Chapter 12, Nested Relational Processing Prototype

(from the book Advanced ANSI SQL Data Modeling and Structure Processing by Michael M David, published and copyrighted 1999 by Artech-House Publishers)

The examples in this chapter show the operation of an SQL nested relational database processor prototype that is driven by the inherent data modeling capability of the ANSI-92 Outer join. It utilizes Advanced Data Access Technologies' patented Data Structure Extraction (DSE) technology, described in Chapter 9, to dynamically extract the data structure meta information naturally present in Outer join specifications. This automatically available information is used to control the nested relational processing. This processing produces results that are semantically superior to standard SQL and are more semantically accurate. It enables the nested relational processor prototype to seamlessly conform to the underlying data structure of the SQL query request.

This automatic conforming to the data structure can not be performed with current standard nested relational processors which require that the data structure be predefined and the data stored as a fixed structure. This nested relational prototype does not require that the data be in fixed format or that the data structure be predefined. The data is stored as standard first normal form relational tables. This is a major advantage this prototype has over standard nested relational processors, giving it data and structure independence. This is in contrast with standard nested relational database systems that store their data as a pre-defined structured that greatly restricts data and structure independence.

12.1 Nested Relational Prototype Operation

Nested relational databases access and process data in non-first normal form (structured format). This eliminates having to flatten the data into first normal form (table format) as standard relational systems do. This flattening of the data can introduce unnecessary replicated data. By not having to flatten the data, nested relational processing can preserve the data structure so that all aggregate and summary operations will be accurate and can be controlled with more flexibility. This is reflected in the data structure of the displayed nested relational output in the examples shown in this chapter where a blank data field indicates that the previous column value is still in effect. Because a missing data value can inadvertently indicate that the previous column value is still in effect, missing values have a dash inserted in their column to indicate they are missing.

The first entry of each example is the Outer join specification that is processed directly by the SQL nested relational prototype. The prototype then extracts the data structure meta information embedded in the Outer join specification using the DSE technology described in Chapter 9, Section 9.1, and displays its meta data structure information in table form. This meta data structure information includes an Outer join semantic optimization indication which is flagged under its ACCESS column when a table in the

data structure does not require access. Each data structure example includes an hierarchical structure diagram to help you visualize the data structure being processed, it is not output from the prototype.

Lastly, using the data structure meta information supplied from the Outer join specification, the prototype accesses its internal first normal form (flat) relational database in a manner that will produce the structured data results shown in WYSIWYG (What You See Is What You Get) format. This nested relational processing can be implemented in any standard SQL system relying only on the data structure meta information supplied from Outer join statements.

12.2 Basic Data Modeling

The examples in Figures 12.1 and 12.2 below demonstrate the data modeling capabilities of the ANSI-92 SQL Outer join. They show how the nested relational prototype using the DSE technology can process standard relational data in a nested relational form. In these examples, three tables, Department, Employee and Dependent are joined in two different ways using the same relationships to form two different data structures involving one-to-many and many-to-one relationships. Notice in the query outputs that there is no unnecessary data replication. All the data replication counts are accurate regardless at what data structure level the data is at or if there are multiple legs in the data structure as in Figure 12.2. This allows aggregate operations applied anywhere in the data structure to be accurate. While the example in Figure 12.2 does show replicated data (HR and Acct), this correctly reflects the many-to-one data structure relationship of employee over department and its semantics (i.e. many employees have the same department). And notice further, that these replication occurrence counts are correct, in a standard relational first normal form result, HR would have been replicated three times instead of the correct two.

Besides the two different data structures in Figures 12.1 and 12.2, there is also a difference with the data values displayed or not displayed in the two examples. The first example's query output in Figure 12.1 includes a department named "MIS" while the second example doesn't. The second example's query output in Figure 12.2 includes an employee named "Irv" with a dependent named "Ben" while the first example in Figure 12.1 doesn't. These differences are properly reflected in the semantics of the data structures involved. The "MIS" department isn't included in the example's query output in Figure 12.2 because this query models an employee view (Employee over Department and Dependent) and there are no employees in the "MIS" department. The employee "Irv" and his dependent "Ben" aren't included in the first example's query output in Figure 12.1 because this query models a department view (Department over Employee over Dependent) and "Irv" and his dependent "Ben" do not belong to any known department. This was covered in Chapter 5, Section 5.1.

