE DEPARTMENT_CENTRE OF DEPARTMENT_CENTRE_TYPE;

Rule F.5
Many-to-many association is transformed into a single reference association; link attribute becomes a nested table. Or one can create double reference association when there is an established reason for using double reference association between two classes such as equal frequency of traversal access to one class through another and vice versa.

**Input:** Conceptual (UML) model containing two classes $C_R$ and $C_S$ with many-to-many association (reference rule R.2).

**Output:** Create two object-relational tables with single reference association $T(a_1, \ldots a_n, @Ref)$, $V(b_1, \ldots b_n)$; or two object-relational tables with double reference association $T(a_1, \ldots a_n, @Ref)$, $V(b_1, \ldots b_n, N(@Ref))$.

EXAMPLE 3.12 of two object-relational tables with double reference association.
CREATE TYPE PRODUCT_TYPE
/
CREATE TYPE ORDER_LINE_TYPE AS OBJECT
  (QUANTITY NUMBER (4),
   Q REF PRODUCT_TYPE);
/
CREATE TYPE ORDER_LINE_T AS TABLE OF ORDER_LINE_TYPE;
/
CREATE TYPE ORDER_TYPE AS OBJECT
  (ORDER_ID NUMBER (6),
   ORDER_DATE DATE,
   P REF CUSTOMER_TYPE,
   ORDER_L ORDER_LINE_T);
/
CREATE TABLE ORDER_TABLE OF ORDER_TYPE
  NESTED TABLE ORDER_L STORE AS NESTED_ORDER_LINE;
/
/* 'many' side of the association is implemented as ORDERS columns of type nested table of references to ORDER_TABLE objects */

CREATE TYPE ORDERREF_TYPE AS OBJECT
  (ORDERREF REF ORDER_TYPE);
/
CREATE TYPE ORDERREF_SET AS TABLE OF ORDERREF_TYPE;
CREATE TYPE PRODUCT_TYPE AS OBJECT

(PRODUCT_ID NUMBER(6),
NAME VARCHAR2(40),
UNIT_PRICE NUMBER(6,2),
DESCRIPTION VARCHAR2(40),
ORDERS ORDERREF_SET);

CREATE TABLE PRODUCT OF PRODUCT_TYPE
NESTED TABLE ORDERS STORE AS NESTED_ORDERS;

3.3.3 APPLYING TRANSFORMATION RULES TO TRANSFORM THE CONCEPTUAL (UML) MODEL INTO AN OBJECT-RELATIONAL DATABASE

Figure 3.5, presents a conceptual (UML) model of relational database obtained from Figure 3.1. In the following database data is accessed by queries using the following paths. As seen from access path it is evident that the association between EMPLOYEE, HOURLY_EMPLOYEE, SALARIED_EMPLOYEE and CONSULTANT classes is by single reference association; and association between EMPLOYEE and DEPARTMENT CENTRE is also by single reference.
Figure 3.5 Conceptual (UML) model

Note: Supertype/subtype relationship has been omitted in the original design and replaced by one-to-one association. This design has been working perfectly in the relational database, so there is no need to change it and complicate the design of the object-relational database.

Below is object-relational database script

/*Create user-defined object types*/
CREATE TYPE DEPARTMENT_CENTRE_TYPE AS OBJECT
  (DEP_ID NUMBER(6),
   LOCATION VARCHAR2(40),
   DATE_ASSIGN DATE,
   D REF EMPLOYEE_TYPE);
/

CREATE TYPE DEPENDENT_TYPE AS OBJECT
  (D_NAME VARCHAR2(15),
   DOB DATE);
/

CREATE TYPE DEPENDENT_T AS TABLE OF DEPENDENT_TYPE;
/

CREATE TYPE H_EMP_TYPE AS OBJECT
  (H_RATE NUMBER(6,2));
/CREATE TYPE S_EMP_TYPE AS OBJECT
  (ANNUAL_SAL NUMBER(6,2),
  STOCK_OPTION NUMBER(6));
/
CREATE TYPE C_EMP_TYPE AS OBJECT
  (BILLING_RATE NUMBER(6,2),
  CONTACT_NO NUMBER(10));
/
CREATE TYPE EMPLOYEE_TYPE AS OBJECT
  (EMPLOYEE_ID NUMBER(6),
  NAME VARCHAR2(40),
  ADDRESS VARCHAR2(40),
  EMP_TYPE VARCHAR2(2),
  DEP_ID NUMBER(6),
  LOCATION VARCHAR2(40),
  MANAGER_ID NUMBER(6),
  H REF H_EMP_TYPE,
  S REF S_EMP_TYPE,
  C REF C_EMP_TYPE,
  DEPENDENTS DEPENDENT_T);
/
CREATE TABLE EMPLOYEE_TABLE OF EMPLOYEE_TYPE
NESTED TABLE DEPENDENTS STORE AS NESTED_DEPENDENTS;

/*Create tables using object types*/
CREATE TABLE H_EMP OF H_EMP_TYPE;
CREATE TABLE S_EMP OF S_EMP_TYPE;
CREATE TABLE C_EMP OF C_EMP_TYPE;
CREATE TABLE DEPARTMENT_CENTRE OF DEPARTMENT_CENTRE_TYPE;

Based on rule (F.4) and direction of query access, create a single reference association between object-relational tables with objects DEPARTMENT_CENTRE and EMPLOYEE; H_EMP, S_EMP, C_EMP and EMPLOYEE. Bases on rule
(F.3), create a nested table DEPENDENT in an object-relational table EMPLOYEE.

Figure 3.6 presents object-relational database schema.

![Diagram of object-relational database schema]

3.4 TRANSFORMATION OF RELATIONAL SCHEMA INTO OBJECT-RELATIONAL SCHEMA.

The above sections explain in detail Database Reverse Engineering transformation rules (marked by R.n) and Database Forward Engineering transformation rules
(marked by F.n). The following section will describe concatenation of some of transformation rules (marked by RF.n).

**Rule RF.1**

The following rule is a combination derived from rules R.1 and F.1. The combination of these two rules does not contradict with the necessity for having the conceptual (UML) model.

If relational schema transformed into object-relational schema, then relational table becomes table with objects

**EXAMPLE 3.13**

**Input:** Relational table $R (P_1, F_1, a_1, ...a_n)$, where $P_1$ is a primary key and $F_1$ is a foreign key.
Output: Object-relational table T (a₁, ..., aₙ, b₁, ..., bₙ).

\[
T: a₁, ..., aₙ, b₁, ..., bₙ
\]

During transformation from relational schema to object-relational schema foreign key \( F \) is dropped, because it serves the purpose of creating an association. Primary key \( P₁ = \{b₁, ..., bₙ\} \).

Rule RF.2

The following rule is the combination of rules R.2, R.5, F.4, F.5. It is a general rule that will not provide detailed semantic relationships.

If there is in existence relational tables R (P, a₁, a₂, ..., aₙ) and S (F, b₁, b₂, ..., bₙ), such that P is a primary key of R and F is a foreign key of S (which references the primary key P of R) then when relational tables R and S have transformed into object-relational tables, one must add a single reference (object-relational tables T(a₁, ..., aₙ) and V(b₁, ..., bₙ, @Ref)) or a double reference associations (object-relational tables T(a₁, ..., aₙ, N(@Ref)) and V(b₁, ..., bₙ, @Ref)) depending on the frequency and direction of traverse from one table with objects to another.

EXAMPLE 3.14 of single reference association
Relational tables PRODUCT and ORDER. Product_ID is primary key of PRODUCT. Order_ID is primary key of ORDER and Product_ID is foreign key of ORDER.

```
PRODUCT (PRODUCT_ID, PRODUCT_NAME);
ORDER (ORDER_ID, ORDER_DATE, PRODUCT_ID);
```

Object-relational tables with objects ORDER and PRODUCT, created with user-defined object type. The object-relational table PRODUCT has a reference to table ORDER that creates a single reference association.

```
CREATE TYPE ORDER_TYPE AS OBJECT
   (ORDER_ID NUMBER (6),
    ORDER_DATE DATE);
/
CREATE TABLE ORDER OF ORDER_TYPE;

CREATE TYPE PRODUCT_TYPE AS OBJECT
   (PRODUCT_ID NUMBER(6),
    NAME VARCHAR2(40),
    O REF ORDER_TYPE);
/
CREATE TABLE PRODUCT OF PRODUCT_TYPE
```

Rule RF.3

The following rules are a combination of database reverse engineering, rule R.6 and database forward engineering rule, F.1 and F.2. It is a general rule and will not provide detailed semantic relationship.
If there is in existence relational tables $R(\ldots a_n)$ and $S(\ldots b_n)$, then attributes in table $S$ are considered to be multivalued attributes of table $R$. Next, add nested table $S(b_1, \ldots b_n) \rightarrow N(b_1, \ldots b_n)$, creating nested object-relational tables $T(a_1, \ldots, a_n, N(b_1, \ldots, b_n))$

EXAMPLE 3.14

Relational table SKILL has attribute considered to be multivalued to the relational table EMPLOYEE

```sql
EMPLOYEE (EMPLOYEE_ID, NAME, DATE_HIRED);
SKILL (EMPLOYEE_ID, SKILL);
```

Object-relational table EMPLOYEE has nested tables SKILLS, which created with user-defined object types.

```sql
CREATE TYPE SKILL_TYPE AS OBJECT
    (SKILL VARCHAR2 (15));
/
CREATE TYPE SKILL_T AS TABLE OF SKILL_TYPE;
/
CREATE TYPE EMPLOYEE_TYPE AS OBJECT
    (EMPLOYEE_ID NUMBER (6),
     NAME VARCHAR2(40),
     SKILLS SKILL_T);
/
CREATE TABLE EMPLOYEE_TABLE OF EMPLOYEE_TYPE
    NESTED TABLE SKILLS STORE AS NESTED_SKILL;
```
Rule RF.4

If there is in existence relational table \( R(a_1, \ldots, a_n) \), and some attributes are considered to be part of a composite attribute (can be grouped together), then one can create a separate object in an object-relational table.

EXAMPLE 3.15

Relational table CUSTOMER has attributes that can be considered composite, they are ADDRESS, CITY, STATE, CODE.

```sql
CREATE TABLE CUSTOMER
    (CUSTOMER_ID INTEGER NOT NULL,
     NAME VARCHAR2(40) NOT NULL,
     ADDRESS VARCHAR2(40),
     CITY VARCHAR2(25),
     STATE VARCHAR2(10),
     CODE NUMBER(5),
    CONSTRAINT CUSTOMER_PK PRIMARY KEY (CUSTOMER_ID));
```

In object-relational database composite attribute transformed into separate object type.

```sql
CREATE TYPE ADDRESS_TYPE AS OBJECT(
    ADDRESS VARCHAR(40),
    CITY VARCHAR(25),
    STATE VARCHAR(10),
    CODE NUMBER(5));
/
CREATE TYPE CUSTOMER_TYPE AS OBJECT(
    CUSTOMER_ID INTEGER,
```

```
NAME VARCHAR(40),
ADDRESS ADDRESS_TYPE);
/
CREATE TABLE CUSTOMER OF CUSTOMER_TYPE;
Chapter 4
Correctness of transformation rules

All Data Manipulation statements (DELETE, UPDATE, INSERT), Data Definition statements (CREATE INDEX, CREATE VIEW, CREATE TABLE) and Data Retrieval statements (SELECT) fall into two categories

- Statements that change database state
- Statements that change the state of variables

4.1 STATEMENTS THAT CHANGE DATABASE STATE

Let \( S(R) (v_1, ..., v_n) \) be SQL DML statements that manipulate on relational database \( R \), where \( v_1, ..., v_n \) are the host variables and let \( S^{(O-R)} (v_1, ..., v_n) \) be O-R SQL DML statements that manipulate on object-relational database \( O-R \), where \( v_1, ..., v_n \) are the host variables. Let \( T \) be a set of transformation rules that translate relational database \( R \) into object-relational database \( O-R \). Let \( R^{d(R)} \) be a state of relational database and \( R^{d(O-R)} \) be as state of object-relational database after applying SQL DML statements \( S^{(R)} \) and \( S^{(O-R)} \) respectively.

Statements \( S^{(R)} (v_1, ..., v_n) \) and \( S^{(O-R)} (v_1, ..., v_n) \) are semantically equivalent iff \( T(R) = (O-R) \) then after execution of \( R^{d(R)} := S^{(R)} (v_1, ..., v_n) [R] \) and \( R^{d(O-R)} := S^{(O-R)} (v_1, ..., v_n) [O-R] \).
... vn) [O-R] the state of relational and object-relational databases are the same
T(R^{(R)}) = R^{(O-R)}

4.2 STATEMENTS THAT CHANGE THE STATE OF VARIABLES

Let S^{(R)}(v_1, ..., v_n) be SQL DML statements that manipulate on relational database
R, where v_1, ..., v_n are the host variables, and let S^{(O-R)}(v_1, ..., v_n) be O-R SQL
DML statements that manipulate on object-relational database OR, where v_1, ..., v_n
are the host variables. Let T be a set of transformation rules that translate relational
database R into the object-relational database O-R. Let R^{(R)} be a state of relational
database and R^{(O-R)} be as state of object-relational database after applying SQL
DML statements S^{(R)} and S^{(O-R)} respectively.

Statements S^{(R)}(v_1, ..., v_n) and S^{(O-R)}(v_1, ..., v_n) are semantically equivalent if T(R)
= (O-R) then after execution of R^{(R)} := S^{(R)}(v_1, ..., v_n) [R] and R^{(O-R)} := S^{(O-R)}(v_1,
..., v_n) [O-R] the state of the host variables are equivalent v'_1 = v''_1, v'_2 = v''_2, ... v'_n
= v''_n.

4.3 CORRECTNESS OF TRANSFORMATION RULES

Path A, Figure 4.1 is a translation of relational SQL statement S^{(R)}(v_1, ..., v_n) into
object-relational O-R SQL statement S^{(O-R)}(v_1, ..., v_n) using translation rules T^S and
execution of an O-R SQL statement on object-relational database O-R.
Path B, Figure 4.1 is the execution of relational SQL statement $S^{(O-R)}(v_1, \ldots, v_n)$ on relational database $R$ and relational database $R$ transformed into object-relational database O-R by applying transformation rules $T^D$.

![Figure 4.1 Translation of statements](image)

We can say that if going through path A or path B object-relational database results in the same state then relational SQL statement $S^{(R)}(v_1, \ldots, v_n)$ and object-relational O-R SQL statement $S^{(O-R)}(v_1, \ldots, v_n)$ are semantically equivalent.

Consider the following example

There is a line of script from Java program (SQLJ) that inserts values into the relational table PERSONS [17].

