Accelerated analytics - faster aggregations using the IBM DB2 for i encoded vector index (EVI) technology

Effectively including non-key aggregate values in an EVI

Nick Lawrence

October 31, 2013

This article describes an IBM DB2® for i 7.1 technology that can improve the performance of business intelligence (BI) and analytic workloads by including aggregate values for non-key columns in an encoded vector index (EVI).

Prerequisites

This article assumes that the reader has a basic understanding of EVIs, DB2 for i indexing strategies, DB2 for i star-schema join support, and materialized query tables (MQTs). The DB2 for i Center of Excellence team has published a number of white papers that provide this background information. If necessary, review the links to these papers in the References section.

This article includes both example SQL queries and visual explanations of the queries to demonstrate how EVIs are used to improve performance. The visual explanations were created using the Visual Explain tool that is included with IBM i Navigator. Refer to the database query optimization references that are included in the References section for more information on this tool.

The references section also includes a link to a description of an SQL performance workshop that is offered by the DB2 for i Center of Excellence team. This workshop is a first-rate way to obtain an in-depth understanding of the DB2 for i performance analysis and optimization concepts that are referenced in this paper. Refer to the link in the References section for more detailed information.

The functions described in this article assume that the business intelligence or analytics application is built on DB2 for i 7.1, with an interim fix level of at least SF99701 Level 18. IBM recommends that customers apply the most recent updates in order to receive the most recent fixes and enhancements. You can find a link to the DB2 for i technology updates page in the References section.

Overview

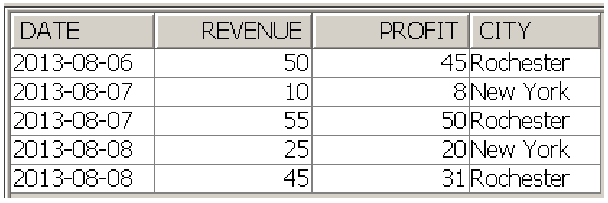
In a typical BI or analytics environment, fact tables contain one or more measures that need to be aggregated by grouping on a dimension. For example, a report might need to provide the sum of a revenue measure, grouped by a location dimension. When the fact table contains millions (or billions) of rows, an efficient mechanism to calculate the aggregations is critical for the performance of ad hoc queries.

EVIs are not a recent addition to DB2 for i and have always been very well suited for the types of ad hoc queries over large data sets that occur in a BI environment. Based on a patented technology developed by IBM research, these indexes first became available in 1998 with IBM OS/400® Version 4 Release 3. Conventional use of an EVI improves the performance of queries by allowing the database manager to efficiently implement local selection of rows using skip sequential processing; that is, the interesting rows are visited in their physical order (as opposed to random access), and ranges of rows that do not meet the selection criteria are skipped over. Another conventional advantage of an EVI is that statistical and descriptive information is stored in the EVI's symbol table for each distinct key, allowing the optimizer to use the EVI to make very intelligent choices when optimizing a query. This article discusses the benefits of an enhancement in DB2 for i 7.1 that allows aggregate values for measure columns to be included in the EVI's symbol table.

Including aggregate values of measure columns in an EVI's symbol table is one easy way to enhance your existing indexing strategy to get additional performance improvements. The inclusion means that each symbol in the EVI's symbol table also includes one or more aggregate values for all rows associated with that symbol. Including aggregate values for measures in the EVI provides the DB2 optimizer with a quick option for calculating these values in a query, if the keys of the index are used as the grouping criteria.

For example, the table shown in Figure 1 contains revenue and profit numbers by date and city.

Figure 1: Revenue and profit numbers by date and city



DATE	REVENUE	PROFIT	CITY
2013-08-06	50	45	Rochester
2013-08-07	10	8	New York
2013-08-07	55	50	Rochester
2013-08-08	25	20	New York
2013-08-08	45	31	Rochester

If an EVI is created using city as a key column, the EVI can include useful aggregates of the measure columns such as `SUM(revenue)` and `AVG(profit)`. After creation, the EVI's symbol table will look as shown in Table 1.

Table 1: EVI symbol table

SUM(revenue)	AVG(profit)	CITY
35	14	New York
150	42	Rochester

A query that needs to calculate `SUM(revenue)` or `AVG(profit)`, while grouping on city, only needs to access the EVI symbol table and retrieve the included aggregations, and there is no need to access the (usually many) rows in the fact table.

Comparison with MQTs

Many BI solutions use MQTs to store pre-aggregated values. The use of MQTs has been an important performance technique for many years due to the DB2 for i optimizer's ability to take advantage of the pre-aggregated data in the MQT. The primary disadvantage of MQTs is that they need to be frequently refreshed and can be expensive to maintain.

As an alternative, the DB2 for i optimizer can use pre-aggregated values that have been included in an EVI. Storing pre-aggregated values in an index allows the values to be immediately maintained; as a result, there is no need to refresh the aggregate values to keep the stored values consistent with the data. The addition of an aggregate value does not significantly increase the index maintenance cost.

The AVG, COUNT, COUNT_BIG, SUM, STDDEV, STDDEV_SAMP, VARIANCE, and VARIANCE_SAMP functions can be included as aggregate value expressions in an EVI. A restriction for EVI's is that the aggregate MIN and MAX values cannot be included for a non-key column.

The DB2 for i optimizer is most likely able to optimize a query by using the included aggregate values in the EVI under the following conditions:

- The GROUP BY is from a single table
- The referenced columns of the GROUP BY are in the key definition
- The selection is based on the keys of the index
- The commitment control level is *NONE or *CHG

These conditions describe the most common type of query that will be optimized to make use of the aggregate values in EVI's symbol table; the conditions are not absolute restrictions on when the aggregate values will be useful to the optimizer. Examples of complex queries that will be optimized to make the most of the included aggregate values are provided later in this article.

When an MQT performs aggregations and the GROUP BY refers to a sin gle table, it might be possible to replace the MQT with an EVI that includes the aggregate values. As the data in the base table changes, EVIs are maintained by the system whereas MQTs are not. MQT maintenance is the user's responsibility.

For queries where the DB2 optimizer is unlikely to use aggregate values that have been included in an EVI, an MQT continues to be a possible option. The DB2 optimizer can match a query with an MQT when the GROUP BY is over more than one table. MQTs can be matched to a query that contains the MIN or MAX aggregate function. Another feature of MQTs is that the commit level of the MQT needs to be greater than or equal to the commit level of the query, but does not necessarily have to be *NONE or *CHG.

When to include an aggregate value in an EVI

It can be beneficial to create an EVI that includes aggregate values instead of an MQT that contains the same aggregations; an EVI is favorable in circumstances where it is advantageous to allow the system to maintain the aggregate values automatically. One strategy for creating useful EVIs (with included aggregates) is to design an optimal MQT strategy, and then identify and replace suitable MQTs with EVIs. An MQT is suitable for replacement with an EVI if the EVI is likely to be useful for queries that might otherwise use the MQT. An MQT that performs aggregations over a single table is an example of an MQT that might be suitable for replacement.

