Advanced Capabilities of the Outer Join

Michael M. David
2210 Wilshire Blvd., Suite 167
Santa Monica, CA 90403
(310) 395-6889

Abstract

This paper demonstrates that the modeling of complex data structures can be performed easily and naturally in SQL using the direct outer join operation as defined in the proposed ISO-ANSI SQL2 standard. This paper goes on to demonstrate four advanced capabilities that can be implemented by SQL vendors utilizing the data modeling ability of the outer join. These capabilities are: powerful optimization techniques that can dynamically shorten the access path length; intelligent join view updates that utilize the semantics in the data structure being modeled; direct disparate heterogeneous database access that is transparent and efficient; and automatic conversion of multi-table structures into nested relations allowing for more powerful SQL operations.

1. Background

The outer join examples shown in this paper use the sample tables shown in Figure 1. These tables are related by the fields in the Department and Employee number domains.

Department table:

DEPTNO	DEPTNAME
101	Acct
102	Maint
103	HR

Employee table:

EMPNO	EMPNAME	DEPTFK
10	John	101
20	Mary	102
30	Mike	102

Dependent table:

DEPDNO	DEPDNAME	EMPFK
521	Jason	20
522	Steph	20

Figure 1 Sample tables used in examples

To assist in demonstrating the data modeling ability of the outer join, Figure 2 shows the tables in Figure 1 joined in exactly the same manner being modeled or viewed in two ways [MD3]. The qualified term user view will be used when necessary to avoid possible confusion between these conceptual views and SQL views.

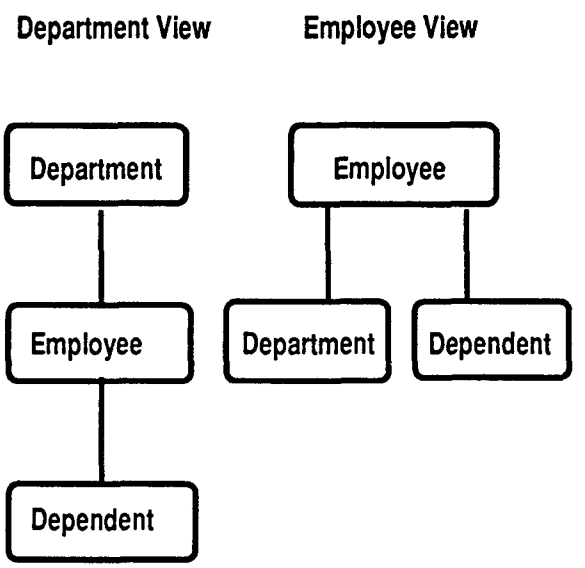

Figure 2 Two user views of tables related identically

Each user view in Figure 2 has its own semantics associated with it. For instance, in the Department view it is possible to have a department that has no employees, while in the Employee view it is not possible to have a department that has no employees.

To demonstrate the direct outer join operation, a subset of the proposed ISO-ANSI SQL2 outer join syntax and semantics [JM] will be used. SQL implementations, like Tandem's NonStop SQL [TC] and ShareBase's SSQL [SB], have closely followed this subset of the proposed ISO-ANSI SQL2 definition of the outer join. The data modeling examples using the ISO-ANSI SQL2 outer join in this paper have been verified on these vendor's SQL implementations.

With the ISO-ANSI SQL2 outer join, tables are joined two at a time where the table on the: left, right, both, or neither side can

have unmatched tuples preserved by specifying the corresponding keyword: LEFT, RIGHT, FULL and INNER. In the examples in this paper, NULL values, represented by question marks in the output, are used as place-holders for the missing data values caused by preserving unmatched tuples.

2. Outer Join Performs Data Modeling

The ability of the outer join to perform data modeling [MD1] is hidden in a very interesting and powerful operational characteristic. This characteristic being that the order in which the tables are joined can affect the result of the outer join, allowing control over its semantics. This beneficial lack of commutativity and associativity with the outer join is a direct result of its data preservation capability.

2.1 Simple data modeling

The table join order of the outer join becomes significant when three or more tables are joined. For this reason, the syntax of the ISO-ANSI SQL2 outer join operation results in the join table order being explicitly specified. Using this ability, Figure 3 shows an example of an outer join modeling the Department user view shown in Figure 2.

```
SELECT Deptname, Empname, Depdname
FROM   Department
    LEFT JOIN Employee  ON Deptno = Deptfk
    LEFT JOIN Dependent ON Empno = Empfk
```

DEPTNAME	EMPNAME	DEPDNAME
Acct	John	?
Maint	Mary	Jason
Maint	Mary	Steph
Maint	Mike	?
HR	?	?

