



Multidimensional Grouping Made Easy

Introducing DB2 for V6R1 i5/OS's new SQL group functions

by Mike Cain

AS BUSINESS SOLUTIONS INCREAS-ingly rely on robust data-centric processing, the capabilities of the database management system must continue to rise to the challenge and provide not just fast and efficient data access, but also the ability to handle complex query requests that involve sophisticated data manipulations. With the release of i5/OS V6R1, the integrated version of DB2 delivers many new features. One of the features is the addition of advanced support for grouping sets and super groups, which allows for online analytical processing (OLAP) via SQL. This new advanced support also brings the new keywords ROLLUP, CUBE, GROUPING SETS, and GROUPING. With these new grouping functions, application developers can more easily deliver results for multidimensional requests.

How the Functions Work

SQL OLAP functions meet query and reporting requirements that call for a result set to contain not only aggregation for each specific group (i.e., totals or averages for each concatenated set of grouping columns), but they also include intermediate and final results for different sets of grouping columns. In effect, you will be able to calculate all combinations with one query.

The following fictitious table definition illustrates how these functions work:

```
CREATE TABLE Orders (  
  OrderNo INTEGER NOT NULL,  
  CustomerNo INTEGER NOT NULL,  
  OrderYear INTEGER NOT NULL,  
  OrderQuarter INTEGER NOT NULL,  
  OrderMonth INTEGER NOT NULL,  
  DiscCode CHAR(3),  
  OrderAmount DECIMAL(5,2) NOT NULL  
);
```

The fictitious data represents orders for the end of 2006 and the beginning of 2007, such that

- OrderYear contains two distinct values: 2006 and 2007
- OrderQuarter contains four distinct values: 1, 2, 3, and 4
- OrderMonth contains 10 distinct values: 1, 2, 3, 4, 5, 8, 9, 10, 11, and 12
- DiscCode contains three distinct values: 'ABC', 'DEF', and NULL

Before the new support, a query request that specified the grouping expression consisting of columns OrderYear, OrderQuarter, and OrderMonth would result in information only for each distinct value of OrderYear+OrderQuarter+OrderMonth (Figure 1).

```
SELECT      OrderYear,  
            OrderQuarter,  
            OrderMonth,  
            SUM(OrderAmount)  
FROM        Orders  
GROUP BY   OrderYear,  
            OrderQuarter,  
            OrderMonth;
```

FIGURE 1

Resulting order table

OrderYear	OrderQuarter	OrderMonth	OrderAmount
2006	3	8	210
2006	3	9	275
2006	4	10	305
2006	4	11	320
2006	4	12	450
2007	1	1	200
2007	1	2	260
2007	1	3	350
2007	2	4	310
2007	2	5	365

FIGURE 2

GROUP BY ROLLUP query

```
SELECT *
FROM (
SELECT      OrderYear, OrderQuarter, OrderMonth, SUM(OrderAmount)
FROM        Orders
GROUP BY   OrderYear, OrderQuarter, OrderMonth

UNION ALL

SELECT      OrderYear, OrderQuarter, NULL, SUM(OrderAmount)
FROM        Orders
GROUP BY   OrderYear, OrderQuarter, NULL

UNION ALL

SELECT      OrderYear, NULL, NULL, SUM(OrderAmount)
FROM        Orders
GROUP BY   OrderYear, NULL, NULL

UNION ALL

SELECT      NULL, NULL, NULL, SUM(OrderAmount)
FROM        Orders
GROUP BY   NULL, NULL, NULL
)
```

Now imagine that the application requires not only sales figures for the detail-level groups represented in OrderYear+OrderQuarter+OrderMonth, but also the intermediate and final totals for each level. Typically, you need to execute queries to obtain figures for each level — detail, subtotal, and final total. Multiple queries mean that you do more work to get all the information that you need.

