Star-schema join support within DB2 for i

Michael W. Cain
DB2 for i Center for Excellence
IBM Systems and Technology Group
December 2011

Table of contents

Abstract

This paper explains how IBM DB2 for i supports star-schema join queries in a business intelligence environment.

Introduction

This paper provides information on a special IBM® DB2® for IBM i® optimization strategy for star-schema and snowflake-schema join queries. The paper does not attempt to introduce or explain database query optimization techniques or implementation methods.

It is strongly recommended that database administrators, analysts and developers who are either new to the IBM i platform or unfamiliar with SQL programming techniques attend the “DB2 for i SQL and Query Performance Monitoring, Analysis and Tuning” workshop. This education covers the proper way to architect and implement a high-performing DB2 for i solution. More information regarding this course can be found at: http://www.ibm.com/systems/i/software/db2/db2performance.html.

Overview of star schema

There are various kinds of data models that are used to support business requirements. In some environments, a specialized data model exists for business analysis. This data model is commonly referred to as a star schema. A star-schema model contains a centralized table (known as the “fact table”) and is surrounded by highly normalized tables (known as “dimension tables”). The name “star schema” reflects the fact that the data model looks like a star; the dimension tables appear as points of a star surrounding the fact table (see Figure 1).

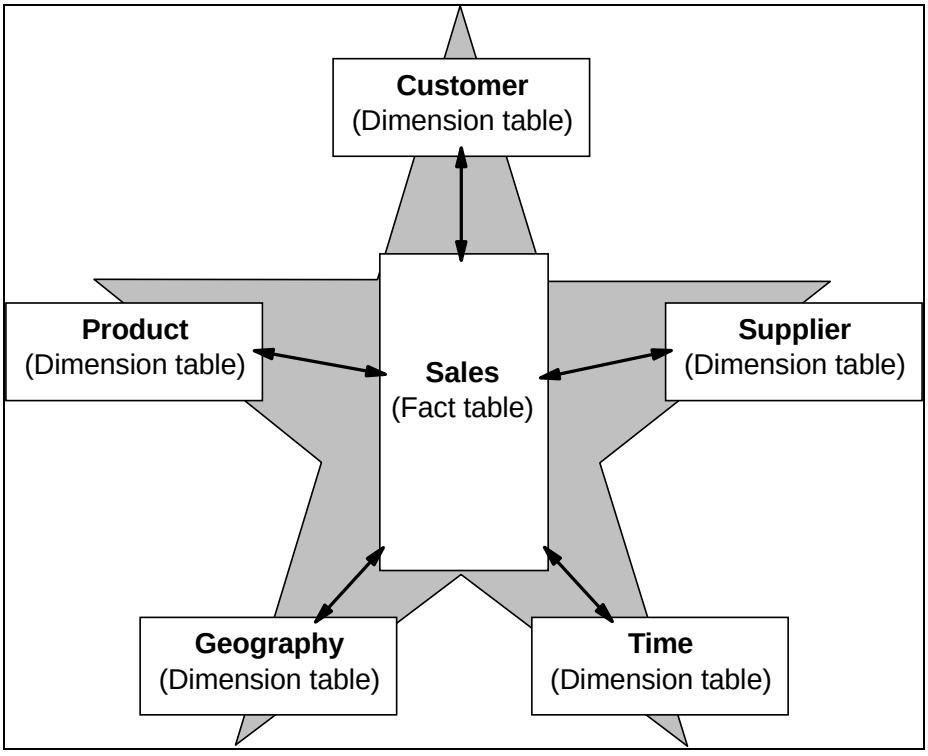

Figure 1. The star-schema model

Star-schema database models and star-schema join queries are primarily found and used in data warehousing and business intelligence environments. Most, if not all, of the major software and solution providers who support analytical processing with a relational database management system use some type of star-schema database model and SQL-based, star-schema join queries. These types of SQL queries can be difficult to optimize and execute unless the optimizer and database engine have specific techniques for recognizing, optimizing and running the star-schema joins.

The attributes of a star-schema database model usually include:

- A relatively large fact table containing millions or billions of rows that hold the measurable or additive facts (such as sales types of transactions or events)
- Relatively small and highly normalized dimension tables containing descriptive data about the facts in the central fact table (such as customer or location information)
- A central fact table that is dependent on the surrounding dimension tables using a parent / child relationship, with the fact table as the child and the dimension tables as the parent

If the dimension tables are further normalized, the results are dimensions that might have additional tables supporting them. This is known as a “snowflake” schema or model (see Figure 2).

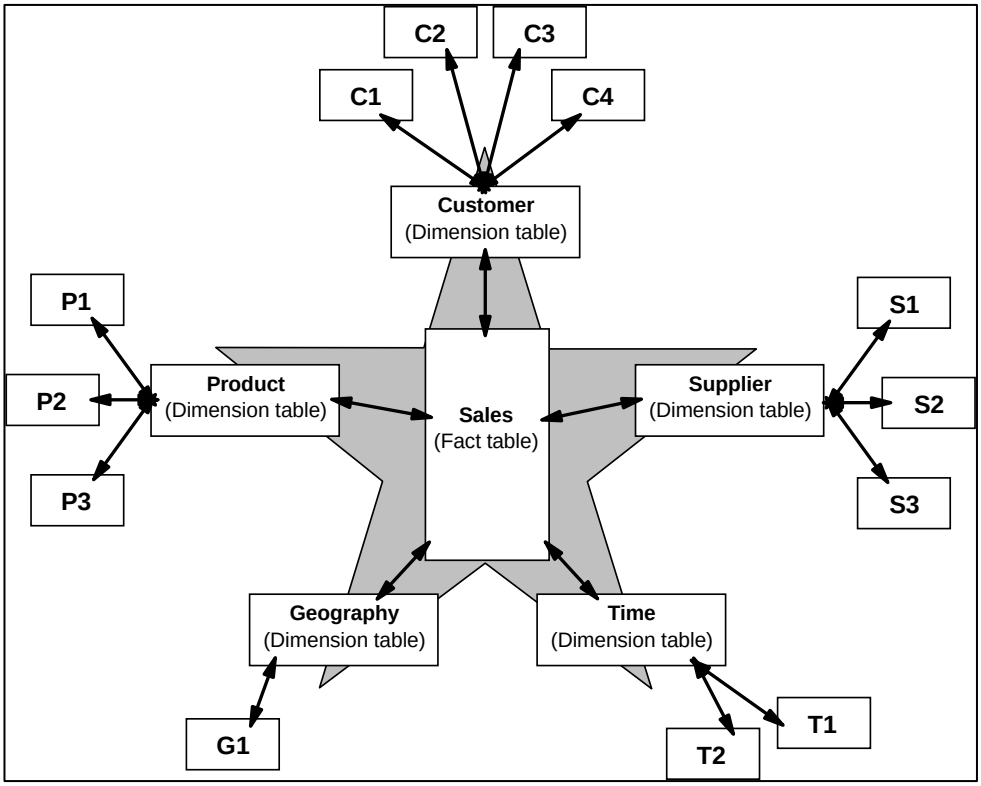

Figure 2. Snowflake-schema model

Star-schema join queries

A star-schema join query is usually a multidimensional request in support of online analytical processing (OLAP). An example of this type of analysis is a request to show sales figures for a given customer, in a given year, in a given month and for a given product. The result set represents the sales figures in the fact table found at the intersection of the specified customer, year, month and product (see Figure 3).