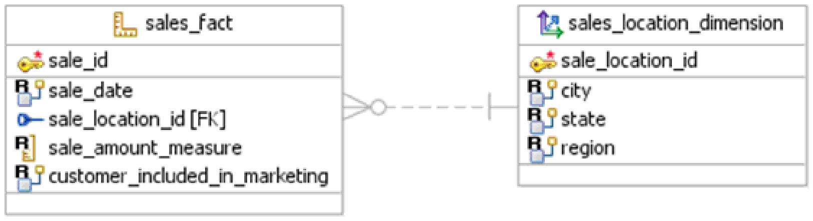
Another approach is to create an EVI with included aggregate values as part of a superior indexing strategy. When using this approach, an aggregate over a measure column should be included in an EVI when the EVI is beneficial for traditional reasons, and the measure column is frequently aggregated by grouping on the EVI's key. The optimizer then has the additional aggregate information available for the group, which it can choose to use in the implementation of the query. When the optimizer is not able to use the included information for aggregating, the index can still be used to retrieve accurate statistical information; in addition, (when it is appropriate) the EVI can be used for table selection, skip sequential plans, or for index ANDing and ORing plans. Thus, including the aggregate value can produce outstanding performance in a subset of queries (at a low maintenance cost), and the EVI continues to be used by the optimizer to perform a wide range of optimizations for other queries.

It will be easier to understand the effect of including an aggregate value by looking at some examples.

Example star schema

Figure 2 shows a very simple star schema, named sales_bi. A fact table (sales_fact) has one measure column (sale_amount_measure). The fact table provides two dimensions for grouping; a date dimension (sale_date), and location (sale_location_id). In addition, each of these dimensions provides opportunities for further grouping. For example, the sale_date column also allows grouping by year, quarter and month, and the sale_location_id column allows for grouping by city, state, or region (using the sales_location_dimension table). The customer_included_in_marketing column contains a 'Y' value if the customer who made the purchase did so because of a marketing campaign.

Figure 2: sales_bi schema



Optimizing SUM() over GROUP BY sale_location_id

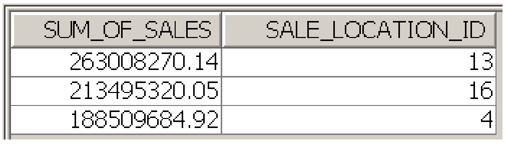
Listing 1 shows a trivial query that returns the sum of the sales, grouped by location ID, in descending order. The selection criteria in the HAVING clause ensures that only the locations 4, 16, and 13 are included in the results.

Listing 1: Aggregate sale_amount_measure for specific locations

```
SELECT SUM(sale_amount_measure) sum_of_sales,
       sale_location_id
FROM   sales_fact
GROUP BY sale_location_id
HAVING sale_location_id IN (4, 16, 13)
ORDER BY sum_of_sales DESC
```

The example output from the query in Listing 1 is shown in Figure 3.

Figure 3: Sum of sales, grouped by sale_location_id



SUM_OF_SALES	SALE_LOCATION_ID
263008270.14	13
213495320.05	16
188509684.92	4

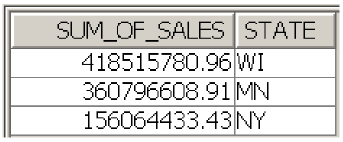
The query in Listing 1 is a simple but uncommon example; most analytical queries will involve a join with a dimension table. For example, the query in Listing 2 does a join so that the grouping of the aggregation is by state, rather than a location ID. Only the states New York, Wisconsin, and Minnesota are included.

Listing 2: Aggregate sale_amount_measure by state

```
SELECT SUM(sale_amount_measure) AS sum_of_sales, state
FROM   sales_fact
INNER JOIN sales_location_dimension
ON (sales_fact.sale_location_id = sales_location_dimension.sale_location_id)
GROUP BY state
HAVING state IN ('NY', 'WI', 'MN')
ORDER BY sum_of_sales
DESC
```

The output from the query in Listing 2 is shown in Figure 4.

Figure 4: Sum of sales grouped by state



SUM_OF_SALES	STATE
418515780.96	WI
360796608.91	MN
156064433.43	NY

The queries in Listing 1 and Listing 2 both need to access an enormous number of rows in the sales_fact table, which will result in poor performance. Thus, improving the performance of these queries by creating indexes is fundamental to success.

Database developers who have reviewed the background information on DB2 for i star-join support and DB2 for i indexing strategies (found in the references section) can recognize that the fact table's sale_location_id column is a good candidate to be a key for an EVI. A secondary observation is that the key for the prospective EVI is used as the grouping column for the query in Listing 1, and in Listing 2, the grouping of the state column is based on a hierarchy over the key column. Because both queries also need to calculate `SUM(sale_amount_measure)`, this aggregation is a good candidate to include in the index.

The DB2 for i Index Advisor does not recommend the inclusion of aggregate values in an EVI. However, as aggregate values are usually included in EVIs that need to be created for conventional reasons, the Index Advisor can be used to help identify the indexes that must be created. After a prospective EVI has been identified, the queries that caused the EVI to be recommended can be examined to determine whether there are aggregate values that should be included in the EVI. (The ability to view statements that caused an index create to be advised is a technology update for DB2 for i 7.1. For more details, refer to the link in the References section.)

The DB2 for i optimizer can determine whether to use an EVI's aggregate value in a query. The additional maintenance cost of including the values is small, and is not a major consideration.

Creating an EVI with included aggregate values

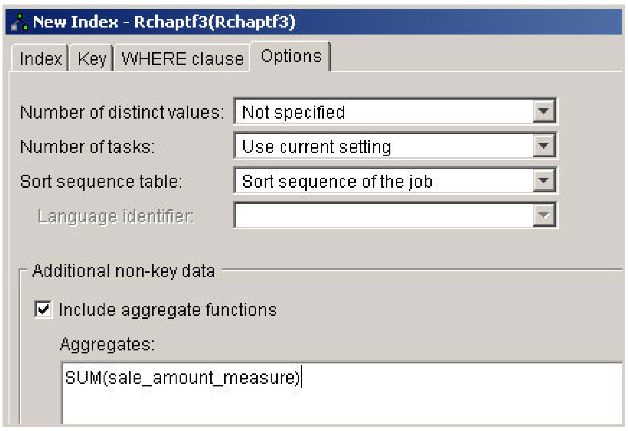
Creating an EVI and including one or more aggregate values is straightforward. The CREATE ENCODED VECTOR INDEX statement has an INCLUDE clause that can be used to specify the non-key aggregate values. Listing 3 shows the SQL syntax for an encoded vector index that defines the sale_location_id column as a key and includes SUM(sale_amount_measure) as additional non-key data.