SELECT DeptName, EmpName, DpndName FROM Department LEFT Employee ON DeptNo=EmpDeptNo LEFT Dependent ON EmpNo=DpndEmpNo

TABLE	NAME	LEVEL	PARENT	ACCESS
1	Department	1	0	Yes
2	Employee	2	1	Yes
3	Dependent	3	2	Yes

DEPTNAME	EMPNAME	DPNDNAME
Acct	Mike	-
	John	-
HR	Mary	Jay
		Ken
	Mark	Kay
MIS	-	-

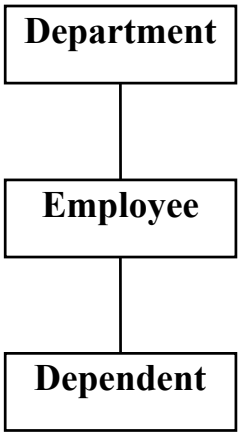

Figure 12.1 Department view processed by nested relational processor

---

SELECT EmpName, DeptName, DpndName FROM Employee LEFT Department ON DeptNo=EmpDeptNo LEFT Dependent ON EmpNo=DpndEmpNo

TABLE	NAME	LEVEL	PARENT	ACCESS
1	Employee	1	0	Yes
2	Department	2	1	Yes
3	Dependent	3	1	Yes

EMPNAME	DEPTNAME	DPNDNAME
Mike	Acct	-
John	Acct	-
Mary	HR	Jay
	-	Ken
Mark	HR	Kay
Irv	-	Ben

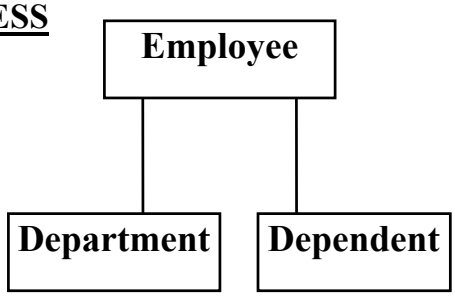

Figure 12.2 Employee view processed by nested relational processor

12.3 Many-to-many Relationships

The examples in Figures 12.4 and 12.5 operate on a Parts and Suppliers, many-to-many, relationship described in Chapter 7, Section 7.4. In this relationship, one supplier can have many parts and one part can have many suppliers. This does not present a problem for the nested relational prototype and both application views in the examples in Figures 12.4 and 12.5 produce a hierarchically structured (many-to-many) result. Most texts on data modeling state that many-to-many relationships form one-to-many hierarchical relationships. A many-to-many relationship is actually a combination of many-to-one and one-to-many. In the one-to-many portion, replications are suppressed while in the one-to-many portion they are not. In the example in Figure 12.4, Parts over Suppliers, parts are not replicated, but suppliers are (P1 occurs once while S1 occurs three times).

It is worth noting that many-to-one relationships are found naturally in the data base and do not require special considerations for processing or printing. But with one-to-many relationships, special handling considerations are needed because the data is nested and requires special consideration when processing and displaying.

Many-to-many relationships require the use of an association table as described in Chapter 7, Section 7.4. The association table used in the SQL examples in Figures 12.4 and 12.5 is PartSupplier and is also shown below in Figure 12.3. It contains keys (Part, Supplier) from both sides of the relationship to maintain the many-to-many relationship in both directions. In the example in Figure 12.4, Parts over Suppliers, the association table is hidden in the result because no column from this table is requested for display.

The example in Figure 12.5, Suppliers over Parts, does reference the association table to include the QNT column. This value is known as intersecting data, data that is meaningful at the point of intersection (i.e. the quantity of a given part from a given supplier) also explained in Chapter 7, Section 7.4. This intersecting data value (Qnt) from the association table would appear to be a value associated with the Parts table. Actually values in the association table will always appear to be a value from the lower level table.