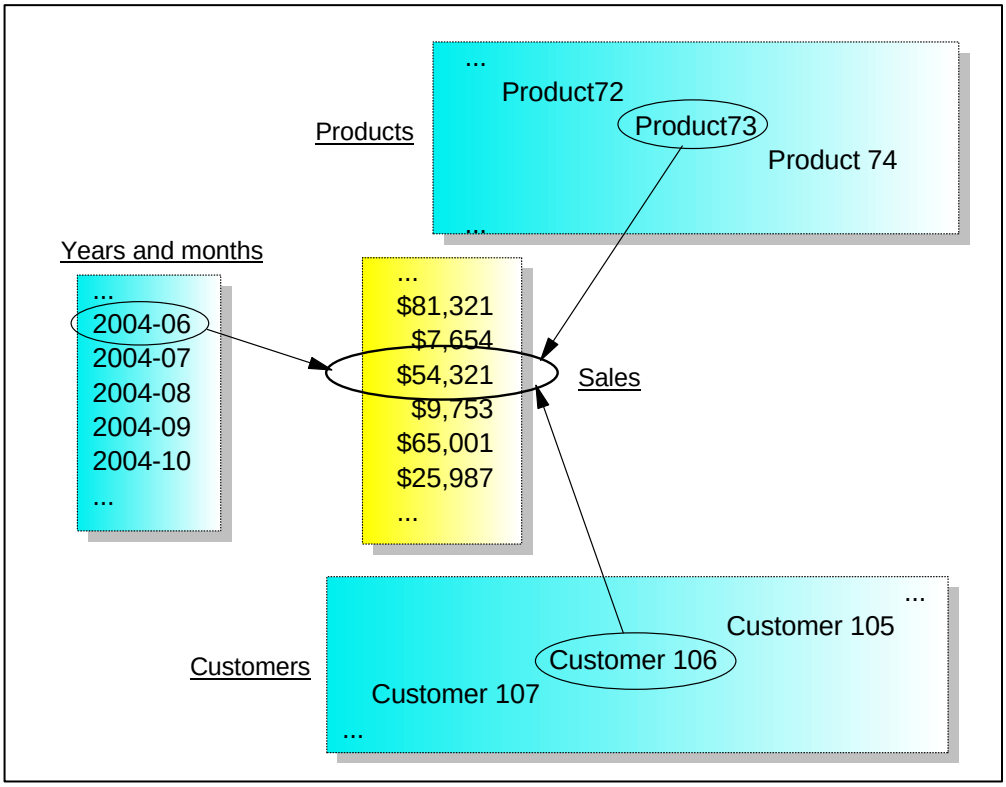

Figure 3.The result set of a star-schema join query

OLAP queries have different business requirements and thus, place different technical requirements on the query optimizer and database engine than online transaction processing (OLTP) queries. Where the OLTP environment might produce a large number of relatively short duration queries, the OLAP environment can produce a relatively small number of queries that process very large volumes of data. In addition, the OLAP queries can be ad-hoc and unpredictable in nature.

In a normalized star-schema model, the fact table does not contain the descriptive information, only transaction data or events (the measurable or additive facts). To decode or describe the data in the fact table, each row must be joined to the relevant dimension table(s). This normally means that the user relies on the dimension tables not only to describe the information, but also to select the data and the associated facts from the fact table.

The properties of a star-schema join query include:

- Multiple tables participating in the query
- Local selection predicates on the dimension tables rather than the larger fact table
- Equi-join predicates between the dimension tables and the fact table used to locate and select the relevant fact table rows and to decode and describe the fact-table data
- The equi-join predicate between a one-dimension table and the fact table can result in the selection of a very large number of fact-table rows. In contrast, the intersection of the equi-join predicates of multiple-dimension tables can result in the selection of a relatively small number of fact table rows.

SQE optimizer enhancements for star-schema join queries

The DB2 for i SQL Query Engine (SQE) automatically supports optimizing star-schema join queries. This support does not require the use of the QAQQINI options or any other “configuration” parameters. Recognizing and optimizing a star-schema query is just a normal part of handling any query request.

The key feature of this SQE support is referred to as “look-ahead predicate generation” or “LPG.” The patented SQE LPG support has some specific benefits and uses which are described and illustrated below.

Support details for star-schema joins

The following diagrams and accompanying descriptions explain how the star-schema join support works. Although not explicitly described, the optimizer and database engine can choose to use parallel methods for selecting data from the dimension tables, building the temporary hash tables, accessing the EVIs, building the bitmaps or relative record number (RRN) lists, and selecting the data from the fact table. These parallel methods are available when the optional DB2 Symmetrical Multiprocessing feature is installed and enabled. For details on the technology and benefits of DB2 SMP see: ibm.com/partnerworld/wps/servlet/ContentHandler/servers/enable/site/education/ibp/4aea/index.html

Given a star-schema model with Fact_Table supported by three dimension tables: Item_Dim, Store_Dim and Date_Dim; a multidimensional query is issued to find sales and quantity figures with local selection on items, stores and dates (see

Figure 4). There is no local selection on the fact table.

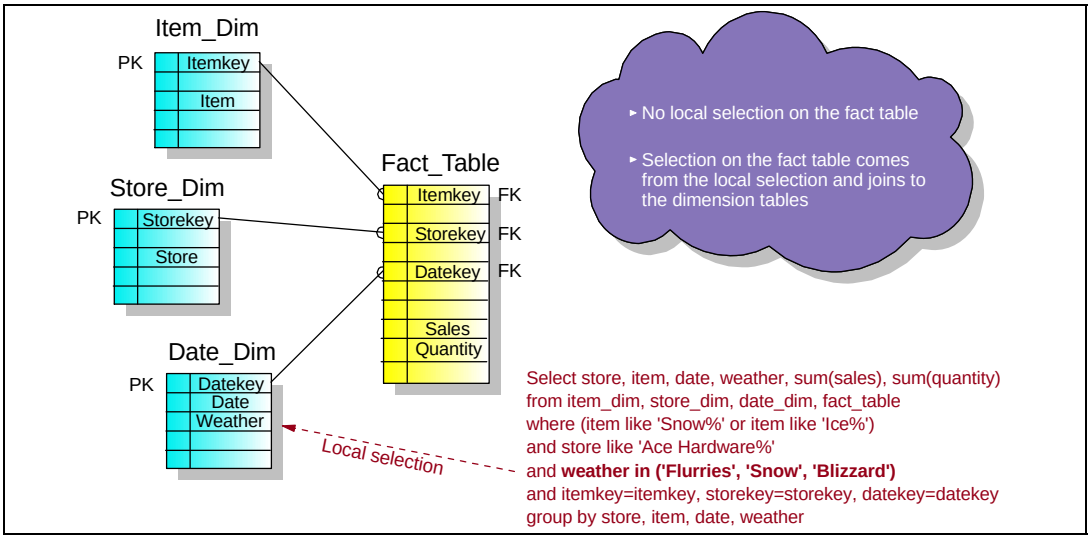

Figure 4. Issuing a multidimensional query

The ad-hoc query requests sales for any winter items (for example, Snow shoes, Snow shovels, Ice melting devices and Ice chippers) sold in any Acme Hardware store when the weather was wintery (that is, Flurries, Snow and Blizzards) during some time period. Thus, any itemkey, storekey or datekey that matches the criteria is a candidate for the join to the fact table.

