Chapter 11
Primary and Referential Horizontal Partitioning Selection Problems: Concepts, Algorithms and Advisor Tool

Ladjel Bellatreche
University of Poitiers, France

Kamel Boukhalfa
University of Poitiers, France

Pascal Richard
University of Poitiers, France

ABSTRACT

Horizontal partitioning has evolved significantly in recent years and widely advocated by the academic and industrial communities. Horizontal Partitioning affects positively query performance, database manageability and availability. Two types of horizontal partitioning are supported: primary and referential. Horizontal fragmentation in the context of relational data warehouses is to partition dimension tables by primary fragmentation then fragmenting the fact table by referential fragmentation. This fragmentation can generate a very large number of fragments which may make the maintenance task very complicated. In this paper, we first focus on the evolution of horizontal partitioning in commercial DBMS motivated by decision support applications. Secondly, we give a formalization of the referential fragmentation schema selection problem in the data warehouse and we study its hardness to select an optimal solution. Due to its high complexity, we develop two algorithms: hill climbing and simulated annealing with several variants to select a near optimal partitioning schema. We present ParAdmin, an advisor tool assisting administrators to use primary and referential partitioning during the physical design of their data warehouses. Finally, extensive experimental studies are conducted using the data set of APB1 benchmark to compare the quality the proposed algorithms using a mathematical cost model. Based on these experiments, some recommendations are given to ensure the well use of horizontal partitioning.

DOI: 10.4018/978-1-60960-537-7.ch011
INTRODUCTION

Data warehouses store large volumes of data mainly in relational models such as star or snowflake schemas. A star schema contains a large fact table and various dimension tables. A star schema is usually queried in various combinations involving many tables. The most used operations are joins, aggregations and selections (Papadomanolakis & Ailamaki, 2004). Joins are well known to be expensive operations, especially when the involved relations are substantially larger than the size of the main memory (Lei & Ross, 1999) which is usually the case of business intelligence applications.

The typical queries defined on the star schema are commonly referred to as star join queries, and exhibit the following two characteristics: (1) there is a multi-table join among the large fact table and multiple smaller dimension tables, and (2) each of the dimension tables involved in the join operation has multiple selection predicates on its descriptive attributes. To speed up star join queries, many optimization techniques were proposed. In (Bellatreche, Moussaoui, Necir & Drias, 2008), we classified them into two main categories: redundant techniques such as materialized views, advanced index schemes, vertical partitioning, parallel processing with replication and non-redundant techniques like ad-hoc joins, where joins are performed without additional data structures like indexes (hash join, nested loop, sort merge, etc.), horizontal partitioning, parallel processing without replication. In this paper, we concentrate only on horizontal partitioning, since it is more adapted to reduce the cost of star join queries.

Horizontal partitioning has been mainly used in logical distributed and parallel databases design in last decades (Özsu & Valduriez, 1999; Sacca & Wiederhold, 1985). Recently, it becomes a crucial part of physical database design (Sanjay, Narasayya & Yang, 2004; Papadomanolakis & Ailamaki, 2004; Eadon et al., 2008; Tzoumas, Deshpande & Jensen, 2010; Bellatreche, Cuzzocrea & Benkrid, 2009), where most of today’s commercial DBMS offer native DDL (data definition language) support for defining horizontal partitions (fragments) of a table (Sanjay et al., 2004). In context of relational warehouses, horizontal partitioning allows tables, indexes and materialised views to be partitioned into disjoint sets of rows that are physically stored and accessed separately. Contrary to materialised views and indexes, horizontal data partitioning does not replicate data, thereby reducing space requirements and minimising the update overhead. The main characteristic of data partitioning is its ability to be combined with some redundant optimization techniques such as indexes and materialized views. It also affects positively query performance, database manageability and availability. Query performance, is guaranteed by performing partition elimination. If a query includes a partition key as a predicate in the WHERE clause, the query optimizer will automatically route the query to only relevant partitions. Partitioning can also improve the performance of multi-table joins, by using a technique known as partition-wise joins. It can be applied when two tables are being joined together, and at least one of these tables is partitioned on the join key. Partition-wise joins break a large join into smaller joins. With partitioning, maintenance operations can be focused on particular portions of tables. For maintenance operations across an entire database object, it is possible to perform these operations on a per-partition basis, thus dividing the maintenance process into more manageable chunks. The Administrator can also allocating partitions in different machines (Stöhr, Märtens & Rahm, 2000; Bellatreche, Cuzzocrea & Benkrid, 2010). Another advantage of using partitioning is when it is time to remove data, an entire partition can be dropped which is very efficient and fast, compared to deleting each row individually. Partitioned database objects provide partition independence. This characteristic of partition independence can be an important part of a high-availability strategy. For example, if one partition of a partitioned table is unavailable, all