PartSupplier Association Table

Part	Supplier	Qnt
P1	S1	100
P1	S2	200
P2	S1	150
P2	S2	300
P3	S2	350

Figure 12.3 Association table used in many-to-many relationship

SELECT PartNo, Desc, SuppNo, Addr FROM Parts LEFT PartSupplier ON PartNo=Part LEFT Suppliers ON Supplier=SuppNo

TABLE	NAME	LEVEL	PARENT	ACCESS
1	Parts	1	0	Yes
2	PartSupplier	2	1	Yes
3	Suppliers	3	2	Yes

PARTNO	DESC	SUPPNO	ADDR
P1	Part1	S1	Wash
		S2	Denv
P2	Part2	S1	Wash
		S2	Denv
P3	Part3	S1	Wash

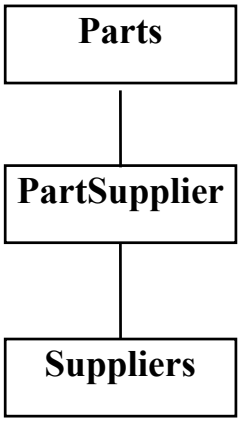

Figure 12.4 Part-supplier view processed by nested relational prototype

---

SELECT SuppNo, Addr, PartSupplier.Qnt, PartNo, Desc FROM Suppliers LEFT PartSuppliers ON SuppNo=Supplier LEFT Parts ON PartNo=Part

TABLE	NAME	LEVEL	PARENT	ACCESS
1	Suppliers	1	0	Yes
2	PartSupplier	2	1	Yes
3	Parts	3	2	Yes

SUPPNO	ADDR	QNT	PARTNO	DESC
S1	Wash	100	P1	Part1
		150	P2	Part2
		350	P3	Part3
S2	Denv	200	P1	Part1
		300	P2	Part2

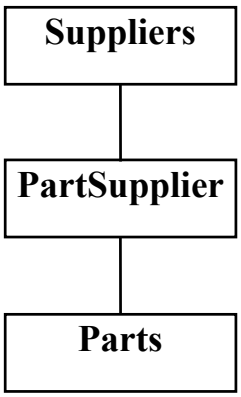

Figure 12.5 Supplier-part view processed by nested relational prototype

12.4 Embedded Views

The following example in Figure 12.6 demonstrates that stored views containing Outer join defined data structures can be seamlessly combined to form larger data structures using the same standard ANSI-92 SQL Outer join syntax already demonstrated. The nested relational prototype identifies stored queries by their view name. They are printed out when expanded as shown below in Figure 12.6. The example in Figure 12.6 uses two views shown earlier in this chapter, the Supplier view (Suppliers over Parts) and the Department view (Department Over Employee over Dependent). In this case, the Supplier view is joined over the Department view using the DeptSuppNo column in the Department table. Notice that this combined data structure properly reflects its new structure, the replication counts are accurate and the data displayed is consistent with the previously shown data structures in this chapter.

SELECT SuppNo, PartNo, DeptName, EmpName, DpndName
FROM SupplierView LEFT DepartmentView ON SuppNo=DeptSuppNo

Inserted **SupplierView:** Suppliers LEFT PartSupplier ON SuppNo=Supplier LEFT Parts ON PartNo=Part
Inserted **DepartmentView:** Department LEFT Employee ON DeptNo=EmpDeptNo LEFT Dependent ON EmpNo=DpndEmpNo

TABLE	NAME	LEVEL	PARENT	ACCESS
1	Suppliers	1	0	Yes
2	PartSupplier	2	1	Yes
3	Department	2	1	Yes
4	Parts	3	2	Yes
5	Employee	3	3	Yes
6	Dependent	4	5	Yes

SUPPNO	PARTNO	DEPTNAME	EMPNAME	DPNDNAME
S1	P1	ACCT	Mike	-
	-		John	-
	P2	HR	Mary	Jay
				Ken
	-		Mark	Kay
	P3	-	-	-
S2	P1	MIS	-	-
	P2	-	-	-