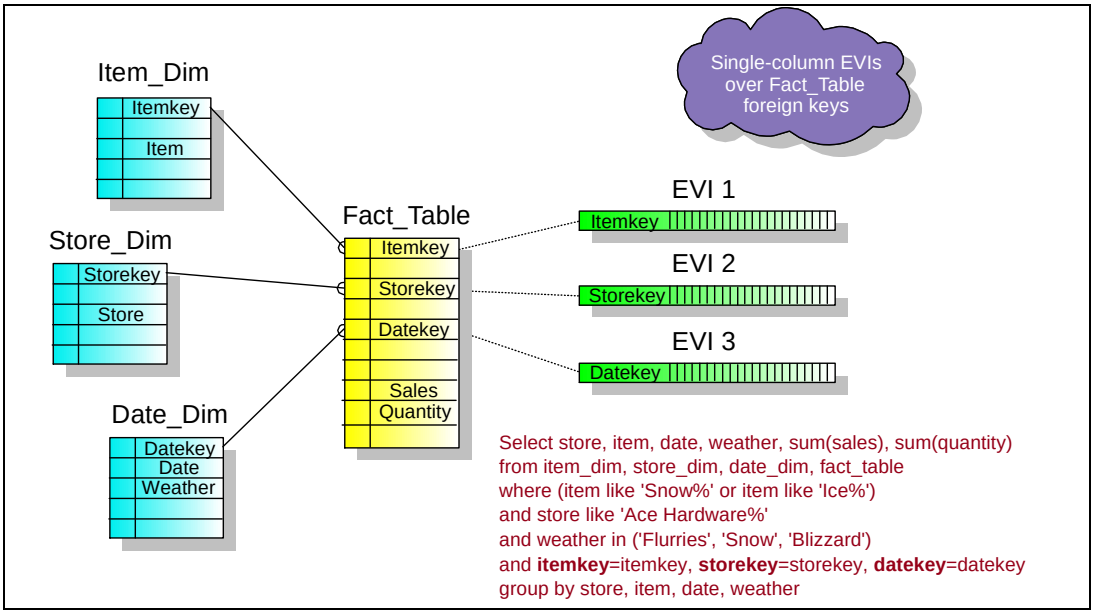

Figure 5. Ad-hoc query for winter items

Single-column EVIs that are created over the foreign key columns of the fact table are usually optimal for the SQE optimizer to implement its star-schema join techniques. If the column is unique or has high

cardinality then a radix index is best. If there are referential constraints on the foreign keys in the fact table, then DB2 will have the radix indexes are already in place to support the constraint validation process. Implementing constraints is considered a data-centric processing best practice.

As part of the join optimization process, SQE identifies the largest table in the query (that is, the fact table) and optimizes the join order of all the tables based on new strategies and methods. This can cause the fact table to be placed somewhere other than the first join position. Since the fact table might be "joined to" from another dimension table, it is also advantageous to create single-key radix indexes on the foreign key columns of the fact table. Using this index might be optimal if the fact table is joined from a dimension table in join position 1.

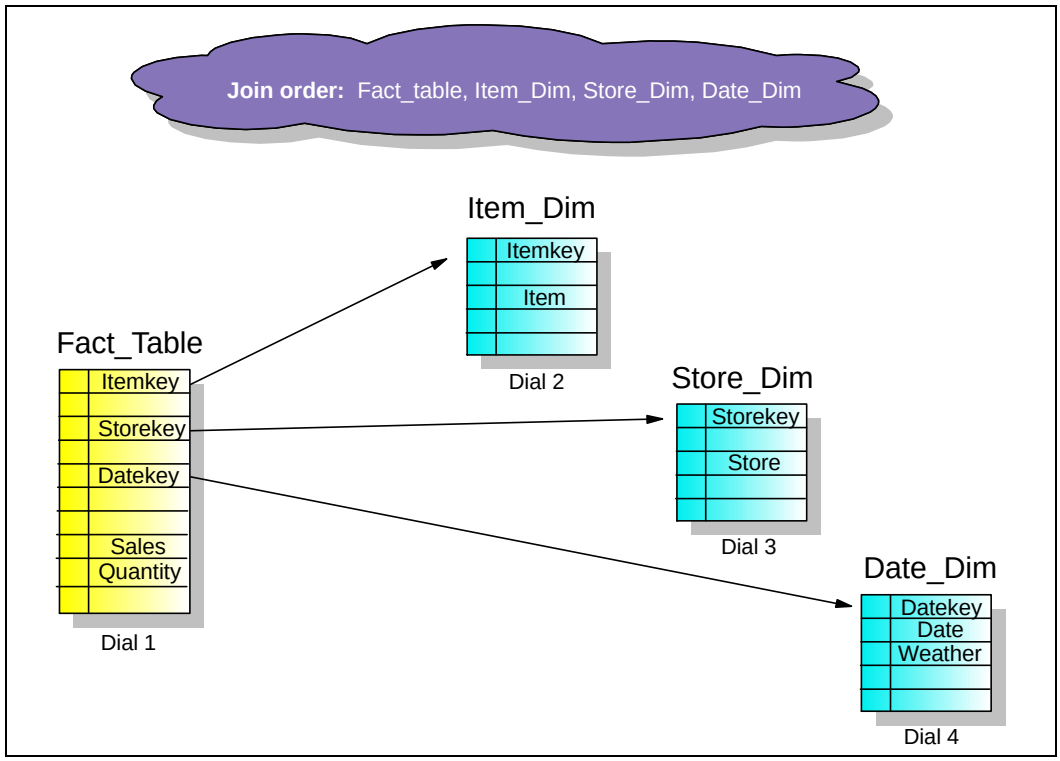

Figure 6. Query join orders

When the query is optimized, hash join is normally chosen as the method for joining the fact table to the dimension tables. To accomplish this, the original query is broken up into multiple parts or steps. For each dimension table, an internal query runs to access the rows that match the local selection predicates, using the best available access method (scan or probe, with or without parallelism). The data required for the query is then used to build a temporary hash table. The original dimension table is no longer required by the query – all of the relevant and required data is in the temporary data structure, which is likely memory resident.

In addition to building the hash tables, the join-column values of the selected dimension table rows are used to populate a list of distinct keys (see Figure 7). This list represents all the join-column values (that match the corresponding local selection) and is used to identify the join-column values in the fact table.