Listing 3: Create an EVI statement

```
CREATE ENCODED VECTOR INDEX sales_fact_location_id_evi
ON sales_fact(sale_location_id ASC)
INCLUDE(SUM(sale_amount_measure))
```

In addition, the IBM i Navigator has been enhanced so that the new index dialogue contains an option to add non-key aggregate values, as shown in Figure 5.

Figure 5: The Options tab with additional non-key data



Number of distinct values: Not specified
Number of tasks: Use current setting
Sort sequence table: Sort sequence of the job
Language identifier:

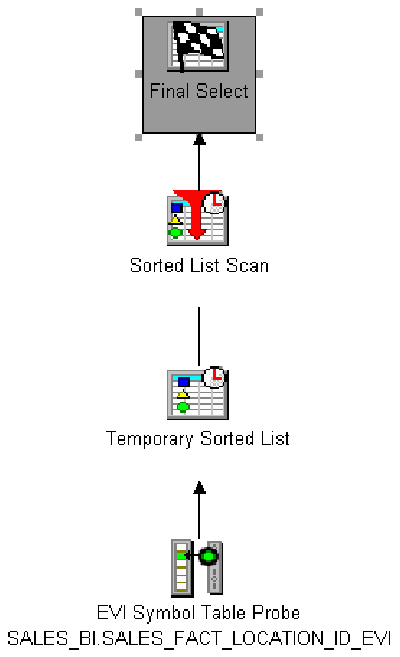
Additional non-key data
[✓] Include aggregate functions
Aggregates:
SUM(sale_amount_measure)

This example includes only a single aggregate value for the index. In practice, it is common to include more than one aggregate value when creating the index. It is not necessary to create separate EVIs for each aggregate measure that needs to be included.

Usage of the EVIs included aggregate values

After the index in Listing 3 is created, the visual explanation for the query shown in Listing 1 is shown in Figure 6.

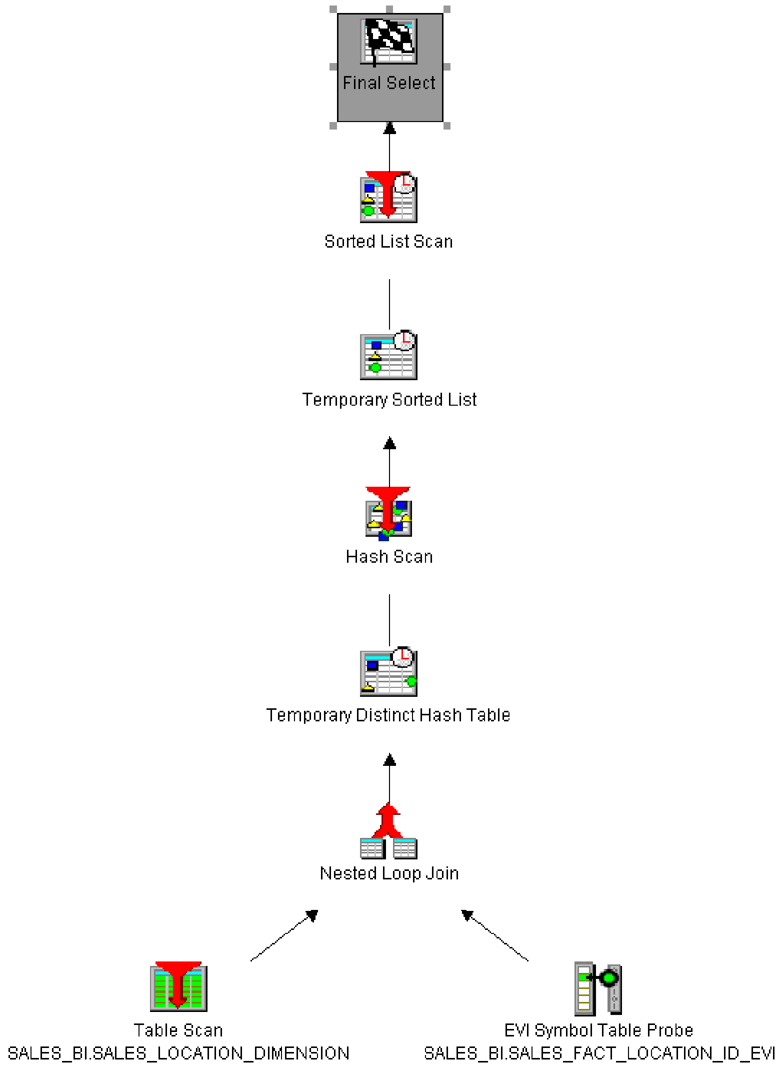
Figure 6: Visual explanation shows an EVI symbol table probe



Only the EVI's symbol table needs to be accessed, there is no need to access the fact table. This works because the symbol table now has the result of `SUM(sale_amount_measure)` maintained for each distinct value of the sale_location_id column.