```java
/* Insert a new PERSON object into the persons table */
try {
    #sql {
        INSERT INTO persons
```
VALUES (:new_name, :new_ssn, :new_street, :new_city, :new_state, :new_zip);}

The above SQL statement is translated into the O-R SQL statement and the result is placed in the Java program instead of the relational SQL statement. Below is a line of script from Java program (SQLJ) that inserts values into object-relational nested table PERSONS.

/* Insert a new PERSON object into the nested persons table*/
try {
  #sql {
    INSERT INTO persons
    VALUES (PERSON(:new_name, :new_ssn, ADDRESS(:new_street, :new_city, :new_state, :new_zip)))
  }
}

Above SQL statement and O-R SQL statements produce the same results when used in the Java program.

4.4 CORRECTNESS OF TRANSFORMATION RULES FOR DATA MIGRATION

From relational database R had been generated INSERT relational SQL statements $S^{(R \text{INSERT})}(v_1, \ldots, v_n)$ then applying translation rules $T^S$ translate INSERT relational SQL statements into INSERT object-relational SQL statements $S^{(O-R \text{INSERT})}(v_1, \ldots, v_n)$ and execute them on object-relational database O-R, Figure 4.2.
If input variables \((v_1, \ldots, v_n)\) into INSERT relational and object-relational SQL statements and output variables \((v_1', \ldots, v_n')\) from INSERT relational and object-relational SQL statements are the same then translation rules \(T^S\) are correct.

\[
S^{(R\ INSERT)}(v_1, \ldots, v_n) \xrightarrow{T^S} S^{(O-R\ INSERT)}(v_1', \ldots, v_n')
\]

*Figure 4.2 Data migration*

For the example please refer to example 6.1 page 125
Chapter 5
Translation of relational SQL statements to object-relational SQL statements

When translating SQL statements into O-R SQL statements we need to consider three general types of statements:

- Data Definition statements
  - CREATE TABLE, CREATE INDEX, and ALTER statements are used when database is created.

- Data Manipulation statements
  - INSERT, DELETE and UPDATE statements used for data manipulation

- Data Retrieval statements
  - SELECT statement used to retrieve data from database

5.1 DATA DEFINITION STATEMENTS

5.1.1 CREATE INDEX STATEMENT

If there is an index defined on a relational (flat) table, that same index can be defined on an object table or on the storage table for a nested table column or attribute, just
as on other tables, when a relational (flat) table is transformed into an object-relational table. Also, where indexes on REF attributes or columns if the REF is scoped [18], where REF is a logical ‘pointer’ to a row object REFS and a collection of REFS provides associations among objects illuminating the need for foreign keys. Indexes can be defined on leaf-level scalar attributes of column objects.

In the following example, UNIQUE index UNIQ_PROJ_INDX is created on storage table NESTED_PROJECT_TABLE. Including pseudocolumn NESTED_TABLE_ID ensures distinct rows in nested table column PROJS_MANAGED:

```sql
CREATE TYPE proj_type AS OBJECT
    (proj_num NUMBER, proj_name VARCHAR2(20));
CREATE TYPE proj_table_type AS TABLE OF proj_type;
CREATE TABLE employee
    (emp_num NUMBER, emp_name CHAR(31),
    projs_managed proj_table_type )
NESTED TABLE projs_managed STORE AS nested_project_table;
CREATE UNIQUE INDEX uniq proj indx ON nested_project_table
    (NESTED_TABLE_ID, proj_num);
```

5.1.2 CREATE VIEW

An object view can be created that supports object datatypes, such as REFS, nested tables, or varray types, on top of the existing view mechanism. Collections, both nested tables and varrays, can be columns in views. Select these collections from
underlying collection columns or synthesize them using subqueries. The CAST-MULTISET operator provides a way of synthesizing such collections [18]. Consider the following example:-

```
CREATE TABLE emp(
    empno NUMBER PRIMARY KEY,
    empname VARCHAR2(20),
    salary NUMBER,
    deptno NUMBER REFERENCES dept(deptno));
```

Construct a dept_view with the department number, name, address and a collection of employees belonging to the department.

Define an employee type and a nested table type for the employee type:

```
CREATE TYPE employee_t AS OBJECT(
    eno NUMBER,
    ename VARCHAR2(20),
    salary NUMBER);
```

```
CREATE TYPE employee_list_t AS TABLE OF employee_t;
```

The dept_view can now be defined:

```
CREATE VIEW dept_view AS
    SELECT d.deptno, d.deptname,
    address_t(d.deptstreet,d.deptcity,d.deptstate,d.deptzip) AS deptaddr,
    CAST(  MULTISET (
        SELECT e.empno, e.empname, e.salary
        FROM emp e
        WHERE e.deptno = d.deptno)
    AS employee_list_t)
    AS emp_list
```
5.1.3 CREATE TABLE STATEMENT

The following transformation rules apply to data definition CREATE TABLE statement.

Rule DD.1

Translation of CREATE TABLE (flat) statement to CREATE TABLE (nested) statement.

Input:

1. Relational tables $R(P_1, a_1, \ldots, a_n)$ and $S(K_1, b_1, \ldots, b_n)$ where $P_1$ and $K_1$ are primary keys of $R$ and $S$ respectively and $K_1$ is foreign key of $S$ that references the primary key $P_1$ of $R$. Attributes $(b_1, \ldots, b_n)$ can be viewed as attributes to relational table $R$.

2. Object-Relational table $T(a_1, \ldots, a_n, N(b_1, \ldots, b_n))$ obtained from application of transformation rule $S(b_1, \ldots, b_n) \rightarrow N(b_1, \ldots, b_n)$ (reference rule F1, F2, RF.3)

3. CREATE TABLE (flat) statement
CREATE TABLE R
(a_1   datatype (size),
 ....
 a_n   datatype (size)
CONSTRAINT R_PK PRIMARY KEY (a_1));

Output:

CREATE TYPE N_TYPE AS OBJECT
(b_1   datatype (size));
/
CREATE TYPE N_T AS TABLE OF N_TYPE;
/
CREATE TYPE T_TYPE AS OBJECT
(a_1   datatype (size),
 ....
 a_n   datatype (size),
 N   N_T);

CREATE TABLE T OF T_TYPE
NESTED TABLE N STORE AS NESTED_N;

EXAMPLE 5.1

<table>
<thead>
<tr>
<th>Relational tables</th>
<th>Object-Relational tables - Nested tables</th>
</tr>
</thead>
</table>
| CREATE TABLE EMPLOYEE
 (EMPLOYEE_ID INTEGER NOT NULL,
  NAME VARCHAR2(40) NOT NULL,
  DATE_HIRED DATE,
 CONSTRAINT EMPLOYEE_PK PRIMARY KEY (EMPLOYEE_ID)); | CREATE TYPE SKILL_TYPE AS OBJECT
 (SKILL VARCHAR2 (15));
 /
 CREATE TYPE SKILL_T AS TABLE OF SKILL_TYPE;
 /
 CREATE TYPE EMPLOYEE_TYPE AS OBJECT
 (EMPLOYEE_ID NUMBER (6), |
<table>
<thead>
<tr>
<th>CREATE TABLE EMPLOYEE_SKILL</th>
<th>NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>(EMPLOYEE_ID INTEGER NOT NULL,</td>
<td>DATE_HIRED DATE,</td>
</tr>
<tr>
<td>SKILL VARCHAR2(15) NOT NULL,</td>
<td>SKILLS SKILL_T);</td>
</tr>
<tr>
<td>CONSTRAINT EMPLOYEE_SKILL_PK</td>
<td>CREATE TABLE EMPLOYEE_TABLE</td>
</tr>
<tr>
<td>PRIMARY KEY (EMPLOYEE_ID,</td>
<td>OF EMPLOYEE_TYPE</td>
</tr>
<tr>
<td>SKILL),</td>
<td>NESTED TABLE SKILLS STORE AS</td>
</tr>
<tr>
<td>CONSTRAINT EMPLOYEE_SKILL_PK</td>
<td>NESTED_SKILL;</td>
</tr>
<tr>
<td>FOREIGN KEY (EMPLOYEE_ID)</td>
<td></td>
</tr>
<tr>
<td>REFERENCE EMPLOYEE</td>
<td></td>
</tr>
<tr>
<td>(EMPLOYEE_ID));</td>
<td></td>
</tr>
</tbody>
</table>

Table 5.1 CREATE TABLE statements for relational and object-relational (nested) tables

Rule DD.2

Translating a CREATE TABLE (flat) statement to a CREATE TABLE statement with single reference association.

Input:

1. Relational tables R(P, a₁, ..., aₙ) and S(F, b₁, ..., bₙ) where P is a primary key of R, F is a foreign key of S which references the primary key P from a relational table R.

2. Object-Relational tables T(a₁, ..., aₙ) and V(b₁, ..., bₙ, @Ref) with single reference association, obtained from the application of transformation rule R(P), S(F) → T, V (@Ref) (reference rule R.2, R.5, F.4, F.5, RF.2)
(3) CREATE TABLE (flat) statement

CREATE TABLE R
(a_1 datatype (size),
....
a_n datatype (size)
CONSTRAINT R_PK PRIMARY KEY (a_1));

CREATE TABLE S
(b_1 datatype (size),
....
b_n datatype (size)
CONSTRAINT S_PK PRIMARY KEY (b_1),
CONSTRAINT S_FK FOREIGN KEY (b_1) REFERENCES R_PK (a_1));

Output

CREATE TYPE T_TYPE AS OBJECT
(a_1 datatype (size),
...a_n datatype (size));
/
CREATE TABLE T OF T_TYPE;

CREATE TYPE V_TYPE AS OBJECT
(b_1 datatype(size),
...b_n datatype(size),
Ref REF T_TYPE);
/
CREATE TABLE V OF V_TYPE

EXAMPLE 5.2
<table>
<thead>
<tr>
<th>Relational tables</th>
<th>Object-relational tables – single reference association</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATE TABLE PRODUCT</td>
<td>CREATE TYPE ORDER_TYPE AS OBJECT</td>
</tr>
<tr>
<td>(PRODUCT_ID INTEGER NOT NULL,</td>
<td>(ORDER_ID NUMBER (6),</td>
</tr>
<tr>
<td>PRODUCT_NAME VARCHAR2(40),</td>
<td>ORDER_DATE DATE);</td>
</tr>
<tr>
<td>CONSTRAINT PRODUCT_PK PRIMARY KEY (PRODUCT_ID));;</td>
<td>/</td>
</tr>
<tr>
<td>CREATE TABLE ORDER</td>
<td>CREATE TABLE ORDER OF ORDER_TYPE;</td>
</tr>
<tr>
<td>(ORDER_ID INTEGER NOT NULL,</td>
<td>CREATE TYPE PRODUCT_TYPE AS OBJECT</td>
</tr>
<tr>
<td>ORDER_DATE DATE DEFAULT SYSDATE,</td>
<td>(PRODUCT_ID NUMBER (6),</td>
</tr>
<tr>
<td>PRODUCT_ID INTEGER NOT NULL,</td>
<td>NAME VARCHAR2(40),</td>
</tr>
<tr>
<td>CONSTRAINT ORDER_PK PRIMARY KEY (ORDER_ID),</td>
<td>REF ORDER_TYPE);</td>
</tr>
<tr>
<td>CONSTRAINT ORDER_FK FOREIGN KEY (PRODUCT_ID) REFERENCES PRODUCT (PRODUCT_ID));</td>
<td>/</td>
</tr>
<tr>
<td>CREATE TABLE PRODUCT_TYPE</td>
<td>CREATE TABLE PRODUCT OF PRODUCT_TYPE</td>
</tr>
</tbody>
</table>

Table 5.2 CREATE TABLE statements for relational and object-relational with single-reference association tables.

**Rule DD.3**

Translating a CREATE TABLE (flat) statement to a CREATE TABLE statement with double reference association.

**Input:**
(1) Relational tables $R(P, a_1, ..., a_n)$ and $S(F, b_1, ..., b_n)$ where $P$ is a primary key of $R$, $F$ is a foreign key of $S$ which references the primary key $P$ of $R$ from a relational database.

(2) Object-Relational tables $T(a_1, ..., a_n, N(@Ref))$ and $V(b_1, ..., b_n, @Ref)$ with double reference association obtained from the application of transformation rule $R(P), S(F) \rightarrow T(N(@Ref)), V(@Ref)$ (reference rules R.2, R.5, F.4, F.5, RF.2).

(3) CREATE TABLE (flat) statement

CREATE TABLE R
(a_1 datatype (size),
....
 a_n datatype (size)
CONSTRAINT R_PK PRIMARY KEY (a_1));

CREATE TABLE S
(b_1 datatype (size),
....
 b_n datatype (size)
CONSTRAINT S_PK PRIMARY KEY (b_1),
CONSTRAINT S_FK FOREIGN KEY (b_1) REFERENCES R_PK (a_1));

Output

CREATE TYPE V_TYPE
/
CREATE TYPE T_TYPE AS OBJECT
(a_i datatype (size),
...
CREATE TABLE T_TABLE OF T_TYPE;
/
CREATE TYPE NREF_TYPE AS OBJECT
  (NREF        REF T_TYPE);
/
CREATE TYPE NREF_SET AS TABLE OF NREF_TYPE;
/
CREATE TYPE V_TYPE AS OBJECT
  (bx datatype (size),
   bn datatype (size),
   N NREF_SET);
/
CREATE TABLE V OF V_TYPE
NESTED TABLE N STORE AS NESTED_N;

EXAMPLE 5.3

<table>
<thead>
<tr>
<th>Relational tables</th>
<th>Object-relational tables – single reference association</th>
</tr>
</thead>
</table>
| CREATE TABLE PRODUCT (PRODUCT_ID INTEGER NOT NULL,                               | CREATE TYPE PRODUCT_TYPE
|   PRODUCT_NAME VARCHAR2(40),                                                      | /
|   CONSTRAINT PRODUCT_PK PRIMARY KEY (PRODUCT_ID));                               | CREATE TYPE ORDER_TYPE AS OBJECT
|                                                                                | (ORDER_ID NUMBER (6),
|                                                                                |   ORDER_DATE DATE,
|                                                                                |   P                 REF PRODUCT_TYPE);                                         |
| CREATE TABLE ORDER (ORDER_ID INTEGER NOT NULL, ORDER_DATE DATE DEFAULT SYSDATE, | /
|   PRODUCT_ID INTEGER NOT NULL,                                                   | CREATE TABLE ORDER_TABLE OF ORDER_TYPE
|   CONSTRAINT ORDER_PK PRIMARY KEY (ORDER_ID),                                     | CREATE TYPE ORDERREF_TYPE AS OBJECT
|   CONSTRAINT ORDER_FK FOREIGN KEY                                                 | (ORDERREF       REF ORDER_TYPE);                                               |
CREATE TYPE ORDERREF_SET AS TABLE OF ORDERREF_TYPE;
/
CREATE TYPE PRODUCT_TYPE AS OBJECT
(PRODUCT_ID
NUMBER(6),
NAME VARCHAR2(40),
UNIT_PRICE
NUMBER(6,2),
ORDERS ORDERREF_SET);
/
CREATE TABLE PRODUCT OF
PRODUCT_TYPE
NESTED TABLE ORDERS STORE AS
NESTED_ORDERS;

Table 5.3 CREATE TABLE for relational and object-relational with double reference association tables

5.2 DATA MANIPULATION STATEMENTS

5.2.1 TRANSFORMATION RULES

The following transformation rules apply to data manipulation statements INSERT, UPDATE, DELETE.

INSERT

Rule DM.1

Translation of INSERT (flat) statement to INSERT (nested) statement.
Input:

1. Relational tables $R(P_1, a_1, ... a_n)$ and $S(K_1, b_1, ... b_n)$ where $P_1$ and $K_1$ are primary keys of $R$ and $S$ respectively and $K_1$ is the foreign key of $S$ that references the primary key $P_1$ of $R$. Attributes $(b_1, ..., b_n)$ can be viewed as attributes to relational table $R$.

2. Object-Relational table $T(a_1, ..., a_n, N(b_1, ..., b_n))$ obtained from application of transformation rule $S(b_1, ..., b_n) \rightarrow N(b_1, ... b_n)$ (reference rule F.1, F.2, RF.3)

3. INSERT statement