Figure 3 Outer join modeling Department user view

Notice that the result of the left outer join shown in Figure 3 coincides with the semantics of the Department user view shown in Figure 2. This means that department HR can exist without any employees and that employees John and Mike can exist without any dependents.

It should also be apparent from the example in Figure 3, that the ON clause was added to the ISO-ANSI outer join syntax to specify join criteria at the point it is required during the join operation. That is, each table on the right of the JOIN keyword is joined with the previous join result or table. Using the LEFT outer join keyword option along with the table join order, as shown in this example, produces the desired semantics of the structure being modeled.

2.2 Complex Data Modeling

Figure 4 demonstrates how the outer join can also model the more complex Employee user view shown in Figure 2 using the same three tables and interrelationships. Notice how the result from this outer join matches the semantics of the Employee user view. In this view, department HR does not exist because its semantics only allow departments that have employees while employees can exist who have no department or dependents recorded in the database. This structure is a multi-leg hierarchy as can be seen by both ON clauses referring to the controlling Employee table to model it.

```
SELECT Deptname, Empname, Depdname
FROM   Employee
    LEFT JOIN Department ON Deptfk = Deptno
    LEFT JOIN Dependent  ON Empno = Empfk
```

DEPTNAME	EMPNAME	DEPDNAME
Acct	John	?
Maint	Mary	Jason
Maint	Mary	Steph
Maint	Mike	?

Figure 4 Outer join modeling Employee user view

Notice that both of the previous outer join examples specified the LEFT outer join keyword option throughout the join. Left outer joins are used most frequently in outer joins because they model hierarchical data structures (used extensively in user views) in the simplest and most natural way. Use of the INNER and FULL keyword options can not be used to model hierarchical structures since they do not model hierarchical constructs.

It is both interesting and important to note that the NATURAL outer join can not generally be used to model hierarchical or higher level data structures. The natural outer join operation is associative when it is coalescing across a common set of join fields, causing the table join order to have no effect on the result [CD3]. This results in a lack of control that is required to freely model data structures.

3. Advanced Outer Join Capabilities

Besides being used directly by the SQL user, the hierarchical data modeling capability of the outer join can also be utilized by SQL vendors to support advanced features and capabilities such as: dynamically optimized views, seamless disparate heterogeneous database access, and intelligent join view updates. These capabilities are possible because of the hierarchical model's well defined and unambiguous semantics. The hierarchical structure being modeled and its associated semantics can be easily determined from the outer join specification automatically and utilized by the SQL engine transparently.

3.1 Internal Optimizations

The first optimization capability identified here can be described as dynamically optimized views. It allows a single general use (universal) SQL view to be used in a wide range of queries without introducing any operational penalties. This is because outer join views modeling hierarchical structures only need to access the tables that are necessary for each unique invocation of the view. This semantic optimization technique uses principles set forth in the universal relation concept [JU].

To demonstrate the internal optimization possible with hierarchically modeled views, two SQL views, Empview and Deptview, will be used. These SQL views are defined from the outer joins that model the Department and Employee user views shown in figures 3 and 4. The optimization examples using these two SQL views will demonstrate that the processing of view invocations often does not require accessing all tables that are part of the underlying query. This is because tables in an underlying hierarchical modeled query that are not referenced on the view invocation often do not require access. Take for example the SQL Empview view invocation in Figure 5.

```
SELECT Empname, Depdname
FROM   Empview
```

EMPNAME	DEPDNAME
John	?
Mary	Jason
Mary	Steph
Mike	?

Figure 5 Employee view with no Department reference

The Empview view invocation in Figure 5 does not reference the Department table that is in the underlying SQL query shown in Figure 4. Since the Department table is in an unreferenced leg of the hierarchically structured Employee user view (Figure 2), it does not need to be accessed by the SQL access engine. This is shown in this query's result since it is consistent with the result from its underlying SQL query in Figure 4 (after taking into account that no columns from the Department table were referenced in the view invocation in Figure 5).

In a standard inner join view, all the tables (or their index tables) always need to be accessed. This is because unreferenced tables can also cause the result to have missing data which is required to properly reflect the result-set of the underlying query. Otherwise, tuples can be introduced into the view result that do not belong. Fortunately, the outer join is not subject to this lost data effect that must be accounted for in inner join views.

The query in Figure 6 demonstrates that this same optimization technique can be applied to a single leg of a hierarchically modeled query. The SQL view Deptview in this example models the Department user view in Figure 2. In this view invocation, the Dependent table is not referenced and does not need to be accessed because it is located lower on the hierarchical leg than any referenced table on that leg.