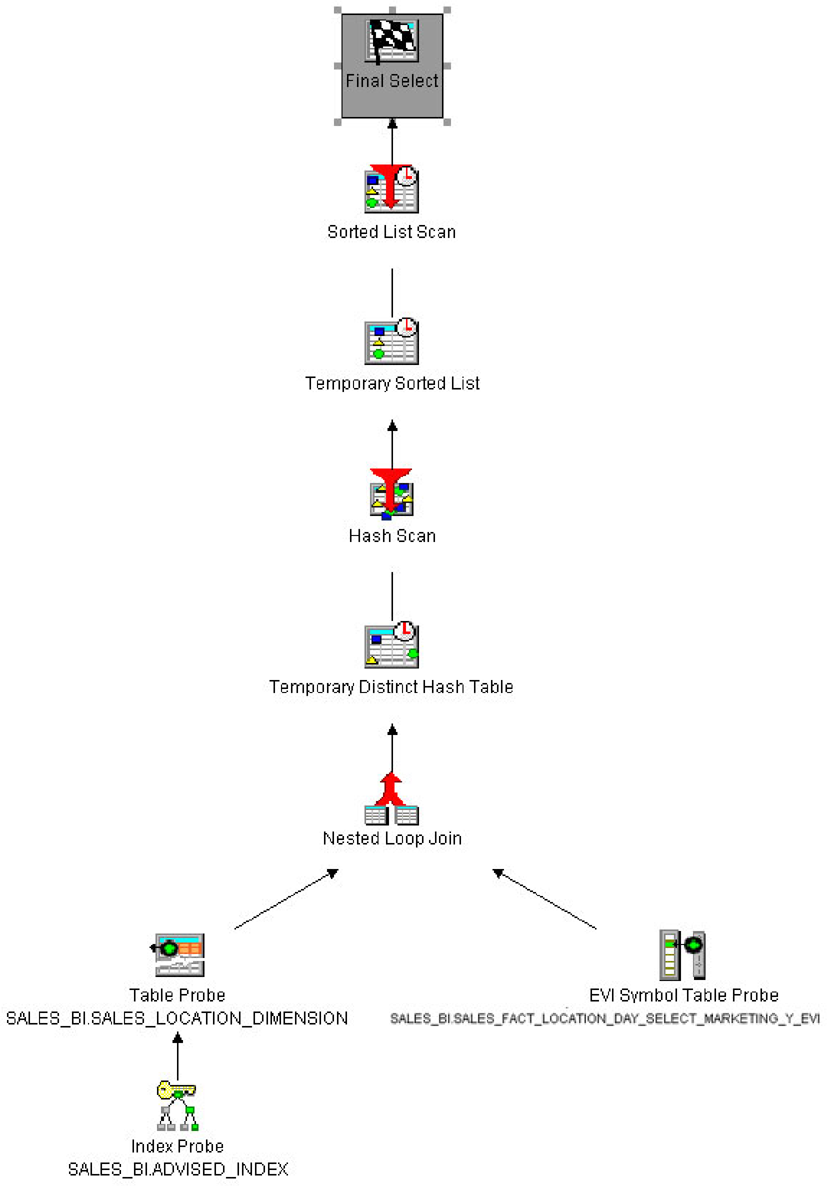
In general, the query optimizer is best able to apply this technique when the query involves a single table, because a join is evaluated before the aggregation. However, if we look at the visual explanation for Listing 2 (shown in Figure 7), we see that the optimizer is sometimes able to rewrite a query so that the join is with the EVI's symbol table. The rewrite allows the included aggregate values in the EVI to be used. The partial aggregations in the symbol table for the location IDs can be further aggregated into the values for the state grouping, without probing or scanning the fact table.

Figure 7: Visual explanation shows a join with the EVI's symbol table



Including aggregate measures in the EVI's symbol table is a component of an overall indexing strategy. Alert readers can discern that Figure 7 shows a table scan to select rows with specific state values from the sales_location_dimension table. The table scan is suboptimal because only a very small number of rows in the table match the selection criteria. As an improvement, the DB2 for i optimizer advises the creation of a binary radix over the state column. After creating the advised binary radix index, the visual explanation for the query in Listing 2 is shown in Figure 8.

Figure 8: Visual explanation after creating the advised binary radix index



Optimizing SUM() over GROUP BY ROLLUP

The next example demonstrates a common type of OLAP query where a GROUP BY ROLLUP is performed over the year, quarter, and month values that are derived from the sale_date dimension in the sales_fact table.

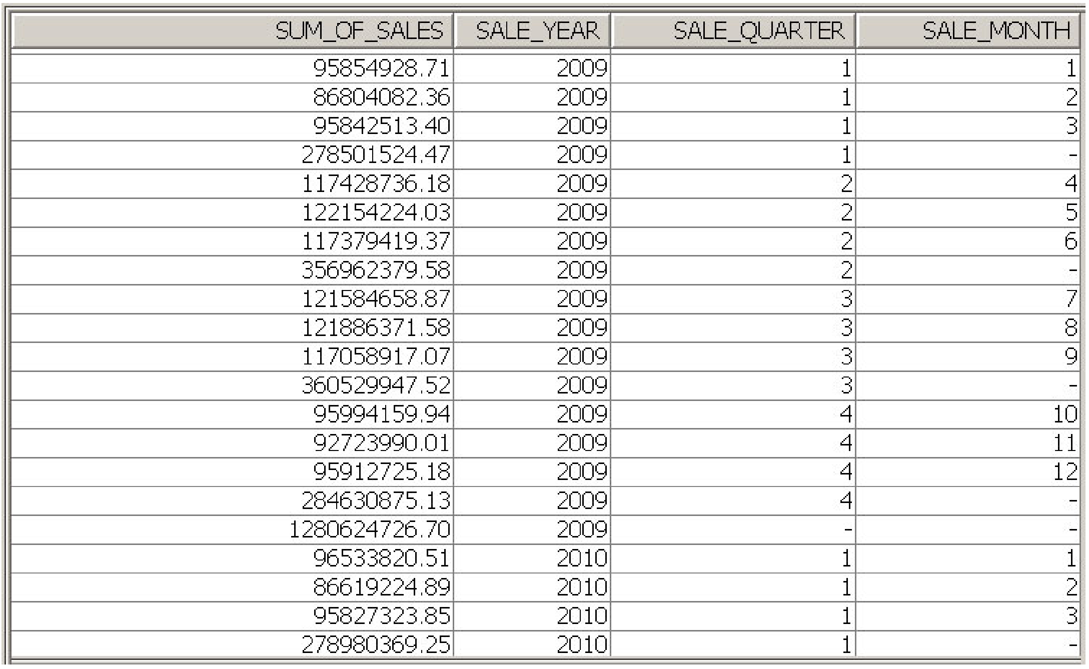
Listing 4 shows a query where the total sales is rolled up by year, quarter, and month.

Listing 4: GROUP BY ROLLUP (year, quarter, month)

```
SELECT SUM(sale_amount_measure)        AS sum_of_sales,
       YEAR(sale_date)                 AS sale_year,
       QUARTER(sale_date)              AS sale_quarter,
       MONTH(sale_date)                AS sale_month
FROM sales_fact
GROUP BY ROLLUP(YEAR(sale_date),
               QUARTER(sale_date),
               MONTH(sale_date))
ORDER BY YEAR(sale_date),
         QUARTER(sale_date),
         MONTH(sale_date)
```

Some sample results from the query in Listing 4 are (partially) shown in Figure 9.

Figure 9: Result set for GROUP BY ROLLUP (year, quarter, month)



SUM_OF_SALES	SALE_YEAR	SALE_QUARTER	SALE_MONTH
95854928.71	2009	1	1
86804082.36	2009	1	2
95842513.40	2009	1	3
278501524.47	2009	1	-
117428736.18	2009	2	4
122154224.03	2009	2	5
117379419.37	2009	2	6
356962379.58	2009	2	-
121584658.87	2009	3	7
121886371.58	2009	3	8
117058917.07	2009	3	9
360529947.52	2009	3	-
95994159.94	2009	4	10
92723990.01	2009	4	11
95912725.18	2009	4	12
284630875.13	2009	4	-
1280624726.70	2009	-	-
96533820.51	2010	1	1
86619224.89	2010	1	2
95827323.85	2010	1	3
278980369.25	2010	1	-

In other ad hoc queries, the year, quarter, and month values are likely to be used for local selection. This makes these expressions an excellent candidate key for an EVI.