```
INSERT INTO R (a_1, ..., a_n) VALUES (v_1, v_2, ..., v_n);
INSERT INTO S (b_1, ..., b_n) VALUES (w_1, ..., w_n);
```

Output:

```
INSERT INTO T VALUES (v_1, N(N_T(w_1 # w_n)));
```

Nested table only

```
INSERT INTO TABLE
(SELECT T.N
 FROM T
WHERE 'condition')
```
EXAMPLE 5.4

<table>
<thead>
<tr>
<th>Relational tables</th>
<th>Object-Relational tables – Nested tables</th>
</tr>
</thead>
<tbody>
<tr>
<td>INSERT INTO Applicant VALUES (1, 'JOHN', 'SMITH');</td>
<td>INSERT INTO Applicant VALUES (1, 'John', 'Smith', SKILL_SET (SKILL_TYPE('VB', 'expert')));</td>
</tr>
<tr>
<td>INSERT INTO Skills VALUES (1, 1, 'VB', 'EXPERT');</td>
<td>Nested table only</td>
</tr>
<tr>
<td>INSERT INTO TABLE (SELECT A.Skills FROM Applicant A WHERE A.f_name = 'John') VALUES ('VB', 'expert');</td>
<td></td>
</tr>
</tbody>
</table>

Table 5.4 INSERT statement for relational and object-relational (nested) tables

Rule DM.2

Translation of INSERT statements on relational tables to INSERT statements on object-relational tables with single reference association.

Input:

(1) Relational tables R(P, a₁, ..., aₙ) and S(F, b₁, ..., bₙ) where P is a primary key of R, F is a foreign key of S which references the primary key P from a relational table R.
(2) Object-Relational tables $T(a_1, \ldots, a_n)$ and $V(b_1, \ldots, b_n, @Ref)$ obtained from the application of transformation rule $R(P), S(F) \rightarrow T, V (@Ref)$. (rule R.2, R.5, F.4, F.5, RF.2)

(3) INSERT statement

\[
\text{INSERT INTO } R \text{ VALUES } (v_1, v_2, \ldots, v_n); \\
\text{INSERT INTO } S \text{ VALUES } (w_1, w_2, \ldots, w_n); \\
\]

Output:

\[
\text{INSERT INTO } T \text{ VALUES } (v_1, v_2, \ldots, v_n); \\
\text{INSERT INTO } V \text{ VALUES } (w_1, w_2, \ldots, w_n, \text{NULL}); \\
\]

EXAMPLE 5.5

<table>
<thead>
<tr>
<th>Relational tables</th>
<th>Object-Relational tables – single reference association</th>
</tr>
</thead>
<tbody>
<tr>
<td>INSERT INTO Position VALUES (1, 1, 'Database Administrator', 90000);</td>
<td>INSERT INTO Position VALUES (1, 'Database Administrator', 90000);</td>
</tr>
<tr>
<td>INSERT INTO Applicant VALUES (1, 'John', 'Smith');</td>
<td>INSERT INTO Applicant VALUES (1, 'John', 'Smith', NULL);</td>
</tr>
</tbody>
</table>

*Table 5.5 Insert statement for relational and object-relational with single reference association tables.*
Rule DM.3

Translation of INSERT statements on relational tables to INSERT statements on object-relational tables with double reference association.

Input:

1. Relational tables \( R(P, a_1, \ldots, a_n) \) and \( S(F, b_1, \ldots, b_n) \) where \( P \) is a primary key of \( R \), \( F \) is a foreign key of \( S \) which references the primary key \( P \) of \( R \) from a relational database.

2. Object-Relational tables \( T(a_1, \ldots, a_n, N(@Ref)) \) and \( V(b_1, \ldots, b_n, @Ref) \) obtained from the application of transformation rule \( R(P), S(F) \rightarrow T(N(@Ref)), V(@Ref) \) (reference rule R.2, R.5, F.4, F.5, RF.2).

3. INSERT statement

\[
\text{INSERT INTO } R \text{ VALUES } (v_1, v_2, \ldots, v_n); \\
\text{INSERT INTO } S \text{ VALUES } (w_1, w_2, \ldots, w_n); \\
\]

Output:

\[
\text{INSERT INTO } T \text{ VALUES } (v_1, v_2, \ldots, v_n, N()); \\
\text{INSERT INTO } V \text{ VALUES } (w_1, w_2, \ldots, w_n, \text{NULL}); \\
\]

To set up links between tables.
INSERT INTO TABLE (SELECT N
FROM T
WHERE 'condition')
(SELECT REF (T)
FROM V
WHERE 'condition');

EXAMPLE 5.6

<table>
<thead>
<tr>
<th>Relational tables</th>
<th>Object-Relational tables – double reference association</th>
</tr>
</thead>
<tbody>
<tr>
<td>INSERT INTO Position VALUES (1, 1, 'Database Administrator', 90000);</td>
<td>INSERT INTO Position VALUES (1, 'Database Administrator', 90000, APPLICANTREF_SET());</td>
</tr>
<tr>
<td>INSERT INTO Applicant VALUES (1, 'John', 'Smith');</td>
<td>INSERT INTO Applicant VALUES (1, 'John', 'Smith', NULL);</td>
</tr>
</tbody>
</table>

To set up links between tables

INSERT INTO TABLE (SELECT P.Applicants
FROM Position P
WHERE P.P# = 3)
(SELECT REF(A)
FROM Applicant A
WHERE A.A# = 1);

Table 5.6 INSERT statement for relational and object-relational with double reference association tables.

UPDATE

Rule DM.4
Translation of UPDATE (flat) statements to UPDATE (nested) statements

Input:

(1) Relational tables $R(P_i, a_1, ..., a_n)$ and $S(K_i, b_1, ..., b_n)$ where $P_i$ and $K_i$ are primary keys of $R$ and $S$ respectively and $K_i$ is the foreign key of $S$ that references the primary key $P_i$ of $R$. Attributes $(b_1, ..., b_n)$ can be viewed as attributes to relational table $R$.

(2) Object-Relational table $T(a_1, ..., a_n, N(b_1, ..., b_n))$ obtained from the application of transformation rule $S(b_1, ..., b_n) \rightarrow N(b_1, ..., b_n)$. (reference rule RF.3)

(3) UPDATE statement

\[
\text{UPDATE } R \\
\text{SET } a_1 = v_1 \\
\text{WHERE } '\text{condition}' ;
\]

Output:

\[
\text{UPDATE } T \\
\text{SET } a_1 = v_1 \\
\text{WHERE } '\text{condition}' ;
\]

Nested table only

\[
\text{UPDATE TABLE(SELECT } N
\]
FROM T  
WHERE 'condition')U  
SET U.a₁ = w₁  
WHERE 'condition';

EXAMPLE 5.7

<table>
<thead>
<tr>
<th>Relational tables</th>
<th>Object-Relational tables - Nested tables</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPDATE Applicant</td>
<td>UPDATE Applicant</td>
</tr>
<tr>
<td>SET f_name = 'Jon'</td>
<td>SET f_name = 'Jon'</td>
</tr>
<tr>
<td>WHERE A# = 1;</td>
<td>WHERE A# = 1;</td>
</tr>
<tr>
<td></td>
<td>Nested table only</td>
</tr>
<tr>
<td></td>
<td>UPDATE TABLE</td>
</tr>
<tr>
<td></td>
<td>(SELECT A.Skills</td>
</tr>
<tr>
<td></td>
<td>FROM Applicant A</td>
</tr>
<tr>
<td></td>
<td>WHERE A.A# = 2) S</td>
</tr>
<tr>
<td></td>
<td>SET S.slevel = 'not expert'</td>
</tr>
<tr>
<td></td>
<td>WHERE S.sname = 'singing'</td>
</tr>
</tbody>
</table>

Table 5.7 UPDATE statement for relational and object-relational nested tables

Rule DM.5

Translation of UPDATE statements on relational tables to UPDATE statements on object-relational tables with a single reference association.

Input:
(1) Relational tables $R(P, a_1, \ldots, a_n)$ and $S(F, b_1, \ldots, b_n)$ where $P$ is a primary key of $R$, $F$ is a foreign key of $S$ which references the primary key $P$ of $R$ from a relational database.

(2) Object-Relational tables $T(a_1, \ldots, a_n)$ and $V(b_1, \ldots, b_n, @Ref)$ obtained from the application of transformation rule $R(P), S(F) \rightarrow T, V (@Ref)$. (reference rule $R.2, R.5, F.4, F.5, RF.2$).

(3) UPDATE statement

```
UPDATE R
SET $a_1 = v_1$
WHERE 'condition';

UPDATE S
SET $b_1 = w_1$
WHERE 'condition';
```

Output:

```
UPDATE T
SET $a_1 = v_1$
WHERE 'condition';
```

To set up references

```
UPDATE V
SET $Ref = (SELECT REF(T)
FROM T
WHERE 'condition')$
WHERE 'condition';
```

EXAMPLE 5.8
Table 5.8 UPDATE statements for relational and object-relational with single reference association tables

<table>
<thead>
<tr>
<th>Relational tables</th>
<th>Object-Relational tables – single reference association</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPDATE Applicant</td>
<td>UPDATE Position</td>
</tr>
<tr>
<td>SET f_name = 'Jon'</td>
<td>SET salary = 85000</td>
</tr>
<tr>
<td>WHERE A# = 1;</td>
<td>WHERE title = 'Database Administrator';</td>
</tr>
<tr>
<td></td>
<td>Set up references</td>
</tr>
<tr>
<td></td>
<td>UPDATE Applicant</td>
</tr>
<tr>
<td></td>
<td>SET Applies = (SELECT REF(P)</td>
</tr>
<tr>
<td></td>
<td>FROM Position P</td>
</tr>
<tr>
<td></td>
<td>WHERE P.P# = 3)</td>
</tr>
<tr>
<td></td>
<td>WHERE A# = 1;</td>
</tr>
</tbody>
</table>

Table 5.8 UPDATE statements for relational and object-relational with single reference association tables

Rule DM.6

Translation of UPDATE statements on relational tables to UPDATE statements on object-relational tables with double reference association.

Input:

1. Relational tables R(P, a₁, ..., aₙ) and S(F, b₁, ..., bₙ) where P is a primary key of R, F is a foreign key of S which references the primary key P of R from a relational database
(2) Object-Relational tables $T(a_1, \ldots, a_n, N(@Ref))$ and $V(b_1, \ldots, b_m, @Ref)$ obtained from the application of transformation rule $R(P), S(F) \rightarrow T(N(@Ref)), V(@Ref)$ (reference rule R.2, R.5, F.4, F.5, RF.2).

(3) UPDATE statement

```sql
UPDATE R
SET $a_1 = v_1$
WHERE 'condition';

UPDATE S
SET $b_1 = w_1$
WHERE 'condition';
```

Output:

```sql
UPDATE T
SET $a_1 = v_1$
WHERE 'condition';

UPDATE V
SET Ref = (SELECT REF(T)
            FROM T
            WHERE 'condition')
WHERE 'condition';

Linking the objects

```sql
UPDATE V
SET Ref = (SELECT REF(T)
            FROM T
            WHERE 'condition')
WHERE 'condition';
```

Nested table only

```sql
UPDATE TABLE(SELECT N
              FROM T
              WHERE 'condition')U
SET U.$a_1 = v_1$
WHERE 'condition';
```
**EXAMPLE 5.9**

<table>
<thead>
<tr>
<th>Relational tables</th>
<th>Object-Relational tables – double reference association</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>UPDATE Applicant</strong>&lt;br&gt;SET f_name = 'Jon'&lt;br&gt;WHERE A# = 1;</td>
<td><strong>UPDATE Position</strong>&lt;br&gt;SET salary = 85000&lt;br&gt;WHERE title = 'Database Administrator';</td>
</tr>
<tr>
<td><strong>UPDATE Position</strong>&lt;br&gt;SET salary = 85000&lt;br&gt;WHERE title = 'Database Administrator';</td>
<td>Linking the objects&lt;br&gt;<strong>UPDATE Applicant</strong>&lt;br&gt;SET Applies = (SELECT REF(P)&lt;br&gt;FROM Position P&lt;br&gt;WHERE P.P# = 3)&lt;br&gt;WHERE A# = 1;</td>
</tr>
<tr>
<td>Nested table only&lt;br&gt;<strong>UPDATE TABLE</strong>&lt;br&gt;(SELECT P.Applicants&lt;br&gt;FROM Position P&lt;br&gt;WHERE P.P# = 2) A</td>
<td><strong>UPDATE TABLE</strong>&lt;br&gt;(SELECT P.Applicants&lt;br&gt;FROM Position P&lt;br&gt;WHERE P.P# = 2) A&lt;br&gt;SET A.Applicantref = 1;</td>
</tr>
</tbody>
</table>

*Table 5.9 UPDATE statements on relational and object-relational with double reference association tables*

**DELETE**

**Rule DM.7**

Translation of DELETE (flat) statements to DELETE (nested) statements

**Input:**

96
(1) Relational tables \( R(P_1, a_1, \ldots, a_n) \) and \( S(K_1, b_1, \ldots, b_n) \) where \( P_1 \) and \( K_1 \) are primary keys of \( R \) and \( S \) respectively and \( K_1 \) is a foreign key of \( S \) that references the primary key \( P_1 \) of \( R \). Attributes \( (b_1, \ldots, b_n) \) can be viewed as attributes to relational table \( R \).

