Preserving Aggregation in an Object-Relational DBMS

Johanna Wenny Rahayu[1] and David Taniar[2]

[1] Department of Computer Science and Computer Engineering, La Trobe University, Australia
wenny@cs.latrobe.edu.au

[2] School of Business Systems, Monash University, PO Box 63B, Clayton, Vic 3800, Australia
David.Taniar@infotech.monash.edu.au

Abstract. *Aggregation* is an important concept in database design where composite objects can be modelled during the design of database applications. Therefore, preserving the aggregation concept in database implementation is essential. In this paper, we propose models for implementation of aggregation in an *Object-Relational Database Management System* (ORDBMS) through the use of *index clusters* and *nested tables*. ORDBMS is a commercial Relational Database Management Systems (RDBMS), like Oracle, which support some object-oriented concepts. We will also show how queries can be performed on index clusters and nested tables.

1 Introduction

Aggregation is a composition (part-of) relationship, in which a composite object ("whole") consists of other component objects ("parts") (Rumbaugh et al, 1991). Aggregation concept is a powerful tool in database design, and consequently, preserving aggregation in database implementation is essential.

Our previous work in this field was mainly in mapping object-oriented design, including mapping of aggregation of direct implementation in pure RDBMS (Rahayu, et al, 1996, 1999). The main reason was that the design must be able to be directly implemented in any commercial RDBMS without any necessary modification to the relational kernel.

Since the last few years, there is a growing trend that many RDBMS vendors (such as Oracle, DB2, Informix, and others) to include some object-oriented concepts (e.g. object type, inheritance, collection types, composite objects) in their products. This is a way of acknowledging the importance of object-oriented paradigm in database applications without sacrificing the foundation of relational theory. This new era marks the birth of *Object-Relational Database Management Systems* (ORDBMS) (Stonebraker and Moore, 1996). The term ORDBMS as in this early stage is often interpreted differently by different people. However, in this paper, we refer ORDBMS as commercial RDBMS that provides some support of object-oriented concepts, such as Oracle 9*i* (Muller, 2002).

It is the aim of this paper to propose models for preserving aggregation in ORDBMS, in particular we introduce the use of *index clusters* and *nested tables* in Oracle 9*i*.

T. Yakhno (Ed.): ADVIS 2002, LNCS 2457, pp. 1–10, 2002.

2 Background: Mapping Aggregation to Relational DBMS – Our Previous Work

Object-relational transformation methodology is to transform an object-oriented conceptual model into relational tables. Our previous work outlined in this section is the transformation methodology of aggregation concept into relational tables (Rahayu, et al, 1996, 1999). The first step in transforming an aggregation structure to relational tables is to create a table for each class in the aggregation hierarchy. The Object Identifier (OID) of each class becomes the Primary Key (PK) of the respective table. In addition to these tables, an aggregate table that stores the whole-part relationship is also created. The relationship between the "whole" and the "parts" is intermediated by the aggregate table. Since the aggregate table PK is a composite attribute comprising the PK of the "whole" and the "part" tables, the PK of the "whole" and each of the "parts" relate to the PK of the aggregate.

As an example for our illustration is a PC class which consists of several part classes: Harddisk, Monitor, Keyboard, and CPU. Figure 1 shows an example of Create Table statements in SQL to create the aggregation structure using a RDBMS.

```
CREATE TABLE PC (
   PCoid  VARCHAR(10) NOT NULL,
   <other details … >
   Primary Key (PCoid));

CREATE TABLE Aggregate (
   PCoid        VARCHAR(10) NOT NULL,
   PartOid      VARCHAR(10) NOT NULL,
   PartType     VARCHAR(20),
   Primary Key (PCoid, PartOid),
   Foreign Key (PCoid) Reference PC (PCoid)
         ON Delete Cascade
         ON Update Restrict);

CREATE TABLE Harddisk (
   HDOid  VARCHAR(10) NOT NULL,
   <other details … >
   Primary Key (HDOid));

CREATE TABLE Monitor (
   Moid   VARCHAR(10) NOT NULL,
   <other details … >
   Primary Key (Moid));

CREATE TABLE Keyboard (
   Koid   VARCHAR(10) NOT NULL,
   <other details … >
   Primary Key (Koid));

CREATE TABLE Cpu (
   CPUoid VARCHAR(10) NOT NULL,
   <other details … >
   Primary Key (CPUoid));
```

Fig. 1. Create Tables for an Aggregation

Notice from Figure 1 that apart from each table for each class, an Aggregate table is also created. The PartType attribute of table Aggregate represents the type of the part classes (i.e. Harddisk, Monitor, Keyboard, or CPU). In this example, the value of PartType attribute is the full string. It could have been encoded to, for example, one character instead of a full string, such as 'H' for 'Harddisk', 'M' for 'Monitor', and so on.