The index advisor does not advise derived expressions as possible keys to an EVI, and therefore, a database designer's ability to recognize scenarios similar to this one is important. An EVI over the derived keys can make a significant impact on performance.

As the prospective EVI's key columns are also used for grouping the `SUM(sale_amount_measure)` aggregate values, it is possible to create a single EVI for the derived key expression, and include the aggregate values.

Create the EVI with derived keys and included values

The SQL statement for the CREATE INDEX statement is shown in Listing 5. This specifies the expressions as keys, and includes the additional non-key aggregate value.

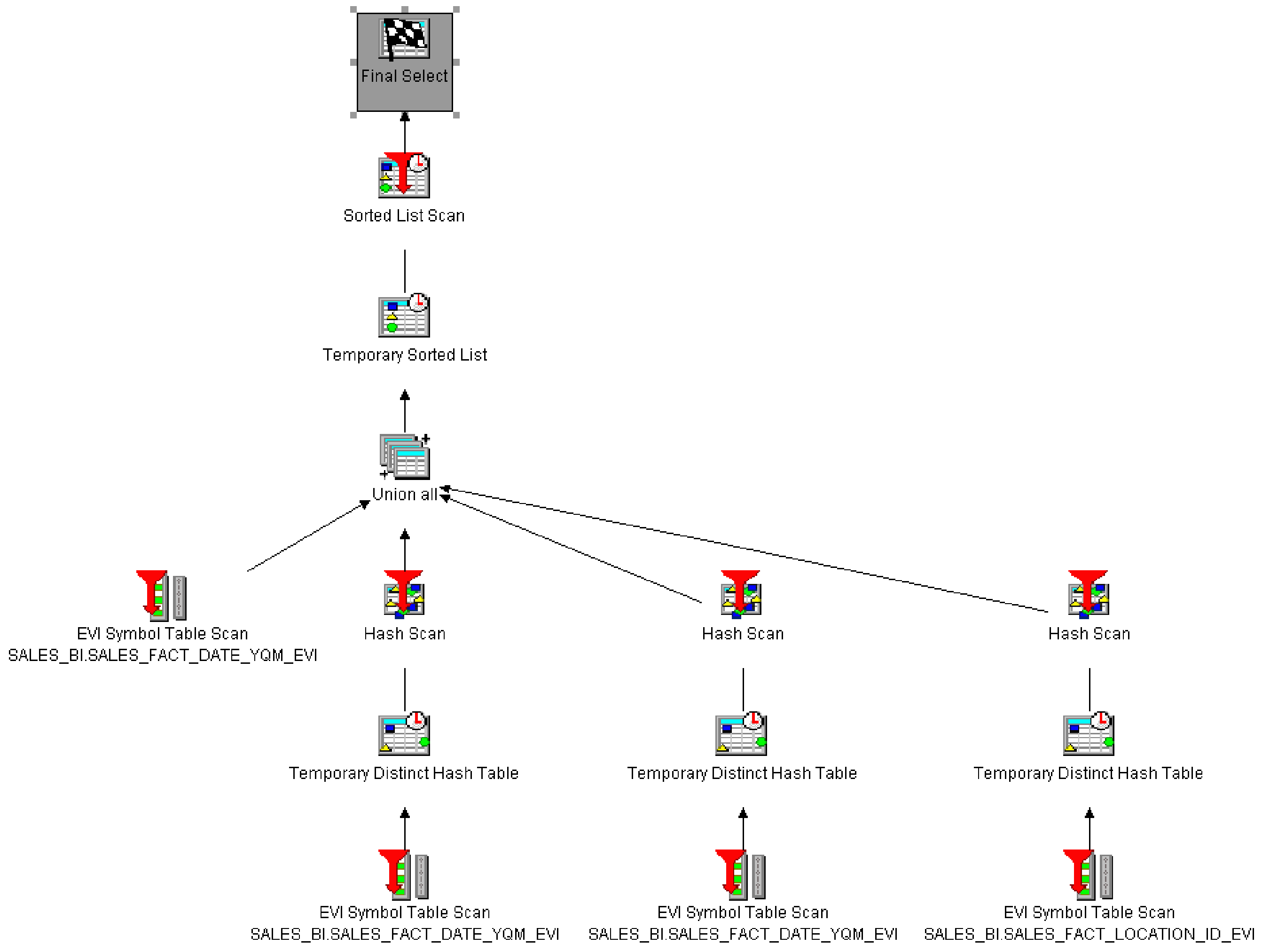
Listing 5: Create the encoded vector index over year, quarter, and month

```
CREATE ENCODED VECTOR INDEX sales_fact_date_yqm_evi
ON sales_bi.sales_fact
  (YEAR(sale_date),
   QUARTER(sale_date),
   MONTH(sale_date)
  )
INCLUDE(SUM(sale_amount_measure))
```

Usage of the EVI in GROUP BY ROLLUP

The visual explanation for the query in Listing 4 is shown in Figure 10. The query no longer needs to access the fact table, because all of the required information to perform GROUP BY ROLLUP is now in the EVI.

Figure 10: Visual explanation for GROUP BY ROLLUP using the EVI's symbol table


Final Select
Sorted List Scan
Temporary Sorted List
Union all
EVI Symbol Table Scan
SALES_BI.SALES_FACT_DATE_YQM_EVI
Hash Scan
Hash Scan
Hash Scan
Temporary Distinct Hash Table
Temporary Distinct Hash Table
Temporary Distinct Hash Table
EVI Symbol Table Scan
SALES_BI.SALES_FACT_DATE_YQM_EVI
EVI Symbol Table Scan
SALES_BI.SALES_FACT_DATE_YQM_EVI
EVI Symbol Table Scan
SALES_BI.SALES_FACT_LOCATION_ID_EVI

Similar to GROUP BY ROLLUP, the DB2 for i optimizer can use the EVI's symbol table when the grouping involves GROUP BY CUBE or GROUP BY GROUPING SETS.

The ability for the optimizer to use the EVI's included aggregates in GROUP BY ROLLUP, GROUP BY CUBE, or GROUP BY GROUPING SETS was added as part of a DB2 for i 7.1 technology update (PTF Group SF99701 Level 18). A link to the technology update's description of these capabilities is included in the References section.

Optimizing AVG() over a subset of rows

It is often a requirement in BI applications to perform aggregations over a subset of the rows in the fact table, using selection criteria that are independent of the grouping criteria. For example, a business analyst might want to perform an aggregation over the sale_amount_measure column in the sales_fact table (using one or more dimensions for grouping), but only for rows that have a 'Y' value in the customer_included_in_marketing column.