(2) Object-Relational table \( T(a_1, \ldots, a_m, N(b_1, \ldots, b_n)) \) obtained from the application of transformation rule \( S(b_1, \ldots, b_n) \rightarrow N(b_1, \ldots, b_n) \). (reference rule F1, F.2, RF.3)

(3) DELETE statement

```sql
DELETE FROM R
WHERE 'condition';
```

Output:

```sql
DELETE FROM T
WHERE 'condition';
```

Nested table only

```sql
DELETE FROM TABLE
(SELECT N
FROM T
WHERE 'condition') U
WHERE U.a_i = v_i;
```

EXAMPLE 5.10
<table>
<thead>
<tr>
<th>Relational tables</th>
<th>Object-Relational tables - Nested tables</th>
</tr>
</thead>
<tbody>
<tr>
<td>DELETE FROM Skills WHERE A# = 1 AND sname = 'programming';</td>
<td>DELETE FROM Applicant WHERE A# = 1;</td>
</tr>
<tr>
<td>Nested table only</td>
<td>DELETE FROM TABLE (SELECT A.Skills FROM Application A WHERE A.A# = 1) N WHERE N.sname = 'programming';</td>
</tr>
</tbody>
</table>

Table 5.10 DELETE statements on relational and object-relational nested tables

Rule DM.8

Translation of UPDATE statements on relational tables to UPDATE statements on object-relational tables with single reference association.

Input:

(1) Relational tables R(P, a₁, ..., aₙ) and S(F, b₁, ..., bₙ) where P is a primary key of R, F is a foreign key of S which references the primary key P of R from a relational database

(2) Object-Relational tables T(a₁, ..., aₙ) and V(b₁, ..., bₙ, @Ref) obtained from the application of transformation rule R(P), S(F) → T, V (@Ref) (reference rule R.2, R.5, F.4, F.5, RF.2).
(3) DELETE statement

DELETE FROM R
WHERE 'condition';

Output:

DELETE FROM T
WHERE 'condition';

DELETE FROM V
WHERE 'condition';

EXAMPLE 5.11

<table>
<thead>
<tr>
<th>Relational tables</th>
<th>Object-Relational tables – double reference association</th>
</tr>
</thead>
<tbody>
<tr>
<td>DELETE FROM Position</td>
<td>DELETE FROM Applicant</td>
</tr>
<tr>
<td>WHERE P# = 1;</td>
<td>WHERE A.Applies.P# = 1;</td>
</tr>
<tr>
<td></td>
<td>DELETE FROM Position</td>
</tr>
<tr>
<td></td>
<td>WHERE P# = 1;</td>
</tr>
</tbody>
</table>

*Table 5.11* DELETE statements for relational and object-relational with double reference association tables.

Rule DM.9

Translation of DELETE statements on relational tables to DELETE statements on object-relational tables with double reference association.
Input:

(1) Relational tables $R(P, a_1, ..., a_n)$ and $S(F, b_1, ..., b_n)$ where $P$ is a primary key of $R$, $F$ is a foreign key of $S$ which references the primary key $P$ of $R$ from a relational database

(2) Object-Relational tables $T(a_1, ..., a_n, N(\text{Ref}))$ and $V(b_1, ..., b_n, (@\text{Ref})$ obtained from the application of transformation rule $R(P), S(F) \rightarrow T(N(@\text{Ref})), V(@\text{Ref})$ (reference rule R.2, R.5, F.4, F.5, RF.2).

(3) DELETE statement

```
DELETE FROM R
WHERE 'condition';
```

Output:

```
DELETE FROM T
WHERE 'condition';
```

Nested tables only

```
DELETE FROM TABLE
(SELECT N
FROM T
WHERE 'condition') U
WHERE U.a_i = v_i;
```
### EXAMPLE 5.12

<table>
<thead>
<tr>
<th>Relational tables</th>
<th>Object-Relational tables – single reference association</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>DELETE FROM Position WHERE P# = 1;</code></td>
<td><code>DELETE FROM Applicant WHERE A.Applies.P# = 1;</code></td>
</tr>
<tr>
<td></td>
<td><code>DELETE FROM Position WHERE P# = 1;</code></td>
</tr>
<tr>
<td></td>
<td>Nested table only</td>
</tr>
<tr>
<td></td>
<td><code>DELETE FROM TABLE (SELECT A.Skills FROM Application A WHERE A.A# = 1) N</code></td>
</tr>
<tr>
<td></td>
<td><code>WHERE N.sname = 'programming';</code></td>
</tr>
</tbody>
</table>

*Table 5.12 DELETE statements for relational and object-relational with single reference association tables.*

### 5.3 DATA RETRIEVAL STATEMENTS

#### 5.3.1 TRANSFORMATION RULES

The following transformation rules apply to data retrieval statements SELECT.

**SELECT**

**Rule DM.10**

*Translation of SELECT (flat) statements to SELECT (nested) statements.*
Input:

(1) Relational tables $R(P_1, a_1, \ldots a_n)$ and $S(K_1, b_1, \ldots b_n)$ where $P_1$ and $K_1$ are primary keys of $R$ and $S$ respectively and $K_1$ is the foreign key of $S$ that references the primary key $P_1$ of $R$. Attributes $(b_1, \ldots, b_n)$ can be viewed as attributes to relational table $R$.

(2) Object-Relational table $T(a_1, \ldots, a_n, N(b_1, \ldots, b_n))$ obtained from the application of transformation rule $S(b_1, \ldots, b_n) \rightarrow N(b_1, \ldots, b_n)$. (reference rule F.1, F.2, RF.3)

(3) SELECT statement

```
SELECT a_1, a_2
FROM R
WHERE 'condition';
```

Output:

```
SELECT a_1, a_2
FROM T
WHERE 'condition';
```

From nested table only

```
SELECT U.a_i
FROM TABLE(SELECT N
FROM T
```

102
WHERE 'condition')U;

EXAMPLE 5.13

<table>
<thead>
<tr>
<th>Relational tables</th>
<th>Object-Relational tables - Nested tables</th>
</tr>
</thead>
<tbody>
<tr>
<td>SELECT *</td>
<td>SELECT *</td>
</tr>
<tr>
<td>FROM Applicant;</td>
<td>FROM Applicant A;</td>
</tr>
<tr>
<td>SELECT *</td>
<td>SELECT A.Skills</td>
</tr>
<tr>
<td>FROM Skills S</td>
<td>FROM Applicant A</td>
</tr>
<tr>
<td>WHERE S.A# = 1;</td>
<td>WHERE A.A# = 1;</td>
</tr>
<tr>
<td></td>
<td>Nested table only</td>
</tr>
<tr>
<td>SELECT S.sname</td>
<td>SELECT S.sname</td>
</tr>
<tr>
<td>FROM TABLE (SELECT A.Skills</td>
<td>FROM TABLE (SELECT A.Skills</td>
</tr>
<tr>
<td></td>
<td>FROM Applicant A</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 5.13 SELECT statements for relational and object-relational (nested) tables

Rule DM.11

Translation of SELECT statements on relational tables to SELECT statements on object-relational tables with single reference association.

Input:

(1) Relational tables R(P, a₁, ..., aₙ) and S(F, b₁, ..., bₙ) where P is a primary key of R, F is a foreign key of S which references the primary key P of R from a relational database
(2) Object-Relational tables \( T(a_1, \ldots, a_n) \) and \( V(b_1, \ldots, b_m, @Ref) \) obtained from the application of transformation rule \( R(P), S(F) \rightarrow T, V\ (\@Ref) \) (reference rule R.2, R.5, F.4, F.5, RF.2).

(3) \textbf{SELECT} statement

\begin{verbatim}
SELECT a_1, a_2
FROM R
WHERE 'condition';
\end{verbatim}

\textbf{Output:}

\begin{verbatim}
SELECT a_1, a_2
FROM T
WHERE 'condition';
\end{verbatim}

\begin{verbatim}
SELECT b_1, DEREF(Ref)
FROM V
WHERE 'condition';
\end{verbatim}

\textbf{EXAMPLE 5.14}

<table>
<thead>
<tr>
<th>Relational tables</th>
<th>Object-Relational tables – single reference association</th>
</tr>
</thead>
<tbody>
<tr>
<td>SELECT A.A#, P.Title FROM Applicant A, Position P WHERE A.A# = P.A#;</td>
<td>SELECT A.A#, A.Applies.Title FROM Applicant A;</td>
</tr>
<tr>
<td></td>
<td>SELECT A.A#, DEREF(Applies) FROM Applicant A;</td>
</tr>
</tbody>
</table>

\textit{Table 5.14 SELECT statements for relational and object-relational with single reference association tables}
Rule DM.12

Translation of SELECT statements on relational tables to SELECT statements on object-relational tables with double reference association.

Input:

(1) Relational tables \( R(P, a_1, \ldots, a_n) \) and \( S(F, b_1, \ldots, b_n) \) where \( P \) is a primary key of \( R \), \( F \) is a foreign key of \( S \) which references the primary key \( P \) of \( R \) from a relational database.

(2) Object-Relational tables \( T(a_1, \ldots, a_n, N(@Ref)) \) and \( V(b_1, \ldots, b_n, @Ref) \) obtained from the application of transformation rule \( R(P), S(F) \rightarrow T(N(@Ref)), V(@Ref) \) (reference rule R.2, R.5, F.4, F.5, RF.2).

(3) SELECT statement

\[
\begin{align*}
\text{SELECT } & a_1, a_2 \\
\text{FROM } & R \\
\text{WHERE } & 'condition';
\end{align*}
\]

Output:

\[
\begin{align*}
\text{SELECT } & b_1, b_2 \\
\text{FROM } & V \\
\text{WHERE } & 'condition';
\end{align*}
\]
SELECT U.N.a
FROM TABLE (SELECT N
FROM T
WHERE 'condition') U;

EXAMPLE 5.15

<table>
<thead>
<tr>
<th>Relational tables</th>
<th>Object-Relational tables – double reference association</th>
</tr>
</thead>
<tbody>
<tr>
<td>SELECT A.A#</td>
<td>SELECT A.A#</td>
</tr>
<tr>
<td>FROM Applicant A, Position P</td>
<td>FROM APPLICANT A;</td>
</tr>
<tr>
<td>WHERE A.A# = P.A#</td>
<td>SELECT APPS.Applicantref.A#</td>
</tr>
<tr>
<td>AND P.P# = 3;</td>
<td>FROM TABLE (SELECT P.Applicants</td>
</tr>
<tr>
<td></td>
<td>FROM Position P</td>
</tr>
<tr>
<td></td>
<td>WHERE P.P# = 3)</td>
</tr>
<tr>
<td></td>
<td>APPS;</td>
</tr>
</tbody>
</table>

Table 5.15 SELECT statements for relational and object-relational with double reference association tables

5.4 NESTED TABLES

There are two relational tables RT₁ and RT₂ with one-to-many association. Table RT₂ contains a multivalued attribute for the entity RT₁.

RT₁

| P | a₁ | a₂ | …… |

RT₂

* | F | b₁ | b₂ | …… |

P is a primary key (PK) in relational table RT₁
F is a foreign key (FK) in relational table RT₂

106
When relational tables \( RT_1 \) and \( RT_2 \) are translated into object-relational tables they become nested tables. Object-relational table \( ORT \) has the following structure.

\[
ORT: a_1, a_2, \ldots, N: b_1, b_2, \ldots
\]

The following relational (Figure 5.2) and object-relational nested tables (Figure 5.3) have been used in the examples to illustrate the use of SQL and O-R SQL statements.

![Figure 5.1 Relational tables Applicant and Skills](image)

![Figure 5.2 Object-Relational Nested tables Applicant](image)
<table>
<thead>
<tr>
<th>Relational Database</th>
<th>Object-Relational Database</th>
</tr>
</thead>
<tbody>
<tr>
<td>INSERT INTO 'tablename' (first_column, last_column) VALUES (first_value, last_value);</td>
<td>INSERT INTO 'tablename' VALUES (first_value,...,'nested_table_name'('nested_type'(first_value,..., last_value)));</td>
</tr>
</tbody>
</table>

**Example**

Because of cardinality constraints if there is an entry in the table SKILLS there must be an entry in table APPLICANT. But if there is an entry in APPLICANT it is not necessary to have an immediate entry into SKILLS.