The relationship between the Aggregate table and the part tables through PK-FK relationship is not possible in the implementation, although conceptually desirable. This is because each value of PK (except NULL) must match with one of the values in each of the PKs. This is simply because PK-FK must exist between one table (FK) in one side and another table (PK) in the other side. It is not possible to have an FK in one side and multiple PKs in the other side. The only solution to this problem is to remove the PK-FK constraints between all of the "part" tables and the Aggregate table. This means PartOid in the Aggregate table is not an FK, although the value of a PartOid must be EITHER one of the PKs of the part tables. In this case, there is no referential integrity between aggregate and part tables. Further restriction imposed by insertion new parts or deletion of the whole object must be handled through stored procedures.

The referential integrity between the "whole" and the aggregate relationship (i.e. PC – Aggregate) depends on the type of the aggregation structure. If it is an *existence-independent*, the deletion is a *cascade*. That means that a deletion of a PC will cause a deletion of matching records in the aggregate table. This, however, does not cause a deletion of any related parts. If the aggregation is an *existence-dependent*, the deletion is also a *cascade*. However, the cascade operation does not have any impact on the "parts". The update operation will adopt a restrict update, as OIDs are not meant to be updated.

The cardinality of the relationship between the "whole" and the aggregate (e.g. PC and Aggregate) is *one*-to-*many*. However, for the "part" to the aggregate table relationship, the cardinality is determined by the type of aggregation structure. If the aggregation structure is exclusive (i.e. non-sharable), the cardinality is *one*-to-*one*, but if the type is non-exclusive (i.e. sharable), the cardinality if *many*-to-*one* (aggregate-part).

An aggregation hierarchy where one 'whole' class consists of several 'part' classes can be repeated to several levels on composition or aggregation. In the case of aggregation hierarchy involving multiple levels, each level is treated as one level aggregation described above. In addition to this, a composite table to capture the overall aggregation hierarchy is created. This composite table consists several attributes including WholeOid, and pairs of PartOid and PartName for each level of aggregation. The composite table structure is shown in Figure 2.

```
CREATE TABLE Composite (
   PCoid        VARCHAR(10) NOT NULL,
   PartOid_1    VARCHAR(10) NOT NULL,
   PartType_1   VARCHAR(20),
   PartOid_2    VARCHAR(10) NOT NULL,
   PartType_2   VARCHAR(20));
```

Fig. 2. Composite table to hold a multi-level aggregation

Details of our previous work on the transformation of an aggregation hierarchy (i.e. one-level and multiple-level aggregation) to relational tables (using a commercially available RDBMS) can be found in Rahayu et al (1996, 1999).

3 The Proposed Models

In this section, we propose an implementation model for aggregation in an ORDBMS. In an ORDBMS (i.e. Oracle version 9), new structures are in place to support object-oriented concept. These were not exist in the original versions of RDBMS, simply because they do not comply with the relational concepts. The structures that we are going to use to represent aggregation are particularly: (*i*) Index Clustering technique, and (*ii*) Nesting technique. In our proposed models, we will show how to preserve aggregation concepts using the above two structures.

3.1 Using Index Clustering Techniques

In this section, we utilize an *index clustering* technique available in Oracle 9*i* in order to preserve an aggregation structure. Figure 3 shows the SQL statements on how aggregation is implemented.

```
CREATE CLUSTER HD_Cluster (
   HDoid        VARCHAR2(10));

CREATE TABLE Harddisk (
   HDoid        VARCHAR2(10) NOT NULL,
   Capacity     VARCHAR2(20),
   Primary Key (HDoid))
   Cluster      HD_Cluster(HDoid);

CREATE TABLE HD_Contr (
   HDoid        VARCHAR2(10) NOT NULL,
   HD_Contr_id  VARCHAR2(10) NOT NULL,
   Description  VARCHAR2(25),
   Primary Key (HDoid, HD_Contr_id),
   Foreign Key (HDoid) References Harddisk (HDoid))
   Cluster      HD_Cluster(HDoid);

CREATE INDEX HD_Cluster_Index
   ON Cluster HD_Cluster;
```

Fig. 3. Implementation of a homogenous aggregation relationship using an *Index Cluster*

It is clear from the implementation shown in Figure 3 that the index clustering technique supports *existence dependent* type of aggregation only, where the existence of the part object is dependent upon the whole object. It is not possible to have a harddisk controller ("part" object) that does not belong to a harddisk ("whole" object). This is enforced by the existence of the cluster key in all the "part" tables.