This access optimization technique can now be roughly stated as: tables that are not referenced or on a path to a referenced table in a hierarchically modeled outer join query do not need to be accessed, or in more precise terms: objects not required to connect the attributes involved in the query can be eliminated from the (outer) join. Further optimization requirements and considerations for the outer join can be found in [MD1].

```
SELECT Deptname, Empname
FROM   Deptview
```

DEPTNAME	EMPNAME
Acct	John
Maint	Mary
Maint	Mike
HR	?

Figure 6 Department view with no Dependent reference

By comparing the output of the Department view (Deptview) shown in Figure 6 to the output from its underlying query shown in Figure 3, a very powerful feature of this optimization can also be seen. The tuple for Employee Mary was not replicated unnecessarily in Figure 6. This would have occurred in an inner join since this tuple matches with two tuples from the unreferenced Dependent table as shown in Figure 3. While not consistent with the inner join, this operational characteristic is consistent with relational theory and is more intuitive. This feature will also reduce the use of the restrictive and usually costly SELECT DISTINCT operation.

The above dynamic view optimization is determined and fixed on or before view invocation. There is also a further optimization that can be applied dynamically during execution of hierarchically modeled queries. By accessing tables hierarchically in a top to bottom fashion, a given occurrence of a tuple being built can have further accesses to lower level referenced tables in a leg abandoned as soon as an unmatched tuple is encountered for that leg. This top to bottom hierarchical access strategy is also consistent with standard optimization strategy since the tables toward the top of the hierarchy usually have fewer tuples requiring accessing than those below them and should normally be driving the join process anyway.

These dynamic optimization techniques can be further optimized by processing each hierarchical leg in parallel. These hierarchical legs are naturally in an optimal parallel table join order because they do not influence each other and can be accessed independently. This independence can be inferred from the example in Figure 5 where only one hierarchical leg of the two in the Employee view is accessed without affecting the result.

3.2 Heterogeneous Access

Disparate heterogeneous database access for a logical hierarchical data model can be accomplished transparently and efficiently in SQL by utilizing the data modeling capability of the outer join. This technique works because hierarchically modeled outer joins can be naturally processed hierarchically in a top to bottom fashion following the structure they are modeling. In fact, as mentioned earlier under internal optimizations in section 3.1, this would probably be the standard access path chosen by the SQL optimizer for hierarchically modeled queries.

Past and current efforts to implement disparate heterogeneous access from SQL have relied on the standard inner join to identify the access path between objects (i.e. DATAPLEX [CW]). But because the inner join can not model data structures, semantic information in the foreign database structure being accessed is lost, resulting in limitations such as: restricting access to a single hierarchical path or requiring the access paths to be predefined. These limitations do not exist when disparate heterogeneous access is based on the outer join operation since it is capable of naturally modeling and processing hierarchical user views.

With the SQL engine capable of accessing hierarchically structured data as it is modeled by the outer join, hierarchically accessible databases can be freely and seamlessly accessed heterogeneously as they naturally exist. This technique retains the semantic structural information of the foreign database and involves no database structure mapping or translation because a common hierarchical data model is used across relational and non relational databases. For this reason, it is efficient and has no apparent limitations in SQL usage because it works in harmony with standard SQL processing through the relational outer join operation.

As a heterogeneous access example, imagine that the Employee and Department components of the Employee user view in Figure 2 are actually segments from an IMS database with Employee as its root segment. The remaining Dependent component of the user view is a relational table. The same outer join modeling the Employee view in Figure 4 could process these two disparate databases just as if they were a single relational database.

The relational engine will have to know that the Employee and Department references on the query statement are not tables, but actually segments from an IMS database and have code or exits to access the segments as they are required. SQL WHERE and ON clause predicates that can be converted to IMS SSA search criteria can be used to optimize the IMS access. (SQL WHERE and ON clauses affect the query operation differently, this must be taken into consideration when combining these predicates.) Those predicates that can not be converted will still be caught and processed easily as non optimizable predicates by the SQL engine after the data is retrieved.

The recognition of IMS and other foreign databases can be handled automatically by data dictionary/repository systems or by modification of the SQL CREATE TABLE statement to accept and supply the required additional information. Database differences such as unsupported data types and overdefined fields [CD2] could be handled at the database interface level. Beyond these definitional changes, the disparate heterogeneous database access associated with SQL manipulations would be transparent and efficient.