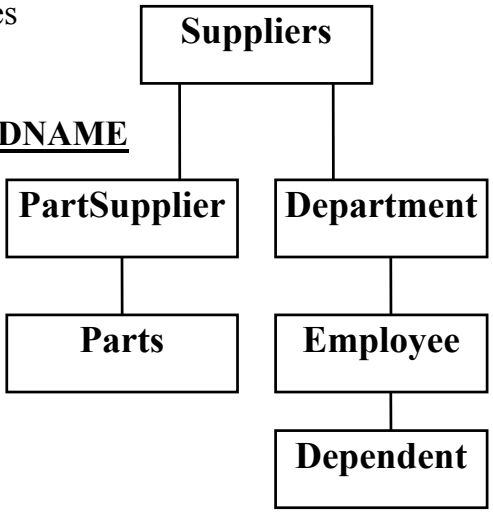

Figure 12.6 Expanded view example

12.5 View Optimization

The final example in Figure 12.7 demonstrates a powerful and very useful optimization for stored views described in detail in Chapter 11, Section 11.3. It significantly enhances the operation and usefulness of SQL's new Outer join data structure processing capability. It often happens that a stored view is used where all the tables defined are not necessary to access for the desired result. With standard Inner join views, it is always necessary that all tables in the view be accessed. This not only results in more overhead, but often incorrect results caused by accessing unneeded tables which in turn can cause replicated data values and lost data. With Outer join views, this unnecessary data access concern is not necessary and can be avoided.

SELECT SuppNo, PartNo, DeptName, EmpName, DpndName
FROM SupplierView LEFT DepartmentView ON SuppNo=DeptSuppNo

Inserted **SupplierView:** Suppliers LEFT PartSupplier ON SuppNo=Supplier LEFT Parts
ON PartNo=Part
Inserted **DepartmentView:** Department LEFT Employee ON DeptNo=EmpDeptNo
LEFT Dependent ON EmpNo=DpndEmpNo

TABLE	NAME	LEVEL	PARENT	ACCESS
1	Suppliers	1	0	Yes
2	PartSupplier	2	1	Yes
3	Department	2	1	Yes
4	Parts	3	2	Yes
5	Employee	3	3	Yes
6	Dependent	4	5	No ←

SUPPNO	PARTNO	DEPTNAME	EMPNAME
S1	P1	ACCT	Mike
	-		John
	P2	HR	Mary
	-	-	Mark
	P3	-	-
S2	P1	MIS	-
	P2	-	-

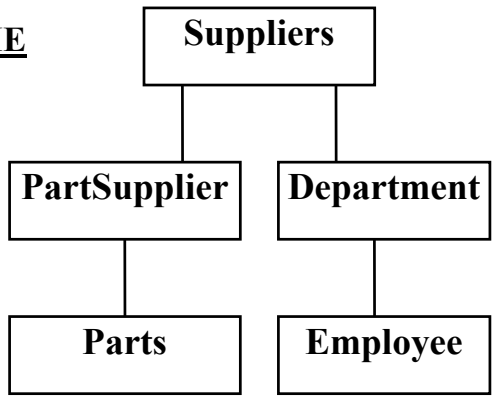

Figure 12.7 View optimization example

The example above in Figure 12.7 is identical to the previous example in Figure 12.6 except in this example no data is selected from the Dependent table. In this case, the DSE prototype determines from the semantics of the data structure that the Dependent table does not need to be accessed (see the ACCESS column in the data structure table above). Notice that the result of the SQL query statement in the example above, without the Dependent data and access to the Dependent table, remains consistent with the previous example. This proves this optimization works in this situation.

Conclusion

The live examples presented in this chapter show that the DSE nested relational prototype proves a number of things. First, that the DSE software works, it does extract the data structure meta information embedded in the Outer join. Second, it can be utilized to develop products like the nested relational processor that would not be possible otherwise with standard SQL. Third, and most importantly, it proves the data modeling technology behind the DSE software is valid and does works. This means the Outer join does indeed inherently support the data modeling of complex data structures. And fourth, it demonstrates this technology is useful and viable.