Specifying a new super grouping function enables the DB2 for i5/OS engine to calculate and provide information for each distinct value

OrderYear+OrderQuarter+OrderMonth
OrderYear+OrderQuarter
OrderYear
Overall totals for the entire set

With the proper super grouping or grouping set syntax, the engine can also return any or all combinations of the grouping columns, such as

OrderYear+OrderQuarter+OrderMonth
OrderYear+OrderQuarter
OrderYear
OrderYear+OrderMonth
OrderQuarter+OrderMonth
OrderQuarter
OrderMonth
Overall totals for the entire set

This is yet another case of driving more data-centric processing from DB2 with a *single* SQL statement.

ROLLUP

The ROLLUP function added to the GROUP BY

clause represents “super grouping.” This function calculates and returns aggregates for each level of the hierarchy implicitly represented in the grouping columns.

Given the hierarchy represented by columns X/Y/Z..., GROUP BY ROLLUP (X, Y, Z) produces the following aggregation results for grouping sets:

(X, Y, Z)
(X, Y)
(X)
() = grand totals

For example, if a query request specifies ROLLUP and the grouping columns of OrderYear, OrderQuarter, and OrderMonth, the database engine provides information for each distinct value of OrderYear+OrderQuarter+OrderMonth; OrderYear+OrderQuarter; OrderYear; and a grand total for the entire set of results.

```
SELECT      OrderYear,
            OrderQuarter,
            OrderMonth,
            SUM(OrderAmount)
FROM        Orders
GROUP BY   ROLLUP ( OrderYear,
                    OrderQuarter,
                    OrderMonth);
```

To help understand the amount of work involved, consider the GROUP BY ROLLUP query as logically equivalent to the *pseudo* SQL in Figure 2. In addition to the 10 rows that the query normally returns, DB2 also produces seven more rows (shown in blue in Figure 3). The seven rows show the subtotals and final total for the grouping column values represented in the hierarchy: months within quarter, quarters within year, and years within total. This represents the *super grouping* work.

Notice that for each subtotal and total, NULL values are used for the rolled-up grouping column. This is true even if that particular column is not NULL capable. (I will explain later how to distinguish NULL values produced by super group operations from NULL values in the underlying table.) Because the sample SQL did not specify an ORDER BY clause, the actual order of the results is unpredictable (more about this later).

CUBE

The CUBE function added to the GROUP BY clause (also considered super grouping) calculates and returns aggregates for all possible distinct combinations represented by the grouping columns. In other words, in addition to grouping along the hierarchy produced by

FIGURE 3

GROUP BY ROLLUP table

OrderYear	OrderQuarter	OrderMonth	OrderAmount
2006	3	8	210
2006	3	9	275
2006	3	-	485
2006	4	10	305
2006	4	11	320
2006	4	12	450
2006	4	-	1075
2006	-	-	1560
2007	1	1	200
2007	1	2	260
2007	1	3	350
2007	1	-	810
2007	2	4	310
2007	2	5	365
2007	2	-	675
2007	-	-	1485
-	-	-	3045

ROLLUP, CUBE also calculates cross-tabular results. Cubing operations are particularly useful to produce multidimensional results in support of OLAP. Imagine a single SQL query that provides information at the various intersections of subjects (e.g., sales figures by department, store, location, date) and effectively produces a “cube” of information.

Given the hierarchy represented by columns X/Y/Z..., GROUP BY CUBE (X, Y, Z) produces aggregation results for grouping sets:

- (X, Y, Z)
- (X, Y)
- (X, Z)
- (X)
- (Y, Z)
- (Y)
- (Z)
- () = grand totals

Note that (X,Y) is equal to (Y,X) and thus redundant and not produced. For example, if a query request specifies cube and the grouping columns of OrderYear, OrderQuarter, and OrderMonth, the database engine provides information for each distinct value of OrderYear+OrderQuarter+OrderMonth; OrderYear+OrderQuarter; OrderYear+OrderMonth; OrderYear; OrderQuarter+OrderMonth; OrderQuarter; OrderMonth; and a grand total for the entire set of results.

```
SELECT      OrderYear,
           OrderQuarter,
           OrderMonth,
           SUM(OrderAmount)
FROM        Orders
```