Moreover, the example also shows a *sharable* aggregation type, where each "part" object can be owned by more than one "whole" object. For example, harddisk controller

with OID ‘HDC1’ may belong to harddisk with OID ‘HD1’ as well as harddisk ‘HD2’. Depending on the situation, the above sharable type may not be desirable. We can enforce non-sharable type of aggregation by creating a single primary key for the “part” object, and treat the cluster key as a foreign key rather than part of the primary key.

The following Figure 4 shows an implementation of the example in Figure 3 as a non-sharable type (the implementation of the cluster and the cluster index remain the same).

```
CREATE TABLE Harddisk (
   HDoid        VARCHAR2(10) NOT NULL,
   Capacity     VARCHAR2(20),
   Primary Key (HDoid))
   Cluster      HD_Cluster(HDoid);

CREATE TABLE HD_Contr (
   HDoid        VARCHAR2(10) NOT NULL,
   HD_Contr_id  VARCHAR2(10) NOT NULL,
   Description  VARCHAR2(25),
   Primary Key (HD_Contr_id),
   Foreign Key (HDoid) References Harddisk (HDoid))
   Cluster      HD_Cluster(HDoid);
```

Fig. 4. Implementation of a homogenous aggregation relationship *non-sharable* type

Each time a record is inserted into the "part" table (i.e. HD_Contr) the value of the cluster key (HDoid) is only stored once. The rows of the "whole" table (i.e. Harddisk) and the rows of the "part" table (HD_Contr) are actually stored together physically (see Figure 5). The index is created in order to enhance the performance of the cluster storage.

HDoid	Capacity	HD_Contr_id	Description
HD001	2GB	Contr11	.....
		Contr22	.....
HD002	6GB	Contr12	.....
		Contr13	.....
		Contr14	.....

Fig. 5. Physical storage of aggregation relationship using index cluster

It is also possible to use a cluster method to implement an aggregation relationship between a "whole’ object with a number of "part" objects. Figure 6 demonstrates the implementation of an aggregation between a PC with Harddisk, Monitor, Keyboard, and CPU.

```
CREATE CLUSTER PC_Cluster (
   PCoid  VARCHAR2(10));

CREATE TABLE PC (
   PCoid        VARCHAR2(10) NOT NULL,
   Type         VARCHAR2(20),
   Primary Key (PC_id))
```

```
    Cluster      PC_Cluster(PC_id);

CREATE TABLE Harddisk (
    PCoid        VARCHAR2(10) NOT NULL,
    HDoid        VARCHAR2(10) NOT NULL,
    Capacity     VARCHAR2(20),
    Primary Key (PCoid, HDoid),
    Foreign Key (PCoid) References PC (PCoid))
    Cluster      PC_Cluster(PCoid);

CREATE TABLE Monitor (
    PCoid        VARCHAR2(10) NOT NULL,
    Moid         VARCHAR2(10) NOT NULL,
    Resolution   VARCHAR2(25),
    Primary Key (PCoid, Moid),
    Foreign Key (PCoid) References PC (PCoid))
    Cluster      PC_Cluster(PCoid);

CREATE TABLE Keyboard (
    PCoid        VARCHAR2(10) NOT NULL,
    Koid         VARCHAR2(10) NOT NULL,
    Type         VARCHAR2(25),
    Primary Key (PCoid, Koid),
    Foreign Key (PCoid) References PC (PCoid))
    Cluster      PC_Cluster(PCoid);

CREATE TABLE Cpu (
    PCoid        VARCHAR2(10) NOT NULL,
    CPUoid       VARCHAR2(10) NOT NULL,
    Speed        VARCHAR2(10),
    Primary Key (PCoid, CPUoid),
    Foreign Key (PCoid) References PC (PCoid))
    Cluster      PC_Cluster(PCoid);

CREATE INDEX PC_Cluster_Index
    ON Cluster PC_Cluster;
```

Fig. 6. Implementation of an aggregation relationship with multiple "part" objects

"whole" id	"whole" attribute	"part" id	"part" attribute
PC001	.....	Harddisk01	.....
		Harddisk02	.....
		Monitor01	.....
		Keyboard01	.....
		Cpu01	.....
PC002	.....	Harddisk03	.....
		Monitor02	.....
		Keyboard02	.....
		Cpu02	.....