INSERT INTO Applicant VALUES (1, 'John', 'Smith');

INSERT INTO Skills VALUES (1, 1, 'VB', 'expert');

If there is no immediate entry into a nested table then the nested table will have a null entry.

INSERT INTO 'tablename' VALUES (first_value,...,'nested_table_name'());

**Example**

INSERT INTO Applicant VALUES (1, 'John', 'Smith', SKILL_SET(SKILL_TYPE('VB', 'expert')));

Entry into nested table only

INSERT INTO TABLE (SELECT T.'nested_table_name' FROM 'tablename' T WHERE T.'column_name' operator VALUE) VALUES ('first_value, ..., 'last_value);
Example

```
INSERT INTO TABLE 
(SELECT A.Skills 
FROM Applicant A 
WHERE A.f_name = 'John') 
VALUES ('VB', 'expert');
```

Table 5.16. Example of INSERT statement in nested tables

**UPDATE**

<table>
<thead>
<tr>
<th>Relational Database</th>
<th>Object-Relational Database</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPDATE 'tablename'</td>
<td>UPDATE Applicant</td>
</tr>
<tr>
<td>SET 'columnname' = 'newvalue' [, 'new_column2' = 'newvalue2' …]</td>
<td>SET f_name = 'Jon'</td>
</tr>
<tr>
<td>WHERE 'columnname' operator 'value'[AND/OR 'column' operator 'value'];</td>
<td>WHERE A# = 1;</td>
</tr>
</tbody>
</table>

Example

Update data in the tables with the full or partial condition on the primary key attribute.

```
UPDATE Applicant 
SET f_name = 'Jon' 
WHERE A# = 1;
```

```
UPDATE Skills 
SET slevel = 'not expert' 
WHERE A# = 2 
AND sname = 'singing';
```

Example

```
UPDATE Applicant 
SET f_name = 'Jon' 
WHERE A# = 1;
```

Update on nested table only

```
UPDATE TABLE 
(SELECT T.'nested_table_name' 
FROM 'tablename' T 
WHERE T.'columnname' = 'value') N 
SET N.'nest_table_column_name' = 'newvalue' [, 

```
'new_nest_table_column_name2' = 'newvalue2', ...]
WHERE N.'nest_table_column_name' operator 'value' [AND/OR 'column' operator 'value'];

Example

UPDATE TABLE
  (SELECT A.Skills
   FROM Applicant A
   WHERE A.A# = 2) S
SET S.slevel = 'not expert'
WHERE S.sname = 'singing';

Update data in the table with the condition on the non-key attribute.
Update skills level to expert for all applicants with the first name John.

SET AUTOPRINT ON
DECLARE
EMP_REC VARCHAR(40);
BEGIN
  FOR EMP_REC IN
    (SELECT f_name
     FROM Applicant
     WHERE f_name = 'John') LOOP
    UPDATE SKILLS
    SET slevel = 'expert';
  END LOOP;
  COMMIT;
END;
/

Table 5.1. Example of UPDATE statement in nested tables

DELETE
<table>
<thead>
<tr>
<th>Relational Database</th>
<th>Object-Relational Database</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DELETE FROM 'tablename'</strong></td>
<td><strong>DELETE FROM 'tablename'</strong></td>
</tr>
<tr>
<td><strong>WHERE 'columnname' operator 'value'</strong></td>
<td><strong>WHERE 'columnname' operator 'value'</strong></td>
</tr>
<tr>
<td>[AND/OR 'column' operator 'value'];</td>
<td>(SELECT T.'columnname' FROM 'tablename' T, TABLE (T.'nest_table_name') N</td>
</tr>
<tr>
<td><strong>DELETE FROM 'tablename'</strong></td>
<td>WHERE N.'nest_table_column_name' operator 'value');</td>
</tr>
<tr>
<td><strong>WHERE 'columnname' IN</strong></td>
<td></td>
</tr>
<tr>
<td>(SELECT T.'columnname' FROM 'tablename' T, TABLE (T.'nest_table_name') N)</td>
<td></td>
</tr>
<tr>
<td><strong>WHERE N.'nest_table_column_name' operator 'value');</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Example</strong></td>
<td><strong>Example</strong></td>
</tr>
<tr>
<td><strong>SET AUTOPRINT ON</strong></td>
<td><strong>DELETE FROM Applicant</strong></td>
</tr>
<tr>
<td><strong>DECLARE</strong></td>
<td><strong>WHERE f_name IN</strong></td>
</tr>
<tr>
<td><strong>EMP_REC VARCHAR(40);</strong></td>
<td>(SELECT A.f_name FROM Applicant A, TABLE (A.Skills) S</td>
</tr>
<tr>
<td><strong>BEGIN</strong></td>
<td>WHERE S.sname = 'C');</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>FOR EMP_REC IN</strong></td>
<td><strong>DELETE FROM Applicant</strong></td>
</tr>
<tr>
<td>(SELECT A.f_name FROM Applicant A, TABLE (A.Skills) S</td>
<td><strong>WHERE f_name IN</strong></td>
</tr>
<tr>
<td><strong>WHERE S.sname = 'C'</strong>)</td>
<td>(SELECT A.f_name FROM Applicant A, TABLE (A.Skills) S</td>
</tr>
<tr>
<td><strong>LOOP</strong></td>
<td>WHERE S.sname = 'C');</td>
</tr>
<tr>
<td><strong>DELETE FROM Applicant;</strong></td>
<td></td>
</tr>
<tr>
<td><strong>END LOOP;</strong></td>
<td></td>
</tr>
<tr>
<td><strong>COMMIT;</strong></td>
<td></td>
</tr>
<tr>
<td><strong>END;</strong></td>
<td></td>
</tr>
<tr>
<td>/</td>
<td><strong>Delete from nested table only</strong></td>
</tr>
<tr>
<td><strong>DELETE FROM TABLE</strong></td>
<td><strong>DELETE FROM TABLE</strong></td>
</tr>
<tr>
<td>(SELECT T.'nest_table_name' FROM 'tablename' T</td>
<td>(SELECT T.'nest_table_name' FROM 'tablename' T</td>
</tr>
<tr>
<td>**WHERE T.'columnname' operator 'value') N</td>
<td>WHERE T.'columnname' operator 'value') N</td>
</tr>
<tr>
<td><strong>WHERE N.'nest_table_column_name' operator 'value');</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Example</strong></td>
<td><strong>Example</strong></td>
</tr>
<tr>
<td><strong>DELETE FROM Skills</strong></td>
<td><strong>DELETE FROM TABLE</strong></td>
</tr>
<tr>
<td><strong>WHERE A# =1</strong></td>
<td>(SELECT A.Skills FROM Application A</td>
</tr>
<tr>
<td><strong>AND sname = 'C';</strong></td>
<td>WHERE A.A# = 1) N</td>
</tr>
<tr>
<td></td>
<td>WHERE N.sname = 'C';</td>
</tr>
<tr>
<td><strong>Table 5.18. Example of DELETE statement in nested tables</strong></td>
<td></td>
</tr>
</tbody>
</table>
5.5 SINGLE REFERENCE ASSOCIATION

There are two relational tables RT₁ and RT₂ with the following table structure.
There are two object-relational tables ORT₁ and ORT₂ with the following table structure.

The following relational (Figure 5.4) and object-relational tables (Figure 5.5) have been used in the examples to illustrate the use of SQL and O-R SQL statements. Pointer references are denoted as @.

### Applicant

<table>
<thead>
<tr>
<th>A#</th>
<th>F_name</th>
<th>L_name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>John</td>
<td>Smith</td>
</tr>
<tr>
<td>2</td>
<td>Mary</td>
<td>Brown</td>
</tr>
<tr>
<td>3</td>
<td>Gary</td>
<td>Locket</td>
</tr>
<tr>
<td>4</td>
<td>Bob</td>
<td>White</td>
</tr>
</tbody>
</table>

### Position

<table>
<thead>
<tr>
<th>A#</th>
<th>P#</th>
<th>Title</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>Database Administrator</td>
<td>90000</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>Web Designer</td>
<td>85000</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>Cook</td>
<td>40000</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>Driver</td>
<td>40000</td>
</tr>
</tbody>
</table>

*Figure 5.3 Relational tables Applicant and Position*
Table 5.4 Object-Relational tables Applicant and Position

<table>
<thead>
<tr>
<th>Applicant</th>
<th>Position</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mary Brown</td>
<td>Web Designer</td>
<td>85000</td>
</tr>
<tr>
<td>Gary Locket</td>
<td>Cook</td>
<td>40000</td>
</tr>
<tr>
<td>Bob White</td>
<td>Driver</td>
<td>40000</td>
</tr>
</tbody>
</table>

Figure 5.4 Object-Relational tables Applicant and Position

**INSERT**

<table>
<thead>
<tr>
<th>Relational Database</th>
<th>Object-Relational Database</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Example</th>
<th>Example</th>
</tr>
</thead>
</table>
| INSERT INTO Position
VALUES (1, 1, 'Database Administrator', 90000); | First, insert the objects with no references (links). |
| INSERT INTO Applicant
VALUES (1, 'John', 'Smith'); | INSERT INTO Position
VALUES (1, 'Database Administrator', 90000); |
| Then set the values of reference attributes |
| INSERT INTO Applicant
VALUES (1, 'John', 'Smith', NULL); | Another way to insert data and link the objects. |
| INSERT INTO Applicant
(SELECT 1, 'John', 'Smith', REF (P)
FROM Position P
WHERE P.P# = 2); | |

Table 5.20. Example of INSERT statements in single reference associations
UPDATE

<table>
<thead>
<tr>
<th>Relational Database</th>
<th>Object-Relational Database</th>
</tr>
</thead>
</table>

UPDATE 'tablename'
SET 'columnname' = 'newvalue' [, 'new_column2' = 'newvalue2' ...]
WHERE 'columnname' operator 'value'[AND/OR 'column' operator 'value'];

Example

UPDATE Applicant
SET f_name = 'Jon'
WHERE A# = 1;

UPDATE Position
SET salary = 85000
WHERE title = 'Database Administrator';

To set the references

UPDATE 'tablenameR'
SET 'Ref_column_name' = (SELECT REF (T)
FROM 'tablename' T
WHERE T.'columnname' operator 'value')
WHERE 'columnnameR' operator 'value';

UPDATE Applicant
SET App = (SELECT REF(P)
FROM Position P
WHERE P.P# = 3)
WHERE A# = 1;

Table 5.21. Example of UPDATE statements in single reference associations

DELETE
### DELETE FROM 'tablename'
**WHERE 'columnname' operator 'value' [AND/OR 'column' operator 'value']:**

**Example**

```
DELETE FROM Position
WHERE P# = 1;
```

In order to execute the above statement, it is first necessary to delete it from another table that has all the references to the mentioned position.

```
DELETE FROM Applicant
WHERE A.App.P# = 1;
```

**Table 5.22. Example of DELETE statements in single reference associations**

### SELECT

**Relational Database**

```
SELECT 'column1', 'column2', etc
FROM 'tablename'
[WHERE 'condition'];
```

**Example**

```
SELECT A.A#, P.Title
FROM Applicant A, Position P
WHERE A.A# = P.A#;
```

**Object-Relational Database**

```
SELECT A.A#, A.App.Title
FROM Applicant A;
```

Select numbers of applicants and the titles of the positions they apply for

Select numbers of applicants and the titles of the positions they apply for

---

116
Find the positions applied by each applicant

```sql
SELECT A.A#, P.Title, P.Salary
FROM Applicant A, Position P
WHERE A.A# = P.A#
```

Find the positions applied by each applicant

```sql
SELECT A.A#, DEREF(App)
FROM Applicant A;
```

*Table 5.23. Example of SELECT statements in single reference associations*

### 5.6 DOUBLE REFERENCE ASSOCIATION

There are two relational tables $RT_1$ and $RT_2$ with the following table structure:

<table>
<thead>
<tr>
<th>$RT_1$</th>
<th>$RT_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_1$</td>
<td>$P_2$</td>
</tr>
<tr>
<td>$a_1$</td>
<td>$F$</td>
</tr>
<tr>
<td>$a_2$</td>
<td>$b_1$</td>
</tr>
</tbody>
</table>

$P_1$ is a primary key (PK) in relational table $RT_1$
$F$ is a foreign key (FK) in relational table $RT_2$
$P_2$ is a primary key (PK) in relational table $RT_2$

There are two object-relational tables $ORT_1$ and $ORT_2$ with the following table structure.

| $ORT_1$: $a_1$, $a_2$, ... | $ORT_2$: $b_1$, ... |

The following relational (Figure 5.6) and object-relational tables (Figure 5.7) have been used in the examples to illustrate the use of SQL and O-R SQL statements. Pointer references are denoted as @.
Applicant

<table>
<thead>
<tr>
<th>A#</th>
<th>F_name</th>
<th>L_name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>John</td>
<td>Smith</td>
</tr>
<tr>
<td>2</td>
<td>Mary</td>
<td>Brown</td>
</tr>
<tr>
<td>3</td>
<td>Gary</td>
<td>Locket</td>
</tr>
<tr>
<td>4</td>
<td>Bob</td>
<td>White</td>
</tr>
</tbody>
</table>

Position

<table>
<thead>
<tr>
<th>A#</th>
<th>P#</th>
<th>Title</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>Database Administrator</td>
<td>90000</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>Web Designer</td>
<td>85000</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>Web Designer</td>
<td>85000</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>Driver</td>
<td>40000</td>
</tr>
</tbody>
</table>

Figure 5.5 Relational tables Applicant and Position

Applicant

<table>
<thead>
<tr>
<th>A#</th>
<th>F_name</th>
<th>L_name</th>
<th>@app</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>John</td>
<td>Smith</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Mary</td>
<td>Brown</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Gary</td>
<td>Locket</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Bob</td>
<td>White</td>
<td></td>
</tr>
</tbody>
</table>

Position

<table>
<thead>
<tr>
<th>@pos</th>
<th>P#</th>
<th>Title</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>Database Administrator</td>
<td>90000</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>Web Designer</td>
<td>85000</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>Driver</td>
<td>40000</td>
</tr>
</tbody>
</table>

Figure 5.6 Object-Relational tables Applicant and Position featuring single reference association.