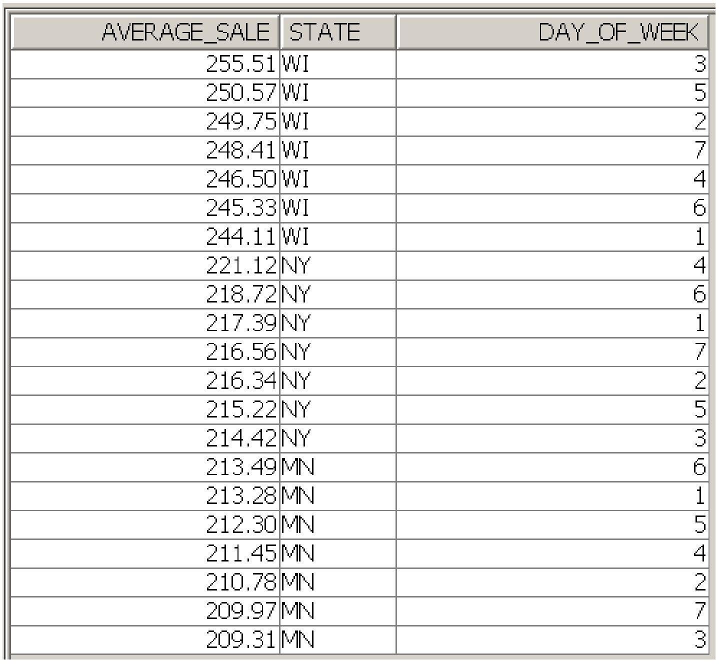
Listing 6 shows an example query that computes the average sale_amount_measure value, grouped by state and day of week, only for those sales where the purchase was made as the result of a marketing campaign. Only the states New York, Wisconsin, and Minnesota are included.

Listing 6: Average sale_amount_measure by state and day of week for included events

```
SELECT
 CAST(AVG(sale_amount_measure) AS DECIMAL(5,2)) average_sale,
 state,
 DAYOFWEEK_ISO(sale_date) day_of_week
FROM
 sales_fact
INNER JOIN
 sales_location_dimension
ON (sales_fact.sale_location_id =
    sales_location_dimension.sale_location_id)
WHERE customer_included_in_marketing = 'Y'
GROUP BY state, DAYOFWEEK_ISO(sale_date)
HAVING state IN ('NY', 'WI', 'MN')
ORDER BY average_sale DESC
```

Sample output from the query in Listing 6 is shown in Figure 11.

Figure 11: Average sales result set



AVERAGE_SALE	STATE	DAY_OF_WEEK
255.51	WI	3
250.57	WI	5
249.75	WI	2
248.41	WI	7
246.50	WI	4
245.33	WI	6
244.11	WI	1
221.12	NY	4
218.72	NY	6
217.39	NY	1
216.56	NY	7
216.34	NY	2
215.22	NY	5
214.42	NY	3
213.49	MN	6
213.28	MN	1
212.30	MN	5
211.45	MN	4
210.78	MN	2
209.97	MN	7
209.31	MN	3

Creating the EVI so that both grouping columns and selection columns are the keys for the index is an approach that would allow included aggregates to be utilized. A sample create index statement for this approach is shown in Listing 7.

Listing 7: Create EVI with grouping and selection columns as keys

```
CREATE ENCODED VECTOR INDEX sales_fact_location_day_marketing_evi
ON sales_fact
 (sale_location_id,
  DAYOFWEEK_ISO(sale_date),
  customer_included_in_marketing)
INCLUDE (AVG(sale_amount_measure))
```

The problem with the approach shown in Listing 7 is that the number of entries (and aggregate values) in the EVI's symbol table might become unnecessarily large. The symbol table contains an aggregate value for each combination of values in the EVI's key columns. If this approach is taken when the selection involves multiple columns with many possible values for each column, the EVI's symbol table can grow to a size where the usefulness of the index is significantly reduced. In addition, columns that are used for selection do not always have a finite range of distinct values, which makes the column an unrealistic choice for an EVI key.

Creating a sparse EVI

For the query in Listing 6, creating a sparse EVI that includes only rows where customer_included_in_marketing has a value of 'Y' is an appropriate solution. In general, this approach must be employed with caution; a sparse index is useful to the optimizer only when the selection criteria of the query exactly match the selection criteria used to create the index. Listing 8 shows the SQL syntax to create the sparse EVI.

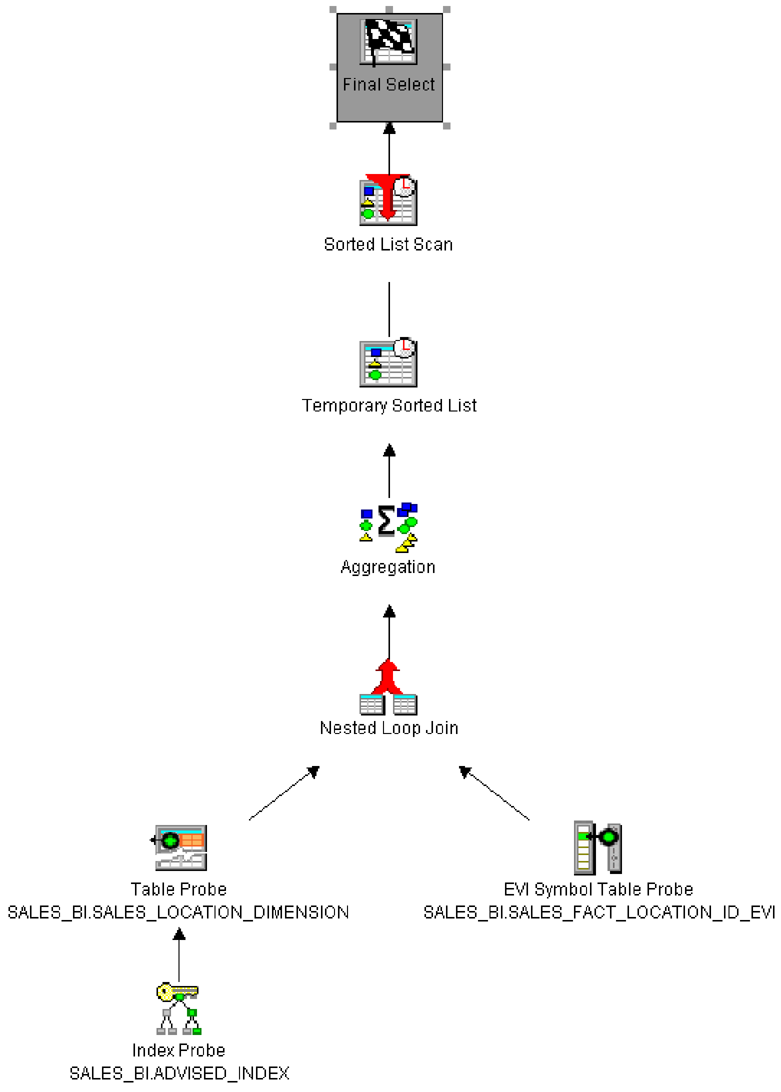
Listing 8: Create a sparse EVI with an included aggregate value

```
CREATE ENCODED VECTOR INDEX
sales_fact_location_day_select_marketing_y_evi
 ON sales_fact
 ( sale_location_id,
   DAYOFWEEK_ISO(sale_date)
 )
WHERE customer_included_in_marketing = 'Y'
INCLUDE (AVG(sale_amount_measure))
```

Usage of a sparse EVI for aggregations

Figure 12 shows the visual explanation for the query in Listing 6, after the creation of the sparse EVI in Listing 8. The sparse EVI's symbol table contains the average values (over the location ID) for rows matching the WHERE clause of the query. As a result, there is no need to access the fact table to implement the query.