Fig. 7. Physical storage of multiple aggregation relationships using an index cluster technique

3.2 Using Nested Tables

Another technique for implementation of aggregation in Oracle is the use of *nested tables*. In this nesting technique, similar to clustering, the "part" information is tightly coupled with the information of the "whole" object. This actually enforces the *existence dependent* type of aggregation where the existence of a "part" object is fully dependent on the "whole" object. If the data of the "whole" object is removed, all associated "part" objects will be removed as well. Because of this reason, nested table technique is only suitable for existence dependent aggregation.

The following Figure 8 describes the link between the “whole” and the “part” table in a nesting structure, whereas Figure 9 shows the implementation of the homogenous aggregation using nested table technique.

Note that there is no concept of primary key nor integrity constraint in the “Part” nested table as shown in Figure 9. For example, if a particular harddisk controller is used by another harddisk from the “whole” table, then all the details of the harddisk controler will be written again as a separate record within the nested table.

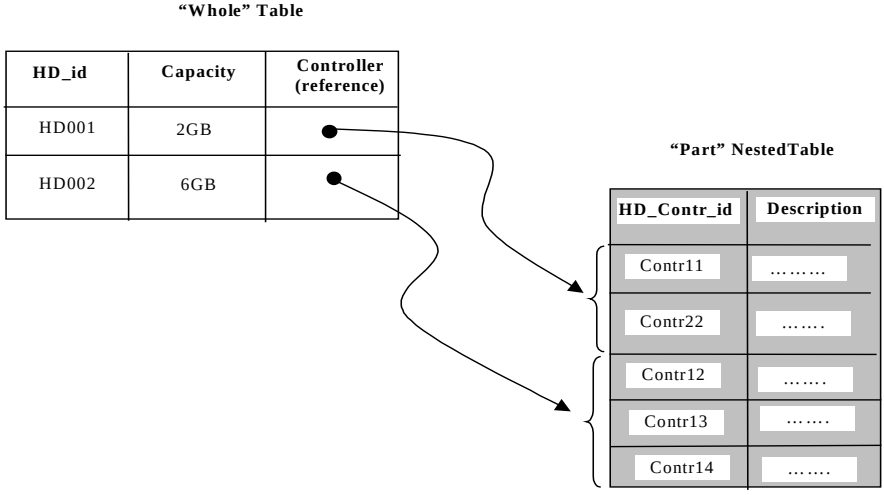

Fig. 8. Aggregation relationships using nested table

```
CREATE OR REPLACE TYPE HD_Contr AS OBJECT
    (HD_Contr_id        VARCHAR2(10),
     description        VARCHAR2(30));
/
CREATE OR REPLACE TYPE HD_Contr_Table AS TABLE OF HD_Contr
/
CREATE TABLE Hard_Disk
    (HD_id       VARCHAR2(10) NOT NULL,
    capacity     VARCHAR2(20),
    controller   HD_Contr_Table,
    Primary Key (HD_id))
    NESTED TABLE controller STORE AS HD_Contr_tab;
```

Fig. 9. Implementation of aggregation relationships using *nested table*

Figure 9 shows that we do not create a standard table for HD Controller. We only need to define a HD Controller type, and define it as a nested table later when we create the Harddisk table. It is also shown that the information of the nested table is stored externally in a table called HD_Contr_tab. This is not a standard table, in which no additional constraints can be attached to this table, and no direct access can be performed to this table without going through the Harddisk table.

Every "whole" object can own any "part" object in nesting technique, even if that particular "part" has been owned by another "whole" object. The record of the HD Controller object will simply be repeated every time a Harddisk claims to own it. This shows a sharable type of aggregation, where a particular "part" object can be shared by more than one "whole" objects.

Because there is no standard table created for HD Controller, we cannot have a primary key for the table, which we usually employ to enforce a non-sharable type of aggregation (see the previous clustering technique).

4 Queries Using Clustering and Nesting Techniques

Previous section describes our proposed model on persevering aggregation through the use of nested tables and index clusters. In this section, we are going to show how queries can be performed on these two data structures.

4.1 Queries Using Clustering Techniques

Queries on aggregation structures are normally categorized as "*Part Queries*" and "*Whole Queries*". A "Part Query" is a query on an aggregation hierarchy to retrieve information of "part" classes where the selection predicates are originated at the "whole" class. On the other hand, a "whole" query is a query to retrieve information of the "whole" class where the selection predicates are originated at the "part" class.