INSERT

<table>
<thead>
<tr>
<th>Relational Database</th>
<th>Object-Relational Database</th>
</tr>
</thead>
<tbody>
<tr>
<td>INSERT INTO ‘tablename’ (first_column, ...., last_column)</td>
<td>INSERT INTO ‘tablename’ VALUES (first_value, ....)</td>
</tr>
</tbody>
</table>
VALUES (first_value, ..., last_value);

Example

INSERT INTO Position
VALUES (1, 1, 'Database Administrator', 90000);

INSERT INTO Applicant
VALUES (1, 'John', 'Smith');

Example

Insert objects without links

INSERT INTO Position
VALUES (1, 'Database Administrator', 90000, APPLICANTREF_SET());

INSERT INTO Applicant
VALUES (1, 'John', 'Smith', NULL);

Linking the object

INSERT INTO TABLE (SELECT T.'nested_table_name'
FROM 'tablename' T
WHERE T.'columnname' operator 'value')
(SELECT REF (TR)
FROM 'tablenameR' TR
WHERE TR.'columnname' operator 'value');

INSERT INTO TABLE
(SELECT P.Applicants
FROM Position P
WHERE P.P# = 3)(SELECT REF(A)
FROM Applicant A
WHERE A.A# = 1);

Table 5.24. Example of INSERT statements in double reference associations

UPDATE

<table>
<thead>
<tr>
<th>Relational Database</th>
<th>Object-Relational Database</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

119
UPDATE 'tablename'
SET 'columnname' = 'newvalue' [, 'new_column2' = 'newvalue2' ...]
WHERE 'columnname' operator 'value' [AND/OR 'column' operator 'value'];

Example

UPDATE Applicant
SET f_name = 'Jon'
WHERE A# = 1;

UPDATE Position
SET salary = 85000
WHERE title = 'Database Administrator';

Table 5.25 Example of UPDATE statements in double reference associations

DELETE

Table 5.25 Example of UPDATE statements in double reference associations

DELETE FROM 'tablename'
WHERE 'columnname' operator 'value' [AND/OR 'column' operator 'value'];
Example

DELETE FROM Position
WHERE P# = 1;

In order to execute the above statement, it is first necessary to delete it from another table that has all the references to the mentioned position.

DELETE FROM Applicant
WHERE A.App.P# = 1;

Delete from nested table only

DELETE FROM TABLE
(SELECT T.'nest_table_name'
FROM 'tablename' T
WHERE T.'columnname' operator 'value') N
WHERE N.'nest_table_column_name'
operator 'value';

Example

DELETE FROM TABLE
(SELECT A.Pos
FROM Position P
WHERE P.P# = 1) N;

Table 5.26 Example of DELETE statements in double reference associations

SELECT

<table>
<thead>
<tr>
<th>Relational Database</th>
<th>Object-Relational Database</th>
</tr>
</thead>
<tbody>
<tr>
<td>SELECT 'column1', 'column2', etc] FROM 'tablename' [WHERE 'condition'];</td>
<td></td>
</tr>
</tbody>
</table>
Select numbers of applicants and the titles of the positions they apply for

<table>
<thead>
<tr>
<th>SQL 1</th>
<th>SQL 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>SELECT A.A#, P.Title FROM Applicant A, Position P WHERE A.A# = P.A#;</td>
<td>SELECT APPS.Applicantref.A#, APPS.Applicantref.App.Titles FROM TABLE (SELECT P.Pos FROM Position P WHERE P.Title = 'Database Administrator') APPS;</td>
</tr>
</tbody>
</table>

Find the numbers of applicants for position number 3.

<table>
<thead>
<tr>
<th>SQL 1</th>
<th>SQL 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>SELECT A.A# FROM Applicant A, Position P WHERE A.A# = P.A# AND P.P# = 3;</td>
<td>SELECT APPS.Applicantref.A# FROM TABLE (SELECT P.Pos FROM Position P WHERE P.P# = 3) APPS;</td>
</tr>
</tbody>
</table>

Find all other applicants who apply for the same position as applicant 1.

<table>
<thead>
<tr>
<th>SQL 1</th>
<th>SQL 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>SELECT Applicant.A# FROM Applicant A, Position P WHERE A.A# = P.A# AND A.A# = 1;</td>
<td>SELECT APPS.Applicantref.A# FROM TABLE (SELECT A.Applies.Pos FROM Applicant A WHERE A.A# = 1) APPS WHERE APPS.Applicantref.A# &lt;&gt; 1;</td>
</tr>
</tbody>
</table>

Table 5.27 Example of SELECT statements in double reference associations

## 5.7 TRANSLATION OF DATABASE APPLICATIONS

The idea behind the translation of Database Application is such that for example there is in existence a program written in any language, such as in Java, C/C++ or PL/SQL. This program has some embedded SQL statements that access a Relational Database. SQL statements perform some query manipulation with data sets and
have some data as the result of the query. The result of the query is always in tabular format, and the rows of results are fetched (retrieved), either a row at a time or in groups.

After the relational database structure has been transformed into object-relational database structure next step is to translate existing SQL statements embedded into some Database Application, Figure 5.7.

![Figure 5.7 Replacement of SQL statements with OR-SQL statements](image)

Those relational SQL statements are need to be replaced with relevant O-R SQL statements that the result from the query manipulation with data sets return the same result as from SQL statements on relational database. Input into relational SQL statements and input into O-R SQL statements are the same and both types of SQL statements must produce the same output results.
Chapter 6
Data Migration

The final step in translation of Relational Database into Object-Relational Database is to migrate all data from one database into another.

This process could be done by using INSERT SQL statements to migrate data and a series of INSERT and UPDATE SQL statements to set up reference links between object-relational tables with objects. The following steps are required to perform to migrate data from one database into another, Figure 6.1.

- Generate INSERT statements from relational database, if the original relational database scripts are unavailable.

- Transform them into O-R SQL INSERT statements using rules described in the previous chapter

- INSERT data into object-relational database
• Use `INSERT` and `UPDATE` statements to set up reference links between object-relational tables with objects.

As seen from the previous chapter, all relational tables when transformed into object-relational tables have one or few of the following, due to structural changes.

- Single reference associations
- Double reference associations
- Nested tables

**EXAMPLE 6.1**

There are two relational tables `APPLICANT` and `SKILL_POSSESSED`.

APPLICANT (A#, FNAME, LNAME, PHONE#);
SKILL_POSSESSED (A#, SNAME);

<table>
<thead>
<tr>
<th>Applicant</th>
<th>Skill_posessed</th>
</tr>
</thead>
<tbody>
<tr>
<td>A#</td>
<td>Fname</td>
</tr>
<tr>
<td>0001</td>
<td>Peter</td>
</tr>
<tr>
<td>002</td>
<td>Jonh</td>
</tr>
<tr>
<td>003</td>
<td>Mary</td>
</tr>
</tbody>
</table>

According to transformation rule (see rules R.6, F.1, F.2, RF.3), one can transform relational tables `APPLICANT` and `SKILL_POSSESSED` into object-relational
nested tables APPLICANT and SKILLS. The following script creates object-relational tables with objects APPLICANT and nested table SKILLS.

```
CREATE TYPE SKILL_TYPE AS OBJECT(
    SNAME VARCHAR(30));
/
CREATE TYPE SKILL_SET AS TABLE OF SKILL_TYPE;
/
CREATE TABLE APPLICANT (
    A# NUMBER(7),
    fname VARCHAR (20),
    lname VARCHAR (30),
    phone# NUMBER (10),
    SKILLS SKILL_SET) /*Nested table skills*/
NESTED TABLE SKILLS STORE AS NESTED_SKILLS;
```

Transform INSERT statements generated from the relational database into O-R SQL INSERT statements using rules described in chapter 5 (rule DM.1).

The following INSERT statements insert values into relational table APPLICANT.

```
INSERT INTO Applicant VALUES (000001, 'PETER', 'JONES', 645278453);
INSERT INTO Applicant VALUES (000002, 'JOHN', 'BLACK', 63569784);
INSERT INTO Applicant VALUES (000003, 'MARY', 'WHITE', 62389541);
```

Transform them into object-relational INSERT statements that insert values into object-relational table APPLICANT.

```
INSERT INTO Applicant VALUES (000001, 'PETER', 'JONES', 645278453,skill_set());
INSERT INTO Applicant VALUES (000002, 'JOHN', 'BLACK', 63569784,skill_set());
INSERT INTO Applicant VALUES (000003, 'MARY', 'WHITE', 62389541,skill_set());
```

The following INSERT statements insert values into relational table SKILL_POSSESSED
INSERT INTO SPossessed VALUES ( 000001, 'JAVA PROGRAMMING');
INSERT INTO SPossessed VALUES ( 000002, 'JAVA PROGRAMMING');
INSERT INTO SPossessed VALUES ( 000003, 'C++ PROGRAMMING');
INSERT INTO SPossessed VALUES ( 000003, 'JAVA PROGRAMMING');

Transform them into object-relational INSERT statements that insert values into nested table SKILLS. For easy handling of INSERT statements create a PL/SQL script that executes INSERT statements in a loop.

/*Insert values into nested table SKILLS*/
SET AUTOPRINT ON
DECLARE
  EMP_REC VARCHAR2(40);
BEGIN
  FOR EMP_REC IN (SELECT A#
                   FROM APPLICANT)
  LOOP
    INSERT INTO TABLE (SELECT A.SKILLS
                        FROM APPLICANT A
                        WHERE A.A# = EMP_REC.A#)
                        (SELECT SNAME FROM SPOSSESSED
                         WHERE A# = EMP_REC.A#);
  END LOOP;
  COMMIT;
END;
/

127
Chapter 7
Conclusion

7.1 CONCLUSION

The aim of this work is to develop a methodology for migration from Relational Database Technology to Object-Relational Database Technology. The proposed methodology describes how to migrate in great details with comprehensive examples.

The basis for proceeding with this thesis arose from an experiment to establish whether an Object-Relational database provides greater results in comparison to a Relational database with an increasing number of rows as an input into queries, Figure 7.1.

For more detailed explanations and more examples refer to Appendix A

The methodology itself contains three major parts: (1) transformation of Database structures, (2) translation of Database applications, (3) Data migration.

1. Transformation of Database structures is done in two stages. First by the transformation of relational schema is put into a conceptual (UML) model; and by second from a conceptual (UML) model to object-relational schema. Each stage has transformation rules with comprehensive examples. In addition to
rules, the thesis describes how to choose single or double reference associations between Object-Relational tables depending on the frequency of application access and direction of traversal from one table to another.

![Performance comparison](image)

*Figure 7.1 Performance comparison, relational v object-relational*

2. Translation of Database application describes how to translate Data Definition and Data Manipulation relational SQL statements into object-relational (O-R) SQL statements. Translation rules cover statements that change the database state as well as statements that change the state of variables.

3. The Data Migration chapter covers how to migrate data from a relational database to an object-relational database. This process could be done by using INSERT SQL statements to migrate data and series of INSERT and UPDATE
SQL statements to set up reference links between object-relational tables with objects.
Bibliography


[10]. Elsenberg, A Melton, J *SQL-1999, formally known as SQL3*, SIGMOND Record, vol 28 no.1 1999


[18]. ORACLE8i SQL Reference. Release 2 (8.1.6), December 1999


[20]. Predator Object-Relational Database. www.distlab.dk/predator


[22]. Soutou C., Modeling relationships in object-relational databases. Data & Knowledge Engineering, vol 36, issue 1, 2001

Appendix A
Performance comparison

This experiment has been conducted to prove that an Object-Relational Database provides a better performance in comparison to a Relational Database based on the query running time.

All programming has been done using Oracle 9i software and relational table CENSUS, which contains 99,762 rows, as data source.

1. OBJECT-RELATIONAL TABLES v RELATIONAL TABLES

Table CENSUS has the structure of a relational table. It has 99,762 rows and 42 columns of data. Data content is the information obtained within the US during the annual census of population in each particular state.

1.1. OBJECT-RELATIONAL TABLES

There is a need to establish which object-relational design of tables is more effective during queries. For this purpose the time from queries performance on two object-relational designs has been compared: tables with objects and tables with objects with single reference association.
The following script generates the creation of table AGE that has 91 rows and one column containing values of age; and table AGE_EDU that has 30,000 rows and six columns with values of education, age, sex, industry_code, occupation_code, citizenship.

```
CREATE TYPE AGE_TYPE AS OBJECT(
    AGE NUMBER (3));
/
CREATE TYPE AGE_EDU_TYPE AS OBJECT(
    EDUCATION VARCHAR(100),
    SEX VARCHAR(15),
    INDUSTRY_CODE NUMBER(5),
    OCCUPATION_CODE NUMBER(5),
    CITIZENSHIP VARCHAR(150),
    AGE NUMBER (3));
/
CREATE TABLE AGE OF AGE_TYPE;
CREATE TABLE AGE_EDU OF AGE_EDU_TYPE;

INSERT INTO AGE (SELECT DISTINCT AGE FROM CENSUS);
INSERT INTO AGE_EDU (SELECT EDUCATION, SEX,
    INDUSTRY_CODE, OCCUPATION_CODE, CITIZENSHIP, AGE FROM CENSUS);
COMMIT;
```

The following script generated two object-relational tables with objects, and sets single reference association from one table to another using pointers. Table AGE has 91 rows and one column containing values of age. Table AGE_EDU has 30,000 rows and six columns with values of education, age, sex, industry_code, occupation_code, citizenship; and one column with reference to the table AGE.
CREATE TYPE AGE_TYPE AS OBJECT (  
    AGE NUMBER (3));
/
CREATE TYPE AGE_EDU_TYPE AS OBJECT (  
    EDUCATION VARCHAR (100),  
    AGE NUMBER (3),  
    SEX VARCHAR (15),  
    INDUSTRY_CODE NUMBER (5),  
    OCCUPATION_CODE NUMBER (5),  
    CITIZENSHIP VARCHAR (150),  
    P REF AGE_TYPE);
/
CREATE TABLE AGE OF AGE_TYPE;
CREATE TABLE AGE_EDU OF AGE_EDU_TYPE;

INSERT INTO AGE (SELECT DISTINCT AGE FROM CENSUS);

INSERT INTO AGE_EDU (EDUCATION, AGE, SEX, INDUSTRY_CODE, OCCUPATION_CODE, CITIZENSHIP) (SELECT EDUCATION, AGE, SEX, INDUSTRY_CODE, OCCUPATION_CODE, CITIZENSHIP FROM CENSUS);

UPDATE AGE_EDU  
SET P = (SELECT REF(R)  
    FROM AGE R  
    WHERE R.AGE = AGE_EDU.AG);

1.2. RELATIONAL TABLES

Two tables have been constructed using relational design. The data source is from relational table CENSUS. Table AGE has 91 rows and one column with the value
of age; the other table AGE_EDU has 30,000 rows and six columns with the value of age education, sex, occupation_code, industry_code and citizenship.

```
CREATE TABLE AGE AS (SELECT DISTINCT AGE FROM CENSUS);
```

```
CREATE TABLE AGE_EDU AS (SELECT AGE, EDUCATION, SEX, OCCUPATION_CODE, INDUSTRY_CODE, CITIZENSHIP FROM CENSUS);
```

1.3. QUERIES ON OBJECT-RELATIONAL AND RELATIONAL TABLES

Tables with objects have been submitted to the test by two queries. One is to find the number of rows where age has same values in two table AGE and AGE_EDU.

```
SELECT COUNT (*)
FROM AGE W, AGE_EDU A
WHERE W.AGE = A.AGE;
```

The second query is to find the number of rows with education where age has the same values in two tables AGE and AGE_EDU.

```
SELECT COUNT (EDUCATION)
FROM AGE W, AGE_EDU A
WHERE W.AGE = A.AGE;
```
Queries A and B are performed on the same data, but the difference is that query B selects and counts a particular value of education from the tables.