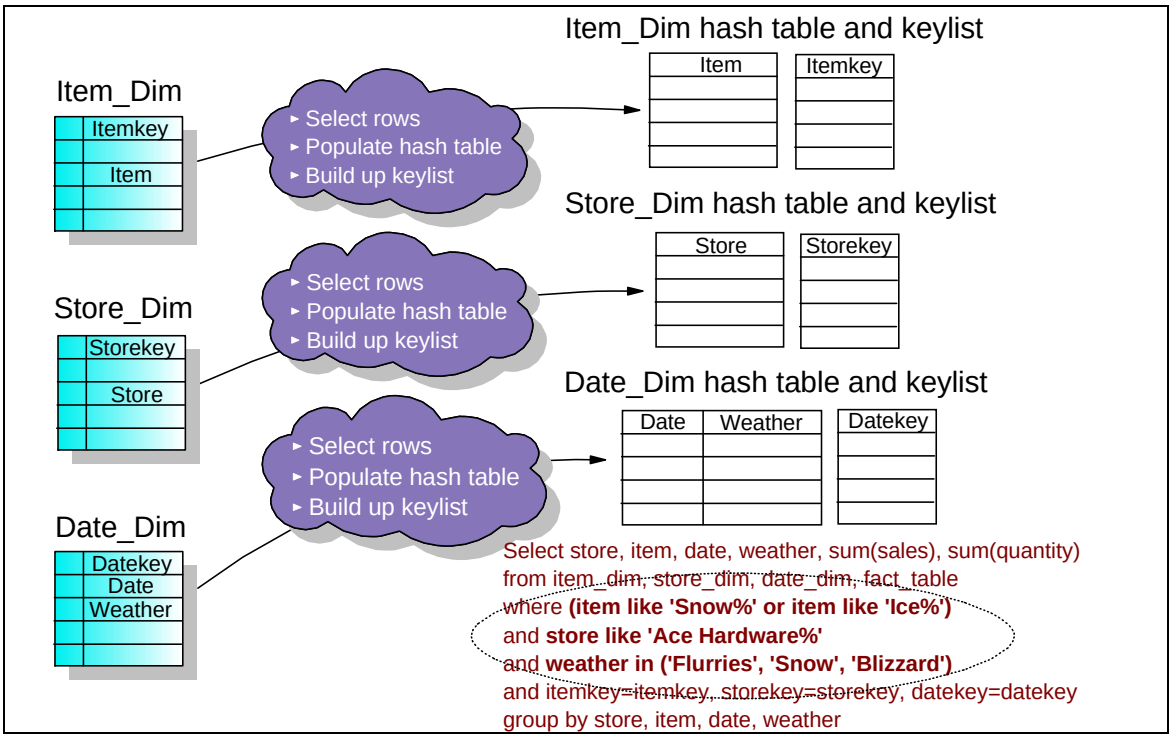

Figure 7. Join-key and join-column values

After the hash tables and distinct key lists are built, the original query is rewritten. The distinct key lists are used to provide local selection on the fact table. This has the effect of transferring the local selection from the dimension tables to the fact table and is referred to as “look-ahead predicate generation” or LPG.

The query is further optimized using the generated local selection predicates on the foreign key columns of the fact table. The optimizer recognizes and uses the EVIs that are created over the join column(s) for statistics and possibly for data access. If an EVI is chosen for implementation, a dynamic bitmap or RRN list is created from the respective EVI. The bitmap or RRN list now represents the local selection on the corresponding join key column. The bitmaps or RRN lists are merged to create a final bitmap or RRN list. The final bitmap or RRN list represents the intersection of all the local selections (see Figure 8). In other words, only the rows that match the local selection are identified in the bitmap or RRN list. Any other local selection that is not implemented with an EVI is handled when reading the table.

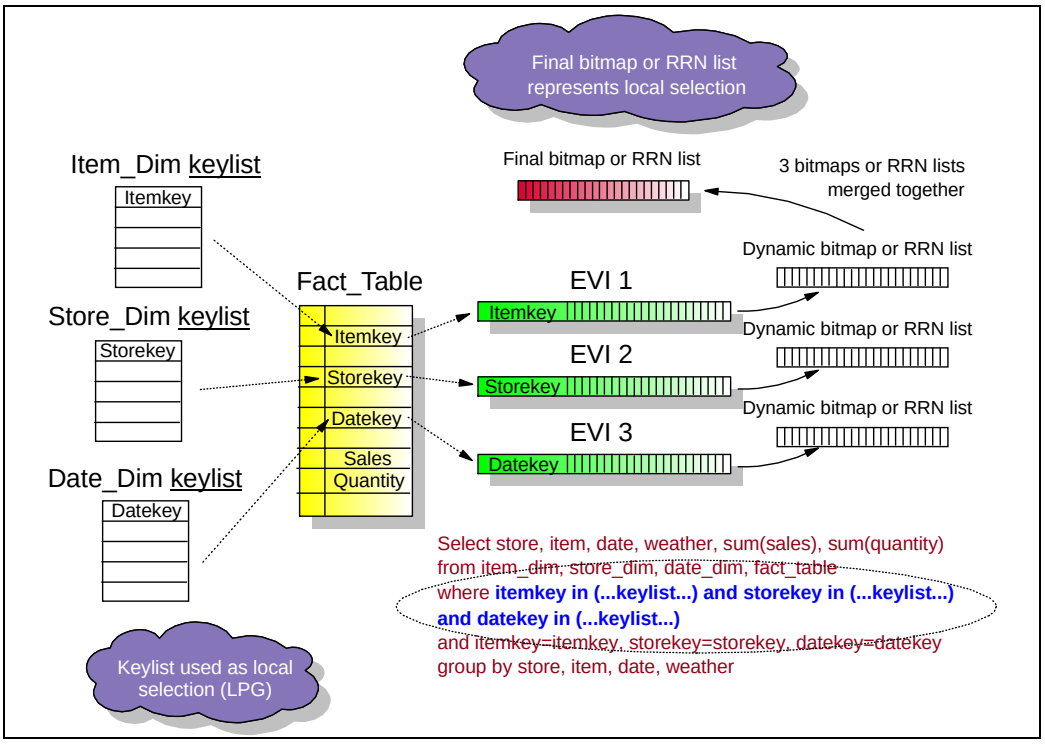

Figure 8. Intersection of local selections

To complete the star-schema join query, the database engine uses the final bitmap or RRN list to perform skip-sequential or clustered-table probe processing on the fact table (see Figure 9). This allows for efficient and fast reading of the large fact table by skipping over the rows that are not part of the query request. If a row is selected, the join is completed by using the join-column values to access the hash tables (that were created earlier from the dimension tables).

