**Summary of “Multi-Table Joins Through Bitmapped Join Indices”**

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 some 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.
* 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 given a join condition.
* 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:

* For one, 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 represented in order automatically.
* 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. Commonly a row identifier is 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, some when records are of variable length, the problem is addressed by defining a maximal number of records per page, and reserving bits accordingly.

Finally, O’Neil et al. outline an execution method for multi-table joins using a bitmapped join index. They execute their query plan on join bitmapped indices built on the joining predicate between a fact table and a dimension table.

An exemplify it with star-joins their query plan can be summarized as follows: Suppose we have