Figure A.1 illustrates the dependence of running time from the number of rows that a query returns. Results of queries are shown in Table A.1.

To understand why there is such a sizable difference in running time between queries A and B let us look at the EXPLAIN PLAN statement.

Output of EXPLAIN PLAN for query B is

```
SELECT STATEMENT
  SORT AGGREGATE
    MERGE JOIN
      SORT JOIN
        TABLE ACCESS FULL AGE_EDU
      SORT JOIN
        TABLE ACCESS FULL AGE
```

Output of EXPLAIN PLAN for query A is

```
SELECT STATEMENT
  MERGE JOIN
    SORT JOIN
      TABLE ACCESS FULL AGE_EDU
    SORT JOIN
      TABLE ACCESS FULL AGE
```
When both outputs are compared it is clear that the access path used by query A is not only shorter but it does not have the `SORT AGGREGATE` statement which sorts all the rows in the combined output for the second time. So, it allows query A to run up to twice as faster as query B.

A table with objects with single reference association has been submitted to the following query

```sql
SELECT COUNT (DEREF (R.P)) (C)
FROM AGE_EDU R;
```

If `EXPLAIN PLAN` is performed for the above query the result would be

```
SELECT STATEMENT
  SORT AGGREGATE
  TABLE ACCESS FULL_AGE
```

As seen from the comparison with `EXPLAIN PLAN` for the table with objects shown previously, `EXPLAIN PLAN` for the table with objects referenced by pointers has only a few statements, and therefore the performance of the query is faster.

As seen from the diagram in Figure A.1 and the information in Table A.1 it is clear that the best performance time is produced by tables with objects referenced by pointers. It is the best object-relational design.
The next step is the necessity to find out either relational or object-relational designs is suit the most for queries performance.
Next, the results are compared from queries processed on tables with objects referenced by pointers and results from queries processed on relational tables.

Figure A.2 illustrates the dependence of running time from the number of rows that the queries returns. Results of queries are shown in Table A.2.

The relational table has been submitted to the following query

```
SELECT COUNT (*)
FROM AGE A, AGE_EDU B
WHERE A.AGE = B.AGE;
```

EXPLAIN PLAN for the above query is the same as for the query A performed on tables with objects.

```
SELECT STATEMENT
  MERGE JOIN
  SORT JOIN
  TABLE ACCESS FULL AGE_EDU
  SORT JOIN
  TABLE ACCESS FULL AGE
```
After comparing all the results from queries (Table A.2), EXPLAIN PLAN statements and diagram of dependence of running time from number of rows (Figure A.2), it is clear that the most effective database table design is object-relational one in particular a table with objects referenced by pointers.
2. NESTED OBJECT-RELATIONAL TABLES v RELATIONAL TABLES

2.1. NESTED TABLES STRUCTURE

Object-relational nested tables have two columns. One column has the value of EDUCATION and another column has nested table AGE. The design of the nested table is as follows:

<table>
<thead>
<tr>
<th>EDUCATION</th>
<th>AGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>10th grade</td>
<td>21</td>
</tr>
<tr>
<td></td>
<td>22</td>
</tr>
<tr>
<td>11th grade</td>
<td>22</td>
</tr>
<tr>
<td></td>
<td>25</td>
</tr>
</tbody>
</table>

/*Create table edu*/
CREATE TABLE EDU AS
(SELECT DISTINCT EDUCATION FROM CENSUS);

/*Creating an table and inserting it into edu as a nested table*/
CREATE TYPE CENSUS_AGE AS OBJECT
    (NAGE NUMBER (3));
/
CREATE TYPE CENSUS_AGE_TABLE AS TABLE OF CENSUS_AGE;
/
ALTER TABLE EDU ADD (CAGE_TABLE CENSUS_AGE_TABLE)
NESTED TABLE CAGE_TABLE STORE AS CAGE_NT;

/*Set an empty nested table to null*/
UPDATE EDU
SET CAGE_TABLE = CENSUS_AGE_TABLE();

/*Insert into the nested table values from the Census table according to values of education in edu table*/
INSERT INTO TABLE
(SELECT E.CAGE_TABLE FROM EDU E
WHERE E.EDUCATION LIKE '%10th grade%')
(SELECT DISTINCT AGE FROM CENSUS
WHERE EDUCATION LIKE '%10th grade%');

UPDATE EDU E
SET E.CAGE_TABLE = CAST(MULTISET
(SELECT CENSUS_AGE(AGE) FROM CENSUS
WHERE EDUCATION LIKE '%10th grade%' ORDER BY AGE)
AS CENSUS_AGE_TABLE)
WHERE EDUCATION LIKE '%10th grade%';

/*Above INSERT and UPDATE statements repeat for each value of education in EDU table*/

2.2 RELATIONAL TABLES

The first table AGE has values of distinct age (it has order from 1 to 91 with the increase of 1) which will be used by table AGE_EDU as a foreign key.

CREATE TABLE AGE AS (SELECT DISTINCT AGE FROM CENSUS);
CREATE TABLE AGE_EDU AS (SELECT AGE, EDUCATION FROM CENSUS);

2.3 QUERY ON NESTED TABLES AND RELATIONAL TABLES

To compare performance between nested tables and relational tables both relational and object-relational tables have been submitted to the following query: How many people have education to the level of a Master's degree.

Nested tables has been submitted to the following query

```
SELECT COUNT (T.NAGE)
FROM TABLE (SELECT E.CAGE_TABLE
            FROM EDU E
            WHERE EDUCATION LIKE '%Masters degree%') T;
```

Relational table has been submitted to the following query

```
SELECT COUNT (A.AGE)
FROM AGE_EDU A, AGE B
WHERE A.AGE = B.AGE
AND A.EDUCATION LIKE '%Masters degree %';
```

Results are summarized in the Table A.3 and Figure A.3
As is seen from Figure A.3, the nested tables design provides a better performance over relational tables. A performance gap is increasing with the amount of data put through for each query. When relational and object-relational tables have 30,000 rows each, the query on nested tables runs twice as fast as the query on relational tables (running time for nested tables is 0.17 sec and running time for relational tables is 0.34 sec).
tables is 0.34 sec). When each table has 480,000 rows the time gap increase to 2.27 seconds. From here the conclusion can be drawn that with the increase in the amount of data, the put through query performance of nested tables will increase significantly.
Appendix B
Example of translation Relational Database into Object-Relational Database

The following Relational Database exists.

LPTITLE (TITLE);
LSTATE (STATE);
LSKILL (SKILL);
APPLICANT (A#, FNAME, LNAME, ADDRESS, CITY, STATE, PHONE#, FAX#, EMAIL, ACOMMENT);
SPOSSESSED (A#, SNAME, SKILLLEVEL);
POSITION (P#, PTITLE, EMPLOYER, SALARY, EXTRAS, SPECIFICATION);
SNEEDED (P#, SNAME, SKILLLEVEL);
APPLIES (A#, P#, APPDATE);
COURSEPASSED (A#, CTITLE, INSTITUTION, COMPDATE, GRADE);
FEMPLOYER (A#, EMPLOYER, FROMDATE, TODATE);
REFEREE (A#, FNAME, LNAME, PHONE#);

The sample database contains information about the applicants (APPLICANT table), skills possessed by the applicants (SPOSSESSED table), courses passed by the applicants (COURSE table), referees provided by the applicants (REFEREE table), former employers (FEMPLOYER table) of the applicants, positions available for the
applicants (POSITION table), skills required for each position (SNEEDED table) and applications submitted by applicants (APPLIES table).

TRANSFORMATION OF DATABASE STRUCTURES

Based on the database source code and original conceptual schema of the relational database construct, the conceptual (UML) model is as follows.

Translation of the conceptual (UML) model into an Object-Relational Database (reference rules DD.1 – DD.4).

```
CREATE TYPE POSITION_TYPE;
/
CREATE TYPE AE_TYPE;
/
CREATE TYPE APPDATE_TYPE;
/
CREATE TYPE FEMPLOYER_TYPE AS OBJECT(
```
CREATE TYPE FEMPLOYER_SET AS TABLE OF FEMPLOYER_TYPE;

CREATE TYPE COURSE_TYPE AS OBJECT(
    CTITLE VARCHAR(40), /*Course title */
    INSTITUTION VARCHAR(40), /*Name of school*/
    COMPDATE DATE, /*Date of completion*/
    GRADE VARCHAR(30)); /*Grade */

CREATE TYPE COURSE_SET AS TABLE OF COURSE_TYPE;

CREATE TYPE REFEREE_TYPE AS OBJECT(
    FNAME VARCHAR(30), /*Referee name */
    LNAME VARCHAR(30),
    PHONE# NUMBER(9)); /*Referee phone#*/

CREATE TYPE REFEREE_SET AS TABLE OF REFEREE_TYPE;

CREATE TYPE SKILL_TYPE AS OBJECT(
    SNAME VARCHAR(30), /*Skill name */
    SLEVEL NUMBER(2)); /*Skill level */

CREATE TYPE SKILL_SET AS TABLE OF SKILL_TYPE;

CREATE TYPE APPLICANT_TYPE AS OBJECT(
    A# NUMBER(7), /*Applicant# */
    FNAME VARCHAR(30),
    LNAME VARCHAR(30),
    STREET VARCHAR(30),
    CITY VARCHAR(30),
    STATE VARCHAR(20),
    PHONE# NUMBER(9),
    FAX# NUMBER(9),
    EMAIL VARCHAR(50),
    APPLIES REF APPDATE_TYPE, /*Ref to APPDATE*/
    LISTS REFEREE_SET, /*Nested table referee*/
    PASSED COURSE_SET, /*Nested table course */
    EMP FEMPLOYER_SET, /*Nested table femployer*/
    SKILLS SKILL_SET); /*Nested table skills*/

CREATE TABLE OR_APPLICANT OF APPLICANT_TYPE
    NESTED TABLE LISTS STORE AS NESTED_REFEREE,
NESTED TABLE PASSED STORE AS NESTED_COURSE,
NESTED TABLE EMP STORE AS NESTED_FEMPLOYER,
NESTED TABLE SKILLS STORE AS NESTED_SKILLS;

CREATE TYPE AE_TYPE AS OBJECT(
    APPDATE DATE,
    P# NUMBER(8),
    A# NUMBER(8),
    PAPPLIES REF POSITION_TYPE);

CREATE TYPE POSITIONREF_SET AS OBJECT(
    POSITIONREF REF AE_TYPE);

CREATE TYPE POSITIONREF_TYPE AS TABLE OF POSITIONREF_SET;

CREATE TYPE POSITION_TYPE AS OBJECT (
    p# NUMBER(  7 ) ,  /*Position# */
    PTITLE VARCHAR(30),  /*Title */
    EMPLOYER VARCHAR(50), /*Employer */
    SALARY NUMBER(9,2),  /*Salary */
    EXTRAS VARCHAR(30),  /*Extras */
    SPECIFICATION VARCHAR(50),  /*Specification*/
    POS POSITIONREF_TYPE,
    PSKILLS SKILL_SET);  /*Nested table pskills*/

CREATE TABLE OR_POSITION OF POSITION_TYPE
NESTED TABLE POS STORE AS NESTED_POS,
NESTED TABLE PSKILLS STORE AS NESTED_PSKILLS;

/*Object-relational table AE plays an intermediate role to
allow set up reference links to OR_POSITION*/

CREATE TABLE AE OF AE_TYPE;

CREATE TYPE APPLICANTREF_TYPE AS OBJECT (  
    APPLICANTREF REF APPLICANT_TYPE);

CREATE TYPE APPLICANTREF_SET AS TABLE OF APPLICANTREF_TYPE;

CREATE TYPE APPLICDATE_TYPE AS OBJECT (  
    P# NUMBER (8),
    APPDATE DATE,
    PAPPLIES REF POSITION_TYPE);

CREATE TYPE APPLICDATE_SET AS TABLE OF APPLICDATE_TYPE;

CREATE TYPE APPDATE_TYPE AS OBJECT (  
    A# NUMBER(8),
    APP APPLICANTREF_SET,
CREATE TABLE APPLDATE OF APPLDATE_TYPE
NESTED TABLE ADATE STORE AS NESTED_APPDATE,
NESTED TABLE APP STORE AS NESTED_APP;

DATA MIGRATION

/*Insert values into OR_APPLICANT object-relational table*/

INSERT INTO OR_APPLICANT (A#, FNAME, LNAME, STREET, CITY,
STATE, PHONE#, FAX#, EMAIL) (SELECT A#, FNAME, LNAME, ADDRESS,
CITY, STATE, PHONE#, FAX#, EMAIL FROM APPLICANT);