The SQL structure for part queries and whole queries using a clustering technique are shown in Figures 10 and 11, respectively. Note that in the clustering technique, the query to access the data along an aggregation hierarchy are simply standard queries to join the whole table with it's associated part tables.

```
SELECT <"part" class attributes>
FROM <table1, table2, …, tablen>
WHERE <join predicates>
AND <"whole" class table.attr =
    &input_class_selection_predicates>
```

Fig. 10. Part query structure using a clustering technique

```
SELECT <"whole" class attributes>
FROM <table1, table2, …, tablen>
WHERE <"part" class table.attr=
    &input_class_selection_predicates>
```

Fig. 11. Whole query structure using a clustering technique

An example of a part query is to display harddisk number and capacity of a given PC (e.g. PC ID = ‘PC001’). The SQL statement for this query is shown in Figure 12.

```
SELECT H.HDoid, H.Capacity
FROM Harddisk H, PC
WHERE H.PCoid = PC.PCoid
    AND PC.PCoid = ‘PC001’;
```

Fig. 12. Part query example using a clustering technique

An example of a whole query is to display details of PCs which as a harddisk capacity of less than 2Gb. The SQL statement for this query is shown in Figure 13.

```
SELECT PC.PCoid, PC.Type
FROM PC, Harddisk H
WHERE PC.PCoid = H.PCoid
    AND H.Capacity < 2;
```

Fig. 13. Whole query example using a clustering technique

In the clustering technique, whole queries are implemented in a very similar manner as those of part queries.

4.2 Queries Using Nesting Techniques

The query representation for part and whole queries are shown in Figures 14 and 15, respectively.

```
SELECT <”part” class attributes>
FROM THE (SELECT “whole” class nested table attribute
          FROM <”whole” class table>
          WHERE <”whole” class table.attr =
                 &input_class_selection_predicates>
```

Fig. 14. Part query structure using a nesting technique

```
SELECT <”whole” class attributes>
FROM <”whole” class table,
    TABLE (“whole” class nested table attribute)>
WHERE <”part” class table.attr=
    &input_class_selection_predicates>
```

Fig. 15. Whole query structure using a nesting technique

In nesting technique, nested tables are referred to using the keyword **THE** in the queries. Using the sample query example as shown in Figure 12, the SQL using a nested table is written as follows (see Figure 16).

```
SELECT HDoid, Capacity
FROM THE (SELECT PC.Harddisk
          FROM PC
          WHERE PC.PCoid = 'PC001');
```

Fig. 16. Part query example using a nesting technique

Again using the whole query example as shown in Figure 13, the following SQL in Figure 17 is an equivalent whole query using a nesting technique.

```
SELECT PC.PCoid, PC.Type
FROM PC, TABLE (PC.Harddisk) H
WHERE H.Capacity < 2;
```

Fig. 17. Whole query example using a nesting technique

5 Conclusions

In this paper we have shown how aggregation hierarchies in database design using an object-oriented paradigm can be preserved in the implementation using an Object-Relational Database Management System (i.e. Oracle 9*i* or above). Preserving aggregation hierarchies in the implementation stage in database development is important in order to maintain object-oriented concepts, found at the design stage, at the implementation stage.

Our proposed models make use of two unique features, namely index clusters and nested tables, for maintaining aggregation. We have shown the DDL (Data Definition Language) using these two features. We have also presented how queries can be performed on these two data structures. We illustrate aggregate queries by focusing on part queries and whole queries – two main important queries involving aggregation hierarchies.

References

1. Muller, R.J., *Oracle 9i Complete: A Comprehensive Reference to Oracle 9*, 2002.
2. Rahayu, J.W., Chang, E., Dillon, T.S., and Taniar, D., "Aggregation versus association in object modelling and databases", *Proceedings of the Australasian Conference on Information Systems*, Hobart, Tasmania, 1996.
3. Rahayu, J.W., Chang, E., and Dillon, T.S., "Composite Indices as a Mechanism for Transforming Multi-Level Composite Objects into Relational Databases", *The OBJECT Journal*, **5**(1), 1999.
4. Rahayu, J.W., Chang, E., Dillon, T.S., and Taniar, D., "Performance Evaluation of the Object-Relational Transformation Methodology", *Data and Knowledge Engineering*, **38**(3), pp. 265–300, 2001.
5. Rumbaugh, J. et al, *Object-Oriented Modelling and Design*, Prentice-Hall, 1991.
6. Stonebraker, M. and Moore, D., *Object-Relational DBMSs: The Next Great Wave*, Morgan Kaufmann, 1996.