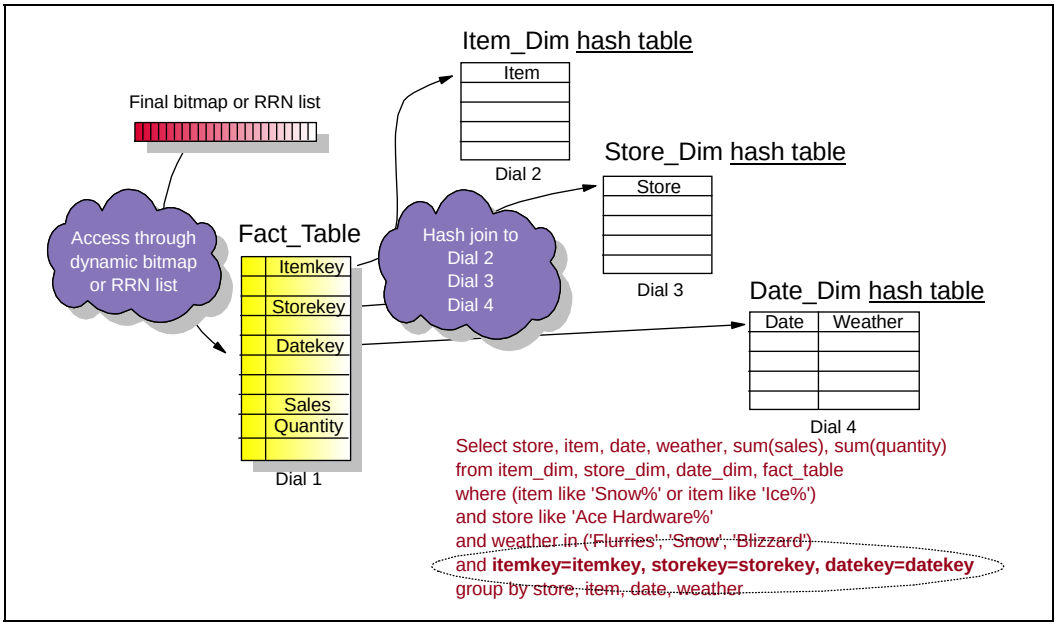

Figure 9. Using a final bitmap or RRN list

The advanced techniques of SQE allow for look-ahead predicate generation to occur anywhere in the query and on any table (see Figure 10). This supports the mixing of methods to provide the best strategy for each part of the query. For example, a highly selective dimension table can be placed first in the join order, followed by the fact table, and then the other dimensions. The SQE LPG support can be used to minimize I/O processing against the fact table in join position two, in addition to the join operation from the dimension table in position one.

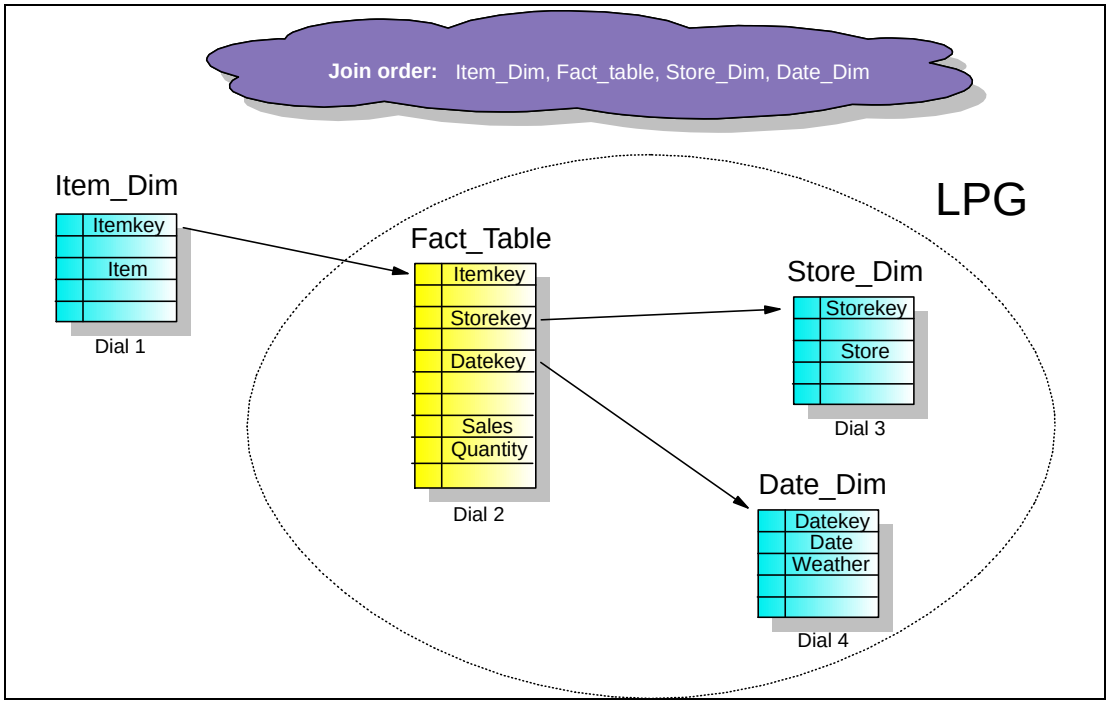

Figure 10. Mixing methods to find the best query strategy

Without the star-schema join support (that is, LPG), the optimizer either chooses a full table scan on the fact table (as the primary) with hash join or index join to the dimension tables, or the optimizer chooses to use one of the dimension tables in the primary join position and use index join to the fact table. These other access plans can require much more I/O resources and result in a slower query response time.

Snowflake schemas

To achieve the benefits of the star-schema join support with a snowflake-schema model, the developer must consider where the join relationships are and where the generated local selection predicates are placed within the query. If the local selection is on the outer-most dimension tables, indexes must be created on the foreign keys of the inner dimension or transformation tables. This allows the DB2 optimizer to generate and transfer the local selection from the outer dimension tables to the inner dimension tables, and then from the inner dimension tables to the fact table.

The simple snowflake model seen in Figure 11, you can see relationships between the outer dimension tables (Location and Season) and the inner dimension tables (Store and Date).

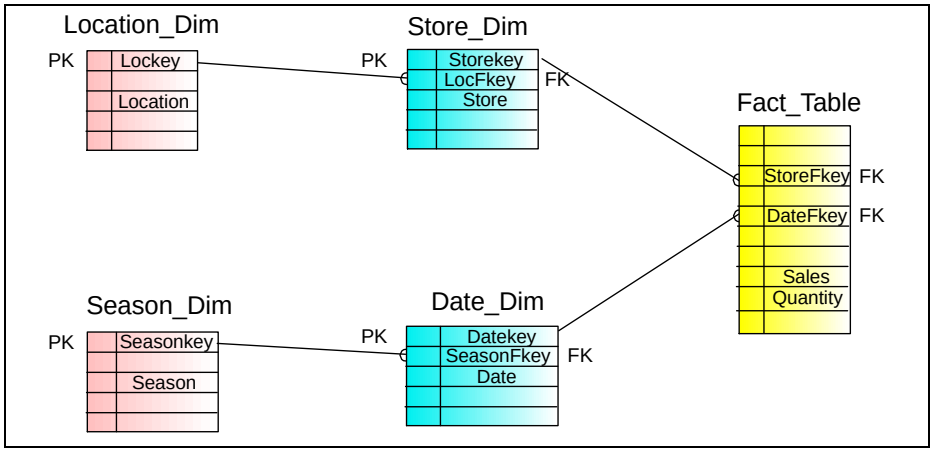

Figure 11. Simple showflake model

Given these relationships and the probability of local-selection predicates on the outer-dimension tables, single-column EVIs are created over the foreign-key columns of the inner-dimension tables and are used by the optimizer to implement the star-join techniques (see Figure 12).