Figure 12: Visual explanation for a query that uses a sparse EVI



Index usage when the included values are not useful

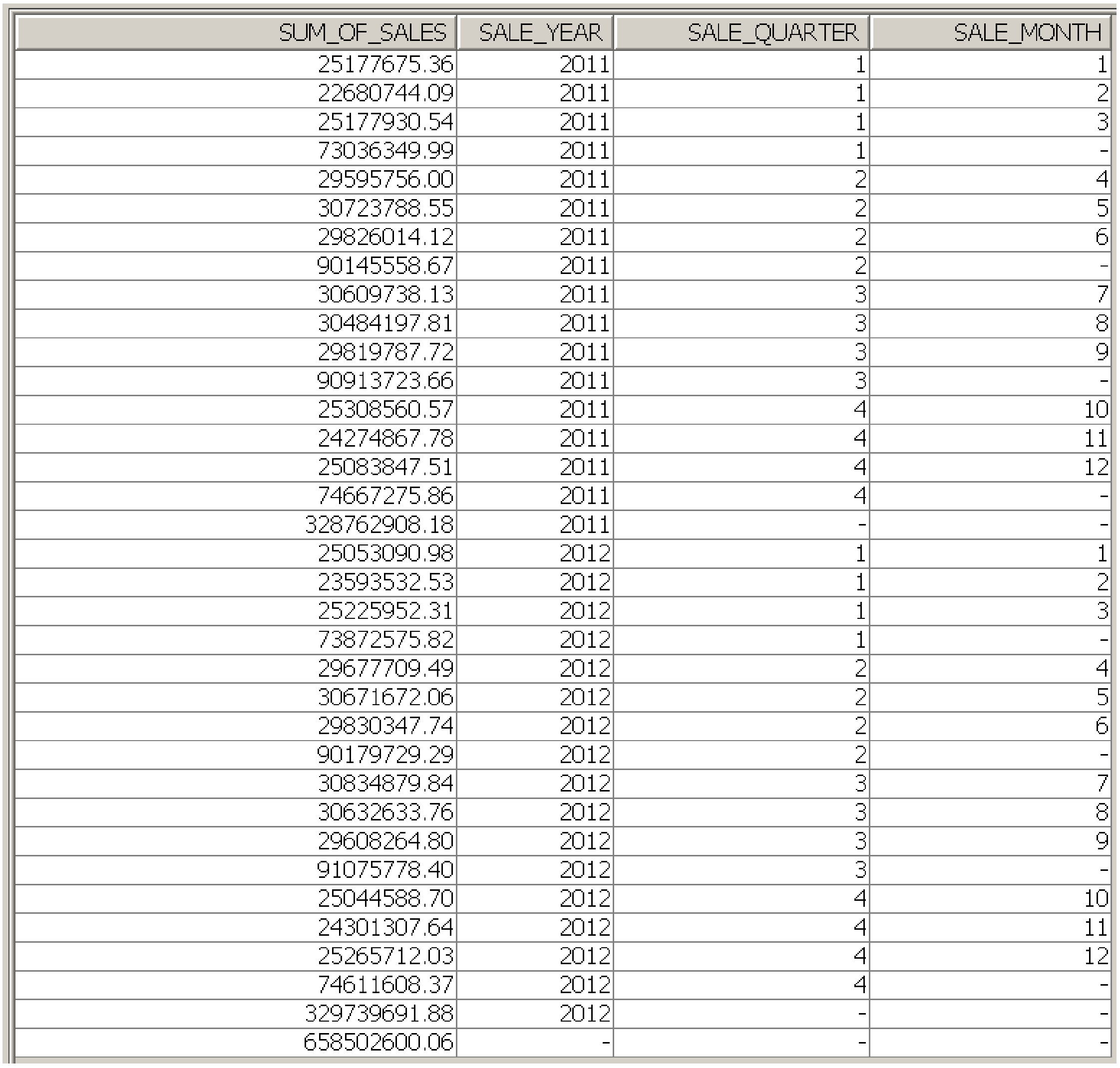
A query cannot always be implemented using aggregate values that are included in an EVI. An example query where the included aggregate values do not help is shown in Listing 9. This query contains a GROUP BY ROLLUP clause that is similar to the query in Listing 4, however the aggregation includes only rows that have a sale_date that is in the 2011 or 2012 years, and a region that has a value of 'EAST'.

Listing 9: Query that does not benefit from included aggregates

```
SELECT SUM(sale_amount_measure) AS sum_of_sales,
       YEAR(sale_date)          AS sale_year,
       QUARTER(sale_date)       AS sale_quarter,
       MONTH(sale_date)         AS sale_month
FROM sales_fact
INNER JOIN sales_location_dimension
      ON (sales_fact.sale_location_id =
          sales_location_dimension.sale_location_id)
WHERE YEAR(sale_date) IN (2011, 2012)
      AND region = 'EAST'
GROUP BY ROLLUP(YEAR(sale_date),
                QUARTER(sale_date),
                MONTH(sale_date))
ORDER BY YEAR(sale_date),
         QUARTER(sale_date),
         MONTH(sale_date)
```

Sample output from Listing 9 is shown in Figure 13.