FIGURE 4

GROUP BY CUBE table

OrderYear	OrderQuarter	OrderMonth	OrderAmount
2006	3	8	210
2006	3	9	275
2006	3	-	485
2006	4	10	305
2006	4	11	320
2006	4	12	450
2006	4	-	1075
2006	-	-	1560
2007	1	1	200
2007	1	2	260
2007	1	3	350
2007	1	-	810
2007	2	4	310
2007	2	5	365
2007	2	-	675
2007	-	-	1485
-	3	-	485
-	4	-	1075
-	1	-	810
-	2	-	675
-	-	1	200
-	-	2	260
-	-	3	350
-	-	4	310
-	-	5	365
-	-	8	210
-	-	9	275
-	-	10	305
-	-	11	320
-	-	12	450
-	3	8	210
-	3	9	275
-	4	10	305
-	4	11	320
-	4	12	450
-	1	1	200
-	1	2	260
-	1	3	350
-	2	4	310
-	2	5	365
2007	-	1	200
2007	-	2	260
2007	-	3	350
2007	-	4	310
2007	-	5	365
2006	-	8	210
2006	-	9	275
2006	-	10	305
2006	-	11	320
2006	-	12	450
-	-	-	3045

```
GROUP BY CUBE (OrderYear,
              OrderQuarter,
              OrderMonth);
```

In addition to the 10 rows that the query normally returns, DB2 also produced 41 more rows (shown in blue in Figure 4) that represent the subtotals and final total for the various combinations of grouping column values. This represents the expansive super grouping

FIGURE 5

GROUPING SET table

OrderYear	OrderQuarter	OrderMonth	OrderAmount
2006	-	-	1560
2007	-	-	1485
-	3	-	485
-	4	-	1075
-	1	-	810
-	2	-	675
2006	-	8	210
2006	-	9	275
2006	-	10	305
2006	-	11	320
2006	-	12	450
2007	-	1	200
2007	-	2	260
2007	-	3	350
2007	-	4	310
2007	-	5	365

FIGURE 6

Table showing super groups

OrderYear	DiscCode	Order YearGroup	DiscCode Group	OrderAmount
-	-	1	1	3425
2006	ABC	0	0	505
2006	-	0	0	675
2006	DEF	0	0	445
2007	-	0	0	800
2007	ABC	0	0	650
2007	DEF	0	0	350
2006	-	0	1	1625
2007	-	0	1	1800

accomplished by the database engine. Again, notice that for each subtotal and total, NULL values are used to represent the empty grouping columns.

GROUPING SETS

The GROUP BY GROUPING SET functionality lets you specify multiple grouping clauses in a single statement where DB2 calculates and returns aggregates for each set. Before this new grouping support, only one set of grouping columns would be specified. Now more than one set of grouping criteria can be provided.

The CUBE and ROLLUP operations can produce a series of grouping sets, as previously illustrated. For n grouping columns of the ROLLUP operation, $n+1$ grouping sets are produced. For n grouping columns of the CUBE operation, 2^n (2 to the power n) grouping sets are produced. The exponential growth of grouping sets and the corresponding increase in data processing are important to keep in mind.

Where CUBE and ROLLUP implicitly predefine which subtotals will be calculated, the GROUPING SET notation lets the application explicitly define

which groups will have subtotals returned. For example, instead of all the cross-tabular grouping combinations returned from a CUBE operation, maybe the application needs only a subset of all possible combinations.

Given the hierarchy represented by columns X/Y/Z..., GROUP BY GROUPING SETS ((X), (Y), (X, Z)) produces aggregation results for grouping sets:

(X)
(Y)
(X, Z)

For example, given a query request that specifies the three grouping sets (OrderYear), (OrderQuarter), and (OrderYear, OrderMonth), the database engine provides aggregate information only for each distinct value of OrderYear (which is set 1), OrderQuarter (set 2), and OrderYear+OrderMonth (set 3).

```
SELECT      OrderYear,
            OrderQuarter,
            OrderMonth,
            SUM(OrderAmount)
FROM        Orders
GROUP BY    GROUPING SETS (
            (OrderYear),
            (OrderQuarter),
            (OrderYear, OrderMonth));
```

Figure 5 shows the resulting GROUPING SETS table.