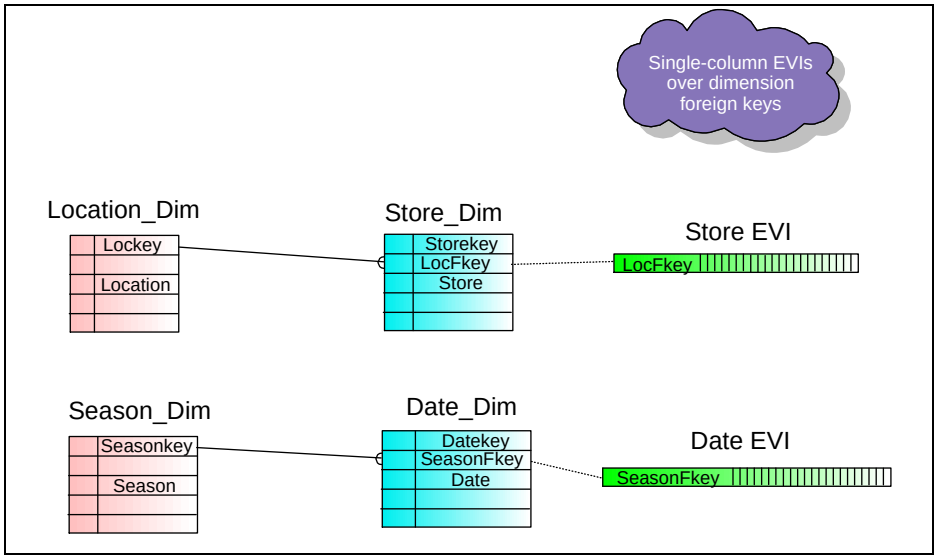

Figure 12. Single-column EVIs over dimension foreign keys

To complete the snowflake model indexing strategy, single-column EVIs are created over the foreign-key column of the fact table (see Figure 13).

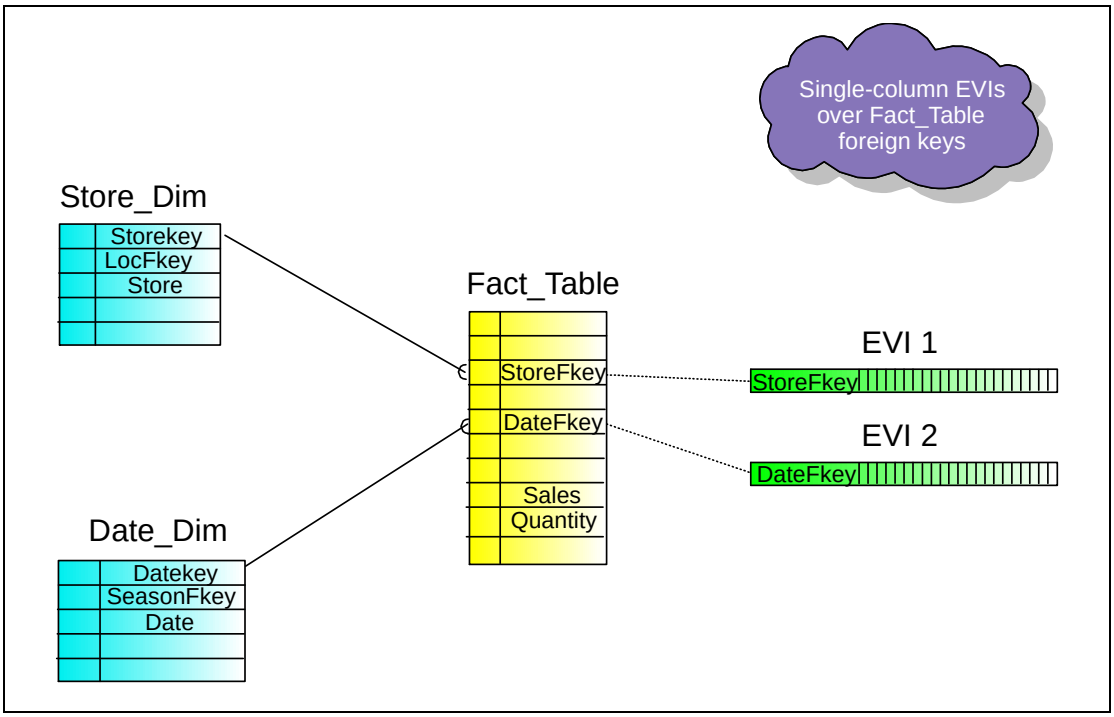

Figure 13. Single-column EVIs over the FACT_Table foreign keys

Star-schema join support requirements

The star-schema support is inherent within DB2 for i. No additional features or functions are required.

Limitations and considerations

Factors such as processor model, number of processors or cores, memory pool size and number of disk units affect the optimizer's costing, determination of the join method, and the effectiveness of the access plan. This includes the star-schema join support.

Single-column EVIs or radix indexes must be created over the join columns of the fact table. LPG is not is not used if the relationship between two tables, i.e. the join is based on more than one column. When designing the star-schema data model, particular attention should be paid to the table relationships.

System i Navigator Visual Explain can be used to illustrate and understand a query plan. To verify that the LPG support is used by the SQE optimizer, check the implementation of the query by reviewing the access plan rendered in Visual Explain. For example, check to see if one or more indexes are used to populate RRN lists. Check the join order and look for temporary hash tables used to facilitate the join(s).

Clicking **Highlight LPG** in the View menu will highlight the icons that are using the look-ahead predicate generation (LPG) (see Figure 14).

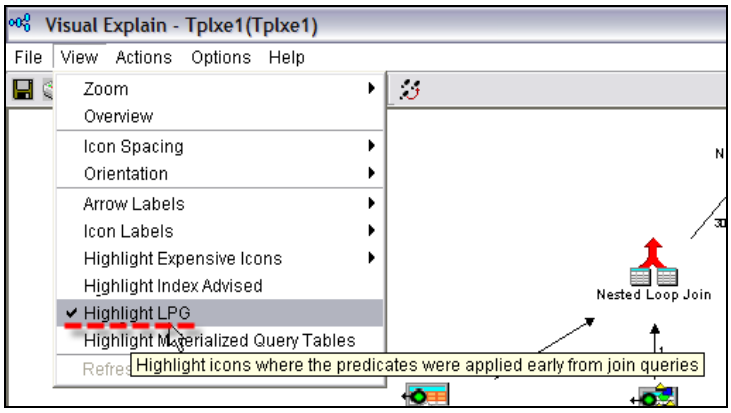

*Figure 14. Click **Highlight LPG** to see the processes that use look-ahead predicate generation*

In Figure 15, two local-selection predicates were generated, and the optimizer chooses two corresponding EVIs for the identification of rows that match this local selection.

