# Chapter 7.21
# Bitmap Join Indexes vs. Data Partitioning

**Ladjel Bellatreche**
*Poitiers University, France*

## INTRODUCTION

Scientific databases and data warehouses store large amounts of data ith several tables and attributes. For instance, the Sloan Digital Sky Survey (SDSS) astronomical database contains a large number of tables with hundreds of attributes, which can be queried in various combinations (Papadomanolakis & Ailamaki, 2004). These queries involve many tables using binary operations, such as joins. To speed up these queries, many optimization structures were proposed that can be divided into two main categories: *redundant structures* like materialized views, advanced indexing schemes (bitmap, bitmap join indexes, etc.) (Sanjay, Chaudhuri & Narasayya, 2000) and vertical partitioning (Sanjay, Narasayya & Yang 2004) and *non redundant structures* like horizontal partitioning (Sanjay, Narasayya & Yang 2004; Bellatreche, Boukhalfa & Mohania, 2007) and parallel processing (Datta, Moon, & Thomas, 2000; Stöhr, Märtens & Rahm, 2000). These optimization techniques are used either in a sequential manner ou combined. These combinations are done intra-structures: materialized views and indexes for redundant and partitioning and data parallel processing for no redundant. Materialized views and indexes *compete for the same resource representing storage*, and *incur maintenance overhead in the presence of updates* (Sanjay, Chaudhuri & Narasayya, 2000). None work addresses the problem of selecting combined optimization structures. In this paper, we propose two approaches; one for combining a non redundant structures horizontal partitioning and a redundant structure bitmap indexes in order to reduce the query processing and reduce the maintenance overhead, and another to exploit algorithms for vertical partitioning to generate bitmap join indexes. To facilitate the understanding of our approaches, for review these techniques in details.

Data partitioning is an important aspect of physical database design. In the context of relational data warehouses, it allows tables, indexes and materialised views to be partitioned into disjoint sets of rows and columns that are physically stored and accessed separately (Sanjay, Narasayya

& Yang 2004). It has a significant impact on performance of queries and manageability of data warehouses. Two types of data partitioning are available: vertical and horizontal partitionings.

The vertical partitioning of a table T splits it into two or more tables, called, sub-tables or vertical fragment, each of which contains a subset of the columns in T. Since many queries access only a small subset of the columns in a table, vertical partitioning can reduce the amount of data that needs to be scanned to answer the query. Note that the key columns are *duplicated* in each vertical fragment, to allow "reconstruction" of an original row in T. Unlike horizontal partitioning, indexes or materialized views, in most of today's commercial database systems there is no native Database Definition Language (DDL) support for defining vertical partitions of a table (Sanjay, Narasayya & Yang 2004). The horizontal partitioning of an object (a table, a vertical fragment, a materialized view, and an index) is specified using a partitioning method (range, hash, list), which maps a given row in an object to a key partition. All rows of the object with the same partition number are stored in the same partition.

Bitmap index is probably the most important result obtained in the data warehouse physical optimization field (Golfarelli, Rizzi & Saltarelli, 2002). The bitmap index is more suitable for low cardinality attributes since its size strictly depends on the number of distinct values of the column on which it is built. Bitmap join indexes (BJIs) are proposed to speed up join operations (Golfarelli, Rizzi & Saltarelli, 2002). In its simplest form, it can be defined as a bitmap index on a table R based on a single column of another table S, where S commonly joins with R in a specific way.

Many studies have recommended the combination of redundant and non redundant structures to get a better performance for a given workload (Sanjay, Narasayya & Yang 2004; Bellatreche, Schneider, Lorinquer & Mohania, 2004). Most of previous work in physical database design did not consider the interdependence between redundant and no redundant optimization structures. Logically, BJIs and horizontal partitioning are two similar optimization techniques - both speed up query execution, pre-compute join operations and concern selection attributes of dimension tables[1]. Furthermore, BJIs and HP can interact with one another, i.e., the presence of an index can make a partitioned schema more efficient and vice versa (since fragments have the same schema of the global table, they can be indexed using BJIs and BJIs can also be partitioned (Sanjay, Narasayya & Yang 2004)).

## BACKGROUND

Note that each BJI can be defined on one or several non key dimension's attributes with a low cardinality (that we call indexed columns) by joining dimension tables owned these attributes and the fact table[2].

**Definition: An indexed attribute** $A_j$ candidate for defining a BJI is a column $A_j$ of a dimension table $D_i$ with a low cardinality (like gender attribute) such that there is a selection predicate of the form: $D_i.A_j\ \theta$ value, $\theta$ is one of six comparison operators {=,<,>,<=,>=}, and value is the predicate constant.

For a large number of indexed attributes candidates, selecting optimal BJIs is an NP-hard problem (Bellatreche, Boukhalfa & Mohania, 2007).

On the other hand, the best way to partition a relational data warehouse is to decompose the fact table based on the fragmentation schemas of dimension tables (Bellatreche & Boukhalfa, 2005). Concretely, (1) partition some/all dimension tables using their simple selection predicates ($D_i.A_j\ \theta$ value), and then (2) partition the facts table using the fragmentation schemas of the fragmented dimension tables (this fragmentation is called derived horizontal fragmentation (Özsu a Valduriez, 1999)). This fragmentation procedure