/*Set up values of nested table to null*/

UPDATE OR_APPLICANT
SET SKILLS = SKILL_SET();

UPDATE OR_APPLICANT
SET LISTS = REFEREE_SET();

UPDATE OR_APPLICANT
SET PASSED = COURSE_SET();

UPDATE OR_APPLICANT
SET EMP = FEMPLOYER_SET();

/*Insert values into OR_POSITION object-relational table*/

INSERT INTO OR_POSITION (P#, PTITLE, EMPLOYER, SALARY, EXTRAS) (SELECT P#, PTITLE, EMPLOYER, SALARY, EXTRAS FROM POSITION);

/*Set up values of nested table to null*/

UPDATE OR_POSITION
SET PSKILLS = SKILL_SET();

UPDATE OR_POSITION
SET POS = POSITIONREF_TYPE();

/*Insert values into AE object-relational table.*/

INSERT INTO AE(APPDATE, P#, A#) (SELECT APPDATE, P#, A# FROM APPLIES);

/*Insert values into APPDATE object-relational table*/

INSERT INTO APPDATE (A#) (SELECT A# FROM APPLICANT);

/*Set up values of nested tables to null*/

UPDATE APPDATE
SET APP = APPLICANTREF_SET();

UPDATE APPDATE
SET ADATE = APPLICDATE_SET();

/*Set up links between AE and OR_POSITION object-relational tables*/

UPDATE AE
SET PAPPLIES = (SELECT REF(P)
FROM OR_POSITION P
WHERE P.P# = AE.P#);

/*Set up links between OR_POSITION and AE object-relational tables */
SET AUTOPRINT ON
DECLARE
AUTOVETPRINT ON
DECLARE
EMP_REC VARCHAR2(40);
BEGIN
FOR EMP_REC IN (SELECT P#
FROM OR_POSITION)
LOOP
153
INSERT INTO TABLE (SELECT P.POS
FROM OR_POSITION P
WHERE P.P# = EMP_REC.P#)
(SELECT REF (A)
FROM AE A
WHERE P# = EMP_REC.P#);
END LOOP;
COMMIT;
END;
/

/*Insert values into ADATE nested table of APPDATE*/
SET AUTOPRINT ON
DECLARE
EMP_REC VARCHAR2(40);
BEGIN
FOR EMP_REC IN (SELECT DISTINCT A#
FROM AE)
LOOP
INSERT INTO TABLE (SELECT A.ADATE
FROM APPDATE A
WHERE A.A# = EMP_REC.A#)
(SELECT P#, APPDATE, PAPPLIES FROM AE
WHERE A# = EMP_REC.A#);
END LOOP;
COMMIT;
END;
/

/*Insert values into nested table SKILLS*/
SET AUTOPRINT ON
DECLARE
EMP_REC VARCHAR2(40);
BEGIN
FOR EMP_REC IN (SELECT A#
FROM OR_APPLICANT)
LOOP
INSERT INTO TABLE (SELECT A.SKILLS
FROM OR_APPLICANT A
WHERE A.A# = EMP_REC.A#)
(SELECT SNAME, SKILLLEVEL FROM SPOSSESSED
WHERE A# = EMP_REC.A#);
END LOOP;
COMMIT;
END;
/

/*Insert values into nested table PSKILLS*/
SET AUTOPRINT ON
DECLARE

EMP_REC VARCHAR2 (40);
BEGIN
  FOR EMP_REC IN (SELECT P#
                 FROM OR_POSITION)
  LOOP
    INSERT INTO TABLE (SELECT P.PSKILLS
                        FROM OR_POSITION P
                        WHERE P.P# = EMP_REC.P#)
                      (SELECT SNAME, SKILLLEVEL FROM SNEEDED
                       WHERE P# = EMP_REC.P#);
  END LOOP;
  COMMIT;
END;
/

/*Insert values into nested table LISTS*/
SET AUTOPRINT ON
DECLARE
  EMP_REC VARCHAR2(40);
BEGIN
  FOR EMP_REC IN (SELECT A#
                   FROM OR_APPLICANT)
  LOOP
    INSERT INTO TABLE (SELECT A.LISTS
                         FROM OR_APPLICANT A
                         WHERE A.A# = EMP_REC.A#)
                       (SELECT FNAME, LNAME, PHONE# FROM REFEREE
                        WHERE A# = EMP_REC.A#);
  END LOOP;
  COMMIT;
END;
/

/*Insert values into nested table PASSED*/
SET AUTOPRINT ON
DECLARE
  EMP_REC VARCHAR2(40);
BEGIN
  FOR EMP_REC IN (SELECT A#
                   FROM OR_APPLICANT)
  LOOP
    INSERT INTO TABLE (SELECT A.PASSED
                         FROM OR_APPLICANT A
                         WHERE A.A# = EMP_REC.A#)
                       (SELECT CTITLE, INSTITUTION, COMPDATE, GRADE
                        FROM COURSEPASSED
                        WHERE A# = EMP_REC.A#);
  END LOOP;
  COMMIT;
END;
/*Insert values into nested table EMP*/
SET AUTOPRINT ON
DECLARE
EMP_REC VARCHAR2(40);
BEGIN
FOR EMP_REC IN (SELECT A#
FROM OR_APPLICANT)
LOOP
    INSERT INTO TABLE (SELECT A.EMP
    FROM OR_APPLICANT A
    WHERE A.A# = EMP_REC.A#)
    (SELECT EMPLOYER, FROMDATE, TODATE FROM
    FEMPLOYER
    WHERE A# = EMP_REC.A#);
END LOOP;
COMMIT;
END;
/

/*Link objects, set references from table OR_APPLICANT to table APPDATE*/
UPDATE OR_APPLICANT
SET APPLIES = (SELECT REF(R)
    FROM APPDATE R
    WHERE R.A# = OR_APPLICANT.A#);

/*Link objects, set up references from table APPDATE to table OR_APPLICANT*/
SET AUTOPRINT ON
DECLARE
EMP_REC VARCHAR (40);
BEGIN
FOR EMP_REC IN (SELECT A#
FROM OR_APPLICANT)
LOOP
    INSERT INTO TABLE (SELECT A.APP
    FROM APPDATE A
    WHERE A.A# = EMP_REC.A#)
    (SELECT REF(B)
    FROM OR_APPLICANT B
    WHERE A# = EMP_REC.A#);
END LOOP;
COMMIT;
END;
/
The HTTP Server component of Oracle 9iAS is an ordinary web server (Apache) equipped with an additional infrastructure for the implementation of static and dynamic database applications over the web. The HTTP Server allows for the application of the languages and technologies like Java (JSP, Servlet, FastCGI), Perl (via mod_perl, CGI), C (via CGI and FastCGI), C++ (FastCGI), and of course the internal programming language of ORACLE DBMS – PL/SQL. The PL/SQL Gateway allows PL/SQL to be used as a scripting language within an HTML page. The HTTP Server provides a back-end support for traditional Web site and page development.

Consider the following 'Master-details' application that finds all courses passed by a selected applicant.

```html
<%@ page language="PL/SQL" %>
<%@ plsql procedure="courses_passed" %>
<%@ plsql parameter="anum" type="number" %>
<! appl_rec APPLICANT%ROWTYPE; %>
<% SELECT * INTO appl_rec FROM APPLICANT WHERE A# = anum; %>
<html>
<head>
<title>Courses passed</title>
</head>
<body>
<b><font face="Arial,Helvetica">
<font size=+3>&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;
Courses passed by <%= appl_rec.FNAME %> <%= appl_rec.LNAME %>
</font></font></b>
<br>
```

157
<table>
<thead>
<tr>
<th>A#</th>
<th>First name</th>
<th>Last name</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>A#</td>
<td>FNAME</td>
<td>LNAME</td>
<td>ADDRESS</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Course title</th>
<th>Institution</th>
<th>Completion date</th>
<th>Grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTITLE</td>
<td>INSTITUTION</td>
<td>COMPDATE</td>
<td>GRADE</td>
</tr>
</tbody>
</table>
| % FOR course_rec IN ( SELECT * FROM COURSEPASSED WHERE A# = anum ) LOOP %
As it has been mentioned earlier in the thesis, to translate a database application is necessary to translate relational SQL statements into O-R SQL statements.

```
SELECT *
INTO APPL_REC (A)
FROM APPLICANT
WHERE A# = ANUM;
```

Query (A) is a relational SQL query performed on the relational table APPLICANT. During transformation of the database structure, the relational table APPLICANT becomes an object-relational table OR_APPLICANT. Next, the query (A) is translated into an object-relational O-R SQL query (B), (reference rule DM.10 – DM.12).

```
SELECT *
INTO APPL_REC (B)
FROM OR_APPLICANT
WHERE A# = ANUM;
```

The following statement from the HTTP application contains a query that performs operation on the relational table COURSEPASSED (C).

```
FOR course_rec IN ( SELECT *
FROM COURSEPASSED (c)
WHERE A# = anum ) LOOP
```
During transformation of database structure relational table COURSEPASSED becomes a nested table PASSED. Next, translate relational SQL query (C) into object-relational O-R SQL query (D), rule DM.10.

\[
\begin{align*}
\text{SELECT } S.* \\
\text{FROM TABLE (SELECT A.PASSED (D) FROM OR_APPLICANT A WHERE A.A# = ANUM) S;}
\end{align*}
\]

Next, replace the relational SQL statements with the O-R SQL statements in the HTTP application.

```sql
<%@ page language="PL/SQL" %>  
<%@ plsql procedure="courses_passed" %>  
<%@ plsql parameter="anum" type="number" %>  
<%! appl_rec APPLICANT%ROWTYPE; %>
<% SELECT * INTO appl_rec FROM OR_APPLICANT WHERE A# = anum; %>
<html>
Courses passed by <%= appl_rec.FNAME %> <%= appl_rec.LNAME %>
</font></font></b>
... ...
<tr>
<td><font face="Arial, Helvetica">%= appl_rec.A# %</font></td>
<td><font face="Arial, Helvetica">%= appl_rec.FNAME %</font></td>
<td><font face="Arial, Helvetica">%= appl_rec.LNAME %</font></td>
<td><font face="Arial, Helvetica">%= appl_rec.ADDRESS %</font></td>
</tr>
</table>
... ...
</table>
<% FOR course_rec IN ( SELECT S.* FROM TABLE(SELECT A.PASSED FROM OR_APPLICANT A
```
WHERE A.A# = anum ) LOOP %>

<tr>
<td><font face="Arial, Helvetica">
<%= course_rec.CTITLE %></font></td>
<td><font face="Arial, Helvetica">
<%= course_rec.INSTITUTION %></font></td>
<td><font face="Arial, Helvetica">
<%= course_rec.COMPDATE %></font></td>
<td><font face="Arial, Helvetica">
<%= course_rec.GRADE %></font></td>
</tr>

<% END LOOP; %>
</table>

<hr WIDTH="100%">
</body>
</html>
Appendix C
Modeling Notation

The conceptual model adopts a *unified modeling language* (UML) notation [4][14].

A class is a description of a group of objects with similar properties (attributes), common behavior and common semantics.

Classes and their relationships are delineated in a class diagram in Figure C.1. Class is denoted by a box with the class name in the top portion of the box. An optional second portion of the box lists attributes.

![Diagram of a class diagram with two boxes labeled 'Employer' and 'Employer'

An association is a description of a group of links with a common structure and common semantics. An association connects related classes and is denoted by a line, Figure C.2. The association name is optional, if the model is unambiguous. Ambiguity arises when there are multiple associations between the same classes.
Multiplicity specifies the number of instances of one class that may relate to a single instance of an associated class. Figure C.3 summarizes multiplicity combinations. A star may denote 'many' multiplicity, meaning zero or more.

![Diagram of an association between Employee and Department]

**Figure C.2 An association**

**Figure C.3 Notation for multiplicity**

Link attribute is a named property of an association that describes a value held by each link of the association. Link attributes are often discovered by abstracting known values. A link attribute is denoted by a box attached to the association by a dashed line, Figure C.4. The top portion of the box may be left blank or may contain the association name. One or more link attributes may appear in the second portion of the box.
Generalization is the relationship between a class (or superclass) and one or more variations of the class (the subclass). Generalization organizes classes by their similarities and differences. The superclass holds common attributes, operations and associations; the subclass adds specific attributes, operations and association. Simple generalization organizes classes into a hierarchy: each subclass has a single immediate superclass. A large arrow denotes generalization, it points to the superclass, Figure C.5. An attribute (the discriminator) means that each one has one value for each subclass; the value indicates which subclass further describes an object.

Figure C.5 Generalization

Attribute multiplicity specifies the possible number of values for an attribute and is listed in brackets after the attribute name. Mandatory single value [1], an optional
single value [0..1] or an unbounded collection with a lower limit [1..*] can be specified. Figure A.6 presents attribute multiplicity.

<table>
<thead>
<tr>
<th>Applicant</th>
</tr>
</thead>
<tbody>
<tr>
<td>applicantname</td>
</tr>
<tr>
<td>address</td>
</tr>
<tr>
<td>skill [1..*]</td>
</tr>
</tbody>
</table>

*Figure C.6 Attribute multiplicity*

Candidate key for a class is a combination of one or more attributes that uniquely identifies objects within a class. The collection of candidate keys must be minimal: no attribute can be discarded from the candidate key without destroying uniqueness. No attribute in a candidate key can be null. Below, a candidate key for a class indicated with the notation CKn in braces next to the appropriate attributes, Figure C.7.

<table>
<thead>
<tr>
<th>Airport</th>
</tr>
</thead>
<tbody>
<tr>
<td>Airportcode {CK1}</td>
</tr>
<tr>
<td>AirportName {CK2}</td>
</tr>
</tbody>
</table>

*Figure C.7 Candidate key for a class*

The degree of association is the number of roles for each link. Association may be binary, ternary, or a higher degree. Ternary association is an association with three roles that cannot be restated as binary associations. The notation is a large diamond; each associated class connects to a vertex of the diamond with a line, Figure C.8
Figure C.8 Ternary association