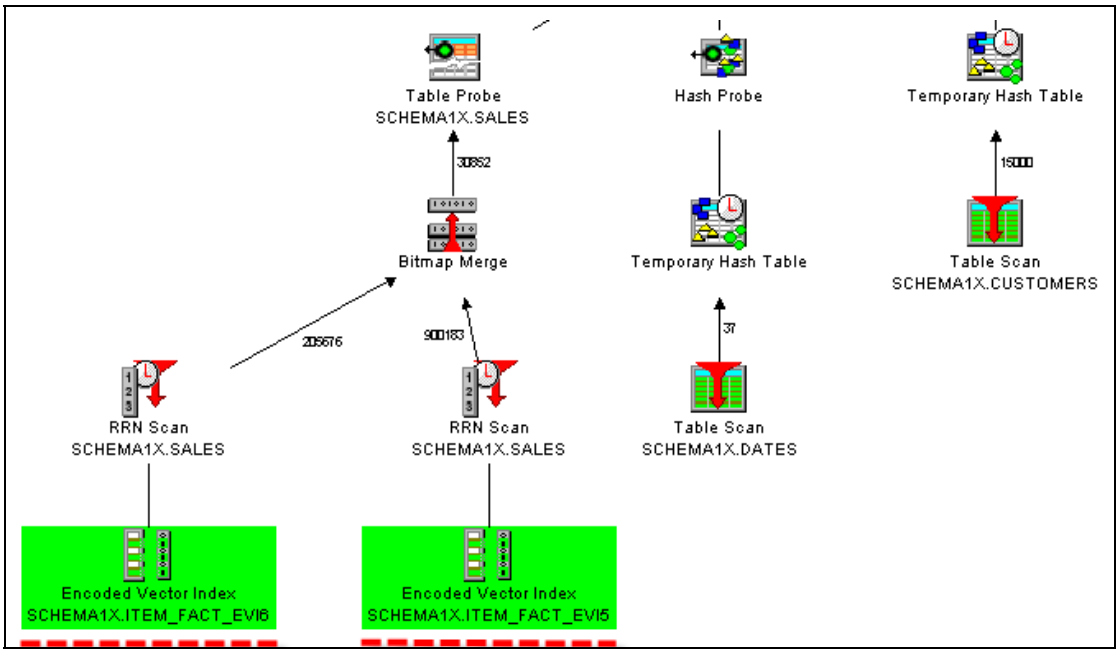

Figure 15. Generating local-selection predicates and EVIs

To help minimize the optimization time for some query types, SQE applies criteria on when and where LPG is considered within a given query plan. For queries that are expected to run in less than 1.5 seconds, LPG strategies are not considered nor applied.

Autonomic Index Advice

Given that SQE has additional strategies and methods for data access the developer must evaluate the indexes available and consider providing radix indexes for columns with only EVIs.

Another benefit of SQE is the ability to provide index advice for local selection predicate columns that are provided by LPG. If a predicate is generated, and the target column does not have a useful index available, the optimizer recommends that an index be created on this column. Normally this will be an EVI.

The index advice can be found through the system-wide index advisor, the SQE system-wide plan cache analysis interface, the SQL performance monitor information or the Visual Explain Index Advisor.

In Visual Explain, index advice can be shown by clicking **Advisor** in the Actions menu (see Figure 16).

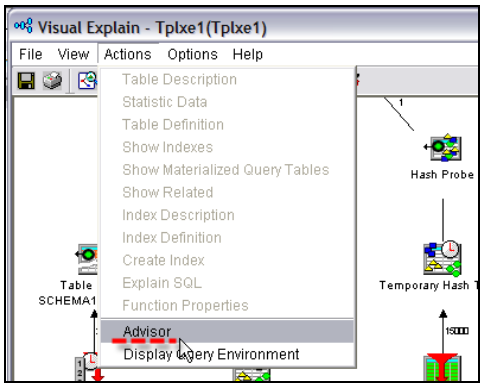

*Figure 16. Clicking **Advisor** to see index advice*

The index advice can also be shown by clicking the **Follow me** icon on the tool bar (see Figure 17).

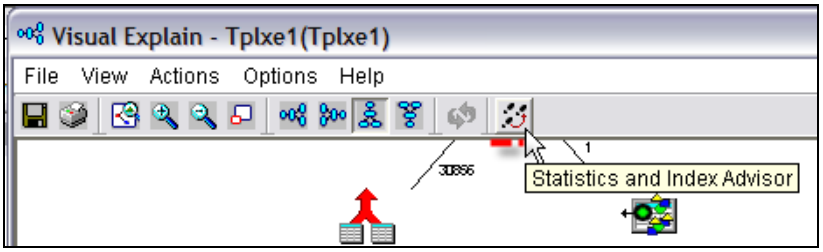

*Figure 17. Clicking the **Follow me** icon to see the index advice*

SQE has the capability to identify and recommend both radix indexes and EVIs. Given that EVIs are usually best when combining multiple indexes together, the optimizer advises a single-column EVI for any column used with LPG (see Figure 18).