The grouping set support is powerful, and it can produce a huge set of results depending on the syntax you use. This is especially significant given that you can specify CUBE (set 1) and/or ROLLUP (set 2) as part of a grouping set specification (i.e., DiscCode — set 3, CustomerNo — set 4).

```
GROUP BY    GROUPING SETS
            (CUBE(OrderYear, OrderQuarter, OrderMonth),
            ROLLUP(OrderMonth, OrderNo),
            (DiscCode),
            (CustomerNo));
```

The CUBE or ROLLUP operation expands into yet more grouping sets, as I previously described. Find more examples and information about the additive and multiplicative nature of grouping set notation in the *DB2 for i5/OS SQL Reference Manual*. It is a good idea to lay out the grouping required and to carefully examine the SQL syntax *before* running the query. If you don't, you may unleash the proverbial "query from hell."

GROUPING Aggregate Function

As I previously stated, part of the grouping sets' and super groups' functionality includes the database

engine setting some or all of the result columns to NULL. The GROUPING aggregate function helps the application understand the value of a particular grouping column. In other words, did DB2 *retrieve* and use a value from the table, or did DB2 *set* a NULL value to represent that no grouping value is present in the query result?

As part of the query's SELECT list, the aggregate function returns a small integer data type, which is set to either 0 or 1. The value 1 is used when DB2 sets the referenced grouping column to NULL because the row was produced from a grouping set or super group operation. The value 0 is used when the referenced grouping column contains an actual *group by* value from the underlying data set.

```
SELECT OrderYear,
       DiscCode,
       GROUPING(OrderYear) AS OrderYearGroup,
       GROUPING(DiscCode) AS DiscCodeGroup,
       SUM(OrderAmount)
FROM Orders
GROUP BY ROLLUP (OrderYear, DiscCode);
```

Recall that the ROLLUP operation produces the following grouping sets:

```
(OrderYear, DiscCode)
(OrderYear) = DiscCode value set to NULL by DB2
() = OrderYear and DiscCode value set to NULL by DB2
```

Given that columns OrderYear and DiscCode are set to NULL as part of the ROLLUP super grouping operation, the GROUPING aggregate function is used on OrderYear and DiscCode to tell the application whether the respective column values are part of the underlying data set. If DB2 sets OrderYear to NULL, the value of column OrderYearGroup is set to 1. If DB2 sets DiscCode to NULL, the value of column DiscCodeGroup is set to 1. In all other situations, 0 is returned in these columns.

The GROUPING function is particularly useful when the grouping column is NULL capable and the application needs to determine whether the NULL value in the result set is part of a super group or the source data. For example, the column DiscCode is defined as NULL capable, so there may be an actual group with a legitimate NULL value (Figure 6).

The rows highlighted in blue in Figure 6 are the super groups. These rows have DiscCode and/or OrderYear set to NULL by DB2. The other rows represent groups from the underlying data set.

Provide Order

While you review the examples, you may notice that the order of the rows in the result sets is random and unpredictable. SQL does not guarantee a particular order or consistency of ordering unless you specify the ORDER BY clause in the query. Without an ORDER BY clause, even the same query (against the same data) with the same result set might produce a different result order.

Given that the new grouping operations can return NULL for grouping column values, remember that the default behavior of DB2 is to order the NULL values last. In other words, the NULL value is higher than all other values.

```
SELECT OrderYear,
       DiscCode,
       GROUPING(OrderYear) AS OrderYearGroup,
       GROUPING(DiscCode) AS DiscCodeGroup,
       SUM(OrderAmount)
FROM Orders
WHERE DiscCode IS NOT NULL
GROUP BY ROLLUP (OrderYear, DiscCode)
ORDER BY OrderYear ASC, DiscCode ASC;
```

If the application expects or assumes a particular order of rows from the query, you need to specify the ORDER BY clause, especially when you use the GROUPING SETS, CUBE, and ROLLUP functions.