Besides the standard IMS optimizations [MD2] and the hierarchical based optimizations described in section 3.1, SQL's set processing capability can be used when accessing IMS databases. This can actually yield better performance than standard COBOL or assembler programs accessing the same IMS database procedurally. With SQL set processing, the accesses to the IMS database being initiated by the SQL access engine can actually be made directly to the underlying IMS VSAM files. This would allow accumulating large amounts of data at one time and eliminate the costly procedural IMS DLI call level access.

Large amounts of data can be accumulated at one time in advance of its use because the data structure and data requests that will be initiated by or on behalf of the outer join are known in advance due to SQL's non-procedural nature. The database access calls made directly to the underlying VSAM files are possible because the IMS internal storage structures are well known and stable. In fact, a number of commercial fast access IMS products (like BMC Software's IMS replacement utilities) take advantage of this same technique.

The same hierarchical access technique described above for heterogeneous database access can also be adapted to work with E-R data models like IBM's Repository and object oriented databases. This is possible because these database types also require inheritance and navigational capabilities which are both possible with the data modeling capability of the outer join.

3.3 Updating Join Views

By utilizing the semantics present in hierarchically modeled outer join views, the updating of these views can be performed intelligently. Using the two user views of the tables shown in Figure 2, it can be shown that the semantics of these structures can be used in determining how the join views modeling these structures can be intuitively updated.

Commonly proposed techniques that have been put forward for controlling the updating of multi-table update operations are to: update both sides of a one-to-one (PK,PK)-join, update the foreign key side of a one-to-many (PK,FK)-join [CD1], or require a DBA supplied procedure to make update decisions [EC1].

The weakness with these update procedures, although necessary in most cases, is that they assume the intent of the update operation and are limited to a small number of tables, or they require user intervention. But when the outer join is modeling hierarchical structures, the semantics are known and the intent of the update operation is more clear. This allows update operations to be successfully performed without arbitrary rules, limits or external aids.

As an example, the Department and Employee SQL views, Deptview and Empview, which contain the outer joins used in Figures 3 and 4 to model the two user views shown in Figure 2 will be used again. Now imagine a DELETE operation is performed against these views using WHERE EMPLOYEE="JOHN" as the restriction criteria. Proposed inner join view update techniques would treat both views exactly the same even though their semantics are different. By utilizing the semantics of the underlying structure, different interpretations of the update can be made for each view.

The following delete examples of hierarchically modeled outer join view updates are not meant to suggest a complete proposal, but to demonstrate that the opportunity exists to utilize the available additional semantics in the data structures being modeled. The examples work in conjunction with referential integrity support (and its required primary and foreign key support) now implemented by many SQL systems. The examples also presume there is a delete integrity rule of CASCADE applied to the Employee table when tuples are deleted from the Department table and a CASCADE delete integrity rule applied to the Dependent table when tuples are deleted from the Employee table.

Under these rules, a delete operation to the Department table would cascade down all three tables, so the result of a delete against any one of these tables is clearly defined. A delete against a SQL view encompassing more than one table is not usually that straightforward. The problem in this case is determining which table or tables are the object or intent of the delete operation. With hierarchically modeled queries, the object of a delete operation can always be based semantically on the root table. This will cause its direct deletion and indirect deletion of lower level tables resulting from cascading referential integrity operations triggered by the deletion of the root table.

For example, when the Department view (Deptview) is used in a delete operation, the object of the delete dictated by its hierarchically modeled structure shown in Figure 2 is the Department root table. This will cause deletion from the Department table which may in turn trigger additional deletes from the Employee and Dependent tables because of the referential integrity rules affecting these tables. In this example, the department that employee Mike works for will be deleted along with all of its employees and their dependents. This delete example is shown in Figure 7. Notice in this example that Mike's department of Maint was deleted along with all of its Employees and their dependents, not just Mike and any dependents he may have.

DELETE FROM Deptview WHERE Employee="Mike"

Would delete:

DEPTNAME	EMPNAME	DEPDNAME
Maint	Mary	Jason
		Steph
	Mike	?

Figure 7 Delete from Department view

The same delete request applied against the hierarchically modeled Employee view (Empview) places the Employee root table as the object of the delete operation as shown semantically in Figure 2. This will cause deletion from the Employee table which in turn may trigger deletes from the Dependent table because of referential integrity constraints on the Employee table. This is shown in the delete example in Figure 8 where employee Mike is deleted with any dependents he may have, but the department table is not affected because there is no applicable referential integrity rule specifying its deletion. So even though Mike was deleted, his department (Maint) was not.

DELETE FROM Empview WHERE Employee="Mike"

Would Delete:

EMPNAME	DEPDNAME
Mike	?

