Indexing Medium-dimensionality Data in Oracle

K. V. Ravi Kanth, Siva Ravada, Jayant Sharma and **Jay Banerjee**
Oracle Corporation, Nashua NH 03060.
{rkothuri, sravada, jsharma, jbanerje}@us.oracle.com

1 Introduction

With the advent of GIS, multi-media, and warehousing technologies, database systems have started focusing on storage and access of multi-dimensional data such as spatial, OLAP, image, audio, and video attributes. As a step in this direction, Oracle8i launched the *interMedia* product to support spatial and image data, and *Materialized Views (MV)* to support warehousing applications. Although 2-dimensional spatial data is efficiently indexed using *Oracle8i Spatial*, and high-dimensional image data using a combination of bitmap indexing and the *Visual Information Retrieval (VIR)* product, there is still a need for efficient indexing mechanisms for medium-dimensionality data such as OLAP, and CAD/CAM applications. In this paper, we describe the implementation of a new indextype, called the R-tree, to support medium-dimensionality data (i.e., data whose dimensionality is in the range of 3-10) using the extensible indexing framework [DDSS95] of Oracle8i. This indextype combines some of the best features of existing R-tree variants [Gut84, BKSS90, WJ96, LLE97, RKV95].

2 Framework

The new indextype is implemented using the extensible indexing framework of Oracle8i (referred to as *cooperative indexing* in [DDSS95]). This framework allows easy creation and maintenance of domain-specific index structures on top of the server layer while reaping the full benefits of operating within a database framework. As a consequence, the new index structure inherits its features such as transactional semantics, integrated backup and recovery, security, and replication from the underlying database.

3 Storage

The R-tree indextype being defined in Oracle8i can index two datatypes: an *sdo_mbr* type, which is a *d*-dimensional rectangle specified by the lower-left and the upper-right corners, or an *sdo_geometry* type, which is an Oracle8i object type that allows for the specification of complex geometries (as defined by OGC).

Data items are stored in a relational table, which we refer to as the *base* table. The R-tree constructed for the data items is stored in the database using a *metadata* table storing the information about the root of the R-tree, its dimensionality and fanout, and an *index* table storing the nodes of the R-tree.

4 Index Creation and Update

4.1 Bulk Creation

The user can specify one of two creation strategies: one using the *sort-tile-recursive* approach [LLE97], and another using the *VAMSplit* R-tree approach [WJ96]. The first one has fast creation times whereas the second approach obtains good query response times [WJ96].

4.2 Inserts and deletes

Inserts and deletes (referred to as updates) are processed in two phases: a *locate* phase, and an *update* phase. In the *locate* phase, updates propagate from the root to a leaf node wherein the update is to be performed. During the *update* phase, the update is processed in the identified leaf node and the changes of the update are propagated up the tree. Note that unlike a B^+-tree, the update may propagate up even when there is no split of the leaf node. This happens when the MBRs of the nodes on the update path need to be adjusted due to update of the lower-level nodes. Oracle read-consistency model ensures that node accesses either during queries or during the locate phase of an update are never blocked. Node accesses are blocked only during the update phase.

Although reads in a query do not observe committed tree changes performed after the start of the query due to Oracle read-consistency model, node-updates in an update operation do see the latest committed versions of the updated nodes. As a result, node-splits and node-deletes may lead to inconsistencies during concurrent updates. To ensure good concurrency without sacrificing tree integrity, we adopt the following strategy. First, we do not allow updates to delete nodes: empty and under-filled nodes are reclaimed during a separate tree reorganization operation that locks the entire tree. Next, for detecting nodes-splits during concurrent updates, we use the node_id information of the nodes.

4.3 Altering the Index

In this operation, the user is allowed to either reconstruct the entire tree from scratch, or to reorganize the tree by first eliminating empty, and under-filled nodes, and then improving the quality of the tree by doing forced reinsertion.

5 Query Operations

The R-tree indextype supports three types of operations: window queries, nearest-neighbor queries, and intersection joins. Window queries specify a query window and retrieve data whose MBRs interact with the query window in one of 4 ways: *intersection*, *containment*, *enclosure* and *exact-match*. Nearest-neighbor queries specify a query point and retrieve the k closest data MBRs. Joins identify the data items of two different datasets that intersect with each other. Note that these queries are processed using the MBRs. For some applications such as GIS data where the bounding rectangles only represent first-level approximations of the data items, the query result may have to be post-processed using the complete extents of the data items to obtain the final result.

All queries are implemented as operators in Oracle8i extensible indexing framework. The operators are evaluated using start, fetch, and close routines of the extensible indexing interface. The query is set up in the start routine (for instance, a stack is initialized with just the root node on top for window queries). It is incrementally processed in the fetch routine, and the associated stacks and queues are cleaned up in the close routine.

Note that to ensure limited memory usage, the queries are processed in a depth-first manner. Consequently the query algorithms for window, nearest-neighbor, and join queries [BKSS90, BKS93, HJR97] are adapted appropriately. For instance, a spatial join operation on two nodes R and S (each corresponding to a different R-tree) identifies a pair of child entries, and propagates to the corresponding pair of child nodes, without identifying the other pairs of entries of R and S that need to be processed later. To facilitate on-the-fly identification of the pair of child entries to process next, the nodes R and S are stored on a stack along with appropriate information. As a result, the memory requirement reduces from $O(n^2)$ (corresponding to all pairs of entries of R and S as in [HJR97]) to $O(2*n)$ (corresponding to storing R and S), where n is number of entries in each node.

6 Extensions

In addition to indexing inherently multi-dimensional columns, R-trees can also be used to index multiple columns so as to answer queries on multiple columns efficiently. Such extensions are being considered for future versions.

References

[BKS93] T. Brinkoff, H. P. Kriegel, and B. Seeger. Efficient processing of spatial joins using R-trees. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 237–246, 1993.

[BKSS90] N. Beckmann, H. Kriegel, R. Schneider, and B. Seeger. The R* tree: An efficient and robust access method for points and rectangles. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 322–331, 1990.

[DDSS95] S. Defazio, A. Daoud, L. A. Smith, and J. Srinivasan. Integrating ir and rdbms using cooperative indexing. In *Proc. of ACM SIGIR Conf. on Information Retrieval*, pages 84–92, 1995.

[Gut84] A. Guttman. R-trees: A dynamic index structure for spatial searching. *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 47–57, 1984.

[HJR97] Yun-Wu Huang, Ning Jing, and Elke A. Rundensteiner. Spatial joins using r-trees: Breadth-first traversal with global optimizations. In *Procȯf the Int. Conf. on Very Large Data Bases*, pages 396–405, 1997.

[LLE97] S. T. Leutenegger, M. A. Lopez, and J. M. Edgington. STR: A simple and efficient algorithm for R-tree packing. In *Proc. Int. Conf. on Data Engineering*, 1997.

[RKV95] N. Roussopoulos, S. Kelley, and F. Vincent. Nearest neighbor queries. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 71–79, May 1995.

[WJ96] D. White and R. Jain. Algorithms and strategies for similarity retrieval. *Proc. of the SPIE Conference*, 1996.