Figure 13: Sales for east region in 2011 and 2012, rolled up by year, quarter, and month

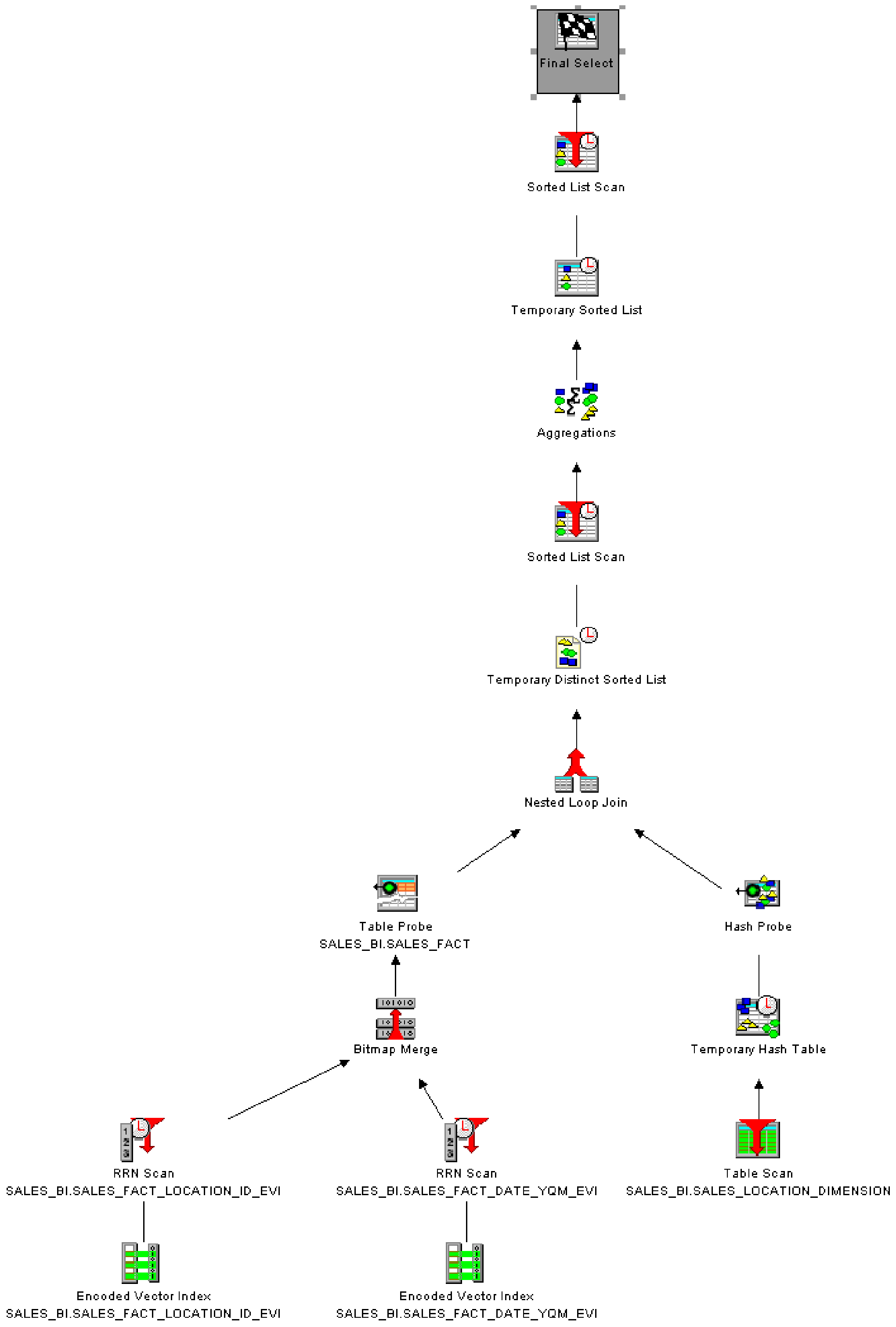


SUM_OF_SALES	SALE_YEAR	SALE_QUARTER	SALE_MONTH
25177675.36	2011	1	1
22680744.09	2011	1	2
25177930.54	2011	1	3
73036349.99	2011	1	-
29595756.00	2011	2	4
30723788.55	2011	2	5
29826014.12	2011	2	6
90145558.67	2011	2	-
30609738.13	2011	3	7
30484197.81	2011	3	8
29819787.72	2011	3	9
90913723.66	2011	3	-
25308560.57	2011	4	10
24274867.78	2011	4	11
25083847.51	2011	4	12
74667275.86	2011	4	-
328762908.18	2011	-	-
25053090.98	2012	1	1
23593532.53	2012	1	2
25225952.31	2012	1	3
73872575.82	2012	1	-
29677709.49	2012	2	4
30671672.06	2012	2	5
29830347.74	2012	2	6
90179729.29	2012	2	-
30834879.84	2012	3	7
30632633.76	2012	3	8
29608264.80	2012	3	9
91075778.40	2012	3	-
25044588.70	2012	4	10
24301307.64	2012	4	11
25265712.03	2012	4	12
74611608.37	2012	4	-
329739691.88	2012	-	-
658502600.06	-	-	-

The query shown in Listing 9 is an example of a common ad hoc BI query where the aggregate values included in indexes won't be used, because the selection and grouping criteria is not limited to information within the symbol table of any single EVI that has been created. However, the EVIs that were created in Listing 3 and Listing 5 still have significant overall value for the query.

Figure 14 shows the visual explanation for the query in Listing 9. Although the aggregations will not be performed using an EVI's included aggregate values, the EVIs in Listing 3 and Listing 5 can be used to generate a relative record number (RRN) list of rows in the fact table that satisfy the predicates in the selection. The RRN list allows the query to access only the pages that have interesting rows, and this can significantly speed up the performance of the query. Because maintaining the included aggregate values has a minimal effect on the index maintenance cost, including this additional information for the other queries that can use it is still beneficial for the application.

Figure 14: Visual explanation for a query that cannot use included aggregate values



Summary

The ability to include aggregate values in an EVI's symbol table is a powerful feature in DB2 for i 7.1 that can help to improve the performance of a BI or analytics application. This article has presented several examples where it is beneficial to include aggregate values in an encoded vector index. Creating indexes with included aggregates is an excellent idea, because the maintenance cost of maintaining the aggregate values is minimal, and the performance gains can be tremendous.

Understanding and maximizing all of DB2 for i SQL's features requires education, training, and experience; the result is an outstanding return on the investment. The Systems and Technology Group Lab Services DB2 for i Center of Excellence team offers many excellent educational opportunities, including a *fantastic* DB2 for i SQL performance workshop. The References section includes a link to the workshop's website, and a link to the IBM Lab Services and Training website.

Acknowledgements

A special thanks to the Systems and Technology Group Lab Services DB2 for i Center of Excellence and the DB2 for i optimizer teams for their review and comments on this article.

References

- Star-schema join support within DB2 for i
- IBM DB2 for i indexing methods and strategies
- Database performance and Query Optimization
  - Encoded Vector Indexes
  - Recommendations for EVI use
  - Viewing your queries with Visual Explain
- DB2 for IBM i Technology updates
  - Index Advisor, Show Statements - improved query identification
  - Fast index only access for CUBE(), ROLLUP(), and GROUPING SETS()
- IBM Lab Services and Training
  - DB2 for i Center of Excellence solution brief
  - DB2 for i SQL Performance Workshop