Figure 8 Delete from Employee view

Examples 7 and 8 show that hierarchically modeled join view update operations can work in conjunction with referential integrity update operations to produce a powerful synergistic effect. These two operations work well together because they are both based on the hierarchical model and can utilize their complimentary update information. Because the update operations are based on the hierarchical data model, they can also be applied uniformly across disparate heterogeneous databases as described in section 3.2.

3.4 Nested Relations

By using the data structure derived from a hierarchical outer join, nested relations that represent the hierarchical structure

can be built automatically. Nested relation systems like UniData's UniSQL [UD] are stored in non-first normal form (non-1NF) which preserves the data structure allowing relational operations to operate more intelligently on them.

Normally, SQL systems that support nested relations require that the data relationships be predefined and the data relations be converted to non-first normal form using a NEST operation. But, using the SQL outer join example in Figure 3 that models the Department view shown in Figure 2, the following non-first normal form structure can be generated automatically and transparently.

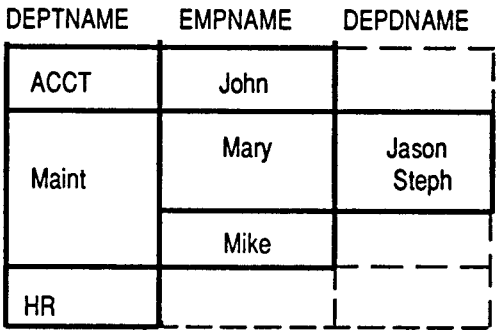

Figure 9 Department view in non-1NF form

The important points to notice in this example is that no data was replicated, the data structure stayed intact, and this was accomplished transparently. As an obvious example of the advantages of this structure over its first normal form counterpart shown in the output of Figure 3, is to perform COUNTs on the DEPTNAME and EMPNAME fields from both of these examples. You would notice that counts for Figure 3 would be wrong because of the replicated data caused by conventional relational databases' first normal form requirement.

4. Conclusion

This paper demonstrated how the newer and now commercially implemented ISO-ANSI SQL2 proposed outer join enables data structures to be modeled. This allows a much broader range of problems that can be solved easily with a relational solution.

Also demonstrated was how the data modeling capability of the SQL2 outer join can be utilized by SQL vendors to: perform substantial internal optimizations using principles set forth in the universal relation concept; allow more opportunity to perform intelligent join view updates because of the additional semantics available; enable transparent disparate heterogeneous database access directly and efficiently from SQL because of the data structure processing ability of the outer join operation; and utilize the power of nested relations more automatically.

Additional information on the outer join can be found in [EC2] and [MD4].

References

[CD1] C. J. Date, "Updating Views", Chapter 17 in book entitled "Relational Database Selected Writings", Addison Wesley, 1986.

[CD2] C.J. Date, "Why is it so Difficult to Provide a Relation Interface to IMS", Chapter 12 in book entitled "Relational Database Selected Writings", Addison Wesley, 1986.

[CD3] C. J. Date, "The Outer Join", Chapter 16 in book entitled "Relational Database Selected Writings", Addison Wesley, 1986

[[CW] C. Chung, "DATAPLEX, An Access to Heterogeneous Distributed Databases", Communications of the ACM 33,1 (Jan.1990).

[EC1] E. F. Codd, "View Updatability", Chapter 17 in book entitled "The Relational Model for Database Management Version 2", Addison Wesley, 1990.

[EC2] E. F. Codd, "The Advanced Operators", Chapter 12 in book entitled "The Relational Model for Database Management Version 2", Addison Wesley, 1990.

[JM] J. Melton, ed. ISO-ANSI Database language SQL2 (working draft), 199X.

[JU] J. Ullman, "The Universal Relation As a User Interface", Chapter 17 in book entitled "Principles of Database and Knowledge-Base Systems", Volume II, Computer Science Press ,1989.

[MD1] M. David, "Ins and Outs of Outer Joins", DATABASE Programming & Design, Feb. 1990.

MD2] M. David, "Heterogeneous Processing of Relational, IMS and VSAM Databases", DATABASE Programming & Design, April 1991.

[MD3] M. David, "The Outer Limits of the Relational Join", 370/390 DATA BASE Management, October 1991.

[MD4] M. David, "The Importance of the Outer Join to Online Environments", DATABASE Programming & Design, Dec. 1990.

[SB] ShareBase, "SSQL Reference Manual", Part number 205-2808-000 Revision B, July 1991.

[TC] Tandem Computers, "NonStop SQL Reference Manual", release version C30/PM, Sept 1990.

[UD] UniData, "UniSQL User's Guide" , Release 2.1, 1991.