SQL Query Optimization

As part of the ongoing DB2 for i5/OS enhancement strategy, virtually all new features and functions are available only when you use the SQL Query Engine (SQE) to optimize and run the query request. The new V6R1 grouping functions are no exception. Queries optimized and executed by the original Classic Query Engine (CQE) *cannot* specify GROUPING SETS, CUBE, or ROLLUP operations. The good news is that almost all SQL requests can now take advantage of SQE's benefits when running on V6R1.

DB2 for i5/OS continues to use sophisticated, and in many cases IBM-patented, technology when it optimizes and runs queries with the new grouping functions. Because one of the primary goals of the DB2 Query Optimizer is to minimize I/O and build the most efficient plan, query rewrite occurs for GROUPING SETS, ROLLUP, and CUBE operations. This rewrite capability lets the integrated version of DB2 perform multiple sets of aggregations with a single pass of the data, convert CUBE to a series of ROLLUPS to take advantage of the single pass process, and combine disparate GROUPING SETS and ROLLUPS into ROLLUPS with depth control.

The query rewrite capability also includes support for materialized query tables (MQTs). This lets DB2 rewrite a super grouping query in order to use a properly created and populated MQT, which further enhances performance and efficiency.

SQL Query Tuning

To help application developers and SQL performance analysts, the DB2 for i5/OS OnDemand Performance Center tools have been enhanced to recognize and support the new grouping functions. The SQL Plan Cache Snapshots, SQL Performance Monitor, and state-of-the-art Visual Explain report on and illustrate the grouping query plans.

As with most queries, a proper indexing strategy gives DB2 statistics and more options to optimize and implement an SQL request that contains GROUPING SETS, CUBE, and ROLLUP. Although DB2 will automatically provide advice, and possibly even create the proper temporary index, the best practice is to provide a permanent index *before* you run in production.

Typically, it's best to create an index with key columns that cover the local selection equality predicates (e.g., WHERE MyColumn = 'ABC') and the grouping column(s). This basic strategy also applies to grouping sets and super groups.

```
SELECT      OrderYear ,
           DiscCode ,
           SUM(OrderAmount)
FROM        Orders
WHERE       CustomerNo = 112358
GROUP BY   ROLLUP (OrderYear, DiscCode)
ORDER BY   OrderYear, DiscCode;
```

```
CREATE INDEX OrdersIndex
ON Orders
(CustomerNo, OrderYear, DiscCode);
```

DB2 for i5/OS has also been enhanced to automatically provide index advice for these new types of grouping queries.

Upgrade to the Latest and Greatest

With the powerful new grouping and OLAP functionality, V6R1 brings yet another level of data-centric capabilities to the application developer. It's time to do more with SQL. It's time to let DB2 for i5/OS do more for you. ■

► **Mike Cain** is a senior technical staff member with IBM, where he focuses on researching, quantifying, and demonstrating i5/OS support for very large database, business intelligence, and SQL query performance. You can e-mail Mike at mcain@us.ibm.com.

Up to Speed on the Essentials?

Our expert-written Essential Guides cover

- Remote Journaling
- IFS Security
- Encryption
- Implementing RFID
- Global Data Synchronization
- WDSc for PDM/SEU Users
- Backup and Recovery
- Connecting Excel and the iSeries
- Stored Procedures
- Thwarting Hackers
- SCM
- XML
- High Availability
- Free-Format RPG

Check them out at SystemiNetwork.com/essentialguides

Your byline could be here!

Help your fellow System i professionals AND earn money doing it. Our most valuable articles, tips, and utilities come from technical professionals like you who write from their own experiences.

Even if you've never written for publication before, your professional experience is a valuable resource. Support the System i community by sharing your field-tested techniques and practices in *System iNEWS*. Our editors will work with you to polish your prose. Contact Vicki Hamende (vhamende@penton.com) for more information or to propose a topic idea.