Chapter 1
The LBF R–Tree
Scalable Indexing and Storage for Data Warehousing Systems

Todd Eavis
Concordia University, Canada

David Cueva
Concordia University, Canada

ABSTRACT

In multi-dimensional database environments, such as those typically associated with contemporary data warehousing, we generally require effective indexing mechanisms for all but the smallest data sets. While numerous such methods have been proposed, the R-tree has emerged as one of the most common and reliable indexing models. Nevertheless, as user queries grow in terms of both size and dimensionality, R-tree performance can deteriorate significantly. Moreover, in the multi-terabyte spaces of today’s enterprise warehouses, the combination of data and indexes — R-tree or otherwise — can produce unacceptably large storage requirements. In this chapter, the authors present a framework that addresses both of these concerns. First, they propose a variation of the classic R-tree that specifically targets data warehousing architectures. Their new LBF R-tree not only improves performance on common user-defined range queries, but gracefully degrades to a linear scan of the data on pathologically large queries. Experimental results demonstrate a reduction in disk seeks of more than 50% relative to more conventional R-tree designs. Second, the authors present a fully integrated, block-oriented compression model that reduces the storage footprint of both data and indexes. It does so by exploiting the same Hilbert space filling curve that is used to construct the LBF R-tree itself. Extensive testing demonstrates compression rates of more than 90% for multi-dimensional data, and up to 98% for the associated indexes.

1 INTRODUCTION

Over the past ten to fifteen years, data warehousing (DW) has become increasingly important to organizations of all sizes. In particular, the representation of historical data across broad time frames allows decision makers to monitor evolutionary patterns and trends that would simply not be possible with operational databases alone. However, this accumulation of historical data comes at a price; namely,
The enormous storage requirements of the tables that house process-specific measurements.

Typically, such databases are constructed as a series of data marts, each designed around a heavily de-normalized logical model known as a Star Schema (or its normalized counterpart, the Snowflake Schema). In its simplest form, the Star schema consists of a central fact table and a series of peripheral dimension tables connected via standard foreign key relationships. Within the fact table, we refer to the foreign keys as feature attributes, while fields identifying the aggregate values for the underlying organizational process are known as measures.

In practice, the dimension tables are relatively small, with their record counts equivalent to the cardinality of the related entity (e.g., date, customer or product). The fact tables, by contrast, can be extremely large and comprise the bulk of the data warehouse. The enormous scale of the fact tables creates two related but distinct processing challenges. First, it is imperative that efficient indexing mechanisms be available. Given the low selectivity of common DW queries (i.e., a large percentage of all records are selected), DW designers often forego indexes altogether and rely upon simple, and expensive, linear table scans. Clearly, this approach can become cost prohibitive in enterprise environments. Second, even in the presence of effective indexing structures, the sheer volume of data can still overwhelm the storage and processing resources of contemporary systems. As such, effective domain-specific compression methods are sorely needed.

In terms of indexing and access methods, it is important to note that the majority of Decision Support queries are multi-dimensional in nature. In other words, the most common access pattern tends to be some form of range query, in which the user submits a request that specifies a range restriction on one or more dimensional columns. Such queries are not well supported by traditional single-dimensional indexes such as the B-tree.

While a significant number of true multi-dimensional techniques have been presented in the literature, the R-tree has emerged as perhaps the most rigorously studied and consistently implemented model. However, while numerous R-tree variations have been presented, none fully support all three of the simple but crucial features of the data warehousing fact table; namely, that it supports medium dimensionality range queries (i.e., 2-10 dimensions), that it is relatively static, and that it is extremely large in practice. The third point, in particular, tends to be under-appreciated in theoretical treatments on the topic.

In this chapter, we discuss the Linear Breadth First R-tree. The LBF R-tree builds directly upon the traditional R-tree model but optimizes its construction and tree traversal in order to consistently reduce the number of costly seeks associated with arbitrary user-defined range queries. Specifically, we utilize a space filling curve to pre-pack the R-tree and then carefully arrange the associated blocks on disk to allow a breadth first traversal pattern to map directly to the physical storage layout. Perhaps more importantly, this linearization model also provides graceful performance degradation in the face of arbitrarily large queries. Experimental results demonstrate a 30% — 50% reduction in the number of seeks relative to existing R-tree packing and query strategies.

As noted, however, indexing alone is not enough to guarantee appropriate performance in practical DW settings. Specifically, given a finite amount of storage space, we are limited in both the number of fact tables that can be constructed, as well as the length of the historical period. Moreover, as fact tables grow in size, real time query costs may become unacceptable to end users. A partial solution to the latter constraint is the materialization of summary views that aggregate the fact table data at coarser levels of granularity. But even here, the space available for such views is limited by the size of the underlying fact tables. As a result, we can conclude that a reduction in fact table size not only improves space utilization but it allows for the construction of additional