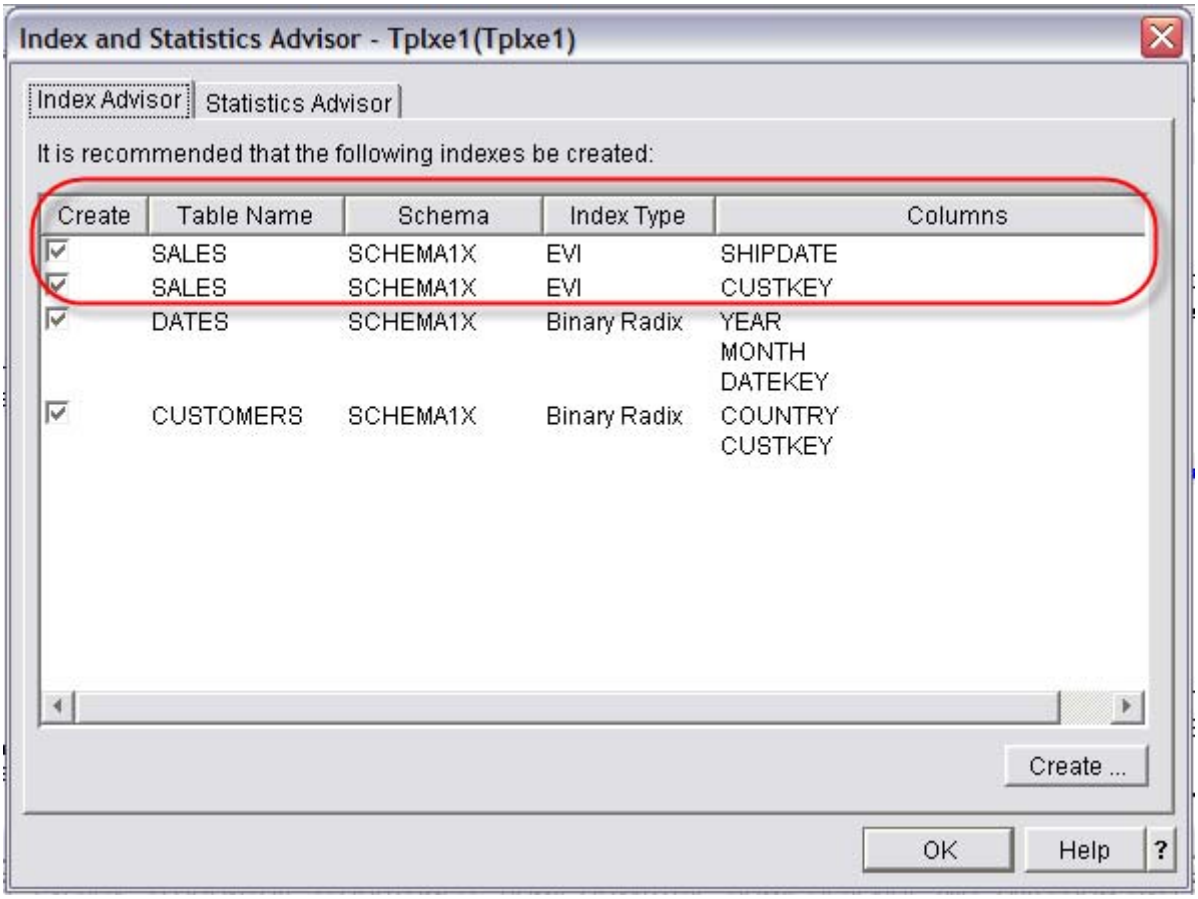

Figure 18. Recommending a single-key EVI

Indexing strategy example

This sample code shows a strategy for indexing to support a star-schema join query and LPG.

```
SELECT        T.CHAR_DATE,
              C.COUNTRY,
              C.CUSTOMER_NAME,
              P.PART_NAME,
              S.SUPPLIER_NAME,
              SUM(F.QUANTITY),
              SUM(F.REVENUE)
FROM          STARLIB.SALES_FACTS F,
              STARLIB.PART_DIM P,
              STARLIB.TIME_DIM T,
              STARLIB.CUST_DIM C,
              STARLIB.SUPP_DIM S
WHERE         F.PARTKEY = P.PARTKEY
AND           F.TIMEKEY = T.TIMEKEY
AND           F.CUSTKEY = C.CUSTKEY
AND           F.SUPPKEY = S.SUPPKEY
AND           T.YEAR = 2011
AND           T.MONTH = 06
AND           T.DAY = 30
AND           C.COUNTRY = 'JAPAN'
AND           P.MFGR = 'Manufacturer#3'
AND           S.SUPPLIER_NAME = 'Supplier#9'
GROUP BY      T.CHAR_DATE,
              C.COUNTRY,
              C.CUSTOMER_NAME,
              P.PART_NAME,
              S.SUPPLIER_NAME
ORDER BY      T.CHAR_DATE,
              C.COUNTRY,
              C.CUSTOMER_NAME,
              P.PART_NAME;

/* Support LPG */
CREATE ENCODED VECTOR INDEX SALES_FACTS_EVI1 ON SALES_FACTS (PARTKEY);
CREATE ENCODED VECTOR INDEX SALES_FACTS_EVI2 ON SALES_FACTS (TIMEKEY);
CREATE ENCODED VECTOR INDEX SALES_FACTS_EVI3 ON SALES_FACTS (CUSTKEY);
CREATE ENCODED VECTOR INDEX SALES_FACTS_EVI4 ON SALES_FACTS (SUPPKEY);

/* Support joins */
CREATE INDEX SALES_FACTS_INDEX1 ON SALES_FACTS (PARTKEY);
CREATE INDEX SALES_FACTS_INDEX2 ON SALES_FACTS (TIMEKEY);
CREATE INDEX SALES_FACTS_INDEX3 ON SALES_FACTS (CUSTKEY);
CREATE INDEX SALES_FACTS_INDEX4 ON SALES_FACTS (SUPPKEY);

/* Support local selection and joins */
CREATE INDEX PART_DIM_INDEX1 ON PART_DIM (MFGR, PARTKEY, PART_NAME);
CREATE INDEX TIME_DIM_INDEX1 ON TIME_DIM (YEAR, MONTH, DAY, TIMEKEY);
CREATE INDEX CUST_DIM_INDEX1 ON CUST_DIM (COUNTRY, CUSTKEY);
CREATE INDEX SUPP_DIM_INDEX1 ON SUPP_DIM (SUPPLIER_NAME, SUPPKEY);
```

As mentioned earlier, when using the star-schema join support, it is not necessary to have radix indexes created over the fact table's join columns; although if primary key and foreign key constraints are defined in the model, the radix indexes will be present. When constraints are not defined, adding these radix indexes allows the optimizer to consider additional strategies and methods for the star schema queries.

At a minimum, radix indexes covering the respective join column of each dimension table should exist. To maximize the speed and efficiency of accessing the dimension-table data during the building of the hash tables, radix indexes over the local selection columns should also exist.

For a snowflake schema or data model, also creating EVIs on the dimension-table foreign key columns is beneficial (and in some cases, required).

Remember that optimizer feedback can provide explicit advice on what indexes to create.

Summary

If developers design and implement a business intelligence solution with DB2 for i, they need to have specialized star-schema join optimization and execution strategies available. These specialized capabilities inherently exist in DB2 for i. With the proper indexing strategy in place, the SQL Query Engine provides robust, high-performance techniques to support an analytical application accessing a star-schema data model.

Appendix A: Resources

These Web sites provide useful references to supplement the information contained in this document:

- IBM eServer i Information Center
  http://publib.boulder.ibm.com/iseries/
- IBM Redbooks™
  www.redbooks.ibm.com/
- DB2 for i home page:
  ibm.com/systems/i/software/db2/index.html
- Indexing and Statistics Strategies for DB2 for i white paper:
  ibm.com/servers/enable/site/education/abstracts/indxng_abs.html
- DB2 for i Education:
  ibm.com/systems/i/software/db2/db2educ_m.html

Appendix B: About the author

Mike Cain

DB2 for i Center of Excellence, IBM Systems and Technology Group Lab Services and Training

Mike Cain is an IBM senior technical staff member and team leader of the DB2 for i Center of Excellence in Rochester, Minnesota. Prior to his current position, he was team leader of the AS/400 Teraplex Integration Center as well as an IBM AS/400 systems engineer and technical consultant.

Trademarks and special notices

References in this document to IBM products or services do not imply that IBM intends to make them available in every country.

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. If these and other IBM trademarked terms are marked on their first occurrence in this information with a trademark symbol (® or ™), these symbols indicate U.S. registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at: www.ibm.com/legal/copytrade.shtml.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Intel, Intel Inside (logos), MMX, and Pentium are trademarks of Intel Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

SET and the SET Logo are trademarks owned by SET Secure Electronic Transaction LLC.

Other company, product, or service names may be trademarks or service marks of others.

Information is provided "AS IS" without warranty of any kind.