Summary of "Multi-Table Joins Through Bitmapped Join Indices"

Eduardo Gutarra Velez

April 8, 2011

O'Neil et al. combine well-known techniques for indexing joins between tables to create a method that can efficiently perform common multi-table joins through bitmap indices. In their paper, they focus mainly on star-joins, but their method can also be applied to other types of joins. A star-join is a join between a fact table and multiple dimension tables.

They review several of the techniques used for performing join operations between tables. These techniques can also be generalized from two tables to multiple tables. The join index is a representation of a pre-computed join between two tables. This representation often associates column values with rows of tables that are being joined. Join indices can be represented as B-Trees or hash indices, and can be organized in any of the following ways:

- Associating row identifiers (RIDs) of both joined tables with the join column value. The RIDs that are associated must satisfy the join condition.
- Given a join condition, identifying a list of RIDs in a second table that corresponds with each RID in a first table.
- Given a join condition, identifying a list of RIDs in a second table that corresponds with a non-join column value of a second table.
- Using variations of these three previous organizations. For example, any of the join indices described above can be extended from working with single column values to working with multi-column values.

As mentioned earlier, join indices may be generalized from two tables to multiple tables. When an index associates an attribute's values with all columns of tables where it occurs, it is called a domain index.

O'Neil et al. explore the possibility of using bitmap indices to represent RIDs in a table. Bitmap indices have some important performance advantages over regular RID list representations:

- First, there is a reduction in I/O (input or output) when a large fraction of a large table is represented as a bitmap rather than by a list of RIDs.

- Second, bitmaps can commonly be pipelined or cached in memory, having RIDs automatically represented in order.

- Lastly, common operations used to combine predicates such as AND and OR may be performed with efficient instructions working on 32 or 64 bits in parallel.

One difficulty when using bitmap indices in database systems is that they require an effective mapping between integer bit positions and the indexed rows. A row identifier is commonly composed of a page number and a slot number within a page where the row is stored. When rows have a fixed size, an equal number of bits can be assigned to consecutive pages to represent their rows on successive slots. However, when records are of variable length, the problem is addressed by defining a maximal number of records per page, and reserving bits accordingly.

O'Neil et al. outline an execution method for multi-table joins using a bitmapped join index. They exemplify their query execution plan for a star-join for which the query is of the following form:

SELECT DISTINCT $T_0.K$, $T_1.A_1$, $T_2.A_2$, ..., $T_n.A_n$
FROM T_0, T_1, T_2,...,T_n
WHERE $T_0.A_1 = T_1.A_1$ AND $T_0.A_2 = T_2.A_2$ AND ... $T_0.A_n = T_n.A_n$
AND $T_1.B_1 = C_1$ AND $T_2.B_2 = C_2$ AND ... $T_n.B_n = C_n$

Here, T_0 is the fact table, and T_i (for $i = 1 \ldots n$) are the dimension tables. The attribute Ai is a key for the dimension table T_i, and is a foreign key for the fact table T_0. Join bitmapped indices are defined on the joining predicates between a fact table and various dimension tables. For example, for the joining predicate $T_0.A_i = T_i.A_i$, they define the join index $T_0.T_i.A_i$ with entries for each RID of T_i. Each RID entry contains a compressed bitmap of all related rows in T_0. A column B_i is a non-key value of the dimension table T_i, and C_i represents a constant value. They presume that indices are also created for non-key columns $T_i.B_i$ (for $i = 1 \ldots n$).

Given the definitions above, their query plan can be summarized as follows:

1. For each dimension table T_i, employ the index $T_i.B_i$ to find rows that satisfy the predicate $T_i.B_i = C_i$.

2. For each resulting RID of the previous step, retrieve the bitmap associated to it in the join index $T_0.T_i.A_i$. These bitmaps are then combined together using OR, creating a bitmap for all related rows of table T_0.

3. As a result from the second step, there should be a bitmap for each predicate of the form $T_i.B_i = C_i$. Take all such resulting bitmaps and intersect them together to generate a bitmap that contains the rows that satisfy all restrictions in the star-join query.

The purpose of using indices is to avoid scanning the entire fact table, or fetching a large number of records from it. The efficiency of the indices depends on the selectivity of the

query. When selectivity is high, the number of records that need to be fetched is small, and thus query response time can be greatly reduced. However, when selectivity is low, query response time can be incremented if the number of rows that need to be scanned is close to the number of rows in the table.

Finally, O'Neil et al. conclude that the challenge facing database systems developers is not in implementing special, new or complex query execution techniques, but in including suitable building blocks that their optimizers can consider when they can improve query time response.